IBM® DB2 Universal Database™

SQL リファレンス
第 2 巻

バージョン 8.2

SC88-9156-01
(英文原典：SC09-4845-01)

IBM® DB2 Universal Database™

SQL リファレンス
第 2 巻

バージョン 8.2

SC88-9156-01
(英文原典：SC09-4845-01)

ご注意!

本書および本書で紹介する製品をご使用になる前に、『*特記事項*』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC09-4845-01
IBM® DB2 Universal Database™
SQL Reference Volume 2
Version 8.2

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.8

目次

本書について

SQL リファレンス (第 1 巻および第 2 巻) は、 DB2 Universal Database バージョン 8 で使用される SQL 言語を定義し、以下の情報について説明しています。

- リレーショナル・データベースの概念、言語エレメント、関数、照会の形式に関する情報 (第 1 巻)。
- SQL ステートメントの構文やセマンティクスに関する情報 (第 2 巻)。

本書の対象読者

本書は、構造化照会言語 (SQL) を使用してデータベースにアクセスする方々を対象にしています。主にプログラマーとデータベース管理者の方々を対象にしていますが、コマンド行プロセッサー (CLP) を使用してデータベースにアクセスする方々にも役立ちます。

本書は学習用ではなく、参照資料です。読者がアプリケーション・プログラムを作成することを前提にしており、したがって、 データベース・マネージャーの機能を詳細に説明しています。

本書の構成

本書では、以下の主なトピックについて説明します。

- 1 ページの『ステートメント』では、 SQL プロシージャー・ステートメントを含むすべての SQL ステートメントについて、構文図、セマンティックの説明、規則、および例を記載しています。

第 1 巻の概要

SQL リファレンスの第 1 巻では、リレーショナル・データベースの概念、言語エレメント、関数、照会について説明します。第 1 巻の特定の章および付録について、以下で簡単に説明します。

- 『概念』では、リレーショナル・データベースと SQL の基本概念について説明します。
- 『言語エレメント』では、多くの SQL ステートメントに共通する SQL 構文および言語エレメントについて説明します。
- 『関数』では、SQL の列およびスカラー関数の構文図、セマンティックの説明、規則、および使用例について説明します。
- 『照会』では、様々な形式の照会について説明します。
- 『SQL の制限値』では、SQL の制限についてのリストを示します。
- 『SQL 連絡域 (SQLCA)』では、SQLCA 構造体について説明します。
- 『SQL 記述子域 (SQLDA)』では、SQLDA 構造体について説明します。
- 『カタログ・ビュー』では、データベース・カタログ・ビューについて説明します。

- 『フェデレーテッド・システム』では、フェデレーテッド・システムのオプションとタイプ・マッピングについて説明します。
- 『サンプル・データベース表』では、例で使用されるサンプルの表について説明します。
- 『予約スキーマ名と予約語』では、 IBM SQL および ISO/ANSI SQL99 標準規格の予約スキーマ名と予約語について示します。
- 『トリガーと制約の相互作用』では、トリガーと参照制約の相互作用について説明します。
- 『Explain 表』では、Explain 表について説明します。
- 『Explain レジスターの値』では、 CURRENT EXPLAIN MODE と CURRENT EXPLAIN SNAPSHOT の特殊レジスター値の相互作用、および PREP コマンドと BIND コマンドとの相互作用について説明します。
- 『例外表』では、SET INTEGRITY ステートメントで使用する、ユーザー作成の表に関する情報を示します。
- 『ルーチンで許可される SQL ステートメント』では、SQL データ・アクセス・コンテキストが異なっても、ルーチン内で実行できる SQL ステートメントのリストを示します。
- 『CALL』では、コンパイル済みステートメントから呼び出せる CALL ステートメントについて説明します。
- 『日本語および繁体字中国語 EUC についての考慮事項』では、拡張 UNIX コード (EUC) 文字セットを使用する際の考慮事項のリストを示します。
- 『DATALINK での BNF の指定』では、 DATALINK でのバッカス正規形式 (BNF) の指定について説明します。

構文図の見方

本書を通じて、構文の説明には次のように定義される構造の図が使用されます。

構文図は、左から右、上から下に、線に沿って読みます。

記号 ▶▶── は、構文図の始まりを示します。

記号 ──▶ は、構文が次の行に続くことを示します。

記号 ▶── は、構文が前の行から続いていることを示します。

記号 ──▶◀ は、構文図の終わりを示します。

構文フラグメントは、記号 ├── で始まり、記号 ──┤ で終わります。

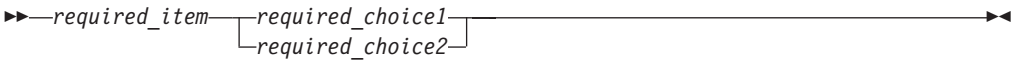
必須項目は、横線 (メインパス) 上に示されます。

オプション項目は、メインパスの下に示されます。

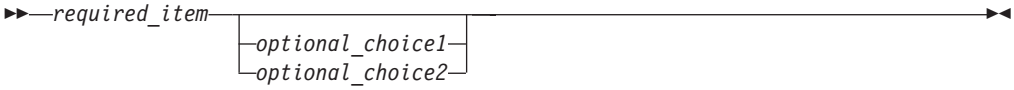
オプション項目をメインパスの上に示すこともありますが、それは構文図を見やすくするためであり、実行には関係しません。

複数の項目からの選択が可能な場合、それらの項目を縦に並べて (スタックに) 示しています。

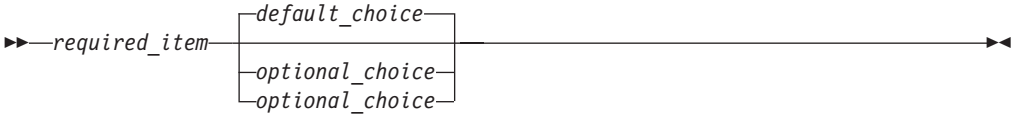
項目から 1 つを選択しなければならない 場合、スタックの項目の 1 つはメインパス上に示されます。



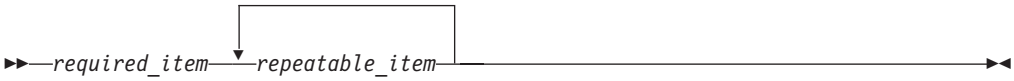
項目から 1 つをオプションで選択できる場合、スタック全体がメインパスよりも下に示されます。



項目の 1 つがデフォルト値の場合、その項目はメインパスより上に示され、残りの選択項目はメインパスよりも下に示されます。



メインパスの上に、左へ戻る矢印がある場合には、項目を繰り返して指定できることを示しています。このような場合、繰り返す項目相互の間は、1 つ以上のブランクで区切らなければなりません。



繰り返しの矢印にコンマが示されている場合は、繰り返し項目をコンマで区切らなければなりません。

スタックの上部の反復の矢印の記号は、そのスタックの中から複数の項目を選択できること、または 1 つの選択項目を繰り返して選択できることを示します。

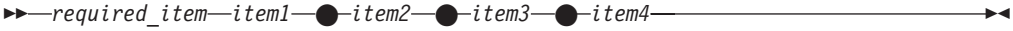
キーワードは英大文字で示してあります (例: FROM)。示されているとおりに入力することが必要です。変数は英小文字で示しています (例: column-name)。このような変数は、構文にユーザーが指定する名前や値を示しています。

句読点、括弧、算術演算子、その他の記号が示されている場合には、それらを構文の一部として入力する必要があります。

1 つの変数が、構文を構成する大きいフラグメントを表すことがあります。たとえば次の図で、変数 parameter-block は、 **parameter-block** というラベルの構文フラグメント全体を表します。

parameter-block:

『黒丸』 (●) ではさまれて隣接しているセグメントは、任意の順序で指定することができます。

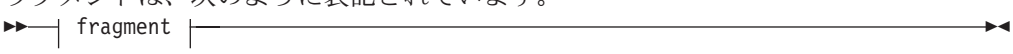


上記の図は、item2 と item3 をどのような順序で指定しても構わないことを示しています。以下はいずれも有効です。

```
required_item item1 item2 item3 item4
required_item item1 item3 item2 item4
```

共通の構文エレメント

構文図で使用されるいくつかの構文フラグメントについて、以下に説明します。フラグメントは、次のように表記されています。

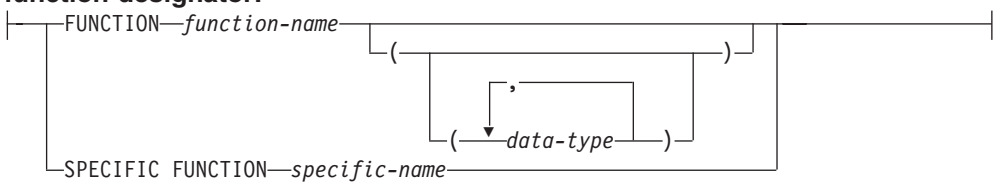


関数指定子

関数指定子は、単一の関数を一意的に識別します。一般的に関数指定子は、関数の DDL ステートメント (DROP または ALTER など) で使用されます。

構文:

function-designator:



説明:

FUNCTION *function-name*

特定の関数を指定します。 *function-name* という名前の関数インスタンスがスキーマ内に 1 つだけ存在している場合にのみ有効です。指定する関数には、任意の数のパラメーターを定義できます。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマにこの名前の関数が存在

しない場合は、エラー (SQLSTATE 42704) になります。指定したスキーマまたは暗黙のスキーマに、この関数のインスタンスが複数存在する場合は、エラー (SQLSTATE 42725) になります。

FUNCTION *function-name (data-type,...)*
関数を固有に指定する関数シグニチャーを指定します。関数解決のアルゴリズムは使用されません。

function-name
関数の名前を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(data-type,...)
値は、CREATE FUNCTION ステートメント上で (対応する位置に) 指定されたデータ・タイプに一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値が、特定の関数インスタンスを識別するのに使用されます。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントで指定された値と完全に一致していなければなりません。

0<*n*<25 は REAL を意味し、24<*n*<54 は DOUBLE を意味するので、FLOAT(*n*) のタイプは、*n* に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つ関数がない場合は、エラー (SQLSTATE 42883) になります。

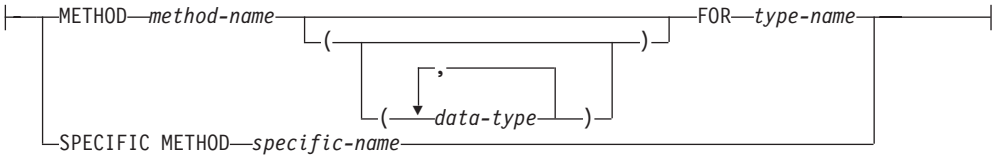
SPECIFIC FUNCTION *specific-name*
関数の作成時に指定された名前、またはデフォルト値として与えられた名前を使用して、特定のユーザー定義関数を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* (特定名) は、指定したスキーマまたは暗黙のスキーマの特定関数のインスタンスを指定していなければなりません。そうでない場合、エラー (SQLSTATE 42704) になります。

メソッド指定子

メソッド指定子は、単一のメソッドを一意的に識別します。一般的にメソッド指定子は、メソッドの DDL ステートメント (DROP または ALTER など) で使用されます。

構文:

method-designator:



説明:

METHOD *method-name*

特定のメソッドを指定します。 *type-name* というサブジェクト・タイプの *method-name* という名前のメソッド・インスタンスが 1 つだけ存在している場合にのみ有効です。このように指定されたメソッドには、任意の数のパラメーターを定義できます。タイプに、指定された名前のメソッドが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。タイプに、そのメソッドのインスタンスが複数存在する場合も、エラーが戻されます (SQLSTATE 42725)。

METHOD *method-name (data-type,...)*

メソッドを一意に指定するメソッド・シグニチャーを指定します。メソッド解決のアルゴリズムは使用されません。

method-name

type-name タイプのメソッドの名前を指定します。

(data-type,...)

値は、 CREATE TYPE ステートメント上で (対応する位置に) 指定されたデータ・タイプと一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値が、特定のメソッド・インスタンスを識別するのに使用されます。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

長さ、精度、または位取りをコーディングする場合、その値は、CREATE TYPE ステートメントで指定された値と完全に一致していなければなりません。

0<*n*<25 は REAL を意味し、24<*n*<54 は DOUBLE を意味するので、FLOAT(*n*) のタイプは、*n* に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つメソッドのタイプがない場合は、エラー (SQLSTATE 42883) になります。

FOR *type-name*
指定されたメソッドを関連付けるタイプを指定します。ここで指定される名前は、カタログにすでに記述されているタイプを示すものでなければなりません (SQLSTATE 42704)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

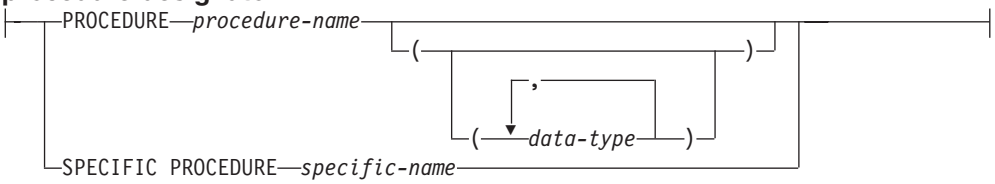
SPECIFIC METHOD *specific-name*
メソッドの作成時に指定された名前か、デフォルト値として与えられた名前を使用して、特定のメソッドを識別します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* に指定される名前は、指定したスキーマまたは暗黙のスキーマに含まれる特定メソッドのインスタンスを識別するものでなければなりません。それ以外の名前が指定された場合は、エラーが戻されます (SQLSTATE 42704)。

プロシージャー指定子

プロシージャー指定子は、単一のストアード・プロシージャーを一意的に識別します。一般的にプロシージャー指定子は、プロシージャーの DDL ステートメント (DROP または ALTER など) で使用されます。

構文:

procedure-designator:



説明:

PROCEDURE *procedure-name*
特定のプロシージャーを指定します。 *procedure-name* という名前のプロシージャー・インスタンスがスキーマ内に 1 つだけ存在している場合にのみ有効です。指定するプロシージャーには、任意の数のパラメーターを定義できます。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のない

オブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマに該当する名前のプロシージャーが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。指定したスキーマまたは暗黙のスキーマに、このプロシージャーのインスタンスが複数存在する場合は、エラー (SQLSTATE 42725) になります。

PROCEDURE *procedure-name (data-type,...)*
プロシージャーを一意的に固有に識別するプロシージャー・シグニチャーを指定します。プロシージャー解決のアルゴリズムは使用されません。

procedure-name
プロシージャーの名前を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(data-type,...)
値は、CREATE PROCEDURE ステートメント上で (対応する位置に) 指定されたデータ・タイプに一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値が、特定のプロシージャー・インスタンスを識別するのに使用されます。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

長さ、精度、または位取りをコーディングする場合、その値は、CREATE PROCEDURE ステートメントで指定された値と完全に一致していなければなりません。

0<*n*<25 は REAL を意味し、24<*n*<54 は DOUBLE を意味するので、FLOAT(*n*) のタイプは、*n* に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つプロシージャーがない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC PROCEDURE *specific-name*
プロシージャーの作成時に指定された名前、またはデフォルト値として与えられた名前を使用して、特定のプロシージャーを指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* に指定される名前は、指定したスキー

マまたは暗黙のスキーマに含まれる特定プロシージャーのインスタンスを識別するものでなければなりません。それ以外の名前が指定された場合は、エラーが戻されます (SQLSTATE 42704)。

本書の表記規則

この項では、本書全体で使用する表記規則について説明します。

エラー条件

本書の本文中で、エラー条件は、そのエラーに対応する SQLSTATE を括弧で囲んで示します。例:

シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

強調表記の規則

本書では、以下の規則を使用しています。

太字 (Bold)	コマンド、キーワード、およびその名前がシステムによって事前定義されているその他の項目を示します。
イタリック (Italic)	次のいずれかを示します。 • ユーザーが指定する名前または値 (変数) • 一般的な強調 • 初出の新しい用語 • 他の情報源への参照
モノスペース (Monospace)	次のいずれかを示します。 • ファイルおよびディレクトリー • コマンド・プロンプトまたはウィンドウで入力するよう指示される情報 • 特定のデータ値の例 • システムが表示するテキストの例 • システム・メッセージの例

関連資料

アプリケーションの作成には、以下の資料が役立ちます。

- *管理ガイド*
  - ローカルに、またはクライアント/サーバー環境でアクセスされるデータベースの設計、インプリメント、および保守を行うのに必要な情報が記載されています。
- *アプリケーション開発ガイド*
  - アプリケーションの開発プロセスについて述べ、組み込み SQL と API を使用してデータベースにアクセスするアプリケーション・プログラムをコーディング、コンパイル、および実行する方法について説明しています。
- *DB2 Universal Database for iSeries SQL 解説書*
  - iSeries (AS/400) 上で DB2 照会マネージャーおよび SQL Development Kit でサポートされる構造化照会言語 (SQL) を定義しています。システム管理、デ

ータベース管理、アプリケーション・プログラミング、および操作に関するタスクについての参照情報が記載されています。 DB2 を実行している iSeries (AS/400) システムで使用される SQL ステートメントごとに、構文、使用上の注意、キーワード、および例が記述されています。

- *DB2 Universal Database for z/OS and OS/390 SQL Reference*
  - DB2 for z/OS (OS/390) で使用される構造化照会言語 (SQL) を定義しています。 DB2 を実行している z/OS (OS/390) システムにおける照会の形式、SQL ステートメント、 SQL プロシージャー・ステートメント、DB2 の制限、SQLCA、 SQLDA、カタログ表、および SQL 予約語が記載されています。
- *DB2 Spatial Extender ユーザーズ・ガイド*
  - 地理情報システム (GIS) を作成して使うアプリケーションを作成する方法を説明しています。 GIS を作成して使用することには、データベースにリソースを提供し、そのデータを照会して、区域内の位置、距離、および分散などの情報を入手することが関係しています。
- *IBM SQL リファレンス*
  - この資料は、データベース製品の IBM のデータベース製品全般に共通するすべての SQL エレメントについて説明しています。 IBM のデータベースを使用して移植可能プログラムを作成するのに役立つ制約事項と規則が示されています。さまざまな規格や製品 (SQL92E、XPG4-SQL、IBM-SQL および IBM リレーショナル・データベースの製品) の間での SQL の拡張機能や非互換性について、リストの形で示されています。
- *American National Standard X3.135-1992, Database Language SQL*
  - SQL の ANSI 規格の定義について説明しています。
- *ISO/IEC 9075:1992, Database Language SQL*
  - SQL の 1992 ISO 規格の定義について説明しています。
- *ISO/IEC 9075-2:1999, Database Language SQL - Part 2: Foundation (SQL/Foundation)*
  - SQL の 1999 ISO 規格の定義についてその大部分を網羅しています。
- *ISO/IEC 9075-4:1999, Database Language SQL - Part 4: Persistent Stored Modules (SQL/PSM)*
  - SQL プロシージャー制御ステートメントの 1999 ISO 規格の定義について説明しています。
- *ISO/IEC 9075-5:1999, Database Language SQL — Part 4: Host Language Bindings (SQL/Bindings)*
  - ホスト言語バインディングと動的 SQL の 1999 ISO 規格の定義について説明しています。

ステートメント

この章には、SQL ルーチン、トリガー、または動的コンパウンド・ステートメントの本体を構成するステートメントを含む、SQL ステートメントの構文図、セマンティックの説明、規則、および使用例があります。

サポートされる SQL ステートメント

次の表には、サポートされる SQL ステートメントがリストされています。

表 1. SQL ステートメント

SQL ステートメント	目的
13 ページの『ALLOCATE CURSOR』	結果セット・ロケーター変数で識別される結果セットにカーソルを割り当てる。
15 ページの『ALTER BUFFERPOOL』	バッファー・プールの定義を変更する。
18 ページの『ALTER DATABASE PARTITION GROUP』	データベース・パーティション・グループの定義を変更する。
22 ページの『ALTER FUNCTION』	関数のプロパティーを変更して既存の関数を変更する。
25 ページの『ALTER METHOD』	メソッドに関連したメソッド本文を変更して既存のメソッドを変更する。
27 ページの『ALTER NICKNAME』	ニックネームの定義を変更する。
34 ページの『ALTER PROCEDURE』	プロシージャーのプロパティーを変更して既存のプロシージャーを変更する。
37 ページの『ALTER SEQUENCE』	シーケンスの定義を変更する。
41 ページの『ALTER SERVER』	フェデレーテッド・システム内のデータ・ソースの定義を変更する。
45 ページの『ALTER TABLE』	表の定義を変更する。
79 ページの『ALTER TABLESPACE』	表スペースの定義を変更する。
87 ページの『ALTER TYPE (構造化)』	構造タイプの定義を変更する。
95 ページの『ALTER USER MAPPING』	ユーザー許可マッピングの定義を変更する。
98 ページの『ALTER VIEW』	参照タイプ列を変更して有効範囲を追加することによって、ビューの定義を変更する。
100 ページの『ALTER WRAPPER』	特定のタイプのデータ・ソースにアクセスするために (ラッパー・モジュールと一緒に) 使用されるオプションを更新する。
102 ページの『ASSOCIATE LOCATORS』	ストアード・プロシージャーから戻される結果セットごとの結果セット・ロケーター値を入手する。
104 ページの『BEGIN DECLARE SECTION』	ホスト変数宣言セクションの始まりを示す。
107 ページの『CALL』	ストアード・プロシージャーを呼び出す。
113 ページの『CASE』	複数の条件に基づいて実行パスを選択する。
116 ページの『CLOSE』	カーソルをクローズする。
118 ページの『COMMENT』	オブジェクトの記述のコメントを置換または追加する。
129 ページの『COMMIT』	作業単位を終了し、その作業単位によって行われたデータベースの変更をコミットする。

表 1. SQL ステートメント (続き)

SQL ステートメント	目的
131 ページの『コンパウンド SQL (動的)』	1 つまたは複数の他の SQL ステートメントを結合して 1 つの動的ブロックにする。
136 ページの『コンパウンド SQL (組み込み)』	1 つまたは複数の他の SQL ステートメントを結合して 1 つの実行可能ブロックにする。
140 ページの『コンパウンド SQL (プロシージャー)』	別のステートメントを SQL プロシージャーにグループ化する。
149 ページの『CONNECT (タイプ 1)』	リモート作業単位の規則に従って、アプリケーション・サーバーに接続する。
157 ページの『CONNECT (タイプ 2)』	アプリケーション制御の分散作業単位の規則に従って、アプリケーション・サーバーに接続する。
165 ページの『CREATE ALIAS』	表、ビュー、または別の別名に対する別名を定義する。
168 ページの『CREATE BUFFERPOOL』	新しいバッファー・プールを作成する。
172 ページの『CREATE DATABASE PARTITION GROUP』	データベース・パーティション・グループを定義する。
175 ページの『CREATE DISTINCT TYPE』	特殊データ・タイプを定義する。
181 ページの『CREATE EVENT MONITOR』	モニターしたいデータベースのイベントを指定する。
199 ページの『CREATE FUNCTION』	ユーザー定義関数を登録する。
200 ページの『CREATE FUNCTION (外部スカラー)』	ユーザー定義外部スカラー関数を登録する。
226 ページの『CREATE FUNCTION (外部表)』	ユーザー定義外部表関数を登録する。
245 ページの『CREATE FUNCTION (OLE DB 外部表)』	ユーザー定義 OLE DB 外部表関数を登録する。
252 ページの『CREATE FUNCTION (ソースまたはテンプレート)』	ユーザー定義ソース関数を登録する。
263 ページの『CREATE FUNCTION (SQL スカラー、表、または行)』	ユーザー定義 SQL 関数を登録および定義する。
272 ページの『CREATE FUNCTION MAPPING』	関数マッピングを定義する。
277 ページの『CREATE INDEX』	表の索引を定義する。
286 ページの『CREATE INDEX EXTENSION』	構造タイプまたは特殊タイプ列を使用して、表で索引を使用するための拡張オブジェクトを定義する。
293 ページの『CREATE METHOD』	以前から定義されているメソッド指定にメソッド本体を関連付ける。
299 ページの『CREATE NICKNAME』	ニックネームを定義する。
311 ページの『CREATE PROCEDURE』	ストアード・プロシージャーを登録する。
312 ページの『CREATE PROCEDURE (外部)』	外部ストアード・プロシージャーを登録する。
327 ページの『CREATE PROCEDURE (SQL)』	SQL ストアード・プロシージャーを登録する。
334 ページの『CREATE SCHEMA』	スキーマを定義する。
337 ページの『CREATE SEQUENCE』	シーケンスを定義する。
342 ページの『CREATE SERVER』	データ・ソースをフェデレーテッド・データベースへ定義する。
347 ページの『CREATE TABLE』	表を定義する。

表 1. SQL ステートメント (続き)

SQL ステートメント	目的
410 ページの『CREATE TABLESPACE』	表スペースを定義する。
420 ページの『CREATE TRANSFORM』	変換関数を定義する。
428 ページの『CREATE TRIGGER』	トリガーを定義する。
440 ページの『CREATE TYPE (構造化)』	構造化データ・タイプを定義する。
466 ページの『CREATE TYPE MAPPING』	データ・タイプ間でのマッピングを定義する。
472 ページの『CREATE USER MAPPING』	ユーザー許可間でのマッピングを定義する。
474 ページの『CREATE VIEW』	1 つまたは複数の表、ビュー、あるいはニックネームのビューを定義する。
489 ページの『CREATE WRAPPER』	ラッパーを登録する。
491 ページの『DECLARE CURSOR』	SQL カーソルを定義する。
497 ページの『DECLARE GLOBAL TEMPORARY TABLE』	グローバル一時表を定義する。
505 ページの『DELETE』	1 つまたは複数の行を表から削除する。
512 ページの『DESCRIBE』	準備済みの SELECT ステートメントの結果列を記述する。
517 ページの『DISCONNECT』	アクティブな作業単位がない場合に 1 つまたは複数の接続を終了する。
520 ページの『DROP』	データベース中のオブジェクトを削除する。
549 ページの『END DECLARE SECTION』	ホスト変数宣言セクションの終わりを示す。
551 ページの『EXECUTE』	準備済み SQL ステートメントを実行する。
558 ページの『EXECUTE IMMEDIATE』	SQL ステートメントを準備して実行する。
561 ページの『EXPLAIN』	選択したアクセス・プランについての情報をキャプチャーする。
566 ページの『FETCH』	行の値をホスト変数に割り当てる。
569 ページの『FLUSH EVENT MONITOR』	イベント・モニターのアクティブな内部バッファーを書き出す。
570 ページの『FLUSH PACKAGE CACHE』	現在パッケージ・キャッシュ内に存在する、キャッシュに入れられたすべての動的 SQL を除去する。
571 ページの『FOR』	表の行ごとに、ステートメントまたはステートメントのグループを実行する。
574 ページの『FREE LOCATOR』	ロケーター変数とその値の間の関連を除去する。
575 ページの『GET DIAGNOSTICS』	以前に実行した SQL ステートメントについての情報を得るために使用される。
578 ページの『GOTO』	SQL プロシージャー内のユーザー定義ラベルに分岐させるために使用される。
580 ページの『GRANT (データベース権限)』	データベース全体に対する権限を付与する。
584 ページの『GRANT (索引特権)』	データベースの索引に対する CONTROL 特権を付与する。
586 ページの『GRANT (パッケージ特権)』	データベースのパッケージに対する特権を付与する。
589 ページの『GRANT (ルーチン特権)』	ルーチン (関数、メソッド、またはプロシージャー) についての特権を付与する。
593 ページの『GRANT (スキーマ特権)』	スキーマについての特権を付与する。
596 ページの『GRANT (シーケンス特権)』	シーケンスについての特権を付与する。
599 ページの『GRANT (サーバー特権)』	特定のデータ・ソースを照会するための特権を付与する。
601 ページの『GRANT (表スペース特権)』	表スペースについての特権を付与する。

表 1. SQL ステートメント (続き)

SQL ステートメント	目的
603 ページの『GRANT (表、ビュー、またはニックネーム特権)』	表、ビュー、およびニックネームについての特権を付与する。
611 ページの『IF』	条件の評価に基づいて実行パスを選択する。
613 ページの『INCLUDE』	ソース・プログラムにコードまたは宣言を挿入する。
615 ページの『INSERT』	1 つまたは複数の行を表に挿入する。
625 ページの『ITERATE』	制御のフローがラベル付きループの最初に戻る。
627 ページの『LEAVE』	プログラム制御をループまたはコンパウンド・ステートメントの外側に移動させる。
629 ページの『LOCK TABLE』	並行処理による表の変更、または並行処理による表の使用のいずれかを禁止する。
631 ページの『LOOP』	ステートメント、またはステートメントのグループの実行を繰り返す。
633 ページの『MERGE』	ソース (表参照の結果) からのデータを使ってターゲット (表またはビュー) を更新する。
642 ページの『OPEN』	FETCH ステートメントが出された時に値を検索するのに使用するカーソルを準備する。
647 ページの『PREPARE』	SQL ステートメント (および任意指定パラメーター) を準備し、実行できるようにする。
657 ページの『REFRESH TABLE』	マテリアライズ照会表のデータをリフレッシュする。
659 ページの『RELEASE (接続)』	1 つまたは複数の接続を解放ペンディング状態にする。
661 ページの『RELEASE SAVEPOINT』	トランザクション内のセーブポイントを解放する。
662 ページの『RENAME』	既存の表の名前を変更する。
664 ページの『RENAME TABLESPACE』	既存の表スペースの名前を変更する。
666 ページの『REPEAT』	検索条件が真になるまでステートメントまたはステートメントのグループを実行する。
668 ページの『RESIGNAL』	エラーまたは警告条件を再通知するのに使用する。
671 ページの『RETURN』	ルーチンから戻るのに使用する。
673 ページの『REVOKE (データベース権限)』	データベース全体から権限を取り消す。
677 ページの『REVOKE (索引特権)』	所定の索引に対する CONTROL 特権を取り消す。
679 ページの『REVOKE (パッケージ特権)』	データベースの所定のパッケージから特権を取り消す。
682 ページの『REVOKE (ルーチン特権)』	ルーチン (関数、メソッド、またはプロシージャー) についての特権を取り消す。
686 ページの『REVOKE (スキーマ特権)』	スキーマの特権を取り消す。
688 ページの『REVOKE (シーケンス特権)』	シーケンスについての特権を取り消す。
691 ページの『REVOKE (サーバー特権)』	特定のデータ・ソースを照会するための特権を取り消す。
693 ページの『REVOKE (表スペース特権)』	指定した表スペースに対する USE 特権を取り消す。
695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』	所定の表、ビュー、またはニックネームから特権を取り消す。
701 ページの『ROLLBACK』	作業単位を終了し、その作業単位によって行われたデータベースの変更をバックアウトする。
704 ページの『SAVEPOINT』	トランザクション内にセーブポイントを設定する。

表 1. SQL ステートメント (続き)

SQL ステートメント	目的
708 ページの『SELECT INTO』	1 行だけの結果表を指定し、その値をホスト変数に割り当てる。
711 ページの『SET CONNECTION』	接続の状態を休止状態から現行状態に変更し、指定した位置を現行サーバーにする。
713 ページの『SET CURRENT DEFAULT TRANSFORM GROUP』	CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターの値を変更する。
715 ページの『SET CURRENT DEGREE』	CURRENT DEGREE 特殊レジスターの値を変更する。
717 ページの『SET CURRENT EXPLAIN MODE』	CURRENT EXPLAIN MODE 特殊レジスターの値を変更する。
720 ページの『SET CURRENT EXPLAIN SNAPSHOT』	CURRENT EXPLAIN SNAPSHOT 特殊レジスターの値を変更する。
723 ページの『SET CURRENT ISOLATION』	CURRENT ISOLATION 特殊レジスターの値を変更する。
724 ページの『SET CURRENT LOCK TIMEOUT』	CURRENT LOCK TIMEOUT 特殊レジスターの値を変更する。
726 ページの『SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION』	CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値を変更する。
728 ページの『SET CURRENT PACKAGE PATH』	CURRENT PACKAGE PATH 特殊レジスターの値を割り当てる。
732 ページの『SET CURRENT PACKAGESET』	パッケージ選択のためのスキーマ名を設定する。
734 ページの『SET CURRENT QUERY OPTIMIZATION』	CURRENT QUERY OPTIMIZATION 特殊レジスターの値を変更する。
737 ページの『SET CURRENT REFRESH AGE』	CURRENT REFRESH AGE 特殊レジスターの値を変更する。
739 ページの『SET ENCRYPTION PASSWORD』	暗号化のためのパスワードを設定する。
741 ページの『SET EVENT MONITOR STATE』	イベント・モニターを活動化または非活動化する。
743 ページの『SET INTEGRITY』	チェック・ペンディング状態を設定し、制約違反の有無についてデータを検査する。
760 ページの『SET PASSTHRU』	データ・ソースのネイティブ SQL を、対象となるデータ・ソースへ直接にサブミットするため、セッションをオープンする。
762 ページの『SET PATH』	CURRENT PATH 特殊レジスターの値を変更する。
764 ページの『SET SCHEMA』	CURRENT SCHEMA 特殊レジスターの値を変更する。
766 ページの『SET SERVER OPTION』	サーバーのオプション設定をセットする。
768 ページの『SET SESSION AUTHORIZATION』	SESSION USER 特殊レジスターの値を変更する。
771 ページの『SET 変数』	NEW 遷移変数に値を割り当てる。
776 ページの『SIGNAL』	エラーまたは警告条件を通知するのに使用する。
779 ページの『UPDATE』	表の 1 つまたは複数の行に含まれる 1 つまたは複数の列の値を更新する。
791 ページの『VALUES INTO』	1 行だけの結果表を指定し、その値をホスト変数に割り当てる。
793 ページの『WHENEVER』	SQL の戻りコードに基づいて実行するアクションを定義する。

表 1. SQL ステートメント (続き)

SQL ステートメント	目的
795 ページの『WHILE』	指定した条件が真である間、ステートメント、またはステートメントのグループの実行を繰り返す。

SQL ステートメントの呼び出し方法

SQL ステートメントは、実行可能と実行不能に分類されます。

*実行可能*ステートメント には、 4 つの呼び出し方法があります。それらは次のとおりです。

- アプリケーション・プログラムに組み込む。
- SQL プロシージャーに組み込む。
- 動的に準備して実行する。
- 対話式に発行する。

ステートメントによっては、これらのいくつかまたはすべての方式を使用することができます。 (REXX に組み込んだステートメントは、動的に準備され実行されます。)

*実行不能*ステートメント は、アプリケーション・プログラムに組み込む方式だけが可能です。

別の SQL ステートメント構成は、select ステートメントです。 *select* ステートメント には、以下のような 3 つの呼び出し方法があります。

- DECLARE CURSOR に組み込んで、 OPEN、FETCH および CLOSE によって暗黙的に実行する。(静的起動)
- 動的に準備し、DECLARE CURSOR で参照して、 OPEN、FETCH および CLOSE によって暗黙的に実行する。(動的起動)
- 対話式に発行する。

アプリケーション・プログラムへのステートメントの組み込み

SQL ステートメントは、プリコンパイラーにサブミットされるソース・プログラムに組み込むことができます。このようなステートメントは、プログラムに*組み込まれている* と言います。組み込みステートメントは、ホスト言語のステートメントが可能な位置であればそのプログラム内のどこにでも組み込むことができます。各組み込みステートメントの前には、キーワード EXEC SQL を付ける必要があります。

実行可能ステートメント

アプリケーション・プログラムに組み込まれた実行可能ステートメントは、ホスト言語ステートメントが実行されるたびに、そこに実行可能ステートメントが指定されていると同じ時点で実行されます。したがって、ループ内のステートメントは、ループが行われるたびに実行され、条件構文内のステートメントは、その条件が満たされた場合にのみ実行されます。

組み込まれたステートメントには、ホスト変数への参照を含むことができます。参照されるホスト変数は、以下のような 2 つの方法で使用することができます。

- 入力として使用する (ホスト変数の現在値がそのステートメントの実行に使用されます)。
- 出力として使用する (ホスト変数には、そのステートメントの実行結果として新しい値が割り当てられます)。

特に、式および述部の中のホスト変数に対する参照はすべて、変数の現在値により置き換えられます。つまり、変数は入力として使用されます。

すべての実行可能ステートメントの後で、必ず SQL 戻りコードのテストを行う必要があります。別の方法として、WHENEVER ステートメント (それ自体は実行不能) を使用して、組み込みステートメントの実行直後の制御の流れを変更することもできます。

データ操作言語 (DML) ステートメントで参照されるオブジェクトはすべて、ステートメントがデータベースにバインドされる時点で存在している必要があります。

実行不能ステートメント

組み込まれた実行不能ステートメントは、プリコンパイラーによってのみ処理されます。プリコンパイラーはステートメントにエラーを検出すると、それを報告します。このようなステートメントは、プログラムの実行時に処理されることはありません。したがって、このようなステートメントの後で SQL 戻りコードのテストを行ってはなりません。

SQL プロシージャーへのステートメントの組み込み

CREATE PROCEDURE ステートメントの SQL プロシージャー本体にステートメントを組み込むことができます。このようなステートメントは、 SQL プロシージャーに組み込まれているといいます。 SQL ステートメントの説明でホスト変数 が参照されるときはいつでも、ステートメントが SQL プロシージャーに組み込まれていれば *SQL 変数* を使用できます。

動的な準備と実行

アプリケーション・プログラムでは、ホスト変数に入った文字ストリングの形式の SQL ステートメントを動的に構築することができます。一般にステートメントは、プログラムが入手可能な何らかのデータから構築されます (たとえば、ワークステーションからの入力)。構築されたステートメント (select ステートメントではない) は、 (組み込み) PREPARE ステートメントによって準備され、 (組み込み) EXECUTE ステートメントによって実行することができます。あるいは、(組み込み) EXECUTE IMMEDIATE ステートメントを使用して、 1 つのステップでステートメントを準備して実行することもできます。

動的に準備されるステートメントには、ホスト変数への参照が含まれていてはなりません。パラメーター・マーカーは含めることができます。 (パラメーター・マーカーの規則に関しては、『PREPARE』 を参照してください。) 準備済みのステートメントが実行される時点で、パラメーター・マーカーは、実際には EXECUTE ステートメントで指定されたホスト変数の現行値に置き換えられます。一度準備したステートメントは、ホスト変数の他の値を用いて何回も実行することができます。パラメーター・マーカーは、EXECUTE IMMEDIATE ステートメントでは使用できません。

ステートメントが正しく実行されたか否かは、 EXECUTE (または EXECUTE IMMEDIATE) ステートメントの実行後の SQLCA への SQL 戻りコードの設定値によって示されます。前述のように、SQL 戻りコードは必ず検査する必要があります。詳しくは、 10 ページの『SQL 戻りコード』を参照してください。

select ステートメントの静的呼び出し

select ステートメントは、(実行不能) DECLARE CURSOR ステートメントの一部として含めることができます。このようなステートメントは、 (組み込み) OPEN ステートメントによってカーソルがオープンされるたびに実行されます。カーソルがオープンされた後で、一連の FETCH ステートメントを実行することにより、結果表を一度に 1 つの行ずつ取り出すことができます。

このように使用する場合、 select ステートメントにホスト変数への参照を含めることができます。これらの参照は、実際には、OPEN ステートメントを実行した時点での変数の値によって置き換えられます。

select ステートメントの動的呼び出し

アプリケーション・プログラムは、ホスト変数に入った文字ストリングの形式で、選択 (SELECT) ステートメントを動的に構築することができます。一般に、ステートメントはプログラムが入手可能な何らかのデータから構築されます (たとえば、ワークステーションから入手した照会)。このように構成されたステートメントは、(組み込み) PREPARE ステートメントによって実行の準備が行われ、 (実行不能) DECLARE CURSOR ステートメントによって参照されます。このようなステートメントは、 (組み込み) OPEN ステートメントによってカーソルがオープンされるたびに実行されます。カーソルがオープンされた後で、一連の FETCH ステートメントを実行することにより、結果表を一度に 1 つの行ずつ取り出すことができます。

このように使用する場合、 select ステートメントにホスト変数への参照を含めることはできません。パラメーター・マーカーは含めることができます。パラメーター・マーカーは、実際には、 OPEN ステートメントに指定されたホスト変数の値によって置き換えられます。

対話式呼び出し

ワークステーションから SQL ステートメントを入力する機能は、データベース・マネージャーのアーキテクチャーの一部です。この方法で入力されたステートメントは、「対話式に発行される」と呼ばれます。このようなステートメントは、アプリケーション・プログラムのコンテキストでのみ認識されるので、パラメーター・マーカーやホスト変数への参照を含まない実行可能ステートメントでなければなりません。

異なるホスト・システムで使用される SQL

SQL ステートメントの構文は、ホスト・システムの種類 (DB2 for z/OS、DB2 for iSeries、 DB2 Universal Database) によって微妙に異なります。アプリケーション内の SQL ステートメントが静的か動的かにかかわらず、別のデータベース・ホスト・システムにアクセスするアプリケーションの場合は、 SQL ステートメントとプリコンパイル/ BIND オプションが、アクセス先のデータベース・システムでサポートされるようにするのは重要なことです。

異なるホスト・システムでの SQL の使用についての詳細情報は、*SQL リファレンス* および *DB2 Universal Database for OS/390 and z/OS SQL Reference* を参照してください。

SQL 戻りコード

実行可能な SQL ステートメントを含むアプリケーション・プログラムは、SQLCODE または SQLSTATE の値のいずれかを使用して、 SQL ステートメントからの戻りコードを処理することができます。アプリケーションでこれらの値にアクセスするには、 2 つの方法があります。

- SQLCA と呼ばれる構造体を組み込む。 SQLCA には SQLCODE という名前の整変数と、 SQLSTATE という名前の文字ストリングが含まれています。 REXX では、SQLCA は自動的に提供されます。他の言語では、INCLUDE SQLCA ステートメントを使用することによって、 SQLCA を入手することができます。
- プリコンパイル・オプションとして LANGLEVEL SQL92E が指定されている場合は、プログラムの SQL 宣言セクションに SQLCODE または SQLSTATE という名前の変数を宣言することができます。これらの値がいずれも SQL 宣言セクションに宣言されていない場合は、プログラムの別のロケーションで SQLCODE という名の変数が宣言されているものと想定されます。 LANGLEVEL SQL92E を使用する場合は、プログラムに INCLUDE SQLCA ステートメントがあってはなりません。

SQLCODE

SQLCODE は、各 SQL ステートメントの実行後に、データベース・マネージャーによって設定されます。すべてのデータベース・マネージャーは、次のように ISO/ANSI SQL 標準規格に準拠しています。

- SQLCODE = 0 で SQLWARN0 がブランクの場合、実行は成功しました。
- SQLCODE = 100 の場合、“データが見つかりませんでした”。たとえば、カーソルが結果表の最後の行より後に設定されていたために、 FETCH ステートメントからデータが戻されませんでした。
- SQLCODE > 0 で、100 ではない場合、実行は警告付きで成功しました。
- SQLCODE = 0 で SQLWARN0 = 'W' の場合、実行は成功しましたが、1 つ以上の警告標識がセットされました。
- SQLCODE < 0 の場合、実行は不成功でした。

0 と 100 以外の SQLCODE の値の意味は、製品によって異なります。

SQLSTATE

SQLSTATE は、各 SQL ステートメントの実行後に、データベース・マネージャーによって設定されます。アプリケーション・プログラムは、SQLCODE ではなく、SQLSTATE をテストすることによって SQL ステートメントの実行を検査することができます。 SQLSTATE は、共通エラー条件に関する共通コードを示します。アプリケーション・プログラムが特定のエラーまたは特定クラスのエラーの有無をテストできます。コード体系は、IBM のどのデータベース・マネージャーでも同じであり、 ISO/ANSI SQL92 標準規格に基づいています。

SQL コメント

静的 SQL ステートメントには、ホスト言語または SQL のコメントを含めることができます。 SQL コメントは、2 つのハイフンによって示されます。

SQL のコメントを使用する際には、以下の規則が適用されます。

- 2 つのハイフンが同一行にあることが必要で、その間にスペースを入れることはできません。
- コメントは、スペースが有効な個所であればどこからでも開始できます (区切りトークンの中、または 'EXEC' と 'SQL' との間を除く)。
- コメントは、行の終わりで終了します。
- PREPARE または EXECUTE IMMEDIATE を使用して動的に準備されるステートメントの中には、コメントは許されません。
- COBOL では、2 つのハイフンの前にスペースを 1 つ入れる必要があります。

例: この例は、C プログラム中の SQL ステートメントにコメントを組み込む方法を示しています。

```
EXEC SQL
  CREATE VIEW PRJ_MAXPER          -- projects with most support personnel
    AS SELECT PROJNO, PROJNAME    -- number and name of project
      FROM PROJECT
      WHERE DEPTNO = 'E21'        -- systems support dept code
      AND PRSTAFF > 1;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『Select-statement』
- 551 ページの『EXECUTE』
- 642 ページの『OPEN』
- 647 ページの『PREPARE』
- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』

SQL 制御ステートメントについて

制御ステートメントとは、構造化プログラミング言語でプログラムを作成する方法と同様に構造化照会言語を使用できるようにする SQL ステートメントです。 SQL 制御ステートメントには、論理の流れを制御し、変数を宣言して設定し、警告や例外を処理する機能が備えられています。ある SQL 制御ステートメントに他のネストされた SQL ステートメントが組み込まれていることもあります。ルーチンの本体、トリガー、または動的コンパウンド・ステートメントで SQL 制御ステートメントを使用できます。

式やホスト変数を指定できるステートメント中のどこでも、 SQL パラメーターや SQL 変数を参照できます。 SQL ルーチン中にホスト変数を指定することはできません。ルーチン中のどこでも SQL パラメーターを参照でき、ルーチン名で修飾できます。 SQL 変数が宣言されているコンパウンド・ステートメント中のどこでも SQL 変数を参照でき、そのコンパウンド・ステートメントの先頭にラベル名を指定して SQL 変数を修飾できます。

SQL パラメーターと SQL 変数はすべて NULL 可能と見なされます。 SQL ルーチン中の SQL パラメーターや SQL 変数の名前を、そのルーチン中で参照されている表やビューの列名と同じ名前にすることもできます。この場合、名前を明示的に修飾して、その名前が列、SQL 変数、または SQL パラメーターのいずれであるかを示す必要があります。

名前を修飾しない場合は、以下の規則により、名前が列、SQL 変数、または SQL パラメーターのいずれであるかが示されます。

- SQL ルーチン本体で指定されている表やビューが、そのルーチンの作成時に既存の場合は、名前は最初に列名として検査されます。列として検出されない場合は、次に SQL 変数名または SQL パラメーター名として検査されます。
- 参照されている表やビューが、そのルーチンの作成時に既存でない場合は、名前は最初に SQL 変数名または SQL パラメーター名として検査されます。検出されない場合は、列として想定されます。

SQL ルーチン中の SQL パラメーターと SQL 変数の名前を、特定の SQL ステートメント中で使用されている ID 名と同じ名前にすることもできます。名前を修飾しない場合は、以下の規則により、名前が ID、SQL パラメーター、または SQL 変数のいずれであるかが示されます。

- SET PATH および SET SCHEMA ステートメント中では、名前は SQL パラメーター名または SQL 変数名として検査されます。 SQL 変数名または SQL パラメーター名として検出されない場合は、ID として使用されます。
- CONNECT ステートメント中では、名前は ID として使用されます。

ALLOCATE CURSOR

ALLOCATE CURSOR ステートメントは、結果セット・ロケーター変数で識別される結果セットにカーソルを割り当てます。結果セット・ロケーター変数については、ASSOCIATE LOCATORS ステートメントの説明を参照してください。

呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

必要ありません。

構文:

説明:

cursor-name
カーソルの名前を指定します。ソース SQL プロシージャーですでに宣言されているカーソルと同じ名前は使用しないでください (SQLSTATE 24502)。

CURSOR FOR RESULT SET *rs-locator-variable*

ホスト変数の規則に従って、ソース SQL プロシージャーで宣言されている結果セット・ロケーター変数の名前を指定します。 SQL 変数の宣言について詳しくは、
『コンパウンド・ステートメント (プロシージャー)』 を参照してください。

結果セット・ロケーター変数には、ASSOCIATE LOCATORS SQL ステートメントで戻された、有効な結果セット・ロケーター値を入れなければなりません (SQLSTATE 24501)。

規則:

- 割り当てカーソルを使用する際には、以下の規則が適用されます。
  - 割り当てカーソルは、 OPEN ステートメントによってオープンすることはできません (SQLSTATE 24502)。
  - 割り当てカーソルは、位置指定 UPDATE または DELETE ステートメントでは使用できません (SQLSTATE 42828)。
  - 割り当てカーソルは、CLOSE ステートメントによってクローズできます。割り当てカーソルをクローズすると、関連付けられたカーソルがクローズされます。
  - 各結果セットに割り当てられるカーソルは 1 つだけです。
- 割り当てカーソルは、ロールバック操作、暗黙的クローズ、または明示的クローズが行われるまで継続します。
- コミット操作を行うと、割り当てカーソルで、 WITH HOLD が定義されていないものが破棄されます。

- 割り当てカーソルを破棄すると、 SQL プロシージャー内の関連付けられたカーソルがクローズされます。

例:

以下の SQL プロシージャーの例では、カーソル C1 を定義し、結果セット・ロケーター変数 LOC1、および SQL プロシージャーによって戻される関連する結果セットに関連付けます。

```
ALLOCATE C1 CURSOR FOR RESULT SET LOC1;
```

関連資料:

- 140 ページの『コンパウンド SQL (プロシージャー)』
- 102 ページの『ASSOCIATE LOCATORS』

ALTER BUFFERPOOL

ALTER BUFFERPOOL ステートメントは、以下を行う場合に使用されます。

- すべてのパーティション、あるいは 1 つのパーティションのバッファー・プールのサイズを変更する
- 拡張ストレージの使用をオンまたはオフにする
- このバッファー・プール定義を新規のデータベース・パーティション・グループに追加する
- ブロック・ベース入出力用のバッファー・プールのブロック域を変更する

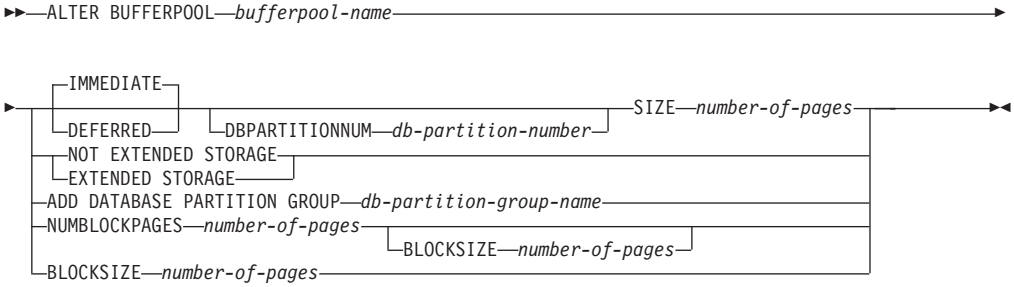
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文:



説明:

bufferpool-name
バッファー・プールの名前を指定します。これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。バッファー・プールは、カタログで記述されている必要があります。

DBPARTITIONNUM *db-partition-number*
そのバッファー・プールのサイズを変更するパーティションを指定します。パーティションは、そのバッファー・プールのデータベース・パーティション・グループのいずれかに入っている必要があります (SQLSTATE 42729)。この文節の指定がない場合、バッファー・プールが存在し、バッファー・プールのサイズにデフォルトのサイズを使用している (つまり、 CREATE BUFFERPOOL ステートメントの *except-on-db-partitions-clause* でサイズが指定されていない) すべてのパーティションで、バッファー・プールのサイズが変更されます。

SIZE *number-of-pages*
バッファー・プールのサイズをページ数で指定します。

IMMEDIATE
バッファー・プールのサイズが直ちに変更されます。メモリーを共用するデ

ータベース内に新規スペースを割り振るのに十分な予約済みのスペースがない場合 (SQLSTATE 01657)、このステートメントは DEFERRED で実行されます。

DEFERRED
データベースが再活動化される時に (すべてのアプリケーションがデータベースから切断される必要があります) 、バッファー・プールのサイズが変更されます。予約済みのメモリー・スペースは必要ありません。 DB2 が、活動状態にあるときにシステムから必要なメモリーを割り振ります。

NOT EXTENDED STORAGE
拡張ストレージが使用可能な場合でも、このバッファー・プールから排除されるページは拡張ストレージにはキャッシュされません。

EXTENDED STORAGE
拡張ストレージが使用可能な場合には、バッファー・プールから排除されたページの 2 次キャッシュとして使用できます。 (拡張ストレージは、データベース構成パラメーター NUM_ESTORE_SEGS と ESTORE_SEG_SIZE をゼロ以外の値に設定することによってオンになります。)

ADD DATABASE PARTITION GROUP *db-partition-group-name*
バッファー・プール定義が適用されるデータベース・パーティション・グループのリストに、このデータベース・パーティション・グループを追加します。バッファー・プールが定義されていないデータベース・パーティション・グループにあるパーティションについては、バッファー・プールに指定されているデフォルト・サイズを使用して、このパーティションにバッファー・プールが作成されます。 *db-partition-group-name* 内の表スペースは、このバッファー・プールを指定できます。データベース・パーティション・グループは、現在データベースに存在している必要があります (SQLSTATE 42704)。

NUMBLOCKPAGES *number-of-pages*
ブロック・ベース域に存在していなければならないページ数を指定します。ページ数は、バッファー・プールのページ数の 98% より小さくしなければなりません (SQLSTATE 54052)。値 0 を指定すると、ブロック入出力は不可になります。使用されている NUMBLOCKPAGES の実際の値は、BLOCKSIZE の倍数になります。

BLOCKSIZE *number-of-pages*
ブロック内のページ数を指定します。ブロック・サイズの値は、2 ～ 256 でなければなりません (SQLSTATE 54053)。デフォルト値は 32 です。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
- バッファー・プール・サイズのみが動的に (直ちに) に変更可能です。他のすべての変更は据え置かれ、データベースが再活動化された後にこれらの変更は効力を持つようになります。

- このステートメントが据え置かれて実行 (Deffered) される場合には、次のことが当てはまります。バッファー・プール定義はトランザクションで、コミット時にバッファー・プール定義に対する変更がカタログ表に反映されますが、実際のバッファー・プールに対する変更は、次回にデータベースが始動されるまでは有効になりません。それまではバッファー・プールの現行の属性が存在し、その間バッファー・プールには何の影響もありません。新しいデータベース・パーティション・グループの表スペースに作成された表は、デフォルトのバッファー・プールを使用します。 キーワードが適用される際のこのステートメントのデフォルトは、IMMEDIATE です。
- すべてのバッファー・プールの合計と、その他のデータベース・マネージャーやアプリケーションの要件に合うように、マシンに十分な実メモリーが必要です。
- 拡張ストレージを使用中のバッファー・プールは、ブロック・ベース入出力 (I/O) を使用するようには変更できません。バッファー・プールは、同時に拡張ストレージとブロック・ベース入出力を使用するようには変更できません。

ALTER DATABASE PARTITION GROUP

ALTER DATABASE PARTITION GROUP ステートメントは、以下の目的で使用されます。

- データベース・パーティション・グループに 1 つまたは複数のパーティションを追加する
- データベース・パーティション・グループから 1 つまたは複数のパーティションをドロップする

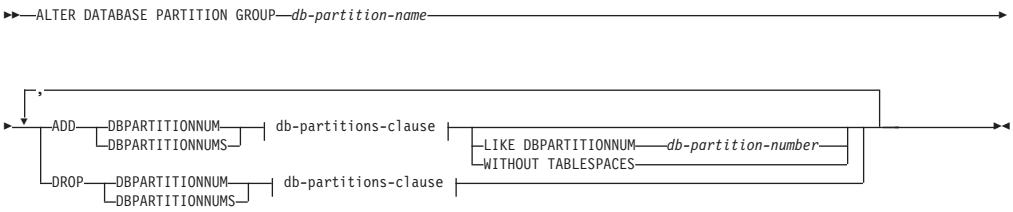
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

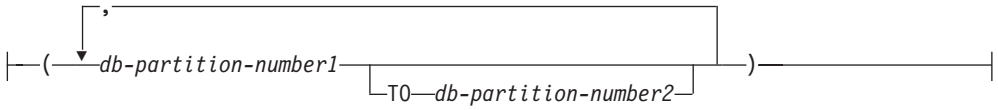
許可:

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文:



db-partitions-clause:



説明:

db-partition-name
データベース・パーティション・グループの名前を指定します。これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。データベース・パーティション・グループはカタログに記述されている必要があります。 IBMCATGROUP および IBMTEMPGROUP は指定できません (SQLSTATE 42832)。

ADD DBPARTITIONNUM
データベース・パーティション・グループに特定の 1 つまたは複数のパーティションを追加することを指定します。 DBPARTITIONNUMS は DBPARTITIONNUM の同義語です。指定するパーティションは、データベース・パーティション・グループにすでに定義済みであってはなりません (SQLSTATE 42728)。

DROP DBPARTITIONNUM
データベース・パーティション・グループから特定の 1 つまたは複数のパーティションをドロップすることを指定します。 DBPARTITIONNUMS は

DBPARTITIONNUM の同義語です。指定するパーティションは、データベース・パーティション・グループにすでに定義されている必要があります (SQLSTATE 42729)。

db-partitions-clause
追加またはドロップする 1 つまたは複数のパーティションを指定します。

db-partition-number1
特定のパーティション番号を指定します。

TO *db-partition-number2*
パーティション番号の範囲を指定します。 *db-partition-number2* の値は、*db-partition-number1* の値よりも大きいか等しい値でなければなりません (SQLSTATE 428A9)。

LIKE DBPARTITIONNUM *db-partition-number*
データベース・パーティション・グループの既存の表スペースのコンテナーが、指定した *db-partition-number* のコンテナーと同じであることを指定します。指定するパーティションは、このステートメントの前にデータベース・パーティション・グループに存在しており、同じステートメントの DROP DBPARTITIONNUM 文節に含まれていないパーティションである必要があります。

WITHOUT TABLESPACES
新たに追加されるパーティションに、デフォルトの表スペースを作成しないことを指定します。 FOR DBPARTITIONNUM 文節を用いた ALTER TABLESPACE ステートメントを使用して、このデータベース・パーティション・グループに対して定義される表スペースで使用するコンテナーを定義する必要があります。このオプションの指定がない場合、そのデータベース・パーティション・グループに対して表スペースが定義されるたびに、新たに追加されるパーティションにデフォルトのコンテナーが指定されます。

規則:

- 番号によって指定するそれぞれのパーティションは、 `db2nodes.cfg` ファイルに定義されていなければなりません (SQLSTATE 42729)。
- ON DBPARTITIONNUMS 文節にリストされる *db-partition-number* は、それぞれユニークなパーティションに対する番号でなければなりません (SQLSTATE 42728)。
- 有効なパーティション番号は、0 ～ 999 (0 と 999 を含む) です (SQLSTATE 42729)。
- 1 つのパーティションを ADD と DROP の両方の文節に指定することはできません (SQLSTATE 42728)。
- データベース・パーティション・グループには少なくとも 1 つのパーティションが残っている必要があります。最後のパーティションをデータベース・パーティション・グループからドロップすることはできません (SQLSTATE 428C0)。
- パーティションを追加する際に、 LIKE DBPARTITIONNUM 文節も WITHOUT TABLESPACES 文節も指定されていない場合、デフォルト解釈により、データベース・パーティション・グループの既存のパーティションの最も小さいパーティション番号 (ここでは *2* とします) が使用され、 LIKE DBPARTITIONNUM *2* が指定された場合と同様の処理が行われます。既存のパーティションをデフォル

ト値として使用する場合、パーティションではデータベース・パーティション・グループ内のすべての表スペースに対してコンテナーが定義されている必要があります (SYSCAT.DBPARTITIONGROUPDEF の列 IN_USE が 'T' でない)。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
    - DBPARTITIONNUMS の代わりに NODES を指定できます。
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
- パーティションがデータベース・パーティション・グループに追加されると、そのパーティションに対するカタログ項目が作成されます (SYSCAT.DBPARTITIONGROUPDEF を参照)。以下のいずれかの場合には、パーティション・マップは直ちに変され、新しいパーティションが、そのパーティションがパーティション・マップにあることを示す標識 (IN_USE) を伴って組み込まれます。
  - データベース・パーティション・グループに表スペースが定義されていない、または
  - データベース・パーティション・グループに定義されている表スペースに表が定義されておらず、 WITHOUT TABLESPACES 文節が指定されていない

  以下のいずれかの場合は、パーティション・マップは変更されず、標識 (IN_USE) はそのパーティションがパーティション・マップに組み込まれていないことを示すように設定されます。
  - データベース・パーティション・グループの表スペースに表が存在する、または
  - データベース・パーティション・グループに表スペースが存在し、WITHOUT TABLESPACES 文節が指定された

  パーティション・マップを変更するには、REDISTRIBUTE DATABASE PARTITION GROUP コマンドを使用する必要があります。このコマンドは、任意のデータを再分散し、パーティション・マップを変更し、標識を変更します。WITHOUT TABLESPACES 文節が指定された場合は、データを再分散する前に表スペース・コンテナーを追加する必要があります。
- パーティションがデータベース・パーティション・グループからドロップされると、そのパーティションのカタログ項目 (SYSCAT.DBPARTITIONGROUPDEF を参照) が更新されます。データベース・パーティション・グループに定義された表スペースに表が定義されていない場合、パーティション・マップが直ちに変更され、ドロップされたパーティションを除外し、データベース・パーティション・グループのそのパーティションに関する項目がドロップされます。表が存在する場合は、パーティション・マップは変更されず、標識 (IN_USE) はそのパーティションがドロップを待機していることを示すように設定されます。REDISTRIBUTE DATABASE PARTITION GROUP コマンドは、データを再分散し、データベース・パーティション・グループからそのパーティションに関する項目をドロップする場合に、使用しなければなりません。

例:

0、1、2、5、7、および 8 というパーティションを持つ、 6 つのパーティションのデータベースがあると想定します。これに、パーティション番号 3 と 6 の 2 つのパーティションをシステムに追加します。

- MAXGROUP という名前のデータベース・パーティション・グループに、パーティション 3 と 6 を追加し、パーティション 2 と同種の表スペース・コンテナーを設定するとします。必要なステートメントは以下のようになります。

```
ALTER DATABASE PARTITION GROUP MAXGROUP
  ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- パーティション 1 をドロップし、パーティション 6 をデータベース・パーティション・グループ MEDGROUP に追加するとします。 ALTER TABLESPACE を使用して、パーティション 6 に対して別個に表スペース・コンテナーを定義します。必要なステートメントは以下のようになります。

```
ALTER DATABASE PARTITION GROUP MEDGROUP
  ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
  DROP DBPARTITIONNUM(1)
```

関連概念:

- *SQL リファレンス 第 1 巻* の『複数のパーティションにわたるデータ・パーティション』

ALTER FUNCTION

ALTER FUNCTION ステートメントは、既存の関数のプロパティーを変更します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- 関数のスキーマに対する ALTERIN 特権
- SYSCAT.ROUTINES の DEFINER 列に記録されているその関数の定義者

関数の EXTERNAL NAME を変更するには、ステートメントの許可 ID の特権に、以下の特権の少なくとも 1 つが含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限

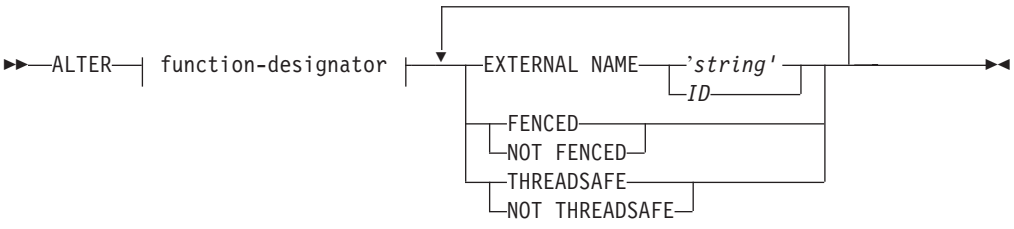
fenced でない関数を変更するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限

fenced である関数を変更するには、さらに別の権限や特権は必要ありません。

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:



説明:

function-designator
変更される関数を一意的に識別します。詳しくは、共通の構文エレメント x ページの『共通の構文エレメント』を参照してください。

EXTERNAL NAME *'string'* **または** *identifier*
関数をインプリメントするユーザー作成コードの名前を指定します。このオプションは、外部関数を変更する際にのみ指定できます (SQLSTATE 42849)。

FENCED または **NOT FENCED**
関数をデータベース・マネージャーのオペレーティング環境のプロセスまたはアドレス・スペースで実行しても安全か (NOT FENCED)、そうでないか (FENCED) を指定します。多くの関数は、FENCED または NOT FENCED のどちらかで実行するように選択することができます。

関数が FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を fenced して、その関数からアクセスされないようにします。一般に、FENCED として実行される関数は、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
適切にコード化、検討、および検査されていない関数に NOT FENCED を使用すると、 DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ユーザー定義関数が使用される場合には、完全な保全性を確保できません。

NOT THREADSAFE を宣言した関数は、NOT FENCED には変更できません (SQLSTATE 42613)。

関数が定義済みの AS LOCATOR の任意のパラメーターを有していて、 NO SQL オプションが指定されていた場合には、この関数は FENCED には変更できません (SQLSTATE 42613)。

| このオプションは LANGUAGE OLE 関数、OLEDB 関数、または CLR 関数を
| 変更できません (SQLSTATE 42849)。

THREADSAFE または **NOT THREADSAFE**
関数を他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

関数が OLE および OLEDB 以外の LANGUAGE で定義される場合:

- 関数が THREADSAFE に定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスで関数を呼び出すことができます。一般に、スレッド・セーフにするには、関数はどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED 関数の両方が THREADSAFE になることが可能です。
- 関数が NOT THREADSAFE に定義される場合には、データベース・マネージャーは関数を他のルーチンと同じプロセスに決して呼び出しません。 fenced された関数だけが、NOT THREADSAFE になり得ます (SQLSTATE 42613)。

このオプションは LANGUAGE OLE 関数または OLEDB 関数を変更できません (SQLSTATE 42849)。

注:

- SYSIBM、SYSFUN、または SYSPROC スキーマの関数は変更できません (SQLSTATE 42832)。
- LANGUAGE SQL で宣言した関数、ソース関数、またはテンプレート関数は変更できません (SQLSTATE 42917)。

例:

関数 MAIL() は完全にテストされました。パフォーマンスを向上させるために、関数が fenced でないように変更します。

```
ALTER FUNCTION MAIL() NOT FENCED
```

関連資料:

- 245 ページの『CREATE FUNCTION (OLE DB 外部表)』
- 200 ページの『CREATE FUNCTION (外部スカラー)』
- 226 ページの『CREATE FUNCTION (外部表)』
- x ページの『共通の構文エレメント』

ALTER METHOD

ALTER METHOD ステートメントは、メソッドに関連付けたメソッド本体を変更して、既存のメソッドを変更します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

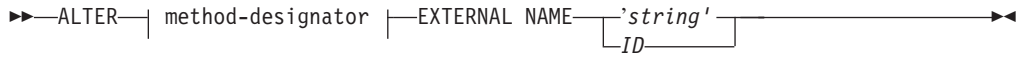
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限、および以下の少なくとも 1 つ。
  - タイプのスキーマに対する ALTERIN 特権
  - SYSCAT.DATATYPES の DEFINER 列に記録されているそのタイプの定義者

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:



説明:

method-designator
変更されるメソッドを一意的に識別します。詳しくは、共通の構文エレメント x ページの『共通の構文エレメント』を参照してください。

EXTERNAL NAME *'string'* **または** *identifier*
メソッドをインプリメントするユーザー作成コードの名前を指定します。このオプションは、外部メソッドを変更する際にのみ指定できます (SQLSTATE 42849)。

注:

- SYSIBM、SYSFUN、または SYSPROC スキーマのメソッドは変更できません (SQLSTATE 42832)。
- LANGUAGE SQL として宣言されたメソッドは変更できません (SQLSTATE 42917)。
- LANGUAGE CLR として宣言されたメソッドは変更できません (SQLSTATE 42849)。
- 指定するメソッドは、変更する前に本体を持っている必要があります (SQLSTATE 42704)。

例:

newaddresslib ライブラリーを使用するように、構造化タイプ ADDRESS_T の DISTANCE() メソッドを変更します。

```
ALTER METHOD DISTANCE()
  FOR TYPE ADDRESS_T
  EXTERNAL NAME 'newaddresslib!distance2'
```

関連資料:

- 293 ページの『CREATE METHOD』
- x ページの『共通の構文エレメント』

ALTER NICKNAME

ALTER NICKNAME ステートメントは、データ・ソース・オブジェクト (表、ビュー、またはファイルなど) に関連したニックネーム情報を変更します。このステートメントは、次のようにして、フェデレーテッド・データベースに保管されている情報を変更します。

- データ・ソース・オブジェクトの列のローカル列名を変更する
- データ・ソース・オブジェクトの列のローカル・データ・タイプを変更する
- ニックネーム・オプションおよび列オプションを追加、設定、またはドロップする
- 主キーを追加またはドロップする
- 1 つ以上のユニーク制約、参照制約、またはチェック制約を追加またはドロップする
- 1 つ以上の参照制約属性またはチェック制約属性を変更する

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

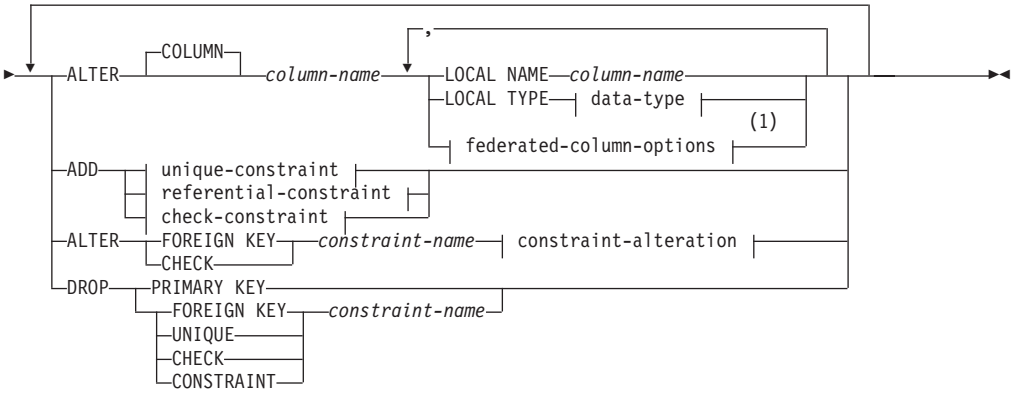
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

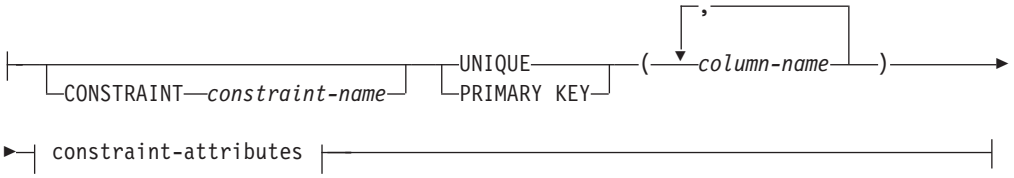
- ステートメントで指定したニックネームに対する ALTER 特権
- ステートメントで指定したニックネームに対する CONTROL 特権
- スキーマに対する ALTERIN 特権 (ニックネームのスキーマ名が存在する場合)。
- SYSCAT.TABLES カタログ・ビューの DEFINER 列に記録されているそのニックネームの定義者
- SYSADM または DBADM 権限

構文:

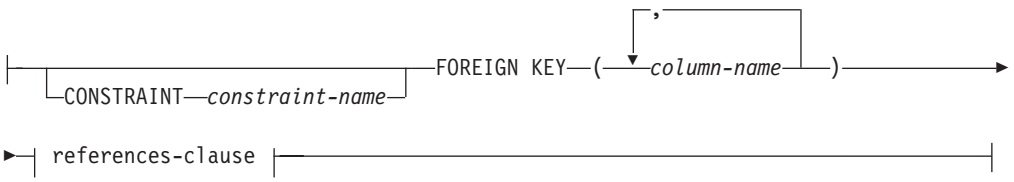
ALTER NICKNAME



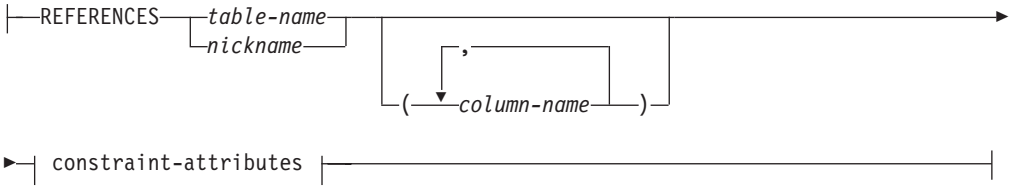
unique-constraint:



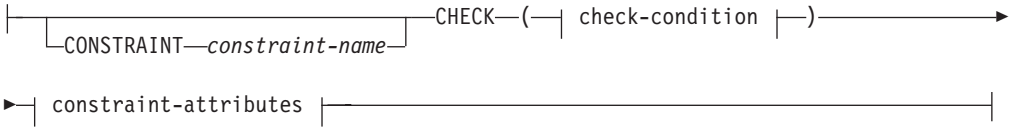
referential-constraint:



references-clause:

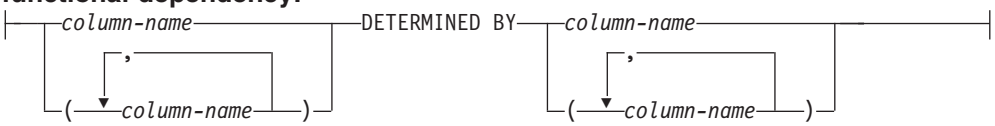


check-constraint:

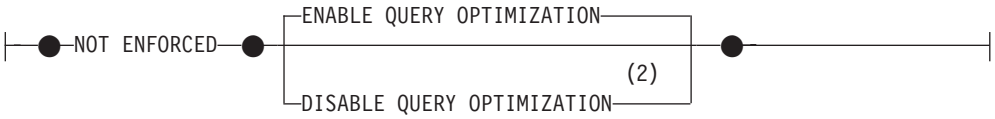


check-condition:

functional-dependency:

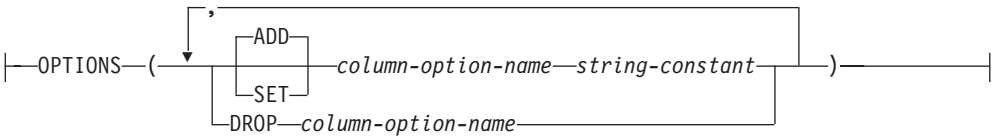


constraint-attributes:

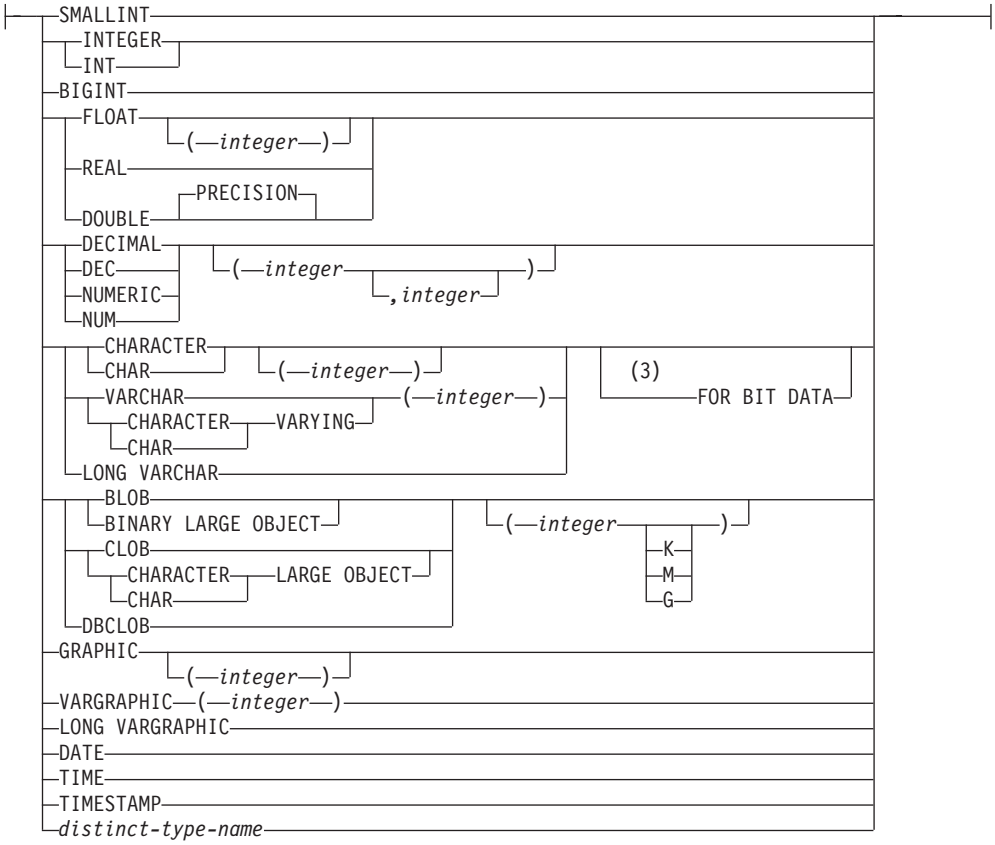


constraint-alteration:

federated-column-options:



data-type:



注:

1 LOCAL NAME パラメーターまたは LOCAL TYPE パラメーター、あるいは

その両方にフェデレーテッド列オプション (federated-column-options) 文節を指定する必要がある場合、この federated-column-options 文節は最後に指定しなければなりません。

2 DISABLE QUERY OPTIMIZATION はユニーク制約または主キー制約をサポートしていません。

3 FOR BIT DATA 文節とその後に続く他の列制約とは、任意の順序で指定できます。

説明:

nickname
変更される列を含むデータ・ソースオブジェクト (表、ビュー、またはファイルなど) のニックネームを指定します。ニックネームはカタログに記述されている必要があります。

ALTER COLUMN *column-name*
変更する列を指定します。 *column-name* は、フェデレーテッド・サーバーでのデータ・ソースにある表またはビューの列の、現在の名前です。 *column-name* は、ニックネームの既存の列を指定するものでなければなりません (SQLSTATE 42703)。同一の ALTER NICKNAME ステートメントで、同じ列名を複数回は参照できません (SQLSTATE 42711)。

LOCAL NAME *column-name*
変更された列を参照するフェデレーテッド・サーバーによって指定される新しい名前は *column-name* です。新しい名前を修飾したり、ニックネームの複数の列に対して同じ名前を使用することはできません (SQLSTATE 42711)。

LOCAL TYPE *data-type*
変更する列のデータ・タイプをマップする、新しいローカル・データ・タイプを指定します。新しいタイプは *data-type* に示されます。

一部のラッパーは、SQL データ・タイプのサブセットのみをサポートします。特定のデータ・タイプの説明は、『CREATE TABLE』 ステートメントの説明を参照してください。

OPTIONS
COLUMN キーワードの後に指定した列のどの列オプションを追加、設定、またはドロップするか指定します。

ADD
列オプションを追加します。

SET
列オプションの設定を変更します。

column-option-name
追加または設定する列オプションを指定します。

string-constant
column-option-name の設定を、文字ストリング定数として指定します。

DROP *column-option-name*
列オプションをドロップします。

ADD *unique-constraint*
ユニーク制約を定義します。 『CREATE NICKNAME』 ステートメントの説明を参照してください。

ADD *referential-constraint*
参照制約を定義します。 『CREATE NICKNAME』 ステートメントの説明を参照してください。

ADD *check-constraint*
チェック制約を定義します。 『CREATE NICKNAME』 ステートメントの説明を参照してください。

ALTER FOREIGN KEY *constraint-name*
参照制約 *constraint-name* の制約属性を変更します。 『CREATE NICKNAME』 ステートメントの説明を参照してください。 *constraint-name* は既存の参照制約を指定する必要があります (SQLSTATE 42704)。

ALTER CHECK *constraint-name*
チェック制約 *constraint-name* の制約属性を変更します。 *constraint-name* には既存のチェック制約を指定する必要があります (SQLSTATE 42704)。

constraint-alteration
参照制約またはチェック制約に関連付けられた属性の変更のオプションを指定します。

ENABLE QUERY OPTIMIZATION
適切な状況下では、制約を照会最適化に使用することができます。

DISABLE QUERY OPTIMIZATION
制約を照会の最適化に使用できません。

DROP PRIMARY KEY
主キーの定義、およびその主キーに従属するすべての参照制約をドロップします。ニックネームには主キーがなければなりません。

DROP FOREIGN KEY *constraint-name*
制約名が *constraint-name* の参照制約をドロップします。 *constraint-name* には、ニックネームに定義されている既存の参照制約を指定する必要があります。

DROP UNIQUE *constraint-name*
ユニーク制約 *constraint-name* の定義、およびこのユニーク制約に従属するすべての参照制約をドロップします。 *constraint-name* には、既存のユニーク制約を指定する必要があります。

DROP CHECK *constraint-name*
制約名が *constraint-name* のチェック制約をドロップします。 *constraint-name* には、ニックネームに定義されている既存のチェック制約を指定する必要があります。

DROP CONSTRAINT *constraint-name*
制約名が *constraint-name* の制約をドロップします。 *constraint-name* には、ニックネームに定義されている既存のチェック制約、参照制約、主キー、またはユニーク制約のいずれかを指定する必要があります。

規則:

- ニックネームがビュー、SQL メソッド、または SQL 関数に使用されている場合、 またはそれらに情報制約が定義されている場合、 そのニックネーム内の列のローカル名やデータ・タイプを変更するために、 ALTER NICKNAME ステートメントを使用することはできません (SQLSTATE 42893)。しかし、列オプション、ニックネーム・オプション、または通知制約を追加、設定、またはドロップするときには、このステートメントを使えます。
- ニックネームがマテリアライズ照会表定義によって参照されている場合、ローカル名、データ・タイプ、列オプション、またはニックネーム・オプションを変更するために、 ALTER NICKNAME ステートメントを使用することはできません (SQLSTATE 42893)。しかし、通知制約を追加、変更、またはドロップするときには、このステートメントを使えます。
- 列オプションは、同じ ALTER NICKNAME ステートメントに複数回指定することはできません (SQLSTATE 42853)。列オプションを使用可能にする、リセットする、あるいはドロップする場合、使用中の他の列オプションには影響はありません。
- リレーショナル・ニックネームの場合、所定の作業単位 (UOW) 内の ALTER NICKNAME ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - このステートメントで参照されているニックネームには、同じ UOW 内でオープンされているカーソルがある。
  - このステートメントで参照されているニックネームに対して、同じ UOW 内ですでに INSERT、DELETE、または UPDATE ステートメントのいずれかが出されている。
- 非リレーショナル・ニックネームの場合、所定の作業単位 (UOW) 内の ALTER NICKNAME ステートメントは、以下の条件の下では処理できません (SQLSTATE 55007)。
  - このステートメントで参照されているニックネームには、同じ UOW 内でオープンされているカーソルがある。
  - このステートメントで参照されているニックネームは、同じ UOW 内の SELECT ステートメントですでに参照されている。
  - このステートメントで参照されているニックネームに対して、同じ UOW 内ですでに INSERT、DELETE、または UPDATE ステートメントのいずれかが出されている。

注:

- ALTER NICKNAME ステートメントを使用してニックネームの列のローカル名を変更する場合、その列に対する照会ではその名前を新しい名前で参照する必要があります。
- 列のデータ・タイプのローカル仕様を変更すると、データベース・マネージャーは、その列に関して収集されたすべての統計 (HIGH2KEY、LOW2KEY、など) を無効にします。

例:

例 1: ニックネーム NICK1 は、T1 という DB2 UDB for iSeries 表を参照します。また、COL1 はこの表の最初の列である C1 を示すローカル名です。 C1 のローカル名を COL1 から NEWCOL に変更します。

```
ALTER NICKNAME NICK1
  ALTER COLUMN COL1
  LOCAL NAME NEWCOL
```

例 2: ニックネーム EMPLOYEE は、EMP という DB2 UDB for z/OS および OS/390 表を参照します。また、SALARY はこの表の列の 1 つである、EMP_SAL を示すローカル名です。列のデータ・タイプ FLOAT は、ローカル・データ・タイプ DOUBLE にマップされます。 FLOAT が DECIMAL (10, 5) へマップされるように、マッピングを変更します。

```
ALTER NICKNAME EMPLOYEE
  ALTER COLUMN SALARY
  LOCAL TYPE DECIMAL(10,5)
```

例 3: Oracle 表において、データ・タイプが VARCHAR の列には、後書きブランクがないことを示します。この表のニックネームは NICK2 で、この列のローカル名は COL1 です。

```
ALTER NICKNAME NICK2
  ALTER COLUMN COL1
  OPTIONS (ADD VARCHAR_NO_TRAILING_BLANKS 'Y')
```

例 4: ニックネーム DRUGDATA1 の表構造化ファイル drugdata1.txt の完全修飾パスを変更します。 FILE_PATH ニックネーム・オプションを使用して、パスを現行値 '/user/pat/drugdata1.txt' から '/usr/kelly/data/drugdata1.txt' に変更します。

```
ALTER NICKNAME DRUGDATA1
  OPTIONS (SET FILE_PATH '/usr/kelly/data/drugdata1.txt')
```

関連タスク:

- *フェデレーテッド・システム・ガイド* の『ニックネームの変更』

関連資料:

- 299 ページの『CREATE NICKNAME』
- *フェデレーテッド・システム・ガイド* の『フェデレーテッド・システムのニックネーム列オプション』

ALTER PROCEDURE

ALTER PROCEDURE ステートメントは、プロシージャーのプロパティーを変更して、既存のプロシージャーを変更します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- プロシージャーのスキーマに対する ALTERIN 特権
- SYSCAT.ROUTINES の DEFINER 列に記録されているそのプロシージャーの定義者

プロシージャーの EXTERNAL NAME を変更するには、ステートメントの許可 ID の特権に、以下の特権の少なくとも 1 つが含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限

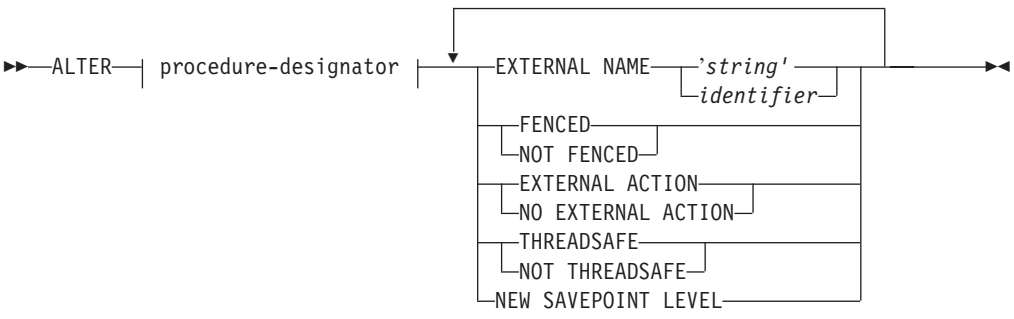
fenced でないようにプロシージャーを変更するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限

fenced であるようにプロシージャーを変更するには、さらに別の権限や特権は必要ありません。

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:



説明:

procedure-designator
変更されるプロシージャーを一意的に識別します。詳しくは、共通の構文エレメント x ページの『共通の構文エレメント』を参照してください。

EXTERNAL NAME *'string'* **または** *identifier*
プロシージャーをインプリメントするユーザー作成コードの名前を指定します。このオプションは、外部プロシージャーを変更する際にのみ指定できます (SQLSTATE 42849)。 LANGUAGE SQL として宣言されたプロシージャー内では EXTERNAL NAME 文節を変更できません (SQLSTATE 42917)。

FENCED または **NOT FENCED**
プロシージャーをデータベース・マネージャーのオペレーティング環境のプロセスまたはアドレス・スペースで実行しても安全か (NOT FENCED)、そうでないか (FENCED) を指定します。多くのプロシージャーは、 FENCED として実行するか NOT FENCED として実行するかの選択が可能です。

プロシージャーが FENCED に変更されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を fenced して、そのプロシージャーからアクセスされないようにします。一般に、FENCED として実行されるプロシージャーは、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
適切にコード化、検討、および検査されていないプロシージャーに NOT FENCED を使用すると、 DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ストアード・プロシージャーが使用される場合には、完全な保全性を確保できません。

このオプションは、外部プロシージャーを変更する際にのみ指定できます (SQLSTATE 42849)。

NOT THREADSAFE を宣言したプロシージャーは、NOT FENCED には変更できません (SQLSTATE 42613)。

プロシージャーが AS LOCATOR を定義した任意のパラメーターを有していて、 NO SQL オプションも指定されている場合には、このプロシージャーは FENCED には変更できません (SQLSTATE 42613)。

LANGUAGE OLE プロシージャーまたは CLR プロシージャーでは、このオプションを変更できません (SQLSTATE 42849)。

LANGUAGE SQL として宣言されたプロシージャー内では FENCED または NOT FENCED 文節を変更できません (SQLSTATE 42917)。

EXTERNAL ACTION または **NO EXTERNAL ACTION**
プロシージャーが、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを取るか (EXTERNAL ACTION)、または取らないか (NO EXTERNAL ACTION) を指定します。 NO EXTERNAL ACTION を指定した場合、プロシージャーが外部に影響を与えないことを前提とした最適化を、システムは使用できます。

THREADSAFE または **NOT THREADSAFE**
プロシージャーを他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

プロシージャーが OLE 以外の LANGUAGE で定義される場合:

- プロシージャーが THREADSAFE として定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスにプロシージャーを呼び出すことができます。一般に、スレッド・セーフになるには、プロシージャーはどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED プロシージャーの両方を THREADSAFE にすることができます。
- プロシージャーが NOT THREADSAFE に定義される場合には、データベース・マネージャーは他のルーチンと同じプロセスにプロシージャーを決して呼び出しません。 fenced されたプロシージャーだけが、NOT THREADSAFE になり得ます (SQLSTATE 42613)。

このオプションは、外部プロシージャーを変更する際にのみ指定できます (SQLSTATE 42849)。

このオプションは LANGUAGE OLE プロシージャーを変更できません (SQLSTATE 42849)。

LANGUAGE SQL として宣言されたプロシージャー内では、 THREADSAFE または NOT THREADSAFE 文節は変更できません (SQLSTATE 42917)。

NEW SAVEPOINT LEVEL

新規セーブポイント・レベルをプロシージャーに対して作成することを指定します。セーブポイント・レベルは、任意のセーブポイント関連ステートメントの参照範囲と、セーブポイント名の比較および参照に使用されるネーム・スペースを参照します。

プロシージャーのプロシージャー・レベルは NEW SAVEPOINT LEVEL にのみ変更可能です。

規則:

- SYSIBM、SYSFUN、または SYSPROC スキーマ (SQLSTATE 42832) のプロシージャーは変更できません。

例:

プロシージャー PARTS_ON_HAND() が fenced でないように変更します。

```
ALTER PROCEDURE PARTS_ON_HAND() NOT FENCED
```

関連資料:

- 311 ページの『CREATE PROCEDURE』
- x ページの『共通の構文エレメント』

ALTER SEQUENCE

ALTER SEQUENCE ステートメントを使用して、シーケンスを以下のように変更できます。

- シーケンスを再始動する
- 将来のシーケンス値の間の増分を変更する
- 最小値または最大値を設定または除去する
- キャッシュ済みシーケンス番号の数を変更する
- シーケンスが循環するかどうかを決定する属性を変更する
- 要求の順序でシーケンス番号が生成されるかどうかを変更する

呼び出し:

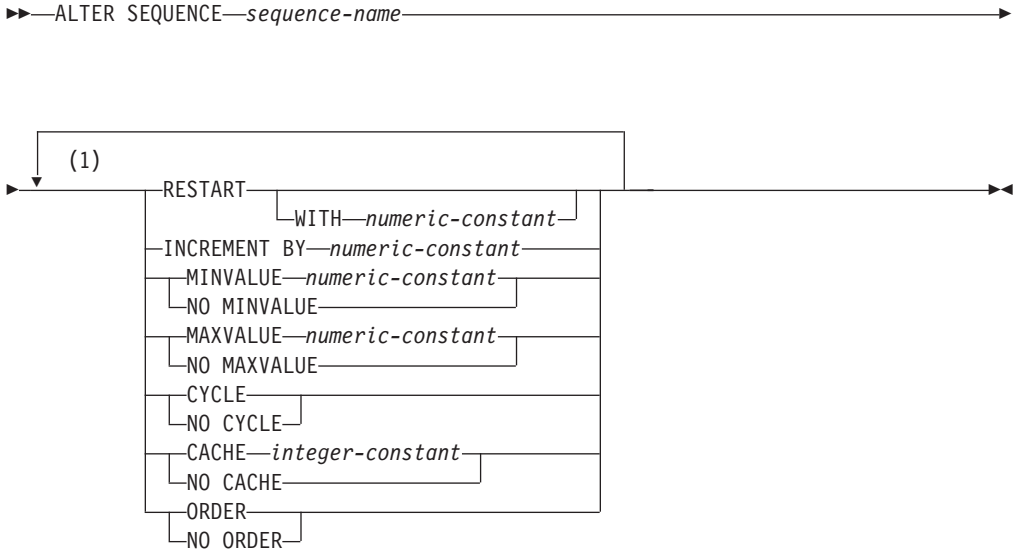
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 変更するシーケンスに対する ALTER 特権
- 暗黙的または明示的に指定されているスキーマの ALTERIN 特権
- SYSADM または DBADM 権限

構文:



注:

1 同じ文節を複数回指定することはできません。

説明:

sequence-name
変更するシーケンスを識別します。この名前 (暗黙的または明示的スキーマ修飾

子を含む) は、現行のサーバーに存在するシーケンスを固有に識別しなければなりません。この名前が示すシーケンスが、明示的または暗黙的に指定されたスキーマに存在しない場合、エラー (SQLSTATE 42704) が戻されます。
sequence-name には、システムが IDENTITY 列に対して生成したシーケンスを指定することはできません (SQLSTATE 428FB)。

RESTART
シーケンスを再始動します。 *numeric-constant* が指定されていない場合、シーケンスは、そのシーケンスを作成した CREATE SEQUENCE ステートメントに開始値として暗黙的または明示的に指定されている値で再始動されます。

WITH *numeric-constant*
指定した値でシーケンスを再始動します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815)。

INCREMENT BY *numeric-constant*
連続したシーケンス値のインターバルを指定します。この値として、長精度整数定数の値を超えない範囲で (SQLSTATE 42820)、シーケンスに関連したデータ・タイプの列に割り当てることのできる任意の正または負の値を指定できます (SQLSTATE 42815)。ただし、小数点の右側に非ゼロの数字が存在してはなりません (SQLSTATE 428FA)。

この値が負の場合、これは降順シーケンスです。この値が 0 の場合、または正の場合は、 ALTER ステートメント以降は昇順になります。

MINVALUE または **NO MINVALUE**
降順シーケンスが値の生成を循環または停止する最小値、あるいは最大値に達した後、昇順シーケンスが循環する最小値を指定します。

MINVALUE *numeric-constant*
最小値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815) が、最大値以下でなければなりません (SQLSTATE 42815)。

NO MINVALUE
昇順シーケンスの場合、値は元の開始値です。降順シーケンスの場合、シーケンスに関連するデータ・タイプの最小値です。

MAXVALUE または **NO MAXVALUE**
昇順シーケンスが値の生成を循環または停止する最大値、あるいは最小値に達した後、降順シーケンスが循環する最大値を指定します。

MAXVALUE *numeric-constant*
最大値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815) が、最小値以上でなければなりません (SQLSTATE 42815)。

NO MAXVALUE
昇順シーケンスの場合、値はシーケンスに関連するデータ・タイプの最大値です。降順シーケンスの場合、値は最初の開始値です。

CYCLE または **NO CYCLE**

その最大値または最小値に達した後、シーケンスが値の生成を続行するかどうかを指定します。シーケンスが境界に達するのは、次の値が境界条件を正確に満たしたとき、またはその値を超えたときです。

CYCLE

最大値または最小値に達した後、このシーケンスについて値の生成を続行することを指定します。このオプションが使用されると、昇順シーケンスが最大値に達した後、その最小値が生成されます。降順シーケンスが最小値に達した後、その最大値が生成されます。シーケンスの最大値および最小値は、循環に使用される範囲を決定します。

CYCLE が有効な場合、DB2 が重複するシーケンス値を生成する場合があります。

NO CYCLE

シーケンスの最大値または最小値に達した後、そのシーケンスについて値は生成されないことを指定します。

CACHE または **NO CACHE**

高速アクセスのため、事前割り振り値のいくつかをメモリーに保管するかどうかを指定します。これはパフォーマンスおよびチューニング・オプションです。

CACHE *integer-constant*

事前割り振りされ、メモリーに保管されるシーケンス値の最大数を指定します。値を事前割り振りしてキャッシュに保管しておくと、シーケンス値を生成するとき、ログへの非同期入出力が少なくなります。

システム障害が起こると、コミットされたステートメントで使用されていないキャッシュ済みシーケンス値はすべて失われます (使用されなくなります)。 CACHE オプションに指定する値は、システム障害の際に失われても構わないシーケンス値の最大数です。

最小値は 2 です (SQLSTATE 42815)。

NO CACHE

シーケンスの値が事前割り振りされないよう指定します。システム障害、シャットダウン、またはデータベース非活動化の際、値が失われることはありません。このオプションが指定されると、シーケンスの値はキャッシュに保管されません。この場合、シーケンスの新しい値が要求されるたびに、ログに対して非同期入出力が行われます。

ORDER または **NO ORDER**

要求の順序でシーケンス番号が生成されるかどうかを指定します。

ORDER

要求の順序でシーケンス番号が生成されるよう指定します。

NO ORDER

要求の順序でシーケンス番号を生成する必要がないことを指定します。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性と整合性:
    - コンマは、複数のシーケンス・オプションを分離するのに使用できます。

  - 以下の構文もサポートされています。
    - NOMINVALUE、NOMAXVALUE、NOCYCLE、NOCACHE、および NOORDER
- 今後のシーケンス番号だけが ALTER SEQUENCE ステートメントによって影響を受けます。
- シーケンスのデータ・タイプは変更できません。代わりに、新しいシーケンスに目的のデータ・タイプを指定して、シーケンスをドロップおよび再作成してください。
- シーケンスが変更されると、キャッシュされている値はすべて失われます。
- シーケンスを再始動、または CYCLE に変更した後、以前にシーケンスによって生成された値と重複するシーケンス番号が生成される可能性があります。

例:

例 1：数値なしで RESTART を指定する理由として考えられるのは、シーケンスを START WITH 値にリセットすることです。この例では、1 から表の行数までの数値を生成し、一時表を使用して表に追加した列にその数値を挿入しています。以降使用する時には、すべての結果行に番号が付けられて結果が返されます。

```
ALTER SEQUENCE ORG_SEQ RESTART
  SELECT NEXT VALUE FOR ORG_SEQ, ORG.* FROM ORG
```

関連サンプル:

- 『DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)』
- 『DbSeq.out -- HOW TO USE A SEQUENCE IN A DATABASE. Connect to 'sample' database using JDBC type 2 driver (JDBC)』

ALTER SERVER

ALTER SERVER ステートメントは、以下の目的で使用されます。

- 特定のデータ・ソースの定義を変更する場合、またはデータ・ソースのカテゴリーの定義を変更する場合。
- 特定のデータ・ソースの構成を変更する場合、またはデータ・ソースのカテゴリーの構成を変更する場合 (この変更は、フェデレーテッド・データベースへ何回か接続する間、継続します)。

このステートメントでは、SERVER という語と、*server-* で始まるパラメーター名は、フェデレーテッド・システムでのデータ・ソースのみを指しています。そのようなシステムでのフェデレーテッド・サーバー、あるいは DRDA アプリケーション・サーバーを指すわけではありません。

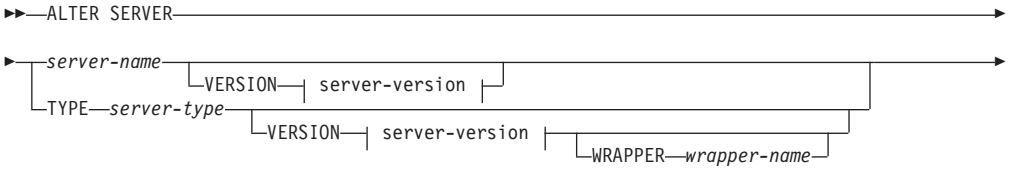
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

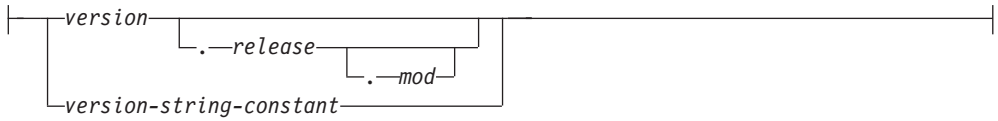
許可:

このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文:



server-version:



説明:

server-name
要求された変更を適用する対象のデータ・ソースに関連するフェデレーテッド・サーバーの名前を指定します。このデータ・ソースは、カタログに記述されているものでなければなりません。

VERSION
server-name の後にある VERSION とそのパラメーターは、 *server-name* に示されている新しいバージョンのデータ・ソースを指定します。

version
バージョン番号を指定します。値は整数でなければなりません。

release
version で示されたバージョンのリリース番号を指定します。値は整数でなければなりません。

mod
release で示されたリリースのモディフィケーション番号を指定します。値は整数でなければなりません。

version-string-constant
バージョンの正式名称を指定します。 *version-string-constant* は単一値 (たとえば、'8i') にすることができます。あるいは、*version*、*release*、そして該当する場合は *mod* を連結した値にすることができます (たとえば、'8.0.3')。

TYPE *server-type*
要求された変更を適用する対象のデータ・ソースのタイプを指定します。

VERSION
server-type の後の VERSION とそのパラメーターでは、サーバー・オプションを使用可能にする、リセットする、あるいはドロップするときの対象となるデータ・ソースのバージョンを指定します。

WRAPPER *wrapper-name*
server-type および *server-version* に示されたタイプおよびバージョンのデータ・ソースと対話するために、フェデレーテッド・サーバーが使用するラッパーの名前を指定します。このラッパーは、カタログにリストされていなければなりません。

OPTIONS
server-name に示されたデータ・ソースに対して、あるいは *server-type* および関連パラメーターに示されたデータ・ソースのカテゴリーに対して、どのサーバー・オプションを使用可能にする、リセットする、またはドロップするかを指定します。

ADD
サーバー・オプションを使用可能にします。

SET
サーバー・オプションの設定を変更します。

server-option-name
使用可能にする、あるいはリセットするサーバー・オプションを指定します。

string-constant
server-option-name の設定を、文字ストリング定数として指定します。

DROP *server-option-name*
サーバー・オプションをドロップします。

注:

- サーバー・オプションは、同じ ALTER SERVER ステートメントに複数回指定することはできません (SQLSTATE 42853)。サーバー・オプションを使用可能にする、リセットする、あるいはドロップする場合、使用中の他のサーバー・オプションには影響はありません。
- 所定の作業単位 (UOW) 内の ALTER SERVER ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - ステートメントが 1 つのデータ・ソースを参照していて、次のいずれかがすでに UOW に含まれている。
    - このデータ・ソース内の表またはビューのニックネームを参照する SELECT ステートメント。
    - このデータ・ソース内の表またはビューのニックネーム上のオープン・カーソル。
    - このデータ・ソース内の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
  - ステートメントがデータ・ソースのカテゴリー (例えば、特定のタイプおよびバージョンのすべてのデータ・ソースなど) を参照しており、次のいずれかがすでに UOW に含まれている。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームを参照する SELECT ステートメント。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネーム上のオープン・カーソル。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
- サーバー・オプションが、データ・ソースのタイプについてある値に設定され、このタイプのインスタンスについてはそれとは別の値に設定される場合、そのインスタンスの値については、前者の値は後者の値によってオーバーライドされます。たとえば、サーバー・タイプ ORACLE について PLAN_HINTS を 'Y' に設定し、 DELPHI という Oracle データ・ソースについては 'N' に設定したとします。このような構成の場合、DELPHI 以外のすべての Oracle データ・ソースで、プランのヒントが使用可能になります。
- 前の alter add server オプション操作で使用可能になったデータ・ソースのカテゴリーに対して、alter set オプションまたは alter drop server オプションのみを実行できます (SQLSTATE 42704)。

例:

例 1: ID が未変更のままとなる場合に、 Oracle 8.0.3 データ・ソースへ許可 ID がいつ送信されるかを確認します。さらに、ローカルのフェデレーテッド・サーバー CPU の速度がデータ・ソース CPU の 2 倍であるとします。オプティマイザーにこの統計を通知します。

```
ALTER SERVER
  TYPE ORACLE
  VERSION 8.0.3
  OPTIONS
     (ADD FOLD_ID 'N',
     SET CPU_RATIO '2.0')
```

例 2: Documentum データ・ソース DCTM_SVR_ASIA がバージョン 4 に変更されていることを指示します。

```
ALTER SERVER DCTM_SVR_ASIA
  VERSION 4
```

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』

関連資料:

- *フェデレーテッド・システム・ガイド* の『フェデレーテッド・システムのサーバー・オプション』

ALTER TABLE

ALTER TABLE ステートメントは、表の定義を変更します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 変更する表に対する ALTER 特権
- 変更する表に対する CONTROL 特権
- 表のスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

外部キーの作成/ドロップの場合、このステートメントの許可 ID には、親表に対する以下の特権が少なくとも 1 つ含まれている必要があります。

- その表に対する REFERENCES 特権
- 指定の親キーのそれぞれの列に対する REFERENCES 特権
- 表に対する CONTROL 特権
- SYSADM または DBADM 権限

表 T の主キーまたはユニーク制約をドロップするには、この親キー T に従属しているすべての表において、ステートメントの許可 ID に以下の特権が少なくとも 1 つ含まれている必要があります。

- その表に対する ALTER 特権
- 表に対する CONTROL 特権
- 表のスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

(全選択を使用して) 表をマテリアライズ照会表に変更するには、このステートメントの許可 ID によって保持されている特権に、以下のうち少なくとも 1 つが含まれている必要があります。

- その表に対する CONTROL
- SYSADM または DBADM 権限

なおかつ、全選択で識別された個々の表またはビューに対する以下の特権が少なくとも 1 つ含まれている必要があります。

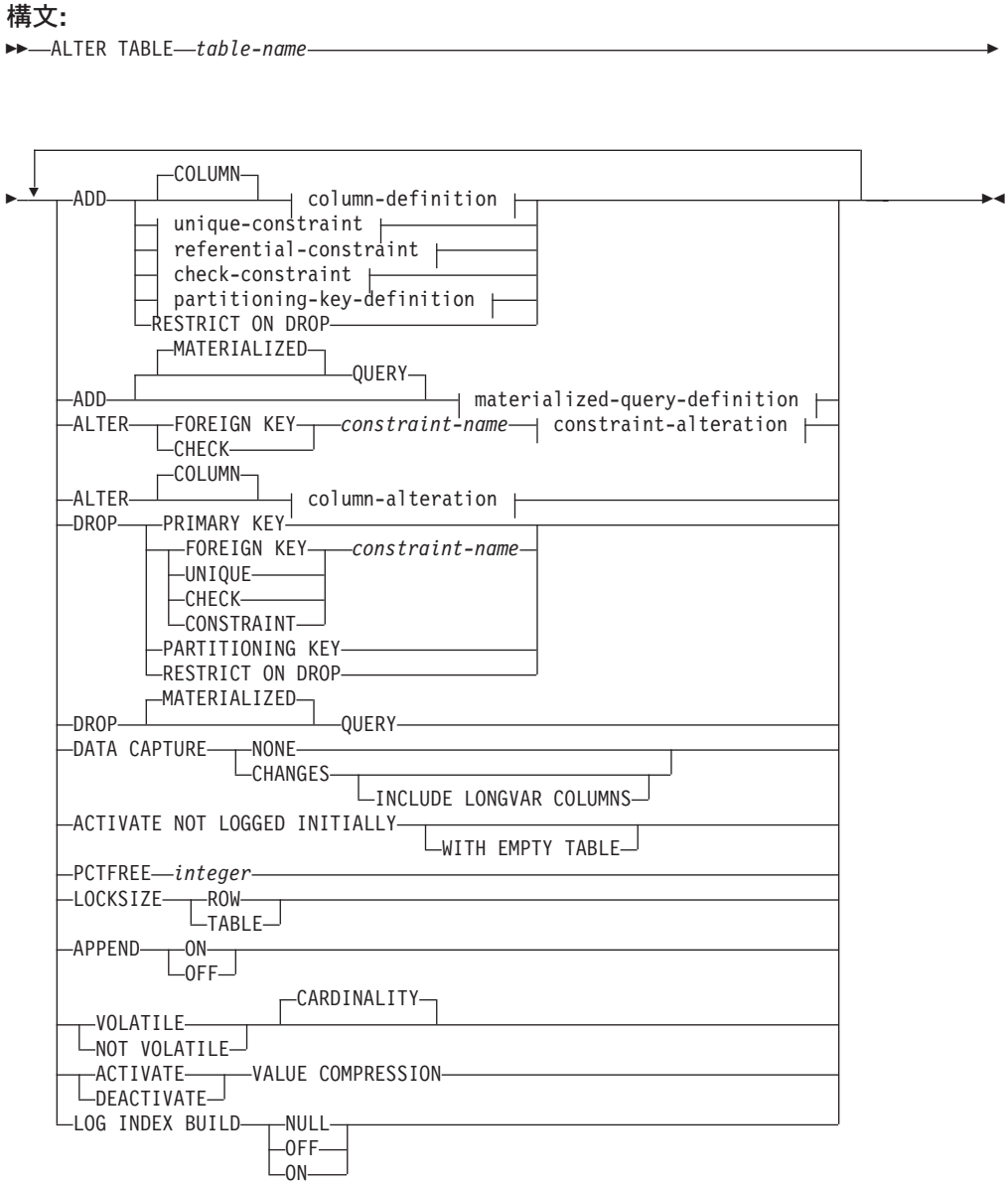
- その表またはビューに対する SELECT および ALTER 特権
- 表またはビューに対する CONTROL 特権
- その表またはビューに対する SELECT 特権と、その表またはビューのスキーマに対する ALTERIN 特権

- SYSADM または DBADM 権限

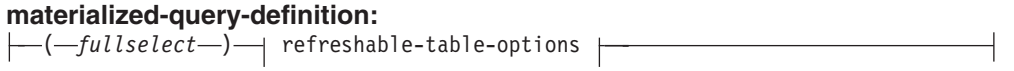
表を変更してマテリアライズ照会表でなくなるようにするには、ステートメントの許可 ID が持っている特権に、このマテリアライズ照会表を定義するのに使用する全選択で識別される各表またはビューに対して、少なくとも以下の 1 つが含まれている必要があります。

- その表またはビューに対する ALTER 特権
- 表またはビューに対する CONTROL 特権
- その表またはビューのスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

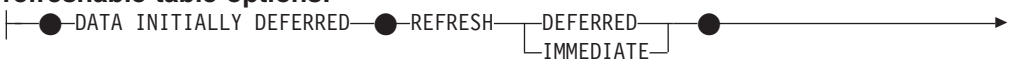
構文:



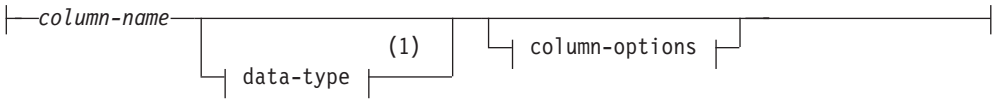
materialized-query-definition:



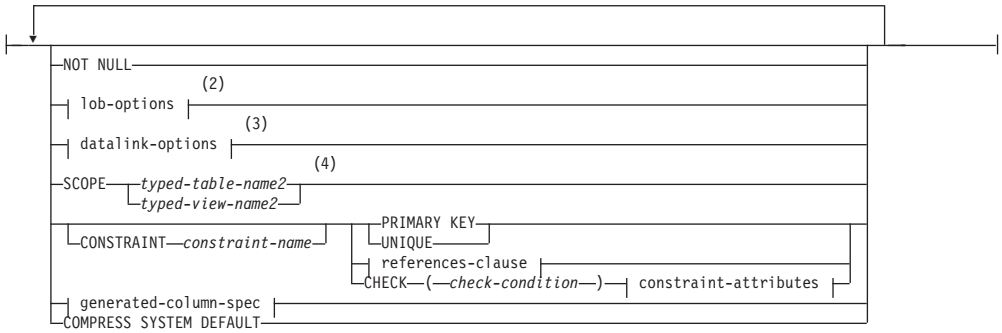
refreshable-table-options:



column-definition:



column-options:



注:

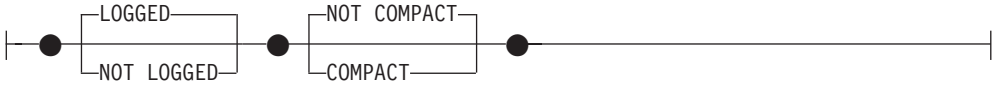
1 最初の column-option で generated-column-spec が選択された場合、 data-type を省略して、generation-expression によって計算するようにできます。

2 lob-options (LOB オプション) 文節は、ラージ・オブジェクト・タイプ（BLOB、CLOB、および DBCLOB）と、ラージ・オブジェクト・タイプに基づく特殊タイプに対してのみ適用されます。

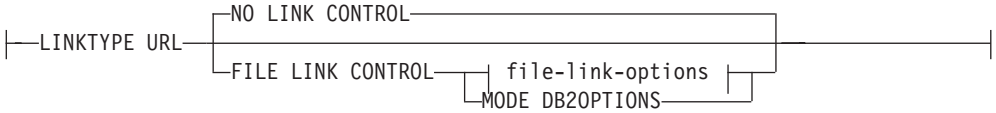
3 datalink-options 文節は、 DATALINK タイプと、DATALINK タイプに基づく特殊タイプに対してのみ適用されます。

4 SCOPE 文節は REF タイプに対してのみ適用されます。

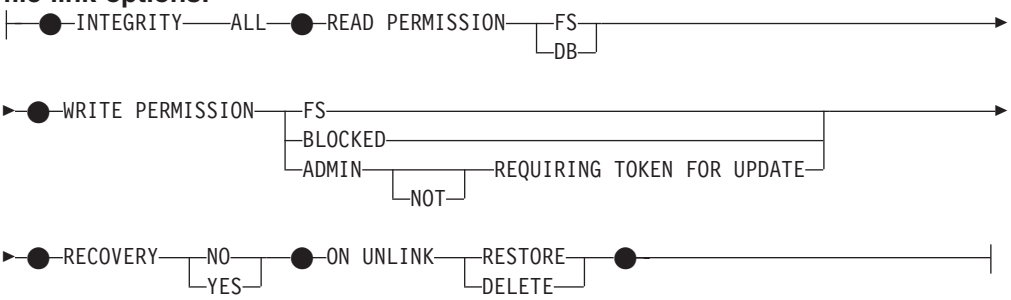
lob-options:



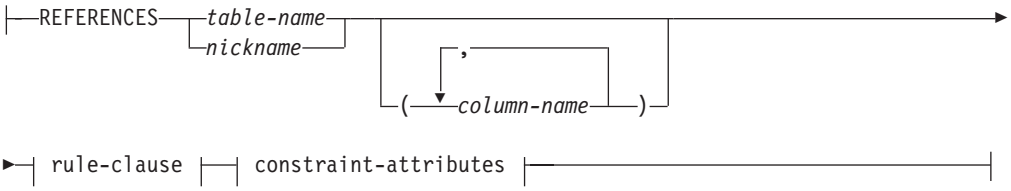
datalink-options:



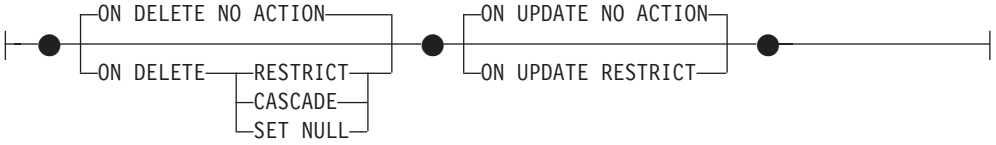
file-link-options:



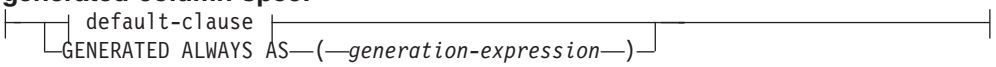
references-clause:



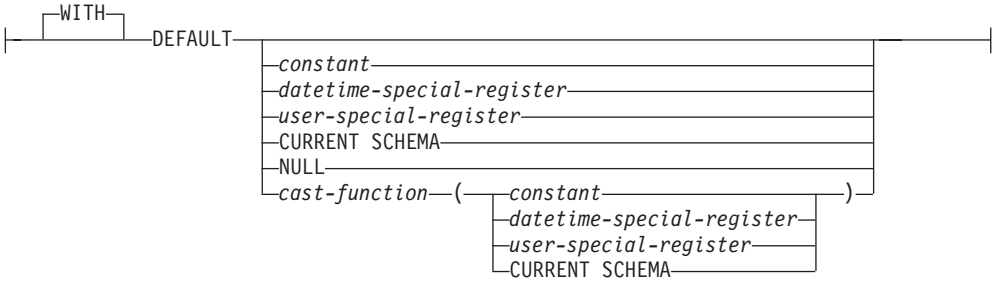
rule-clause:



generated-column-spec:

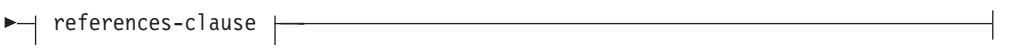


default-clause:

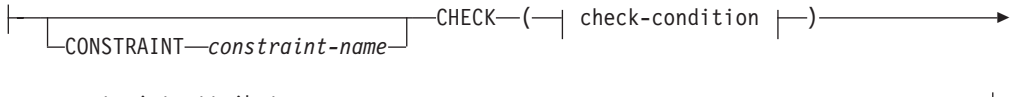


unique-constraint:

referential-constraint:

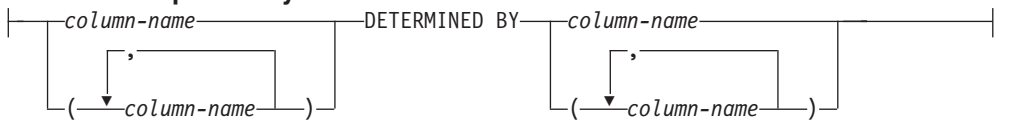


check-constraint:

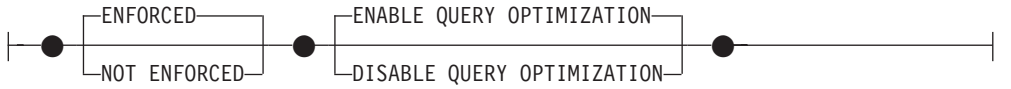


check-condition:

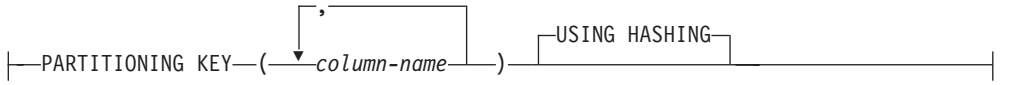
functional-dependency:



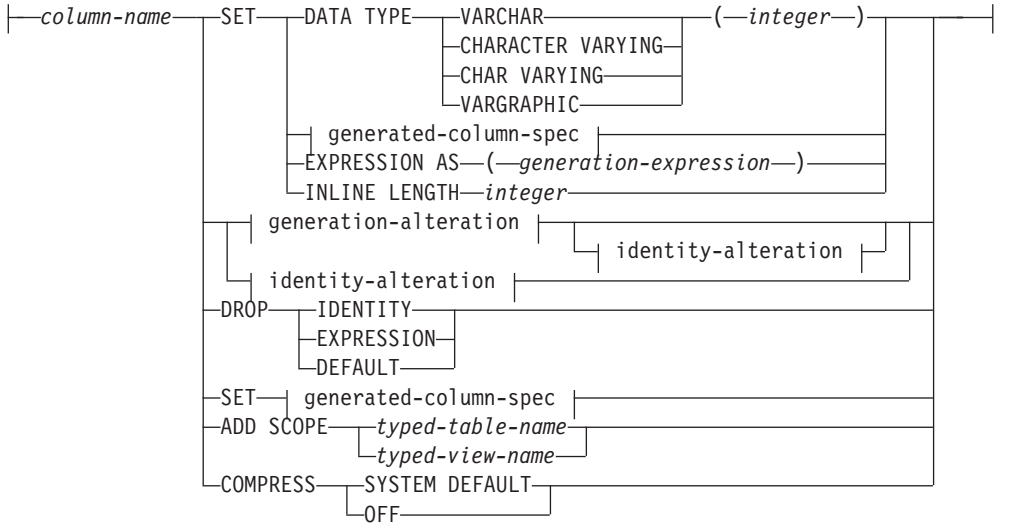
constraint-attributes:



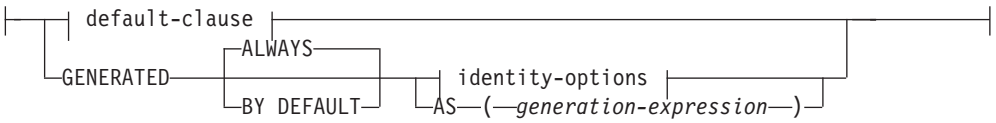
partitioning-key-definition:



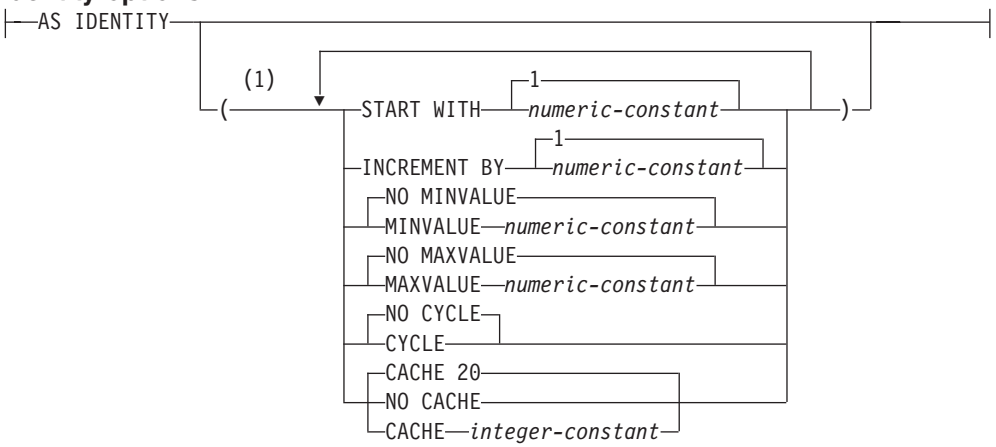
column-alteration:



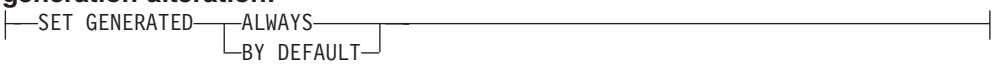
generated-column-spec:



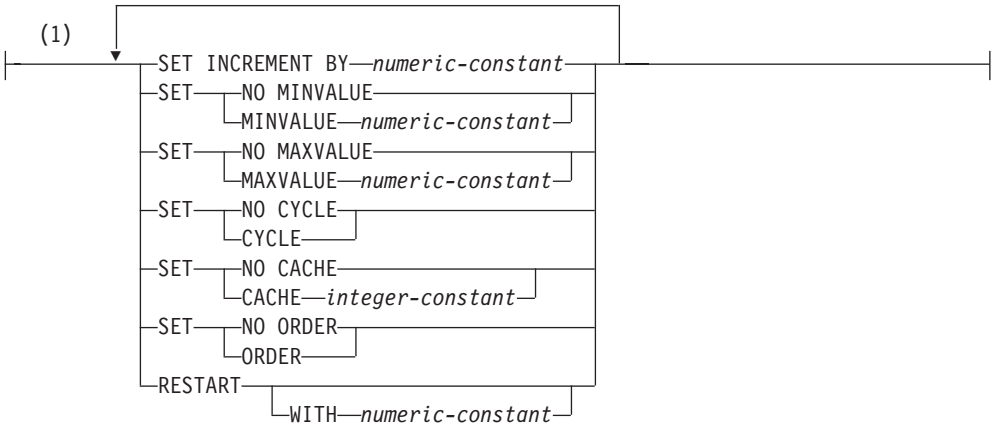
identity-options:



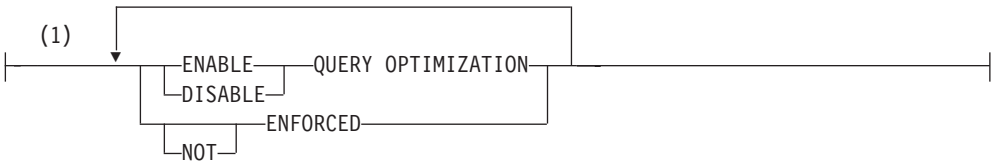
generation-alteration:



identity-alteration:



constraint-alteration:



注:

1 同じ文節を複数回指定することはできません。

説明:

table-name

table-name は、現行サーバーに存在する表を示していなければなりません。これはニックネームであってはならず (SQLSTATE 42809)、ビュー、カタログ表、または宣言済み一時表であってもなりません (SQLSTATE 42995)。

table-name でマテリアライズ照会表を指定している場合、変更は、マテリアライズ照会表の追加またはドロップ、 NOT LOGGED INITIALLY の活動化、RESTRICT ON DROP の追加またはドロップ、 pctfree、locksize、append、または volatile の変更のみを行うことができます。

table-name が範囲クラスター表を示している場合、変更は、制約の追加・変更・ドロップ、NOT LOGGED INITIALLY の活動化、 RESTRICT ON DROP の追加またはドロップ、locksize、data capture、または volatile の変更、および列のデフォルト値の設定のみ行うことができます。

ADD *column-definition*

列を表に追加します。型付き表を使用することはできません (SQLSTATE 428DH)。 表の既存の行に関して、新しい列の値はデフォルト値に設定されます。新しい列はその表の最後の列になります。つまり、当初 *n* 個の列があった場合、追加された列は列 *n*+1 になります。

新しい列を追加する場合、すべての列のバイト・カウントの合計が、最大レコード・サイズを超えてはなりません。

column-name

表に追加する列の名前です。名前は非修飾でなければなりません。表にすでにある列名は使用できません (SQLSTATE 42711)。

data-type

『CREATE TABLE』 の項に示されるデータ・タイプのいずれかです。

NOT NULL

列に NULL 値が入るのを防止します。 *default-clause* (DEFAULT 文節) も指定する必要があります (SQLSTATE 42601)。

lob-options

LOB データ・タイプのオプションを指定します。 『CREATE TABLE』 の *lob-options* を参照してください。

datalink-options

DATALINK データ・タイプのオプションを指定します。 『CREATE TABLE』 の *datalink-options* を参照してください。

SCOPE

参照タイプ列の有効範囲を指定します。

typed-table-name2

型付き表の名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-table-name2* のタイプを表します (SQLSTATE 428DM)。値が *typed-table-name2* の既存行を実際に参照していることを確認するための、 *column-name* のデフォルト値の検査は行われません。

typed-view-name2

型付きビューの名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-view-name2* のタイプを表します

(SQLSTATE 428DM)。値が *typed-view-name2* の既存行を実際に参照していることを確認するための、 *column-name* のデフォルト値の検査は行われません。

CONSTRAINT *constraint-name*
制約の名前を指定します。制約名 *(constraint-name)* は、同じ ALTER TABLE ステートメントにすでに指定されている制約、あるいは表に既存の他の制約の名前であってはなりません (SQLSTATE 42710)。

ユーザーが制約名を指定しない場合は、表に定義されている既存の制約の ID の中でユニークな 18 文字の ID がシステムによって生成されます。(ID は、″SQL″ と、タイム・スタンプに基づいて生成される 15 の数字から構成されます。)

PRIMARY KEY 制約またはユニーク制約とともに使用した場合、この *constraint-name* は、制約をサポートするために作成される索引の名前として使用されます。ユニーク制約に関連した索引名の詳細については、 72 ページの『注』を参照してください。

PRIMARY KEY
これは、1 つの列からなる主キーを定義する簡単な方法として用意されています。つまり、PRIMARY KEY が列 C の定義で指定されている場合、その効果は、PRIMARY KEY(C) 文節が独立した文節として指定された場合と同じです。列に NULL 値を含めることはできないので、 NOT NULL 属性も指定する必要があります (SQLSTATE 42831)。

この後の *unique-constraint* の説明の中の PRIMARY KEY を参照してください。

UNIQUE
これは、1 つの列からなるユニーク・キーを定義する簡単な方法です。すなわち、UNIQUE を列 C の定義に指定すると、 UNIQUE(C) 文節を独立した文節として指定した場合と同じ結果になります。

この後の *unique-constraint* の説明の中の UNIQUE を参照してください。

references-clause
これは、1 つの列からなる外部キーを定義する簡単な方法として用意されています。つまり、references-clause が列 C の定義に指定されている場合、その効果は、列として C しか指定されていない FOREIGN KEY 文節の一部として references-clause が指定された場合と同じになります。

『CREATE TABLE』 の *references-clause* を参照してください。

CHECK (*check-condition***)**
これは、1 つの列に適用されるチェック制約を定義する簡単な方法として用意されています。 『CREATE TABLE』 の *check-condition* を参照してください。

generated-column-spec
列の生成の詳細については、『CREATE TABLE』 を参照してください。

default-clause
列のデフォルト値を指定します。

WITH
オプション・キーワード。

DEFAULT
INSERT で値が提供されなかった場合、もしくは INSERT や UPDATE で DEFAULT が指定されている場合に、デフォルト値を提供します。DEFAULT キーワードの後に特定のデフォルト値の指定がない場合のデフォルト値は、列のデータ・タイプによって異なります。 表 2 を参照してください。列が DATALINK または構造タイプとして定義されている場合、 DEFAULT 文節を指定することはできません。

列が特殊タイプを使用して定義される場合、列のデフォルト値は、特殊タイプにキャストされたソース・データ・タイプのデフォルト値になります。

表 2. デフォルト値 (値が指定されない場合)

データ・タイプ	デフォルト値
数値	0
固定長文字ストリング	ブランク
可変長文字ストリング	長さ 0 のストリング
固定長 GRAPHIC ストリング	2 バイトのブランク
可変長 GRAPHIC ストリング	長さ 0 のストリング
日付	既存の行の場合、0001 年 1 月 1 日に対応する日付。追加行の場合には、現在の日付。
時刻	既存の行の場合、0 時間 0 分 0 秒に対応する時刻。追加行の場合には、現在の時刻。
タイム・スタンプ	既存の行の場合、0001 年 1 月 1 日 0 時 0 分 0 秒 0 マイクロ秒に対応する日付。追加行の場合には、現在のタイム・スタンプ。
バイナリー・ストリング (blob)	長さ 0 のストリング

column-definition から DEFAULT を省略すると、その列のデフォルト値として NULL 値が使用されます。

DEFAULT キーワードに指定できる値のタイプは、次のとおりです。

constant
列のデフォルト値として定数を指定します。指定する定数は、次の条件を満たしていなければなりません。

- 第 3 章に示されている割り当ての規則に従って、その列に割り当てることができる値でなければなりません。
- その列が浮動小数点データ・タイプとして定義されている場合を除き、浮動小数点の定数を指定してはなりません。

- 定数が 10 進定数の場合、その列のデータ・タイプの位取りを超えるゼロ以外の数字を含めてはなりません (たとえば、DECIMAL(5,2) の列のデフォルト値として 1.234 を指定することはできません)。
- 指定する定数が 254 文字を超えてはなりません。この制約には、引用符文字や 16 進定数の X などの接頭部文字も含まれます。さらに、定数が *cast-function* の引き数の場合には、完全修飾された関数名から取った文字や括弧も含めて、この制限を超えてはなりません。

datetime-special-register
INSERT、UPDATE、または LOAD の実行時における日時特殊レジスターの値 (CURRENT DATE、 CURRENT TIME、または CURRENT TIMESTAMP) を、その列のデフォルト値として指定します。その列のデータ・タイプは、指定した特殊レジスターに対応するデータ・タイプでなければなりません (たとえば、 CURRENT DATE を指定した場合、データ・タイプは DATE でなければなりません)。既存の行の場合は、ALTER TABLE ステートメントが処理される時点の現行日付、現行時刻、または現行タイム・スタンプが値として使用されます。

user-special-register
INSERT、UPDATE、または LOAD の実行時におけるユーザー特殊レジスターの値 (CURRENT USER、SESSION_USER、SYSTEM_USER) を、その列のデフォルトとして指定します。その列のデータ・タイプは、ユーザー特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。なお、SESSION_USER の代わりに USER を、 CURRENT USER の代わりに CURRENT_USER を指定することもできます。既存の行の場合、値は ALTER TABLE ステートメントの CURRENT USER、 SESSION_USER、または SYSTEM_USER になります。

CURRENT SCHEMA
INSERT、UPDATE、または LOAD の実行時における CURRENT SCHEMA 特殊レジスターの値を、その列のデフォルト値として指定します。 CURRENT SCHEMA を指定した場合、その列のデータ・タイプは、 CURRENT SCHEMA 特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。既存の行の場合は、 ALTER TABLE ステートメントが処理される時点における CURRENT SCHEMA 特殊レジスターの値です。

NULL
その列のデフォルト値として NULL を指定します。 NOT NULL の指定がある場合には、 DEFAULT NULL を同じ列定義に指定してはなりません。

cast-function
この形式のデフォルト値は、特殊タイプ (distinct type)、 BLOB、または日時 (DATE、TIME、または TIMESTAMP) データ・タイプとして定義された列に対してのみ使用することができます。特殊タイ

プの場合、BLOB や日時タイプに基づく例外があり、関数名が列の特殊タイプの名前に一致していなければなりません。スキーマ名で修飾されている場合には、その特殊タイプのスキーマ名と同じでなければなりません。修飾されていない場合には、関数の解決に用いるスキーマ名は特殊タイプのスキーマ名と同じでなければなりません。日時タイプに基づく特殊タイプで、デフォルト値が定数の場合、必ず関数を使用する必要があります。さらに、その関数名は、暗黙または明示のスキーマ名 SYSIBM を持つ特殊タイプのソース・タイプ名に一致していなければなりません。他の日時列の場合は、対応する日時関数も使用できます。 BLOB に基づく BLOB または特殊タイプの場合も、関数を使用する必要があります。その関数名は、暗黙または明示のスキーマ名 SYSIBM を持つ BLOB でなければなりません。

constant
引き数として定数を指定します。指定する定数は、特殊タイプのソース・タイプに関する定数の規則 (特殊タイプでない場合は、データ・タイプに関する定数の規則) に従っていなければなりません。 cast-function が BLOB の場合には、定数としてストリング定数を指定する必要があります。

datetime-special-register
CURRENT DATE、CURRENT TIME、または CURRENT TIMESTAMP を指定します。列の特殊タイプのソース・タイプは、指定した特殊レジスターに対応するデータ・タイプでなければなりません。

user-special-register
CURRENT USER、SESSION_USER、または SYSTEM_USER を指定します。列の特殊タイプのソース・タイプのデータ・タイプは、少なくとも 8 バイトの長さのストリング・データ・タイプでなければなりません。 *cast-function* が BLOB の場合には、長さ属性が 8 バイト以上でなければなりません。

CURRENT SCHEMA
CURRENT SCHEMA 特殊レジスターの値を指定します。列の特殊タイプのソース・タイプのデータ・タイプは、 CURRENT SCHEMA 特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。 cast-function が BLOB の場合には、長さ属性が 8 バイト以上でなければなりません。

指定した値が無効な場合、エラー (SQLSTATE 42894) が戻されます。

GENERATED
DB2 が列の値を生成することを指定します。

ALWAYS
行が表に挿入されるときや、*generation-expression* の結果値が変更されるたびに、 DB2 は常に列の値を生成することを指定します。この式の結果は、表に保管されます。データ伝搬や、アンロードおよび再ロード

操作を実行しているのでなければ、 GENERATED ALWAYS が推奨されるオプションです。 GENERATED ALWAYS は、生成列に必須指定のオプションです。

BY DEFAULT
行が表に挿入されたり、更新されるときに、明示的に値を指定しないかぎり、列に DEFAULT を指定して DB2 が列に値を生成することを指定します。データ伝搬を使用したり、アンロードおよび再ロードを実行したりするときは、BY DEFAULT が推奨されるオプションです。

identity-options
この文節は、既存の表に列を追加するときには指定できません。

AS (*generation-expression***)**
列定義が式に基づくことを指定します。 SET INTEGRITY ステートメントを使用して表をチェック・ペンディング状態にする必要があります。 ALTER TABLE ステートメントの後、FORCE GENERATED を伴う SET INTEGRITY ステートメントを使用して、新しい式に対してこの列にあるすべての値を更新および検査しなければなりません。 *generation-expression* による列の指定の詳細については、『CREATE TABLE』 を参照してください。

COMPRESS SYSTEM DEFAULT
システム・デフォルト値 (つまり、特定の値が指定されない場合にデータ・タイプとして使用されるデフォルト値) が最小限のスペースを使用して保管されるように指定します。 VALUE COMPRESSION 文節が指定されていない場合には警告が出され (SQLSTATE 01648)、システム・デフォルト値は最小限のスペースを使用しては保管されません。

システム・デフォルト値がこのような方法で保管されると、列に対する許可や更新操作の際に余分な検査が行われるために、若干パフォーマンスが低下します。

基本データ・タイプは、DATE、TIME、または TIMESTAMP であってはなりません (SQLSTATE 42842)。基本データ・タイプが可変長ストリングの場合には、この文節は無視されます。表が VALUE COMPRESSION に設定されている場合は、長さ 0 のストリング値は自動的に圧縮されます。

ADD *unique-constraint*
ユニーク制約または主キー制約を定義します。主キー制約またはユニーク制約を、副表に追加することはできません (SQLSTATE 429B3)。階層最上部のスーパー表の場合、制約はその表および関連する副表すべてに適用されます。

CONSTRAINT *constraint-name*
主キー制約、またはユニーク制約の名前を指定します。詳細については、『CREATE TABLE』 で *constraint-name* を参照してください。

UNIQUE (*column-name***...,)**
指定した列で構成されるユニーク・キーを定義します。指定する列は NOT NULL として定義されていなければなりません。各 *column-name* (列名) は、表の列を指定するものでなければなりません。また、同じ列を複数回指定することはできません。名前は非修飾でなければなりません。指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません。列の長さ属性が 1024 バイト以内に収まる場合でも、LOB、LONG VARCHAR、LONG VARGRAPHIC、 DATALINK、これらの

タイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、ユニーク・キーの一部として使用できません (SQLSTATE 54008)。ユニーク・キーにある一連の列は、主キーまたは他のユニーク・キーの一連の列と同じにすることはできません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。指定した一連の列に存在する値は、ユニークである必要があります (SQLSTATE 23515)。

既存の索引がユニーク・キー定義と一致しているかどうか判別するために、チェックが実行されます (索引内の INCLUDE 列はすべて無視されます)。列の順序や方向 (ASC/DESC) の指定に関係なく、同じ一連の列を指定していると、索引定義は一致します。一致する索引定義が見つかると、その索引の記述は、システムによりその索引が必要であることを示すように変更され、索引がユニークでない場合はユニーク索引に変更されます (ユニーク性を確実にした後)。その表に一致する索引が複数ある場合、既存のユニーク索引が選択されます (選択は任意に行われます)。一致する索引が見つからない場合は、CREATE TABLE で説明するように、その列に対してユニーク索引が自動的に作成されます。ユニーク制約に関連した索引名の詳細については、72 ページの『注』を参照してください。

PRIMARY KEY *...(column-name,)*

指定された列で構成される主キーを定義します。各 *column-name* (列名) は、表の列を指定していなければなりません。また、同じ列を複数回指定することはできません。名前は非修飾でなければなりません。指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません。表には主キーがあってはならず、指定する列は NOT NULL として定義されているものでなければなりません。列の長さ属性が 1024 バイト以内に収まる場合でも、 LOB、LONG VARCHAR、LONG VARGRAPHIC、 DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、主キーの一部として使用できません (SQLSTATE 54008)。主キーの一連の列は、ユニーク・キーの一連の列と同じであってはなりません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。指定した一連の列に存在する値は、ユニークである必要があります (SQLSTATE 23515)。

既存の索引が主キー定義と一致しているかどうか判別するために、チェックが実行されます (索引内の INCLUDE 列はすべて無視されます)。列の順序や方向 (ASC/DESC) の指定に関係なく、同じ一連の列を指定していると、索引定義は一致します。一致する索引定義が見つかると、その索引の記述は、その索引が 1 次索引である (システムが必要としている) ことを示すように変更され、索引がユニークでない場合はユニーク索引に変更されます (ユニーク性を確実にした後)。その表に一致する索引が複数ある場合、既存のユニーク索引が選択されます (選択は任意に行われます)。一致する索引が見つからない場合は、CREATE TABLE で説明するように、その列に対してユニーク索引が自動的に作成されます。ユニーク制約に関連した索引名の詳細については、72 ページの『注』を参照してください。

1 つの表には、主キーを 1 つだけ定義することができます。

ADD *referential-constraint*
参照制約を定義します。 『CREATE TABLE』 の *referential-constraint* を参照してください。

ADD *check-constraint*
チェック制約または機能従属関係を定義します。 『CREATE TABLE』 の *check-constraint* を参照してください。

ADD *partitioning-key-definition*
パーティション・キーを定義します。表は、単一パーティションのデータベース・パーティション・グループにある表スペースに定義する必要があり、すでにパーティション・キーを持っていてはなりません。パーティション・キーが表にすでに存在している場合には、新しいパーティション・キーを追加する前に既存のキーをドロップする必要があります。

パーティション・キーを、副表に追加することはできません (SQLSTATE 428DH)。

PARTITIONING KEY (*column-name***...)**
指定した列を使用して、パーティション・キーを定義します。各 *column-name* (列名) は、表の列を指定していなければなりません。また、同じ列を複数回指定することはできません。名前は非修飾でなければなりません。列のデータ・タイプが LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK、これらのいずれかのタイプの特殊タイプ、または構造タイプである場合、パーティション・キーの一部として列を使用することはできません。

USING HASHING
データ分散のパーティション化方式として、ハッシュ関数を使用することを指定します。これは、サポートされる唯一のパーティション化方式です。

ADD RESTRICT ON DROP
表をドロップできないように、また、表を含む表スペースをドロップできないように指定します。

ADD MATERIALIZED QUERY

materialized-query-definition
照会の最適化で使用するために、正規表をマテリアライズ照会表に変更します。 *table-name* で指定する表には、以下の条件があります。

- マテリアライズ照会表として以前に定義されていてはなりません。
- 型付き表であってはなりません。
- 何らかの定数、ユニーク索引、またはトリガーが定義されていてはなりません。
- 他のマテリアライズ照会表の定義に参照されていてはなりません。

表名が基準に適合しない場合、エラーが戻されます (SQLSTATE 428EW)。

fullselect
表の基礎となる照会を定義します。既存の表の列は、*fullselect* の結果列と以下のような関係になければなりません (SQLSTATE 428EW)。

- 列の数が同数でなければなりません。
- 全く同じデータ・タイプでなければなりません。

- 同じ順序を示す位置に同じ列名がなければなりません。

マテリアライズ照会表に *fullselect* を指定することについての詳細は、『CREATE TABLE』 を参照してください。追加の制限事項としては、*table-name* は全選択で直接的にも間接的にも参照できません。

refreshable-table-options
マテリアライズ照会表を変更するためのリフレッシュ可能オプションを指定します。

DATA INITIALLY DEFERRED
REFRESH TABLE または SET INTEGRITY ステートメントを使用して、表のデータを妥当性検査する必要があります。

REFRESH
表のデータを保守する方法を示します。

DEFERRED
REFRESH TABLE ステートメントを使っていつでも表のデータをリフレッシュできます。表のデータには、 REFRESH TABLE ステートメント処理時のスナップショットである照会結果が反映されるにすぎません。この属性を定義したマテリアライズ照会表には、INSERT、 UPDATE、または DELETE ステートメントを使用できません (SQLSTATE 42807)。

IMMEDIATE
DELETE、INSERT、または UPDATE の一部として基礎表に加えられた変更は、マテリアライズ照会表にカスケードされます。その場合、表の内容は、どのポイント・イン・タイム指定でも、指定した subselect (副選択) を処理する場合と同じ内容になります。この属性を定義したマテリアライズ照会表には、INSERT、 UPDATE、または DELETE ステートメントを使用できません (SQLSTATE 42807)。

ENABLE QUERY OPTIMIZATION
マテリアライズ照会表を照会の最適化に使用できるようにします。

DISABLE QUERY OPTIMIZATION
マテリアライズ照会表を照会の最適化に使用しません。それでもその表を直接照会することはできます。

MAINTAINED BY
マテリアライズ照会表のデータが、システム、ユーザー、またはレプリケーション・ツールのいずれによって保守されるかを指定します。

SYSTEM
マテリアライズ照会表のデータがシステムによって保守されるように指定します。

USER
マテリアライズ照会表のデータがユーザーによって保守されるように指定します。ユーザーは、ユーザー保守済みマテリアライズ照会表に対して、更新、削除、また挿入操作を許可されます。システム保守のマテリアライズ照会表で使用される

REFRESH TABLE ステートメントは、ユーザー保守のマテリアライズ照会表に対しては呼び出せません。 REFRESH DEFERRED マテリアライズ照会表だけが、MAINTAINED BY USER として定義されることが可能です。

FEDERATED_TOOL
マテリアライズ照会表のデータがレプリケーション・ツールによって保守されるように指定します。システム保守のマテリアライズ照会表で使用される REFRESH TABLE ステートメントは、フェデレーテッド・ツール保守のマテリアライズ照会表に対しては呼び出せません。 REFRESH DEFERRED のマテリアライズ照会表だけが、MAINTAINED BY FEDERATED_TOOL として定義されることが可能です。

ALTER FOREIGN KEY *constraint-name*
参照制約 *constraint-name* の制約属性を変更します。 *constraint-name* は既存の参照制約を指定する必要があります (SQLSTATE 42704)。

ALTER CHECK *constraint-name*
チェック制約または機能従属関係 *constraint-name* の制約属性を変更します。 *constraint-name* は、既存のチェック制約または機能従属関係を指定していなければなりません (SQLSTATE 42704)。

constraint-alteration
参照制約またはチェック制約に関連付けられた属性の変更のオプションです。

ENFORCED または **NOT ENFORCED**
挿入、更新、削除などの通常の操作中に、データベース・マネージャーによって制約が課せられるかどうかを指定します。

ENFORCED
制約を ENFORCED に変更します。 機能従属関係に ENFORCED を指定することはできません (SQLSTATE 42621)。

NOT ENFORCED
制約を NOT ENFORCED に変更します。この制約に適合することが分かっている表データだけに、制約を指定してください。

ENABLE QUERY OPTIMIZATION または **DISABLE QUERY OPTIMIZATION**
適切な状況下で、照会の最適化のために、制約または機能従属関係を使用できるかどうかを指定します。

ENABLE QUERY OPTIMIZATION
制約が真であると想定され、照会の最適化のために使用できます。

DISABLE QUERY OPTIMIZATION
制約を照会の最適化に使用できません。

ALTER *column-alteration*
列の定義を変更します。指定した属性だけが変更され、他の属性は変更されません。

column-name
変更される列の名前を指定します。 *column-name* は、表の既存列を指定するものでなければなりません (SQLSTATE 42703)。名前は非修飾でなければ

なりません。 同じ ALTER TABLE ステートメントで追加、ドロップ、または変更される列を指す名前は指定できません (SQLSTATE 42711)。

SET DATA TYPE
列のデータ・タイプを設定します。型付き表を使用することはできません (SQLSTATE 428DH)。

列を変更する場合、すべての列のバイト・カウントの合計が、最大レコード・サイズを超えてはなりません (SQLSTATE 54010)。ユニーク制約または索引で列を使用する場合、新しい長さはユニーク制約または索引の列の保管長の合計が、1024 を超えないようにしなければなりません (SQLSTATE 54008)。

VARCHAR *integer*
既存の VARCHAR 列の長さを増やします。 CHARACTER VARYING または CHAR VARYING を、 VARCHAR キーワードの同義語として使用することができます。 *column-name* のデータ・タイプは VARCHAR でなければならず、また現行の列の最大長は *integer* の値以下でなければなりません (SQLSTATE 42837)。 *integer* の値の上限は、32 672 です。

VARGRAPHIC *integer*
既存の VARGRAPHIC 列の長さを増やします。 *column-name* のデータ・タイプは VARGRAPHIC でなければならず、また現行の列の最大長は *integer* の値以下でなければなりません (SQLSTATE 42837)。 *integer* の値の上限は、16 336 です。

SET EXPRESSION AS (*generation-expression***)**
列の式を、指定された *generation-expression* に変更します。 SET EXPRESSION AS では、 SET INTEGRITY ステートメントを使用して表をチェック・ペンディング状態にする必要があります。 ALTER TABLE ステートメントの後、SET INTEGRITY ステートメントを使用して、新しい式に対してこの列にあるすべての値を更新および検査しなければなりません。列は、式に基づいて生成される列として定義されていなければなりません (SQLSTATE 42837)。また表の DIMENSIONS 文節に現れないようにする必要があります (SQLSTATE 42997)。 generation-expression は、生成される列を定義する際に適用されるのと同じ規則に適合する必要があります。 generation-expression の結果データ・タイプは、列のデータ・タイプに割り当て可能でなければなりません (SQLSTATE 42821)。

SET *generated-column-spec*
列の値を生成するために使用される技法を指定します。これは、特定のデフォルト値、式、または IDENTITY 列としての列の定義という形を取ることができます。列の既存のデフォルトが、別の生成技法に由来する場合は、そのデフォルトをドロップする必要があります。それは、いずれかの DROP 文節を使用して、同じ *column-alteration* の中で行うことができます。

default-clause
変更される列の新規デフォルト値を指定します。その列は IDENTITY 列としてすでに定義されていてはならず、生成式が定義されていてはなりません (SQLSTATE 42837)。指定されるデフォルト値は、『割り当ておよび比較』 で説明されている割り当ての規則に従って、その列に割

り当てることのできる値でなければなりません。デフォルト値を変更しても、既存の行については、この列に関連した値が変更されるわけではありません。

GENERATED ALWAYS または **GENERATED BY DEFAULT**
データベース・マネージャーがその列の値をいつ生成するかを指定します。 GENERATED BY DEFAULT は、値が提供されないときか、列への割り当てに DEFAULT キーワードが使用されたときだけ、値が生成されることを指定します。 GENERATED ALWAYS は、データベース・マネージャーが常にその列の値を生成することを指定します。 GENERATED BY DEFAULT を *generation-expression* と一緒に指定することはできません。

identity-options
その列が表の IDENTITY 列であることを指定します。その列は IDENTITY 列としてすでに定義されていてはならず、生成式を持つことはできず、明示的デフォルトを持つこともできません (SQLSTATE 42837)。 1 つの表には 1 つしか IDENTITY 列があってはなりません (SQLSTATE 428C1)。その列は NULL 可能でないものとして指定される必要があり (SQLSTATE 42997)、列に関連したデータ・タイプは、位取りがゼロの完全な数値データ・タイプでなければなりません (SQLSTATE 42815)。完全な数値データ・タイプは次のいずれかです:
SMALLINT、INTEGER、BIGINT、DECIMAL、位取りがゼロの NUMERIC、またはこれらのいずれかのタイプに基づく特殊タイプ。 IDENTITY オプションの詳細については、『CREATE TABLE』 を参照してください。

AS (*generation-expression***)**
列定義が式に基づくことを指定します。その列は生成式ですでに定義されていてはならず、IDENTITY 列であってはならず、明示的デフォルトを持つこともできません (SQLSTATE 42837)。
generation-expression は、生成される列を定義する際に適用されるのと同じ規則に適合する必要があります。 *generation-expression* の結果データ・タイプは、列のデータ・タイプに割り当て可能でなければなりません (SQLSTATE 42821)。その列は、パーティション・キー列で、または ORGANIZE BY 文節の中で、参照されていてはなりません (SQLSTATE 42997)。

SET INLINE LENGTH *integer*
既存の構造タイプ列のインライン長を変更します。インライン長は、行内の残りの値とともにインラインで保管する構造タイプのインスタンスの最大バイト・サイズを指示します。インラインで保管できない構造タイプのインスタンスは、 LOB 値が処理されるのに似た方法で、基本表の行とは別に保管されます。

column-name のデータ・タイプは、構造タイプでなければなりません (SQLSTATE 42842)。

構造タイプ列のデフォルトの INLINE LENGTH は、このタイプのインライン長になります (明示的に指定するか、または CREATE TYPE ステートメ

ント内のデフォルトとして)。構造タイプのインライン長が 292 未満の場合、列のインライン長には値 292 が使われます。

明示的なインライン長の値は増やすことのみ可能で (SQLSTATE -1)、少なくとも 292 でなければならず、32672 を超えてはなりません (SQLSTATE 54010)。

列を変更する場合、すべての列のバイト・カウントの合計が、最大レコード・サイズを超えてはなりません (SQLSTATE 54010)。

残りの行とは別々に既に保管されたデータは、このステートメントによりインラインに移動されません。構造タイプ列のインライン長の変更の利点を生かすには、列のインライン長を変更した後に指定された表に対して REORG コマンドを呼び出します。

SET GENERATED ALWAYS または **GENERATED BY DEFAULT**
データベース・マネージャーがその列の値をいつ生成するかを指定します。GENERATED BY DEFAULT は、値が提供されないときか、列への割り当てに DEFAULT キーワードが使用されたときだけ、値が生成されることを指定します。 GENERATED ALWAYS は、データベース・マネージャーが常にその列の値を生成することを指定します。その列は、IDENTITY 列に基づく生成列としてすでに定義されている、すなわち、 AS IDENTITY 文節によって定義されている必要があります (SQLSTATE 42837)。

DROP DEFAULT
列の現行デフォルトをドロップします。指定される列には、デフォルト値がなければなりません (SQLSTATE 42837)。

DROP EXPRESSION
列の生成された式属性をドロップし、その列を非生成の列にします。その列が、生成された式列でない場合は、DROP EXPRESSION は許可されません (SQLSTATE 42837)。

DROP IDENTITY
列の IDENTITY 属性をドロップし、その列を単なる数値データ・タイプ列にします。その列が IDENTITY 列でない場合は、DROP IDENTITY は許可されません (SQLSTATE 42837)。

ADD SCOPE
有効範囲が未定義である既存の参照タイプ列に、有効範囲を追加します (SQLSTATE 428DK)。変更する表が型付き表である場合、列をスーパー表から継承することはできません (SQLSTATE 428DJ)。

typed-table-name
型付き表の名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-table-name* のタイプを表します (SQLSTATE 428DM)。値が *typed-table-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

typed-view-name
型付きビューの名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-view-name* のタイプを表します

(SQLSTATE 428DM)。値が *typed-view-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

COMPRESS
この列のデフォルト値をさらに効率よく保管するかどうかを指定します。

SYSTEM DEFAULT
システム・デフォルト値 (つまり、特定の値が指定されない場合にデータ・タイプとして使用されるデフォルト値) が最小限のスペースを使用して保管されるように指定します。活動化された VALUE COMPRESSION 属性に表が設定されていない場合には、警告が出され (SQLSTATE 01648)、システム・デフォルト値は最小限のスペースを使用しては保管されません。

システム・デフォルト値がこのような方法で保管されると、列上での挿入や更新操作の際に余分な検査が行われるために、若干パフォーマンスが低下します。

列の既存のデータは変更されません。既存のデータを使用可能にして、最小限のスペースを使用してシステム・デフォルト値を保管することの利点を生かすため、オフラインでの表再編成を考慮してください。

OFF
システム・デフォルト値が列の正規値として保管されるように指定します。列の既存のデータは変更されません。既存のデータの変更のために、オフラインでの再編成をお勧めします。

基本データ・タイプは、DATE、TIME、または TIMESTAMP であってはなりません (SQLSTATE 42842)。基本データ・タイプが可変長ストリングの場合には、この文節は無視されます。表が VALUE COMPRESSION に設定されている場合は、長さ 0 のストリング値は自動的に圧縮されます。

変更する表が型付き表である場合、列をスーパー表から継承することはできません (SQLSTATE 428DJ)。

identity-alteration
列の IDENTITY 属性を変更します。その列は IDENTITY 列でなければなりません。

SET INCREMENT BY *numeric-constant*
連続した IDENTITY 列値のインターバルを指定します。次に生成される IDENTITY 列の値は、最後に割り当てられた値に増分を適用することによって決まります。列はあらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができ (SQLSTATE 42815)、長精度整数定数の値を超えることはありません (SQLSTATE 42820)。

この値が負の場合、ALTER ステートメント以降は降順になります。この値が 0 の場合、または正の場合は、 ALTER ステートメント以降は昇順になります。

SET NO MINVALUE または **MINVALUE** *numeric-constant*
降順 IDENTITY 列が値の生成を循環または停止する最小値、あるいは最大値に達した後、昇順 IDENTITY 列が循環する最小値を指定します。列は、指定した表の中に存在していなければならず (SQLSTATE 42703)、あらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

NO MINVALUE
昇順シーケンスの場合、値は元の開始値です。降順シーケンスの場合、列のデータ・タイプの最小値になります。

MINVALUE *numeric-constant*
最小値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができます (SQLSTATE 42815) が、最大値以下でなければなりません (SQLSTATE 42815)。

SET NO MAXVALUE または **MAXVALUE** *numeric-constant*
昇順 IDENTITY 列が値の生成を循環または停止する最大値、あるいは最小値に達した後、降順 IDENTITY 列が循環する最大値を指定します。列は、指定した表の中に存在していなければならず (SQLSTATE 42703)、あらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

NO MAXVALUE
昇順シーケンスの場合、値は列のデータ・タイプの最大値です。降順シーケンスの場合、値は最初の開始値です。

MAXVALUE *numeric-constant*
最大値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができます (SQLSTATE 42815) が、最小値よりも大きいかまたは等しくなければなりません (SQLSTATE 42815)。

SET NO CYCLE または **CYCLE**
その最大値または最小値が生成された後、この IDENTITY 列が値の生成を続行するかどうかを指定します。列は、指定した表の中に存在していなければならず (SQLSTATE 42703)、あらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

NO CYCLE
最大値または最小値に達した後、IDENTITY 列について値が生成されないことを指定します。

CYCLE
最大値または最小値に達した後、この列について値の生成が続行されることを指定します。このオプションが使用されると、昇順 IDENTITY 列が最大値に達した後は、その最小値が生成されます。降順 IDENTITY 列が最小値に達した後は、その最大値が生成されます。 IDENTITY 列の最大値および最小値は、循環に使用される範囲を決定します。

CYCLE が有効な場合、IDENTITY 列について重複する値が生成される可能性があります。ユニーク値が必要であれば (実際には必要ありません)、IDENTITY 列を使用して 1 列のユニーク索引を定義することによって、ユニーク性を確実にしてください。このような IDENTITY 列にユニーク索引が存在し、ユニークではない値が生成されると、エラーが起こります (SQLSTATE 23505)。

SET NO CACHE または **CACHE** *integer-constant*
特定の事前割り振り値を、高速アクセスできるようメモリーに保存するかどうかを指定します。これはパフォーマンスおよびチューニング・オプションです。列はあらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

NO CACHE
IDENTITY 列の値を事前割り振りしないことを指定します。データ共用環境では、IDENTITY 値は要求の順序で生成されなければならず、 NO CACHE オプションを使用する必要があります。

このオプションが指定されると、IDENTITY 列の値はキャッシュに保管されません。この場合、新しい IDENTITY 値が要求されるたびに、ログに対して非同期入出力が行われます。

CACHE *integer-constant*
事前割り振りされ、メモリーに保管される IDENTITY シーケンスの値の数を指定します。 IDENTITY 列について値が生成される場合、値を事前割り振りしてキャッシュに保管しておくと、ログへの非同期入出力が少なくなります。

IDENTITY 列に新しい値が必要でも未使用の値がキャッシュにない場合、値の割り振りはログへの入出力を待機する必要があります。ただし、IDENTITY 列に新しい値が必要で、未使用の値がキャッシュにあれば、その IDENTITY 値の割り振りが、ログへの入出力なしで素早く行われます。

システム障害に起因するものであっても通常のものであっても、データベース非活動化が起こると、コミットされたステートメントで使用されていないキャッシュ済みシーケンス値はすべて失われます (使用されなくなります)。 CACHE オプションに指定する値は、システム障害の際に失われても構わない IDENTITY 列の値の最大数です。

最小値は 2 です (SQLSTATE 42815)。

SET NO ORDER または **ORDER**
要求の順序で IDENTITY 列の値が生成されるかどうかを指定します。列は、指定した表の中に存在していなければならず (SQLSTATE 42703)、あらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。

NO ORDER
要求の順序で IDENTITY 列の値を生成する必要がないことを指定します。

ORDER
要求の順序で IDENTITY 列の値が生成されることを指定します。

RESTART または **RESTART WITH** *numeric-constant*
IDENTITY 列に関連付けられたシーケンスの状態をリセットします。 WITH *numeric-constant* が指定されていないと、IDENTITY 列のシーケンスは、作成されたときに開始値として (暗黙的または明示的のいずれかで) 定義された値で再開始されます。

列は、指定した表の中に存在していなければならず (SQLSTATE 42703)、あらかじめ IDENTITY 属性で定義されていなければなりません (SQLSTATE 42837)。 RESTART は、START WITH の元の値を変更することはありません。

numeric-constant は数値定数で、小数点の右側に非ゼロの数字がない (SQLSTATE 42815) かぎり、この列に割り当てることができる正または負の値にすることができます。 *numeric-constant* が列の次の値として使用されます。

DROP PRIMARY KEY
主キーの定義、およびその主キーに従属するすべての参照制約をドロップします。表には主キーがなければなりません。

DROP FOREIGN KEY *constraint-name*
制約名が *constraint-name* の参照制約をドロップします。 *constraint-name* (制約名) は、参照制約を指定していなければなりません。参照制約のドロップにより起こることについては、 72 ページの『注』を参照してください。

DROP UNIQUE *constraint-name*
ユニーク制約 *constraint-name* の定義、およびこのユニーク制約に従属するすべての参照制約をドロップします。 *constraint-name* には、既存のユニーク制約を指定する必要があります。ユニーク制約のドロップにより起こることについては、 72 ページの『注』を参照してください。

DROP CHECK *constraint-name*
制約名が *constraint-name* のチェック制約をドロップします。 *constraint-name* は、表に定義されている既存のチェック制約を指定していなければなりません。

DROP CONSTRAINT *constraint-name*
制約名が *constraint-name* の制約をドロップします。 *constraint-name* は、表に定義されている既存のチェック制約、参照制約、主キー、またはユニーク制約のいずれかを指定していなければなりません。制約のドロップにより起こることについては、 72 ページの『注』を参照してください。

DROP PARTITIONING KEY
パーティション・キーをドロップします。表にはパーティション・キーがある必要があり、表は単一パーティションのデータベース・パーティション・グループで定義されている表スペースに入っている必要があります。

DROP RESTRICT ON DROP
表、および表を含む表スペースのドロップに関する制約事項を削除します。

DROP MATERIALIZED QUERY
表がマテリアライズ照会表と見なされなくなるように、マテリアライズ照会表を変更します。 *table-name* で指定される表は、複製ではないマテリアライズ照会

表として定義されていなければなりません (SQLSTATE 428EW)。 *table-name* の列の定義は変更されませんが、照会の最適化にこの表を使用することはできなくなり、 REFRESH TABLE ステートメントも使用できなくなります。

DATA CAPTURE
データの複製に関する追加情報をログに記録するか否かを指定します。

表が型付き表である場合、このオプションはサポートされません (ルート表の場合は SQLSTATE 428DH で、他の副表の場合は 428DR)。

NONE
追加情報をログに記録しないことを指定します。

CHANGES
この表に対する SQL 変更についての追加情報をログに書き込むことを指定します。このオプションは、表を複製する場合で、キャプチャー・プログラムを使用してログからこの表に対する変更内容をキャプチャーする場合に必須です。

カタログ・パーティション以外のパーティションにデータが置かれるように表が定義されている場合 (複数パーティションのデータベース・パーティション・グループ、またはカタログ・パーティション以外のパーティションを持つデータベース・パーティション・グループ)、このオプションはサポートされません (SQLSTATE 42997)。

表のスキーマ名 (暗黙または明示名) が 18 バイトより長い場合、このオプションはサポートされません (SQLSTATE 42997)。

INCLUDE LONGVAR COLUMNS
データ複製ユーティリティーが、 LONG VARCHAR または LONG VARGRAPHIC 列に対する変更をキャプチャーするようにします。この文節は、LONG VARCHAR または LONG VARGRAPHIC 列のない表に指定することもできます。これは、LONG VARCHAR または LONG VARGRAPHIC 列を含むよう、表を ALTER することができるためです。

ACTIVATE NOT LOGGED INITIALLY
現行の作業単位の表の NOT LOGGED INITIALLY 属性を活動化します。

このステートメントにより表を変更した後に、同一の作業単位の INSERT、DELETE、UPDATE、CREATE INDEX、DROP INDEX、または ALTER TABLE によって表に対して行われた変更は、ログ記録されません。 NOT LOGGED INITIALLY 属性が活動状態にあるときに、 ALTER ステートメントによってシステム・カタログに対して行われた変更は、ログ記録されます。同一の作業単位内でシステム・カタログ情報に対して行われる一連の変更は、ログ記録されます。

現行の作業単位が完了すると、NOT LOGGED INITIALLY 属性は非活動化され、それ以降の作業単位の表で行われるすべての操作はログ記録されます。

カタログ表へのデータの挿入中にロックを避けるためにこの機能を使用する場合、 ALTER TABLE ステートメントにはこの文節だけを指定してください。ALTER TABLE ステートメントでこの文節以外のものを指定すると、カタログはロックされてしまいます。 ALTER TABLE ステートメントでこの文節のみが指定されている場合、 SHARE ロックのみがシステム・カタログ表で獲得され

ます。これにより、このステートメントが実行される時と、このステートメントが実行される作業単位が終了する時の、所要時間の競合が生じるのを、可能な限り抑えることができます。

表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

NOT LOGGED INITIALLY 属性の詳細については、 『CREATE TABLE』 にあるこの属性に関する記述を参照してください。

注: 活動化された NOT LOGGED INITIALLY 属性を持つ表に対してログに記録されない活動が生じ、ステートメントに障害が起こる (ロールバックが発生する)、または ROLLBACK TO SAVEPOINT が実行される場合には、その作業単位全体がロールバックされます (SQL1476N)。さらに、NOT LOGGED INITIALLY 属性が活動化されている表は、ロールバックされた後にアクセス不能としてマークされ、ドロップしかできなくなります。したがって、NOT LOGGED INITIALLY 属性が活動化されている作業単位内のエラーは、最小限に抑えるべきです。

WITH EMPTY TABLE
現在表にあるすべてのデータを除去します。一度データが除去されると、RESTORE 機能を使用しなければ、そのデータの回復を行うことができません。この ALTER ステートメントを発行した作業単位をロールバックしても、表データは元の状態には回復できません。

この処置が必要な場合、対象の表に定義された DELETE トリガーは行われません。その表にある索引もすべて空になります。

PCTFREE *integer*
ロードまたは表再編成操作において、各ページに残すフリー・スペースのパーセンテージを指定します。各ページの最初の行は、制約なしに追加されます。ページに行が追加される際には、少なくとも *integer* で指定された分 (%) のフリー・スペースがページに残されます。 PCTFREE 値は、load および table reorg ユーティリティーでのみ考慮されます。 *integer* の値は 0 ～ 99 です。カタログ (SYSCAT.TABLES) の PCTFREE 値 -1 は、デフォルト値として解釈されます。表ページのデフォルト PCTFREE 値は 0 です。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

LOCKSIZE
表へのアクセス時に使用されるロックのサイズ (細分性) を指定します。表定義でこのオプションを使用しても、通常のロック・エスカレーションが行われます。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

ROW
行ロックの使用を指定します。これは、表の作成時のデフォルトのロック・サイズです。

TABLE
表ロックの使用を指定します。これは、適切な共用ロックまたは排他ロックが表で獲得されており、意図ロック (″意図なし″ は除く) が使用されないことを意味します。この値を使用すると、獲得すべきロック数が限定されるた

め、照会のパフォーマンスが向上します。しかし、すべてのロックが表全体に対して獲得されるので、並行性も限定されます。

APPEND
データを表データの終わりに追加するか、またはデータ・ページの使用可能なフリー・スペースを位置に追加するかを指定します。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

ON
表データが追加され、各ページのフリー・スペース情報は保持されません。表にはクラスター索引があってはなりません (SQLSTATE 428CA)。

OFF
表データは使用可能なスペースに入れられます。これは、表の作成時のデフォルト値です。

APPEND OFF を設定した後に表の再編成が必要となります。これは、使用可能なフリー・スペース情報が不正確となるため、データ挿入時のパフォーマンスの低下につながるからです。

VOLATILE CARDINALITY または **NOT VOLATILE CARDINALITY**
表 *table-name* のカーディナリティーが、ランタイムに相当に変化し得るのかどうかをオプティマイザーに知らせます。揮発性は、表そのものに対してではなく、表の行数に適用されます。 CARDINALITY はオプショナル・キーワードです。デフォルトは NOT VOLATILE です。

VOLATILE
表 *table-name* のカーディナリティーが、空から非常に大きなものに至るまで、ランタイムに変化し得ることを知らせます。表にアクセスするために、オプティマイザーは、その統計に関係なく、表のスキャンではなく索引のスキャンを使います。ただし、その場合、その索引は索引専用である (参照されるすべての列がその索引内にある) か、索引のスキャンで述部を適用できることが条件になります。リスト・プリフェッチ・アクセス方式は、この表へのアクセスには使用されません。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

NOT VOLATILE
table-name のカーディナリティーが揮発性でないことを指定します。この表へのアクセス・プランは、既存の統計と、現行の最適化レベルに基づいて続けられます。

VALUE COMPRESSION
たいていのデータ・タイプで NULL および長さ 0 のデータ値を、さらに効率的に保管するかどうかを指定します。また、使用される行形式を判別します。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

ACTIVATE
データ・タイプが BLOB、CLOB、DBCLOB、LONG VARCHAR、または LONG VARGRAPHIC の列の長さ 0 のデータ値を最小限のスペースを使用して保管するよう指定します。 NULL 値は、追加バイトを使用することなく、それぞれ保管されます。これをサポートする行形式は、各データ・タイ

プのバイト・カウントを識別し、更新の際にデータ・フラグメントの原因となる傾向があります。新しい行形式 (COMPRESS SYSTEM DEFAULT オプションによって列を指定する) を使用しても、列のシステム・デフォルト値をより効率的に保管できます。

DEACTIVATE
今後の更新のためのスペースを確保するように NULL 値を指定します。このスペースは、可変長の列のためには確保されません。使用される行形式は、各データ・タイプのバイト・カウントを識別します。またそのようにすると、列のシステム・デフォルト値のストレージを効率的にサポートしません。列が COMPRESS SYSTEM DEFAULT 属性を既に有していると、警告が出されます (SQLSTATE 01648)。

更新操作により、既存の行を新しい行形式に変更します。既存の行の更新操作のパフォーマンスを向上させるには、オフラインでの表再編成をお勧めします。

LOG INDEX BUILD
この表で索引の作成、再作成、または再編成の操作を行う際に実行されるロギングのレベルを指定します。

NULL
logindexbuild データベース構成パラメーターの値を使用して、索引作成操作を完全にログに記録するかどうかを指定します。これは、表が作成されるときのデフォルトです。

OFF
この表での索引作成操作が最小限ログに記録されることを指定します。この値は、*logindexbuild* データベース構成パラメーターの設定をオーバーライドします。

ON
この表での索引作成操作が完全にログに記録されることを指定します。この値は、*logindexbuild* データベース構成パラメーターの設定をオーバーライドします。

規則:

- 表に対して定義されたユニーク・キー制約または主キー制約は、パーティション・キー (存在する場合) のスーパーセットである必要があります (SQLSTATE 42997)。
- 主キーまたはユニーク・キーは、ディメンションのサブセットにはなりません (SQLSTATE 429BE)。
- 1 つの列の参照は、 1 つの ALTER TABLE ステートメント内の 1 つの ADD または ALTER COLUMN 文節でのみ可能です (SQLSTATE 42711)。
- 表にマテリアライズ照会表があり、その表に従属している場合、列の長さを変更することはできません (SQLSTATE 42997)。
- 以下のことを行う前に、SET INTEGRITY ステートメントを使用して、表をチェック・ペンディング状態に設定しなければなりません (SQLSTATE 55019)。
  - 生成式を使って列を追加する
  - 列の生成式を変更する

– 生成式を持つよう、列を変更する

注:

- 表をマテリアライズ照会表に変更すると、この表はチェック・ペンディング状態になります。表が REFRESH IMMEDIATE として定義されている場合、この表はチェック・ペンディング状態を解除しない限り、全選択で参照されている表に対して INSERT、DELETE、または UPDATE コマンドを実行することはできません。 IMMEDIATE CHECKED オプションを指定して REFRESH TABLE または SET INTEGRITY を使用することで、表のチェック・ペンディング状態を解除し、全選択に基づいて表内のデータを完全にリフレッシュできます。表にあるデータが完全に全選択の結果を反映する場合、 SET INTEGRITY の IMMEDIATE UNCHECKED オプションを使用して、表のチェック・ペンディング状態を解除できます。
- 表を変更して REFRESH IMMEDIATE マテリアライズ照会表にすると、全選択により参照される表に対して INSERT、DELETE、または UPDATE を使用するパッケージはどれも無効になります。
- 表をマテリアライズ照会表から正規表に変更すると、表に関連するパッケージはどれも無効になります。
- 表を MAINTAINED BY FEDERATED_TOOL マテリアライズ照会表から正規表に変更しても、レプリケーション・ツールのサブスクリプション・セットアップには変更は生じません。 MAINTAINED BY SYSTEM マテリアライズ照会表に対する後続の変更は、レプリケーション・ツールの失敗をもたらすので、MAINTAINED BY FEDERATED_TOOL マテリアライズ照会表の変更の際には、サブスクリプション設定を変更する必要があります。
- 据え置かれたマテリアライズ照会表がステージング表に関連付けられる場合、マテリアライズ照会表が正規表に変更されると、ステージング表はドロップされます。
- ADD 列文節は、他のいずれの文節よりも先に処理されます。他の文節は、指定された順序で処理されます。
- ALTER TABLE によって追加される列は、表の既存のビューに自動的に追加されることはありません。
- ユニーク・キー制約または主キー制約に関して索引が自動的に作成される場合、データベース・マネージャーは、指定された制約名を表のスキーマ名と一致するスキーマ名を伴う索引名として使用することを試みます。この名前が既存の索引名と一致する場合、または制約の名前が指定されなかった場合、索引は SYSIBM スキーマに作成され、 ″SQL″ とタイム・スタンプに基づいて生成される 15 個の数字からなるシステム生成の名前が付けられます。
- 表 T での DELETE 操作に関係する可能性のある表は、 T に*連結削除* されている、と言われます。つまり、ある表が T の従属表であるか、または T からの削除のカスケード先の表の従属表である場合、この表は T に対して連結削除されることになります。
- パッケージの中に挿入 (更新/削除) の使用があるといわれるのは、パッケージ内のステートメントによって直接に、あるいは、そのいずれかのステートメントの代わりにパッケージによって実行される制約やトリガーによって間接的に、レコードが T に挿入 (更新または削除) される場合です。同様に、パッケージの中に更新の使用があるといわれるのは、パッケージ内のステートメントによって直接

に、あるいは、そのいずれかのステートメントの代わりにパッケージによって実行される制約やトリガーによって間接的に、列が変更される場合です。

- フェデレーテッド・システムでは、透過性 DDL を使用して作成されたリモート基本表は変更できます。ただし、可能な変更に関して、透過 DDL には以下のようないくつかの制限があります。
  - リモート基本表は、新規の列の追加か、または主キーの指定によってのみ、変更できます。
  - リモート基本表内の既存の列には、コメントを指定できません。
  - リモート基本表内の既存の主キーは、変更またはドロップできません。
  - リモート基本表の変更を行うと、そのリモート基本表表に関連したニックネームに従属するパッケージはいずれも無効になります。
  - リモート・データ・ソースは、ALTER TABLE ステートメントを通じて要求された変更をサポートする必要があります。データ・ソースの、サポートしていない要求への応答方法によって、エラーが返されるか、または要求が無視される可能性があります。
  - 透過性 DDL を使用して作成されたのではないリモート基本表を変更しようとすると、エラーが返されます。
- 主キー、ユニーク・キー、または外部キーに対する変更は、パッケージ、索引、およびその他の外部キーに以下の影響を与えます。
  - 主キーまたはユニーク・キーが追加された場合、
    - パッケージ、外部キー、または既存のユニーク・キーに影響はありません。(主キーまたはユニーク・キーが、前のバージョンで作成された既存のユニーク索引を使用しており、ユニーク性の据え置きをサポートするように変換されていない場合、索引は変換され、関連した表の更新を行うパッケージは無効になります。)
  - 主キーまたはユニーク・キーがドロップされた場合、
    - 制約に関してその索引が自動的に作成されていた場合には、その索引はドロップされます。索引に従属しているパッケージはすべて無効になります。
    - 索引が制約に関してユニークであるように変換されており、現在システムが索引に関してユニークであることを必要としていない場合、索引は非ユニークに戻されます。索引に従属しているパッケージはすべて無効になります。
    - 索引が制約のために使用された既存のユニーク索引だった場合、索引はシステムが必要としていないことを示すよう設定されます。パッケージに影響はありません。
    - すべての従属外部キーはドロップされます。次の項目に示すように、各従属外部キーごとに、さらにアクションが取られます。
  - 外部キーが追加される、ドロップされる、または NOT ENFORCED から ENFORCED に (または ENFORCED から NOT ENFORCED に) 変更される場合:
    - オブジェクト表に対して挿入を行うパッケージは、すべて無効になります。
    - 外部キー内の少なくとも 1 つの列に対して更新使用の指定があるパッケージは、すべて無効になります。
    - 親表の削除を行うパッケージはすべて無効になります。

- 親キーの少なくとも 1 つの列に対して更新使用の指定があるパッケージは、すべて無効になります。

– 外部キーまたは機能従属関係が、 ENABLE QUERY OPTIMIZATION から DISABLE QUERY OPTIMIZATION に変更される場合:

- 最適化のための制約と従属関係にあるパッケージすべては、無効です。

– 表に列を追加すると、変更された表に対して挿入を行うパッケージはすべて無効になります。追加された列が、表内の最初のユーザー定義構造タイプ列である場合、変更された表で DELETE を行おうとするパッケージは無効になります。

– チェック・ペンディング状態ではない既存の表に対してチェック制約または参照制約を追加するか、またはチェック・ペンディング状態にない既存の表の既存のチェック制約か参照制約を NOT ENFORCED から ENFORCED に変更すると、その表の既存の行は、制約に関して直ちに評価されます。検証に失敗すると、エラー (SQLSTATE 23512) になります。表がチェック・ペンディング状態の場合は、チェック制約または参照制約を追加しても、または制約を NOT ENFORCED から ENFORCED に変更しても、制約が直ちに適用されるわけではありません。その場合には、チェック・ペンディング操作で使用された制約タイプ・フラグのうち対応するものが更新されます。制約の適用を開始するには、SET INTEGRITY ステートメントを発行します。

– チェック制約の追加、変更、またはドロップを行うと、対象の表に対する挿入、制約に関係している少なくとも 1 つの列に対する更新、またはパフォーマンスを向上させるための制約の選択使用のいずれかを含むすべてのパッケージが無効になります。

– パーティション・キーを追加すると、パーティション・キーの少なくとも 1 つの列に対して更新使用の指定があるパッケージは、すべて無効になります。

– デフォルトで主キーの最初の列を使用して定義されたパーティション・キーは、主キーのドロップや異なる主キーの追加によっては影響を受けません。

– 列の長さを変更して長くすると、変更された列を含む表を (参照制約またはトリガーによって直接または間接的に) 参照するパッケージはすべて無効になります。

– 列の長さを変更して長くすると、表に従属するビュー (型付きビューを除く) が再生成されます。ビューの再生成時にエラーが生じると、エラーが戻されます (SQLSTATE 56098)。表に従属する型付きビューは、作動不能としてマークされます。

– 列の長さを長く変更すると、トリガーを含むステートメントを準備中またはバインド中に、トリガー処理でエラー (SQLSTATE 54010) が発生する可能性があります。このことは、遷移変数および遷移表列の長さの合計に基づく行の長さが長すぎる場合に生じます。このようなトリガーがドロップされると、それ以降にトリガーを作成しようとしてもエラー (SQLSTATE 54040) となります。

– それぞれ 4000 および 2000 より大きい数値に変更された VARCHAR および VARGRAPHIC 列は、 SYSFUN スキーマの関数での入力パラメーターとして使用しないでください (SQLSTATE 22001)。

– 構造タイプ列を変更してインライン長を長くすると、参照制約またはトリガーによって直接または間接的に表を参照するパッケージはすべて無効になります。
– 構造タイプ列を変更してインライン長を長くすると、表に従属するビューは再生成されます。
– 表の LOCKSIZE を変更すると、変更された表に従属するすべてのパッケージは無効になります。
– ACTIVATE NOT LOGGED INITIALLY 文節は、 FILE LINK CONTROL 属性のある DATALINK 列を表に追加するときには使用できません (SQLSTATE 42613)。
– VOLATILE または NOT VOLATILE CARDINALITY を変更すると、変更された表に従属するすべてのパッケージは無効になります。
– 複製を行うユーザーは、 VARCHAR 列の長さが長くなるときに、特に注意する必要があります。アプリケーション表と関連付けられた変更データ表は、すでに DB2 行サイズの限界近くに設定されている可能性があります。変更データ表をアプリケーション表よりも前に変更するか、これら 2 つを同じ作業単位内で変更するようにして、両方の表で変更が完了できるようにしてください。コピーについても考慮すべき点があります。これも、行サイズの限界近くに設定されていたり、既存の列の長さを長くする機能のないプラットフォームに存在している可能性があります。

  VARCHAR 列の長さを長くしたログ・レコードをキャプチャー・プログラムが処理する前に、変更データ表を変更していなければ、キャプチャー・プログラムが失敗する場合があります。コピーを保持しているサブスクリプションを実行する前に、 VARCHAR 列が含まれるコピーを変更していなければ、そのサブスクリプションは失敗する可能性があります。
– ***互換性***
  - 以前のバージョンの DB2 との互換性:
    • 以下についての ADD キーワードはオプションです。
      – 名前のない PRIMARY KEY 制約
      – 名前のない参照制約
      – FOREIGN KEY 句の後に名前を指定した参照制約
    • CONSTRAINT キーワードは、参照文節を定義する *column-definition* から省略できます。
    • *constraint-name* (制約名) を FOREIGN KEY に続けて (CONSTRAINT キーワードなし) 指定することができます。
    • SET MATERIALIZED QUERY AS の代わりに SET SUMMARY AS を指定できます。
    • DROP MATERIALIZED QUERY の代わりに SET MATERIALIZED QUERY AS DEFINITION ONLY を指定できます。
    • ADD MATERIALIZED QUERY (全選択) の代わりに SET MATERIALIZED QUERY AS (全選択) を指定できます。
  - 以前のバージョンの DB2 との互換性と整合性:

- *identity-options* 文節では、コンマを使って複数のオプションを分離することができます。

- 以下の構文もサポートされています。
  - NOMINVALUE、NOMAXVALUE、NOCYCLE、NOCACHE、および NOORDER

例:

例 1: 1 文字の長さの RATING という名前の新しい列を、DEPARTMENT 表に追加します。

```
ALTER TABLE DEPARTMENT
   ADD RATING CHAR(1)
```

例 2: SITE_NOTES という名前の新しい列を PROJECT 表に追加します。SITE_NOTES は、最大 1000 文字の長さの可変長列として作成します。この列の値には関連する文字セットがなく、変換されません。

```
ALTER TABLE PROJECT
   ADD SITE_NOTES  VARCHAR(1000) FOR BIT DATA
```

例 3: 以下の列が定義された EQUIPMENT という表が存在するものと想定します。

列名	データ・タイプ
EQUIP_NO	INT
EQUIP_DESC	VARCHAR(50)
LOCATION	VARCHAR(50)
EQUIP_OWNER	CHAR(3)

EQUIPMENT 表に、所有者 (EQUIP_OWNER) は DEPARTMENT 表に存在する部門番号 (DEPTNO) でなければならない、という参照制約を追加します。 DEPTNO は、DEPARTMENT 表の主キーです。 DEPARTMENT 表からある部門を削除する場合は、その部門の所有するすべての備品の所有者 (EQUIP_OWNER) の値を割り当て解除する必要があります (つまり NULL 値に設定する必要があります)。制約の名前は、DEPTQUIP です。

```
ALTER TABLE EQUIPMENT
     ADD CONSTRAINT DEPTQUIP
      FOREIGN KEY (EQUIP_OWNER)
        REFERENCES DEPARTMENT
           ON DELETE SET NULL
```

さらに、備品レコードに関係した数量を記録できるようにするため、追加の列が必要になります。特に指定されない限り、EQUIP_QTY 列には値 1 を入れます。NULL 値にしてはなりません。

```
ALTER TABLE EQUIPMENT
  ADD COLUMN EQUIP_QTY
  SMALLINT NOT NULL DEFAULT 1
```

例 4: 表 EMPLOYEE を変更します。各従業員の給与と歩合の合計が $30,000 を超えていなければならない、という定義済みの REVENUE という名前のチェック制約を追加します。

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT REVENUE
  CHECK (SALARY + COMM > 30000)
```

例 5: 表 EMPLOYEE を変更します。前に定義した制約 REVENUE をドロップします。

```
ALTER TABLE EMPLOYEE
  DROP CONSTRAINT REVENUE
```

例 6: SQL の変更内容をデフォルトのフォーマットでログに記録するように表を変更します。

```
ALTER TABLE SALARY1
  DATA CAPTURE NONE
```

例 7: SQL の変更内容を拡張フォーマットでログに記録するように表を変更します。

```
ALTER TABLE SALARY2
  DATA CAPTURE CHANGES
```

例 8: EMPLOYEE 表を変更して、デフォルト値を指定して 4 つの新しい列を追加します。

```
ALTER TABLE EMPLOYEE
   ADD COLUMN HEIGHT MEASURE   DEFAULT MEASURE(1)
   ADD COLUMN BIRTHDAY BIRTHDATE DEFAULT DATE('01-01-1850')
   ADD COLUMN FLAGS BLOB(1M)  DEFAULT BLOB(X'01')
   ADD COLUMN PHOTO PICTURE   DEFAULT BLOB(X'00')
```

デフォルト値の指定時に、これらのデフォルト値はさまざまな関数名を使用します。 MEASURE は INTEGER に基づく特殊タイプなので、MEASURE 関数を使用しています。ちなみに、HEIGHT 列のデフォルト値は、関数を使用しなくても指定することができたはずです。 MEASURE のソース・タイプは、BLOB または日時データ・タイプではないからです。 BIRTHDATE は DATE に基づく特殊タイプなので、 DATE 関数を使用しています (この場合、BIRTHDATE は使用できません)。 FLAGS 列と PHOTO 列では、PHOTO が特殊名であるにもかかわらず、BLOB 関数を使用してデフォルト値が指定されています。 BIRTHDAY、FLAGS、および PHOTO 列のデフォルト値を指定するためには、関数を使用する必要があります。タイプが、BLOB や日時データ・タイプのソースに基づく BLOB や特殊タイプだからです。

例 9: 以下の列のある CUSTOMERS という表が定義されます。

列名	データ・タイプ
BRANCH_NO	SMALLINT
CUSTOMER_NO	DECIMAL(7)
CUSTOMER_NAME	VARCHAR(50)

この表では、主キーは BRANCH_NO 列と CUSTOMER_NO 列からなります。表をパーティション化するには、表に対してパーティション・キーを作成する必要があります。表は単一のデータベース・パーティションからなるデータベース・パーティション・グループの表スペースに定義する必要があります。主キーは、パーティション列のスーパーセットである必要があり、主キーの少なくとも 1 つの列がパーティション・キーとして使用されている必要があります。以下のようにして、BRANCH_NO をパーティション・キーとして定義します。

```
ALTER TABLE CUSTOMERS
  ADD PARTITIONING KEY (BRANCH_NO)
```

例 10: リモート表 EMPLOYEE が、フェデレーテッド・システムに透過 DDL を使用して作成されました。リモート表 EMPLOYEE を変更して、列 PHONE_NO および WORK_DEPT を追加します。また、主キーを既存の列 EMP_NO および新規列 WORK_DEPT に追加します。

```
ALTER TABLE EMPLOYEE
  ADD COLUMN PHONE_NO CHAR(4) NOT NULL
  ADD COLUMN WORK_DEPT CHAR(3)
  ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

例 11: DEPARTMENT 表を変更して機能従属関係 FD1 を追加し、次いで DEPARTMENT 表から機能従属関係をドロップします。

```
ALTER TABLE DEPARTMENT
  ADD COSTRAINT FD1
    CHECK ( DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED

ALTER TABLE DEPARTMENT
  DROP CHECK FD1
```

例 12: EMPLOYEE 表の WORKDEPT 列のデフォルト値を 123 に変更します。

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN WORKDEPT
    SET DEFAULT '123'
```

関連概念:

- *フェデレーテッド・システム・ガイド* の『透過 DDL とは ?』

関連タスク:

- *フェデレーテッド・システム・ガイド* の『透過 DDL を使用したリモート表の変更』

関連資料:

- 87 ページの『ALTER TYPE (構造化)』
- 347 ページの『CREATE TABLE』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』
- *SQL 管理ルーチン* の『ALTOBJ プロシージャー』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『TbGenCol.java -- How to use generated columns (JDBC)』

ALTER TABLESPACE

ALTER TABLESPACE ステートメントは、以下の方法で既存の表スペースを変更する場合に使用されます。

- データベース管理表スペース (DMS) (つまり MANAGED BY DATABASE オプションによって作成される表スペース) にコンテナーを追加する、またはデータベース管理表スペース (DMS) からコンテナーをドロップする。
- データベース管理表スペース (DMS) のコンテナーのサイズを変更する。
- コンテナーのないパーティション上のシステム管理表スペース (SMS) にコンテナーを追加する。
- 表スペースの PREFETCHSIZE 設定値を変更する。
- 表スペースの表に対して使用する BUFFERPOOL を変更する。
- 表スペースの OVERHEAD 設定値を変更する。
- 表スペースの TRANSFERRATE 設定値を変更する。
- 表スペースに対するファイル・システムのキャッシング・ポリシーを変更する。

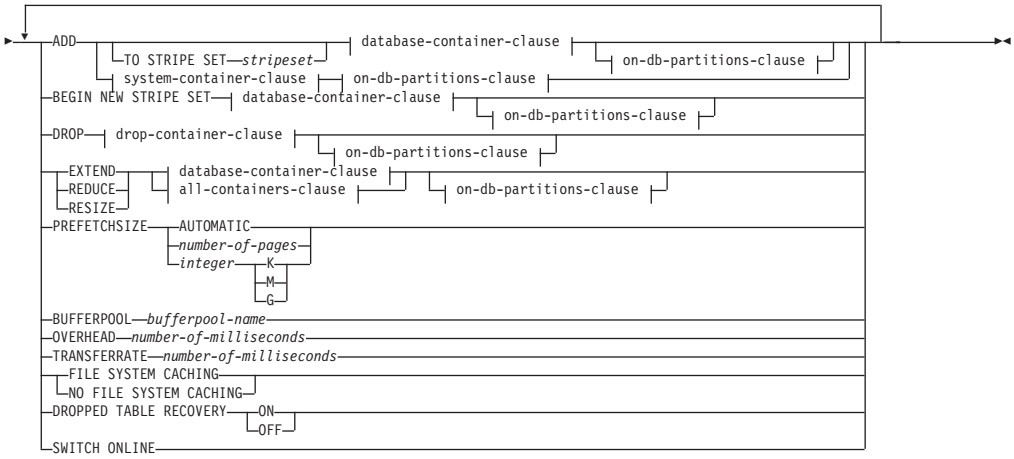
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

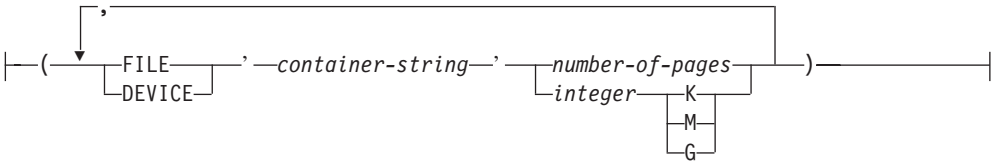
許可:

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

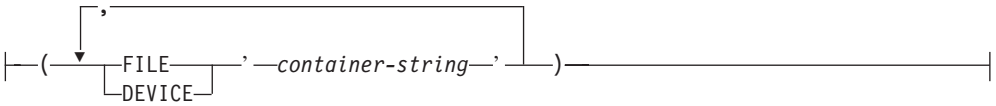
構文:



database-container-clause:

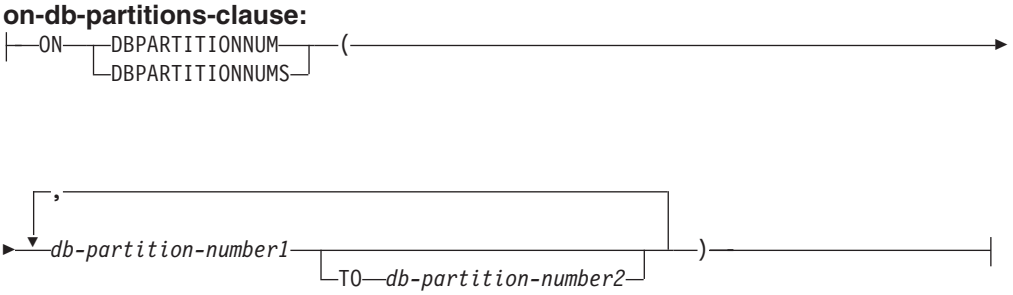


drop-container-clause:

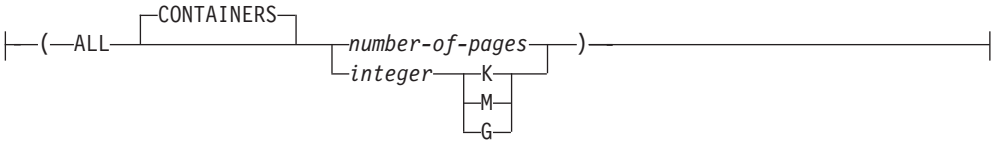


system-container-clause:

on-db-partitions-clause:



all-containers-clause:



説明:

tablespace-name
表スペースの名前を指定します。これは、1 つの部分からなる名前です。これは、長形式 SQL ID です (通常 ID または区切り ID のいずれか)。

ADD
表スペースに 1 つか複数の新しいコンテナーを追加するように指定します。

TO STRIPE SET *stripeset*
1 つか複数の新しいコンテナーを表スペースに追加し、指定されたストライプ・セットに配置するように指定します。

BEGIN NEW STRIPE SET
表スペースに新しいストライプ・セットを作成し、その新しいストライプ・セットに 1 つか複数のコンテナーを追加するように指定します。 ADD オプションを使用して後に追加されるコンテナーは、 TO STRIPE SET が指定されない場合にはこの新しいストライプ・セットに追加されます。

DROP
1 つか複数のコンテナーを表スペースからドロップするように指定します。

EXTEND
既存のコンテナーのサイズを増やすように指示します。指定するサイズは、既存のコンテナーに追加されるサイズです。 *all-containers-clause* が指定されると、表スペースにあるすべてのコンテナーがこのサイズで拡張されます。

REDUCE
既存のコンテナーのサイズを減らすように指示します。指定するサイズは、既存のコンテナーから減らすサイズです。 *all-containers-clause* が指定されると、表スペースにあるすべてのコンテナーがこのサイズだけ縮小されます。

RESIZE
既存のコンテナーのサイズを変更するよう指定します。指定されるサイズが、コンテナーの新しいサイズになります。 *all-containers-clause* が指定されると、表スペースにあるすべてのコンテナーがこのサイズに変更されます。この操作が複数のコンテナーに影響を与える場合には、そうしたコンテナーすべてのサイズは増やすか減らすかのどちらかにする必要があります。一部を増やし、その他を減らすことはできません (SQLSTATE 429BC)。

database-container-clause
データベース管理表スペース (DMS) に 1 つまたは複数のコンテナーを追加します。表スペースは、すでにアプリケーション・サーバーに存在するデータベース管理表スペース (DMS) を指定するものでなければなりません。

drop-container-clause
1 つまたは複数のコンテナーをデータベース管理表スペース (DMS) からドロップします。表スペースは、すでにアプリケーション・サーバーに存在するデータベース管理表スペース (DMS) を指定するものでなければなりません。

system-container-clause
指定のパーティションにあるシステム管理表スペース (SMS) に、1 つまたは複数のコンテナーを追加します。表スペースは、すでにアプリケーション・サーバーに存在するシステム管理表スペース (SMS) を指定するものでなければなりません。表スペースに対して指定するパーティションにコンテナーがあってはなりません (SQLSTATE 42921)。

on-db-partitions-clause
1 つまたは複数のパーティションに対応するコンテナー操作を指定します。

all-containers-clause
データベース管理表スペース (DMS) にあるコンテナーすべてを拡張、縮小またはサイズ変更します。表スペースは、すでにアプリケーション・サーバーに存在するデータベース管理表スペース (DMS) を指定するものでなければなりません。

PREFETCHSIZE
プリフェッチでは、照会に必要なデータがその照会で参照される前に読み取られるため、照会では入出力の実行を待たずにすみます。

AUTOMATIC
表スペースのプリフェッチ・サイズが自動的に更新されるように指定します。プリフェッチ・サイズは、次の公式を使って DB2 により管理されます。

```
プリフェッチ・サイズ =
 (コンテナーの数) *
 (コンテナーごとの物理ディスクの数) *
 (エクステント・サイズ)
```

コンテナーごとの物理ディスクの数のデフォルトは、 DB2_PARALLEL_IO レジストリー変数によって指定されているのでない限り、1 です。

表スペース内のコンテナー数が変化するたびに (1 つ以上のコンテナーを追加またはドロップする ALTER TABLESPACE ステートメントの正常実行に続いて)、 DB2 はプリフェッチ・サイズを自動的に更新します。プリフェッチ・サイズはデータベースの開始時に更新されます。

PREFETCHSIZE 文節に数値を指定することにより、プリフェッチ・サイズの自動更新をオフにすることができます。

number-of-pages
データのプリフェッチの実行中に、表スペースから読み取られる PAGESIZE ページの数を指定します。このプリフェッチ・サイズ値は、後に K (K バイトの場合)、M (M バイトの場合)、または G (G バイトの場合) を付けた整数値としても指定できます。このように指定した場合、バイト数をページ・サイズで割った値よりも小さいか等しい整数で、最大の整数値が、プリフェッチ・サイズのページ値の数を判別するために使用します。

BUFFERPOOL *bufferpool-name*
この表スペースの表に対して使用するバッファー・プールの名前を指定します。バッファー・プールは、現在データベースに存在している必要があります (SQLSTATE 42704)。バッファー・プールに対して、この表スペースのデータベース・パーティション・グループを定義する必要があります (SQLSTATE 42735)。

OVERHEAD *number-of-milliseconds*
入出力制御装置のオーバーヘッドとディスク・シーク待ち時間をミリ秒単位で指定する数値リテラルです (整数、10 進数、または浮動小数点数)。この数値がすべてのコンテナーで同一でない場合、それは表スペースに属するすべてのコンテナーの平均でなければなりません。この値は、照会の最適化の過程で入出力コストを判別するのに使用されます。

TRANSFERRATE *number-of-milliseconds*
1 ページ (4K または 8K) をメモリーに読み込むための時間をミリ秒単位で指定する数値リテラルです (整数、 10 進数、または浮動小数点数)。この数値がすべてのコンテナーで同一でない場合、それは表スペースに属するすべてのコンテナーの平均でなければなりません。この値は、照会の最適化の過程で入出力コストを判別するのに使用されます。

FILE SYSTEM CACHING または **NO FILE SYSTEM CACHING**
入出力操作をファイル・システム・レベルでキャッシュに入れるかどうかを指定します。新しいキャッシング・ポリシーが有効になる前に、データベースへの接続を終了する必要があります。

FILE SYSTEM CACHING
ターゲット表スペース内のすべての入出力操作が、ファイル・システム・レベルでキャッシュに入れられます。

NO FILE SYSTEM CACHING
すべての入出力操作がファイル・システム・レベルのキャッシュをう回します。

DROPPED TABLE RECOVERY
指定された表スペースからドロップされた表は、 ROLLFORWARD コマンドの RECOVER DROPPED TABLE ON オプションを使用して回復させることができます。

SWITCH ONLINE
OFFLINE 状態の表スペースは、コンテナーがアクセス可能であれば、オンラインになります。コンテナーがアクセス可能でなければ、エラーが戻されます (SQLSTATE 57048)。

注:

- ***互換性***
  - バージョン 8 より前のバージョンのキーワードとの互換性:
    - DBPARTITIONNUM を NODE に置換できます。
    - DBPARTITIONNUMS を NODES に置換できます。
- 各コンテナー定義には、53 バイトに加えて、コンテナー名を保管するのに必要なバイト数が必要です。表スペースのすべてのコンテナー名を結合した長さは、20 480 バイトを超えることはできません (SQLSTATE 54034)。
- デフォルトのコンテナー操作は、ALTER TABLESPACE ステートメントで指定されるコンテナー操作ですが、この操作は特定のデータベース・パーティションに明示的に向けられません。こうしたコンテナー操作は、ステートメントにリストされていない任意のデータベース・パーティションに向けられます。デフォルトのコンテナー操作がどのデータベース・パーティションにも向けられない場合には、コンテナー操作ではすべてのデータベース・パーティションに明示的に言及するので、警告が出されます (SQLSTATE 1758W)。
- スペースが表スペースに追加または、表スペースから削除され、トランザクションがコミットされると、表スペースのコンテナーはコンテナー間でバランスの再調整がなさるかもしれません。バランス再調整中も、表スペースへのアクセスは制限されません。
- 表スペースが OFFLINE 状態で、コンテナーがアクセス可能である場合、すべてのアプリケーションを切断してから、もう一度データベースへ接続すれば、表スペースは OFFLINE 状態から脱することができます。別の方法として、SWITCH ONLINE オプションを使用すると、残りのデータベースは稼働状態で使用中のまま、表スペースは OFFLINE から脱する (稼働状態になる) ことができます。
- 表スペースに複数のコンテナーを追加する場合は、バランスの再調整のコストが一度だけで済むように、これらのコンテナーを同じステートメントで追加することをお勧めします。単一トランザクションで別々の ALTER TABLESPACE ステートメントを使用して、同じ表スペースにコンテナーを追加するとエラーになります (SQLSTATE 55041)。

- 存在しないコンテナーについて拡張、縮小、またはサイズ変更、またはドロップをしようとすると、エラーが発生します (SQLSTATE 428B2)。
- コンテナーを拡張、軽減またはサイズ変更する場合、このコンテナー・タイプは、コンテナーが作成されたときに使用されたタイプと適合しなければなりません (SQLSTATE 428B2)。
- 1 つのトランザクションで、同じ表スペースに対して別個の ALTER TABLESPACE ステートメントを使用して、複数のコンテナー・サイズを変更しようとすると、エラーが発生します (SQLSTATE 55041)。
- パーティション・データベースで、複数のデータベース・パーティションが同じ物理ノードに存在する場合、このようなデータベース・パーティションに同じデバイスまたは特定のパスを指定することはできません (SQLSTATE 42730)。この環境の場合、それぞれのデータベース・パーティションごとにユニークな *container-string* を指定するか、または相対パス名を使用してください。
- 表スペース定義はトランザクションであり、表スペース定義に対する変更はコミット時にカタログ表に反映されますが、新しい定義のバッファー・プールは、データベースの次回始動時まで使用することはできません。 ALTER TABLESPACE ステートメントが出されたときに使用中のバッファー・プールは、それまで引き続き使用されます。

規則:

- BEGIN NEW STRIPE SET 文節は、 ADD、DROP、EXTEND、REDUCE、および RESIZE 文節が別のパーティションに向けられない限りは、それらの文節と同じステートメントでは指定できません (SQLSTATE 429BC)。
- TO STRIPE SET 文節を使用して指定されたストライプ・セットは、変更される表スペースの有効な範囲内になければなりません (SQLSTATE 42615)。
- 表スペースにスペースを追加、または表スペースからスペースを削除する場合、以下の規則に従います。
  - EXTEND および RESIZE は、各コンテナーのサイズを拡張する、同じステートメントで使用できます (SQLSTATE 429BC)。
  - REDUCE および RESIZE は、各コンテナーのサイズを軽減する、同じステートメントで使用できます (SQLSTATE 429BC)。
  - EXTEND および REDUCE は、異なるパーティションに向けられない限りは、同じステートメントでは使用できません (SQLSTATE 429BC)。
  - ADD は、REDUCE または DROP が異なるパーティションに向けられない限りは、それらと共に同じステートメントでは使用できません (SQLSTATE 429BC)。
  - DROP は、EXTEND または ADD が異なるパーティションに向けられない限りは、それらと共に同じステートメントでは使用できません (SQLSTATE 429BC)。

例:

例 1: PAYROLL 表スペースにデバイスを追加します。

```
ALTER TABLESPACE PAYROLL
  ADD (DEVICE '/dev/rhdisk9' 10000)
```

例 2: ACCOUNTING 表スペースのプリフェッチ・サイズと入出力オーバーヘッドを変更します。

```
ALTER TABLESPACE ACCOUNTING
  PREFETCHSIZE 64
  OVERHEAD 19.3
```

例 3: 表スペース TS1 を作成した後、コンテナーをサイズ変更して、すべてのコンテナーのサイズが 2000 ページになるようにします。 (このサイズ変更を実行する 3 つの異なる ALTER TABLESPACE ステートメントを示します。)

```
CREATE TABLESPACE TS1
  MANAGED BY DATABASE
  USING (FILE '/conts/cont0' 1000,
         DEVICE '/dev/rcont1' 500,
         FILE 'cont2' 700)
ALTER TABLESPACE TS1
  RESIZE (FILE '/conts/cont0' 2000,
          DEVICE '/dev/rcont1' 2000,
          FILE 'cont2' 2000)
```

または

```
ALTER TABLESPACE TS1
  RESIZE (ALL 2000)
```

または

```
ALTER TABLESPACE TS1
  EXTEND (FILE '/conts/cont0' 1000,
          DEVICE '/dev/rcont1' 1500,
          FILE 'cont2' 1300)
```

例 4: DATA_TS 表スペースにあるすべてのコンテナーを 1000 ページだけ拡張します。

```
ALTER TABLESPACE DATA_TS
  EXTEND (ALL 1000)
```

例 5: INDEX_TS 表スペースにあるすべてのコンテナーのサイズを 100 メガバイト (MB) に変更します。

```
ALTER TABLESPACE INDEX_TS
  RESIZE (ALL 100 M)
```

例 6: 3 つの新規コンテナーを追加します。 1 番目のコンテナーを拡張し、2 番目をサイズ変更します。

```
ALTER TABLESPACE TS0
  ADD (FILE 'cont2' 2000, FILE 'cont3' 2000)
  ADD (FILE 'cont4' 2000)
  EXTEND (FILE 'cont0' 100)
  RESIZE (FILE 'cont1' 3000)
```

例 7: 表スペース TSO がパーティション 0、1 および 2 に存在します。新規コンテナーをデータベース・パーティション 0 に追加します。データベース・パーティション 1 上のすべてのコンテナーを拡張します。明示的に指定されたデータベース・パーティション (つまり、データベース・パーティション 0 および 1) 以外の、すべてのデータベース・パーティション上のコンテナーをサイズ変更します。

```
ALTER TABLESPACE TS0
  ADD (FILE 'A' 200) ON DBPARTITIONNUM (0)
  EXTEND (ALL 200) ON DBPARTITIONNUM (1)
  RESIZE (FILE 'B' 500)
```

この例では RESIZE 文節がデフォルトのコンテナー文節で、データベース・パーティション 2 で実行されます。他の操作は明示的にデータベース・パーティション 0 および 1 に向けられるからです。しかしデータベース・パーティションが 2 つしかない場合、ステートメントは正常に実行されますが、デフォルトのコンテナーが指定されたものの使用されていないことを示す警告が出されます (SQL1758W)。

関連資料:

- 410 ページの『CREATE TABLESPACE』
- *管理ガイド: パフォーマンス* の『システム環境変数』

ALTER TYPE (構造化)

ALTER TYPE ステートメントは、ユーザー定義の構造タイプの属性またはメソッド指定を追加またはドロップします。既存のメソッドのプロパティーも変更が可能です。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- タイプのスキーマに対する ALTERIN 特権
- SYSCAT.DATATYPES の DEFINER 列に記録されているそのタイプの定義者

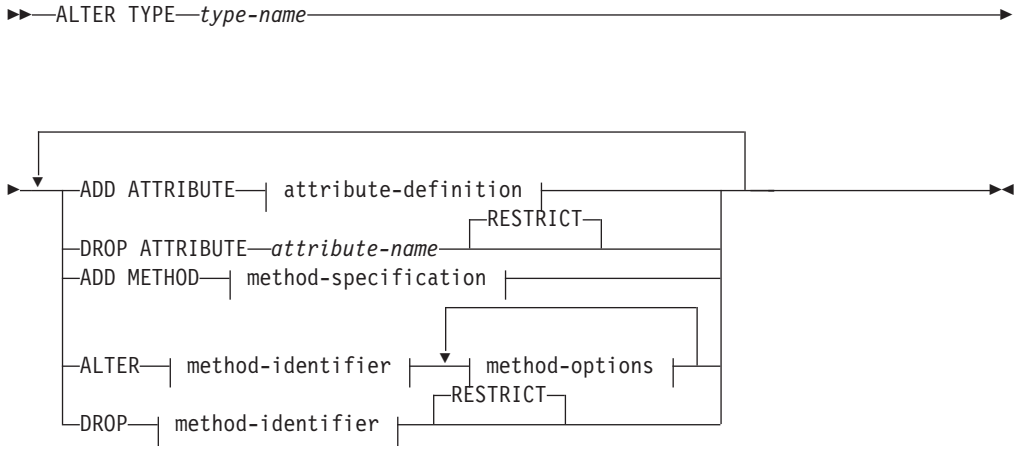
fenced でないようにメソッドを変更するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限

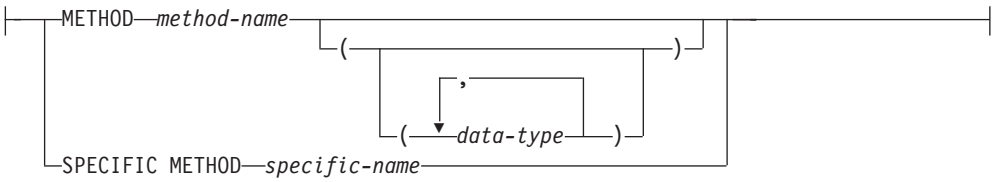
fenced であるようにメソッドを変更するには、さらに別の権限や特権は必要ありません。

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:



method-identifier:



method-options:

説明:

type-name
変更する構造タイプを識別します。指定するタイプは、カタログに定義されている既存のタイプであり (SQLSTATE 42704)、かつ構造タイプでなければなりません (SQLSTATE 428DP)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

ADD ATTRIBUTE
既存の構造タイプの最後の属性の後に、属性を追加します。

attribute-definition
構造タイプの属性を定義します。

attribute-name
属性の名前を指定します。この名前は、この構造タイプの他のどの属性 (継承された属性も含む) とも同じであってはならず、この構造タイプのどのサブタイプとも同じであってはなりません (SQLSTATE 42711)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *attribute-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

data-type 1
属性のデータ・タイプを指定します。これは、CREATE TABLE でリストされているデータ・タイプの 1 つで、 LONG VARCHAR、LONG VARGRAPHIC、または LONG VARCHAR や LONG VARGRAPHIC に基づいた特殊タイプ以外のものです (SQLSTATE 42601)。このデータ・タイプには既存のデータ・タイプを指定する必要があります (SQLSTATE 42704)。 *data-type* がスキーマ名なしで指定される場合、SQL パス上でスキーマを検索することにより、タイプは解決されます。『CREATE TABLE』 に種々のデータ・タイプの説明が記載されています。属性データ・タイプが参照タイプである場合、参照するターゲット・タイプはこのステートメントに既に存在する構造タイプでなければなりません (SQLSTATE 42704)。

タイプ DATALINK の属性を使って定義された構造タイプは、型付き表または型付きビューのデータ・タイプとしてのみ有効に使用することができます (SQLSTATE 01641)。

ランタイムにタイプのインスタンスが直接または間接的に同じタイプやそのサブタイプのインスタンスを含むタイプ定義を避けるために、タイプの定義において、属性タイプのいずれかが直接または間接的にそれ自身を使用するように定義してはならないという制限があります (SQLSTATE 428EP)。

lob-options
LOB タイプと関連したオプション (あるいは LOB に基づく特殊タイプ) を指定します。 lob-options の詳細については、『CREATE TABLE』 を参照してください。

datalink-options
DATALINK タイプと関連したオプション (あるいは DATALINK タイプに基づく特殊タイプ) を指定します。 datalink-options の詳細については、『CREATE TABLE』 を参照してください。

DATALINK タイプまたは DATALINK タイプに基づいている特殊タイプでオプションが指定されないと、 LINKTYPE URL および NO LINK CONTROL がデフォルト・オプションになることに注目してください。

DROP ATTRIBUTE
既存の構造タイプの属性をドロップします。

attribute-name
属性の名前。属性は、そのタイプの属性として存在していなければなりません (SQLSTATE 42703)。

RESTRICT
type-name が既存の表、ビュー、列、列のタイプ内でネストされた属性、または索引拡張のタイプとして使用される場合に、どの属性もドロップできないという規則を課します。

ADD METHOD *method-specification*
メソッド指定を、*type-name* で識別されるタイプに追加します。別個の CREATE METHOD ステートメントを使用してメソッドに本体を与えるまでは、このメソッドを使用することはできません。 *method-specification* についての詳細は、『CREATE TYPE (構造化)』 を参照してください。

ALTER *method-identifier*
変更されるメソッドのインスタンスを一意的に指定します。指定されたメソッドには、既存のメソッド本体があるかもしれませんし、ないかもしれません。 LANGUAGE SQL として宣言されたメソッドは変更できません (SQLSTATE 42917)。

method-identifier

METHOD *method-name*
特定のメソッドを指定します。 *type-name* というサブジェクト・タイプの *method-name* という名前のメソッド・インスタンスが 1 つだけ存在している場合にのみ有効です。このように指定されたメソッドには、任意の数のパラメーターを定義できます。タイプに、指定された名前のメ

ソッドが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。タイプに、そのメソッドのインスタンスが複数存在する場合も、エラーが戻されます (SQLSTATE 42725)。

METHOD *method-name (data-type,...)*
メソッドを一意に指定するメソッド・シグニチャーを指定します。メソッド解決のアルゴリズムは使用されません。

method-name
type-name タイプのメソッドの名前を指定します。

(data-type,...)
値は、 CREATE TYPE ステートメント上で (対応する位置に) 指定されたデータ・タイプと一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値が、特定のメソッド・インスタンスを識別するのに使用されます。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

長さ、精度、または位取りをコーディングする場合、その値は、CREATE TYPE ステートメントで指定された値と完全に一致していなければなりません。

0<*n*<25 は REAL を意味し、24<*n*<54 は DOUBLE を意味するので、 FLOAT(*n*) のタイプは、*n* に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つメソッドのタイプがない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC METHOD *specific-name*
メソッドの作成時に指定された名前か、デフォルト値として与えられた名前を使用して、特定のメソッドを識別します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* に指定される名前は、指定したスキーマまたは暗黙のスキーマに含まれる特定メソッドのインスタンスを識別するものでなければなりません。それ以外の名前が指定された場合は、エラーが戻されます (SQLSTATE 42704)。

method-options
メソッドに対して変更されるオプションを指定します。

FENCED または NOT FENCED
メソッドをデータベース・マネージャーのオペレーティング環境のプロセスまたはアドレス・スペースで実行しても安全か (NOT FENCED)、そうでないか (FENCED) を指定します。多くのメソッドは、FENCED または NOT FENCED のどちらかで実行するように選択することができます。

メソッドが FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を fenced して、そのメソッドからアクセスされないようにします。一般に、FENCED として実行されるメソッドは、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
適切にコード化、検討、および検査されていないメソッドに NOT FENCED を使用すると、 DB2 の保全性に危険を招く場合があります。DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED メソッドが使用される場合には、完全な保全性を確保できません。

NOT THREADSAFE を宣言したメソッドは、NOT FENCED には変更できません (SQLSTATE 42613)。

メソッドが定義済みの AS LOCATOR の任意のパラメーターを有していて、 NO SQL オプションが指定されていた場合には、このメソッドは FENCED には変更できません (SQLSTATE 42613)。

このオプションは LANGUAGE OLE メソッドを変更できません (SQLSTATE 42849)。

THREADSAFE または NOT THREADSAFE
メソッドを他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

メソッドが OLE 以外の LANGUAGE で定義される場合:

- メソッドが THREADSAFE に定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスでメソッドを呼び出すことができます。一般に、スレッド・セーフにするには、メソッドはどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED メソッドの両方が THREADSAFE になることが可能です。メソッドが LANGUAGE OLE とともに定義される場合には、THREADSAFE は指定されません (SQLSTATE 42613)。
- メソッドが NOT THREADSAFE として定義される場合には、データベース・マネージャーは他のルーチンと同じプロセスにメソッドを決して呼び出しません。fenced されたメソッドだけが、NOT THREADSAFE になり得ます (SQLSTATE 42613)。

DROP *method-identifier*
一意的にドロップするメソッドのインスタンスを指定します。指定されたメソッ

ドには、既存のメソッド本体があってはなりません (SQLSTATE 428ER)。 DROP METHOD ステートメントを使用してメソッド本体をドロップしてから、ALTER TYPE DROP METHOD を使用してください。 CREATE TYPE ステートメントで暗黙的に生成されたメソッド (mutators および observers など) は、ドロップできません (SQLSTATE 42917)。

RESTRICT
指定されたメソッドが、既存のメソッド本体を所持できないように制限を受けることを指示します。 DROP METHOD ステートメントを使用してメソッド本体をドロップしてから、 ALTER TYPE DROP METHOD を使用してください。

規則:

- 以下の場合には、タイプ *type-name* で属性を追加またはドロップすることは許可されていません (SQLSTATE 55043)。
  - あるタイプまたはそのタイプのサブタイプの 1 つが既存の表のタイプである場合。
  - タイプが直接または間接的に *type-name* を使用する表の列が存在する場合。*直接使用* および*間接使用* という用語は、構造タイプで定義されています。
  - 索引拡張で、このタイプまたはサブタイプのいずれかが使用される場合。
- 属性の追加によるタイプの変更で、このタイプまたはサブタイプの属性の合計が 4082 を超えてはなりません (SQLSTATE 54050)。
- ADD ATTRIBUTE オプション:
  - ADD ATTRIBUTE は、新しい属性に observer および mutator メソッドを生成します。これらのメソッドは、構造タイプが作成される際に生成されるタイプに類似しています (『CREATE TYPE (構造化)』 を参照)。これらのメソッドが任意の既存のメソッドまたは関数と競合したり、これらをオーバーライドしたりする場合には、 ALTER TYPE ステートメントは失敗します (SQLSTATE 42745)。
  - ユーザーがタイプ (またはこの任意のサブタイプ) の INLINE LENGTH を明示的に 292 よりも小さい値に指定した場合で、追加したこの属性が原因で、指定されたインライン長が、変更されたタイプのコンストラクター関数の結果のサイズよりも小さくなる場合 (32 バイト + 属性ごとに 10 バイト)、エラーになります (SQLSTATE 42611)。
- DROP ATTRIBUTE オプション:
  - 既存のスーパータイプから継承される属性は、ドロップできません (SQLSTATE 428DJ)。
  - DROP ATTRIBUTE は、ドロップされた属性の mutator および observer メソッドをドロップし、これらのドロップされたメソッドの従属性を検査します。
- DROP METHOD オプション:
  - 他のメソッドによってオーバーライドされた元のメソッドは、ドロップできません (SQLSTATE -2)。

注:

- SYSIBM、SYSFUN、または SYSPROC スキーマのメソッドは変更できません (SQLSTATE 42832)。

- 属性を追加またはドロップしてタイプを変更すると、そのタイプまたはそのタイプのサブタイプをパラメーターまたは結果として使用する関数またはメソッドに依存するすべてのパッケージは無効になります。
- 構造タイプから属性を追加またはドロップする場合:
  - タイプが作成されたときにシステムによりタイプの INLINE LENGTH が計算された場合、 INLINE LENGTH 値は自動的に、変更されたタイプについて修正され、そのサブタイプもすべて変更に対応するように修正されます。すべての構造タイプについても、INLINE LENGTH 値は自動的に (再帰的に) 変更されます。この場合、INLINE LENGTH はシステムにより計算され、変更された INLINE LENGTH を持つタイプの属性がタイプに含まれています。
  - 属性の追加またはドロップにより影響を受けるタイプの INLINE LENGTH がユーザーにより明示的に指定されたものである場合、この特定のタイプの INLINE LENGTH は変更されません。明示的に指定されたインライン長については、十分に注意してください。後でタイプに属性が追加されることがある場合、列定義でこのタイプまたはサブタイプの 1 つを使用するために、インスタンス化されたオブジェクトの長さの増加の可能性に対応できるように、インライン長を十分に大きくしておかなければなりません。
  - 新しい属性がアプリケーション・プログラムから見えるようにするには、データ・タイプの新しい構造に適合するように、既存のトランスフォーム機能を修正しなければなりません。
- パーティション・データベース環境では、外部ユーザー定義関数またはメソッドでの SQL の使用はサポートされていません (SQLSTATE 42997)。
- ***特権***

  EXECUTE 特権は、メソッド本体が CREATE METHOD ステートメントを使用して定義されるまでは、 ALTER TYPE ステートメントで明示的に指定されたすべてのメソッドには与えられません。ユーザー定義タイプの定義者には、ALTER TYPE ステートメントを使用して、メソッド指定をドロップする特権があります。

例:

例 1: ALTER TYPE ステートメントを使用して、手動で、参照するタイプおよび表の循環を許可します。 EMPLOYEE および DEPARTMENT という名前の表を手動で参照しているとします。

次のシーケンスで、タイプおよび表の作成ができます。

```
CREATE TYPE DEPT ...
CREATE TYPE EMP ... (タイプ REF(DEPT) の DEPTREF という属性を含む)
ALTER TYPE DEPT ADD ATTRIBUTE MANAGER REF(EMP)
CREATE TABLE DEPARTMENT OF DEPT ...
CREATE TABLE EMPLOYEE OF EMP (DEPTREF WITH OPTIONS SCOPE DEPARTMENT)
ALTER TABLE DEPARTMENT ALTER COLUMN MANAGER ADD SCOPE EMPLOYEE
```

次のシーケンスで、タイプおよび表のドロップができます。

```
DROP TABLE EMPLOYEE   (DEPARTMENT の MANAGER 列が有効範囲解除となる)
DROP TABLE DEPARTMENT
ALTER TYPE DEPT DROP ATTRIBUTE MANAGER
DROP TYPE EMP
DROP TYPE DEPT
```

例 2: ALTER TYPE ステートメントを使用して、サブタイプを参照する属性を持つタイプを作成します。

```
CREATE TYPE EMP ...
CREATE TYPE MGR UNDER EMP ...
ALTER TYPE EMP ADD ATTRIBUTE MANAGER REF(MGR)
```

例 3: ALTER TYPE ステートメントを使用して、属性を追加します。以下のステートメントは、EMP タイプに SPECIAL 属性を追加します。元の CREATE TYPE ステートメントでインライン長が指定されなかったため、DB2 は、 13 (新しい属性の 10 + 属性長 + 非 LOB 属性の 2 バイト) を追加してインライン長を計算しなおします。

```
ALTER TYPE EMP ...
  ADD ATTRIBUTE SPECIAL CHAR(1)
```

例 4: ALTER TYPE ステートメントを使用して、タイプに関連するメソッドを追加します。以下のステートメントは、BONUS というメソッドを追加します。

```
ALTER TYPE EMP ...
  ADD METHOD BONUS (RATE DOUBLE)
    RETURNS INTEGER
    LANGUAGE SQL
    CONTAINS SQL
    NO EXTERNAL ACTION
    DETERMINISTIC
```

CREATE METHOD ステートメントを発行してメソッド本体を作成するまでは、BONUS メソッドは使用できないことに注意してください。タイプ EMP に SALARY という属性が含まれているとすると、メソッド本体の定義例は以下のようになります。

```
CREATE METHOD BONUS(RATE DOUBLE) FOR EMP
  RETURN CAST(SELF.SALARY * RATE AS INTEGER)
```

関連資料:

- 347 ページの『CREATE TABLE』
- 440 ページの『CREATE TYPE (構造化)』
- 293 ページの『CREATE METHOD』
- *SQL リファレンス 第 1 巻* の『ユーザー定義タイプ』

関連サンプル:

- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』

ALTER USER MAPPING

ALTER USER MAPPING ステートメントは、指定したフェデレーテッド・サーバーの許可 ID について、データ・ソースで使用する許可 ID またはパスワードを変更するときに使います。

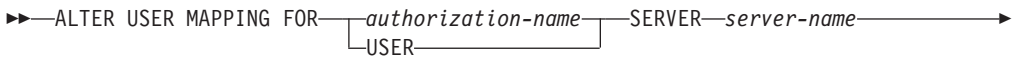
呼び出し:

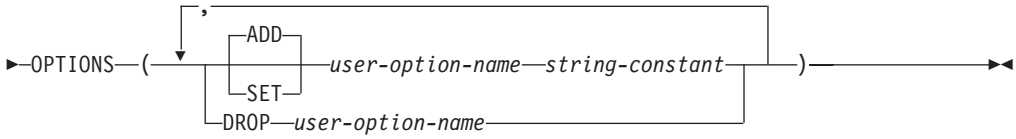
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID が、データ・ソースへマップされる許可名と違う場合、そのステートメントの許可 ID には SYSADM または DBADM 権限がなければなりません。それらの権限がない場合でも、許可 ID と許可名が一致すれば、特権あるいは権限は必要ありません。

構文:





説明:

authorization-name
ユーザーまたはアプリケーションがフェデレーテッド・データベースへ接続するときの、許可名を指定します。

USER
特殊レジスター USER の値。 USER を指定すると、ALTER USER MAPPING ステートメントの許可 ID は、 REMOTE_AUTHID ユーザー・オプションで指定したデータ・ソースの許可 ID にマップされます。

SERVER *server-name*
ローカル許可 ID へマップするリモート許可 ID を使ってアクセスできるデータ・ソースを指定します。このローカル許可 ID は、*authorization-name* で示されるか、または USER によって参照されるものです。

OPTIONS
変更するマッピングに関して、使用可能にする、リセットする、またはドロップするユーザー・オプションを指定します。

ADD
ユーザー・オプションを使用可能にします。

SET
ユーザー・オプションの設定を変更します。

user-option-name
使用可能にする、あるいはリセットするユーザー・オプションを指定します。

string-constant
user-option-name の設定を、文字ストリング定数として指定します。

DROP *user-option-name*
ユーザー・オプションをドロップします。

注:

- ユーザー・オプションは、同じ ALTER USER ステートメントに複数回指定することはできません (SQLSTATE 42853)。ユーザー・オプションを使用可能にする、リセットする、あるいはドロップする場合、使用中の他のユーザー・オプションには影響はありません。
- 所定の作業単位 (UOW) 内の ALTER USER MAPPING ステートメントは、UOW に以下のいずれかがすでに含まれている場合には処理できません (SQLSTATE 55007)。
  - マッピングに含めるソース・データの表またはビューのニックネームを参照する SELECT ステートメント。
  - マッピングに含めるソース・データの表またはビューのニックネーム上のオープン・カーソル。
  - マッピングに含めるソース・データの表またはビューのニックネームに対して発行された、INSERT、 DELETE、または UPDATE ステートメント。

例:

例 1: Jim はローカル・データベースを使い、 ORACLE1 という Oracle データ・ソースに接続します。そして許可 ID KLEEWEIN を使ってローカル・データベースにアクセスします。 KLEEWEIN は、CORONA (ORACLE1 へアクセスするときの許可 ID) へマップします。 Jim は新しい ID である JIMK を使用して ORACLE1 へのアクセスを開始します。ここで、KLEEWEIN は JIMK へマップすることが必要になります。

```
ALTER USER MAPPING FOR KLEEWEIN
  SERVER ORACLE1
  OPTIONS ( SET REMOTE_AUTHID 'JIMK'  )
```

例 2: Mary はフェデレーテッド・データベースを使用して、 DORADO という DB2 Universal Database for z/OS and OS/390 データ・ソースへ接続します。そしてある許可 ID を使って DB2 にアクセスし、別の許可 ID で DORADO にアクセスします。これら 2 つの ID のマッピングは作成してあります。どちらの ID でも同じパスワードを使っていますが、ここで、DORADO の ID 用に固有のパスワード ZNYQ を使うことにしました。その結果、使用しているフェデレーテッド・データベースのパスワードを ZNYQ へマップしなければならなくなります。

```
ALTER USER MAPPING FOR MARY
  SERVER DORADO
  OPTIONS ( ADD REMOTE_PASSWORD 'ZNYQ'  )
```

関連資料:

- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムのユーザー・マッピング・オプション』

ALTER VIEW

ALTER VIEW ステートメントは、参照タイプ列を変更して有効範囲を追加することによって、既存のビューを変更します。

呼び出し:

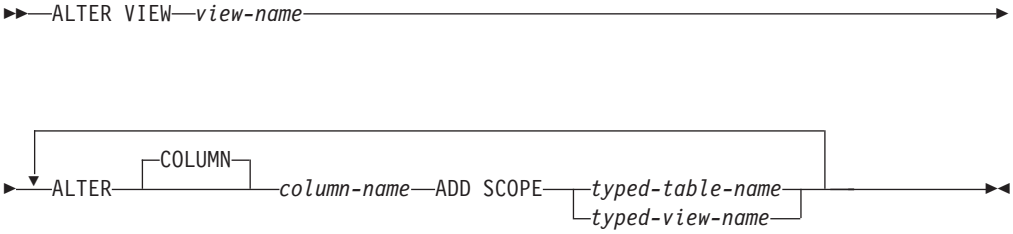
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- ビューのスキーマに対する ALTERIN 特権
- 変更するビューの定義者
- 変更するビューに対する CONTROL 特権

構文:



説明:

view-name
変更するビューを指定します。ビューはカタログに記述されている必要があります。

ALTER COLUMN *column-name*
ビューで変更する列の名前です。 *column-name* は、ビューの既存の列を指定するものでなければなりません (SQLSTATE 42703)。名前は非修飾でなければなりません。

ADD SCOPE
有効範囲が未定義である既存の参照タイプ列に、有効範囲を追加します (SQLSTATE 428DK)。列をスーパービューから継承することはできません (SQLSTATE 428DJ)。

typed-table-name
型付き表の名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-table-name* のタイプを表します (SQLSTATE 428DM)。 値が *typed-table-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

typed-view-name
型付きビューの名前。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-view-name* のタイプを表します (SQLSTATE 428DM)。 値が *typed-view-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

ALTER WRAPPER

ALTER WRAPPER ステートメントは、ラッパーのプロパティーの更新に使用されます。

呼び出し:

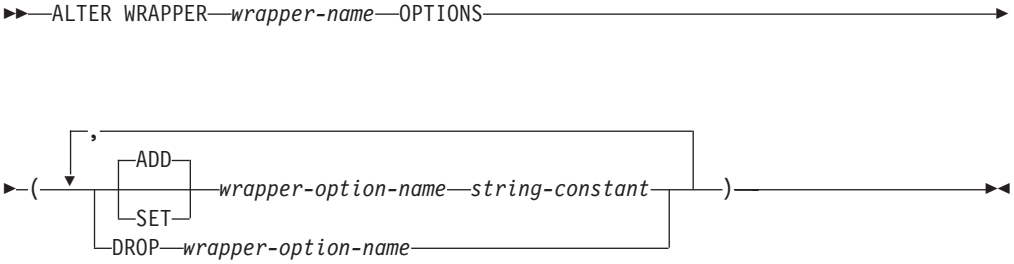
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限

構文:



説明:

wrapper-name
ラッパーの名前を指定します。

OPTIONS
使用可能にする、リセットする、またはドロップするラッパー・オプションを指定します。

ADD
サーバー・オプションを使用可能にします。

SET
ラッパー・オプションの設定を変更します。

wrapper-option-name
使用可能にする、またはリセットするラッパー・オプションを指定します。現在、唯一サポートされているラッパー・オプション名は DB2_FENCED です。

string-constant
wrapper-option-name の設定を、文字ストリング定数として指定します。有効な値は 'Y' または 'N' です。リレーショナル・ラッパーのデフォルト値は 'N' で、非リレーショナル・ラッパーのデフォルト値は 'Y' です。

DROP *wrapper-option-name*
ラッパー・オプションをドロップします。

注:

- ALTER WRAPPER ステートメントの実行には、ラッパー固有のオプションの有効性検査は含まれていません。

例:

例 1: DB2_FENCED オプションをラッパー SQLNET に設定します。

```
ALTER WRAPPER SQLNET OPTIONS (SET DB2_FENCED 'Y')
```

関連資料:

- 489 ページの『CREATE WRAPPER』

ASSOCIATE LOCATORS

ASSOCIATE LOCATORS ステートメントは、ストアード・プロシージャーから戻される結果セットごとの結果セット・ロケーター値を入手します。

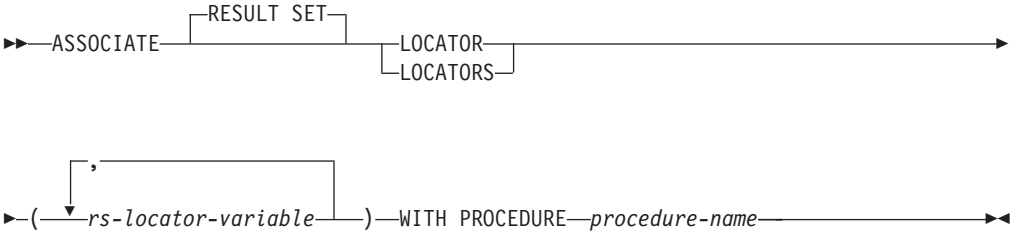
呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

必要ありません。

構文:



説明:

rs-locator-variable
コンパウンド・ステートメントで宣言されている結果セット・ロケーター変数を指定します。

WITH PROCEDURE
ここで指定したプロシージャー名で、結果セット・ロケーターを戻すストアード・プロシージャーを識別します。

procedure-name
プロシージャー名は修飾または非修飾の名前です。この名前の各部分は SBCS 文字で構成される必要があります。

完全修飾のプロシージャー名は、2 つの部分からなる名前です。最初の部分は、ストアード・プロシージャーのスキーマ名を含んでいる ID です。最後の部分は、ストアード・プロシージャーの名前を含んでいる ID です。それぞれの部分の間はピリオドで区切らなければなりません。それらの部分の一部またはすべてを区切り ID にできます。

プロシージャー名が非修飾の場合、暗黙的なスキーマ名が修飾子としてプロシージャー名に追加されることはないので、プロシージャー名に含まれる名前は 1 つだけです。 ASSOCIATE LOCATOR ステートメントを正常に実行するのに必要なことは、ステートメント内の非修飾のプロシージャー名を、最近実行した CALL ステートメント (ただし、非修飾のプロシージャー名を使用するように指定したもの) でのプロシージャー名と同じにすることだけです。その CALL ステートメントでの非修飾の名前の暗黙的なスキーマ名は、マッチングにおいて考慮されません。プロシージャー名を指定する際に従わなければならない規則について、以下で説明します。

ASSOCIATE LOCATORS ステートメントを実行する場合、プロシージャーの名前または指定で、 CALL ステートメントを使用して要求側がすでに呼び出しているストアード・プロシージャーを識別しなければなりません。 ASSOCIATE LOCATORS ステートメントのプロシージャー名は、 CALL ステートメントで指定したのと同じ方法で指定しなければなりません。たとえば CALL ステートメントで 2 つの部分からなる名前を指定した場合、 ASSOCIATE LOCATORS ステートメントでも 2 つの部分からなる名前を使用しなければなりません。

規則:

- 結果セットには、複数のロケーターを割り当てることができます。別々の結果セット・ロケーター変数を指定して、同じ ASSOCIATE LOCATORS ステートメントを複数回発行することができます。
- ASSOCIATE LOCATORS ステートメントでリストされている結果セット・ロケーター変数の数が、ストアード・プロシージャーで戻されるロケーターの数より少ない場合、ステートメント内のすべての変数に値が割り当てられ、警告が出されます。
- ASSOCIATE LOCATORS ステートメントでリストされている結果セット・ロケーター変数の数が、ストアード・プロシージャーで戻されるロケーターの数より多い場合、超過した変数に値 0 が割り当てられます。
- あるストアード・プロシージャーが同じ呼び出し側から複数回呼び出される場合、アクセス可能なのは、最新の結果セットだけです。

例:

以下の例のステートメントは、SQL プロシージャーに組み込まれることを想定しています。

例 1: 結果セット・ロケーター変数 LOC1 および LOC2 を使用して、ストアード・プロシージャー P1 から戻される 2 つの結果セットの結果セット・ロケーター値を入手します。ストアード・プロシージャーが 1 つの部分だけからなる名前で呼び出されることを想定しています。

```
CALL P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2)
  WITH PROCEDURE P1;
```

例 2: 例 1 のシナリオを繰り返しますが、スキーマ MYSCHEMA で確実にストアード・プロシージャー P1 が使用されるように、 2 つの部分からなる名前を使用し、ストアード・プロシージャーの明示的なスキーマ名を指定します。

```
CALL MYSCHEMA.P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2)
  WITH PROCEDURE MYSCHEMA.P1;
```

BEGIN DECLARE SECTION

BEGIN DECLARE SECTION ステートメントは、ホスト変数宣言セクションの始まりを示します。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、実行可能ステートメントではありません。また、REXX に指定することはできません。

許可:

必要ありません。

構文:

説明:

BEGIN DECLARE SECTION ステートメントは、ホスト言語の規則に従って変数宣言が許される個所であれば、アプリケーション・プログラムのどのような個所にもコーディングできます。これは、ホスト変数宣言セクションの始まりを示すのに使用されます。ホスト変数セクションは、 END DECLARE SECTION ステートメントで終了します。

規則:

- BEGIN DECLARE SECTION と END DECLARE SECTION ステートメントは、対にして使用する必要があり、ネストすることはできません。
- 宣言セクションに SQL ステートメントを含めることはできません。
- REXX 以外のすべてのホスト言語において、SQL ステートメントで参照される変数は、宣言セクションで宣言する必要があります。また、そのセクションは、変数に対する最初の参照より前になければなりません。一般に、REXX では、LOB ロケーターとファイル参照変数を除いて、ホスト変数は宣言されません。それらは BEGIN DECLARE SECTION では宣言されません。
- 宣言セクションの外部で宣言される変数の名前を、宣言セクションで宣言されている変数と同じ名前にすることはできません。
- LOB データ・タイプのデータ・タイプと長さの前には、 SQL TYPE IS キーワードを付ける必要があります。

例:

例 1: C プログラムで、ホスト変数 hv_smint (smallint)、hv_vchar24 (varchar(24))、hv_double (double)、 hv_blob_50k (blob(51200))、hv_struct (構造タイプ ″struct_type″ は blob(10240)) を定義します。

```
EXEC SQL  BEGIN DECLARE SECTION;
  short  hv_smint;
  struct {
  short hv_vchar24_len;
  char  hv_vchar24_value[24];
 }                              hv_vchar24;
```

```
  double hv_double;
  SQL TYPE IS BLOB(50K) hv_blob_50k;
  SQL TYPE IS struct_type AS BLOB(10k) hv_struct;
EXEC SQL  END DECLARE SECTION;
```

例 2: COBOL プログラムで、ホスト変数 HV-SMINT (smallint)、 HV-VCHAR24 (varchar(24))、HV-DEC72 (dec(7,2))、および HV-BLOB-50k (blob(51200)) を定義します。

```
WORKING-STORAGE SECTION.
      EXEC SQL BEGIN DECLARE SECTION  END-EXEC.
01  HV-SMINT              PIC S9(4)       COMP-4.
01  HV-VCHAR24.
    49 HV-VCHAR24-LENGTH  PIC S9(4)       COMP-4.
    49 HV-VCHAR24-VALUE   PIC X(24).
01  HV-DEC72              PIC S9(5)V9(2)  COMP-3.
01 HV-BLOB-50K            USAGE SQL TYPE IS BLOB(50K).
      EXEC SQL END DECLARE SECTION  END-EXEC.
```

例 3: FORTRAN プログラムで、ホスト変数 HVSMINT (smallint)、 HVVCHAR24 (char(24))、HVDOUBLE (double)、および HVBLOB50k (blob(51200)) を定義します。

```
EXEC SQL  BEGIN DECLARE SECTION
  INTEGER*2      HVSMINT
  CHARACTER*24   HVVCHAR24
  REAL*8         HVDOUBLE
  SQL TYPE IS BLOB(50K)  HVBLOB50K
EXEC SQL  END DECLARE SECTION
```

注: FORTRAN では、予期される値が 254 文字を超える場合には、 CLOB ホスト変数を使用する必要があります。

例 4: REXX プログラムで、ホスト変数 HVSMINT (smallint)、 HVBLOB50K (blob(51200))、および HVCLOBLOC (CLOB ロケーター) を定義します。

```
DECLARE :HVCLOBLOC  LANGUAGE TYPE CLOB LOCATOR
call sqlexec 'FETCH c1 INTO :HVSMINT, :HVBLOB50K'
```

変数 HVSMINT と HVBLOB50K は、FETCH ステートメントで使用することによって、暗黙に定義されています。

関連資料:

- 549 ページの『END DECLARE SECTION』

関連サンプル:

- 『advsql.sqb -- How to read table data using CASE (MF COBOL)』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『spclient.sqc -- Call various stored procedures (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『spclient.sqC -- Call various stored procedures (C++)』
- 『tut_read.sqC -- How to read tables (C++)』

- 『udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)』

CALL

CALL ステートメントはプロシージャーを呼び出します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

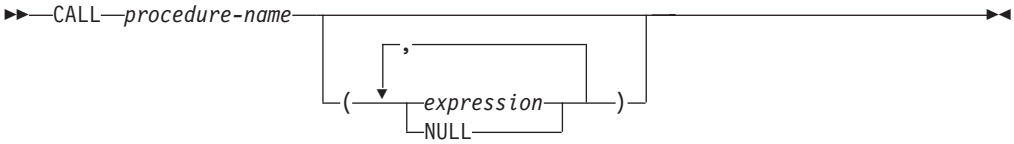
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- プロシージャーでの EXECUTE 特権
- SYSADM または DBADM 権限

実行の許可を与えられていないステートメントの許可 ID を持つ一致するプロシージャーが存在する場合、エラー (SQLSTATE 42501) が出されます。

構文:



説明:

procedure-name
呼び出されるプロシージャーを指定します。プロシージャーはカタログに記述されている必要があります。呼び出す特定のプロシージャーは、プロシージャー解決を使用して選択します。 (詳細については、このステートメントの『注』セクションを参照してください。)

expression **または NULL**
指定される *expression* または NULL のどちらかは、CALL の引き数です。CALL ステートメントの *n* 番目の引き数は、プロシージャーの CREATE PROCEDURE ステートメントで定義されている *n* 番目のパラメーターに対応します。

CALL の各引き数は、以下のようなプロシージャー定義における対応するパラメーターと互換性がなければなりせん。

- IN パラメーター
  - 引き数は、パラメーターに割り当て可能でなければなりません。
  - ストリング引き数の割り当てには、ストレージ割り当て規則を使用します。
- OUT パラメーター
  - 引き数は、単一の変数またはパラメーター・マーカーでなければなりません (SQLSTATE 42886)。
  - 引き数は、パラメーターに割り当て可能でなければなりません。

  - ストリング引き数の割り当てには、検索割り当て規則を使用します。
- INOUT パラメーター
  - 引き数は、単一の変数またはパラメーター・マーカーでなければなりません (SQLSTATE 42886)。
  - 引き数は、パラメーターに割り当て可能でなければなりません。
  - ストリング引き数の割り当てには、呼び出しについてはストレージ割り当て規則、および戻りに関しては検索割り当て規則を使用します。

注:

- ***プロシージャー・シグニチャー:***

  プロシージャーは、そのスキーマ、プロシージャー名、およびパラメーター数によって識別されます。これはプロシージャー・シグニチャーと呼ばれ、データベース内でユニークである必要があります。プロシージャーごとにパラメーターの数が違っていれば、 1 つのスキーマに同じ名前のプロシージャーが複数存在してもかまいません。

- ***SQL パス:***

  プロシージャーは、修飾名 (スキーマおよびプロシージャー名) を参照して呼び出すことができます。修飾名の後に、括弧で閉じられた引き数のオプションのリストが続きます。また、スキーマ名を指定せずにプロシージャーを呼び出すことも可能であり、その場合は、同じ数のパラメーターを持つ異なるスキーマのプロシージャーが選択可能になります。このような場合、プロシージャー解決に役立つ SQL パスが使用されます。 SQL パスとは、同じ名前、同じパラメーター数を持つプロシージャーを識別するために探索されるスキーマのリストです。静的 CALL ステートメントに対する SQL パスは、 FUNCPATH BIND オプションを使って指定されます。動的 CALL ステートメントの場合、 SQL パスは CURRENT PATH 特殊レジスターの値です。

- ***プロシージャー解決:***

  特定のプロシージャーの呼び出しに対して、データベース・マネージャーは、同じ名前をもつ呼び出し可能なプロシージャーから呼び出すプロシージャーを判別する必要があります。プロシージャー解決は、以下の手順で行われます。

  1. カタログ (SYSCAT.ROUTINES) から、以下のすべての条件が真となるすべてのプロシージャーを探します。
     - スキーマ名が指定された呼び出し (修飾子付き参照) の場合、スキーマ名とプロシージャー名が呼び出し名に一致する。
     - スキーマ名が指定されていない呼び出し (修飾子なし参照) の場合、プロシージャー名が呼び出し名に一致し、SQL パス中のスキーマの 1 つに一致するスキーマ名がある。
     - 定義済みパラメーターの数が呼び出しと一致している。
     - 呼び出し側が、プロシージャーで EXECUTE 特権を持っている。
  2. スキーマがその SQL パスで最初に出現するプロシージャーが選択されます。

  ステップ 1 の後で候補となるプロシージャーが残らなかった場合は、エラー (SQLSTATE 42884) になります。

- ***SQL プロシージャーからの RETURN_STATUS の検索:***

  SQL プロシージャーが RETURN ステートメントを状況値とともに正常に発行すると、この値が SQLCA の最初の SQLERRD フィールドに戻されます。 SQL プロシージャーで CALL ステートメントが発行される場合、 GET DIAGNOSTICS ステートメントを使用して RETURN_STATUS 値を検索します。SQLSTATE がエラーを示す場合は、値は -1 になります。エラーが出ないで、RETURN ステートメントがプロシージャーで指定されなかった場合には、値は 0 になります。

- ***プロシージャーから戻される結果セット:***

  呼び出し側プログラムが CLI、JDBC、または SQLJ を使用して作成されている場合、または呼び出し側が SQL プロシージャーの場合には、結果セットを呼び出し側に直接戻すことができます。プロシージャーは、結果セットにカーソルを宣言して、その結果セットでカーソルをオープンし、プロシージャー終了時にカーソルをオープンしたままにすることによって、結果セットを戻すよう指定します。

  プロシージャーの終了時には、

  – オープンされたままのカーソルすべてについて、結果セットは呼び出し側に戻されるか、 (WITH RETURN TO CLIENT カーソルの場合) クライアントに直接戻されます。
  – 未読の行だけが戻されます。たとえば、カーソルの結果セットに 500 行が含まれていて、そのうち 150 行がプロシージャーの終了時にプロシージャーによって読み取られた場合、第 151 行から第 500 行までが呼び出し側またはアプリケーションに戻されます (該当する場合)。

  プロシージャーが CLI または JDBC から呼び出され、複数のカーソルがオープンされたままの場合、結果セットはカーソルがオープンされた順序でのみ処理が可能です。

- ***パフォーマンスの向上:***

  すべての引き数の値は、アプリケーションからプロシージャーへ渡されます。この操作のパフォーマンスを向上させるには、 OUT パラメーターに対応し、数バイト以上の長さを持つホスト変数を、 CALL ステートメントを実行する前に NULL に設定しなければなりません。

- ***CALL ステートメントのネスト:***

  プロシージャーは、ルーチンやアプリケーション・プログラムから呼び出すことができます。プロシージャーをルーチンから呼び出す場合、その呼び出しはネストされるものと見なされます。

  プロシージャーが照会結果セットを戻す場合には、その結果セットは以下のように戻されます。

  – RETURN TO CALLER 結果セットは、直前のネスト・レベルにあるプログラムでのみ可視です。
  – RETURN TO CLIENT 結果セットは、そのプロシージャーがネストされた一連のプロシージャーから呼び出された場合に限り可視になります。呼び出しチェーンのどこかで関数やメソッドが実行されると、結果セットは不可視になりま

す。結果セットが可視の場合、最初のプロシージャーが呼び出されたクライアント・アプリケーションでのみ可視になります。

以下の例について考慮します。

```
Client program:
EXEC SQL CALL PROCA;

  PROCA:
  EXEC SQL CALL PROCB;

    PROCB:
    EXEC SQL DECLARE B1 CURSOR WITH RETURN TO CLIENT ...;
    EXEC SQL DECLARE B2 CURSOR WITH RETURN TO CALLER ...;
    EXEC SQL DECLARE B3 CURSOR FOR SELECT UDFA FROM T1;

      UDFA:
      EXEC SQL CALL PROCC;

        PROCC:
        EXEC SQL DECLARE C1 CURSOR WITH RETURN TO CLIENT ...;
        EXEC SQL DECLARE C2 CURSOR WITH RETURN TO CALLER ...;
```

プロシージャー PROCB から:

– カーソル B1 はクライアント・アプリケーションでは可視ですが、プロシージャー PROCA では不可視です。
– カーソル B2 は PROCA では可視ですが、クライアントには不可視です。

プロシージャー PROCC から:

– カーソル C1 は UDFA でもクライアント・アプリケーションでも不可視です。 (UDFA はクライアントと PROCC との間の呼び出しチェーンで現れ、結果セットはクライアントに戻されないからです。)
– カーソル C2 は UDFA では可視ですが、上位のプロシージャーでは不可視です。

- ***トリガー、動的コンパウンド・ステートメント、関数、またはメソッド内のネストされたプロシージャー:***

トリガー、動的コンパウンド・ステートメント、関数、またはメソッドの中で、プロシージャーが呼び出されるときは、次のとおりです。

– プロシージャーは COMMIT または ROLLBACK ステートメントを発行してはなりません。
– プロシージャーから戻される結果セットにはアクセスできません。
– プロシージャーが READS SQL DATA または MODIFIES SQL DATA として定義されている場合、プロシージャーを呼び出したステートメントによって変更されている表には、プロシージャー内のどのステートメントからもアクセスできません (SQLSTATE 57053)。また、プロシージャーが MODIFIES SQL DATA として定義されている場合は、プロシージャーを呼び出したステートメントによって読み取りまたは変更されている表には、プロシージャー内のどのステートメントからもアクセスできません (SQLSTATE 57053)。

関数またはメソッドの中でプロシージャーが呼び出されるときは、次のとおりです。

- プロシージャーは、呼び出された関数またはメソッドと同じ表アクセスに関する制約事項を持ちます。
- 関数またはメソッドが呼び出される前に定義されたセーブポイントは、プロシージャーでは不可視です。またプロシージャー内で定義されたセーブポイントは関数またはメソッド以外では不可視になります。
- プロシージャーから戻される RETURN TO CLIENT 結果セットに、クライアントからアクセスすることはできません。

- ***互換性:***
  - CALL_RESOLUTION DEFERRED オプションを使用してアプリケーションを事前にコンパイルすると、アプリケーションに組み込める旧書式の CALL ステートメントがあります。このオプションは SQL プロシージャーには使用できません。

例:

例 1:

Java プロシージャーが、以下のステートメントを使用してデータベースに定義されています。

```
CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER,
                                                OUT COST DECIMAL(7,2),
                                                OUT QUANTITY INTEGER)
                EXTERNAL NAME 'parts!onhand'
                LANGUAGE JAVA
                PARAMETER STYLE DB2GENERAL;
```

Java アプリケーションは、以下のコードを使用してこのプロシージャーを呼び出します。

```
...
CallableStatement stpCall;

String sql = "CALL PARTS_ON_HAND (?, ?, ?)";

stpCall = con.prepareCall(sql); /*con is the connection */

stpCall.setInt(1, hvPartnum);
stpCall.setBigDecimal(2, hvCost);
stpCall.setInt(3, hvQuantity);

stpCall.registerOutParameter(2, Types.DECIMAL, 2);
stpCall.registerOutParameter(3, Types.INTEGER);

stpCall.execute();

hvCost = stpCall.getBigDecimal(2);
hvQuantity = stpCall.getInt(3);
...
```

このアプリケーションのコード部分は、クラス parts の Java メソッド onhand を呼び出します。これは、CALL ステートメントで指定されたプロシージャー名がデータベースで検出され、外部名 parts!onhand を持っているためです。

例 2:

4 つの異なるスキーマに 6 個の FOO プロシージャーがあり、以下のように登録されているとします (必須キーワードの一部は省略されています)。

```
CREATE PROCEDURE AUGUSTUS.FOO (INT) SPECIFIC FOO_1 ...
CREATE PROCEDURE AUGUSTUS.FOO (DOUBLE, DECIMAL(15, 3)) SPECIFIC FOO_2 ...
CREATE PROCEDURE JULIUS.FOO (INT) SPECIFIC FOO_3 ...
CREATE PROCEDURE JULIUS.FOO (INT, INT, INT) SPECIFIC FOO_4 ...
CREATE PROCEDURE CAESAR.FOO (INT, INT) SPECIFIC FOO_5 ...
CREATE PROCEDURE NERO.FOO (INT,INT) SPECIFIC FOO_6 ...
```

以下のようにプロシージャーが参照されるとします (I1 および I2 は INTEGER 値です)。

```
CALL FOO(I1, I2)
```

この参照を行うアプリケーションの SQL パスが次のようになっているとします。

```
"JULIUS", "AUGUSTUS", "CAESAR"
```

アルゴリズムに従っていきます。

スキーマ ″NERO″ が SQL パスに含まれていないため、FOO_6 は候補から除かれます。パラメーターの数が違うため、FOO_1、FOO_3、および FOO_4 は候補から除かれます。残った候補は順番に考慮され、SQL パスにより判別します。引き数およびパラメーターのタイプは無視されることに注意してください。 FOO_5 のパラメーターは CALL の引き数と正確に一致しますが、 SQL パスで ″CAESAR″ の前に ″AUGUSTUS″ が現れるため FOO_2 が選ばれます。

関連資料:

- 575 ページの『GET DIAGNOSTICS』
- *SQL リファレンス 第 1 巻* の『CURRENT PATH 特殊レジスター』
- *SQL リファレンス 第 1 巻* の『コンパイル済みステートメントから呼び出される CALL』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

関連サンプル:

- 『outcli.sqb -- Call stored procedures using the SQLDA structure (MF COBOL)』
- 『spclient.c -- Call various stored procedures』
- 『spclient.sqc -- Call various stored procedures (C)』
- 『spclient.sqC -- Call various stored procedures (C++)』
- 『SpClient.sqlj -- Call a variety of types of stored procedures from SpServer.sqlj (SQLj)』

CASE

CASE ステートメントは、複数の条件に基づいて実行パスを選択します。 CASE ステートメントを CASE 式と混同しないでください。 CASE 式を使用すると、1 つ以上の条件の評価に基づいて 1 つの式を選択できます。

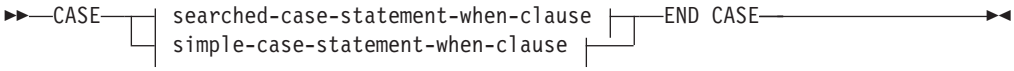
呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

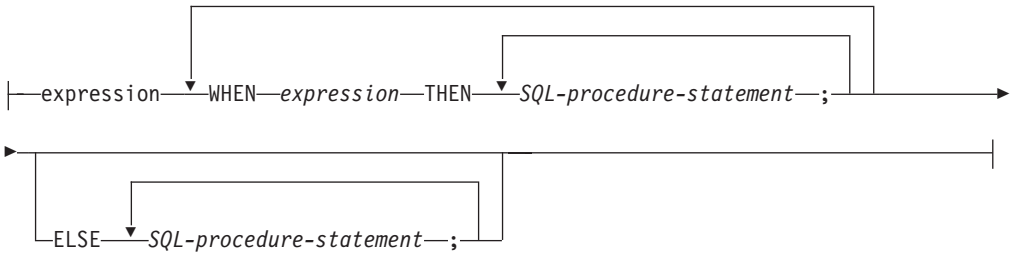
許可:

CASE ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 CASE ステートメントに組み込まれている SQL ステートメントおよび式を呼び出すために必要な特権がなければなりません。

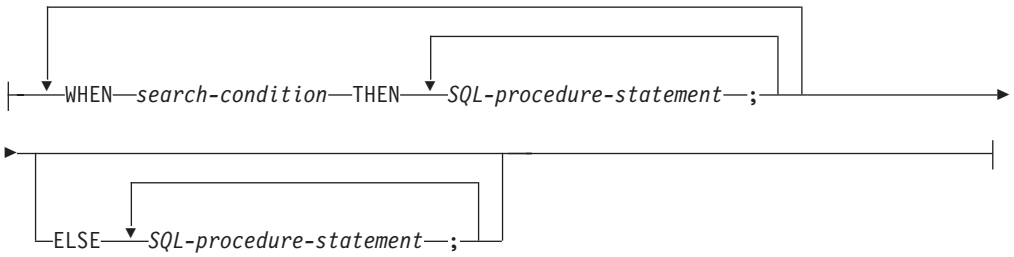
構文:



simple-case-statement-when-clause:



searched-case-statement-when-clause:



説明:

CASE

case-statement を開始します。

simple-case-statement-when-clause
最初の WHEN キーワードの前の *expression* (式) の値が、その WHEN キーワードの後にある各 *expression* の値と等しいかどうかが検査されます。検索条件が真の場合、THEN ステートメントが実行されます。結果が不明または偽の場合、処理は次の検索条件まで継続されます。結果が検索条件のいずれにも一致せず、なおかつ ELSE 文節が使用されている場合、 ELSE 文節内のステートメントが処理されます。

searched-case-statement-when-clause
WHEN キーワードの後の *search-condition* が評価されます。真であると評価された場合、関連する THEN 文節内のステートメントが評価されます。偽または不明であると評価された場合、次の *search-condition* が評価されます。真であると評価される *search-condition* がなく、なおかつ ELSE 文節が使用されている場合、ELSE 文節内のステートメントが処理されます。

SQL-procedure-statement
呼び出すステートメントを指定します。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

END CASE
case-statement を終了します。

注:

- WHEN で指定した条件がすべて真ではなく、なおかつ ELSE 文節が指定されていない場合、実行時にエラーが出され、CASE ステートメントの実行が終了します (SQLSTATE 20000)。
- 使用する CASE ステートメントでは、考えられるあらゆる実行条件を全て網羅するようにしてください。

例:

SQL 変数 v_workdept の値によっては、表 DEPARTMENT 内の更新列 DEPTNAME を適当な名前で更新しなければなりません。

以下の例では、*simple-case-statement-when-clause* の構文を使用して、これを行う方法を示しています。

```
CASE v_workdept
  WHEN'A00'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 1';
  WHEN 'B01'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 2';
  ELSE UPDATE department
    SET deptname = 'DATA ACCESS 3';
END CASE
```

以下の例では、*searched-case-statement-when-clause* の構文を使用して、これを行う方法を示しています。

```
CASE
  WHEN v_workdept = 'A00'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 1';
  WHEN v_workdept = 'B01'
```

```
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 2';
  ELSE UPDATE department
    SET deptname = 'DATA ACCESS 3';
END CASE
```

関連資料:

- *SQL リファレンス 第 1 巻* の『式』
- 140 ページの『コンパウンド SQL (プロシージャー)』

関連サンプル:

- 『advsql.sqb -- How to read table data using CASE (MF COBOL)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『TbTrig.java -- How to use triggers (JDBC)』
- 『TbTrig.sqlj -- How to use triggers (SQLj)』

CLOSE

CLOSE ステートメントは、カーソルをクローズします。カーソルのオープン時に結果表が作成された場合、その表は破棄されます。

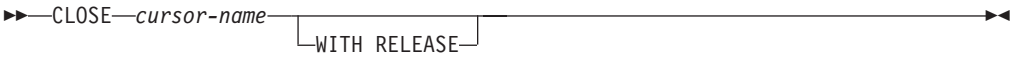
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。カーソルの使用に必要な許可については、 『DECLARE CURSOR』を参照してください。

構文:



説明:

cursor-name
クローズするカーソルを識別します。 DECLARE CURSOR ステートメントの項で説明されているように、 *cursor-name* は、宣言されたカーソルを指定しなければなりません。 CLOSE ステートメントを実行する場合、カーソルはオープン状態でなければなりません。

WITH RELEASE
カーソルのために保留されていた、すべてのロックを解放しようとします。すべてのロックを解放する必要はないことに注意してください。これらのロックは他の操作または活動のために保留することができます。

注:

- 作業単位の終了時には、アプリケーション・プロセスに属し、 WITH HOLD オプションを指定せずに宣言されたすべてのカーソルは暗黙にクローズされます。
- WITH RELEASE 文節は、関数またはメソッドで定義されたカーソルのクローズには効力を持っていません。またこの文節は、関数またはメソッドから呼び出されるストアード・プロシージャーで定義されたカーソルのクローズに対しても効力を持ちません。
- WITH RELEASE 文節は、分離レベル CS または UR で機能しているカーソルに対しては影響を与えません。また、分離レベル RS または RR で機能しているカーソルに対して WITH RELEASE を指定した場合には、それらの分離レベルの保証の一部が終了させられます。特に、カーソルを再オープンする場合には、RS カーソルが ’反復不可読み取り’ 状態になったり、 RR カーソルが ’反復不可読み取り’ か ’幻像読み取り’ 状態のどちらかになる可能性があります。

  もともと RR か RS だったカーソルが、WITH RELEASE 文節を使用してクローズされたのちに、再オープンされると、新しいロックを獲得できます。
- クローズされずに呼び出し側プログラムに戻ったストアード・プロシージャー内のカーソルには、特殊な規則が適用されます。

- カーソルがオープンしている間は (つまり、まだクローズしていない場合)、そのカーソルをステートメントが呼び出した結果 (たとえば、 NEXT VALUE 式が組み込まれたカーソルをシーケンスに使用した FETCH または UPDATE) 生じたシーケンス値への変更が、そのカーソルが示したシーケンスを PREVIOUS VALUE に更新することはありません。このように影響を受けるシーケンスの PREVIOUS VALUE 値は、 CLOSE ステートメントを使ってカーソルを明示的にがクローズした際に更新されます。パーティション・データベース環境では、コミットやロールバックによってカーソルを暗黙的にクローズした場合、 PREVIOUS VALUE はシーケンスについて生成された最新の値に更新されない場合があります。

例:

カーソルを使用して、C プログラム変数 dnum、dname、および mnum の中に、一度に 1 行ずつ取り出します。最後にカーソルをクローズします。再びカーソルをオープンすると、再びその位置は取り出される行の始めになります。

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM TDEPT
  WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {                                          .
  EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;
     .
     .
}
EXEC SQL CLOSE C1;
```

関連資料:

- 107 ページの『CALL』
- 491 ページの『DECLARE CURSOR』

関連サンプル:

- 『dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『tut_read.sqC -- How to read tables (C++)』

COMMENT

COMMENT ステートメントは、種々のオブジェクトのカタログ記述にコメントを追加するか、または置き換えます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

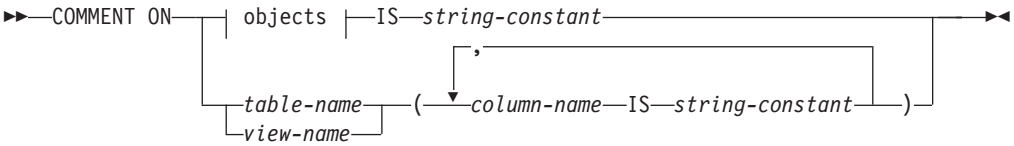
許可:

COMMENT ステートメントの許可 ID が持つ特権には、以下の特権のいずれかが含まれている必要があります。

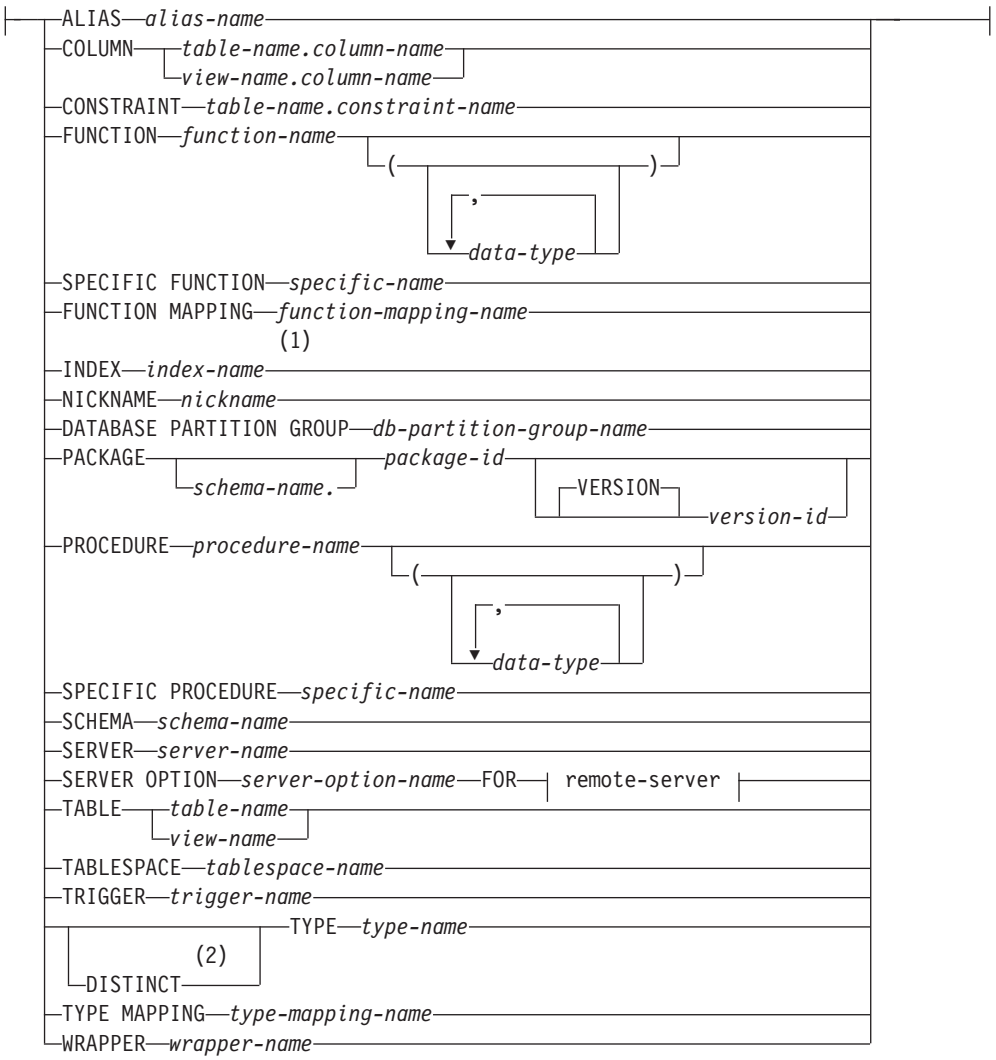
- SYSADM、または DBADM
- オブジェクトのカタログ・ビューの DEFINER 列 (スキーマの場合は OWNER 列) に記録されているオブジェクトの定義者 (列または制約の場合は基礎表)
- スキーマに対する ALTERIN 特権 (複数部分の名前を使用可能なオブジェクトにのみ適用される)
- オブジェクトに対する CONTROL 特権 (索引、パッケージ、表、およびビューの各オブジェクトにのみ適用される)
- オブジェクトに対する ALTER 特権 (表オブジェクトにのみ適用される)

表スペースまたはデータベース・パーティション・グループの場合、許可 ID は SYSADM 権限または SYSCTRL 権限を持っている必要がある点に注意してください。

構文:

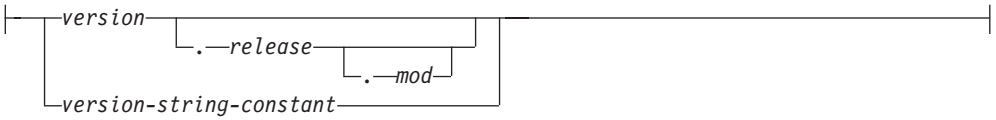


objects:



remote-server:

server-version:



注:

1 *Index-name* には、索引、あるいは索引指定のどちらかの名前を指定できます。

2 DISTINCT の同義語としてキーワード DATA を使用できます。

説明:

ALIAS *alias-name*

alias-name (別名) に対するコメントの追加または置き換えを行うことを指定します。 *alias-name* (別名) は、カタログに記述されている別名を指定する名前で

なければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TABLES カタログ・ビューの別名を記述する行の REMARKS 列の値を置き換えます。

COLUMN *table-name.column-name* または *view-name.column-name*
列に対するコメントを追加または置き換えることを指定します。
table-name.column-name (表名.列名) または *view-name.column-name* (ビュー名.列名) の組み合わせは、カタログに記述されている列と表の組み合わせを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.COLUMNS カタログ・ビューのその列を記述する行の REMARKS 列の値を置き換えます。

作動不能ビューの列にコメントを作成することはできません (SQLSTATE 51024)。

CONSTRAINT *table-name.constraint-name*
制約に対するコメントの追加または置き換えを指定します。
table-name.constraint-name (表名.制約名) の組み合わせは、制約とそれが制約する表を指定していなければなりません。これらは、カタログに記述されていなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TABCONST カタログ・ビューのその制約を記述する行の REMARKS 列の値を置き換えます。

FUNCTION
関数に対するコメントの追加または置き換えを指定します。指定する関数インスタンスは、カタログに記述されたユーザー定義関数、または関数テンプレートでなければなりません。

関数のインスタンスを指定する方法としては、次のようにいくつかの方法があります。

FUNCTION *function-name*
特定の関数を指定します。 *function-name* (関数名) の関数がちょうど 1 つだけ存在している場合にのみ有効です。このように指定する関数には、任意の数のパラメーターが定義されていても構いません。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマにこの名前の関数が存在しない場合は、エラー (SQLSTATE 42704) になります。指定したスキーマまたは暗黙のスキーマに、この関数の特定インスタンスが複数存在する場合は、エラー (SQLSTATE 42725) になります。

FUNCTION *function-name* **(***data-type***,...)**
コメントを付ける関数名を固有に識別する関数シグニチャーを指定します。関数選択のアルゴリズムは使用されません。

function-name
コメントを付ける関数名を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(*data-type***,...)**
これは、CREATE FUNCTION ステートメント上で (対応する位置に)

指定されたデータ・タイプに一致していなければなりません。データ・タイプ *(data-type)* の数、およびそれらのデータ・タイプを論理的に連結したものが、コメントを追加または置換する特定の関数を識別するのに使用されます。

data-type が修飾なしの場合は、 SQL パス上でスキーマを検索することによってタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。代わりに、空の括弧をコーディングすることによって、データ・タイプの一致を調べる際にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントにおける指定に完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

(FOR BIT DATA 属性は、一致検索のためのシグニチャーの一部とは見なされません。したがって、たとえばシグニチャーの中に CHAR FOR BIT DATA が指定されている場合、それは CHAR とだけ定義されている関数と一致し、シグニチャーに CHAR とだけ指定されているものは、 CHAR FOR BIT DATA と指定されている関数と一致することになります。)

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つ関数がない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC FUNCTION *specific-name*
関数のコメントを追加または置換することを指定します (関数を指定する他の方法については、 FUNCTION の部分を参照)。関数の作成時に指定された特定の関数名、またはデフォルト値として使用された特定の関数名を使用して、コメントを付ける特定のユーザー定義関数を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* (特定名) は、指定したスキーマまたは暗黙のスキーマの特定関数のインスタンスを指定していなければなりません。そうでない場合、エラー (SQLSTATE 42704) になります。

SYSIBM、SYSFUN、または SYSPROC スキーマ (SQLSTATE 42832) の関数についてのコメントを付けることはできません。

コメントは、SYSCAT.ROUTINES カタログ・ビューのうち、その関数を記述する行の REMARKS 列の値を置き換えます。

FUNCTION MAPPING *function-mapping-name*
関数マッピングに対するコメントの追加または置き換えを指定します。 *function-mapping-name* (関数マッピング名) は、カタログに記述されている関数マッピングを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.FUNCMAPPINGS カタログ・ビューのうち、その関数マッピングを記述する行の REMARKS 列の値を置き換えます。

INDEX *index-name*
索引または索引指定に対するコメントを追加または置換することを指定します。 *index-name* (索引名) は、カタログに記述されている特定の索引、または索引指定のいずれかを指定していなければなりません (SQLSTATE 42704)。 コメントは、SYSCAT.INDEXES カタログ・ビューのうち、その索引または索引指定を記述する行の REMARKS 列の値を置き換えます。

NICKNAME *nickname*
ニックネームに対するコメントの追加または置き換えを指定します。 *nickname* (ニックネーム) は、カタログに記述されているニックネームでなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TABLES カタログ・ビューのニックネームを記述する行の REMARKS 列の値を置き換えます。

DATABASE PARTITION GROUP *db-partition-group-name*
データベース・パーティション・グループに対するコメントの追加または置き換えを指定します。 *db-partition-group-name* は、カタログに記述されている特定のデータベース・パーティション・グループを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.DBPARTITIONGROUPS カタログ・ビューのうち、そのデータベース・パーティション・グループを記述する行の REMARKS 列の値を置き換えます。

PACKAGE *schema-name.package-id*
パッケージに対するコメントを追加または置換することを指定します。スキーマ名が指定されていない場合、パッケージ ID は暗黙的にデフォルト・スキーマで修飾されます。スキーマ名およびパッケージ ID は、明示的または暗黙的に指定された バージョン ID とともに、カタログに記述されているパッケージを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.PACKAGES カタログ・ビューのうち、そのパッケージを記述する行の REMARKS 列の値を置き換えます。

VERSION *version-id*
コメントを付けるパッケージ・バージョンを指定します。値が指定されない場合には、空ストリングがバージョンのデフォルトになります。パッケージ名は同じですがバージョンが異なる複数のパッケージが存在する場合には、COMMENT ステートメントの単一の呼び出しで、1 つのパッケージ・バージョンにのみ、コメントを付けることができます。 次のような場合は、バージョン ID を二重引用符で区切ってください。

- バージョン ID が VERSION(AUTO) プリコンパイラー・オプションによって生成された場合
- バージョン ID が数字で始まる場合
- バージョン ID が小文字であったり、大小混合である場合

ステートメントをオペレーティング・システムのコマンド・プロンプトから呼び出す場合は、各二重引用符の区切り文字の前に円記号を置いて、オペレーティング・システムによって区切り文字が外されないようにします。

PROCEDURE

プロシージャーに対するコメントを追加または置換することを指定します。指定するプロシージャー・インスタンスは、カタログに記述されたストアード・プロシージャーでなければなりません。

プロシージャー・インスタンスを指定する方法としては、次のようにいくつかの方法があります。

PROCEDURE *procedure-name*

特定のプロシージャーを指定します。スキーマに *procedure-name* のプロシージャーが 1 つだけ存在している場合にのみ有効です。この方法で指定するプロシージャーには、パラメーターがいくつ定義されていても構いません。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマに該当する名前のプロシージャーが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。指定したスキーマまたは暗黙のスキーマにこのプロシージャーの特定のインスタンスが複数存在する場合は、エラーが戻されます (SQLSTATE 42725)。

PROCEDURE *procedure-name* **(***data-type***,...)**

これは、コメントを付けるプロシージャーを固有に識別するプロシージャー・シグニチャーを指定するのに使用されます。

procedure-name

コメントを付けるプロシージャーのプロシージャー名を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(*data-type***,...)**

データ・タイプを指定します。ここで指定されるデータ・タイプは、CREATE PROCEDURE ステートメントの対応する位置に指定されたデータ・タイプと一致していなければなりません。データ・タイプの数、およびそれらのデータ・タイプを論理的に連結したものが、コメントを追加または置換する特定のプロシージャーを識別するのに使用されます。

data-type が修飾なしの場合は、 SQL パス上でスキーマを検索することによってタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。代わりに、空の括弧をコーディングすることによって、データ・タイプの一致を調べる際にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE PROCEDURE ステートメントにおける指定に完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つプロシージャーがない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC PROCEDURE *specific-name*
プロシージャーのコメントを追加または置換することを指定します (プロシージャーを指定する他の方法については、 PROCEDURE を参照)。プロシージャーの作成時に指定されたか、またはデフォルト値として付けられた特定のプロシージャー名を使用して、コメントを付ける特定のストアード・プロシージャーを指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* に指定される名前は、指定したスキーマまたは暗黙のスキーマに含まれる特定プロシージャーのインスタンスを識別するものでなければなりません。それ以外の名前が指定された場合は、エラーが戻されます (SQLSTATE 42704)。

SYSIBM、SYSFUN、または SYSPROC スキーマ (SQLSTATE 42832) のプロシージャーについてのコメントを付けることはできません。

コメントは、SYSCAT.ROUTINES カタログ・ビューのうち、そのプロシージャーを記述する行の REMARKS 列の値を置き換えます。

SCHEMA *schema-name*
スキーマに対するコメントを追加または置換することを指定します。 *schema-name* に指定するスキーマ名は、カタログに記述されているスキーマを識別するものでなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.SCHEMATA カタログ・ビューのうち、そのスキーマを記述する行の REMARKS 列の値を置き換えます。

SERVER *server-name*
データ・ソースに対するコメントの追加または置き換えを指定します。 *server-name* に指定するサーバー名は、カタログに記述されているデータ・ソースを識別するものでなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.SERVERS カタログ・ビューのうち、そのデータ・ソースを記述する行の REMARKS 列の値を置き換えます。

SERVER OPTION *server-option-name* **FOR** *remote-server*
サーバー・オプションに対するコメントの追加または置き換えを指定します。

server-option-name
サーバー・オプションを指定します。このオプションは、カタログに記述されているものでなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.SERVEROPTIONS カタログ・ビューのうち、そのサーバー・オプションを記述する行の REMARKS 列の値を置き換えます。

remote-server
server-option が適用されるデータ・ソースを示します。

SERVER *server-name*
server-option が適用されるデータ・ソースを指定します。 *server-name* (サーバー名) は、カタログに記述されているデータ・ソースを指定していなければなりません。

TYPE *server-type*
server-option が適用されるデータ・ソースのタイプを指定します。たとえば、DB2 Universal Database for OS/390 または Oracle など。*server-type* の指定は、大文字でも小文字でもかまいません。カタログには大文字で格納されます。

VERSION
server-name で指定したデータ・ソースのバージョンを指定します。

version
バージョン番号を指定します。 *version* は整数でなければなりません。

release
version で示されたバージョンのリリース番号を指定します。*release* は整数でなければなりません。

mod
release で示されたリリースのモディフィケーション番号を指定します。 *mod* は整数でなければなりません。

version-string-constant
バージョンの正式名称を指定します。 *version-string-constant* は単一値 (たとえば、'8i') にすることができます。あるいは、*version*、*release*、そして該当する場合は *mod* を連結した値にすることができます (たとえば、'8.0.3')。

WRAPPER *wrapper-name*
server-name で示されるデータ・ソースにアクセスするときに使用するラッパーを指定します。

TABLE *table-name* または *view-name*
表またはビューに対するコメントを追加または置換することを指定します。*table-name* (表名) または *view-name* (ビュー名) は、カタログに記述されている表またはビュー (別名またはニックネームではない) を指定していなければならず (SQLSTATE 42704)、宣言済み一時表を指定してはなりません (SQLSTATE 42995)。コメントは、 SYSCAT.TABLES カタログ・ビューのうち、その表またはビューを記述する行の REMARKS 列の値を置き換えます。

TABLESPACE *tablespace-name*
表スペースに対するコメントを追加または置換することを指定します。

tablespace-name (表スペース名) は、カタログに記述されている特定の表スペースを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TABLESPACES カタログ・ビューのうち、その表スペースを記述する行の REMARKS 列の値を置き換えます。

TRIGGER *trigger-name*
トリガーに対するコメントを追加または置換することを指定します。
trigger-name (トリガー名) は、カタログに記述されている特定のトリガーを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TRIGGERS カタログ・ビューのうち、そのトリガーを記述する行の REMARKS 列の値を置き換えます。

TYPE *type-name*
ユーザー定義タイプのコメントを追加または置換することを指示します。
type-name (タイプ名) は、カタログに記述されているユーザー定義タイプを指定していなければなりません (SQLSTATE 42704)。 DISTINCT 文節が指定されている場合、*type-name* (タイプ名) は、カタログに記述されている特殊タイプを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.DATATYPES カタログ・ビューの REMARKS 列の値を、ユーザー定義タイプを記述する行に置き換えます。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

TYPE MAPPING *type-mapping-name*
ユーザー定義のデータ・タイプのマッピングに対するコメントを追加または置換することを指定します。 *type-mapping-name* (タイプ・マッピング名) は、カタログに記述されているデータ・タイプ・マッピングを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.TYPEMAPPINGS カタログ・ビューのうち、そのマッピングを記述する行の REMARKS 列の値を置き換えます。

WRAPPER *wrapper-name*
ラッパーに対するコメントの追加または置き換えを指定します。
wrapper-name (ラッパー名) は、カタログに記述されているラッパーを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.WRAPPERS カタログ・ビューのうち、そのラッパーを記述する行の REMARKS 列の値を置き換えます。

IS *string-constant*
追加または置換するコメントを指定します。 *string-constant* (ストリング定数) には、最大 254 バイトの任意の文字ストリング定数を指定できます。 (復帰文字 (CR) と改行文字 (LF) はそれぞれ 1 バイトとカウントされます。)

table-name|*view-name* **({** *column-name* **IS** *string-constant* **} ...)**
この形式の COMMENT ステートメントを使用すると、表またはビューの複数の列に対するコメントを指定することができます。列名は修飾できず、各名前は指定する表またはビューの列を指定するものでなければなりません。また、その表またはビューはカタログに記述されていなければなりません。 *table-name* を宣言済み一時表にすることはできません (SQLSTATE 42995)。

作動不能ビューの列にコメントを作成することはできません (SQLSTATE 51024)。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。

例:

例 1: EMPLOYEE 表についてのコメントを追加します。

```
COMMENT ON TABLE EMPLOYEE
  IS 'Reflects first quarter reorganization'
```

例 2: EMP_VIEW1 ビューについてのコメントを追加します。

```
COMMENT ON TABLE EMP_VIEW1
  IS 'View of the EMPLOYEE table without salary information'
```

例 3: EMPLOYEE 表の EDLEVEL 列についてのコメントを追加します。

```
COMMENT ON COLUMN EMPLOYEE.EDLEVEL
  IS 'highest grade level passed in school'
```

例 4: EMPLOYEE 表の異なる 2 つの列についてのコメントを追加します。

```
COMMENT ON EMPLOYEE
  (WORKDEPT IS 'see DEPARTMENT table for names',
  EDLEVEL IS 'highest grade level passed in school' )
```

例 5: Pellow は、自身の PELLOW スキーマに作成した CENTRE 関数についてのコメントを付けます。シグニチャーを使用して、コメントを付ける特定の関数を指定します。

```
COMMENT ON FUNCTION CENTRE (INT,FLOAT)
  IS 'Frank''s CENTRE fctn, uses Chebychev method'
```

例 6: McBride は、PELLOW スキーマに作成した別の CENTRE 関数にコメントを付けます。特定の名前を使用して、コメントを付ける関数インスタンスを指定します。

```
COMMENT ON SPECIFIC FUNCTION PELLOW.FOCUS92 IS
  'Louise''s most triumphant CENTRE function, uses the
       Brownian fuzzy-focus technique'
```

例 7: CHEM スキーマの関数 ATOMIC_WEIGHT にコメントを付けます。このスキーマではこの名前の関数は 1 つしかないことが分かっています。

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT
  IS 'takes atomic nbr, gives atomic weight'
```

例 8: Eigler は、自身の EIGLER スキーマに作成した SEARCH プロシージャーについてのコメントを付けます。シグニチャーを使用して、コメントを付ける特定のプロシージャーを指定します。

```
COMMENT ON PROCEDURE SEARCH (CHAR,INT)
  IS 'Frank''s mass search and replace algorithm'
```

例 9: Macdonald は、EIGLER スキーマに作成した別の SEARCH 関数にコメントを付けます。特定の名前を使用して、コメントを付けるプロシージャー・インスタンスを指定します。

```
COMMENT ON SPECIFIC PROCEDURE EIGLER.DESTROY IS
  'Patrick''s mass search and destroy algorithm'
```

例 10: BIOLOGY スキーマのプロシージャー OSMOSIS にコメントを付けます。このスキーマではこの名前のプロシージャーは 1 つしかないことが分かっています。

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS
  IS 'Calculations modelling osmosis'
```

例 11: INDEXSPEC という索引指定にコメントを付けます。

```
COMMENT ON INDEX INDEXSPEC
  IS 'An index specification that indicates to the optimizer
  that the table referenced by nickname NICK1 has an index.'
```

例 12: デフォルト名が NET8 のラッパーにコメントを付けます。

```
COMMENT ON WRAPPER NET8
  IS 'The wrapper for data sources associated with
      Oracle's Net8 client software.'
```

COMMIT

COMMIT ステートメントは、作業単位を終了し、その作業単位で行われたデータベースへの変更をコミットします。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

必要ありません。

構文:

説明:

COMMIT ステートメントが実行される作業単位が終了すると、新しい作業単位が開始されます。その作業単位で実行された、以下のステートメントによって行われた変更がすべてコミットされます。

ALTER、COMMENT、CREATE、DROP、GRANT、LOCK TABLE、REVOKE、SET INTEGRITY、SET 変数、およびデータ変更ステートメント (INSERT、DELETE、MERGE、UPDATE)。クエリー内でネストされているものを含みます。

ただし、以下のステートメントはトランザクションによって制御されず、これらのステートメントによって行われた変更は COMMIT ステートメントとは独立しています。

- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET EVENT MONITOR STATE
- SET PASSTHRU

  注: SET PASSTHRU ステートメントはトランザクションによって制御されませんが、ステートメントによって開始されたパススルー・セッションはトランザクションにより制御されます。
- SET PATH
- SET SCHEMA

- SET SERVER OPTION

その開始以降に作業単位によって獲得されたロックは、 WITH HOLD を宣言されたオープン・カーソルに必要なロック以外のすべてのロックが解放されます。 WITH HOLD を定義されていないすべてのオープン・カーソルはクローズされます。 WITH HOLD を定義されたオープン・カーソルはオープンされたままになり、そのカーソルは、結果表の次の論理行の前に置かれます。 (位置による UPDATE ステートメントまたは DELETE ステートメントが出される前に、FETCH を実行する必要があります。) LOB ロケーターはすべて解放されます。 WITH HOLD 特性を持つカーソルによって取り出された LOB 値に関連したロケーターも、すべて解放されます。

トランザクション内に設定されたセーブポイントはすべて解放されます。

注:

- 各アプリケーション・プロセスを終了する前に、その作業単位を明示的に終了することを強くお勧めします。アプリケーション・プログラムが COMMIT ステートメントまたは ROLLBACK ステートメントを実行せずに正常に終了すると、データベース・マネージャーはアプリケーションの環境に応じてコミットまたはロールバックを試みます。
- キャッシュされた動的 SQL ステートメントに対する COMMIT の影響については、 『EXECUTE』 を参照してください。
- 宣言済み一時表に対する COMMIT の影響の可能性については、 『DECLARE GLOBAL TEMPORARY TABLE』 を参照してください。

例:

最後のコミット・ポイント以降にデータベースに対して行われた変更をコミットします。

```
COMMIT WORK
```

関連資料:

- 551 ページの『EXECUTE』
- 497 ページの『DECLARE GLOBAL TEMPORARY TABLE』

関連サンプル:

- 『dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『tbsavept.sqc -- How to use external savepoints (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『tut_use.sqc -- How to modify a database (C)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『tut_use.sqC -- How to modify a database (C++)』

コンパウンド SQL (動的)

コンパウンド・ステートメントは、別々のステートメントを実行可能ブロックにグループ化します。動的準備済みアトミック・コンパウンド・ステートメントの内部で SQL 変数を宣言することができます。

呼び出し:

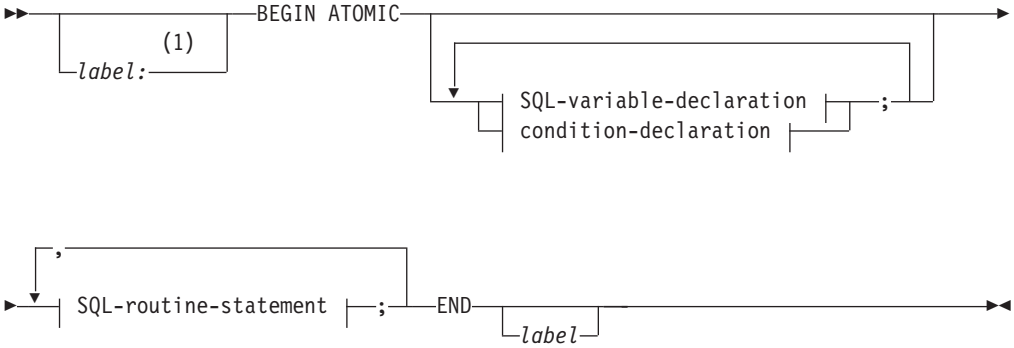
このステートメントはトリガー、SQL 関数、または SQL メソッドに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

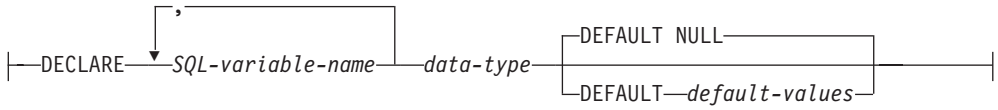
動的コンパウンド・ステートメントを呼び出すために、特権は必要ありません。ただし、コンパウンド・ステートメントの許可 ID に、コンパウンド・ステートメントに組み込まれている SQL ステートメントを呼び出すために必要な特権がなければなりません。

構文:

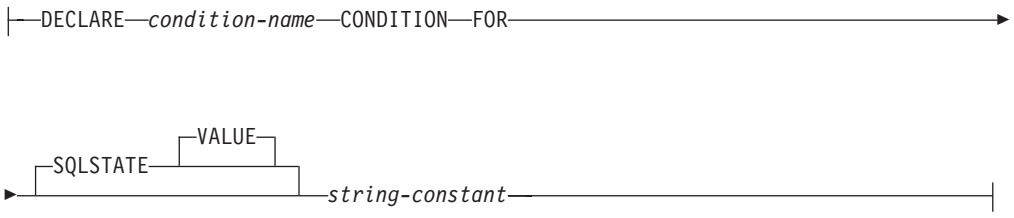
dynamic-compound-statement



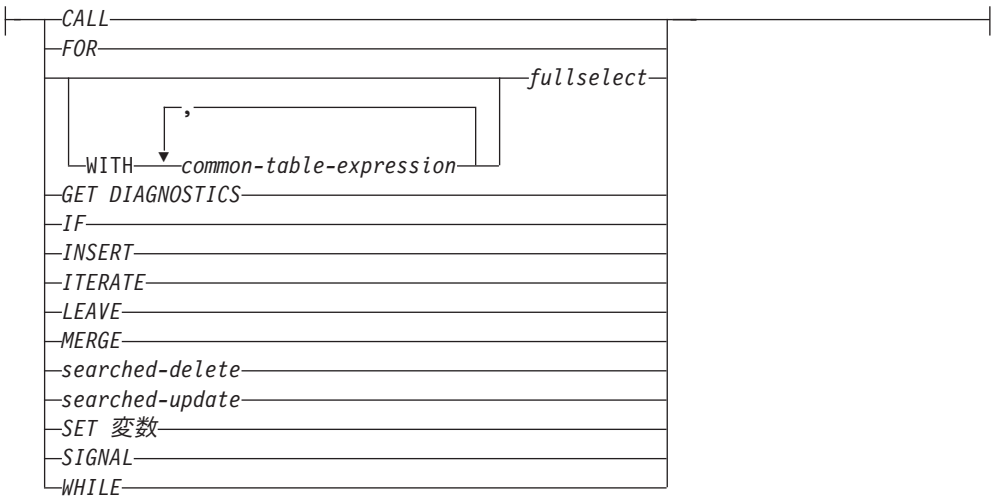
SQL-variable-declaration:



condition-declaration:



SQL-routine-statement:



注:

1 ステートメントが関数、メソッド、またはトリガー定義にある場合のみ、ラベルを指定できます。

説明:

label
コード・ブロックのラベルを定義します。開始ラベルが指定されている場合、そのラベルを使用して動的コンパウンド・ステートメントで宣言されている SQL 変数を修飾したり、ラベルを LEAVE ステートメントに指定したりすることができます。終了ラベルを指定する場合、そのラベルは開始ラベルと同じでなければなりません。

ATOMIC
ATOMIC は、コンパウンド・ステートメントでエラーが起こった場合、そのコンパウンド・ステートメント内の SQL ステートメントがすべてロールバックされ、以降の SQL ステートメントは処理されないことを指示します。

SQL-routine-statement
動的コンパウンド・ステートメント内で使用する SQL ステートメントを指定します。 SQL 関数または SQL メソッド内の動的コンパウンド・ステートメントでは、 RETURN ステートメントも使用できます。コンパウンド SQL 内のニックネームに対する検索更新、検索削除、挿入、またはマージ操作はサポートされません。

SQL-variable-declaration
動的コンパウンド・ステートメントに対してローカルである変数を宣言します。

SQL-variable-name
ローカル変数の名前を定義します。 DB2 は SQL 変数をすべて大文字に変換します。この名前を以下のものと同じにすることはできません。

- コンパウンド・ステートメント内の別の SQL 変数
- パラメーター名

SQL 変数および列参照と同じ名前の ID が SQL ステートメントに含まれている場合、 DB2 はその ID を列と解釈します。

data-type
変数のデータ・タイプを指定します。

DEFAULT *default-values* **または NULL**
SQL 変数のデフォルトを定義します。動的コンパウンド・ステートメントが呼び出されると、この変数は初期化されます。デフォルト値が指定されていない場合、変数は NULL に初期化されます。

condition-declaration
条件名および対応する SQLSTATE 値を宣言します。

condition-name
条件の名前を指定します。条件名はプロシージャー本体内でユニークでなければならず、宣言されたコンパウンド・ステートメント内でのみ参照が可能です。

FOR SQLSTATE *string-constant*
条件に関連する SQLSTATE を指定します。 *string-constant* は引用符で囲んだ 5 文字で指定しなければなりませんが、 '00000' にすることはできません。

注:

- 動的コンパウンド・ステートメントは、DB2 によって単一ステートメントとしてコンパイルされます。このステートメントは、小さな制御フロー・ロジックを含む短いスクリプトに有効ですが、大きな意味を持つデータ・フローには有効ではありません。複雑な制御フローのネストまたは条件処理が必要な大きな構成の場合、 SQL プロシージャーの使用をお勧めします。 SQL プロシージャーの使用法の詳細については、『CREATE PROCEDURE』 を参照してください。
- コンパウンド・ステートメント内で呼び出されるプロシージャーは、 COMMIT または ROLLBACK ステートメントを発行できません (SQLSTATE 42985)。
- ***表アクセスの制限:***

  プロシージャーが READS SQL DATA または MODIFIES SQL DATA として定義されている場合は、プロシージャー内のステートメントは、このプロシージャーを呼び出したコンパウンド・ステートメントによって変更される表にアクセスすることはできません (SQLSTATE 57053)。プロシージャーが MODIFIES SQL DATA として定義されている場合は、プロシージャー内のステートメントは、このプロシージャーを呼び出したコンパウンド・ステートメントによって読み取られるまたは変更される表を変更できません (SQLSTATE 57053)。

例:

例 1:

この例では、データ・クレンジングを行うために、データウェアハウジング・シナリオでインライン SQL PL を使用する方法を示します。

この例には、3 つの表があります。 ″target″ 表には、クレンジングされたデータが入ります。 ″except″ 表にはクレンジングできない行 (例外) が保管され、 ″source″ 表にはクレンジングするロー・データが入ります。

データを分類して変更するために、″discretize″ という単純な SQL 関数が使用されます。これは、不良データの場合はすべて NULL を戻します。次いで、動的コンパ

ウンド・ステートメントがデータをクレンジングします。このステートメントは、FOR ループでソース表の中のすべての行を処理し、 ″discretize″ 関数の結果に従って、現在行を ″target″ 表と ″except″ 表のどちらに挿入するのかを決定します。この技法を使用して、より複雑なメカニズム (複数ステージのクレンジング) にすることができます。

SQL プロシージャー、その他の任意のプロシージャー、またはホスト言語のアプリケーションで、同じコードを作成できます。ただし、動的コンパウンド・ステートメントには独特の利点があります。つまり、FOR ループでカーソルが開くことはなく、単一行挿入も実際の単一行挿入ではありません。実際には、共用選択からの複数表挿入という効果的な論理になっています。

これは、動的コンパウンドを単一ステートメントとしてコンパイルすることによって達成されます。ビューを使用する照会に統合され、照会コンテキスト内で全体としてコンパイルおよび最適化される本体を持つビューと同様に、DB2 オプティマイザーは、制御フローとデータ・フローの両方をコンパイルおよび最適化します。したがって、全体の論理は、DB2 の実行時間内で実行されます。ストアード・プロシージャーの場合とは異なり、 DB2 のコア・エンジンの外部に移動されるデータはありません。

最初のステップでは必要な表を作成します。

```
CREATE TABLE target
  (pk INTEGER NOT NULL
  PRIMARY KEY, c1 INTEGER)
```

クレンジングされたデータを入れるための TARGET という表が作成されます。

```
CREATE TABLE except
  (pk INTEGER NOT NULL
  PRIMARY KEY, c1 INTEGER)
```

例外を入れるための EXCEPT という表が作成されます。

```
CREATE TABLE source
  (pk INTEGER NOT NULL
  PRIMARY KEY, c1 INTEGER)
```

クレンジングするデータを保持する SOURCE という表が作成されます。

次に、[0..1000] の範囲外にあるすべての値を取り除き、それらの値を 10 のステップに整列させることによってデータをクレンジングするための、 ″discretize″ 関数を作成します。

```
CREATE FUNCTION discretize(raw INTEGER) RETURNS INTEGER
  RETURN CASE
    WHEN raw < 0 THEN CAST(NULL AS INTEGER)
    WHEN raw > 1000 THEN NULL
    ELSE ((raw/10) * 10) + 5
  END
```

次いで、次のようにして値を挿入します。

```
INSERT INTO source (pk, c1)
  VALUES (1,   -5),
         (2, NULL),
```

```
          (3, 1200),
          (4,   23),
          (5,   10),
          (6,  876)
```

次のようにして関数を呼び出します。

```
BEGIN ATOMIC
  FOR row AS
    SELECT pk, c1, discretize(c1) AS d FROM source
  DO
    IF row.d is NULL THEN
      INSERT INTO except VALUES(row.pk, row.c1);
    ELSE
      INSERT INTO target VALUES(row.pk, row.d);
    END IF;
  END FOR;
END
```

次のようにして結果をテストできます。

```
SELECT * FROM except ORDER BY 1
PK          C1
----------- -----------
          1          -5
          2           -
          3        1200
  3 record(s) selected.

SELECT * FROM target ORDER BY 1
PK          C1
----------- -----------
          4          25
          5          15
          6         875
  3 record(s) selected.
```

最後のステップとして、次のようにしてクリーンアップを行います。

```
DROP FUNCTION discretize
DROP TABLE source
DROP TABLE target
DROP TABLE except
```

関連資料:

• 311 ページの『CREATE PROCEDURE』

関連サンプル:

• 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』

コンパウンド SQL (組み込み)

1 つまたは複数の異なる SQL ステートメント（サブステートメント）を結合して、 1 つの実行可能なブロックにします。

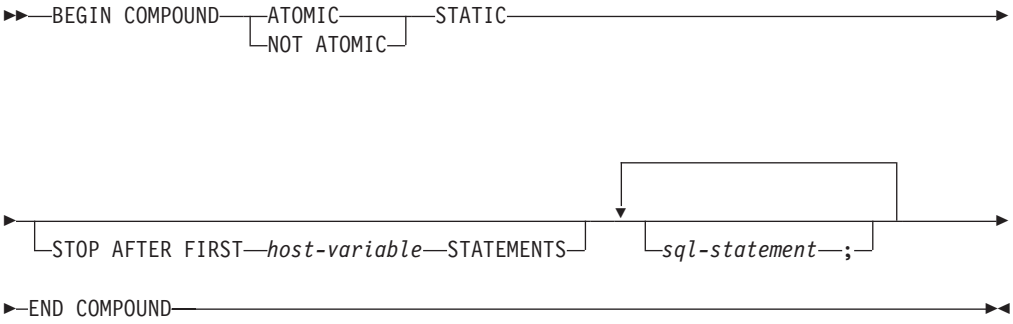
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。コンパウンド SQL ステートメント構成全体は、動的に準備できない実行可能ステートメントです。このステートメントは REXX ではサポートされません。

許可:

コンパウンド SQL ステートメント自体には必要ありません。コンパウンド SQL ステートメントの許可 ID には、コンパウンド SQL ステートメントに含まれる個々のステートメントすべてに対する適切な権限が必要です。

構文:



説明:

ATOMIC
コンパウンド SQL ステートメント内のいずれかのサブステートメントが失敗した場合に、正常なサブステートメントによる変更も含め、サブステートメントによってデータベースに対して行われた変更すべてを取り消すことを指定します。

NOT ATOMIC
サブステートメントのエラーには関係なく、他のサブステートメントによってデータベースに対して行われた変更をコンパウンド SQL ステートメントが取り消さないことを指定します。

STATIC
すべてのサブステートメントに対する入力変数が、その当初の値を保持することを指定します。たとえば、

```
SELECT ... INTO :abc ...
```

上記のステートメントの後に、次のステートメントが続いていると想定します。

```
UPDATE T1 SET C1 = 5 WHERE C2 = :abc
```

この UPDATE ステートメントは、SELECT INTO の後に続く値ではなく、コンパウンド SQL ステートメントの実行開始時の :abc の値が使用されます。

1 つの同じ変数が複数のサブステートメントによって設定される場合、コンパウンド SQL ステートメントの後のその変数の値は、最後のサブステートメントで設定された値になります。

注: 非静的動作はサポートされません。つまり、サブステートメントは、順次ではない方法で実行されているものとして見る必要があり、サブステートメントに相互関係があってはなりません。

STOP AFTER FIRST
特定の数のサブステートメントだけを実行することを指定します。

host-variable
実行されるサブステートメントの数を指定する短整数。

STATEMENTS
STOP AFTER FIRST *host-variable* 文節を完結します。

sql-statement
組み込み静的コンパウンド SQL ステートメントには、以下のものを除くすべての実行可能ステートメントを含めることができます。

```
CALL                FETCH
CLOSE               OPEN
CONNECT             PREPARE
Compound SQL        RELEASE (Connection)
DESCRIBE            ROLLBACK (Transaction)
DISCONNECT          SET CONNECTION
EXECUTE IMMEDIATE
```

注: コンパウンド SQL では、INSERT、UPDATE、および DELETE でのニックネームの使用がサポートされません。

COMMIT ステートメントを含める場合、それは最後のサブステートメントでなければなりません。 COMMIT がこの位置にある場合には、STOP AFTER FIRST *host-variable* STATEMENTS 文節ですべてのサブステートメントが実行されるわけではないことが指定されている場合でも、その COMMIT は発行されます。たとえば、100 個のサブステートメントのあるコンパウンド SQL ブロックで、 COMMIT が最後のサブステートメントであるとします。 STOP AFTER FIRST STATEMENTS 文節で 50 個のサブステートメントしか実行できないことが指定されている場合、 COMMIT は 51 番目のサブステートメントになります。

CONNECT TYPE 2 を使用している場合や、XA 分散トランザクション処理環境で実行している場合に、 COMMIT を組み込むと、エラーが戻されます (SQLSTATE 25000)。

規則:

- DB2 Connect では、コンパウンド SQL ブロックの LOB 列を選択する SELECT ステートメントは、サポートされていません。
- コンパウンド SQL ステートメント内にホスト言語コードを使用することはできません。すなわち、コンパウンド SQL ステートメントを構成するサブステートメント相互間にホスト言語コードを使用することはできません。

- NOT ATOMIC コンパウンド SQL ステートメントのみが、 DB2 Connect により受け入れられます。
- コンパウンド SQL ステートメントはネストできません。
- 準備済み COMMIT ステートメントは、 ATOMIC コンパウンド SQL ステートメントでは許可されていません。

注:

コンパウンド SQL ステートメント全体に対して、1 つの SQLCA が戻されます。その SQLCA の情報の多くは、最後のサブステートメントの処理時にアプリケーション・サーバーによって設定された値を反映しています。たとえば、

- 通常、SQLCODE および SQLSTATE は、最後のサブステートメントに関するものです (例外については次の点で説明します)。
- 'no data found' (データが見つからない) の警告 (SQLSTATE '02000') が戻されると、その警告には他の警告よりも高い優先順位が与えられ、 WHENEVER NOT FOUND 例外を立ち上げることができるようになります。 (これは、最終的にアプリケーションに戻される SQLCA 内の SQLCODE、SQLERRML、SQLERRMC、および SQLERRP の各フィールドが、 'no data found' の警告を引き起こしたサブステートメントによるものであることを意味しています。コンパウンド SQL ステートメントに複数の 'no data found' の警告が生じた場合、戻されるフィールドは最後のサブステートメントに関するフィールドです。)
- SQLWARN 標識は、すべてのサブステートメントに対する標識の累計になります。

NOT ATOMIC コンパウンド SQL の実行の過程で 1 つまたは複数のエラーが発生し、その中に重大なエラーが含まれていない場合には、SQLERRMC には、それらのエラーのうち最大 7 つに関する情報が入れられます。 SQLERRMC の最初のトークンは、発生したエラーの合計数を示します。残りのトークンには、コンパウンド SQL ステートメント内でエラーが生じたサブステートメントの順序位置と SQLSTATE が含まれます。これは、次の形式の文字ストリングです。

```
nnnXsssccccc
```

X で始まるサブストリングは最大 6 回まで繰り返され、各ストリング・エレメントは、次のように定義されます。

nnn エラーが生じたステートメントの合計数。 (数が 999 を超えると、カウントは 0 から再び始まります。) このフィールドは左寄せされ、ブランクで埋められます。

X トークン区切り記号 X'FF'。

sss エラーが生じたステートメントの順序位置。 (数が 999 を超えると、カウントは 0 から再び始まります。) たとえば、最初のステートメントが失敗した場合、このフィールドには数字の 1 が左寄せで入れられます ('1 ')。

ccccc エラーの SQLSTATE。

2 番目の SQLERRD フィールドには、エラーが生じたステートメント (負の SQLCODE が戻される) の数が入れられます。

SQLCA の 3 番目の SQLERRD フィールドは、すべてのサブステートメントによって影響を受けた行の数の累計です。

SQLCA の 4 番目の SQLERRD は、正常に実行されたサブステートメントの数を示します。たとえば、コンパウンド SQL ステートメント内の 3 番目のサブステートメントが失敗した場合、 4 番目の SQLERRD フィールドは 2 に設定され、 2 つのサブステートメントが正常に処理された後でエラーが発生したことを示します。

SQLCA の 5 番目の SQLERRD フィールドは、制約活動を引き起こしたすべてのサブステートメントに対して参照保全制約を課すことによって更新または削除された行の数の累計です。

例:

例 1: C プログラムで、ACCOUNTS 表と TELLERS 表の両方を更新するコンパウンド SQL ステートメントを発行します。ステートメントのいずれかにエラーがある場合は、すべてのステートメントの結果を取り消します (ATOMIC)。エラーがない場合は、現行の作業単位をコミットします。

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
  UPDATE ACCOUNTS SET ABALANCE = ABALANCE + :delta
    WHERE AID = :aid;
  UPDATE TELLERS SET TBALANCE = TBALANCE + :delta
    WHERE TID = :tid;
  INSERT INTO TELLERS (TID, BID, TBALANCE) VALUES (:i, :branch_id, 0);
  COMMIT;
END COMPOUND;
```

例 2: C プログラムで、データベースに 10 行のデータを挿入します。ホスト変数 :nbr の値は 10 であり、また S1 は準備済みの INSERT ステートメントであると想定します。さらに、エラーに関係なくすべての挿入を試行するものとします (NOT ATOMIC)。

```
EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC STOP AFTER FIRST :nbr STATEMENTS
  EXECUTE S1 USING DESCRIPTOR :*sqlda0;
  EXECUTE S1 USING DESCRIPTOR :*sqlda1;
  EXECUTE S1 USING DESCRIPTOR :*sqlda2;
  EXECUTE S1 USING DESCRIPTOR :*sqlda3;
  EXECUTE S1 USING DESCRIPTOR :*sqlda4;
  EXECUTE S1 USING DESCRIPTOR :*sqlda5;
  EXECUTE S1 USING DESCRIPTOR :*sqlda6;
  EXECUTE S1 USING DESCRIPTOR :*sqlda7;
  EXECUTE S1 USING DESCRIPTOR :*sqlda8;
  EXECUTE S1 USING DESCRIPTOR :*sqlda9;
END COMPOUND;
```

関連資料:

- 140 ページの『コンパウンド SQL (プロシージャー)』

関連サンプル:

- 『dbuse.sqc -- How to use a database (C)』
- 『dbuse.sqC -- How to use a database (C++)』

コンパウンド SQL (プロシージャー)

コンパウンド・ステートメントは、別のステートメントを SQL プロシージャーにグループ化します。コンパウンド・ステートメントで宣言できるのは、SQL 変数、カーソル、および条件ハンドラーです。

呼び出し:

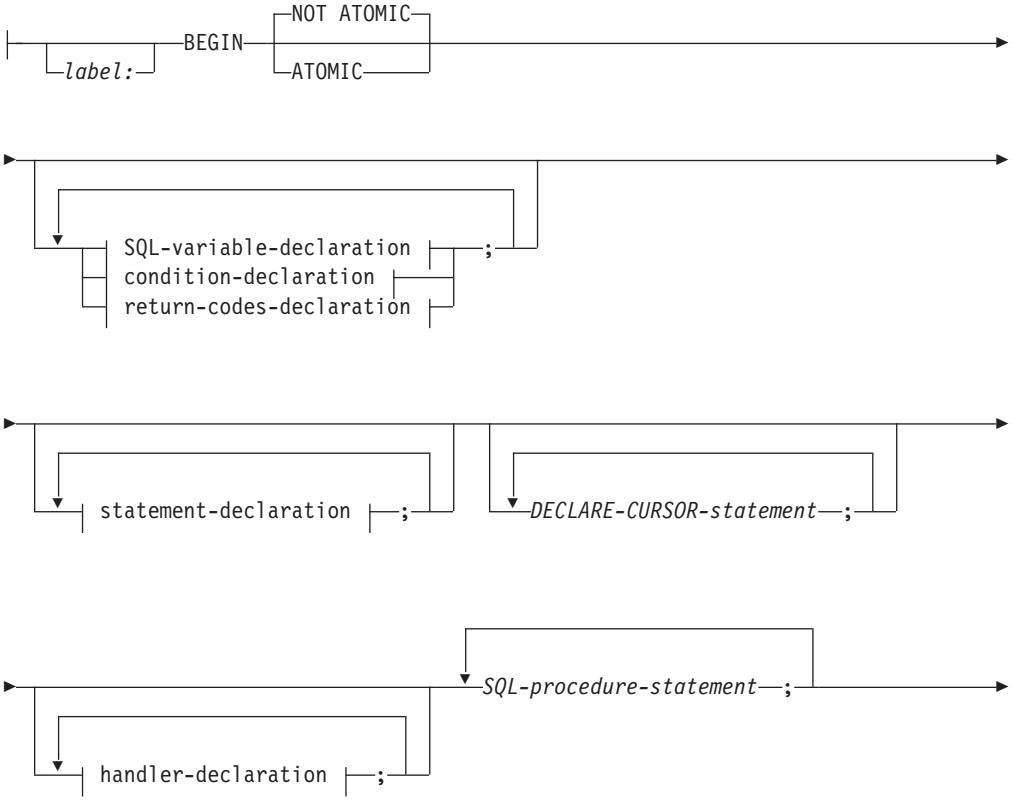
このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

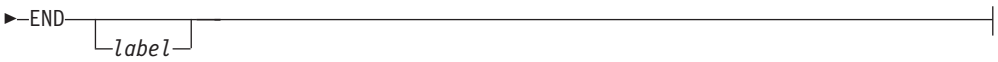
許可:

プロシージャー・コンパウンド・ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID に、プロシージャー・コンパウンド・ステートメントに組み込まれている SQL ステートメントを呼び出すために必要な特権がなければなりません。カーソルの使用に必要な許可については、DECLARE CURSOR ステートメントの説明を参照してください。

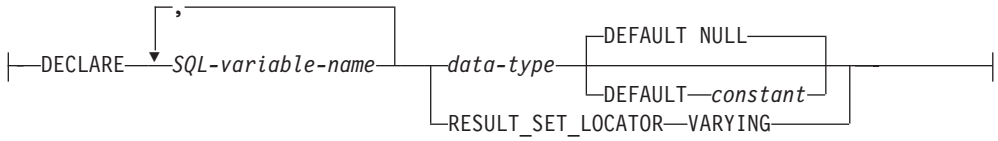
構文:

procedure-compound-statement:

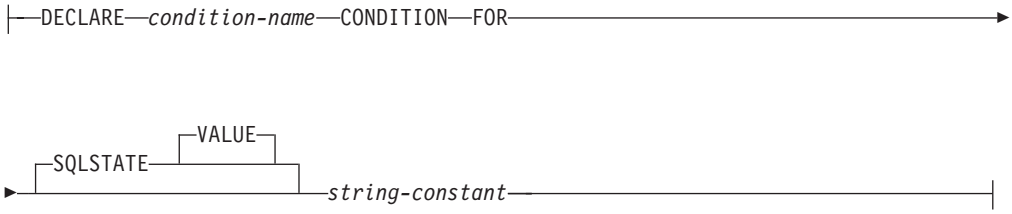




SQL-variable-declaration:

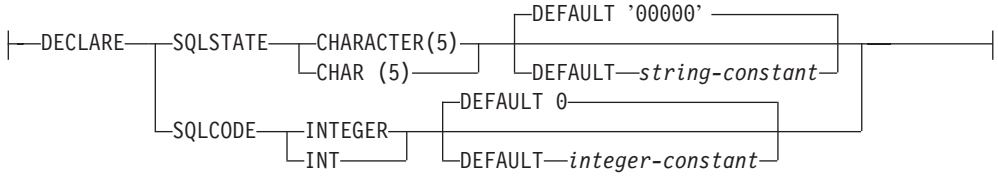


condition-declaration:

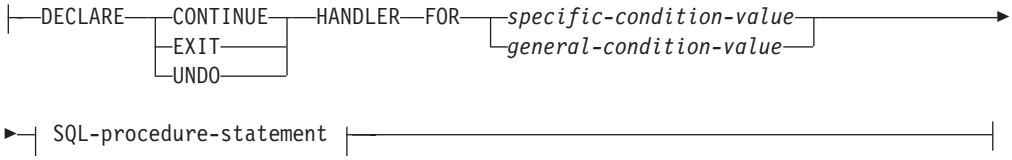


statement-declaration:

return-codes-declaration:



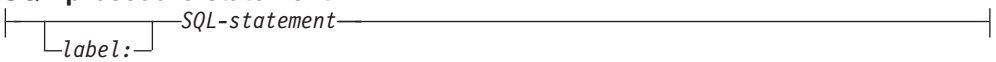
handler-declaration:



specific-condition-value:

general-condition-value:

SQL-procedure-statement:



説明:

label
コード・ブロックのラベルを定義します。開始ラベルを指定した場合、そのラベルを使用して、コンパウンド・ステートメントで宣言する SQL 変数を修飾することができます。また、開始ラベルは LEAVE ステートメントで指定することもできます。終了ラベルを指定する場合、そのラベルは開始ラベルと同じでなければなりません。

ATOMIC または NOT ATOMIC
ATOMIC は、コンパウンド・ステートメントで未処理の例外条件が発生したときに、そのコンパウンド・ステートメント内のすべての SQL ステートメントをロールバックします。 NOT ATOMIC は、コンパウンド・ステートメントで未処理の例外条件が発生しても、そのコンパウンド・ステートメントをロールバックしません。

SQL-variable-declaration
コンパウンド・ステートメントに対してローカルな変数を宣言します。

SQL-variable-name
ローカル変数の名前を定義します。 DB2 は SQL 変数をすべて大文字に変換します。この名前は、同じコンパウンド・ステートメント内にある別の SQL 変数と同じにすることはできず、パラメーター名と同じにすることもできません。 SQL 変数名は、列名と同じにすることはできません。 SQL 変数および列参照と同じ名前の ID が SQL ステートメントに含まれている場合、 DB2 はその ID を列と解釈します。変数が宣言されているコンパウンド・ステートメントにラベルを付けると、変数の使用法をラベルで修飾できます。たとえば、ラベル C を付けたコンパウンド・ステートメント中で変数 V を宣言すると、その変数を C.V として参照できます。

data-type
変数のデータ・タイプを指定します。 LONG VARCHAR、LONG VARGRAPHIC、DATALINK、REFERENCE、およびユーザー定義構造タイプはサポートされません (SQLSTATE 429BB)。

DEFAULT *constant* **または NULL**
SQL 変数のデフォルトを定義します。 SQL プロシージャーが呼び出された時点で、この変数は初期化されます。デフォルト値が指定されていない場合、変数は NULL に初期化されます。

RESULT_SET_LOCATOR VARYING
結果セット・ロケーター変数のデータ・タイプを指定します。

condition-declaration
条件名および対応する SQLSTATE 値を宣言します。

condition-name
条件の名前を指定します。条件名はプロシージャー本体内でユニークでなければならず、宣言されたコンパウンド・ステートメント内でのみ参照が可能です。

FOR SQLSTATE *string-constant*
条件に関連付ける SQLSTATE を指定します。 string-constant は単一引用符で囲まれている 5 つの文字として指定しなければならず、 '00000' にすることはできません。

statement-declaration
コンパウンド・ステートメントの 1 つ以上のローカルの名前を宣言します。ステートメント名は、同じコンパウンド・ステートメント内の別のステートメント名と同じにすることはできません。

return-codes-declaration
SQLSTATE および SQLCODE という特殊変数を宣言します。これらの変数は、SQL ステートメントの処理後に戻される値に自動的に設定されます。SQLSTATE および SQLCODE 変数は両方とも、 SQL プロシージャー本体の最外部のコンパウンド・ステートメントでしか宣言できません。これらの変数は、SQL プロシージャーごとに一度しか宣言できません。

declare-cursor-statement
プロシージャー本体にカーソルを宣言します。カーソルごとにユニークな名前を使用しなければなりません。カーソルを参照できるのは、コンパウンド・ステートメントの中からだけです。カーソルをオープンする場合は OPEN ステートメントを、カーソルを使用して行を読み取る場合は FETCH ステートメントを使用します。 SQL プロシージャーからクライアント・アプリケーションに結果セットを戻す場合、 WITH RETURN 文節を使用してカーソルを宣言しなければなりません。以下の例では、クライアント・アプリケーションに結果セットを 1 つ戻します。

```
CREATE PROCEDURE RESULT_SET()
  LANGUAGE SQL
  RESULT SETS 1
  BEGIN
    DECLARE C1 CURSOR WITH RETURN FOR
      SELECT id, name, dept, job
        FROM staff;
    OPEN C1;
END
```

注: 結果セットを処理する場合、 DB2 コール・レベル・インターフェース (DB2 CLI)、ODBC (Java Database Connectivity)、 JDBC (Java Database Connectivity)、 Java Embedded SQL (SQLJ) のいずれかのアプリケーション・プログラミング・インターフェースを使用して、クライアント・アプリケーションを作成しなければなりません。

カーソルの宣言について詳しくは、『DECLARE CURSOR』 を参照してください。

handler-declaration
ハンドラー (つまり、コンパウンド・ステートメントで例外または完了条件が発

生した場合に実行する *SQL-procedure-statement*) を指定します。
SQL-procedure-statement は、ハンドラーが制御を受け取る際に実行するステートメントです。

ハンドラーがアクティブになる対象は、そのハンドラーが宣言されているコンパウンド・ステートメント (ネストされたコンパウンド・ステートメントを含む) 中の、 *handler-declarations* の集合の後の *SQL-procedure-statements* の集合です。

条件ハンドラーには以下の 3 つのタイプがあります。

CONTINUE
ハンドラーが正常に呼び出された後に、例外を発生したステートメントの後の SQL ステートメントに制御が戻されます。例外を発生したエラーが FOR、 IF、 CASE、 WHILE、または REPEAT ステートメント (ただし、それらのいずれかのステートメントの中の SQL-procedure-statement は除く) の場合、制御は、 END FOR、 END IF、 END CASE、 END WHILE、または END REPEAT の後のステートメントに戻されます。

EXIT
ハンドラーが正常に呼び出された後に、ハンドラーを宣言したコンパウンド・ステートメントの最後に制御が戻されます。

UNDO
ハンドラーが呼び出される前に、コンパウンド・ステートメントで行われたあらゆる SQL の変更がロールバックされます。ハンドラーが正常に呼び出された後に、ハンドラーを宣言したコンパウンド・ステートメントの最後に制御が戻されます。 UNDO を指定する場合は、ハンドラーを宣言しているコンパウンド・ステートメントを ATOMIC にしなければなりません。

以下のようなハンドラーを活動化する条件を handler-declaration 中に定義します。

specific-condition-value
ハンドラーが、*特定条件*ハンドラー であることを指定します。

SQLSTATE *string*
ハンドラーが呼び出される SQLSTATE を指定します。 SQLSTATE 値の先頭 2 文字は "00" でなければなりません。

condition-name
ハンドラーが呼び出される条件名を指定します。条件名は、条件宣言であらかじめ定義していなければなりません。

general-condition-value
ハンドラーが、*一般条件*ハンドラー であることを指定します。

SQLEXCEPTION
例外条件が発生した場合に呼び出されるハンドラーを指定します。例外条件は、最初の 2 文字が "00"、"01"、または "02" ではない SQLSTATE 値で表されます。

SQLWARNING
警告条件が発生した場合に呼び出されるハンドラーを指定します。警告条件は、最初の 2 文字が "01" の SQLSTATE 値で表されます。

NOT FOUND
NOT FOUND 条件が発生した場合に呼び出されるハンドラーを指定します。 NOT FOUND 条件は、最初の 2 文字が "02" の SQLSTATE 値で表されます。

SQL-procedure-statement
SQL プロシージャー・ステートメントを指定します。

label
SQL プロシージャー・ステートメントのラベルを指定します。ラベルは、リスト内でネストされたコンパウンド・ステートメントを含め、SQL プロシージャー・ステートメントのリスト内でユニークでなければなりません。ネストされていないコンパウンド・ステートメントは、同じラベルを使用できることに注意してください。 SQL プロシージャー・ステートメントのリストは、おそらく SQL 制御ステートメントの中にあります。

SQL-statement
SQL プロシージャーの本体には、以下を除き、実行可能なすべての SQL ステートメントを入れることができます。

- ALTER
- CONNECT
- 索引、表、またはビュー以外のオブジェクトの CREATE
- DESCRIBE
- DISCONNECT
- 索引、表、またはビュー以外のオブジェクトの DROP
- FLUSH EVENT MONITOR
- REFRESH TABLE
- RELEASE (接続のみ)
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- SET CONNECTION
- SET INTEGRITY
- SET PASSTHRU
- SET SERVER OPTION

以下のステートメントは、SQL プロシージャーの有効範囲内でのみサポートされます。

- ALLOCATE CURSOR
- ASSOCIATE LOCATORS
- CASE
- GOTO

- LOOP
- コンパウンド SQL (プロシージャー)
- REPEAT
- RESIGNAL

規則:

- ATOMIC コンパウンド・ステートメントはネストできません。
- ハンドラーの宣言には、以下の規則が適用されます。
  - ハンドラーの宣言では、同一の *condition-name* または SQLSTATE 値を複数回含めることはできません。また、SQLSTATE 値と、その同じ SQLSTATE 値を表す *condition-name* を含めることもできません。
  - コンパウンド・ステートメント中で複数の条件ハンドラーが宣言されている場合、以下の規則が適用されます。
    - 2 つのハンドラー宣言に、同一の一般条件カテゴリー (SQLEXCEPTION、SQLWARNING、 NOT FOUND) を指定することはできません。
    - 2 つのハンドラー宣言に、同一の値を表す SQLSTATE 値または *condition-name* として、同一の特定条件カテゴリーを指定することはできません。
  - 例外または完了条件が発生した場合、その条件に最も適したハンドラーが有効になります。以下の考慮事項に基づいて、最も適したハンドラーが判別されます。
    - ハンドラー宣言 *H* の有効範囲は、*H* のあるコンパウンド・ステートメント中に含まれるハンドラー宣言の後の *SQL-procedure-statement* のリストです。したがって、*H* の有効範囲には、条件ハンドラー *H* の本体中に含まれるステートメントは入りません。つまり、条件ハンドラーは条件ハンドラー自体の本体中で生じる条件を処理できないことになります。同様に、同一のコンパウンド・ステートメント中で *H1* と *H2* という 2 つのハンドラーが宣言されている場合、*H1* は *H2* の本体中で生じる条件を処理できず、 *H2* は *H1* の本体中で生じる条件を処理できません。
    - 内側の有効範囲で宣言された *specific-condition-value* または *general-condition-value* C のハンドラーは、外側の有効範囲で宣言された C の別のハンドラーより優先します。
    - 条件 *C* に関する特定ハンドラーと、同じく *C* を処理する一般ハンドラーが、同一の有効範囲内で宣言されている場合は、特定ハンドラーの方が一般ハンドラーより優先します。

    適したハンドラーのない例外条件が生じた場合は、失敗したステートメントに含まれる SQL プロシージャーは、未処理の例外条件で終了します。該当するハンドラーのない完了条件が生じた場合は、引き続き次の SQL ステートメントが実行されます。

例:

以下の処置を実行するコンパウンド・ステートメントが含まれている、プロシージャー本体を作成します。

1. SQL 変数を宣言します。

2. IN パラメーターによって判別される部門の従業員の給与を戻すカーソルを宣言します。 SELECT ステートメントで、 *salary* 列のデータ・タイプを DECIMAL から DOUBLE にキャストします。
3. 条件 NOT FOUND (ファイル終わり) に EXIT ハンドラーを宣言します。これにより、値 '6666' が OUT パラメーター medianSalary に割り当てられます。
4. 指定された部門の従業員の数を選択して SQL 変数 numRecords に入れます。
5. 50% + 1 の従業員が検索されるまで、WHILE ループのカーソルから行を取り出します。
6. 給与の中央値を戻します。

```
CREATE PROCEDURE DEPT_MEDIAN
  (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INTEGER DEFAULT 1;
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE c1 CURSOR FOR
       SELECT CAST(salary AS DOUBLE) FROM staff
         WHERE DEPT = deptNumber
         ORDER BY salary;
    DECLARE EXIT HANDLER FOR NOT FOUND
      SET medianSalary = 6666;
-- initialize OUT parameter
      SET medianSalary = 0;
      SELECT COUNT(*) INTO v_numRecords FROM staff
        WHERE DEPT = deptNumber;
      OPEN c1;
      WHILE v_counter < (v_numRecords/2 + 1) DO
            FETCH c1 INTO medianSalary;
            SET v_counter = v_counter + 1;
      END WHILE;
      CLOSE c1;
END
```

以下の例は、RESIGNAL の結果として別の条件から UNDO ハンドラーが活動化される場合を仮定して、実行の流れを図示しています。

```
CREATE PROCEDURE A()
LANGUAGE SQL
CS1: BEGIN ATOMIC
  DECLARE C CONDITION FOR SQLSTATE '12345';
  DECLARE D CONDITION FOR SQLSTATE '23456';

  DECLARE UNDO HANDLER FOR C
  H1: BEGIN
    -- Rollback after error, perform final cleanup, and exit
    -- procedure A.

    -- ...

    -- When this handler completes, execution continues after
    -- compound statement CS1; procedure A will terminate.
    END;

  -- Perform some work here ...
  CS2: BEGIN
    DECLARE CONTINUE HANDLER FOR D
    H2: BEGIN
      -- Perform local recovery, then forward the error
      -- condition to the outer handler for additional
      -- processing.

      -- ...
```

```
        RESIGNAL C; -- will activate UNDO handler H1; execution
                    -- WILL NOT return here.  Any local cursors
                    -- declared in H2 and CS2 will be closed.
      END;

      -- Perform some more work here ...

      -- Simulate raising of condition D by some SQL statement
      -- in compound statement CS2:
      SIGNAL D; -- will activate H2
    END;
  END
```

関連資料:

- 491 ページの『DECLARE CURSOR』
- *SQL リファレンス 第 1 巻* の『データ・タイプ』

CONNECT (タイプ 1)

CONNECT (タイプ 1) ステートメントは、リモート作業単位の規則に従って、指定したアプリケーション・サーバーにアプリケーション・プロセスを接続します。

1 つのアプリケーション・プロセスは、一時点で 1 つのアプリケーション・サーバーにのみ接続できます。これは、*現行サーバー* と呼ばれます。デフォルトのアプリケーション・サーバーは、アプリケーション・リクエスターの初期設定時に確立されます。暗黙接続が使用可能な場合にアプリケーション・プロセスが開始されると、暗黙でデフォルトのアプリケーション・プロセスに接続されます。そのアプリケーション・プロセスで CONNECT TO ステートメントを使用することによって、それとは別のアプリケーション・サーバーに明示的に接続することもできます。接続は、CONNECT RESET ステートメント、または DISCONNECT が出されるまで、あるいは別の CONNECT TO ステートメントによってアプリケーション・サーバーが変更されるまで継続します。

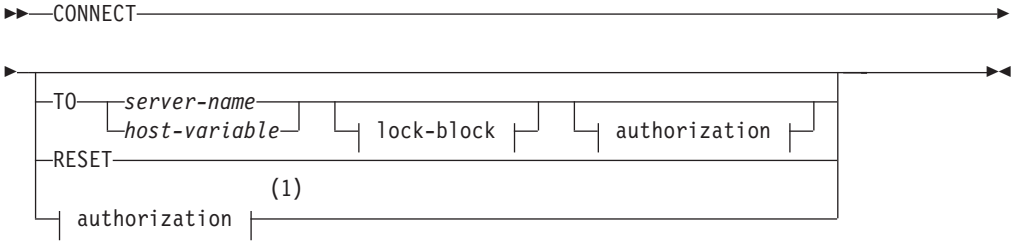
呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

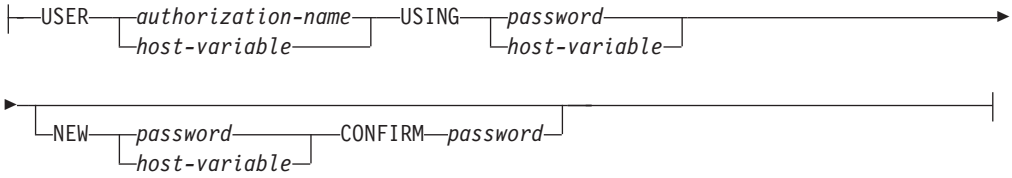
許可:

このステートメントの許可 ID には、指定されたアプリケーション・サーバーに接続するための許可が必要です。データベースの認証の設定値によっては、クライアントまたはサーバーのいずれかによって許可検査が行われる場合があります。パーティション・データベースの場合、ユーザーとグループの定義は、パーティションのすべてにわたって同一である必要があります。

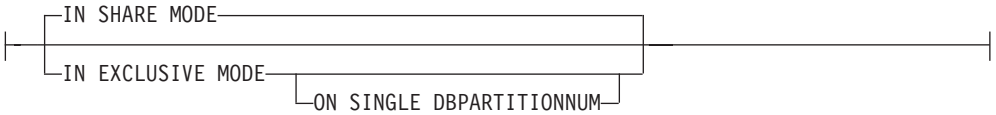
構文:



authorization:



lock-block:



注:

1 この形式は、暗黙の接続が使用可能な場合にのみ有効です。

説明:

CONNECT (オペランドなし)
現行サーバーに関する情報を戻します。この情報は、「正常に接続された場合」の項に説明されているとおり、 SQLCA の SQLERRP フィールドに戻されます。

接続状態が存在する場合、許可 ID とデータベース別名が、SQLCA の SQLERRMC フィールドに入れられます。権限 ID が 8 バイトを超える場合、これは 8 バイトに切り捨てられ、 SQLCA の SQLWARN0 および SQLWARN1 フィールドにそれぞれ 'W' と 'A' のフラグが付きます。データベース構成パラメーター DYN_QUERY_MGMT が使用可能の場合、 SQLCA の SQLWARN0 および SQLWARN7 フィールドにはそれぞれ 'W' と 'E' のフラグが付きます。

接続が存在せず、暗黙接続が可能な場合は、暗黙接続が試みられます。暗黙接続が使用可能でない場合、この試みはエラーになります (既存の接続がない)。接続がない場合、SQLERRMC フィールドはブランクになります。

アプリケーション・サーバーのテリトリー・コードとコード・ページは、SQLERRMC フィールドに入れられます (正常に実行される CONNECT TO ステートメントの場合と同じ)。

この形式の CONNECT を使用する場合、

- アプリケーション・プロセスが接続可能状態である必要はありません。
- 接続されている場合、接続状態は変わりません。
- 接続されておらず、暗黙接続が使用可能な場合は、デフォルトのアプリケーション・サーバーとの接続が行われます。この場合、正常に実行される CONNECT TO ステートメントの場合と同様に、アプリケーション・サーバーの国または地域別コードとコード・ページが、SQLERRMC フィールドに入れられます。
- 未接続で暗黙的接続が不能な場合、アプリケーション・プロセスは未接続のままになります。
- カーソルをクローズしません。

TO *server-name* または *host-variable*
server-name (サーバー名) またはそのサーバー名が入る *host-variable* (ホスト変数) を指定することによって、アプリケーション・サーバーを指定します。

host-variable (ホスト変数) を指定する場合、それは、長さ属性が 8 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。その *host-variable* に入る *server-name* は、左寄せする必要があり、引用符で区切ることはできません。

server-name は、アプリケーション・サーバーを指定するデータベース別名である点に注意してください。 この名前は、アプリケーション・リクエスターのローカル・ディレクトリーにリストされている必要があります。

注: DB2 UDB for OS/390 and z/OS は 16 バイトのロケーション名をサポートし、 DB2 UDB for iSeries は 18 バイトのターゲット・データベース名をサポートします。 DB2 バージョン 8 では、 SQL CONNECT ステートメントに対して 8 バイトのデータベース別名を使用することだけがサポートされています。ただし、データベース別名は、データベース接続サービス・ディレクトリーによって、 18 バイトのデータベース名にマッピングすることができます。

CONNECT TO ステートメントが実行される時点で、アプリケーション・プロセスは接続可能状態でなければなりません。

正常に接続された場合

CONNECT TO ステートメントが正常に実行された場合、

- オープンされていたカーソルはすべてクローズされ、準備済みのステートメントはすべて破棄されて、以前のアプリケーション・サーバーからのすべてのロックが解放されます。
- アプリケーション・プロセスはそれ以前のアプリケーション・サーバー (ある場合) から切断され、指定されたアプリケーション・サーバーに接続されます。
- そのアプリケーション・サーバーの実際の名前 (別名ではなく) が CURRENT SERVER 特殊レジスターに入れられます。
- アプリケーション・サーバーに関する情報が、SQLCA の SQLERRP フィールドに入れられます。アプリケーション・サーバーが IBM 製品の場合、その情報は *pppvvrrm* の形式です。ここで、
  - *ppp* は、以下のように製品を示します。
    - DB2 UDB for OS/390 and z/OS の場合は DSN
    - DB2 Server for VSE & VM の場合は ARI
    - DB2 UDB for iSerie の場合は QSQ
    - DB2 UDB for UNIX および Windows の場合は SQL
  - *vv* は、2 桁のバージョン ID です ('08'など)。
  - *rr* は、2 桁のリリース ID です ('01' など)。
  - *m* は、1 桁の修正レベル ID です ('0'など)。

  DB2 UDB for UNIX および Windows のこのリリース (バージョン 8) は、'SQL08010' として識別されます。
- SQLCA の SQLERRMC フィールドは、次の値を含む (X'FF' で区切って) ように設定されます。
  1. アプリケーション・サーバーの国または地域コード (DB2 Connect を使用している場合はブランク)。
  2. アプリケーション・サーバーのコード・ページ (DB2 Connect を使用している場合は CCSID)。

3. 権限 ID (最初の 8 バイトまで)。
4. データベース別名。
5. アプリケーション・サーバーのプラットフォーム・タイプ。現在識別される値は次のとおりです。

トークン	サーバー
QAS	DB2 Universal Database for iSeries
QDB2	DB2 Universal Database for OS/390 and z/OS
QDB2/2	DB2 Universal Database for OS/2
QDB2/6000	DB2 Universal Database for AIX
QDB2/HPUX	DB2 Universal Database for HP-UX
QDB2/LINUX	DB2 Universal Database for Linux
QDB2/NT	DB2 Universal Database for Windows NT、2000、および XP
QDB2/SUN	DB2 Universal Database for Solaris Operating System
QSQLDS/VM	DB2 Server for VM
QSQLDS/VSE	DB2 Server for VSE

6. エージェント ID。これは、アプリケーションに代わってデータベース・マネージャー内で実行されるエージェントを指定します。このフィールドは、データベース・モニターによって戻される agent_id エレメントと同じです。
7. エージェント索引。これは、エージェントの索引を識別し、サービスに使用されます。
8. パーティション番号。非パーティション・データベースの場合は、この値は常に 0 です (存在する場合)。
9. アプリケーション・クライアントのコード・ページ。
10. パーティション・データベースのパーティションの数。データベースをパーティション化できない場合、値は 0 (ゼロ) になります。トークンは、バージョン 5 またはそれ以降の場合にのみ存在します。

- SQLCA の SQLERRD(1) フィールドは、アプリケーション・コード・ページからデータベース・コード・ページへの変換が行われる際に見込まれる混合文字データ（CHAR データ・タイプ）の長さの最大差を示します。値 0 または 1 は、拡張がないことを示します。 1 より大きい値は、その長さの分の拡張が見込まれることを示します。負の値は、切り捨てが見込まれることを示します。
- SQLCA の SQLERRD(2) フィールドは、データベース・コード・ページからアプリケーション・コード・ページへの変換が行われる際に見込まれる混合文字データ（CHAR データ・タイプ）の長さの最大差を示します。値 0 または 1 は、拡張がないことを示します。 1 より大きい値は、その長さの分の拡張が見込まれることを示します。負の値は、切り捨てが見込まれることを示します。

- SQLCA の SQLERRD(3) フィールドは、接続されているデータベースが更新可能か否かを示します。データベースは、最初は更新可能ですが、その許可 ID で更新を実行できないことが作業単位で明らかになると、読み取り専用に変更されます。この値は、次のいずれかです。
  - 1 - 更新可能
  - 2 - 読み取り専用
- SQLCA の SQLERRD(4) フィールドは、接続の特定の特性を戻します。この値は、次のいずれかです。

  0 なし (1 フェーズ・コミットで更新元でもある下位レベル・クライアントから実行している場合のみ可能)。

  1 1 フェーズ・コミット。

  2 1 フェーズ・コミット。読み取り専用 (TP モニター環境にある DRDA1 データベースとの接続にのみ適用可能)。

  3 2 フェーズ・コミット。
- SQLCA の SQLERRD(5) フィールドは、接続のための認証タイプを戻します。この値は、次のいずれかです。

  0 サーバーで認証された。

  1 クライアントで認証された。

  2 DB2 Connect を使用して認証された。

  4 暗号化を伴ってサーバーで認証された。

  5 暗号化を伴い、DB2 Connect を使用して認証された。

  7 外部 Kerberos セキュリティー機構を使用して認証された。

  8 外部 Kerberos セキュリティー機構を使用して、または暗号化を伴ってサーバーで認証された。

  9 外部 GSS API プラグイン・セキュリティー機構を使用して認証された。

  10 外部 GSS API プラグイン・セキュリティー機構を使用して、または暗号化を伴ってサーバーで認証された。

  255 認証は指定されていない。
- SQLCA の SQLERRD(6) フィールドは、データベースがパーティション化されている場合に、接続されたパーティションのパーティション番号を戻します。区分化されていない場合、値 0 が戻されます。
- 正常に接続された許可 ID の長さが 8 バイトを超える場合には、 SQLCA の SQLWARN1 フィールドは 'A' に設定されます。これは、切り捨てが発生したことを示します。 SQLCA にある SQLWARN0 フィールドは、'W' に設定されて警告を示します。
- データベースでデータベース構成パラメーター DYN_QUERY_MGMT が使用可能になっている場合には、 SQLCA の SQLWARN7 フィールドは 'E' に設定されます。 SQLCA にある SQLWARN0 フィールドは、'W' に設定されて警告を示します。

接続が正常に実行されなかった場合

CONNECT TO ステートメントが正常に実行されなかった場合、

- SQLCA の SQLERRP フィールドは、エラーを検出したアプリケーション・リクエスターのモジュール名に設定されます。モジュール名の最初の 3 文字は、製品を識別します。
- アプリケーション・プロセスが接続可能状態でないために CONNECT TO ステートメントがエラーになった場合、アプリケーション・プロセスの接続状態は変更されません。
- *server-name* がローカル・ディレクトリーのリストにないために CONNECT TO ステートメントがエラーになった場合は、エラー・メッセージ (SQLSTATE 08001) が出され、アプリケーション・プロセスの接続状態は変更されません。
  - アプリケーション・リクエスターがアプリケーション・サーバーに接続されなかった場合、アプリケーション・プロセスは接続されないままです。
  - アプリケーション・リクエスターがアプリケーション・サーバーにすでに接続されていた場合、アプリケーション・プロセスはそのアプリケーション・サーバーに接続されたままです。それ以降のステートメントは、そのアプリケーション・サーバーで実行されます。
- その他の理由で CONNECT TO ステートメントがエラーになる場合、アプリケーション・プロセスは接続されていない状態になります。

IN SHARE MODE
データベースへの他の同時接続を可能にし、他のユーザーがデータベースに排他モードで接続しないようにします。

IN EXCLUSIVE MODE
排他ロックを保持するユーザーと同じ許可 ID を持つ場合を除き、複数の並行アプリケーション・プロセスがアプリケーション・サーバーで何らかの操作を実行するのを防止します。このオプションは DB2 Connect ではサポートされません。

ON SINGLE DBPARTITIONNUM
コーディネーターのデータベース・パーティションを排他モードで接続し、その他のすべてのデータベース・パーティションを共用モードで接続することを指定します。このオプションは、パーティション・データベースでのみ有効です。

RESET
アプリケーション・プロセスを現行サーバーから切断します。コミット操作が行われます。暗黙接続が使用可能な場合、アプリケーション・プロセスは SQL ステートメントが発行されるまで未接続のままになります。

USER *authorization-name/host-variable*
アプリケーション・サーバーに接続するユーザー ID を指定します。ホスト変数を指定する場合、それは、長さ属性が 8 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。 *host-variable* に入るユーザー ID は、左寄せする必要があり、引用符で区切ってはなりません。

USING *password/host-variable*
アプリケーション・サーバーに接続しようとするユーザー ID のパスワードを識別します。 *password* (パスワード) または *host-variable* (ホスト変数) には、

最大 18 文字を使用することができます。ホスト変数を指定する場合、それは、長さ属性が 18 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。

NEW *password/host-variable* **CONFIRM** *password*
USER オプションによって識別されるユーザー ID に割り当てられる新規パスワードを識別します。 *password* (パスワード) または *host-variable* (ホスト変数) には、最大 18 文字を使用することができます。ホスト変数を指定する場合、それは、長さ属性が 18 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。パスワードが変更されるシステムは、ユーザー認証がどのように設定されているかによって異なります。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
- アプリケーション・プロセスによって実行される最初の SQL ステートメントを CONNECT TO ステートメントにすることは望ましいことです。
- 異なるユーザー ID およびパスワードを用いて CONNECT TO ステートメントを現行のアプリケーション・サーバーに出すと、会話が割り振り解除され、割り振りし直されます。 すべてのカーソルは、データベース・マネージャーによってクローズされます (WITH HOLD オプションが使用された場合は、カーソル位置は失われます)。
- 同じユーザー ID およびパスワードを用いて、 CONNECT TO ステートメントが現行のアプリケーション・サーバーに出された場合、会話の割り振り解除および再割り振りは行われません。 この場合、カーソルはクローズされません。
- 複数パーティション・データベース環境を使用するには、ユーザーまたはアプリケーションは `db2nodes.cfg` ファイルにリストされたパーティションのいずれかに接続する必要があります。コーディネーターのパーティションと同じパーティションをすべてのユーザーが使用しないようにする必要があります。
- *authorization-name* として SYSTEM を CONNECT ステートメント内で明示的に指定することはできません。ただし、Windows オペレーティング・システムでは、ローカル・システム・アカウントの下で稼働しているローカル・アプリケーションは暗黙的にデータベースに接続することができ、その場合、ユーザー ID が SYSTEM になります。
- Windows Server に明示的に接続するときは、たとえば 'Domain\User' のような、Microsoft Windows NT Security Account Manager (SAM) 互換名を使用して、*authorization-name* またはユーザー *host-variable* を指定することができます。

例:

例 1: C プログラムで、データベース別名 TOROLAB、ユーザー ID FERMAT、およびパスワード THEOREM を使用して、アプリケーション・サーバー TOROLAB に接続します。

```
EXEC SQL  CONNECT TO TOROLAB USER FERMAT USING THEOREM;
```

例 2: C プログラムで、データベース別名がホスト変数 APP_SERVER (varchar(8)) に入っているアプリケーション・サーバーに接続します。正常に接続された後、アプリケーション・サーバーの 3 文字の製品 ID を、変数 PRODUCT (char(3)) にコピーします。

```
EXEC SQL  CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
  strncpy(PRODUCT,sqlca.sqlerrp,3);
```

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』
- *SQL リファレンス 第 1 巻* の『複数のパーティションにわたるデータ・パーティション』

関連サンプル:

- 『advsql.sqb -- How to read table data using CASE (MF COBOL)』
- 『dbmcon.sqc -- How to use multiple databases (C)』
- 『dbmcon.sqC -- How to use multiple databases (C++)』

CONNECT (タイプ 2)

CONNECT (タイプ 2) ステートメントは、指定したアプリケーション・サーバーにアプリケーション・プロセスを接続し、アプリケーション制御の分散作業単位の規則を確立します。このサーバーは、そのプロセスの現行サーバーになります。

CONNECT (タイプ 1) ステートメントのほとんどの性質は、 CONNECT (タイプ 2) ステートメントにも適用されます。この項では、それらを繰り返して説明するのではなく、タイプ 2 のエレメントのうちタイプ 1 とは異なる部分だけを説明します。

呼び出し:

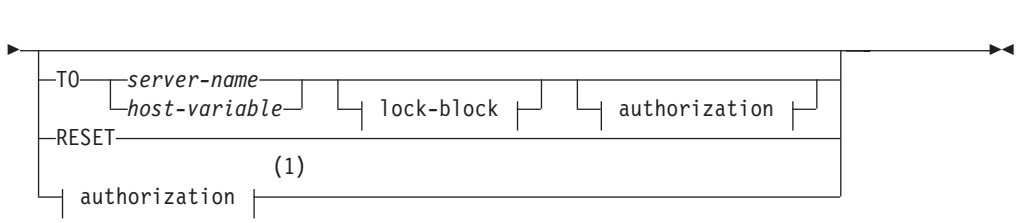
対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

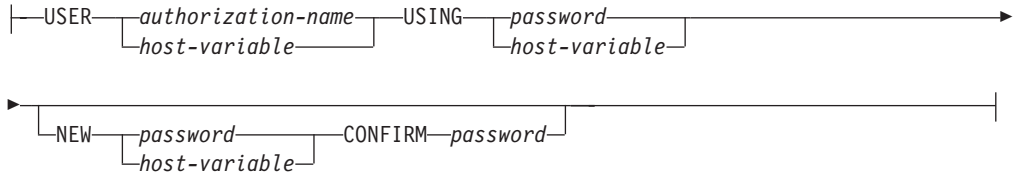
このステートメントの許可 ID には、指定されたアプリケーション・サーバーに接続するための許可が必要です。データベースの認証の設定値によっては、クライアントまたはサーバーのいずれかによって許可検査が行われる場合があります。パーティション・データベースの場合、ユーザーとグループの定義は、パーティションのすべてにわたって同一である必要があります。

構文:

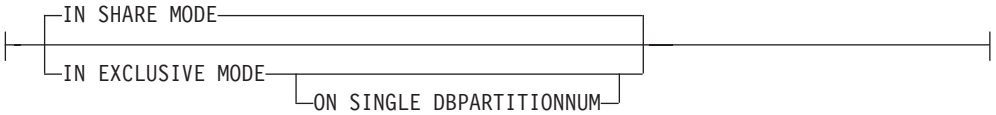
タイプ 1 とタイプ 2 のどちらを選択するかは、プリコンパイラー・オプションによって決められます。それらのオプションの概要については、『分散リレーショナル・データベース』を参照してください。



authorization:



lock-block:



注:

1 この形式は、暗黙の接続が使用可能な場合にのみ有効です。

説明:

TO *server-name/host-variable*
サーバーの名前のコーディング規則は、タイプ 1 と同じです。

SQLRULES(STD) オプションが有効な場合、*server-name* は、アプリケーション・プロセスの既存の接続を指定するものであってはなりません。そのように指定すると、エラー (SQLSTATE 08002) になります。

SQLRULES(DB2) オプションが有効で、 *server-name* がアプリケーション・プロセスの既存の接続を指定している場合、その接続が現行接続になり、古い接続は休止状態になります。つまり、この状況での CONNECT ステートメントの効果は、 SET CONNECTION ステートメントの効果と同じです。

SQLRULES の指定の詳細に関しては、『分散作業単位のセマンティクスを制御するオプション』を参照してください。

正常に接続された場合

CONNECT TO ステートメントが正常に実行された場合、

- アプリケーション・サーバーとの接続は作成されるか、または休止でない状態になり、現行接続かつ保留状態になります。
- CONNECT TO が現行サーバー以外のサーバーに出されると、現行の接続は休止状態になります。
- CURRENT SERVER 特殊レジスターと SQLCA は、CONNECT (タイプ 1) の
  と同じ方法で更新されます。

接続が正常に実行されなかった場合

CONNECT TO ステートメントが正常に実行されなかった場合、

- エラーの理由に関係なく、アプリケーション・プロセスの接続状態とその接続の状態は変更されません。
- 失敗したタイプ 1 の CONNECT の場合と同様に、SQLCA の SQLERRP フィールドは、エラーを検出したアプリケーション・リクエスターまたはサーバーのモジュール名に設定されます。

CONNECT (オペランドなし)、 **IN SHARE/EXCLUSIVE MODE**、**USER**、および **USING**
接続が存在する場合、タイプ 2 の動作はタイプ 1 と同様です。許可 ID とデータベース別名が、SQLCA の SQLERRMC フィールドに入れられます。接続が存在しない場合、暗黙接続の試みは行われず、 SQLERRP および SQLERRMC の各フィールドはブランクを戻します。 (アプリケーションでは、これらのフィールドを調べることによって、現行接続が存在しているか否かの検査を行うことができます。)

USER と USING を含むオペランドのない CONNECT は、DB2DBDFT 環境変数を使用することによって、アプリケーション・プロセスをデータベースに接続することができます。この方法は、タイプ 2 の CONNECT RESET に相当しますが、ユーザー ID とパスワードの使用が可能です。

RESET

デフォルトのデータベースが使用可能な場合、そのデータベースへの明示接続と同等です。デフォルトのデータベースが使用できない場合、アプリケーション・プロセスの接続状態とその接続の状態は変更されません。

デフォルトのデータベースが使用可能か否かは、インストール・オプション、環境変数、および認証設定値によって決まります。

規則:

- 『分散作業単位のセマンティクスを制御するオプション』で概略を説明するように、一連の接続オプションによって、接続管理のセマンティクスが制御されます。すべてのプリプロセス済みソース・ファイルには、デフォルト値が割り当てられます。 1 つのアプリケーションが、異なるさまざまな接続オプションでプリコンパイルされた複数のソース・ファイルで構成されている場合もあります。

  SET CLIENT コマンドまたは API を最初に実行しない限り、ランタイムに実行される最初の SQL ステートメントを含むソース・ファイルのプリプロセスに使用された接続オプションが、実際の接続オプションになります。

  ソース・ファイルの 1 つの CONNECT ステートメントが、異なる接続オプションで次々にプリプロセスされ、その合間に SET CLIENT コマンドまたは API を実行することなく実行されると、エラー (SQLSTATE 08001) が戻されます。
  SET CLIENT コマンドまたは API を実行すると、アプリケーション内のすべてのソース・ファイルをプリプロセスするために使用された接続オプションは無視されます。

  このステートメントの『例』セクションの例 1 では、こうした規則について説明されています。
- CONNECT TO ステートメントを使用して接続を確立したり切り替えたりすることができますが、 USER/USING 文節を使用した CONNECT TO は、指定したサーバーとの現行接続や休止接続がない場合にしか受け入れられません。
  USER/USING 文節によって同じサーバーとの接続を発行するには、その前に接続を解放する必要があります。そうでない場合、リジェクトされます (SQLSTATE 51022)。接続を解放するには、DISCONNECT ステートメントまたは RELEASE ステートメントを出し、次に COMMIT ステートメントを出します。

注:

- 暗黙接続は、タイプ 2 の接続を行うアプリケーションの最初の SQL ステートメントでサポートされます。 SQL ステートメントをデフォルトのデータベースに対して実行するには、まず CONNECT RESET ステートメントまたは CONNECT USER/USING ステートメントを使用して、接続を確立する必要があります。オペランドのない CONNECT ステートメントでは、現行接続があればそれに関する情報が表示されますが、現行接続がない場合にはデフォルトのデータベースには接続しません。

- *authorization-name* として SYSTEM を CONNECT ステートメント内で明示的に指定することはできません。ただし、Windows オペレーティング・システムでは、ローカル・システム・アカウントの下で稼働しているローカル・アプリケーションは暗黙的にデータベースに接続することができ、その場合、ユーザー ID が SYSTEM になります。
- Windows Server に明示的に接続するときは、たとえば 'Domain\User' のような、Microsoft Windows NT Security Account Manager (SAM) 互換名を使用して、*authorization-name* またはユーザー *host-variable* を指定することができます。

タイプ 1 とタイプ 2 の CONNECT ステートメントの比較

CONNECT ステートメントのセマンティクスは、CONNECT プリコンパイラー・オプションまたは SET CLIENT API によって決まります (『分散作業単位のセマンティクスを制御するオプション』を参照)。 CONNECT タイプ 1 または CONNECT タイプ 2 は指定することができ、それらのプログラムの CONNECT ステートメントは、それぞれタイプ 1 およびタイプ 2 の CONNECT ステートメントと呼ばれます。それらのセマンティクスについて、以下に説明します。

CONNECT TO の使用

タイプ 1	タイプ 2
各作業単位は、1 つのアプリケーション・サーバーに対してのみ接続を確立できます。	各作業単位は、複数のアプリケーション・サーバーとの接続を確立することができます。
他のアプリケーション・サーバーと接続するためには、その前に、現行の作業単位をコミットまたはロールバックする必要があります。	他のアプリケーション・サーバーと接続する前に、現行の作業単位をコミットまたはロールバックする必要はありません。
CONNECT ステートメントは、現行接続を確立します。後続の SQL 要求は、他の CONNECT によって変更されるまで、この接続に送られます。	最初の接続を確立する場合はタイプ 1 の CONNECT と同じです。休止接続に切り替える際に SQLRULES が STD に設定されている場合には、 SET CONNECTION ステートメントを使用する必要があります。
現行接続への接続が有効であり、現行接続を変更しません。	SQLRULES プリコンパイラー・オプションが DB2 に設定されている場合は、タイプ 1 の CONNECT と同じです。 SQLRULES が STD に設定されている場合、SET CONNECTION ステートメントを使用する必要があります。

タイプ 1	タイプ 2
他のアプリケーション・サーバーに接続すると、現行接続が切断されます。新しい接続が現行接続になります。 1 つの作業単位で維持される接続は 1 つだけです。	別のアプリケーション・サーバーに接続すると、現行接続は*休止状態* になります。新しい接続が現行接続になります。 1 つの作業単位で複数の接続を維持できます。 CONNECT が休止接続のアプリケーション・サーバーに対するものである場合、それが現行接続になります。 CONNECT を使用した休止接続への接続は、SQLRULES(DB2) が指定されている場合にのみ可能です。 SQLRULES(STD) が指定されている場合は、 SET CONNECTION ステートメントを使用する必要があります。
SET CONNECTION ステートメントはタイプ 1 の接続でサポートされていますが、有効な接続先は現行接続だけです。	タイプ 2 の接続では、接続状態を休止から現行に変更する SET CONNECTION ステートメントがサポートされています。

CONNECT...USER...USING の使用

タイプ 1	タイプ 2
USER...USING 文節を使用した接続では、現行接続が切断され、指定した許可名とパスワードで新しい接続が確立されます。	USER/USING 文節を使用した接続は、指定したその同じサーバーに対する現行接続も休止接続もない場合にのみ、受け入れられます。

暗黙の CONNECT、CONNECT RESET の使用、および切断

タイプ 1	タイプ 2
CONNECT RESET を使用して、現行接続を切断することができます。	CONNECT RESET は、デフォルトのアプリケーション・サーバーがシステムに定義されている場合、それに対する明示的接続に相当します。 接続は、COMMIT の正常実行時にアプリケーションによって切断できます。コミットの前には、RELEASE ステートメントを使用して、接続を解放ペンディングにします。このような接続はすべて、その次の COMMIT 時に切断されます。 あるいは RELEASE ステートメントの代わりに、プリコンパイラー・オプションの DISCONNECT(EXPLICIT)、DISCONNECT(CONDITIONAL)、DISCONNECT(AUTOMATIC)、または DISCONNECT の各ステートメントを使用することができます。

タイプ 1	タイプ 2
CONNECT RESET を使用して現行接続を切断した場合、その次の SQL ステートメントが CONNECT ステートメントでなく、デフォルトのアプリケーション・サーバーがシステムに定義されていれば、それに対する暗黙接続が行われます。	デフォルトのアプリケーション・サーバーがシステムに定義されている場合、CONNECT RESET はそれに対する明示接続に相当します。
連続して CONNECT RESET を出すと、エラーになります。	連続する CONNECT RESET を発行してエラーになるのは、 SQLRULES(STD) が指定されている場合だけです。このオプションを指定すると既存の接続に対して CONNECT を使用できなくなります。
CONNECT RESET では、現行の作業単位のコミットも暗黙のうちに行われます。	CONNECT RESET では、現行の作業単位はコミットされません。
何らかの理由で既存の接続がシステムによって切断された場合、このデータベースに対して CONNECT 以外の SQL ステートメントを連続して出すと、 08003 という SQLSTATE を受け取ることになります。	既存の接続がシステムによって切断された場合でも、COMMIT、ROLLBACK、および SET CONNECTION の各ステートメントを使用することができます。
アプリケーション・プロセスが正常に終了した時点で、作業単位が暗黙のうちにコミットされます。	タイプ 1 と同じ。
アプリケーション・プロセスが終了すると、すべての接続 (1 つだけ) が切断されます。	アプリケーション・プロセスが終了すると、すべての接続 (現行、休止、および解放ペンディングのもの) が切断されます。

CONNECT のエラー

タイプ 1	タイプ 2
ローカル・ディレクトリーにサーバー名が定義されていないというエラー以外で CONNECT がエラーになった場合、現行接続があるかどうかに関係なく、アプリケーション・プロセスは未接続状態になります。それ以降の CONNECT 以外のステートメントは、08003 の SQLSTATE を受け取ることになります。	CONNECT がエラーになったときに現行接続があっても、その現行接続は影響を受けません。 CONNECT がエラーになったときに、現行接続がなかった場合、プログラムは未接続状態になります。それ以降の CONNECT 以外のステートメントは、08003 の SQLSTATE を受け取ることになります。

例:

例 1:

この例では、複数のソース・プログラム (枠の中に示される) の使用法を示します。いくつかは異なる接続オプション (コードの上に示される) を指定してプリプロセスされ、そのうち 1 つは SET CLIENT API 呼び出しを含んでいます。

PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tbl1;
...
```

PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tbl2;
...
```

PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2  SQLRULES DB2  DISCONNECT EXPLICIT [1]
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO :hv1
FROM tbl3;
...
```

1 注: SET CLIENT API の実際の構文ではありません。

PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO :hv1
FROM tbl4;
...
```

アプリケーションが PGM1 に続いて PGM2 を実行すると、次のようになります。

- OTTAWA への接続が実行されます。 connect=2、 sqlrules=DB2、 disconnect=CONDITIONAL
- QUEBEC への接続はエラー (SQLSTATE 08001) になります。これは、 SQLRULES と DISCONNECT が共に異なっているためです。

アプリケーションが PGM1 に続いて PGM3 を実行すると、次のようになります。

- OTTAWA への接続が実行されます。 connect=2、 sqlrules=DB2、 disconnect=CONDITIONAL
- LONDON への接続が実行されます。 connect=2、 sqlrules=DB2、 disconnect=EXPLICIT

2 番目の CONNECT ステートメントの前に SET CLIENT API が実行されるため、問題ありません。

アプリケーションが PGM1 に続いて PGM4 を実行すると、次のようになります。

- OTTAWA への接続が実行されます。 connect=2、 sqlrules=DB2、 disconnect=CONDITIONAL
- REGINA への接続が実行されます。 connect=2 sqlrules=DB2、 disconnect=CONDITIONAL

これは、PGM1 のプリプロセッサー・オプションが PGM4 と同じなので、問題ありません。

例 2:

この例では、CONNECT (タイプ 2)、SET CONNECTION、RELEASE、および DISCONNECT の各ステートメントの相互関係を示します。 S0、S1、S2、および S3 は 4 つのサーバーを示します。

順序	ステートメント	現行サーバー	休止接続	解放ペンディング
0	ステートメントはない	なし	なし	なし
1	SELECT * FROM TBLA	S0 (デフォルト値)	なし	なし
2	CONNECT TO S1	S1	S0	なし
	SELECT * FROM TBLB	S1	S0	なし
3	CONNECT TO S2	S2	S0、S1	なし
	UPDATE TBLC SET ...	S2	S0、S1	なし
4	CONNECT TO S3	S3	S0、S1、S2	なし
	SELECT * FROM TBLD	S3	S0、S1、S2	なし
5	SET CONNECTION S2	S2	S0、S1、S3	なし
6	RELEASE S3	S2	S0、S1	S3
7	COMMIT	S2	S0、S1	なし
8	SELECT * FROM TBLE	S2	S0、S1	なし
9	DISCONNECT S1	S2	S0	なし
	SELECT * FROM TBLF	S2	S0	なし

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』

関連資料:

- 149 ページの『CONNECT (タイプ 1)』

関連サンプル:

- 『dbmcon.sqc -- How to use multiple databases (C)』
- 『dbmcon.sqC -- How to use multiple databases (C++)』

CREATE ALIAS

CREATE ALIAS ステートメントは、表、ビュー、ニックネーム、または他の別名に対する別名を定義します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

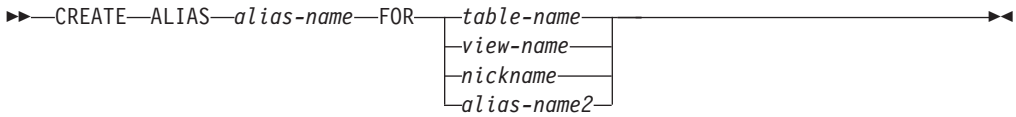
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (別名の暗黙または明示のスキーマ名が存在しない場合)
- スキーマに対する CREATEIN 特権 (別名のスキーマ名が既存のスキーマを指している場合)

別名によって参照されるオブジェクトを使用するには、そのオブジェクトに対して、オブジェクトそのものを使用する場合に必要な特権と同じ特権が必要です。

構文:



説明:

alias-name
別名を指定します。この名前が、現行データベースに存在する表、ビュー、ニックネーム、または別名を指定していてはなりません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まるスキーマ名は使用できません (SQLSTATE 42939)。

別名を定義する際の規則は、表名の定義に使用される規則と同じです。

FOR *table-name*、*view-name*、*nickname*、 または *alias-name2*
alias-name を定義する対象の表名、ビュー名、ニックネーム、または別名を指定します。他の別名 *(alias-name2)* を指定する場合、その別名は、定義される新しい *alias-name* (完全修飾形式の) と同じであってはなりません。 *table-name* を宣言済み一時表にすることはできません (SQLSTATE 42995)。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - ALIAS の代わりに SYNONYM を指定できます。

- 新しく作成した別名の定義は、SYSCAT.TABLES に保管されます。
- 別名は、定義時に存在していないオブジェクトに対しても定義できます。オブジェクトが存在しない場合、警告が出されます (SQLSTATE 01522)。ただし、参照されるオブジェクトは、その別名を含む SQL ステートメントのコンパイル時には存在していなければなりません。そうでない場合、エラーになります (SQLSTATE 52004)。
- 他の別名を参照する別名を別名チェーンの一部として定義することは可能ですが、 SQL ステートメントで使用する場合には、単一の別名と同じ制約がそのチェーンにも適用されます。別名チェーンは、単一の別名と同じ方法で解決されます。ビュー定義、パッケージ内のステートメント、または別名チェーンを指すトリガーで別名が使用された場合、そのビュー、パッケージ、またはチェーンの各別名についてのトリガーに従属関係が記録されます。別名チェーンの中に反復サイクルがあってはならず、それは別名定義時に検出されます。
- まだ存在していないスキーマ名を用いて別名を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。

例:

例 1: HEDGES は表 T1 に対して別名を作成します (どちらも修飾なし)。

```
CREATE ALIAS A1 FOR T1
```

HEDGES.T1 に対して別名 HEDGES.A1 が作成されます。

例 2: HEDGES は表に対して別名を作成します (どちらも修飾付き)。

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
```

MCKNIGHT.A1 に対して別名 HEDGES.T1 が作成されます。

例 3: HEDGES は表に対して別名を作成します (異なるスキーマ内の別名。HEDGES は DBADM ではなく、HEDGES はスキーマ MCKNIGHT に対して CREATEIN を持っていない)。

```
CREATE ALIAS  MCKNIGHT.A1 FOR MCKNIGHT.T1
```

この例はエラーになります (SQLSTATE 42501)。

例 4: HEDGES は未定義の表に対して別名を作成します (どちらも修飾付き。FUZZY.WUZZY は存在しない)。

```
CREATE ALIAS HEDGES.A1 FOR FUZZY.WUZZY
```

このステートメントは成功しますが、警告 (SQLSTATE 01522) が出されます。

例 5: HEDGES は別名に対して別名を作成します (どちらも修飾付き)。

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
CREATE ALIAS HEDGES.A2 FOR HEDGES.A1
```

最初のステートメントは成功します (例 2 と同じ)。

2 番目のステートメントも成功して、別名チェーンが作成されます。つまり、HEDGES.A2 が HEDGES.A1 を参照し、その HEDGES.A1 が MCKNIGHT.T1 を参照することになります。 HEDGES に MCKNIGHT.T1 に対する特権があるかどうかは関係ありません。別名は、表の特権に関係なく作成されます。

例 6: ニックネーム FUZZYBEAR の別名として、A1 を指定します。

```
CREATE ALIAS A1 FOR FUZZYBEAR
```

例 7: ある大規模な組織に、 D108 という会計部門と D577 という人事部門があります。 D108 は、DB2 RDBMS に存在する表に、ある情報を保持しています。 D577 は、Oracle RDBMS に存在する表に、いくつかのレコードを保持しています。 DBA は、この 2 つの RDBMS をフェデレーテッド・システム内のデータ・ソースとして定義し、それぞれの表に DEPTD108 および DEPTD577 というニックネームを付けます。フェデレーテッド・システムのユーザーはこれらの表の結合を作成する必要がありますが、英数字のニックネームではなく、もっと意味のある名前で参照できるようにしたいことがあります。それで、DEPTD108 の別名として FINANCE を定義し、 DEPTD577 の別名として PERSONNEL を定義します。

```
CREATE ALIAS FINANCE FOR DEPTD108
CREATE ALIAS PERSONNEL FOR DEPTD577
```

CREATE BUFFERPOOL

CREATE BUFFERPOOL ステートメントは、データベース・マネージャーにより使用される新しいバッファー・プールを作成します。

パーティション・データベースでは、特定のパーティションを上書きする可能性のあるデフォルトのバッファー・プール定義が、それぞれのパーティションに対して指定されます。またパーティション・データベースでは、データベース・パーティション・グループを指定しない限り、すべてのパーティションにバッファー・プールが定義されます。データベース・パーティション・グループを指定すると、そのデータベース・パーティション・グループのパーティションにだけバッファー・プールが作成されます。

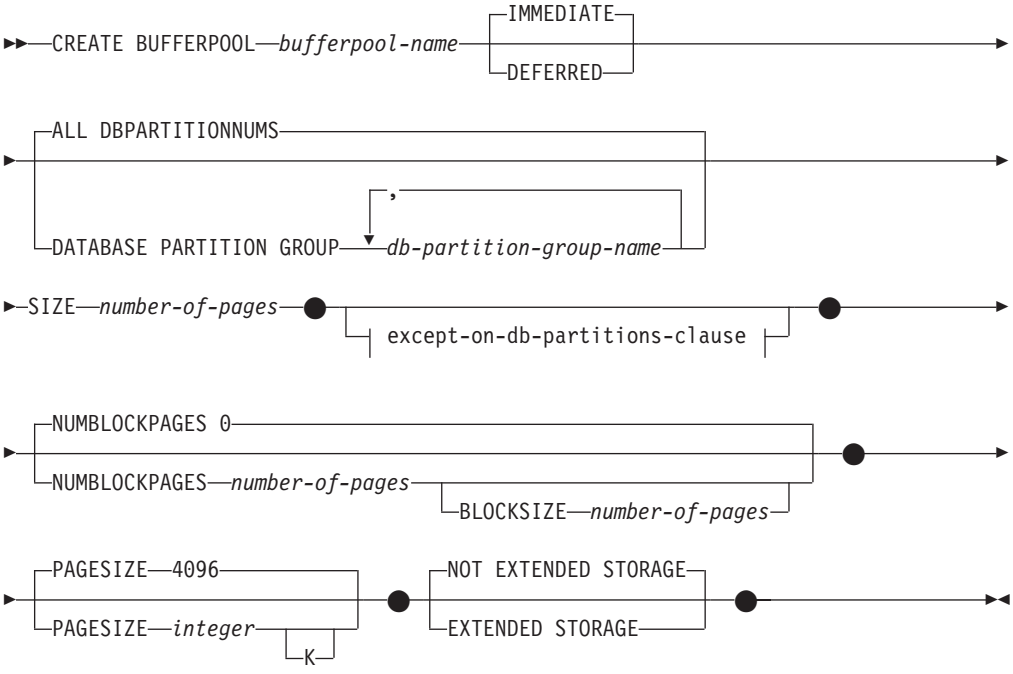
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

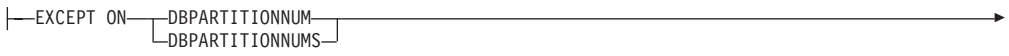
許可:

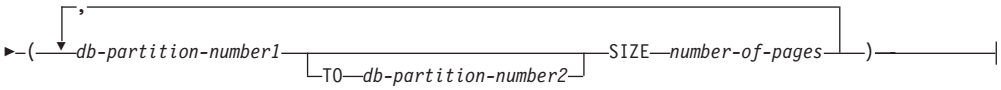
このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文:



except-on-db-partitions-clause:





説明:

bufferpool-name
バッファー・プールの名前を指定します。これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。 *bufferpool-name* には、すでにカタログに存在するバッファー・プールを指定していてはなりません (SQLSTATE 42710)。 *bufferpool-name* を文字 'SYS' および 'IBM' で始めることはできません (SQLSTATE 42939)。

IMMEDIATE
バッファー・プールは直ちに作成されます。メモリーを共用するデータベース内に新規バッファー・プールを割り振るのに十分な予約済みのスペースがない場合、警告 (SQLSTATE 01657) が出され、ステートメントは DEFERRED で実行されます。

DEFERRED
データベースが非活動状態になると、バッファー・プールが作成されます (すべてのアプリケーションがデータベースから切断される必要があります)。予約済みのメモリー・スペースは必要ありません。 DB2 が、システムから必要なメモリーを割り振ります。

ALL DBPARTITIONNUMS
このバッファー・プールは、データベースのすべてのパーティションに作成されます。

DATABASE PARTITION GROUP *db-partition-group-name, ...*
バッファー・プール定義を適用するデータベース・パーティション・グループ (単一または複数) を指定します。これを指定すると、バッファー・プールはこれらのデータベース・パーティション・グループのパーティションにだけ作成されます。それぞれのデータベース・パーティション・グループは、現在データベースに存在している必要があります (SQLSTATE 42704)。 DATABASE PARTITION GROUP キーワードを指定しないと、このバッファー・プールはすべてのパーティション (およびその後データベースに追加されるパーティション) に作成されます。

SIZE *number-of-pages*
バッファー・プールのサイズをページ数で指定します。パーティション・データベースでは、このサイズはバッファー・プールが存在するすべてのパーティションのデフォルトのサイズになります。

NUMBLOCKPAGES *number-of-pages*
ブロック・ベース域に存在していなければならないページ数を指定します。ページ数は、バッファー・プールのページ数の 98% より小さくしなければなりません (SQLSTATE 54052)。値 0 を指定すると、ブロック入出力は不可になります。使用されている NUMBLOCKPAGES の実際の値は、BLOCKSIZE の倍数になります。

BLOCKSIZE *number-of-pages*
ブロック内のページ数を指定します。ブロック・サイズの値は、2 ～ 256 でなければなりません (SQLSTATE 54053)。デフォルト値は 32 です。

except-on-db-partitions-clause
そのバッファー・プールのサイズをデフォルトのサイズとは異なるサイズにしたいパーティションを指定します。 この文節の指定がない場合、すべてのパーティションが、このバッファー・プールに対して指定したのと同じプール・サイズを持つことになります。

EXCEPT ON DBPARTITIONNUMS
特定のパーティションを指定することを示すキーワードです。 DBPARTITIONNUM は DBPARTITIONNUMS の同義語です。

db-partition-number1
バッファー・プールが作成されるパーティションに含まれる特定のパーティション番号を指定します。

TO *db-partition-number2*
パーティション番号の範囲を指定します。 *db-partition-number2* の値は、 *db-partition-number1* の値よりも大きいか等しい値でなければなりません (SQLSTATE 428A9)。指定するパーティション番号の範囲 (指定するパーティション番号を含む) のすべてのパーティションは、バッファー・プールを作成するパーティションに含まれていなければなりません (SQLSTATE 42729)。

SIZE *number-of-pages*
バッファー・プールのサイズをページ数で指定します。

PAGESIZE *integer* **[K]**
バッファー・プールに使用されるページのサイズを定義します。接尾部 K を持たない *integer* の有効値は、 4 096、8 192、16 384、または 32 768 です。接尾部 K を持つ *integer* の有効値は、4、8、16、または 32 です。ページ・サイズがこれらのいずれかの値でない場合は、エラーが生じます (SQLSTATE 428DE)。 デフォルト値は 4 096 バイト (4K) ページです。 *integer* と K の間には、任意の数のスペースを使用できます (スペースなしでも可)。

EXTENDED STORAGE
拡張ストレージが使用可能な場合、このバッファー・プールから排除されるページは、拡張ストレージにキャッシュされます。 (拡張ストレージは、データベース構成パラメーター NUM_ESTORE_SEGS と ESTORE_SEG_SIZE をゼロ以外の値に設定することによってオンになります。)

NOT EXTENDED STORAGE
拡張ストレージが使用可能な場合でも、このバッファー・プールから排除されるページは拡張ストレージにはキャッシュされません。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
    - DBPARTITIONNUMS の代わりに NODES を指定できます。
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
- DEFERRED オプションを使用してバッファー・プールが作成されると、このバッファー・プール内に作成される任意の表スペースは、データベースが次に活動状

態になるときまで、同じページ・サイズの小さいシステム・バッファー・プールを使用します。バッファー・プールを再アクティブ化して、新たなバッファー・プールに対する表スペースの割り当てを有効にするには、データベースを再始動する必要があります。デフォルト・オプションは IMMEDIATE です。

- バッファー・プールは、拡張ストレージおよびブロック・ベース入出力の両方を使用して作成はできません。
- すべてのバッファー・プールの合計と、その他のデータベース・マネージャーやアプリケーションの要件に合うように、マシンに十分な実メモリーが必要です。DB2 が正規バッファー・プールで必要な合計のメモリーを入手できない場合には、各ページ・サイズ (4K、 8K、16K および 32K) に対して小さいシステム・バッファー・プールの始動を試みます。この場合、ユーザーに警告が出され (SQLSTATE 01626)、すべての表スペースからのページはシステム・バッファー・プールを使用します。

関連資料:

- *管理ガイド: パフォーマンス* の『database_memory - 「データベース共有メモリー・サイズ」構成パラメーター』

関連サンプル:

- 『tscreate.sqc -- How to create and drop buffer pools and table spaces (C)』
- 『tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)』

CREATE DATABASE PARTITION GROUP

CREATE DATABASE PARTITION GROUP ステートメントは、データベースに新しいデータベース・パーティション・グループを作成し、パーティションをデータベース・パーティション・グループに割り当て、データベース・パーティション・グループ定義をカタログに記録します。

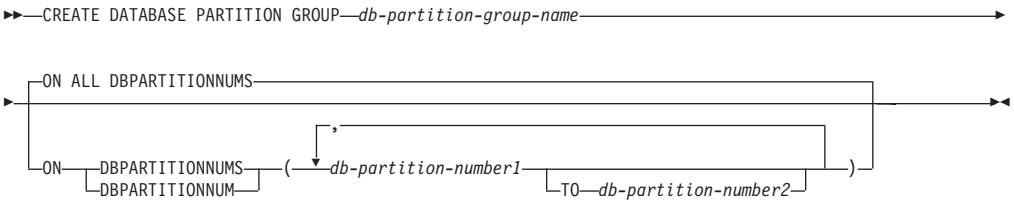
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文:



説明:

db-partition-group-name
データベース・パーティション・グループの名前を指定します。 これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。 *db-partition-group-name* は、すでにカタログに存在するデータベース・パーティション・グループを指定するものであってはなりません (SQLSTATE 42710)。 *db-partition-group-name* を文字 'SYS' または 'IBM' で始めることはできません (SQLSTATE 42939)。

ON ALL DBPARTITIONNUMS
データベース・パーティション・グループの作成時に、データベース (`db2nodes.cfg` ファイル) に定義されているすべてのパーティションにわたってデータベース・パーティション・グループを定義することを指定します。

データベース・システムにパーティションが追加された場合、 ALTER DATABASE PARTITION GROUP ステートメントを使用して、この新しいパーティションをデータベース・パーティション・グループ (IBMDEFAULTGROUP を含む) に組み込む必要があります。さらに、REDISTRIBUTE DATABASE PARTITION GROUP コマンドを使用して、そのパーティションにデータを移す必要があります。

ON DBPARTITIONNUMS
データベース・パーティション・グループに入れる特定のパーティションを指定します。 DBPARTITIONNUM は DBPARTITIONNUMS の同義語です。

db-partition-number1
特定のパーティション番号を指定します。 (前のバージョンとの互換性を保つため、形式 NODE*nnnnn* の *node-name* も指定できます。)

TO *db-partition-number2*
パーティション番号の範囲を指定します。 *db-partition-number2* の値は、*db-partition-number1* の値よりも大きいか等しい値でなければなりません (SQLSTATE 428A9)。指定したパーティション番号の範囲 (指定した番号を含む) のすべてのパーティションが、データベース・パーティション・グループに入れられます。

規則:

- 番号によって指定するそれぞれのパーティションは、 db2nodes.cfg ファイルに定義されていなければなりません (SQLSTATE 42729)。
- ON DBPARTITIONNUMS 文節にリストするそれぞれの *db-partition-number* は、同じであってはなりません (SQLSTATE 42728)。
- 有効な *db-partition-number* は、 0 〜 999 (両端を含む) です (SQLSTATE 42729)。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
    - DBPARTITIONNUMS の代わりに NODES を指定できます。
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
- このステートメントは、データベース・パーティション・グループに対するパーティション・マップを作成します。それぞれのパーティション・マップごとに、パーティション・マップ ID (PMAP_ID) が生成されます。この情報はカタログに記録され、SYSCAT.DBPARTITIONGROUPS と SYSCAT.PARTITIONMAPS から検索することができます。パーティション・マップのそれぞれの項目は、ハッシュされた行が常駐するターゲット・パーティションを指定します。単一パーティションのデータベース・パーティション・グループの場合、対応するパーティション・マップの項目は 1 つだけです。複数パーティションのデータベース・パーティション・グループの場合、対応するパーティション・マップには 4 096 の項目があり、パーティション番号がマップ項目にラウンドロビン方式 (デフォルト) で割り当てられます。

例:

0、1、2、5、7、および 8 として定義された 6 つのパーティションを持つパーティション・データベースがあると想定します。

- 6 つのパーティションすべてに対して、 MAXGROUP という名前のデータベース・パーティション・グループを作成すると想定します。必要なステートメントは以下のようになります。

```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```

- パーティション 0、1、2、5、および 8 に対して、 MEDGROUP と呼ばれるデータベース・パーティション・グループを作成すると想定します。必要なステートメントは以下のようになります。

```
CREATE DATABASE PARTITION GROUP MEDGROUP
  ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```

- パーティション 7 に対して、単一パーティションのデータベース・パーティション・グループ MINGROUP を作成すると想定します。必要なステートメントは以下のようになります。

```
CREATE DATABASE PARTITION GROUP MINGROUP
  ON DBPARTITIONNUM (7)
```

関連概念:

- *SQL リファレンス 第 1 巻* の『複数のパーティションにわたるデータ・パーティション』

CREATE DISTINCT TYPE

CREATE DISTINCT TYPE ステートメントは、特殊 (distinct) タイプを定義します。特殊タイプは、常に組み込みデータ・タイプのいずれかに基づいています。このステートメントの正常な実行により、該当の特殊タイプとそのソース・タイプとの間をキャストする関数も生成され、また必要に応じてその特殊タイプで使用する比較演算子 (=、<>、<、<=、>、および >=) に対するサポートが生成されます。

呼び出し:

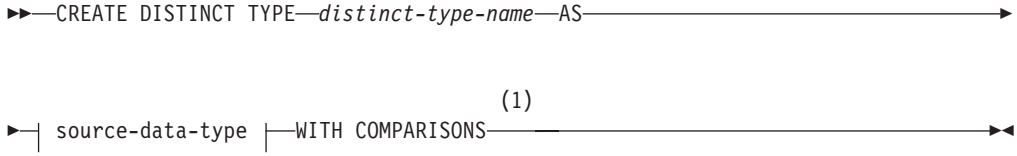
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

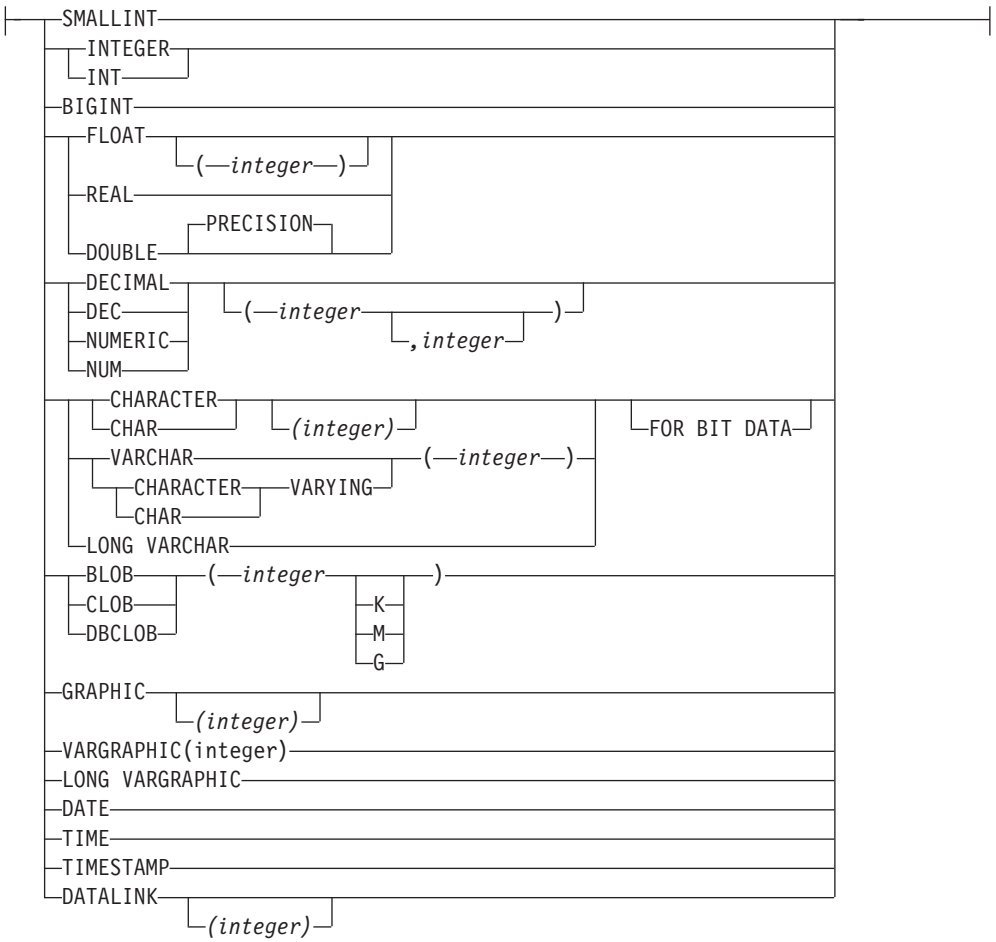
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (特殊タイプのスキーマ名が既存のスキーマを指していない場合)。
- スキーマに対する CREATEIN 特権 (特殊タイプのスキーマ名が既存のスキーマを指している場合)。

構文:



source-data-type:



注:

1 すべてのソース・データ・タイプに必須。ただし LOB、LONG VARCHAR、および LONG VARGRAPHIC はサポートされません。

説明:

distinct-type-name

特殊タイプの名前を指定します。暗黙または明示の修飾子を含む名前は、カタログに記述されている特殊タイプを指定するものであってはなりません。非修飾名は、source-data-type または BOOLEAN と同一のものであってはなりません (SQLSTATE 42918)。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

スキーマ名 (明示指定または暗黙指定) は、8 バイト以下でなければなりません (SQLSTATE 42622)。

述部でキーワードとして使用されるいくつかの名前は、システム使用として予約されており、 *distinct-type-name* として使用することはできません。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、

LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。この規則に違反すると、エラーになります (SQLSTATE 42939)。

2 つの部分からなる *distinct-type-name* を指定する場合、スキーマ名を 'SYS' で始めることはできません。違反すると、エラー (SQLSTATE 42939) になります。

source-data-type
特殊タイプの内部表示のベースとして使用されるデータ・タイプを指定します。

WITH COMPARISONS
特殊タイプの 2 つのインスタンスを比較するシステム生成の比較演算子を作成することを指定します。ソース・データ・タイプが BLOB、 CLOB、 DBCLOB、 LONG VARCHAR、 LONG VARGRAPHIC、または DATALINK の場合、これらのキーワードは指定できません。指定した場合には、警告 (SQLSTATE 01596) が出され、比較演算子は生成されません。それ以外のソース・データ・タイプの場合、WITH COMPARISONS キーワードは必須です。

注:

- ***特権***

  ユーザー定義タイプの定義者は、特殊タイプに関して自動的に生成されるすべての関数で、 EXECUTE 特権 WITH GRANT OPTION を必ず与えられます。

  CREATE DISTINCT TYPE の間に自動的に生成されるすべての関数での EXECUTE 特権は、 PUBLIC に与えられます。
- まだ存在していないスキーマ名を用いて特殊タイプを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- ソース・タイプとの間のキャストに必要な次の関数が生成されます。
  - 特殊タイプをソース・タイプに変換する関数
  - ソース・タイプを特殊タイプに変換する関数
  - ソース・タイプが SMALLINT の場合、INTEGER から特殊タイプに変換する関数
  - ソース・タイプが CHAR の場合、VARCHAR から特殊タイプに変換する関数
  - ソース・タイプが GRAPHIC の場合、VARGRAPHIC から特殊タイプに変換する関数

一般に、これらの関数の形式は次のようになります。

```
CREATE FUNCTION source-type-name (distinct-type-name)
  RETURNS source-type-name ...

CREATE FUNCTION distinct-type-name (source-type-name)
  RETURNS distinct-type-name ...
```

ソース・タイプがパラメーター化タイプである場合、特殊タイプをソース・タイプに変換する関数の関数名は、パラメーターなしのソース・タイプの名前になります (詳細については、 178 ページの表 3 を参照)。この関数の戻り値のタイプには、CREATE DISTINCT TYPE ステートメントに指定されたパラメーターが含められます。ソー

ス・タイプを特殊タイプに変換するための関数の入力パラメーターは、そのパラメーターを含むソース・タイプになります。たとえば、

```
CREATE DISTINCT TYPE T_SHOESIZE AS CHAR(2)
  WITH COMPARISONS

CREATE DISTINCT TYPE T_MILES AS DOUBLE
  WITH COMPARISONS
```

上記の指定により、次の関数が生成されます。

```
FUNCTION CHAR (T_SHOESIZE) RETURNS CHAR (2)

FUNCTION T_SHOESIZE (CHAR (2))
RETURNS T_SHOESIZE

FUNCTION DOUBLE (T_MILES) RETURNS DOUBLE

FUNCTION T_MILES (DOUBLE) RETURNS T_MILES
```

生成された cast 関数のスキーマは、特殊タイプのスキーマと同じです。この名前と同じ名前でシグニチャーも同じ他の関数が、データベースにすでに存在していてはなりません (SQLSTATE 42710)。

次の表は、事前定義されているすべてのデータ・タイプについて、特殊タイプをソース・タイプに変換する関数、およびソース・タイプを特殊タイプに変換する関数の名前を示しています。

表 3. 特殊タイプに対する CAST 関数

ソース・タイプ名	関数名	パラメーター	戻りタイプ
CHAR	*distinct-type-name*	CHAR (*n*)	*distinct-type-name*
	CHAR	*distinct-type-name*	CHAR (*n*)
	distinct-type-name	VARCHAR (*n*)	*distinct-type-name*
VARCHAR	*distinct-type-name*	VARCHAR (*n*)	*distinct-type-name*
	VARCHAR	*distinct-type-name*	VARCHAR (*n*)
LONG VARCHAR	*distinct-type-name*	LONG VARCHAR	*distinct-type-name*
	LONG_VARCHAR	*distinct-type-name*	LONG VARCHAR
CLOB	*distinct-type-name*	CLOB (*n*)	*distinct-type-name*
	CLOB	*distinct-type-name*	CLOB (*n*)
BLOB	*distinct-type-name*	BLOB (*n*)	*distinct-type-name*
	BLOB	*distinct-type-name*	BLOB (*n*)
GRAPHIC	*distinct-type-name*	GRAPHIC (*n*)	*distinct-type-name*
	GRAPHIC	*distinct-type-name*	GRAPHIC (*n*)
	distinct-type-name	VARGRAPHIC (*n*)	*distinct-type-name*
VARGRAPHIC	*distinct-type-name*	VARGRAPHIC (*n*)	*distinct-type-name*
	VARGRAPHIC	*distinct-type-name*	VARGRAPHIC (*n*)
LONG VARGRAPHIC	*distinct-type-name*	LONG VARGRAPHIC	*distinct-type-name*
	LONG_VARGRAPHIC	*distinct-type-name*	LONG VARGRAPHIC
DBCLOB	*distinct-type-name*	DBCLOB (*n*)	*distinct-type-name*
	DBCLOB	*distinct-type-name*	DBCLOB (*n*)

表 3. 特殊タイプに対する CAST 関数 (続き)

ソース・タイプ名	関数名	パラメーター	戻りタイプ
SMALLINT	*distinct-type-name*	SMALLINT	*distinct-type-name*
	distinct-type-name	INTEGER	*distinct-type-name*
	SMALLINT	*distinct-type-name*	SMALLINT
INTEGER	*distinct-type-name*	INTEGER	*distinct-type-name*
	INTEGER	*distinct-type-name*	INTEGER
BIGINT	*distinct-type-name*	BIGINT	*distinct-type-name*
	BIGINT	*distinct-type-name*	BIGINT
DECIMAL	*distinct-type-name*	DECIMAL (*p,s*)	*distinct-type-name*
	DECIMAL	*distinct-type-name*	DECIMAL (*p,s*)
NUMERIC	*distinct-type-name*	DECIMAL (*p,s*)	*distinct-type-name*
	DECIMAL	*distinct-type-name*	DECIMAL (*p,s*)
REAL	*distinct-type-name*	REAL	*distinct-type-name*
	distinct-type-name	DOUBLE	*distinct-type-name*
	REAL	*distinct-type-name*	REAL
FLOAT(*n*) ただし *n*<=24	*distinct-type-name*	REAL	*distinct-type-name*
	distinct-type-name	DOUBLE	*distinct-type-name*
	REAL	*distinct-type-name*	REAL
FLOAT(*n*) ただし *n*>24	*distinct-type-name*	DOUBLE	*distinct-type-name*
	DOUBLE	*distinct-type-name*	DOUBLE
FLOAT	*distinct-type-name*	DOUBLE	*distinct-type-name*
	DOUBLE	*distinct-type-name*	DOUBLE
DOUBLE	*distinct-type-name*	DOUBLE	*distinct-type-name*
	DOUBLE	*distinct-type-name*	DOUBLE
DOUBLE PRECISION	*distinct-type-name*	DOUBLE	*distinct-type-name*
	DOUBLE	*distinct-type-name*	DOUBLE
DATE	*distinct-type-name*	DATE	*distinct-type-name*
	DATE	*distinct-type-name*	DATE
TIME	*distinct-type-name*	TIME	*distinct-type-name*
	TIME	*distinct-type-name*	TIME
TIMESTAMP	*distinct-type-name*	TIMESTAMP	*distinct-type-name*
	TIMESTAMP	*distinct-type-name*	TIMESTAMP
DATALINK	*distinct-type-name*	DATALINK	*distinct-type-name*
	DATALINK	*distinct-type-name*	DATALINK

注: NUMERIC および FLOAT は、移植可能アプリケーションのユーザー定義タイプを作成する場合にはお勧めできません。代わりに DECIMAL および DOUBLE を使用してください。

上記の表には、特殊タイプが定義されている場合に自動的に生成される関数だけを示しています。したがって、CREATE FUNCTION ステートメントを使用して、特殊タイプに対応するユーザー定義関数を登録し、それらのユーザー定義関数を適切な組み込み関数に基づくものにしてからでなければ、どの組み込み関数

(AVG、MAX、LENGTH など) も、特殊タイプに対してサポートされません。特に、組み込み列関数に基づくユーザー定義関数を登録することが可能である点に注意してください。

WITH COMPARISONS 文節を使用して特殊タイプが作成された場合、システム生成の比較演算子が作成されます。これらの比較演算子の作成により、SYSCAT.ROUTINES カタログ・ビューに新しい関数としての項目が生成されます。

これらの演算子や cast 関数を SQL ステートメントで正しく使用するには、 SQL パスに特殊タイプのスキーマ名が含まれていなければなりません。または FUNCPATH BIND オプションを参照してください。

例:

例 1: INTEGER データ・タイプに基づく、SHOESIZE という名前の特殊タイプを作成します。

```
CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS
```

またこの結果、比較演算子 (=、<>、<、<=、>、>=)、 INTEGER を戻す cast 関数 INTEGER(SHOESIZE)、および SHOESIZE を戻す cast 関数 SHOESIZE(INTEGER) が作成されます。

例 2: DOUBLE データ・タイプに基づく、MILES という名前の特殊タイプを作成します。

```
CREATE DISTINCT TYPE MILES AS DOUBLE WITH COMPARISONS
```

またこの結果、比較演算子 (=、<>、<、=、>、>=)、 DOUBLE を戻す cast 関数 DOUBLE(MILES)、および MILES を戻す cast 関数 MILES(DOUBLE) が作成されます。

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 199 ページの『CREATE FUNCTION』
- 347 ページの『CREATE TABLE』
- 762 ページの『SET PATH』
- *SQL リファレンス 第 1 巻* の『ユーザー定義タイプ』

関連サンプル:

- 『dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)』
- 『udfcli.sqc -- Call a variety of types of user-defined functions (C)』
- 『dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)』
- 『udfcli.sqC -- Call a variety of types of user-defined functions (C++)』
- 『DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)』
- 『DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)』

CREATE EVENT MONITOR

CREATE EVENT MONITOR ステートメントは、データベースの使用中に発生する特定のイベントを記録するモニターを定義します。各イベント・モニターの定義には、データベースがイベントを記録するロケーションも指定します。

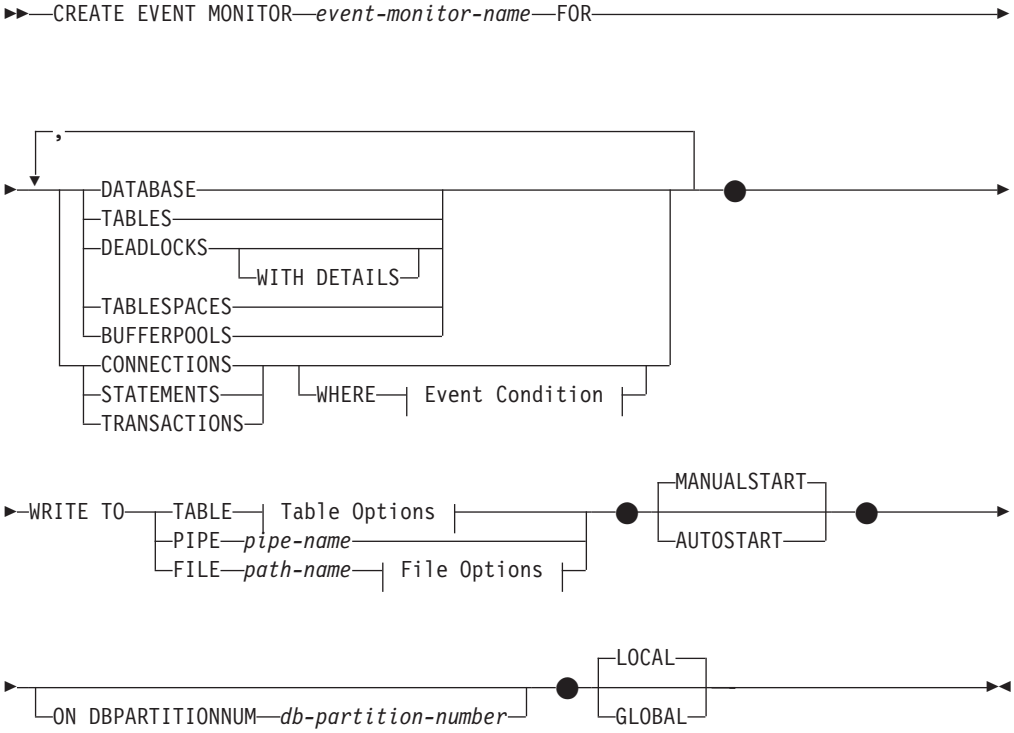
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

許可 ID の特権には、 SYSADM 権限または DBADM 権限のいずれかが含まれていなければなりません (SQLSTATE 42502)。

構文:



Event Condition:

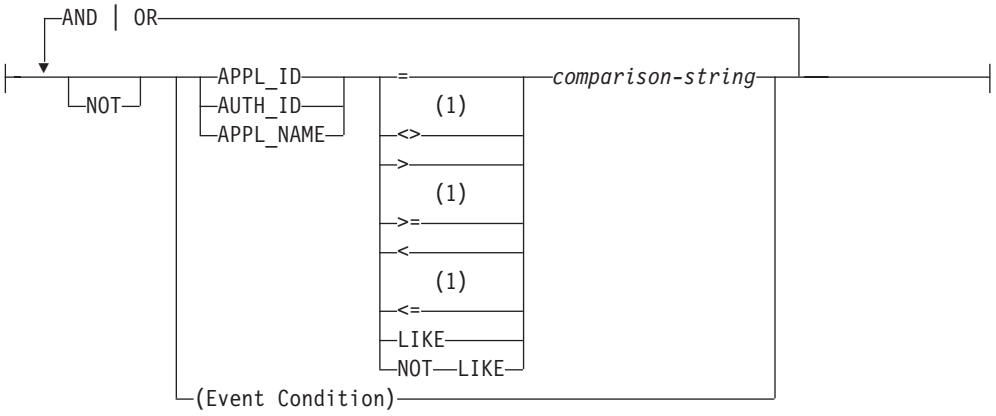
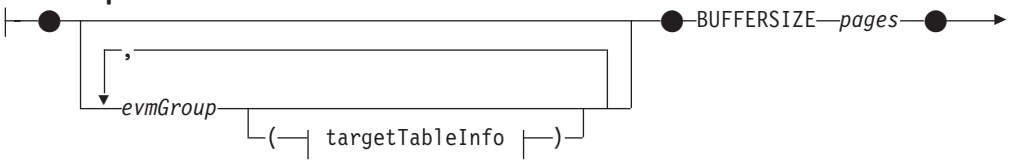
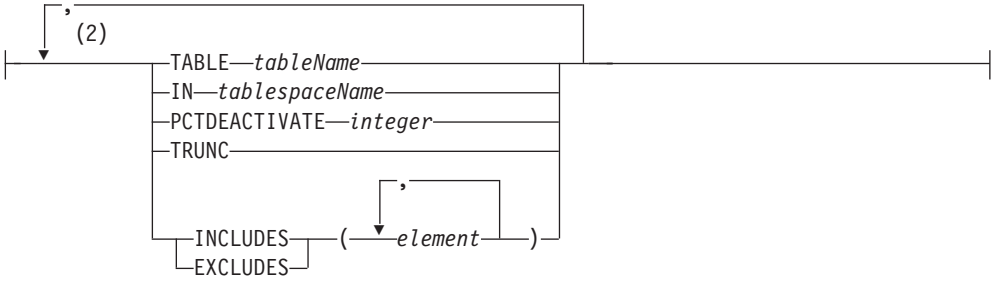


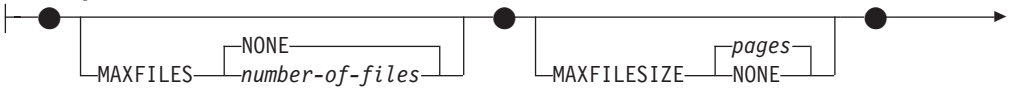
Table Options:

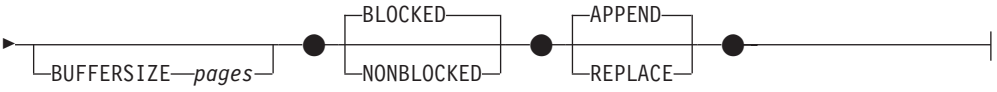


targetTableInfo:



File Options:





注:

1 これらの演算子の他の形式もサポートされます。

2 各文節は一度だけ指定できます。

説明:

event-monitor-name
イベント・モニターの名前を指定します。これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。 *event-monitor-name* (イベント・モニター名) は、すでにカタログに存在するイベント・モニターを指定する名前であってはなりません (SQLSTATE 42710)。

FOR
記録するイベント・タイプをこの後に指定します。

DATABASE
最後のアプリケーションがデータベースから切断された場合に、イベント・モニターがそのデータベース・イベントを記録することを指定します。

TABLES
最後のアプリケーションがデータベースから切断された場合に、イベント・モニターがアクティブな各表の表イベントを記録することを指定します。アクティブな表とは、データベースに最初に接続した時点以降に変更が行われた表です。

DEADLOCKS
デッドロックが発生した場合に、イベント・モニターがデッドロック・イベントを記録することを指定します。 WITH DETAILS オプションを指定すると、デッドロックに関係する各アプリケーションのより詳細なデッドロック接続イベントが生成されます。この追加の詳細には、以下の情報が含まれます。

- デッドロックが生じたときにアプリケーションが実行していたステートメントに関する情報。たとえば、ステートメント・テキスト。
- デッドロックが生じた際にアプリケーションによって保持されていたロック。パーティション・データベース環境では、デッドロックが生じたときにアプリケーションがロックを待機していたデータベース・パーティション上のロックだけが、ロックに含まれます。

DEADLOCKS および DEADLOCKS WITH DETAILS の両方を同時に同じステートメントでは指定できません (SQLSTATE 42613)。

TABLESPACES
最後のアプリケーションがデータベースから切断された場合に、イベント・モニターが各表スペースに関する表スペース・イベントを記録することを指定します。

BUFFERPOOLS
最後のアプリケーションがデータベースから切断された場合に、イベント・モニターがバッファー・プール・イベントを記録することを指定します。

CONNECTIONS
アプリケーションがデータベースから切断された場合に、イベント・モニターが接続イベントを記録することを指定します。

STATEMENTS
SQL ステートメントの実行が完了した時点で、イベント・モニターがステートメント・イベントを記録することを指定します。

TRANSACTIONS
トランザクションが完了した時点で (すなわち、コミットまたはロールバックの操作が行われた時点で)、イベント・モニターがトランザクション・イベントを記録することを指定します。

WHERE *event condition*
どの接続が CONNECTION、STATEMENT、または TRANSACTION イベントを引き起こすかを判別するフィルターを定義します。特定の接続に関してイベント条件 (*event condition*) の結果が真の場合、その接続は要求されたイベントを生成します。

この文節は、WHERE 文節の特殊な形式です。標準的な検索条件と混同しないようにしてください。

アプリケーションが特定のイベント・モニターに対するイベントを生成するかどうかを判別するために、この WHERE 文節は次のように評価されます。

1. イベント・モニターが初めてオンになった時点でアクティブである各接続がまず評価されます。
2. それ以後のデータベースへの新たな接続は、その接続時に評価されます。

WHERE 文節は各イベントごとに評価されるわけではありません。

WHERE 文節の指定がない場合、指定したイベント・タイプのイベントがすべてモニターされます。

イベント条件文節は、データベース・コード・ページ内で、 32 678 バイト以内の長さでなければなりません。(SQLSTATE 22001)

APPL_ID
該当の接続が CONNECTION、 STATEMENT、または TRANSACTION のいずれのイベント (指定による) を生成する必要があるかどうかを判別するために各接続のアプリケーション ID と *comparison-string* (比較ストリング) とを比較しなければならないことを指定します。

AUTH_ID
該当の接続が CONNECTION、 STATEMENT、または TRANSACTION のいずれのイベント (指定による) を生成する必要があるかどうかを判別するために各接続の許可 ID と *comparison-string* とを比較しなければならないことを指定します。

APPL_NAME
該当の接続が CONNECTION、 STATEMENT、または TRANSACTION のいずれのイベント (指定による) を生成する必要があるかどうかを判別するために各接続のアプリケーション・プログラム名と *comparison-string* とを比較しなければならないことを指定します。

アプリケーション・プログラム名は、 (最後のパス区切り記号の後の) アプリケーション・プログラム・ファイル名の最初の 20 バイトです。

comparison-string
データベースに接続する各アプリケーションの APPL_ID、 AUTH_ID、または APPL_NAME と比較するストリングを指定します。

comparison-string (比較ストリング) は、ストリング定数でなければなりません (ホスト変数や他のストリング式は使用できません)。

WRITE TO
データの出力先をこの後に指定します。

TABLE
イベント・モニターのデータの出力先が一連のデータベース表であることを示します。イベント・モニターは、データ・ストリームを 1 つまたは複数の論理データ・グループに分離し、各グループを別個の表に挿入します。ターゲット表のあるグループのデータは保持されますが、ターゲット表のないグループのデータは破棄されます。グループに含まれる各モニター・エレメントは、同じ名前の表列にマップされます。対応する表列を持つエレメントだけが表に挿入されます。他のエレメントは破棄されます。

Table Options
表フォーマット・オプションを指定します。

evmGroupInfo
論理データ・グループのターゲット表を定義します。この文節は、記録される各グループごとに指定しなければなりません。しかし evmGroupInfo 文節が指定されない場合には、イベント・モニター・タイプのすべてのグループが記録されます。

evmGroup
ターゲット表が定義する論理データ・グループを指定します。以下の表に示されているように、値はイベント・モニターのタイプに基づきます。

イベント・モニターのタイプ	evmGroup 値
データベース	DB
	CONTROL[1]
	DBMEMUSE
表	TABLE
	CONTROL[1]
デッドロック	CONNHEADER
	DEADLOCK
	DLCONN
	CONTROL[1]
詳細なデッドロック	CONNHEADER
	DEADLOCK
	DLCONN[2]
	DLLOCK[3]
	CONTROL[1]
表スペース	TABLESPACE
	CONTROL[1]
バッファー・プール	BUFFERPOOL
	CONTROL[1]

イベント・モニターのタイプ	evmGroup 値
接続	CONNHEADER
	CONN
	CONTROL[1]
	CONMEMUSE
ステートメント	CONNHEADER
	STMT
	SUBSECTION[4]
	CONTROL[1]
トランザクション	CONNHEADER
	XACT
	CONTROL[1]

[1] 論理データ・グループ dbheader (conn_time エレメントのみ)、開始およびオーバーフローは、CONTROL グループに書き込まれます。イベント・モニターがブロック化されておらず、イベントが破棄された場合に、オーバーフロー・グループが書き込まれます。

[2] DETAILED_DLCONN イベントに相当します。

[3] 各 DETAILED_DLCONN イベント内で発生する LOCK 論理データに相当します。

[4] パーティション・データベース環境に対してのみ作成されます。

targetTableInfo

グループのターゲット表を指定します。 *targetTableInfo* の値が指定されないと、 CREATE EVENT MONITOR の処理は以下のように続行されます。

- 派生した表名が使用されます (以下で説明します)。
- デフォルトの表スペースが選択されます (以下で説明します)。
- すべてのエレメントを含めます。
- PCTDEACTIVATE および TRUNC は指定されません。

TABLE *tableName*

ターゲット表の名前を指定します。名前が修飾なしの場合には、現行の許可 ID のスキーマが表スキーマのデフォルトになります。名前が指定されない場合、以下のように非修飾名は *evmGroup* および *event-monitor-name* から派生します。

```
substring(evmGroup || "_" || event-monitor-name,
  1,128)
```

IN *tablespaceName*

表を作成する表スペースの名前を指定します。表スペース名が指定されない場合には、以下のように表スペースが選択されます。

```
IF ユーザーが USE 特権を持っている
   表スペース IBMDEFAULTGROUP が存在する場合には
THEN それを選択します。
ELSE IF ユーザーが USE 特権をもっている
```

```
    表スペースが存在する場合には
THEN それを選択します。
ELSE エラーを出します (SQLSTATE 42727)
```

PCTDEACTIVATE *integer*
表がデータベース管理表スペース (DMS) に作成される場合には、 PCTDEACTIVATE パラメーターは、どの程度表スペースが満たされた時点でイベント・モニターが非活動化されるかを指定します。パーセンテージを表す値は、0 ～ 100 の範囲で指定可能です。デフォルト値は 100 です (表スペースが完全にいっぱいになるときにイベント・モニターが非活動化されることを意味します)。このオプションは、システム管理表スペース (SMS) では指定できません。

TRUNC
STMT_TEXT 列を VARCHAR(*n*) で定義することを指定します。ここで *n* は、表行に収めることができる最大サイズです。この場合、*n* より大きいステートメント・テキストは切り捨てられます。以下の例は、*n* の値の計算方法を例示しています。次のような前提があります。

- STMT 表は、32K ページを使用する表スペースに作成されます。
- 表内の他のすべての列の合計は、357 バイトです。

この場合、表の最大行サイズは 32677 バイトです。ですから、STMT_TEXT は VARCHAR(32316) に定義されます (つまり、32677 - 357 - 4)。 TRUNC が指定されない場合には、STMT_TEXT 列は CLOB(64K) に定義されます。STMT_TEXT が STMT グループおよび DLCONN グループで見つかることに注意してください (詳細なイベント・モニターのあるデッドロックの場合)。

INCLUDES
続くエレメントが表に含まれるように指定します。

EXCLUDES
続くエレメントが表に含まれないように指定します。

element
モニター・エレメントを指定します。エレメント情報は、以下の書式で提供されます。

- エレメント情報を指定しない。この場合、すべてのエレメントは CREATE TABLE ステートメントに組み込まれます。
- 書式 INCLUDES (element1, element2, ..., element*n*) のエレメントが組み込まれるように指定する。こうしたエレメントの表列だけが作成されます。
- 書式 EXCLUDES (element1, element2, ..., element*n*) のエレメントを除外するよう指定する。指定されたすべてのエレメントを除外した表列だけが作成されます。

db2evtbl コマンドを使用して、グループのエレメントの完全なリストを含む CREATE EVENT MONITOR ステートメントを構築します。

BUFFERSIZE *pages*
イベント・モニター・バッファーのサイズを指定します (4K ページ単位)。表イベント・モニターはバッファーからのすべてのデータを挿入し、バッファーが処理されると COMMIT を発行します。バッファーが大きいほど、イベント・モニターによって使用されるコミット有効範囲は大きくなります。活動頻度の高いイベント・モニターには、比較的活動頻度の低いイベント・モニターよりも大きいバッファーを用意する必要があります。モニターが開始されると、指定したサイズの 2 つのバッファーが割り振られます。イベント・モニターは、二重バッファリングを使用して、非同期入出力を可能にします。

各バッファーの最小 (およびデフォルト) サイズは 4 ページです (つまり、それぞれサイズが 16K の 2 つのバッファー)。バッファーはヒープから割り振られるので、バッファーの最大サイズはモニター・ヒープのサイズによって制約されます。多くのイベント・モニターを同時に使用する場合には、 *mon_heap_sz* データベース構成パラメーターのサイズを大きくします。

BLOCKED
エージェントが 2 つのイベント・バッファーがいっぱいであると判断した場合、イベントを生成するそのエージェントはイベント・バッファーがディスクへ書き込まれるのを待機しなければならないことを指定します。イベント・データが失われるのを防止したい場合には、BLOCKED を選択する必要があります。これはデフォルトのオプションです。

NONBLOCKED
エージェントが 2 つのイベント・バッファーがいっぱいであると判断した場合、イベントを生成するそのエージェントはイベント・バッファーがディスクへ書き込まれるのを待機しないことを指定します。NONBLOCKED のイベント・モニターは、BLOCKED のイベント・モニターほどには、データベース操作の速度を低下させません。ただし、NONBLOCKED のイベント・モニターは、アクティブ度の高いシステムではデータの消失の可能性が高くなります。

PIPE
イベント・モニター・データの出力先が名前付きパイプであることを指定します。イベント・モニターは、データを単一のストリーム (単一の無限に長いファイルであるかのように) でパイプに書き込みます。データをパイプに書き込む時点で、イベント・モニターはブロック化書き込みを行いません。パイプ・バッファーに余地がない場合、イベント・モニターはそのデータを廃棄します。データを失いたくない場合、モニターするアプリケーション側でデータを迅速に読み取る必要があります。

pipe-name
イベント・モニターがデータを書き込むパイプの名前 (AIX では FIFO) を指定します。

パイプの命名規則は、プラットフォームごとに異なります。 UNIX オペレーティング・システムでは、パイプ名はファイル名と同様に扱われます。したがって、相対パイプ名を使用することができ、相対パス名と同様に扱われます (下記の *path-name* を参照)。ただし、Windows NT または Windows 2000 では、パイプ名に関して特殊な構文があります。その結果、Windows NT または Windows 2000 では、絶対パイプ名が必要です。

パイプの存在は、イベント・モニターの作成時には検査されません。モニター・アプリケーションは、イベント・モニターが活動化された時点で、読み取り用パイプを作成し、オープンしておく必要があります。この時点でパイプが使用不可である場合には、イベント・モニターはオフになり、エラーがログに記録されます。 (つまり、AUTOSTART オプションの結果としてイベント・モニターがデータベースの開始時に活動化された場合、イベント・モニターはエラーをシステム・エラー・ログに記録します。) SET EVENT MONITOR STATE SQL ステートメントによってイベント・モニターが活動化された場合、そのステートメントはエラーになります (SQLSTATE 58030)。

FILE

イベント・モニターのデータの出力先がファイル (または一連のファイル) であることを示します。イベント・モニターは、拡張子 “evt” が付いた一連の 8 文字の番号のファイル (たとえば、00000000.evt、00000001.evt、および 00000002.evt) に、データ・ストリームを書き出します。データが細かく分割されている場合でも、データは 1 つの論理ファイルと見なす必要があります (つまり、データ・ストリームの最初はファイル 00000000.evt の最初のバイトであり、データ・ストリームの最後は、ファイル nnnnnnnn.evt の最後のバイトになります)。

各ファイルの最大サイズとファイルの最大数とを指定することができます。イベント・モニターは、1 つのイベント・レコードを 2 つのファイルに分割することはありません。ただしイベント・モニターは、互いに関連する複数のレコードを 2 つの異なるファイルに記録する場合があります。そのデータを使用するアプリケーションでは、イベント・ファイルの処理時にこのような関連する情報を追跡する必要があります。

path-name

イベント・モニターがイベント・ファイルのデータを書き込む先のディレクトリーの名前を指定します。パスはサーバーにおいて既知である必要があります。ただし、パス自体は別のパーティションにある可能性があります (たとえば UNIX 系のシステムでは、 NFS にマウントされたファイルである場合もあります)。 *path-name* (パス名) の指定には、ストリング定数を使用する必要があります。

ディレクトリーは、CREATE EVENT MONITOR の時に存在している必要はありません。ただし、イベント・モニターが活動化される時点で、ターゲット・パスの存否の検査が行われます。その時点で、ターゲット・パスが存在しない場合は、エラー (SQLSTATE 428A3) になります。

絶対パス (AIX の場合にルート・ディレクトリーで始まるパス、または Windows NT または Windows 2000 の場合にディスク ID で始まるパス) を指定すると、指定したパスが使用されます。相対パス (ルートか

ら始まっていないパス) が指定されている場合は、データベース・ディレクトリーの DB2EVENT ディレクトリーからの相対パスが使用されます。

相対パスが指定されている場合、それを絶対パスに変換するために DB2EVENT ディレクトリーが使用されます。したがって、絶対パスと相対パスの間に区別はありません。絶対パスは SYSCAT.EVENTMONITORS カタログ・ビューに保管されます。

複数のイベント・モニターに指定するターゲット・パスを同じパスにすることはできます。ただし、イベント・モニターの 1 つが最初に活動化されると、ターゲット・ディレクトリーが空でないかぎり、他のイベント・モニターはいずれも活動化することはできなくなります。

ファイル・オプション
ファイル形式のオプションを指定します。

MAXFILES NONE
イベント・モニターが作成するイベント・ファイルの数に制限がないことを指定します。これがデフォルトです。

MAXFILES *number-of-files*
特定の 1 つのイベント・モニターについて、 1 時点で存在するイベント・モニター・ファイルの数に限界があることを指定します。イベント・モニターがファイルをもう 1 つ作成しなければならない場合、ディレクトリー内の .evt ファイルの数が *number-of-files* よりも少ないかどうかが検査されます。すでにこの限界に達している場合、イベント・モニターはオフになります。

書き込み済みのイベント・ファイルを、アプリケーションがディレクトリーから削除した場合は、イベント・モニターが作成するファイルの合計数が *number-of-files* を超えることがあります。このオプションの使用によって、ユーザーはイベント・データによるディスク・スペースの消費量が指定量を超えることがないようにすることができます。

MAXFILESIZE *pages*
各イベント・モニター・ファイルのサイズに限界があることを指定します。イベント・モニターは、新しいイベント・レコードをファイルに書き込む場合、そのファイルが *pages* (4K ページ単位のページ数) を超えないかどうかを調べます。結果のファイルが大きすぎる場合、イベント・モニターはその次のファイルに切り替えます。このオプションのデフォルト値は次のとおりです。

- Windows NT または Windows 2000 - 200 個の 4K ページ
- UNIX - 1000 個の 4K ページ

ページ数は、少なくともイベント・バッファーのサイズ (ページ数) よりも大きくなければなりません。この要件が満たされていない場合、エラー (SQLSTATE 428A4) になります。

MAXFILESIZE NONE
ファイルのサイズに限界を設定しないことを指定します。
MAXFILESIZE NONE を指定すると、MAXFILES 1 も指定する必

要があります。このオプションは、特定のイベント・モニターのイベント・データすべてを 1 つのファイルに入れることを示します。このような場合、イベント・ファイルは 00000000.evt だけになります。

BUFFERSIZE *pages*
イベント・モニター・バッファーのサイズを指定します (4K ページ単位)。イベント・モニターのパフォーマンスを向上させるために、すべてのイベント・モニターのファイル入出力はバッファーに入れられます。バッファーが大きいほど、イベント・モニターによって行われる入出力は少なくなります。活動頻度の高いイベント・モニターには、比較的活動頻度の低いイベント・モニターよりも大きいバッファーを用意する必要があります。モニターが開始されると、指定したサイズの 2 つのバッファーが割り振られます。イベント・モニターは、二重バッファリングを使用して、非同期入出力を可能にします。

各バッファーの最小サイズとデフォルト・サイズ (このオプションが指定されていなければ) は、4 ページです (つまり、それぞれサイズが 16 K の 2 つのバッファー)。バッファーはヒープから割り振られるので、バッファーの最大サイズはモニター・ヒープ (MON_HEAP) のサイズによって制約されます。多くのイベント・モニターを同時に使用する場合には、 MON_HEAP データベース構成パラメーターのサイズを大きくしてください。

データをパイプに書き込むイベント・モニターにも、それぞれサイズが 1 ページの 2 つの内部 (構成不能) バッファーがあります。これらのバッファーも、モニター・ヒープ (MON_HEAP) から割り振られます。出力先がパイプである各アクティブ・イベント・モニターごとに、データベース・ヒープのサイズを 2 ページ分大きくしてください。

BLOCKED
エージェントが 2 つのイベント・バッファーがいっぱいであると判断した場合、イベントを生成するそのエージェントはイベント・バッファーがディスクへ書き込まれるのを待機しなければならないことを指定します。イベント・データが失われるのを防止したい場合には、BLOCKED を選択する必要があります。これはデフォルトのオプションです。

NONBLOCKED
エージェントが 2 つのイベント・バッファーがいっぱいであると判断した場合、イベントを生成するそのエージェントはイベント・バッファーがディスクへ書き込まれるのを待機しないことを指定します。 NONBLOCKED の指定を伴うイベント・モニターは、BLOCKED の指定を伴うイベント・モニターほどには、データベース操作の速度を低下させません。ただし、NONBLOCKED のイベント・モニターは、アクティブ度の高いシステムではデータの消失の可能性が高くなります。

APPEND
イベント・モニターがオンになった時点でイベント・データ・ファ

イルがすでに存在する場合、そのイベント・モニターは新しいイベント・データをデータ・ファイルの既存のストリームに付加するように指定します。イベント・モニターが再活動化されると、それはオフにならなかったかのように、イベント・ファイルへの書き込みを再開します。 APPEND はデフォルトのオプションです。

新しく作成されたイベント・モニターがイベント・データを書き込むディレクトリーに既存のイベント・データがない場合、 CREATE EVENT MONITOR 時に APPEND オプションは適用されません。

REPLACE
イベント・モニターがオンになった時点でイベント・データ・ファイルがすでに存在する場合、そのイベント・モニターが、イベント・ファイルをすべて削除して、ファイル 00000000.evt へのデータの書き込みを開始するように指定します。

MANUALSTART
データベースが開始されるたびに、イベント・モニターが自動的には開始されないことを指定します。 MANUALSTART オプションが指定されたイベント・モニターは、 SET EVENT MONITOR STATE ステートメントを使用して、手操作で活動化する必要があります。これはデフォルトのオプションです。

AUTOSTART
データベースが開始されるたびに、イベント・モニターが自動的に開始されることを指定します。

ON DBPARTITIONNUM *db-partition-number*
イベント・モニターを実行するデータベース・パーティションを指定します。この文節は、ファイルおよびパイプのイベント・モニターには有効ですが、write-to-table イベント・モニターには無効です。パーティション・データベース環境では、write-to-table イベント・モニターは、ターゲット表のための表スペースが定義されているすべてのパーティションで、イベントの実行と書き込みを行います。

モニターの有効範囲が GLOBAL として定義されている場合、すべてのデータベース・パーティションが指定のデータベース・パーティション番号に報告を行います。 I/O コンポーネントは指定のデータベース・パーティションで物理的に稼働し、指定されたファイルまたはパイプにレコードを書き込みます。

GLOBAL
イベント・モニターはすべてのデータベース・パーティションについて報告します。 DB2 Universal Database バージョン 8 のパーティション・データベースの場合、 GLOBAL として定義できるのは DEADLOCKS および DEADLOCKS WITH DETAILS イベント・モニターだけです。

LOCAL
イベント・モニターはモニター稼働しているデータベース・パーティションについてのみ報告します。この報告は、データベース活動の部分的なトレースです。これがデフォルトです。

規則:

- 各イベント・タイプ (DATABASE、TABLES、DEADLOCK、...) は、特定のイベント・モニターの定義に 1 回だけ指定できます。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
- イベント・モニターの定義は、SYSCAT.EVENTMONITORS カタログ・ビューに記録されます。イベント自体は、SYSCAT.EVENTS カタログ・ビューに記録されます。ターゲット表の名前は、SYSCAT.EVENTTABLES カタログ・ビューに記録されます。
- DEADLOCKS ではなく DEADLOCKS WITH DETAILS を使用すると、パフォーマンスに影響します。デッドロックが生じると、データベース・マネージャーは、追加のデッドロック情報を記録する余分の時間を必要とします。
- 通常 CONNHEADER イベントは、接続が確立されるといつでも書き込まれます。しかしイベント・モニターが DEADLOCKS WITH DETAILS に対してのみ作成される場合には、 CONNHEADER イベントは、接続が初めてデッドロックになったときだけ書き込まれます。
- BUFFERSIZE パラメーターは、STMT および DETAILED_DLCONN イベントのサイズを制約します。 STMT イベントがバッファーの大きさに合わない場合、ステートメント・テキストを切り捨ててそのイベントを切り捨てます。DETAILED_DLCONN イベントがバッファーの大きさに合わない場合、ロックを除去してそのイベントを切り捨てます。それでもまだ大きさが合わない場合には、ステートメント・テキストを切り捨てます。
- ***表イベント・モニターへの書き込み:***
  - 一般的な注:
    - すべてのターゲット表は、CREATE EVENT MONITOR ステートメントの実行時に作成されます。
    - 何らかの理由により表の作成に失敗すると、エラーがアプリケーション・プログラムに戻され、 CREATE EVENT MONITOR ステートメントは失敗します。
    - 1 つのターゲット表は、1 つのイベント・モニターでのみ使用可能です。CREATE EVENT MONITOR 処理時に、別のイベント・モニターによって既に定義されているターゲット表が検出されると、 CREATE EVENT MONITOR ステートメントは失敗し、エラーがアプリケーション・プログラムに戻されます。表名が SYSCAT.EVENTTABLES カタログ・ビューの値と一致する場合には、その表は別のイベント・モニターによって使用されるよう定義されています。
    - CREATE EVENT MONITOR 処理時に、表が既に存在するものの別のイベント・モニターによって使用されるよう定義されていない場合には、表は作成されず、処理は続行されます。警告が、アプリケーション・プログラムに出されます。
    - CREATE EVENT MONITOR ステートメントが実行される前に、すべての表スペースが存在しなければなりません。 CREATE EVENT MONITOR ステートメントは、表スペースを作成しません。
    - 指定されている場合、LOCAL および GLOBAL キーワードは無視されます。 WRITE TO TABLE イベント・モニターの場合、イベント・モニター

の出力処理またはスレッドは、インスタンスの各データベース・パーティションで開始され、そうした個々の処理は実行しているデータベース・パーティションのデータのみを報告します。

- フラット・モニター・ログ・ファイルからの以下のイベント・タイプまたはパイプ・フォーマットは、表イベント・モニターへの書き込みにより記録されません。
  - LOG_STREAM_HEADER
  - LOG_HEADER
  - DB_HEADER (エレメント db_name および db_path は記録されません。エレメント conn_time は CONTROL に記録されます。)
- パーティション・データベース環境では、表スペースが存在するパーティション上のターゲット表だけにデータが書き込まれます。ターゲット表のための表スペースがいずれかのパーティションに存在しない場合は、そのターゲット表についてのデータは無視されます。この動作によってユーザーは、特定のパーティション上にのみ存在する表スペースを作成して、モニターするパーティションのサブセットを選択できます。

  パーティション・データベース環境で、いくつかのターゲット表がパーティション上に存在しないものの、その同じパーティションに他のターゲット表がある場合には、そのパーティションにあるターゲット表についてのデータだけが記録されます。
- ユーザーは、すべてのターゲット表を手動で整理する必要があります。

– 表列:

- 表の列名は、イベント・モニター・エレメント ID と一致します。タイプ sqlm_time (経過時間) のモニター変数は、例外です。このタイプの列名は TYPE_NAME_S および TYPE_NAME_MS であり、それぞれ秒単位およびマイクロ秒単位で時間を保管する列を表しています。対応するターゲット表列のないイベント・モニター・エレメントは、無視されます。
- db2evtbl コマンドを使用して、グループのエレメントの完全なリストを含む CREATE EVENT MONITOR コマンドを構築します。
- モニター・エレメントに使用されている列のタイプは、以下のマッピングに相関します。

SQLM_TYPE_STRING	CHAR[n], VARCHAR[n] or CLOB(n) (イベント・モニター内のデータが *n* バイトを超えた場合は 切り捨てが行われる。)
SQLM_TYPE_U8BIT and SQLM_TYPE_8BIT	SMALLINT, INTEGER or BIGINT
SQLM_TYPE_16BIT and SQLM_TYPE_U16BIT	SMALLINT, INTEGER or BIGINT
SQLM_TYPE_32BIT and SQLM_TYPE_U32BIT	INTEGER or BIGINT
SQLM_TYPE_U64BIT and SQLM_TYPE_64BIT	BIGINT
sqlm_timestamp	TIMESTAMP
sqlm_time(elapsed time)	BIGINT
sqlca:	
sqlerrmc	VARCHAR[72]
sqlstate	CHAR[5]
sqlwarn	CHAR[11]
other fields	INTEGER or BIGINT

- 列は、NOT NULL になるよう定義されています。
- CLOB 列のある表のパフォーマンスは VARCHAR 列を含む表より劣るので、 STMT evmGroup (または、DEADLOCKS WITH DETAILS イベント・

タイプを使用する場合には DLCONN evmGroup) を指定する場合には、TRUNC キーワードの使用を考慮してください。

- 他のターゲット表とは異なり、 CONTROL 表の列はモニター・エレメント ID と一致しません。列は、以下のように定義されます。

列名	データ・タイプ	ヌル可能性	説明
PARTITION_KEY	INTEGER	N	パーティション・キー (パーティション・データベースのみ)
PARTITION_NUMBER	INTEGER	N	パーティション番号 (パーティション・データベースのみ)
EVMONNAME	VARCHAR(128)	N	イベント・モニターの名前
MESSAGE	VARCHAR(128)	N	MESSAGE_TIME 列の性質を記述します。 これは、次のいずれかになります。 - FIRST_CONNECT (活動化の前にデータベースに最初に接続する時刻) - EVMON_START (EVMONNAME にリストされているイベント・モニターが開始された時刻) - OVERFLOWS:*n* (バッファー・オーバーフローのため、*n* 個のレコードが破棄されたことを示します)
MESSAGE_TIME	TIMESTAMP	N	タイム・スタンプ

- パーティション・データベース環境では、各表の最初の列は PARTITION_KEY という名前が付けられ、 NOT NULL で、INTEGER タイプです。この列は、表のパーティション・キーとして使用されます。各イベント・モニター処理は、この列の値を選択し、処理を実行中のデータベース・パーティションにデータを挿入します。つまり、挿入操作はイベント・モニター処理を実行しているデータベース・パーティションでローカルに実行します。任意のデータベース・パーティションで、PARTITION_KEY フィールドは同じ値を含みます。これは、データ・パーティションがドロップされてデータ再分散が実行される場合に、ドロップされるデータベース・パーティション上のすべてのデータは、公平に分散されるのでなく、単一の任意のデータベース・パーティションに渡されることを意味します。ですから、データベース・パーティションを除去する前に、そのデータベース・パーティション上のすべての表行の削除を考慮してください。
- パーティション・データベース環境では、PARTITION_NUMBER という名前の列を各表で定義できます。この列は NOT NULL で INTEGER タイプです。データが挿入されたパーティションの番号が含まれています。PARTITION_KEY 列とは異なり、PARTITION_NUMBER 列は必須ではありません。 PARTITION_NUMBER 列は、パーティション・データベース環境以外では使用できません。

– 表属性:

- デフォルトの表属性が使用されます。パーティション・キーを除き (パーティション・データベースのみ)、表の作成時には追加のオプションは指定されません。
- 表の索引を作成できます。
- 別の表属性 (揮発性、RI、トリガー、制約など) を追加できますが、イベント・モニター処理 (またはスレッド) はそれらを無視します。

- 表属性として ″not logged initially″ (最初はログ記録されない) が追加されると、最初の COMMIT 時にオフになり、戻せません。

– イベント・モニターの活動化:
  - イベント・モニターを活動化する際、すべてのターゲット表名が SYSCAT.EVENTTABLES カタログ・ビューから取り出されます。
  - パーティション・データベース環境では、インスタンスの各パーティションで活動化処理が行われます。特定のパーティションで、活動化処理は、それぞれのターゲット表ごとに、表スペースとデータベース・パーティション・グループを判別します。イベント・モニターは、パーティションに少なくとも 1 つのターゲット表が存在する場合のみ、そのパーティションで活動化します。さらに、パーティションにいずれかのターゲット表が見つからない場合は、そのターゲット表にはフラグが立てられ、そのターゲット表を宛先とするデータが実行時処理中にドロップされるようにします。
  - イベント・モニターが活動化する際にターゲット表が存在しない場合 (またはパーティション・データベース環境で、表スペースがパーティション・データベースにない場合) には、活動化は続行され、この表に挿入されるはずであったデータは無視されます。
  - 活動化処理は、各ターゲット表の妥当性検査をします。妥当性検査がうまくいかないと、イベント・モニターの活動化は失敗し、管理ログにメッセージが書き込まれます。
  - パーティション・データベース環境における活動化の際に、FIRST_CONNECT および EVMON_START の CONTROL 表行は、カタログ・データベース・パーティションでのみ挿入されます。これには、コントロール表の表スペースがカタログ・データベース・パーティションに存在することが必要です。カタログ・データベース・パーティションにない場合には、挿入は実行されません。
  - パーティション・データベース環境では、表イベント・モニターへの書き込みがアクティブであるときにパーティションがまだアクティブにない場合には、そのパーティションはイベント・モニターがアクティブ化される前にアクティブ化されます。この場合、データベース活動化は、 SQL CONNECT ステートメントがデータベースのすべてのパーティションを活動化するのと同様に行われます。

– ランタイム:
  - イベント・モニターは DBADM 権限で実行されます。
  - イベント・モニターがアクティブであるときに、ターゲット表への挿入操作が失敗すると、
    - コミットされていない変更がロールバックされます。
    - メッセージが、管理ログに書き込まれます。
    - イベント・モニターが非活動になります。
  - イベント・モニターがアクティブである場合には、イベント・モニター・バッファーの処理を終えるとローカル COMMIT が実行されます。
  - パーティション・データベース環境では、アプリケーション・コーディネーターのデータベース・パーティションで実行されているイベント・モニター処理によって、大きさが 65 535 バイトまで可能な実際のステートメント・

テキストだけが保管されます (STMT または DLCONN 表に)。他のデータベース・パーティションでは、この値の長さはゼロです。

- パーティション・データベース以外の環境では、最後のアプリケーションが終了する際 (およびデータベースが明示的に活動化されたなかった時) に、表イベント・モニターへのすべての書き込みは非活動状態になります。パーティション・データベース環境では、カタログ・パーティションが非活動状態になると、表イベント・モニターへの書き込みも非活動化します。
- DROP EVENT MONITOR ステートメントはターゲット表をドロップしません。

例:

例 1: 次の例では、SMITHPAY と呼ばれるイベント・モニターを作成します。このイベント・モニターは、データベースと、 JSMITH 許可 ID が所有する PAYROLL アプリケーションによって実行される SQL ステートメントに関するイベント・データを収集します。データは、絶対パス/home/jsmith/event/smithpay/ に付加されます。最大 25 のファイルが作成されます。各ファイルの最大長は 4K ページ 1 024 個分です。ファイル入出力は非ブロック化されます。

```
CREATE EVENT MONITOR SMITHPAY
  FOR DATABASE, STATEMENTS
  WHERE APPL_NAME = 'PAYROLL' AND AUTH_ID = 'JSMITH'
  WRITE TO FILE '/home/jsmith/event/smithpay'
  MAXFILES 25
  MAXFILESIZE 1024
  NONBLOCKED
  APPEND
```

例 2: 次の例では、DEADLOCKS_EVTS と呼ばれるイベント・モニターを作成します。このイベント・モニターは、デッドロック・イベントを収集して、それらを相対パス DLOCKS に書き込みます。 1 つのファイルに書き込まれ、ファイル・サイズに限界はありません。イベント・モニターが活動化されるたびに、ファイル 00000000.evt が存在する場合にはそこにイベント・データが付加されます。このイベント・モニターは、データベースが開始されるたびに開始されます。入出力はデフォルト解釈によりブロック化されます。

```
CREATE EVENT MONITOR DEADLOCK_EVTS
  FOR DEADLOCKS
  WRITE TO FILE 'DLOCKS'
  MAXFILES 1
  MAXFILESIZE NONE
  AUTOSTART
```

例 3: この例では、DB_APPLS と呼ばれるイベント・モニターを作成します。このイベント・モニターは、接続イベントを収集し、それらのデータを名前付きパイプ /home/jsmith/applpipe に書き込みます。

```
CREATE EVENT MONITOR DB_APPLS
  FOR CONNECTIONS
  WRITE TO PIPE '/home/jsmith/applpipe'
```

例 4: この例では、パーティション・データベース環境であることが前提で、 FOO と呼ばれるイベント・モニターを作成します。このイベント・モニターは SQL ステートメント・イベントを収集し、以下の導出された名前で SQL 表に書き込みます。

- CONNHEADER_FOO

- STMT_FOO
- SUBSECTION_FOO
- CONTROL_FOO

表スペース情報が提供されていないので、 IN *tablespaceName* 文節で説明されていた規則に基づいて、システムが選択した表スペースにすべての表が作成されます。すべての表がこれらのグループのすべてのエレメントを含みます (つまり、列名がエレメント名と同じになるように列は定義されます。)

```
CREATE EVENT MONITOR FOO
  FOR STATEMENTS
  WRITE TO TABLE
```

例 5 この例では、パーティション・データベース環境であることが前提で、 BAR と呼ばれるイベント・モニターを作成します。このイベント・モニターは SQL ステートメントおよびトランザクション・イベントを収集し、以下のように表に書き込みます。

- STMT グループからのすべてのデータは、MYDEPT.MYSTMTINFO 表に書き込まれます。この表は、MYTABLESPACE 表スペースに作成されます。ROWS_READ、 ROWS_WRITTEN、および STMT_TEXT エレメントに対してのみ列を作成します。グループの他のエレメントは破棄されます。
- SUBSECTION グループからのすべてのデータは、MYDEPT.MYSUBSECTIONINFO 表に書き込まれます。この表は、MYTABLESPACE 表スペースに作成されます。この表は、START_TIME、STOP_TIME、および PARTIAL_RECORD 以外のすべての列を含みます。
- XACT グループからのすべてのデータは、XACT_BAR 表に書き込まれます。表スペース情報が提供されていないので、 IN *tablespaceName* 文節で説明されていた規則に基づいて、システムが選択した表スペースに表が作成されます。 この表には、XACT グループにあるすべてのエレメントが含まれます。
- CONNHEADER または CONTROL に対しては表は作成されません。これらのグループのすべてのデータは破棄されます。

```
CREATE EVENT MONITOR BAR
  FOR STATEMENTS, TRANSACTIONS
  WRITE TO TABLE
  STMT(TABLE MYDEPT.MYSTMTINFO, IN MYTABLESPACE,
    INCLUDES(ROWS_READ, ROWS_WRITTEN, STMT_TEXT)),
  SUBSECTION(TABLE MYDEPT.MYSUBSECTIONINFO, IN MYTABLESPACE,
    EXCLUDES(START_TIME, STOP_TIME, PARTIAL_RECORD)),
  XACT
```

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- *システム・モニター ガイドおよびリファレンス* の『イベント・モニターの論理データ・グループおよびモニター・エレメント』

CREATE FUNCTION

このステートメントは、ユーザー定義の関数または関数テンプレートをアプリケーション・サーバーに登録または定義する場合に使用されます。

このステートメントを使用して作成できる関数には、異なる 5 つのタイプがあります。これらのそれぞれについて、個々に説明します。

- 外部スカラー。この関数はプログラミング言語で書かれ、1 つのスカラー値を戻します。外部の実行可能プログラムが、関数の種々の属性を伴ってデータベースに登録されます。
- 外部表。この関数はプログラミング言語で書かれ、完全な表を戻します。外部の実行可能プログラムが、関数の種々の属性を伴ってデータベースに登録されます。
- OLE DB 外部表。ユーザー定義の OLE DB 外部表関数はデータベースに登録されて、 OLE DB Provider からデータをアクセスします。
- ソースまたはテンプレート。ソース関数は、データベースにすでに登録されている他の関数 (組み込み、外部、SQL、またはソースのいずれか) を呼び出すことによってインプリメントされます。

  関数テンプレート という部分関数を作成し、戻される値のタイプを定義することができますが、実行可能コードを含めることはできません。ユーザーはこれをフェデレーテッド・システム内のデータ・ソース関数にマップし、フェデレーテッド・データベースからそのデータ・ソース関数を呼び出せるようにします。関数テンプレートは、フェデレーテッド・サーバーとして指定されたアプリケーション・サーバーにだけ登録できます。
- SQL スカラー、表、または行。関数本体は SQL で書かれ、データベースに登録で定義されます。これは、スカラー値、表、または単一の行を戻します。

関連資料:

- 245 ページの『CREATE FUNCTION (OLE DB 外部表)』
- 263 ページの『CREATE FUNCTION (SQL スカラー、表、または行)』
- 200 ページの『CREATE FUNCTION (外部スカラー)』
- 226 ページの『CREATE FUNCTION (外部表)』
- 252 ページの『CREATE FUNCTION (ソースまたはテンプレート)』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』
- 『udfcli.sqc -- Call a variety of types of user-defined functions (C)』
- 『udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)』
- 『udfcli.c -- How to work with different types of user-defined functions (UDFs)』
- 『udfcli.sqC -- Call a variety of types of user-defined functions (C++)』
- 『udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)』
- 『UDFCreate.db2 -- How to catalog the Java UDFs contained in UDFsrv.java 』
- 『UDFjCreate.db2 -- How to catalog the Java UDFs contained in UDFjsrv.java 』

CREATE FUNCTION (外部スカラー)

このステートメントは、ユーザー定義の外部スカラー関数をアプリケーション・サーバーに登録する場合に使用されます。*スカラー関数* は、呼び出されるたびに 1 つの値を戻し、 SQL 式が使用可能な個所であれば一般に使用することができます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限、および以下の少なくとも 1 つ。
  - データベースに対する IMPLICIT_SCHEMA 権限 (関数のスキーマ名が既存のスキーマを指していない場合)
  - スキーマに対する CREATEIN 特権 (関数のスキーマ名が既存のスキーマを指している場合)

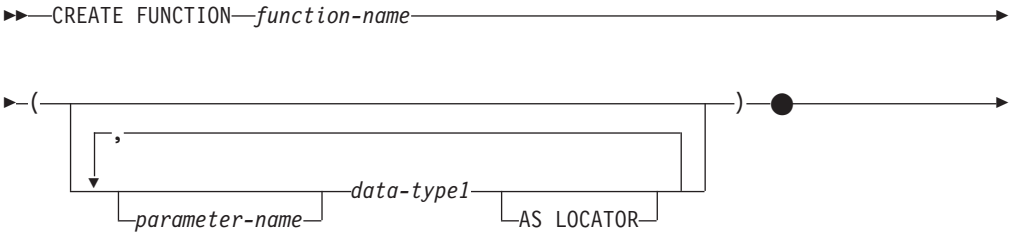
非 fenced の関数を作成するには、ステートメントの許可 ID の特権に、以下の特権の少なくとも 1 つが含まれている必要があります。

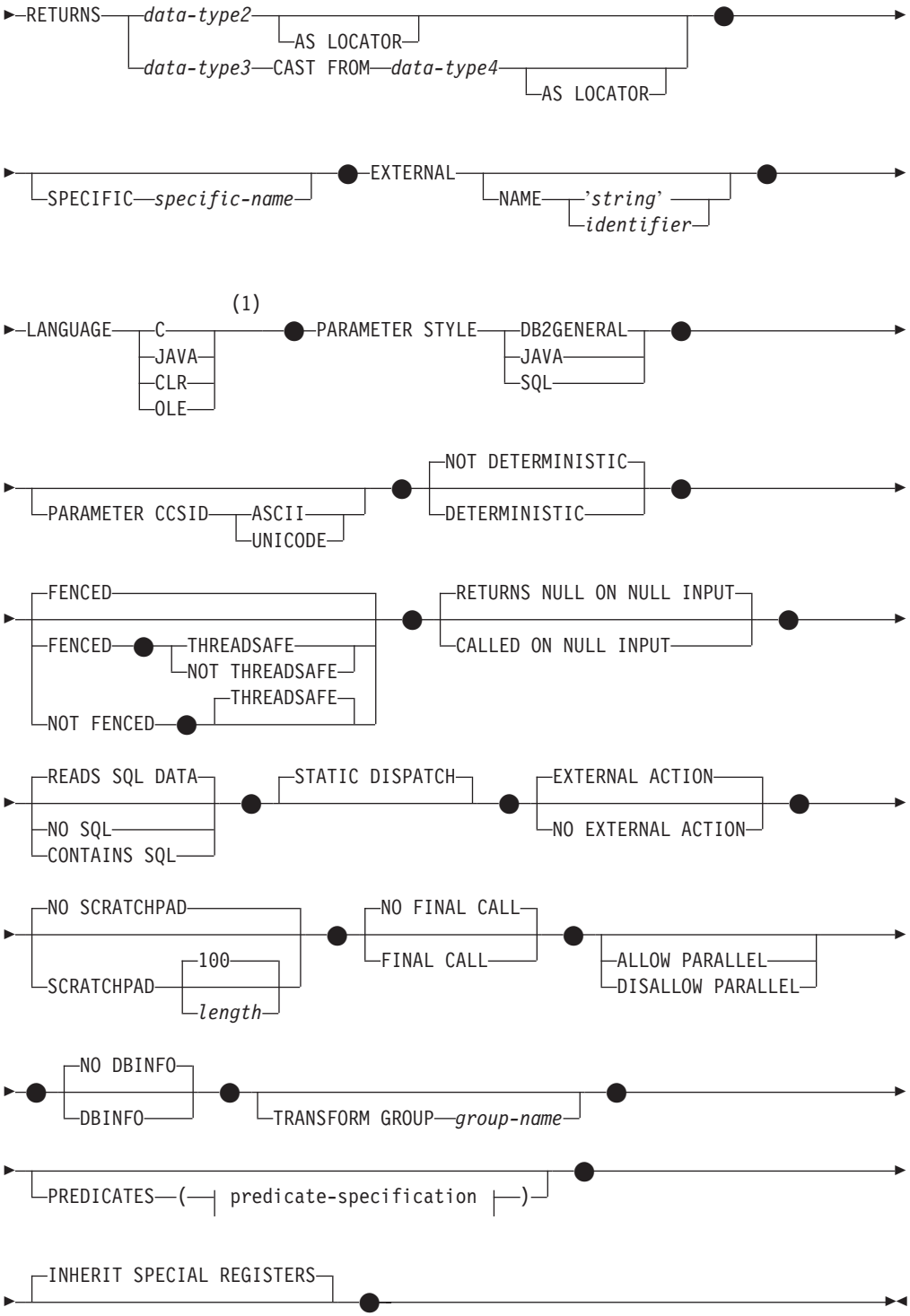
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限
- SYSADM または DBADM 権限

fenced 関数を作成する場合には、さらに別の権限や特権は必要ありません。

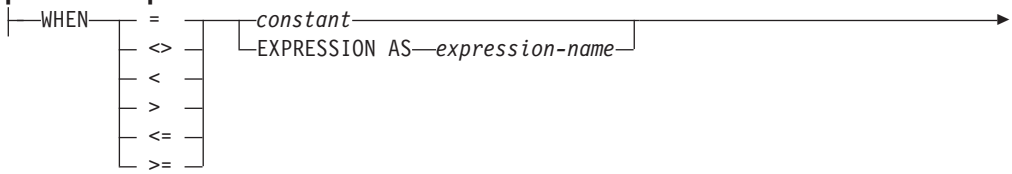
許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

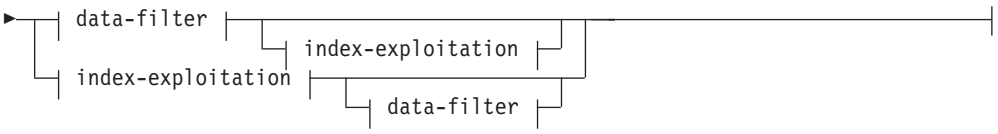
構文:



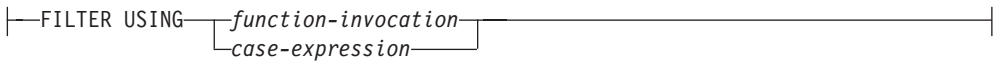


predicate-specification:

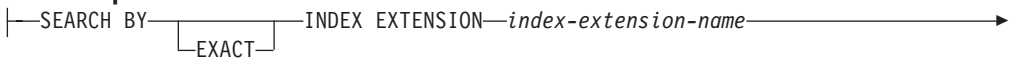




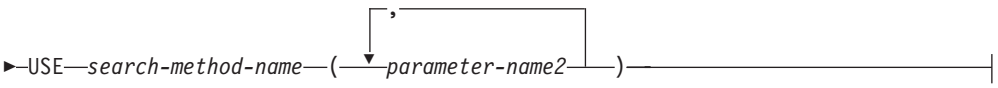
data-filter:



index-exploitation:



exploitation-rule:



注:

1 LANGUAGE SQL もサポートされています。

説明:

function-name

定義する関数の名前を指定します。これは、関数を指定する修飾または非修飾の名前です。 *function-name* (関数名) の非修飾形式は SQL ID です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。最初のパラメーターが構造タイプの場合、修飾名は、最初のパラメーターのデータ・タイプと同じであってはなりません。

暗黙または明示の修飾子を含む名前、およびパラメーターの数と各パラメーターのデータ・タイプ (データ・タイプの長さ、精度、または位取りの各属性には関係なく) は、カタログに記述されている関数またはメソッドを指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数およびデータ・タイプとの組み合わせは、そのスキーマ内では当然ユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分から成る名前を指定する場合、 “SYS” で始まる *schema-name* (スキーマ名) は使用できません。使用した場合、エラー (SQLSTATE 42939) になります。

述部のキーワードとして使用される多くの名前は、システム使用として予約されており、 *function-name* として使用することはできません。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。この規則に違反すると、エラーになります (SQLSTATE 42939)。

一般に、関数のシグニチャーに何らかの差異がある場合には、同じ名前を複数の関数に使用することができます。

禁止されてはいませんが、意図的にオーバーライドを行う場合を除き、外部ユーザー定義関数の名前として、組み込み関数と同じ名前を指定するのは避けるべきです。異なる意味を持つ関数に組み込みのスカラー関数または列関数と同じ名前 (たとえば、 LENGTH、VALUE、MAX など) を与えることは、たとえその引き数が一致していたとしても、動的 SQL ステートメントの過程で、あるいは静的 SQL アプリケーションの再バインド時に問題が生じます。すなわち、アプリケーションが失敗することがあり、また、さらに悪いケースとして、外見上は正常に実行されていても、結果が異なる場合があります。

parameter-name

後続の関数定義で使用できるパラメーターを指定します。パラメーター名は、述部指定の *index-exploitation* 文節にある関数のパラメーターを参照するのに必要です。

(data-type1,...)

関数の入力パラメーターの数を指定するとともに、各パラメーターのデータ・タイプを指定します。このリストには、関数が受け取ることを予期している各パラメーターごとに 1 つの項目を指定する必要があります。パラメーターの数は 90 を超えることはできません。この限界を超えると、エラー (SQLSTATE 54023) になります。

パラメーターのない関数も登録可能です。この場合、指定するデータ・タイプがない場合でも、括弧はコーディングする必要があります。たとえば、以下のようになります。

```
CREATE FUNCTION WOOFER() ...
```

その対応するすべてのパラメーターのタイプがまったく同じである場合でも、 1 つのスキーマ中に名前が同じ 2 つの関数があってはなりません。このタイプの比較では長さ、精度、および位取りは考慮されません。したがって、CHAR(8) と CHAR(35)、また DECIMAL(11,2) と DECIMAL (4,3) は、それぞれ同じタイプと見なされます。 Unicode データベースの場合には、CHAR(13) と GRAPHIC(8) は、それぞれ同じタイプと見なされます。さらに、DECIMAL と NUMERIC などのように、この目的で複数のタイプが同じタイプとして扱われることがあります。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

たとえば、以下のステートメントの場合、

```
CREATE FUNCTION PART (INT, CHAR(15)) ...
CREATE FUNCTION PART (INTEGER, CHAR(40)) ...
CREATE FUNCTION ANGLE (DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (DEC(10,7)) ...
```

2 番目と 4 番目のステートメントは、重複する関数と見なされ、エラーになります。

data-type1
パラメーターのデータ・タイプを指定します。

- CREATE TABLE ステートメントの *data-type1* の定義で指定が可能で、関数の作成に使用されている言語において対応するものがある SQL データ・タイプ仕様と省略形を指定できます。
- DECIMAL (および NUMERIC) は、LANGUAGE C と OLE では無効です (SQLSTATE 42815)。
- CLR は 28 より大きい DECIMAL スケールをサポートしていません (SQLSTATE 42613)。
- REF(*type-name*) は、パラメーターのタイプとして指定できます。ただし、パラメーターに有効範囲を指定してはなりません。
- 適切なトランスフォーム関数が、関連するトランスフォーム・グループに存在する場合には、構造タイプを指定できます。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターを UDF に渡すことを指定します。これにより、UDF に渡すバイト数を大幅に減らすことができ、パフォーマンスも向上します (特に、UDF にとって実際に必要になる値が数バイトだけである場合)。

以下の例は、パラメーター定義の AS LOCATOR 文節の使用法を示しています。

```
CREATE FUNCTION foo (CLOB(10M) AS LOCATOR, IMAGE AS LOCATOR)
  ...
```

ここで、IMAGE は LOB タイプの 1 つに基づく特殊タイプであると想定します。

また、引き数のプロモーション目的には、AS LOCATOR 文節の効果はないことに注意してください。この例では、タイプはそれぞれ CLOB と IMAGE であると見なされるので、関数に CHAR 引き数または VARCHAR 引き数が最初の引き数として渡されます。同様に、関数シグニチャーに対して AS LOCATOR の効果はありません。関数シグニチャーは、(a) ″関数解決″ と呼ばれるプロセスによって DML で参照された場合、 (b) COMMENT ON や DROP などの DDL ステートメントで参照された場合に関数をマッチングする際に使用されます。実際に、この文節はシグニチャーの指定のない COMMENT ON や DROP で使用しても、しなくても構いません。

LOB または LOB に基づく特殊タイプ以外のタイプに対して AS LOCATOR を指定すると、エラー (SQLSTATE 42601) が発生します。

関数が FENCED で NO SQL オプションを持っている場合、 AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

RETURNS

これは必須の文節であり、関数の出力を指定します。

data-type2

出力のデータ・タイプを指定します。

この場合、上記の関数パラメーター *data-type1* で説明した外部関数のパラメーターと同じ考慮事項が適用されます。

AS LOCATOR

LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターが UDF から渡されることを示します。

data-type3 **CAST FROM** *data-type4*

出力のデータ・タイプを指定します。

この形式の RETURNS 文節は、関数コードから戻されたデータ・タイプとは異なるデータ・タイプを、呼び出しステートメントに戻すのに使用されます。たとえば、以下の例で、

```
CREATE FUNCTION GET_HIRE_DATE(CHAR(6))
  RETURNS DATE CAST FROM CHAR(10)
  ...
```

CHAR(10) の値が関数コードからデータベース・マネージャーに戻され、データベース・マネージャーは、その値を DATE に変換して、変換された値を呼び出し側ステートメントに渡します。 *data-type4* は、 *data-type3* パラメーターにキャスト可能でなければなりません。キャスト可能でない場合、エラー (SQLSTATE 42880) になります。

data-type3 の長さ、精度または位取りは、 *data-type4* から推断することができるので、 *data-type3* に指定されるパラメーター化タイプの長さ、精度、または位取りを指定する必要はありません (指定は可能です)。代わりに、空の括弧を使用できます (たとえば、VARCHAR() など)。パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、FLOAT() を使用することはできません (SQLSTATE 42601)。

特殊タイプおよび構造タイプは、 *data-type4* に指定するタイプとしては無効です (SQLSTATE 42815)。

キャスト操作は、変換エラーになる可能性があるランタイム・チェックの対象にもなります。

AS LOCATOR

data-type4 の指定が、 LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターが UDF から戻されることを示します。

SPECIFIC *specific-name*

定義する関数のインスタンスに対するユニーク名を指定します。この特定名は、この関数をソース関数として使用する場合、この関数をドロップする場合、また

はこの関数にコメントを付ける場合に使用することができます。これは、関数の呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の関数インスタンスまたはメソッド指定を識別するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) になります。

specific-name は、既存の *function-name (関数名)* と同じでも構いません。

修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

EXTERNAL

この文節は、外部プログラミング言語で作成され、文書化されたリンケージの規則とインターフェースに準拠している新しい関数を登録するのに、 CREATE FUNCTION ステートメントが使用されていることを示します。

NAME 文節を指定しない場合、″NAME *function-name*″ が想定されます。

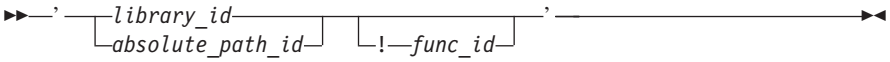
NAME *'string'*

この文節は、定義している関数をインプリメントするユーザー作成コードの名前を指定します。

'string' オプションは、最大 254 文字のストリング定数です。 ストリングに使用される形式は、指定した LANGUAGE によって異なります。

- LANGUAGE C の場合:

  指定する *string* (ストリング) は、作成しているユーザー定義関数を実行するためにデータベース・マネージャーが呼び出すライブラリー名と、そのライブラリー中の関数名です。ライブラリー (およびそのライブラリー中の関数) は、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数が SQL ステートメントで使用される時点では、そのライブラリーとそのライブラリー中の該当の関数は存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラー (SQLSTATE 42724) が戻されます。



  単一引用符内に、余分なブランクを使用することはできません。

  library_id

  関数を含むライブラリー名を指定します。データベース・マネージャーは、次のようにしてこのライブラリーを特定します。

  – UNIX 系システムの場合、*library_id* が 'myfunc' と指定されており、データベース・マネージャーが /u/production から実行されて

いると、データベース・マネージャーはライブラリー /u/production/sqllib/function/myfunc で関数を特定します。

– Windows オペレーティング・システムの場合、データベース・マネージャーは LIBPATH または PATH 環境変数に指定されているディレクトリー・パスから関数を特定します。

absolute_path_id
関数を含んでいるファイルの絶対パス名を指定します。

たとえば、UNIX 系システムの場合、'/u/jchui/mylib/myfunc' を指定すると、データベース・マネージャーは /u/jchui/mylib を調べて myfunc 共用ライブラリーを探します。

Windows オペレーティング・システムの場合、'd:¥mylib¥myfunc' を指定すると、データベース・マネージャーは d:¥mylib ディレクトリーからダイナミック・リンク・ライブラリー myfunc.dll をロードします。絶対パス ID がルーチン本体の識別に使用されている場合は、.dll 拡張子を必ず付加してください。

! func_id
呼び出される関数の入り口点名を指定します。 ! は、ライブラリー ID と関数 ID との間の区切り文字です。 *! func_id* を省略すると、データベース・マネージャーはライブラリーのリンク時に確立されたデフォルトの入り口点を使用します。

たとえば、UNIX 系システムで 'mymod!func8' と指定すると、データベース・マネージャーはライブラリー $inst_home_dir/sqllib/function/mymod を調べて、そのライブラリー内の入り口点 func8 を使用します。

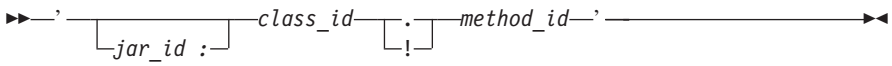
Windows オペレーティング・システムの場合 'mymod!func8' を指定すると、データベース・マネージャーは mymod.dll ファイルをロードして、そのダイナミック・リンク・ライブラリー (DLL) の func8() 関数を呼び出します。

ストリングの形式が正しくない場合には、エラー (SQLSTATE 42878) になります。

すべての外部関数の本体は、データベースのすべてのパーティションで使用可能なディレクトリーにある必要があります。

- LANGUAGE JAVA の場合:

  指定する *string* には、作成中のユーザー定義関数を実行するためにデータベース・マネージャーが呼び出す、任意指定の jar ファイル、クラス ID、およびメソッド ID が含まれています。クラス ID とメソッド ID は、 CREATE FUNCTION ステートメントの実行時には存在している必要はありません。 *jar_id* を指定する場合、 ID は、 CREATE FUNCTION ステートメントの実行時に存在していなければなりません。ただし、関数を SQL ステートメントで使用する時点で、メソッド ID は存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラー (SQLSTATE 42724) になります。



単一引用符内に、余分なブランクを使用することはできません。

jar_id
jar の集合をデータベースへインストールしたときに、その jar の集合に付けられた jar ID を指定します。これは、単純 ID またはスキーマ修飾 ID のいずれかにすることができます。たとえば、'myJar' や 'mySchema.myJar' のようになります。

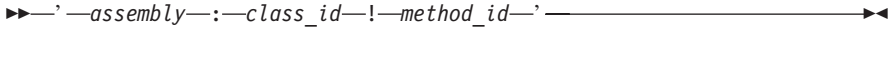
class_id
Java オブジェクトのクラス ID を指定します。クラスがパッケージの一部である場合、クラス ID の部分に完全なパッケージ接頭部 (例: 'myPacks.UserFuncs') が含まれている必要があります。 Java 仮想マシンは、ディレクトリー '.../myPacks/UserFuncs/' の中のクラスを探します。 Windows オペレーティング・システムでは、 Java 仮想マシンはディレクトリー '...¥myPacks¥UserFuncs¥' を探索します。

method_id
呼び出す Java オブジェクトのメソッド名を指定します。

- LANGUAGE CLR の場合:

指定された *string* は、作成する関数を実行するためにデータベース・マネージャーが呼び出す .NET アセンブリー (ライブラリーまたは実行可能モジュール)、そのアセンブリー内のクラス、およびそのクラス内のメソッドを表します。モジュール、クラス、およびメソッドは、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数を SQL ステートメントで使用する時点では、モジュール、クラス、およびメソッドは存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラーが戻されます (SQLSTATE 42724)。

'/clr' コンパイラー・オプションで管理対象コード拡張を指定してコンパイルされている C++ ルーチンは、 'LANGUAGE C' ではなく 'LANGUAGE CLR' としてカタログする必要があります。 DB2 は、必要な実行時の決定を行えるようにするために、 .NET インフラストラクチャーがユーザー定義関数内で使用されていることを認識している必要があります。 .NET インフラストラクチャーを使用するすべてのユーザー定義関数は、 'LANGUAGE CLR' としてカタログする必要があります。



名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

assembly
クラスを含む DLL ファイルまたは他のアセンブリー・ファイルを指定します。ファイル拡張子 (.dll など) まで指定します。絶対パス名を指定しない場合、ファイルは DB2 インストール・パスの関数ディレクトリー (たとえば、 c:¥sqllib¥function) にあるものとされます。

ファイルがインストール関数ディレクトリーのサブディレクトリーにある場合は、絶対パスを指定せずに、ファイル名の前にサブディレクトリーを指定します。たとえば、インストール・ディレクトリーが c:¥sqllib であり、アセンブリー・ファイルが c:¥sqllib¥function¥myprocs¥mydotnet.dll であるなら、アセンブリーの指定は 'myprocs¥mydotnet.dll' とするだけで十分です。このパラメーターの大文字小文字が区別されるかどうかは、ファイル・システムの設定と同じです。

class_id
呼び出すメソッドが属するアセンブリー内のクラスの名前を指定します。クラスがネーム・スペース内にある場合は、クラスだけでなく絶対ネーム・スペースも指定することが必要です。たとえば、クラス EmployeeClass がネーム・スペース MyCompany.ProcedureClasses にあるのであれば、 MyCompany.ProcedureClasses.EmployeeClass をクラスとして指定します。一部の .NET 言語用のコンパイラーはクラスのネーム・スペースとしてプロジェクト名を追加するため、コマンド行コンパイラーと GUI コンパイラーのどちらを使用するかで動作が異なってくるので注意してください。このパラメーターには、大文字と小文字の区別があります。

method_id
指定したクラス内で呼び出されるメソッドを指定します。このパラメーターには、大文字と小文字の区別があります。

- LANGUAGE OLE の場合:

  指定する *string* は、作成中のユーザー定義関数を実行するためにデータベース・マネージャーが呼び出す、 OLE のプログラム ID (progid) またはクラス ID (clsid)、およびメソッド ID です。プログラム ID またはクラス ID、およびメソッド ID は、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数を SQL ステートメントで使用する時点で、メソッド ID は存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラー (SQLSTATE 42724) になります。

  単一引用符内に、余分なブランクを使用することはできません。

  progid
  OLE オブジェクトのプログラム ID を指定します。

  progid は、データベース・マネージャーには解釈されず、ランタイムに OLE API に転送されるだけです。指定する OLE オブジェクトは、作成可能である必要があり、実行時バインディング (ディスパッチに基づくバインディングとも呼ばれる) をサポートしている必要があります。

  clsid
  作成する OLE オブジェクトのクラス ID を指定します。 OLE オブ

ジェクトが *progid* を指定して登録されていない場合に、 progid を指定する代わりに使用することができます。 *clsid* の形式は次のとおりです。

{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}

ここで 'n' は英数字です。 *clsid* は、データベース・マネージャーには解釈されず、ランタイムに OLE API に転送されるだけです。

method_id
呼び出す OLE オブジェクトのメソッド名を指定します。

NAME *identifier*
指定する *identifier* は SQL ID です。 SQL ID は、ストリングの *library-id* として使用されます。区切られた ID でない場合、ID は大文字に変換されます。 ID がスキーマ名で修飾されている場合、スキーマ名の部分は無視されます。この形式の NAME は、LANGUAGE C でのみ使用可能です。

LANGUAGE
この文節は必須で、ユーザー定義関数の本体が準拠している言語インターフェース規則を指定するのに使用します。

C これは、データベース・マネージャーが、ユーザー定義関数を C の関数であるかのように呼び出すことを意味します。ユーザー定義関数は、標準 ANSI C プロトタイプで定義されている C 言語の 呼び出しおよびリンケージの規則に準拠していなければなりません。

JAVA データベース・マネージャーは、 Java クラスのメソッドとしてユーザー定義関数を呼び出します。

CLR データベース・マネージャーは、 .NET クラスのメソッドとしてユーザー定義関数を呼び出します。 LANGUAGE CLRは、Windowsオペーレーティング・システム上で実行するユーザ定義機能のみサポートされます。 NOT FENCED は CLR ルーチンに指定できません (SQLSTATE 42601)。

OLE データベース・マネージャーは、 OLE 自動化オブジェクトによって公開されたメソッドとして、ユーザー定義関数を呼び出します。ユーザー定義関数は、「*OLE Automation Programmer's Reference*」に説明されている、 OLE 自動化データ・タイプと呼び出しメカニズムに準拠している必要があります。

LANGUAGE OLE は、 DB2 (Windows オペレーティング・システム版) で保管されたユーザー定義関数に対してのみサポートされます。 LANGUAGE OLE を指定した UDF には、THREADSAFE は指定できません (SQLSTATE 42613)。

PARAMETER STYLE
この文節は、関数にパラメーターを渡し、関数から値を戻すのに用いる規則を指定するために使用します。

DB2GENERAL
Java クラスのメソッドとして定義された外部関数との間で、パラメーターを渡し、値を戻す場合に用いる規則を指定します。これは、 LANGUAGE JAVA を使用する場合にだけ指定する必要があります。

DB2GENERAL の同義語として値 DB2GENRL が使用可能です。

JAVA
関数は、Java 言語および SQLJ ルーチンの仕様に準拠する、パラメーターの受け渡し規則を使用します。 これは、LANGUAGE JAVA が使用され、パラメーターまたは戻りタイプに構造タイプがない場合にのみ指定できます (SQLSTATE 429B8)。 PARAMETER STYLE JAVA 関数は、 FINAL CALL、SCRATCHPAD または DBINFO 文節をサポートしていません。

SQL
C 言語の呼び出しとリンケージの規則、OLE 自動化オブジェクトによって公開されたメソッド、 または .NET オブジェクトの共用静的メソッドに準拠する規則を、この外部メソッドとの間でパラメーターを渡し、値を戻す場合の規則として指定します。これは、LANGUAGE C、LANGUAGE CLR、または LANGUAGE OLE を使用する場合に指定する必要があります。

PARAMETER CCSID
関数とやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。関数が呼び出されるときのアプリケーション・コード・ページはデータベース・コード・ページです。

UNICODE
ストリング・データが Unicode でエンコードされることを指定します。データベースが Unicode データベースの場合、文字データは UTF-8、GRAPHIC データは UCS-2 になります。データベースが Unicode データベースでない場合は、文字データは UTF-8 になります。いずれの場合も、関数が呼び出されるときのアプリケーション・コード・ページは 1208 です。

データベースが Unicode データベースではないのに、 PARAMETER CCSID UNICODE を指定した関数を作成すると、その関数は GRAPHIC タイプやユーザー定義タイプを取ることができません (SQLSTATE 560C1)。

データベースが Unicode ではなく、データベース構成に代替照合シーケンスが指定されている場合、 PARAMETER CCSID ASCII または PARAMETER CCSID UNICODE を指定した関数を作成できます。関数とやり取りされるすべてのストリング・データは、適切なコード・ページに変換されます。

この文節を LANGUAGE OLE、LANGUAGE JAVA、または LANGUAGE CLR とともに指定することはできません (SQLSTATE 42613)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は任意指定で、特定の引き数の値に対して関数が常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存して関数の結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC を伴う関数は、同じ入力を指定して呼び出しが行われた場合に常に同じ結果を戻します。

NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。乱数を生成する関数は、NOT DETERMINISTIC 関数の例です。入力の平方根を求める関数は、DETERMINISTIC 関数の例です。

FENCED または **NOT FENCED**

この文節は、関数をデータベース・マネージャーの操作環境のプロセスまたはアドレス・スペースで実行しても “安全” かどうかを指定します。

関数が FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を保護して、その関数からアクセスされないようにします。多くの関数は、FENCED または NOT FENCED のどちらかで実行するように選択することができます。一般に、FENCED として実行される関数は、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
適切にコード化、検討、およびテストされていない関数に NOT FENCED を使用すると、 DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ユーザー定義関数が使用される場合には、完全な保全性を確保できません。

LANGUAGE OLE または NOT THREADSAFE を指定した関数には、FENCED のみを指定できます (SQLSTATE 42613)。

関数が FENCED で NO SQL オプションを持っている場合、 AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

ユーザー定義関数を NOT FENCED として登録するには、SYSADM 権限、DBADM 権限、または CREATE_NOT_FENCED_ROUTINE 権限が必要です。

NOT FENCED 文節を指定している場合は、LANGUAGE CLR ユーザー定義関数を作成できません (SQLSTATE 42601)。

THREADSAFE または **NOT THREADSAFE**

関数を他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

関数が OLE 以外の LANGUAGE で定義される場合:

- 関数が THREADSAFE に定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスで関数を呼び出すことができます。一般に、スレッド・セーフにするには、関数はどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED 関数の両方が THREADSAFE になることが可能です。
- 関数が NOT THREADSAFE に定義される場合には、データベース・マネージャーは関数を他のルーチンと同じプロセスに決して呼び出しません。

FENCED 関数の場合、LANGUAGE が JAVAまたは CLR なら THREADSAFE がデフォルトです。これ以外のすべての言語の場合は、NOT THREADSAFE がデフォルトです。関数が LANGUAGE OLE に定義される場合には、THREADSAFE は指定できません (SQLSTATE 42613)。

NOT FENCED 関数の場合には、THREADSAFE がデフォルトです。 NOT THREADSAFE を指定することはできません (SQLSTATE 42613)。

RETURNS NULL ON NULL INPUT または **CALLED ON NULL INPUT**
このオプション文節を使用すると、引き数のいずれかが NULL 値の場合に、外部関数を呼び出さないようにすることができます。ユーザー定義関数がパラメーターなしで定義されている場合、この NULL 引き数条件は引き起こされることはないので、この仕様のコーディング方法はそれほど重要ではなくなります。

RETURNS NULL ON NULL INPUT が指定されており、実行時に関数の引き数のいずれかが NULL 値の場合、このユーザー定義関数は呼び出されず、結果は NULL 値になります。

CALLED ON NULL INPUT が指定されると、引き数が NULL 値か否かに関係なくユーザー定義関数が呼び出されます。これは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。ただし、NULL の引き数値の有無のテストは UDF が行う必要があります。

値 NULL CALL は、後方互換性またはファミリーの互換性のために、CALLED ON NULL INPUT の同義語として使うことができます。同様に、NOT NULL CALL は、RETURNS NULL ON NULL INPUT の同義語として使用できます。

NO SQL、CONTAINS SQL、READS SQL DATA
関数から SQL ステートメントが発行されるかどうかと、もし発行されればどのタイプかを示します。

NO SQL
関数はどの SQL ステートメントも実行できないことを指示します (SQLSTATE 38001)。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、関数で実行できることを指定します (SQLSTATE 38004 または 42985)。どの関数でもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、関数で実行できることを指定します (SQLSTATE 38002 または 42985)。どの関数でもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

STATIC DISPATCH
このオプション文節は、問題解決時に DB2 が関数のパラメーターの静的タイプ (宣言済みタイプ) に基づいて関数を選択するよう指示します。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
このオプションの文節は、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを関数が行うかどうかを指定します。 EXTERNAL ACTION を指定すると、関数に外部の影響がないことを前提とした最適化ができなくなります。 (たとえば、メッセージの送信、警報音による通知、ファイルへのレコードの書き込みなど。)

NO SCRATCHPAD または **SCRATCHPAD** *length*

この文節はオプションであり、この外部関数に対してスクラッチパッドを用意するか否かを指定するのに使用することができます。 (ユーザー定義関数を再入可能にすることを強くお勧めします。再入可能にすると、スクラッチパッドによってある呼び出しと次の呼び出しとの間に関数が “状態を保管する” 手段が用意されます。)

SCRATCHPAD を指定すると、ユーザー定義関数の最初の呼び出し時に、その外部関数によって使用されるスクラッチパッドにメモリーが割り振られます。このスクラッチパッドには、次の特性があります。

- *length* を指定すると、スクラッチパッドのサイズをバイト単位で設定できます。この値は 1 ～ 32 767 で指定できます (SQLSTATE 42820)。デフォルト・サイズは 100 バイトです。
- すべて X'00' に初期化されます。
- その有効範囲は、該当の SQL ステートメントです。 SQL ステートメントでの外部関数に対する参照ごとに 1 つのスクラッチパッドがあります。したがって、以下のステートメントの関数 UDFX が SCRATCHPAD キーワードを指定して定義されている場合、 3 つのスクラッチパッドが割り当てられます。

```
SELECT A, UDFX(A) FROM TABLEB
  WHERE UDFX(A) > 103 OR UDFX(A) < 19
```

  ALLOW PARALLEL が指定されているか、またはデフォルト値として使用された場合、その有効範囲は上記とは異なります。関数が複数のパーティションで実行される場合、関数が処理されるそれぞれのパーティションにおいて、 SQL ステートメントでの関数へのそれぞれの参照ごとにスクラッチパッドが割り当てられます。同様に、パーティション内並列処理をオンにして照会が実行される場合、 3 つ以上のスクラッチパッドが割り当てられることがあります。
- スクラッチパッドは持続します。その内容は、外部関数のある呼び出しから次の呼び出しになっても存続します。外部関数のある呼び出しによってスクラッチパッドに対して行われた変更はいずれも、次の呼び出し時に存続しています。データベース・マネージャーは、各 SQL ステートメントの実行開始時に、スクラッチパッドを初期設定します。各副照会の実行開始時には、データベース・マネージャーによってスクラッチパッドがリセットされます。FINAL CALL オプションが指定されている場合、システムは、スクラッチパッドのリセットに先立って、最終呼び出しを行います。
- これは、外部関数が獲得するシステム・リソース (メモリーなどの) の中央点として使用することもできます。関数は、最初の呼び出しでメモリーを獲得し、そのアドレスをスクラッチパッドに保管して、後の呼び出しでそれを参照することができます。

  (このようにシステム・リソースが獲得される場合、FINAL CALL キーワードも指定する必要があります。これにより、ステートメントの最後で特殊な呼び出しが行われ、外部関数は獲得したシステム・リソースをすべて解放することができます。)

SCRATCHPAD を指定すると、ユーザー定義関数を呼び出すたびに、スクラッチパッドをアドレッシングする外部関数に追加の引き数が渡されます。

NO SCRATCHPAD を指定すると、外部関数に対してスクラッチパッドは割り振られず、渡されません。

SCRATCHPAD は、PARAMETER STYLE JAVA 関数ではサポートされていません。

FINAL CALL または **NO FINAL CALL**

この文節はオプションであり、外部関数に対する最終呼び出しが行われるか否かを指定します。このような最終呼び出しの目的は、外部関数が、獲得したシステム・リソースすべてを解放できるようにすることです。外部関数がメモリーなどのシステム・リソースを獲得し、それをスクラッチパッドに固定するような状況では、これを SCRATCHPAD キーワードと共に使用すると便利です。 FINAL CALL を指定した場合、実行時点で以下が行われます。

- 呼び出しのタイプを指定する追加の引き数が外部関数に渡されます。呼び出しのタイプは次のとおりです。
  - 通常呼び出し。SQL 引き数が渡され、結果が戻されることが予期されます。
  - 最初の呼び出し。この SQL ステートメントのユーザー定義関数に対する参照に対応する外部関数の最初の呼び出し。最初の呼び出しは通常呼び出しです。
  - 最終呼び出し。外部関数がリソースを解放できるようにするための、その関数に対する最終呼び出し。最終呼び出しは、通常呼び出しではありません。この最終呼び出しは、以下の時点で行われます。
    - ステートメント終了時。これは、カーソル指向型のステートメントでカーソルがクローズされた場合、あるいはステートメントが実行を終了した場合に発生します。
    - 並列タスクの終了時。これは、関数が並列タスクで実行された場合に発生します。
    - トランザクション終了時または割り込み時。これは、通常のステートメント終了が発生しなかった場合に発生します。たとえば、何らかの理由で、アプリケーションのロジックが、カーソルをクローズしないようになっている場合があります。このタイプの最終呼び出しの際、CLOSE カーソル以外は、 SQL ステートメントが発行されない可能性があります (SQLSTATE 38505)。このタイプの最終呼び出しは、「呼び出しタイプ」の引き数の特殊値で指示されます。

      WITH HOLD として定義されたカーソルがオープンされている間に、コミット操作が発生すると、それ以降のカーソルのクローズ時、またはアプリケーションの終了時に最終呼び出しが行われます。

  NO FINAL CALL を指定すると、“呼び出しタイプ” の引き数は外部関数に渡されず、最終呼び出しは行われません。

FINAL CALL は、PARAMETER STYLE JAVA 関数ではサポートされていません。

ALLOW PARALLEL または **DISALLOW PARALLEL**

この文節はオプションで、関数への 1 つの参照に対して、関数の呼び出しを並列化できるか否かを指定します。一般に、ほとんどのスカラー関数の呼び出しは

並列化が可能ですが、並列化できない関数 (1 つのスクラッチパッドのコピーに依存する関数など) もあります。スカラー関数に対して ALLOW PARALLEL または DISALLOW PARALLEL を指定すると、 DB2 はその指定を受け入れます。関数にどちらのキーワードが当てはまるかを判別するには、以下の点について検討する必要があります。

- UDF のすべての呼び出しが、互いに完全に独立していますか? YES の場合には、ALLOW PARALLEL を指定します。
- UDF を呼び出すごとに、次の呼び出しに関係する値を提供するスクラッチパッドが更新されますか? (たとえば、カウンターの増分など。) YES の場合には、DISALLOW PARALLEL を指定するか、またはデフォルトを受け入れます。
- 1 つのパーティションでのみ起こる必要のある外部アクションが UDF によって実行されますか? YES の場合には、DISALLOW PARALLEL を指定するか、またはデフォルトを受け入れます。
- コストのかかる初期化処理の実行回数を最小にするためだけに、スクラッチパッドを使用していますか? YES の場合には、ALLOW PARALLEL を指定します。

いずれの場合も、すべての外部関数の本体は、データベースのすべてのパーティションで使用可能なディレクトリーにある必要があります。

ステートメントで以下の 1 つ以上のオプションが指定されている場合以外は、デフォルト値は ALLOW PARALLEL です。

- NOT DETERMINISTIC
- EXTERNAL ACTION
- SCRATCHPAD
- FINAL CALL

これらのオプションのいずれかが指定または暗黙指定されている場合は、デフォルト値は DISALLOW PARALLEL です。

INHERIT SPECIAL REGISTERS

このオプション文節は、関数の更新可能な特殊レジスターが、呼び出しステートメントの環境からの初期値を継承することを指定します。カーソルの 選択ステートメントで呼び出される関数の場合、初期値はカーソルがオープンした際の環境から継承します。ネストされたオブジェクト (たとえば、トリガーまたはビュー) に呼び出されるルーチンの場合、初期値は (オブジェクト定義から継承するのではなく) ランタイム環境から継承します。

特殊レジスターに対する変更が、関数の呼び出し側に戻されることはありません。

更新不能の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、デフォルト値に設定されます。

NO DBINFO または **DBINFO**

この文節はオプションで、 DB2 において既知である特定の情報を追加の呼び出し時引き数として UDF に渡すか (DBINFO)、または渡さないか (NO DBINFO) を指定します。 NO DBINFO がデフォルト値です。 DBINFO は、LANGUAGE OLE ではサポートされません (SQLSTATE 42613)。また、PARAMETER STYLE JAVA でもサポートされません。

DBINFO を指定すると、以下の情報を含む構造が UDF に渡されます。

- データベース名 - 現在接続されているデータベースの名前。
- アプリケーション ID - データベースへの接続ごとに確立された、ユニークなアプリケーション ID。
- アプリケーション許可 ID - アプリケーション・ランタイムの許可 ID。この UDF とアプリケーションとの中間でネストされている UDF は無関係。
- コード・ページ - データベースのコード・ページを識別します。
- スキーマ名 - 表名とまったく同じ条件のもとでは、スキーマの名前が入ります。その他の場合はブランクです。
- 表名 - UDF 参照が UPDATE ステートメントの SET 文節の右側にある場合、または INSERT ステートメントの VALUES リストの項目である場合のいずれかに限り、更新または挿入される表の非修飾名が入ります。その他の場合はブランクです。
- 列名 - 表名とまったく同じ条件で、更新または挿入される列の名前が入ります。その他の場合はブランクです。
- データベースのバージョン/リリース - UDF を呼び出すデータベース・サーバーのバージョン、リリースおよび修正レベルを識別します。
- プラットフォーム - サーバーのプラットフォーム・タイプが入ります。
- 表関数の結果の列番号 - 外部スカラー関数には当てはまりません。

TRANSFORM GROUP *group-name*

関数を呼び出す際のユーザー定義の構造タイプのトランスフォーメーションに使用するトランスフォーム・グループを指定します。関数定義にパラメーターまたは RETURNS データ・タイプとしてユーザー定義の構造タイプが含まれている場合、トランスフォームが必要になります。この文節が指定されない場合には、デフォルトのグループ名 DB2_FUNCTION が使用されます。指定した (またはデフォルトの) *group-name* が、参照された構造タイプに定義されていない場合、エラーになります (SQLSTATE 42741)。指定した group-name または構造タイプに必須の FROM SQL または TO SQL トランスフォーム関数が定義されていない場合には、エラーになります (SQLSTATE 42744)。

トランスフォーム関数は、FROM SQL および TO SQL の両方とも、指定された場合も暗黙的に指定されている場合でも、構造タイプと組み込みタイプ属性とのトランスフォームを適切に実行する SQL 関数でなければなりません。

PREDICATES

述部でこの関数が使用されるときに実行される、フィルター操作や索引拡張の活用を定義します。述部仕様では、検索条件のオプションの SELECTIVITY 文節を指定できます。 PREDICATES 文節が指定された場合、関数は NO EXTERNAL ACTION を指定した DETERMINISTIC として定義しなければなりません (SQLSTATE 42613)。 PREDICATES 文節が指定されており、データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 42613)。

WHEN *comparison-operator*

比較演算子 ("=", "<", ">", ">=", "<=", "<>") を使用した述部での、関数の特定の使用を導入します。

constant
関数の RETURNS タイプに比較可能なデータ・タイプを使用して、定数値を指定します (SQLSTATE 42818)。述部が同じ比較演算子とこの定数でこの関数を使用する場合、指定されたフィルターおよび索引の活用がオプティマイザーにより考慮されます。

EXPRESSION AS *expression-name*
式に名前を提供します。述部が同じ比較演算子と式でこの関数を使用する場合、指定されたフィルターおよび索引の活用が行われます。この式には、式名が割り当てられ、検索関数の引き数として使用できるようになっています。 *expression-name* は、作成されている関数のいずれかの *parameter-name* と同じにすることはできません (SQLSTATE 42711)。式が指定される際に、その式のタイプが識別されます。

FILTER USING
結果表をさらにフィルター操作する際に使用する、外部関数またはケース式の指定を許可します。

function-invocation
結果表の追加のフィルター操作の実行に使用できるフィルター関数を指定します。これは定義された関数のバージョンであり (述部で使用)、ユーザー定義述部で実行される行数を減らし、行を限定するかどうかを判別します。索引により生成される結果が、ユーザー定義述部に期待される結果に近い場合には、フィルター関数を適用する効果はあまりありません。これを指定しない場合は、データのフィルター操作は実行されません。

この関数は、任意の *parameter-name*、*expression-name*、または定数を引き数として使用でき (SQLSTATE 42703)、整数を戻します (SQLSTATE 428E4)。戻り値 1 の場合は行が保持され、その他の場合は破棄されます。

この関数は、以下の要件を満たしていなければなりません。

- LANGUAGE SQL で定義されていないこと (SQLSTATE 429B4)。
- NOT DETERMINISTIC または EXTERNAL ACTION で定義されていないこと (SQLSTATE 42845)。
- いずれかのパラメーターのデータ・タイプとして構造化データ・タイプがないこと (SQLSTATE 428E3)。
- 副照会が含まれていないこと (SQLSTATE 428E4)。

引き数が他の関数またはメソッドを呼び出す場合、このネストされた関数またはメソッドにもこれらの 4 つの規則が課されます。ただし、引き数が組み込みデータ・タイプに評価されるかぎり、システム生成の observer メソッドをフィルター関数 (または、引き数として使用される任意の関数またはメソッド) への引き数として使用することができます。

関数の定義者は、指定されたフィルター関数に対して EXECUTE 特権を持っていなければなりません。

function-invocation 文節は、データベース・コード・ページ内で、65 536 バイト以内の長さでなければなりません (SQLSTATE 22001)。

case-expression
結果表をさらにフィルター操作するためのケース式を指定します。 *searched-when-clause* および *simple-when-clause* では、*parameter-name*、*expression-name*、または定数を使用できます (SQLSTATE 42703)。 FILTER USING *function-invocation* に指定された規則を使って、外部関数を結果式として使用することができます。 *case-expression* で参照される関数またはメソッドはすべて、*function-invocation* にリストされている 4 つの規則にも適合していなければなりません。

case-expression の中では副照会は使用できません (SQLSTATE 428E4)。

ケース式は整数を戻さなければなりません (SQLSTATE 428E4)。結果式で戻り値が 1 の場合は行が保持され、その他の場合は破棄されます。

case-invocation 文節は、データベース・コード・ページ内で、 65 536 バイト以内の長さでなければなりません (SQLSTATE 22001)。

index-exploitation
索引を活用するために使用する索引拡張の検索メソッドによって、規則のセットを定義します。

SEARCH BY INDEX EXTENSION *index-extension-name*
索引拡張を指定します。 *index-extension-name* は、既存の索引拡張を指定する必要があります。

EXACT
述部評価の時に索引検索が厳密に行われることを指定します。索引検索後、オリジナルのユーザー定義の述部関数も、フィルターも適用する必要がないことを DB2 に指示するのに EXACT を使用します。 EXACT 述部は、索引検索が述部と同じ結果を戻す場合に便利です。

EXACT が指定されない場合には、索引検索後、オリジナルのユーザー定義述部が適用されます。索引が類似した述部を提供するのにとどまると思われる場合には、 EXACT オプションは指定しないでください。

索引検索が使用されない場合には、フィルター関数とオリジナルの述部を適用する必要があります。

exploitation-rule
検索ターゲットおよび検索引き数を記述し、さらにこれらを使用して索引拡張で定義した検索メソッドを介して索引検索を実行する方法を記述します。

WHEN KEY (*parameter-name1***)**
検索ターゲットを定義します。 1 つのキーにつき 1 つしか、探索ターゲットを指定できません。 *parameter-name1* 値は、定義された関数のパラメーター名を指定します (SQLSTATE 42703 または 428E8)。

データ・タイプ *parameter-name1* は、索引拡張で指定したソース・キーのデータ・タイプに適合しなければなりません (SQLSTATE 428EY)。この適合は、組み込みおよび特殊データ・タイプで厳密に一致しなければならず、構造タイプの同じタイプ階層内になければなりません。

指定されたパラメーターの値が、指定された索引拡張に基づく索引により網羅される列である場合、この文節は真となります。

USE *search-method-name*(*parameter-name2,...*)
検索引き数を定義します。索引拡張で定義されている検索メソッドから、使用する検索メソッドを指定します。 *search-method-name* は、索引拡張で定義される検索メソッドと適合しなければなりません (SQLSTATE 42743)。 *parameter-name2* 値は、定義された関数のパラメーター名、または EXPRESSION AS 文節の *expression-name* を指定します (SQLSTATE 42703)。これは、検索ターゲットに指定したパラメーター名と異なっていなければなりません (SQLSTATE 428E9)。パラメーターの数と各 *parameter-name2* のデータ・タイプは、索引拡張の検索メソッドに定義されるパラメーターに適合しなければなりません (SQLSTATE 42816)。この適合は、組み込みおよび特殊データ・タイプで厳密に一致しなければならず、構造タイプの同じタイプ階層内になければなりません。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - ASUTIME NO LIMIT
      - NO COLLID
      - PROGRAM TYPE SUB
      - STAY RESIDENT NO
      - Unicode データベースでの CCSID UNICODE
      - PARAMETER CCSID UNICODE が指定されていない場合、非 Unicode データベース内での CCSID ASCII
  - 以前のバージョンの DB2 との互換性:
    - PARAMETER STYLE SQL の代わりに PARAMETER STYLE DB2SQL を指定できます。
    - DETERMINISTIC の代わりに NOT VARIANT を、また NOT DETERMINISTIC の代わりに VARIANT を指定することができます。
    - CALLED ON NULL INPUT の代わりに NULL CALL を、また RETURNS NULL ON NULL INPUT の代わりに NOT NULL CALL を指定できます。
- あるデータ・タイプが他のデータ・タイプにキャスト可能かどうかの判別では、CHAR や DECIMAL などのパラメーター化データ・タイプの長さまたは精度と位取りは考慮されません。したがって、ソース・データ・タイプの値をターゲット・データ・タイプの値にキャストしようとすると、関数の使用時にエラーになる可能性があります。たとえば、VARCHAR は DATE にキャストできますが、実際にはソース・タイプが VARCHAR(5) と定義されている場合には、関数の使用時にエラーになります。
- ユーザー定義関数のパラメーターのデータ・タイプを選択する場合は、入力値に影響を与えるプロモーションの規則を考慮してください (『データ・タイプのプロモーション』 を参照してください)。たとえば、入力値として使用できる定数のデータ・タイプは、予期される以外の組み込みデータ・タイプである可能性があり、さらには、予期されるデータ・タイプにプロモートできない場合があります。プロモーションの規則に従って、一般にパラメーターには次のデータ・タイプを使用するようにしてください。

  - SMALLINT ではなく INTEGER
  - REAL ではなく DOUBLE
  - CHAR ではなく VARCHAR
  - GRAPHIC ではなく VARGRAPHIC
- プラットフォーム間での UDF の移植性を保つためには、以下のデータ・タイプは使用しないようにする必要があります。
  - FLOAT- 代わりに DOUBLE または REAL を使用してください。
  - NUMERIC- 代わりに DECIMAL を使用してください。
  - LONG VARCHAR- 代わりに CLOB (または BLOB) を使用してください。
- 関数とメソッドは、オーバーライド関係になってはなりません (SQLSTATE 42745)。オーバーライドについての詳細は、『CREATE TYPE (構造化)』 を参照してください。
- 関数のシグニチャーは、メソッドのシグニチャーと同じであってはなりません (関数の最初の *parameter-type* と、メソッドの *subject-type* を比較) (SQLSTATE 42723)。
- まだ存在していないスキーマ名を用いて関数を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- パーティション・データベース環境では、外部ユーザー定義関数またはメソッドでの SQL の使用はサポートされていません (SQLSTATE 42997)。
- 索引拡張を定義するには、NO SQL として定義されたルーチンしか使用できません (SQLSTATE 428F8)。
- 関数が SQL を許可する場合、外部プログラムは、フェデレーテッド・オブジェクトへのアクセスを試行してはなりません (SQLSTATE 55047)。
- NOT FENCED として定義される Java ルーチンは、FENCED THREADSAFE として定義されているかのように呼び出されます。
- ***表アクセスの制限***

  関数が READS SQL DATA に定義されている場合には、関数のいかなるステートメントも、関数を呼び出したステートメントによって変更されている表にはアクセスできません (SQLSTATE 57053)。たとえば、 ユーザー定義関数 BONUS() が READS SQL DATA に定義されているとします。ステートメント UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) が呼び出される場合、 BONUS 関数の SQL ステートメントは、EMPLOYEE 表からの読み取りを行えません。
- ***特権***
  - 関数の定義者は、関数に対する WITH GRANT OPTION 付きの EXECUTE 特権と、関数をドロップする権利を常に与えられます。
  - 関数を SQL ステートメントで使用する時点で、関数の定義者はその関数によって使用されるすべてのパッケージに対して EXECUTE 特権を持っていなければなりません。

例:

例 1: Pellow は、自身の PELLOW スキーマに CENTRE 関数を登録します。デフォルト値のあるキーワードはデフォルト値を使い、関数特定名はシステムに生成させることにします。

```
CREATE FUNCTION CENTRE (INT,FLOAT)
  RETURNS FLOAT
  EXTERNAL NAME 'mod!middle'
  LANGUAGE C
  PARAMETER STYLE SQL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
```

例 2: ここで、McBride (DBADM 権限を持つ) が PELLOW スキーマに別の CENTRE 関数を登録し、関数にデータ定義言語でその後使用するための明示的な特定名を付け、すべてのキーワード値を明示的に指定します。また、この関数はスクラッチパッドを使用し、おそらく後続の結果に影響するデータをスクラッチパッドに累積します。 DISALLOW PARALLEL が指定されているので、関数への参照は並列化されず、したがって 1 つのスクラッチパッドを使用して一度限りの初期化と結果の保管が行われます。

```
CREATE FUNCTION PELLOW.CENTRE (FLOAT, FLOAT, FLOAT)
  RETURNS DECIMAL(8,4) CAST FROM FLOAT
  SPECIFIC FOCUS92
  EXTERNAL NAME 'effects!focalpt'
  LANGUAGE C    PARAMETER STYLE SQL
  DETERMINISTIC   FENCED   NOT NULL CALL   NO SQL   NO EXTERNAL ACTION
  SCRATCHPAD   NO FINAL CALL
  DISALLOW PARALLEL
```

例 3: 以下の例は、次の規則をインプリメントするために書かれた、 C 言語のユーザー定義関数プログラムです。

```
output = 2 * input - 4
```

入力が NULL 値の場合には (そしてその場合のみ)、NULL 値を戻します。これは、CREATE FUNCTION ステートメントで NOT NULL CALL を指定することにより、より簡単に (つまり NULL 値チェックを行わずに) 作成することができます。 CREATE FUNCTION ステートメントは、次のとおりです。

```
CREATE FUNCTION ntest1 (SMALLINT)
  RETURNS SMALLINT
  EXTERNAL NAME 'ntest1!nudft1'
  LANGUAGE C    PARAMETER STYLE SQL
  DETERMINISTIC   NOT FENCED   NULL CALL
  NO SQL   NO EXTERNAL ACTION
```

プログラム・コードは、次のとおりです。

```
#include "sqlsystm.h"
/* NUDFT1 IS A USER_DEFINED SCALAR FUNCTION */
/* udft1 accepts smallint input
and produces smallint output
implementing the rule:
if (input is null)
set output = null;
else
set output = 2 * input - 4;
*/
void SQL_API_FN nudft1
(short *input,      /* ptr to input arg */
short *output,     /* ptr to where result goes */
short *input_ind,  /* ptr to input indicator var */
```

```
short *output_ind, /* ptr to output indicator var */
char sqlstate[6],  /* sqlstate, allows for null-term */
char fname[28],    /* fully qual func name, nul-term */
char finst[19],    /* func specific name,  null-term */
char msgtext[71])  /* msg text buffer,     null-term */
{
/* first test for null input */
if (*input_ind == -1)
{
/* input is null, likewise output */
*output_ind = -1;
}
else
{
/* input is not null.  set output to 2*input-4 */
*output = 2 * (*input) - 4;
/* and set out null indicator to zero */
*output_ind = 0;
}
/* signal successful completion by leaving sqlstate as is */
/* and exit */
return;
}
/* end of UDF: NUDFT1 */
```

例 4: 次の例では、ストリングの中で最初に現れる母音の位置を戻す Java UDF を登録します。 UDF は Java で書かれており、fenced して実行されるクラス javaUDFs の findvwl メソッドです。

```
CREATE FUNCTION findv ( CLOB(100K))
  RETURNS INTEGER
  FENCED
  LANGUAGE JAVA
  PARAMETER STYLE JAVA
  EXTERNAL NAME 'javaUDFs.findvwl'
  NO EXTERNAL ACTION
  CALLED ON NULL INPUT
  DETERMINISTIC
  NO SQL
```

例 5: この例では、タイプ SHAPE の 2 つのパラメーター g1 および g2 を入力として取るユーザー定義述部 WITHIN を概説します。

```
CREATE FUNCTION within (g1 SHAPE, g2 SHAPE)
     RETURNS INTEGER
     LANGUAGE C
     PARAMETER STYLE SQL
     NOT VARIANT
     NOT FENCED
     NO SQL
     NO EXTERNAL ACTION
EXTERNAL NAME 'db2sefn!SDESpatilRelations'
     PREDICATES
WHEN = 1
FILTER USING mbrOverlap(g1..xmin, g1..ymin, g1..xmax, g1..max,
g2..xmin, g2..ymin, g2..xmax, g2..ymax)
     SEARCH BY INDEX EXTENSION gridIndex
WHEN KEY(g1) USE withinExplRule(g2)
WHEN KEY(g2) USE withinExplRule(g1)
```

WITHIN 関数の記述は、任意のユーザー定義の関数の記述に類似しているものの、以下を追加することにより、この関数がユーザー定義の述部で使用できることを指定することができます。

- **PREDICATES WHEN = 1** は、DML ステートメントの WHERE 文節でこの関数が

  ```
  within(g1, g2) = 1
  ```

  と表されるときに、述部はユーザー定義の述部として扱われ、索引拡張 *gridIndex* で定義される索引は、この述部に適合する行を検索するのに使用されるように指定します。定数が指定される場合には、DML ステートメントで指定される定数は、索引の作成ステートメントで指定される定数と完全に一致していなければなりません。この条件は、主に、結果タイプが 1 または 0 のいずれかになるブール式に対応するように提供されています。他の場合には、EXPRESSION 文節を選択するとよいでしょう。
- **FILTER USING mbrOverlap** は、フィルター関数 mbrOverlap を参照します。これは、WITHIN 述部の低コスト・バージョンです。上の例では、mbrOverlap 関数は入力として最小の境界長方形を使用し、これらがオーバーラップするかどうかを素早く判別します。 2 つの入力の形の最小の境界長方形がオーバーラップしない場合、g1 が g2 に含まれることはありません。このようにして、タプルを安全に廃棄でき、コストの高い WITHIN 述部のアプリケーションを避けることができます。
- **SEARCH BY INDEX EXTENSION** 文節は、索引拡張と検索ターゲットの組み合わせをこのユーザー定義の述部で使用できることを指定します。

例 6: この例では、タイプ POINT の 2 つのパラメーター P1 および P2 を入力として取るユーザー定義述部 DISTANCE を概説します。

```
CREATE FUNCTION distance (P1 POINT, P2 POINT)
  RETURNS INTEGER
  LANGUAGE C
  PARAMETER STYLE SQL
  NOT VARIANT
  NOT FENCED
  NO SQL
  NO EXTERNAL ACTION
  EXTERNAL NAME 'db2sefn!SDEDistances'
  PREDICATES
  WHEN > EXPRESSION AS distExpr
  SEARCH BY INDEX EXTENSION gridIndex
  WHEN KEY(P1) USE distanceGrRule(P2, distExpr)
  WHEN KEY(P2) USE distanceGrRule(P1, distExpr)
```

DISTANCE 関数の記述は、任意のユーザー定義関数の記述に類似しているものの、以下の追加により、この関数が述部で使用される場合に、この述部がユーザー定義述部であることを指定します。

- **PREDICATES WHEN > EXPRESSION AS distExpr** も、有効な述部指定です。 WHEN 文節で式が指定されると、この述部が DML ステートメントのユーザー定義述部であるかどうかを判別するために、この式の結果タイプが使用されます。たとえば、以下のようになります。

  ```
  SELECT T1.C1
    FROM T1, T2
    WHERE distance (T1.P1, T2.P1) > T2.C2
  ```

  述部指定 distance は、2 つのパラメーターを入力として使用し、タイプ INTEGER の T2.C2 を使用して結果を比較します。 (特定の定数を使用する場合

とは異なり) 式の右辺のデータ・タイプのみ問題となるため、 CREATE FUNCTION DDL にある EXPRESSION 文節を選択して、比較値としてワイルドカードを指定するとよいでしょう。

別の方法として、以下のものも有効なユーザー定義述部です。

```
SELECT T1.C1
  FROM T1, T2
  WHERE distance(T1.P1, T2.P1) > distance (T1.P2, T2.P2)
```

現在のところ、右辺しか式として扱われないという制限があります。左辺の項は、ユーザー定義述部用のユーザー定義関数です。

- **SEARCH BY INDEX EXTENSION** 文節は、索引拡張と検索ターゲットの組み合わせをこのユーザー定義の述部に使用できることを指定します。 distance 関数の場合、distExpr として指定された式も範囲生成関数 (索引拡張の一部として定義) に渡される検索引き数の 1 つです。式の ID は、式の名前を定義するのに使用され、引き数として範囲生成関数に渡されます。

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 440 ページの『CREATE TYPE (構造化)』
- 263 ページの『CREATE FUNCTION (SQL スカラー、表、または行)』
- *SQL リファレンス 第 1 巻* の『ルーチンで使用可能な SQL ステートメント』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』
- *SQL リファレンス 第 1 巻* の『データ・タイプのプロモーション』
- *SQL リファレンス 第 1 巻* の『データ・タイプ間のキャスト』

CREATE FUNCTION (外部表)

このステートメントは、ユーザー定義の外部表関数をアプリケーション・サーバーに登録する場合に使用されます。

表関数 は、SELECT の FROM 文節で使用することができ、行を一度に 1 行戻すことによって、SELECT に表を戻します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限、および以下の少なくとも 1 つ。
  - データベースに対する IMPLICIT_SCHEMA 権限 (関数の暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (関数のスキーマ名が存在する場合)

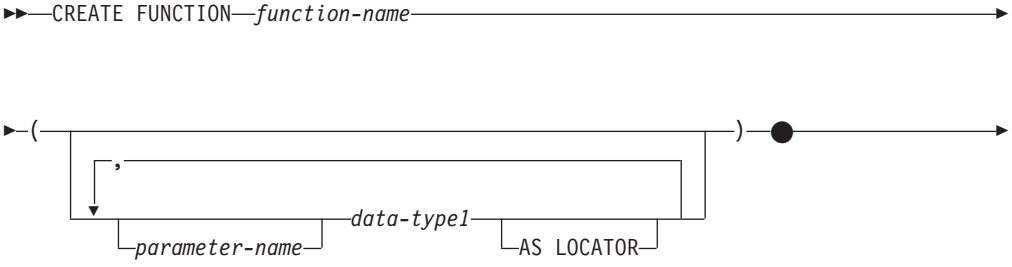
非 fenced の関数を作成するには、ステートメントの許可 ID の特権に、以下の特権の少なくとも 1 つが含まれている必要があります。

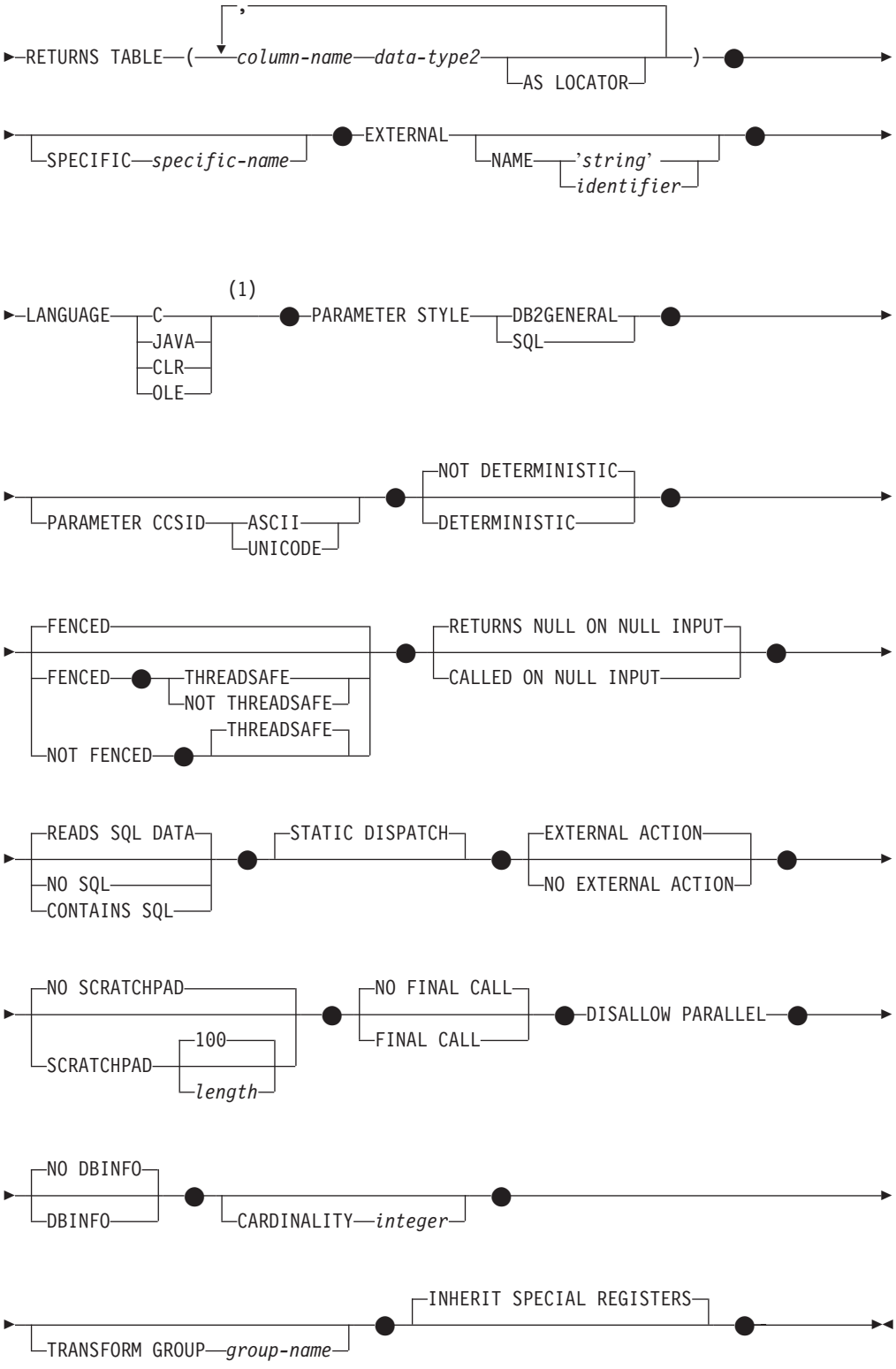
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限
- SYSADM または DBADM 権限

fenced 関数を作成する場合には、さらに別の権限や特権は必要ありません。

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:





注:

1 LANGUAGE OLE DB 外部表関数の作成の詳細は、『CREATE FUNCTION

(OLE DB 外部表)』 を参照してください。 LANGUAGE SQL 表関数の作成の詳細は、 『CREATE FUNCTION (SQL スカラー、表または行)』 を参照してください。

説明:

function-name
定義する関数の名前を指定します。これは、関数を指定する修飾または非修飾の名前です。 *function-name* (関数名) の非修飾形式は SQL ID です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。最初のパラメーターが構造タイプの場合、修飾名は、最初のパラメーターのデータ・タイプと同じであってはなりません。

暗黙または明示の修飾子を含む名前、およびパラメーターの数と各パラメーターのデータ・タイプ (データ・タイプの長さ、精度、または位取りの各属性には関係なく) は、カタログに記述されている関数を指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数およびデータ・タイプとの組み合わせは、そのスキーマ内では当然ユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まる *schema-name* (スキーマ名) は使用できません (SQLSTATE 42939)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *function-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

関数のシグニチャーに何らかの差異があれば、同じ名前を複数の関数に使用することができます。禁止されてはいませんが、外部ユーザー定義表関数の名前として、組み込み関数と同じ名前を指定すべきではありません。

parameter-name
パラメーターのオプショナル名を指定します。これは、この関数の他のパラメーターすべての名前と異なる名前にする必要があります。

(data-type1,...)
関数の入力パラメーターの数を指定するとともに、各パラメーターのデータ・タイプを指定します。このリストには、関数が受け取ることを予期している各パラメーターごとに 1 つの項目を指定する必要があります。パラメーターの数は 90 を超えることはできません。この限界を超えると、エラー (SQLSTATE 54023) になります。

パラメーターのない関数も登録可能です。この場合、指定するデータ・タイプがない場合でも、括弧はコーディングする必要があります。たとえば、以下のようになります。

```
CREATE FUNCTION WOOFER() ...
```

その対応するすべてのパラメーターのタイプがまったく同じである場合でも、 1 つのスキーマ中に名前が同じ 2 つの関数があってはなりません。このタイプの比較では長さ、精度、および位取りは考慮されません。したがって、CHAR(8) と CHAR(35)、また DECIMAL(11,2) と DECIMAL (4,3) は、それぞれ同じタイプと見なされます。 Unicode データベースの場合には、CHAR(13) と GRAPHIC(8) は、それぞれ同じタイプと見なされます。さらに、DECIMAL と NUMERIC などのように、この目的で複数のタイプが同じタイプとして扱われることがあります。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

たとえば、以下のステートメントの場合、

```
CREATE FUNCTION PART (INT, CHAR(15)) ...
CREATE FUNCTION PART (INTEGER, CHAR(40)) ...

CREATE FUNCTION ANGLE (DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (DEC(10,7)) ...
```

2 番目と 4 番目のステートメントは、重複する関数と見なされ、エラーになります。

data-type1
パラメーターのデータ・タイプを指定します。

- CREATE TABLE ステートメントの *data-type* の定義で指定が可能で、関数の作成に使用されている言語において対応するものがある SQL データ・タイプ仕様と省略形を指定できます。
- DECIMAL (および NUMERIC) は、LANGUAGE C と OLE では無効です (SQLSTATE 42815)。
- CLR は 28 より大きい DECIMAL スケールをサポートしていません (SQLSTATE 42613)。
- REF(*type-name*) は、パラメーターのデータ・タイプとして指定できます。ただし、パラメーターに有効範囲を指定してはなりません (SQLSTATE 42997)。
- 適切なトランスフォーム関数が、関連するトランスフォーム・グループに存在する場合には、構造タイプを指定できます。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターを UDF に渡すことを指定します。これにより、UDF に渡すバイト数を大幅に減らすことができ、パフォーマンスも向上します (特に、UDF にとって実際に必要になる値が数バイトだけである場合)。

以下の例は、パラメーター定義の AS LOCATOR 文節の使用法を示しています。

```
CREATE FUNCTION foo ( CLOB(10M) AS LOCATOR, IMAGE AS LOCATOR)
  ...
```

ここで、IMAGE は LOB タイプの 1 つに基づく特殊タイプであると想定します。

また、引き数のプロモーション目的には、AS LOCATOR 文節の効果はないことに注意してください。この例では、タイプはそれぞれ CLOB と IMAGE であると見なされるので、関数に CHAR 引き数または VARCHAR 引き数が最初の引き数として渡されます。同様に、関数シグニチャーに対して AS LOCATOR の効果はありません。関数シグニチャーは、(a) ″関数解決″ と呼ばれるプロセスによって DML で参照された場合、 (b) COMMENT ON や DROP などの DDL ステートメントで参照された場合に関数をマッチングする際に使用されます。実際に、この文節はシグニチャーの指定のない COMMENT ON や DROP で使用しても、しなくても構いません。

LOB または LOB に基づく特殊タイプ以外のタイプに対して AS LOCATOR を指定すると、エラー (SQLSTATE 42601) が発生します。

関数が FENCED で NO SQL オプションを持っている場合、 AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

RETURNS TABLE

関数の出力が表であることを指定します。このキーワードに続く括弧は、表の列の名前とタイプのリストを区切るもので、他の指定 (たとえば、制約) のない単純な CREATE TABLE ステートメントの形式と類似しています。 255 列以内が許可されます (SQLSTATE 54011)。

column-name

この列の名前を指定します。名前を修飾することはできず、表の複数の列に対して同じ名前を使用することはできません。

data-type2

列のデータ・タイプを指定します。構造タイプ以外であれば、特定の言語において、 UDF 作成のパラメーターとしてサポートされるどのようなデータ・タイプでも構いません (SQLSTATE 42997)。

AS LOCATOR

data-type2 が LOB タイプまたは LOB タイプに基づく特殊タイプの場合、このオプションを使用すると、関数は結果表でインスタンス化される LOB 値のロケーターを戻します。

この文節で使用できる有効なタイプについては、 『CREATE FUNCTION (外部スカラー)』 で説明されています。

SPECIFIC *specific-name*

定義する関数のインスタンスに対するユニーク名を指定します。この特定名は、この関数をソース関数として使用する場合、この関数をドロップする場合、またはこの関数にコメントを付ける場合に使用することができます。これは、関数の呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の関数インスタンスを指定するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) になります。

specific-name は、既存の *function-name (関数名)* と同じでも構いません。

修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

EXTERNAL
この文節は、外部プログラミング言語で作成され、文書化されたリンケージの規則とインターフェースに準拠している新しい関数を登録するのに、 CREATE FUNCTION ステートメントが使用されていることを示します。

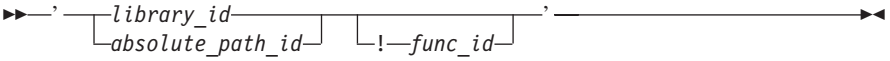
NAME 文節を指定しない場合、″NAME *function-name*″ が想定されます。

NAME *'string'*
この文節は、定義する関数をインプリメントするためのユーザー作成コードを指定します。

'string' オプションは、最大 254 文字のストリング定数です。 ストリングに使用される形式は、指定した LANGUAGE によって異なります。

- LANGUAGE C の場合

  指定する *string* (ストリング) は、作成しているユーザー定義関数を実行するためにデータベース・マネージャーが呼び出すライブラリー名と、そのライブラリー中の関数名です。ライブラリー (およびそのライブラリー中の関数) は、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数が SQL ステートメントで使用される時点では、そのライブラリーとそのライブラリー内の関数が存在していなければならず、しかもデータベース・サーバーのマシンからアクセス可能でなければなりません。



  単一引用符内に、余分なブランクを使用することはできません。

  library_id
  関数を含むライブラリー名を指定します。データベース・マネージャーは、次のようにしてこのライブラリーを特定します。

  – UNIX 系システムの場合、*library_id* が 'myfunc' と指定されており、データベース・マネージャーが /u/production から実行されていると、データベース・マネージャーはライブラリー /u/production/sqllib/function/myfunc で関数を特定します。
  – Windows オペレーティング・システムの場合、データベース・マネージャーは LIBPATH または PATH 環境変数に指定されているディレクトリー・パスから関数を特定します。

  absolute_path_id
  関数を含んでいるファイルの絶対パス名を指定します。

たとえば、UNIX 系システムの場合、'/u/jchui/mylib/myfunc' を指定すると、データベース・マネージャーは /u/jchui/mylib を調べて myfunc 共用ライブラリーを探します。

Windows オペレーティング・システムの場合、'd:¥mylib¥myfunc' を指定すると、データベース・マネージャーは d:¥mylib ディレクトリーからダイナミック・リンク・ライブラリー myfunc.dll をロードします。絶対パス ID がルーチン本体の識別に使用されている場合は、.dll 拡張子を必ず付加してください。

! func_id
呼び出される関数の入り口点名を指定します。 ! は、ライブラリー ID と関数 ID との間の区切り文字です。 *! func_id* を省略すると、データベース・マネージャーはライブラリーのリンク時に確立されたデフォルトの入り口点を使用します。

たとえば、UNIX 系システムで 'mymod!func8' と指定すると、データベース・マネージャーはライブラリー $inst_home_dir/sqllib/function/mymod を調べて、そのライブラリー内の入り口点 func8 を使用します。

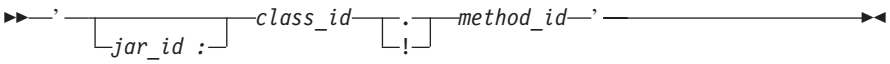
Windows 32 ビット・オペレーティング・システム の場合 'mymod!func8' を指定すると、データベース・マネージャーは mymod.dll ファイルをロードして、そのダイナミック・リンク・ライブラリー (DLL) の func8() 関数を呼び出します。

ストリングの形式が正しくない場合には、エラー (SQLSTATE 42878) になります。

いずれの場合も、すべての外部関数の本体は、データベースのすべてのパーティションで使用可能なディレクトリーにある必要があります。

- LANGUAGE JAVA の場合:

指定する *string* には、作成中のユーザー定義関数を実行するためにデータベース・マネージャーが呼び出す、任意指定の jar ファイル、クラス ID、およびメソッド ID が含まれています。クラス ID とメソッド ID は、 CREATE FUNCTION ステートメントの実行時には存在している必要はありません。 *jar_id* を指定する場合、 ID は、 CREATE FUNCTION ステートメントの実行時に存在していなければなりません。ただし、関数を SQL ステートメントで使用する時点で、メソッド ID は存在しなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。



単一引用符内に、余分なブランクを使用することはできません。

jar_id
jar の集合をデータベースへインストールしたときに、その jar の集合に付けられた jar ID を指定します。これは、単純 ID またはスキーマ修飾 ID のいずれかにすることができます。たとえば、'myJar' や 'mySchema.myJar' のようになります。

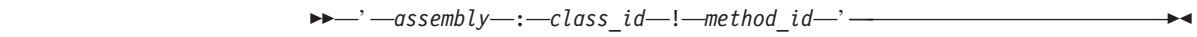
class_id
Java オブジェクトのクラス ID を指定します。クラスがパッケージの一部である場合、クラス ID の部分に完全なパッケージ接頭部 (例: 'myPacks.UserFuncs') が含まれている必要があります。 Java 仮想マシンは、ディレクトリー '.../myPacks/UserFuncs/' の中のクラスを探します。 Windows 32 ビット・オペレーティング・システム では、Java 仮想マシンはディレクトリー '...¥myPacks¥UserFuncs¥' を探索します。

method_id
呼び出す Java オブジェクトのメソッド名を指定します。

- LANGUAGE CLR の場合:

指定された *string* は、作成する関数を実行するためにデータベース・マネージャーが呼び出す .NET アセンブリー (ライブラリーまたは実行可能モジュール)、そのアセンブリー内のクラス、およびそのクラス内のメソッドを表します。モジュール、クラス、およびメソッドは、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数を SQL ステートメントで使用する時点では、モジュール、クラス、およびメソッドは存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラーが戻されます (SQLSTATE 42724)。

'/clr' コンパイラー・オプションで管理対象コード拡張を指定してコンパイルされている C++ ルーチンは、 'LANGUAGE C' ではなく 'LANGUAGE CLR' としてカタログする必要があります。 DB2 は、必要な実行時の決定を行えるようにするために、 .NET インフラストラクチャーがユーザー定義関数内で使用されていることを認識している必要があります。 .NET インフラストラクチャーを使用するすべてのユーザー定義関数は、 'LANGUAGE CLR' としてカタログする必要があります。



名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

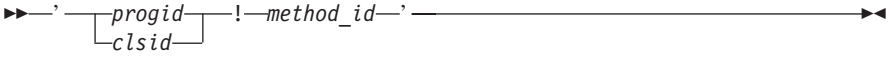
assembly
クラスを含む DLL ファイルまたは他のアセンブリー・ファイルを指定します。ファイル拡張子 (.dll など) まで指定します。絶対パス名を指定しない場合、ファイルは DB2 インストール・パスの関数ディレクトリー (たとえば、 c:¥sqllib¥function) にあるものとされます。ファイルがインストール関数ディレクトリーのサブディレクトリーにある場合は、絶対パスを指定せずに、ファイル名の前にサブディレクトリーを指定します。たとえば、インストール・ディレクトリーが c:¥sqllib であり、アセンブリー・ファイルが c:¥sqllib¥function¥myprocs¥mydotnet.dll であるなら、アセンブリーの指定は 'myprocs¥mydotnet.dll' とするだけで十分です。このパラメーターの大文字小文字が区別されるかどうかは、ファイル・システムの設定と同じです。

class_id
呼び出すメソッドが属するアセンブリー内のクラスの名前を指定します。クラスがネーム・スペース内にある場合は、クラスだけでなく絶対ネーム・スペースも指定することが必要です。たとえば、クラス EmployeeClass がネーム・スペース MyCompany.ProcedureClasses にあるのであれば、 MyCompany.ProcedureClasses.EmployeeClass をクラスとして指定します。一部の .NET 言語用のコンパイラーはクラスのネーム・スペースとしてプロジェクト名を追加するため、コマンド行コンパイラーと GUI コンパイラーのどちらを使用するかで動作が異なってくるので注意してください。このパラメーターには、大文字と小文字の区別があります。

method_id
指定したクラス内で呼び出されるメソッドを指定します。このパラメーターには、大文字と小文字の区別があります。

- LANGUAGE OLE の場合:

指定する *string* は、作成中のユーザー定義関数を実行するためにデータベース・マネージャーが呼び出す、 OLE のプログラム ID (progid) またはクラス ID (clsid)、およびメソッド ID です。プログラム ID またはクラス ID、およびメソッド ID は、 CREATE FUNCTION ステートメントの実行時に存在している必要はありません。ただし、関数を SQL ステートメントで使用する時点で、メソッド ID は存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラー (SQLSTATE 42724) になります。



単一引用符内に、余分なブランクを使用することはできません。

progid
OLE オブジェクトのプログラム ID を指定します。

progid は、データベース・マネージャーには解釈されず、ランタイムに OLE API に転送されるだけです。指定する OLE オブジェクトは、作成可能である必要があり、実行時バインディング (ディスパッチに基づくバインディングとも呼ばれる) をサポートしている必要があります。

clsid
作成する OLE オブジェクトのクラス ID を指定します。 OLE オブジェクトが *progid* を指定して登録されていない場合に、 progid を指定する代わりに使用することができます。 *clsid* の形式は次のとおりです。

{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}

ここで 'n' は英数字です。 *clsid* は、データベース・マネージャーには解釈されず、ランタイムに OLE API に転送されるだけです。

method_id
呼び出す OLE オブジェクトのメソッド名を指定します。

NAME *identifier*

この文節は、定義している関数をインプリメントするユーザー作成コードの名前を指定します。指定する *identifier* は SQL ID です。 SQL ID は、ストリングの *library-id* として使用されます。区切られた ID でない場合、ID は大文字に変換されます。 ID がスキーマ名で修飾されている場合、スキーマ名の部分は無視されます。この形式の NAME は、LANGUAGE C でのみ使用可能です。

LANGUAGE

この文節は必須で、ユーザー定義関数の本体が準拠している言語インターフェース規則を指定するのに使用します。

C これは、データベース・マネージャーが、ユーザー定義関数を C の関数であるかのように呼び出すことを意味します。ユーザー定義関数は、標準 ANSI C プロトタイプで定義されている C 言語の呼び出しおよびリンケージの規則に準拠していなければなりません。

JAVA データベース・マネージャーは、 Java クラスのメソッドとしてユーザー定義関数を呼び出します。

CLR データベース・マネージャーは、 .NET クラスのメソッドとしてユーザー定義関数を呼び出します。 LANGUAGE CLRは、Windowsオペーレーティング・システム上で実行するユーザ定義機能のみサポートされます。 NOT FENCED は CLR ルーチンに指定できません (SQLSTATE 42601)。

OLE データベース・マネージャーは、 OLE 自動化オブジェクトによって公開されたメソッドとして、ユーザー定義関数を呼び出します。ユーザー定義関数は、「*OLE Automation Programmer's Reference*」に説明されている、 OLE 自動化データ・タイプと呼び出しメカニズムに準拠している必要があります。

LANGUAGE OLE は、 DB2 (Windows 32 ビット・オペレーティング・システム版) で保管されたユーザー定義関数に対してのみサポートされます。

LANGUAGE OLE DB 外部表関数の作成の詳細は、『CREATE FUNCTION (OLE DB 外部表)』 を参照してください。

PARAMETER STYLE

この文節は、関数にパラメーターを渡し、関数から値を戻すのに用いる規則を指定するために使用します。

DB2GENERAL

Java クラスのメソッドとして定義された外部関数との間で、パラメーターを渡し、値を戻す場合に用いる規則を指定します。これは、LANGUAGE JAVA を使用する場合にだけ指定する必要があります。

SQL

C 言語の呼び出しとリンケージの規則、OLE 自動化オブジェクトによって公開されたメソッド、 または .NET オブジェクトの共用静的メソッドに準拠する規則を、この外部メソッドとの間でパラメーターを渡し、値を戻す場合の規則として指定します。これは、LANGUAGE C、LANGUAGE CLR、または LANGUAGE OLE を使用する場合に指定する必要があります。

PARAMETER CCSID

関数とやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII

ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、 PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。関数が呼び出されるときのアプリケーション・コード・ページはデータベース・コード・ページです。

UNICODE

ストリング・データが Unicode でエンコードされることを指定します。データベースが Unicode データベースの場合、文字データは UTF-8、GRAPHIC データは UCS-2 になります。データベースが Unicode データベースでない場合は、文字データは UTF-8 になります。いずれの場合も、関数が呼び出されるときのアプリケーション・コード・ページは 1208 です。

データベースが Unicode データベースではないのに、 PARAMETER CCSID UNICODE を指定した関数を作成すると、その関数は GRAPHIC タイプやユーザー定義タイプを取ることができません (SQLSTATE 560C1)。

データベースが Unicode データベースでない場合、表関数を PARAMETER CCSID UNICODE を指定して作成できますが、以下の規則が適用されます。

- 表関数を作成するより前に、代替照合シーケンスをデータベース構成に指定する必要があります (SQLSTATE 56031)。 PARAMETER CCSID UNICODE 表関数は、データベース構成に指定されている代替照合シーケンスと照合されます。
- CCSID ASCII を指定して作成された表または表関数と、 CCSID UNICODE を指定して作成された表または表関数とを、 1 つの SQL ステートメント内で両方とも使用することはできません (SQLSTATE 53090)。このことは、ステートメント内で直接参照されている表および表関数、および間接的に (たとえば、参照保全制約、トリガー、マテリアライズ照会表、およびビューの本体内の表によって) 参照されている表および表関数に適用されます。
- PARAMETER CCSID UNICODE を指定して作成された表関数は、 SQL 関数または SQL メソッド内では参照できません (SQLSTATE 560C0)。
- PARAMETER CCSID UNICODE を指定して作成された表関数を参照する SQL ステートメントは、 SQL 関数または SQL メソッドを呼び出すことができません (SQLSTATE 53090)。
- GRAPHIC タイプおよびユーザー定義タイプは、 PARAMETER CCSID UNICODE 表関数へのパラメーターとしては使用できません (SQLSTATE 560C1)。
- PARAMETER CCSID UNICODE 表関数を参照するステートメントは、 DB2 バージョン 8.1 以降のクライアントからのみ呼び出すことができます (SQLSTATE 42997)。

- SQL ステートメントは常にデータベース・コード・ページで解釈されます。特にこのことは、リテラル、16 進数リテラル、および区切り ID 内のすべての文字がデータベース・コード・ページで表記されていなければならないということを意味します。そうでないと、文字は置換文字によって置き換えられてしまいます。

データベースが Unicode ではなく、データベース構成に代替照合シーケンスが指定されている場合、 PARAMETER CCSID ASCII または PARAMETER CCSID UNICODE を指定した関数を作成できます。関数とやり取りされるすべてのストリング・データは、適切なコード・ページに変換されます。

この文節を LANGUAGE OLE、LANGUAGE JAVA、または LANGUAGE CLR とともに指定することはできません (SQLSTATE 42613)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は任意指定で、特定の引き数の値に対して関数が常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存して関数の結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC 関数は、同一の入力で正しい呼び出しが行われたとき、常に同じ表を返します。 NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。 NOT DETERMINISTIC 表関数の例として、ファイルなどのデータ・ソースからデータを検索する関数があります。

FENCED または **NOT FENCED**
この文節は、関数をデータベース・マネージャーの操作環境のプロセスまたはアドレス・スペースで実行しても「安全」か (NOT FENCED)、そうでないか (FENCED) を指定します。

関数が FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を保護して、その関数からアクセスされないようにします。多くの関数は、FENCED または NOT FENCED のどちらかで実行するように選択することができます。一般に、FENCED として実行される関数は、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
適切にコード化、検討、およびテストされていない関数に NOT FENCED を使用すると、 DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ユーザー定義関数が使用される場合には、完全な保全性を確保できません。

LANGUAGE OLE または NOT THREADSAFE を指定した関数には、FENCED のみを指定できます (SQLSTATE 42613)。

関数が FENCED で NO SQL オプションを持っている場合、 AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

ユーザー定義関数を NOT FENCED として登録するには、SYSADM 権限、DBADM 権限、または CREATE_NOT_FENCED_ROUTINE 権限が必要です。

NOT FENCED 文節を指定している場合は、LANGUAGE CLR ユーザー定義関数を作成できません (SQLSTATE 42601)。

THREADSAFE または **NOT THREADSAFE**
関数を他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

関数が OLE 以外の LANGUAGE で定義される場合:

- 関数が THREADSAFE に定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスで関数を呼び出すことができます。一般に、スレッド・セーフにするには、関数はどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED 関数の両方が THREADSAFE になることが可能です。
- 関数が NOT THREADSAFE に定義される場合には、データベース・マネージャーは関数を他のルーチンと同じプロセスに決して呼び出しません。

FENCED 関数の場合、LANGUAGE が JAVAまたは CLR なら THREADSAFE がデフォルトです。これ以外のすべての言語の場合は、NOT THREADSAFE がデフォルトです。関数が LANGUAGE OLE に定義される場合には、THREADSAFE は指定できません (SQLSTATE 42613)。

NOT FENCED 関数の場合には、THREADSAFE がデフォルトです。 NOT THREADSAFE を指定することはできません (SQLSTATE 42613)。

RETURNS NULL ON NULL INPUT または **CALLED ON NULL INPUT**
このオプション文節を使用すると、引き数のいずれかが NULL 値の場合に、外部関数を呼び出さないようにすることができます。ユーザー定義関数がパラメーターなしで定義されている場合、この NULL 引き数条件は引き起こされることはないので、この仕様のコーディング方法はそれほど重要ではなくなります。

RETURNS NULL ON NULL INPUT が指定されており、表関数 OPEN が実行されるときに、関数の引き数のいずれかが NULL 値の場合、ユーザー定義関数は呼び出されません。試行した表関数スキャンの結果は、空の表 (行のない表) になります。

CALLED ON NULL INPUT が指定されると、引き数が NULL 値か否かに関係なくユーザー定義関数が呼び出されます。これは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。ただし、NULL の引き数値の有無のテストは UDF が行う必要があります。

値 NULL CALL は、後方互換性またはファミリーの互換性のために、CALLED ON NULL INPUT の同義語として使うことができます。同様に、NOT NULL CALL は、RETURNS NULL ON NULL INPUT の同義語として使用できます。

NO SQL、CONTAINS SQL、READS SQL DATA
関数から SQL ステートメントが発行されるかどうかと、もし発行されればどのタイプかを示します。

NO SQL
関数はどの SQL ステートメントも実行できないことを指示します (SQLSTATE 38001)。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、関数で

実行できることを指定します (SQLSTATE 38004 または 42985)。どの関数でもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、関数で実行できることを指定します (SQLSTATE 38002 または 42985)。どの関数でもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

STATIC DISPATCH
このオプション文節は、問題解決時に DB2 が関数のパラメーターの静的タイプ (宣言済みタイプ) に基づいて関数を選択するよう指示します。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
このオプションの文節は、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを関数が行うかどうかを指定します。 EXTERNAL ACTION を指定すると、関数に外部の影響がないことを前提とした最適化ができなくなります。 (たとえば、メッセージの送信、警報音による通知、ファイルへのレコードの書き込みなど。)

NO SCRATCHPAD または **SCRATCHPAD** *length*
この文節はオプションであり、この外部関数に対してスクラッチパッドを用意するか否かを指定するのに使用することができます。 (ユーザー定義関数を再入可能にすることを強くお勧めします。再入可能にすると、スクラッチパッドによってある呼び出しと次の呼び出しとの間に関数が “状態を保管する” 手段が用意されます。)

SCRATCHPAD を指定すると、ユーザー定義関数の最初の呼び出し時に、その外部関数によって使用されるスクラッチパッドにメモリーが割り振られます。このスクラッチパッドには、次の特性があります。

- *length* を指定すると、スクラッチパッドのサイズをバイト単位で設定できます。この値は 1 ～ 32 767 で指定できます (SQLSTATE 42820)。デフォルト値は 100 です。
- すべて X'00' に初期化されます。
- その有効範囲は、該当の SQL ステートメントです。 SQL ステートメントでの外部関数に対する参照ごとに 1 つのスクラッチパッドがあります。したがって、以下のステートメントの UDFX 関数が、SCRATCHPAD キーワードを使用して定義されると、 2 つのスクラッチパッドが割り当てられます。

```
SELECT A.C1, B.C2
  FROM TABLE (UDFX(:hv1)) AS A,
       TABLE (UDFX(:hv1)) AS B
    WHERE ...
```

- スクラッチパッドは持続します。スクラッチパッドは、ステートメントの実行開始時に初期化され、ある呼び出しから次の呼び出しにスクラッチパッドの状態を保存するために、外部表関数で使用することができます。 UDF に FINAL CALL キーワードも指定されている場合、 DB2 がスクラッチパッドを変更することはありません。また、特殊 FINAL 呼び出しがなされると、スクラッチパッドに固定されていたすべてのリソースが解放されます。

NO FINAL CALL が指定またはデフォルト指定されている場合は、 DB2 が OPEN 呼び出しごとにスクラッチパッドを初期化し直すので、外部表関数は CLOSE 呼び出し時に、スクラッチパッドに固定されているすべてのリソースに対して終結処理を行います。 FINAL CALL または NO FINAL CALL の判別、およびスクラッチパッドの関連する動作は、重要な考慮事項です。表関数が副照会または結合で使用されるときは、ステートメントの実行中に複数の OPEN 呼び出しが生じ得るので、特に重要です。

- これは、外部関数が獲得するシステム・リソース (メモリーなどの) の中央点として使用することもできます。関数は、最初の呼び出しでメモリーを獲得し、そのアドレスをスクラッチパッドに保管して、後の呼び出しでそれを参照することができます。

(上で概説したように、FINAL CALL/NO FINAL CALL キーワードは、スクラッチパッドの再初期化を制御するために使用され、スクラッチパッドに固定されているリソースを外部表関数が解放する時期を指示します。)

SCRATCHPAD を指定すると、ユーザー定義関数を呼び出すたびに、スクラッチパッドをアドレッシングする外部関数に追加の引き数が渡されます。

NO SCRATCHPAD を指定すると、外部関数に対してスクラッチパッドは割り振られず、渡されません。

FINAL CALL または **NO FINAL CALL**

この文節はオプションであり、外部関数に対する最終呼び出し (および別個の最初の呼び出し) が行われるか否かを指定します。この文節は、スクラッチパッドが再初期化される時期も制御します。 NO FINAL CALL が指定されている場合は、DB2 はオープン、取り出しおよびクローズの 3 つのタイプの表関数の呼び出ししか行うことができません。しかし、FINAL CALL が指定されている場合は、オープン、取り出しおよびクローズに加えて、表関数に対して最初の呼び出しと最終呼び出しを行うことができます。

外部表関数の場合、どのオプションが選択されたかにかかわらず、呼び出しタイプ引き数は常に存在します。

割り込みかトランザクションの終了のために最終呼び出しが行われると、 UDF は CLOSE カーソル以外の SQL ステートメントを発行できません (SQLSTATE 38505)。こうした最終呼び出しの状況の場合には、「呼び出しタイプ」の引き数に特殊値が渡されます。

DISALLOW PARALLEL

この文節は、関数への単一の参照に対して、関数の呼び出しを並列化できないことを指定します。表関数は常に 1 つのパーティションで実行されます。

NO DBINFO または **DBINFO**

この文節はオプションで、 DB2 において既知である特定の情報を追加の呼び出し時引き数として UDF に渡すか (DBINFO)、または渡さないか (NO DBINFO) を指定します。 NO DBINFO がデフォルト値です。 DBINFO は、LANGUAGE OLE ではサポートされません (SQLSTATE 42613)。

DBINFO を指定すると、以下の情報を含む構造が UDF に渡されます。

- データベース名 - 現在接続されているデータベースの名前。

- アプリケーション ID - データベースへの接続ごとに確立された、ユニークなアプリケーション ID。
- アプリケーション許可 ID - アプリケーション・ランタイムの許可 ID。この UDF とアプリケーションとの中間でネストされている UDF は無関係。
- コード・ページ - データベースのコード・ページを識別します。
- スキーマ名 - 外部表関数には適用されません。
- 表名 - 外部表関数には適用されません。
- 列名 - 外部表関数には適用されません。
- データベースのバージョン/リリース - UDF を呼び出すデータベース・サーバーのバージョン、リリースおよび修正レベルを識別します。
- プラットフォーム - サーバーのプラットフォーム・タイプが入ります。
- 表関数の結果の列番号 - この関数を参照する特定のステートメントに実際に必要な、表関数の結果の列番号の配列。表関数の場合に限り、すべての列の値でなく必要な列の値だけを戻すことによって、 UDF を最適化することを可能にします。

CARDINALITY *integer*

この文節はオプションで、関数によって戻されると予想される行数の見積もりを最適化のために指定します。 *integer* の値の有効範囲は、 0 ～ 9 223 372 036 854 775 807 (両端の値を含む) です。

表関数に対して CARDINALITY 文節の指定がない場合、 DB2 はデフォルト値として有限の値を想定します (RUNSTATS ユーティリティーが統計を収集していない表に対して想定される値と同じ)。

警告: 関数が事実上無限のカーディナリティーを持っている (すなわち、呼び出されるといつでも行を戻し、″end-of-table″ 条件を戻さない) 場合、″end-of-table″ 条件を必要とする照会は無限に実行されるので、照会を中断させる必要があります。このような照会の例としては、GROUP BY 文節や ORDER BY 文節を含む照会があります。このような UDF を作成することは推奨されていません。

TRANSFORM GROUP *group-name*

関数を呼び出す際のユーザー定義の構造タイプのトランスフォーメーションに使用するトランスフォーム・グループを指定します。関数定義にパラメーター・データ・タイプとしてユーザー定義の構造タイプが含まれている場合、トランスフォームが必要になります。この文節が指定されない場合には、デフォルトのグループ名 DB2_FUNCTION が使用されます。指定した (またはデフォルトの) *group-name* が、参照された構造タイプに定義されていない場合、エラーになります (SQLSTATE 42741)。指定した group-name または構造タイプに必須の FROM SQL 変換関数が定義されていない場合には、エラーになります (SQLSTATE 42744)。

INHERIT SPECIAL REGISTERS

このオプション文節は、関数の更新可能な特殊レジスターが、呼び出しステートメントの環境からの初期値を継承することを指定します。カーソルの 選択ステートメントで呼び出される関数の場合、初期値はカーソルがオープンした際の環境から継承します。ネストされたオブジェクト (たとえば、トリガーまたはビュー) に呼び出されるルーチンの場合、初期値は (オブジェクト定義から継承するのではなく) ランタイム環境から継承します。

特殊レジスターに対する変更が、関数の呼び出し側に戻されることはありません。

更新不能の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、デフォルト値に設定されます。

注:

- ユーザー定義関数のパラメーターのデータ・タイプを選択する場合は、入力値に影響を与えるプロモーションの規則を考慮してください。たとえば、入力値として使用できる定数のデータ・タイプは、予期される以外の組み込みデータ・タイプである可能性があり、さらには、予期されるデータ・タイプにプロモートできない場合があります。プロモーションの規則に従って、一般にパラメーターには次のデータ・タイプを使用するようにしてください。
  - SMALLINT ではなく INTEGER
  - REAL ではなく DOUBLE
  - CHAR ではなく VARCHAR
  - GRAPHIC ではなく VARGRAPHIC
- プラットフォーム間での UDF の移植性を保つためには、以下のデータ・タイプの使用をお勧めします。
  - FLOAT ではなく DOUBLE または REAL
  - NUMERIC ではなく DECIMAL
  - LONG VARCHAR ではなく CLOB (または BLOB)
- まだ存在していないスキーマ名を用いて関数を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- パーティション・データベース環境では、外部ユーザー定義関数またはメソッドでの SQL の使用はサポートされていません (SQLSTATE 42997)。
- 索引拡張を定義するには、NO SQL として定義されたルーチンしか使用できません (SQLSTATE 428F8)。
- 関数が SQL を許可する場合、外部プログラムは、フェデレーテッド・オブジェクトへのアクセスを試行してはなりません (SQLSTATE 55047)。
- NOT FENCED として定義される Java ルーチンは、FENCED THREADSAFE として定義されているかのように呼び出されます。
- ***表アクセスの制限***

  関数が READS SQL DATA に定義されている場合には、関数のいかなるステートメントも、関数を呼び出したステートメントによって変更されている表にはアクセスできません (SQLSTATE 57053)。たとえば、 ユーザー定義関数 BONUS() が READS SQL DATA に定義されているとします。ステートメント UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) が呼び出される場合、 BONUS 関数の SQL ステートメントは、EMPLOYEE 表からの読み取りを行えません。
- ***互換性***
  - DB2 for z/OS および OS/390 との互換性:

- 以下の構文はデフォルトの振る舞いとして受け入れられます。
  - ASUTIME NO LIMIT
  - NO COLLID
  - PROGRAM TYPE SUB
  - STAY RESIDENT NO
  - Unicode データベースでの CCSID UNICODE
  - PARAMETER CCSID UNICODE が指定されていない場合、非 Unicode データベース内での CCSID ASCII

– 以前のバージョンの DB2 との互換性:
- PARAMETER STYLE SQL の代わりに PARAMETER STYLE DB2SQL を指定できます。
- DETERMINISTIC の代わりに NOT VARIANT を指定できます。
- NOT DETERMINISTIC の代わりに VARIANT を指定できます。
- CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
- RETURNS NULL ON NULL INPUT の代わりに NOT NULL CALL を指定できます。
- DB2GENERAL の代わりに DB2GENRL を指定できます。

• ***特権***
– 関数の定義者は、関数に対する WITH GRANT OPTION 付きの EXECUTE 特権と、関数をドロップする権利を常に与えられます。
– 関数を SQL ステートメントで使用する時点で、関数の定義者はその関数によって使用されるすべてのパッケージに対して EXECUTE 特権を持っていなければなりません。

例:

例 1: 以下の例では、テキスト管理システムにおいて既知の各文書の 1 つの文書 ID 列からなる行を戻す表関数を登録しています。最初のパラメーターは指定された対象領域をマッチングし、 2 番目パラメーターには指定されたストリングが入ります。

単一セッションのコンテキスト内では UDF は常に同じ表を戻すため、 UDF は DETERMINISTIC として定義されています。 DOCMATCH からの出力を定義する RETURNS 文節に注意してください。それぞれの表関数に対して、FINAL CALL を指定する必要があります。さらに、この表関数は並列して実行できないので、DISALLOW PARALLEL キーワードが追加されています。 DOCMATCH の出力のサイズは大きく変動しますが、 DB2 オプティマイザーにとって有用な CARDINALITY 20 が代表値として指定されています。

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16))
  EXTERNAL NAME '/common/docfuncs/rajiv/udfmatch'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  NOT FENCED
```

```
  SCRATCHPAD
  FINAL CALL
  DISALLOW PARALLEL
  CARDINALITY 20
```

例 2: 以下の例では、 Microsoft Exchange のメッセージのメッセージ・ヘッダー情報と、部分的なメッセージ・テキストの検索に使用する OLE 表関数を登録しています。

```
CREATE FUNCTION MAIL()
  RETURNS TABLE (TIMERECEIVED DATE,
                 SUBJECT VARCHAR(15),
                 SIZE INTEGER,
                 TEXT VARCHAR(30))
  EXTERNAL NAME 'tfmail.header!list'
  LANGUAGE OLE
  PARAMETER STYLE SQL
  NOT DETERMINISTIC
  FENCED
  CALLED ON NULL INPUT
  SCRATCHPAD
  FINAL CALL
  NO SQL
  EXTERNAL ACTION
  DISALLOW PARALLEL
```

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 245 ページの『CREATE FUNCTION (OLE DB 外部表)』
- 263 ページの『CREATE FUNCTION (SQL スカラー、表、または行)』
- 200 ページの『CREATE FUNCTION (外部スカラー)』
- *SQL リファレンス 第 1 巻* の『ルーチンで使用可能な SQL ステートメント』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』
- *SQL リファレンス 第 1 巻* の『データ・タイプのプロモーション』

CREATE FUNCTION (OLE DB 外部表)

このステートメントは、OLE DB Provider からデータをアクセスするための、ユーザー定義の OLE DB 外部表関数をアプリケーション・サーバーに登録する場合に使用します。

表関数 は、SELECT の FROM 文節で使用できます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

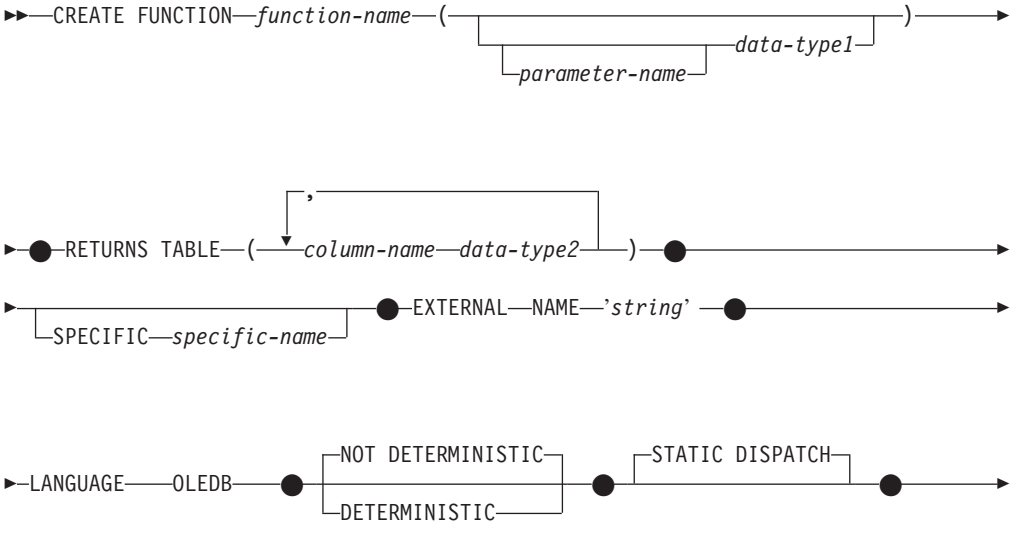
許可:

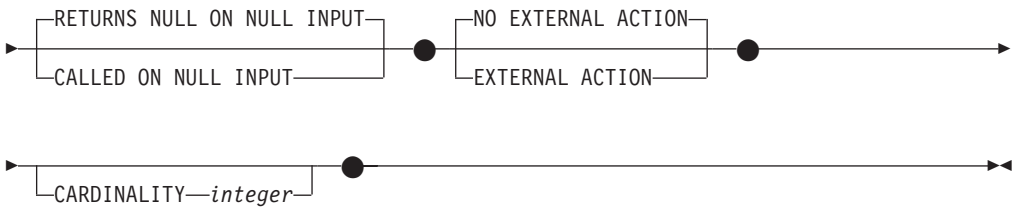
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限、および以下の少なくとも 1 つ。
  - データベースに対する IMPLICIT_SCHEMA 権限 (関数の暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (関数のスキーマ名が存在する場合)

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:





説明:

function-name
定義する関数の名前を指定します。これは、関数を指定する修飾または非修飾の名前です。 *function-name* (関数名) の非修飾形式は SQL ID です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

暗黙または明示の修飾子を含む名前、およびパラメーターの数と各パラメーターのデータ・タイプ (データ・タイプの長さ、精度、または位取りの各属性には関係なく) は、カタログに記述されている関数を指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数およびデータ・タイプとの組み合わせは、そのスキーマ内では当然ユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まる *schema-name* (スキーマ名) は使用できません (SQLSTATE 42939)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *function-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

関数のシグニチャーに何らかの差異があれば、同じ名前を複数の関数に使用することができます。禁止されてはいませんが、外部ユーザー定義表関数の名前として、組み込み関数と同じ名前を指定すべきではありません。

parameter-name
パラメーターにオプションの名前を指定します。

data-type1
関数の入力パラメーターを指定するとともに、そのパラメーターのデータ・タイプを指定します。入力パラメーターを指定しないと、データは、外部ソースから取り出されます (多くの場合、照会最適化によってサブセット化されます)。入力パラメーターは、任意の文字または GRAPHIC ストリング・データ・タイプにすることができ、コマンド・テキストを OLE DB Provider に渡します。

パラメーターのない関数も登録可能です。この場合、指定するデータ・タイプがない場合でも、括弧はコーディングする必要があります。たとえば、以下のようになります。

```
CREATE FUNCTION WOOFER() ...
```

その対応するすべてのパラメーターのタイプがまったく同じである場合でも、 1 つのスキーマ中に名前が同じ 2 つの関数があってはなりません。このタイプの比較では、長さは考慮されません。したがって、CHAR(8) と CHAR(35) は、それぞれ同じタイプと見なされます。 Unicode データベースの場合には、CHAR(13) と GRAPHIC(8) は、それぞれ同じタイプと見なされます。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

RETURNS TABLE

関数の出力が表であることを指定します。このキーワードに続く括弧は、表の列の名前とタイプのリストを区切るもので、他の指定 (たとえば、制約) のない単純な CREATE TABLE ステートメントの形式と類似しています。

column-name

列の名前を指定します。これは、対応する rowset の列名と同じでなければなりません。名前を修飾することはできず、表の複数の列に対して同じ名前を使用することはできません。

data-type2

列のデータ・タイプを指定します。

SPECIFIC *specific-name*

定義する関数のインスタンスに対するユニーク名を指定します。この特定名は、この関数をソース関数として使用する場合、この関数をドロップする場合、またはこの関数にコメントを付ける場合に使用することができます。これは、関数の呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の関数インスタンスを指定するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) になります。

specific-name は、既存の *function-name (関数名)* と同じでも構いません。

修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

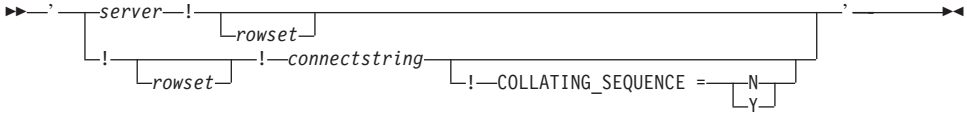
specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

EXTERNAL NAME *'string'*

この文節は、外部表と OLE DB Provider を指定します。

'string' オプションは、最大 254 文字のストリング定数です。

指定されるストリングは、OLE DB Provider との接続およびセッションを確立し、 rowset からデータを取り出すときに使われます。 OLE DB Provider とデータ・ソースは、 CREATE FUNCTION ステートメントの実行時には存在している必要はありません。



server
『CREATE SERVER』 で定義されているように、データ・ソースのローカル名を指定します。

rowset
OLE DB Provider によって公開された rowset (表) を指定します。カタログまたはスキーマ名をサポートする OLE DB Provider の、完全修飾表名を指定する必要があります。

connectstring
データ・ソースへ接続するときに必要な、初期化プロパティーのストリング・バージョン。接続ストリングの基本形式は、ODBC 接続ストリングに基づいています。このストリングには、セミコロンで区切られた、一連のキーワード/値の対が含まれています。等号 (=) により、各キーワードとその値を区切ります。キーワードは、OLE DB 初期化プロパティー (プロパティー・セット DBPROPSET_DBINIT) の記述か、プロバイダー固有のキーワードです。

COLLATING_SEQUENCE
データ・ソースが、DB2 Universal Database と同じ照合順序を使うかどうかを指定します。詳細については、『CREATE SERVER』 を参照してください。有効な値は、以下のとおりです。

Y = 同じ照合順序

N = 異なる照合順序

COLLATING_SEQUENCE を指定しない場合、データ・ソースと DB2 Universal Database の照合順序は異なるものと見なされます。

server を指定する場合、外部名として *connectstring* または COLLATING_SEQUENCE を使うことはできません。それらは、サーバー・オプション CONNECTSTRING および COLLATING_SEQUENCE として定義されています。 *server* を指定しないのであれば、*connectstring* を指定する必要があります。 *rowset* を指定しないのであれば、表関数には、コマンド・テキストを OLE DB Provider に渡すための入力パラメーターが必要です。

LANGUAGE OLEDB
これを指定すると、データベース・マネージャーは、組み込まれた汎用 OLE DB の消費者情報を配置し、OLE DB Provider からデータを取り出します。開発者側で表関数をインプリメントする必要はありません。

LANGUAGE OLEDB 表関数は、任意のプラットフォームで作成できますが、Microsoft OLE DB によってサポートされているプラットフォーム上でのみ実行できます。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は任意指定で、特定の引き数の値に対して関数が常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存して関数の結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC 関数は、同一の入力で正しい呼び出しが行われたとき、常に同じ表を返します。 NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。

STATIC DISPATCH
このオプション文節は、問題解決時に DB2 が関数のパラメーターの静的タイプ (宣言済みタイプ) に基づいて関数を選択するよう指示します。

RETURNS NULL ON NULL INPUT または **CALLED ON NULL INPUT**
このオプション文節を使用すると、引き数のいずれかが NULL 値の場合に、外部関数を呼び出さないようにすることができます。ユーザー定義関数がパラメーターなしで定義される場合、この NULL 引き数条件が引き起こされることはありません。

RETURNS NULL ON NULL INPUT が指定されており、実行時に関数の引き数のいずれかが NULL 値の場合、ユーザー定義関数は呼び出されず、結果は空の表、すなわち行のない表になります。

CALLED ON NULL INPUT が指定されると、引き数が NULL 値か否かに関係なく実行時にユーザー定義関数が呼び出されます。関数の論理によって、空の表を戻すことも戻さないこともあります。ただし、NULL の引き数値の有無のテストは UDF が行う必要があります。

値 NULL CALL は、後方互換性またはファミリーの互換性のために、CALLED ON NULL INPUT の同義語として使うことができます。同様に、NOT NULL CALL は、RETURNS NULL ON NULL INPUT の同義語として使用できます。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
このオプションの文節は、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを関数が行うかどうかを指定します。 EXTERNAL ACTION を指定すると、関数に外部の影響がないことを前提とした最適化ができなくなります。 (たとえば、メッセージの送信、警報音による通知、ファイルへのレコードの書き込みなど。)

CARDINALITY *integer*
この文節はオプションで、関数によって戻されると予想される行数の見積もりを最適化のために指定します。 *integer* の値の有効範囲は、 0 ～ 2 147 483 647 (両端の値を含む) です。

表関数に対して CARDINALITY 文節の指定がない場合、 DB2 はデフォルト値として有限の値を想定します (RUNSTATS ユーティリティーが統計を収集していない表に対して想定される値と同じ)。

警告: 関数が事実上無限のカーディナリティーを持っている (すなわち、呼び出されるといつでも行を戻し、″end-of-table″ 条件を戻さない) 場合、″end-of-table″ 条件を必要とする照会は無限に実行されるので、照会を中断させる必要があります。このような照会の例としては、GROUP BY 文節や ORDER BY 文節を含む照会があります。このような UDF を作成することは推奨されていません。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DETERMINISTIC の代わりに NOT VARIANT を指定できます。
    - NOT DETERMINISTIC の代わりに VARIANT を指定できます。

  - CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
  - RETURNS NULL ON NULL INPUT の代わりに NOT NULL CALL を指定できます。

- FENCED、 FINAL CALL、 SCRATCHPAD、 PARAMETER STYLE SQL、 DISALLOW PARALLEL、 NO DBINFO、NOT THREADSAFE、および NO SQL は、ステートメントでは暗黙的であり、指定できます。
- ユーザー定義関数のパラメーターのデータ・タイプを選択する場合は、入力値に影響を与えるプロモーションの規則を考慮してください。たとえば、入力値として使用できる定数のデータ・タイプは、予期される以外の組み込みデータ・タイプである可能性があり、さらには、予期されるデータ・タイプにプロモートできない場合があります。プロモーションの規則に従って、一般にパラメーターには次のデータ・タイプを使用するようにしてください。
  – CHAR ではなく VARCHAR
  – GRAPHIC ではなく VARGRAPHIC
- プラットフォーム間での UDF の移植性を保つためには、以下のデータ・タイプの使用をお勧めします。
  – FLOAT ではなく DOUBLE または REAL
  – NUMERIC ではなく DECIMAL
  – LONG VARCHAR ではなく CLOB (または BLOB)
- まだ存在していないスキーマ名を用いて関数を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- ***特権***

  関数の定義者は、関数に対する WITH GRANT OPTION 付きの EXECUTE 特権と、関数をドロップする権利を常に与えられます。

例:

例 1: 次の例では、OLE DB 表関数を登録し、 Microsoft Access データベースから受注情報を取り出します。外部名として接続ストリングが定義されています。

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER,
                 customerid CHAR(5),
                 employeeid INTEGER,
                 orderdate TIMESTAMP,
                 requireddate TIMESTAMP,
                 shippeddate TIMESTAMP,
                 shipvia INTEGER,
                 freight dec(19,4))
  LANGUAGE OLEDB
  EXTERNAL NAME '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
                           Data Source=c:¥sqllib¥samples¥oledb¥nwind.mdb
  !COLLATING_SEQUENCE=Y';
```

例 2: 次の例では、OLE DB 表関数を登録し、 Oracle データベースから顧客情報を取り出します。接続ストリングは、サーバー定義で指定されています。外部名では、表名は完全に修飾されたものです。ローカル・ユーザーである john が、リモート・ユーザーの dave にマップされます。他のユーザーは、接続ストリングでゲスト・ユーザー ID を使用します。

```
CREATE SERVER spirit
 WRAPPER OLEDB
 OPTIONS (CONNECTSTRING 'Provider=MSDAORA;Persist Security Info=False;
                         User ID=guest;password=pwd;Locale Identifier=1033;
                         OLE DB Services=CLIENTCURSOR;Data Source=spirit');

CREATE USER MAPPING FOR john
  SERVER spirit
  OPTIONS (REMOTE_AUTHID 'dave', REMOTE_PASSWORD 'mypwd');

CREATE FUNCTION customers ()
  RETURNS TABLE (customer_id INTEGER,
                 name  VARCHAR(20),
                 address VARCHAR(20),
                 city VARCHAR(20),
                 state VARCHAR(5),
                 zip_code INTEGER)
  LANGUAGE OLEDB
  EXTERNAL NAME 'spirit!demo.customer';
```

例 3: 次の例では、OLE DB 表関数を登録し、 MS SQL Server 7.0 データベースから店舗についての情報を取り出します。外部名として接続ストリングが指定されています。表関数には、コマンド・テキストを OLE DB Provider に渡すための入力パラメーターがあります。外部名として rowset 名を指定する必要はありません。照会例では、SQL ステートメント・テキストを渡し、上位 3 店舗についての情報を取り出します。

```
CREATE FUNCTION favorites (varchar(600))
 RETURNS TABLE (store_id CHAR (4),
                name  VARCHAR (41),
                sales INTEGER)
 SPECIFIC favorites
 LANGUAGE OLEDB
 EXTERNAL NAME '!!Provider=SQLOLEDB.1;Persist Security Info=False;
                User ID=sa;Initial Catalog=pubs;Data Source=WALTZ;
                Locale Identifier=1033;Use Procedure for Prepare=1;
                Auto Translate=False;Packet Size=4096;Workstation ID=WALTZ;
                OLE DB Services=CLIENTCURSOR;';

SELECT *
  FROM TABLE (favorites
             (' select top 3 sales.stor_id as store_id, ' ||
                ' stores.stor_name as name, ' ||
                ' sum(sales. qty) as sales ' ||
              ' from sales, stores ' ||
              ' where sales.stor_id = stores.stor_id ' ||
              ' group by sales.stor_id, stores.stor_name ' ||
              ' order by sum(sales.qty) desc ')) as f;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 342 ページの『CREATE SERVER』
- 472 ページの『CREATE USER MAPPING』
- 489 ページの『CREATE WRAPPER』
- 226 ページの『CREATE FUNCTION (外部表)』
- *SQL リファレンス 第 1 巻* の『データ・タイプのプロモーション』

CREATE FUNCTION (ソースまたはテンプレート)

このステートメントは、以下の目的で使用されます。

- 他の既存のスカラー関数または列関数に基づくユーザー定義関数を、アプリケーション・サーバーに登録する。
- フェデレーテッド・サーバーとして指定されたアプリケーション・サーバーに、関数テンプレートを登録する。*関数テンプレート* とは、実行可能コードを含まない部分関数のことです。ユーザーは、データ・ソース関数へマッピングする目的でこれを作成します。マッピングを作成したら、フェデレーテッド・サーバーへサブミットする照会に、その関数テンプレートを指定できます。そのような照会を処理する場合、フェデレーテッド・サーバーは、テンプレートのマップ先のデータ・ソース関数を呼び出し、値を戻します。この値のデータ・タイプは、テンプレートの定義の RETURNS 部分にある値に対応するものです。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

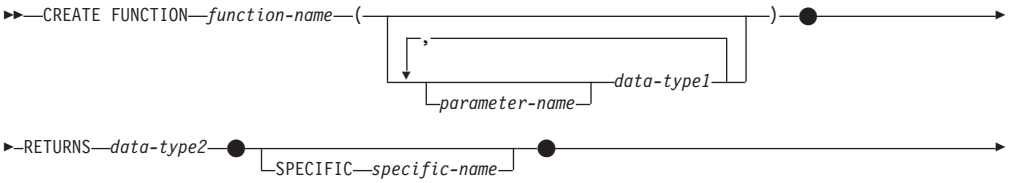
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

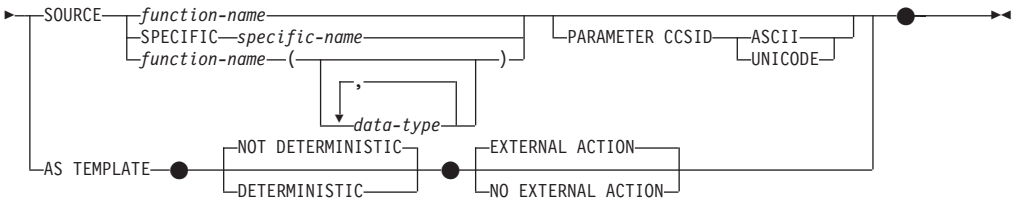
- データベースに対する IMPLICIT_SCHEMA 特権 (関数の暗黙または明示のスキーマ名が存在しない場合)
- スキーマに対する CREATEIN 特権 (関数のスキーマ名が存在する場合)
- SYSADM または DBADM 権限

このステートメントの許可 ID が持つ特権に SYSADM または DBADM 権限が含まれておらず、かつ SOURCE 文節が指定されている場合、ステートメントの許可 ID が保持する特権にはソース関数に対する EXECUTE 特権も含まれている必要があります。

許可 ID の権限が不十分で操作を実行できない場合には、エラーが戻されます (SQLSTATE 42502)。

構文:





説明:

function-name

定義する関数または関数テンプレートを指定します。これは、関数を指定する修飾または非修飾の名前です。 *function-name* (関数名) の非修飾形式は SQL ID です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

暗黙または明示の修飾子を含む名前、およびパラメーターの数と各パラメーターのデータ・タイプ (データ・タイプの長さ、精度、または位取りの各属性には関係なく) は、カタログに記述されている関数または関数テンプレートを指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数およびデータ・タイプとの組み合わせは、そのスキーマ内では当然ユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まる *schema-name* (スキーマ名) は使用できません (SQLSTATE 42939)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *function-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

ユーザー定義特殊タイプと同じ機能をサポートする目的で、既存の関数に基づくユーザー定義関数に名前を付ける場合は、元の関数と同じ名前を使用することができます。これにより、ユーザーは、追加定義の必要性を特に意識することなく、同じ関数のユーザー定義特殊タイプ版を使用することができます。一般に、関数のシグニチャーに何らかの差異がある場合には、同じ名前を複数の関数に使用することができます。

(data-type,...)

関数または関数テンプレートの入力パラメーターの数を指定するとともに、各パラメーターのデータ・タイプを指定します。このリストには、関数または関数テンプレートが受け取ることを予期している各パラメーターごとに、 1 つの項目を指定する必要があります。パラメーターの数は 90 を超えることはできません。この限界を超えると、エラー (SQLSTATE 54023) になります。

パラメーターのない関数または関数テンプレートも登録可能です。この場合、指定するデータ・タイプがない場合でも、括弧はコーディングする必要があります。たとえば、

```
CREATE FUNCTION WOOFER() ...
```

その対応するすべてのパラメーターのタイプがまったく同じである場合でも、 1 つのスキーマ中に名前が同じ 2 つの関数または関数テンプレートがあってはなりません。 (この制限は、同じ名前を持つスキーマ内の関数または関数テンプレートにも適用されます。) このタイプの比較では長さ、精度、および位取りは考慮されません。したがって、CHAR(8) と CHAR(35)、また DECIMAL(11,2) と DECIMAL (4,3) は、それぞれ同じタイプと見なされます。 Unicode データベースの場合には、CHAR(13) と GRAPHIC(8) は、それぞれ同じタイプと見なされます。さらに、DECIMAL と NUMERIC などのように、この目的で複数のタイプが同じタイプとして扱われることがあります。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

たとえば、以下のステートメントの場合、

```
CREATE FUNCTION PART (INT, CHAR(15)) ...
CREATE FUNCTION PART (INTEGER, CHAR(40)) ...

CREATE FUNCTION ANGLE (DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (DEC(10,7)) ...
```

2 番目と 4 番目のステートメントは、重複する関数と見なされ、エラーになります。

parameter-name
パラメーターのオプショナル名を指定します。これは、この関数の他のパラメーターすべての名前と異なる名前にする必要があります。

data-type1
パラメーターのデータ・タイプを指定します。

ソース・スカラー関数では、SOURCE 文節で指定された関数の対応するパラメーターのタイプにキャスト可能であれば、任意の有効な SQL データ・タイプを使用できます。 REF(*type-name*) データ・タイプをパラメーターのデータ・タイプとして指定することはできません (SQLSTATE 42997)。

関数がソース関数から派生するので、パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません (指定は可能です)。その代わりに、CHAR() のように空の括弧を使用できます。*パラメーター化データ・タイプ* は、特定の長さ、位取り、または精度によって定義可能なデータ・タイプです。パラメーター化データ・タイプは、ストリング・データ・タイプと 10 進データ・タイプです。

関数テンプレートを使用すると、パラメーター化データ・タイプに長さ、精度、または位取りを指定する代わりに、空の括弧も使用できます。パラメーター化データ・タイプには空の括弧を使用することが推奨されています。空の括弧を使用する場合、長さ、精度、または位取りはリモート関数のものと同じです。それらは関数マッピング作成時に関数テンプレートがリモート関数にマップされるときに決定されます。括弧をすべて省略した場合は、データ・タイプのデフォルト長が使用されます (CREATE TABLE ステートメントの説明を参照)。

RETURNS
この文節は必須で、関数または関数テンプレートの出力を指定します。

data-type2
出力のデータ・タイプを指定します。

ソース・スカラー関数では、ソース関数の結果のタイプからキャスト可能であれば、任意の有効な SQL データ・タイプ (特殊タイプも同様) を指定できます。

上記のような、ソース関数のパラメーターの場合、パラメーター化タイプのパラメーターを指定する必要はありません。その代わりに、VARCHAR() のように、空の括弧を使用できます。

関数が他の関数に基づいている場合に RETURNS 文節のデータ・タイプの指定に適用される考慮事項と規則については、このステートメントの『規則』セクションを参照してください。

関数テンプレートを使用する場合は、空の括弧を使用できません (SQLSTATE 42611)。パラメーター化データ・タイプには、長さ、精度、または位取りを指定する必要があります。リモート関数と同じ長さ、精度、または位取りを指定することが推奨されています。

SPECIFIC *specific-name*

定義する関数のインスタンスに対するユニーク名を指定します。この特定名は、この関数をソース関数として使用する場合、この関数をドロップする場合、またはこの関数にコメントを付ける場合に使用することができます。これは、関数の呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の関数インスタンスを指定するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) が戻されます。

specific-name は、既存の *function-name (関数名)* と同じでも構いません。

修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) が戻されます。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

SOURCE

作成する関数が、データベース・マネージャーにとって既知の他の関数 (ソース関数) によってインプリメントされる関数であることを指定します。ソース関数は、組み込み関数 (COALESCE、 DBPARTITIONNUM、 NULLIF、 HASHEDVALUE、 TYPE_ID、 TYPE_NAME、 TYPE_SCHEMA、または VALUE を除く) か、以前に作成したユーザー定義のスカラー関数のいずれかになります。

SOURCE 文節は、スカラー関数または列関数に対してのみ指定が可能で、表関数には指定できません。

SOURCE 文節によって、他の関数と同一の機能を持たせることが可能になります。

function-name

ソースとして使用される特定の関数を指定します。ステートメントの許可 ID が EXECUTE 特権を持ち、この *function-name* を名前とする特定の関数

が、スキーマに実際に 1 つだけ存在している場合にのみ有効です。この構文変形は、組み込み関数であるソース関数に対しては無効です。

非修飾名を指定すると、現行 SQL パス (CURRENT PATH 特殊レジスターの値) がその関数を見つけるのに使用されます。 EXECUTE 特権のあるステートメントの許可 ID の名前を持つ関数が含まれている、関数パスの最初のスキーマが選択されます。

指定されたスキーマにこの名前の関数が見つからないか、あるいは名前が修飾されておらず、この名前の関数が関数パスにない場合は、エラー (SQLSTATE 42704) が戻されます。指定したスキーマまたは見つかったスキーマに、この関数の許可された特定インスタンスが複数個ある場合には、エラー (SQLSTATE 42725) が戻されます。その名前の関数が存在し、ステートメントの許可 ID が関数に対して EXECUTE 特権を持っていない場合には、エラー (SQLSTATE 42501) が戻されます。

SPECIFIC *specific-name*
関数の作成時に指定されたか、またはデフォルト値として使用された *specific-name* (特定名) を使用して、ソースとして使用する特定のユーザー定義関数を指定します。この構文変形は、組み込み関数であるソース関数に対しては無効です。

非修飾名を指定すると、現行 SQL パスがその関数を見つけるのに使用されます。 EXECUTE 特権のあるステートメントの許可 ID の特定の名前を持つ関数が含まれている、関数パスの最初のスキーマが選択されます。

指定されたスキーマにこの *specific-name* の関数が見つからないか、あるいは名前が修飾されておらず、この *specific-name* の関数が SQL パスにない場合は、エラー (SQLSTATE 42704) が戻されます。 *specific-name* の関数が存在し、ステートメントの許可 ID が関数に対して EXECUTE 特権を持っていない場合には、エラー (SQLSTATE 42501) が出されます。

function-name (data-type,...)
ソース関数を固有に指定する関数シグニチャーを指定します。組み込み関数であるソース関数に対しては、これが唯一有効な構文変形です。

同じ関数名と SOURCE 文節に指定されたデータ・タイプを持つ複数の関数の中から 1 つの関数を選択するために、関数解決の規則が適用されます。ただし、選択された関数の各パラメーターのデータ・タイプは、ソース関数に指定された対応するデータ・タイプと、まったく同じタイプでなければなりません。

function-name
ソース関数の関数名を指定します。非修飾名を指定すると、ユーザーの SQL パスのスキーマが考慮されます。

data-type
これは、CREATE FUNCTION ステートメントで対応する位置 (コンマで区切られた) に指定されたデータ・タイプに一致していなければなりません。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。代わりに、空の括弧をコーディングすることによって、データ・タイプの一致を調べる際にそれらの属性を無視するように

指定することができます。たとえば、DECIMAL() は、データ・タイプが DECIMAL(7,2) として定義されているパラメーターと一致します。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントにおける指定に完全に一致していなければなりません。これは、意図した通りの関数が確実に使用されるようにする場合に便利です。また、データ・タイプの同義語は一致と見なされることにも注意してください (たとえば、 DEC と NUMERIC は一致します)。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つ関数がない場合は、エラー (SQLSTATE 42883) が戻されます。

PARAMETER CCSID
関数とやり取りされるすべてのストリング・データに使用されるコード化スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。関数が呼び出されるときのアプリケーション・コード・ページはデータベース・コード・ページです。

UNICODE
ストリング・データが Unicode でエンコードされることを指定します。データベースが Unicode データベースの場合、文字データは UTF-8、GRAPHIC データは UCS-2 になります。データベースが Unicode データベースでない場合は、文字データは UTF-8 になります。いずれの場合も、関数が呼び出されるときのアプリケーション・コード・ページは 1208 です。

PARAMETER CCSID 文節には、ソース関数と同じエンコーディング・スキームを指定することが必要です (SQLSTATE 53090)。

AS TEMPLATE
このステートメントが、実行可能コードを含む関数ではなく、関数テンプレートを作成するために使われることを示しています。

NOT DETERMINISTIC または **DETERMINISTIC**
関数が同一の入力引き数に同じ結果を戻すかどうかを指定します。デフォルトは NOT DETERMINISTIC です。

NOT DETERMINISTIC
同じ入力引き数を指定して呼び出しても、関数が同じ結果を戻さない場合があることを指定します。関数は、結果に影響する状態値によって左右されます。データベース・マネージャーは、SQL ステートメントの最適化中に、この情報を使用します。 deterministic ではない (常には同じ結果が戻されない) 関数の例としては、乱数を生成する関数があります。

deterministic ではない関数は、並列タスクによって実行されると、不正確な結果を受け取る場合があります。

DETERMINISTIC
同じ入力引き数を指定して呼び出すと、関数から常に同じ結果が戻されることを指定します。データベース・マネージャーは、SQL ステートメントの最適化中に、この情報を使用します。 deterministic な (常に同じ結果を戻す) 関数の例としては、入力引き数の平方根を計算する関数があります。

SQL ルーチンの本体は、暗黙的または明示的な DETERMINISTIC または NOT DETERMINISTIC の指定と整合している必要があります。DETERMINISTIC と定義された関数は NOT DETERMINISTIC と定義された別の関数を呼び出すことはできず、特殊レジスターを参照することもできません (SQLSTATE 428C2)。たとえば、RETURN ステートメント内で RAND 組み込み関数を呼び出す SQL 関数は deterministic ではない関数として作成されていることが必要です。

EXTERNAL ACTION または **NO EXTERNAL ACTION**
関数が、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを取るかどうかを指定します。外部アクションの例としては、メッセージの送信やファイルへのレコードの書き込みがあります。デフォルトは EXTERNAL ACTION です。

EXTERNAL ACTION
関数が、データベース・マネージャーによって管理されていないオブジェクトの状態を変更できることを指定します。

外部アクションが指定された関数は、並列タスクによって実行されると、不正確な結果を受け取る場合があります。たとえば、ある関数から外部アクションが指定された関数への初期呼び出しごとに注釈が送信される場合、関数に対して 1 つの注釈が送信されるのではなく、並列タスクごとに 1 つの注釈が送信されます。

NO EXTERNAL ACTION
関数が、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを取らないことを指定します。データベース・マネージャーは、SQL ステートメントの最適化中に、この情報を使用します。

SQL ルーチン本体が EXTERNAL ACTION を指定して定義されている関数を呼び出す場合は、 EXTERNAL ACTION を暗黙的または明示的に指定することが必要です (SQLSTATE 428C2)。

規則:

- 便宜上、この項では作成する関数を CF と呼び、 SOURCE 文節で指定する関数を SF と呼びます (許される 3 つの構文のどれが SF の指定に使用されたかは関係ありません)。
  - CF と SF のそれぞれの非修飾名が異なる名前であっても構いません。
  - 他の関数のソースとして指定された関数自体が、別の関数をソースとして使用した関数であっても構いません。間接的に呼び出された関数がエラーを戻すと、アプリケーションをデバッグすることが極めて難しくなるので、この機能を使用する場合には細心の注意を払う必要があります。
  - 以下の文節は、SOURCE 文節と共に指定した場合には無効になります (CF はこれらの属性を SF から継承するからです)。
    - CAST FROM ...
    - EXTERNAL ...
    - LANGUAGE ...
    - PARAMETER STYLE ...
    - DETERMINISTIC/NOT DETERMINISTIC
    - FENCED/NOT FENCED
    - RETURNS NULL ON NULL INPUT/CALLED ON NULL INPUT
    - EXTERNAL ACTION/NO EXTERNAL ACTION
    - NO SQL/CONTAINS SQL/READS SQL DATA
    - SCRATCHPAD/NO SCRATCHPAD
    - FINAL CALL/NO FINAL CALL
    - RETURNS TABLE (...)
    - CARDINALITY ...
    - ALLOW PARALLEL/DISALLOW PARALLEL
    - DBINFO/NO DBINFO
    - THREADSAFE/NOT THREADSAFE
    - INHERIT SPECIAL REGISTERS

  これらの規則に違反すると、エラー (SQLSTATE 42613) になります。
- CF の入力パラメーターの数は、SF のパラメーターの数と同じでなければなりません。異なる場合には、エラー (SQLSTATE 42624) が戻されます。
- 次の場合には、CF にパラメーター化データ・タイプの長さ、精度、または位取りを指定する必要がありません。
  - 関数の入力パラメーター
  - その RETURNS パラメーター

  代わりに、VARCHAR() のように、データ・タイプの一部として空の括弧を使用することにより、長さ、精度、および位取りがソース関数と同じであるか、あるいはキャストによって決定されるように指定することができます。

  ただし、長さ、精度、または位取りを指定した場合には、 CF における値と SF の対応する値とが比較検査されます。これについては、以下で入力パラメーターと戻り値とに分けて説明します。

- CF の入力パラメーターの指定は、SF の入力パラメーターの指定と比較検査されます。 CF の各パラメーターのデータ・タイプは、 SF の対応するパラメーターのデータ・タイプと同じであるか、あるいはキャスト*可能* でなければなりません。同じタイプでないか、あるいはキャスト可能ではないパラメーターがある場合には、エラー (SQLSTATE 42879) が戻されます。

  この規則は、CF の使用時に発生し得るエラーに対して何らかの保証を与えるものではありません。 CF パラメーターのデータ・タイプや長さ、または精度属性に一致する引き数は、対応する SF パラメーターの方が長さが短かったり精度が劣る場合には、割り当てることができません。一般に、CF のパラメーターの長さ属性や精度属性は、対応する SF パラメーターのそれよりも大きくしてはなりません。
- CF の RETURNS データ・タイプの指定は、SF のそれと比較検査されます。キャスト後の SF の最終の RETURNS データ・タイプは、 CF の RETURNS のデータ・タイプと同じか、あるいはそれにキャスト可能でなければなりません。そうでない場合、エラー (SQLSTATE 42866) が戻されます。

  この規則は、CF の使用時に発生し得るエラーに対して何らかの保証を与えるものではありません。 SF の RETURNS データ・タイプのデータ・タイプや長さもしくは精度属性に一致する結果の値は、 CF の RETURNS データ・タイプの方が長さが短かったり精度が劣る場合には、割り当てることができません。長さもしくは精度属性が SF の RETURNS データ・タイプのそれよりも小さい CF の RETURNS データ・タイプを指定することにした場合は、十分に注意を払ってください。

注:

- あるデータ・タイプが他のデータ・タイプにキャスト可能かどうかの判別では、CHAR や DECIMAL などのパラメーター化データ・タイプの長さまたは精度と位取りは考慮されません。したがって、ソース・データ・タイプの値をターゲット・データ・タイプの値にキャストしようとすると、関数の使用時にエラーになる可能性があります。たとえば、VARCHAR は DATE にキャストできますが、実際にはソース・タイプが VARCHAR(5) と定義されている場合には、関数の使用時にエラーになります。
- ユーザー定義関数のパラメーターのデータ・タイプを選択する場合は、入力値に影響を与えるプロモーションの規則を考慮してください (『データ・タイプのプロモーション』 を参照してください)。たとえば、入力値として使用できる定数のデータ・タイプは、予期される以外の組み込みデータ・タイプである可能性があり、さらには、予期されるデータ・タイプにプロモートできない場合があります。プロモーションの規則に従って、一般にパラメーターには次のデータ・タイプを使用するようにしてください。
  - SMALLINT ではなく INTEGER
  - REAL ではなく DOUBLE
  - CHAR ではなく VARCHAR
  - GRAPHIC ではなく VARGRAPHIC
- まだ存在していないスキーマ名を用いて関数を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。

- データ・ソース関数を認識するフェデレーテッド・サーバーの場合、この関数はフェデレーテッド・データベースにあるもう一方の関数にマップする必要があります。データベースに対となる関数がない場合、ユーザーがそれを作成してマッピングしなければなりません。

  対となるもう一方は、関数 (スカラーまたはソース) か、関数テンプレートとすることができます。ユーザーが関数を作成して必要なマッピングを行うと、その関数を指定する照会を処理するたびに、DB2 は、 (1) その関数を呼び出すときの戦略と、データ・ソース関数を呼び出すときの戦略を比べ、 (2) オーバーヘッドが少ないと思われる関数を呼び出します。

  ユーザーが関数テンプレートとマッピングを作成すると、そのテンプレートを指定する照会を処理するたびに、DB2 は、マッピング先のデータ・ソース関数を呼び出します (ただし、この関数を呼び出すためのアクセス・プランが存在する場合)。

- ***特権***

  関数の定義者は、関数に対する EXECUTE 特権と、関数をドロップする権利を常に与えられます。以下のいずれかが真の場合には、関数の定義者には WITH GRANT OPTION も与えられます。

  – ソース関数が、組み込み関数である。
  – 関数の定義者が、ソース関数に対して WITH GRANT OPTION 付きの EXECUTE 特権を持っている。
  – 関数が、テンプレートである。

例:

例 1: Pellow がオリジナルの CENTRE 外部スカラー関数を作成した後に、別のユーザーがその関数に基づいて関数を作成します。この関数は、整数引き数のみを受け入れるという点だけが異なります。

```
CREATE FUNCTION MYCENTRE (INTEGER, INTEGER)
  RETURNS FLOAT
  SOURCE PELLOW.CENTRE (INTEGER, FLOAT)
```

例 2: 組み込みの INTEGER データ・タイプに基づく特殊タイプ HATSIZE が作成済みです。それぞれの部門の平均の hat size を計算するには、AVG 関数を使用すると便利です。これは、以下のようにして簡単に実行できます。

```
CREATE FUNCTION AVG (HATSIZE) RETURNS HATSIZE
  SOURCE SYSIBM.AVG (INTEGER)
```

特殊タイプの作成により必要な cast 関数が生成され、引き数の場合は HATSIZE から INTEGER に、関数の結果の場合は INTEGER から HATSIZE にキャストすることが可能です。

例 3: フェデレーテッド・システムで、表統計を倍精度浮動小数点付きの値で戻す Oracle UDF を起動します。フェデレーテッド・サーバーは、この関数とフェデレーテッド・データベース側の対となる関数との間でマッピングが行われる場合にだけ、この関数を認識することができます。ところが、そのような対となる関数は存在していません。それで、対となる関数を関数テンプレートの形で指定して、このテンプレートを NOVA というスキーマに割り当てることを決定します。次のコードを使用して、テンプレートをフェデレーテッド・サーバーに登録します。

```
CREATE FUNCTION NOVA.STATS (DOUBLE, DOUBLE)
  RETURNS DOUBLE
  AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
```

例 4: フェデレーテッド・システムで、特定の組織の従業員に対して支給されるボーナスの総額を戻す Oracle UDF を呼び出します。フェデレーテッド・サーバーは、この関数とフェデレーテッド・データベース側の対となる関数との間でマッピングが行われる場合にだけ、この関数を認識することができます。ところがそのような対となる関数は存在しないため、ユーザーが関数テンプレートの形で作成します。次のコードを使用して、このテンプレートをフェデレーテッド・サーバーに登録します。

```
CREATE FUNCTION BONUS ()
  RETURNS DECIMAL (8,2)
  AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
```

関連資料:

- *SQL リファレンス 第 1 巻* の『関数』
- *SQL リファレンス 第 1 巻* の『基本述部』
- 272 ページの『CREATE FUNCTION MAPPING』
- *SQL リファレンス 第 1 巻* の『データ・タイプのプロモーション』
- *SQL リファレンス 第 1 巻* の『データ・タイプ間のキャスト』

CREATE FUNCTION (SQL スカラー、表、または行)

このステートメントは、ユーザー定義の SQL スカラー、表、または行関数を定義するのに使用されます。*スカラー関数* は、呼び出されるたびに 1 つの値を戻し、SQL 式が有効な個所であればどこでも有効です。*表関数* は、FROM 文節で使用でき、表を戻します。*行関数* は、トランスフォーム関数として使用でき、行を戻します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- 全選択に指定された表、ビュー、またはニックネームのそれぞれに対して、
  - その表、ビュー、またはニックネームに対する CONTROL 特権、または
  - その表、ビュー、またはニックネームに対する SELECT 特権

  および以下の少なくとも 1 つ
  - データベースに対する IMPLICIT_SCHEMA 権限 (関数の暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (関数のスキーマ名が既存のスキーマを指している場合)

PUBLIC 以外のグループ特権は、 CREATE FUNCTION ステートメントで指定された表やビューに対しては考慮されません。

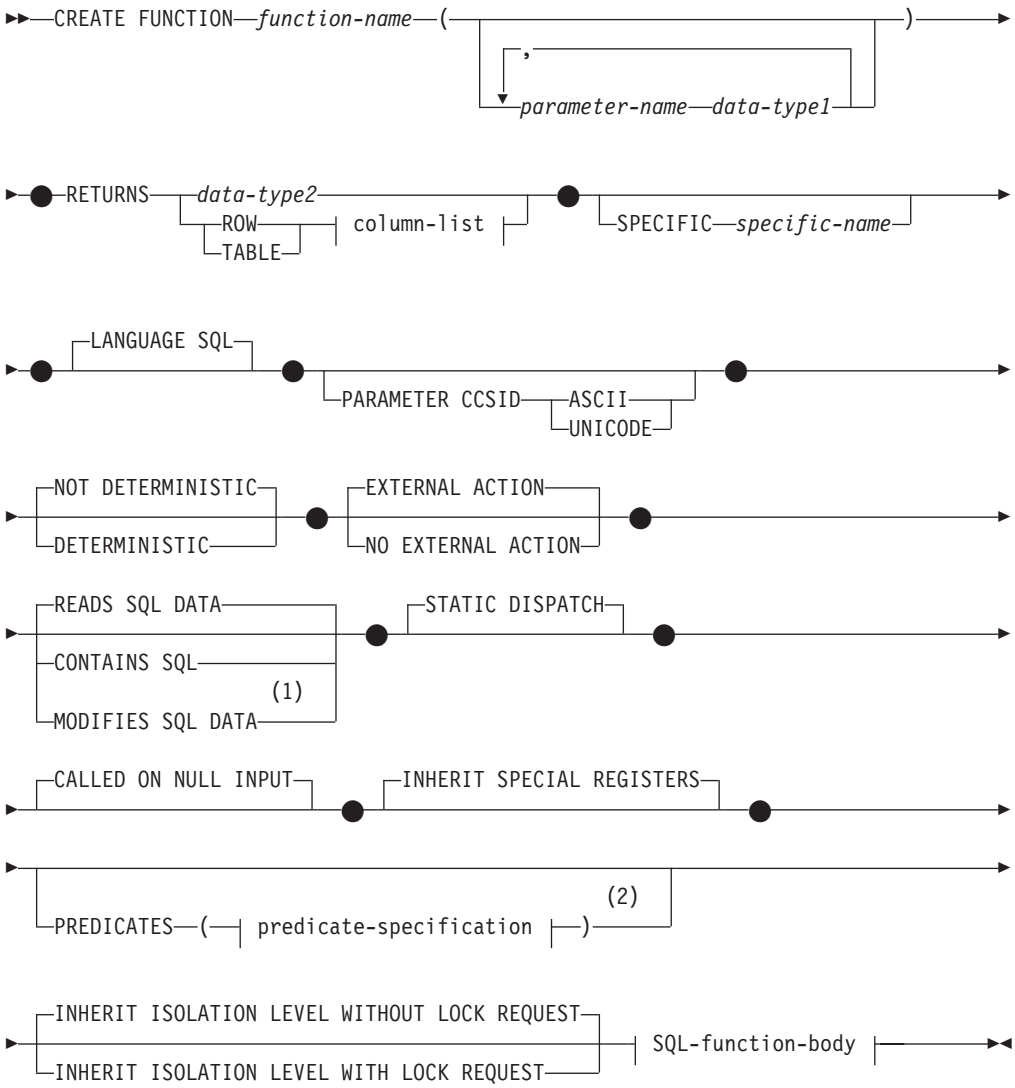
このニックネームで示されている表またはビューのデータ・ソースの許可要件は、関数が呼び出される時に適用されます。接続の許可 ID は、別のリモート許可 ID へマップできます。

関数の定義者が SYSADM 権限を持つために、関数の作成しかできない場合、関数作成のため、その定義者には暗黙的に DBADM 権限が付与されます。

このステートメントの許可 ID に SYSADM 権限または DBADM 権限がない場合には、ステートメントの許可 ID が持つ特権に、関数本体で指定される SQL ステートメントを呼び出すのに必要なすべての特権が含まれていることも必要です。

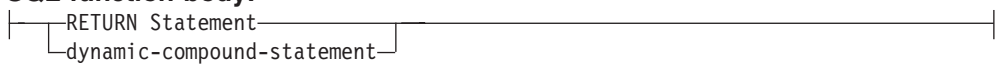
許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:



column-list:

SQL-function-body:



注:

1 RETURNS が表 (TABLE *column-list*) を指定している場合にのみ有効

2 RETURNS がスカラー結果 (*data-type2*) を指定している場合のみ有効です。

説明:

function-name
定義する関数の名前を指定します。これは、関数を指定する修飾または非修飾の名前です。 *function-name* (関数名) の非修飾形式は SQL ID です (最大長

18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

暗黙または明示の修飾子を含む名前、およびパラメーターの数と各パラメーターのデータ・タイプ (データ・タイプの長さ、精度、または位取りの各属性には関係なく) は、カタログに記述されている関数を指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数およびデータ・タイプとの組み合わせは、そのスキーマ内では当然ユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まる *schema-name* (スキーマ名) は使用できません (SQLSTATE 42939)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *function-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

関数のシグニチャーに何らかの差異があれば、同じ名前を複数の関数に使用することができます。禁止されてはいませんが、外部ユーザー定義表関数の名前として、組み込み関数と同じ名前を指定すべきではありません。

parameter-name
この関数の他のすべてのパラメーター名と異なる名前。

data-type1
パラメーターのデータ・タイプを指定します。

- CREATE TABLE ステートメントの *data-type1* の定義で指定可能な SQL データ・タイプ仕様と省略形を指定することができます。
- REF を指定することはできますが、このとき REF の有効範囲は指定できません。システムがパラメーターまたは結果の有効範囲を推測することはしません。関数本体では、最初に参照タイプをキャストして有効範囲を指定してからでなければ、逆参照操作は使用できません。同様に、SQL 関数により戻される参照も、最初にキャストしてこれに有効範囲を指定してからでなければ逆参照操作は使用できません。
- LONG VARCHAR および LONG VARGRAPHIC データ・タイプは使用できません (SQLSTATE 42815)。

RETURNS
これは必須の文節であり、関数の出力のタイプを指定します。

data-type2
出力のデータ・タイプを指定します。

このステートメントでは、前述の関数パラメーター *data-type1* で説明した SQL 関数のパラメーターと同じ考慮事項が適用されます。

ROW *column-list*
関数の出力が単一の行であることを指定します。関数が複数の行を出力する

場合、エラーになります (SQLSTATE 21505)。 *column-list* には、少なくとも 2 つの列を組み込まなければなりません (SQLSTATE 428F0)。

行関数は、構造タイプの変換関数としてのみ使用できます (1 つの構造タイプをパラメーターとして使用し、基本タイプのみ戻します)。

TABLE *column-list*
関数の出力が表であることを指定します。

column-list
ROW または TABLE 関数で戻される列名およびデータ・タイプのリスト。

column-name
この列の名前を指定します。名前を修飾することはできず、行の複数の列に対して同じ名前を使用することはできません。

data-type3
列のデータ・タイプを指定します。 SQL 関数のパラメーターによりサポートされていれば、どのデータ・タイプでも構いません。

SPECIFIC *specific-name*
定義する関数のインスタンスに対するユニーク名を指定します。この特定名は、この関数をソース関数として使用する場合、この関数をドロップする場合、またはこの関数にコメントを付ける場合に使用することができます。これは、関数の呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の関数インスタンスを識別するものであってはなりません。さもないと、エラーになります (SQLSTATE 42710)。

specific-name は、既存の *function-name (関数名)* と同じでも構いません。

修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示または暗黙の修飾子と同じでなければなりません。そうでない場合は、エラーになります (SQLSTATE 42882)。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

LANGUAGE SQL
関数が SQL を使用して書かれていることを指定します。

PARAMETER CCSID
関数とやり取りされるすべてのストリング・データに使用されるコード化スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。

UNICODE
文字データは UTF-8 で記述され、GRAPHIC データは UCS-2 で記述されることを指定します。データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 56031)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は任意指定で、特定の引き数の値に対して関数が常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存して関数の結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC 関数は、同一の入力で正しい呼び出しが行われたとき、常に同じ表を返します。 NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。

関数の本体が特殊レジスターにアクセスしたり、他の非 deterministic 関数を呼び出す場合には、 NOT DETERMINISTIC を明示的または暗黙的に指定しなければなりません (SQLSTATE 428C2)。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
このオプションの文節は、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを関数が行うかどうかを指定します。 NO EXTERNAL ACTION を指定すると、システムは関数が外部に影響を与えないことを前提とした最適化を使用できます。

関数の本体が外部アクションのある別の関数を呼び出す場合、 EXTERNAL ACTION を明示的または暗黙的に指定しなければなりません (SQLSTATE 428C2)。

CONTAINS SQL、**READS SQL DATA**、または **MODIFIES SQL DATA**
どのタイプの SQL ステートメントを実行できるかを指示します。

CONTAINS SQL
SQL データの読み取りも変更もしない SQL ステートメントを、関数により実行できるように指示します (SQLSTATE 42985)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、関数により実行できるように指示します (SQLSTATE 42985)。

MODIFIES SQL DATA
dynamic-compound-statement でサポートされているすべての SQL ステートメントが関数によって実行可能であることを示します。

STATIC DISPATCH
このオプション文節は、問題解決時に DB2 が関数のパラメーターの静的タイプ (宣言済みタイプ) に基づいて関数を選択するよう指示します。

CALLED ON NULL INPUT
この文節は、なんらかの引き数が NULL 値であるかどうかにかかわりなく、関数が呼び出されることを指定します。これは、NULL 値を戻す場合も、NULL 以外の値を戻す場合もあります。 NULL の引き数値の有無のテストはユーザー定義関数が行う必要があります。

CALLED ON NULL INPUT の代わりに NULL CALL という句を使用できます。

INHERIT SPECIAL REGISTERS
このオプション文節は、関数の更新可能な特殊レジスターが、呼び出しステートメントの環境からの初期値を継承するよう指示します。カーソルの 選択ステートメントに呼び出された関数の場合、初期値はカーソルがオープンした際の環境から継承します。ネストされたオブジェクト (たとえば、トリガーまたはビュー) に呼び出されるルーチンの場合、初期値はランタイム環境 (オブジェクト定義ではない) から継承します。

特殊レジスターに対する変更が、関数の呼び出し元に戻されることはありません。

一部の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、呼び出し元からの継承は行われません。

PREDICATES
この関数を使用する述部の場合、この文節は索引拡張を活用できることを示し、オプションの SELECTIVITY 文節を使用して述部の検索条件を指定できます。PREDICATES 文節が指定された場合、関数は NO EXTERNAL ACTION を指定した DETERMINISTIC として定義しなければなりません (SQLSTATE 42613)。 PREDICATES 文節が指定されており、データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 42613)。

predicate-specification
述部指定に関する詳細は、『CREATE FUNCTION (外部スカラー)』 を参照してください。

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST または **INHERIT ISOLATION LEVEL WITH LOCK REQUEST**
関数が呼び出し元のステートメントの分離レベルを継承している場合に、ロック要求をステートメントの分離レベルに関連付けることができるかどうかを指定します。デフォルトは INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST です。

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
関数が呼び出し元のステートメントの分離レベルを継承している場合に、指定した isolation-clause (分離文節) の一部として lock-request-clause (ロック要求文節) が含まれている SQL ステートメントのコンテキストでそのメソッドを呼び出すことができません (SQLSTATE 42601)。

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
関数が呼び出し元のステートメントの分離レベルを継承している場合に、指定した lock-request-clause (ロック要求文節) をメソッドが継承することを指定します。

SQL-function-body
関数の本体を指定します。パラメーター名を SQL-function-body で参照することができます。あいまい参照を避けるため、パラメーター名は関数名で修飾できます。

SQL-function-body が動的コンパウンド・ステートメントであれば、ここには少なくとも 1 つの RETURN ステートメントが組み込まれていなければならず、また関数が呼び出されるときに RETURN ステートメントが実行されなければなりません (SQLSTATE 42632)。関数が表または行関数であれば、RETURN ステ

ートメントは、動的コンパウンド・ステートメントの最後の関数でなければなりません。RETURN ステートメントを 1 つだけ組み込むことができます (SQLSTATE 429BD)。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - Unicode データベースでの CCSID UNICODE
      - 非 Unicode データベースでの CCSID ASCII
  - 以前のバージョンの DB2 との互換性:
    - CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
- 関数本体内での関数呼び出しの解決は、 CREATE FUNCTION ステートメントに対して有効な関数パスに従って実行され、関数が作成された後も変更されません。
- SQL 関数に、何らかの日付または時刻の特殊レジスターへの参照が複数含まれている場合、すべての参照は同じ値を戻します。そして、関数を呼び出したステートメントでのレジスター呼び出しにより戻される値と同じになります。
- SQL 関数の本体には、これ自体または他の関数やこれを呼び出すメソッドに対する再帰呼び出しを組み込むことはできません。そのような関数は、呼び出しの対象として存在できないからです。
- 関数またはメソッドを作成するすべてのステートメントで、以下の規則が課されます。
  - 関数のシグニチャーは、メソッドのシグニチャーと同じであってはなりません (関数の最初の *parameter-type* と、メソッドの *subject-type* を比較)。
  - 関数とメソッドは、オーバーライド関係にあってはなりません。つまり、関数が、最初のパラメーターをサブジェクトとするメソッドである場合、これが他のメソッドをオーバーライドしたり、他のメソッドによりオーバーライドされたりすることはできません。オーバーライド・メソッドについての詳細は、『CREATE TYPE (構造化)』 ステートメントを参照してください。
  - 関数にはオーバーライドが適用されないため、2 つの関数がメソッドである場合に、一方が他方をオーバーライドする状態で存在することは可能です。

  上記の規則で parameter-types を比較する目的で、以下のようになります。
  - パラメーター名、長さ、AS LOCATOR、および FOR BIT DATA は無視されます。
  - サブタイプとそのスーパータイプは異なるものと見なされます。
- ***表アクセスの制限***

  関数が READS SQL DATA に定義されている場合には、関数のいかなるステートメントも、関数を呼び出したステートメントによって変更されている表にはアクセスできません (SQLSTATE 57053)。たとえば、 ユーザー定義関数 BONUS() が READS SQL DATA に定義されているとします。ステートメント UPDATE

EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) が呼び出される場合、 BONUS 関数の SQL ステートメントは、EMPLOYEE 表からの読み取りを行えません。

ネストされた CALL ステートメントが MODIFIES SQL DATA で定義された関数に含まれている場合、この関数によって (関数定義またはこの関数を呼び出すステートメントによって) 変更される表への読み取りアクセスは許可されません (SQLSTATE 57053)。

- ***特権***

  関数の定義者は、関数に対する EXECUTE 特権と、関数をドロップする権利を常に与えられます。関数を定義するのに必要なすべての特権に対して定義者が WITH GRANT OPTION を持っている場合、または定義者が SYSADM か DBADM 権限を持っている場合には、関数の定義者には、関数に対する WITH GRANT OPTION も与えられます。

  関数の定義者にそれらの特権が与えられるのは、それらの特権の派生元の特権が関数の作成時に存在している場合に限ります。定義者は、これらの特権を直接持っているか、または PUBLIC の特権として持っていることが必要です。関数の定義者がメンバーであるグループを持つ特権は考慮されません。関数を使用する場合、接続済みのユーザーの許可 ID には、そのデータ・ソースでニックネームが参照する表またはビューに対する適切な特権がなければなりません。

例:

例 1: 既存のサインおよびコサイン関数を使用して、値のタンジェントを戻すスカラー関数を定義します。

```
CREATE FUNCTION TAN (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SIN(X)/COS(X)
```

例 2: 構造タイプ PERSON のトランスフォーム関数を定義します。

```
CREATE FUNCTION FROMPERSON (P PERSON)
  RETURNS ROW (NAME VARCHAR(10), FIRSTNAME VARCHAR(10))
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN VALUES (P..NAME, P..FIRSTNAME)
```

例 3: 指定された部門番号の従業員を戻す表関数を定義します。

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
```

```
    SELECT EMPNO, LASTNAME, FIRSTNME
        FROM EMPLOYEE
        WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

例 4: ストリングを反転するスカラー関数を定義します。

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
  RETURNS VARCHAR(4000)
  DETERMINISTIC NO EXTERNAL ACTION CONTAINS SQL
  BEGIN ATOMIC
  DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT '';
  DECLARE LEN INT;
  IF INSTR IS NULL THEN
  RETURN NULL;
  END IF;
  SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));
  WHILE LEN > 0 DO
  SET (REVSTR, RESTSTR, LEN)
    = (SUBSTR(RESTSTR, 1, 1) || REVSTR,
    SUBSTR(RESTSTR, 2, LEN - 1),
    LEN - 1);
  END WHILE;
  RETURN REVSTR;
END
```

例 5: 例 4 から監査付きで表関数を定義します。

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO AUDIT
    VALUES (USER,
            'Table: EMPLOYEE Prd: DEPTNO = ' || DEPTNO);
    RETURN
      SELECT EMPNO, LASTNAME, FIRSTNME
        FROM EMPLOYEE
        WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
  END
```

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 440 ページの『CREATE TYPE (構造化)』
- 671 ページの『RETURN』
- 131 ページの『コンパウンド SQL (動的)』
- 200 ページの『CREATE FUNCTION (外部スカラー)』
- *SQL リファレンス 第 1 巻* の『ルーチンで使用可能な SQL ステートメント』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』

CREATE FUNCTION MAPPING

CREATE FUNCTION MAPPING ステートメントは、以下の目的で使用されます。

- フェデレーテッド・データベース関数 (または関数テンプレート) と、データ・ソース関数との間でマッピングを作成する。マッピングによって、フェデレーテッド・データベース関数またはテンプレートを以下に含まれている関数に関連付けることができます。
  - 指定したデータ・ソース
  - データ・ソースの範囲。たとえば、特定のタイプおよびバージョンのすべてのデータ・ソース
- フェデレーテッド・データベース関数とデータ・ソース関数との間での、デフォルトのマッピングを使用不可にする。

1 つの関数に対して複数の関数マッピングが適用可能である場合、最新のマッピングが適用されます。

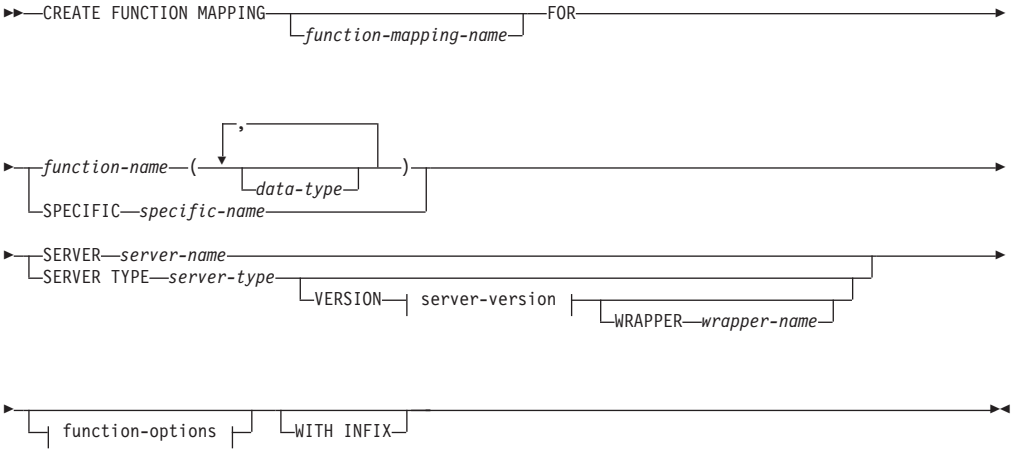
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

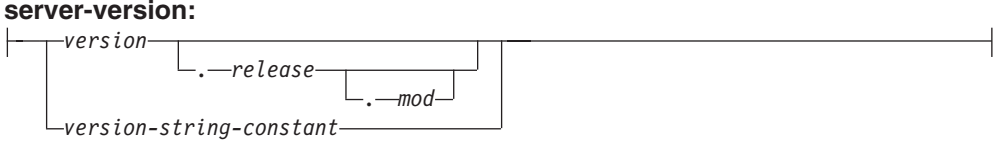
許可:

このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

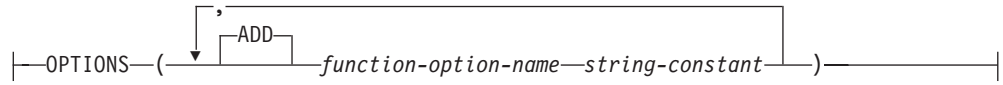
構文:



server-version:



function-options:



説明:

function-mapping-name
関数マッピングを指定します。この名前は、カタログですでに記述されている関数マッピングを指定するものであってはなりません (SQLSTATE 42710)。

function-mapping-name を省略すると、システムが生成したユニークな名前が割り当てられます。

function-name
マップ元となるフェデレーテッド・データベース関数またはフェデレーテッド・データベース関数テンプレートの修飾名または非修飾名を指定します。

data-type
入力パラメーターのある関数または関数テンプレートの場合、 *data-type* にそれらのパラメーターのデータ・タイプを指定します。 *data type* を、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、またはユーザー定義タイプにすることはできません。

パラメーター化データ・タイプに長さ、精度、または位取りを指定する代わりに、空の括弧を使用できます。パラメーター化データ・タイプには空の括弧を使用することが推奨されています。たとえば、CHAR() のようにします。特定の長さ、位取り、または精度を指定して定義可能なデータ・タイプのことを、パラメーター化データ・タイプといいます。パラメーター化データ・タイプは、ストリング・データ・タイプと 10 進データ・タイプです。指定する長さ、精度、または位取りは、関数テンプレートのものと同じでなければなりません。括弧をすべて省略した場合は、データ・タイプのデフォルト長が使用されます (CREATE TABLE ステートメントの説明を参照)。

SPECIFIC *specific-name*
マップ元の関数または関数テンプレートを指定します。フェデレーテッド・データベースにおいて、関数または関数テンプレートに、ユニークな *function-name* がない場合、*specific-name* を指定してください。

SERVER *server-name*
マップ元の関数を含むデータ・ソースを指定します。

TYPE *server-type*
マップ元の関数を含むデータ・ソースのタイプを指定します。

VERSION
server-type に示されたデータ・ソースのバージョンを指定します。

version
バージョン番号を指定します。値は整数でなければなりません。

release
version で示されたバージョンのリリース番号を指定します。値は整数でなければなりません。

mod

release で示されたリリースのモディフィケーション番号を指定します。値は整数でなければなりません。

version-string-constant

バージョンの正式名称を指定します。 *version-string-constant* は単一値 (たとえば、'8i') にすることができます。あるいは、*version*、*release*、そして該当する場合は *mod* を連結した値にすることができます (たとえば、'8.0.3')。

WRAPPER *wrapper-name*

server-type および *server-version* に示されたタイプおよびバージョンのデータ・ソースと対話するために、フェデレーテッド・サーバーが使用するラッパーの名前を指定します。

OPTIONS

使用可能にする関数マッピング・オプションを指定します。

ADD

1 つ以上の関数マッピング・オプションを使用可能にします。

function-option-name

関数マッピング、またはマッピングに含まれるデータ・ソース関数のいずれかに適用される、関数マッピング・オプションを指定します。

string-constant

function-option-name の設定を、文字ストリング定数として指定します。

WITH INFIX

データ・ソース関数が infix 形式で生成されることを指定します。フェデレーテッド・データベース・システムは接頭表記法をリモート・データ・ソースが使用する中置表記法に変換します。

注:

- フェデレーテッド・データベースの関数または関数テンプレートは、以下の場合に、データ・ソース関数へマッピングすることができます。
  - フェデレーテッド・データベース関数またはテンプレートに、データ・ソース関数と同じ数の入力パラメーターがある場合。
  - フェデレーテッド・データベース関数またはテンプレートに定義されたデータ・タイプと、データ・ソース関数に定義された対応するデータ・タイプとが互換性のある場合。
- 分散要求が、データ・ソース関数へマップされる DB2 関数を参照する場合、オプティマイザーは、要求の処理時にいずれかの関数を呼び出すための戦略を開発します。 DB2 関数を呼び出すときのオーバーヘッドが、データ・ソース関数を呼び出す場合より少なければ、DB2 関数が呼び出されます。しかし、DB2 関数の方がオーバーヘッドが大きいのであれば、データ・ソース関数が呼び出されます。
- 分散要求が、データ・ソース関数へマップされる DB2 関数テンプレートを参照する場合、要求の処理時には、データ・ソース関数だけを呼び出せます。テンプレートには実行可能コードがないので呼び出せません。
- デフォルトの関数マッピングを使用不可にして、操作できないよう設定することができます (ドロップすることはできません)。デフォルトの関数マッピングを使

用不可にするには、 CREATE FUNCTION MAPPING ステートメントをコーディングし、マッピング内に DB2 関数の名前を指定して DISABLE オプションを 'Y' に設定します。

- SYSIBM スキーマ内の関数には特定の名前はありません。 SYSIBM スキーマの関数のデフォルト関数マッピングをオーバーライドするには、修飾子 SYSIBM と関数名 (LENGTH など) で構成される *function-name* を指定します。
- 所定の作業単位 (UOW) 内の CREATE FUNCTION MAPPING ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - ステートメントが 1 つのデータ・ソースを参照していて、次のいずれかがすでに UOW に含まれている。
    - このデータ・ソース内の表またはビューのニックネームを参照する SELECT ステートメント。
    - このデータ・ソース内の表またはビューのニックネーム上のオープン・カーソル。
    - このデータ・ソース内の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
  - ステートメントがデータ・ソースのカテゴリー (例えば、特定のタイプおよびバージョンのすべてのデータ・ソースなど) を参照しており、次のいずれかがすでに UOW に含まれている。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームを参照する SELECT ステートメント。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネーム上のオープン・カーソル。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。

例:

例 1: 関数テンプレートを、すべての Oracle データ・ソースでアクセスできる UDF にマップします。このテンプレートは STATS といい、NOVA というスキーマに属するものです。 Oracle UDF は STATISTICS といい、STAR というスキーマに属しています。

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN1
  FOR NOVA.STATS (DOUBLE, DOUBLE)
  SERVER TYPE ORACLE
  OPTIONS (REMOTE_NAME 'STAR.STATISTICS')
```

例 2: BONUS という関数テンプレートを、 ORACLE1 という Oracle データ・ソースで使われている UDF (これも BONUS という) へマップします。

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN2
  FOR BONUS()
  SERVER ORACLE1
  OPTIONS (REMOTE_NAME 'BONUS')
```

例 3: フェデレーテッド・データベースに対して定義されている WEEK システム関数と、 Oracle データ・ソースに対して定義されている同様の関数との間で、デフォルトの関数マッピングが行われているとします。 Oracle データを要求し、WEEK を参照する照会が処理されると、オプティマイザーによるオーバーヘッドを少なくするための見積もりに応じて、 WEEK 関数またはその Oracle 側の対となる関数の

いずれかが呼び出されます。 DBA は、そのときの照会で WEEK だけが呼び出された場合に、パフォーマンスがどの程度影響を受けるかを確認するようにしてください。 WEEK が毎回確実に呼び出されるようにするには、 DBA はマッピングを使用不可にしなければなりません。

```
CREATE FUNCTION MAPPING
  FOR SYSFUN.WEEK(INT)
  TYPE ORACLE
  OPTIONS (DISABLE 'Y')
```

例 4: ローカル関数 UCASE(CHAR) を、 ORACLE2 という Oracle データ・ソースで使っている UDF へマップします。 Oracle UDF の呼び出しごとの見積命令数を組み込みます。

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN4
  FOR SYSFUN.UCASE(CHAR)
  SERVER ORACLE2
  OPTIONS
    (REMOTE_NAME 'UPPERCASE',
    INSTS_PER_INVOC '1000')
```

関連概念:

- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムでの関数マッピング』
- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムでの関数マッピングの働き』
- フェデレーテッド・システム・ガイド の『関数テンプレート』
- フェデレーテッド・システム・ガイド の『関数マッピングの作成方法』

関連タスク:

- フェデレーテッド・システム・ガイド の『デフォルトの関数マッピングを使用不可にする』

関連資料:

- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムの関数マッピング・オプション』

CREATE INDEX

CREATE INDEX ステートメントは、以下の目的で使用されます。

- DB2 表に対する索引を作成します。
- 索引指定、つまり、データ・ソース表に索引があることをオプティマイザーに通知するメタデータを作成します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

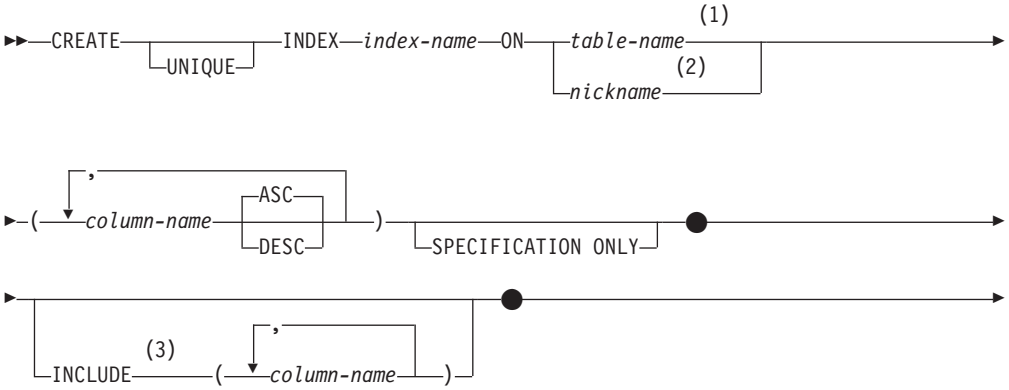
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

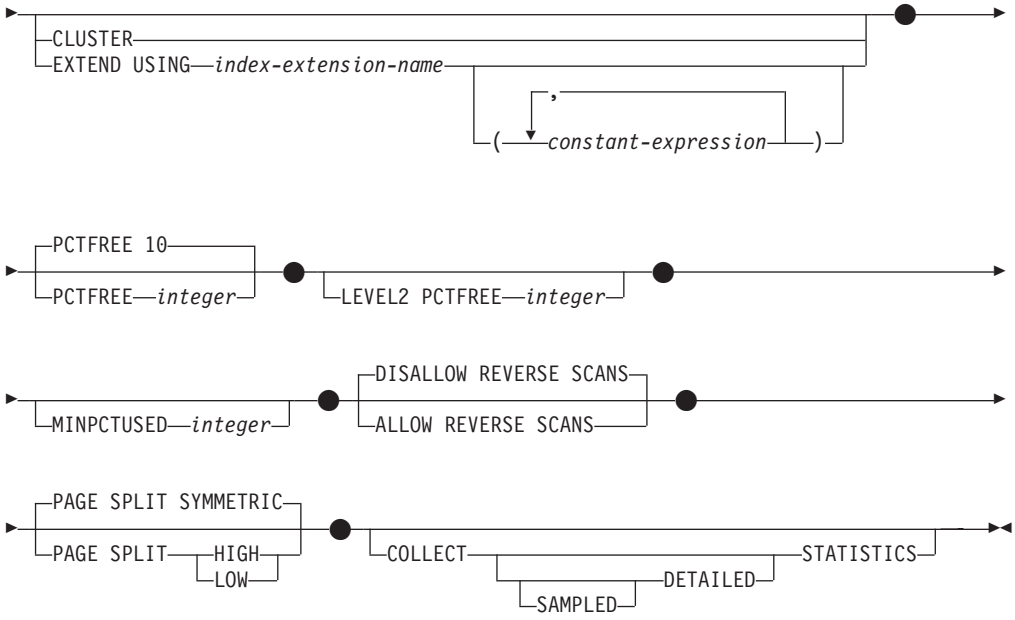
- SYSADM または DBADM 権限
- 以下のいずれか
  - その索引の定義されている表またはニックネームに対する CONTROL 特権。
  - その索引の定義されている表またはニックネームに対する INDEX 特権。

  および以下のいずれか
  - データベースに対する IMPLICIT_SCHEMA 権限 (索引の暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (索引のスキーマ名が既存のスキーマを指している場合)。

宣言済み一時表の索引を作成するのに、明示特権は必要ありません。

構文:





注:

1 フェデレーテッド・システムでは、*table-name* には、フェデレーテッド・データベース内の表を指定します。データ・ソース表を指定することはできません。

2 *nickname* を指定すると、 CREATE INDEX ステートメントは、索引指定を作成します。この場合、INCLUDE、CLUSTER、EXTEND USING、PCTFREE、MINPCTUSED、DISALLOW REVERSE SCANS、ALLOW REVERSE SCANS、PAGE SPLIT、または COLLECT STATISTICS は指定できません。

3 INCLUDE 文節は UNIQUE が指定されている場合のみ指定できます。

説明:

UNIQUE

ON *table-name* を指定する場合、 UNIQUE により、表には索引キーの値が同じである複数の行を含めることができなくなります。行の更新、または新しい行の挿入を行う SQL ステートメントの終了時に、固有性が確保されます。

この固有性は、CREATE INDEX ステートメントの実行の過程でも検査されます。重複するキー値を含む行がすでに表に含まれている場合、索引は作成されません。

UNIQUE を使用する場合、NULL 値は他の値と同様に扱われます。たとえば、キーが NULL 可能の単一列である場合、その列では 1 つの NULL 値しか含めることができません。

UNIQUE オプションの指定があり、しかも表にパーティション・キーがある場合、索引キーの列はパーティション・キーのスーパーセットである必要があります。つまり、ユニーク索引キーに対して指定される列は、パーティション・キーのすべての列を含んでいる必要があります (SQLSTATE 42997)。

主キーまたはユニーク・キーは、ディメンションのサブセットにはなりません (SQLSTATE 429BE)。

ON *nickname* が指定されている場合、索引キーのデータ・ソース表の各行にユニークな値が含まれている場合に限り、UNIQUE を指定するようにします。固有性は検査されません。

INDEX *index-name*
索引または索引指定を指定します。暗黙または明示の修飾子を含む名前は、カタログに記述されている索引または索引指定、または宣言済み一時表の既存の索引を識別するものであってはなりません (SQLSTATE 42704)。修飾子は、SYSIBM、 SYSCAT、 SYSFUN、または SYSSTAT であってはなりません (SQLSTATE 42939)。

宣言済みグローバル一時表の索引の暗黙または明示の修飾子は、SESSION でなければなりません (SQLSTATE 428EK)。

ON *table-name* **または** *nickname*
table-name には、索引を作成する表を識別します。表は、基本表 (ビューではない) か、カタログに記述されたマテリアライズ照会表か、または宣言済み一時表でなければなりません。宣言済み一時表の名前は、SESSION によって修飾される必要があります。 *table-name* に、カタログ表を指定することはできません (SQLSTATE 42832)。 UNIQUE が指定されており、 *table-name* が型付き表である場合には、副表を指定することはできません (SQLSTATE 429B3)。

nickname は、索引指定を作成するニックネームです。この *nickname* により、索引が索引指定によって記述されているデータ・ソース表、またはそのような表に基づくデータ・ソース・ビューのいずれかが参照されます。この *nickname* は、カタログにリストされていなければなりません。

column-name
索引の場合、*column-name* には索引キーを構成する列を指定します。索引指定の場合、*column-name* は、フェデレーテッド・サーバーがデータ・ソース表の列を参照するときの名前になります。

各 *column-name* は、表の列を指定する非修飾名でなければなりません。列は 16 個まで指定できます。 *table-name* が型付き表である場合は、15 列まで指定できます。 *table-name* が副表であるならば、副表内の少なくとも 1 つの *column-name* はスーパー表から継承するのではなく、新たに導入する必要があります。同じ *column-name* を繰り返すことはできません (SQLSTATE 42711)。

指定された列の保管長の合計は、1024 バイトを超えてはなりません。*table-name* が型付き表である場合は、索引キーの長さ制限は 4 バイト減ります。

この長さは、列のデータ・タイプや NULL 可能か否かによって変動するシステムのオーバーヘッドにより、より小さい値になることがあります。この制限に影響を与えるオーバーヘッドの詳細については、 『CREATE TABLE』 の『バイト・カウント』を参照してください。

LOB 列、DATALINK 列、または LOB や DATALINK の特殊タイプ列を索引の一部として使用することはできません。列の長さ属性が 1024 バイトの限界内に収まる場合でも同じです (SQLSTATE 54008)。構造タイプ列は、EXTEND USING 文節も指定されている場合にのみ指定できます (SQLSTATE 42962)。

EXTEND USING 文節が指定される場合、列は 1 つしか指定できず、列のタイプは構造タイプか、あるいは LOB、DATALINK、LONG VARCHAR、 LONG VARGRAPHIC を基にしていたのではない特殊タイプでなければなりません (SQLSTATE 42997)。

ASC
索引項目が、列の値の昇順で保持されるように指定します。これがデフォルト設定です。 ASC は、EXTEND USING で定義される索引に指定することはできません (SQLSTATE 42601)。

DESC
索引項目が、列の値の降順で保持されるように指定します。 DESC は、EXTEND USING で定義される索引に指定することはできません (SQLSTATE 42601)。

SPECIFICATION ONLY
このステートメントが、 *nickname* で参照するデータ・ソース表に適用される索引指定を作成するときに使われることを示します。 SPECIFICATION ONLY は、*nickname* を指定した場合に、指定しなければなりません (SQLSTATE 42601)。 *table-name* を指定した場合には、指定することはできません (SQLSTATE 42601)。

宣言済み一時表の索引が作成される場合には、この文節は使用できません (SQLSTATE 42995)。

INCLUDE
このキーワードは、一連の索引キー列に付加する追加の列を指定する文節を新たに指定します。この文節によって組み込まれる列は、固有性を強制するために使用されることはありません。これらの組み込み列を使用して索引のみアクセスを実行することにより、一部の照会のパフォーマンスを向上させることができます。この列は、固有性を強制するために使用される列とは区別する必要があります (SQLSTATE 42711)。列の数および長さ属性の合計に対する制限は、ユニーク・キーと索引にあるすべての列にも適用されます。

この文節は、宣言済み一時表と共には使用できません (SQLSTATE 42995)。

column-name
索引には組み込まれているものの、ユニーク索引キーの一部ではない列を指定します。ユニーク索引キーの列に定義された規則と同様の規則が適用されます。 column-name に続けてキーワード ASC または DESC を指定しても構いませんが、順序に影響はありません。

INCLUDE は、EXTEND USING で定義される索引に指定することはできません。 *nickname* が指定されている場合にも使用できません (SQLSTATE 42601)。

CLUSTER
当該の索引を表のクラスター索引として指定します。クラスター索引のクラスター係数は、関連する表にデータが挿入される時に、動的に維持または改善されます。これは、この索引のキー値が同じ範囲にある行と物理的に近い位置に、新しい行の挿入を試みることによって行われます。ただし、表のクラスター索引は 1 つだけなので、 CLUSTER が表の既存の索引の定義に使用されていて、

CLUSTER が指定できないということもありえます (SQLSTATE 55012)。追加モードを使用するように定義されている表では、クラスター索引を作成できない場合があります (SQLSTATE 428D8)。

CLUSTER は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。この文節は、宣言済み一時表 (SQLSTATE 42995) や範囲クラスター表 (SQLSTATE 429BG) では使用できません。

EXTEND USING *index-extension-name*
この索引を管理するのに使用する *index-extension* を指定します。この文節を指定する場合、1 つだけ *column-name* を指定しなければならず、この列は構造タイプまたは特殊タイプでなければなりません (SQLSTATE 42997)。
index-extension-name は、カタログに記述されている索引拡張を指定しなければなりません (SQLSTATE 42704)。特殊タイプの場合には、列が、索引拡張でソース・キー・パラメーターに対応するタイプと完全に一致していなければなりません。構造タイプ列では、対応するソース・キー・パラメーターのタイプが、列タイプのタイプまたはスーパータイプと同じでなければなりません (SQLSTATE 428E0)。

この文節は、宣言済み一時表と共には使用できません (SQLSTATE 42995)。

constant-expression
索引拡張に必要な引き数の値を指定します。各式は、対応する索引拡張パラメーターの定義されたデータ・タイプ (長さまたは精度、およびスケールも含む) に完全に一致するデータ・タイプを持つ定数値でなければなりません (SQLSTATE 428E0)。 この文節は、データベース・コード・ページ内で、32 768 バイト以内の長さでなければなりません。(SQLSTATE 22001)

PCTFREE *integer*
索引を構築する際に、各索引ページに残すフリー・スペースのパーセンテージを指定します。ページの最初の項目は、制限なしで追加されます。索引ページに項目を追加する場合には、各ページに少なくとも *integer* パーセントをフリー・スペースとして残します。 *integer* の値は 0 ～ 99 です。ただし、10 よりも大きな値を指定しても、ノンリーフ・ページには 10% のフリー・スペースしか残されません。デフォルト値は 10 です。

PCTFREE は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。この文節は、宣言済み一時表と共には使用できません (SQLSTATE 42995)。

LEVEL2 PCTFREE *integer*
索引を作成する際に、索引レベル 2 の各ページで何 % をフリー・スペースとして残すかを指定します。 *integer* の値は 0 ～ 99 です。 LEVEL2 PCTFREE が設定されない場合は、すべての非リーフ・ページに最低 10 % または PCTFREE で指定された分 (%) のフリー・スペースが残されます。 LEVEL2 PCTFREE が設定された場合は、 *integer* で指定された分 (%) のフリー・スペースがレベル 2 の中間ページに残され、レベル 3 以上の中間ページには最低 10 % または *integer* で指定された分 (%) のフリー・スペースが残されます。

nickname が指定されている場合は、LEVEL2 PCTFREE は使用できません (SQLSTATE 42601)。この文節は、宣言済み一時表と共には使用できません (SQLSTATE 42995)。

MINPCTUSED *integer*
索引のリーフ・ページをオンラインでマージするかどうか、および索引のリーフ・ページで使用されるスペースの最小パーセンテージの限界値を指定します。索引のリーフ・ページからキーを除去した後、そのページで使用されているスペースのパーセントが *integer* のパーセントを下回る場合、このページにある残りのキーを近隣のページのキーにマージするよう試行されます。いずれかのページに十分なスペースがあれば、マージが行われ、いずれかのページが削除されます。 *integer* の値は 0 ～ 99 です。しかし、パフォーマンス上の理由のため、50 以下の値をお勧めします。 このオプションを指定すると、更新および削除のパフォーマンスに影響があります。タイプ 2 の索引の場合、排他的表ロックがあると、更新および削除操作の実行中に限りマージされます。 排他的表ロックがない場合には、更新および削除操作の間にキーは疑似的に削除されたものとしてマークされ、マージは実行されません。リーフ・ページをマージするには、CREATE INDEX の MINPCTUSED を使うのではなく、 REORG INDEXES の CLEANUP ONLY ALL オプションを使用することを考慮してください。

MINPCTUSED は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。この文節は、宣言済み一時表と共には使用できません (SQLSTATE 42995)。

DISALLOW REVERSE SCANS
索引において、前方向スキャン、すなわち INDEX CREATE の実行時に定義した順序でのスキャンだけをサポートすることを指定します。これはデフォルトです。

DISALLOW REVERSE SCANS は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。

ALLOW REVERSE SCANS
索引が前方向スキャンと反対方向スキャンの両方、すなわち、 INDEX CREATE の実行時に定義した順序と、その反対 (つまり逆) の順序とをサポートすることを指定します。

ALLOW REVERSE SCANS は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。

PAGE SPLIT
索引分割の振る舞いを指定します。デフォルトは SYMMETRIC です。

SYMMETRIC
ページを大まかに半分で分割するよう指定します。

HIGH
索引キーの値が特定のパターンに従って挿入されている場合に、索引ページのスペースを効率的に使う索引ページの分割の振る舞いを指定します。索引キーには、1 つ以上の列が含まれていなければなりません。索引キー値のサブセットについては、索引の左端の列 (複数の場合もある) には常に同じ値が含まれていなければならず、索引の右端の列 (複数の場合もある) には挿入ごとに増加する値が含まれていなければなりません。詳細は、『CREATE INDEX ステートメントのオプション』を参照してください。

LOW
索引キーの値が特定のパターンに従って挿入されている場合に、索引ページのスペースを効率的に使う索引ページの分割の振る舞いを指定します。索引

キーには、1 つ以上の列が含まれていなければなりません。索引キー値のサブセットについては、索引の左端の列 (複数の場合もある) には常に同じ値が含まれていなければならず、索引の右端の列 (複数の場合もある) には挿入ごとに減少する値が含まれていなければなりません。詳細は、『CREATE INDEX ステートメントのオプション』を参照してください。

COLLECT STATISTICS
索引の作成時に基本索引統計が収集されるように指定します。

DETAILED
索引の作成時に、拡張索引統計 (CLUSTERFACTOR および PAGE_FETCH_PAIRS) も収集されるように指定します。

SAMPLED
拡張索引統計のコンパイル時に、サンプリングを使用できるように指定します。

規則:

- 既存の索引に一致する索引を作成しようとすると、 CREATE INDEX ステートメントはエラーになります (SQLSTATE 01550)。 2 つの索引記述は、以下の場合に重複していると見なされます。
  - 列の集合 (キーと組み込み列の両方) と、索引内でのそれらの順序が、既存の索引と同じであり、かつ
  - 順序付け属性が同じであり、しかも
  - 以前に存在していた索引と作成中の索引の両方がユニークでないか、または以前に存在していた索引がユニークであり、さらに
  - 以前に存在していた索引と作成中の索引の両方がユニークである場合、作成中の索引のキー列は、以前に存在していた索引のキー列と同一であるか、またはそのスーパーセットになります。

注:

- ***互換性***
  - DB2 for OS/390 との互換性:
    - 以下の構文は許容されますが、無視されます。
      - CLOSE
      - DEFINE
      - FREEPAGE
      - GBPCACHE
      - PIECESIZE
      - TYPE 2
      - using-block
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - COPY NO
      - DEFER NO
- 索引が作成されている間は、表に同時に読み取り/書き込みアクセスできます。索引が作成されると、索引の作成時に表に加えられた変更は、新しい索引に送られ適用されます。表への書き込みアクセスは、新しい索引が使用できるようになってから、索引の作成が完了するまでの短時間ブロックされます。

このデフォルトの動作を回避するには、LOCK TABLE ステートメントを使用して、 CREATE INDEX ステートメントが発行される前に表を明示的にロックします。 (表は SHARE か EXCLUSIVE モードのいずれかでロックできます。読み取りアクセスが許可されているかどうかによります。)

- 指定した表にすでにデータが含まれる場合、CREATE INDEX は、そのデータの索引項目を作成します。表にまだデータが含まれていない場合、CREATE INDEX は索引記述を作成します。索引項目は、データが表に挿入される時点で作成されます。
- 索引が作成され、データが表にロードされた時点で、 RUNSTATS コマンドを実行することをお勧めします。 RUNSTATS コマンドは、データベース表、列、および索引について収集された統計値を更新します。これらの統計値は、表への最適アクセス・パスを判別するために使用されます。 RUNSTATS コマンドを実行することによって、データベース・マネージャーが新しい索引の特性を判別することができます。 CREATE INDEX ステートメントが発行される前にデータをロードする場合には、 CREATE INDEX ステートメントの COLLECT STATISTICS オプションを、 RUNSTATS コマンドの代わりに使用することをお勧めします。
- まだ存在していないスキーマ名を用いて索引を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- オプティマイザーは、実際の索引を作成する前に、複数の索引を推奨することがあります。
- 索引のあるデータ・ソース表に索引指定を定義している場合、その索引指定の名前は索引の名前と一致していなくても構いません。
- オプティマイザーは索引指定を使用して、その指定を適用するデータ・ソース表へのアクセスを改善します。
- COLLECT STATISTICS オプションは、宣言済み一時表ではサポートされません (SQLSTATE 42995)。
- COLLECT STATISTICS オプションは、ニックネームが指定される場合にはサポートされません (SQLSTATE 42601)。
- マテリアライズ照会表 (MQT) でユニーク索引を作成するときには、他の処理に対するこのユニーク性制約の影響を考慮してください。例えば、ユニーク索引が、即時リフレッシュのシステム保守 MQT に対するマテリアライズ照会のユニーク性属性に合致していない場合、それは、基礎表に対する挿入または更新操作の際にユニーク性違反を捉える MQT 上の索引になります。据え置きリフレッシュのシステム保守 MQT での同様のシナリオでは、 REFRESH TABLE ステートメントが失敗することになります。一般に、MQT 上のユニーク索引は、基礎表に基づくデータに対してすでに存在するユニーク性制約 (あるいは MQT に関連した照会から示唆されるユニーク性制約) に合致しているべきです。

例:

例 1: PROJECT 表に対して UNIQUE_NAM という名前の索引を作成します。この索引の目的は、プロジェクト名 (PROJNAME) の値が同じ 2 つの項目が表に作成されないようにすることです。索引項目は昇順に並べます。

```
CREATE UNIQUE INDEX UNIQUE_NAM
  ON PROJECT(PROJNAME)
```

例 2: EMPLOYEE 表に対して JOB_BY_DPT という名前の索引を作成します。索引項目は、各部門 (WORKDEPT) の中ではジョブ名 (JOB) 順に昇順で並べます。

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

例 3: ニックネーム EMPLOYEE は、CURRENT_EMP というデータ・ソース表を参照します。このニックネームを作成した後、索引が CURRENT_EMP で定義されます。索引キー用に選んだ列は WORKDEBT と JOB です。この索引を記述する索引指定を作成します。この指定を参照することにより、オプティマイザーは、索引が存在することと索引に含まれるキーを知ることになります。この情報を利用して、オプティマイザーは、表をアクセスするときの戦略を改善することができます。

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

例 4: 構造タイプ列の位置に、拡張索引タイプ SPATIAL_INDEX を作成します。索引拡張 GRID_EXTENSION の記述が SPATIAL_INDEX を保守するのに使用されます。リテラルが GRID_EXTENSION に指定されて、索引格子サイズを作成します。

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
  EXTEND USING (GRID_EXTENSION (x'00010010001000100040001 0'))
```

例 5: TAB1 という名前の表に IDX1 という名前の索引を作成し、索引 IDX1 の基本索引統計を収集します。

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

例 6: TAB1 という名前の表に IDX2 という名前の索引を作成し、索引 IDX2 の詳細な索引統計を収集します。

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

例 7: TAB1 という名前の表に IDX3 という名前の索引を作成し、サンプリングを使用して索引 IDX3 の詳細な索引統計を収集します。

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

関連概念:

- *管理ガイド: インプリメンテーション* の『CREATE INDEX ステートメントのオプション』
- *フェデレーテッド・システム・ガイド* の『フェデレーテッド・システムでの索引の指定』
- *フェデレーテッド・システム・ガイド* の『索引の指定』

関連資料:

- 347 ページの『CREATE TABLE』
- *SQL リファレンス 第 1 巻* の『トリガーと制約の相互作用』
- 286 ページの『CREATE INDEX EXTENSION』

関連サンプル:

- 『dbstat.sqb -- Reorganize table and run statistics (MF COBOL)』
- 『TbGenCol.java -- How to use generated columns (JDBC)』

CREATE INDEX EXTENSION

CREATE INDEX EXTENSION ステートメントは、構造タイプまたは特殊タイプ列のある表で索引を使用するための拡張オブジェクトを作成します。

呼び出し:

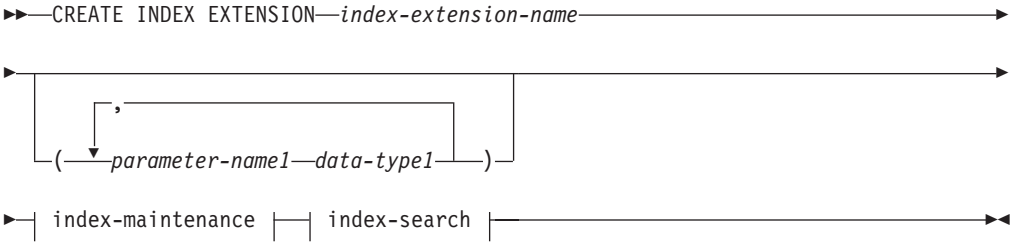
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

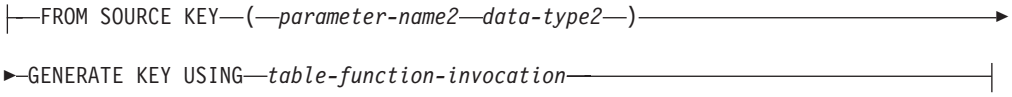
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (索引拡張のスキーマ名が既存のスキーマを指していない場合)
- スキーマに対する CREATEIN 特権 (索引拡張のスキーマ名が既存のスキーマを指している場合)

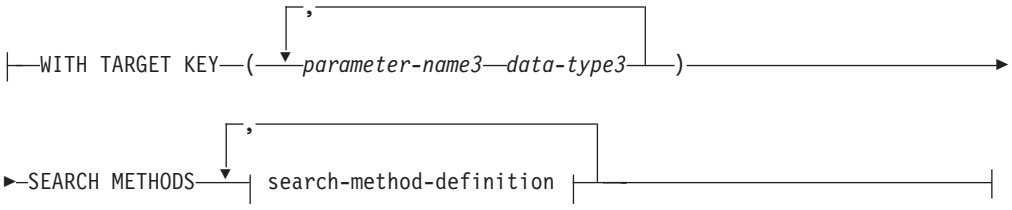
構文:



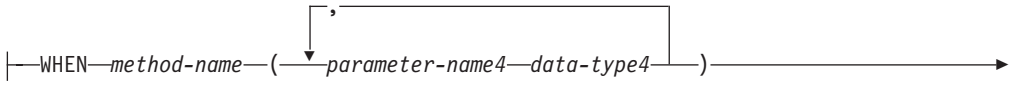
index-maintenance:

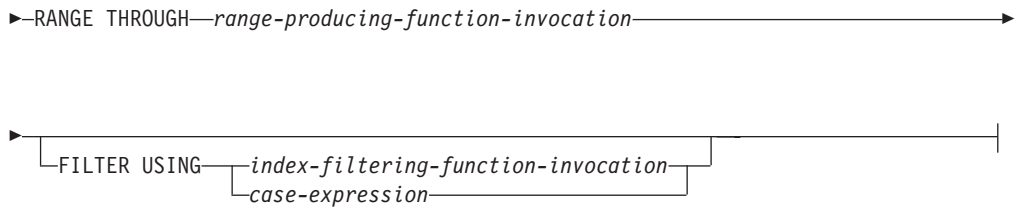


index-search:



search-method-definition:





説明:

index-extension-name
索引拡張を指定します。暗黙または明示の修飾子を含む名前は、カタログに記述されている索引拡張を識別するものであってはなりません。 2 つの部分から成る *index-extension-name* を指定する場合、スキーマ名を 'SYS' で始めることはできません。違反すると、エラーが戻されます (SQLSTATE 42939)。

parameter-name1
CREATE INDEX 時に索引拡張に渡されるパラメーターを指定して、この索引拡張の実際の振る舞いを定義します。索引拡張に渡されるパラメーターは、 インスタンス・パラメーター と呼ばれます。この値が索引拡張の新しいインスタンスを定義するためです。

parameter-name1 は、索引拡張の定義内でユニークでなければなりません。パラメーターの数は 90 を超えることはできません。この限界を超えると、エラー (SQLSTATE 54023) になります。

data-type1
各パラメーターのデータ・タイプを指定します。このリストには、索引拡張が受け取ることを予期している各パラメーターごとに、 1 つの項目を指定する必要があります。指定できる SQL データ・タイプは、VARCHAR、INTEGER、 DECIMAL、 DOUBLE、または VARGRAPHIC など、定数として使用できるタイプだけです (SQLSTATE 429B5)。 CREATE INDEX の索引拡張により受け取られるパラメーター値は、長さ、精度、およびスケールとも、 *data-type1* に完全に一致していなければなりません (SQLSTATE 428E0)。

index-maintenance
構造タイプまたは特殊タイプの列の索引キーを保守する方法を指定します。索引保守は、ソース列をターゲット・キーに変換するプロセスです。変換プロセスは、データベースで以前に定義されている表関数を使用して定義されます。

FROM SOURCE KEY (*parameter-name2 data-type2***)**
この索引拡張によりサポートされるソース・キー列の、構造データ・タイプまたは特殊タイプを指定します。

parameter-name2
ソース・キー列に関連するパラメーターを指定します。ソース・キー列は、 *data-type2* と同じデータ・タイプの索引キー列です (CREATE INDEX で定義)。

data-type2
parameter-name2 のデータ・タイプを指定します。 *data-type2* は、LOB、DATALINK、LONG VARCHAR、 LONG VARGRAPHIC を基に

したのではないユーザー定義のタイプまたは特殊タイプでなければなりません (SQLSTATE 42997)。 CREATE INDEX 時に索引拡張が索引に関連付けられる場合、索引キー列のデータ・タイプは以下のようでなければなりません。

- 特殊タイプの場合、*data-type2* に完全に一致しなければなりません。あるいは、
- 構造タイプの場合、*data-type2* のタイプまたはサブタイプと同じなければなりません。

これ以外の場合には、エラーになります (SQLSTATE 428E0)。

GENERATE KEY USING *table-function-invocation*

ユーザー定義の表関数を使用して索引キーが生成される方法を指定します。単一のソース・キー・データ値に複数の索引項目を生成できます。単一のソース・キー・データ値から索引項目を複製することはできません (SQLSTATE 22526)。この関数は、引き数として *parameter-name1*、*parameter-name2*、または定数を使用できます。データ・タイプ *parameter-name2* が構造タイプの場合、この引き数では、この構造タイプの observer メソッドしか使用できません (SQLSTATE 428E3)。 TARGET KEY 指定では、GENERATE KEY 関数の出力を指定しなければなりません。関数の出力は、FILTER USING 文節で指定される索引フィルター関数の入力としても使用できます。

table-function-invocation で使用される関数は、次のようでなければなりません。

- 表関数に解決されること (SQLSTATE 428E4)
- このデータベースがユニコード・データベースではない場合、PARAMETER CCSID UNICODE で定義しないこと (SQLSTATE 428E4)
- LANGUAGE SQL で定義されていないこと (SQLSTATE 428E4)
- NOT DETERMINISTIC (SQLSTATE 428E4) または EXTERNAL ACTION (SQLSTATE 428E4) で定義されていないこと
- NO SQL で定義されていること (SQLSTATE 428E4)
- パラメーターのデータ・タイプに構造データ・タイプ、 LOB、DATALINK、LONG VARCHAR、または LONG VARGRAPHIC がないこと (SQLSTATE 428E3)。ただし、システム生成のオブザーバー・メソッドだけは例外です。
- 副照会が含まれていないこと (SQLSTATE 428E3)
- EXTEND USING 文節なしで定義された索引の列のデータ・タイプの制限に従うデータ・タイプを持つ列を戻すこと

引き数が他の操作またはルーチンを呼び出す場合、それはオブザーバー・メソッドでなければなりません (SQLSTATE 428E3)。

索引拡張の定義者は、この関数に対して EXECUTE 特権を持っている必要があります。

index-search

検索引き数から検索範囲へのマッピングを提供することにより、検索の実行方法を指定します。

WITH TARGET KEY
GENERATE KEY USING 文節で指定されるキー生成関数の出力であるターゲット・キー・パラメーターを指定します。

parameter-name3
指定されるターゲット・キーに関連するパラメーターを指定します。*parameter-name3* は、 GENERATE KEY USING 文節の表関数で指定された RETURNS 表の列に対応します。指定されるパラメーターの数は、表関数で戻される列の数と一致しなければなりません (SQLSTATE 428E2)。

data-type3
それぞれの *parameter-name3* の対応するデータ・タイプを指定します。*data-type3* は、GENERATE KEY USING 文節の表関数で指定されたように、 RETURNS 表のそれぞれに対応する出力列のデータ・タイプに厳密に一致しなければなりません (SQLSTATE 428E2)。これには、長さ、精度、およびタイプが含まれます。

SEARCH METHODS
索引に定義される検索メソッドを導入します。

search-method-definition
索引検索のメソッドの詳細を指定します。これは、メソッド名、検索引き数、範囲生成関数、およびオプションの索引フィルター関数で構成されます。

WHEN *method-name*
検索メソッドの名前。これは、索引活用規則 (ユーザー定義関数の PREDICATES 文節にある) で指定されるメソッド名に関連する SQL ID です。検索メソッド定義で *search-method-name* を参照できる WHEN 文節は 1 つだけです (SQLSTATE 42713)。

parameter-name4
検索引き数のパラメーターを指定します。これらの名前は、RANGE THROUGH および FILTER USING 文節で使用されます。

data-type4
検索パラメーターに関連付けられるデータ・タイプ。

RANGE THROUGH *range-producing-function-invocation*
検索範囲を生成する外部表関数を指定します。この関数は *parameter-name1*、*parameter-name4*、または定数を引き数として使用し、検索範囲のセットを戻します。

range-producing-function-invocation で使用される表関数は、以下のようでなければなりません。

- 表関数に解決されること (SQLSTATE 428E4)
- その引き数に副照会 (SQLSTATE 428E3) または SQL 関数 (SQLSTATE 428E4) が含まれていないこと
- このデータベースがユニコード・データベースではない場合、PARAMETER CCSID UNICODE で定義しないこと (SQLSTATE 428E4)
- LANGUAGE SQL で定義されていないこと (SQLSTATE 428E4)
- NOT DETERMINISTIC または EXTERNAL ACTION で定義されていないこと (SQLSTATE 428E4)
- NO SQL で定義されていること (SQLSTATE 428E4)

- この関数の結果の数およびタイプが、以下のように GENERATE KEY USING 文節で指定した表関数の結果に関連していること (SQLSTATE 428E1)。
  - キー・トランスフォーム関数で戻される数の 2 倍以内の数の列を戻す。
  - 偶数の列があり、戻りコードの前半で範囲の開始 (開始キー値) を定義し、戻りコードの後半で範囲の終了 (停止キー値) を定義する。
  - 対応する停止キー列と同じタイプの開始キー列がある。
  - 対応するキー・トランスフォーム関数列と同じタイプの開始キー列がある。

厳密には、$a_1:t_1$, ..., $a_n:t_n$ を、関数結果列およびキー・トランスフォーム関数のデータ・タイプにします。 *range-producing-function-invocation* の関数結果列は、 $b_1:t_1$, ..., $b_m:t_m$, $c_1:t_1$, ..., $c_m:t_m$ でなければなりません。ここで、m <= n および "b" 列は開始キー列で、"c" 列は停止キー列です。

range-producing-function-invocation が開始または停止キー値として NULL 値を戻す場合、セマンティクスは未定義です。

索引拡張の定義者は、この関数に対して EXECUTE 特権を持っている必要があります。

FILTER USING

範囲生成関数の適用後に戻された索引項目をフィルター操作する際に使用する、外部関数またはケース式の指定を許可します。

index-filtering-function-invocation

索引項目をフィルター操作するのに使用する外部関数を指定します。この関数は *parameter-name1*、*parameter-name3*、 *parameter-name4*、または定数を引き数として使用し (SQLSTATE 42703)、整数を戻します (SQLSTATE 428E4)。戻される値が 1 の場合、索引項目に対応する行が表から取り出されます。その他の場合、索引項目をさらに処理することはありません。

これを指定しない場合は、索引のフィルター操作は実行されません。

index-filtering-function-invocation で使用される関数は、以下のようでなければなりません。

- このデータベースがユニコード・データベースではない場合、PARAMETER CCSID UNICODE で定義しないこと (SQLSTATE 428E4)
- LANGUAGE SQL で定義されていないこと (SQLSTATE 429B4)
- NOT DETERMINISTIC または EXTERNAL ACTION で定義されていないこと (SQLSTATE 42845)
- NO SQL で定義されていること (SQLSTATE 428E4)
- どのパラメーターのデータ・タイプにも、構造データ・タイプがないこと (SQLSTATE 428E3)
- 副照会が含まれていないこと (SQLSTATE 428E3)

引き数が他の関数またはメソッドを呼び出す場合、このネストされた関数またはメソッドにもこれらの 4 つの規則が課されます。ただし、引き数が組み込みデータ・タイプになるかぎり、システム生成のオブザーバー・メソッ

ドをフィルター関数 (または、引き数として使用される任意の関数またはメソッド) への引き数として使用することができます。

索引拡張の定義者は、この関数に対して EXECUTE 特権を持っている必要があります。

case-expression
索引項目をフィルター操作するためのケース式を指定します。 *searched-when-clause* および *simple-when-clause* では、 *parameter-name1*、*parameter-name3*、*parameter-name4*、または定数を使用できます (SQLSTATE 42703)。 FILTER USING *index-filtering-function-invocation* に指定された規則を使って、外部関数を *result-expression* として使用できます。 *case-expression* で参照される関数またはメソッドはすべて、*index-filtering-function-invocation* でリストされている 4 つの規則に適合することも必要です。加えて、副照会は、*case-expression* の中では使用できません (SQLSTATE 428E4)。ケース式は整数を戻さなければなりません (SQLSTATE 428E4)。 *result-expression* で戻り値が 1 の場合は索引項目が保持され、その他の場合は索引項目は破棄されます。

注:

- まだ存在していないスキーマ名を用いて索引拡張を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。

例:

例 1: ここでは、 *gridEntry* という表関数で構造タイプ SHAPE 列を使用する索引拡張 *grid_extension* を作成して、7 つの索引ターゲット・キーを生成します。この索引拡張は 2 つの索引検索メソッドも提供して、検索引き数が指定される際の検索範囲を生成します。

```
CREATE INDEX EXTENSION GRID_EXTENSION (LEVELS VARCHAR(20) FOR BIT DATA)
  FROM SOURCE KEY (SHAPECOL SHAPE)
  GENERATE KEY USING GRIDENTRY(SHAPECOL..MBR..XMIN,
                               SHAPECOL..MBR..YMIN,
                               SHAPECOL..MBR..XMAX,
                               SHAPECOL..MBR..YMAX,
                               LEVELS)
  WITH TARGET KEY (LEVEL INT, GX INT, GY INT,
                   XMIN INT, YMIN INT, XMAX INT, YMAX INT)
  SEARCH METHODS
  WHEN SEARCHFIRSTBYSECOND (SEARCHARG SHAPE)
  RANGE THROUGH GRIDRANGE(SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX,
                               LEVELS)
  FILTER USING
    CASE WHEN (SEARCHARG..MBR..YMIN > YMAX) OR
       SEARCHARG..MBR..YMAX < YMIN) THEN 0
    ELSE CHECKDUPLICATE(LEVEL, GX, GY,
                               XMIN, YMIN, XMAX, YMAX,
                               SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX,
                               LEVELS)
    END
```

```
WHEN SEARCHSECONDBYFIRST (SEARCHARG SHAPE)
RANGE THROUGH GRIDRANGE(SEARCHARG..MBR..XMIN,
                        SEARCHARG..MBR..YMIN,
                        SEARCHARG..MBR..XMAX,
                        SEARCHARG..MBR..YMAX,
                        LEVELS)
FILTER USING
  CASE WHEN (SEARCHARG..MBR..YMIN > YMAX) OR
     SEARCHARG..MBR..YMAX < YMIN) THEN 0
  ELSE MBROVERLAP(XMIN, YMIN, XMAX, YMAX,
                        SEARCHARG..MBR..XMIN,
                        SEARCHARG..MBR..YMIN,
                        SEARCHARG..MBR..XMAX,
                        SEARCHARG..MBR..YMAX)
 END
```

関連資料:

- *SQL リファレンス 第 1 巻* の『定数』

CREATE METHOD

このステートメントは、すでにユーザー定義の構造タイプの定義の一部となっているメソッド指定に、メソッド本体を関連付けるために使用されます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- CREATE METHOD ステートメントで参照される構造タイプのスキーマに対する CREATEIN 特権
- CREATE METHOD ステートメントで参照される構造タイプの DEFINER

外部メソッド本体をそのメソッド指定に関連付けるためには、ステートメントの許可 ID に以下の少なくとも 1 つが含まれている必要もあります。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限

SQL メソッドを作成する場合、全選択で識別される表、ビュー、またはニックネームのそれぞれに対して、ステートメントの許可 ID に以下の特権が含まれている必要もあります。

- その表、ビュー、またはニックネームに対する CONTROL 特権、または
- その表、ビュー、またはニックネームに対する SELECT 特権

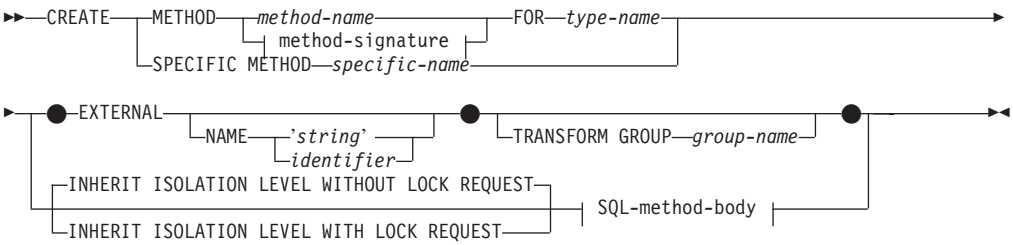
SQL メソッドの定義者が SYSADM 権限を持つために、メソッドの作成しかできない場合、メソッド作成のため、その定義者には暗黙的に DBADM 権限が付与されます。

PUBLIC 以外のグループ特権は、 CREATE METHOD ステートメントで指定された表やビューに対しては考慮されません。

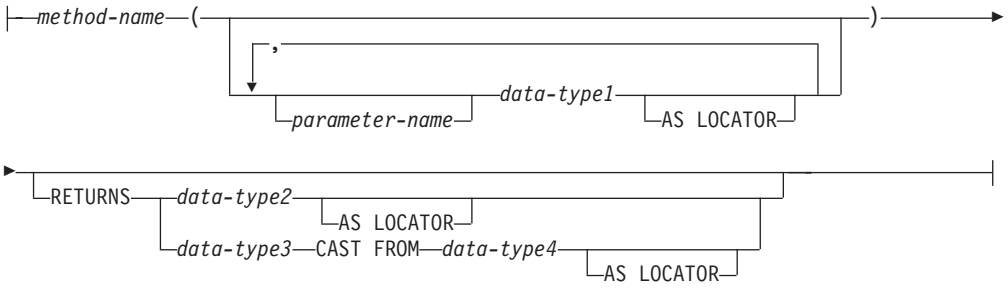
このニックネームで示されている表またはビューのデータ・ソースの許可要件は、メソッドが呼び出される時に適用されます。接続の許可 ID は、別のリモート許可 ID へマップできます。

許可 ID の権限が不十分で、操作を実行できない場合には、エラーになります (SQLSTATE 42502)。

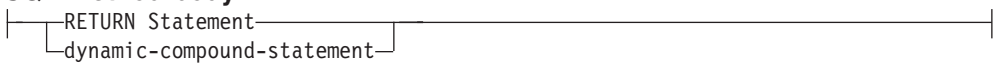
構文:



method-signature:



SQL-method-body:



説明:

METHOD

ユーザー定義の構造タイプに関連付けられる既存のメソッド指定を識別します。メソッド指定は、以下のいずれかの方法で識別できます。

method-name

メソッド本体に対して定義するメソッド指定の名前を指定します。暗黙的スキーマは、サブジェクト・タイプ (*type-name*) のスキーマです。この *method-name* のある *type-name* には、 1 つしかメソッドを指定できません (SQLSTATE 42725)。

method-signature

定義するメソッドを一意的に識別できるメソッド・シグニチャーを指定します。このメソッド・シグニチャーは、 CREATE TYPE または ALTER TYPE ステートメントで提供されたメソッド指定と一致しなければなりません (SQLSTATE 42883)。

method-name

メソッド本体に対して定義するメソッド指定の名前を指定します。暗黙的スキーマは、サブジェクト・タイプ (*type-name*) のスキーマです。

parameter-name

パラメーター名を指定します。パラメーター名がメソッド・シグニチャーにより提供される場合、これらは適合するメソッド指定の対応する部分と全く同じでなければなりません。このステートメントでは、文書化だけのためにパラメーター名がサポートされています。

data-type1

各パラメーターのデータ・タイプを指定します。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。

RETURNS
この文節は、メソッドの出力を指定します。 RETURNS 文節がメソッド・シグニチャーにより提供される場合、これは CREATE TYPE の対応するメソッド指定の対応する部分と全く同じでなければなりません。このステートメントでは、文書化だけのために RETURN 文節がサポートされています。

data-type2
出力のデータ・タイプを指定します。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターが、メソッドにより戻されることを指定します。

data-type3 **CAST FROM** *data-type4*
この形式の RETURNS 文節は、関数コードから戻されたデータ・タイプとは異なるデータ・タイプを、呼び出しステートメントに戻すのに使用されます。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、AS LOCATOR 文節を使用して、 LOB ロケーターが実際の値の代わりにメソッドから戻されるように指定できます。

FOR *type-name*
指定されたメソッドを関連付けるタイプを指定します。この名前は、カタログにすでに記述されているタイプを示すものでなければなりません (SQLSTATE 42704)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

SPECIFIC METHOD *specific-name*
CREATE TYPE 時に指定されたか、デフォルト値として与えられた値を使用して、特定のメソッドを識別します。 specific-name は、指定したスキーマまたは暗黙のスキーマのメソッド指定を識別しなければなりません。そうでない場合、エラーになります (SQLSTATE 42704)。

EXTERNAL
この文節は、この CREATE METHOD ステートメントを使用して登録するメソッドが、外部プログラミング言語で作成されたコードに基づいており、文書化されたリンケージの規則とインターフェースに従っていることを示します。
CREATE TYPE で適合するメソッド指定は、SQL 以外の LANGUAGE を指定しなければなりません。このメソッドが呼び出されると、メソッドのサブジェクトが、暗黙の最初のパラメーターとしてインプリメンテーションに渡されます。

NAME 文節の指定がない場合、"NAME *method-name*" が想定されます。

NAME
この文節は、定義するメソッドをインプリメントするユーザー作成コードの名前を指定します。

'*string*'
'string' オプションは、最大 254 文字のストリング定数です。ストリングに使用される形式は、指定した LANGUAGE によって異なります。特定の言語規則に関する詳細は、『CREATE FUNCTION (外部スカラー)』 を参照してください。

identifier
指定する identifier は SQL ID です。 SQL ID は、ストリングのライブラリー ID として使用されます。区切られた ID でない場合、ID は大文字に変換されます。 ID がスキーマ名で修飾されている場合、スキーマ名の部分は無視されます。この形式の NAME は、LANGUAGE C でのみ使用可能です (CREATE TYPE のメソッド指定で定義)。

TRANSFORM GROUP *group-name*
メソッドを呼び出す際のユーザー定義の構造タイプのトランスフォーメーションに使用するトランスフォーム・グループを指定します。メソッド定義には、ユーザー定義の構造タイプが含まれているため、トランスフォームが必要です。

ここで、トランスフォーム・グループ名を指定することを強くお勧めします。この文節が指定されない場合、使用されるデフォルトのグループ名は DB2_FUNCTION です。参照された構造タイプに、指定した (またはデフォルトの) グループ名が定義されていない場合には、エラーになります (SQLSTATE 42741)。同様に、指定したグループ名または構造タイプに、必須の FROM SQL または TO SQL トランスフォーム関数が定義されていない場合には、エラーになります (SQLSTATE 42744)。

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST または **INHERIT ISOLATION LEVEL WITH LOCK REQUEST**
メソッドが呼び出し元のステートメントの分離レベルを継承している場合に、ロック要求をステートメントの分離レベルに関連付けることができるかどうかを指定します。デフォルトは INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST です。

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
メソッドが呼び出し元のステートメントの分離レベルを継承している場合に、指定した isolation-clause (分離文節) の一部として lock-request-clause (ロック要求文節) が含まれている SQL ステートメントのコンテキストでそのメソッドを呼び出すことができません (SQLSTATE 42601)。

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
メソッドが呼び出し元のステートメントの分離レベルを継承している場合に、指定した lock-request-clause (ロック要求文節) をメソッドが継承することを指定します。

SQL-method-body
CREATE TYPE のメソッド仕様が LANGUAGE SQL の場合、SQL-method-body は、どのようにメソッドがインプリメントされるかを定義します。

SQL-method-body は、以下のメソッド仕様のパーツに従っていなければなりません。

- DETERMINISTIC または NOT DETERMINISTIC (SQLSTATE 428C2)
- EXTERNAL ACTION または NO EXTERNAL ACTION (SQLSTATE 428C2)
- CONTAINS SQL または READS SQL DATA (SQLSTATE 42985)

パラメーター名を SQL-method-body で参照することができます。メソッドのサブジェクトは、暗黙的な最初のパラメーター SELF としてメソッド・インプリメンテーションに渡されます。

詳細については、『コンパウンド SQL (動的)』 および 『RETURN ステートメント』を参照してください。

規則:

- CREATE TYPE または ALTER TYPE ステートメントを使用して、前もってメソッド指定を定義していなければ、CREATE METHOD は使用できません (SQLSTATE 42723)。
- 作成されるメソッドがオーバーライド・メソッドの場合には、以下のメソッドに従属するパッケージは無効になります。
  - オリジナル・メソッド
  - 作成されるスーパータイプのメソッドをサブジェクトとして持つ、他のオーバーライド・メソッド

注:

- メソッドが SQL を許可する場合、外部プログラムは、フェデレーテッド・オブジェクトへのアクセスを試行してはなりません (SQLSTATE 55047)。
- ***特権***

  メソッドの定義者は、メソッドに対する EXECUTE 特権と、メソッドをドロップする権利を常に与えられます。

  EXTERNAL メソッドが作成されると、メソッドの定義者は WITH GRANT OPTION 付きの EXECUTE 特権を常に受け取ります。

  SQL メソッドが作成されると、メソッドの定義者がメソッドを定義するために必要なすべての特権に対して WITH GRANT OPTION を持っている場合、または定義者が SYSADM や DBADM 権限を持っている場合には、メソッドに対する WITH GRANT OPTION 付きの EXECUTE 特権のみが定義者に与えられます。SQL メソッドの定義者にそれらの特権が与えられるのは、それらの特権の派生元の特権がメソッドの作成時に存在している場合に限ります。定義者は、これらの特権を直接持っているか、または PUBLIC の特権として持っていることが必要です。メソッドの定義者がメンバーであるグループを持つ特権は考慮されません。メソッドを使用する場合、接続済みのユーザーの許可 ID には、そのデータ・ソースでニックネームが参照する表またはビューに対する適切な特権がなければなりません。
- ***表アクセスの制限***

  メソッドが READS SQL DATA として定義されている場合は、メソッド内のステートメントは、このメソッドを呼び出したステートメントによって変更される表にアクセスすることはできません (SQLSTATE 57053)。

例:

例 1:

```
CREATE METHOD BONUS (RATE DOUBLE)
  FOR EMP
  RETURN SELF..SALARY * RATE
```

例 2:

```
CREATE METHOD SAMEZIP (addr address_t)
  RETURNS INTEGER
  FOR address_t
  RETURN
    (CASE
      WHEN (self..zip = addr..zip)
        THEN 1
      ELSE 0
    END)
```

例 3:

```
CREATE METHOD DISTANCE (address_t)
  FOR address_t
  EXTERNAL NAME 'addresslib!distance'
  TRANSFORM GROUP func_group
```

関連資料:

- 671 ページの『RETURN』
- 131 ページの『コンパウンド SQL (動的)』
- 200 ページの『CREATE FUNCTION (外部スカラー)』

CREATE NICKNAME

CREATE NICKNAME ステートメントは、データ・ソース・オブジェクトのニックネームを作成します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

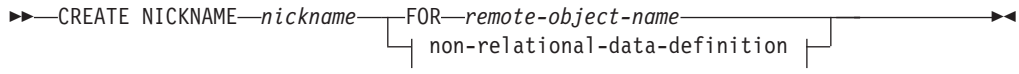
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

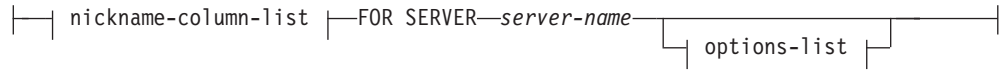
- フェデレーテッド・データベースに対する IMPLICIT_SCHEMA 権限 (ニックネームの暗黙または明示のスキーマ名が存在しない場合)
- スキーマに対する CREATEIN 特権 (ニックネームのスキーマ名が存在する場合)
- SYSADM または DBADM 権限

ユーザー・マッピングが必要なデータ・ソースの場合、データ・ソースで許可 ID が保持する特権に、ニックネームが表すオブジェクトからデータを選択する特権が組み込まれている必要があります。

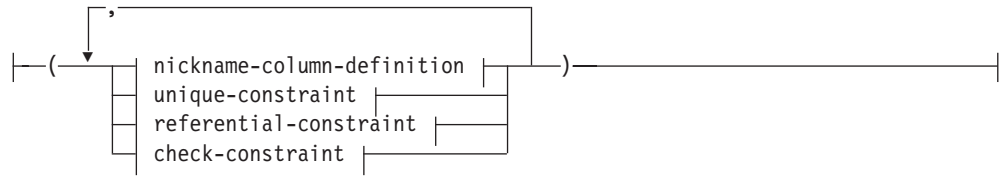
構文:



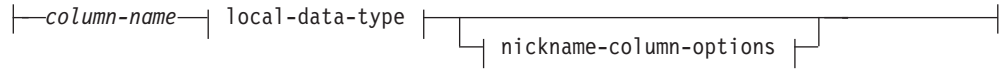
non-relational-data-definition:



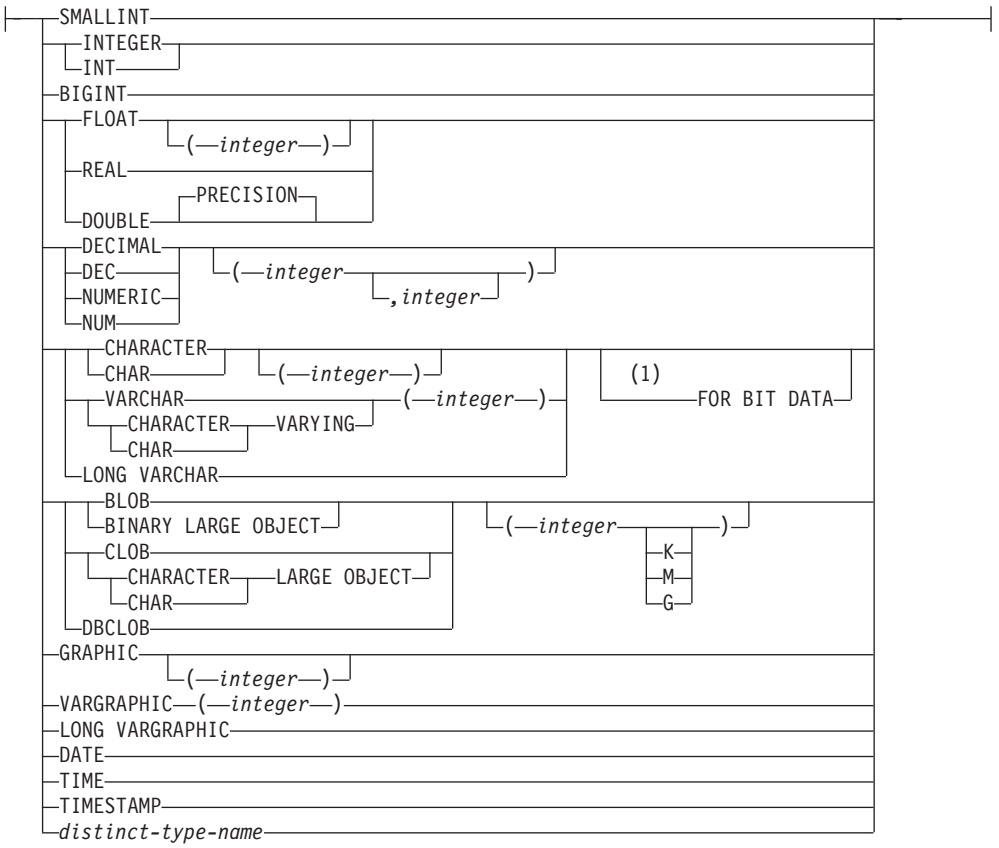
nickname-column-list:



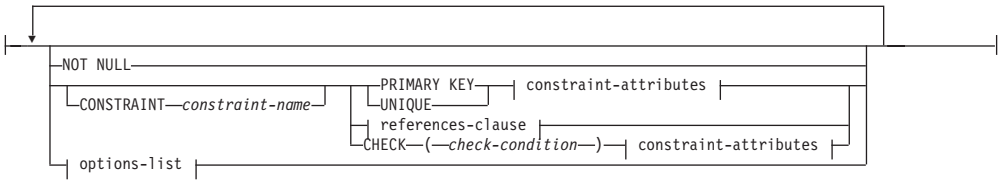
nickname-column-definition:



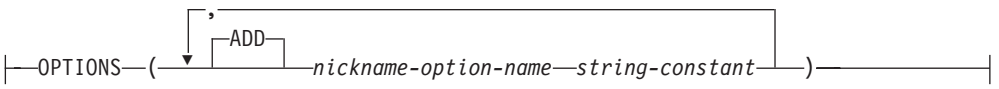
local-data-type:



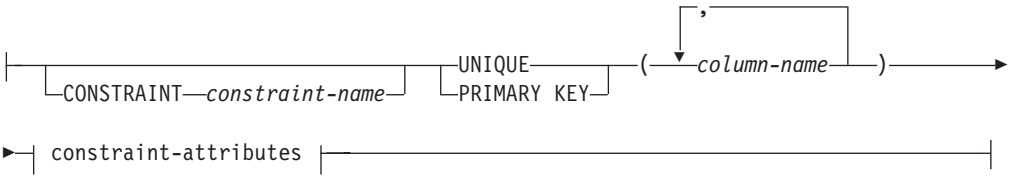
nickname-column-options:



options-list:

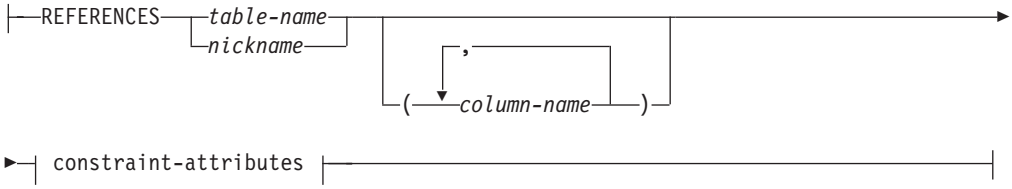


unique-constraint:

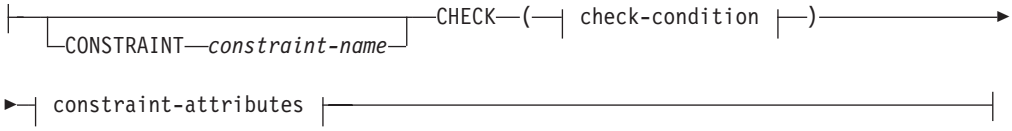


referential-constraint:

references-clause:

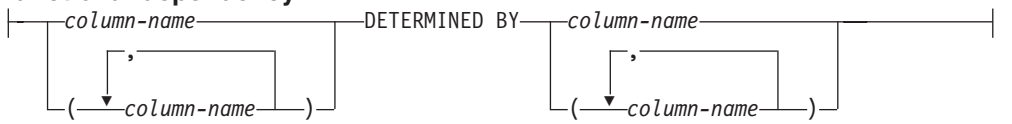


check-constraint:

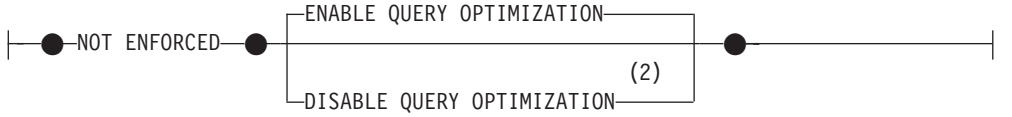


check-condition:

functional-dependency:



constraint-attributes:



注:

1 FOR BIT DATA 文節とその後に続く他の列制約とは、任意の順序で指定できます。

2 DISABLE QUERY OPTIMIZATION はユニーク制約または主キー制約をサポートしていません。

説明:

nickname
ニックネーム、つまりデータ・ソース・オブジェクト用にフェデレーテッド・サーバーが使用する ID を指定します。暗黙または明示の修飾子を含むニックネームは、カタログに記述されている表、ビュー、ニックネーム、または別名を指

定するものであってはなりません。データ・ソース・オブジェクトを DB2 別名にすることはできません。スキーマ名を、'SYS' で始めることはできません (SQLSTATE 42939)。

FOR *remote-object-name*
ID を指定します。スキーマ名をサポートするデータ・ソースの場合、これは *data-source-name.remote-schema-name.remote-table-name* という形式の 3 つの部分からなる ID です。スキーマ名をサポートしないデータ・ソースの場合は、これは *data-source-name.remote-table-name* という形式の 2 つの部分からなる ID です。

data-source-name
ニックネームを作成する対象となる表またはビューを含むデータ・ソースを指定します。 *data-source-name* は、 CREATE SERVER ステートメント内の *server-name* に割り当てられた名前と同じです。

remote-schema-name
表またはビューが属するスキーマを指定します。リモート・スキーマ名に特殊文字や小文字が含まれる場合には、二重引用符で囲まなければなりません。

remote-table-name
ニックネームを作成している、特定のデータ・ソース・オブジェクト (表またはビューなど) の名前。表を宣言済み一時表 (SQLSTATE 42995) にすることはできません。リモート表名に特殊文字や小文字が含まれる場合には、二重引用符で囲まなければなりません。

non-relational-data-definition
非リレーショナル・ラッパーを通してアクセスするデータを定義します。

nickname-column-definition
ニックネームの列のローカル属性を定義します。一部のラッパーは属性を指定する必要がありますが、他のラッパーはデータ・ソースから属性を判別できます。

column-name
列のローカル名を指定します。この名前は、対応する列の *remote-object-name* とは異なる可能性があります。

local-data-type
列のローカル・データ・タイプを指定します。一部のラッパーは、SQL データ・タイプのサブセットのみをサポートします。特定のデータ・タイプの説明は、『CREATE TABLE』 ステートメントの説明を参照してください。

nickname-column-options
ニックネームの列に関連した追加オプションを指定します。

NOT NULL
列で NULL 値を許可しないように指定します。

CONSTRAINT *constraint-name*
制約の名前を指定します。 *constraint-name* (制約名) は、同じ CREATE NICKNAME ステートメントにすでに指定されている制約を指定するものであってはなりません (SQLSTATE 42710)。

この文節が省略された場合は、ニックネームに定義されている既存の制約の ID の中でユニークな 18 文字の ID がシステムによって生成されます。 (ID は、'SQL' と、タイム・スタンプ関数によって生成される 15 の数字で構成されます。)

PRIMARY KEY 制約またはユニーク制約とともに使用した場合、この *constraint-name* は、制約をサポートするために作成される索引の指定の名前として使用されます。

PRIMARY KEY
これは、1 つの列からなる主キーを定義する簡単な方法として用意されています。つまり、PRIMARY KEY が列 C の定義で指定されている場合、その効果は、PRIMARY KEY(C) 文節を独立した文節として指定する場合と同じです。

後述の *unique-constraint* の説明の中の PRIMARY KEY を参照してください。

UNIQUE
これは、1 つの列からなるユニーク・キーを定義する簡単な方法です。すなわち、UNIQUE を列 C の定義に指定すると、UNIQUE(C) 文節を独立した文節として指定した場合と同じ結果になります。

後述の *unique-constraint* の説明の中の UNIQUE を参照してください。

references-clause
これは、1 つの列からなる外部キーを定義する簡単な方法として用意されています。つまり、references-clause が列 C の定義に指定されている場合、その効果は、列として C しか指定されていない FOREIGN KEY 文節の一部として references-clause が指定された場合と同じになります。

後述の *referential-constraint* の *references-clause* の項を参照してください。

CHECK (*check-condition***)**
これは、1 つの列に適用されるチェック制約を定義する簡単な方法として用意されています。後述の CHECK (*check-condition*) を参照してください。

OPTIONS
ニックネームを作成したときに追加される列オプションを指示します。特定の列オプションを指定する必要のあるラッパーもあります。

ADD
列オプションを追加します。

nickname-option-name
オプションの名前を指定します。

string-constant
column-option-name の設定を、文字ストリング定数として指定します。

unique-constraint
ユニーク制約または主キー制約を定義します。

CONSTRAINT *constraint-name*
主キー制約、またはユニーク制約の名前を指定します。

UNIQUE (*column-name***,...)**
指定した列で構成されるユニーク・キーを定義します。指定する列は NOT NULL として定義されていなければなりません。各 *column-name* (列名) は、ニックネームの列を指定するものでなければなりません。また、同じ列を複数回指定することはできません。

指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。列の長さ属性が 1024 バイト以内に収まる場合でも、 LOB、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、ユニーク・キーの一部として使用できません (SQLSTATE 54008)。

ユニーク・キーにある一連の列は、主キーまたは他のユニーク・キーの一連の列と同じにすることはできません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。

カタログに記録されているニックネームの記述には、ユニーク・キーとその索引の指定が含まれます。索引の指定は、それぞれの列について昇順に指定された順序で、列に対して自動的に作成されます。索引の指定の名前は、ニックネームの作成時にスキーマに存在する既存の索引または索引の指定と競合しない場合、*constraint-name* (制約名) と同じになります。索引の指定名が競合する場合は、名前は SQL の後に文字のタイム・スタンプ (*yymmddhhmmssxxx*) が続き、スキーマ名として SYSIBM を伴う名前になります。

PRIMARY KEY (*column-name***,...)**
指定された列で構成される主キーを定義します。この文節を複数回指定することはできず、指定する列は NOT NULL として定義されていなければなりません。各 *column-name* (列名) は、ニックネームの列を指定していなければなりません。また、同じ列を複数回指定することはできません。

指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。 列の長さ属性が 1024 バイト以内に収まる場合でも、 LOB、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、主キーの一部として使用できません (SQLSTATE 54008)。

主キーの一連の列は、ユニーク・キーの一連の列と同じであってはなりません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。

1 つのニックネームには、主キーを 1 つだけ定義することができます。

カタログに記録されているニックネームの記述には、主キーとその索引の指定が含まれます。索引の指定は、それぞれの列について昇順に指定された順序で、列に対して自動的に作成されます。索引の指定の名前は、ニックネームの作成時にスキーマに存在する既存の索引または索引の指定と競合しない場合、*constraint-name* (制約名) と同じになります。索引の指定名が競合する場合は、名前は SQL の後に文字のタイム・スタンプ (*yymmddhhmmssxxx*) が続き、スキーマ名として SYSIBM を伴う名前になります。

referential-constraint
参照制約を定義します。

CONSTRAINT *constraint-name*
参照制約の名前を指定します。

FOREIGN KEY (*column-name***,...)**
指定した *constraint-name* (制約名) の参照制約を定義します。

N1 を、ステートメントの対象となるニックネームであると想定します。参照制約の外部キーは、指定された列で構成されます。列名リストの各名前は、N1 の列を指定していなければならず、同じ列を複数回指定することはできません。指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。外部キーは、255 バイトよりも大きい長さの可変長列で定義できます。
LOB、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプの列を、外部キーの一部として使用することはできません (SQLSTATE 42962)。外部キーの列の数は、親キーの列の数と同じでなければならず、対応する列のデータ・タイプは互換性があることが必要です (SQLSTATE 42830)。 2 つの列の記述は、それらの列が互換性のあるデータ・タイプ (両方の列が数字、文字ストリング、GRAPHIC、日時であるか、または同じ特殊タイプ) であれば互換性があります。

references-clause
参照制約の親表または親ニックネーム、および親キーを指定します。

REFERENCES *table-name* または *nickname*
REFERENCE 文節に指定される表またはニックネームは、カタログに記述された基本表またはニックネームを識別している必要がありますが、カタログ表を示すものであってはなりません。

参照制約の外部キー、親キー、および親表または親ニックネームが、以前に指定した参照制約の外部キー、親キー、および親表または親ニックネームと同じである場合、参照制約は重複しています。重複した参照制約は無視され、警告が戻されます (SQLSTATE 01543)。

以下の説明では、N2 は指定した親表または親ニックネームを示し、 N1 は作成する (または変更する) ニックネームを示します。N1 と N2 は同じニックネームです。

指定された外部キーの列の数は、N2 の親キーと同じ数でなければなりません。また、外部キーの *n* 番目の列の記述は、その親キーの *n* 番目の列の記述と互換性がなければなりません。この規則において、日時の列はストリング列と互換性があるとは見なされません。

FOREIGN KEY 文節で指定される参照制約は、 N2 が親であり、N1 が従属である関係を定義します。

(*column-name***,...)**
参照制約の親キーは、指定された列で構成されます。各 *column-name* は、N2 の列を指定する非修飾名でなければなりません。同じ列を重複して指定することはできません。

列名のリストは、主キーまたは N2 に存在するユニーク制約の一連の列と一致している (順序は任意) 必要があります (SQLSTATE 42890)。列名のリストの指定がない場合、N2 に主キーがある必要があります (SQLSTATE 42888)。列名リストを省略すると、指定されているとおりの順序でその主キーの列が暗黙に指定されます。

constraint-attributes
参照保全またはチェック制約に関連付けられた属性を定義します。

NOT ENFORCED
この制約は、挿入、更新、削除などの通常の操作中にデータベース・マネージャーによって課せられません。

ENABLE QUERY OPTIMIZATION
制約は真であると想定され、適切な状況下では照会最適化に使用することができます。

DISABLE QUERY OPTIMIZATION
制約を照会の最適化に使用できません。

check-constraint
チェック制約を定義します。 *check-constraint (チェック制約)* は、偽以外に評価されなければならない、または列間の機能従属関係を定義する *search-condition (検索条件)* です。

CONSTRAINT *constraint-name*
チェック制約の名前を指定します。

CHECK (*check-condition***)**
チェック制約を定義します。 *check-condition* (チェック条件) はニックネームのすべての行について、真または不明でなければなりません。

search-condition
search-condition には、以下の制限があります。

- 列参照は作成するニックネームの列に対する参照でなければなりません。
- *search-condition* に TYPE 述部を入れることはできません。
- これには、以下のどれも入れることはできません (SQLSTATE 42621)。
  - 副照会

- 逆参照操作または、有効範囲をもつ参照引き数がオブジェクト ID (OID) 列以外の列である DEREF 関数
- SCOPE 文節をもつ CAST 指定
- 列関数
- deterministic 関数でない関数
- 外部アクションをもつと定義された関数
- CONTAINS SQL または READS SQL DATA のいずれかによって定義されたユーザー定義関数
- ホスト変数
- パラメーター・マーカー
- 特殊レジスター
- ID 列以外の生成列の参照

functional-dependency
列間の機能従属関係を定義します。

列の親セットには、DETERMINED BY 文節の直前に来る指定された列が含まれます。列の子セットには、DETERMINED BY 文節の直後に来る指定された列が含まれます。 *search-condition* の制約事項すべては、親セット列と子セット列に適用され、列のセットには単純な列参照のみが許可されています (SQLSTATE 42621)。機能従属関係には同じ列を 2 度以上指定することはできません (SQLSTATE 42709)。列のデータ・タイプを LOB データ・タイプ、LOB データ・タイプに基づいた特殊タイプ、または構造化型にすることはできません (SQLSTATE 42962)。列の子セットの列を NULL 可能列にすることはできません (SQLSTATE 42621)。

column-definition の一部としてチェック制約を指定する場合、その同じ列に対してのみ列参照を行うことができます。ニックネーム定義の一部として指定されたチェック制約には、それ以前に CREATE NICKNAME ステートメントで定義されている列を指定する列参照を含めることができます。チェック制約の矛盾、重複条件、または同等条件については検査されません。したがって、矛盾したチェック制約や冗長なチェック制約が定義可能であるため、実行時にエラーになる可能性があります。

FOR SERVER *server-name*
CREATE SERVER ステートメントを使用して登録されたサーバーを指定します。このサーバーは、ニックネームのデータにアクセスするのに使用されます。

OPTIONS
ニックネームを作成したときに使用可能にされるニックネーム・オプションを指示します。

ADD
ニックネーム・オプションを追加します。

nickname-option-name
オプションの名前を指定します。

string-constant
nickname-option-name の設定を、文字ストリング定数として指定します。

注:

- リレーショナル・データ・ソース・オブジェクトの例は、表とビューです。非リレーショナル・データ・ソース・オブジェクトの例は、 Documentum オブジェクトまたは登録済み表、テキスト・ファイル (.txt)、 BLAST 検索の実行対象にできるオブジェクト、および Microsoft Excel ファイル (.xls) です。
- ニックネームで示されているデータ・ソース・オブジェクトは、*remote-object-name* の最初の修飾子によって示されているデータ・ソースに存在していなければなりません。
- サポートされるデータ・ソース・データ・タイプのリストは、ラッパーごとに異なります。 DATALINK、構造化型、および REF タイプなどの DB2 データ・タイプに対応するデータ・ソースのデータ・タイプは、どのラッパーによってもサポートされていません。 CREATE NICKNAME ステートメントに、サポートされないデータ・タイプの列を持つ *remote-object-name* を指定すると、エラーが戻されます。

  LONG VARCHAR および LONG VARGRAPHIC データ・ソースのデータ・タイプは、 CLOB および DBCLOB データ・タイプにそれぞれマップされます。 LONG VARCHAR FOR BIT DATA は BLOB にマップされます。
- DB2 索引名に許可されている最大長は 18 文字です。索引の名前がこれより長いリレーショナル表にニックネームを作成する場合、その名前の全体のカタログが作成されることはありません。 DB2 側で 18 文字に切り捨てます。そのような文字で構成されているストリングが、索引の属するスキーマ内でユニークでない場合、 DB2 は最後の文字を 0 に置き換えてユニークなストリングにしようとします。その結果もユニークでない場合は、DB2 は最後の文字を 1 に変えます。それでも駄目であれば、DB2 はこのプロセスを 2〜9 までの数字を使って続けます。必要であれば、名前の 17 番目の文字を 0〜9 まで、さらに 16 番目の文字を 0〜9 まで、というように、ユニークな名前が生成されるまで繰り返していきます。たとえば、データ・ソース表の索引を ABCDEFGHIJKLMNOPQRSTUVWXYZ とします。この索引が属するスキーマ内に、 ABCDEFGHIJKLMNOPQR および ABCDEFGHIJKLMNOPQ0 という名前がすでに存在します。新しい名前は 18 文字を超過してしまうため、 DB2 側で ABCDEFGHIJKLMNOPQR に切り捨てます。この名前はすでにスキーマ内に存在しているため、 DB2 は切り捨てた名前を ABCDEFGHIJKLMNOPQ0 に変更します。この名前も存在しているため、DB2 は切り捨てた名前を ABCDEFGHIJKLMNOPQ1 に変えます。スキーマ内にはこの名前は存在しないので、DB2 はこの名前を新しい名前として受け入れます。
- データ・ソース・オブジェクトにニックネームを作成すると、 DB2 はニックネーム列の名前をカタログに保管します。データ・ソース・オブジェクトが表またはビューの場合には、 DB2 はニックネーム列名を表またはビュー列名と同じにします。この名前の長さが DB2 列名に許可されている最大長を超える場合、DB2 は名前をこの長さに切り捨てます。切り捨てられた名前が、表またはビューにある他の列の名前と同じでユニークでない場合、 DB2 は前の段落で説明されている手順に従い、名前をユニークなものに変更します。

- リモート・データ・ソース・オブジェクトの定義が変更される場合 (たとえば、列の削除やデータ・タイプの変更)、ニックネームをドロップしてから再作成しなければなりません。そうしないと、ニックネームを SQL ステートメント内で使用するときに、エラーが戻される場合があります。

例:

例 1: HEDGES というスキーマにある、 DEPARTMENT というビューのニックネームを作成します。このビューは、OS390A という DB2 UDB for z/OS および OS/390 のデータ・ソースに保管されます。

```
CREATE NICKNAME DEPT
FOR OS390A.HEDGES.DEPARTMENT
```

例 2: 例 1 でニックネームを作成したときのビューから、すべてのレコードを選択します。このビューは、ニックネームで参照する必要があります。リモート・ビューは、パススルー・セッション時に限ってデータ・ソースで認識されるときの名前を使用して参照できます。

```
SELECT * FROM DEPT                          (ニックネーム DEPT が作成された後に有効)

SELECT * FROM OS390A.HEDGES.DEPARTMENT      (無効)
```

例 3: salesdata という名前のスキーマにあるリモート表 JAPAN のニックネームを作成します。データ・ソース上のスキーマ名と表名は小文字で保管されるため、リモート・スキーマ名と表名には二重引用符を付けて指定します。

```
CREATE NICKNAME JPSALES
  FOR asia."salesdata"."japan"
```

例 4: 表構造のファイル DRUGDATA1.TXT のニックネームを作成します。ステートメントに FILE_PATH、COLUMN DELIMITER、KEY_COLUMN、および VALIDATE_DATA_FILE ニックネーム・オプションを組み込みます。

```
CREATE NICKNAME DRUGDATA1
  (Dcode         INTEGER,
  DRUG           CHAR(20),
  MANUFACTURER   CHAR(20))
   FOR SERVER biochem_lab
   OPTIONS
    (FILE_PATH '/usr/pat/DRUGDATA1.TXT',
     COLUMN_DELIMITER ',',
     KEY_COLUMN 'DCODE',
     SORTED 'Y',
     VALIDATE_DATA_FILE 'Y')
```

例 5: 指定したディレクトリー・パス /home/db2user にある複数の XML ファイルに対して親ニックネーム CUSTOMERS を作成します。以下のオプションを組み込みます。

- 列オプション:
  - ID という名前の VARCHAR(5) 列に XPATH 列オプションを指定し、列データを抽出する XML ファイルのエレメントまたは属性を指示します。
  - NAME という名前の VARCHAR(16) 列に XPATH 列オプションを指定し、列データを抽出する XML ファイルのエレメントまたは属性を指示します。
  - ADDRESS という名前の VARCHAR(30) 列に XPATH 列オプションを指定し、列データを抽出する XML ファイルのエレメントまたは属性を指示します。

- CID という名前の VARCHAR(16) 列に PRIMARY_KEY 列オプションを指定し、ニックネームの階層で親ニックネームとなるカスタマー・ニックネームを指示します。

- ニックネーム・オプション:
  - DIRECTORY_PATH ニックネーム・オプションは、データを提供する XML ファイルの場所を指示します。
  - XPATH ニックネーム・オプションは、XML ファイル内でデータが始まるエレメントを指示します。
  - STREAMING ニックネーム・オプションは、XML ソース・データがエレメントで区切られており、エレメントごとに処理されることを指示します。この例では、エレメントはカスタマー・レコードです。

```
CREATE NICKNAME customers
  (id      VARCHAR(5)   OPTIONS(XPATH './@id'),
   name    VARCHAR(16)  OPTIONS(XPATH './/name'),
   address VARCHAR(30)  OPTIONS(XPATH './/address/@street'),
   cid     VARCHAR(16)  OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS
     (DIRECTORY_PATH '/home/db2user',
      XPATH '//customer',
      STREAMING 'YES')
```

関連資料:

- 347 ページの『CREATE TABLE』
- 27 ページの『ALTER NICKNAME』
- 342 ページの『CREATE SERVER』
- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムのニックネーム列オプション』
- フェデレーテッド・システム・ガイド の『有効なデータ・ソース・オブジェクト』

CREATE PROCEDURE

CREATE PROCEDURE ステートメントは、アプリケーション・サーバーによってプロシージャーを定義します。

このステートメントを使用して作成できるプロシージャーには、異なる 2 つのタイプがあります。これらのそれぞれについて、個々に説明します。

- 外部。このプロシージャーはプログラミング言語で書かれています。 外部実行可能ファイルは、アプリケーション・サーバーやプロシージャーの様々な属性によって定義されたプロシージャーにより参照されます。
- SQL。このプロシージャーは SQL で書かれています。 プロシージャー本体は、アプリケーション・サーバーやプロシージャーの様々な属性によって定義されます。

関連資料:

- 312 ページの『CREATE PROCEDURE (外部)』
- 327 ページの『CREATE PROCEDURE (SQL)』

CREATE PROCEDURE (外部)

CREATE PROCEDURE (外部) ステートメントは、アプリケーション・サーバーで外部プロシージャーを定義するのに使用します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATE_EXTERNAL_ROUTINE 権限、および以下の少なくとも 1 つ。
  - データベースに対する IMPLICIT_SCHEMA 権限 (プロシージャーのスキーマ名が既存のスキーマを指していない場合)
  - スキーマに対する CREATEIN 特権 (プロシージャーのスキーマ名が既存のスキーマを指している場合)

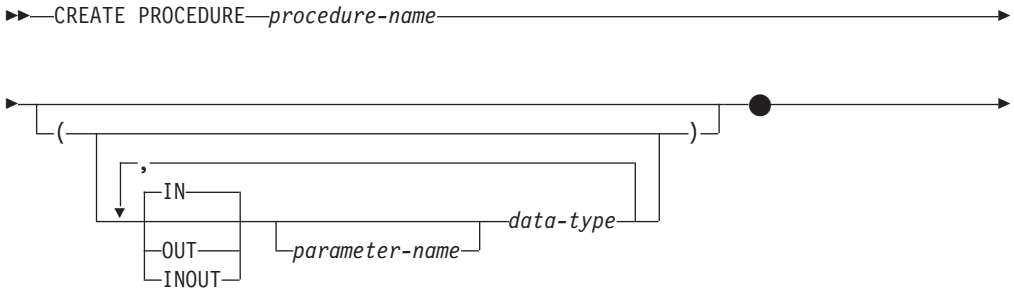
非 fenced のストアード・プロシージャーを作成するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

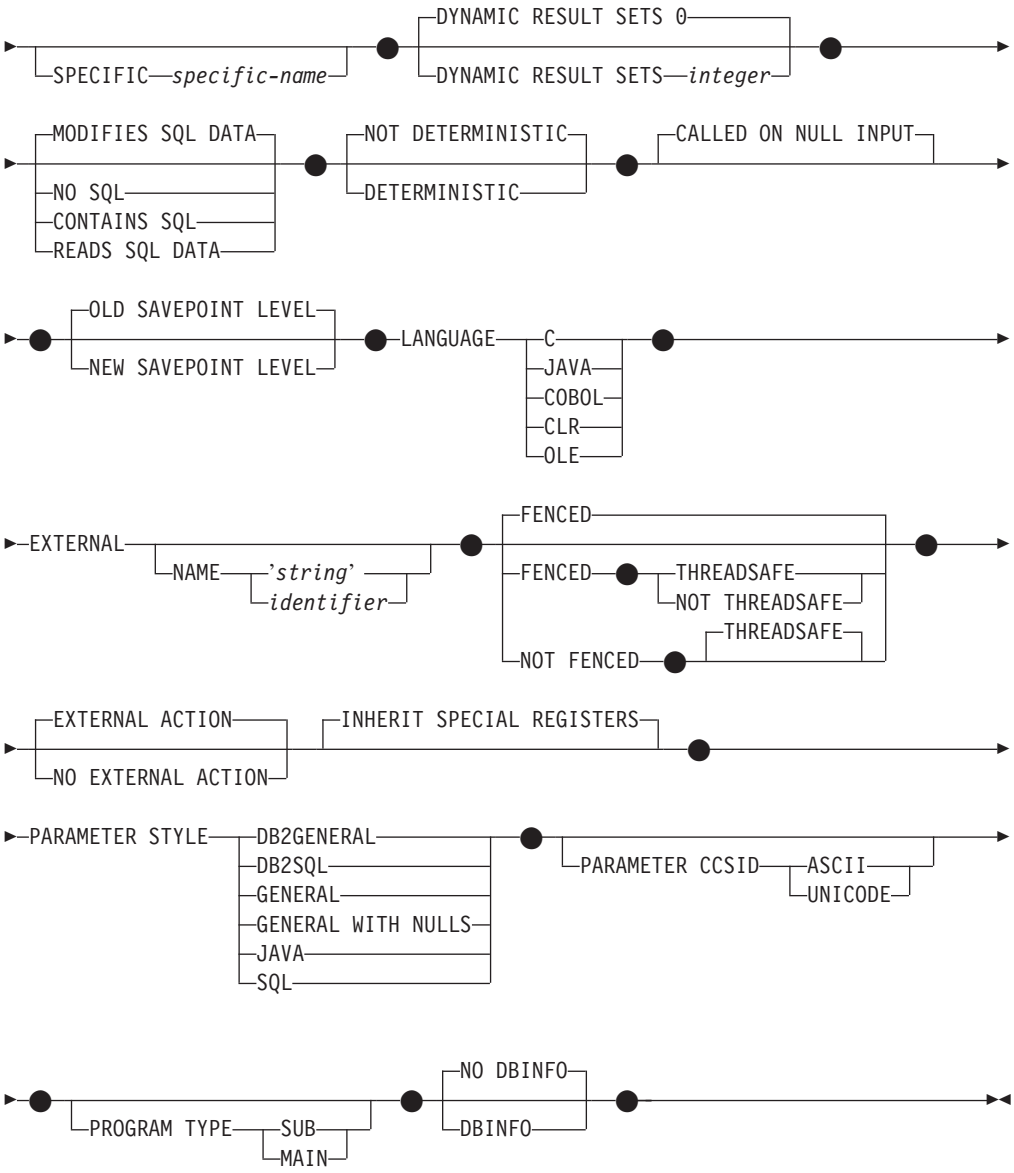
- データベースに対する CREATE_NOT_FENCED_ROUTINE 権限
- SYSADM または DBADM 権限

fenced ストアード・プロシージャーを作成する場合、追加の権限や特権は必要ありません。

許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文:





説明:

procedure-name

定義するプロシージャーの名前を指定します。この名前は、プロシージャーを指定する修飾または非修飾の名前です。 *procedure-name* (プロシージャー名) の非修飾形式は SQL ID です (最大長 128)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

暗黙または明示の修飾子を含む名前と、パラメーターの数との組み合わせは、カタログにすでに記述されているプロシージャーを指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数との組み合わせは、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、 'SYS' で始まる *schema-name* (スキーマ名) は使用できません (SQLSTATE 42939)。

(IN | OUT | INOUT *parameter-name data-type,...)*
プロシージャーのパラメーターを指定し、各パラメーターのモード、データ・タイプ、およびオプション名を指定します。このリストには、プロシージャーが予期する各パラメーターごとに 1 つの項目を指定する必要があります。

1 つのスキーマに同じ名前の 2 つのプロシージャーがある場合、パラメーターの数をまったく同一にすることはできません。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

たとえば、次のステートメントの場合、2 番目のステートメントは失敗します。その理由は、データ・タイプが異なっていてもプロシージャーのパラメーターの数が同じだからです。

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

IN パラメーターをプロシージャーの入力パラメーターとして指定します。 プロシージャー内でパラメーターに加えられるすべての変更は、制御が戻されると SQL アプリケーションの呼び出しは行なえなくなります。デフォルトは IN です。

OUT
パラメーターをプロシージャーの出力パラメーターとして指定します。

INOUT
パラメーターを、プロシージャーの入力および出力パラメーターの両方として指定します。

parameter-name
パラメーターの名前を任意に指定します。パラメーター名は、プロシージャーでユニークでなければなりません (SQLSTATE 42734)。

data-type
パラメーターのデータ・タイプを指定します。

- CREATE TABLE ステートメントのデータ・タイプ の定義に指定することが可能で、プロシージャーの作成に使用されている言語に対応するものがある SQL データ・タイプ指定と省略形を指定できます。
- ユーザー定義データ・タイプはサポートされていません (SQLSTATE 42601)。
- LONG VARCHAR および LONG VARGRAPHIC データ・タイプは、外部プロシージャーのパラメーター・タイプとしてサポートされていません。
- CLR は 28 より大きい DECIMAL スケールをサポートしていません (SQLSTATE 42613)。

SPECIFIC *specific-name*
定義するプロシージャーのインスタンスに対するユニーク名を指定します。この特定名は、このプロシージャーをドロップする場合、またはこのプロシージャーにコメントを付ける場合に使用することができます。これは、プロシージャーの呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続

きます。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別のルーチン・インスタンスを指定するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) になります。

specific-name は、既存の *procedure-name* と同じでも構いません。

修飾子を指定しない場合、*procedure-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニークな名前は、 SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmsshhn)。

DYNAMIC RESULT SETS *integer*
ストアード・プロシージャーから戻される結果セットの上限の見積もりを指定します。

NO SQL、CONTAINS SQL、READS SQL DATA、MODIFIES SQL DATA
ストアード・プロシージャーから SQL ステートメントが発行されるかどうかと、もし発行されればどのタイプかを示します。

NO SQL
ストアード・プロシージャーはどの SQL ステートメントも実行できないことを指示します (SQLSTATE 38001)。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、ストアード・プロシージャーで実行できることを指定します (SQLSTATE 38004)。どのストアード・プロシージャーでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、ストアード・プロシージャーで実行できることを指定します (SQLSTATE 38002 または 42985)。どのストアード・プロシージャーでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003)。

MODIFIES SQL DATA
このストアード・プロシージャーは、ストアード・プロシージャーでサポートされていないステートメント以外のすべての SQL ステートメントを実行できることを指定します (SQLSTATE 38003)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は、同一の引き数値に対してプロシージャーが常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存してプロシージャーの結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC が指定されたプロシージャーは、同じ入力を指定して正しく呼び出した場合に常に同じ結果を戻します。

現在、この文節はストアード・プロシージャーの処理に影響を与えません。

CALLED ON NULL INPUT
CALLED ON NULL INPUT は、ストアード・プロシージャーに常に適用されます。これは、任意の引き数が NULL かどうかにかかわりなく、ストアード・プ

ロシージャーが呼び出されることを意味します。 OUT または INOUT パラメーターは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。 NULL の引き数値の有無のテストはストアード・プロシージャーで行う必要があります。

OLD SAVEPOINT LEVEL または **NEW SAVEPOINT LEVEL**
このストアード・プロシージャーが、セーブポイント名と影響について新しいセーブポイント・レベルを設定するかどうかを指定します。 OLD SAVEPOINT LEVEL がデフォルトの動作です。セーブポイント・レベルについて詳しくは、SAVEPOINT ステートメントの説明にある『規則』のセクションを参照してください。

LANGUAGE
この文節は必須で、ストアード・プロシージャーの本体が準拠している言語インターフェース規則を指定するのに使用されます。

C データベース・マネージャーは、ストアード・プロシージャーを C プロシージャーであるかのように呼び出します。ストアード・プロシージャーは、標準 ANSI C プロトタイプで定義されている C 言語の呼び出し規則およびリンケージ規則に準拠していなければなりません。

JAVA
データベース・マネージャーは、 Java クラス内のメソッドとしてストアード・プロシージャーを呼び出します。

COBOL
データベース・マネージャーは、プロシージャーを COBOL プロシージャーであるかのように呼び出します。

CLR
データベース・マネージャーは、 .NET クラス内のメソッドとしてストアード・プロシージャーを呼び出します。 LANGUAGE CLRは、Windowsオペーレーティング・システム上で実行するストアード・プロシージャーのみサポートされます。 NOT FENCED は CLR ルーチンに指定できません (SQLSTATE 42601)。

OLE
データベース・マネージャーは、 OLE 自動化オブジェクトによって公開されたメソッドであるものとしてストアード・プロシージャーを呼び出します。ストアード・プロシージャーは、OLE 自動化データ・タイプと呼び出しメカニズムに準拠している必要があります。さらに OLE 自動化オブジェクトは、プロセス内サーバー (DLL) としてインプリメントされる必要もあります。これらの制約事項については、*OLE Automation Programmer's Reference* で説明されています。

LANGUAGE OLE は、 DB2 (Windows オペレーティング・システム版) に保管されているストアード・プロシージャーに対してのみサポートされます。 LANGUAGE OLE を指定したプロシージャーには、THREADSAFE は指定できません (SQLSTATE 42613)。

EXTERNAL
この文節は、この CREATE PROCEDURE ステートメントを使用して登録する

新しいプロシージャーが、外部プログラミング言語で作成されたコードに基づいており、文書化されたリンケージの規則とインターフェースに従っていることを示します。

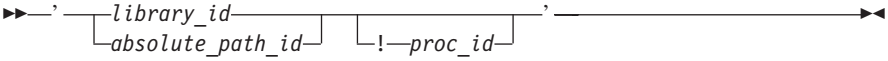
NAME 文節の指定がない場合、『NAME *procedure-name*』 が想定されます。
NAME 文節のフォーマットが正しくない場合、エラーが戻されます (SQLSTATE 42878)。

NAME *'string'*
この文節は、定義するプロシージャーをインプリメントするユーザー作成コードの名前を指定します。

'string' オプションは、最大 254 文字のストリング定数です。 ストリングに使用される形式は、指定した LANGUAGE によって異なります。

- LANGUAGE C の場合

  指定する *string* は、ライブラリー名と作成しているストアード・プロシージャーを実行するためにデータベース・マネージャーが呼び出すそのライブラリー中のプロシージャーです。ライブラリー (およびそのライブラリー中のプロシージャー) は、 CREATE PROCEDURE ステートメントの実行時に存在している必要はありません。ただし、プロシージャーが呼び出される時点では、該当のライブラリーとそのライブラリー中の該当のプロシージャーは存在していなければならず、またデータベース・サーバーのマシンからアクセス可能でなければなりません。



  名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

  library_id
  該当のプロシージャーが入っているライブラリーの名前を指定します。データベース・マネージャーは、次のようにしてこのライブラリーを特定します。

  – UNIX 系システムの場合、*library_id* が 'myfunc' と指定されており、データベース・マネージャーが /u/production から実行されていると、データベース・マネージャーは FENCED が指定されていればライブラリー /u/production/sqllib/function/myproc で、 NOT FENCED が指定されていればライブラリー /u/production/sqllib/function/unfenced/myproc でプロシージャーを特定します。
  – Windows オペレーティング・システムの場合、データベース・マネージャーは LIBPATH または PATH 環境変数に指定されているディレクトリー・パスから関数を特定します。

  これらのディレクトリーのいずれかに存在しているストアード・プロシージャーは、登録済み属性を使用しません。

  absolute_path_id
  プロシージャーの絶対パス名を指定します。

たとえば、UNIX 系システムの場合、'/u/jchui/mylib/myproc' を指定すると、データベース・マネージャーは /u/jchui/mylib を調べて myproc プロシージャーを調べます。

Windows オペレーティング・システムの場合、'd:¥mylib¥myproc.dll' を指定すると、データベース・マネージャーは d:¥mylib ディレクトリーからダイナミック・リンク・ライブラリー myproc.dll をロードします。絶対パス ID がルーチン本体の識別に使用されている場合は、.dll 拡張子を必ず付加してください。

! proc_id
呼び出すプロシージャーの入り口点の名前を指定します。感嘆符 (!) は、ライブラリー ID とプロシージャー ID との間の区切り文字です。 *! proc_id* を省略すると、データベース・マネージャーはライブラリーのリンク時に確立されたデフォルトの入り口点を使用します。

たとえば、'!proc8' を指定すると、データベース・マネージャーはシステム規則に基づいて指定されたプロシージャー ID を調べて、そのライブラリー内の入り口点 proc8 を使用します。

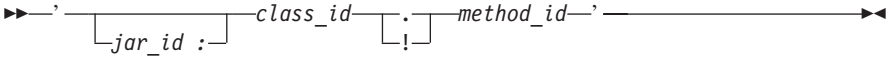
同様に、'!proc8' を指定すると、データベース・マネージャーは *absolute_path_id* によって指定されたライブラリーを調べて、そのライブラリー内の入り口点 proc8 を使用します。

ストリングの形式が正しくない場合には、エラーが戻されます (SQLSTATE 42878)。

ストアード・プロシージャーの本体は、マウントされてデータベースのすべてのパーティションで使用可能なディレクトリーに入っていなければなりません。

- LANGUAGE JAVA の場合:

  指定する *string* には、作成中のストアード・プロシージャーを実行するためにデータベース・マネージャーが呼び出す、任意指定の jar ファイル、クラス ID、およびメソッド ID が含まれています。クラス ID とメソッド ID は、 CREATE PROCEDURE ステートメントの実行時には存在している必要はありません。 *jar_id* を指定する場合、ID は、CREATE PROCEDURE ステートメントの実行時に存在していなければなりません。ただし、プロシージャーを呼び出す時点では、該当のクラス ID とメソッド ID が存在し、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラーが戻されます (SQLSTATE 42884)。



名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

jar_id
jar の集合をデータベースへインストールしたときに、その jar の集合に付けられた jar ID を指定します。これは、単純 ID またはスキ

ーマ修飾 ID のいずれかにすることができます。たとえば、'myJar' や 'mySchema.myJar' のようになります。

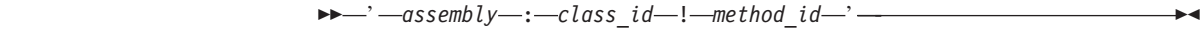
class_id
Java オブジェクトのクラス ID を指定します。クラスがパッケージの一部である場合、クラス ID の一部に完全なパッケージ接頭部 (たとえば、 'myPacks.StoredProcs') が含まれている必要があります。 Java 仮想マシンは、ディレクトリー '../myPacks/StoredProcs/' の中のクラスを探します。 Windows オペレーティング・システムでは、 Java 仮想マシンはディレクトリー '..¥myPacks¥StoredProcs¥' を探索します。

method_id
呼び出す Java クラスのメソッド名を指定します。

- LANGUAGE CLR の場合:

指定された *string* は、作成するプロシージャーを実行するためにデータベース・マネージャーが呼び出す .NET アセンブリー (ライブラリーまたは実行可能モジュール)、そのアセンブリー内のクラス、およびそのクラス内のメソッドを表します。モジュール、クラス、およびメソッドは、CREATE PROCEDURE ステートメントの実行時に存在している必要はありません。ただし、プロシージャーを呼び出す時点では、モジュール、クラス、およびメソッドは存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラーが戻されます (SQLSTATE 42284)。

'/clr' コンパイラー・オプションで管理対象コード拡張を指定してコンパイルされている C++ ルーチンは、 'LANGUAGE C' ではなく 'LANGUAGE CLR' としてカタログする必要があります。 DB2 は、必要な実行時の決定を行えるようにするために、 .NET インフラストラクチャーがストアード・プロシージャー内で使用されていることを認識している必要があります。 .NET インフラストラクチャーを使用するすべてのストアード・プロシージャーは、 'LANGUAGE CLR' としてカタログする必要があります。



名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

assembly
クラスを含む DLL ファイルまたは他のアセンブリー・ファイルを指定します。ファイル拡張子 (.dll など) まで指定します。絶対パス名を指定しない場合、ファイルは DB2 インストール・パスの関数ディレクトリー (たとえば、 c:¥sqllib¥function) にあるものとされます。ファイルがインストール関数ディレクトリーのサブディレクトリーにある場合は、絶対パスを指定せずに、ファイル名の前にサブディレクトリーを指定します。たとえば、インストール・ディレクトリーが c:¥sqllib であり、アセンブリー・ファイルが c:¥sqllib¥function¥myprocs¥mydotnet.dll であるなら、アセンブリーの指

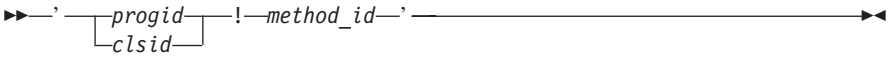
定は 'myprocs¥mydotnet.dll' とするだけで十分です。このパラメーターの大文字小文字が区別されるかどうかは、ファイル・システムの設定と同じです。

class_id
呼び出すメソッドが属するアセンブリー内のクラスの名前を指定します。クラスがネーム・スペース内にある場合は、クラスだけでなく絶対ネーム・スペースも指定することが必要です。たとえば、クラス EmployeeClass がネーム・スペース MyCompany.ProcedureClasses にあるのであれば、 MyCompany.ProcedureClasses.EmployeeClass をクラスとして指定します。一部の .NET 言語用のコンパイラーはクラスのネーム・スペースとしてプロジェクト名を追加するため、コマンド行コンパイラーと GUI コンパイラーのどちらを使用するかで動作が異なってくるので注意してください。このパラメーターには、大文字と小文字の区別があります。

method_id
指定したクラス内で呼び出されるメソッドを指定します。このパラメーターには、大文字と小文字の区別があります。

- LANGUAGE OLE の場合

指定するストリングは、ステートメントが作成しているストアード・プロシージャーを実行するためにデータベース・マネージャーが呼び出す OLE のプログラム ID (*progid*) またはクラス ID (*clsid*)、およびメソッド ID (*method_id*) です。プログラム ID またはクラス ID、およびメソッド ID は、 CREATE PROCEDURE ステートメントの実行時に存在している必要はありません。ただし、プロシージャーを CALL ステートメントで使用する時点で、メソッド ID は存在していなければならず、データベース・サーバーのマシンからアクセス可能でなければなりません。そうでない場合、エラー (SQLSTATE 42724) になります。



名前は、単一引用符で囲む必要があります。余分なブランクを使用することはできません。

progid
OLE オブジェクトのプログラム ID を指定します。

progid は、データベース・マネージャーには解釈されず、ランタイムに OLE に転送されるだけです。指定する OLE オブジェクトは、作成可能である必要があり、実行時バインディング (ディスパッチに基づくバインディングとも呼ばれる) をサポートしている必要があります。規約では、progid は次のような形式になります。

```
<program_name>.<component_name>.<version>
```

これは規約でしかなく、厳密な規則ではないので、 *progids* をこれとは異なる形式にしてもかまいません。

clsid
作成する OLE オブジェクトのクラス ID を指定します。 OLE オブ

ジェクトが progid を指定して登録されていない場合に、 progid を指定する代わりに使用することができます。 *clsid* の形式は次のとおりです。

```
{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}
```

ここで 'n' は英数字です。 *clsid* は、データベース・マネージャーには解釈されず、ランタイムに OLE API に転送されるだけです。

method_id
呼び出す OLE オブジェクトのメソッド名を指定します。

NAME *identifier*
指定する *identifier* は SQL ID です。 SQL ID は、ストリングの *library-id* として使用されます。区切られた ID でない場合、ID は大文字に変換されます。 ID がスキーマ名で修飾されている場合、スキーマ名の部分は無視されます。この形式の NAME は、LANGUAGE C でのみ使用可能です。

FENCED または **NOT FENCED**
この文節は、ストアード・プロシージャーをデータベース・マネージャーのオペレーティング環境のプロセスまたはアドレス・スペースで実行しても 『安全』か (NOT FENCED)、否か (FENCED) を指定します。

ストアード・プロシージャーが FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を保護して、そのプロシージャーからアクセスされないようにします。 すべてのプロシージャーは、 FENCED として実行するか NOT FENCED として実行するかの選択が可能です。一般に、FENCED として実行されるプロシージャーは、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
十分に検査されていないプロシージャーに NOT FENCED を使用すると、DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ストアード・プロシージャーが使用される場合には、完全な保全性を確保できません。

ストアード・プロシージャーを NOT FENCED として登録するには、SYSADM 権限、 DBADM 権限、または CREATE_NOT_FENCED 権限が必要です。 LANGUAGE OLE または NOT THREADSAFE を指定したストアード・プロシージャーには、FENCED のみを指定できます。

NOT FENCED 文節を指定している場合は、LANGUAGE CLR ストアード・プロシージャーを作成できません (SQLSTATE 42601)。

THREADSAFE または **NOT THREADSAFE**
プロシージャーを他のルーチンと同じプロセスで実行しても安全か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

プロシージャーが OLE 以外の LANGUAGE で定義される場合:

- プロシージャーが THREADSAFE として定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスにプロシージャーを呼び出すことができます。一般に、スレッド・セーフになるには、プロシージャーはどのグローバルあるいは静的データ域をも使用してはなりません。多く

のプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED プロシージャーの両方を THREADSAFE にすることができます。

- プロシージャーが NOT THREADSAFE に定義される場合には、データベース・マネージャーは他のルーチンと同じプロセスにプロシージャーを決して呼び出しません。

FENCED プロシージャーの場合、LANGUAGE が JAVAまたは CLR なら THREADSAFE がデフォルトです。これ以外のすべての言語の場合は、NOT THREADSAFE がデフォルトです。プロシージャーが LANGUAGE OLE に定義される場合には、THREADSAFE は指定できません (SQLSTATE 42613)。

NOT FENCED プロシージャーの場合には、THREADSAFE がデフォルトです。NOT THREADSAFE を指定することはできません (SQLSTATE 42613)。

EXTERNAL ACTION または **NO EXTERNAL ACTION**

プロシージャーが、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを取るか (EXTERNAL ACTION)、または取らないか (NO EXTERNAL ACTION) を指定します。デフォルトは EXTERNAL ACTION です。 NO EXTERNAL ACTION を指定した場合、プロシージャーが外部に影響を与えないことを前提とした最適化を、システムは使用できます。

INHERIT SPECIAL REGISTERS

このオプション文節は、プロシージャーの更新可能な特殊レジスターが、呼び出しステートメントの環境からの初期値を継承するよう指定します。

特殊レジスターに対する変更が、プロシージャーの呼び出し元に戻されることはありません。

更新不能の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、デフォルト値に設定されます。

PARAMETER STYLE

この文節は、ストアード・プロシージャーとの間でパラメーターを渡し、値を戻すのに用いる規則を指定するのに使用されます。

DB2GENERAL

ストアード・プロシージャーは、 Java メソッドを使用するために定義された規則に従ったパラメーターの受け渡し規則を使用します。これは、LANGUAGE JAVA を使用する場合にだけ指定する必要があります。

DB2SQL

CALL ステートメントのパラメーターの他に、以下の引き数がストアード・プロシージャーに渡されます。

- CALL ステートメントの各パラメーターの NULL 標識を含むベクトル
- DB2 へ戻される SQLSTATE
- ストアード・プロシージャーの修飾名
- ストアード・プロシージャーの特定名
- DB2 へ戻される SQL 診断ストリング

これは、LANGUAGE C、COBOL、CLR、または OLE を使用する場合にだけ、指定することができます。

GENERAL
ストアード・プロシージャーは、パラメーター受け渡しメカニズムを使用します。ここでは、ストアード・プロシージャーは CALL で指定したパラメーターを受け取ります。パラメーターは言語ごとに直接に渡されることになっているので、SQLDA 構造は使われません。これは、LANGUAGE C、COBOL、または CLR を使用する場合にだけ、指定することができます。

NULL 標識がプログラムに直接渡されることはありません。

GENERAL WITH NULLS
GENERAL で指定した CALL ステートメントのパラメーターの他に、別の引き数がストアード・プロシージャーに渡されます。この別の引き数は、CALL ステートメントの各パラメーター用の、NULL 標識のベクトルです。これは、C では短精度整数の配列になります。これは、LANGUAGE C、COBOL、または CLR を使用する場合にだけ、指定することができます。

JAVA
ストアード・プロシージャーは、 Java 言語および SQLJ ルーチンの仕様に準拠する規則に従ったパラメーターの受け渡し規則を使用します。 IN/OUT および OUT パラメーターは、戻り値を処理するために単一項目配列として渡されます。これは、LANGUAGE JAVA を使用する場合にだけ指定する必要があります。

PARAMETER STYLE JAVA プロシージャーでは、 DBINFO または PROGRAM TYPE 文節はサポートされていません。

SQL
CALL ステートメントのパラメーターの他に、以下の引き数がストアード・プロシージャーに渡されます。

- CALL ステートメントの各パラメーターの NULL 標識
- DB2 へ戻される SQLSTATE
- ストアード・プロシージャーの修飾名
- ストアード・プロシージャーの特定名
- DB2 へ戻される SQL 診断ストリング

これは、LANGUAGE C、COBOL、CLR、または OLE を使用する場合にだけ、指定することができます。

PARAMETER CCSID
プロシージャーとやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、

PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。プロシージャーが呼び出されるときのアプリケーション・コード・ページはデータベース・コード・ページです。

UNICODE
ストリング・データが Unicode でエンコードされることを指定します。データベースが Unicode データベースの場合、文字データは UTF-8、GRAPHIC データは UCS-2 になります。データベースが Unicode データベースでない場合は、文字データは UTF-8 になります。いずれの場合も、プロシージャーが呼び出されるときのアプリケーション・コード・ページは 1208 です。

データベースが Unicode データベースではないのに、 PARAMETER CCSID UNICODE を指定したプロシージャーを作成すると、そのプロシージャーは GRAPHIC タイプやユーザー定義タイプを取ることができません (SQLSTATE 560C1)。 PARAMETER CCSID UNICODE プロシージャーは、 DB2 バージョン 8.1 以降のクライアントからのみ呼び出すことができます (SQLSTATE 42997)。

データベースが Unicode ではなく、データベース構成に代替照合シーケンスが指定されている場合、 PARAMETER CCSID ASCII または PARAMETER CCSID UNICODE を指定したプロシージャーを作成できます。プロシージャーとやり取りされるすべてのデータは、適切なコード・ページに変換されます。

この文節を LANGUAGE OLE、LANGUAGE JAVA、または LANGUAGE CLR とともに指定することはできません (SQLSTATE 42613)。

PROGRAM TYPE
ストアード・プロシージャーでのパラメーターのスタイルが、メインルーチンなのかサブルーチンなのかを指定します。デフォルトは SUB です。

SUB
ストアード・プロシージャーのパラメーターは、別々の引き数として渡されます。

MAIN
ストアード・プロシージャーのパラメーターは、引き数カウンター、および引き数のベクトルとして渡されます (argc、argv)。呼び出すストアード・プロシージャーの名前も、″main″ となります。このタイプのストアード・プロシージャーは、独立した実行可能ファイルではなく、共用ライブラリーと同じ方法で作成する必要があります。PROGRAM TYPE MAIN は、LANGUAGE 文節に C、COBOL、または CLR のいずれかが指定されている場合にのみ有効です。

DBINFO または **NO DBINFO**
DB2 において既知である特定の情報が呼び出されたときに、その情報を追加の呼び出し時引き数としてストアード・プロシージャーに渡すか (DBINFO)、または渡さないか (NO DBINFO) を指定します。 NO DBINFO がデフォルト値です。 DBINFO は、LANGUAGE OLE ではサポートされません (SQLSTATE 42613)。これは PARAMETER STYLE JAVA、または DB2GENERAL でもサポートされません。

DBINFO を指定すると、以下の情報を含む構造がストアード・プロシージャーに渡されます。

- データベース名 - 現在接続されているデータベースの名前。
- アプリケーション ID - データベースへの接続ごとに確立された、ユニークなアプリケーション ID。
- アプリケーション許可 ID - アプリケーション・ランタイムの許可 ID。
- コード・ページ - データベースのコード・ページを識別します。
- データベースのバージョン/リリース - ストアード・プロシージャーを呼び出すデータベース・サーバーのバージョン、リリース、および修正レベルを識別します。
- プラットフォーム - サーバーのプラットフォーム・タイプが入ります。

DBINFO 構造はすべての外部ルーチンで共通で、プロシージャーに関係ない追加のフィールドを含みます。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - ASUTIME NO LIMIT
      - COMMIT ON RETURN NO
      - NO COLLID
      - STAY RESIDENT NO
      - Unicode データベースでの CCSID UNICODE
      - PARAMETER CCSID UNICODE が指定されていない場合、非 Unicode データベース内での CCSID ASCII
  - 以前のバージョンの DB2 との互換性:
    - DYNAMIC RESULT SETS の代わりに RESULT SETS を指定できます。
    - CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
    - DB2GENERAL の代わりに DB2GENRL を指定できます。
    - GENERAL の代わりに SIMPLE CALL を指定できます。
    - GENERAL WITH NULLS の代わりに SIMPLE CALL WITH NULLS を指定できます。
    - PARAMETER STYLE DB2DARI はサポートされています。
- まだ存在していないスキーマ名を用いてプロシージャーを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- NOT FENCED として定義される Java ルーチンは、FENCED THREADSAFE として定義されているかのように呼び出されます。
- 動的コンパウンド・ステートメント内から呼び出されるプロシージャーは、プロシージャー作成時に OLD SAVEPOINT LEVEL が指定またはデフォルト設定されていたとしても、 NEW SAVEPOINT LEVEL を指定して作成されたかのように実行されます。

- ***特権***
  - プロシージャーの定義者は、プロシージャーに対する WITH GRANT OPTION 付きの EXECUTE 特権と、プロシージャーをドロップする権利を常に与えられます。
  - プロシージャーを SQL ステートメントで使用する時点で、プロシージャーの定義者はそのプロシージャーによって使用されるすべてのパッケージに対して EXECUTE 特権を持っていなければなりません。

例:

例 1: Java で書かれたストアード・プロシージャーのプロシージャー定義を作成します。このプロシージャーは、パーツ番号を渡されて、パーツの価格と現在入手可能な数量を戻します。

```
CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER,
    OUT COST DECIMAL(7,2),
    OUT QUANTITY INTEGER)
EXTERNAL NAME 'parts.onhand'
LANGUAGE JAVA PARAMETER STYLE JAVA
```

例 2: C で書かれたストアード・プロシージャーのプロシージャー定義を作成します。このプロシージャーは、部品番号を渡されて、部品を構成するパーツの数とパーツの合計価格、およびパーツ番号、数量、各パーツの単価をリストする結果セットを戻します。

```
CREATE PROCEDURE ASSEMBLY_PARTS (IN ASSEMBLY_NUM INTEGER,
    OUT NUM_PARTS INTEGER,
    OUT COST DOUBLE)
  EXTERNAL NAME 'parts!assembly'
  DYNAMIC RESULT SETS 1 NOT FENCED
  LANGUAGE C PARAMETER STYLE GENERAL
```

関連資料:

- 704 ページの『SAVEPOINT』
- *SQL リファレンス 第 1 巻* の『ルーチンで使用可能な SQL ステートメント』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』

関連サンプル:

- 『spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)』
- 『spcreate.db2 -- Catalog the DB2 CLI stored procedures contained in spserver.c (CLI)』
- 『SpCreate.db2 -- How to catalog the stored procedures contained in SpServer.java』
- 『SpCreate.db2 -- How to catalog the stored procedures contained in SpServer.sqlj』

CREATE PROCEDURE (SQL)

CREATE PROCEDURE (SQL) ステートメントは、アプリケーション・サーバーで SQL プロシージャーを定義するのに使用します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

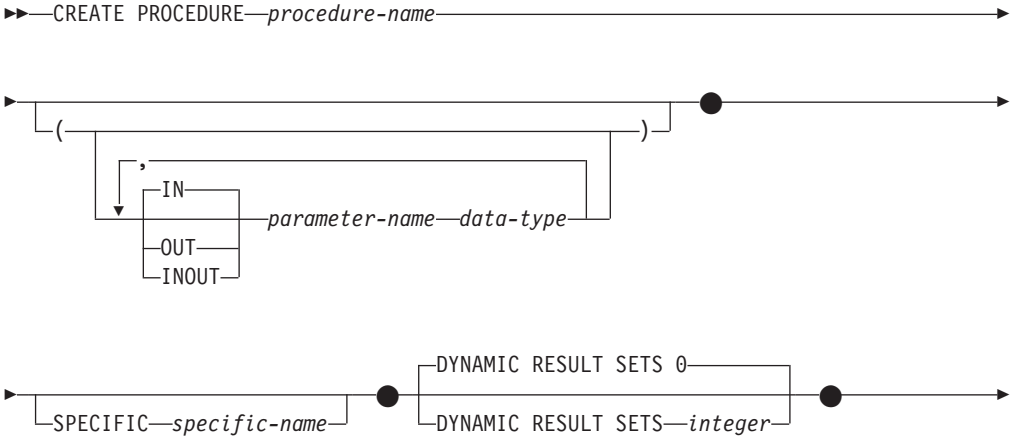
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

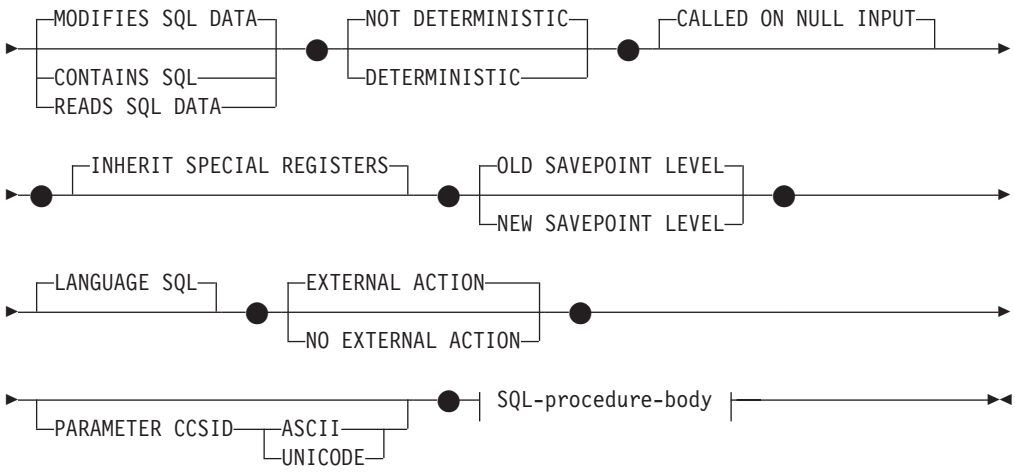
- SYSADM または DBADM 権限
- データベースに対する BINDADD 特権、および以下のどちらか。
  - データベースに対する IMPLICIT_SCHEMA 特権 (プロシージャーの暗黙または明示的なスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (プロシージャーのスキーマ名が既存のスキーマを指している場合)

このステートメントの許可 ID に SYSADM 権限または DBADM 権限がない場合には、ステートメントの許可 ID が持つ特権に、プロシージャー本体で指定される SQL ステートメントを呼び出すのに必要なすべての特権が含まれていることも必要です。

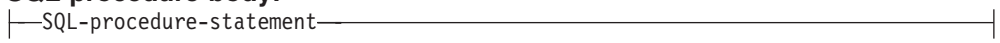
許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) が戻されます。

構文:





SQL-procedure-body:



説明:

procedure-name
定義するプロシージャーの名前を指定します。この名前は、プロシージャーを指定する修飾または非修飾の名前です。 *procedure-name* (プロシージャー名) の非修飾形式は SQL ID です (最大長 128)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。

暗黙または明示の修飾子を含む名前と、パラメーターの数との組み合わせは、カタログにすでに記述されているプロシージャーを指定するものであってはなりません (SQLSTATE 42723)。非修飾名とパラメーターの数との組み合わせは、そのスキーマ内ではユニークですが、複数のスキーマ間でユニークである必要はありません。

2 つの部分からなる名前を指定する場合、*schema-name* を 'SYS' で始めることはできません。違反すると、エラーが戻されます (SQLSTATE 42939)。

*(***IN | OUT | INOUT** *parameter-name data-type,...)*
プロシージャーのパラメーターを指定し、各パラメーターのモード、名前、およびデータ・タイプを指定します。このリストには、プロシージャーが予期する各パラメーターごとに 1 つの項目を指定する必要があります。

パラメーターのないプロシージャーも登録可能です。この場合、指定するデータ・タイプがない場合でも、括弧はコーディングする必要があります。たとえば以下のようにします。

```
CREATE PROCEDURE SUBWOOFER() ...
```

1 つのスキーマに同じ名前の 2 つのプロシージャーがある場合、パラメーターの数をまったく同一にすることはできません。シグニチャーが重複していると、SQL エラー (SQLSTATE 42723) になります。

たとえば、次のステートメントの場合、2 番目のステートメントは失敗します。その理由は、データ・タイプが異なっていてもプロシージャーのパラメーターの数が同じだからです。

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

IN | OUT | INOUT
パラメーターのモードを指定します。

IN パラメーターをプロシージャーの入力パラメーターとして指定します。 プロシージャー内でパラメーターに加えられるすべての変更は、制御が戻されると SQL アプリケーションの呼び出しは行なえなくなります。デフォルトは IN です。

OUT パラメーターをプロシージャーの出力パラメーターとして指定します。

INOUT
パラメーターを、プロシージャーの入力および出力パラメーターの両方として指定します。

parameter-name
パラメーターの名前を指定します。パラメーター名は、プロシージャーでユニークでなければなりません (SQLSTATE 42734)。

data-type
パラメーターのデータ・タイプを指定します。

- CREATE TABLE ステートメントのデータ・タイプ の定義に指定することが可能で、プロシージャーの作成に使用されている言語に対応するものがある SQL データ・タイプ指定と省略形を指定できます。
- LONG VARCHAR、LONG VARGRAPHIC、DATALINK、REFERENCE、およびユーザー定義構造タイプはサポートされません (SQLSTATE 429BB)。

SPECIFIC *specific-name*
定義するプロシージャーのインスタンスに対するユニーク名を指定します。この特定名は、このプロシージャーをドロップする場合、またはこのプロシージャーにコメントを付ける場合に使用することができます。これは、プロシージャーの呼び出しには使用できません。 *specific-name* (特定名) の非修飾形式は SQL ID です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL ID が続きます。暗黙または明示の修飾子も含めて、その名前が、アプリケーション・サーバーに存在する他のプロシージャー・インスタンスを指定するものであってはなりません。そうでない場合、エラー (SQLSTATE 42710) になります。

specific-name は、既存の *procedure-name* と同じにすることができます。

修飾子を指定しない場合、*procedure-name* に使用された修飾子が使用されます。修飾子を指定する場合は、*procedure-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQL*yymmddhhmmsshhn*)。

DYNAMIC RESULT SETS *integer*
ストアード・プロシージャーから戻される結果セットの上限の見積もりを指定します。

CONTAINS SQL、READS SQL DATA、MODIFIES SQL DATA
プロシージャーに含まれる SQL ステートメントのデータ・アクセスのレベルを示します。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、ストアード・プロシージャーで実行できることを指定します (SQLSTATE 38004 または 42985)。どのストアード・プロシージャーでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、ストアード・プロシージャーで実行できることを指定します (SQLSTATE 38002 または 42985)。どのストアード・プロシージャーでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

MODIFIES SQL DATA
このストアード・プロシージャーは、ストアード・プロシージャーでサポートされていないステートメント以外のすべての SQL ステートメントを実行できることを指定します (SQLSTATE 38003 または 42985)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節は、同一の引き数値に対してプロシージャーが常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に依存してプロシージャーの結果が影響を受けるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC が指定されたプロシージャーは、同じ入力を指定して正しく呼び出した場合に常に同じ結果を戻します。

現在、この文節はストアード・プロシージャーの処理に影響を与えません。

CALLED ON NULL INPUT
CALLED ON NULL INPUT は、ストアード・プロシージャーに常に適用されます。これは、任意の引き数が NULL かどうかにかかわりなく、ストアード・プロシージャーが呼び出されることを意味します。 OUT または INOUT パラメーターは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。 NULL の引き数値の有無のテストはストアード・プロシージャーで行う必要があります。

INHERIT SPECIAL REGISTERS
このオプション文節は、プロシージャーの更新可能な特殊レジスターが、呼び出しステートメントの環境からの初期値を継承するよう指定します。ネストされたオブジェクト (たとえば、トリガーまたはビュー) に呼び出されるルーチンの場合、初期値は (オブジェクト定義から継承するのではなく) ランタイム環境から継承します。

特殊レジスターに対する変更が、プロシージャーの呼び出し元に戻されることはありません。

更新不能の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、デフォルト値に設定されます。

OLD SAVEPOINT LEVEL または **NEW SAVEPOINT LEVEL**
このストアード・プロシージャーが、セーブポイント名と影響について新しいセーブポイント・レベルを設定するかどうかを指定します。 OLD SAVEPOINT LEVEL がデフォルトの動作です。セーブポイント・レベルについて詳しくは、SAVEPOINT ステートメントの説明にある『規則』のセクションを参照してください。

LANGUAGE SQL
この文節は、プロシージャー本体が SQL 言語に書き込まれるように指定するのに使用します。

EXTERNAL ACTION または **NO EXTERNAL ACTION**
プロシージャーが、データベース・マネージャーによって管理されていないオブジェクトの状態を変更するアクションを取るか (EXTERNAL ACTION)、または取らないか (NO EXTERNAL ACTION) を指定します。デフォルトは EXTERNAL ACTION です。 NO EXTERNAL ACTION を指定した場合、プロシージャーが外部に影響を与えないことを前提とした最適化を、システムは使用できます。

PARAMETER CCSID
プロシージャーとやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。

UNICODE
文字データは UTF-8 で記述され、GRAPHIC データは UCS-2 で記述されることを指定します。データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 56031)。

SQL-procedure-body
SQL プロシージャーの本体である SQL ステートメントを指定します。プロシージャー・コンパウンド・ステートメント内に複数の SQL-procedure-statement を指定することができます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

規則:

- 動的コンパウンド・ステートメント内から呼び出されるプロシージャーは、プロシージャー作成時に OLD SAVEPOINT LEVEL が指定またはデフォルト設定されていたとしても、 NEW SAVEPOINT LEVEL を指定して作成されたかのように実行されます。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - ASUTIME NO LIMIT
      - COMMIT ON RETURN NO
      - NO COLLID
      - STAY RESIDENT NO
  - 以前のバージョンの DB2 との互換性:
    - DYNAMIC RESULT SETS の代わりに RESULT SETS を指定できます。
    - CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
  - まだ存在していないスキーマ名を用いてプロシージャーを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
  - ***特権***

    プロシージャーの定義者は、プロシージャーに対する WITH GRANT OPTION 付きの EXECUTE 特権と、プロシージャーをドロップする権利を常に与えられます。

例:

例 1: 社員の給与の中央値を戻す SQL プロシージャーを作成します。給与の中央値を超える給与を得ている全社員の氏名、肩書き、および給与の入った結果セットを戻します。

```
CREATE PROCEDURE MEDIAN_RESULT_SET (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
BEGIN
  DECLARE v_numRecords INT DEFAULT 1;
  DECLARE v_counter INT DEFAULT 0;

  DECLARE c1 CURSOR FOR
    SELECT CAST(salary AS DOUBLE)
      FROM staff
      ORDER BY salary;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, CAST(salary AS INTEGER)
      FROM staff
      WHERE salary > medianSalary
      ORDER BY salary;

  DECLARE EXIT HANDLER FOR NOT FOUND
    SET medianSalary = 6666;

  SET medianSalary = 0;
  SELECT COUNT(*) INTO v_numRecords
    FROM STAFF;
  OPEN c1;
  WHILE v_counter < (v_numRecords/2 + 1)
  DO
    FETCH c1 INTO medianSalary;
    SET v_counter = v_counter + 1;
```

```
    END WHILE;
    CLOSE c1;
    OPEN c2;
  END
```

関連資料:

- 704 ページの『SAVEPOINT』
- 140 ページの『コンパウンド SQL (プロシージャー)』
- *SQL リファレンス 第 1 巻* の『ルーチンで使用可能な SQL ステートメント』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』

関連サンプル:

- 『basecase.db2 -- To create the UPDATE_SALARY SQL procedure 』
- 『nestcase.db2 -- To create the BUMP_SALARY SQL procedure 』
- 『nestedsp.db2 -- To create the OUT_AVERAGE, OUT_MEDIAN and MAX_SALARY SQL procedures』
- 『rsultset.db2 -- To register and create the MEDIAN_RESULT_SET SQL procedure』

CREATE SCHEMA

CREATE SCHEMA ステートメントは、スキーマを定義します。また、オブジェクトを作成して、このステートメントでそのオブジェクトに関する特権を与えることも可能です。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

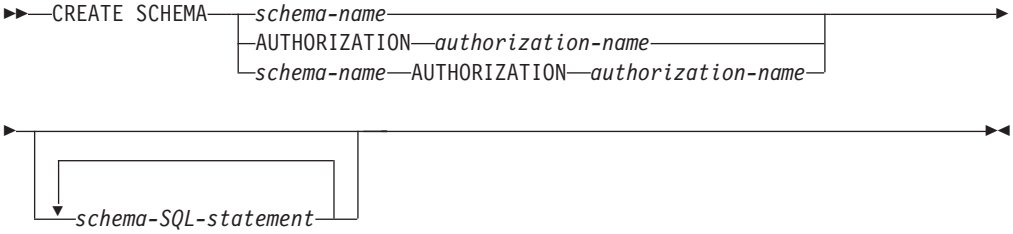
SYSADM 権限または DBADM 権限のある許可 ID は、任意の有効な *schema-name* または *authorization-name* を指定してスキーマを作成できます。

SYSADM 権限または DBADM 権限がない許可 ID は、ステートメントの許可 ID に一致する *schema-name* または *authorization-name* を指定しなければスキーマを作成できません。

ステートメントに *schema-SQL-statement* が含まれている場合、 *authorization-name* (指定されていない場合、ステートメントの許可 ID がデフォルト解釈される) が持つ特権には、以下の特権の少なくとも 1 つが含まれている必要があります。

- それぞれの *schema-SQL-statement* を実行するために必要な特権
- SYSADM または DBADM 権限

構文:



説明:

schema-name
スキーマの名前を指定します。この名前は、カタログですでに記述されているスキーマを指定するものであってはなりません (SQLSTATE 42710)。 'SYS' で始まる名前は使用できません (SQLSTATE 42939)。スキーマの所有者は、ステートメントを発行した許可 ID です。

AUTHORIZATION *authorization-name*
スキーマの所有者であるユーザーを指定します。 *authorization-name* の値は、スキーマの名前の指定にも使用されます。 *authorization-name* は、カタログですでに記述されているスキーマを指定するものであってはなりません (SQLSTATE 42710)。

schema-name **AUTHORIZATION** *authorization-name*
schema-name のスキーマを識別します。その所有者が *authorization-name* です。

schema-name は、カタログですでに記述されているスキーマを指定するものであってはなりません (SQLSTATE 42710)。 *schema-name* には 'SYS' で始まる名前は使用できません (SQLSTATE 42939)。

schema-SQL-statement
CREATE SCHEMA ステートメントに組み込むことができる SQL ステートメントは、次のとおりです。

- CREATE TABLE ステートメント (型付き表およびマテリアライズ照会表は除く)
- CREATE VIEW ステートメント (型付きビューは除く)
- CREATE INDEX ステートメント
- COMMENT ステートメント
- GRANT ステートメント

注:

- スキーマの所有者は、以下のように決定されます。
  - AUTHORIZATION 文節が指定されている場合は、指定された *authorization-name* がスキーマの所有者になります。
  - AUTHORIZATION 文節の指定がない場合は、 CREATE SCHEMA ステートメントを発行した許可 ID がスキーマの所有者になります。
- スキーマの所有者は、ユーザーであることが想定されます (グループではなく)。
- CREATE SCHEMA ステートメントを使用してスキーマを明示的に作成すると、スキーマの所有者はそのスキーマに関して CREATEIN 特権、DROPIN 特権、および ALTERIN 特権を与えられ、これらの特権を他のユーザーに与えることができます。
- CREATE SCHEMA ステートメントの一部として作成されるオブジェクトの定義者は、スキーマの所有者になります。スキーマの所有者は、 CREATE SCHEMA ステートメントの一環として与えられる特権の付与者でもあります。
- CREATE SCHEMA ステートメント中の SQL ステートメント中の非修飾のオブジェクト名は、作成されたスキーマの名前によって暗黙的に修飾されます。
- CREATE ステートメントに、作成するオブジェクトの修飾名が含まれる場合、その修飾名に指定されたスキーマ名は作成されるスキーマの名前と同じでなければなりません (SQLSTATE 42875)。ステートメントで参照されるその他のオブジェクトは、任意の有効なスキーマ名で修飾することができます。
- スキーマ名として SESSION を使用することは推奨されません。宣言済み一時表は SESSION で修飾されていなければならないので、アプリケーションで、持続表と同一の名前を付けた一時表を宣言することがあり得ます。スキーマ名 SESSION の付いた表を参照する SQL ステートメントは、同一名の持続表ではなく宣言済み一時表に解決されてしまいます (ステートメントのコンパイル時に)。静的組み込みおよび動的な組み込み SQL ステートメントでは、別々の時点で SQL ステートメントのコンパイルが行われるので、結果は、宣言済み一時表がいつ定義されたかによって異なってしまいます。持続表、ビュー、または別名が、SESSION というスキーマ名を使って定義されていなければ、これらの事項にとらわれる必要はありません。

例:

例 1: DBADM 権限のあるユーザーが、 RICK という名前のスキーマをユーザー RICK を所有者として作成します。

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

例 2: 部品の在庫表と部品番号の索引があるスキーマを作成します。ユーザー JONES に、表に対する権限を与えます。

```
CREATE SCHEMA INVENTRY

  CREATE TABLE PART (PARTNO   SMALLINT NOT NULL,
                     DESCR    VARCHAR(24),
                     QUANTITY INTEGER)

  CREATE INDEX PARTIND ON PART (PARTNO)

  GRANT ALL ON PART TO JONES
```

例 3: 2 つの表がある PERS という名前のスキーマを作成します。それぞれの表には他の表を参照する外部キーがあります。これは、 ALTER TABLE ステートメントを使用せずにこのような表のペアを作成する CREATE SCHEMA ステートメントの機能の一例です。

```
CREATE SCHEMA PERS

  CREATE TABLE ORG (DEPTNUMB  SMALLINT NOT NULL,
                    DEPTNAME VARCHAR(14),
                    MANAGER  SMALLINT,
                    DIVISION VARCHAR(10),
                    LOCATION VARCHAR(13),
                    CONSTRAINT PKEYDNO
                      PRIMARY KEY (DEPTNUMB),
                    CONSTRAINT FKEYMGR
                      FOREIGN KEY (MANAGER)
                      REFERENCES STAFF (ID) )

  CREATE TABLE STAFF (ID        SMALLINT NOT NULL,
                      NAME     VARCHAR(9),
                      DEPT     SMALLINT,
                      JOB      VARCHAR(5),
                      YEARS    SMALLINT,
                      SALARY   DECIMAL(7,2),
                      COMM     DECIMAL(7,2),
                      CONSTRAINT PKEYID
                        PRIMARY KEY (ID),
                      CONSTRAINT FKEYDNO
                        FOREIGN KEY (DEPT)
                        REFERENCES ORG (DEPTNUMB) )
```

関連資料:

- 118 ページの『COMMENT』
- 277 ページの『CREATE INDEX』
- 347 ページの『CREATE TABLE』
- 474 ページの『CREATE VIEW』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』

CREATE SEQUENCE

CREATE SEQUENCE ステートメントは、アプリケーション・サーバーでのシーケンスを作成します。

呼び出し:

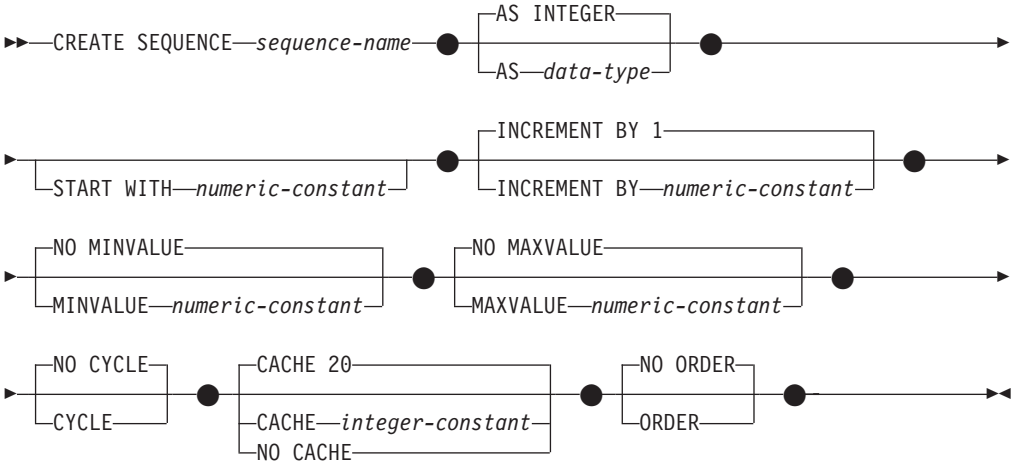
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 特権 (シーケンスの暗黙または明示的なスキーマ名が存在しない場合)
- CREATEIN 特権 (シーケンスのスキーマ名が既存のスキーマを参照している場合)

構文:



説明:

sequence-name
シーケンスを指定します。名前の組み合わせ、また暗黙および明示スキーマ名は、現行のサーバーに存在するシーケンスを識別することはできません (SQLSTATE 42710)。

sequence-name の非修飾フォームは SQL ID です。修飾フォームは、ピリオドと SQL ID が後ろに続く修飾子です。修飾子はスキーマ名です。

シーケンス名がスキーマ名で明示的に修飾されている場合、そのスキーマ名の先頭を 'SYS' にすると、エラーが起こります (SQLSTATE 42939)。

AS *data-type*
シーケンス値に使用されるデータ・タイプを指定します。データ・タイプは、ゼロの位取りの整数値タイプ (SMALLINT、INTEGER、BIGINT、または

DECIMAL) か、ソース・タイプがゼロの位取りの整数値タイプであるユーザー定義の特殊タイプまたは参照タイプにすることができます (SQLSTATE 42815)。デフォルトは INTEGER です。

START WITH *numeric-constant*
シーケンスの最初の値を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815)。デフォルトは、昇順シーケンスであれば MINVALUE、降順シーケンスであれば MAXVALUE です。

この値は、シーケンスの最大または最小値に達した後、そのシーケンスが循環する値である必要はありません。 START WITH 文節を使用して、循環に使用される範囲外のシーケンスを開始することができます。循環に使用される範囲は、MINVALUE および MAXVALUE によって定義されています。

INCREMENT BY *numeric-constant*
連続したシーケンス値のインターバルを指定します。この値として、長精度整数定数の値を超えない範囲で (SQLSTATE 42820)、シーケンスに関連したデータ・タイプの列に割り当てることのできる任意の正または負の値を指定できます (SQLSTATE 42815)。ただし、小数点の右側に非ゼロの数字が存在してはなりません (SQLSTATE 428FA)。

この値が負の場合、これは降順シーケンスです。 この値が 0 の場合、または正の場合は、昇順になります。 デフォルトは 1 です。

MINVALUE または NO MINVALUE
降順シーケンスが値の生成を循環または停止する最小値、あるいは最大値に達した後、昇順シーケンスが循環する最小値を指定します。

MINVALUE *numeric-constant*
最小値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815) が、最大値以下でなければなりません (SQLSTATE 42815)。

NO MINVALUE
昇順シーケンスの場合、値は START WITH 値で、START WITH が指定されない場合には 1 です。降順シーケンスの場合、シーケンスに関連するデータ・タイプの最小値です。これがデフォルトです。

MAXVALUE または NO MAXVALUE
昇順シーケンスが値の生成を循環または停止する最大値、あるいは最小値に達した後、降順シーケンスが循環する最大値を指定します。

MAXVALUE *numeric-constant*
最大値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、シーケンスに関連するデータ・タイプの列に割り当てられる正または負の値にすることができます (SQLSTATE 42815) が、最小値以上でなければなりません (SQLSTATE 42815)。

NO MAXVALUE
昇順シーケンスの場合、値はシーケンスに関連するデータ・タイプの最大値です。降順シーケンスの場合、値は START WITH 値で、START WITH が指定されない場合には -1 です。

CYCLE または NO CYCLE
その最大値または最小値に達した後、シーケンスが値の生成を続行するかどうかを指定します。次の値が境界条件を正確に満たしたとき、またはその値を超えることによって、シーケンスの境界に達します。

CYCLE
最大値または最小値に達した後、このシーケンスについて値の生成を続行することを指定します。このオプションが使用されると、昇順シーケンスが最大値に達した後、その最小値が生成されます。降順シーケンスが最小値に達した後、その最大値が生成されます。シーケンスの最大値および最小値は、循環に使用される範囲を決定します。

CYCLE が有効な場合、重複するシーケンス値が生成される場合があります。

NO CYCLE
シーケンスの最大値または最小値に達した後、そのシーケンスについて値は生成されないことを指定します。これがデフォルトです。

CACHE または NO CACHE
高速アクセスのため、事前割り振り値のいくつかをメモリーに保管するかどうかを指定します。これはパフォーマンスおよびチューニング・オプションです。

CACHE *integer-constant*
事前割り振りされ、メモリーに保管されるシーケンス値の最大数を指定します。値を事前割り振りしてキャッシュに保管しておくと、シーケンス値を生成するとき、ログへの非同期入出力が少なくなります。

システム障害が起こると、コミットされたステートメントで使用されていないキャッシュ済みシーケンス値はすべて失われます (使用されなくなります)。 CACHE オプションに指定する値は、システム障害の際に失われても構わないシーケンス値の最大数です。

最小値は 2 です (SQLSTATE 42815)。デフォルト値は CACHE 20 です。

NO CACHE
シーケンスの値が事前割り振りされないよう指定します。システム障害、シャットダウン、またはデータベース非活動化の際、値が失われることはありません。このオプションが指定されると、シーケンスの値はキャッシュに保管されません。この場合、シーケンスの新しい値が要求されるたびに、ログに対して非同期入出力が行われます。

NO ORDER または ORDER
要求の順序でシーケンス番号が生成されるかどうかを指定します。

ORDER
要求の順序でシーケンス番号が生成されるよう指定します。

NO ORDER
要求の順序でシーケンス番号を生成する必要がないことを指定します。これがデフォルトです。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - コンマは、複数のシーケンス・オプションを分離するのに使用できます。
  - 以下の構文もサポートされています。
    - NOMINVALUE、NOMAXVALUE、NOCYCLE、NOCACHE、および NOORDER。
- 定数シーケンス (常に定数値を返す) を定義することも可能です。これは、INCREMENT 値にゼロを指定して START WITH 値には MAXVALUE を超えない値を指定するか、あるいは START WITH、MINVALUE、および MAXVALUE に同じ値を指定することによって実行できます。定数シーケンスの場合には、シーケンスに関する NEXT VALUE が呼び出されるたびに、同じ値が戻ります。定数シーケンスは、数値グローバル変数として使用することができます。 ALTER SEQUENCE を使用すると、定数シーケンスのために生成される値を調整することができます。
- ALTER SEQUENCE ステートメントを使用して、シーケンスを手動で循環させることができます。 NO CYCLE が暗黙的または明示的に指定されている場合、ALTER SEQUENCE ステートメントでシーケンスを再始動または拡張し、そのシーケンスの最大または最小値に達した後でも値の生成を続行できます。
- CYCLE キーワードを指定して、シーケンスが循環するように明示的に指定できます。シーケンスを定義する際に CYCLE オプションを使用して、生成された値が境界に達するたびに循環するよう指示します。シーケンスが自動的に循環するように定義されると (つまり CYCLE が明示的に指定された場合)、増分値が 1 または -1 以外の場合には、シーケンスに対して生成される最大または最小値は、実際に指定された MAXVALUE または MINVALUE ではない可能性があります。 たとえば、`START WITH=1, INCREMENT=2, MAXVALUE=10` と定義されたシーケンスは、最大値 9 を生成し、値 10 は生成しないはずです。シーケンスに CYCLE を定義する際、MINVALUE、MAXVALUE、および START WITH の値への影響を考慮してください。
- シーケンス番号のキャッシュは、シーケンス番号の範囲を高速アクセスのためにメモリーに保管することを意味しています。アプリケーションが、次のシーケンス番号をキャッシュから割り振ることができるシーケンスにアクセスしていると、シーケンス番号の割り振りは素早く行われます。ただし、次のシーケンス番号をキャッシュから割り振ることができないシーケンスにアクセスしている場合、シーケンス番号の割り振りは、永続記憶域への入出力操作を待機しなければならない場合があります。 CACHE の値を選択するとき、パフォーマンスとアプリケーション要件の関係を考慮しておく必要があります。
- シーケンスの定義者には、WITH GRANT OPTION 付きの ALTER および USAGE 特権が付与されます。また定義者はシーケンスをドロップできます。

例:

例 1: 1 で始まり、1 つずつ増分し、循環しない、同時に 24 の値をキャッシュに入れる ORG_SEQ というシーケンスを作成します。

```
CREATE SEQUENCE ORG_SEQ
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 24
```

関連サンプル:

- 『DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)』

CREATE SERVER

CREATE SERVER ステートメントは、データ・ソースをフェデレーテッド・データベースへ定義します。このステートメントでは、SERVER という語と、*server-* で始まるパラメーター名は、フェデレーテッド・システムでのデータ・ソースを指しています。そのようなシステムでのフェデレーテッド・サーバー、あるいは DRDA アプリケーション・サーバーを指すわけではありません。

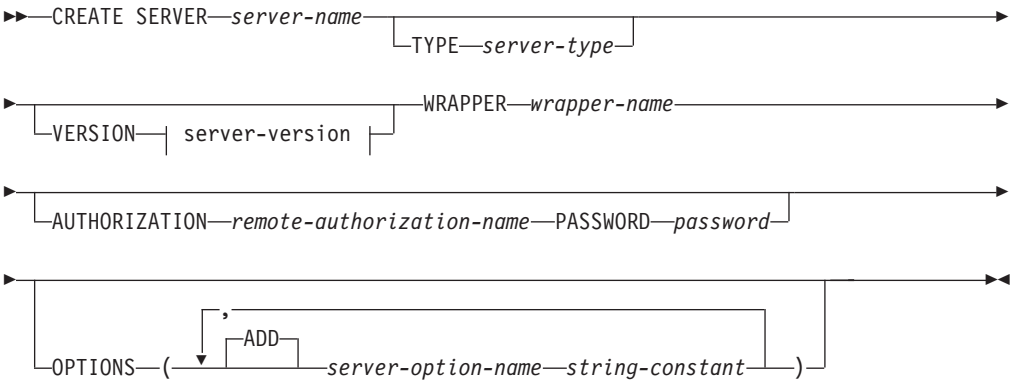
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

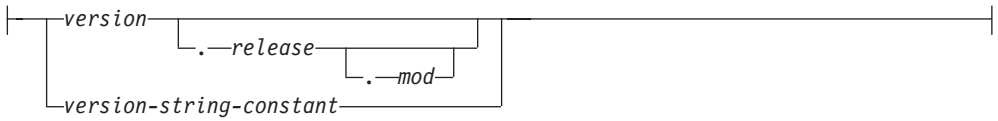
許可:

このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文:



server-version:



説明:

server-name

フェデレーテッド・データベースに対して定義するデータ・ソースを指定します。この名前は、カタログに記述されているデータ・ソースを指すものであってはなりません。 *server-name* は、フェデレーテッド・データベース内の表スペースの名前と同じであってはなりません。

リレーショナル・データ・ソースのサーバー定義は、通常リモート・データベースを表します。 Oracle など一部のリレーショナル・データベース管理システムでは、各インスタンス内に複数のデータベースを取ることが許可されていません。その代わりに、各インスタンスはフェデレーテッド・システム内のサーバーを表します。

非リレーショナル・データ・ソースの場合、サーバー定義の目的はデータ・ソースごとに異なります。検索タイプやデーモン、Web サイト、または Web サーバーにマップされるサーバー定義もあります。その他の非リレーショナル・データ・ソースの場合、フェデレーテッド・オブジェクトの階層ではデータ・ソース・ファイル (ニックネームで識別される) が特定のサーバー・オブジェクトに関連付けられていることが必要なため、サーバー定義が作成されます。

TYPE *server-type*

server-name に示されるデータ・ソースのタイプを指定します。一部のラッパーには、このパラメーターが必須です。

VERSION

server-name に示されるデータ・ソースのバージョンを指定します。一部のラッパーには、このパラメーターが必須です。

version

バージョン番号を指定します。値は整数でなければなりません。

release

version で示されたバージョンのリリース番号を指定します。値は整数でなければなりません。

mod

release で示されたリリースのモディフィケーション番号を指定します。値は整数でなければなりません。

version-string-constant

バージョンの正式名称を指定します。 *version-string-constant* は単一値 (たとえば、'8i') にすることができます。あるいは、*version*、*release*、そして該当する場合は *mod* を連結した値にすることができます (たとえば、'8.0.3')。

WRAPPER *wrapper-name*

server-name で指定されたサーバー・オブジェクトと対話するために、フェデレーテッド・サーバーが使用するラッパーの名前を指定します。

AUTHORIZATION *remote-authorization-name*

DB2 ファミリーのデータ・ソースにのみ必要です。 CREATE SERVER ステートメントの処理時に、必要なアクションをデータ・ソースで実行するときの許可 ID を指定します。この ID には、実行するアクションに必要な権限 (BINDADD または同等の権限) がなければなりません。
remote-authorization-name が大文字小文字混合または小文字で指定される場合 (かつリモート・データ・ソースに大文字小文字の区別のある許可名がある場合) には、 *remote-authorization-name* を二重引用符で囲んでください。

PASSWORD *password*

DB2 ファミリーのデータ・ソースにのみ必要です。 *remote-authorization-name* で表された許可 ID に関連付けられているパスワードを指定します。 *password* が大文字小文字混合または小文字で指定される場合 (かつリモート・データ・ソースに大文字小文字の区別のあるパスワードがある場合) には、 *password* を二重引用符で囲んでください。

OPTIONS

サーバー定義を作成したときに使用可能にされるオプションを指示します。サーバー・オプションはサーバー定義を構成するときに使用されます。任意のデー

タ・ソースのサーバー定義を作成するために使用できるサーバー・オプションもあります。特定のデータ・ソースに固有のサーバー・オプションもあります。

ADD
1 つ以上のサーバー・オプションを使用可能にします。

server-option-name
server-name で示されたデータ・ソースを構成するとき、あるいはそれについての情報を提供するときのいずれかに使うサーバー・オプションを指定します。

string-constant
server-option-name の設定を、文字ストリング定数として指定します。

注:

- データ・ソースがパスワードを必要とする場合は、 *password* を指定します。 *password* のいずれかの文字が小文字であれば、 *password* を引用符で囲みます。
- CREATE SERVER ステートメントを使って DB2 ファミリー・インスタンスをデータ・ソースとして定義する場合、 DB2 で特定のパッケージをそのインスタンスにバインドする必要があるかもしれません。バインドする必要がある場合、ステートメント内の *remote-authorization-name* に BIND 権限がなければなりません。バインド操作の完了に要する時間は、データ・ソースの速度とネットワーク接続の速度によって異なります。

例:

例 1: DB2 for z/OS および OS/390 バージョン 7.1 データ・ソースにアクセスするためのサーバー定義を登録します。 CRANDALL は、DB2 for z/OS および OS/390 サーバー定義に割り当てられる名前です。 DRDA は、このデータ・ソースにアクセスするために使用するラッパーの名前です。さらに、以下のものを指定します。

- GERALD および drowssap は、このステートメントの処理時に、パッケージを CRANDALL でバインドするときの許可 ID とパスワードです。
- CATALOG DATABASE ステートメントに指定された DB2 for z/OS および OS/390 データベースの別名は、CLIENTS390 です。
- CRANDALL にアクセスするときの許可 ID とパスワードは、 CRANDALL へ大文字で送信されます。
- CLIENTS390 とフェデレーテッド・データベースは、同じ照合順序を使用します。

```
 CREATE SERVER CRANDALL
TYPE DB2/ZOS
VERSION 7.1
WRAPPER DRDA
AUTHORIZATION "GERALD"
PASSWORD drowssap
  OPTIONS
    (DBNAME 'CLIENTS390',
      FOLD_ID 'U',
      FOLD_PW 'U',
    COLLATING_SEQUENCE 'Y')
```

例 2: Oracle 9 データ・ソースにアクセスするためのサーバー定義を登録します。 CUSTOMERS は、Oracle サーバー定義に割り当てられる名前です。 NET8 は、このデータ・ソースにアクセスするために使用するラッパーの名前です。さらに、以下のものを指定します。

- ABC は、Oracle データベース・サーバーが置かれているノードの名前です。
- フェデレーテッド・サーバーの CPU の動作速度が、 CUSTOMERS をサポートする CPU の 2 倍であること。
- フェデレーテッド・サーバーの入出力装置のデータ処理速度が、 CUSTOMERS の入出力装置の 1.5 倍であること。

```
CREATE SERVER CUSTOMERS
 TYPE ORACLE
VERSION 9
WRAPPER NET8
  OPTIONS
    (NODE 'ABC',
      CPU_RATIO '2.0',
    IO_RATIO '1.5')
```

例 3: Excel ラッパーのサーバー定義を登録します。このサーバー定義は、フェデレーテッド・オブジェクトの階層を保持するために必要です。 BIOCHEM_LAB は Excel サーバー定義に割り当てられる名前です。 EXCEL_2000_WRAPPER は、このデータ・ソースにアクセスするために使用するラッパーの名前です。

```
CREATE SERVER BIOCHEM_DATA
WRAPPER EXCEL_2000_WRAPPER
```

例 4: BLAST データ・ソースにアクセスするためのサーバー定義を登録します。 BLAST_SERVER は BLAST サーバー定義に割り当てられる名前です。このサーバー定義がサポートする検索のタイプは、BLASTn 検索タイプです。 VERSION は BLAST 検索プログラムのバージョンです。 BLAST_WRAPPER は、このデータ・ソースにアクセスするために使用するラッパーの名前です。さらに、以下のものを指定します。

- NODE は、BLAST デーモン・プロセスを実行するサーバーのホスト名です。
- BLAST デーモンが、BLAST ラッパーによってサブミットされたジョブを listen するときに使用するポート番号は、4007 です。

```
CREATE SERVER BLAST_SERVER
TYPE BLASTn
VERSION 2.1.2
WRAPPER BLAST_WRAPPER
  OPTIONS
    (NODE 'big.rs.company.com',
   DAEMON_PORT '4007')
```

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』
- *フェデレーテッド・システム・ガイド* の『サーバー定義およびサーバー・オプション』

関連資料:

- *フェデレーテッド・システム・ガイド* の『フェデレーテッド・システムのサーバー・オプション』

- フェデレーテッド・システム・ガイド の『SQL ステートメントで有効なサーバーのタイプ』

CREATE TABLE

CREATE TABLE ステートメントは表を定義します。定義には、その表の名前と、その列の名前および属性を含める必要があります。定義には、主キーやチェック制約など、表の他の属性を含めることができます。

グローバル一時表を宣言するには、DECLARE GLOBAL TEMPORARY TABLE ステートメントを使用します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- データベースに対する CREATETAB 権限および表スペースに対する USE 特権に加えて、以下のいずれか。
  - データベースに対する IMPLICIT_SCHEMA 権限 (表の暗黙的または明示的スキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (表のスキーマ名が既存のスキーマを指す場合)

  副表を定義する場合には、許可 ID は表階層のルート表の定義者と同じでなければなりません。

外部キーを定義する場合には、ステートメントの許可 ID が保持する特権として、親表に対する以下のいずれかが必要になります。

- その表に対する REFERENCES 特権
- 指定の親キーのそれぞれの列に対する REFERENCES 特権
- 表に対する CONTROL 特権
- SYSADM または DBADM 権限

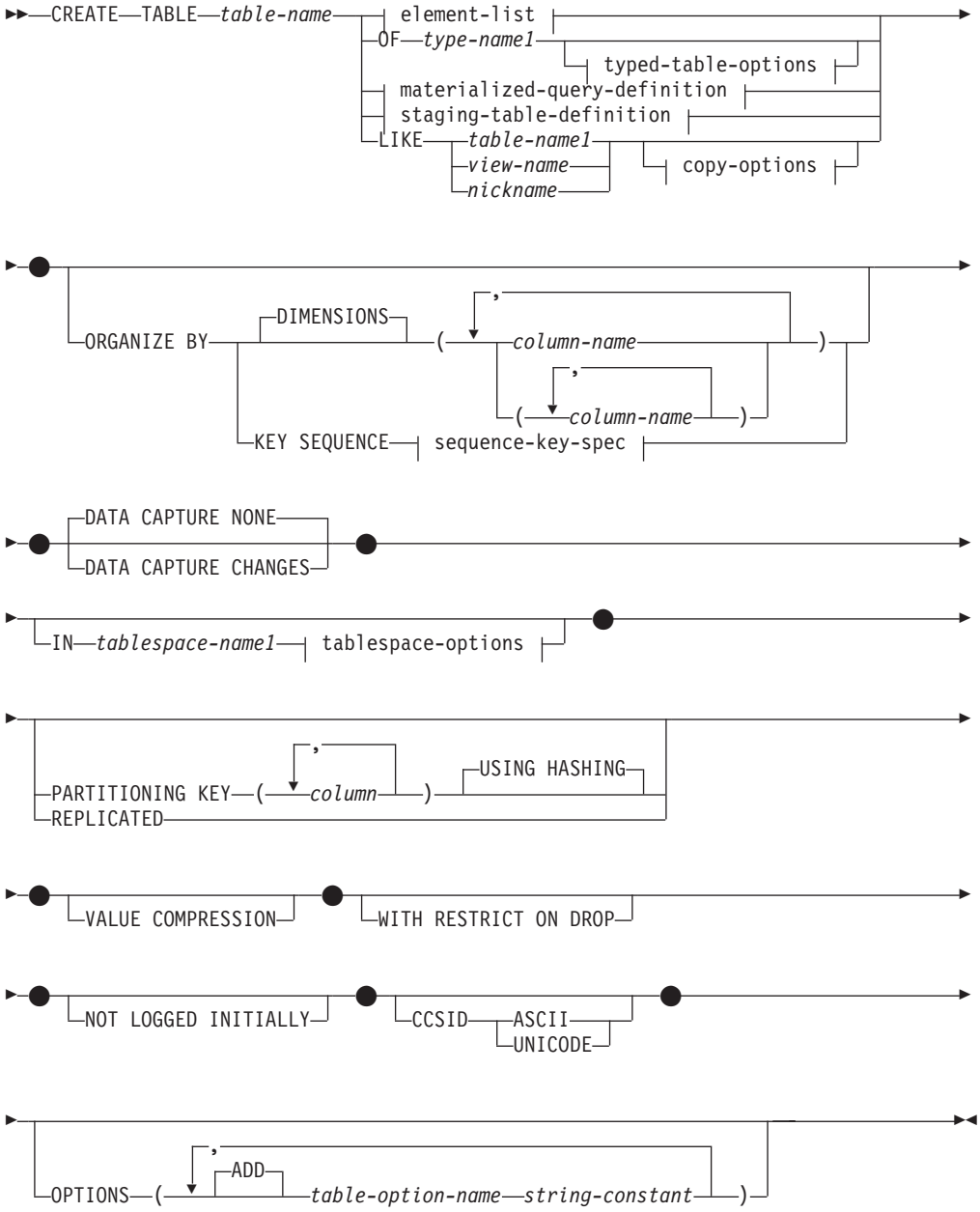
(全選択を使用して) マテリアライズ照会表を定義するには、このステートメントの許可 ID によって保持される特権には、全選択で識別された個々の表またはビューに対する以下のうち少なくとも 1 つ含まれている必要があります。

- 表またはビューに対する SELECT 特権と、 REFRESH DEFERRED または REFRESH IMMEDIATE が指定されている場合には ALTER 特権
- 表またはビューに対する CONTROL 特権
- SYSADM または DBADM 権限

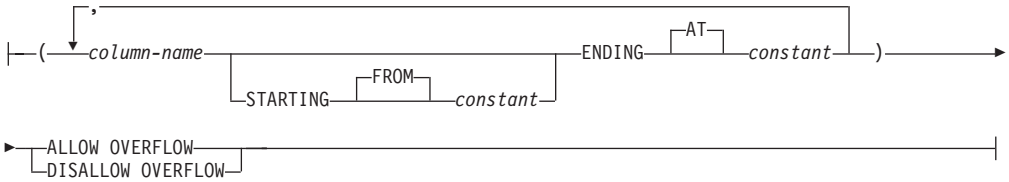
マテリアライズ照会表に関連付けられたステージング表を定義するには、ステートメントの許可 ID に、以下の特権の少なくとも 1 つが含まれている必要があります。

- マテリアライズ照会表での CONTROL 特権または ALTER 特権、なおかつ、マテリアライズ照会表での全選択で識別された個々の表またはビューに対する以下の特権が少なくても 1 つ含まれている必要があります。
  - その表またはビューに対する SELECT 特権および ALTER 特権
  - 表またはビューに対する CONTROL 特権
- SYSADM または DBADM 権限

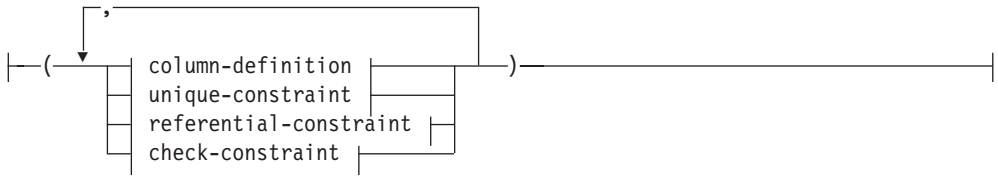
構文:



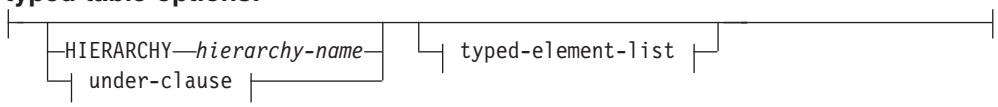
sequence-key-spec:



element-list:

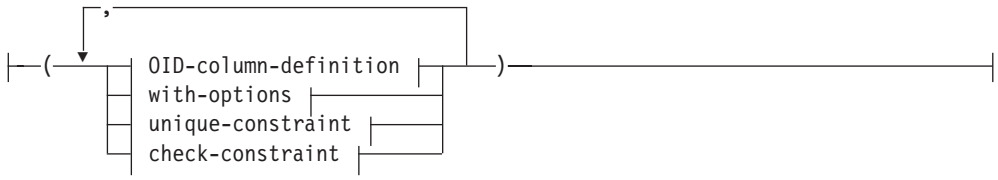


typed-table-options:

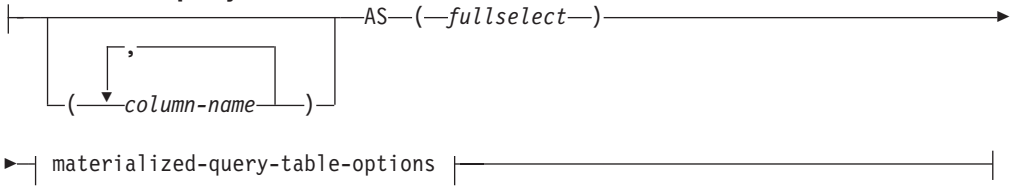


under-clause:

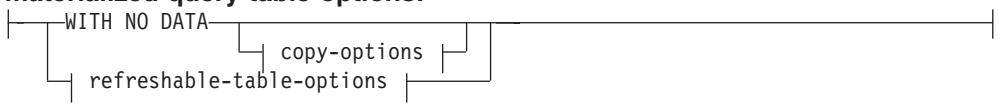
typed-element-list:



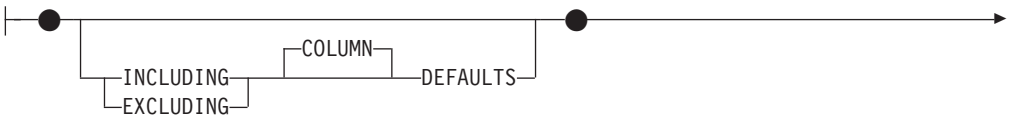
materialized-query-definition:

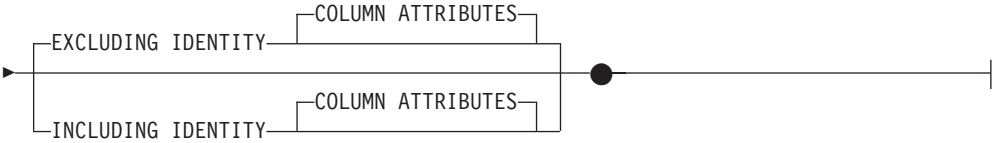


materialized-query-table-options:

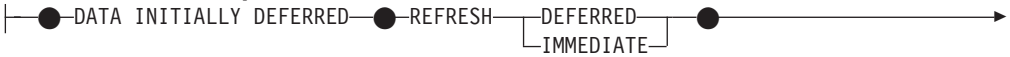


copy-options:

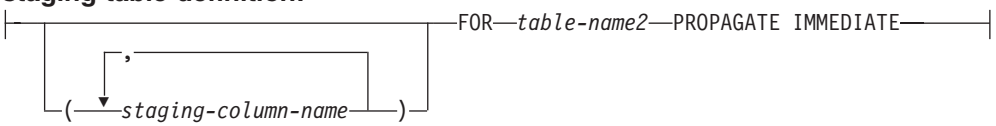




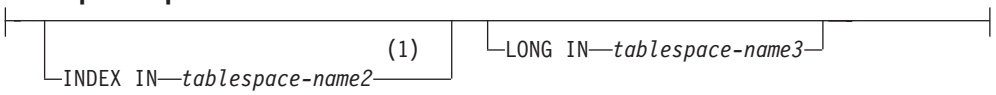
refreshable-table-options:



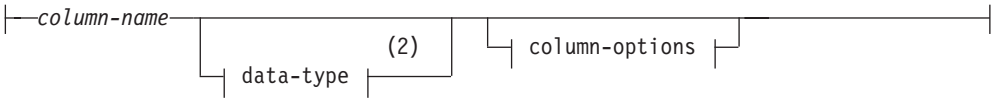
staging-table-definition:



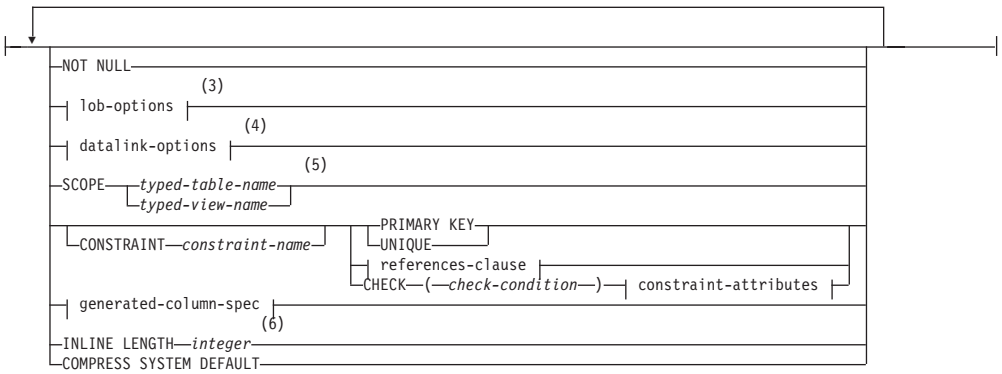
tablespace-options:



column-definition:



column-options:



注:

1 表の索引をどの表スペースに含めるかを指定できるのは、表を作成する場合だけです。

2 選択する最初の column-option が、 generation-expression を指定した generated-column-spec の場合、data-type を省略することができます。これは、generation-expression の処理結果のデータ・タイプから判別されます。

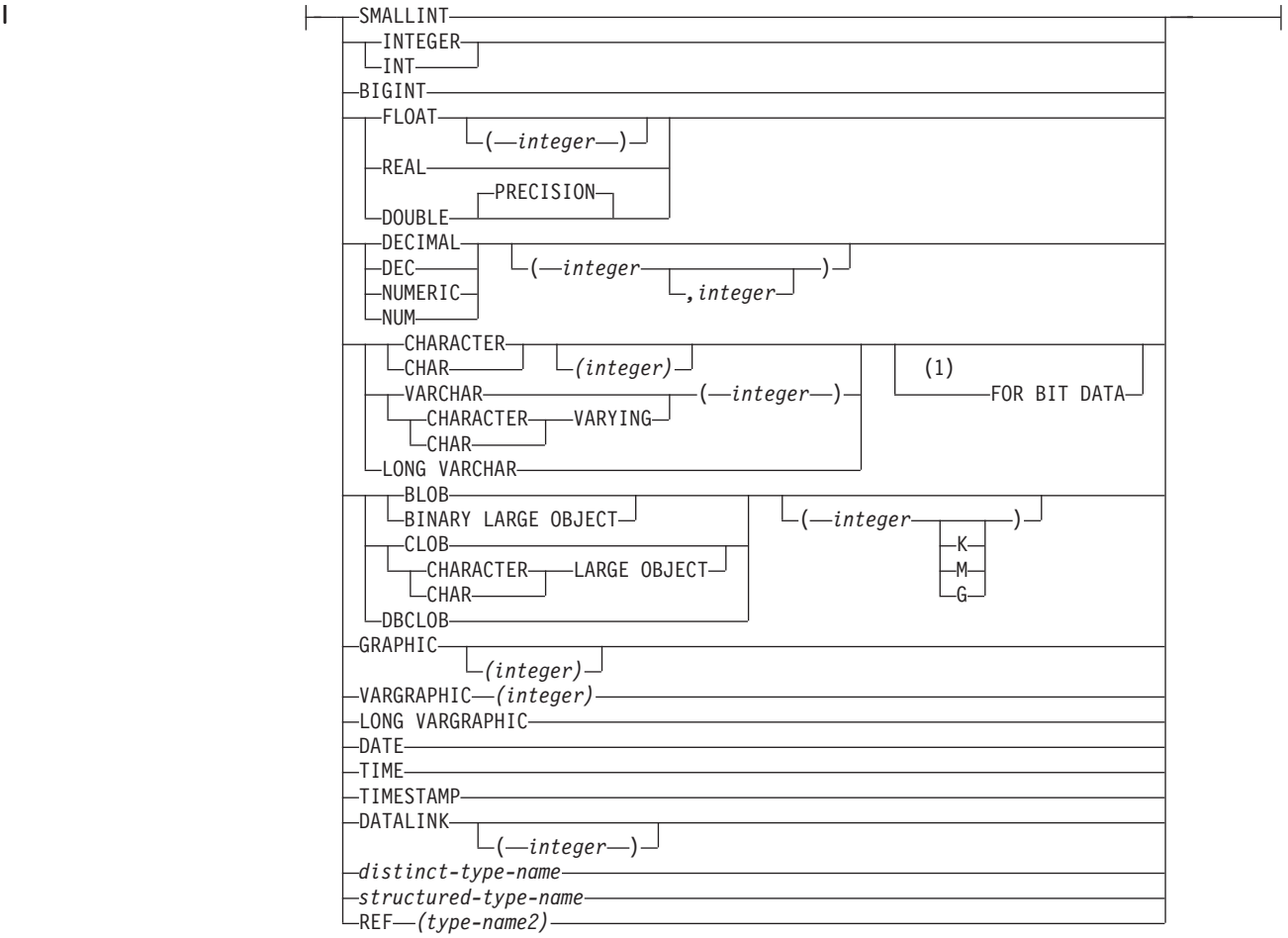
3 lob-options (LOB オプション) 文節は、ラージ・オブジェクト・タイプ（BLOB、CLOB、および DBCLOB）と、ラージ・オブジェクト・タイプに基づく特殊タイプに対してのみ適用されます。

4 datalink-options 文節は、 DATALINK タイプと、DATALINK タイプに基づく特殊タイプに対してのみ適用されます。

5 SCOPE 文節は REF タイプに対してのみ適用されます。

6 INLINE LENGTH は、構造タイプとして定義された列に対してのみ用います。

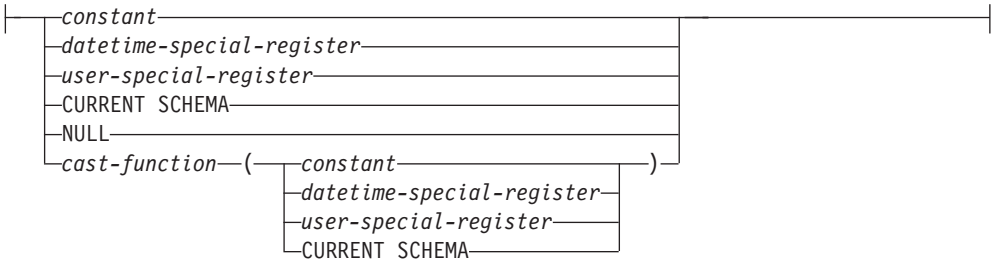
data-type:



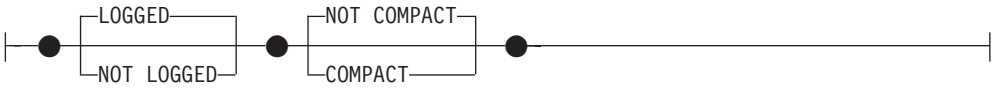
注:

1 FOR BIT DATA 文節とその後に続く他の列制約とは、任意の順序で指定できます。

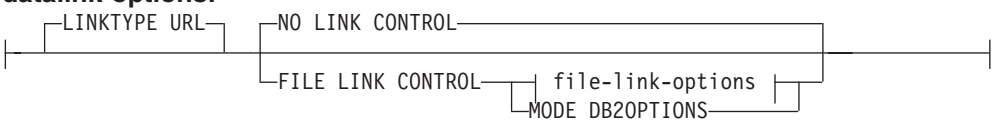
default-values:



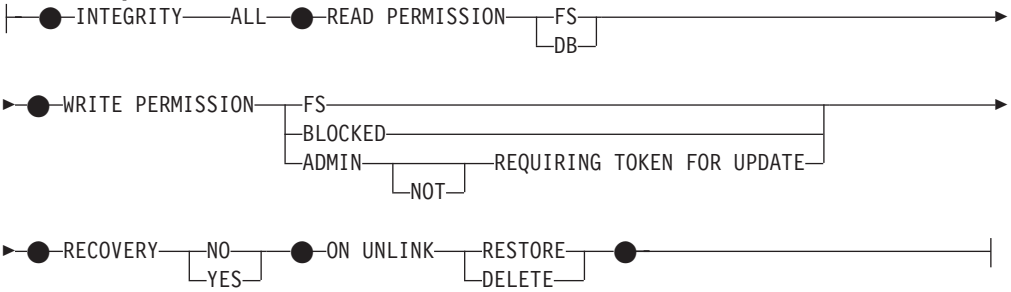
lob-options:



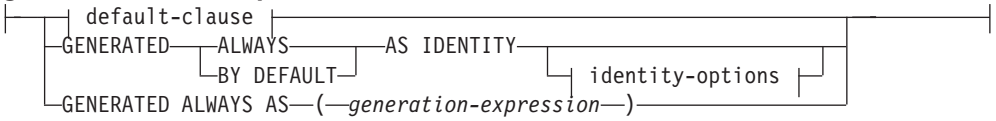
datalink-options:



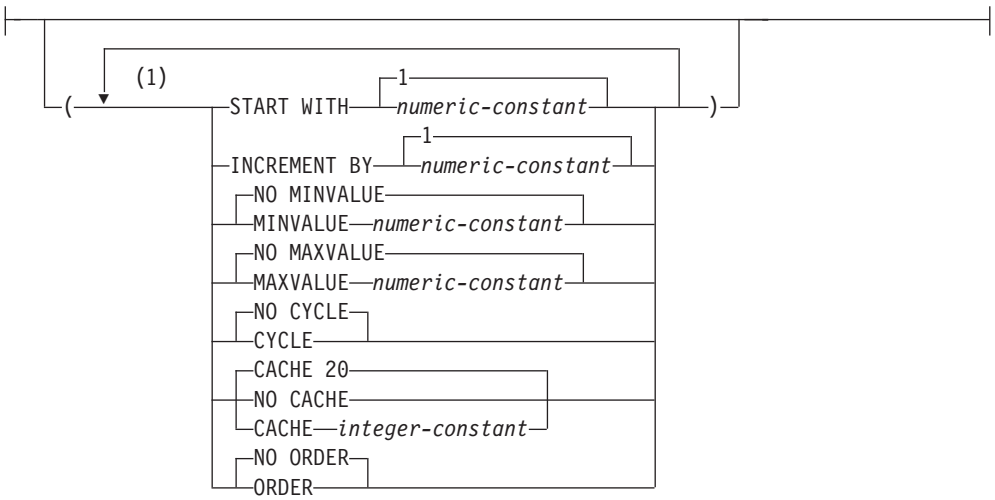
file-link-options:



generated-column-spec:



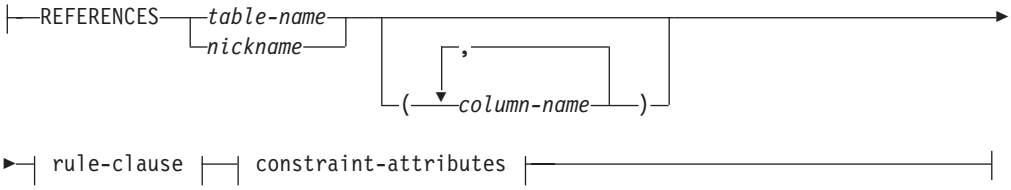
identity-options:



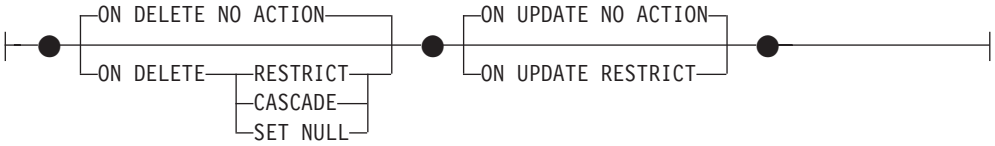
注:

1 同じ文節を複数回指定することはできません。

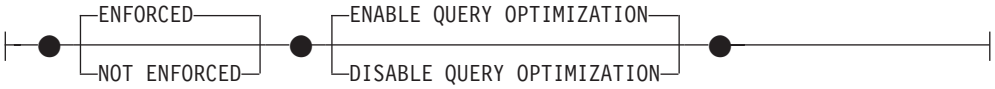
references-clause:



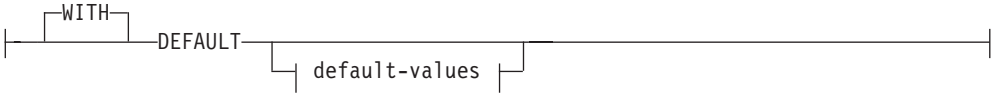
rule-clause:



constraint-attributes:



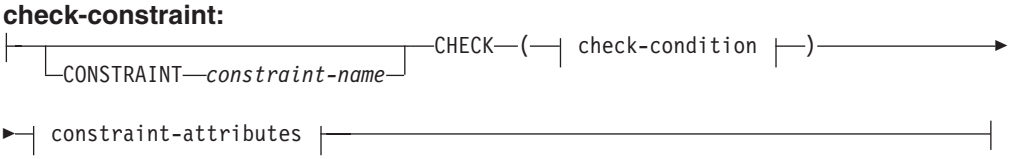
default-clause:



unique-constraint:

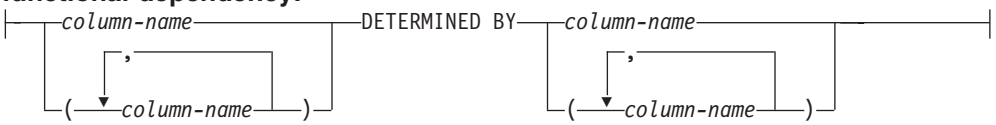
referential-constraint:

check-constraint:



check-condition:

functional-dependency:



OID-column-definition:

with-options:

説明:

システム保守済みマテリアライズ照会表とユーザー保守済みマテリアライズ照会表は、それぞれを個別に識別する必要が生じない限り、どちらもマテリアライズ照会表 と呼びます。

table-name
表の名前を指定します。 暗黙または明示の修飾子を含む名前は、カタログに記述されている表、ビュー、ニックネーム、または別名を指定するものであってはなりません。 スキーマ名は SYSIBM、SYSCAT、SYSFUN、または SYSSTAT であってはなりません (SQLSTATE 42939)。

OF *type-name1*
表の列が *type-name1* で指定される構造タイプの属性に基づいていることを指定します。 *type-name1* の指定にスキーマ名が含まれていない場合、そのタイプ名は SQL パス上のスキーマを探索することによって決まります (このパスは、静的 SQL の場合は FUNCPATH プリプロセス・オプションによって、動的 SQL の場合は CURRENT PATH レジスターによって定義されます)。このタイプ名は、既存のユーザー定義タイプ名である (SQLSTATE 42704) 必要があり、また、少なくとも 1 つの属性があって (SQLSTATE 42997)、しかもインスタンス化可能な構造タイプでなければなりません (SQLSTATE 428DP)。

UNDER が指定されていない場合には、オブジェクト ID 列を指定する必要があります (*OID-column-definition* を参照)。このオブジェクト ID 列は、その表の最初の列になります。オブジェクト ID 列の後に、*type-name1* の属性に基づく列が続きます。

HIERARCHY *hierarchy-name*
表階層に関連する階層表を指定します。同時に、これは階層のルート表としても作成されます。型付き表階層に含まれる副表のデータはすべて、この階層表に保管されます。階層表を SQL ステートメントで直接に参照することはできません。 *hierarchy-name* は *table-name* になります。暗黙または明示のスキーマ名の入った *hierarchy-name* は、カタログに記述されている表、ニックネーム、ビュー、または別名を指定するものであってはなりません。スキーマ名を指定する場合、作成する表のスキーマ名と同じにする必要があります (SQLSTATE 428DQ)。ルート表の定義時にこの文節を省略すると、システムによって名前が生成されます。この名前は、作成する表の名前とその後のユニークな接尾部で構成される ID であり、既存の表、ビュー、別名、およびニックネームの ID の中でユニークなものです。

UNDER *supertable-name*
表が *supertable-name* の副表であることを指定します。スーパー表は既存の表でなければならず (SQLSTATE 42704)、かつ表は *type-name1* のすぐ上のスーパータイプである構造タイプを使用して定義しなければなりません (SQLSTATE 428DB)。 *table-name* と *supertable-name* のスキーマ名は、同じでなければなりません (SQLSTATE 428DQ)。 *supertable-name* で指定される表には、*type-name1* で既に定義された既存の副表を含めることはできません (SQLSTATE 42742)。

表の列には、スーパー表のオブジェクト ID 列が含まれています。この列のタイプは、REF(*type-name1*) に変更されており、 *type-name1* の属性に基づく列が続きます (ここでいうタイプには、スーパータイプの属性も含まれていることを念頭に置いてください)。属性名は OID 列名と同じものにすることはできません (SQLSTATE 42711)。

表スペース、データ・キャプチャーなど他の表オプションは、最初のうちログされません。また、パーティション・キーを指定することはできません。これらのオプションはスーパー表から継承されます (SQLSTATE 42613)。

INHERIT SELECT PRIVILEGES
スーパー表に対して SELECT 特権を保持するユーザーやグループはすべて、新しく作成した副表に対しても同様の特権を付与されます。この特権は、副表定義者によって付与されたものと見なされます。

element-list
表のエレメントを定義します。これには、表の列と制約の定義が含まれます。

typed-element-list
型付き表の追加エレメントを定義します。これには、列の追加オプション、オブジェクト ID 列 (ルート表のみ) の追加、表の制約事項などが含まれます。

materialized-query-definition
表の定義が照会の結果に基づいている場合に、その表は照会に基づくマテリアライズ照会表となります。

column-name
表の列の名前を指定します。 列名のリストを指定する場合、リスト中の列の名前の数は、全選択の結果表の列の数と同じ数でなければなりません。各 *column-name* (列名) は、ユニークで、しかも非修飾でなければなりません。列名のリストの指定がない場合、表の列は、全選択の結果表の列名を継承します。

全選択の結果表に、無名列の重複列名がある場合には、列名のリストを指定する必要があります (SQLSTATE 42908)。 無名列とは、定数、関数、式、またはセット演算から派生した列で、選択リストの AS 文節によって名前が指定されていない列を指します。

AS
表の定義に使用され、表に含まれるデータを判別する照会をこの後に指定します。

fullselect
表の基礎となる照会を定義します。作成される列定義は、同じ照会で定義したビューの定義と同じになります。

各選択リスト・エレメントには名前が必要です (式には AS 文節を使用します)。 *materialized-query-definition* は、マテリアライズ照会表の属性を定義します。選択されたオプションは、次のような全選択の内容も定義します。

WITH NO DATA が指定されている場合、型付き表または型付きビューを参照しない有効な全選択を指定することができます。

REFRESH DEFERRED または REFRESH IMMEDIATE が指定されていると、全選択で次のものを指定できません (SQLSTATE 428EC)。

- マテリアライズ照会表、宣言済み一時表、または任意の FROM 文節での型付き表への参照
- ビューの全選択が、マテリアライズ照会表の全選択に関してリストされたいずれかの制限に違反する場合の、そのビューへの参照
- 参照タイプまたは DATALINK タイプ (あるいはこれらのタイプに基づく特殊タイプ) である式
- 次のいずれかの属性を持つ関数:
  - EXTERNAL ACTION
  - LANGUAGE SQL
  - CONTAINS SQL
  - READS SQL DATA
  - MODIFIES SQL DATA
- 物理的特性に依存する関数 (たとえば、DBPARTITIONNUM、HASHEDVALUE)
- システム・オブジェクトに対する表またはビュー参照 (Explain 表も指定できません)
- 構造タイプまたは LOB タイプ (または LOB タイプに基づいた特殊タイプ) である式

REPLICATED を指定する場合は、以下の制限が適用されます。

- GROUP BY 文節は許可されていません。
- マテリアライズ照会表は単一の表だけを参照できます。すなわち、結合を組み込むことはできません。

REFRESH IMMEDIATE は、以下の場合に指定されます。

- 照会は副選択でなければなりません。ただし、例外として、UNION ALL は GROUP BY の入力表式においてサポートされます。
- 照会は再帰的であってはなりません。
- 照会に以下のものを含めることはできません。
  - ニックネームへの参照
  - deterministic 関数でない関数
  - スカラー全選択
  - 全選択を持つ述部
  - 特殊レジスター
  - SELECT DISTINCT
- FROM 文節で複数の表またはビューを参照している場合、明示的な INNER JOIN 構文を使わずに内部結合を 1 つだけ定義できます。
- GROUP BY 文節を指定する場合、以下の考慮事項が当てはまります。
  - サポートされている列関数は SUM、COUNT、COUNT_BIG、および GROUPING (DISTINCT は指定しない)。選択リストには COUNT(*) または COUNT_BIG(*) 列が含まれていなければなりません。マテリアライズ照会表の選択リストには、SUM(X) が含まれています。この X には、NULL 可能な引き数が入ります。マテリアライズ照会表の選択リストには、COUNT(X) も含まれていなければなりません。これらの列関数は、式の一部とすることはできません。
  - HAVING 文節は許可されていません。
  - 複数パーティションのデータベース・パーティション・グループ内の場合、パーティション・キーは GROUP BY 項目のサブセットでなければなりません。
- マテリアライズ照会表には重複した行があってはならず、 GROUP BY 文節が指定されているかどうかによって、この固有性要件に特有の、以下の制限が適用されます。
  - GROUP BY 文節を指定する場合、以下の固有性に関連した制限が当てはまります。
    - すべての GROUP BY 項目が選択リストに含まれていること。
    - GROUP BY に GROUPING SETS、CUBE、または ROLLUP が含まれている場合、選択リスト内の GROUP BY 項目とそれに関連した GROUPING 列関数は、結果セットのユニーク・キーを形成していなければなりません。したがって、以下の制約事項が満たされていなければなりません。
      - どのグループ・セットも反復することはできない。例えば、`ROLLUP(X,Y),X` は指定できません。これは `GROUPING SETS((X,Y),(X),(X))` と同等だからです。

- X が、GROUPING SETS、CUBE、または ROLLUP 内に出現する NULL 可能な GROUP BY 項目である場合、選択リスト内に GROUPING(X) がなければならない。

– GROUP BY 文節を指定しない場合、以下の固有性に関連した制限が当てはまります。

- マテリアライズ照会表のユニーク性要件は、それぞれの基礎表に定義されているユニーク・キー制約の 1 つからマテリアライズ・ビュー用のユニーク・キーを導出することによって達成されます。したがって、基礎表には少なくとも 1 つのユニーク・キー制約が定義されている必要があり、それらのキーの列が、マテリアライズ照会表定義の選択リストに現れていなければなりません。

- MAINTAINED BY FEDERATED_TOOL を指定する場合は、ニックネームへの参照だけが FROM 文節で許可されます。

REFRESH DEFERRED が指定され、その後のステートメントで関連したステージング表を提供する予定でマテリアライズ照会表を作成する場合、マテリアライズ照会表の全選択は、 REFRESH IMMEDIATE オプションによってマテリアライズ照会表を作成するのに使用した全選択と同じ制限事項および規則に従う必要があります。

全選択に GROUP BY 文節が含まれるマテリアライズ照会表は、全選択で参照される表からの要約されたデータです。そのようなマテリアライズ照会表は、*サマリー表* と呼ばれます。サマリー表は、マテリアライズ照会表が特殊化したタイプの表です。

WITH NO DATA
照会は、表を定義するときにだけ使われます。表は照会結果を使って移植されず、REFRESH TABLE ステートメントは使用できません。 CREATE TABLE ステートメントが完了すると、表はマテリアライズ照会表と見なされなくなります。

表の列は、全選択の結果である列の定義に基づいて定義されます。全選択によって FROM 文節の 1 つの表が参照される場合、その表の列である選択リスト項目は、参照される表の列名、データ・タイプ、そして NULL 可能特性を使って定義されます。

refreshable-table-options
マテリアライズ照会表の属性の再生可能オプションを定義します。

DATA INITIALLY DEFERRED
データは CREATE TABLE ステートメントの一部として表に挿入されません。データを表に挿入するには、 *table-name* (表名) を指定する REFRESH TABLE ステートメントが使用されます。

REFRESH
表のデータを保守する方法を示します。

DEFERRED
REFRESH TABLE ステートメントを使っていつでも表のデータをリフレッシュできます。表のデータには、 REFRESH TABLE ステートメント処理時のスナップショットである照会結果が反映されるにすぎません。この属性を定義したシステム保守済みマテリアライズ照会表には、

INSERT、 UPDATE、または DELETE ステートメントを使用できません (SQLSTATE 42807)。この属性を定義したユーザー保守のマテリアライズ照会表には、 INSERT、UPDATE、または DELETE ステートメントを使用できます。

IMMEDIATE
DELETE、INSERT、または UPDATE の一部として基礎表に加えられた変更は、マテリアライズ照会表にカスケードされます。その場合、表の内容は、どのポイント・イン・タイム指定でも、指定した *subselect* (副選択) を処理する場合と同じ内容になります。この属性を定義したマテリアライズ照会表には、INSERT、 UPDATE、または DELETE ステートメントを使用できません (SQLSTATE 42807)。

ENABLE QUERY OPTIMIZATION
適切な状況下では、マテリアライズ照会表を照会最適化に使用することができます。

DISABLE QUERY OPTIMIZATION
マテリアライズ照会表を照会の最適化に使用しません。それでもその表を直接照会することはできます。

MAINTAINED BY
マテリアライズ照会表のデータが、システム、ユーザー、またはレプリケーション・ツールのいずれによって保守されるかを指定します。デフォルトは SYSTEM です。

SYSTEM
マテリアライズ照会表のデータがシステムによって保守されるように指定します。

USER
マテリアライズ照会表のデータがユーザーによって保守されるように指定します。ユーザーは、ユーザー保守済みマテリアライズ照会表に対して、更新、削除、また挿入操作を許可されます。システム保守のマテリアライズ照会表で使用される REFRESH TABLE ステートメントは、ユーザー保守のマテリアライズ照会表に対しては呼び出せません。
REFRESH DEFERRED マテリアライズ照会表だけが、MAINTAINED BY USER として定義されることが可能です。

FEDERATED_TOOL
マテリアライズ照会表のデータがレプリケーション・ツールによって保守されるように指定します。システム保守のマテリアライズ照会表で使用される REFRESH TABLE ステートメントは、フェデレーテッド・ツール保守のマテリアライズ照会表に対しては呼び出せません。
REFRESH DEFERRED のマテリアライズ照会表だけが、MAINTAINED BY FEDERATED_TOOL として定義されることが可能です。

staging-table-definition
関連付けられたマテリアライズ照会表を通して間接的に、ステージング表によりサポートされる照会を定義します。マテリアライズ照会表の基礎表も、関連付けられたステージング表の基礎表です。ステージング表は、基礎表の内容と同期化するためにマテリアライズ照会表に適用する必要のある変更を収集します。

staging-column-name
ステージング表の列の名前を指定します。列名のリストを指定する場合、ステージング表を定義するマテリアライズ照会表の列の名前の数よりも、 *2* つ名前の数が多くなければなりません。そのマテリアライズ照会表がマテリアライズ照会表の複製である場合、またはマテリアライズ照会表を定義している照会が GROUP BY 文節を含んでいない場合には、ステージング表を定義するマテリアライズ照会表の列の名前の数よりも、 *3* つ名前の数が多くなければなりません。各 column-name (列名) は、ユニークかつ非修飾でなければなりません。列名のリストの指定がない場合、表の列は、関連付けられたマテリアライズ照会表の列名を継承します。追加の列は GLOBALTRANSID および GLOBALTRANSTIME と名づけられ、 3 番目の列が必要な場合には OPERATIONTYPE という名前が付けられます。

表 4. ステージング表に加えれる追加の列

列名	データ・タイプ	列の説明
GLOBALTRANSID	CHAR(8) FOR BIT DATA	伝搬された各行のグローバル・トランザクション ID
GLOBALTRANSTIME	CHAR(13) FOR BIT DATA	トランザクションのタイム・スタンプ
OPERATIONTYPE	INTEGER	伝搬された行に対する操作。挿入、更新、または削除のいずれか。

関連付けられたマテリアライズ照会表の任意の列が、生成された列名と重複する場合には、列名のリストを指定する必要があります (SQLSTATE 42711)。

FOR *table-name2*
ステージング表の定義に使用されるマテリアライズ照会表を指定します。名前 (暗黙的または明示的なスキーマ名を含む) は、 REFRESH DEFERRED に定義された現行サーバーに存在するマテリアライズ照会表を指定していなければなりません。関連付けられたマテリアライズ照会表の全選択は、REFRESH IMMEDIATE オプションによってマテリアライズ照会表を作成するのに使用した全選択と同じ制限事項および規則に従う必要があります。

ステージング表の内容が関連付けられたマテリアライズ照会表および基礎ソース表と整合する場合には、マテリアライズ照会表をリフレッシュするために、REFRESH TABLE ステートメントを呼び出して、ステージング表の内容を使用できます。

PROPAGATE IMMEDIATE
削除、挿入、または更新操作の一部として基礎表に加えられた変更は、ステージング表の同じ削除、挿入、または更新操作にカスケードされます。ステージング表が不整合としてマークされていないのであれば、任意のポイント・イン・タイムに、ステージング表の内容は最後にマテリアライズ照会表をリフレッシュしてから基礎表までの差分変更します。

LIKE *table-name1* **または** *view-name* **または** *nickname*
表の列の名前と記述が、指定された表 (*table-name1*)、ビュー (*view-name*)、またはニックネーム (*nickname*) の列とまったく同じであることを指定します。
LIKE の後に指定する名前は、カタログに存在する表、ビューまたはニックネー

ム、あるいは宣言済み一時表を識別するものでなければなりません。型付き表または型付きビューを指定することはできません (SQLSTATE 428EC)。

LIKE は、*n* 列を暗黙的な定義で使います。*n* は、指定した表、ビューまたはニックネームにおける列数です。

- 表が特定されると、暗黙的な定義には *table-name1* のそれぞれの列の列名、データ・タイプ、および NULL 可能特性が入ります。 EXCLUDING COLUMN DEFAULTS を指定しないと、列のデフォルト値も入ります。
- ビューが特定されると、暗黙的な定義には *view-name* に指定した全選択のそれぞれの結果列の列名、データ・タイプ、および NULL 可能特性が入ります。
- ニックネームが特定されると、暗黙的な定義には *nickname* のそれぞれの列の列名、データ・タイプ、および NULL 可能特性が入ります。

copy-attributes 文節に基づいて、列のデフォルトと IDENTITY 列属性を組み込んだり除外したりすることができます。さらにこの暗黙的な定義には、指定した表、ビュー、またはニックネームの他の属性は含まれません。したがって、新しい表にはユニーク制約、外部キー制約、トリガー、または索引はありません。表は IN 文節で暗黙的にまたは明示的に指定した表スペースの中に作成されます。また、任意指定の他の文節を指定した場合に限り、この表にその任意指定の文節が含まれます。

copy-options
これらのオプションは、ソース結果表の定義 (表、ビュー、または全選択) の追加属性をコピーするかどうかを指定します。

INCLUDING COLUMN DEFAULTS
ソース結果表の定義の更新可能な各列の列デフォルトをコピーします。更新可能でない列では、作成される表の対応列にデフォルトが定義されないことになります。

LIKE *table-name* が指定され、しかも *table-name* が基本表または宣言済み一時表を示す場合、 INCLUDING COLUMN DEFAULTS がデフォルトになります。

EXCLUDING COLUMN DEFAULTS
ソース結果表の定義から列デフォルトはコピーされません。

この文節がデフォルトです。ただし、LIKE *table-name* が指定され、かつ *table-name* が基本表または宣言済み一時表を示す場合を除きます。

INCLUDING IDENTITY COLUMN ATTRIBUTES
可能であれば、ソース結果表の定義から IDENTITY 列属性がコピーされます。 IDENTITY 列属性をコピーできるのは、表、ビュー、または全選択内の対応する列のエレメントが、識別特性をもつ基本表列名に直接または間接にマップされる表列の名前またはビュー列の名前である場合です。これら以外の場合はすべて、新規表の列には識別特性は備わりません。たとえば、以下のような場合です。

- 全選択の選択リストに、IDENTITY 列名の複数のインスタンスが入っている場合 (つまり、同一列の複数回の選択の場合)
- 全選択の選択リストに複数の IDENTITY 列が含まれている (つまり、結合が関与している) 場合

- IDENTITY 列が選択リスト内の式に組み込まれている場合
- 全選択にセット演算 (UNION (合併)、EXCEPT (差)、または INTERSECT (論理積)) が含まれている場合

EXCLUDING IDENTITY COLUMN ATTRIBUTES
ソース結果表の定義から IDENTITY 列属性はコピーされません。

ORGANIZE BY DIMENSIONS *(column-name,...)*
表データをクラスター化するために使用する、各列または列のグループのディメンションを指定します。ディメンション・リストで括弧を使用すると、列のグループは 1 つのディメンションとして扱われように指定されます。
DIMENSIONS キーワードはオプションです。

クラスタリング・ブロック索引はそれぞれ指定されたディメンション用に自動的に保持され、文節で使用されるすべての列で構成されるブロック索引は、どのクラスタリング・ブロック索引にもすべての列が含まれていない場合には保持されます。 ORGANIZE BY 文節で使用される列の集合は、CREATE INDEX ステートメントの規則に従う必要があります。

ORGANIZE BY 文節で指定された各列名は、表に対して定義されなければならず (SQLSTATE 42703)、ディメンションはディメンション・リストで複数回現れることはありません (SQLSTATE 42709)。

表のページは、表スペースのエクステント・サイズと同じサイズのブロックに配置され、各ブロックの行すべては同じディメンション値の組み合わせを含みます。

ORGANIZE BY KEY SEQUENCE *sequence-key-spec*
表を、指定された範囲のキー・シーケンスの値に基づいて、固定サイズの昇順キー・シーケンスに編成するよう指定します。このような方法で編成された表を、*レンジ・クラスター表* といいます。定義された範囲内にある有効なキー値のそれぞれには、物理表上の位置があらかじめ決定されています。レンジ・クラスター表に必要なストレージは、表が作成される時点で使用可能になっていなければならず、また、ストレージには、範囲内にある行の数に行のサイズを乗算しただけのスペースが必要です (スペース所要量の決定に関する詳細は、 398 ページの『行サイズ』および 398 ページの『バイト・カウント』を参照してください)。

column-name
レンジ・クラスター表のシーケンスを決定するユニーク・キーに含まれる、表の列を指定します。列のデータ・タイプは SMALLINT、INTEGER、または BIGINT (SQLSTATE 42611) でなければならず、また列は NOT NULL (SQLSTATE 42831) として定義される必要があります。シーケンス・キーの中で、同じ列を複数回指定することはできません。指定する列の数が 16 を超えてはなりません (SQLSTATE 54008)。

列ごとに昇順で指定されたキー・シーケンスで、列のカタログにユニーク索引項目が自動的に作成されます。索引の名前は、SQL の後に文字のタイム・スタンプ (*yymmddhhmmssxxx*) が続き、スキーマ名として SYSIBM がついたものになります。表の編成はこのキーによって配列されているため、実際の索引オブジェクトはストレージに作成されません。同じ列に、レンジ・クラスター表のシーケンス・キーとして主キーまたはユニークな制約が定義されている場合は、この同じ索引項目が制約に使用されます。

キー・シーケンスの指定には、列の制約を反映するチェック制約が存在します。 DISALLOW OVERFLOW 文節が指定されていると、チェック制約の名前が RCT になり、チェック制約が施行されます。 ALLOW OVERFLOW 文節が指定されていると、チェック制約の名前が RCT_OFLOW になり、チェック制約は施行されません。

STARTING FROM *constant*
column-name の範囲の下限となる定数を指定します。指定された定数より小さい値は、 ALLOW OVERFLOW オプションが指定されている場合以外、許可されません。 *column-name* が SMALLINT または INTEGER 列の場合は、定数を INTEGER 定数にする必要があります。 *column-name* が BIGINT 列の場合は、定数を INTEGER または BIGINT 定数にする必要があります (SQLSTATE 42821)。開始の定数が指定されない場合、デフォルト値は 1 です。

ENDING AT *constant*
column-name の範囲の上限となる定数を指定します。指定された定数より大きい値は、 ALLOW OVERFLOW オプションが指定されている場合以外、許可されません。なお、終了の定数は、開始の定数より大きくなければなりません。 *column-name* が SMALLINT または INTEGER 列の場合は、定数を INTEGER 定数にする必要があります。 *column-name* が BIGINT 列の場合は、定数を INTEGER または BIGINT 定数にする必要があります (SQLSTATE 42821)。

ALLOW OVERFLOW
レンジ・クラスター表で、定義された範囲外の値を行のキー値として許可することを指定します。レンジ・クラスター表がオーバーフローを許可するように作成される場合、キー値が範囲外にある行にはあらかじめ決定された配列がなく、行は定義された範囲の一番下に置かれます。これらのオーバーフロー行に関係する操作は、定義された範囲内にキー値がある行での操作と比べて非効率的です。

DISALLOW OVERFLOW
レンジ・クラスター表で、定義された範囲外の値を行のキー値として許可しないことを指定します (SQLSTATE 23513)。オーバーフロー行を許可しないレンジ・クラスター表では、常に、すべての行が昇順キー・シーケンスで保守されます。

column-definition
列の属性を定義します。

column-name
表を構成する列の名前を指定します。名前を修飾したり、表の複数の列に対して同じ名前を使用することはできません (SQLSTATE 42711)。

表には、以下のものを指定できます。

- 4K ページ・サイズの場合、最大 500 列。列のバイト・カウントは 4 005 を超えてはなりません。
- 8K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 8 101 を超えてはなりません。
- 16K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 16 293 を超えてはなりません。

- 32K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 32 677 を超えてはなりません。

詳細については、 398 ページの『行サイズ』を参照してください。

data-type
以下のリストのタイプのいずれかを指定します。データ・タイプは、以下のとおりです。

SMALLINT
短整数。

INTEGER または **INT**
長精度整数。

BIGINT
64 ビット整数。

FLOAT(*integer***)**
単精度または倍精度の浮動小数点数 (*integer* の値によって異なる)。integer の値は、1 ～ 53 の範囲の整数でなければなりません。 1 ～ 24 の値は単精度、25 ～ 53 の値は倍精度を示します。

また、以下を指定することもできます。

REAL	単精度浮動小数点。
DOUBLE	倍精度浮動小数点。
DOUBLE PRECISION	倍精度浮動小数点。
FLOAT	倍精度浮動小数点。

DECIMAL*(precision-integer, scale-integer)* または **DEC***(precision-integer, scale-integer)*
10 進数。最初の整数は数値の精度、つまり数字の総桁数です。これは、1 ～ 31 の範囲で指定できます。 2 番目の整数は、数値の位取り、つまり、小数点の右側の桁数です。これは、0 ～ 数値の精度までの範囲で指定できます。

精度と位取りが指定されない場合、5,0 のデフォルト値が使用されます。 **NUMERIC** および **NUM** は、 **DECIMAL** および **DEC** の同義語として使用可能です。

CHARACTER*(integer)*、または **CHAR** *(integer)*、または **CHARACTER**、または **CHAR**
長さ *integer* (整数) の固定長文字ストリング。長さは、1 ～ 254 の範囲で指定できます。 長さの指定がない場合は、1 文字の長さを指定したものと見なされます。

VARCHAR*(integer)*、または **CHARACTER VARYING***(integer)*、または **CHAR VARYING***(integer)*
長さが *integer* の可変長文字ストリング。長さは、1 ～ 32,672 の範囲で指定できます。

LONG VARCHAR
最大長 32 700 の可変長文字ストリング。

FOR BIT DATA
列の内容をビット (バイナリー) データとして扱うように指定します。他のシステムとのデータ交換の過程で、コード・ページ変換は行われません。比較は、データベース照合シーケンスに関係なくバイナリーで行われます。

BLOB または BINARY LARGE OBJECT*(integer [K | M | G])*
バイナリー・ラージ・オブジェクト・ストリング (最大長をバイト単位で指定)。

長さは、1 ～ 2 147 483 647 バイトの範囲で指定できます。

integer (整数) だけを指定した場合は、それが最大長になります。

integer K (大文字または小文字) を指定した場合、最大長は *integer* の 1 024 倍になります。 *integer* の最大値は 2 097 152 です。計算結果が 2 147 483 648 を超える K、M、または G の倍数を指定すると、使用される実際の値は 2 147 483 647 (2 ギガバイト - 1 バイト) になります。これは LOB 列の最大長です。

integer M を指定した場合、最大長は *integer* の 1 048 576 倍になります。 *integer* の最大値は 2 048 です。

integer G を指定した場合、最大長は *integer* の 1 073 741 824 倍になります。 *integer* の最大値は 2 です。

長さの指定がない場合、長さが 1 048 576 (1 メガバイト) であると想定されます。

1 ギガバイトを超える BLOB ストリングを作成するには、 NOT LOGGED オプションを指定する必要があります。

integer と K、M、または G の間には、任意の数のスペースを使用できます。スペースは必須ではありません。たとえば、次の例はすべて有効です。

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

CLOB または CHARACTER (CHAR) LARGE OBJECT*(integer [K | M | G])*
文字ラージ・オブジェクト・ストリング (最大長をバイト単位で指定)。

integer K | M | G の意味は、BLOB の場合と同じです。

長さの指定がない場合、長さが 1 048 576 (1 メガバイト) であると想定されます。

1 ギガバイトを超える CLOB ストリングを作成するには、 NOT LOGGED オプションを指定する必要があります。

CLOB 列の場合に、FOR BIT DATA 文節を指定することはできません。ただし、CLOB 列に CHAR FOR BIT DATA ストリングを割り当てることができ、 CLOB ストリングに CHAR FOR BIT DATA ストリングを連結することができます。

DBCLOB*(integer [K | M | G])*
2 バイト文字ラージ・オブジェクト・ストリング (最大長を 2 バイト文字の数で指定)。

integer *K* | *M* | *G* の意味は、BLOB の場合に類似しています。指定する数値が 2 バイト文字 1 個を 1 文字と数えた値であることと、最大サイズが 2 バイト文字 1 073 741 823 個であるという点が違います。

長さの指定がない場合、長さが 1 048 576 (2 バイト文字) であると想定されます。

1 ギガバイトを超える DBCLOB ストリングを作成するには、 NOT LOGGED オプションを指定する必要があります。

GRAPHIC*(integer)*
長さ *integer* (整数) の固定長 GRAPHIC ストリング。長さは、1 ～ 127 の範囲で指定できます。長さの指定がない場合、長さが 1 であると想定されます。

VARGRAPHIC*(integer)*
長さが *integer* の可変長 GRAPHIC ストリング。長さは、1 ～ 16,336 の範囲で指定できます。

LONG VARGRAPHIC
最大長 16,350 の可変長 GRAPHIC ストリングを示します。

DATE
日付を示します。

TIME
時刻を示します。

TIMESTAMP
タイム・スタンプを示します。

DATALINK または **DATALINK***(integer)*
データベース外に保管されたデータ・ファイルへのリンクを示します。

表内の列は ″アンカー値″ で構成されます。このアンカー値には、外部データだけではなく、任意コメントへのリンクを確立したり保持したりするための参照情報が含まれています。

DATALINK 列の長さは 200 バイトです。 *integer* を指定する場合、200 でなければなりません。長さの指定がない場合、長さは 200 バイトであると想定されます。

DATALINK 値とは、一連の組み込みスカラー関数を持つカプセル化された値です。 DATALINK 値は DLVALUE という名前の関数が作成し、また特別な環境で DATALINK 値を構成するためには DLNEWCOPY、 DLPREVIOUSCOPY、および DLREPLACECONTENT という名前の関数が使用されます。 (DLVALUE は、正規の DATALINK 値を構成するために使用してください。) DATALINK 値から属性を抽出するために、以下の関数を使用することができます。

- DLCOMMENT
- DLLINKTYPE
- DLURLCOMPLETE
- DLURLCOMPLETEONLY
- DLURLCOMPLETEWRITE

- DLURLPATH
- DLURLPATHONLY
- DLURLPATHWRITE
- DLURLSCHEME
- DLURLSERVER

DATALINK 列には、以下の制限事項があります。

- 列は索引の一部であってはならない。したがって、この列を主キーまたはユニーク制約の列として組み込むことはできません (SQLSTATE 42962)。
- 列は参照制約の外部キーであってはならない (SQLSTATE 42830)。
- この列にデフォルト値 (WITH DEFAULT) を指定することはできない。列を NULL にすることができる場合、この列のデフォルト値は NULL です (SQLSTATE 42894)。

distinct-type-name

ユーザー定義タイプの中で特殊タイプであるものを示します。スキーマ名を伴わない特殊タイプ名を指定した場合、その特殊タイプ名は SQL パスのスキーマから探索することによって解決されます (このパスは、静的 SQL の場合は FUNCPATH プリプロセス・オプションによって、動的 SQL の場合は CURRENT PATH レジスターによって定義されます)。

特殊タイプを使用して列を定義する場合、その列のデータ・タイプはその特殊タイプになります。列の長さと位取りは、それぞれ特殊タイプのソース・タイプの長さと位取りになります。

特殊タイプを使用して定義された列が参照制約の外部キーである場合、主キーの対応する列のデータ・タイプは、同じ特殊タイプでなければなりません。

structured-type-name

ユーザー定義タイプの中で構造タイプであるものを示します。構造タイプ名の指定にスキーマ名が含まれていない場合、その構造タイプ名は SQL パスのスキーマから探索することによって決まります (このパスは、静的 SQL の場合は FUNCPATH プリプロセス・オプションによって、動的 SQL の場合は CURRENT PATH レジスターによって定義されます)。

構造タイプを使用して列を定義する場合、その列の静的データ・タイプはその構造タイプになります。その列には、 *structured-type-name* のサブタイプである動的タイプをもつ値を組み込むことができます。

構造タイプを使用して定義された列を、主キー、ユニーク制約、外部キー、索引キー、またはパーティション・キー内で使うことはできません (SQLSTATE 42962)。

列が、構造タイプを使用して定義されていて、ネストのいずれかのレベルで参照タイプ属性をもっている場合、その参照タイプ属性の有効範囲は解除されます。そのような属性を逆参照操作で使用するには、 CAST 指定を使って SCOPE を明示的に指定する必要があります。

タイプ DATALINK の属性をもつ構造タイプか、または DATALINK をソースとして派生された特殊タイプを使って列が定義されている場合、その列は NULL にしかできません。このようなタイプの場合にコンストラクター関数を使うと、エラーが戻される (SQLSTATE 428ED) ので、このタイプのインスタンスは列に挿入できません。

REF (*type-name2***)**
型付き表への参照。 *type-name2* の指定にスキーマ名が含まれていない場合、そのタイプ名は SQL パス上のスキーマを探索することによって決まります (このパスは、静的 SQL の場合は FUNCPATH プリプロセス・オプションによって、動的 SQL の場合は CURRENT PATH レジスターによって定義されます)。この列の基礎を成すデータ・タイプは、CREATE TYPE ステートメントの REF USING 文節で *type-name2* に対して指定された表示データ・タイプに基づくか、または *type-name2* の入ったデータ・タイプ階層のルート・タイプに基づきます。

column-options
表の列に関連した追加オプションを定義します。

NOT NULL
列に NULL 値が入るのを防止します。

NOT NULL を指定しない場合、列に NULL 値を含めることができます。また、そのデフォルト値は、NULL 値または WITH DEFAULT 文節で指定される値のいずれかになります。

lob-options
LOB データ・タイプのオプションを指定します。

LOGGED
列に対して行われた変更をログに書き込むことを指定します。このような列のデータは、データベース・ユーティリティー (RESTORE DATABASE など) によってリカバリー可能です。 LOGGED はデフォルト値です。

1 ギガバイトを超える LOB はログ記録することができず (SQLSTATE 42993)、 10 メガバイトを超える LOB はログ記録されない可能性があります。

NOT LOGGED
列に対して行われた変更をログに書き込まないことを指定します。

NOT LOGGED は、コミットやロールバックの操作には影響しません。つまり、トランザクションがロールバックされても、 LOB の値がログ記録されるか否かに関係なくデータベースの整合性は保持されます。ロギングされないので、ロールフォワード操作中、バックアップまたはロード操作の後の LOB データは、ロールフォワード操作中にログ・レコードを再生させることになった LOB 値をゼロで置換したものになります。クラッシュ・リカバリーの過程で、コミットされた変更とロールバックされた変更すべてに、予期された結果が反映されます。

COMPACT
後続の付加操作で使用するためのスペースを LOB ストレージの最

後に残すことなく、 LOB 列の値で消費されるディスク・スペースを最小限にすることを指定します (LOB 値が使用する最後のグループ内の余分なディスク・スペースすべてを解放します)。このようにしてデータを保管した場合、列に対する付加操作 (長さを増加する操作) でパフォーマンスが低下することがあります。

NOT COMPACT
列の LOB 値に対する将来の変更に備えて、いくらかのスペースを挿入するように指定します。これはデフォルトです。

datalink-options
DATALINK データ・タイプに関連したオプションを指定します。

LINKTYPE URL
リンク・タイプを URL として定義します。

NO LINK CONTROL
ファイルが存在するかどうかを判別する検査を行わないことを指定します。 URL の構文だけが検査されます。データベース・マネージャーによってファイルが制御されることはありません。

FILE LINK CONTROL
ファイルが存在するかどうかを判別する検査が行われるように指定します。データベース・マネージャーがファイルをより一層制御できるように、追加オプションが使用されます。

file-link-options
データベース・マネージャーによるファイル・リンク制御の度合いを定義する追加オプション。

INTEGRITY
DATALINK 値と実ファイルの間に存在するリンクの保全レベルを指定します。

ALL
DATALINK 値として指定されているものは、すべてデータベース・マネージャーによって制御されており、それらは標準的なファイル・システム・プログラム・インターフェースを使って削除したり名前変更したりすることはできません。

READ PERMISSION
DATALINK 値に指定されているファイルの読み取り許可を決定する方法を指定します。

FS
読み取りアクセス許可は、ファイル・システム許可によって決められます。列からファイル名を検索することなく、そのようなファイルにアクセスできます。

DB
読み取りアクセス許可は、データベースによって決められます。有効なファイル・アクセス・トークンを渡してから、表から取り出された DATALINK 値をオープン操作で戻すことによってのみファイルへのアクセスが可能になります。

WRITE PERMISSION
DATALINK 値に指定されているファイルへの書き込み許可を決定する方法を指定します。

FS
書き込みアクセス許可は、データベースによって決められます。列からファイル名を検索することなく、そのようなファイルにアクセスできます。

BLOCKED
書き込みアクセスはブロックされます。このファイルはどのインターフェースを介しても直接更新することはできません。この情報を更新するには、代替メカニズムを使用する必要があります。たとえば、ファイルがコピーされて、そのコピーが更新されるとします。その場合には、DATALINK 値もファイルの新しいコピーを指すように更新されなければなりません。

ADMIN
書き込み許可は、Data Links Manager によって判別されます。有効な書き込みトークンを渡してから、表から取り出された DATALINK 値をオープン操作で DLURLCOMPLETEWRITE または DLURLPATHWRITE スカラー関数を使用して戻すことによってのみ、ファイルへの書き込みアクセスが可能になります。 この値は、READ PERMISSION DB も指定された場合にのみ指定が可能です。

指定されたリンク・ファイルに対するアクセス権は、Data Links Manager で定義され、保守されます。

ユーザー (更新者) が有効な書き込みトークンを持って書き込み用にファイルをオープンすると、他のユーザーは有効な読み取りまたは書き込みトークンを使用して、読み取り用にファイルをオープンすることは依然として可能です。しかし同じ書き込みトークンを使用して、書き込み用にファイルを繰り返しオープンできるのは、同じ更新者だけです。また同じ更新者は、その後に読み取り操作を実行するためにも同じ書き込みトークンが必要です。

REQUIRING TOKEN FOR UPDATE
ファイル更新を完了するには、ファイルをオープンして変更するのに使用した書き込みトークンは、 SQL UPDATE ステートメントに DLNEWCOPY または DLPREVIOUSCOPY スカラー関数を呼び出す際に指定したファイル参照に含まれていなければなりません。

NOT REQUIRING TOKEN FOR UPDATE
ファイル更新を完了するためには、 SQL UPDATE ステートメントに DLNEWCOPY または DLPREVIOUSCOPY スカラー関数を呼び出す際に指定したファイル参照には書き込みトークンは必要ではありません。

RECOVERY
この列の値によって参照されているファイルの特定時点でのリカバリーを DB2 がサポートしているかどうかを指定します。

YES
DB2 は、この列の値で参照されているファイルの特定時点でのリカバリーをサポートしています。この値は、INTEGRITY ALL と WRITE PERMISSION BLOCKED または WRITE PERMISSION ADMIN とが指定されている場合にだけ指定することができます。

NO
ポイント・イン・タイム・リカバリーがサポートされていないことを指定します。

ON UNLINK
DATALINK 値が変更または削除された (リンク解除された) ときにファイル上で実行する処置を指定します。 WRITE PERMISSION FS が使用される際には、適用できないことに注意してください。

RESTORE
ファイルがリンク解除されたときに、ファイルがリンクされていた時点で許可を持つ所有者に、データ・リンク・ファイル・マネージャーがファイルを戻すように指定します。ユーザーがファイル・サーバーから登録を解除されている場合には、ファイルは特別に事前定義された ″dfmunknown″ ユーザー ID に割り当てられます。この値は、INTEGRITY ALL と WRITE PERMISSION BLOCKED または WRITE PERMISSION ADMIN とが指定されている場合にだけ指定することができます。

DELETE
リンク解除される際にファイルが削除されるように指定します。この値を指定することができるのは、 READ PERMISSION DB および WRITE PERMISSION BLOCKED または WRITE PERMISSION ADMIN も指定されている場合だけです。

MODE DB2OPTIONS
このモードは、一連のデフォルト・ファイル・リンク・オプションを定義します。 DB2OPTIONS によって定義されるデフォルト値は、以下のとおりです。

- INTEGRITY ALL
- READ PERMISSION FS
- WRITE PERMISSION FS
- RECOVERY NO

WRITE PERMISSION FS が使用されている場合、ON UNLINK は適用できません。

SCOPE
参照タイプ列の有効範囲を指定します。

逆参照演算子の左オペランド、または DEREF 関数の引き数として使用する列には、すべて有効範囲を指定する必要があります。ターゲット表が定義されるように、後続する ALTER TABLE ステートメントまで参照タイプ列の指定を遅らせることができます (通常は、相互参照表の場合に適用する)。

typed-table-name
型付き表の名前。この表は既に存在しているものか、作成する表と同じ名前のものでなければなりません (SQLSTATE 42704)。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-table-name* のタイプを表します (SQLSTATE 428DM)。 *column-name* に割り当てられた値が、 *typed-table-name* に存在する行を実際に参照しているかどうかを示す検査は行われません。

typed-view-name
型付きビューの名前。このビューは既に存在しているものか、作成するビューと同じ名前のものでなければなりません (SQLSTATE 42704)。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-view-name* のタイプを表します (SQLSTATE 428DM)。 *column-name* に割り当てられた値が、 *typed-view-name* に存在する行を実際に参照しているかどうかを示す検査は行われません。

CONSTRAINT *constraint-name*
制約の名前を指定します。 *constraint-name* (制約名) は、同じ CREATE TABLE ステートメントにすでに指定されている制約を指定するものであってはなりません (SQLSTATE 42710)。

この文節が省略された場合は、表に定義されている既存の制約の ID の中でユニークな 18 文字の ID がシステムによって生成されます。 (ID は、″SQL″ と、タイム・スタンプに基づいて生成される 15 の数字から構成されます。)

PRIMARY KEY 制約またはユニーク制約とともに使用した場合、この *constraint-name* は、制約をサポートするために作成される索引の名前として使用されます。

PRIMARY KEY
これは、1 つの列からなる主キーを定義する簡単な方法として用意されています。つまり、PRIMARY KEY が列 C の定義で指定されている場合、その効果は、PRIMARY KEY(C) 文節を独立した文節として指定する場合と同じです。

表が副表である場合、主キーはスーパー表から継承されるので (SQLSTATE 429B3)、主キーを指定することはできません。

この後の *unique-constraint* の説明の中の PRIMARY KEY を参照してください。

UNIQUE
これは、1 つの列からなるユニーク・キーを定義する簡単な方法です。

すなわち、UNIQUE を列 C の定義に指定すると、 UNIQUE(C) 文節を独立した文節として指定した場合と同じ結果になります。

表が副表である場合、ユニーク制約はスーパー表から継承されるので (SQLSTATE 429B3)、ユニーク制約を指定することはできません。

この後の *unique-constraint* の説明の中の UNIQUE を参照してください。

references-clause
これは、1 つの列からなる外部キーを定義する簡単な方法として用意されています。つまり、references-clause が列 C の定義に指定されている場合、その効果は、列として C しか指定されていない FOREIGN KEY 文節の一部として references-clause が指定された場合と同じになります。

後述の *referential-constraint* の *references-clause* の項を参照してください。

CHECK (*check-condition***)**
これは、1 つの列に適用されるチェック制約を定義する簡単な方法として用意されています。後述の CHECK (*check-condition*) を参照してください。

INLINE LENGTH *integer*
このオプションは、構造タイプを使って定義された列に対してだけ有効であり (SQLSTATE 42842)、行内の残りの値とともにインラインで保管する構造タイプのインスタンスの最大バイト・サイズを指示します。インラインで保管できない構造タイプのインスタンスは、 LOB 値が処理されるのに似た方法で、基本表の行とは別に保管されます。これは自動的に行われます。

構造タイプ列のデフォルトの INLINE LENGTH は、このタイプのインライン長になります (明示的に指定するか、または CREATE TYPE ステートメント内のデフォルトとして)。構造タイプの INLINE LENGTH が 292 未満の場合、列の INLINE LENGTH には値 292 が使われます。

注: サブタイプのインライン長は、デフォルトのインライン長には数えられませんが、それは、CREATE TABLE 時に明示的に INLINE LENGTH を指定して、現在および将来のサブタイプに対処できるようにしておかないと、サブタイプのインスタンスはインラインに適合しないことがあることを意味します。

明示的な INLINE LENGTH の値は少なくとも 292 でなければならず、32672 を超えてはなりません (SQLSTATE 54010)。

COMPRESS SYSTEM DEFAULT
システム・デフォルト値 (つまり、特定の値が指定されない場合にデータ・タイプとして使用されるデフォルト値) が最小限のスペースを使用して保管されるように指定します。 VALUE COMPRESSION 文節が指定されていない場合には警告が出され (SQLSTATE 01648)、システム・デフォルト値は最小限のスペースを使用しては保管されません。

システム・デフォルト値がこのような方法で保管されると、列に対する許可や更新操作の際に余分な検査が行われるために、若干パフォーマンスが低下します。

基本データ・タイプは、DATE、TIME、または TIMESTAMP であってはなりません (SQLSTATE 42842)。基本データ・タイプが可変長ストリングの場合には、この文節は無視されます。表が VALUE COMPRESSION に設定されている場合は、長さ 0 のストリング値は自動的に圧縮されます。

generated-column-spec

default-clause
列のデフォルト値を指定します。

WITH
オプション・キーワード。

DEFAULT
INSERT で値が提供されなかった場合、もしくは INSERT や UPDATE で DEFAULT が指定されている場合に、デフォルト値を提供します。 DEFAULT キーワードの後にデフォルト値が指定されていない場合、使用されるデフォルト値は列のデータ・タイプによって異なります。 『ALTER TABLE』 を参照してください。

列を DATALINK として定義する場合、デフォルト値は指定できません (SQLSTATE 42613)。可能なデフォルト値は NULL だけです。

列が型付き表の列に基づいている場合、デフォルト値の定義時には特定のデフォルト値を指定する必要があります。型付き表のオブジェクト ID の列には、デフォルト値を指定することはできません (SQLSTATE 42997)。

列が特殊タイプを使用して定義される場合、列のデフォルト値は、特殊タイプにキャストされたソース・データ・タイプのデフォルト値になります。

構造タイプを使用して列を定義する場合は、 *default-clause* を指定できません (SQLSTATE 42842)。

column-definition から DEFAULT を省略すると、その列のデフォルト値として NULL 値が使用されます。そのような列を NOT NULL と定義すると、その列には有効なデフォルトはなくなります。

default-values
default-values に指定できるデフォルト値のタイプは、以下のとおりです。

constant
列のデフォルト値として定数を指定します。指定する定数は、次の条件を満たしていなければなりません。

- 第 3 章に示されている割り当ての規則に従って、その列に割り当てることができる値でなければなりません。

- その列が浮動小数点データ・タイプとして定義されている場合を除き、浮動小数点の定数を指定してはなりません。
- 定数が 10 進定数の場合、その列のデータ・タイプの位取りを超えるゼロ以外の数字を含めてはなりません (たとえば、 DECIMAL(5,2) の列のデフォルト値として 1.234 を指定することはできません)。
- 指定する定数が 254 文字を超えてはなりません。この制約には、引用符文字や 16 進定数の X などの接頭部文字も含まれます。さらに、定数が *cast-function* の引き数の場合には、完全修飾された関数名から取った文字や括弧も含めて、この制限を超えてはなりません。

datetime-special-register
INSERT、UPDATE、または LOAD の実行時における日時特殊レジスターの値 (CURRENT DATE、 CURRENT TIME、または CURRENT TIMESTAMP) を、その列のデフォルト値として指定します。その列のデータ・タイプは、指定した特殊レジスターに対応するデータ・タイプでなければなりません (たとえば、 CURRENT DATE を指定した場合、データ・タイプは DATE でなければなりません)。

user-special-register
INSERT、UPDATE、または LOAD の実行時におけるユーザー特殊レジスターの値 (CURRENT USER、SESSION_USER、SYSTEM_USER) を、その列のデフォルトとして指定します。その列のデータ・タイプは、ユーザー特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。なお、SESSION_USER の代わりに USER を、 CURRENT USER の代わりに CURRENT_USER を指定することもできます。

CURRENT SCHEMA
INSERT、UPDATE、または LOAD の実行時における CURRENT SCHEMA 特殊レジスターの値を、その列のデフォルト値として指定します。 CURRENT SCHEMA を指定した場合、その列のデータ・タイプは、 CURRENT SCHEMA 特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。

NULL
その列のデフォルト値として NULL を指定します。 NOT NULL の値が指定された場合は、 DEFAULT NULL を同じ列定義に指定できますが、その列をデフォルト値に設定しようとするとエラーが生じます。

cast-function
この形式のデフォルト値は、特殊タイプ (distinct type)、BLOB、または日時 (DATE、TIME、または TIMESTAMP) データ・タイプとして定義された列に対してのみ使用することができます。特殊タイプの場合、BLOB や日時タイプに

基づく例外があり、関数名が列の特殊タイプの名前に一致していなければなりません。スキーマ名で修飾されている場合には、その特殊タイプのスキーマ名と同じでなければなりません。修飾されていない場合には、関数の解決に用いるスキーマ名は特殊タイプのスキーマ名と同じでなければなりません。日時タイプに基づく特殊タイプで、デフォルト値が定数の場合、必ず関数を使用する必要があります。さらに、その関数名は、暗黙または明示のスキーマ名 SYSIBM を持つ特殊タイプのソース・タイプ名に一致していなければなりません。他の日時列の場合は、対応する日時関数も使用できます。 BLOB に基づく BLOB または特殊タイプの場合も、関数を使用する必要があります。その関数名は、暗黙または明示のスキーマ名 SYSIBM を持つ BLOB でなければなりません。

constant
引き数として定数を指定します。指定する定数は、特殊タイプのソース・タイプに関する定数の規則 (特殊タイプでない場合は、データ・タイプに関する定数の規則) に従っていなければなりません。 *cast-function* が BLOB の場合には、定数としてストリング定数を指定する必要があります。

datetime-special-register
CURRENT DATE、CURRENT TIME、または CURRENT TIMESTAMP を指定します。列の特殊タイプのソース・タイプは、指定した特殊レジスターに対応するデータ・タイプでなければなりません。

user-special-register
CURRENT USER、SESSION_USER、または SYSTEM_USER を指定します。列の特殊タイプのソース・タイプのデータ・タイプは、少なくとも 8 バイトの長さのストリング・データ・タイプでなければなりません。 *cast-function* が BLOB の場合には、長さ属性が 8 バイト以上でなければなりません。

CURRENT SCHEMA
CURRENT SCHEMA 特殊レジスターの値を指定します。列の特殊タイプのソース・タイプのデータ・タイプは、 CURRENT SCHEMA 特殊レジスターの長さ属性よりも長いか等しい文字ストリングでなければなりません。 cast-function が BLOB の場合には、長さ属性が 8 バイト以上でなければなりません。

指定した値が無効な場合、エラーが戻されます (SQLSTATE 42894)。

GENERATED
DB2 が列の値を生成することを指示します。 その列が IDENTITY 列と見なされることになる場合には、GENERATED を指定する必要があります。

ALWAYS
行が表に挿入されるときや、*generation-expression* の結果値が変更されるたびに、 DB2 が常に列の値を生成することを指定します。この式の結果は、表に保管されます。データ伝搬、またはアンロードおよび再ロード操作を実行しているのでなければ、 GENERATED ALWAYS の値をお勧めします。 GENERATED ALWAYS は、生成列に必須の値です。

BY DEFAULT
DEFAULT 文節を指定して行が挿入または更新されるときに、明示的に値を指定しないかぎり、DB2 が列に値を生成することを指定します。データ伝搬を使用したり、アンロードおよび再ロードを実行したりするときは、 BY DEFAULT が推奨される値です。

明示的な要求ではありませんが、値のユニーク性を確保するために、ユニークな 1 列の索引を生成列で定義すべきです。

AS IDENTITY
列をこの表の IDENTITY 列にすることを指定します。 1 つの表には 1 つしか IDENTITY 列があってはなりません (SQLSTATE 428C1)。列に関連付けられたデータ・タイプがゼロの位取りの完全な数値タイプになっているか、ソース・タイプのユーザー定義特殊タイプがゼロの位取りの完全な数値タイプになっている場合だけ、IDENTITY キーワードが指定可能です (SQLSTATE 42815)。 ゼロの位取りの SMALLINT、INTEGER、BIGINT、または DECIMAL や、これらのタイプのうちのいずれかに基づいた特殊タイプは、完全な数値タイプと見なされます。これに対して、単精度および倍精度の浮動小数点は、近似数値データ・タイプと見なされます。参照タイプは、完全な数値タイプで表されていても、IDENTITY 列と定義することはできません。

IDENTITY 列は暗黙で NOT NULL になります。 IDENTITY 列は DEFAULT 文節を持つことができません (SQLSTATE 42623)。

START WITH *numeric-constant*
IDENTITY 列の最初の値を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 42815) かぎり、この列に割り当てることができる正または負の値にすることができます。デフォルトは、昇順シーケンスであれば MINVALUE、降順シーケンスであれば MAXVALUE です。

INCREMENT BY *numeric-constant*
連続した IDENTITY 列値のインターバルを指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができ (SQLSTATE 42815)、長精度整数定数の値を超えることはありません (SQLSTATE 42820)。

この値が負の場合、これは降順シーケンスです。 この値が 0 の場合、または正の場合は、昇順シーケンスになります。デフォルトは 1 です。

NO MINVALUE または **MINVALUE**
降順 IDENTITY 列が値の生成を循環または停止する最小値、あるいは最大値に達した後、昇順 IDENTITY 列が循環する最小値を指定します。

NO MINVALUE
昇順シーケンスの場合、値は START WITH 値、または START WITH が指定されなかった場合には 1 です。降順シーケンスの場合、列のデータ・タイプの最小値になります。これはデフォルトです。

MINVALUE *numeric-constant*
最小値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができます (SQLSTATE 42815) が、最大値以下でなければなりません (SQLSTATE 42815)。

NO MAXVALUE または **MAXVALUE**
昇順 IDENTITY 列が値の生成を循環または停止する最大値、あるいは最小値に達した後、降順 IDENTITY 列が循環する最大値を指定します。

NO MAXVALUE
昇順シーケンスの場合、値は列のデータ・タイプの最大値です。降順シーケンスの場合、値は START WITH 値、または START WITH が指定されなかった場合には -1 です。これはデフォルトです。

MAXVALUE *numeric-constant*
最大値にする数値定数を指定します。この値は、小数点の右側に非ゼロの数字がない (SQLSTATE 428FA) かぎり、この列に割り当てることができる正または負の値にすることができます (SQLSTATE 42815) が、最小値よりも大きいかまたは等しくなければなりません (SQLSTATE 42815)。

NO CYCLE または **CYCLE**
その最大値または最小値が生成された後、この IDENTITY 列が値の生成を続行するかどうかを指定します。

NO CYCLE
最大値または最小値に達した後、IDENTITY 列について値が生成されないことを指定します。これはデフォルトです。

CYCLE
最大値または最小値に達した後、この列について値の生成が続行されることを指定します。このオプションが使用されると、昇順 IDENTITY 列が最大値に達した後は、その最小値が生成されます。降順 IDENTITY 列が最小値に達した後は、その最大値が生成されます。 IDENTITY 列の最大値および最小値は、循環に使用される範囲を決定します。

CYCLE が有効な場合、DB2 が IDENTITY 列について重複する値を生成する可能性があります。ユニーク値が望ましい

場合、明示的には要求されませんが、値のユニーク性を確保するために、ユニークな 1 列の索引を生成列で定義する必要があります。このような IDENTITY 列にユニーク索引が存在し、ユニークではない値が生成されると、エラーが起こります (SQLSTATE 23505)。

NO CACHE または **CACHE**
特定の事前割り振り値を、高速アクセスできるようメモリーに保存するかどうかを指定します。 IDENTITY 列で新しい値が必要になった場合に、キャッシュの中のものを使用できないときは、新しいキャッシュ・ブロックの末尾をログ記録する必要があります。ただし、IDENTITY 列で新しい値が必要になった場合に、キャッシュの中に未使用の値があるときは、その識別値を割り振ったほうが、ロギングしなくて済むので高速化されます。これはパフォーマンスおよびチューニング・オプションです。

NO CACHE
IDENTITY 列の値を事前割り振りしないことを指定します。

このオプションが指定されると、IDENTITY 列の値はキャッシュに保管されません。この場合、新しい ID 値が要求されるたびに、ログに対して非同期入出力が行われます。

CACHE *integer-constant*
事前割り振りされ、メモリーに保管される IDENTITY シーケンスの値の数を指定します。 IDENTITY 列について値が生成される場合、値を事前割り振りしてキャッシュに保管しておくと、ログへの非同期入出力が少なくなります。

IDENTITY 列に新しい値が必要でも未使用の値がキャッシュにない場合、値の割り振りによりログへの入出力の待機が呼び出されます。ただし、IDENTITY 列に新しい値が必要で、未使用の値がキャッシュにあれば、その IDENTITY 値の割り振りが、ログへの入出力なしで素早く行われます。

システム障害に起因するものであっても通常のものであっても、データベースの活動解除が起こると、コミットされたステートメントで使用されていないキャッシュ済みシーケンス値はすべて失われます。つまり使用されなくなります。データベースの活動解除が起きたら失われる可能性のある IDENTITY 列値の最大数は、 CACHE オプションに指定された値になります。 (データベースが ACTIVATE コマンドまたは API を使用して明示的に活動化されない場合には、最終アプリケーションの接続をデータベースから切断すると、暗黙の活動解除が行われます。)

最小値は 2 です (SQLSTATE 42815)。デフォルト値は CACHE 20 です。

NO ORDER または **ORDER**
要求の順序で ID 値が生成されるかどうかを指定します。

NO ORDER
要求の順序での値を生成する必要がないことを指定します。これはデフォルトです。

ORDER
要求の順序で値が生成されることを指定します。

GENERATED ALWAYS AS (*generation-expression***)**
列定義が式に基づくことを指定します。(GENERATED ALWAYS 列の式にユーザー定義の外部関数が入っている場合に、その関数の実行可能ファイルを変更する (引き数ごとに異なる結果を得るため) と、データの不整合を生じることがあります。 これが生じないようにするには、SET INTEGRITY ステートメントを使って、新しい値を強制的に生成させます。) *generation-expression* には、以下のいずれも入れることができません (SQLSTATE 42621)。

- 副照会
- 列関数
- 逆参照操作または DEREF 関数
- 非 deterministic であるユーザー定義関数または組み込み関数
- EXTERNAL ACTION オプションを使用するユーザー定義関数
- CONTAINS SQL または READS SQL DATA のいずれかによって定義されたユーザー定義関数
- ホスト変数またはパラメーター・マーカー
- 特殊レジスター
- 列リスト内で後から定義されている列の参照
- 他の生成列の参照

列のデータ・タイプは *generation-expression* の結果データ・タイプに基づいています。 CAST 指定を使って特定のデータ・タイプを強制的に使用し、有効範囲を設けることができます (参照タイプの場合のみ)。 *data-type* を指定すると、適切な割り当て規則に従って、値が列に割り当てられます。 NOT NULL 列オプションを使わない限り、生成された列は暗黙で NULL 可能と見なされます。生成される列のデータ・タイプは、同等性を定義されているものでなければなりません。ただし、LONG VARCHAR、LONG VARGRAPHIC、または DATALINK の各タイプ、 LOB データ・タイプ、構造化タイプ、およびこれらのいずれかのタイプに基づいた特殊タイプの列を除きます (SQLSTATE 42962)。

OID-column-definition
型付き表のオブジェクト ID 列を定義します。

REF IS *OID-column-name* **USER GENERATED**
オブジェクト ID 列 (OID) を表の最初の列として定義することを指定します。表階層のルート表では、OID が必須です (SQLSTATE 428DX)。この表は副表以外の型付き表 (OF 文節が必須) でなければなりません (SQLSTATE 42613)。この列の名前は *OID-column-name* という形式で定義されますが、構造タイプ *type-name1* のどの属性の名前とも同一にすることはできません

(SQLSTATE 42711)。さらに、この列はタイプ REF (*type-name1*), NOT NULL で定義され、システム必須のユニーク索引 (デフォルトの索引名) が生成されます。この列はオブジェクト *ID 列* または *OID 列* として参照されます。 USER GENERATED というキーワードは、行を挿入する際にユーザーが OID 列の初期値を提供しなければならないことを指しています。行を挿入した後は、OID 列を更新することはできません (SQLSTATE 42808)。

with-options
型付き表の列に適用される追加オプションを定義します。

column-name
追加オプションを指定する列の名前を指定します。 *column-name* (列名) は、同じくスーパー表の列ではない表の列の名前に対応していなければなりません (SQLSTATE 428DJ)。列名は、ステートメント内の 1 つの WITH OPTIONS 文節に 1 回だけしか指定できません (SQLSTATE 42613)。

タイプ定義 (CREATE TYPE) の一部としてオプションが既に指定されている場合には、ここで指定されているオプションは CREATE TYPE のオプションをオーバーライドします。

WITH OPTIONS *column-options*
指定した列にオプションを定義します。前述の *column-options* を参照してください。表が副表である場合、主キーまたはユニーク制約を指定することはできません (SQLSTATE 429B3)。

DATA CAPTURE
データベース間のデータのレプリケーションに関する余分な情報を、ログに書き込むかどうかを指定します。この文節は、副表を作成する際には指定できません (SQLSTATE 42613)。

表が型付き表である場合、このオプションはサポートされません (SQLSTATE 428DH または 42HDR)。

NONE
追加情報をログに記録しないことを指定します。

CHANGES
この表に対する SQL 変更についての追加情報をログに書き込むことを指定します。このオプションは、表を複製する場合で、キャプチャー・プログラムを使用してログからこの表に対する変更内容をキャプチャーする場合に必須です。

カタログ・パーティション以外のパーティションにデータが置かれるように表が定義されている場合 (複数パーティションのデータベース・パーティション・グループ、またはカタログ・パーティション以外のパーティションを持つデータベース・パーティション・グループ)、このオプションはサポートされません (SQLSTATE 42997)。

表のスキーマ名 (暗黙または明示名) が 18 バイトより長い場合、このオプションはサポートされません (SQLSTATE 42997)。

WITH RESTRICT ON DROP
表をドロップできないこと、また、表を含む表スペースをドロップできないことを指定します。

IN *tablespace-name1*
表を作成するための表スペースを指定します。その表スペースは存在していなければならず、ステートメントの許可 ID がもつ USE 特権の対象の REGULAR 表スペースでなければなりません。他の表スペースが指定されていない場合、表のすべての部分がこの表スペースに保管されます。副表は表階層のルート表から表スペースを継承するので、この文節を作成の際に指定することはできません (SQLSTATE 42613)。この文節を指定しない場合には、この表の表スペースは次のように決められます。

```
IF ユーザーが USE 特権を持つ表スペース IBMDEFAULTGROUP が
   十分なページ・サイズをもって存在する場合
    THEN それを選択します
ELSE IF ユーザーが USE 特権を持つ表スペースが
  十分なページ・サイズをもって存在する場合
  (複数表スペース修飾の場合は下記を参照)
    THEN それを選択します
 ELSE エラーを出します (SQLSTATE 42727)
```

ELSE IF 条件で複数の表スペースが指定されている場合、ステートメントの許可 ID がもつ USE 特権の対象の最小限必要なページ・サイズをもつ表スペースを選択します。複数の表スペースがそれにあてはまる場合、以下のどれに USE 特権が付与されているかに応じて優先順位が決められます。

1. 許可 ID
2. 許可 ID を保有するグループ
3. PUBLIC

それでも複数の表スペースがそれにあてはまる場合は、最終選択はデータベース・マネージャーによって行われます。

表スペースの決定は、以下の時点で変更することができます。

- 表スペースをドロップまたは作成するとき
- USE 特権を付与または取り消すとき

十分な表のページ・サイズは、行のバイト・カウントか列の数のいずれかによって決まります。詳細については、398 ページの『行サイズ』を参照してください。

tablespace-options
索引または長形式列の値 (あるいはその両方) が保管される表スペースを指定します。表スペースのタイプについては、『CREATE TABLESPACE』 を参照してください。

INDEX IN *tablespace-name2*
表に索引を作成するための表スペースを指定します。このオプションは、IN 文節に指定された PRIMARY 表スペースがデータベース管理表スペース (DMS) である場合にのみ使用できます。指定する表スペースは、既存であり、ステートメントの許可 ID がもつ USE 特権が対象とする REGULAR または LARGE DMS 表スペースで

あり、 *tablespace-name1* と同じデータベース・パーティション・グループになければなりません (SQLSTATE 42838)。

表の索引を入れる表スペースの指定は、その表の作成時にのみ行うことができる点に注意してください。索引用の表スペースに対する USE 特権の検査は、表の作成時にしか行われません。その後の索引の作成時に CREATE INDEX ステートメントの許可 ID が、表スペースに対する USE 特権をもつことをデータベース・マネージャーによって要求されることはありません。

LONG IN *tablespace-name3*
長形式列 (LONG VARCHAR、LONG VARGRAPHIC、LOB データ・タイプ、これらのいずれかをソース・タイプとする特殊タイプ、またはインラインで保管できない値をもつユーザー定義の構造タイプで定義されたすべての列) の値が保管される表スペースを指定します。このオプションは、IN 文節に指定された PRIMARY 表スペースがデータベース管理表スペース (DMS) である場合にのみ使用できます。表スペースは、既存であり、ステートメントの許可 ID がもつ USE 特権が対象とする LARGE DMS 表スペースであり、 *tablespace-name1* と同じデータベース・パーティション・グループになければなりません (SQLSTATE 42838)。

表の長形式列と LOB 列を入れる表スペースの指定は、その表の作成時にのみ行うことができる点に注意してください。長形式列と LOB 列用の表スペースに対する USE 特権の検査は、表の作成時にしか行われません。その後の長形式列と LOB 列の追加時に ALTER TABLE ステートメントの許可 ID が、表スペースに対する USE 特権をもつことがデータベース・マネージャーで要求されることはありません。

PARTITIONING KEY (*column-name***,...)**
表のデータがパーティション化されている場合に、パーティション・キーを指定します。各 *column-name* (列名) は、表の列を指定するものでなければなりません。また、同じ列を複数回指定することはできません。 LONG VARCHAR、 LONG VARGRAPHIC、 BLOB、 CLOB、 DBCLOB、 DATALINK、これらのタイプのいずれかに基づいた特殊タイプ、または構造タイプであるデータ・タイプの列を、パーティション・キーの一部として使用することはできません (SQLSTATE 42962)。副表では表階層のルート表からパーティション・キーが継承されるため、パーティション・キーを指定することはできません (SQLSTATE 42613)。

この文節の指定がなく、この表が複数パーティションのデータベース・パーティション・グループに存在する場合、そのパーティション・キーは次のように定義されます。

- 表が型付き表である場合、オブジェクト ID 列がパーティション・キーになります。
- 主キーが指定されている場合は、その主キーの最初の列がパーティション・キーになります。
- これら以外の場合、 LOB、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK 列、これらのタイプのうちのいずれか

に基づいた特殊タイプ、または構造タイプの列以外のデータ・タイプの最初の列がパーティション・キーになります。

デフォルトのパーティション・キーの要件を満たす列が存在しない場合、表はパーティション・キーなしで作成されます。このような表は、単一パーティションのデータベース・パーティション・グループに対して定義された表スペースでのみ許されます。

単一パーティションのデータベース・パーティション・グループに対して定義された表スペースの表の場合、長形式以外の一連の列はいずれもパーティション・キーの定義に使用することができます。このパラメーターの指定がない場合、パーティション・キーは作成されません。

パーティション・キーに関する制約事項については、393 を参照してください。

USING HASHING
データ分散のパーティション化方式として、ハッシュ関数を使用することを指定します。これは、サポートされる唯一のパーティション化方式です。

REPLICATED
表が定義される表スペースのデータベース・パーティション・グループの各データベース・パーティションに対して、表に保管されたデータを物理的に複製することを指定します。つまり、これらのデータベース・パーティションにはそれぞれ、表のデータすべてのコピーが存在することになります。このオプションは、マテリアライズ照会表にのみ指定できます (SQLSTATE 42997)。

VALUE COMPRESSION
ほとんどのデータ・タイプで NULL および長さ 0 のデータ値を、さらに効率的に保管するよう指定します。また、使用される行形式を判別します。表が型付き表である場合、このオプションがサポートされるのは、型付き表階層のルート表でだけです (SQLSTATE 428DR)。

データ・タイプが BLOB、CLOB、DBCLOB、LONG VARCHAR、または LONG VARGRAPHIC の列の長さ 0 のデータ値を、最小限のスペースを使用して保管します。 NULL 値は、追加バイトを使用することなく、それぞれ保管されます。これをサポートする行形式は、各データ・タイプのバイト・カウントを識別し、更新の際にデータ・フラグメントの原因となる傾向があります。新しい行形式 (COMPRESS SYSTEM DEFAULT オプションによって列を指定する) を使用しても、列のシステム・デフォルト値をより効率的に保管できます。

NOT LOGGED INITIALLY
表を作成する作業単位と同一の作業単位の挿入、削除、更新、索引の作成、索引のドロップ、 または表の変更操作によって表に対して行われた変更はログ記録されません。このオプションを使用する際の他の考慮事項については、このステートメントの『注』のセクションを参照してください。

カタログの変更と、ストレージに関連する情報は、以後の作業単位で表に対して行われた操作と同様にすべてログ記録されます。

注: 活動化された NOT LOGGED INITIALLY 属性を持つ表に対してログに記録されない活動が生じ、ステートメントに障害が起こる (ロールバックが発生する)、または ROLLBACK TO SAVEPOINT が実行される場合には、その作業単位全体がロールバックされます (SQL1476N)。さらに、NOT LOGGED INITIALLY 属性が活動化されている表は、ロールバックされた後にアクセス不能としてマークされ、ドロップしかできなくなります。したがって、NOT LOGGED INITIALLY 属性が活動化されている作業単位内のエラーは、最小限に抑えるべきです。

CCSID

表に格納されるストリング・データのコード化・スキームを指定します。CCSID 文節を指定しない場合のデフォルトは、Unicode データベースでは CCSID UNICODE、他のすべてのデータベースでは CCSID ASCII になります。

ASCII

ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、CCSID ASCII を指定することはできません (SQLSTATE 56031)。

UNICODE

ストリング・データが Unicode でエンコードされることを指定します。データベースが Unicode データベースの場合、文字データは UTF-8、GRAPHIC データは UCS-2 になります。データベースが Unicode データベースでない場合は、文字データは UTF-8 になります。

データベースが Unicode データベースでない場合、 CCSID UNICODE を指定して表を作成できますが、以下の規則が適用されます。

- 表を作成するより前に、代替照合シーケンスをデータベース構成に指定する必要があります (SQLSTATE 56031)。 CCSID UNICODE 表は、データベース構成に指定されている代替照合シーケンスと照合されます。
- CCSID ASCII を指定して作成された表または表関数と、 CCSID UNICODE を指定して作成された表または表関数とを、 1 つの SQL ステートメント内で両方とも使用することはできません (SQLSTATE 53090)。このことは、ステートメント内で直接参照されている表および表関数、および間接的に (たとえば、参照保全制約、トリガー、マテリアライズ照会表、およびビューの本体内の表によって) 参照されている表および表関数に適用されます。
- CCSID UNICODE を指定して作成された表は、 SQL 関数または SQL メソッド内では参照できません (SQLSTATE 560C0)。
- CCSID UNICODE を指定して作成された表を参照する SQL ステートメントは、 SQL 関数または SQL メソッドを呼び出すことができません (SQLSTATE 53090)。
- GRAPHIC タイプおよびユーザー定義タイプは CCSID UNICODE 表内では使用できません (SQLSTATE 560C1)。
- 同じ表に CCSID UNICODE 文節と DATA CAPTURE CHANGES 文節の両方を指定することはできません (SQLSTATE 42613)。
- Explain 表は CCSID UNICODE では作成できません (SQLSTATE 55002)。

- 宣言済みグローバル一時表は CCSID UNICODE では作成できません (SQLSTATE 56031)。
- CCSID UNICODE 表は CREATE SCHEMA ステートメントでは作成できません (SQLSTATE 53090)。
- ロード操作の例外表の CCSID は、この操作のターゲット表と同じでなければなりません (SQLSTATE 428A5)。
- SET INTEGRITY ステートメントの例外表の CCSID は、このステートメントのターゲット表と同じでなければなりません (SQLSTATE 53090)。
- イベント・モニター・データのターゲット表は CCSID UNICODE として宣言されていてはなりません (SQLSTATE 55049)。
- CCSID UNICODE 表を参照するステートメントは、 DB2 バージョン 8.1 以降のクライアントからのみ呼び出すことができます (SQLSTATE 42997)。
- SQL ステートメントは常にデータベース・コード・ページで解釈されます。特にこのことは、リテラル、16 進数リテラル、および区切り ID 内のすべての文字がデータベース・コード・ページで表記されていなければならないということを意味します。そうでないと、文字は置換文字によって置き換えられてしまいます。

呼び出される SQL ステートメント内の表の CCSID に関係なく、アプリケーション内のホスト変数は常にアプリケーション・コードで表記されます。 DB2 は、アプリケーション・コード・ページとセクション・コード・ページ間でのデータ変換の必要に応じて、コード・ページ変換を実行します。レジストリー変数 DB2CODEPAGE をクライアント側で設定して、アプリケーション・コード・ページを変更することができます。

OPTIONS (ADD *table-option-name string-constant*, **...)**
表オプションは、リモート基本表を識別するために使用します。 *table-option-name* はオプションの名前です。 *string-constant* は、表オプションの設定を指定します。 *string-constant* は単一引用符で囲む必要があります。

リモート・サーバー (CREATE SERVER ステートメントに指定されたサーバー名) は、 OPTIONS 文節に指定します。 OPTIONS 文節を使用して、作成中のリモート基本表のスキーマまたは被修飾名をオーバーライドすることもできます。

スキーマ名を指定することが推奨されています。リモート・スキーマ名が指定されていない場合、表名の修飾子が使用されます。表名に修飾子がない場合、ステートメントの許可 ID が使用されます。

リモート基本表の被修飾名が指定されていない場合、 *table-name* が使用されます。

unique-constraint
ユニーク制約または主キー制約を定義します。表にパーティション・キーがある場合、ユニーク・キーまたは主キーはパーティション・キーのスーパーセットである必要があります。副表である表では、ユニーク制約または主キー制約を指定することはできません (SQLSTATE 429B3)。主キーまたはユニーク・キーは、ディメンションのサブセットにはなりません (SQLSTATE 429BE)。表がルート表である場合、表とそのすべての副表に対して制約が適用されます。

CONSTRAINT *constraint-name*
主キー制約、またはユニーク制約の名前を指定します。

UNIQUE (*column-name***,...)**
指定した列で構成されるユニーク・キーを定義します。指定する列は NOT NULL として定義されていなければなりません。各 *column-name* (列名) は、表の列を指定するものでなければなりません。また、同じ列を複数回指定することはできません。

指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。 変数キー部分をもつユニーク・キーは、DB2_INDEX_2BYTEVARLEN レジストリー変数が ON になっている場合には、 255 より大きいサイズが可能です。列の長さ属性が 1024 バイト以内に収まる場合でも、 LOB、 LONG VARCHAR、 LONG VARGRAPHIC、DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、ユニーク・キーの一部として使用できません (SQLSTATE 54008)。

ユニーク・キーにある一連の列は、主キーまたは他のユニーク・キーの一連の列と同じにすることはできません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。

表が副表である場合、ユニーク制約はスーパー表から継承されるので、(SQLSTATE 429B3)、ユニーク制約を指定することはできません。

カタログに記録されている表の記述には、ユニーク・キーとそのユニーク索引が含まれます。ユニーク索引は、それぞれの列について昇順に指定された順序で、列に対して自動的に作成されます。索引の名前は、表の作成時にスキーマに存在する既存の索引と競合しない場合、 *constraint-name* (制約名) と同じになります。索引名が競合する場合は、名前は SQL の後に文字のタイム・スタンプ (*yymmddhhmmssxxx*) が続き、スキーマ名として SYSIBM を伴う名前になります。

PRIMARY KEY (*column-name***,...)**
指定された列で構成される主キーを定義します。この文節を複数回指定することはできず、指定する列は NOT NULL として定義されていなければなりません。各 *column-name* (列名) は、表の列を指定していなければなりません。また、同じ列を複数回指定することはできません。

指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。 変数キー部分をもつ主キーは、DB2_INDEX_2BYTEVARLEN レジストリー変数が ON になっている場合には、 255 より大きいサイズが可能です。列の長さ属性が 1024 バイト以内に収まる場合でも、 LOB、 LONG VARCHAR、 LONG VARGRAPHIC、DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプは、主キーの一部として使用できません (SQLSTATE 54008)。

主キーの一連の列は、ユニーク・キーの一連の列と同じであってはなりません (SQLSTATE 01543)。 LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。

1 つの表には、主キーを 1 つだけ定義することができます。

表が副表である場合、主キーはスーパー表から継承されるので (SQLSTATE 429B3)、主キーを指定することはできません。

カタログに記録されている表の記述には、主キーとその主索引が含まれます。ユニーク索引は、それぞれの列について昇順に指定された順序で、列に対して自動的に作成されます。索引の名前は、表の作成時にスキーマに存在する既存の索引と競合しない場合、 *constraint-name* (制約名) と同じになります。索引名が競合する場合は、名前は SQL の後に文字のタイム・スタンプ (*yymmddhhmmssxxx*) が続き、スキーマ名として SYSIBM を伴う名前になります。

表にパーティション・キーがある場合、 *unique-constraint* (ユニーク制約) の列はパーティション・キーの列のスーパーセットである必要があります。列の順序は重要ではありません。

referential-constraint
参照制約を定義します。

CONSTRAINT *constraint-name*
参照制約の名前を指定します。

FOREIGN KEY (*column-name***,...)**
指定した *constraint-name* (制約名) の参照制約を定義します。

T1 を、ステートメントの対象となる表であると想定します。参照制約の外部キーは、指定された列で構成されます。列名リストの各名前は、T1 の列を指定していなければならず、同じ列を複数回指定することはできません。指定する列の数は 16 を超えてはならず、保管されるそれらの長さの合計は 1024 を超えてはなりません (保管される長さの詳細は、 398 ページの『バイト・カウント』を参照)。外部キーは、255 バイトよりも大きい長さの可変長列で定義できます。 LOB、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプの列を、外部キーの一部として使用することはできません (SQLSTATE 42962)。外部キーの列の数は、親キーの列の数と同じでなければならず、対応する列のデータ・タイプは互換性があることが必要です (SQLSTATE 42830)。 2 つの列の記述は、それらの列が互換性のあるデータ・タイプ (両方の列が数字、文字ストリング、GRAPHIC、日付 / 時間であるか、または同じ特殊タイプ) であれば互換性があります。

references-clause
参照制約の親表または親ニックネーム、および親キーを指定します。

REFERENCES *table-name* または *nickname*
REFERENCE 文節に指定される表またはニックネームは、カタログに記述された基本表またはニックネームを識別している必要がありますが、カタログ表を示すものであってはなりません。

参照制約の外部キー、親キー、および親表または親ニックネームが、以前に指定した参照制約の外部キー、親キー、および親表または親ニックネームと同じである場合、参照制約は重複しています。重複した参照制約は無視され、警告が戻されます (SQLSTATE 01543)。

以下の説明では、T2 は指定した親表を示し、T1 は作成する (または変更する) 表を示します。 (T1 と T2 は同じ表である可能性もあります。)

指定された外部キーの列の数は、T2 の親キーと同じ数でなければなりません。また、外部キーの *n* 番目の列の記述は、その親キーの *n* 番目の列の記述と互換性がなければなりません。この規則において、日時の列はストリング列と互換性があるとは見なされません。

(*column-name***,...)**
参照制約の親キーは、指定された列で構成されます。各 *column-name* は、T2 の列を指定する非修飾名でなければなりません。同じ列を重複して指定することはできません。

列名のリストは、主キーまたは T2 に存在するユニーク制約の一連の列と一致している (順序は任意) 必要があります (SQLSTATE 42890)。列名のリストの指定がない場合、T2 に主キーがある必要があります (SQLSTATE 42888)。列名リストを省略すると、指定されているとおりの順序でその主キーの列が暗黙に指定されます。

FOREIGN KEY 文節で指定される参照制約は、 T2 が親であり、T1 が従属である関係を定義します。

rule-clause
従属表に対するアクションを指定します。

ON DELETE
親表の行が削除された場合、従属表でどのようなアクションを行うかを指定します。次の 4 つのアクションがあります。

- NO ACTION (デフォルト値)
- RESTRICT
- CASCADE
- SET NULL

削除規則は、T2 の行が DELETE または伝搬による削除操作の対象であり、その行の従属行が T1 にある場合に、適用されます。 *p* は、そのような T2 の行を表すと想定します。

- RESTRICT または NO ACTION を指定すると、エラーになり、行は削除されません。
- CASCADE を指定すると、 T1 の *p* の従属行に削除操作が伝搬します。
- SET NULL が指定された場合、 T1 の *p* のそれぞれの従属行の外部キーの NULL 可能な列が NULL 値に設定されます。

SET NULL は、外部キーの列に NULL 可能な列がない限り指定してはなりません。この文節を省略すると、暗黙に ON DELETE NO ACTION が指定されます。

T1 が複数のパスで T2 に連結削除されている場合は、重複する外部キー定義を使用して 2 つの SET NULL 規則を定義することはで

きません。たとえば、T1 (i1, i2, i3) という場合があります。 Rule1 に外部キー (i1, i2) を使用し、Rule2 に外部キー (i2, i3) を使用するということはできません。

規則の配列順序は次のとおりです。

1. RESTRICT
2. SET NULL OR CASCADE
3. NO ACTION

T1 の任意の行が 2 つの異なる規則によって影響される場合、エラーとなり行は削除されません。

複数の表が関係し、削除規則の 1 つが RESTRICT または SET NULL になっている循環によって表が自身を連結削除するような参照制約は定義できません (SQLSTATE 42915)。

複数のパスによって表が自身や他の表を連結削除する参照制約は、以下の場合を除き、定義できます (SQLSTATE 42915)。

- 表は、CASCADE 関係 (自己参照、または別の表を参照)、および削除規則が RESTRICT または SET NULL の自己参照関係のいずれの従属表であってもなりません。
- キーとキーの間に同じ列が 1 つでもあると、キーはもう 1 方のキーにオーバーラップします。表が、外部キーにオーバーラップしている複数の関係を通して別の表に連結削除される場合は、それらの関係の間で削除規則が一致していなければなりません。また、いずれの削除規則も SET NULL になっていてはなりません。
- 表が複数の関係を通して別の表に連結削除される場合で、それらの関係のうち、少なくとも 1 つに SET NULL の削除規則が指定されている場合、これらの関係の外部キー定義には、パーティション・キーまたは MDC キー列が含まれていてはなりません。
- CASCADE 関係を通して 2 つの表が同じ表に連結削除されている場合、各連結削除パスの最後の関係の削除規則が RESTRICT または SET NULL であるときは、 2 つの表を相互に連結削除することはできません。

T1 の何からの行が別の削除規則の影響を受ける場合、結果は、これらの規則で指定されたすべてのアクションの影響を受けます。すべてのアクションの影響は、T1 の AFTER トリガーと CHECK 制約からも認識されます。たとえば、例として、上位の表へのある連結削除パスによってヌルに設定されるターゲットになっていて、同じ上位の表への 2 番目の連結削除パスによって削除されるターゲットになっている行があるとします。この場合、結果として行は削除されます。この派生表では、AFTER DELETE トリガーは活動化されますが、 AFTER UPDATE トリガーは活動化されません。

親表または従属表が型付き表階層のメンバーである参照制約に対する上記の規則の適用の場合、それぞれの階層内の任意の表に対して適用されるすべての参照制約が考慮に入れられます。

ON UPDATE
親表の行が更新された場合に従属表に対して行うアクションを指定します。この文節はオプションです。 ON UPDATE NO ACTION はデフォルト値であり、 ON UPDATE RESTRICT はそれに代わって指定できる唯一のものです。

NO ACTION と RESTRICT の差異については、このステートメントの『注』のセクションを参照してください。

constraint-attributes
参照保全またはチェック制約に関連付けられた属性を定義します。

ENFORCED または **NOT ENFORCED**
挿入、更新、削除などの通常の操作中に、データベース・マネージャーによって制約が課せられるかどうかを指定します。デフォルトは ENFORCED です。

ENFORCED
データベース・マネージャーによって制約が課せられます。 機能従属関係に ENFORCED を指定することはできません (SQLSTATE 42621)。 ENFORCED は、参照制約がニックネームを参照しているときは指定できません (SQLSTATE 428G7)。

NOT ENFORCED
データベース・マネージャーによって制約が課せられません。この制約に適合することが分かっている表データだけに、制約を指定してください。

ENABLE QUERY OPTIMIZATION または **DISABLE QUERY OPTIMIZATION**
適切な状況下で、照会の最適化のために、制約または機能従属関係を使用できるかどうかを指定します。デフォルトは ENABLE QUERY OPTIMIZATION です。

ENABLE QUERY OPTIMIZATION
制約が真であると想定され、照会の最適化のために使用できます。

DISABLE QUERY OPTIMIZATION
制約を照会の最適化に使用できません。

check-constraint
チェック制約を定義します。 *check-constraint (チェック制約)* は、偽以外に評価されなければならない *search-condition (検索条件)*、 または列間に定義された機能従属関係です。

CONSTRAINT *constraint-name*
チェック制約の名前を指定します。

CHECK (*check-condition***)**
チェック制約を定義します。 *search-condition* は、表のすべての行について、真または不明でなければなりません。

search-condition
search-condition には、以下の制限があります。

- 列参照は、作成する表の列に対するものでなければなりません。

- *search-condition* に TYPE 述部を入れることはできません。
- *search-condition* には、以下のいずれも入れることができません (SQLSTATE 42621)。
  - 副照会
  - 逆参照操作または、有効範囲をもつ参照引き数がオブジェクト ID (OID) 列以外の列である DEREF 関数
  - SCOPE 文節をもつ CAST 指定
  - 列関数
  - deterministic 関数でない関数
  - 外部アクションをもつと定義された関数
  - CONTAINS SQL または READS SQL DATA のいずれかによって定義されたユーザー定義関数
  - ホスト変数
  - パラメーター・マーカー
  - 特殊レジスター
  - ID 列以外の生成列の参照

functional-dependency
列間の機能従属関係を定義します。

column-name **DETERMINED BY** *column-name* または **(***column-name***,...) DETERMINED BY (***column-name***,...)**
列の親セットには、DETERMINED BY 文節の直前に来る指定された列が含まれます。列の子セットには、DETERMINED BY 文節の直後に来る指定された列が含まれます。 *search-condition* の制約事項すべては、親セット列と子セット列に適用され、列のセットには単純な列参照のみが許可されています (SQLSTATE 42621)。機能従属関係には同じ列を 2 度以上指定することはできません (SQLSTATE 42709)。列のデータ・タイプを LOB データ・タイプ、LOB データ・タイプに基づいた特殊タイプ、または構造化型にすることはできません (SQLSTATE 42962)。列の子セットの列を NULL 可能列にすることはできません (SQLSTATE 42621)。

column-definition の一部としてチェック制約を指定する場合、その同じ列に対してのみ列参照を行うことができます。表定義の一部として指定されたチェック制約には、それ以前に CREATE TABLE ステートメントで定義されている列を指定する列参照を含めることができます。チェック制約の矛盾、重複条件、または同等条件については検査されません。したがって、矛盾したチェック制約や冗長なチェック制約が定義可能であるため、実行時にエラーになる可能性があります。

search-condition として 『IS NOT NULL』 も指定できますが、列の NOT NULL 属性を使用することによって直接的に NULL 値可を指定するようにしてください。たとえば、salary が NULL に設定された場合に、 `CHECK (salary + bonus > 30000)` は受け入れられます。これは、CHECK 制約は満たされるか未知かのどちらかでなければならず、この場合 salary は未知

であるためです。一方、給与 (salary) が NULL に設定されている場合に、CHECK (salary IS NOT NULL) は偽となり、制約違反と見なされます。

search-condition を伴うチェック制約は、表に対して行の挿入または更新が行われる時点で適用されます。表で定義されるチェック制約は、その表の副表すべてに自動的に適用されます。

挿入、更新、削除、保全性設定などの通常の操作中には、データベース・マネージャーによって機能従属関係が課せられません。機能従属関係は、照会を最適化するために、照会の書き直しの際に使用できるでしょう。機能従属関係の保全性が維持されないと、間違った結果が戻される可能性があります。

規則:

- すべての構造タイプ列のインライン長さも含め、列のバイト・カウントの合計は、表スペースのページ・サイズに基づく行サイズの限界を超えてはなりません (SQLSTATE 54010)。詳しくは、 10 ページの『SQL 戻りコード』を参照してください。型付き表の場合、表階層のルート表の列や表階層内の各副表で新たに追加される列すべてに対しては、バイト・カウントが適用されます (追加される副表列は、 NULL 不可として定義されたとしても、バイト・カウントの際には NULL 可能と見なされます)。また、各行がどの副表からきたものかを識別するため、4 バイトのオーバーヘッドが追加されます。
- 表内に存在する列の数は、1,012 個を超えてはなりません (SQLSTATE 54011)。型付き表の場合は、表階層内のすべての副表タイプに含まれている属性の合計が 1,010 個を超えてはなりません。
- 型付き表のオブジェクト ID 列は更新できません (SQLSTATE 42808)。
- 表に対して定義されたユニーク・キー制約または主キー制約は、パーティション・キーのスーパーセットでなければなりません (SQLSTATE 42997)。
- 次の表には、*file-link-options* でサポートされている DATALINK オプションの組み合わせが示されています (SQLSTATE 42613)。 WRITE PERMISSION ADMIN は、READ PERMISSION DB とだけ組み合わせることが可能です。(RECOVERY と ON UNLINK 文節の他の組み合わせはサポートされます。)

表 5. 有効な DATALINK ファイル制御オプションの組み合わせ： この表にない組み合わせはすべてサポートされず、SQLSTATE 42613 となります。

INTEGRITY	READ PERMISSION	WRITE PERMISSION	RECOVERY	ON UNLINK
ALL	FS	FS	NO	該当なし
ALL	FS	BLOCKED	NO	RESTORE
ALL	FS	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	NO	RESTORE
ALL	DB	BLOCKED	NO	DELETE
ALL	DB	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	YES	DELETE
ALL	DB	ADMIN	NO	RESTORE
ALL	DB	ADMIN	NO	DELETE
ALL	DB	ADMIN	YES	RESTORE

表 5. 有効な DATALINK ファイル制御オプションの組み合わせ (続き): この表にない組み合わせはすべてサポートされず、SQLSTATE 42613 となります。

INTEGRITY	READ PERMISSION	WRITE PERMISSION	RECOVERY	ON UNLINK
ALL	DB	ADMIN	YES	DELETE

- 以下の規則は、パーティション・データベースに対してのみ適用されます。
  - LOB、LONG VARCHAR、LONG VARGRAPHIC、 DATALINK、これらのタイプのうちのいずれかに基づく特殊タイプ、または構造タイプの列だけで構成された表は、単一パーティションのデータベース・パーティション・グループで定義されている表スペース内でしか作成することができません。
  - 複数パーティションのデータベース・パーティション・グループに対して定義された、表スペースの表のパーティション・キー定義は変更できません。
  - 型付き表のパーティション・キー列は OID 列にする必要があります。
- 複数のデータベース・パーティションがあるデータベースでは、レンジ・クラスター表を指定できません (SQLSTATE 42997)。
- レンジ・クラスター表には、次のような制限が適用されます。
  - VALUE COMPRESSION を活動化できません。
  - クラスター索引は作成できません。
  - 列を追加するための表の変更はサポートされていません。
  - 列のデータ・タイプを変更するための表の変更はサポートされていません。
  - PCTFREE を変更するための表の変更はサポートされていません。
  - APPEND ON を設定するための表の変更はサポートされていません。
  - DETAILED 統計は使用できません。
  - 表の移植にロード・ユーティリティーを使用することはできません。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - CONSTRAINT キーワードは、参照文節を定義する *column-definition* から省略できます。
    - *constraint-name* (制約名) を FOREIGN KEY に続けて (CONSTRAINT キーワードなし) 指定することができます。
    - SUMMARY は CREATE の後に任意に指定できます。
    - WITH NO DATA の代わりに DEFINITION ONLY を指定できます。
  - 以前のバージョンの DB2 との互換性と整合性:
    - *identity-options* 文節では、コンマを使って複数のオプションを分離することができます。
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文はデフォルトの振る舞いとして受け入れられます。
      - IN database-name.tablespace-name
      - IN DATABASE database-name
      - FOR MIXED DATA

| • FOR SBCS DATA
- 以下の構文もサポートされています。
  • NOMINVALUE、NOMAXVALUE、NOCYCLE、NOCACHE、および NOORDER

- まだ存在していないスキーマ名を用いて表を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- 外部キーが指定されると、
  – 親表の削除を行うパッケージはすべて無効になります。
  – 親キーの少なくとも 1 つの列に対して更新使用の指定があるパッケージは、すべて無効になります。
- 副表を作成すると、表階層内の表のいずれかに従属しているパッケージがすべて無効になります。
- それぞれ 4,000 および 2,000 より大きい数値の VARCHAR および VARGRAPHIC 列は、 SYSFUN スキーマの関数での入力パラメーターとして使用しないでください。関数にこの長さを超過する引き数値を指定して呼び出すと、エラーが発生します (SQLSTATE 22001)。
- 参照制約の削除規則または更新規則として NO ACTION または RESTRICT を使用すると、制約がいつ適用されるかが決まります。 RESTRICT の削除規則または更新規則は、 CASCADE や SET NULL などの変更規則を伴う参照制約を含む | 他のすべての制約の*前に* 適用されます。 NO ACTION の削除規則または更新規 | 則は、他の参照制約の*後で* 適用されます。この動作の違いが明白になる例の 1 | つとして、互いに関連する複数の表の UNION ALL として定義されたビューから | の行の削除があります。

```
Table T1 is a parent of table T3; delete rule as noted below.
Table T2 is a parent of table T3; delete rule CASCADE.

CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2

DELETE FROM V1
```

表 T1 が RESTRICT の削除規則を伴う表 T3 の親である場合に、T3 に T1 の親キーの子行があると、制約違反 (SQLSTATE 23001) になります。

表 T1 が表 T3 の親で、T3 の削除規則が NO ACTION である場合、 T1 からの削除に対して NO ACTION 削除規則が適用される前に T2 から行を削除すると、削除規則 CASCADE によって、その子行が削除される場合があります。 T2 からの削除で、T3 の T1 の親キーの子行すべてが削除されたわけではない場合は、制約違反 (SQLSTATE 23504) になります。

戻される SQLSTATE は、削除規則または更新規則が RESTRICT か NO ACTION かによって異なります。

- 複数パーティションのデータベース・パーティション・グループに対して定義された表スペースの複数の表の場合、パーティション・キーを選択する際に表のコロケーションについて考慮する必要があります。以下に考慮事項をリストします。

  - コロケーションのためには、各表は同じデータベース・パーティション・グループにある必要があります。表スペースは異なっていても構いませんが、同じデータベース・パーティション・グループに定義されている必要があります。
  - コロケーションのためには、各表のパーティション・キーの列の数は同じである必要があり、対応するキーの列はパーティション互換である必要があります。
  - パーティション・キーの選択も、結合のパフォーマンスに影響します。表を他の表と頻繁に結合する場合は、結合する列を両方の表のパーティション・キーにすることを考慮してください。
- NOT LOGGED INITIALLY 文節は、表の中に、 FILE LINK CONTROL 属性のある DATALINK 列が存在するときには使用できません (SQLSTATE 42613)。
- 代替のソース (別の表やファイル) からのデータを使用して大きな結果セットを作成する必要があり、表のリカバリーが不要な場合は、NOT LOGGED INITIALLY オプションが有用です。このオプションを使用すると、データのロギングにかかるオーバーヘッドが節減されます。このオプションを指定する場合、以下の考慮事項が適用されます。
  - 作業単位がコミットされると、その作業単位の過程で表に対して行われた変更はすべてディスクにフラッシュされます。
  - ロールフォワード・ユーティリティーを実行した際に、データベース中の表がロード・ユーティリティーによって移植されたか、または NOT LOGGED INITIALLY オプションを使用して作成されたことを示すログ記録が見つかると、表は使用不能としてマークされます。 その後 DROP TABLE ログが見つかると、表はロールフォワード・ユーティリティーによってドロップされます。除去しない場合、データベースの回復後に表にアクセスを試みると、エラーが出されます (SQLSTATE 55019)。許される唯一の操作は表のドロップです。
  - データベースまたは表スペースのバックアップの一環として、このような表をバックアップすると、表のリカバリーが可能になります。
- CURRENT REFRESH AGE を ANY にセットし、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION をシステム保守済みマテリアライズ照会表に含まれるようにセットすると、照会の処理を最適化するときに、 ENABLE QUERY OPTIMIZATION を指定して定義された REFRESH DEFERRED システム保守済みマテリアライズ照会表を使用できます。 CURRENT REFRESH AGE を ANY にセットし、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION をユーザー保守済みマテリアライズ照会表に含まれるようにセットすると、照会の処理を最適化するときに、 ENABLE QUERY OPTIMIZATION を指定して定義された REFRESH DEFERRED ユーザー保守済みマテリアライズ照会表を使用できます。 ENABLE QUERY OPTIMIZATION を指定して定義された REFRESH IMMEDIATE マテリアライズ照会表は、必ず最適化の対象として考慮に入れられます。この最適化で REFRESH DEFERRED または REFRESH IMMEDIATE マテリアライズ照会表を使用できるようにするには、すでに説明された規則以外の特定の規則に全選択を適合させる必要があります。全選択の条件は、以下の規則に従っていなければなりません。
  - GROUP BY 文節を指定した副選択、または 1 つの表参照を指定した副選択になっている。
  - 選択リストのどこにも DISTINCT が含まれていない。

  - 特殊レジスターが含まれていない。
  - deterministic 関数でない関数が含まれていない。

  マテリアライズ照会表の作成時に指定した照会が上記の規則に適合しなければ、警告が戻されます (SQLSTATE 01633)。
- マテリアライズ照会表が REFRESH IMMEDIATE で定義されている場合、またはステージング表が PROPAGATE IMMEDIATE で定義されている場合は、基礎表の挿入、更新、または削除操作を行うことになる変更をしようとすると、エラーになる可能性があります。エラーが発生すると、基礎表の挿入、更新、または削除の操作は失敗します。
- ロード操作中や SET INTEGRITY ステートメントの実行中など、制約が大量にチェックされている場合、マテリアライズ照会表やステージング表は使用できません。
- REFRESH IMMEDIATE で定義されたマテリアライズ照会表、または REFRESH DEFERRED に関連ステージング表を指定して定義したマテリアライズ照会表によって参照されている表に対しては、次のような操作を実行することができません。
  - IMPORT REPLACE は使用できません。
  - ALTER TABLE NOT LOGGED INITIALLY WITH EMPTY TABLE は実行できません。
- フェデレーテッド・システムでは、リレーショナル・データ・ソースまたはローカル表のニックネームは、マテリアライズ照会表を作成する基礎表として使用できます。非リレーショナル・データ・ソースのニックネームは、サポートされていません。ニックネームが基礎表のうちの 1 つである場合は、REFRESH DEFERRED オプションを使用する必要があります。パーティション・データベース環境では、ニックネームを参照するシステム保守のマテリアライズ照会表はサポートされていません。
- ***透過性 DDL:*** フェデレーテッド・システムでは、DB2 UDB SQL を使用してリモート基本表を作成、変更、またはドロップすることができます。この機能は、*透過性 DDL* として知られています。リモート基本表がデータ・ソース上に作成される前に、フェデレーテッド・サーバーはそのデータ・ソースへアクセスするように構成されなければなりません。この構成には、データ・ソースのラッパーの作成、リモート基本表を置くサーバーのサーバー定義の指定、フェデレーテッド・サーバーとデータ・ソース間のユーザー・マッピングの作成が含まれます。

  CREATE TABLE ステートメントに含むことができるものに関して、透過性 DDL には以下のようないくつかの制約があります。
  - 列および主キーのみがリモート基本表に作成可能です。
  - リモート・データ・ソースは次のものをサポートする必要があります。
    - DB2 列データ・タイプがマップされるリモート列データ・タイプ
    - CREATE TABLE ステートメントにおける主キー・オプション

    データ・ソースの、サポートしていない要求への応答方法によって、エラーが返されるか、または要求が無視される可能性があります。

  リモート基本表が透過性 DDL を使用して作成された場合、そのリモート基本表に対してニックネームが自動的に作成されます。

- 親表または従属表が表階層の一部を成すように参照制約を定義することができます。その場合、参照制約は次のような効果を生じます。
  1. INSERT、UPDATE、および DELETE ステートメントの効果は次のとおりです。
     - PT が親表で DT が従属表である参照制約が存在する場合、非 NULL の外部キーをもつ DT の行 (またはその副表のいずれか) ごとに、それに合致する親キーをもつ行が PT (またはその副表のいずれか) 内に必ず存在することが制約によって確実になります。アクションの開始の仕方に関係なく、この規則は、 PT または DT の行に影響を与えるすべてのアクションに対して適用されます。
  2. DROP TABLE ステートメントの効果は次のとおりです。
     - ドロップ済みの表が親表または従属表である参照制約では、制約はドロップされます。
     - ドロップ済みの表のスーパー表が親表である参照制約では、そのドロップ済みの表の行は、スーパー表からの削除を考慮されます。参照制約が検査されて、削除行ごとに削除規則が呼び出されます。
     - ドロップ済みの表のスーパー表が従属表である参照制約の場合、制約は検査されません。従属表から行を削除しても、参照制約違反にはなりません。
- ***特権:*** 表が作成されると、その表の定義者には CONTROL 特権が付与されます。副表が作成されると、各ユーザーまたはグループが持っているそのすぐ上のスーパー表に対する SELECT 特権が副表に対しても自動的に付与され、その場合は表定義者から特権が付与されたことになります。
- ***行サイズ:*** 表の行で許可される最大バイト数は、表が作成される表スペースのページ・サイズによって決まります (*tablspace-name1*)。次のリストでは、各表スペースのページ・サイズに関連した行サイズの制限と列数の制限を示します。

表 6. 各表スペースのページ・サイズの列数および行サイズの制限

ページ・サイズ	行サイズの制限	列数の制限
4K	4 005	500
8K	8 101	1 012
16K	16 293	1 012
32K	32 677	1 012

表の実際の列数については、次の公式によってさらに制限されます。

```
Total Columns * 8 + Number of LOB Columns * 12 +
  Number of Datalink Columns * 28 <= row size limit for page size.
```

- ***バイト・カウント:*** 次の表は、 NULL 値を使用できない列のバイト・カウントを、列のデータ・タイプ別に示したものです。値の圧縮をしない表では、NULL が許される各列には追加のバイトが必要です。

  表が構造タイプに基づいている場合には、副表が定義されているか否かにかかわらず、副表の行を識別するために 4 バイトのオーバーヘッドが確保されます。さらに、追加される副表列は、NULL 不可と定義されたとしたら、バイト・カウント用に NULL 可能なものと見なされる必要があります。

保管長の計算の際に、索引または制約 (制約は索引によって実施されることに注意してください) で 1024 バイトの制限を超えていないこと、オーバーヘッドは 4 バイトではなく 2 バイトであることを確認してください。

表 7. データ・タイプごとの列のバイト・カウント

データ・タイプ	**VALUE COMPRESSION** が表に対して活動状態になるときのバイト・カウント	**VALUE COMPRESSION** が表に対して暗黙的または明示的に非活動状態であるときのバイト・カウント。列が **NULL** を受け入れる場合、示されたバイト・カウントに **1** が加算される
ROW OVERHEAD	2	0
INTEGER	6	4
SMALLINT	4	2
BIGINT	10	8
REAL	6	4
DOUBLE	10	8
DECIMAL	(*p*/2)+3 の整数部分 (*p* は精度)	(*p*/2)+1 の整数部分 (*p* は精度)
CHAR(*n*)	*n*+2	*n*
VARCHAR(*n*)	*n*+2	*n*+4 (表); *n*+2 (索引)
LONG VARCHAR	22	24
GRAPHIC(*n*)	*n**2+2	*n**2
VARGRAPHIC(*n*)	(*n**2)+2	(*n**2)+4 (表); (*n**2)+2 (索引)
LONG VARGRAPHIC	22	24
DATE	6	4
TIME	5	3
TIMESTAMP	12	10
DATALINK(*n*)	*n*+52	*n*+54
LOB の最大長 1024	70[1]	72
LOB の最大長 8192	94	96
LOB の最大長 65 536	118	120
LOB の最大長 524 000	142	144
LOB の最大長 4 190 000	166	168
LOB の最大長 134 000 000	198	200
LOB の最大長 536 000 000	222	224
LOB の最大長 1 070 000 000	254	256
LOB の最大長 1 470 000 000	278	280
LOB の最大長 2 147 483 647	314	316

表 7. データ・タイプごとの列のバイト・カウント (続き)

データ・タイプ	VALUE COMPRESSION が表に対して活動状態になるときのバイト・カウント	VALUE COMPRESSION が表に対して暗黙的または明示的に非活動状態であるときのバイト・カウント。列が NULL を受け入れる場合、示されたバイト・カウントに 1 が加算される

[1] 各 LOB 値は、その基底レコードに、実際の値の位置へのポインターとなる *LOB 記述子* を持っています。その記述子のサイズは、列に定義されている最大長によって異なります。

特殊タイプ の場合、バイト・カウントは特殊タイプのソース・タイプの長さに相当します。*参照タイプ* の場合には、バイト・カウントは、参照タイプの基礎となる組み込みデータ・タイプの長さに相当します。*構造タイプ* では、バイト・カウントは INLINE LENGTH + 4 です。 INLINE LENGTH は、 *column-options* 文節内の列に指定された (または暗黙で計算された) 値です。

- ***ディメンション列:*** ディメンション列の各特殊値は、表の別々のブロックに割り当てられるので、 ″INTEGER(ORDER_DATE)/100″ などの式でのクラスタリングはお勧めできません。この場合、表に生成された列が定義可能で、その後この生成された列は ORGANIZE BY DIMENSIONS 文節で使用されるかもしれません。式が表の列に関連して単調な場合には、この列の述部範囲を満たすためにディメンション索引が DB2 によって使用される可能性があります。たとえば、式が単に *column-name* + *some-positive-constant* の場合には、これは単調な増加です。ユーザー定義関数、特定の組み込み機能、および 1 つの式で複数の列を使用すると、単調化やその検出を防げます。

  非単調な式を持つ、または単調化を識別できない、生成された列に関係するディメンションを作成できますが、スライスの範囲照会やこうしたディメンションのセル境界はサポートされません。同等性および IN 述部は、スライスまたはセルによって処理できます。

  生成された関数 fn に関係して以下の事柄が真の場合には、生成された列は単調です。

  – 単調な増加。

    値 x1 および x2 のペアのすべての可能性において、x2>x1 ならば fn(x2)>fn(x1) となります。たとえば:

    ```
    SALARY - 10000
    ```

  – 単調な減少。

    値 x1 および x2 のペアのすべての可能性において、x2>x1 ならば fn(x2)<fn(x1) となります。たとえば:

    ```
    -SALARY
    ```

  – 単調な非減少。

    値 x1 および x2 のペアのすべての可能性において、x2>x1 ならば fn(x2)>=fn(x1) となります。たとえば:

    ```
    SALARY/1000
    ```

– 単調な非増加。

値 x1 および x2 のペアのすべての可能性において、x2>x1 ならば fn(x2)<=fn(x1) となります。たとえば:

```
-SALARY/1000
```

式 ″PRICE*DISCOUNT″ は単調ではありません。表の複数の列が関係するからです。

- ***レンジ・クラスター表:*** キー・シーケンスによる表の編成は、特定のタイプの表に対して効果があります。表は、有効な値の範囲上で高密度にクラスタリングされた整数キーを持っている必要があります。この整数キーの列は、NULL 可能であってはならず、キーは論理的に表の主キーでなければなりません。レンジ・クラスター表では、表の編成上、キー値によって指定された行や、キー値の範囲で指定された一定範囲の行に対する直接アクセスを提供する、別個のユニーク索引オブジェクトを必要としません。定義されたキー・シーケンスの範囲内にある行の完全セットに対するすべてのスペースの割り振りは、表の作成時に行われ、レンジ・クラスター表を定義する際に考慮される必要があります。最初の時点で行に削除のマークが付いていたとしても、ストレージ・スペースを他の用途に使用することはできません。キー・シーケンスの範囲全体が、長期間にわたってデータのみを追加するためのものである場合、この表編成は適切な選択ではありません。

例:

例 1: DEPARTX 表スペースに表 TDEPT を作成します。 DEPTNO、DEPTNAME、 MGRNO、および ADMRDEPT は列の名前です。 CHAR は、列が文字データを含むことを意味しています。 NOT NULL は、列に NULL 値を含めることができないことを示します。 VARCHAR は、列のデータが可変長文字データであることを意味します。主キーは、列 DEPTNO で構成されます。

```
CREATE TABLE TDEPT
  (DEPTNO   CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
  MGRNO CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   PRIMARY KEY(DEPTNO))
IN DEPARTX
```

例 2: SCHED 表スペースに表 PROJ を作成します。 PROJNO、 PROJNAME、DEPTNO、 RESPEMP、 PRSTAFF、 PRSTDATE、 PRENDATE、および MAJPROJ は列の名前です。 CHAR は、列が文字データを含むことを意味しています。DECIMAL は、列のデータがパック 10 進数データであることを意味します。 5,2 の 5 は 10 進数の桁数、2 は小数点以下の桁数を示します。 NOT NULL は、列に NULL 値を含めることができないことを示します。 VARCHAR は、列のデータが可変長文字データであることを意味します。 DATE は、列のデータが 3 つの部分からなる形式 (年、月、日) の日付情報であることを示しています。

```
CREATE TABLE PROJ
  (PROJNO   CHAR(6)      NOT NULL,
   PROJNAME VARCHAR(24)  NOT NULL,
   DEPTNO   CHAR(3)      NOT NULL,
   RESPEMP  CHAR(6)      NOT NULL,
   PRSTAFF  DECIMAL(5,2)         ,
```

```
  PRSTDATE DATE                ,
  PRENDATE DATE                ,
  MAJPROJ  CHAR(6)      NOT NULL)
 IN SCHED
```

例 3: 不明の給与はすべて 0 と見なされる EMPLOYEE_SALARY という名前の表を作成します。表スペースが指定されていないので、 *IN tablespace-name1* 文節について記述された規則に基づいてシステムが選択した表スペースに、表が作成されます。

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO   CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
   EMPNO    CHAR(6)      NOT NULL,
   SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT)
```

例 4: 給与 (SALARY) と距離 (MILES) の合計用の特殊タイプを作成し、デフォルト表スペースに作成される表の列として使用します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが JOHNDOE で、 CURRENT PATH がデフォルト値であると想定します (″SYSIBM″,″SYSFUN″,″JOHNDOE″)。

SALARY の値の指定がない場合には、それを 0 に設定します。また、
LIVING_DIST の値の指定がない場合には、それを 1 マイルに設定します。

```
CREATE DISTINCT TYPE JOHNDOE.T_SALARY AS INTEGER WITH COMPARISONS

CREATE DISTINCT TYPE JOHNDOE.MILES  AS FLOAT WITH COMPARISONS

CREATE TABLE EMPLOYEE
  (ID          INTEGER NOT NULL,
   NAME        CHAR (30),
   SALARY      T_SALARY NOT NULL WITH DEFAULT,
   LIVING_DIST MILES    DEFAULT MILES(1) )
```

例 5: 画像 (IMAGE) と音声 (AUDIO) 用の特殊タイプを作成し、表の列として使用します。表スペースが指定されていないので、 IN *tablespace-name1* 文節について記述された規則に基づいてシステムが選択した表スペースに、表が作成されます。 CURRENT PATH はデフォルト値であると想定します。

```
CREATE DISTINCT TYPE IMAGE AS BLOB (10M)

CREATE DISTINCT TYPE AUDIO AS BLOB (1G)

CREATE TABLE PERSON
  (SSN    INTEGER NOT NULL,
   NAME   CHAR (30),
   VOICE  AUDIO,
   PHOTO  IMAGE)
```

例 6: HUMRES 表スペースに表 EMPLOYEE を作成します。表には、次のような制約を定義します。

- 部門番号 (DEPT) の値は、10 ～ 100 の範囲でなければならない。
- 従業員のジョブ (JOB) は、'Sales'、'Mgr'、または 'Clerk' のいずれかでなければならない。
- 1986 年以前からの従業員の給与 (SALARY) はすべて $40,500 以上でなければならない。

注: チェック制約に含まれる列が NULL 可能である場合、それらも NULL になる可能性があります。

```
CREATE TABLE EMPLOYEE
  (ID          SMALLINT NOT NULL,
   NAME        VARCHAR(9),
   DEPT        SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
   JOB         CHAR(5) CHECK (JOB IN ('Sales','Mgr','Clerk')),
   HIREDATE    DATE,
   SALARY      DECIMAL(7,2),
   COMM        DECIMAL(7,2),
   PRIMARY KEY (ID),
   CONSTRAINT  YEARSAL CHECK (YEAR(HIREDATE) > 1986
     OR SALARY > 40500)
  )
  IN HUMRES
```

例 7: PAYROLL 表スペースに全体が含まれる表を作成します。

```
CREATE TABLE EMPLOYEE .....
  IN PAYROLL
```

例 8: データ部分が ACCOUNTING にあり、索引部分が ACCOUNT_IDX にある表を作成します。

```
CREATE TABLE SALARY.....
  IN ACCOUNTING INDEX IN ACCOUNT_IDX
```

例 9: 表を作成して、SQL の変更内容をデフォルトのフォーマットでログ記録します。

```
CREATE TABLE SALARY1 .....
```

または

```
CREATE TABLE SALARY1 .....
  DATA CAPTURE NONE
```

例 10: 表を作成して、SQL の変更内容を拡張フォーマットでログ記録します。

```
CREATE TABLE SALARY2 .....
  DATA CAPTURE CHANGES
```

例 11: SCHED 表スペースに表 EMP_ACT を作成します。 EMPNO、 PROJNO、ACTNO、 EMPTIME、 EMSTDATE、および EMENDATE は列の名前です。表には、次のような制約を定義します。

- すべての行の一連の列、EMPNO、PROJNO、および ACTNO の値は、ユニークでなければならない。
- PROJNO の値は、PROJECT 表の PROJNO 列の既存の値と一致していなければならず、プロジェクトが削除される場合、そのプロジェクトを参照する EMP_ACT の行もすべて削除される。

```
CREATE TABLE EMP_ACT
  (EMPNO       CHAR(6) NOT NULL,
   PROJNO      CHAR(6) NOT NULL,
   ACTNO       SMALLINT NOT NULL,
   EMPTIME     DECIMAL(5,2),
   EMSTDATE    DATE,
   EMENDATE    DATE,
   CONSTRAINT EMP_ACT_UNIQ UNIQUE (EMPNO,PROJNO,ACTNO),
```

```
    CONSTRAINT FK_ACT_PROJ FOREIGN KEY (PROJNO)
                           REFERENCES PROJECT (PROJNO) ON DELETE CASCADE
  )
  IN SCHED
```

ユニーク制約を課すために、 EMP_ACT_UNIQ という名前のユニーク索引が同じスキーマ内に自動的に作成されます。

例 12: アイス・ホッケーの栄誉の殿堂に入る、有名なゴールについての情報を保持する表を作成します。この表には、ゴールをきめたホッケー選手の名前、ゴールをきめられたゴールキーパーの名前、日付と場所、ゴールについての説明文などの情報がリストされます。さらに、可能ならば、その試合についての新聞記事やゴールのスチール写真と動画の保管先を示します。新聞記事はリンク接続されるので、削除したり、名前を変更したりできませんが、この間、既存の表示アプリケーションや更新アプリケーションは操作を続ける必要があります。スチール写真やムービーはリンクされてからアクセスできるようになりますが、この操作はすべて DB2 によって制御されます。リンクが解除されると、スチール写真はリカバリーされて元の所有者に戻されます。一方、ムービー写真はリカバリー対象とならず、リンクが解除された時点で削除されます。説明列と 3 つの DATALINK 列は NULL 可能です。

```
CREATE TABLE HOCKEY_GOALS
  ( BY_PLAYER      VARCHAR(30)   NOT NULL,
    BY_TEAM        VARCHAR(30)   NOT NULL,
    AGAINST_PLAYER VARCHAR(30)   NOT NULL,
    AGAINST_TEAM   VARCHAR(30)   NOT NULL,
    DATE_OF_GOAL   DATE          NOT NULL,
    DESCRIPTION    CLOB(5000),
    ARTICLES       DATALINK  LINKTYPE URL FILE LINK CONTROL MODE DB2OPTIONS,
    SNAPSHOT       DATALINK  LINKTYPE URL FILE LINK CONTROL
                             INTEGRITY ALL
                             READ PERMISSION DB WRITE PERMISSION BLOCKED
                             RECOVERY YES ON UNLINK RESTORE,
    MOVIE          DATALINK  LINKTYPE URL FILE LINK CONTROL
                             INTEGRITY ALL
                             READ PERMISSION DB WRITE PERMISSION BLOCKED
                             RECOVERY NO ON UNLINK DELETE )
```

例 13: EMPLOYEE 表に例外表が必要であるとします。これは、以下のステートメントを使用して作成できます。

```
CREATE TABLE EXCEPTION_EMPLOYEE AS
  (SELECT EMPLOYEE.*,
    CURRENT TIMESTAMP AS TIMESTAMP,
    CAST ('' AS CLOB(32K)) AS MSG
  FROM EMPLOYEE
  ) WITH NO DATA
```

例 14: 次に示されている属性を持つ以下のような表スペースがあるとします。

```
TBSPACE            PAGESIZE    USER   USERAUTH
------------------ ----------- ------ --------
DEPT4K                    4096 BOBBY  Y
PUBLIC4K                  4096 PUBLIC Y
DEPT8K                    8192 BOBBY  Y
DEPT8K                    8192 RICK   Y
PUBLIC8K                  8192 PUBLIC Y
```

- RICK が以下のような表を作成した場合、バイト・カウントは 4005 未満なので、その表は表スペース PUBLIC4K に入れられます。しかし BOBBY が同じ表

を作成した場合、以下のような明示的な権限付与があって BOBBY は USE 特権を保有しているので、その表は表スペース DEPT4K に入れられます。

```
CREATE TABLE DOCUMENTS
  (SUMMARY    VARCHAR(1000),
   REPORT     VARCHAR(2000))
```

- BOBBY が以下のような表を作成した場合、バイト・カウントは 4005 を超えるので、その表は表スペース DEPT8K に入れられます。また、明示的な権限付与によって BOBBY は USE 特権を保有します。しかし DUNCAN が同じ表を作成すると、それは表スペース PUBLIC8K に入れられます。 DUNCAN には該当する特権がないからです。

```
CREATE TABLE CURRICULUM
  (SUMMARY    VARCHAR(1000),
   REPORT     VARCHAR(2000),
   EXERCISES  VARCHAR(1500))
```

例 15: 構造タイプ EMP を使って定義された LEAD 列をもつ表を作成します。LEAD 列に 300 バイトの INLINE LENGTH を指定します。これは、300 バイト以内に収まらない LEAD のインスタンスをすべて、その表以外に保管すること (LOB 値の処理方法と同様に、基本表の行とは別個に) を指示します。

```
CREATE TABLE PROJECTS (PID INTEGER,
  LEAD EMP INLINE LENGTH 300,
  STARTDATE DATE,
            ...)
```

例 16: DEPTNO、DEPTNAME、MGRNO、ADMRDEPT、および LOCATION という名前の 5 つの列を持つ表 DEPT を作成します。 DB2 によって常に値が生成されるよう、列 DEPT を ID 列として定義することにします。 DEPT 列の値は、500 から始まり、1 ずつ増分する必要があります。

```
CREATE TABLE DEPT
  (DEPTNO     SMALLINT      NOT NULL
                GENERATED ALWAYS AS IDENTITY
                  (START WITH 500, INCREMENT BY 1),
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30))
```

例 17: YEAR 列で区画に分割され、 REGION および YEAR 列にディメンションを持つ SALES 表を作成します。データは、YEAR 列のハッシュ値に従って、パーティション内に分散しています。各パーティションで、それらのパーティション上の REGION および YEAR 列の値のユニークな組み合わせを基にして、データはエクステントに編成されます。

```
CREATE TABLE SALES
  (CUSTOMER   VARCHAR(80),
   REGION     CHAR(5),
   YEAR       INTEGER)
PARTITIONING KEY (YEAR)
ORGANIZE BY DIMENSIONS (REGION, YEAR)
```

例 18: PURCHASEDATE 列から生成された、 PURCHASEYEARMONTH 列を持つ SALES 表を作成します。式を使用して、元の PURCHASEDATE 列に対して単調な列を作成し、それによって、ディメンションとして使用するのに適切です。表は REGION 列で区画に分割されており、各パーティションで PURCHASEYEARMONTH 列に従って、エクステントに編成されています。つま

り、異なる領域は異なるパーティションにあり、異なる購入月はこれらのパーティション内の異なるセル (またはエクステントのセット) に属します。

```
CREATE TABLE SALES
  (CUSTOMER            VARCHAR(80),
   REGION              CHAR(5),
   PURCHASEDATE        DATE,
   PURCHASEYEARMONTH   INTEGER
                          GENERATED ALWAYS AS (INTEGER(PURCHASEDATE)/100))
PARTITIONING KEY (REGION)
ORGANIZE BY DIMENSIONS (PURCHASEYEARMONTH)
```

例 19 CUSTOMERNUM 列から生成された、 CUSTOMERNUMDIM 列を持つ CUSTOMER 表を作成します。式を使用して、元の CUSTOMERNUM 列に対して単調な列を作成し、それによって、ディメンションとして使用するのに適切です。表は CUSTOMERNUMDIM 列に従ってセルに編成するされ、表内のそれぞれのセルには、50 人の顧客が入っています。ユニーク索引が CUSTOMERNUM に作成された場合、カスタマー番号は、表内のエクステントの特定のセットに、50 の値のセットがあるように、クラスター化されます。

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME     VARCHAR(80),
   ADDRESS          VARCHAR(200),
   CITY             VARCHAR(50),
   COUNTRY          VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM   INTEGER
                       GENERATED ALWAYS AS (CUSTOMERNUM/50))
ORGANIZE BY DIMENSIONS (CUSTOMERNUMDIM)
```

例 20: ORASERVER という Oracle サーバー上に、EMPLOYEE というリモート基本表を作成します。この新たに作成されたリモート基本表を参照する、EMPLOYEE というニックネームも自動的に作成されます。

```
CREATE TABLE EMPLOYEE
  (EMP_NO       CHAR(6)       NOT NULL,
   FIRST_NAME   VARCHAR(12)   NOT NULL,
   MID_INT      CHAR(1)       NOT NULL,
   LAST_NAME    VARCHAR(15)   NOT NULL,
   HIRE_DATE    DATE,
   JOB          CHAR(8),
   SALARY       DECIMAL(9,2),
   PRIMARY KEY (EMP_NO))
OPTIONS
  (REMOTE_SERVER 'ORASERVER',
   REMOTE_SCHEMA 'J15USER1',
   REMOTE_TABNAME 'EMPLOYEE')
```

以下の CREATE TABLE ステートメントは、表名 (または表名と明示的リモート基本表名) を指定して、大文字小文字のいずれか希望する文字にする方法を示したものです。 employee という小文字の ID は、ID が大文字に暗黙的に変換されることを示すために使用されています。

Informix サーバー上に EMPLOYEE (大文字) というリモート基本表を作成し、その表に EMPLOYEE (大文字) というニックネームを作成します。

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

REMOTE_TABNAME オプションが指定されておらず、かつ *table-name* が引用符で区切られていない場合、通常リモート・データ・ソースでは名前が小文字で保管されるとしても、リモート基本表名は大文字になります。

Informix サーバー上に employee (小文字) というリモート基本表を作成し、その表に EMPLOYEE (大文字) というニックネームを作成します。

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER',
   REMOTE_TABNAME 'employee')
```

区切り ID をサポートするリモート・データ・ソースで表を作成するときには、REMOTE_TABNAME と、大文字小文字のいずれかで表名を指定した文字ストリング定数を使用します。

Informix サーバー上に employee (小文字) というリモート基本表を作成し、その表に employee (小文字) というニックネームを作成します。

```
CREATE TABLE "employee"
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

REMOTE_TABNAME オプションが指定されておらず、かつ *table-name* が引用符で区切られている場合、リモート基本表名は *table-name* と同一になります。

例 21: 生徒 ID を使用して生徒を探すのに使用できるレンジ・クラスター表を作成します。各生徒のレコードには、学校 ID、プログラム ID、生徒番号、生徒 ID、生徒のファーストネーム、生徒のラストネーム、および生徒の成績平均値 (GPA) が含まれます。

```
CREATE TABLE STUDENTS
  (SCHOOL_ID     INTEGER   NOT NULL,
   PROGRAM_ID    INTEGER   NOT NULL,
   STUDENT_NUM   INTEGER   NOT NULL,
   STUDENT_ID    INTEGER   NOT NULL,
   FIRST_NAME    CHAR(30),
   LAST_NAME     CHAR(30),
   GPA           DOUBLE)
ORGANIZE BY KEY SEQUENCE
  (STUDENT_ID
    STARTING FROM 1
    ENDING AT 1000000)
  DISALLOW OVERFLOW
```

各レコードのサイズは、列の合計に位置合わせとレンジ・クラスター表の行のヘッダーを加算して求めます。この場合は、4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (NULL 可能列) + 1 (位置合わせ) + 10 (ヘッダー) で、 98 バイトが行のサイズです。ページ・サイズが 4 KB (4096 バイト) なら、ページのオーバーヘッドを差し引いて 4038 バイトが使用可能なので、ページあたりに 41 のレコードを収められるスペースがある計算になります。つまり、100 万人の生徒のレコードを収容するためには、 24 391 ページ (100 万をページあたりのレコード数 41 で除算) が必要です。表のオーバーヘッド用に 2 ページを追加するとして、表を作成する際に割り振る 4 KB ページの最終的な数は、24 393 になります。

例 22: 制約名の指定されていない、機能従属関係を持つ DEPARTMENT という表を作成します。

```
CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT      NOT NULL,
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30),
  CHECK (DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED)
```

関連概念:

- *管理ガイド: プランニング* の『マルチディメンション・クラスター化索引』
- *フェデレーテッド・システム・ガイド* の『透過 DDL とは ?』

関連タスク:

- *フェデレーテッド・システム・ガイド* の『透過 DDL を使用した新規リモート表の作成』

関連資料:

- *SQL リファレンス 第 1 巻* の『副選択』
- 45 ページの『ALTER TABLE』
- 410 ページの『CREATE TABLESPACE』
- 497 ページの『DECLARE GLOBAL TEMPORARY TABLE』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』
- *SQL リファレンス 第 1 巻* の『パーティションの互換性データ・タイプ』

関連サンプル:

- 『dtudt.c -- How to create, use, and drop user-defined distinct types.』
- 『tbconstr.c -- How to work with constraints associated with tables』
- 『tbcreate.c -- How to create, alter and drop tables』
- 『dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『tbcreate.sqc -- How to create and drop tables (C)』
- 『tbident.sqc -- How to use identity columns (C)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『tbcreate.sqC -- How to create and drop tables (C++)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)』
- 『TbConstr.java -- How to create, use and drop constraints (JDBC)』
- 『TbCreate.java -- How to create and drop tables (JDBC)』
- 『TbGenCol.java -- How to use generated columns (JDBC)』
- 『TbIdent.java -- How to use Identity Columns (JDBC)』
- 『TbTrig.java -- How to use triggers (JDBC)』
- 『DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)』

- 『TbConstr.sqlj -- How to create, use and drop constraints (SQLj)』
- 『TbCreate.sqlj -- How to create and drop tables (SQLj)』
- 『TbIdent.sqlj -- How to use Identity Columns (SQLj)』
- 『TbTrig.sqlj -- How to use triggers (SQLj)』
- 『impexp.sqb -- Export and import tables with table data (MF COBOL)』

CREATE TABLESPACE

CREATE TABLESPACE ステートメントは、データベースに新しい表スペースを作成し、その表スペースにコンテナーを割り当て、その表スペース定義と属性をカタログに記録します。

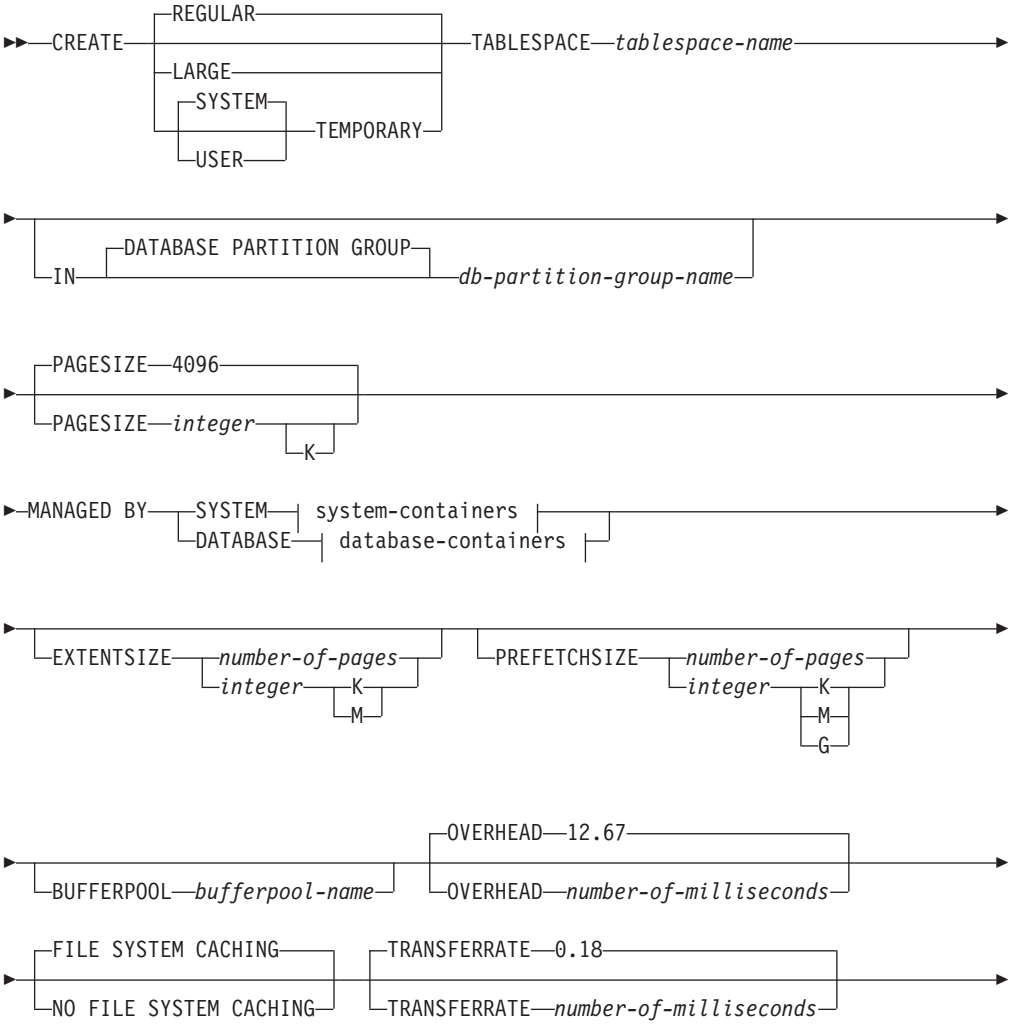
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。 DYNAMICRULES がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

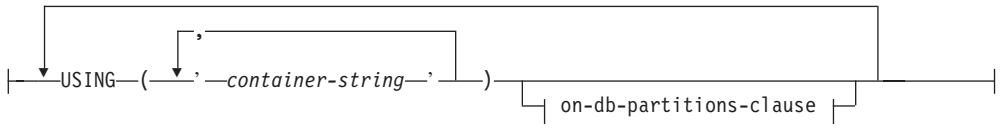
許可:

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

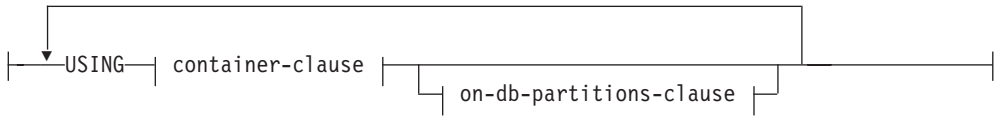
構文:



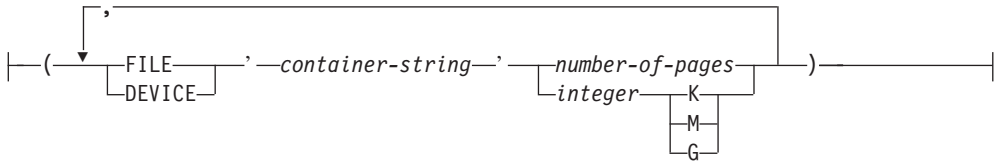
system-containers:



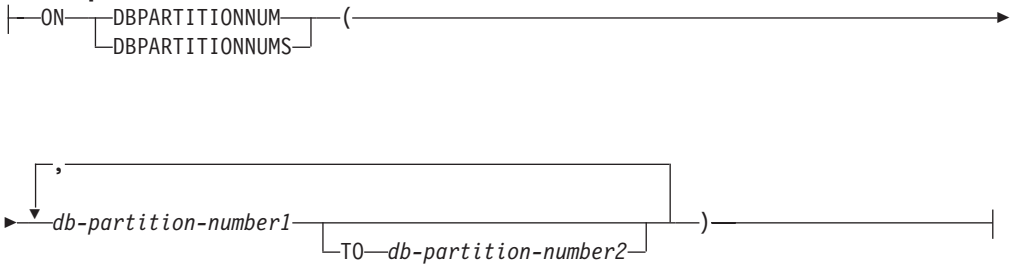
database-containers:



container-clause:



on-db-partitions-clause:



説明:

REGULAR
一時表を除くすべてのデータを保管します。

LARGE
LONG 表形式または LOB の表の列を保管します。また、構造タイプ列または索引データを保管することもできます。表スペースはデータベース管理表スペース (DMS) でなければなりません。

SYSTEM TEMPORARY
一時表 (データベース・マネージャーがソートや結合などの操作を実行するのに使用する作業域) を保管します。キーワード SYSTEM は任意指定です。一時表はこのような表スペースにのみ保管することができるので、データベースには、常に少なくとも 1 つの SYSTEM TEMPORARY 表スペースがなければならない点に注意してください。 TEMPSPACE1 表スペースはデータベースの作成時に自動的に作成されます。

USER TEMPORARY
宣言済みグローバル一時表を保管します。データベースの作成時にユーザー TEMPORARY 表スペースは存在しないことに注意してください。宣言済み一時表を定義できるよう、該当する USE 特権を使って少なくとも 1 つの USER TEMPORARY 表スペースを作成する必要があります。

tablespace-name
表スペースの名前を指定します。これは、1 つの部分からなる名前です。これは、SQL ID です (通常 ID または区切り ID)。 *tablespace-name* (表スペース名) は、すでにカタログに存在している表スペースを指定するものであってはなりません (SQLSTATE 42710)。 *tablespace-name* を文字 'SYS' で始めることはできません (SQLSTATE 42939)。

IN DATABASE PARTITION GROUP *db-partition-group-name*
表スペースのデータベース・パーティション・グループを指定します。該当のデータベース・パーティション・グループは存在していなければなりません。SYSTEM TEMPORARY 表スペースの作成の際に指定できるデータベース・パーティション・グループは、IBMTEMPGROUP だけです。 DATABASE PARTITION GROUP キーワードはオプションです。

データベース・パーティション・グループを指定しないと、デフォルトのデータベース・パーティション・グループ (IBMDEFAULTGROUP) が、REGULAR、LARGE、および USER TEMPORARY 表スペースに使用されます。 SYSTEM TEMPORARY 表スペースには、デフォルト・データベース・パーティション・グループ IBMTEMPGROUP が使われます。

PAGESIZE *integer* **[K]**
表スペースに使用するページのサイズを定義します。接尾部 K を持たない *integer* の有効値は、 4 096 または 8 192、16 384、または 32 768 です。接尾部 K を持つ *integer* の有効値は、4 または 8、16、または 32 です。ページ・サイズがこれらのいずれの値にも該当しない場合 (SQLSTATE 428DE)、あるいはページ・サイズが表スペースと関連付けられたバッファー・プールのページ・サイズと同じではない場合 (SQLSTATE 428CB) には、エラーが起こります。 デフォルト値は 4 096 バイト (4K) ページです。 *integer* と K の間には、任意の数のスペースを使用できます (スペースなしでも可)。

MANAGED BY SYSTEM
表スペースを、システム管理スペース (SMS) 表スペースとして指定します。

system-containers
システム管理表スペース (SMS) に対するコンテナーを指定します。

USING *('container-string',...)*
システム管理表スペース (SMS) に対して、表スペースに属し、表スペースのデータの保管先となる 1 つまたは複数のコンテナーを指定します。コンテナー・ストリング (*container-string*) の長さは、240 バイトを超えてはなりません。

各 *container-string* は、絶対ディレクトリー名または相対ディレクトリー名にすることができます。 ディレクトリー名が絶対ではない場合は、データベース・ディレクトリーからの相対ディレクトリーになります。ディレクトリー名のコンポーネントのいずれかが存在しない場合は、それがデータベース・マネージャーによって作成されます。表スペースをドロップすると、デ

ータベース・マネージャーによって作成されたすべてのコンポーネントが削除されます。コンテナー・ストリングで指定されたディレクトリーが存在する場合、そのディレクトリーにはファイルやサブディレクトリーがあってはなりません (SQLSTATE 428B2)。

container-string の形式は、オペレーティング・システムによって異なります。 コンテナーは、オペレーティング・システムの通常の方法で指定されます。たとえば、Windows のディレクトリー・パスはドライブ文字と ":" から始まり、 UNIX 系システムでは "/" から始まります。

リモート・リソース (LAN でリダイレクトされたドライブや NFS でマウントされたファイル・システムなど) は、現在、Network Appliance Filers、IBM iSCSI、IBM Network Attached Storage、 Network Appliance iSCSI、NEC iStorage S2100、S2200、または S4100、 または NEC iStorage NS シリーズを Windows DB2 サーバーで使用する場合にのみサポートされます。 NEC iStorage NS シリーズは、無停電電源装置 (UPS) を使用した場合にのみサポートされています。 (スタンバイではなく) 連続 UPS を推奨します。

on-db-partitions-clause
パーティション・データベースにおいて、コンテナーを作成するパーティションを指定します。この文節を指定しない場合、他のどの *on-db-partitions-clause* にも明示的に指定されていないデータベース・パーティション・グループ内のパーティションでコンテナーが作成されます。データベース・パーティション・グループ IBMTEMPGROUP で定義されている SYSTEM TEMPORARY 表スペースについては、 *on-db-partitions-clause* を指定しないと、データベースに追加されたすべての新しいパーティションでもコンテナーが作成されます。

MANAGED BY DATABASE
表スペースを、データベース管理スペース (DMS) として指定します。

database-containers
データベース管理表スペース (DMS) に対するコンテナーを指定します。

USING
container-clause を導きます。

container-clause
データベース管理表スペース (DMS) に対するコンテナーを指定します。

(FILE|DEVICE *'container-string' number-of-pages,...***)**
データベース管理表スペース (DMS) に対して、表スペースに属し、表スペースのデータの保管先となる 1 つまたは複数のコンテナーを指定します。コンテナーのタイプ (FILE または DEVICE) とそのサイズ (PAGESIZE ページの数) を指定します。 このサイズは整数値としても指定でき、その後に K (K バイトの場合)、M (M バイトの場合)、または G (G バイトの場合) を付けます。このように指定した場合、ページ・サイズで割ったバイト数の値よりも小さいか等しい整数で、最大の整数値は、コンテナーのページ数を判別するために使用します。 FILE と DEVICE のコンテナーを混合して指定できます。コンテナー・ストリング (*container-string*) の長さは、 254 バイトを超えてはなりません。

FILE コンテナーの場合、*container-string* は、絶対ファイル名または相対ファイル名でなければなりません。 絶対ファイル名以外のファイル名は、データベース・ディレクトリーからの相対パス名になります。ディレクトリー名のコンポーネントのいずれかが存在しない場合は、それがデータベース・マネージャーによって作成されます。ファイルが存在しない場合、データベース・マネージャーによってそのファイルが作成され、指定されたサイズに初期化されます。表スペースをドロップすると、データベース・マネージャーによって作成されたすべてのコンポーネントが削除されます。

注: ファイルが存在する場合は上書きされ、指定したサイズより小さい場合には拡張されます。 指定したサイズよりもファイルの方が大きくても、ファイルは切り捨てられません。

DEVICE コンテナーの場合、*container-string* は装置名でなければなりません。 その装置はすでに存在していなければなりません。

すべてのコンテナーは、データベース全体を通してユニークでなければなりません。 1 つのコンテナーは、1 つの表スペースにのみ属することができます。 コンテナーごとに異なるサイズにすることができますが、すべてのコンテナーが同じサイズの場合にパフォーマンスは最高になります。 *container-string* の正しい形式は、オペレーティング・システムによって異なります。 コンテナーは、オペレーティング・システムの通常の方法で指定されます。

リモート・リソース (LAN でリダイレクトされたドライブや NFS でマウントされたファイル・システムなど) は、現在、Network Appliance Filers、IBM iSCSI、IBM Network Attached Storage、 Network Appliance iSCSI、NEC iStorage S2100、S2200、または S4100、 または NEC iStorage NS シリーズを Windows DB2 サーバーで使用する場合にのみサポートされます。 NEC iStorage NS シリーズは、無停電電源装置 (UPS) を使用した場合にのみサポートされています。 (スタンバイではなく) 連続 UPS を推奨します。

on-db-partitions-clause
パーティション・データベースにおいて、コンテナーを作成するパーティションを指定します。この文節を指定しない場合、他のどの *on-db-partitions-clause* にも明示的に指定されていないデータベース・パーティション・グループ内のパーティションでコンテナーが作成されます。データベース・パーティション・グループ IBMTEMPGROUP で定義されている SYSTEM TEMPORARY 表スペースについては、*on-db-partitions-clause* を指定しないと、データベースに追加されたすべての新しいパーティションでもコンテナーが作成されます。

on-db-partitions-clause
パーティション・データベースにおいて、コンテナーを作成するパーティションを指定します。

ON DBPARTITIONNUMS
特定のパーティションを指定することを示すキーワードです。
DBPARTITIONNUM は DBPARTITIONNUMS の同義語です。

db-partition-number1
データベース・パーティション番号を指定します。

TO *db-partition-number2*
パーティション番号の範囲を指定します。 *db-partition-number2* の値は、 *db-partition-number1* の値よりも大きいか等しい値でなければなりません (SQLSTATE 428A9)。この表スペースのデータベース・パーティション・グループにパーティションが組み込まれると、コンテナーが作成されるパーティションに、指定したパーティション番号の範囲 (指定したパーティション番号を含む) のすべてのパーティションが組み込まれます。

番号によって指定するパーティションと、パーティションの範囲内のすべてのパーティションは、表スペースを定義するデータベース・パーティション・グループに存在している必要があります (SQLSTATE 42729)。 あるパーティション番号を明示的に、または範囲の中で指定できるのは、このステートメントのただ 1 つの *on-db-partitions-clause* の中だけです (SQLSTATE 42613)。

EXTENTSIZE *number-of-pages*
次のコンテナーに移る前にコンテナーに書き込まれる PAGESIZE ページの数を指定します。このエクステント・サイズ値は、後に K (K バイトの場合)、または M (M バイトの場合) を付けた整数値として指定することもできます。このように指定した場合、バイト数をページ・サイズで割った値よりも小さいか等しい整数で、最大の整数値が、エクステント・サイズの値を判別するために使用します。データが保管されていくにつれて、データベース・マネージャーはコンテナー間を繰り返し循環します。

デフォルト値は DFT_EXTENT_SZ データベース構成パラメーターによって指定されます。有効な範囲は 2 ～ 256 ページです。

PREFETCHSIZE *number-of-pages*
データのプリフェッチの実行中に、表スペースから読み取られる PAGESIZE ページの数を指定します。このプリフェッチ・サイズ値は、後に K (K バイトの場合)、M (M バイトの場合)、または G (G バイトの場合) を付けた整数値としても指定できます。このように指定した場合、バイト数をページ・サイズで割った値よりも小さいか等しい整数で、最大の整数値が、プリフェッチ・サイズのページ値の数を判別するために使用します。プリフェッチでは、照会に必要なデータがその照会で参照される前に読み取られるため、照会では入出力の実行を待たずにすみます。

デフォルト値は DFT_PREFETCH_SZ 構成パラメーターによって指定されます。

BUFFERPOOL *bufferpool-name*
この表スペースの表に対して使用するバッファー・プールの名前を指定します。バッファー・プールは存在している必要があります (SQLSTATE 42704)。 これを指定しない場合、デフォルトのバッファー・プール (IBMDEFAULTBP) が使用されます。バッファー・プールのページ・サイズは、表スペースに指定された (またはデフォルト指定された) ページ・サイズと一致していなければなりません (SQLSTATE 428CB)。バッファー・プ

ールに対して、この表スペースのデータベース・パーティション・グループを定義する必要があります (SQLSTATE 42735)。

OVERHEAD *number-of-milliseconds*
入出力制御装置のオーバーヘッドとディスク・シーク待ち時間をミリ秒単位で指定する数値リテラルです (整数、10 進数、または浮動小数点数)。この数値がすべてのコンテナーで同一でない場合、それは表スペースに属するすべてのコンテナーの平均値でなければなりません。この値は、照会の最適化の過程で入出力コストを判別するのに使用されます。

FILE SYSTEM CACHING または **NO FILE SYSTEM CACHING**
入出力操作をファイル・システム・レベルでキャッシュに入れるかどうかを指定します。デフォルトは FILE SYSTEM CACHING です。

FILE SYSTEM CACHING
ターゲット表スペース内のすべての入出力操作が、ファイル・システム・レベルでキャッシュに入れられるように指定します。

NO FILE SYSTEM CACHING
すべての入出力操作がファイル・システム・レベル・キャッシュをバイパスするように指定します。

TRANSFERRATE *number-of-milliseconds*
1 ページをメモリーに読み込むための時間をミリ秒単位で指定する数値リテラルです (整数、10 進数、浮動小数点数)。この数値がすべてのコンテナーで同一でない場合、それは表スペースに属するすべてのコンテナーの平均でなければなりません。この値は、照会の最適化の過程で入出力コストを判別するのに使用されます。

DROPPED TABLE RECOVERY
ROLLFORWARD コマンドの RECOVER TABLE ON オプションを使用すれば、指定した表スペースでドロップされた表を回復できる場合があります。この文節は、REGULAR 表スペースにのみ指定できます (SQLSTATE 42613)。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DBPARTITIONNUM の代わりに NODE を指定できます。
    - DBPARTITIONNUMS の代わりに NODES を指定できます。
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
    - LARGE の代わりに LONG を指定できます。
- 表スペースをデータベース管理表スペース（DMS）にするか、システム管理表スペース（SMS)にするかは、トレードオフの関係にあります。(それぞれの特徴をふまえた上で、どちらが要件に適切かを検討して下さい)
- 各コンテナー定義には、53 バイトに加えて、コンテナー名を保管するのに必要なバイト数が必要です。表スペースのすべてのコンテナー名を結合した長さは、20 480 バイトを超えることはできません (SQLSTATE 54034)。

- データベースに複数の一時 (TEMPORARY) 表スペースが存在する場合、それらは、使用率のバランスを調整するために、「ラウンドロビン」式に使用されます。
- パーティション・データベースで、複数のデータベース・パーティションが同じ物理ノードに存在する場合、このようなデータベース・パーティションに同じ装置または特定のパスを指定することはできません (SQLSTATE 42730)。この環境の場合、それぞれのデータベース・パーティションごとにユニークな *container-string* を指定するか、または相対パス名を使用してください。
- SMS または DMS コンテナーの作成時にコンテナー・ストリング構文にデータベース・パーティション式を指定することができます。データベース・パーティション式は一般に、パーティション・データベース・システムで複数の論理データベース・パーティションを使用する場合に指定します。この指定により、コンテナー名がノード (データベース・パーティション・サーバー) 間でユニークなものとなります。この式を指定する場合、データベース・パーティション番号はコンテナー名の一部となるか、あるいは、追加の引き数を指定すれば、引き数の結果はコンテナー名の一部となります。

  データベース・パーティション式を示すには、引き数 ″ $N″ ([ブランク]$N) を使用します。データベース・パーティション式はコンテナー名内で自由に使用できます。また、複数のデータベース・パーティションを指定できます。データベース・パーティション式を終了するにはスペース文字を使用します。スペースの後ろにあるものは、データベース・パーティション式が評価された後でコンテナー名に付加されます。コンテナー名の中で、データベース・パーティション式の後ろにスペースがない場合は、ストリングの残りは式の一部であると見なされます。引き数は、以下のいずれかの形式でのみ使用できます。

表 8. コンテナーを作成するための引き数: 演算子は左から右へと評価されます。この例では、データベース・パーティション番号は 5 であると想定します。

構文	例	値
[blank]$N	" $N"	5
[blank]$N+[number]	" $N+1011"	1016
[blank]$N%[number]	" $N%3" [a]	2
[blank]$N+[number]%[number]	" $N+12%13"	4
[blank]$N%[number]+[number]	" $N%3+20"	22
[a] % はモジュラスです。		

例:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
  (device '/dev/rcont $N' 20000)

On a two database partition system, the following containers
 would be created:
  /dev/rcont0  - on DATABASE PARTITION 0
  /dev/rcont1  - on DATABASE PARTITION 1


CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
  (file '/DB2/containers/TS2/container $N+100' 10000)

On a four database partition system, the following containers
 would be created:
  /DB2/containers/TS2/container100  - on DATABASE PARTITION 0
```

```
  /DB2/containers/TS2/container101  - on DATABASE PARTITION 1
  /DB2/containers/TS2/container102  - on DATABASE PARTITION 2
  /DB2/containers/TS2/container103  - on DATABASE PARTITION 3


CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
  ('/TS3/cont $N%2','/TS3/cont $N%2+2')

On a two database partition system, the following containers
 would be created:
  /TS3/cont0  - On DATABASE PARTITION 0
  /TS3/cont2  - On DATABASE PARTITION 0
  /TS3/cont1  - On DATABASE PARTITION 1
  /TS3/cont3  - On DATABASE PARTITION 1


データベース・パーティションが 5 の場合、以下のコンテナーは、

  '/dbdir/node $N /cont1'
  '/ $N+1000 /file1'
  ' $N%10 /container'
  '/dir/ $N%5+2000 /dmscont'

次のように作成されます。

  '/dbdir/node5/cont1'
  '/1005/file1'
  '5/container'
  '/dir/2000/dmscont'
```

例:

例 1: UNIX 系システムで、それぞれ 10 000 の 4K ページの 3 つのデバイスを使用する通常のデータベース管理表スペース (DMS) を作成します。それらの入出力特性も指定します。

```
CREATE TABLESPACE PAYROLL
  MANAGED BY DATABASE
  USING (DEVICE'/dev/rhdisk6' 10000,
    DEVICE '/dev/rhdisk7' 10000,
    DEVICE '/dev/rhdisk8' 10000)
  OVERHEAD 12.67
  TRANSFERRATE 0.18
```

例 2: 3 つの別個のドライブの 3 つのディレクトリーを使用し、エクステント・サイズを 64 ページ、プリフェッチ・サイズを 32 ページに指定して、 Windows NT または Windows 2000 で通常の システム管理表スペース (SMS) を作成します。

```
CREATE TABLESPACE ACCOUNTING
  MANAGED BY SYSTEM
  USING ('d:¥acc_tbsp', 'e:¥acc_tbsp', 'f:¥acc_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
```

例 3: それぞれ 50,000 ページの 2 つのファイル、および 256 ページのエクステント・サイズを使用して、UNIX で一時 DMS 表スペースを作成します。

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY DATABASE
  USING (FILE '/tmp/tempspace2.f1' 50000,
    FILE '/tmp/tempspace2.f2' 50000)
  EXTENTSIZE 256
```

例 4: UNIX のデータベース・パーティションで、データベース・パーティション・グループ ODDNODEGROUP (パーティション 1、3、5) にデータベース管理表スペース (DMS) を作成します。すべてのパーティションで、装置 /dev/rhdisk0 の 10 000 個の 4K ページを使用します。また、それぞれのパーティションに、 40 000 個の 4K ページがあるパーティション固有の装置を指定します。

```
CREATE TABLESPACE PLANS
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn1hd01' 40000)
  ON DBPARTITIONNUM (1)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
  ON DBPARTITIONNUM (3)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
  ON DBPARTITIONNUM (5)
```

関連サンプル:

- 『tbtemp.sqc -- How to use a declared temporary table (C)』
- 『TbTemp.java -- How to use Declared Temporary Table (JDBC)』

CREATE TRANSFORM

CREATE TRANSFORM ステートメントは、グループ名で識別されるトランスフォーメーション関数またはメソッドを定義します。この関数は、ホスト言語プログラムおよび外部関数とメソッドを相手に構造タイプ値を交換するために使います。

呼び出し:

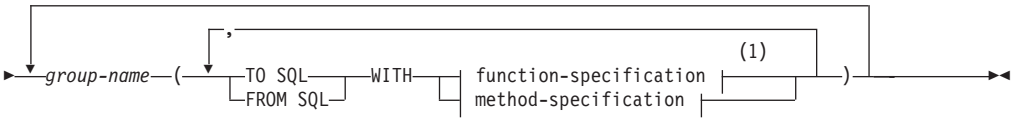
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントの中で発行することができます。 DYNAMICRULES がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

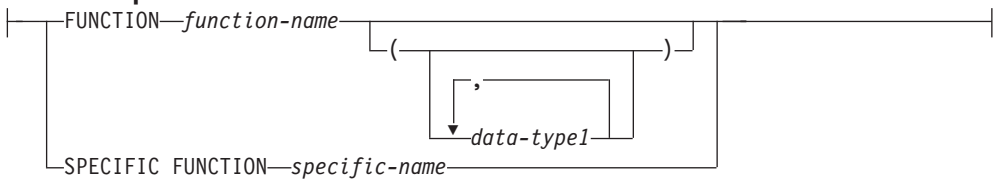
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- *type-name* で指定されたタイプの定義者と、指定された関数ごとの EXECUTE 特権

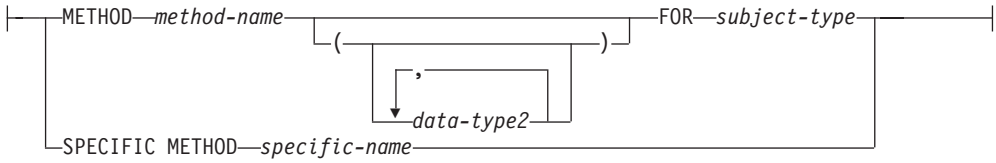
構文:



function-specification:



method-specification:



注:

1 同じ文節を複数回指定することはできません。

説明:

TRANSFORM または **TRANSFORMS**
1 つ以上のトランスフォーム・グループを定義することを指示します。いずれかのバージョンのキーワードを指定することができます。

FOR *type-name*
トランスフォーム・グループを定義する対象となるユーザー定義構造タイプの名前を指定します。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のない *type-name* の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のない *type-name* の修飾子が暗黙指定されます。この *type-name* は既存のユーザー定義タイプであり (SQLSTATE 42704)、しかも構造タイプでなければなりません (SQLSTATE 42809)。構造タイプまたはこれと同じタイプ階層内の他の構造タイプが、特定のグループ名を使ってトランスフォームをすでに定義済みであってはなりません (SQLSTATE 42739)。

group-name
TO SQL および FROM SQL 関数またはメソッドをもつトランスフォーム・グループの名前を指定します。その名前はユニーク名である必要はありません。しかし、その名前の付いたトランスフォーム・グループ (同じ TO SQL や FROM SQL 指示が定義されている) が、指定の *type-name* ですでに定義済みであってはなりません (SQLSTATE 42739)。 *group-name* は、SQL ID である必要があり、長さは最大 18 文字であり (SQLSTATE 42622)、そして修飾子接頭部が付いていてはなりません (SQLSTATE 42601)。 *group-name* を接頭部 SYS で始めることはできません。これは、データベースで使うために予約されているからです (SQLSTATE 42939)。

どのグループに対しても、 FROM SQL と TO SQL 関数のそれぞれの指名を最大で 1 つずつ指定することができます (SQLSTATE 42628)。

TO SQL
SQL ユーザー定義構造タイプ・フォーマットに値をトランスフォームするのに使用する特定の関数を定義します。この関数では、すべてのパラメーターを組み込みデータ・タイプとして使用しなければならず、戻りタイプは *type-name* です。

FROM SQL
SQL ユーザー定義構造タイプを表す組み込みデータ・タイプ値に値をトランスフォームするのに使用する特定の関数を定義します。この関数では、データ・タイプ *type-name* の 1 つのパラメーターを使う必要があり、 1 つの組み込みデータ・タイプ (または一連の組み込みデータ・タイプ) を戻します。

WITH *function-specification*
関数インスタンスの指定には、いくつかの方法を使うことができます。

FROM SQL を指定する場合、*function-specification* に、次のような要件を満たす関数を指定しなければなりません。

- タイプ *type-name* の 1 つのパラメーターがある
- 戻りタイプは、組み込みタイプであるか、またはすべてが組み込みタイプの列をもつ行である
- シグニチャーは、LANGUAGE SQL を指定しているか、または LANGUAGE SQL をもつ別の FROM SQL トランスフォーム関数の使用を指定している

TO SQL を指定する場合、*function-specification* に、次のような要件を満たす関数を指定しなければなりません。

- すべてのパラメーターに組み込みタイプがある
- 戻りタイプは *type-name* である
- シグニチャーは、LANGUAGE SQL を指定しているか、または LANGUAGE SQL をもつ別の TO SQL トランスフォーム関数の使用を指定している

function-specification に、これらの要件 (FROM SQL トランスフォーム関数として使用する場合の要件、または TO SQL トランスフォーム関数として使用する場合の要件) を満たさない関数を指定すると、エラー (SQLSTATE 428DC) になります。

FUNCTION *function-name*

特定の関数を名前で指定します。 *function-name* (関数名) の関数がちょうど 1 つだけ存在している場合にのみ有効です。指定する関数には、任意の数のパラメーターを定義できます。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

指定したスキーマまたは暗黙のスキーマにこの名前の関数が存在しない場合は、エラー (SQLSTATE 42704) になります。指定したスキーマまたは暗黙のスキーマに、この関数の特定インスタンスが複数存在する場合、エラー (SQLSTATE 42725) になります。関数選択の標準アルゴリズムは使用されません。

FUNCTION *function-name* **(***data-tape1,...***)**

使用する関数を固有に指定する関数シグニチャーを指定します。関数選択の標準アルゴリズムは使用されません。

function-name

関数の名前を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(*data-type1,...***)**

ここで指定するデータ・タイプは、 CREATE FUNCTION ステートメント上で (対応する位置に) 指定されたデータ・タイプに一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値を使って、特定の関数を識別します。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。代わりに、空の括弧をコーディングすることによって、データ・タイプの一致を調べる際にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。 ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントにおける指定に完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。 FOR BIT DATA 属性は、一致検索のためのシグニチャーの一部とは見なされません。たとえばシグニチャーの中に CHAR FOR BIT DATA が指定されている場合、それは CHAR と定義されている関数とだけ一致することになります。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つ関数がない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC FUNCTION *specific-name*
関数の作成時に指定された特定関数名、またはデフォルト値として使用された特定関数名を使用して、特定のユーザー定義関数を指定します。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。
specific-name (特定名) は、指定したスキーマまたは暗黙のスキーマの特定関数のインスタンスを指定していなければなりません。そうでない場合、エラー (SQLSTATE 42704) になります。

WITH *method-specification*
メソッド・インスタンスの指定には、いくつかの方法を使うことができます。

FROM SQL を指定する場合、*method-specification* に、次のような要件を満たすメソッドを指定しなければなりません。

- *subject-type* で指定されるタイプは、タイプ *type-name* である
- 戻りタイプは、組み込みタイプであるか、またはすべてが組み込みタイプの列を使用する行である
- シグニチャーは、LANGUAGE SQL を指定しているか、または LANGUAGE SQL をもつ別の FROM SQL トランスフォーム関数またはメソッドの使用を指定している

TO SQL を指定する場合、*method-specification* に、次のような要件を満たすメソッドを指定しなければなりません。

- *subject-type* で指定されるタイプは、タイプ *type-name* である
- 暗黙 SELF パラメーター以外の他のすべてのパラメーター・タイプは、組み込みタイプである
- 戻りタイプは *type-name* である
- メソッドが SELF AS RESULT として宣言される

- シグニチャーは、LANGUAGE SQL を指定しているか、または LANGUAGE SQL をもつ別の TO SQL トランスフォーム関数またはメソッドの使用を指定している

method-specification に、これらの要件 (FROM SQL トランスフォーム・メソッドとして使用する場合の要件、または TO SQL トランスフォーム・メソッドとして使用する場合の要件) を満たさないメソッドを指定すると、エラー (SQLSTATE 428DC) になります。

METHOD *method-name* **FOR** *subject-type*
特定のメソッドを名前で指定します。指定したプロパティーを持つメソッドが 1 つだけ存在している場合にのみ有効です。 *method-name* ID は、非修飾名です。このように指定されたメソッドには、任意の数のパラメーターを定義できます。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

暗黙のスキーマ内で、指定したタイプにこの名前のメソッドが存在しない場合は、エラー (SQLSTATE 42704) になります。暗黙のスキーマに、このメソッドの特定インスタンスが複数存在する場合、エラー (SQLSTATE 42725) になります。メソッド解決の標準アルゴリズムは使用されません。

METHOD *method-name* **(***data-tape2,...***) FOR** *subject-type*
使用するメソッドを固有に指定するメソッド・シグニチャーを指定します。メソッド解決の標準アルゴリズムは使用されません。

method-name **FOR** *subject-type*
メソッドの名前を指定します。 *method-name* ID は、非修飾名です。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(*data-type2,...***)**
ここで指定するデータ・タイプは、メソッド作成ステートメント上で (対応する位置に) 指定されたデータ・タイプに一致していなければなりません。データ・タイプの数、およびデータ・タイプを論理的に連結した値を使って、特定のメソッドを識別します。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。代わりに、空の括弧をコーディングすることによって、データ・タイプの一致を調べる際にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE

42601)。ただし、長さ、精度、または位取りをコーディングする場合、その値は、メソッド作成ステートメントにおける指定に完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。 FOR BIT DATA 属性は、一致検索のためのシグニチャーの一部とは見なされません。たとえばシグニチャーの中に CHAR FOR BIT DATA が指定されている場合、それは CHAR と定義されているメソッドとだけ一致することになります。

暗黙のスキーマに、指定したシグニチャーを持つメソッドがない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC METHOD *specific-name*

メソッドの作成時に指定された特定関数名、またはデフォルト値として使用された特定関数名を使用して、特定のユーザー定義メソッドを指定します。

動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

specific-name (特定名) は、指定したスキーマまたは暗黙のスキーマの特定メソッドのインスタンスを指定していなければなりません。そうでない場合、エラー (SQLSTATE 42704) になります。

規則:

- FROM SQL 関数またはメソッドから戻された 1 つ以上の組み込みタイプが、TO SQL 関数またはメソッドのパラメーターである 1 つ以上の組み込み関数と直接対応しているべきです。これが、この 2 つの関数の相反する関係の論理的結末です。 FROM トランスフォームと TO トランスフォームの間のこの関連が保持されない場合は、エラー (SQLSTATE -3) になります。

注:

- 静的 SQL の場合は TRANSFORM GROUP プリコンパイル・オプションまたは BIND オプションを使って、動的 SQL の場合は SET CURRENT DEFAULT TRANSFORM GROUP ステートメントを使って、アプリケーション・プログラム内でトランスフォーム・グループを指定しない場合に、そのアプリケーション・グループが、 *type-name* で指定されたユーザー定義の構造タイプに基づいているホスト変数を検索または送信しようとすると、トランスフォーム・グループ 'DB2_PROGRAM' 内のトランスフォーム関数またはメソッドが使われます (定義されている場合)。データ・タイプ *type-name* の値を検索すると、 FROM SQL トランスフォームが呼び出され、構造タイプは、トランスフォーム関数またはメソッドから戻された組み込みデータ・タイプにトランスフォームされます。同様に、データ・タイプ *type-name* の値に割り当てられることになるホスト変数を送信すると、 TO SQL トランスフォームが呼び出され、組み込みデータ・タイプ値は構造タイプ値にトランスフォームされます。 TO SQL トランスフォームがメソッドの場合は、適切な動的タイプのオブジェクトが作成され、 TO SQL トランスフォーム・メソッドが、新しく作成されたオブジェクトを対象引き数として呼び出されます。ユーザー定義のトランスフォーム・グループを指定しない場

合や、 'DB2_PROGRAM' グループ (特定の構造タイプのもの) が定義されていない場合、エラーが生じます (SQLSTATE 42741)。

- 構造化ホスト変数を表す組み込みデータタイプは、次のように割当てる必要があります。
  - [割当元]
    プリコンパイル時に (検索割当規則を使用し) TRANSFORM GROUP オプションで定義した構造化タイプ用の FROM SQL トランスフォーム関数結果
  - [割当先]
    プリコンパイル時に (ストレージ割当規則を使用し) TRANSFORM GROUP オプションで定義された構造化タイプ用の TO SQLトランスフォーム関数のパラメーター

  ホスト変数が、該当するトランスフォーム関数またはメソッドでの規定のタイプと互換性のある割り当てでない場合、エラーが起きます (組み込みの場合は SQLSTATE 42821、バインドアウトの場合は SQLSTATE 42806)。ストリングの割り当てが原因のエラーの詳細は、『ストリング割り当て』を参照してください。
- パラメーターまたは戻りタイプとしてデータ・タイプ *type-name* を使って、 SQL で作成されていないユーザー定義関数またはメソッドを呼び出す場合、そのたびに 'DB2_FUNCTION' という名前のデフォルト・トランスフォーム・グループ内で指定されているトランスフォーム関数またはメソッドが使用されます。これが行われるのは、関数またはメソッドに TRANSFORM GROUP 文節を指定しない場合です。データ・タイプ *type-name* の引き数を使って関数またはメソッドを呼び出す場合、 FROM SQL トランスフォームが実行され、構造タイプは、トランスフォーム関数またはメソッドから戻された組み込みデータ・タイプにトランスフォームされます。同様に、関数またはメソッドの戻りデータ・タイプがデータ・タイプ *type-name* である場合、 TO SQL トランスフォームが呼び出され、外部関数プログラムから戻された組み込みデータ・タイプ値は、構造タイプ値にトランスフォームされます。

  TO SQL トランスフォームがメソッドの場合で、 SELF AS RESULT として宣言されているメソッドのメソッド呼び出しの結果を、 TO SQL トランスフォームを使用してトランスフォームする場合は、対象の動的タイプのオブジェクトが作成されます。メソッドが SELF AS RESULT として宣言されていない場合は、最も具体的なディスパッチ可能メソッドの結果の宣言済みタイプのオブジェクトが作成されます。メソッドが SELF AS RESULT として宣言される場合は、解決済みの関数またはメソッドの結果の宣言済みタイプのオブジェクトが作成されます。 TO SQL トランスフォーム・メソッドは、実際の関数またはメソッド呼び出しから戻される基本タイプの引き数と一緒に、新しく作成されたオブジェクトを対象引き数として呼び出されます。
- 構造タイプの中に、やはり構造タイプである属性が入っている場合、それに関連したトランスフォーム関数またはメソッドは、すべてのネストされた構造タイプを繰り返し展開 (またはアセンブル) する必要があります。つまり、トランスフォーム関数またはメソッドの結果やパラメーターは、サブジェクト構造タイプ (ネストされたすべての構造タイプも含む) のすべての基本属性を表す一連の組み込みタイプだけで構成されることを意味します。 ネストされた構造タイプを処理するために、トランスフォーム関数またはメソッドが「カスケード化」されることはありません。

- このステートメントに指定する関数またはメソッドは、このステートメントの実行時に上記の概説どおりの規則に従って解決されます。これらの関数は、この後の SQL ステートメント内で (暗黙的に) 使用された場合、これ以外の解決プロセスをたどりません。このステートメントで定義されたトランスフォーム関数またはメソッドは、このステートメントで解決されるとおりに記録されます。ただし、トランスフォーム・メソッドが呼び出されると、 REFER (メソッドの動的ディスパッチ) のアルゴリズムが適用されます。
- 特定のタイプの属性またはサブタイプを作成またはドロップしたときは、ユーザー定義構造タイプのトランスフォーム関数も変更する必要があります。
- ある特定のトランスフォーム・グループについて、FROM SQL トランスフォームと TO SQL トランスフォームを指定できるのは、同じ *group-name* 文節、別の *group-name* 文節、または別の CREATE TRANSFORM ステートメントのいずれかにおいてです。 既存のグループ定義をあらかじめドロップしておかないと、FROM SQL トランスフォームまたは TO SQL トランスフォームの指定を再定義できないことが唯一の制約事項です。それによって、たとえば、特定のグループの FROM SQL トランスフォームを先に定義しておいてから、後で同じグループ用の対応する TO SQL トランスフォームを定義することができます。

例:

例 1: ユーザー定義構造タイプの多角形を、 C 用にカスタマイズしたトランスフォーム関数と Java 用に特殊化したトランスフォーム関数に関連付ける 2 つのトランスフォーム・グループを作成します。

```
CREATE TRANSFORM FOR POLYGON
  mystruct1 (FROM SQL WITH FUNCTION myxform_sqlstruct,
             TO SQL WITH FUNCTION myxform_structsql)
  myjava1   (FROM SQL WITH FUNCTION myxform_sqljava,
             TO SQL WITH FUNCTION myxform_javasql)
```

例 2 ユーザー定義構造タイプの多角形を、 C 用にカスタマイズしたトランスフォーム・メソッドと Java 用に特殊化したトランスフォーム・メソッドに関連付ける 2 つのトランスフォーム・グループを作成します。

```
CREATE TRANSFORM FOR POLYGON
  mystruct1 (FROM SQL WITH METHOD myxform_sqlstruct FOR POLYGON,
             TO SQL WITH METHOD myxform_structsql FOR POLYGON)
  myjava1   (FROM SQL WITH METHOD myxform_sqljava FOR POLYGON,
             TO SQL WITH METHOD myxform_javasql FOR POLYGON)
```

例 3: 現行スキーマ内に存在しないタイプのトランスフォーム・グループを作成します。

```
CREATE TRANSFORM FOR NEWTON.POLYGON
  mystruct1 (FROM SQL WITH METHOD myxform_sqlstruct FOR NEWTON.POLYGON,
             TO SQL WITH METHOD myxform_structsql FOR NEWTON.POLYGON)
```

関連資料:

- *SQL リファレンス 第 1 巻* の『割り当てと比較』

CREATE TRIGGER

CREATE TRIGGER ステートメントは、データベースにトリガーを定義します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

トリガーを作成する際に、このステートメントの許可 ID には以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- BEFORE または AFTER トリガーを定義する表に対する ALTER 特権
- INSTEAD OF トリガーを定義するビューに対する CONTROL 特権
- INSTEAD OF トリガーを定義するビューの定義者
- トリガーを定義する表またはビューのスキーマに対する ALTERIN 特権

および以下のいずれか

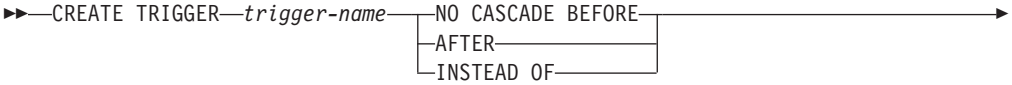
- データベースに対する IMPLICIT_SCHEMA 権限 (トリガーの暗黙または明示のスキーマ名が存在しない場合)
- スキーマに対する CREATEIN 特権 (トリガーのスキーマ名が既存のスキーマを指している場合)

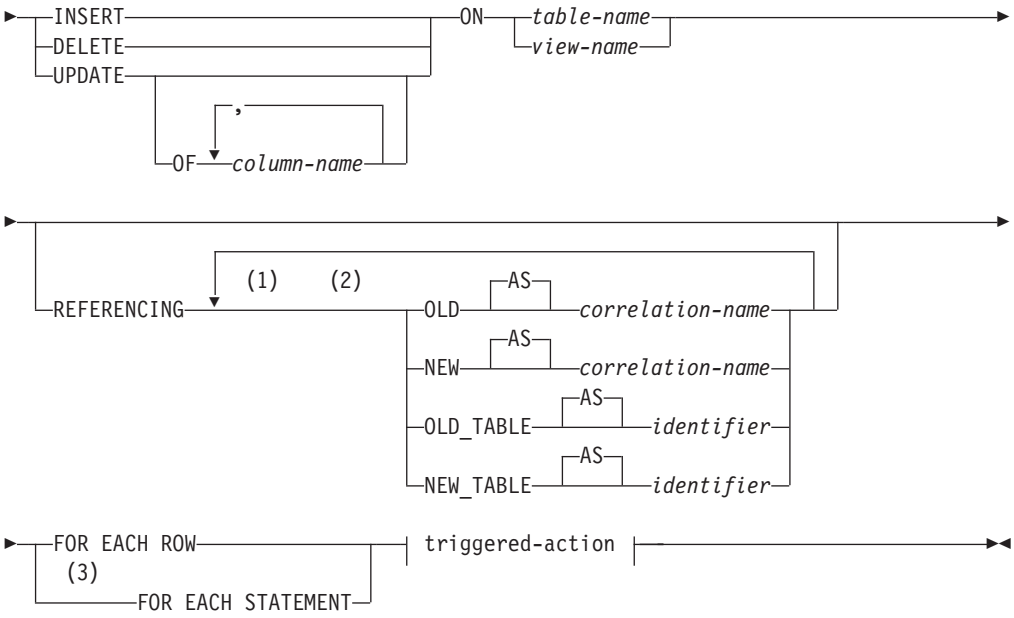
このステートメントの許可 ID に SYSADM 権限または DBADM 権限がない場合には、トリガーが存在する限り、ステートメントの許可 ID が持つ特権 (PUBLIC 特権またはグループ特権は考慮に入れない) に以下のすべてが含まれている必要があります。

- 遷移変数または遷移表を指定する場合は、トリガーを定義する表に対する SELECT 特権
- トリガー・アクション条件で参照される表またはビューに対する SELECT 特権
- 指定したトリガー SQL ステートメントを呼び出すために必要な特権

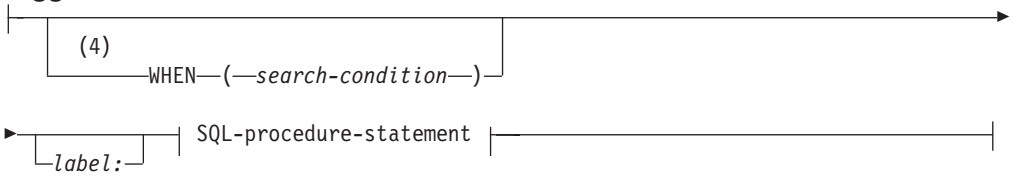
SYSADM 権限を所持しているために、トリガーの定義者がトリガーの作成しか行えないような場合、その定義者には、トリガーを作成できるようにする目的で明示的な DBADM 権限が付与されます。

構文:

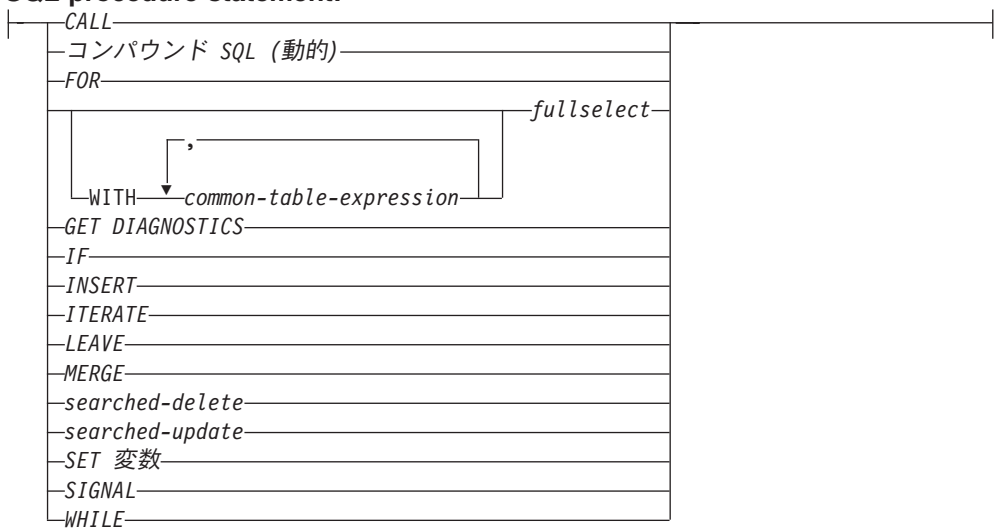




triggered-action:



SQL-procedure-statement:



注:

1 OLD および NEW は、それぞれ一度だけ指定できます。

2 OLD_TABLE および NEW_TABLE はそれぞれ一度だけ、AFTER トリガーまたは INSTEAD OF トリガーにのみ指定できます。

3 FOR EACH STATEMENT を BEFORE トリガーまたは INSTEAD OF トリガーに指定することはできません。

4 WHEN 条件を INSTEAD OF トリガーに指定することはできません。

説明:

trigger-name
トリガーの名前を指定します。暗黙のスキーマ名または明示スキーマ名を含む名前は、カタログにすでに記述されているトリガーを指定するものであってはなりません (SQLSTATE 42710)。 2 つの部分からなる名前を指定する場合、 'SYS' で始まるスキーマ名は使用できません (SQLSTATE 42939)。

NO CASCADE BEFORE
対象となる表の実際の更新によって生じる変更がデータベースに適用される前に、関連するトリガー・アクションを適用することを指定します。また、このトリガーのトリガー・アクションが、他のトリガーを活動化することがないことも指定します。

AFTER
対象となる表の実際の更新によって生じる変更がデータベースに適用された後で、関連するトリガー・アクションを適用することを指定します。

INSTEAD OF
関連したトリガー・アクションが、対象となるビューに対するアクションを置換することを指定します。 INSTEAD OF トリガーは、対象となる特定のビューに対する各操作ごとに 1 つだけ許可されます (SQLSTATE 428FP)。

INSERT
対象となる表または対象となるビューに INSERT 操作が適用される場合には必ず、このトリガーに関連するトリガー・アクションを実行することを指定します。

DELETE
対象となる表または対象となるビューに DELETE 操作が適用される場合には必ず、このトリガーに関連するトリガー・アクションを実行することを指定します。

UPDATE
指定した列または暗黙に指定される列に従って、対象となる表または対象となるビューに UPDATE 操作が適用される場合には必ず、このトリガーに関連するトリガー・アクションを実行することを指定します。

オプションの *column-name* のリストの指定がない場合、暗黙に表のすべての列が指定されます。 したがって、*column-name* リストを省略すると、表の列のいずれかの更新によってトリガーが活動化されることが暗に指定されます。

OF *column-name*,...
指定する各 *column-name* (列名) は、基本表の列でなければなりません (SQLSTATE 42703)。トリガーが BEFORE トリガーである場合、指定される *column-name* は、 ID 列以外の生成列にすることはできません (SQLSTATE 42989)。 *column-name* リストに 1 つの *column-name* を複数回指定することはできません (SQLSTATE 42711)。 トリガーは、*column-name*

リストに指定した列が更新される場合にのみ活動化されることになります。この文節は、INSTEAD OF トリガーには指定できません (SQLSTATE 42613)。

ON

table-name
BEFORE トリガーまたは AFTER トリガー定義の対象となる表を指定します。その名前は、基本表、または基本表の名前がわかる別名を指定しなければなりません (SQLSTATE 42704 または 42809)。この名前に、カタログ表 (SQLSTATE 42832)、マテリアライズ照会表 (SQLSTATE 42997)、宣言済み一時表 (SQLSTATE 42995)、あるいはニックネーム (SQLSTATE 42809) を指定することはできません。

view-name
INSTEAD OF トリガー定義の対象となるビューを指定します。その名前は、タイプなしビュー、またはタイプなしビューがわかる別名を指定しなければなりません (SQLSTATE 42704 または 42809)。指定する名前は、カタログ・ビューであってはなりません (SQLSTATE 42832)。指定する名前は、WITH CHECK OPTION (対称ビュー) を使用して定義されるビュー、または対称ビューが直接的または間接的に定義されたビューであってはなりません (SQLSTATE 428FQ)。

REFERENCING
遷移変数 の相関名と*遷移表* の表名を指定します。相関名には、トリガーとなる SQL 操作によって影響を受ける一連の行の中の特定の行を指定します。表名には、影響を受ける行の集合全体を指定します。トリガーとなる SQL 操作によって影響を受ける各行をトリガー・アクションで使用するには、次のようにして指定される *correlation-name* (相関名) によって列を修飾します。

OLD AS *correlation-name*
トリガーとなる SQL 操作の前の時点での行の状態を指定する相関名を指定します。

NEW AS *correlation-name*
トリガーとなる SQL 操作および、すでに実行された BEFORE トリガーの SET ステートメントによって、変更された時の行の状態を指定する相関名を指定します。

トリガーとなる SQL 操作によって影響を受ける行全体の集合をトリガー・アクションで使用するには、次のように指定される一時表名を使用します。

OLD_TABLE AS *identifier*
影響を受ける行の集合の、トリガーとなる SQL 操作の前のものを識別する一時表名を指定します。

NEW_TABLE AS *identifier*
トリガーとなる SQL 操作および、すでに実行された BEFORE トリガーの SET ステートメントによって、変更された行の状態を識別する一時表名を指定します。

REFERENCING 文節には、次の規則が適用されます。

- OLD および NEW の相関名と、OLD_TABLE および NEW_TABLE の名前は、いずれも同じであってはなりません (SQLSTATE 42712)。

- 1 つのトリガーには、*correlation-name* として、 1 つの OLD と 1 つの NEW だけしか指定できません (SQLSTATE 42613)。
- 1 つのトリガーには、 *identifier* として 1 つの OLD_TABLE と 1 つの NEW_TABLE しか指定できません (SQLSTATE 42613)。
- OLD *correlation-name* と OLD_TABLE *identifier* は、トリガー・イベントが DELETE 操作または UPDATE 操作のいずれかである場合にしか使用できません (SQLSTATE 42898)。操作が DELETE 操作の場合、OLD *correlation-name* は、削除された行の値を捕らえるものとなります。操作が UPDATE 操作の場合、その UPDATE 操作の前の時点での行の値を捕らえるものとなります。同じことが OLD_TABLE *identifier* とそれによって影響を受ける行の集合にも適用されます。
- NEW *correlation-name* と NEW_TABLE *identifier* は、トリガー・イベントが INSERT 操作または UPDATE 操作のいずれかである場合にしか使用できません (SQLSTATE 42898)。どちらの操作でも、NEW の値は、元の操作によって提供されたが、その時点までに実行された BEFORE トリガーによってさらに変更された、行の新しい状態を捕らえます。同じことが NEW_TABLE *identifier* とそれによって影響を受ける行の集合にも適用されます。
- OLD_TABLE *identifier* と NEW_TABLE *identifier* は、 BEFORE トリガーには定義できません (SQLSTATE 42898)。
- OLD *correlation-name* と NEW *correlation-name* は、 FOR EACH STATEMENT トリガーには定義できません (SQLSTATE 42899)。
- 遷移表は変更できません (SQLSTATE 42807)。
- トリガー・アクションの中での遷移表の列と遷移変数への参照の合計回数が、表内の列数の限界を超えてはなりません。また、その長さの合計が、表の中の行の最大長を超えてはなりません (SQLSTATE 54040)。
- 各 *correlation-name* と各 *identifier* の有効範囲は、トリガー定義全体です。

FOR EACH ROW
トリガーとなる SQL 操作によって影響を受ける対象となる表またはビューの各行ごとに、トリガー・アクションが一度ずつ適用されるよう指定します。

FOR EACH STATEMENT
トリガー・アクションが、ステートメント全体で一度だけ適用されることを指定します。このタイプのトリガー精度は、BEFORE トリガーまたは INSTEAD OF トリガーには指定できません (SQLSTATE 42613)。指定すると、トリガーとなる UPDATE または DELETE ステートメントによって影響を受ける行がない場合でも、 UPDATE トリガーまたは DELETE トリガーが活動化されることになってしまいます。

triggered-action
トリガーを活動化するときに実行されるアクションを指定します。
triggered-action は *SQL-procedure-statement*、および *SQL-procedure-statement* 実行のオプション条件から構成されています。

WHEN (*search-condition***)**
真、偽、または不明である条件を指定します。 *search-condition* によって、あるトリガー処置を実行すべきかどうかを決定する機能が与えられます。

関連するアクションは、指定された検索条件が真である場合のみ実行されます。 WHEN 文節が省略されると、関連する *SQL-procedure statement* が常に実行されます。

WHEN 文節を INSTEAD OF トリガーに指定することはできません (SQLSTATE 42613)。

label:

SQL プロシージャー・ステートメントのラベルを指定します。ラベルは、リスト内でネストされたコンパウンド・ステートメントを含め、SQL プロシージャー・ステートメントのリスト内でユニークでなければなりません。ネストされていないコンパウンド・ステートメントは、同じラベルを使用できることに注意してください。 SQL プロシージャー・ステートメントのリストは、おそらく SQL 制御ステートメントの中にあります。

FOR ステートメント、WHILE ステートメント、および動的コンパウンド・ステートメントだけにラベルを組み込むことができます。

SQL-procedure-statement

トリガー・アクションの一部にする SQL ステートメントを指定します。コンパウンド SQL 内のニックネームに対する検索更新、検索削除、挿入、またはマージ操作はサポートされません。

SQL-procedure-statement には、サポートされていないステートメントを入れることはできません (SQLSTATE 42987)。

SQL-procedure-statement は、未定義の遷移変数 (SQLSTATE 42703)、フェデレーテッド・オブジェクト (SQLSTATE 42997)、または宣言済み一時表 (SQLSTATE 42995) を参照できません。

BEFORE トリガー内の *SQL-procedure-statement* には、以下の制限があります。

- MODIFIES SQL DATA で定義されたプロシージャーを呼び出す CALL ステートメント、または MERGE ステートメントにすることができません (SQLSTATE 42987)。
- REFRESH IMMEDIATE で定義されたマテリアライズ照会表を参照できません (SQLSTATE 42997)。
- NEW 遷移変数内の ID 列以外の生成された列を参照できません (SQLSTATE 42989)。

注:

- すでに行が含まれている表にトリガーを追加しても、トリガー・アクションは活動化されません。そのため、トリガーが表内のデータに制約を適用するように設計されている場合、既存の行についてはそれらの制約が満たされない可能性があります。
- 2 つのトリガーのイベントが同時に発生する場合 (たとえばイベント、活動化のタイミング、および対象の表が同じである場合)、最初に作成されたトリガーが最初に実行されます。
- トリガーの定義後に対象の表に列が追加された場合、次の規則が適用されます。
  - トリガーが、明示的な列リストなしで指定された UPDATE トリガーである場合、新しい列への更新によってトリガーが活動化されます。

  - その列は、それ以前に定義されたトリガーのトリガー・アクションからは見えません。
  - OLD_TABLE および NEW_TABLE の各変位表に、この列は含まれません。したがって、遷移表に対して "SELECT *" を実行しても、追加列は含められません。
- トリガー・アクションで参照される表に 1 つの列を追加した場合、新しい列はそのトリガー・アクションからは見えません。
- *SQL-procedure-statement* に指定されている全選択の結果は、トリガーの内部または外部では使用不可です。
- トリガー・コンパウンド・ステートメント内で呼び出されるプロシージャーは、COMMIT または ROLLBACK ステートメントを発行できません (SQLSTATE 42985)。
- 検索 UPDATE ステートメント、検索 DELETE ステートメント、または検索 INSERT ステートメント内のニックネームへの参照を含むプロシージャーはサポートされていません (SQLSTATE 25000)。
- ***表アクセスの制限:***

  プロシージャーが READS SQL DATA または MODIFIES SQL DATA として定義されている場合は、プロシージャー内のステートメントは、このプロシージャーを呼び出したコンパウンド・ステートメントによって変更される表にアクセスすることはできません (SQLSTATE 57053)。プロシージャーが MODIFIES SQL DATA として定義されている場合は、プロシージャー内のステートメントは、このプロシージャーを呼び出したコンパウンド・ステートメントによって読み取られるまたは変更される表を変更できません (SQLSTATE 57053)。
- カスケードされた参照制約のサイクルに関係のある表に対して定義された BEFORE DELETE トリガーは、そのトリガーが定義されている表への参照や、参照保全制約のサイクルの評価中にカスケードされて変更された他の表への参照には含めないでください。そのようなトリガーを含めると、結果がデータによってまちまちになり、一貫性のない状態が生じてしまう可能性があります。

  このようなトリガーを最も簡明な形式にする方法は、自己参照の参照制約のある表に対する BEFORE DELETE トリガーや CASCADE の削除規則に、*triggered-action* に関係のある表への参照を含めないようにすることです。
- トリガーを作成すると、特定のパッケージは無効として扱われるようになります。
  - 明示的な列リストなしの UPDATE トリガーを作成した場合、ターゲット表またはビューに対して更新操作を使用するパッケージは無効になります。
  - 列リストを指定した UPDATE トリガーを作成した場合、ターゲット表に対して更新操作を使用するパッケージは、そのパッケージにおいて、 CREATE TRIGGER ステートメントの *column-name* リストの中の少なくとも 1 つの列に対しても更新を使用する場合にのみ無効になります。
  - INSERT トリガーを作成した場合、ターゲット表またはビューに対して挿入操作を使用するパッケージは無効になります。
  - 削除トリガーを作成した場合、ターゲット表またはビューに対して削除操作を使用するパッケージは無効になります。

- パッケージは、アプリケーション・プログラムが明示的にバインドまたは再バインドされるまで、あるいはアプリケーション・プログラムが実行されてデータベース・マネージャーが自動的にそれを再バインドするまで、無効のままになります。
- ***作動不能トリガー***: *作動不能トリガー* は、使用可能でなくなったために活動化されないトリガーです。以下の場合、トリガーは操作不能になります。
  - トリガーを実行するため、そのトリガーの作成者が持っていなければならない特権が取り消された。
  - トリガーが定義された、表、ビュー、または別名といったオブジェクトがドロップされた。
  - トリガーが定義されたビューが操作不能になった。
  - トリガーのサブジェクト表である別名がドロップされた。

  実際、操作不能トリガーは、DROP または REVOKE ステートメントのカスケード規則の結果、トリガー定義がドロップされたトリガーです。たとえば、ビューがドロップされると、そのビューを使用して *SQL-procedure-statement* が定義されているトリガーが操作不能になります。

  トリガーが操作不能になると、そのトリガーを活動化していた操作を実行するステートメントを持つパッケージはすべて無効とマークされます。パッケージが (明示的または暗黙的に) 再バインドされると、操作不能トリガーは完全に無視されます。同様に、トリガーを活動化していた操作を実行する動的 SQL ステートメントを含むアプリケーションも、作動不能トリガーを完全に無視します。

  トリガー名は、DROP TRIGGER および COMMENT ON TRIGGER の各ステートメントにも指定できます。

  操作不能トリガーは、その定義テキストを使用して CREATE TRIGGER ステートメントを出すことによって再作成できます。このトリガー定義テキストは、SYSCAT.TRIGGERS カタログ・ビューの TEXT 列に保管されています。操作不能トリガーを再作成するため、そのトリガーを明示的にドロップする必要はありません。操作不能トリガーとして同じ *trigger-name* で CREATE TRIGGER ステートメントを出すと、警告とともに、その操作不能トリガーは置換されます (SQLSTATE 01595)。

  作動不能トリガーであることは、 SYSCAT.TRIGGERS カタログ・ビューの VALID 列が X であることによって示されます。
- ***トリガー実行中のエラー***: triggered-SQL-statements の実行時に発生したエラーは、エラーが重大であると見なされた場合以外は SQLSTATE 09000 を用いて戻されます。重大エラーであれば、重大エラー SQLSTATE が返されます。重大エラーでない場合、SQLCA の SQLERRMC フィールドには、トリガー名、SQLCODE、 SQLSTATE、および障害のあるトークンから入るだけの数のトークンが組み込まれます。

  SQL-procedure-statement には SIGNAL SQLSTATE ステートメントまたは RAISE_ERROR 関数が組み込まれていることがあります。どちらの場合も、返される SQLSTATE は、SIGNAL SQLSTATE ステートメントまたは RAISE_ERROR 条件に指定されているものです。

- まだ存在していないスキーマ名を用いてトリガーを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- データベース・マネージャーが ID 列用に生成する値は、どの BEFORE トリガーの実行よりも前に生成されます。したがって、生成される識別値は BEFORE トリガーにとって可視の値です。
- 生成された式列用にデータベース・マネージャーが生成する値は、どの BEFORE トリガーの実行よりも後に生成されます。したがって、その式で生成される値は、BEFORE トリガーにとって可視でない値です。
- ***トリガーおよび型付き表***: 表階層のどのレベルの型付き表にも、BEFORE または AFTER トリガーを付加することができます。 SQL ステートメントが複数のトリガーを活動化する場合、それらのトリガーは、それぞれ型付き表階層の別々の表に付加されていても、作成順に実行されます。

  トリガーが活動化されたとき、その遷移変数 (OLD、NEW、OLD_TABLE、および NEW_TABLE) 内に副表の行が入っていることがあります。ただし、付加先の表で定義されている列しか入っていません。

  INSERT、UPDATE、および DELETE ステートメントの効果は次のとおりです。

  – 行トリガー: SQL ステートメントを使って表行の INSERT、UPDATE、または DELETE を行うと、このステートメントは、行の入った最も限定的な表とその表のすべてのスーパー表に付加されている行トリガーを活動化します。 SQL ステートメントがどのように表にアクセスするかに関係なく、常にこのようになります。たとえば、UPDATE EMP コマンドを実行すると、更新済みの行の一部が、副表 MGR に入ることがあります。 EMP 行の場合、EMP とそのスーパー表に付加されている行トリガーが活動化されます。 MGR 行の場合、MGR とそのスーパー表に付加されている行トリガーが活動化されます。

  – ステートメント・トリガー: INSERT、UPDATE、または DELETE ステートメントは、このステートメントによって影響を受ける可能性のある表 (およびそのスーパー表) に付加されているステートメント・トリガーを活動化します。そのような表内の実際の行が影響を受けたかどうかに関係なく、常にそのようになります。たとえば、INSERT INTO EMP コマンドで、 EMP とそのスーパー表のステートメント・トリガーを活動化します。別の例として、副表行が更新も削除もされていない場合でも、 UPDATE EMP または DELETE EMP コマンドで、 EMP とそのスーパー表のステートメント・トリガーが活動化されます。同様に、UPDATE ONLY (EMP) または DELETE ONLY (EMP) コマンドは、 EMP とそのスーパー表のステートメント・トリガーを活動化しますが、副表のステートメント・トリガーは活動化しません。

  DROP TABLE ステートメントの効果: DROP TABLE ステートメントは、ドロップしようとしている表に付加されているどのトリガーも活動化しません。ただし、ドロップされる表が従属表である場合、そのドロップされる表の行すべては、スーパー表から削除されるものと見なされます。したがって、表 T の場合は次のようになります。

  – 行トリガー: DROP TABLE T は、T の行ごとに、 T のすべてのスーパー表に付加されている行タイプの削除トリガーを活動化します。

– ステートメント・トリガー: DROP TABLE T は、T に行が入っているかどうかに関係なく、 T のすべてのスーパー表に付加されているステートメント・タイプの削除トリガーを活動化します。

ビューでのアクション: ビューでのアクションによってどのトリガーが活動化されるかを予測するには、ビュー定義を使ってそのアクションを、基本表でのアクションに変換します。たとえば:

1. SQL ステートメントで UPDATE V1 を実行します。 V1 は、サブビュー V2 をもつ型付きビューです。 V1 は基礎表 T1 をもち、V2 は基礎表 T2 をもっているとします。ステートメントは、T1、T2、およびそれらの副表内の行に影響を与える可能性があるので、 T1 と T2 およびそのすべての副表とスーパー表のステートメント・トリガーが活動化されます。
2. SQL ステートメントで UPDATE V1 を実行します。 V1 は、サブビュー V2 をもつ型付きビューです。 V1 は SELECT ... FROM ONLY(T1) と定義されていて、 V2 は SELECT ... FROM ONLY(T2) と定義されていると仮定します。ステートメントは、T1 と T2 の副表内の行には影響を与えないので、T1 と T2 およびそれぞれのスーパー表のステートメント・トリガーは活動化されますが、これらの表の副表のものは活動化されません。
3. SQL ステートメントで UPDATE ONLY(V1) を実行します。 V1 は、SELECT ... FROM T1 と定義された型付きビューです。ステートメントは、T1 とその副表に影響を与える可能性があります。したがって、T1 とそのすべての副表とスーパー表のステートメント・トリガーが活動化されます。
4. SQL ステートメントで UPDATE ONLY(V1) を実行します。 V1 は、SELECT ... FROM ONLY(T1) と定義された型付きビューです。この場合、V1 がサブビューをもち T1 が副表をもっていても、 T1 だけがステートメントから影響を受けることができます。したがって、T1 とそのスーパー表のステートメント・トリガーが活動化されます。

- ***MERGE ステートメントおよびトリガー***: MERGE ステートメントは、更新、削除、および挿入操作を実行できます。 MERGE ステートメントの適用可能な UPDATE、DELETE、または INSERT トリガーは、更新、削除、または挿入操作の実行時に活動化されます。

例:

例 1: 会社が管理する従業員の数の自動追跡を実行する 2 つのトリガーを作成します。このトリガーは、次の表に作用します。

EMPLOYEE 表 (列は ID、NAME、ADDRESS、および POSITION)
COMPANY_STATS 表 (列は NBEMP、NBPRODUCT、および REVENUE)

最初のトリガーは、新しい従業員を採用するたびに (つまり EMPLOYEE 表に新しい行が挿入されるたびに)、従業員数に 1 を加算します。

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

2 番目のトリガーは、従業員が会社を退職するたびに (つまり EMPLOYEE 表から行が削除されるたびに)、従業員数から 1 を減算します。

```
CREATE TRIGGER FORMER_EMP
  AFTER DELETE ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP - 1
```

例 2: 部品のレコードが更新されると、以下の検査と (必要ならば) アクションを実行するトリガーを作成します。

手持ち数量 (ON_HAND) が最大在庫量 (MAX_STOCKED) の 10% 未満になった場合、その部品の品目数として最大在庫量から手持ち数量を引いた数を指定した出荷依頼書を発行します。

このトリガーは、PARTNO、DESCRIPTION、ON_HAND、MAX_STOCKED、および PRICE の列を含む PARTS 表に作用します。

ISSUE_SHIP_REQUEST は、追加部品の注文書を、発注先に送るユーザー定義関数です。

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (N.ON_HAND < 0.10 * N.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N.MAX_STOCKED - N.ON_HAND, N.PARTNO));
  END
```

例 3: 更新の結果、現行の給与の 10 % を超える昇給になった場合にエラーを生じさせるトリガーを作成します。

```
CREATE TRIGGER RAISE_LIMIT
  AFTER UPDATE OF SALARY ON EMPLOYEE
  REFERENCING NEW AS N OLD AS O
  FOR EACH ROW
  WHEN (N.SALARY > 1.1 * O.SALARY)
         SIGNAL SQLSTATE '75000' SET MESSAGE_TEXT='Salary increase>10%'
```

例 4: 株価の変更を記録し追跡するアプリケーションについて考えます。データベースには、CURRENTQUOTE および QUOTEHISTORY という 2 つの表が含まれています。

```
Tables: CURRENTQUOTE (SYMBOL, QUOTE, STATUS)
        QUOTEHISTORY (SYMBOL, QUOTE, QUOTE_TIMESTAMP)
```

CURRENTQUOTE の QUOTE (相場) 列が更新されると、新しい相場とタイム・スタンプを QUOTEHISTORY 表にコピーするようにします。 CURRENTQUOTE の STATUS (状況) 列も、次のような株の状況が反映されるように更新します。

1. 値上がり
2. 今年の新高値
3. 値下がり
4. 今年の新安値
5. 変わらず

これを実現する CREATE TRIGGER ステートメントは、次のようになります。

- 状況を設定するトリガーの定義

```
CREATE TRIGGER STOCK_STATUS
  NO CASCADE BEFORE UPDATE OF QUOTE ON CURRENTQUOTE
  REFERENCING NEW AS NEWQUOTE OLD AS OLDQUOTE
  FOR EACH ROW
  BEGIN ATOMIC
     SET NEWQUOTE.STATUS =
       CASE
          WHEN NEWQUOTE.QUOTE >
                (SELECT MAX(QUOTE) FROM QUOTEHISTORY
                WHERE SYMBOL = NEWQUOTE.SYMBOL
                AND YEAR(QUOTE_TIMESTAMP) = YEAR(CURRENT DATE) )
             THEN 'High'
          WHEN NEWQUOTE.QUOTE <
(SELECT MIN(QUOTE) FROM QUOTEHISTORY
                WHERE SYMBOL = NEWQUOTE.SYMBOL
                AND YEAR(QUOTE_TIMESTAMP) = YEAR(CURRENT DATE) )
             THEN 'Low'
          WHEN NEWQUOTE.QUOTE > OLDQUOTE.QUOTE
             THEN 'Rising'
          WHEN NEWQUOTE.QUOTE < OLDQUOTE.QUOTE
             THEN 'Dropping'
          WHEN NEWQUOTE.QUOTE = OLDQUOTE.QUOTE
             THEN 'Steady'
       END;
  END
```

- 変更内容を QUOTEHISTORY 表に記録するトリガーの定義

```
CREATE TRIGGER RECORD_HISTORY
  AFTER UPDATE OF QUOTE ON CURRENTQUOTE
  REFERENCING NEW AS NEWQUOTE
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO QUOTEHISTORY
      VALUES (NEWQUOTE.SYMBOL, NEWQUOTE.QUOTE, CURRENT TIMESTAMP);
  END
```

関連資料:

- 131 ページの『コンパウンド SQL (動的)』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『trigsql.sqb -- How to use a trigger on a table (MF COBOL)』
- 『TbTrig.java -- How to use triggers (JDBC)』
- 『TbTrig.sqlj -- How to use triggers (SQLj)』

CREATE TYPE (構造化)

CREATE TYPE ステートメントは、ユーザー定義の構造化タイプを定義します。ユーザー定義構造化タイプには、属性を含めないこともできますし、複数の属性も含めることもできます。構造化タイプには、スーパータイプからの属性を継承するサブタイプを指定することができます。ステートメントの実行が正常に完了すると、属性値の検索と更新のためのメソッドが生成されます。また、このステートメントの実行が正常に完了すると、列内で使用する構造化タイプのインスタンスを作成する関数と、該当の参照タイプとその表示タイプとをキャストする関数、およびその参照タイプ上の比較演算子 (=、<>、<、<=、>、および >=) をサポートする関数も生成されます。

また、CREATE TYPE ステートメントは、ユーザー定義構造化タイプと一緒に使用されるユーザー定義メソッドの任意のメソッド仕様も定義します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

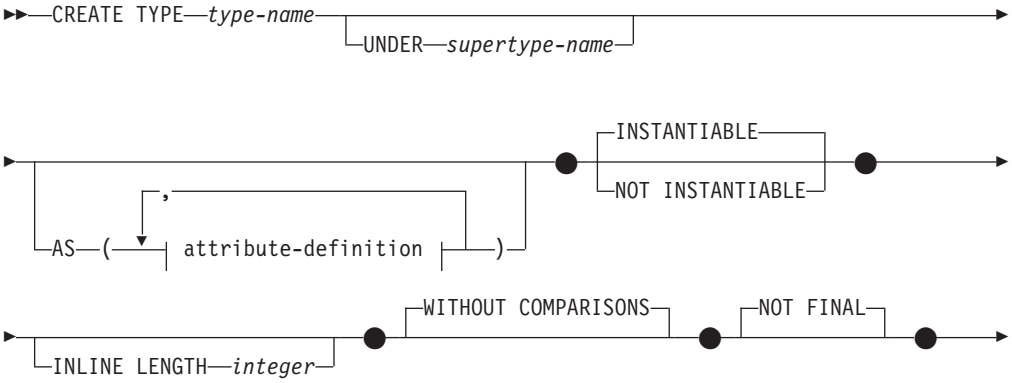
許可:

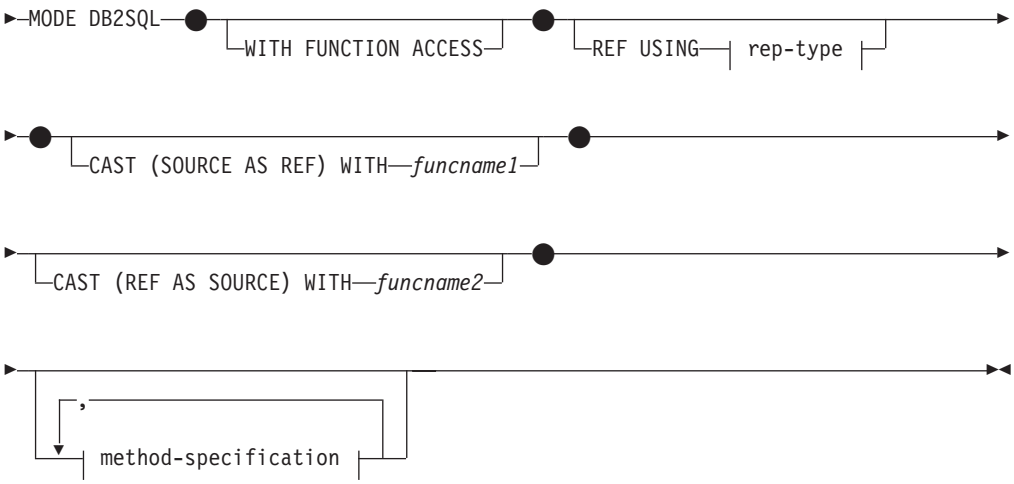
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (このタイプのスキーマ名が既存のスキーマを指していない場合)
- スキーマに対する CREATEIN 特権 (このタイプのスキーマ名が既存のスキーマを指している場合)

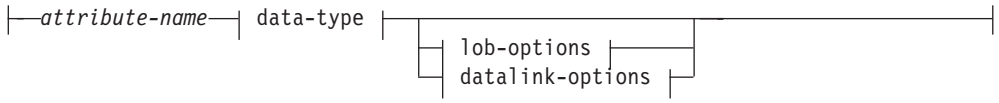
UNDER が指定されていて、このステートメントの許可 ID がタイプ階層のルート・タイプの定義者と同じではない場合には、 SYSADM または DBADM 権限が必要です。

構文:

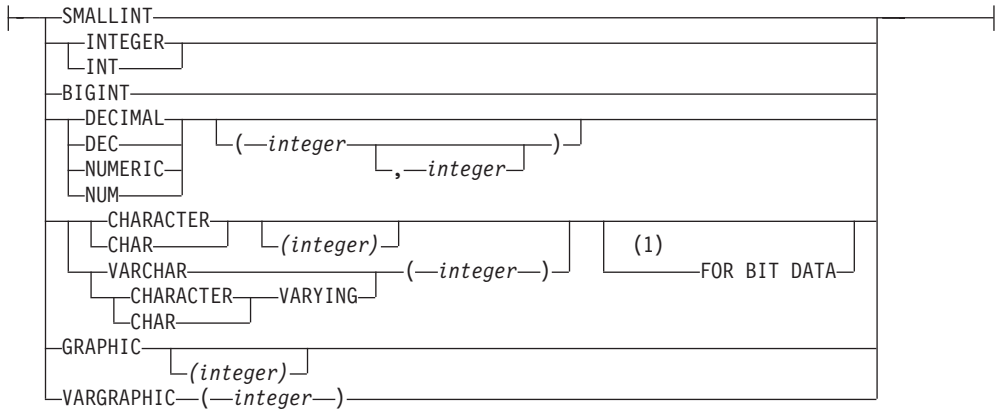




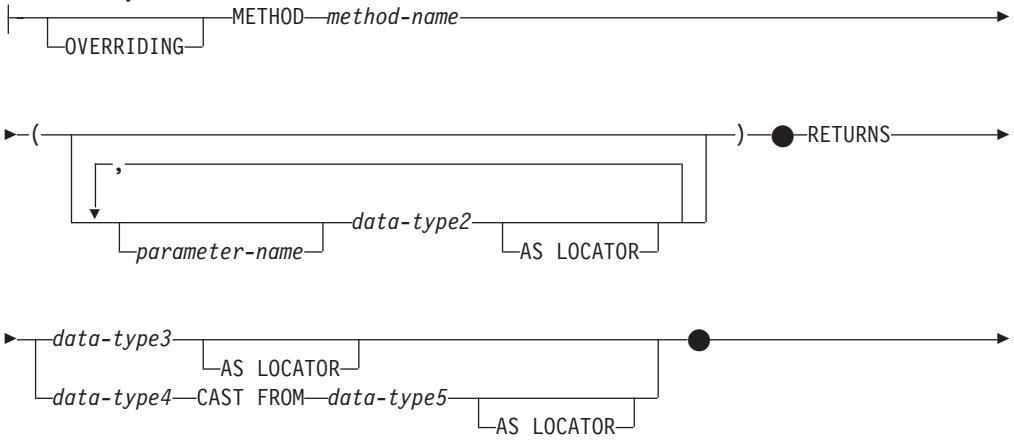
attribute-definition:

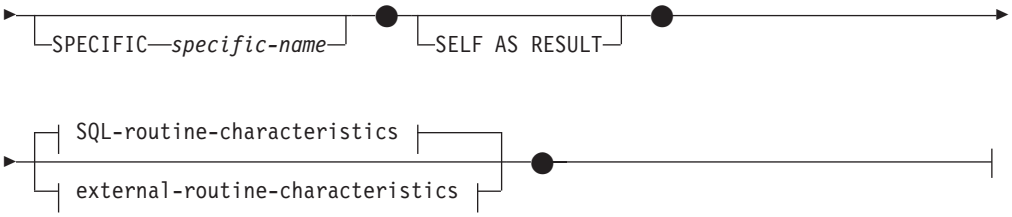


rep-type:

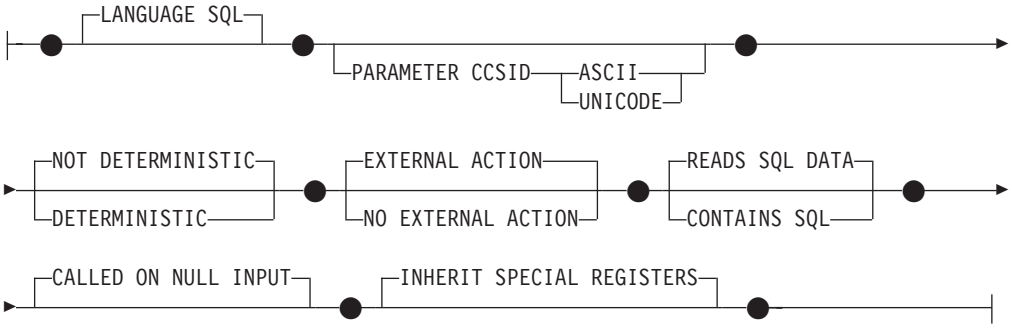


method-specification:

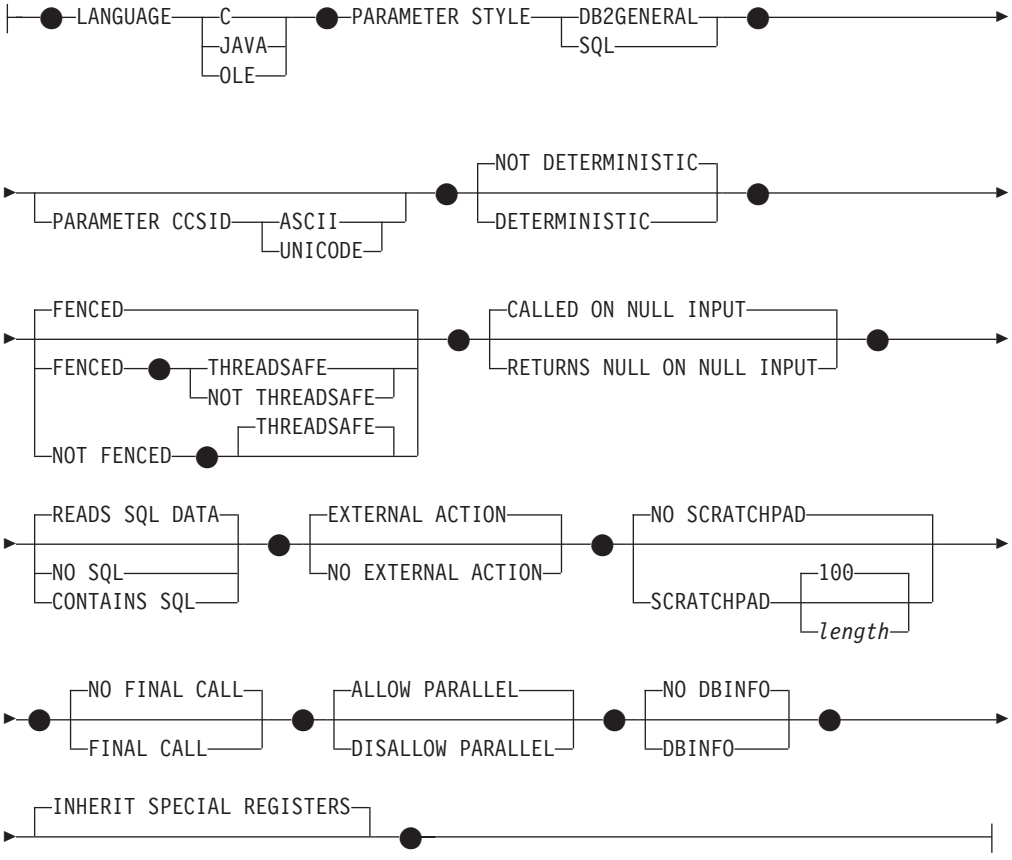




SQL-routine-characteristics:



external-routine-characteristics:



注:

1 FOR BIT DATA 文節とその後に続く他の列制約とは、任意の順序で指定できます。

説明:

type-name
タイプの名前を指定します。名前 (暗黙または明示の修飾子を含む) は、カタログに既に記述されているその他のタイプ (組み込みタイプ、構造タイプ、特殊タイプを含む) と同じであってはなりません。 非修飾名は、組み込みデータ・タイプ名と同一のものまたは BOOLEAN であってはなりません (SQLSTATE 42918)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

スキーマ名 (明示指定または暗黙指定) は、8 バイト以下でなければなりません (SQLSTATE 42622)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *type-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

2 つの部分から成る *type-name* を指定する場合、スキーマ名を 'SYS' で始めることはできません。違反すると、エラーが戻されます (SQLSTATE 42939)。

UNDER *supertype-name*
この構造タイプが指定した *supertype-name* のサブタイプであることを指定します。 *supertype-name* は既存の構造タイプを指定する必要があります (SQLSTATE 42704)。 *supertype-name* がスキーマ名なしで指定される場合、SQL パス上でスキーマを検索することにより、タイプは解決されます。構造タイプには、スーパータイプの属性すべてと、それに続く *attribute-definition* の追加属性が含められます。

attribute-definition
構造タイプの属性を定義します。

attribute-name
属性の名前です。この構造タイプのその他の属性またはスーパータイプと同じ *attribute-name* を付けることはできません (SQLSTATE 42711)。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *attribute-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

data-type
属性のデータ・タイプです。 これは、『CREATE TABLE』 でリストされているデータ・タイプの 1 つで、 LONG VARCHAR、LONG VARGRAPHIC、または LONG VARCHAR や LONG VARGRAPHIC に基づいた特殊タイプ以外のものです (SQLSTATE 42601)。このデータ・タイプには既存のデータ・タイプを指定する必要があります (SQLSTATE 42704)。 *data-type* がスキーマ名なしで指定される場合、 SQL パス上でスキーマを検索することにより、タイプは解決されます。 『CREATE TABLE』 に種々のデータ・タイプの説明が記載されています。属性データ・タイプが参照タイプである場合、参照するターゲット・タイプはこのステートメントに既に

存在する構造タイプであるか、またはこのステートメントで作成されたものでなければなりません (SQLSTATE 42704)。

タイプ DATALINK の属性を使って定義された構造タイプは、型付き表または型付きビューのデータ・タイプとしてのみ有効に使用することができます (SQLSTATE 01641)。

ランタイムに、該当タイプのインスタンスが、同一タイプまたはそのサブタイプの別のインスタンスを直接または間接に取り込むことを許容するタイプ定義を防止するため、その属性のいずれかが、自身を直接または間接に使用する仕方でタイプを定義することはできません (SQLSTATE 428EP)。

lob-options
LOB タイプと関連したオプション (あるいは LOB に基づく特殊タイプ) を指定します。 *lob-options* の詳細については、『CREATE TABLE』 を参照してください。

datalink-options
DATALINK タイプと関連したオプション (あるいは DATALINK タイプに基づく特殊タイプ) を指定します。 *datalink-options* の詳細については、『CREATE TABLE』 を参照してください。

DATALINK タイプまたは DATALINK に基づいている特殊タイプでオプションが指定されないと、 LINKTYPE URL および NO LINK CONTROL オプションがデフォルト値になることに注目してください。

INSTANTIABLE または **NOT INSTANTIABLE**
構造タイプのインスタンスを作成できるかどうかを指定します。インスタンス化不能な構造タイプとは、以下のような意味です。

- インスタンス化不能タイプには、コンストラクター関数が生成されない
- インスタンス化不能タイプは、表またはビューのタイプとして使用することができない (SQLSTATE 428DP)
- インスタンス化不能タイプは、列のタイプとして使用することができる (その列には、 NULL 値またはインスタンス化可能なサブタイプのインスタンスだけを挿入することができる)

インスタンス化不能タイプのインスタンスを作成するには、インスタンス化可能サブタイプを作成する必要があります。 NOT INSTANTIABLE を指定すると、新規のタイプのインスタンスを作成できなくなります。

INLINE LENGTH *integer*
このオプションは、表の行内の残りの値とインラインで保管する構造タイプ列のインスタンスの最大サイズ (バイト数) を指示します。指定したインライン長よりも長い構造タイプまたはそのサブタイプのインスタンスは、 LOB 値が処理されるのと同様の方法で、基本表の行とは別に保管されます。

指定した INLINE LENGTH が、新たに作成したタイプのコンストラクター関数の結果サイズよりも小さく (32 バイトに、属性ごとに 10 バイトを加算したもの)、しかも 292 バイトより小さいと、エラーが生じます (SQLSTATE 429B2)。属性数には、タイプのスーパータイプから継承されたすべての属性が含まれることに注意してください。

タイプの INLINE LENGTH は、指定値またはデフォルト値のどちらであっても、構造タイプを使用する列のデフォルトのインライン長になります。このデフォルトは、CREATE TABLE 時にオーバーライドすることができます。

型付き表のタイプとして構造タイプを使用すると、INLINE LENGTH には何の意味もなくなります。

構造タイプのデフォルトの INLINE LENGTH はシステムによって計算されます。この後に示す公式では、以下のような用語を使います。

短い属性 (short attribute)
SMALLINT、 INTEGER、 BIGINT、 REAL、 DOUBLE、 FLOAT、 DATE、または TIME のデータ・タイプのいずれかをもつ属性を指します。さらに、これらのタイプに基づいた特殊タイプまたは参照タイプも含まれます。

短くない属性 (non-short attribute)
残りのデータ・タイプのいずれか、またはこれらのデータ・タイプに基づく特殊タイプの属性を指します。

システムは、次のようにデフォルトのインライン長を計算します。

1. 以下のような公式を使って、短くない属性の追加スペース所要量を割り出します。

   space_for_non_short_attributes = SUM(*attributelength* + n)

   n は以下のように定義されます。
   - ネストされた構造タイプの属性には 0 バイト
   - 非 LOB 属性には 2 バイト
   - LOB 属性には 9 バイト

   attributelength は、表 9 に示すとおり、属性に指定されているデータ・タイプに基づく値です。
2. 以下のような公式を使って、デフォルトの合計インライン長を計算します。

   default_length(*structured_type*) = (*number_of_attributes* * 10) + 32 + *space_for_non-short_attributes*

   number_of_attributes は、スーパータイプから継承される属性も含めた構造タイプの合計属性数です。ただし、*number_of_attributes* には、 *structured_type* の任意のサブタイプに定義されているどの属性も含まれません。

表 9. 属性データ・タイプのバイト・カウント

属性データ・タイプ	バイト・カウント
DECIMAL	(p/2)+1 の整数部分 (p は精度)
CHAR (n)	n
VARCHAR (n)	n
GRAPHIC (n)	n * 2
VARGRAPHIC (n)	n * 2
TIMESTAMP	10

表 9. 属性データ・タイプのバイト・カウント (続き)

属性データ・タイプ	バイト・カウント	
DATALINK(n)	n + 54	
LOB タイプ	各 LOB 属性は、構造タイプ・インスタンス内に、実際の値の位置へのポインターとなる LOB 記述子を持っています。その記述子のサイズは、その LOB 属性に定義されている最大長によって異なります。	
	LOB の最大長	LOB 記述子のサイズ
	1 024	72
	8 192	96
	65 536	120
	524 000	144
	4 190 000	168
	134 000 000	200
	536 000 000	224
	1 070 000 000	256
	1 470 000 000	280
	2 147 483 647	316
特殊タイプ	特殊タイプのソース・タイプの長さ	
参照タイプ	参照タイプの基礎となる組み込みデータ・タイプの長さ	
構造タイプ	inline_length(*attribute_type*)	

WITHOUT COMPARISONS
構造タイプのインスタンスで比較関数がサポートされていないことを示します。

NOT FINAL
この構造タイプをスーパータイプとして使用できることを示します。

MODE DB2SQL
この文節は必須であり、このタイプでコンストラクター関数を直接呼び出すために使用します。

WITH FUNCTION ACCESS
将来作成されるメソッドを含め、該当タイプとそのサブタイプのすべてのメソッドに対して、関数表記を使ってアクセスできることを指示します。この文節を指定できるのは、 UNDER 文節が指定されていない構造タイプの階層のルート・タイプだけです (SQLSTATE 42613)。この文節は、メソッドを呼び出す表記よりもこの形式の表記のほうが望ましいアプリケーションで、関数表記を使用できるようにするために提供されています。

REF USING *rep-type*
この構造タイプの参照タイプの表示 (基礎データ・タイプ) として使用される組み込みデータ・タイプとそのサブタイプをすべて定義します。 この文節を指定できるのは、 UNDER 文節が指定されていない構造タイプの階層のルート・タイプだけです (SQLSTATE 42613)。 *rep-type* は、 LONG VARCHAR、 LONG VARGRAPHIC、 BLOB、 CLOB、 DBCLOB、 DATALINK、または構造タイプであってはならず、 32 672 バイト以下の長さでなければなりません (SQLSTATE 42613)。

構造タイプの階層のルート・タイプにこの文節を指定しない場合、 REF USING VARCHAR(16) FOR BIT DATA が想定されます。

CAST (SOURCE AS REF) WITH *funcname1*
システムにより生成される関数で、データ・タイプ *rep-type* の値を、この構造タイプの参照タイプにキャストする関数の名前を定義します。 *funcname1* の一部としてスキーマ名を指定することはできません (SQLSTATE 42601)。 cast 関数は、構造タイプと同じスキーマ内で生成されます。この文節を指定しない場合、 *funcname1* のデフォルト値は *type-name* (構造タイプの名前) になります。 *funcname1(rep-type)* に一致する関数シグニチャーが、同じスキーマ内に存在してはなりません (SQLSTATE 42710)。

CAST (REF AS SOURCE) WITH *funcname2*
システムにより生成される関数で、この構造タイプの参照タイプ値を、データ・タイプ *rep-type* にキャストする関数の名前を定義します。 *funcname2* の一部としてスキーマ名を指定することはできません (SQLSTATE 42601)。 cast 関数は、構造タイプと同じスキーマ内で生成されます。この文節を指定しない場合、 *funcname2* のデフォルト値は *rep-type* (表示タイプの名前) になります。

method-specification
このタイプのメソッドを定義します。メソッドは、CREATE METHOD ステートメントで本体を与えられて初めて、実際に使用できるようになります (SQLSTATE 42884)。

OVERRIDING
定義するメソッドが、定義するタイプのスーパータイプのメソッドをオーバーライドすることを指定します。オーバーライドすることによって、サブタイプのメソッドを再インプリメントできるようになるので、より具体的な機能が提供されます。オーバーライドは、以下のタイプのメソッドではサポートされません。

- 表メソッドおよび行メソッド
- PARAMETER STYLE JAVA を使用して宣言される外部メソッド
- 索引拡張で述部として使用できるメソッド
- システムによって生成される mutator メソッドまたは observer メソッド

このようなメソッドをオーバーライド使用とすると、エラーになります (SQLSTATE 42745)。

メソッドを有効なオーバーライド・メソッドにする場合は、定義するタイプの適切なスーパータイプの 1 つに元のメソッドが 1 つ存在していなければならず、オーバーライド・メソッドと元のメソッドの間に以下の関係が存在している必要があります。

- 定義するメソッドと元のメソッドのメソッド名が同じである。
- 定義するメソッドと元のメソッドのパラメーターの数が同じである。
- 定義するメソッドの各パラメーターのデータ・タイプと、元のメソッドの対応するパラメーターのデータ・タイプが同一である。この要件では、暗黙の SELF パラメーターは考慮されません。

このような元のメソッドが存在しない場合は、エラーが戻されます (SQLSTATE 428FV)。

オーバーライド・メソッドは、以下の属性を元のメソッドから継承します。

- 言語
- 決定性の指示
- 外部アクションの指示
- 引き数に NULL 値がある場合にメソッドを呼び出すかどうかの指示
- 結果のキャスト (元のメソッドで指定されている場合)
- SELF AS RESULT の指示
- SQL データ・アクセスまたは CONTAINS SQL の指示
- 外部メソッドの場合は、以下のとおりです。
  - パラメーターのスタイル
  - パラメーターと結果のロケーターの指示 (元のメソッドで指定されている場合)
  - FENCED、SCRATCHPAD、FINAL CALL、ALLOW PARALLEL、および DBINFO の指示
  - INHERIT SPECIAL REGISTER および THREADSAFE の指示

method-name
定義しようとするメソッドを指定します。これは、修飾されていない SQL ID でなければなりません (SQLSTATE 42601)。メソッド名は、CREATE TYPE に使用されるスキーマで暗黙的に修飾されます。

述部のキーワードとして使用されるいくつかの名前は、システム使用に予約されており、 *method-name* として使用することはできません (SQLSTATE 42939)。それらの名前は、SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および比較演算子です。

一般に、メソッドのシグニチャーがそれぞれ異なっている場合は、同じ名前を複数のメソッドに使用することができます。

parameter-name
パラメーター名を指定します。その名前は SELF であってはなりません。これは、メソッドの暗黙のサブジェクト・パラメーターの名前です (SQLSTATE 42734)。メソッドが SQL メソッドである場合、そのすべてのパラメーターに名前が付いていなければなりません (SQLSTATE 42629)。宣言するメソッドが別のメソッドをオーバーライドする場合は、パラメーター名は、オーバーライドされるメソッドの対応するパラメーターの名前と正確に一致している必要があります。そうでないと、エラーが戻されます (SQLSTATE 428FV)。

data-type2
各パラメーターのデータ・タイプを指定します。メソッドが受け取るはずの各パラメーターごとに 1 つの項目をこのリストに指定する必要があります。暗黙の SELF パラメーターを含め、90 を超える数のパラメーターを使うことはできません。この限界を超えると、エラーになります (SQLSTATE 54023)。

CREATE TABLE ステートメントに列タイプとして指定でき、しかもメソッドの作成に使用されている言語に対応するような SQL データ・タイプ指定と省略形を指定することができます。ユーザー定義関数とメソ

ッドに関する SQL データ・タイプとホスト言語データ・タイプの対応については、「*DB2 アプリケーション開発ガイド*」の言語別の項を参照してください。

注: 該当する SQL データ・タイプが構造タイプである場合、ホスト言語データ・タイプに対するデフォルト・マッピングはありません。構造タイプとホスト言語データ・タイプとをマッピングするには、ユーザー定義のトランスフォーム関数を使用する必要があります。

DECIMAL (および NUMERIC) は、C 言語と OLE では無効です (SQLSTATE 42815)。

REF を指定することができますが、これには定義された有効範囲はありません。メソッドの本体で、まず参照タイプをキャストして有効範囲をもたせて初めて、パス式内でその参照タイプを使用できるようになります。同様に、メソッドから戻された参照も、まずキャストして有効範囲をもたせて初めて、パス式内で使用できるようになります。

AS LOCATOR

LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターをメソッドに渡すことを指定します。これにより、メソッドに渡すバイト数を大幅に削減することができ、パフォーマンスも向上します。メソッドにとって実際に必要になる値が数バイトだけである場合は特にそうです。

LOB または LOB に基づく特殊タイプ以外のタイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

メソッドが FENCED の場合や、LANGUAGE が SQL の場合、AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

宣言するメソッドが別のメソッドをオーバーライドする場合は、パラメーターの AS LOCATOR 指示は、オーバーライドされるメソッドの対応するパラメーターの AS LOCATOR 指示と正確に一致している必要があります (SQLSTATE 428FV)。

宣言するメソッドが別のメソッドをオーバーライドする場合は、各パラメーターの FOR BIT DATA 指示は、オーバーライドされるメソッドの対応するパラメーターの FOR BIT DATA 指示と正確に一致している必要があります。 (SQLSTATE 428FV)。

RETURNS

これは必須の文節であり、メソッドの結果を指定します。

data-type3

メソッドの結果のデータ・タイプを指定します。この場合、上記のメソッドのパラメーター data-type2 の項で説明したのと全く同じ考慮事項があてはまります。

AS LOCATOR

LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターがメソッドから渡されることを示します。

LOB または LOB に基づく特殊タイプ以外のタイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

メソッドが FENCED の場合や、LANGUAGE が SQL の場合、AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

定義するメソッドが別のメソッドをオーバーライドする場合は、この文節を指定できません (SQLSTATE 428FV)。

メソッドが別のメソッドをオーバーライドする場合、 *data-type3* は、データ・タイプが構造タイプであれば、オーバーライドされるメソッドの結果のデータ・タイプのサブタイプでなければなりません。そうでない場合は、両方のデータ・タイプは同じでなければなりません (SQLSTATE 428FV)。

data-type4 **CAST FROM** *data-type5*
メソッドの結果のデータ・タイプを指定します。

この文節は、メソッド・コードから戻されたデータ・タイプとは異なるデータ・タイプを、呼び出しステートメントに戻すのに使用されます。 *data-type5* は、 *data-type4* パラメーターにキャスト可能でなければなりません。キャスト可能でないと、エラーになります (SQLSTATE 42880)。

data-type4 の長さ、精度または位取りは、 *data-type5* から推断することができるので、 *data-type4* に指定されるパラメーター化タイプの長さ、精度、または位取りを指定する必要はありません (指定は可能です)。代わりに、VARCHAR() のような空の括弧を使用できます。パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

特殊タイプは、data-type5 に指定するタイプとしては無効です (SQLSTATE 42815)。

キャスト操作は実行時検査の対象にもなり、その結果、変換エラーになる可能性があります。

AS LOCATOR
LOB タイプまたは LOB タイプに基づく特殊タイプの場合、 AS LOCATOR 文節を追加することができます。これは、実際の値の代わりに LOB ロケーターがメソッドから渡されることを示します。

LOB または LOB に基づく特殊タイプ以外のタイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

メソッドが FENCED の場合や、LANGUAGE が SQL の場合、AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

定義するメソッドが別のメソッドをオーバーライドする場合は、この文節を指定できません (SQLSTATE 428FV)。

定義するメソッドが別のメソッドをオーバーライドする場合は、 FOR BIT DATA 文節を指定できません (SQLSTATE 428FV)。

SPECIFIC *specific-name*
定義するメソッドのインスタンスに対するユニーク名を指定します。この名前は、メソッドの本体の作成やメソッドのドロップのときに使用することができます。これは、メソッドの呼び出しには使用できません。 *specific-name* (特定名)

の非修飾形式は SQL ID です (最大長 18)。修飾形式は、スキーマ名とその後に続くピリオドと SQL ID です。暗黙または明示の修飾子も含め、その名前が、アプリケーション・サーバーに存在する別の個別メソッド名を指定するものであってはなりません。そうでない場合は、エラーになります (SQLSTATE 42710)。

specific-name は、既存の *method-name* と同じでも構いません。

修飾子を指定しない場合、*type-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *type-name* の明示または暗黙の修飾子と同じでなければなりません。そうでない場合は、エラーになります (SQLSTATE 42882)。

specific-name の指定がない場合、ユニークな名前がデータベース・マネージャーによって生成されます。生成されるユニーク名は、SQL の後に文字のタイム・スタンプが続く名前です (SQLyymmddhhmmssxxx)。

SELF AS RESULT
このメソッドがタイプ保存メソッドであることを指定します。その意味は次のとおりです。

- 宣言された戻りタイプは、宣言されたサブジェクト・タイプと同じでなければなりません (SQLSTATE 428EQ)。
- SQL ステートメントがコンパイルされ、タイプ保存メソッドに解決されると、そのメソッド結果の静的タイプは、サブジェクト引き数の静的タイプと同じになります。
- メソッドをインプリメントする場合、結果の動的タイプが、サブジェクト引き数の静的タイプと同じになる (SQLSTATE 2200G) ようにし、そしてその結果 NULL にならない (SQLSTATE 22004) ようにする必要があります。

定義するメソッドが別のメソッドをオーバーライドする場合は、この文節を指定できません (SQLSTATE 428FV)。

SQL-routine-characteristics
CREATE METHOD を使ってこのタイプに定義されるメソッド本体の特性を指定します。

LANGUAGE SQL
この文節を使って、単一の RETURN ステートメントを使って SQL でメソッドを作成することを指示します。メソッド本体は、CREATE METHOD ステートメントを使って指定します。

PARAMETER CCSID
SQL メソッドとやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。

UNICODE
文字データは UTF-8 で記述され、GRAPHIC データは UCS-2 で記述されることを指定します。データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 56031)。

NOT DETERMINISTIC または **DETERMINISTIC**
この文節はオプションですが、特定の引き数の値に対してメソッドが常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に応じてメソッドの結果が異なるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC メソッドは、同じ入力を使用して正しく呼び出した場合に常に同じ結果を戻します。 NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。メソッド本体が特殊レジスターにアクセスしたり、別の非 deterministic ルーチンを呼び出したりする場合、明示的または暗黙的に NOT DETERMINISTIC を指定しなければなりません (SQLSTATE 428C2)。

EXTERNAL ACTION または **NO EXTERNAL ACTION**
この文節はオプションであり、データベース・マネージャーによって管理されていないオブジェクトの状態を変更する処置をメソッドが行うか否かを指定します。 EXTERNAL ACTION を指定すると、外部からメソッドへの影響がないことを前提とした最適化ができなくなります。 (たとえば、メッセージの送信、警報音による通知、ファイルへのレコードの書き込みなど。)

READS SQL DATA または **CONTAINS SQL**
どのタイプの SQL ステートメントを実行できるかを指示します。サポートされている SQL ステートメントは RETURN ステートメントであるので、式が副照会であるかどうかで区別を行います。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、メソッドで実行できることを指定します (SQLSTATE 42985)。 SQL ステートメント内でニックネームを参照することはできません (SQLSTATE 42997)。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、メソッドで実行できることを指定します (SQLSTATE 42985)。

CALLED ON NULL INPUT
このオプション文節は、引き数が NULL 値か否かに関係なくユーザー定義メソッドを呼び出すことを指定します。これは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。ただし、NULL の引き数値の有無のテストはメソッドが行う必要があります。

定義するメソッドが別のメソッドをオーバーライドする場合は、この文節を指定できません (SQLSTATE 428FV)。

NULL CALL は、CALLED ON NULL INPUT の同義語として使うことができます。

INHERIT SPECIAL REGISTERS
このオプションの文節は、メソッド内の更新可能特殊レジスターが、初期値を呼び出し側ステートメントの環境から継承することを指定します。カーソルの選択ステートメントから呼び出されるメソッドの場合は、 -初期値はカ

ーソルがオープンされた環境から継承されます。ネストされたオブジェクトで呼び出されるルーチン (トリガーまたはビューなど) の場合は、初期値はランタイム環境から継承されます (オブジェクト定義からは継承されません)。

特殊レジスターに対する変更が、関数の呼び出し側に戻されることはありません。

更新不能の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、デフォルト値に設定されます。

external-routine-characteristics

LANGUAGE

この文節は必須で、ユーザー定義メソッドの本体が準拠している言語インターフェース規則を指定するのに使用します。

C これは、データベース・マネージャーが、ユーザー定義メソッドを C の関数であるかのように呼び出すことを意味します。ユーザー定義メソッドは、標準 ANSI C プロトタイプで定義されている C 言語の呼び出しおよびリンケージの規則に準拠していなければなりません。

JAVA

データベース・マネージャーは、 Java クラスのメソッドとしてユーザー定義メソッドを呼び出します。

OLE

データベース・マネージャーは、 OLE 自動化オブジェクトによって公開されたメソッドとして、ユーザー定義メソッドを呼び出します。メソッドは、「*OLE Automation Programmer's Reference*」に説明されている OLE 自動化データ・タイプと呼び出しメカニズムに準拠している必要があります。

LANGUAGE OLE は、Windows 32 ビット・オペレーティング・システムで保管されたユーザー定義メソッドに対してのみサポートされます。THREADSAFE は、LANGUAGE OLE で定義されたメソッドに指定することはできません (SQLSTATE 42613)。

PARAMETER STYLE

この文節は、メソッドに対してパラメーターを渡し、そこから値を戻すのに用いる規則を指定するのに使用されます。

DB2GENERAL

Java クラスのメソッドとして定義された外部メソッドとの間で、パラメーターを渡し、値を戻す場合に用いる規則を指定します。これは、LANGUAGE JAVA を使用する場合にだけ指定する必要があります。

DB2GENERAL の同義語として値 DB2GENRL が使用可能です。

SQL

C 言語の呼び出しとリンケージの規則、または OLE 自動化オブジェクトによって公開されたメソッドに準拠する規則を、この外部メソッドとの間でパラメーターを渡し、値を戻す場合の規則として指定します。これは、LANGUAGE C または LANGUAGE OLE を使用する場合に指定する必要があります。

PARAMETER CCSID
外部メソッドとやり取りされるすべてのストリング・データに使用されるエンコーディング・スキームを指定します。 PARAMETER CCSID 文節を指定しない場合のデフォルトは、 Unicode データベースでは PARAMETER CCSID UNICODE、他のすべてのデータベースでは PARAMETER CCSID ASCII になります。

ASCII
ストリング・データがデータベース・コード・ページでエンコードされることを指定します。データベースが Unicode データベースの場合は、PARAMETER CCSID ASCII を指定することはできません (SQLSTATE 56031)。

UNICODE
文字データは UTF-8 で記述され、GRAPHIC データは UCS-2 で記述されることを指定します。データベースが Unicode データベースでない場合は、 PARAMETER CCSID UNICODE は指定できません (SQLSTATE 56031)。

この文節を LANGUAGE OLE とともに指定することはできません (SQLSTATE 42613)。

DETERMINISTIC または **NOT DETERMINISTIC**
この文節はオプションですが、特定の引き数の値に対してメソッドが常に同じ結果を戻すか (DETERMINISTIC)、それとも状態値に応じてメソッドの結果が異なるか (NOT DETERMINISTIC) を指定します。つまり DETERMINISTIC メソッドは、同じ入力を使用して正しく呼び出した場合に常に同じ結果を戻します。 NOT DETERMINISTIC を指定すると、同じ入力によって常に同じ結果が生じる利点に基づく最適化ができなくなります。

NOT DETERMINISTIC メソッドの例として、ある部署の社員の通し番号をランダムに戻すメソッドが挙げられます。 DETERMINISTIC メソッドの例として、多角形の面積を計算するメソッドが挙げられます。

FENCED または **NOT FENCED**
この文節は、データベース・マネージャーの操作環境のプロセスまたはアドレス・スペースでメソッドを実行しても「安全」か (NOT FENCED)、そうでないか (FENCED) を指定します。

メソッドが FENCED として登録されると、データベース・マネージャーは、その内部リソース (データ・バッファーなど) を保護して、そのメソッドからアクセスされないようにします。多くのメソッドは、FENCED または NOT FENCED のどちらかで実行するように選択することができます。一般に、FENCED として実行されるメソッドは、 NOT FENCED として実行されるものと同じようには実行されません。

注意:
十分にチェックされていないメソッドに NOT FENCED を使用すると、DB2 の保全性に危険を招く場合があります。 DB2 では、発生する可能性のある一般的な不注意による障害の多くに対して、いくつかの予防措置がとられていますが、 NOT FENCED ユーザー定義メソッドが使用される場合には、完全な保全性を確保できません。

LANGUAGE OLE または NOT THREADSAFE を指定したメソッドには、FENCED のみを指定できます (SQLSTATE 42613)。

メソッドが FENCED で NO SQL オプションが指定されている場合、 AS LOCATOR 文節を指定できません (SQLSTATE 42613)。

メソッドを NOT FENCED として登録するには、SYSADM 権限、DBADM 権限、または CREATE_NOT_FENCED_ROUTINE 権限が必要です。

THREADSAFE または **NOT THREADSAFE**
メソッドを他のルーチンと同じプロセスで実行しても「安全」か (THREADSAFE)、そうでないか (NOT THREADSAFE) を指定します。

メソッドが OLE 以外の LANGUAGE で定義される場合:

- メソッドが THREADSAFE に定義されている場合には、データベース・マネージャーは他のルーチンと同じプロセスでメソッドを呼び出すことができます。一般に、スレッド・セーフにするには、メソッドはどのグローバルあるいは静的データ域をも使用してはなりません。多くのプログラミング解説書には、スレッド・セーフ・ルーチンの作成に関する説明が含まれています。 FENCED および NOT FENCED メソッドの両方が THREADSAFE になることが可能です。
- メソッドが NOT THREADSAFE として定義される場合には、データベース・マネージャーは他のルーチンと同じプロセスにメソッドを決して呼び出しません。

FENCED メソッドについては、LANGUAGE が JAVA の場合、THREADSAFE がデフォルトです。これ以外のすべての言語の場合は、NOT THREADSAFE がデフォルトです。メソッドが LANGUAGE OLE とともに定義される場合には、THREADSAFE は指定されません (SQLSTATE 42613)。

NOT FENCED メソッドについては、THREADSAFE がデフォルトです。NOT THREADSAFE を指定することはできません (SQLSTATE 42613)。

RETURNS NULL ON NULL INPUT または **CALLED ON NULL INPUT**
このオプション文節を使用すると、非サブジェクト引き数のいずれかが NULL 値の場合に、外部メソッドを呼び出さないようにすることができます。

RETURNS NULL ON NULL INPUT が指定されており、実行時にメソッドの引き数のいずれかが NULL 値の場合、このメソッドは呼び出されず、結果は NULL 値になります。

CALLED ON NULL INPUT を指定すると、 NULL 値の引き数の数に関係なくメソッドが呼び出されます。これは、NULL 値を戻す場合も、通常の (NULL 以外の) 値を戻す場合もあります。ただし、NULL の引き数値の有無のテストはメソッドが行う必要があります。

値 NULL CALL は、後方互換性またはファミリーの互換性のために、CALLED ON NULL INPUT の同義語として使うことができます。同様に、NOT NULL CALL は、RETURNS NULL ON NULL INPUT の同義語として使用できます。

以下の 2 つのケースでは、この指定が無視されます。

- 対象となる引き数が NULL の場合。この場合、メソッドは実行されずに結果は NULL になります。
- パラメーターがないものとしてメソッドを定義した場合。この場合、この NULL 引き数条件が成立することはありません。

NO SQL、CONTAINS SQL、READS SQL DATA
メソッドから SQL ステートメントが発行されるかどうかと、もし発行されればどのタイプかを示します。

NO SQL
メソッドはどの SQL ステートメントも実行できないことを指示します (SQLSTATE 38001)。

CONTAINS SQL
SQL データの読み取りも変更も行わない SQL ステートメントを、メソッドで実行できることを指定します (SQLSTATE 38004 または 42985)。どのメソッドでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

READS SQL DATA
SQL データを変更しない SQL ステートメントを、メソッドで実行できることを指定します (SQLSTATE 38002 または 42985)。どのメソッドでもサポートされていないステートメントは、これとは異なるエラーを戻します (SQLSTATE 38003 または 42985)。

EXTERNAL ACTION または **NO EXTERNAL ACTION**
この文節はオプションであり、データベース・マネージャーによって管理されていないオブジェクトの状態を変更する処置をメソッドが行うか否かを指定します。 EXTERNAL ACTION を指定すると、外部からメソッドへの影響がないことを前提とした最適化ができなくなります。

NO SCRATCHPAD または **SCRATCHPAD** *length*
この文節はオプションであり、この外部メソッドに対してスクラッチパッドを用意するか否かを指定するのに使用できます。メソッドを再入可能にすることを強くお勧めします。再入可能にすると、スクラッチパッドが、呼び出しのたびにメソッドに「状態を保管」させる手段になります。

SCRATCHPAD を指定すると、ユーザー定義メソッドの最初の呼び出し時に、その外部メソッドによって使用されるスクラッチパッドにメモリーが割り振られます。このスクラッチパッドには、次の特性があります。

- *length* を指定して、スクラッチパッドのバイト単位のサイズを設定します。これは 1 ～ 32,767 でなければなりません (SQLSTATE 42820)。デフォルト値は 100 です。
- すべて X'00' に初期化されます。
- その有効範囲は、該当の SQL ステートメントです。 SQL ステートメントでの外部メソッドに対する参照ごとに 1 つのスクラッチパッドがあります。

したがって、次のステートメントのメソッド X が SCRATCHPAD キーワードを指定して定義されると、 3 つのスクラッチパッドが割り当てられます。

```
SELECT A, X..(A) FROM TABLEB
     WHERE X..(A) > 103 OR X..(A) < 19
```

ALLOW PARALLEL が指定されているか、またはデフォルト値として使用された場合、その有効範囲は上記とは異なります。メソッドが複数のパーティションで実行される場合、メソッドが処理されるそれぞれのパーティションにおいて、 SQL ステートメントでのメソッドへのそれぞれの参照ごとにスクラッチパッドが割り当てられます。同様に、パーティション内並列処理をオンにして照会が実行される場合、 3 つ以上のスクラッチパッドが割り当てられることがあります。

スクラッチパッドは持続します。その内容は、外部メソッドの呼び出しごとに保存されます。外部メソッドのある呼び出しによってスクラッチパッドに加えられた変更はいずれも、次の呼び出し時に存続しています。データベース・マネージャーは、各 SQL ステートメントの実行開始時に、スクラッチパッドを初期設定します。各副照会の実行開始時には、データベース・マネージャーによってスクラッチパッドがリセットされます。 FINAL CALL オプションが指定されている場合、システムは、スクラッチパッドのリセットに先立って、最終呼び出しを行います。

スクラッチパッドは、外部メソッドが獲得できるシステム・リソース (メモリーなど) の中央点として使用することもできます。メソッドは、最初の呼び出しでメモリーを獲得し、そのアドレスをスクラッチパッドに保管して、後の呼び出しでそれを参照することができます。

このようにシステム・リソースが獲得される場合、FINAL CALL キーワードも指定する必要があります。そうすると、ステートメントの最後で特殊な呼び出しが行われ、外部メソッドは獲得したシステム・リソースをすべて解放することができます。

SCRATCHPAD を指定すると、ユーザー定義メソッドを呼び出すたびに、スクラッチパッドをアドレッシングする外部メソッドに追加の引き数が渡されます。

NO SCRATCHPAD を指定すると、外部メソッドに対してスクラッチパッドは割り振られず、渡されません。

NO FINAL CALL または **FINAL CALL**

この文節はオプションであり、外部メソッドに対する最終呼び出しが行われるか否かを指定します。このような最終呼び出しの目的は、外部メソッドが獲得したシステム・リソースすべてを解放できるようにすることです。外部メソッドがメモリーなどのシステム・リソースを獲得し、それをスクラッチパッドに固定するような状況では、これを SCRATCHPAD キーワードと共に使用すると便利です。

FINAL CALL を指定すると、実行時に、呼び出しのタイプを指定する外部メソッドに追加の引き数が渡されます。呼び出しのタイプは次のとおりです。

- 通常呼び出し。SQL 引き数が渡され、結果が戻されることが予期されます。

- 最初の呼び出し。この SQL ステートメントのメソッドに対する参照に対応する外部メソッドの最初の呼び出しです。最初の呼び出しは通常呼び出しです。
- 最終呼び出し。外部メソッドがリソースを解放できるようにするそのメソッドに対する最終呼び出しです。最終呼び出しは、通常呼び出しではありません。この最終呼び出しは、以下の時点で行われます。
  - ステートメント終了時。これは、カーソル指向型のステートメントでカーソルがクローズされた場合、あるいはステートメントが実行を終了した場合に発生します。
  - トランザクション終了時。これは、通常のステートメント終了が発生しなかった場合に発生します。たとえば、何らかの理由で、アプリケーションのロジックが、カーソルをクローズしないようになっている場合があります。

  WITH HOLD として定義されたカーソルがオープンされている間に、コミット操作が発生すると、それ以降のカーソルのクローズ時、またはアプリケーションの終了時に最終呼び出しが行われます。

NO FINAL CALL を指定すると、「呼び出しタイプ」の引き数は外部メソッドに渡されず、最終呼び出しは行われません。

ALLOW PARALLEL または **DISALLOW PARALLEL**

この文節はオプションで、メソッドへの 1 つの参照で、メソッドの呼び出しを並列化できるか否かを指定します。一般には、ほとんどのスカラー・メソッドの呼び出しは並列化可能ですが、並列化できないメソッド (1 つのスクラッチパッドのコピーに依存するメソッドなど) もあります。スカラー・メソッドに対して ALLOW PARALLEL または DISALLOW PARALLEL を指定すると、 DB2 はその指定を受け入れます。

メソッドにどちらのキーワードが当てはまるかを判別するには、以下の点を検討する必要があります。

- メソッドのすべての呼び出しが、互いに完全に独立していますか? YES の場合には、ALLOW PARALLEL を指定します。
- メソッドを呼び出すごとに、次の呼び出しに関係する値を提供するスクラッチパッドが更新されますか? (たとえば、カウンターの増分によって。) YES の場合には、DISALLOW PARALLEL を指定するか、またはデフォルトを受け入れます。
- 1 つのパーティションでのみ起こる必要のある外部アクションがメソッドによって実行されますか? YES の場合には、DISALLOW PARALLEL を指定するか、またはデフォルトを受け入れます。
- コストのかかる初期化処理の実行回数を最小にするためだけに、スクラッチパッドを使用していますか? YES の場合には、ALLOW PARALLEL を指定します。

いずれの場合も、すべての外部メソッドの本体は、データベースのすべてのパーティションで使用可能なディレクトリーにある必要があります。

構文図は、デフォルト値が ALLOW PARALLEL であることを示しています。しかし、ステートメントで以下のオプションの少なくとも 1 つが指定されている場合は、デフォルトは DISALLOW PARALLEL です。

- NOT DETERMINISTIC
- EXTERNAL ACTION
- SCRATCHPAD
- FINAL CALL

NO DBINFO または **DBINFO**
この文節はオプションで、 DB2 において既知である特定の情報を追加の呼び出し時に引き数としてメソッドに渡すか (DBINFO)、または渡さないか (NO DBINFO) を指定します。 NO DBINFO がデフォルト値です。 DBINFO は、LANGUAGE OLE ではサポートされません (SQLSTATE 42613)。定義するメソッドが別のメソッドをオーバーライドする場合は、この文節を指定できません (SQLSTATE 428FV)。

DBINFO を指定すると、以下の情報をもつ構造がメソッドに渡されます。

- データベース名 - 現在接続されているデータベースの名前。
- アプリケーション ID - データベースへの接続ごとに確立された、ユニークなアプリケーション ID。
- アプリケーション許可 ID - アプリケーション実行時の許可 ID。このメソッドとアプリケーションとの中間でネストされているメソッドは無関係です。
- コード・ページ - データベースのコード・ページを識別します。
- スキーマ名 - 表名とまったく同じ条件のもとでは、スキーマの名前が入ります。その他の場合はブランクです。
- 表名 - メソッド参照が UPDATE ステートメントの SET 文節の右側にある場合、または INSERT ステートメントの VALUES リストの項目である場合のいずれかに限り、更新または挿入される表の非修飾名が入ります。その他の場合はブランクです。
- 列名 - 表名とまったく同じ条件で、更新または挿入される列の名前が入ります。その他の場合はブランクです。
- データベースのバージョン/リリース - メソッドを呼び出すデータベース・サーバーのバージョン、リリース、および修正レベルを識別します。
- プラットフォーム - サーバーのプラットフォーム・タイプが入ります。
- 表メソッドの結果の列番号 - メソッドには当てはまりません。

INHERIT SPECIAL REGISTERS
このオプションの文節は、メソッド内の特殊レジスターが、初期値を呼び出し側ステートメントから継承することを指定します。カーソルの場合は、初期値はカーソルがオープンされる時に継承されます。

特殊レジスターに対する変更が、メソッドの呼び出し元に戻されることはありません。

一部の特殊レジスター (日時特殊レジスターなど) は、現在実行中のステートメントのプロパティーを反映するので、呼び出し元からの継承は行われません。

注:

- ***互換性***

- DB2 UDB for OS/390 and z/OS との互換性:
  - 以下の構文が許容されます。
    - DETERMINISTIC の代わりに NOT VARIANT を指定できます。
    - NOT DETERMINISTIC の代わりに VARIANT を指定できます。
    - CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。
    - RETURNS NULL ON NULL INPUT の代わりに NOT NULL CALL を指定できます。
  - 以下の構文は、外部メソッドのデフォルトの振る舞いとして受け入れられます。
    - ASUTIME NO LIMIT
    - NO COLLID
    - PROGRAM TYPE SUB
    - STAY RESIDENT NO
    - Unicode データベースでの CCSID UNICODE
    - PARAMETER CCSID UNICODE が指定されていない場合、非 Unicode データベース内での CCSID ASCII
  - 以下の構文は、SQL メソッドのデフォルトの振る舞いとして受け入れられます。
    - Unicode データベースでの CCSID UNICODE
    - 非 Unicode データベースでの CCSID ASCII
- 以前のバージョンの DB2 との互換性:
  - PARAMETER STYLE SQL の代わりに PARAMETER STYLE DB2SQL を指定できます。

- まだ存在していないスキーマ名を用いて構造タイプを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- 属性なしで定義された構造化サブタイプは、属性をすべてスーパータイプから継承するサブタイプを定義します。 UNDER 文節も他のどの属性も指定しない場合、タイプは、属性なしの、タイプ階層のルート・タイプになります。
- タイプ階層に新たにサブタイプを追加すると、パッケージが無効になることがあります。パッケージは、その新しいタイプのスーパータイプに依存していると、無効になることがあります。このような従属関係は、TYPE 述部または TREAT 指定を使用した結果として生じます。
- 構造タイプは 4082 個を超える属性をもつことはできません (SQLSTATE 54050)。
- 関数と同じシグニチャーをもつメソッド (関数の最初のパラメーター・タイプと、メソッドのサブジェクト・タイプの比較による) を指定することはできません。
- 元のメソッドは、別のメソッドをオーバーライドしたり、元のメソッドによってオーバーライドされたりしてはなりません (SQLSTATE 42745)。さらに、関数とメソッドは、オーバーライド関係にあってはなりません。つまり、関数は、サブジェクト S を第 1 パラメーターとしてもつメソッドであると見なされる場合、

S のスーパータイプの別のメソッドをオーバーライドしてはならず、 S のサブタイプの別のメソッドによってオーバーライドされてはならないという意味です (SQLSTATE 42745)。

- ある構造タイプを作成すると、そのタイプで使用される一連の関数とメソッドが自動的に生成されます。これらの関数とメソッドはすべて、構造タイプと同じスキーマ内で生成されます。生成された関数またはメソッドのシグニチャーが、このスキーマに存在する関数のシグニチャーと競合またはそれをオーバーライドする場合、このステートメントは失敗します (SQLSTATE 42710)。構造タイプをドロップしないで、生成された関数またはメソッドをドロップすることはできません (SQLSTATE 42917)。次のような関数とメソッドが生成されます。
  - 関数
    - 参照比較

      REF(*type-name*) という参照タイプでは、 =、<>、<、<=、>、>= という名前の 6 つの比較関数が生成されます。 これらの関数はそれぞれ REF(*type-name*) というタイプのパラメーターを 2 つ受け取ってから、真、偽、または不明という値を戻します。 REF(*type-name*) の比較演算子は、REF(*type-name*) の基礎データ・タイプと同じ動作をするように定義されます。 (タイプ階層に含まれる参照表示タイプはすべて同一のものです。これにより、REF(S) と REF(T) の比較が可能になります (S と T が共通のスーパータイプを持っている場合)。表の OID 列は、表階層だけで固有になるので、1 つの表階層の REF(T) 値を別の表階層の REF(T) 値と「等しい」ものにすることができます (それぞれが異なった行を参照していても)。)

      参照タイプの有効範囲は比較の対象にはなりません。

    - Cast 関数

      生成された参照タイプである REF(*type-name*) とこの参照タイプの基礎データ・タイプとの間をキャストするために 2 つの cast 関数が生成されます。
      - 基礎タイプから参照タイプへとキャストする関数の名前は、暗黙的または明示的な *funcname1* です。

        この関数の形式は以下のとおりです。

        ```
        CREATE FUNCTION funcname1 (rep-type)
          RETURNS REF(type-name) ...
        ```

      - 参照タイプから基礎タイプ (参照タイプの) へとキャストする関数の名前は、暗黙的または明示的な *funcname2* です。

        この関数の形式は以下のとおりです。

        ```
        CREATE FUNCTION funcname2 ( REF(type-name) )
          RETURNS rep-type ...
        ```

      ある種の rep-type には、定数からのキャストを操作する *funcname1* を使って生成された追加の cast 関数があります。
      - *rep-type* が SMALLINT の場合、追加で生成された cast 関数の形式は以下のとおりです。

        ```
        CREATE FUNCTION funcname1 (INTEGER)
          RETURNS REF(type-name)
        ```

- *rep-type* が CHAR(n) の場合、追加で生成された cast 関数の形式は以下のとおりです。

```
CREATE FUNCTION funcname1 ( VARCHAR(n))
  RETURNS REF(type-name)
```

- *rep-type* が GRAPHIC(n) の場合、追加で生成された cast 関数の形式は以下のとおりです。

```
CREATE FUNCTION funcname1 (VARGRAPHIC(n))
  RETURNS REF(type-name)
```

それらの演算子や cast 関数を SQL ステートメントで正しく使用するには、 SQL パスに構造タイプのスキーマ名が組み込まれていなければなりません。

- コンストラクター関数

コンストラクター関数は、そのタイプの新しいインスタンスを構成可能にするために生成されます。この新しいインスタンスでは、スーパータイプから継承する属性も含め、そのタイプのどの属性も NULL になります。

生成されるコンストラクター関数の形式は、以下のとおりです。

```
CREATE FUNCTION type-name ( )
  RETURNS type-name
 ...
```

NOT INSTANTIABLE を指定すると、コンストラクター関数は生成されません。構造タイプがタイプ DATALINK の属性を持っていると、コンストラクター機能の呼び出しは失敗します (SQLSTATE 428ED)。

– メソッド

- observer メソッド

構造タイプの各属性ごとに observer メソッドが定義されます。 observer メソッドは、各属性ごとに属性タイプを戻します。対象が NULL の場合、observer メソッドは、属性タイプの NULL 値を戻します。

たとえば、C1..STREET、C1..CITY、C1..COUNTRY、および C1..CODE を使って、構造タイプ ADDRESS のインスタンスの属性を監視することができます。

生成される observer メソッドのメソッド・シグニチャーは、次のようなステートメントが実行された場合に似ています。

```
CREATE TYPE  type-name
    ...
 METHOD attribute-name()
    RETURNS attribute-type
```

type-name は、構造タイプ名です。

- mutator メソッド

構造タイプの各属性ごとに、タイプ保存の mutator メソッドが定義されます。構造タイプのインスタンス内の属性を変更するには、mutator メソッド

を使用します。 mutator メソッドは、各属性ごとに、サブジェクトのコピーの指定属性に引き数を割り当てることで変更されたそのコピーを戻します。

たとえば、C1..CODE('M3C1H7') を使って、構造タイプ ADDRESS のインスタンスを変異することができます。サブジェクトが NULL の場合、mutator メソッドはエラーを生じます (SQLSTATE 2202D)。

生成される mutator メソッドのメソッド・シグニチャーは、次のようなステートメントが実行された場合に似ています。

```
CREATE TYPE type-name
     ...
  METHOD attribute-name (attribute-type)
   RETURNS type-name
```

属性のデータ・タイプが SMALLINT、REAL、CHAR、または GRAPHIC である場合、定数を使用する変異をサポートするため、次のような追加の mutator メソッドが生成されます。

- *attribute-type* が SMALLINT の場合、追加の mutator はタイプ INTEGER の引き数をサポートします。
- *attribute-type* が REAL の場合、追加の mutator はタイプ DOUBLE の引き数をサポートします。
- *attribute-type* が CHAR の場合、追加の mutator はタイプ VARCHAR の引き数をサポートします。
- *attribute-type* が GRAPHIC の場合、追加の mutator はタイプ VARGRAPHIC の引き数をサポートします。

- 列タイプとして構造タイプを使用する場合、そのタイプのインスタンスの長さは、実行時に 1 GB を超えてはなりません (SQLSTATE 54049)。

- 既存の構造タイプの新しいサブタイプを作成する (列タイプとして使用するため) 場合、それに関連した既存の構造タイプのサポートとしてすでに作成されているすべてのトランスフォーム関数を再検査し、必要があれば更新してください。その新しいタイプが、特定のタイプとして同じ階層内にあっても、あるいはネストされたタイプの階層内にあっても、そのタイプに関連した既存のトランスフォーム関数を変更して、新規のサブタイプによって導入される新しい属性の一部または全部を組み込む必要があると考えられます。概して、それは、UDF とクライアント・アプリケーションから構造タイプにアクセスさせるための特定のタイプ (またはタイプ階層) に関連した一連のトランスフォーム関数であるため、特定の複合階層内のすべての属性 (つまり、すべてのサブタイプとそのネストされた構造タイプの推移的な閉止を含む) をサポートするように、トランスフォーム関数を作成しなければなりません。

  既存のタイプの新しいサブタイプを作成すると、作成されたタイプのスーパータイプで定義されるメソッドで、しかもオーバーライドが可能なメソッドに従属するすべてのパッケージは無効になります。

- ***表アクセスの制限***

  メソッドが READS SQL DATA として定義されている場合は、メソッド内のステートメントは、このメソッドを呼び出したステートメントによって変更される表にアクセスすることはできません (SQLSTATE 57053)。たとえば、メソッド BONUS() が READS SQL DATA として定義されているとします。ステートメン

ト UPDATE DEPTINFO SET SALARY = SALARY + EMP..BONUS() が呼び出されると、 BONUS メソッド内の SQL ステートメントは、EMPLOYEE 表を読み取ることができません。

- ***特権***
  - ユーザー定義タイプの定義者は、構造タイプ用に自動的に生成されるすべてのメソッドおよび関数に対する EXECUTE 特権 WITH GRANT OPTION を常に受け取ります。 EXECUTE 特権は、CREATE METHOD ステートメントを使用してメソッド本体が定義されない限り、 CREATE TYPE ステートメントで明示的に指定されるメソッドに対しては付与されません。ユーザー定義タイプの定義者には、 ALTER TYPE ステートメントを使用してメソッド指定をドロップする権利があります。 CREATE DISTINCT TYPE の間に自動的に生成されるすべての関数での EXECUTE 特権は、 PUBLIC に与えられます。
  - SQL ステートメントで外部メソッドを使用する場合は、メソッドの定義者は、メソッドが使用するどのパッケージに対しても EXECUTE 特権を持っている必要があります。
- パーティション・データベース環境では、外部ユーザー定義関数またはメソッドでの SQL の使用はサポートされていません (SQLSTATE 42997)。
- 索引拡張を定義するには、NO SQL として定義されたルーチンしか使用できません (SQLSTATE 428F8)。
- NOT FENCED として定義される Java ルーチンは、FENCED THREADSAFE として定義されているかのように呼び出されます。

例:

例 1: 部門のタイプを作成します。

```
CREATE TYPE DEPT AS
   (DEPT NAME     VARCHAR(20),
      MAX_EMPS INT)
      REF USING INT
  MODE DB2SQL
```

例 2: 従業員タイプおよびマネージャー・サブタイプから構成される階層タイプを作成します。

```
CREATE TYPE EMP AS
  (NAME      VARCHAR(32),
  SERIALNUM INT,
  DEPT      REF(DEPT),
  SALARY    DECIMAL(10,2))
  MODE DB2SQL

CREATE TYPE MGR UNDER EMP AS
  (BONUS     DECIMAL(10,2))
  MODE DB2SQL
```

例 3: アドレスのタイプ階層を作成します。アドレスは、列のタイプとして使用するためのものです。インライン長は指定されていないので、DB2 がデフォルト長を計算します。該当のアドレスが、特定の入力アドレスにどのくらい近いかを計算する外部メソッドを、アドレス・タイプ定義内にカプセル化します。 CREATE METHOD ステートメントを使ってメソッド本体を作成します。

```
CREATE TYPE address_t AS
  (STREET     VARCHAR(30),
  NUMBER     CHAR(15),
  CITY       VARCHAR(30),
```

```
  STATE      VARCHAR(10))
  NOT FINAL
  MODE DB2SQL
    METHOD SAMEZIP (addr address_t)
    RETURNS INTEGER
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    NO EXTERNAL ACTION

    METHOD DISTANCE (address_t)
    RETURNS FLOAT
    LANGUAGE C
    DETERMINISTIC
    PARAMETER STYLE SQL
    NO SQL
    NO EXTERNAL ACTION

CREATE TYPE germany_addr_t UNDER address_t AS
  (FAMILY_NAME VARCHAR(30))
  NOT FINAL
  MODE DB2SQL

CREATE TYPE us_addr_t UNDER address_t AS
  (ZIP VARCHAR(10))
  NOT FINAL
  MODE DB2SQL
```

例 4: ネストされた構造タイプ属性をもつタイプを作成します。

```
CREATE TYPE PROJECT AS
  (PROJ_NAME  VARCHAR(20),
   PROJ_ID    INTEGER,
   PROJ_MGR   MGR,
   PROJ_LEAD  EMP,
   LOCATION   ADDR_T,
   AVAIL_DATE DATE)
  MODE DB2SQL
```

関連資料:

- *SQL リファレンス 第 1 巻* の『基本述部』
- 347 ページの『CREATE TABLE』
- 762 ページの『SET PATH』
- *SQL リファレンス 第 1 巻* の『特殊レジスター』

関連サンプル:

- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』

CREATE TYPE MAPPING

CREATE TYPE MAPPING ステートメントは、以下のデータ・タイプ間のマッピングを作成します。

- フェデレーテッド・データベースに定義される予定の、データ・ソース表またはビューの列のデータ・タイプ。
- フェデレーテッド・データベースに定義済みの、対応するデータ・タイプ。

マッピングによって、フェデレーテッド・データベース・データ・タイプを以下に含まれているデータ・タイプに関連付けることができます。

- 指定したデータ・ソース
- データ・ソースの範囲。たとえば、特定のタイプおよびバージョンのすべてのデータ・ソース

データ・タイプのマッピングは、既存のデータ・タイプでは不十分な場合にのみ作成する必要があります。

ニックネームの作成時または表の作成時 (透過性 DDL) に複数のタイプ・マッピングが適用できる場合、最新のマッピングが適用されます。

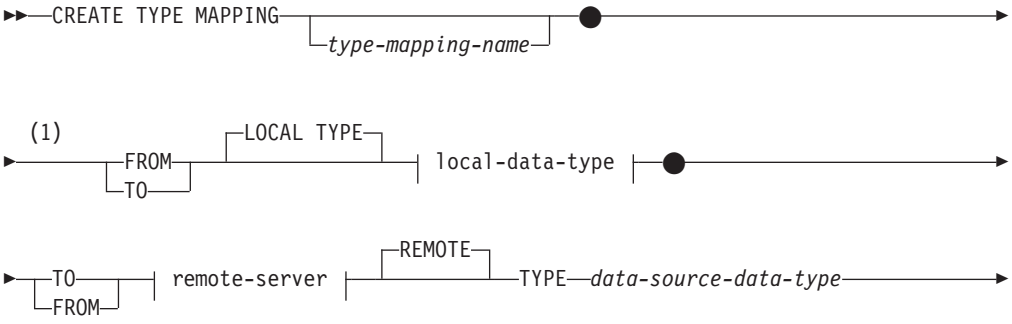
呼び出し:

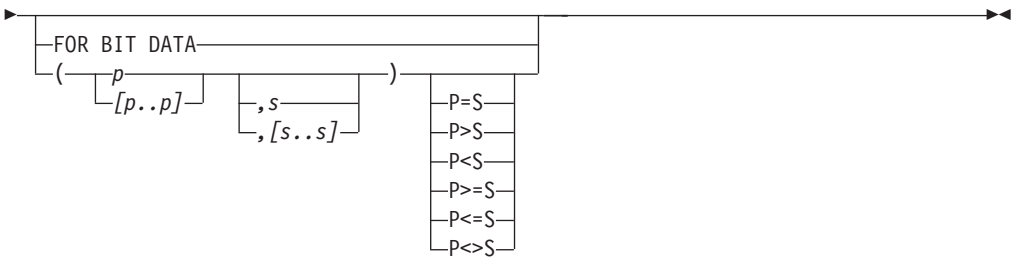
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。パッケージに対して DYNAMICRULES が有効な場合、動的にプリペアーすることが可能なステートメントです (SQLSTATE 42509)。

許可:

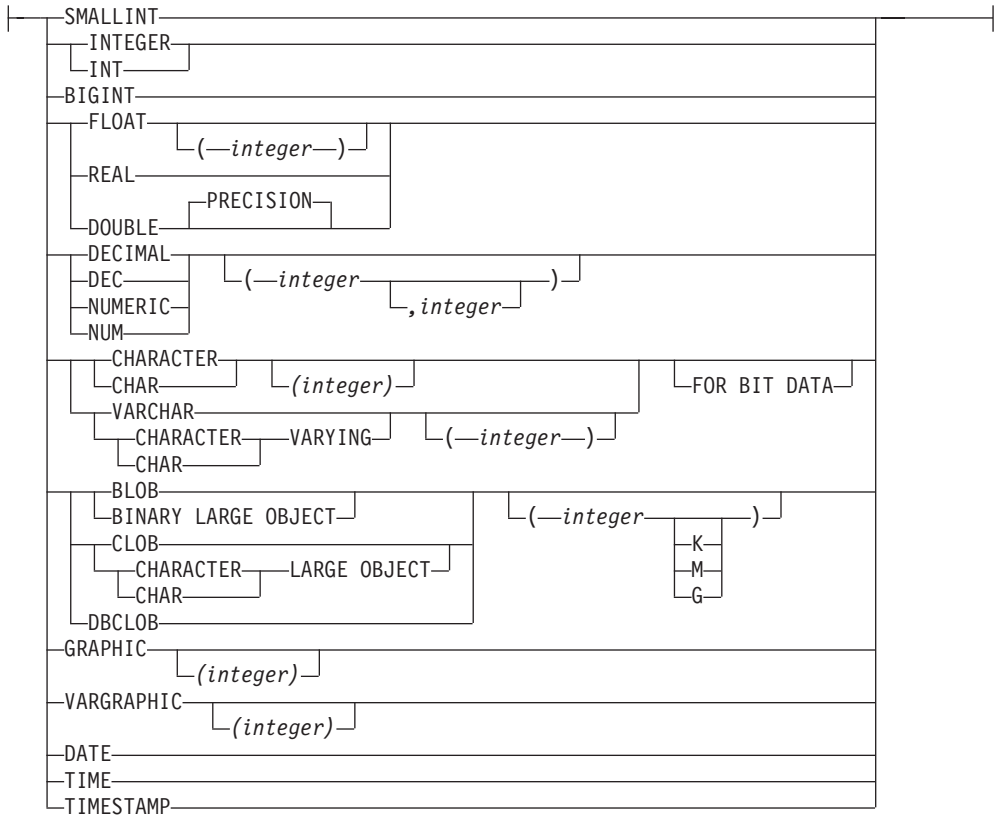
このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文:

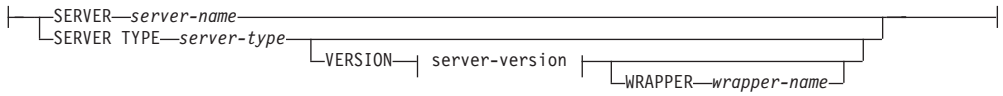




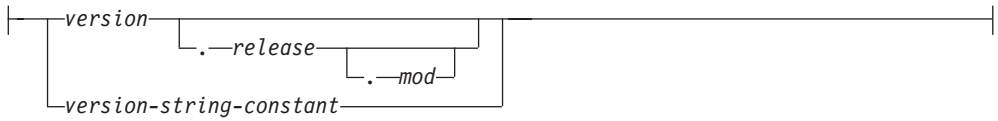
local-data-type:



remote-server:



server-version:



注:

1 CREATE TYPE MAPPING ステートメントには、 TO キーワードと FROM キーワードの両方を指定する必要があります。

説明:

type-mapping-name
データ・タイプ・マッピングに名前を付けます。この名前は、カタログですでに記述されているデータ・タイプ・マッピングを指定するものであってはなりません。 *type-mapping-name* を指定しないと、ユニークな名前が生成されます。

FROM または **TO**
リバースまたはフォワード・タイプ・マッピングを指定します。

FROM
local-data-type が続く場合はフォワード・タイプ・マッピングを、*remote-server* が続く場合はリバース・タイプ・マッピングを指定します。

TO
remote-server が続く場合はフォワード・タイプ・マッピングを、*local-data-type* が続く場合はリバース・タイプ・マッピングを指定します。

local-data-type
フェデレーテッド・データベースに定義したデータ・タイプを指定します。*local-data-type* がスキーマ名なしで指定される場合、 SQL パスでスキーマを検索することにより、タイプ名は解決されます。

パラメーター化データ・タイプには、空の括弧を使用できます。特定の長さ、位取り、または精度を指定して定義可能なデータ・タイプのことを、パラメーター化データ・タイプといいます。フォワード・タイプ・マッピングに CHAR() のような空の括弧を指定すると、長さはリモート表の列の長さから判別されます。リバース・タイプ・マッピングに空の括弧を指定すると、タイプ・マッピングはその長さのデータ・タイプにも適用されます。括弧をすべて省略した場合は、データ・タイプのデフォルト長が使用されます (CREATE TABLE ステートメントの説明を参照)。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

local-data-type を、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、またはユーザー定義タイプにすることはできません (SQLSTATE 42611)。

SERVER *server-name*
data-source-data-type が定義されているデータ・ソースを指名します。

SERVER TYPE *server-type*
data-source-data-type が定義されているデータ・ソースのタイプを指定します。

VERSION
data-source-data-type が定義されているデータ・ソースのバージョンを指定します。

version
バージョン番号を指定します。値は整数でなければなりません。

release
version で示されたバージョンのリリース番号を指定します。値は整数でなければなりません。

mod
release で示されたリリースのモディフィケーション番号を指定します。値は整数でなければなりません。

version-string-constant
バージョンの正式名称を指定します。 *version-string-constant* は単一値 (たとえば、'8i') にすることができます。あるいは、*version*、*release*、そして該当する場合は *mod* を連結した値にすることができます (たとえば、'8.0.3')。

WRAPPER *wrapper-name*
server-type および *server-version* に示されたタイプおよびバージョンのデータ・ソースと対話するために、フェデレーテッド・サーバーが使用するラッパーの名前を指定します。

TYPE *data-source-data-type*
ローカル・データ・タイプとの間でマッピングされるデータ・ソースのデータ・タイプを指定します。

パラメーター化データ・タイプには、空の括弧を使用できます。フォワード・タイプ・マッピングに CHAR() のような空の括弧を指定すると、タイプ・マッピングはどの長さのデータ・タイプにも適用されます。リバース・タイプ・マッピングに空の括弧を指定すると、長さは透過性 DDL に指定されている列の長さから判別されます。括弧をすべて省略した場合は、データ・タイプのデフォルト長が使用されます。

data-source-data-type は、組み込みデータ・タイプでなければなりません。ユーザー定義タイプを指定することはできません。

server-name がタイプ・マッピングとともに指定されているか、または既存のサーバーがタイプ・マッピングの影響を受ける場合、タイプ・マッピング作成時に *data-source-data-type*、*p*、および *s* が検査されます (SQLSTATE 42611)。

p *p* が指定されている場合、 *p* と等しい長さまたは精度を持つデータ・タイプだけがタイプ・マッピングの影響を受けます。

[p1..p2]
フォワード・タイプ・マッピングのみ。 10 進データ・タイプの場合、*p1* と *p2* は値が取る最小および最大桁数を指定します。ストリング・データ・タイプの場合、 *p1* と *p2* は値が取る最小および最大文字数を指定します。いずれにせよ、最大値は最小値以上の値にする必要があります。また、最大値と最小値は両方とも、そのデータ・タイプに関して有効なものでなければなりません。

s *s* が指定されている場合、 *s* と等しいスケールを持つデータ・タイプだけがタイプ・マッピングの影響を受けます。

[s1..s2]
フォワード・タイプ・マッピングのみ。 10 進データ・タイプの場合、 *s1* と *s2* は小数点以下の桁数の最小および最大数を指定します。最大値は最小値以上の値にする必要があります。また、最大値と最小値は両方とも、そのデータ・タイプに関して有効なものでなければなりません。

P [operand] **S**
10 進データ・タイプの場合、 P *[operand]* S は精度と小数点以下の最大桁数との比較を指定します。たとえば、operand (オペランド) に = を指定すると、精度と小数部分に許容できる最大桁数が同じである場合に、タイプ・マッピングが適用されることを示します。

FOR BIT DATA

data-source-data-type が、ビット・データ用かどうかを示します。 データ・ソース・タイプの列に バイナリー値が含まれる場合、これらのキーワードは必須です。この属性が文字データ・タイプで指定されていない場合、データベース・マネージャーがこの属性を決定します。

注:

- 所定の作業単位 (UOW) 内の CREATE TYPE MAPPING ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - ステートメントが 1 つのデータ・ソースを参照していて、次のいずれかがすでに UOW に含まれている。
    - このデータ・ソース内の表またはビューのニックネームを参照する SELECT ステートメント。
    - このデータ・ソース内の表またはビューのニックネーム上のオープン・カーソル。
    - このデータ・ソース内の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
  - ステートメントがデータ・ソースのカテゴリー (例えば、特定のタイプおよびバージョンのすべてのデータ・ソースなど) を参照しており、次のいずれかがすでに UOW に含まれている。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームを参照する SELECT ステートメント。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネーム上のオープン・カーソル。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
- 複数のタイプ・マッピングが適用できる場合は、最新のマッピングが使用されます。 SYSCAT.TYPEMAPPINGS カタログ・ビューの CREATE_TIME 列を照会することにより、タイプ・マッピングの作成時間を検索できます。

例:

例 1: Oracle データ・タイプ DATE とデータ・タイプ SYSIBM.DATE との間のフォワード・タイプ・マッピングを作成します。このマッピングが定義された後に作成されるすべてのニックネームについて、データ・タイプ DATE の Oracle 列はデータ・タイプ DATE の DB2 列にマップします。

```
CREATE TYPE MAPPING MY_ORACLE_DATE
  FROM LOCAL TYPE SYSIBM.DATE
  TO SERVER TYPE ORACLE
  REMOTE TYPE DATE
```

例 2: データ・タイプ SYSIBM.DECIMAL(10,2) とデータ・ソース ORACLE1 の Oracle データ・タイプ NUMBER([10..38],2) との間のフォワード・タイプ・マッピングを作成します。データ・タイプ NUMBER(11,2) の Oracle 表にある列は、 11 が 10 と 38 の間に位置するので、データ・タイプ DECIMAL(10,2) の列にマップされます。

```
CREATE TYPE MAPPING MY_ORACLE_DEC
FROM LOCAL TYPE SYSIBM.DECIMAL(10,2)
 TO SERVER ORACLE1
REMOTE TYPE NUMBER([10..38],2)
```

例 3: データ・タイプ SYSIBM.VARCHAR(*p*) とデータ・ソース ORACLE1 の Oracle データ・タイプ CHAR(*p*) との間のフォワード・タイプ・マッピングを作成します (*p* は任意の長さ) 。データ・タイプ CHAR(10) の Oracle 表にある列は、データ・タイプ VARCHAR(10) の列にマップされます。

```
CREATE TYPE MAPPING MY_ORACLE_CHAR
FROM LOCAL TYPE SYSIBM.VARCHAR()
 TO SERVER ORACLE1
REMOTE TYPE CHAR()
```

例 4: データ・ソース ORACLE2 の Oracle データ・タイプ NUMBER(10,2) とデータ・タイプ SYSIBM.DECIMAL(10,2) との間のリバース・タイプ・マッピングを作成します。透過性 DDL を使用して Oracle 表を作成し、データ・タイプ DECIMAL(10,2) の列を指定すると、 DB2 はデータ・タイプ NUMBER(10,2) の列を持つ Oracle 表を作成します。

```
CREATE TYPE MAPPING MY_ORACLE_DEC
TO LOCAL TYPE SYSIBM.DECIMAL(10,2)
FROM SERVER ORACLE2
REMOTE TYPE NUMBER(10,2)
```

関連資料:

- 347 ページの『CREATE TABLE』

CREATE USER MAPPING

CREATE USER MAPPING ステートメントは、フェデレーテッド・データベースを使用する許可 ID と、指定したデータ・ソースで使用する許可 ID およびパスワードとの間のマッピングを定義します。

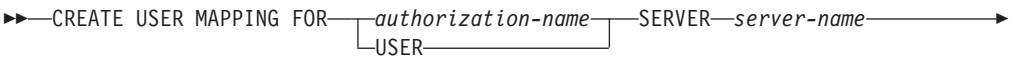
呼び出し:

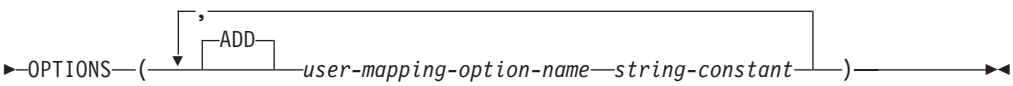
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID がデータ・ソースへマップされる許可名とは異なる場合、そのステートメントの許可 ID が持つ特権には SYSADM または DBADM 権限が含まれている必要があります。一方、許可 ID と許可名が一致する場合は、特権あるいは権限は必要ありません。

構文:





説明:

authorization-name
ユーザーまたはアプリケーションがフェデレーテッド・データベースへ接続するときの、許可名を指定します。 *authorization_name* は REMOTE_AUTHID ユーザー・マッピング・オプションにマップされます。

USER
USER 特殊レジスターの値。USER が指定されている場合、 CREATE USER MAPPING ステートメントを出す許可 ID は REMOTE_AUTHID ユーザー・マッピング・オプションにマップされます。

SERVER *server-name*
authorization-name (許可名) がアクセスできるデータ・ソースのサーバー・オブジェクトを指定します。 *server-name* はフェデレーテッド・データベースに登録されているリモート・サーバーのローカル名です。

OPTIONS
ユーザー・マッピングを作成したときに使用可能にされるオプションを指示します。

ADD
1 つ以上のユーザー・マッピング・オプションを使用可能にします。

user-mapping-option-name
オプションの名前を指定します。

string-constant
user-mapping-option-name の設定を文字ストリング定数として指定します。

注:

- ユーザー・マッピングは、DB2 ファミリー製品、Documentum、Informix、Microsoft SQL Server、ODBC、 Oracle、Sybase、および Teradata のデータ・ソースにのみ必要です。
- REMOTE_PASSWORD オプションはユーザー・マッピングには常に必要です。

例:

例 1: DB2 for z/OS および OS/390 データ・ソース・サーバー・オブジェクト SERVER390 へのユーザー・マッピングを登録します。ローカル・フェデレーテッド・データベースの許可名を SERVER390 のユーザー ID とパスワードにマップします。許可名は RSPALTEN です。 SERVER390 のユーザー ID は SYSTEM です。 SERVER390 のパスワードは MANAGER です。

```
CREATE USER MAPPING FOR RSPALTEN
  SERVER SERVER390
    OPTIONS
  (REMOTE_AUTHID 'SYSTEM',
  REMOTE_PASSWORD 'MANAGER')
```

例 2: Oracle データ・ソース・サーバー・オブジェクト ORACLE1 へのユーザー・マッピングを登録します。 MARCR は、ローカルのフェデレーテッド・データベースの許可名で、ORACLE1 のユーザー ID です。許可名とユーザー ID が同じなので、ユーザー・マッピングに REMOTE_AUTHID オプションを指定する必要はありません。 ORACLE1 上の MARCR のパスワードは NZXCZY です。

```
CREATE USER MAPPING FOR MARCR
    SERVER ORACLE1
    OPTIONS
  (REMOTE_PASSWORD 'NZXCZY')
```

関連資料:

- *フェデレーテッド・システム・ガイド* の『フェデレーテッド・システムのユーザー・マッピング・オプション』

CREATE VIEW

CREATE VIEW ステートメントは、1 つまたは複数の表、ビュー、またはニックネームに基づくビューを作成します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限、または
- 全選択に指定された表、ビュー、またはニックネームのそれぞれに対して、
  - その表またはビューに対する CONTROL 特権、または
  - その表またはビューに対する SELECT 特権

  および以下の少なくとも 1 つ
  - データベースに対する IMPLICIT_SCHEMA 権限 (ビューの暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (ビューのスキーマ名が既存のスキーマを指している場合)

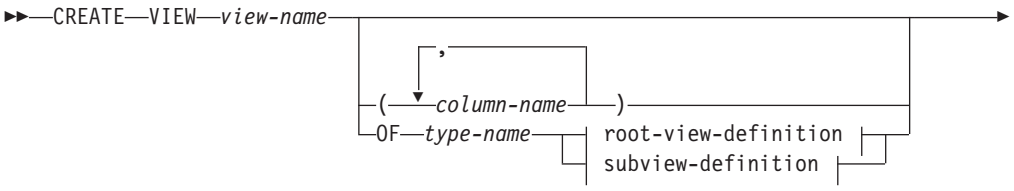
  サブビューを作成するには、このステートメントの許可 ID が以下の条件に適合している必要があります。
  - 表階層のルート表の定義者と同じ名前である。
  - サブビューの基礎表に対して SELECT WITH GRANT 権限を持っているか、またはスーパービューがこのビューの定義者以外にはだれにも SELECT 権限を付与していない。

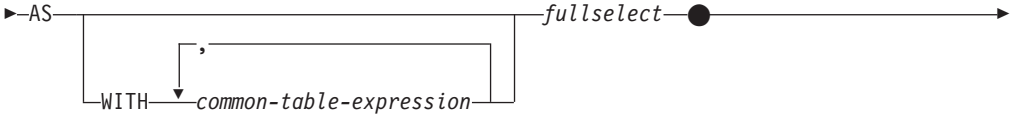
グループ特権は、CREATE VIEW ステートメントで指定された表やビューに対しては考慮されません。

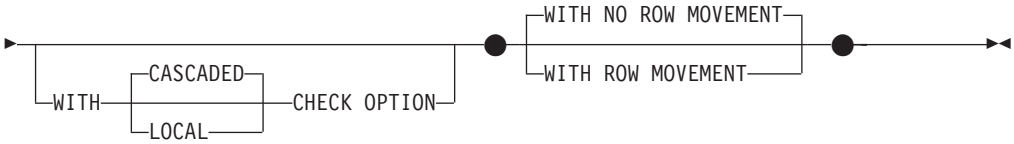
特権は、フェデレーテッド・データベースのニックネームにビューを定義するときには考慮されません。 このニックネームで示されている表またはビューのデータ・ソースの許可要件は、照会の処理時に適用されます。ステートメントの許可 ID は、別のリモート許可 ID へマップできます。

ビューの定義者が SYSADM 権限を持つために、ビューの作成しかできない場合、ビュー作成のため、その定義者には明示的な DBADM 権限が付与されます。

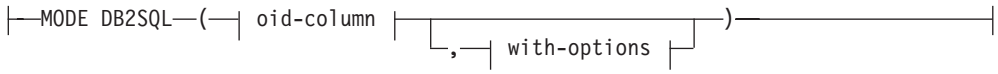
構文:



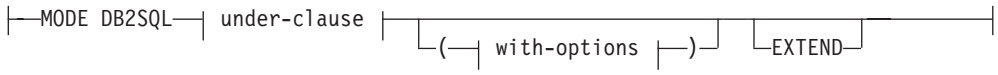




root-view-definition:

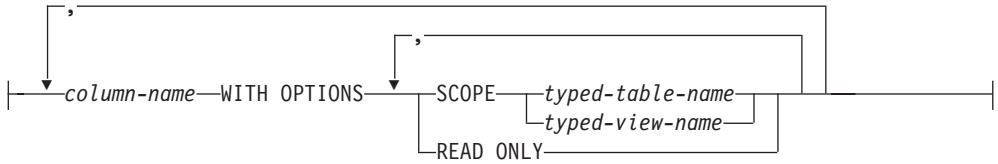


subview-definition:



oid-column:

with-options:



under-clause:

説明:

view-name

ビューの名前を指定します。暗黙または明示の修飾子を含む名前は、カタログに記述されている表、ビュー、ニックネーム、または別名を指定するものであってはなりません。修飾子は、SYSIBM、SYSCAT、SYSFUN、または SYSSTAT であってはなりません (SQLSTATE 42939)。

この名前は、作動不能なビューの名前と同じであっても構いません (483 ページの『作動不能ビュー』を参照)。このような場合、作動不能なビューは、CREATE VIEW ステートメントに指定した新しいビューによって置き換えられ

ます。作動不能なビューが置き換えられると、ユーザーに警告 (SQLSTATE 01595) が出されます。 BIND オプション SQLWARN を NO に設定してアプリケーションがバインドされた場合は、警告は戻されません。

column-name
ビューの列の名前を指定します。列名のリストを指定する場合、リスト中の列の名前の数は、全選択の結果表の列の数と同じ数でなければなりません。各 *column-name* (列名) は、ユニークで、しかも非修飾でなければなりません。列名のリストの指定がない場合、ビューの列は、全選択の結果表の列名を継承します。

全選択の結果表の列名が重複している場合、または無名の列がある場合には、列名のリストを指定する必要があります (SQLSTATE 42908)。 無名列とは、定数、関数、式、またはセット演算から派生した列で、選択リストの AS 文節によって名前が指定されていない列を指します。

OF *type-name*
ビューの列が *type-name* で指定される構造タイプの属性に基づいていることを指定します。 *type-name* の指定にスキーマ名が含まれていない場合、そのタイプ名は SQL パス上のスキーマを探索することによって決まります (このパスは、静的 SQL の場合は FUNCPATH プリプロセス・オプションによって、動的 SQL の場合は CURRENT PATH レジスターによって定義されます)。ここに指定するタイプ名は、既存のユーザー定義タイプ名で (SQLSTATE 42704)、かつインスタンス化の可能な構造タイプでなければなりません (SQLSTATE 428DP)。

MODE DB2SQL
この文節は、型付きビューのモードを指定するために使用されます。 これは、現在サポートされている唯一有効なモードです。

UNDER *superview-name*
このビューが *superview-name* のサブビューであることを指定します。 スーパービューは既存のビューでなければならず (SQLSTATE 42704)、このビューは *type-name* のすぐ上位にあるスーパータイプである構造タイプで定義する必要があります (SQLSTATE 428DB)。 *view-name* と *superview-name* のスキーマ名は、同じでなければなりません (SQLSTATE 428DQ)。 *superview-name* で指定されるビューには、 *type-name* で既に定義された既存のサブビューを含めることはできません (SQLSTATE 42742)。

表の列には、スーパービューのオブジェクト ID 列が含まれています。オブジェクト ID 列のタイプは REF(*type-name*) に変更されており、 *type-name* の属性に基づく列が続きます (ここでいうタイプには、スーパータイプの属性も含まれていることを念頭に置いてください)。

INHERIT SELECT PRIVILEGES
スーパービューに対して SELECT 特権を持つユーザーやグループはすべて、新しく作成したサブビューに対しても同様の特権を付与されます。この特権は、サブビュー定義者によって付与されたものと見なされます。

OID-column
型付きビューのオブジェクト ID 列を定義します。

REF IS *OID-column-name* **USER GENERATED**
オブジェクト ID (OID) 列をビューの最初の列として定義することを指定し

ます。ビュー階層のルート・ビューには、OID が必須です (SQLSTATE 428DX)。このビューはサブビュー以外の型付きビュー (OF 文節が必須) でなければなりません (SQLSTATE 42613)。この列の名前は *OID-column-name* という形式で定義されますが、構造タイプ *type-name* のどの属性の名前とも同一にすることはできません (SQLSTATE 42711)。*fullselect* で指定した最初の列は、 REF(*type-name*) というタイプでなければなりません (キャストして適切なタイプにする必要があるかもしれません)。UNCHECKED を指定しない場合、索引 (主キー、ユニーク制約、ユニーク索引、または OID 列) を使用してユニーク性を強制できる列 (NULL 可能ではない) に基づいている必要があります。この列をオブジェクト *ID 列* または *OID 列* といいます。 USER GENERATED というキーワードは、行を挿入する際にユーザーが OID 列の初期値を提供しなければならないことを指しています。行を挿入した後は、OID 列を更新することはできません (SQLSTATE 42808)。

UNCHECKED
固有であることをシステムが証明できない場合でも、型付きビュー定義のオブジェクト ID の列を固有であると見なすように定義します。この属性は、次のような型付きビュー階層に定義されている表またはビューでの使用を想定しています。すなわち、そのデータが固有性規則に準拠しているものの、システムが固有性を証明できる規則には準拠していないことをユーザーが認識しているという場合です。 UNCHECKED オプションは、複数の階層や従来型の表またはビューにまでわたっているビューの階層に必須のオプションです。 UNCHECKED を指定する場合、ユーザーの責任でビューの各行にユニークな OID が確実にあるようにします。ユーザーがこの特性を保証しなかったために、ビューに重複した OID 値が入ってしまうと、ユニークでない OID 値のどれかを含むパスの式または DEREF 演算子はエラーになります (SQLSTATE 21000)。

with-options
型付きビューの列に適用される追加オプションを定義します。

column-name **WITH OPTIONS**
追加オプションを指定する列の名前を指定します。 *column-name* は、ビューの *type-name* に定義されている (継承されてはいない) 属性名に対応していなければなりません。 この列は参照タイプである必要があります (SQLSTATE 42842)。 また、すでにスーパービューに存在する列に対応することはできません (SQLSTATE 428DJ)。 列名は、ステートメント内の 1 つの WITH OPTIONS SCOPE 文節に 1 回しか指定できません (SQLSTATE 42613)。

SCOPE
参照タイプ列の有効範囲を指定します。逆参照演算子の左オペランド、または DEREF 関数の引き数として使用する列には、すべて有効範囲を指定する必要があります。

ターゲット表またはターゲット・ビューが定義するために、後続する ALTER VIEW ステートメント (有効範囲が継承されていない場合) まで、参照タイプ列の有効範囲指定を遅らせることができます (通常は、相互参照表および相互参照ビューの場合に適用する)。 ビューの参照タイプ列で有効範囲が指定されていないのに、基礎表またはビュー列の有効範囲が指定され

た場合、基礎列の有効範囲は参照タイプ列によって継承されます。 基礎表またはビューの列に有効範囲がない場合には、この列に有効範囲は指定されません。 有効範囲と参照タイプ列についての詳細は、481 を参照してください。

typed-table-name
型付き表の名前。この表は既に存在しているものか、作成する表と同じ名前のものでなければなりません (SQLSTATE 42704)。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-table-name* のタイプを表します (SQLSTATE 428DM)。値が *typed-table-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

typed-view-name
型付きビューの名前。このビューは既に存在しているものか、作成するビューと同じ名前のものでなければなりません (SQLSTATE 42704)。 *column-name* のデータ・タイプは REF(*S*) でなければなりません。 *S* は *typed-view-name* のタイプを表します (SQLSTATE 428DM)。値が *typed-view-name* の既存行を実際に参照していることを確認するための、 *column-name* の既存値の検査は行われません。

READ ONLY
列を読み取り専用列として指定します。このオプションは、列を読み取り専用に指定し、サブビュー定義は、同じ列の式を暗黙的に読み取り専用に指定することができます。

AS
ビュー定義を指定します。

WITH *common-table-expression*
後続の全選択で使用する共通表式を定義します。型付きビューを定義するときには、共通表式は指定できません。

全選択 (fullselect)
ビューを定義します。どのような場合でも、ビューは、SELECT が実行されたとしたらその結果となるような行で構成されます。全選択でホスト変数、パラメーター・マーカー、または宣言済み一時表を参照することはできません。ただし、パラメーター化されたビューを SQL 表関数として作成することは可能です。

全選択では、FROM 文節に SQL データ変更ステートメントを組み込めません (SQLSTATE 428FL)。

型付きビューおよびサブビューの場合 : *fullselect* は、以下の規則に準拠していなければなりません。そうでない場合、エラーが戻されます (SQLSTATE 428EA 何も指定されていない場合)。

- 全選択に、DBPARTITIONNUM または HASHEDVALUE 関数、非 deterministic 関数、または外部アクションを持つように定義されている関数への参照を含めることはできません。
- ビューの本体は、単一の副選択か複数の副選択の UNION ALL で構成する必要があります。ビューの本体に直接加わっている各副選択を、ビューの*分岐*と呼びます。ビューには、1 つかそれ以上の分岐がある場合があります。

- 各分岐の FROM 文節は、単一の表またはビュー (その分岐の*基礎* 表またはビューといい、必ずしも型付きではない) で構成される必要があります。
- 各分岐の基礎表またはビューは、別々の階層にする必要があります (つまり、ビューは、同じ階層内の基礎表またはビューが付いた複数の分岐を持つことはできません)。
- 型付きビュー定義の分岐はいずれも GROUP BY または HAVING を指定できません。
- ビューの本体に UNION ALL が含まれる場合、階層内にあるルート・ビュー の OID 列に UNCHECKED オプションを指定する必要があります。

ビューおよびサブビューの階層の場合 : BR1 および BR2 が、階層内のビュー定義に現れる分岐になるようにします。 T1 を BR1 の基礎表またはビューに、T2 を BR2 の基礎表またはビューにします。 この場合は以下のようになります。

- T1 および T2 が同じ階層でない場合、ビュー階層にあるルート・ビューの OID 列に UNCHECKED オプションを指定する必要があります。
- T1 および T2 が同じ階層にある場合、行セットが結合しないことを十分保証する述部または ONLY 文節を、 BR1 および BR2 に含める必要があります。

EXTEND AS を使って定義された型付きのビューの場合: サブビューの本体内の各分岐について:

- 各分岐の基礎表は、即時スーパービューのいくつかの基礎表の副表 (必ずしも適切ではない) でなければなりません。
- SELECT リストの式は、サブビューの非継承列に割り当てることのできるものでなければなりません (SQLSTATE 42854)。

AS (EXTEND なし) を使って定義された型付きサブビューの場合:

- サブビューの本体内にあるそれぞれの分岐について、SELECT リストにある式は、サブビューの継承列と非継承列の宣言済みタイプに割り当てられるようにする必要があります (SQLSTATE 42854)。
- サブビューで指定した階層上のそれぞれの分岐の OID 式は、ルート・ビュー内の同じ階層上の分岐の OID 式と同じでなければなりません (キャスト以外)。
- スーパービュー内の READ ONLY として (暗黙的または明示的に) 指定されていない列の式は、そのサブビュー内の同じ基礎階層上のすべての分岐と同じでなければなりません。

WITH CHECK OPTION

ビューによって挿入または更新される行すべてが、ビューの定義に従っていなければならないという制約を指定します。 ビューの定義に従わない行とは、ビューの検索条件を満たしていない行です。

WITH CHECK OPTION は、以下のいずれかの条件が真である場合には指定できません。

- ビューが読み取り専用である場合 (SQLSTATE 42813)。挿入が許されていない更新可能なビューに対して WITH CHECK OPTION を指定すると、制約は更新にのみ適用されます。

- ビューが、NODENUMBER または PARTITION 関数、非 deterministic 関数、または外部アクションを伴う関数を参照する場合 (SQLSTATE 42997)。
- ニックネームがビューの更新の対象である場合。
- ビューの更新の対象である INSTEAD OF トリガーが定義されているビューである場合 (SQLSTATE 428FQ)。

WITH CHECK OPTION を省略すると、ビューを使用するどのような挿入操作または更新操作のチェックにおいても、ビューの定義は使用されません。 ただし、ビューが WITH CHECK OPTION が指定された他のビューに直接または間接的に従属する場合には、挿入操作または更新操作の過程で、何らかのチェックが行われる場合があります。 この場合、ビューの定義が使用されるわけではないため、ビューの定義に従っていないビューを介して、行が挿入または更新される可能性があります。

CASCADED
ビュー *V* に対する WITH CASCADED CHECK OPTION 制約は、 *V* が従属するいずれかの更新可能ビューから、制約としての検索条件を *V* が継承することを意味します。 さらに、*V* に従属するすべての更新可能ビューも、このような制約の対象になります。 したがって、*V* の検索条件と、 *V* が従属している各ビューの検索条件との AND を取ったものが、 *V* あるいは *V* に従属するいずれかのビューの挿入または更新に対して適用される制約となります。

LOCAL
ビュー *V* に対する WITH LOCAL CHECK OPTION 制約は、 *V* の検索条件が、 *V* または *V* に従属するいずれかのビューの挿入あるいは更新に対する制約として適用されることを意味しています。

次の例は、CASCADED と LOCAL の差異を示しています。 次のような更新可能なビューを想定します (Y は、下記の表の列見出しに示しているように、LOCAL または CASCADED に置き換えます)。

```
V1 defined on table T
V2 defined on V1 WITH Y CHECK OPTION
V3 defined on V2
V4 defined on V3 WITH Y CHECK OPTION
V5 defined on V4
```

次の表は、挿入または更新された行を検査するのに使われる検索条件を示しています。

	Y が LOCAL の場合	**Y が CASCADED の場合**
V1 でのチェック条件:	対象となるビューなし	対象となるビューなし
V2 でのチェック条件:	V2	V2、V1
V3 でのチェック条件:	V2	V2、V1
V4 でのチェック条件:	V2、V4	V4、V3、V2、V1
V5 でのチェック条件:	V2、V4	V4、V3、V2、V1

また、次のような更新可能ビューについても考えてみます。これは、デフォルトの CASCADED オプションを使用した場合の WITH CHECK OPTION の効果を示しています。

```
CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CHECK OPTION

CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100
```

次の INSERT ステートメントは *V1* を使用するものですが、 *V1* に WITH CHECK OPTION が指定されておらず、また *V1* が、WITH CHECK OPTION の指定された他のどのビューにも従属していないため、このステートメントは成功します。

```
INSERT INTO V1 VALUES(5)
```

次の INSERT ステートメントは *V2* を使用するものですが、 *V2* に WITH CHECK OPTION が指定されており、挿入 (INSERT) によって *V2* の定義に従っていない行が作成されるため、このステートメントはエラーになります。

```
INSERT INTO V2 VALUES(5)
```

次の INSERT ステートメントでは *V3* を使用しています。 *V3* に WITH CHECK OPTION は指定されていませんが、これは、WITH CHECK OPTION の指定された *V2* の従属であるため、エラーになります (SQLSTATE 44000)。

```
INSERT INTO V3 VALUES(5)
```

次の INSERT ステートメントも、*V3* を使用しています。これは *V3* の定義に準拠していませんが、成功します (*V3* には WITH CHECK OPTION が指定されていません)。これは、WITH CHECK OPTION の指定された *V2* の定義に従ったものになっています。

```
INSERT INTO V3 VALUES(200)
```

WITH NO ROW MOVEMENT または **WITH ROW MOVEMENT**
基礎表のチェック制約に違反する方法で行が更新されたときに、更新可能 UNION ALL ビューに対して行うアクションを指定します。デフォルトは WITH NO ROW MOVEMENT です。

WITH NO ROW MOVEMENT
基礎表のチェック制約に違反する方法で行が更新されたときに、エラー (SQLSTATE 23513) を戻すよう指定します。

WITH ROW MOVEMENT
表のチェック制約に対する違反があっても、更新された行を該当する基礎表に移動させるよう指定します。

行の移動には、チェック制約に違反する行の削除と、それらの行のビューへの再挿入が関係します。 WITH ROW MOVEMENT 文節を指定できるのは、 UNION ALL ビューの列がすべて更新可能になっている場合だけです (SQLSTATE 429BJ)。行が削除されたのと同じ基礎表に行が挿入される (おそらく、トリガー起動の後) 場合は、エラーが戻されます (SQLSTATE 23524)。 WITH ROW MOVEMENT 文節を使用して定義されたビューには、最外部の全選択を除き、ネストされた UNION ALL 操作を含めることはできません (SQLSTATE 429BJ)。

注:

- ***互換性:***

  - 以前のバージョンの DB2 との互換性:
    - FEDERATED キーワードは、キーワード CREATE および VIEW の間に指定できます。しかし FEDERATED キーワードは無視されます。これはフェデレーテッド・オブジェクトがビュー定義で使用される場合は、警告は戻されないからです。
- まだ存在していないスキーマ名を用いてビューを作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙的に作成されます。 そのスキーマの所有者は SYSIBM です。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。
- ビュー列は、列が式から派生するとき以外は、基本表またはビューの NOT NULL WITH DEFAULT 属性を継承します。 基本表に制約が定義されている場合、更新可能ビューに行が挿入または更新されるとき、それらの制約 (主キー、参照保全性、およびチェック制約) に対するチェックが行われます。
- 定義の中で作動不能ビューを使用して新しいビューを作成することはできません (SQLSTATE 51024)。
- このステートメントでは、宣言済み一時表をサポートしていません (SQLSTATE 42995)。
- ***削除可能ビュー:*** 削除操作の INSTEAD OF トリガーがビューに定義されている場合、または次のいずれかが当てはまる場合、ビューは*削除可能* です。
  - 外部全選択の各 FROM 文節には、基本表 (OUTER 文節なし)、削除可能ビュー (OUTER 文節なし)、削除可能なネストした表式、または削除可能な共通表式 (ニックネームが識別不能) のいずれかが 1 つだけ指定されている
  - 外部全選択に VALUES 文節が含まれない
  - 外部全選択に GROUP BY 文節も HAVING 文節も含まれない
  - 外部全選択の選択リストに列関数が含まれない
  - 外部全選択に、UNION ALL を除く SET 演算 (UNION、EXCEPT、または INTERSECT) が含まれない
  - UNION ALL のオペランドの基本表が同じ表ではなく、各オペランドが削除可能
  - 外部全選択の選択リストに DISTINCT が含まれない
  - 最外部の全選択の FROM 文節に *data-change-table-reference* が含まれない
- ***更新可能ビュー:*** 更新操作の INSTEAD OF トリガーがビューに定義されている場合、または次のいずれかが当てはまる場合、ビューの列は*更新可能* です。
  - ビューが削除可能 (削除の INSTEAD OF トリガーとは無関係) であり、列の解決結果が基本表の列 (逆参照操作は使用しない) となり、READ ONLY オプションが指定されない
  - ビューの全選択に UNION ALL が含まれる場合、 UNION ALL のオペランドの対応するすべての列のデータ・タイプおよびデフォルト値が正確に一致する (長さ、または精度と位取りを含む)

  ビューのいずれかの 列が更新可能なら、ビューは更新可能です。
- ***挿入可能ビュー:***

  - 挿入操作の INSTEAD OF トリガーがビューに定義されている場合、またはビューの少なくとも 1 つの列が更新可能であり、ビューの全選択に UNION ALL が含まれない場合、そのビューは挿入可能です。
  - 特定の行が基本表のうちの正確に 1 つのチェック制約を満たしている場合にのみ、その行をビュー (UNION ALL を含む) に挿入できます。
  - 更新不可の列が含まれているビューに挿入するには、それらの列を列リストから除外する必要があります。
- ***読み取り専用ビュー:*** ビューが*読み取り専用* であるのは、削除可能でも、更新可能でも、挿入可能でもない 場合です。

  ビューが読み取り専用かどうかは、SYSCAT.VIEWS カタログ・ビューの READONLY 列に示され、 INSTEAD OF トリガーが考慮されることはありません。
- 共通表式とネストした表式は、削除可能かどうか、更新可能かどうか、挿入可能かどうか、または読み取り専用かどうかを判別する上で、同じ一連の規則に従います。
- ***作動不能ビュー:*** *作動不能ビュー* とは、 SQL ステートメントで使用できなくなったビューのことです。ビューは、次の場合に作動不能になります。
  - ビュー定義が従属している特権が取り消された場合
  - ビュー定義が従属している表、ニックネーム、別名、または関数などのオブジェクトがドロップされた場合
  - ビュー定義が従属しているビューが作動不能になった場合
  - ビュー定義のスーパービューであるビュー (サブビュー) が作動不能になった場合

  実際には、作動不能ビューとは、ビュー定義が間違ってドロップされたビューです。たとえば、別名がドロップされると、その別名を使用して定義されているビューすべてが作動不能になります。それに従属するすべてのビューも作動不能になり、そのビューに従属するパッケージは無効になります。

  作動不能ビューを明示的に再作成するか、あるいはドロップされるまで、その作動不能ビューを使用するステートメントのコンパイルはできません (SQLSTATE 51024)。ただし、CREATE ALIAS、CREATE VIEW、DROP VIEW、および COMMENT ON TABLE の各ステートメントは例外です。作動不能ビューが明示的にドロップされるまで、その修飾名を使って別の表や別名を作成することはできません (SQLSTATE 42710)。

  作動不能ビューは、作動不能ビューの定義テキストを使用して、 CREATE VIEW ステートメントを発行することにより、再作成することができます。このビュー定義テキストは、SYSCAT.VIEWS カタログの TEXT 列に保管されます。作動不能ビューを再作成する場合は、他のユーザーがそのビューに対して必要となる特権すべてを明示的に付与する必要があります。これは、ビューが作動不能と見なされると、ビューのすべての許可レコードが削除されるためです。作動不能ビューを再作成するために、それを明示的にドロップする必要はありません。作動不能ビューと同じ *view-name* を指定して CREATE VIEW ステートメントを発行すると、作動不能ビューは置き換えられ、CREATE VIEW ステートメントは警告を戻します (SQLSTATE 01595)。

作動不能ビューであることは、SYSCAT.VIEWS カタログ・ビューの VALID 列が X、また SYSCAT.TABLES カタログ・ビューの STATUS 列が X であることによって示されます。

- ***特権:***

  ビューの定義者は、ビューに対する SELECT 特権と、ビューをドロップする権利を常に与えられます。ビューの定義者は、その定義者が全選択で指定されたすべての基本表、ビューまたはニックネームに対する CONTROL 特権を持っている場合、あるいは定義者が SYSADM 権限または DBADM 権限を持っている場合にのみ、そのビューに対する CONTROL 特権が付与されます。

  ビューの定義者は、そのビューが読み取り専用でなく、定義者が基礎となるオブジェクトに対して対応する特権を持っている場合に、そのビューに対する INSERT、 UPDATE、列レベルの UPDATE、または DELETE の特権を与えられます。

  WITH ROW MOVEMENT でビューが定義された場合は、定義者がビューのすべての列に対する UPDATE 特権を持っており、またすべての基礎表およびビューに対する INSERT および DELETE 特権を持っているなら、その定義者に、そのビューのみでの UPDATE 特権が与えられます。

  ビューの定義者にそれらの特権が与えられるのは、それらの特権の派生元の特権がビューの作成時に存在している場合に限ります。定義者は、これらの特権を直接持っているか、または PUBLIC の特権として持っていることが必要です。特権は、フェデレーテッド・データベースのニックネームにビューを定義するときには考慮されません。ただし、ニックネームにビューを使用する場合、ユーザーの許可 ID には、そのニックネームがデータ・ソースで参照する表またはビューに対する適切な選択特権がなければなりません。もしその特権がなければ、エラーが戻されます。ビューの定義者がメンバーであるグループを持つ特権は考慮されません。

  サブビューが作成されると、すぐ上のスーパービューに対して持っている SELECT 特権が自動的にサブビューに対しても付与されます。

- ***有効範囲および REF 列:***

  ビュー定義の全選択で参照タイプ列を選択する際には、必要なターゲット・タイプと有効範囲について考慮してください。

  – 必要なターゲット・タイプおよび有効範囲が基本表または基礎ビューのものと同じ場合には、参照列をそのまま選択することができます。
  – 有効範囲を変更する必要がある場合には、WITH OPTIONS SCOPE 文節を使って、必要な有効範囲の表とビューを定義します。
  – 参照のターゲット・タイプを変更する必要がある場合には、まず列を参照の対象となっている参照タイプにキャストしてから、新規の参照タイプへもキャストする必要があります。この場合、有効範囲は参照タイプへのキャストで指定できますし、 WITH OPTIONS SCOPE 文節を使っても指定できます。 たとえば、REF(TYP1) SCOPE TAB1 として定義された Y 列を選択したとしましょう。 そして、この列を REF(VTYP1) SCOPE VIEW1 として定義するとします。 この場合、選択リスト項目は次のようになります。

    ```
    CAST(CAST(Y AS VARCHAR(16) FOR BIT DATA) AS REF(VTYP1) SCOPE VIEW1)
    ```

- ***ID 列:*** ビューの列は、ビュー定義の全選択内にある対応する列のエレメントが表の ID 列の名前であるか、基本表の ID 列の名前に直接または間接的にマップされたビューの列の名前である場合に ID 列と見なされます。

  これ以外の場合はすべて、ビューの列は ID のプロパティーを取得しません。たとえば、以下のようになります。

  – ビュー定義の選択リストに ID 列の名前のインスタンスが複数含まれている (つまり、同じ列を複数回選択している) 場合
  – ビュー定義に結合が関与している場合
  – ビュー定義の列に ID 列を参照する式が含まれている場合
  – ビュー定義に UNION が含まれている場合

  挿入先のビューにおいて、ビュー定義の選択リストに直接または間接的に基本表の ID 列の名前が含まれている場合は、 INSERT ステートメントが基本表の ID 列を直接参照する場合と同じ規則が適用されます。
- ***フェデレーテッド・ビュー:*** フェデレーテッド・ビューとは、全選択内にあるいずれかのニックネームへの参照を含むビューのことです。この種のニックネームが存在する場合、そのビューで使用する許可モデルは、ビューが照会で順番に参照されるときに変更されます。

  ビューを作成しても、ビュー定義者がニックネームの基礎データ・ソース表またはビューにアクセスできるかどうかを判別する特権検査は行われません。フェデレーテッド・データベースでの表またはビューに対する特権検査は、最低でもそうしたオブジェクトに対する SELECT 特権をビュー定義者に要求することにより、通常どおり行われます。

  フェデレーテッド・ビューが照会で順番に参照される場合、その照会を発行したデータ・ソースおよび許可 ID (または、その照会がマッピングするリモート許可 ID) に対する照会になるニックネームには、データ・ソース表またはビューにアクセスするのに必要な特権がなければなりません。フェデレーテッド・ビューを参照する照会を発行する許可 ID には、フェデレーテッド・サーバーに存在する (フェデレーテッドでない) 表またはビューに対する追加の特権は必要ありません。
- ***ROW MOVEMENT、トリガー、および制約:*** WITH ROW MOVEMENT 文節を使用して定義されたビューが更新される場合、トリガーや制約の操作のシーケンスは次のようになります。

  1. 最終的には移動される行も含め、更新されるすべての行に対して BEFORE UPDATE トリガーが活動化されます。
  2. 更新操作が処理されます。
  3. 更新されるすべての行に対して制約が処理されます。
  4. 更新操作の後、制約を満たしたすべての行に対し、作成された順番で AFTER UPDATE トリガーが (行レベルとステートメント・レベルの両方で) 活動化されます。これは UPDATE ステートメントであるため、すべての基礎表に対して、UPDATE ステートメント・レベルのすべてのトリガーが活動化されます。
  5. 更新操作の後、制約を満たさなかったすべての行 (移動される行) に対して BEFORE DELETE トリガーが活動化されます。

6. 削除操作が処理されます。
7. 削除されるすべての行に対して制約が処理されます。
8. 削除されるすべての行に対し、作成された順番で AFTER DELETE トリガーが (行レベルとステートメント・レベルの両方で) 活動化されます。ステートメント・レベルのトリガーは、削除操作に関係する表に対してのみ活動化されます。
9. 挿入されるすべての行 (つまり、移動される行) に対して BEFORE INSERT トリガーが活動化されます。 BEFORE INSERT トリガーの新しい遷移表には、ユーザーからの入力データが含まれています。
10. 入力操作が処理されます。
11. 挿入されるすべての行に対して制約が処理されます。
12. 挿入されるすべての行に対し、作成された順番で AFTER INSERT トリガーが (行レベルとステートメント・レベルの両方で) 活動化されます。ステートメント・レベルのトリガーは、挿入操作に関係する表に対してのみ活動化されます。

- ***ネストされた UNION ALL ビュー:*** UNION ALL を使用して定義されたビューや、やはり UNION ALL を使用して定義されたビューを直接または間接的に基礎として使用しているビューは、いずれかのビューが WITH ROW MOVEMENT 文節を使用して定義されていると、更新できません (SQLSTATE 429BK)。

例:

例 1: PROJECT 表に基づくビュー MA_PROJ を作成します。このビューには、文字 'MA' で始まるプロジェクト番号 (PROJNO) を持つ行だけを入れます。

```
CREATE VIEW MA_PROJ  AS SELECT *
  FROM PROJECT
   WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

例 2: 例 1 と同様にビューを作成します。ただし、プロジェクト番号 (PROJNO)、プロジェクト名 (PROJNAME)、およびプロジェクトの責任者 (RESPEMP) の列だけを選択します。

```
CREATE VIEW MA_PROJ
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

例 3: 例 2 と同様のビューを作成します。ただし、ビューの中でプロジェクトの責任者の列を IN_CHARGE と呼びます。

```
CREATE VIEW MA_PROJ
  (PROJNO, PROJNAME, IN_CHARGE)
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

注: 列名のいずれか 1 つだけを変更する場合でも、ビューの 3 つの列すべての名前を MA_PROJ の後の括弧内に指定する必要があります。

例 4: PRJ_LEADER という名前のビューを作成します。このビューには、PROJECT 表の最初の 4 つの列 (PROJNO、 PROJNAME、 DEPTNO、 RESPEMP) と、プロジェクトの責任者 (RESPEMP) のラストネーム (LASTNAME) を入れま

す。ラストネームは、EMPLOYEE 表の EMPNO を PROJECT 表の RESPEMP と突き合わせることによって、 EMPLOYEE 表から入手します。

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
```

例 5: 例 4 と同様のビューを作成します。ただし、列 PROJNO、 PROJNAME、DEPTNO、 RESPEMP、および LASTNAME に加えて、担当者の給与総額 (SALARY + BONUS + COMM) を入れます。また、平均スタッフ数 (PRSTAFF) が 1 より大きいプロジェクトだけを選択します。

```
CREATE VIEW PRJ_LEADER
(PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, TOTAL_PAY )
AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, SALARY+BONUS+COMM
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
  AND PRSTAFF > 1
```

全選択で、式 SALARY+BONUS+COMM に TOTAL_PAY という名前を付けることによって、次のように、列名リストの指定を省略することができます。

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP,
            LASTNAME, SALARY+BONUS+COMM AS TOTAL_PAY
    FROM PROJECT, EMPLOYEE
    WHERE RESPEMP = EMPNO AND PRSTAFF > 1
```

例 6: 次の図に示す表とビューがあると想定します。

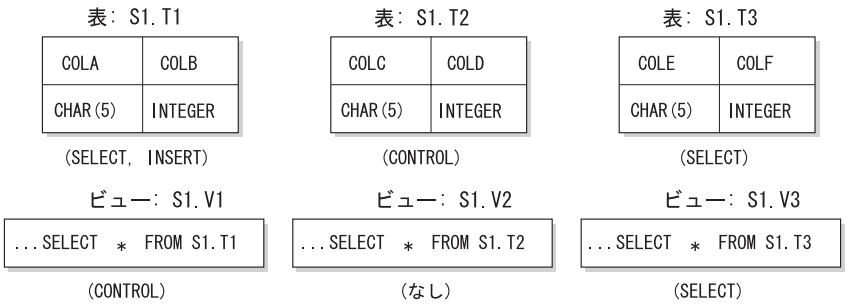


図 1. 例 6 の表とビュー

ユーザー ZORPIE (DBADM または SYSADM のいずれの権限も持たない) は、各オブジェクトの下の大括弧内に示している権限を与えられています。

1. 次のステートメントによって作成するビューに対して、ZORPIE は CONTROL 特権を獲得します。

   ```
   CREATE VIEW VA AS SELECT * FROM S1.V1
   ```

   これは、ZORPIE が S1.V1 に対して CONTROL 権限を持っているからです。DBADM または SYSADM のいずれかの権限を持つユーザーが、 S1.V1 に対する CONTROL 権限を ZORPIE に与えている必要があります。基礎となる基本表に対して、どのような特権が与えられているかは関係ありません。
2. ZORPIE は、次のようなビューの作成は許されません。

   ```
   CREATE VIEW VB AS SELECT * FROM S1.V2
   ```

これは、ZORPIE には、S1.V2 に対する CONTROL も SELECT も与えられていないからです。基礎となる基本表 (S1.T2) に対して CONTROL を与えられているかどうかは、関係ありません。

3. 次のステートメントによって作成するビューに対して、ZORPIE は CONTROL 特権を獲得します。

```
CREATE VIEW VC (COLA, COLB, COLC, COLD)
  AS SELECT * FROM S1.V1, S1.T2
  WHERE COLA = COLC
```

これは、ZORPIE.VC の全選択では、ビュー S1.V1 と S1.T2 を参照しており、ZORPIE はその両方に対する CONTROL を持っているからです。ビュー VC は読み取り専用で、INSERT、UPDATE、または DELETE のいずれの権限も ZORPIE には与えられないことに注意してください。

4. 次のステートメントによって作成するビューに対して、ZORPIE は SELECT 特権を入手します。

```
CREATE VIEW VD (COLA,COLB, COLE, COLF)
  AS SELECT * FROM S1.V1, S1.V3
  WHERE COLA = COLE
```

これは、ZORPIE.VD の全選択で 2 つのビュー S1.V1 および S1.V3 を参照しており、 ZORPIE はその 1 つに対しては SELECT 特権を、もう 1 つに対しては CONTROL 特権を与えられているからです。 ZORPIE.VD に対する特権として、 2 つの特権のうち低い方の特権である SELECT が ZORPIE に与えられます。

5. 以下のビュー定義では、ZORPIE はビュー VE に対して WITH GRANT OPTION を伴う INSERT、 UPDATE および DELETE の各特権と、SELECT 特権を与えられます。

```
CREATE VIEW VE
   AS SELECT * FROM S1.V1
  WHERE COLA > ANY
         (SELECT COLE FROM S1.V3)
```

ZORPIE の VE に対する特権は、主として S1.V1 に対する特権によって決定されます。 S1.V3 は副照会で参照されているだけなので、ビュー VE の作成には S1.V3 に対する SELECT 特権だけが必要です。ビューの定義者は、ビューの定義で参照されているすべてのオブジェクトに対して CONTROL を持っている場合のみ、そのビューに対して CONTROL を入手します。 ZORPIE は S1.V3 に対する CONTROL を持っていないので、VE に対する CONTROL は与えられません。

関連資料:

- 263 ページの『CREATE FUNCTION (SQL スカラー、表、または行)』
- *SQL リファレンス 第 1 巻* の『SQL 照会』

CREATE WRAPPER

CREATE WRAPPER ステートメントは、ラッパーをフェデレーテッド・サーバーに登録します。ラッパーは、フェデレーテッド・サーバーがデータ・ソースの特定のタイプと対話するためのメカニズムです。

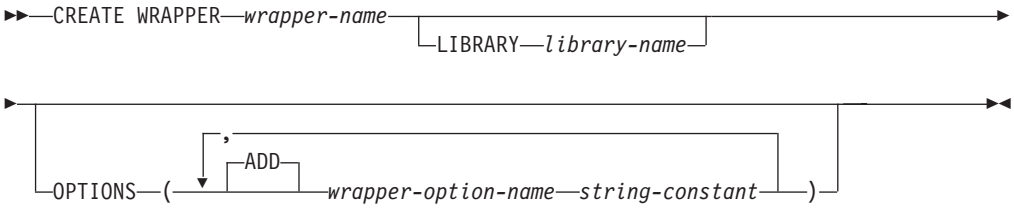
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。パッケージに対して DYNAMICRULES が有効な場合、動的にプリペアーすることが可能なステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文:



説明:

wrapper-name
ラッパーの名前を指定します。次のような名前にすることができます。

- 事前定義名。事前定義名を指定すると、フェデレーテッド・サーバーは *library-name* に自動的にデフォルト値を割り当てます。
- ユーザーが指定する名前。ユーザー指定の名前を提供する場合は、そのラッパーおよびオペレーティング・システムとともに使用する適切な *library-name* も指定する必要があります。

LIBRARY *library-name*
ラッパー・ライブラリー・モジュールを含むファイルの名前を指定します。

ライブラリー名は、絶対パス名または単に基本名 (パスなし) として指定できます。基本名だけを指定する場合は、ライブラリーは DB2 インストール・パスの lib (UNIX)、または bin (Windows) サブディレクトリーに存在していなければなりません。 *library-name* は単一引用符で囲む必要があります。

LIBRARY オプションが必要なのは、ユーザーが指定した *wrapper-name* を使用する場合だけです。事前定義された *wrapper-name* を指定する場合には、このオプションは使用しません。

OPTIONS (ADD *wrapper-option-name string-constant*, **...)**
ラッパー・オプションは、ラッパーを構成するため、または DB2 によるラッパーの使用法を定義するために使用します。 *wrapper-option-name* はオプションの名前です。 *string-constant* は、ラッパー・オプションの設定を指定します。

string-constant は単一引用符で囲む必要があります。すべてのラッパーによって使用されるラッパー・オプションもあれば、特定のラッパー固有のオプションもあります。

例:

例 1: Oracle データ・ソースにアクセスするために、フェデレーテッド・サーバー上の NET8 ラッパーを登録します。 Oracle データ・ソースへのアクセスに使用できるラッパーは 2 つあり、 *NET8* はそのうちの 1 つの事前定義名です。

```
CREATE WRAPPER NET8
```

例 2: ODBC データ・ソースにアクセスするために、 Linux オペレーティング・システムを使用する DB2 フェデレーテッド・サーバー上にラッパーを登録します。フェデレーテッド・データベースに登録されているラッパーに *odbc* という名前を割り振ります。 ODBC Driver Manager を含むライブラリーの絶対パスは、ラッパー・オプション *MODULE '/usr/lib/odbc.so'* に定義されています。

```
CREATE WRAPPER odbc OPTIONS (MODULE '/usr/lib/odbc.so')
```

例 3: ODBC データ・ソースにアクセスするために、 Windows オペレーティング・システムを使用する DB2 フェデレーテッド・サーバー上にラッパーを登録します。 ODBC ラッパーのライブラリー名は *'db2rcodbc.dll'* です。

```
CREATE WRAPPER odbc LIBRARY 'db2rcodbc.dll'
```

例 4: Entrez データ・ソースにアクセスするために、 AIX オペレーティング・システムを使用する DB2 フェデレーテッド・サーバー上にラッパーを登録します。ラッパーの名前を *entrez_wrapper* と指定します。 AIX フェデレーテッド・サーバー上で、 Entrez ラッパーのライブラリー・ファイルは *libdb2lsentrez.a* です。 Entrez ラッパーをフェデレーテッド・サーバーに登録する場合は、EMAIL オプションが必要です。

```
CREATE WRAPPER entrez_wrapper LIBRARY 'libdb2lsentrez.a'
OPTIONS (EMAIL 'jeff@someplace.com')
```

DECLARE CURSOR

DECLARE CURSOR ステートメントは、カーソルを定義します。

呼び出し:

対話式SQLの機能として、対話式に実行するインターフェースが提供されているように見えますが、実はアプリケーション・プログラムに組み込まれているだけで、動的にプリペアーして実行することはできません。

許可:

「カーソルの SELECT ステートメント」という用語は、以下の許可規則を示すために使用されます。 カーソルの SELECT ステートメントは、次のいずれかです。

- *statement-name (ステートメント名)* によって識別され、準備される 選択ステートメント。
- 指定された *select-statement (選択ステートメント)*

カーソルの SELECT ステートメントに (直接または別名を使用して) 指定する表またはビューのそれぞれについて、ステートメントの許可 ID の特権に、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- *select-statement* で指定された表またはビューのそれぞれに対する以下のいずれかの特権。
  - 表またはビューに対する SELECT 特権、または
  - 表またはビューに対する CONTROL 特権

select-statement に SQL データ変更ステートメントが含まれている場合は、そのステートメントの許可要件も DECLARE CURSOR ステートメントに適用されます。

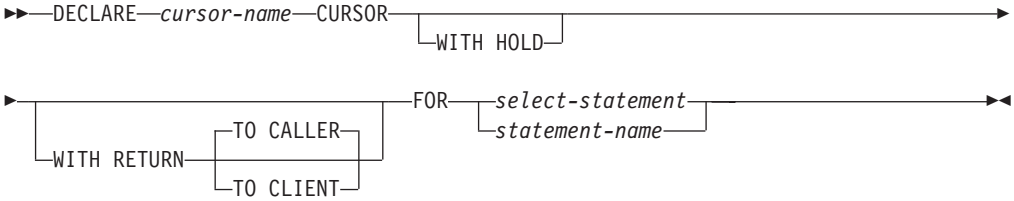
***statement-name* を指定した場合:**

- ステートメントの許可 ID は、ランタイム許可 ID になります。
- 許可検査は、選択ステートメントが準備される時点で行われます。
- 選択ステートメントの準備が成功しない限り、カーソルはオープンされません。

***select-statement* を指定した場合:**

- GROUP 特権は検査されません。
- ステートメントの許可 ID は、プログラム作成時に指定される許可 ID になります。

構文:



説明:

cursor-name
ソース・プログラムの実行時に作成されるカーソルの名前を指定します。 この名前は、ソース・プログラムに宣言されている他のカーソルの名前と同じであってはなりません。 カーソルは、その使用に先立ってオープンする必要があります。

WITH HOLD
複数の作業単位を通してリソースを維持します。 WITH HOLD カーソル属性の効果は次のとおりです。

- COMMIT で終了する作業単位の場合:
  - WITH HOLD として定義されたオープン・カーソルはオープンされたままです。 カーソルは、結果表の次の論理行の前に位置づけられます。

    WITH HOLD カーソルとの接続に対して、 COMMIT ステートメントの後で DISCONNECT ステートメントが出された場合、保留されたカーソルを明示的にクローズする必要があります。そうしない場合、 (SQL ステートメントが全く発行されていない場合でも単に WITH HOLD カーソルをオープンしたままにすることによって) その接続が作業を行っていると想定され、その DISCONNECT ステートメントはエラーになります。
  - オープンされている WITH HOLD カーソルの現行カーソル位置を保護するロック以外のすべてのロックが解放されます。保留されるロックには、表に対するロックと、並列環境の場合はカーソルが現在位置している行に対するロックがあります。 パッケージと動的 SQL セクション (存在する場合) に対するロックは保留されます。
  - WITH HOLD の定義されたカーソルに対して、COMMIT 要求の直後に有効な操作は、次のとおりです。
    - FETCH: カーソルの次の行を取り出します。
    - CLOSE: カーソルをクローズします。
  - UPDATE および DELETE CURRENT OF CURSOR は、同一作業単位内で取り出された行に対してのみ有効です。
  - LOB ロケーターは解放されます。
  - 以下によって変更された行のセットがコミットされます。
    - データ変更ステートメント
    - オープン WITH HOLD カーソルに組み込まれている、SQL データを変更するルーチン
- ROLLBACK で終了する作業単位の場合:
  - オープン・カーソルはすべてクローズされます。
  - その作業単位の過程で獲得したロックはすべて解除されます。
  - LOB ロケーターは解放されます。
- 特殊な COMMIT の場合:
  - パッケージは、パッケージをバインドすることによって明示的に再作成されるか、または無効になった後、それが初めて参照されるときに動的に再作成されることにより暗黙のうちに再作成されます。 保留されたカーソルはすべて、パッケージの再バインド時にはクローズされます。 そのような場合、それ以後の実行時にエラーになる場合があります。

WITH RETURN

この文節は、カーソルがストアード・プロシージャーからの結果セットとして使用されるよう意図されていることを示します。 WITH RETURN が使用されるのは、 DECLARE CURSOR ステートメントにストアード・プロシージャーのソース・コードが含まれている場合だけです。これ以外の場合は、プリコンパイラーがこの文節を受け入れても、この文節は効力を持ちません。

SQL プロシージャーでは、 WITH RETURN 文節を使用して宣言されたカーソルは SQL プロシージャーの終了後もクローズされずに残り、 SQL プロシージャーからの結果セットを定義します。そして、その他のオープン・カーソルは、SQL プロシージャーが終了するときにすべてクローズされます。外部ストアード・プロシージャー (LANGUAGE SQL を使用して定義されていないもの) では、すべてのカーソルのデフォルトは WITH RETURN TO CALLER です。したがって、外部プロシージャーの終了時に残っているオープン・カーソルがすべて結果セットと見なされます。

TO CALLER

カーソルが呼び出し側に結果セットを返すよう指定します。たとえば、他のストアード・プロシージャーから呼び出しが行われた場合は、そのストアード・プロシージャーに結果セットが返されます。また、呼び出し側がクライアント・アプリケーションであるなら、そのクライアント・アプリケーションに結果セットが返されます。

TO CLIENT

カーソルがクライアント・アプリケーションに結果セットを返すよう指定します。このカーソルは、中間にネストされたプロシージャーからは認識されません。関数またはメソッドがプロシージャーを直接または間接的に呼び出す場合は、結果セットはクライアントに返されず、プロシージャーの終了後にカーソルがクローズされます。

select-statement

カーソルの SELECT ステートメントを指定します。 その *select-statement* には、パラメーター・マーカーを含めることはできませんが、ホスト変数への参照は含めることができます。 参照されるホスト変数の宣言は、ソース・プログラムにおいて DECLARE CURSOR ステートメントよりも前になければなりません。

statement-name

カーソルの SELECT ステートメントは、カーソルのオープン時に *statement-name* によって指定される準備済み SELECT ステートメントです。 *statement-name* は、ソース・プログラムの他の DECLARE CURSOR ステートメントに指定されている *statement-name* と同じであってはなりません。

準備済み SELECT ステートメントについては、『PREPARE』 を参照してください。

注:

- 他のプログラムから呼び出されたプログラム、または同じプログラムの別のソース・ファイルから呼び出されたプログラムで、呼び出し側プログラムによってオープンされたカーソルを使用することはできません。
- SQL 以外の LANGUAGE を使用する、ネストされていないストアード・プロシージャーには、 WITH RETURN 文節を使用せずに DECLARE CURSOR が指定

されるとデフォルトで WITH RETURN TO CALLER を使用し、カーソルをクローズせずにプロシージャーに残すという性質があります。このようにすることによって、適当なクライアント・アプリケーションに結果セットを返すことができる以前のバージョンのストアード・プロシージャーにも対応することができます。この性質を無効にするには、プロシージャーでオープンされているカーソルをすべてクローズしてください。

- カーソルの SELECT ステートメントが CURRENT DATE、CURRENT TIME、または CURRENT TIMESTAMP を含む場合、これらの特殊レジスターを参照すると、それぞれの FETCH でそれぞれの同一の日時値が与えられます。この値は、カーソルがオープンされた時点で決まります。 この値は、カーソルのオープン時に決まります。
- データをより効率的に処理するために、データベース・マネージャーでは、リモート・サーバーからデータを検索するときに、読み取り専用カーソルに対してはデータ変更を禁止することができます。 FOR UPDATE 文節を使用するなら、データベース・マネージャーで、カーソルが更新可能かどうかを決めることができます。 更新可能性は、アクセス・パス選択を決めるためにも使用されます。 カーソルを位置指定 UPDATE または DELETE ステートメントで使用しない場合は、 FOR READ ONLY として宣言してください。
- オープン状態のカーソルは、結果表と、その表の行に対する相対位置を示します。 表は、カーソルの SELECT ステートメントによって指定される結果表です。
- カーソルは、以下の各項目が真となる場合に*削除可能* です。
  - 外部全選択の各 FROM 文節に、OUTER 文節を使用しないで、基本表または削除可能ビュー (ネストした表式や共通表式またはニックネームを指定できない) が指定されている
  - 外部全選択に VALUES 文節が含まれない
  - 外部全選択に GROUP BY 文節も HAVING 文節も含まれない
  - 外部全選択の選択リストに列関数が含まれない
  - 外部全選択に、UNION ALL を除く SET 演算 (UNION、EXCEPT、または INTERSECT) が含まれない
  - 外部全選択の選択リストに DISTINCT が含まれない
  - 外部全選択に ORDER BY 文節が含まれておらず (ORDER BY 文節がビューにネストされていてもよい)、 FOR UPDATE 文節が指定されていない
  - 選択ステートメントに FOR READ ONLY 文節が含まれない
  - 最外部の全選択の FROM 文節に *data-change-table-reference* が含まれない
  - 次の 1 つまたは複数が真である
    - FOR UPDATE 文節が指定されている
    - カーソルが静的に定義されており、STATICREADONLY BIND オプションが YES になっていない
    - LANGLEVEL BIND オプションが MIA または SQL92E である

  カーソルに関連する外部全選択の選択リスト内の列は、以下の各項目が真となる場合に、*更新可能* です。
  - カーソルが削除可能である

– 列の解決結果が基本表の列となる
– LANGLEVEL BIND オプションが MIA の場合、 SQL92E または select-statement が FOR UPDATE 文節を含んでいる (列が FOR UPDATE 文節で明示的または暗黙的に指定されている必要があります)

カーソルが*読み取り専用* であるのは、削除可能でない場合です。

カーソルは、以下の各項目が真となる場合に*未確定* です。
– 選択ステートメントが動的に準備される
– 選択ステートメントに FOR READ ONLY 文節も FOR UPDATE 文節も含まれていない
– LANGLEVEL BIND オプションが SAA1 である
– それ以外の点では、カーソルは削除可能カーソルの条件を満たしている

未確定カーソルは、BLOCKING BIND オプションが ALL の場合には読み取り専用と見なされます。そうでない場合は、更新可能と見なされます。

- CLI を使用して作成されたアプリケーション・プログラムによって呼び出されるストアード・プロシージャーの中のカーソルは、クライアント・アプリケーションに直接返される結果表を定義するために使用することができます。また、SQL プロシージャーが WITH RETURN 文節を使用して定義される場合に限り、そのプロシージャーの中のカーソルを呼び出し側の SQL プロシージャーに返すこともできます。
- WITH HOLD を宣言したカーソルから直接または間接的に呼び出されるルーチン内で宣言されるカーソルは、 WITH HOLD オプションを継承しません。したがって、ルーチン内のカーソルが明示的に WITH HOLD と定義されない限り、カーソルはアプリケーションの COMMIT によってクローズされます。

次のようなアプリケーションと 2 つの UDF について考慮します。

アプリケーション:

```
   DECLARE APPCUR CURSOR WITH HOLD FOR SELECT UDF1() ...
   OPEN APPCUR
   FETCH APPCUR ...
   COMMIT

UDF1:

   DECLARE UDF1CUR CURSOR FOR SELECT UDF2() ...
   OPEN UDF1CUR
   FETCH UDF1CUR ...

UDF2:

   DECLARE UDF2CUR CURSOR WITH HOLD FOR SELECT UDF2() ...
   OPEN UDF2CUR
   FETCH UDF2CUR ...
```

アプリケーションがカーソル APPCUR を取り出した後は、3 つのカーソルすべてがオープンになります。アプリケーションが COMMIT ステートメントを発行すると、 APPCUR は、WITH HOLD と宣言されているのでオープンのままになります。しかし、UDF1 では、カーソル UDF1CUR は、 WITH HOLD オプションを指定して定義されていないのでクローズされます。カーソル UDF1CUR がクローズされると、対応する選択ステートメント内のすべてのルーチン呼び出しが

完了します (最終呼び出しを受け取るように定義されている場合は、それを受け取ります)。 UDF2 が完了し、UDF2CUR がクローズされます。

例:

例 1: DECLARE CURSOR ステートメントは、SELECT の結果にカーソル名 C1 を関連付けます。

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DEPARTMENT
  WHERE ADMRDEPT = 'A00';
```

例 2: EMPLOYEE 表が、生成された列 WEEKLYPAY (年間の給与に基づいて週ごとの支払いを計算する) を追加するように調整されていると想定します。カーソルを宣言して、挿入される行からシステムが生成した列の値を取り出します。

```
EXEC SQL DECLARE C2 CURSOR FOR
  SELECT E.WEEKLYPAY
  FROM NEW TABLE
    (INSERT INTO EMPLOYEE
     (EMPNO, FIRSTNME, MIDINIT, LASTNAME, EDLEVEL, SALARY)
     VALUES('000420', 'Peter', 'U', 'Bender', 16, 31842) AS E;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『Select-statement』
- 107 ページの『CALL』
- 642 ページの『OPEN』
- 647 ページの『PREPARE』

関連サンプル:

- 『cursor.sqb -- How to update table data with cursor statically (MF COBOL)』
- 『tabsql.sqb -- Demonstrates common table expressions using SQL (MF COBOL)』
- 『fnuse.sqc -- How to use built-in SQL functions (C)』
- 『tbinfo.sqc -- How to get information at the table level (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『fnuse.sqC -- How to use built-in SQL functions (C++)』
- 『tbinfo.sqC -- How to get information at the table level (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『tut_read.sqC -- How to read tables (C++)』

DECLARE GLOBAL TEMPORARY TABLE

DECLARE GLOBAL TEMPORARY TABLE ステートメントは、現行セッションの一時表を定義します。宣言済み一時表の記述は、システム・カタログには現れません。これは永続的なものではなく、他のセッションと共用することもできません。同じ名前の宣言済みグローバル一時表を定義するセッションであっても、一時表の記述はそのセッションによって異なります。セッションが終了すると、表の行は削除され、一時表の記述はドロップされます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

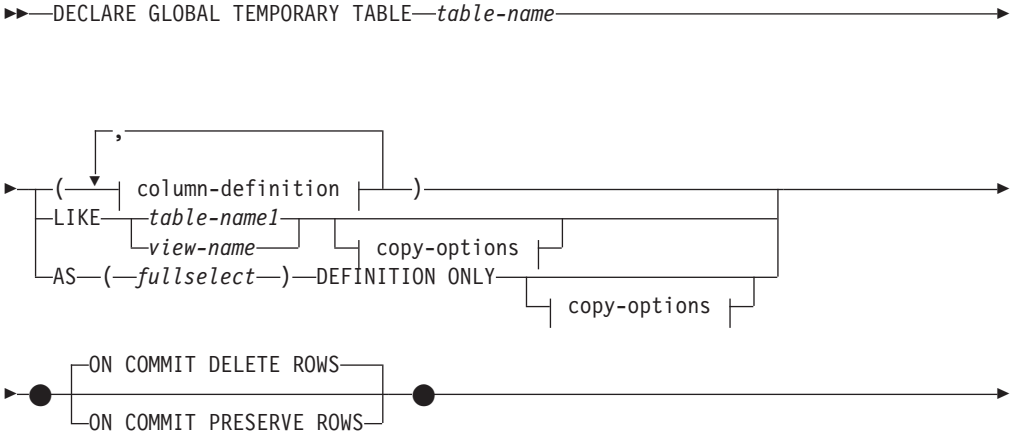
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

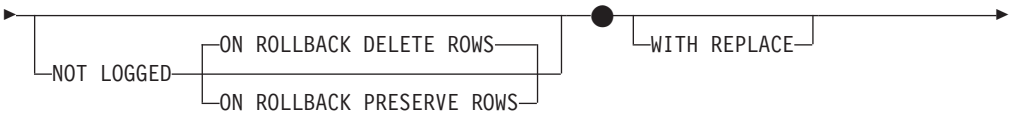
- SYSADM または DBADM 権限
- USER TEMPORARY 表スペースでの USE 権限

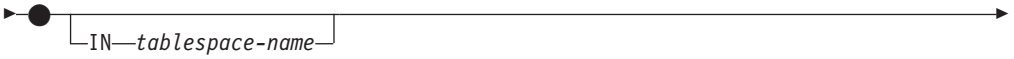
LIKE または全選択を使用して表を定義する場合、ステートメントの許可 ID の特権に、識別されているそれぞれの表またはビューに対する以下の権限が少なくとも 1 つ以上含まれている必要があります。

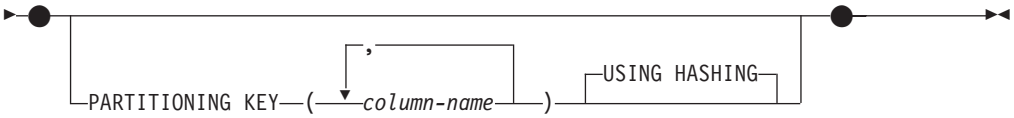
- その表またはビューに対する SELECT 特権
- その表またはビューに対する CONTROL 特権
- SYSADM または DBADM 権限

構文:

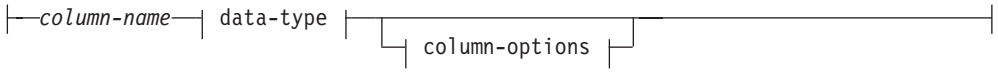




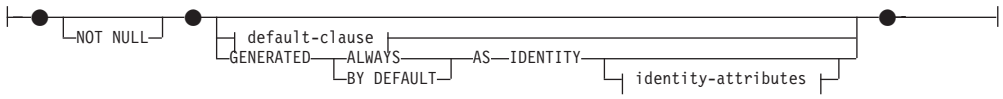




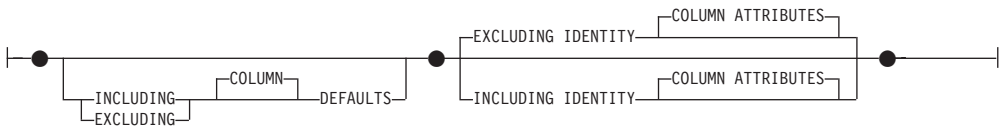
column-definition:



column-options:



copy-options:



説明:

table-name

一時表の名前を示します。修飾子を明示的に指定する場合は、SESSION でなければなりません。そうしないと、エラーになります (SQLSTATE 428EK)。修飾子が指定されなければ、暗黙的に SESSION が指定されます。

同じ *table-name* の宣言済みグローバル一時表を定義するセッションであっても、その宣言済みグローバル一時表の記述はそれぞれのセッションによって異なります。 *table-name* を使用する宣言済み一時表がセッション内にすでに存在している場合は、 WITH REPLACE 文節を指定する必要があります (SQLSTATE 42710)。

表、ビュー、別名、またはニックネームについては、同じ名前および同じスキーマ名 (SESSION) を持つものがカタログ内にすでに存在していても構いません。このような場合には、次のような処理が行われます。

- 宣言されているグローバル一時表 *table-name* は、正常に定義されます (エラーや警告は戻されません)。

- SESSION.*table-name* への参照はすべて、カタログ内ですでに定義されている SESSION.*table-name* ではなく、宣言済みグローバル一時表に対して行われます。

column-definition
一時表の列の属性を定義します。

column-name
表を構成する列の名前を指定します。名前を修飾したり、表の複数の列に対して同じ名前を使用することはできません (SQLSTATE 42711)。

表には、以下のものを指定できます。

- 4K ページ・サイズの場合、最大 500 列。列のバイト・カウントは 4 005 を超えてはなりません。
- 8K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 8 101 を超えてはなりません。
- 16K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 16 293 を超えてはなりません。
- 32K ページ・サイズの場合、最大 1 012 列。列のバイト・カウントは 32 677 を超えてはなりません。

詳細については、『CREATE TABLE』の『行サイズ』を参照してください。

data-type
許容されるタイプについては、『CREATE TABLE』の *data-type* を参照してください。宣言済みグローバル一時表では、BLOB、 CLOB、 DBCLOB、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK、参照、および構造タイプを使用できませんのでご注意ください (SQLSTATE 42962)。なお、この例外として、これらの制限されたタイプをソースとする特殊タイプがあります。

FOR BIT DATA は、文字ストリング・データ・タイプの一部として指定することができます。

column-options
表の列に関連した追加オプションを定義します。

NOT NULL
列に NULL 値が入るのを防止します。 NULL 値の指定については、『CREATE TABLE』の NOT NULL を参照してください。

default-clause
デフォルトの指定については、『CREATE TABLE』の *default-clause* を参照してください。

IDENTITY および identity-attributes
ID 列の指定については、 『CREATE TABLE』の IDENTITY および *identity-attributes* を参照してください。

LIKE *table-name1* または *view-name*
表の列の名前と記述が、指定された表 (*table-name1*)、ビュー (*view-name*)、またはニックネーム (*nickname*) の列とまったく同じであることを示します。 LIKE の後に指定する名前は、カタログに存在している表、ビュー、またはニックネー

ム、あるいは宣言済み一時表を識別するものでなければなりません。型付き表または型付きビューを指定することはできません (SQLSTATE 428EC)。

LIKE は、*n* 列の暗黙的な定義で使用します。 *n* は、指定した表またはビューに含まれる列の数を表します。

- 表が特定されると、暗黙的な定義には *table-name1* のそれぞれの列の列名、データ・タイプ、および NULL 可能特性が入ります。 EXCLUDING COLUMN DEFAULTS を指定しないと、列のデフォルト値も入ります。
- ビューが特定されると、暗黙的な定義には *view-name* に指定した全選択のそれぞれの結果列の列名、データ・タイプ、および NULL 可能特性が入ります。

copy-attributes 文節に基づいて、列のデフォルトと ID 列属性を組み込んだり除外したりすることができます。

指定した表やビューのこの他の属性は、暗黙的な定義には含まれません。したがって、新しい表にはユニーク制約、外部キー制約、トリガー、または索引はありません。表は、IN 文節の指定に従って、表スペースの中に明示的または暗黙的に作成されます。

table-name1 や *view-name* には、作成するグローバル一時表と同じ名前を使用することはできません (SQLSTATE 428EC)。

AS (*fullselect*) DEFINITION ONLY
表定義が、照会式の結果による列定義に基づいていることを示します。 AS (*fullselect*) は、宣言済みグローバル一時表に対する *n* 列の暗黙的な定義で使用されます。 *n* は、*fullselect* の結果として得られる列の数を表します。新しい表の列は、これらの *fullselect* で得られた列に基づいて定義されます。選択リストの各エレメントの名前は、それぞれユニークなものでなければなりません (SQLSTATE 42711)。 SELECT 文節で AS 文節を使用すると、それぞれのエレメントにユニークな名前を付けることができます。

暗黙的な定義では、*fullselect* の各結果列について、その列名、データ・タイプ、および NULL 可能特性を定義します。

copy-options
これらのオプションでは、ソースの結果表定義 (表、ビュー、または全選択) から付加的な属性をコピーするかどうかを指定します。

INCLUDING COLUMN DEFAULTS
ソース結果表の定義の更新可能な各列の列デフォルトをコピーします。更新可能でない列では、作成される表の対応列にデフォルトが定義されないことになります。

LIKE *table-name1* が指定されており、かつ *table-name1* が基本表か宣言済み一時表である場合に限り、この INCLUDING COLUMN DEFAULTS がデフォルトとして使用されます。

EXCLUDING COLUMN DEFAULTS
列のデフォルトは、ソースの結果表定義からコピーされません。

この文節がデフォルトです。ただし、LIKE *table-name* が指定され、かつ *table-name* が基本表または宣言済み一時表を示す場合を除きます。

INCLUDING IDENTITY COLUMN ATTRIBUTES
この文節を使用すると、ソースの結果表定義から ID 列の属性 (START WITH、INCREMENT BY、および CACHE の値) がコピーされます。これらの属性をコピーできるのは、表、ビュー、または全選択内の対応する列のエレメントが、 ID のプロパティーが含まれている基本表の列名に直接または間接的にマップされた表の列の名前、またはビューの列の名前である場合です。これ以外の場合は、新しい一時表の列に ID のプロパティーは定義されません。たとえば:

- 全選択の選択リストに ID 列の名前のインスタンスが複数含まれている (つまり、同じ列を複数回選択している) 場合
- 全選択の選択リストに複数の ID 列が含まれている (つまり、結合が関与している) 場合
- ID 列が選択リスト内の式に組み込まれている場合
- 全選択にセット演算 (UNION (合併)、EXCEPT (差)、または INTERSECT (論理積)) が含まれている場合

EXCLUDING IDENTITY COLUMN ATTRIBUTES
ソース結果表の定義から ID 列属性はコピーされません。

ON COMMIT
COMMIT 操作の実行時にグローバル一時表で行うアクションを指定します。

DELETE ROWS
表にオープンされている WITH HOLD カーソルがなければ、すべての行が表から削除されます。これがデフォルトです。

PRESERVE ROWS
表の行が保存されます。

NOT LOGGED
表の作成を含め、表に対して行われた変更は記録されません。 ROLLBACK (または ROLLBACK TO SAVEPOINT) 操作が実行されると、作業単位 (またはセーブポイント) で表を作成していた場合には、その表はドロップされます。そして作業単位 (またはセーブポイント) で表をドロップしていた場合には、表がリストアされますが、行はリストアされません。

ON ROLLBACK
ROLLBACK (または ROLLBACK TO SAVEPOINT) 操作が実行されるときに、記録されていないグローバル一時表に対して取られるアクションを指定します。

DELETE ROWS
表データが変更されている場合は、すべての行が削除されます。これがデフォルトです。

PRESERVE ROWS
表の行が保存されます。

WITH REPLACE
ユーザーが指定した名前を持つ宣言済みグローバル一時表がすでに存在している場合は、既存の表をこのステートメントで定義した一時表で置き換える (および既存の表の行をすべて削除する) よう指示します。

WITH REPLACE が指定されていない場合は、現行セッションにすでに存在している宣言済みグローバル一時表の名前を指定することはできません (SQLSTATE 42710)。

IN *tablespace-name*
グローバル一時表をインスタンス化する表スペースを指定します。ここでは、既存の USER TEMPORARY 表スペースを指定する必要があります (SQLSTATE 42838)。また、ステートメントの許可 ID にはその表スペースに対する USE 特権が含まれていなければなりません (SQLSTATE 42501)。この文節が指定されない場合、表をインスタンス化する表スペースは USER TEMPORARY 表スペースの中から選択され、その中のステートメントの許可 ID に USE 特権が含まれている表スペースで、かつ必要なページ・サイズに最も適したサイズの表スペースが使用されます。複数の表スペースがそれにあてはまる場合、以下のどれに USE 特権が付与されているかに応じて優先順位が決められます。

1. 許可 ID
2. 許可 ID を保有するグループ
3. PUBLIC

それでも複数の表スペースがそれにあてはまる場合は、最終選択はデータベース・マネージャーによって行われます。条件に合う USER TEMPORARY 表スペースがない場合はエラーが戻されます (SQLSTATE 42727)。

表スペースの決定は、以下の時点で変更することができます。

- 表スペースをドロップまたは作成するとき
- USE 特権を付与または取り消すとき

十分な表のページ・サイズは、行のバイト・カウントか列の数のいずれかによって決まります。詳細については、『CREATE TABLE』の『行サイズ』を参照してください。

PARTITIONING KEY (*column-name,...***)**
表のデータがパーティション化されている場合に、パーティション・キーを指定します。各 *column-name* (列名) は、表の列を指定するものでなければなりません。また、同じ列を複数回指定することはできません。

この文節の指定がなく、この表が複数パーティション・データベースのパーティション・グループに存在する場合は、そのパーティション・キーが宣言済み一時表の最初の列として定義されます。

宣言済み一時表では、単一パーティションのデータベース・パーティション・グループに定義された表スペースにおいて、すべての列の集合をパーティション・キーの定義に使用することができます。このパラメーターの指定がない場合、パーティション・キーは作成されません。

USING HASHING
データ分散のパーティション化方式として、ハッシュ関数を使用することを指定します。これは、サポートされる唯一のパーティション化方式です。

注:

- ***互換性***
  - DB2 for OS/390 and z/OS との互換性を保つために、以下のようにすることができます。

- 以下の構文はデフォルトの振る舞いとして受け入れられます。
  - CCSID ASCII
  - CCSID UNICODE

- USER TEMPORARY 表スペースは、ユーザー定義の一時表が宣言される前に、存在しなくてはなりません (SQLSTATE 42727)。
- **宣言済みグローバル一時表の参照:** 宣言済みグローバル一時表の記述は DB2 カタログ (SYSCAT.TABLES) に現れないため、この記述は永続的なものではなく、またデータベース接続によって共有することもできません。従って、同じ *table-name* という宣言済みグローバル一時表を定義するセッションであっても、その宣言済みグローバル一時表の記述はそれぞれのセッションによって異なる可能性があります。

  SQL ステートメント (DECLARE GLOBAL TEMPORARY TABLE ステートメントは除く) を使用して宣言済みグローバル一時表を参照するためには、その表をスキーマ名 SESSION で明示的または暗黙的に修飾する必要があります。*table-name* が SESSION で修飾されていない場合、宣言済みグローバル一時表は参照を決定する際に認識されません。

  グローバル一時表が名前によって宣言されていない接続で SESSION.*table-name* を参照する場合は、カタログ内の持続オブジェクトから参照先が決定されます。そのオブジェクトが存在しない場合はエラーが戻されます (SQLSTATE 42704)。
- バインドしているパッケージに、 SESSION によって暗黙的または明示的に修飾された表を参照する静的 SQL ステートメントが含まれている場合、それらのステートメントは静的にはバインドされません。これらのステートメントは、呼び出されると、パッケージのバインドにおいて VALIDATE オプションが選択されているかどうかにかかわらず増分的にバインドされます。ステートメントの実行時には、宣言済み一時表が存在する場合はその一時表に対して、存在しない場合は永続表に対して各表の参照が行われます。このどちらも存在しない場合はエラーが戻されます (SQLSTATE 42704)。
- **特権:** 宣言済みグローバル一時表を定義する場合、その表を定義するユーザーには、表をドロップする権限も含めて、その表に対するすべての表特権が付与されます。加えて、PUBLIC に対しても同様の特権が GRANT されます。 (GRANT オプションによって GRANT される特権はありません。また、これらの特権はいずれもカタログ表には現れません。) これらの特権を持つユーザーは、すでに定義されている宣言済みグローバル一時表を参照するセッションで、すべての SQL ステートメントを実行することができます。
- **インスタンス化と終了:** 以下の説明において、それぞれ P はセッションを、T はセッション P の中の宣言済みグローバル一時表を示しています。
  - T の空のインスタンスは、 P で実行される DECLARE GLOBAL TEMPORARY TABLE ステートメントの結果として作成されます。
  - P で実行されるすべての SQL ステートメントでは T を参照することができます。そして、P で T を参照する場合は、すべてその同じインスタンスが参照されます。
  - SQL プロシージャーのコンパウンド・ステートメント (BEGIN と END で定義される) で DECLARE GLOBAL TEMPORARY TABLE ステートメントを指定すると、宣言済みグローバル一時表の有効範囲がコンパウンド・ステートメ

ントだけでなく接続にまで広がり、表はコンパウンド・ステートメントの外部からも認識されるようになります。表はコンパウンド・ステートメントの END で暗黙的にドロップされません。宣言済みグローバル一時表は、その表が明示的にドロップされない限り、同じ名前を使用してセッション内の他のコンパウンド・ステートメントで複数回定義することはできません。

- ON COMMIT DELETE ROWS 文節が暗黙的または明示的に指定された場合は、 P においてコミット操作が作業単位で終了し、 T に属する WITH HOLD カーソルが 1 つも P にオープンされていない状態になると、操作 DELETE FROM SESSION.T がコミットに組み込まれます。
- P において、作業単位またはセーブポイントでロールバック操作が終了し、その作業単位またはセーブポイントに SESSION.T への変更が含まれている場合、 NOT LOGGED が指定されていると、このロールバック操作には SESSION.T からの DELETE 操作が含まれます。そうでない場合は、T の変更は取り消されます。

  P において、作業単位またはセーブポイントでロールバック操作が終了し、その作業単位またはセーブポイントに SESSION.T への宣言が含まれている場合、このロールバック操作には DROP SESSION.T 操作が含まれます。

  P において、作業単位またはセーブポイントでロールバック操作が終了し、その作業単位またはセーブポイントに宣言済み一時表 SESSION.T のドロップが含まれている場合、このロールバック操作によって表のドロップは取り消されます。 NOT LOGGED が指定されている場合は、表も空にされます。
- アプリケーションによって、宣言された T が終了されたり、データベースとの接続が切断された場合、 T はドロップされ、そのインスタンス化された行は破棄されます。
- T の宣言が行われたサーバーへの接続が終了すると、 T はドロップされ、そのインスタンス化された行は破棄されます。

- **宣言済みグローバル一時表の使用に関する制限:** 宣言済みグローバル一時表には、以下のような使用上の制限があります。
  - ALTER、COMMENT、GRANT、LOCK、RENAME、または REVOKE ステートメントでこの一時表を指定することはできません (SQLSTATE 42995)。
  - CREATE ALIAS、CREATE FUNCTION (SQL スカラー、表、または行)、CREATE TRIGGER、または CREATE VIEW ステートメントでこの一時表を参照することはできません (SQLSTATE 42995)。
  - 参照制約でこの一時表を指定することはできません (SQLSTATE 42995)。

関連資料:

- 347 ページの『CREATE TABLE』

関連サンプル:

- 『tbtemp.sqc -- How to use a declared temporary table (C)』
- 『TbTemp.java -- How to use Declared Temporary Table (JDBC)』

DELETE

DELETE ステートメントは、表、ニックネーム、またはビュー、あるいは指定した全選択の基礎表、ニックネーム、またはビューから、行を削除します。 ニックネームから行を削除すると、そのニックネームの参照先のデータ・ソース・オブジェクトから行を削除することになります。ビューから行を削除すると、このビューに対する削除操作に INSTEAD OF トリガーが定義されていない場合は、そのビューの基本となる表から行を削除することになります。 INSTEAD OF トリガーが定義されている場合は、トリガーが代わりに実行されます。

このステートメントには、以下の 2 つの形式があります。

- *検索 (Searched)* DELETE 形式は、 1 つまたは複数の行を削除するのに使用します (削除する行は検索条件によって、自由に限定できます)。
- *位置指定 (Positioned)* DELETE 形式は、 1 行だけを削除する場合に使用します (削除される行は、カーソルの現在位置によって決まります)。

呼び出し:

DELETE ステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントのどの形式を実行する場合も、ステートメントの許可 ID に、以下の特権が少なくとも 1 つ含まれている必要があります。

- 削除する行を含む表、ビュー、またはニックネームに対する DELETE 特権
- 削除する行を含む表、ビュー、またはニックネームに対する CONTROL 特権
- SYSADM または DBADM 権限

検索 DELETE ステートメントを実行する場合、副照会で参照される表、ビュー、またはニックネームのそれぞれに対して、ステートメントの許可 ID に、以下の特権が少なくとも 1 つ含まれている必要があります。

- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

ステートメントを処理するために使用されるパッケージが SQL92 規則を使用してプリコンパイルされる場合 (SQL92E または MIA の値を指定したオプション LANGLEVEL) で、検索 DELETE ステートメント形式の *search-condition* に表またはビューの列への参照が含まれている場合には、このステートメントの許可 ID の特権には以下の特権のうち少なくとも 1 つが含まれている必要があります。

- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

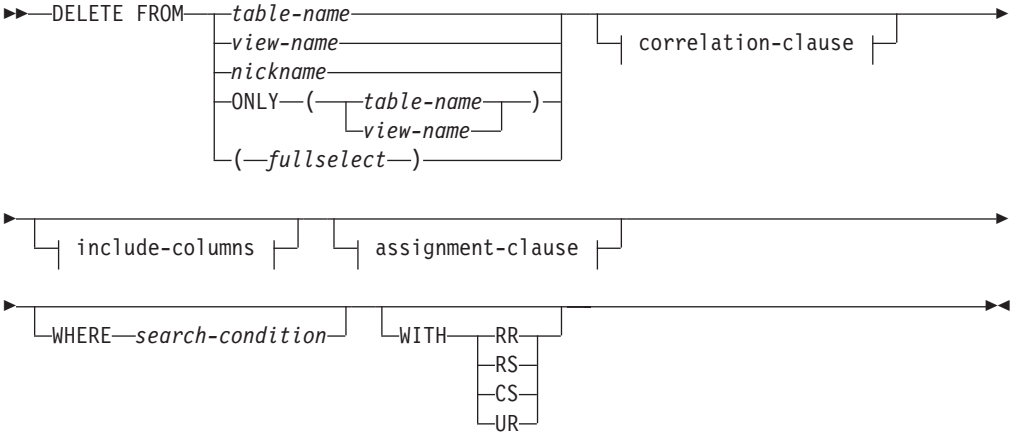
指定した表またはビューが ONLY キーワードの後にくる場合、ステートメントの許可 ID が持つ特権にも、指定した表またはビューの副表またはサブビューごとに SELECT 特権が含まれている必要があります。

静的 DELETE ステートメントの場合、グループ特権は検査されません。

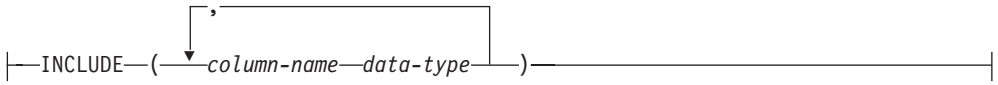
削除操作の対象がニックネームの場合は、データ・ソースのオブジェクトに対する特権は、ステートメントがデータ・ソースで実行されるまで考慮されません。この時点で、データ・ソースに接続するために使用される許可 ID は、データ・ソースのオブジェクトに対して操作を行うのに必要な特権を持っている必要があります。ステートメントの許可 ID は、データ・ソースの別の許可 ID へマップできます。

構文:

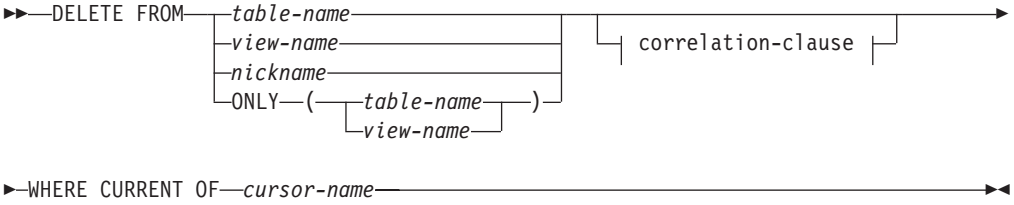
検索削除:



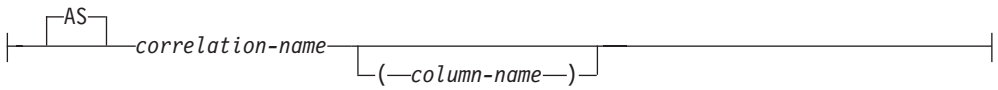
include-columns:



位置指定削除:



correlation-clause:



説明:

FROM *table-name*、*view-name*、 *nickname*、**または** *(fullselect)*

削除操作の対象のオブジェクトを指定します。 この名前は、カタログに存在する表またはビューを指定するものでなければならず、カタログ表、カタログ・ビュー、システムによって保守されるマテリアライズ照会表、または読み取り専用のビューは指定できません。

table-name が型付きの表である場合は、このステートメントを使用して、その表またはそれに関係する副表の行を削除できます。

view-name が型付きビューである場合は、このステートメントを使用して、その基礎表の行またはそのビューに関係するサブビューの基礎表を削除できます。 *view-name* が基礎表 (型付き表) を伴う通常のビューである場合、このステートメントを使用して、その型付き表またはそれに関係する副表の行を削除できます。

削除操作のオブジェクトが全選択である場合、全選択は、CREATE VIEW ステートメントの説明の『注』にある、『削除可能ビュー』の項目で定義されているように、削除可能になっている必要があります。

WHERE 文節内で参照できるのは、指定された表の列だけです。 位置指定 DELETE の場合は、 FROM 文節に指定されているのと同じ表またはビューを、関連するカーソルにも ONLY を使用せずに指定しなければなりません。

FROM ONLY (*table-name***)**
型付き表の場合に適用できます。 ONLY キーワードは、指定された表のデータだけにステートメントを適用し、その表に関係する副表の行は削除されないことを指定します。 位置指定 DELETE の場合は、FROM 文節に指定されているのと同じ表を、関連するカーソルにも ONLY を使用して指定しなければなりません。 *table-name* が型付き表でない場合は、このステートメントに ONLY キーワードを使用しても効果はありません。

FROM ONLY (*view-name***)**
このステートメントは型付きビューのみに適用されます。 ONLY キーワードは、指定されたビューのデータだけにこのステートメントが適用されることを指定します。サブビューの行は、このステートメントでは削除されません。 位置指定 DELETE の場合は、FROM 文節に指定されているのと同じビューを、関連するカーソルにも ONLY を使用して指定しなければなりません。 *view-name* が型付きビューでない場合は、このステートメントに ONLY キーワードを使用しても効果はありません。

correlation-clause
search-condition で、表、ビュー、ニックネームまたは全選択を指定するのに使用できます。 *correlation-clause* についての説明は、『副選択』の説明にある『table-reference』を参照してください。

include-columns
全選択の FROM 文節にネストされているとき、 *table-name* や *view-name* などの列と一緒に DELETE ステートメントの中間結果表に組み込まれている列セットを指定します。 *include-columns* は、 *table-name* や *view-name* で指定されている列のリストの最後に付加されます。

INCLUDE
DELETE ステートメントの中間結果表に組み込まれる列のリストを指定します。

column-name
DELETE ステートメントの中間結果表の列を指定します。名前は、他の組み込み列や、 *table-name* または *view-name* の列と同じ名前であってはなりません (SQLSTATE 42711)。

data-type
組み込み列のデータ・タイプを指定します。データ・タイプは、CREATE TABLE ステートメントでサポートされているものでなければなりません。

assignment-clause
UPDATE ステートメントの *assignment-clause* の説明を参照してください。同じ規則が適用されます。 *include-columns* は、 *assignment-clause* を使用して設定できる唯一の列です (SQLSTATE 42703)。

WHERE
削除する行を選択する条件を指定します。 この文節は、省略するか、検索条件を指定するか、あるいはカーソルの名前を指定できます。 この文節を省略すると、表またはビューのすべての行が削除されます。

search-condition
副照会以外の検索条件の各 *column-name (列名)* は、表またはビューの列を指定していなければなりません。

search-condition は、該当の表、ビュー、またはニックネームの各行に適用されます。 *search-condition* の結果が「真」の行だけが削除されます。

検索条件に副照会が含まれる場合、その副照会は、*search-condition (検索条件)* が行に適用されるたびに実行され、その結果が *search-condition (検索条件)* の適用の対象として使用されます。 実際には、相関参照が含まれていない副照会は一度実行されるのに対し、相関参照の含まれている副照会は各行ごとに一度ずつ実行しなければならない場合があります。 副照会で DELETE ステートメントの対象の表、または削除規則が CASCADE あるいは SET NULL の従属表が参照されている場合、その副照会は行が削除される前に完全に評価されます。

CURRENT OF *cursor-name*
プログラムの DECLARE CURSOR ステートメントで定義されたカーソルを指定します。 DECLARE CURSOR ステートメントは、DELETE ステートメントよりも前になければなりません。

指定する表、ビュー、またはニックネームは、そのカーソルの SELECT ステートメントの FROM 文節でも指定されていなければならず、またそのカーソルの結果表が読み取り専用であってはなりません。 (読み取り専用の結果表については、『DECLARE CURSOR』 を参照してください。)

DELETE ステートメントが実行される場合、カーソルは行の位置になければなりません。その行が削除されます。 削除後、カーソル位置はその結果表の次の行の前になります。 次の行がない場合、カーソル位置は最後の行の後になります。

WITH
削除する行を検索しているときに使用される分離レベルを指定します。

RR
反復可能読み取り

RS
読み取り固定

CS
カーソル固定

UR
非コミット読み取り

ステートメントのデフォルト分離レベルは、ステートメントがバインドされているパッケージの分離レベルです。

規則:

- ***トリガー:*** DELETE ステートメントによってトリガーの実行が引き起こされる場合があります。トリガーが他のステートメントの実行を引き起こす場合や、削除された行に起因するエラーが発生する場合があります。ビューの DELETE ステートメントが INSTEAD OF トリガーの発生を引き起こす場合は、参照保全は、トリガーの発生を引き起こしたビューの基礎表に対してではなく、トリガー内で実行される更新に対して検査されます。
- ***参照保全:*** 指定する表または指定するビューの基本表が親である場合、削除のために選択する行は RESTRICT の削除規則との関係において従属であってはならず、 DELETE は RESTRICT の削除規則との関係において従属である下層行にカスケードしてはなりません。

  削除操作が RESTRICT の削除規則によって禁止されていなければ、選択された行は削除されます。選択された行の従属行もすべて影響を受けます。
  - 削除規則が SET NULL の関係において、すべての従属行の外部キーの NULL 可能列は、 NULL 値に設定されます。
  - 削除規則が CASCADE の関係において、すべての従属行も削除され、上記の規則はこれらの行にも適用されます。

  他の参照制約が実施された後で、非 NULL の外部キーが既存の親行を指すようにするために、 NO ACTION の削除規則が検査されます。

注:

- 複数行の DELETE の実行中にエラーが起こった場合、データベースは変更されません。
- 適切なロックがすでに存在するのでない限り、正常な DELETE ステートメントの実行中には、1 つまたは複数の排他ロックが獲得されます。 COMMIT ステートメントまたは ROLLBACK ステートメントを発行すると、それらのロックは解放されます。 ロックがコミットまたはロールバック操作によって解放される時まで、削除操作の効果は次のものにしか認識されません。
  - 削除を実行したアプリケーション・プロセス
  - 分離レベル UR を使用する別のアプリケーション・プロセス

  ロックにより、他のアプリケーション・プロセスが、表に対して操作を実行するのを防ぐことができます。
- アプリケーション・プロセスがそのカーソルのいずれかがある行を削除すると、それらのカーソルの位置はその結果表の中の次の行の前になります。 C をカーソルとし、それが (OPEN、C による DELETE、その他の何らかのカーソルによる DELETE、または検索 DELETE の結果として) 行 R の前の位置にあるとします。 R の派生元の基本表に影響する INSERT、UPDATE、および DELETE 操作があると、 C を参照する次の FETCH 操作では、必ずしも C の位置が R にある必要はありません。たとえば、この操作によって C が R' の位置になることがあります (R' は操作結果の表で次の行となった新しい行)。

- 削除操作の対象となる行数が SQLCA の SQLERRD(3) に示されます。 SQL プロシージャー・ステートメントでは、値は GET DIAGNOSTICS ステートメントの ROW_COUNT 変数を使用して検索できます。参照制約による影響およびトリガーにより実行されたステートメントによる影響を受けた行の数は、 SQLCA の SQLERRD(5) に示されます。これには、CASCADE 削除規則の結果として削除された行と、 SET NULL 削除規則の結果として外部キーが NULL に設定された行とが含まれます。トリガーにより実行されたステートメントについては、挿入、更新、または削除された行数が含まれます。
- あるエラーが起きたために検索条件に合う行の削除がすべて完了しなかった場合、および既存の参照制約に必要なすべての操作が行われなかった場合、表は変更されずエラーが戻されます。
- ニックネームの場合は、外部サーバー・オプション `iud_app_svpt_enforce` によってさらに制限が加えられます。詳細については、フェデレーテッド・システムの資料を参照してください。
- 一部のデータ・ソースの場合、データが矛盾している可能性があるために、ニックネームに対する削除を行うと SQLCODE -20190 が戻される場合があります。詳細については、フェデレーテッド・システムの資料を参照してください。
- 削除された行の中に DATALINK 列によってリンクされたファイルが含まれている場合には、それらのファイルはリンク解除されてから、データ・リンク列の定義に応じてリストアされたり削除されたりします。

  値の中で参照されているファイル・サーバーがデータベース・サーバーに登録されていない場合、 DATALINK 値を削除しようとするとエラーが起きるかもしれません (SQLSTATE 55022)。

  削除時に使用できないサーバーにリンクされている行を削除すると、エラーが起こる可能性があります (SQLSTATE 57050)。

例:

例 1: DEPARTMENT 表から部門 (DEPTNO) 'D11' を削除します。

```
DELETE FROM DEPARTMENT
  WHERE DEPTNO = 'D11'
```

例 2: DEPARTMENT 表からすべての部門を削除します (つまり、表を空にします)。

```
DELETE FROM DEPARTMENT
```

例 3: EMPLOYEE 表から 1995 年中に売上が 1 つもなかったセールス担当員または現場担当員を削除します。

```
DELETE FROM EMPLOYEE
  WHERE LASTNAME NOT IN
    (SELECT SALES_PERSON
      FROM SALES
      WHERE YEAR(SALES_DATE)=1995)
      AND JOB IN ('SALESREP','FIELDREP')
```

例 4: EMPLOYEE 表から、重複している従業員の行をすべて削除します。従業員の行は、ラストネームが一致していれば、重複していると考えられます。従業員の行のファーストネームは、字句順に、できるだけ短くしておきます。

```
DELETE FROM
  (SELECT ROWNUMBER() OVER (PARTITON BY LASTNAME ORDER BY>/ph> FIRSTNME)
    FROM EMPLOYEE) AS E(RN)
    WHERE RN = 1
```

関連資料:

- *SQL リファレンス 第 1 巻* の『検索条件』
- *SQL リファレンス 第 1 巻* の『副選択』
- 474 ページの『CREATE VIEW』
- 491 ページの『DECLARE CURSOR』
- 779 ページの『UPDATE』
- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbmod.c -- How to modify table data』
- 『dbuse.sqc -- How to use a database (C)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『tbmod.sqc -- How to modify table data (C)』
- 『dbuse.sqC -- How to use a database (C++)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『tbmod.sqC -- How to modify table data (C++)』
- 『delet.sqb -- How to delete table data (MF COBOL)』
- 『updat.sqb -- How to update, delete and insert table data (MF COBOL)』
- 『DbUse.java -- How to use a database (JDBC)』
- 『TbConstr.java -- How to create, use and drop constraints (JDBC)』
- 『TbMod.java -- How to modify table data (JDBC)』
- 『DbUse.sqlj -- How to use a database (SQLj)』
- 『TbConstr.sqlj -- How to create, use and drop constraints (SQLj)』
- 『TbMod.sqlj -- How to modify table data (SQLj)』

DESCRIBE

DESCRIBE ステートメントは、準備されたステートメントについての情報を入手します。

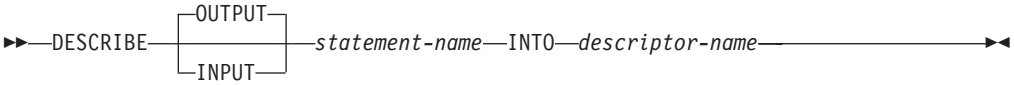
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。

構文:



説明:

OUTPUT
記述ユーティリティーが、準備済みステートメント内の選択リスト列についての情報、またはストアード・プロシージャー呼び出し用の OUT および INOUT パラメーターに関連した出力パラメーター・マーカーについての情報を入手することを指示するオプショナル・キーワードです。

INPUT
記述ユーティリティーが、準備済みステートメント内の入力パラメーター・マーカーについての情報を入手することを指定します。 CALL ステートメントの場合は、この情報には、ストアード・プロシージャー用の IN および INOUT パラメーターに関連したパラメーター・マーカーも含まれます。入力パラメーター・マーカーは、使用法に関係なく常に NULL 可能と見なされます。

statement-name
その情報を入手する対象のステートメントを指定します。 DESCRIBE ステートメントを実行する時点で、この名前はすでに準備済みのステートメントを指定していなければなりません。

INTO *descriptor-name*
SQL 記述子域 (SQLDA) を指定します。 DESCRIBE ステートメントを実行する前に、SQLDA 内の以下の変数を設定しておく必要があります。

SQLN SQLVAR によって表示される変数の数を指定します。 (SQLVAR 配列の大きさは SQLN によって指定されます。) DESCRIBE ステートメントを実行する前に、SQLN にゼロ以上の値を設定する必要があります。

DESCRIBE ステートメントを実行すると、データベース・マネージャーは、以下のように SQLDA の変数に値を割り当てます。

SQLDAID
最初の 6 バイトは 'SQLDA ' に設定されます (5 文字の英字の後、6 文字目はスペース文字です)。

第 7 バイト (SQLDOUBLED) は、 SQLDA において各選択リスト項目 (結果表の列) につき SQLVAR 項目が 2 つずつ含まれている場合に、 '2' に設定されます。結果列として LOB、特殊タイプ、構造タイプ、または参照タイプを可能にするために、このようになっています。それ以外の場合、SQLDOUBLED はスペース文字に設定されます。

SQLDA の中に DESCRIBE の応答全体が入るだけのスペースがない場合、ダブル・フラグはスペースに設定されます。

8 番目のバイトは、スペース文字に設定されます。

SQLDABC
SQLDA の長さ。

SQLD もし準備済みステートメントが SELECT である場合は、結果表の中の列数。それ以外の場合は 0。

SQLVAR
SQLD の値が 0 の場合、または SQLD の値が SQLN の値より大きい場合は、 SQLVAR のエレメントには値は割り当てられません。

SQLD の値が *n* (*n* は 0 より大きく、 SQLN の値以下) の場合、SQLVAR の最初のエレメントには結果表の最初の列に関する記述が入り、 SQLVAR の 2 番目のエレメントには結果表の 2 番目の列に関する記述が入るというように、 SQLVAR の最初の *n* 個のエレメントに値が割り当てられます。1 つの列の記述は、 SQLTYPE、 SQLLEN、 SQLNAME、SQLLONGLEN、および SQLDATATYPE_NAME に割り当てられている値で構成されます。

基本 SQLVAR

SQLTYPE
列のデータ・タイプと、その列に NULL 値が入るかどうかを示すコード。

SQLLEN
結果列のデータ・タイプによって決まる長さを示す値。 LOB データ・タイプの場合、SQLLEN は 0 になります。

SQLNAME
sqlname は、以下のように導出されます。

- SQLVAR が、SELECT ステートメントの選択リスト内の単純な列参照用の派生列に対応する場合は、 sqlname はその列の名前です。
- SQLVAR が、ストアード・プロシージャーのパラメーター・リスト内の式の一部ではないパラメーター・マーカーに対応する場合は、 CREATE PROCEDURE でパラメーターが指定されていれば、sqlname にはそのパラメーターの名前が含まれます。
- CREATE PROCEDURE でパラメーターが指定されていなければ、 sqlname には SQLDA 内の SQLVAR の位置を表す ASCII 数値リテラル値が含まれます。

副次 SQLVAR

これらの変数は、LOB、特殊タイプ、構造タイプ、または参照タイプの列を含めることができるよう、 SQLVAR の項目の数が 2 倍にされた場合にのみ使用されます。

SQLLONGLEN
BLOB、CLOB、または DBCLOB の列の長さ属性。

SQLDATATYPE_NAME
データベース・マネージャーは、すべてのユーザー定義タイプ (特殊タイプまたは構造タイプ) の列で、この名前を完全修飾ユーザー定義タイプ名に設定します。また、参照タイプの列では、この名前を参照のターゲット・タイプの完全修飾ユーザー定義タイプ名に設定します。それ以外の場合、スキーマ名は SYSIBM となり、タイプ名は SYSCAT.DATATYPES カタログ・ビューの TYPENAME 列に含まれている名前になります。

注:

- DESCRIBE ステートメントを実行する前に、SQLN の値を SQLDA の中の SQLVAR のオカレンスの数に設定し、 SQLN 個のオカレンスが入るだけの十分なストレージを割り振っておく必要があります。たとえば、準備済み SELECT ステートメントの結果表の列の記述を入手するには、 SQLVAR のオカレンス数は列数以上でなければなりません。
- 大きいサイズの LOB が予想される場合、このラージ・オブジェクトの処理がアプリケーション・メモリーに与える影響について考慮してください。そのような状況では、ロケーターまたはファイル参照変数を使用することを考えてください。 DESCRIBE ステートメントを実行してからストレージを割り振るまでの間に、 SQLDA を修正して、SQLLEN などの他のフィールドに対する対応する変更を使用して、SQL_TYP_xLOB の SQLTYPE を SQL_TYP_xLOB_LOCATOR または SQL_TYP_xLOB_FILE に変更してください。その後、SQLTYPE に基づいてストレージを割り振ってから、処理を継続します。
- 拡張 UNIX コード (EUC) コード・ページと DBCS コード・ページとの間でコード・ページ変換を行うと、文字長が長くなったり短くなったりする場合があります。
- 構造タイプが選択されているのに、 FROM SQL トランスフォームが定義されていない (TRANSFORM GROUP が CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターで指定されたため (SQLSTATE 428EM)、あるいは指定されたグループに FROM SQL 変換機能が定義されていないため (SQLSTATE 42744)) 場合は、 DESCRIBE からエラーが戻されます。
- **SQLDA の割り振り:** SQLDA を割り振るための可能な方法としては、以下の 3 通りがあります。

  方式 1: アプリケーションで処理する必要のある選択リストが入るだけの十分な数の SQLVAR のオカレンスを含む SQLDA を割り振ります。表に LOB、特殊タイプ、構造タイプ、または参照タイプの列が含まれている場合には、 SQLVAR の数を最大列数の 2 倍にしてください。それ以外の場合は、その数を最大列数と同じにします。割り振りを行ったなら、アプリケーションでこの SQLDA を繰り返し使用できるようになります。

このテクニックでは、大量のストレージを使用し、そのストレージのほとんどが特定の選択リストで使用されるわけではない場合でも決して割り振り解除されることがありません。

方式 2: 選択リストを処理するたびに、以下の 2 つのステップを繰り返し実行します。

1. SQLVAR のオカレンスのない SQLDA (SQLN を 0 にした SQLDA) を指定した DESCRIBE ステートメントを実行します。 SQLD の戻り値は、結果表の列数となります。これは、必要な SQLVAR のオカレンス数か、またはその数の半分のいずれかになります。 SQLVAR 項目がないので、SQLSTATE 01005 の警告が出されます。その警告の SQLCODE が +237、+238、または +239 のいずれかである場合、 SQLVAR 項目の数は SQLD の戻り値の 2 倍でなければなりません。 (上記の正の SQLCODE の戻り値は、 SQLWARN BIND オプションが YES (正の SQLCODE を戻す) に設定されていることが前提となっています。 SQLWARN が NO に設定されている場合でも +238 が戻されて、 SQLVAR 項目の数が SQLD の戻り値の 2 倍でなければならないことを示します。)
2. SQLVAR のオカレンス数として十分大きい数を指定した SQLDA を割り振ります。 この新しい SQLDA を使用して、DESCRIBE ステートメントをもう一度実行します。

この方式では、方式 1 よりもストレージを効率的に管理できます。しかし、DESCRIBE ステートメントの数は 2 倍になります。

方式 3: 選択リストのほとんど (そしておそらくは全部) が入るほどのある程度の大きさではあるが、適度に小さい SQLDA を割り振ります。 DESCRIBE を実行して SQLD 値を調べます。 必要なら、SQLVAR のオカレンス数として SQLD 値を使用することにより、もっと大きな SQLDA を割り振ります。

この方式は、最初の 2 つの方式の折衷方式です。 その効果は、元の SQLDA サイズを適切に選択することに依存しています。

例:

C プログラムの中で、SQLVAR オカレンスのない SQLDA を指定して DESCRIBE ステートメントを実行します。 SQLD が 0 より大きい場合、その値を使って必要な数の SQLVAR のオカレンスを含む SQLDA を割り振り、その SQLDA を使って DESCRIBE ステートメントを実行します。

```
EXEC SQL  BEGIN DECLARE SECTION;
  char    stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

... /* code to prompt user for a query, then to generate */
    /* a select-statement in the stmt1_str            */
EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;

... /* code to set SQLN to zero and to allocate the SQLDA */
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;

... /* code to check that SQLD is greater than zero, to set */
    /* SQLN to SQLD, then to re-allocate the SQLDA          */
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;
```

```
... /* code to prepare for the use of the SQLDA            */
    /* and allocate buffers to receive the data            */
EXEC SQL  OPEN DYN_CURSOR;

... /* loop to fetch rows from result table                */
EXEC SQL  FETCH DYN_CURSOR USING DESCRIPTOR :sqlda;
.
.
.
```

関連資料:

- 647 ページの『PREPARE』
- *SQL リファレンス 第 1 巻* の『SQLDA (SQL 記述子域)』

関連サンプル:

- 『tbread.sqc -- How to read tables (C)』
- 『tbread.sqC -- How to read tables (C++)』

DISCONNECT

DISCONNECT ステートメントは、アクティブな作業単位がない場合に (つまり、コミットまたはロールバックの操作の後)、1 つまたは複数の接続を破棄します。DISCONNECT ステートメントの対象が単一の接続の場合、その接続は、アクティブな作業単位があるかどうかにかかわりなく、そのデータベースが既存の作業単位に関係していない場合にのみ破棄されます。たとえば、他のいくつかのデータベースの作業が終了し、ステートメントの対象については終了していない場合、接続を破棄せずに切断される場合があります。

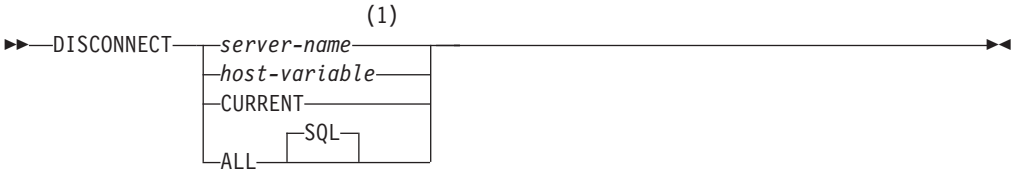
呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。

構文:



注:

1 CURRENT または ALL という名前のアプリケーション・サーバーは、ホスト変数によってのみ指定することができます。

説明:

server-name または *host-variable*

server-name (サーバー名) またはその *server-name* を含む *host-variable* (ホスト変数) によって、アプリケーション・サーバーを指定します。

host-variable (ホスト変数) を指定する場合、それは、長さ属性が 8 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。その *host-variable* に入る *server-name* は、左寄せする必要があり、引用符で区切ることはできません。

server-name は、アプリケーション・サーバーを指定するデータベース別名である点に注意してください。 この名前は、アプリケーション・リクエスターのローカル・ディレクトリーにリストされている必要があります。

指定されたデータベース別名、またはホスト変数に含まれているデータベース別名は、そのアプリケーション・プロセスの既存の接続を指定するものでなければなりません。 データベース別名が既存の接続を指定していない場合、エラー (SQLSTATE 08003) になります。

CURRENT
アプリケーション・プロセスの現行接続を指定します。 アプリケーション・プロセスは、接続された状態でなければなりません。 接続されていない場合、エラー (SQLSTATE 08003) になります。

ALL
アプリケーション・プロセスの既存の接続を全部破棄することを指定します。ステートメント実行時に接続が存在していない場合でも、エラーまたは警告のメッセージは出されません。任意に選択できるキーワードである SQL は RELEASE ステートメントの構文との一貫性を持たせるために含められています。

規則:

- 一般に、DISCONNECT ステートメントは作業単位の中では実行できません。実行すると、エラー (SQLSTATE 25000) になります。この規則の例外は、単一の接続を切断することを指定し、データベースが既存の作業単位に加わっていない場合です。この場合、DISCONNECT ステートメントが発行される時にアクティブな作業単位があるかどうかは問題になりません。
- DISCONNECT ステートメントは、トランザクション処理 (TP) モニター環境の中では全く実行できません (SQLSTATE 25000)。これは、SYNCPOINT プリコンパイラー・オプションが TWOPHASE に設定されている場合に使用されるものです。

注:

- DISCONNECT ステートメントが正常に処理されると、指定されたそれぞれの接続は破棄されます。

  DISCONNECT ステートメントが正常に処理されない場合、アプリケーション・プロセスの接続状態とその接続の状態は変更されません。
- DISCONNECT を使って現行接続を破棄する場合、その次に実行する SQL ステートメントは、CONNECT または SET CONNECTION でなければなりません。
- タイプ 1 CONNECT では、DISCONNECT を使用できないわけではありません。ただし、DISCONNECT CURRENT と DISCONNECT ALL は、使用することはできますが、CONNECT RESET ステートメントの場合と違って、コミット操作は行われません。

  タイプ 1 CONNECT では一度に 1 つの接続しかサポートされないため、DISCONNECT ステートメントに *server-name* または *host-variable* を指定する場合、それは現行接続を指定するものでなければなりません。一般に、“規則” に示されている例外を除き、DISCONNECT は作業単位内で実行すると失敗します。
- リモート接続を作成し保守するには、さまざまなリソースが必要になります。したがって、再使用の予定がないリモート接続は、できるだけ破棄する必要があります。
- 接続は、接続オプションの効果のためにコミット操作中に破棄されることもあります。そのような接続オプションには、AUTOMATIC、 CONDITIONAL、または EXPLICIT があります。それらは、プリコンパイラー・オプションとして設定されたり、ランタイムに SET CLIENT API によって設定されます。
  DISCONNECT オプションの指定については、『分散リレーショナル・データベース』を参照してください。

例:

例 1: IBMSTHDB への SQL 接続は、アプリケーションではもはや必要でなくなりました。コミットかロールバックの操作を行った後、次のステートメントを実行してその接続を破棄します。

```
EXEC SQL DISCONNECT IBMSTHDB;
```

例 2: 現行の接続は、アプリケーションでもはや必要でなくなりました。コミットかロールバックの操作を行った後、次のステートメントを実行してその接続を破棄します。

```
EXEC SQL DISCONNECT CURRENT;
```

例 3: 既存の接続は、アプリケーションでもはや必要でなくなりました。コミットかロールバックの操作を行った後、次のステートメントを実行して接続をすべて破棄します。

```
EXEC SQL DISCONNECT ALL;
```

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』

関連サンプル:

- 『dbconn.sqc -- How to connect to and disconnect from a database (C)』
- 『dbmcon.sqc -- How to use multiple databases (C)』
- 『dbconn.sqC -- How to connect to and disconnect from a database (C++)』
- 『dbmcon.sqC -- How to use multiple databases (C++)』
- 『Util.java -- Utilities for JDBC sample programs (JDBC)』
- 『Util.sqlj -- Utilities for SQLJ sample programs (SQLj)』

DROP

DROP ステートメントは、オブジェクトを削除します。そのオブジェクトに直接または間接的に従属するオブジェクトがある場合、それらも削除されるか、または作動不能になります。オブジェクトを削除すると、その記述がカタログから削除され、そのオブジェクトを参照するパッケージがあれば無効になります。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

2 つの部分からなる名前が使用可能なオブジェクトをドロップする場合、 DROP ステートメントの許可 ID が持つ特権には、以下のいずれかが含まれている必要があります。これらが 1 つも含まれていない場合はエラーが戻されます (SQLSTATE 42501)。

- SYSADM または DBADM 権限
- そのオブジェクトのスキーマに対する DROPIN 特権
- そのオブジェクトのカタログ・ビューの DEFINER 列に記録されているそのオブジェクトの定義者
- オブジェクトに対する CONTROL 特権 (索引、索引指定、ニックネーム、パッケージ、表、およびビューにのみあてはまる)
- カタログ・ビュー SYSCAT.DATATYPES の DEFINER 列に記録されているユーザー定義タイプの定義者 (ユーザー定義タイプに関連したメソッドをドロップしている場合にのみあてはまる)

表またはビューの階層をドロップする場合、DROP ステートメントの許可 ID は、その階層内にあるそれぞれの表またはビューについて、上記の特権のいずれかを持っている必要があります。

スキーマをドロップする場合、DROP ステートメントの許可 ID は SYSADM 権限または DBADM 権限を持っているか、または SYSCAT.SCHEMATA の OWNER 列に記録されているスキーマ所有者でなければなりません。

バッファー・プール、データベース・パーティション・グループ、または表スペースをドロップする場合、 DROP ステートメントの許可 ID は SYSADM 権限または SYSCTRL 権限を持っている必要があります。

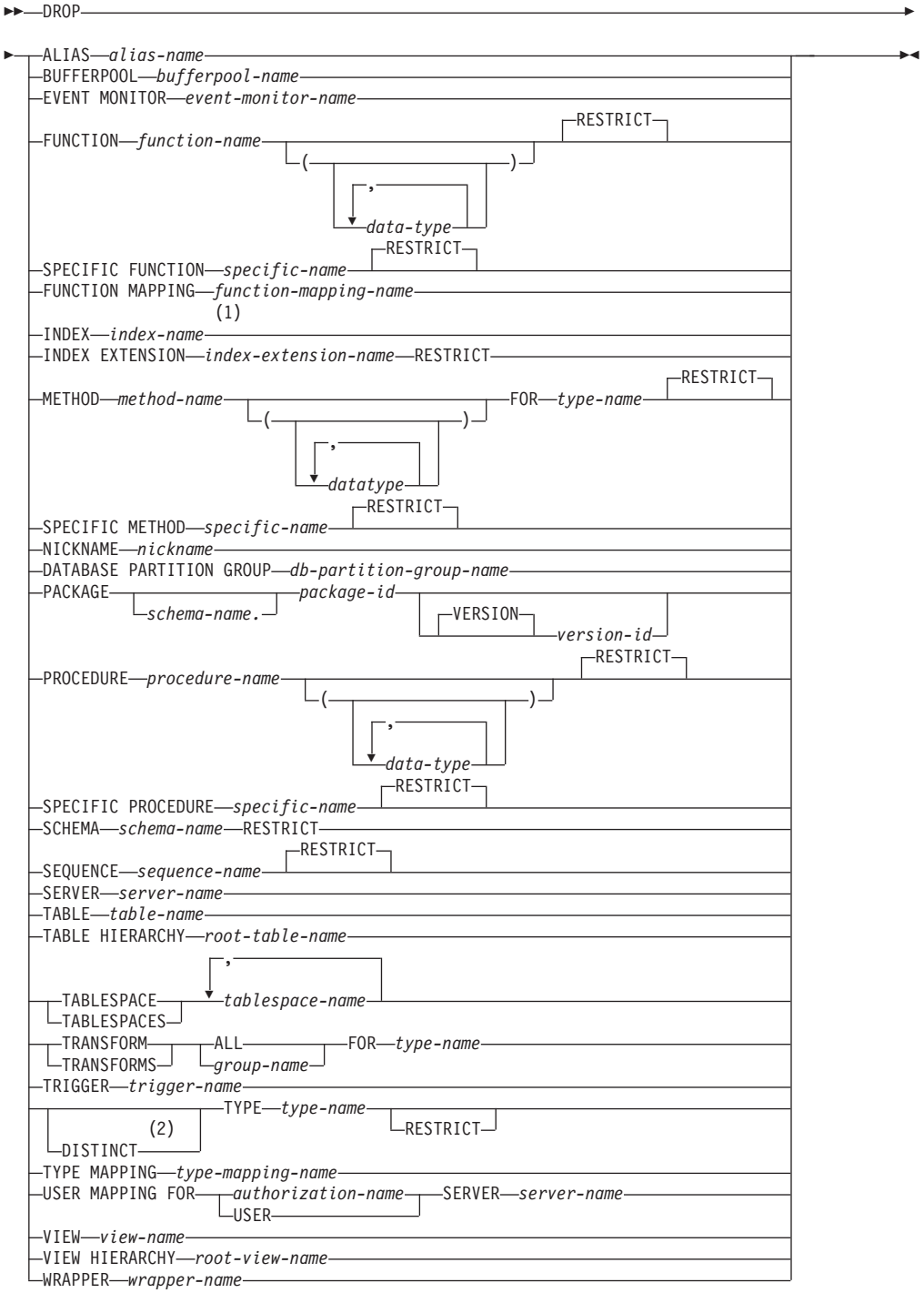
イベント・モニター、サーバー定義、データ・タイプ・マッピング、関数マッピング、またはラッパーをドロップする場合、 DROP ステートメントの許可 ID は SYSADM 権限または DBADM 権限を持っている必要があります。

ユーザー・マッピングをドロップする際、この許可 ID がマッピング内にあるフェデレーテッド・データベースの許可名と異なる場合には、 DROP ステートメントの

許可 ID に SYSADM または DBADM 権限が必要です。 許可 ID と許可名が一致する場合には、必要とされる特権または権限はありません。

トランスフォームをドロップする際、DROP ステートメントの 許可 ID は SYSADM 権限か DBADM 権限を保持しているか、または *type-name* の DEFINER でなければなりません。

構文:



注:

1 *Index-name* には、索引、あるいは索引指定のどちらかの名前を指定できます。

2 任意のユーザー定義タイプをドロップするのに DATA を使用することもできます。

説明:

ALIAS *alias-name*
ドロップする別名を指定します。 *alias-name* (別名) は、カタログに記述されている別名を指定する名前でなければなりません (SQLSTATE 42704)。指定した別名は削除されます。

別名を参照するすべての表、ビュー、およびトリガーは作動不能になります。(これには、CREATE TRIGGER ステートメントの ON 文節で参照されている表と、トリガー SQL ステートメントで参照されているすべての表が含まれます。)

BUFFERPOOL *bufferpool-name*
ドロップするバッファー・プールを指定します。 *bufferpool-name* (バッファー・プール名) は、カタログに記述されているバッファー・プールを指定していなければなりません (SQLSTATE 42704)。そのバッファー・プールに表スペースが割り当てられていない場合もあります (SQLSTATE 42893)。 IBMDEFAULTBP バッファー・プールはドロップできません (SQLSTATE 42832)。 DB2 で使用するバッファー・プール・メモリーは、すぐに解放されます。ディスク装置は、次にデータベースへ接続するときまで解放できません。

EVENT MONITOR *event-monitor-name*
ドロップするイベント・モニターを指定します。 *event-monitor-name* (イベント・モニター名) は、すでにカタログに存在するイベント・モニターを指定していなければなりません (SQLSTATE 42704)。

指定したイベント・モニターが ON の場合は、エラー (SQLSTATE 55034) が戻されます。それ以外の場合は、イベント・モニターは削除されます。

イベント・モニターをドロップする時点でそのイベント・モニターのターゲット・パスにイベント・ファイルが存在する場合、そのイベント・ファイルは削除されません。ただし、それと同じターゲット・パスを指定した新しいイベント・モニターが作成されると、それらのイベント・ファイルは削除されます。

WRITE TO TABLE イベント・モニターをドロップする場合、表情報は SYSCAT.EVENTTABLES カタログ・ビューからドロップされますが、表そのものはドロップされません。

FUNCTION
ドロップするユーザー定義関数 (完全な関数または関数テンプレートのいずれか) のインスタンスを指定します。指定する関数インスタンスは、カタログに記述されたユーザー定義関数でなければなりません。 CREATE DISTINCT TYPE ステートメントによって暗黙に生成された関数はドロップできません。

関数のインスタンスを指定する方法としては、次のようにいくつかの方法があります。

FUNCTION *function-name*
特定の関数を指定します。 *function-name* (関数名) の関数インスタンスが 1

つだけ存在している場合にのみ有効です。このように指定する関数には、任意の数のパラメーターが定義されていても構いません。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマにこの名前の関数が存在しない場合は、エラー (SQLSTATE 42704) になります。指定したスキーマまたは暗黙のスキーマに、この関数の特定インスタンスが複数存在する場合は、エラー (SQLSTATE 42725) になります。

FUNCTION *function-name* **(***data-type***,...)**
ドロップする関数を固有に指定する関数シグニチャーを指定します。関数選択のアルゴリズムは使用されません。

function-name
ドロップする関数の関数名を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(data-type,...)
これは、CREATE FUNCTION ステートメント上で (対応する位置に) 指定されたデータ・タイプに一致していなければなりません。データ・タイプ (*data-type*) の数、およびデータ・タイプを論理的に連結した値が、ドロップする特定の関数インスタンスを識別するのに使用されます。

data-type が修飾なしの場合は、 SQL パス上でスキーマを検索することによってタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントで指定された値と完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

RESTRICT
RESTRICT キーワードを指定すると、以下のいずれかの従属関係が存在する場合は、関数をドロップしないという規則が適用されます。

- 別のルーチンがその関数に基づいている。
- ビューがその関数を使用している。
- トリガーがその関数を使用している。
- マテリアライズ照会表が、その定義で関数を使用することはできません。

RESTRICT は、デフォルトの動作です。

指定したスキーマまたは暗黙のスキーマに、指定したシグニチャーを持つ関数がない場合は、エラー (SQLSTATE 42883) になります。

SPECIFIC FUNCTION *specific-name*
関数の作成時に指定された特定関数名、またはデフォルト値として使用された特定関数名を使用して、ドロップする特定のユーザー定義関数を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。
specific-name (特定名) は、指定したスキーマまたは暗黙のスキーマの特定関数のインスタンスを指定していなければなりません。そうでない場合、エラー (SQLSTATE 42704) になります。

RESTRICT
RESTRICT キーワードを指定すると、以下のいずれかの従属関係が存在する場合は、関数をドロップしないという規則が適用されます。

- 別のルーチンがその関数に基づいている。
- ビューがその関数を使用している。
- トリガーがその関数を使用している。

RESTRICT は、デフォルトの動作です。

SYSIBM、SYSFUN、または SYSPROC スキーマ (SQLSTATE 42832) の関数はドロップできません。

他のオブジェクトが関数に従属している場合があります。そのような関数をドロップする場合には、その前にそのような従属オブジェクトをすべてドロップしておく必要があります (作動不能としてマークされるパッケージは除く)。従属オブジェクトを伴う関数をドロップしようとすると、エラー (SQLSTATE 42893) になります。それらの従属関係のリストについては、537 ページを参照してください。

関数がドロップ可能な場合、その関数がドロップされます。

ドロップする特定関数に従属するパッケージがある場合には、それは作動不能としてマークされます。そのようなパッケージが暗黙のうちに再バインドされることはありません。 BIND コマンドまたは REBIND コマンドを使って再バインドするか、 PREP コマンドを使って再作成する必要があります。

FUNCTION MAPPING *function-mapping-name*
ドロップする関数マッピングを指定します。 *function-mapping-name* は、カタロ

グに記述されているユーザー定義関数マッピングを指定していなければなりません (SQLSTATE 42704)。その関数マッピングはデータベースから削除されます。

デフォルトの関数マッピングは、ドロップできませんが、 CREATE FUNCTION MAPPING ステートメントを使用することによって、使用不可にすることができます。デフォルトの関数マッピングをオーバーライドするために作成されたユーザー定義の関数マッピングをドロップすると、デフォルトの関数マッピングが復元されます。

ドロップされる関数マッピングに従属しているパッケージは、無効になります。

INDEX *index-name*

ドロップする索引または索引指定を指定します。 *index-name* (索引名) は、カタログに記述されている索引または索引指定を識別していなければなりません (SQLSTATE 42704)。索引は、システムに必須の主キーまたはユニーク制約の索引であってはならず、複製されたマテリアライズ照会表の索引であってもなりません (SQLSTATE 42917)。指定した索引または索引指定は削除されます。

ドロップされる索引または索引指定に従属しているパッケージは、無効になります。

INDEX EXTENSION *index-extension-name* **RESTRICT**

ドロップする索引拡張を指定します。 *index-extension-name* (索引拡張名) は、カタログに記述されている索引拡張を指定する名前でなければなりません (SQLSTATE 42704)。 RESTRICT キーワードは、この索引拡張の定義に従って索引を定義できないという規則を課します (SQLSTATE 42893)。

METHOD

ドロップするメソッド本体を指定します。指定するメソッドの本体は、カタログに記述されているメソッドでなければなりません (SQLSTATE 42704)。 CREATE TYPE ステートメントによって暗黙的に生成されたメソッド本体をドロップすることはできません。

DROP METHOD によって、メソッドの本体は削除されますが、メソッドの指定 (シグニチャー) はサブジェクト・タイプの定義の一部として残されます。メソッドの本体をドロップした後、メソッドの指定は ALTER TYPE DROP METHOD を使用してサブジェクト・タイプの定義から削除することができます。

ドロップするメソッド本体は、以下のようないくつかの方法で指定することができます。

METHOD *method-name*

ドロップする特定のメソッドを指定します。この方法は、対象となるタイプ *type-name* に、*method-name* という名前のメソッド・インスタンスが 1 つしかないことが明らかな場合にのみ有効です。この方法を用いる場合は、メソッドにいくつのパラメーターが定義されていても構いません。タイプ *type-name* に、指定された名前のメソッドが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。指定されたデータ・タイプに、そのメソッドの特定のインスタンスが複数存在する場合も、エラーが戻されます (SQLSTATE 42725)。

METHOD *method-name* **(***data-type***,...)**
ドロップするメソッドを一意的に識別できるメソッド・シグニチャーを指定します。メソッド選択のアルゴリズムは使用されません。

method-name
指定したタイプの中から、ドロップするメソッドのメソッド名を指定します。指定する名前は、修飾なしの ID でなければなりません。

(*data-type***,...)**
データ・タイプを指定します。ここで指定されるデータ・タイプは、CREATE TYPE または ALTER TYPE ステートメントで、メソッドの指定の対応する位置に指定されたデータ・タイプと一致していなければなりません。データ・タイプの数とデータ・タイプを論理的に連結した値から、ドロップする特定のメソッドが識別されます。

data-type が修飾なしの場合は、SQL パス上でスキーマを検索してタイプ名が決定されます。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE TYPE ステートメントで指定された値と完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定されたデータ・タイプに、指定されたシグニチャーを持つメソッドが存在しない場合は、エラーが戻されます (SQLSTATE 42883)。

FOR *type-name*
指定したメソッドのドロップを行うタイプを指定します。ここで指定される名前は、カタログにすでに記述されているタイプを示すものでなければなりません (SQLSTATE 42704)。動的 SQL ステートメントでは、 CURRENT SCHEMA 特殊レジスターが、修飾子のないタイプ名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/BIND オプションによって、修飾子のないタイプ名に修飾子が暗黙指定されます。

RESTRICT
RESTRICT キーワードを指定すると、以下のいずれかの従属関係が存在する場合は、メソッドをドロップしないという規則が適用されます。

- 別のルーチンがそのメソッドに基づいている。
- ビューがそのメソッドを使用している。
- トリガーがそのメソッドを使用している。

RESTRICT は、デフォルトの動作です。

SPECIFIC METHOD *specific-name*
CREATE TYPE または ALTER TYPE ステートメントにおいてユーザーが指定した名前、もしくはデフォルトで指定された名前を使用して、ドロップする特定のメソッドを識別します。特定名 (specific-name) に修飾子が付いていない場合、動的 SQL ステートメントでは、 CURRENT SCHEMA 特殊レジスターが、修飾子のない特定名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル / BIND オプションにより、修飾子のない特定名に修飾子が暗黙指定されます。 specific-name に指定される名前は、メソッドの名前でなければなりません。メソッド名ではない名前が指定された場合は、エラーになります (SQLSTATE 42704)。

RESTRICT
RESTRICT キーワードを指定すると、以下のいずれかの従属関係が存在する場合は、メソッドをドロップしないという規則が適用されます。

- 別のルーチンがそのメソッドに基づいている。
- ビューがそのメソッドを使用している。
- トリガーがその関数を使用している。

RESTRICT は、デフォルトのメソッドです。

メソッドに他のオブジェクトが従属している場合があります。そのような場合は、メソッドをドロップする前にそれらの従属関係をすべて除去する必要があります (ただし、そのメソッドがドロップされると作動不能としてマークされるパッケージは例外です)。そのような従属関係を持つメソッドをドロップしようとすると、エラーが戻されます (SQLSTATE 42893)。

ドロップできる状態にあれば、メソッドはドロップされます。

ドロップする特定のメソッドに従属しているパッケージは、そのメソッドがドロップされると、作動不能としてマークされます。そのようなパッケージが暗黙的に再バインドされることはありません。これらのパッケージは、BIND コマンドまたは REBIND コマンドを使用して再バインドするか、あるいは PREP コマンドを使用して再作成する必要があります。

ドロップされる特定のメソッドが別のメソッドをオーバーライドする場合、オーバーライドされるメソッド (および、ドロップされる特定のメソッドのスーパータイプでこのメソッドをオーバーライドするメソッド) に従属したパッケージはすべて無効になります。

NICKNAME *nickname*
ドロップするニックネームを指定します。ニックネームは、カタログにリストされていなければなりません (SQLSTATE 42704)。そのニックネームはデータベースから削除されます。

ニックネームに関連した列および索引に関するすべての情報が、カタログから削除されます。ニックネームに従属したマテリアライズ照会表はドロップされます。ニックネームに従属した索引指定はドロップされます。ニックネームに従属するビューは、作動不能としてマークされます。ドロップされた索引指定または作動不能ビューに従属するパッケージはいずれも無効になります。ニックネームが参照するデータ・ソース表は影響を受けません。

SQL 関数またはメソッドがニックネームに依存している場合、そのニックネームはドロップできません (SQLSTATE 42893)。

DATABASE PARTITION GROUP *db-partition-group-name*
ドロップするデータベース・パーティション・グループを指定します。*db-partition-group-name* パラメーターは、カタログに記述されているデータベース・パーティション・グループを指定していなければなりません (SQLSTATE 42704)。これは、1 つの部分からなる名前です。

データベース・パーティション・グループをドロップすると、データベース・パーティション・グループで定義されたすべての表スペースがドロップされます。そのような表スペース内の表に対して従属関係がある既存のデータベース・オブジェクト (パッケージや参照制約など) は、ドロップされるか、または無効になり (該当する場合)、従属するビューとトリガーは作動不能になります。

システム定義のデータベース・パーティション・グループはドロップできません (SQLSTATE 42832)。

現在データ再分散が行われているデータベース・パーティション・グループに対して DROP DATABASE PARTITION GROUP ステートメントを発行すると、データベース・パーティション・グループのドロップ操作は失敗し、エラーが戻されます (SQLSTATE 55038)。ただし、部分的に再分散されたデータベース・パーティション・グループはドロップできます。データベース・パーティション・グループは、 REDISTRIBUTE DATABASE PARTITION GROUP コマンドが完了するまで実行されなかった場合は、部分的に再分散の状態になります。これは、エラーまたは FORCE APPLICATION ALL コマンドによって割り込まれた場合に起こる可能性があります。 (部分的に再分散されたデータベース・パーティション・グループの場合、 SYSCAT.DBPARTITIONGROUPS カタログの REBALANCE_PMAP_ID は -1 ではありません。)

PACKAGE *schema-name.package-id*
ドロップするパッケージを指定します。スキーマ名が指定されていない場合、パッケージ ID は暗黙的にデフォルト・スキーマで修飾されます。スキーマ名およびパッケージ ID は、明示的または暗黙的に指定された バージョン ID とともに、カタログに記述されているパッケージを指定していなければなりません (SQLSTATE 42704)。指定したパッケージが削除されます。ドロップするパッケージが、*schema-name.package-id* で指定された唯一のパッケージである場合 (つまり、他のバージョンは存在しない場合)、そのパッケージに対する特権もすべて削除されます。

VERSION *version-id*
ドロップするパッケージ・バージョンを指定します。値が指定されない場合には、空ストリングがバージョンのデフォルトになります。同じパッケージ名が付けられていてもバージョンは異なる、複数のパッケージが存在する場合、 DROP ステートメントを 1 回呼び出すときに、1 つのパッケージ・バージョンだけをドロップできます。 次のような場合は、バージョン ID を二重引用符で区切ってください。

- バージョン ID が VERSION(AUTO) プリコンパイラー・オプションによって生成された場合
- バージョン ID が数字で始まる場合
- バージョン ID が小文字であったり、大小混合である場合

ステートメントをオペレーティング・システムのコマンド・プロンプトから呼び出す場合は、各二重引用符の区切り文字の前に円記号を置いて、オペレーティング・システムによって区切り文字が外されないようにします。

PROCEDURE
ドロップするストアード・プロシージャーのインスタンスを指定します。指定するプロシージャー・インスタンスは、カタログに記述されたストアード・プロシージャーでなければなりません。

プロシージャー・インスタンスを指定する方法としては、次のようにいくつかの方法があります。

PROCEDURE *procedure-name*
特定のプロシージャーを指定します。この方法は、*procedure-name* で指定したプロシージャー・インスタンスがスキーマ内に 1 つしか存在しないことが明らかな場合にのみ有効です。この方法で指定するプロシージャーには、パラメーターがいくつ定義されていても構いません。指定したスキーマまたは暗黙のスキーマに該当する名前のプロシージャーが存在しない場合は、エラーが戻されます (SQLSTATE 42704)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。指定したスキーマまたは暗黙のスキーマにこのプロシージャーの特定インスタンスが複数存在する場合は、エラーが戻されます (SQLSTATE 42725)。

RESTRICT
RESTRICT キーワードを指定すると、トリガー定義、SQL 関数、または SQL メソッドにプロシージャーの名前が付けられた CALL ステートメントが含まれる場合に、そのプロシージャーはドロップされずに済みます。 RESTRICT は、デフォルトの動作です。

PROCEDURE *procedure-name* **(***data-type***,...)**
ドロップするプロシージャーを一意に識別するプロシージャー・シグニチャーを指定します。プロシージャー選択のアルゴリズムは使用されません。

procedure-name
ドロップするプロシージャーのプロシージャー名を指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。

(data-type,...)
データ・タイプを指定します。ここで指定されるデータ・タイプは、CREATE PROCEDURE ステートメントの対応する位置に指定されたデータ・タイプと一致していなければなりません。データ・タイプ (*data-type*) の数、およびデータ・タイプを論理的に連結した値を使用して、ドロップする特定のプロシージャーが識別されます。

data-type が修飾なしの場合は、 SQL パス上でスキーマを検索することによってタイプ名が決定されます。 REFERENCE タイプに指定するデータ・タイプ名にも同様の規則が当てはまります。

パラメーター化データ・タイプの長さ、精度、または位取りを指定する必要はありません。空の括弧をコーディングすることによって、一致データ・タイプの検索時にそれらの属性を無視するように指定することができます。

パラメーター値が異なるデータ・タイプ (REAL または DOUBLE) を示しているため、 FLOAT() を使用することはできません (SQLSTATE 42601)。

ただし、長さ、精度、または位取りをコーディングする場合、その値は、CREATE FUNCTION ステートメントにおける指定に完全に一致していなければなりません。

0<n<25 は REAL を意味し、24<n<54 は DOUBLE を意味するので、FLOAT(n) のタイプは、n に定義された値と一致している必要はありません。マッチングは、タイプが REAL か DOUBLE かに基づいて行われます。

指定したスキーマまたは暗黙のスキーマに、指定されたシグニチャーを持つプロシージャーがない場合は、エラーが戻されます (SQLSTATE 42883)。

SPECIFIC PROCEDURE *specific-name*
プロシージャーの作成時にユーザーが指定した特定のプロシージャー名か、デフォルト値として与えられたプロシージャー名を使用して、ドロップする特定のストアード・プロシージャーを識別します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *specific-name* に指定される名前は、指定したスキーマまたは暗黙のスキーマに含まれる特定プロシージャーのインスタンスを識別するものでなければなりません。それ以外の名前が指定された場合は、エラーが戻されます (SQLSTATE 42704)。

RESTRICT
RESTRICT キーワードを指定すると、トリガー定義、SQL 関数、または SQL メソッドにプロシージャーの名前が付けられた CALL ステートメントが含まれる場合に、そのプロシージャーはドロップされずに済みます。 RESTRICT は、デフォルトの動作です。

SYSIBM、SYSFUN、または SYSPROC スキーマのプロシージャーはドロップできません (SQLSTATE 42832)。

SCHEMA *schema-name* **RESTRICT**
ドロップする特定のスキーマを指定します。 *schema-name* に指定するスキーマ名は、カタログに記述されているスキーマを識別するものでなければなりません (SQLSTATE 42704)。 RESTRICT キーワードは、データベースから削除するスキーマとして指定したスキーマにオブジェクトを定義できないという規則を課します (SQLSTATE 42893)。

SEQUENCE *sequence-name*
ドロップする特定のシーケンスを識別します。暗黙的または明示的スキーマ名を含む *sequence-name* は、現在のサーバーに存在するシーケンスを固有に識別し

ていなければなりません。この名前によるシーケンスが、明示的または暗黙的に指定されたスキーマに存在しない場合、エラー (SQLSTATE 42704) が起こります。

RESTRICT オプションはデフォルトで、以下のいずれかの従属関係が存在する場合は、シーケンスがドロップされないようにします。

- トリガーの NEXT VALUE または PREVIOUS VALUE 式がこのシーケンスを指定するようなトリガーが存在する (SQLSTATE 42893)。
- ルーチン本体の NEXT VALUE 式がこのシーケンスを指定するような SQL 関数または SQL メソッドが存在する (SQLSTATE 42893)。

SERVER *server-name*

カタログから定義をドロップするデータ・ソースを指定します。 *server-name* に指定するサーバー名は、カタログに記述されているデータ・ソースを識別するものでなければなりません (SQLSTATE 42704)。そのデータ・ソースの定義は削除されます。

データ・ソースに常駐する表およびビューのニックネームはすべてドロップされます。また、これらのニックネームに従属する索引指定もすべてドロップされます。ドロップされたサーバー定義に従属するユーザー定義関数マッピング、ユーザー定義タイプ・マッピング、およびユーザー・マッピングもすべてドロップされます。ドロップされたサーバー定義、関数マッピング、ニックネーム、および索引指定に依存するパッケージはすべて無効になります。

TABLE *table-name*

ドロップする基本表または宣言済み一時表を指定します。 *table-name* に指定する表名は、カタログに記述されている表が、宣言済み一時表を指定する場合は、スキーマ名 SESSION によって決められ、アプリケーションに存在する一時表の名前でなければなりません (SQLSTATE 42704)。型付き表の副表は、それぞれスーパー表に従属しています。したがって、スーパー表をドロップする前には、副表をすべてドロップする必要があります (SQLSTATE 42893)。指定された表はデータベースから削除されます。

その表を参照するすべての索引、主キー、外部キー、チェック制約、マテリアライズ照会表、およびステージング表はドロップされます。表を参照するすべてのビューおよびトリガーは、作動不能になります。 (これには、CREATE TRIGGER ステートメントの ON 文節で参照されている表と、トリガー SQL ステートメントで参照されているすべての表が含まれます。) ドロップされたオブジェクトまたは作動不能としてマークされたオブジェクトに従属するすべてのパッケージは無効になります。これには、副表よりも上位の階層であるスーパー表に従属するパッケージが含まれます。参照列の中で、ドロップされた表を参照の有効範囲として定義したものがあれば、参照範囲は無効になります。

宣言済み一時表にパッケージが従属することはありません。したがって、宣言済み一時表がドロップされてもパッケージが無効になることはありません。

DATALINK 列にリンクされたファイルはすべてリンク解除されます。リンク解除操作は非同期で実行されるので、ファイルを他の操作ですぐに使用することはできない場合があります。

フェデレーテッド・システムでは、透過性 DDL を使用して作成されたリモート表はドロップできます。リモート表をドロップすると、その表に関連したニックネームもドロップされ、そのニックネームに従属するパッケージが無効化されます。

表階層から副表をドロップすると、その副表に関連した列はアクセスできなくなります (ただし、列の数や行のサイズの制限に関しては考慮されます)。副表をドロップすると、スーパー表から副表の行がすべて削除されてしまいます。その結果、スーパー表に定義したトリガーや参照保全制約が活動化することがあります。

宣言済み一時表が、現在の作業単位またはセーブポイントがアクティブになる前に作成されたものである場合は、その一時表をドロップすると機能上で表がドロップされてしまうため、アプリケーションからその一時表にアクセスすることができなくなります。しかし、表スペースでは、作業単位がコミットされるまで、あるいはセーブポイントが終了するまで、依然としてこの表が予約された状態にあるため、USER TEMPORARY 表スペースをドロップしたり、 USER TEMPORARY 表スペースのデータベース・パーティション・グループを再分散することはできません。宣言済み一時表がドロップされると、DROP がコミットされたかロールバックされたかにかかわらず、表に含まれていたデータはすべて破棄されます。

表に RESTRICT ON DROP 属性があると、その表はドロップできません。

TABLE HIERARCHY *root-table-name*
ドロップする型付き表階層を指定します。 *root-table-name* で指定する型付き表は、型付き表階層のルート表でなければなりません (SQLSTATE 428DR)。 *root-table-name* で指定する型付き表とその表のすべての副表が、データベースから削除されます。

ドロップされた表を参照するすべての索引、マテリアライズ照会表、ステージング表、主キー、外部キー、およびチェック制約はドロップされます。ドロップされた表を参照するすべてのビューおよびトリガーは、作動不能になります。ドロップされたオブジェクトまたは作動不能としてマークされたオブジェクトに従属するすべてのパッケージは無効になります。参照列の中で、ドロップされた表を参照の有効範囲として定義したものがあれば、参照範囲は無効になります。

DATALINK 列にリンクされたファイルはすべてリンク解除されます。リンク解除操作は非同期で実行されるので、ファイルを他の操作ですぐに使用することはできない場合があります。

単一の副表をドロップする場合とは違い、表階層をドロップしても、階層内にある任意の表の削除トリガーが活動化したり、削除された行が記録されたりすることはありません。

TABLESPACE または **TABLESPACES** *tablespace-name*
ドロップされる表スペースを指定します。 *tablespace-name* (表スペース名) は、カタログに記述されている表スペースを指定していなければなりません (SQLSTATE 42704)。これは、1 つの部分からなる名前です。

表の一部がドロップされる表スペースに保管され、 1 つかそれ以上の部分がドロップされない別の表スペースに保管されている場合、この表スペースはドロップされません (このような表は前もってドロップする必要があります)。また、その表スペースに存在する表に RESTRICT ON DROP 属性がある場合も、この

表スペースはドロップされません (SQLSTATE 55024)。システム表スペースはドロップできません (SQLSTATE 42832)。データベースに TEMPORARY 表スペースが 1 つしか存在しない場合は、 SYSTEM TEMPORARY 表スペースをドロップすることはできません (SQLSTATE 55026)。宣言済み一時表が作成されている USER TEMPORARY 表スペースはドロップできません (SQLSTATE 55039)。 USER TEMPORARY 表スペースでは、宣言済み一時表がドロップされても、 DROP TABLE を含む作業単位がコミットされるまでは、その表スペースは使用中と見なされます。

表スペースをドロップすると、その表スペースに定義されているオブジェクトはすべてドロップされます。パッケージや参照制約などのその表スペースに従属する既存のすべてのデータベース・オブジェクトはドロップされるか、または無効になり、従属しているビューやトリガーは作動不能になります。

ユーザーによって作成されたコンテナーは削除されません。 CREATE TABLESPACE でデータベース・マネージャーによって作成されたコンテナー名のパスに含まれているディレクトリーは、いずれも削除されます。データベース・ディレクトリーの下にあるすべてのコンテナーは削除されます。 DROP TABLESPACE がコミットされると、可能なら、指定された表スペースの DMS ファイル・コンテナーや SMS コンテナーが削除されます。コンテナーが削除できない場合 (たとえば、別のエージェントによってオープンされたままになっている場合など) は、ファイルが長さ 0 に切り捨てられます。これらの長さ 0 のファイルは、すべての接続が終了するか、DEACTIVATE DATABASE コマンドが発行されたときに削除されます。

TRANSFORM ALL FOR *type-name*

ユーザー定義データ・タイプ *type-name* に定義されたすべてのトランスフォーム・グループがドロップされることを示します。これらのグループで参照されるトランスフォーム関数はドロップされません。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *type-name* に指定されるタイプ名は、カタログに記述されているユーザー定義タイプを識別するものでなければなりません (SQLSTATE 42704)。

type-name に定義されているトランスフォームが存在しない場合は、エラーが戻されます (SQLSTATE 42740)。

DROP TRANSFORM は、CREATE TRANSFORM の逆の処理を行います。 DROP TRANSFORM は、指定されたデータ・タイプで特定のグループに関連付けられたトランスフォーム関数を未定義の状態にします。これらのグループに関連付けられていた関数は引き続き存在しており、明示的に呼び出すことができますが、これらの関数にはもはやトランスフォーム・プロパティーは含まれていないので、ホスト言語環境で値を交換するためにこれらの関数が暗黙的に呼び出されることはありません。

トランスフォーム・グループの中に SQL 以外の言語で書かれたユーザー定義関数 (またはメソッド) があり、その関数が、ユーザー定義タイプ *type-name* に定義されたそのグループのトランスフォーム関数のいずれかに従属している場合、そのトランスフォーム・グループはドロップされません (SQLSTATE 42893)。このようなユーザー定義関数が従属しているトランスフォーム関数は、

type-name で定義された参照先のトランスフォーム・グループに関連付けられています。そのため、パッケージが属しているトランスフォーム関数が、指定されたトランスフォーム・グループと関連付けられていると、そのパッケージは作動不能としてマークされてしまいます。

TRANSFORMS *group-name* **FOR** *type-name*
ユーザーが定義したデータ・タイプ *type-name* から、指定したトランスフォーム・グループがドロップされることを示します。このグループで参照されるトランスフォーム関数はドロップされません。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないオブジェクト名の修飾子が暗黙指定されます。 *type-name* に指定されるタイプ名は、カタログに記述されているユーザー定義タイプを識別するものでなければなりません (SQLSTATE 42704)。また、*group-name* には、 *type-name* に存在しているトランスフォーム・グループを指定しなければなりません。

TRIGGER *trigger-name*
ドロップするトリガーを指定します。 *trigger-name* (トリガー名) は、カタログに記述されているトリガーを指定していなければなりません (SQLSTATE 42704)。 指定したトリガーは削除されます。

トリガーをドロップすると、特定のパッケージが無効としてマークされます。

trigger-name がビューに対して INSTEAD OF トリガーを指定する場合、そのビューに対する更新を行うことにより、他のトリガーはそのトリガーに従属できます。

TYPE *type-name*
ドロップするユーザー定義タイプを指定します。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターが、修飾子のないオブジェクト名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションにより、修飾子のないオブジェクト名の修飾子が暗黙指定されます。構造タイプでは、関連した参照タイプもドロップされます。 *type-name* (タイプ名) は、カタログに記述されているユーザー定義タイプを指定していなければなりません。 DISTINCT 文節が指定されている場合、*type-name* (タイプ名) は、カタログに記述されている特殊タイプを指定していなければなりません。

RESTRICT
以下の場合、このタイプはドロップされません (SQLSTATE 42893)。

- 表またはビューの列のタイプとして使用されるタイプである。
- サブタイプが含まれている。
- 型付き表または型付きビューのデータ・タイプとして使用されている構造タイプである。
- 他の構造タイプの属性として使用されるタイプである。
- 表の列のタイプに *type-name* のインスタンスが含まれている可能性がある。これには、列のタイプが *type-name* である場合や、列に関連付けられたタイプ階層以外のロケーションで *type-name* が使用される場合などがあります。もっと典型的な例としては、どのタイプ (T) であれ、表の

列のタイプで *type-name* が直接または間接的に使用されている場合には、 T をドロップすることはできません。

- タイプが、表またはビューの参照タイプ列のターゲット・タイプ、または別の構造タイプの参照タイプ属性である。
- このタイプ、あるいはこのタイプを参照する値が、関数やメソッドのパラメーター・タイプまたは戻り値タイプである。
- このタイプ、またはこのタイプを参照する値が SQL 関数やメソッドの本体で使用されているが、パラメーター・タイプや戻り値タイプではない。
- このタイプがチェック制約、トリガー、ビュー定義、または索引の拡張で使用されている。

RESTRICT が指定されていない場合の振る舞いは、そのタイプを使用する関数およびメソッドの場合を除いて RESTRICT の振る舞いと同じです。

ドロップされるタイプを使用する関数の場合、ユーザー定義タイプがドロップ可能であると、ドロップするそのタイプ (またはドロップするタイプを参照するもの) のパラメーターまたは戻り値が含まれているすべての関数 (F) (特定名は SF) に、以下の DROP FUNCTION ステートメントが実行されることになります。

```
DROP SPECIFIC FUNCTION SF
```

このステートメントがカスケードして、従属する関数もドロップされる可能性があります。ユーザー定義タイプへの従属関係に基づいて、それらの関数もすべてドロップ・リストに含まれている場合には、ユーザー定義タイプのドロップは正常に処理されます (そうでない場合、SQLSTATE 42893 のエラーになります)。

ドロップされるタイプを使用するメソッドの場合、ユーザー定義タイプがドロップ可能であると、ドロップするそのタイプ (またはドロップするタイプを参照するもの) のパラメーターまたは戻り値が含まれているタイプ T1 のメソッド (M) (特定名は SM) に、以下のステートメントが実行されることになります。

```
DROP SPECIFIC METHOD SM
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

これらのメソッドに従属しているオブジェクトがあると、DROP TYPE 操作が失敗する場合があります。

ドロップするタイプのスーパータイプで定義されるメソッドに従属し、オーバーライドに適したパッケージはすべて、無効になります。

TYPE MAPPING *type-mapping-name*
ドロップするユーザー定義のデータ・タイプ・マッピングを指定します。 *type-mapping-name* (タイプ・マッピング名) は、カタログに記述されているデータ・タイプ・マッピングを指定していなければなりません (SQLSTATE 42704)。 指定したデータ・タイプ・マッピングがデータベースから削除されます。

その他にドロップされるオブジェクトはありません。

USER MAPPING FOR *authorization-name* | **USER SERVER** *server-name*
ドロップするユーザー・マッピングを指定します。このマッピングは、フェデレーテッド・データベースにアクセスするために使う許可名を、データ・ソースに

アクセスするために使う許可名に関連付けます。これら 2 つのうち最初の許可名は、*authorization-name* で指定されるか、または特殊レジスター USER によって参照されます。 *server-name* は、アクセスするのに 2 番目の許可名を使用するデータ・ソースを指定します。

authorization-name は、カタログにリストされていなければなりません (SQLSTATE 42704)。 *server-name* に指定するサーバー名は、カタログに記述されているデータ・ソースを識別するものでなければなりません (SQLSTATE 42704)。ユーザー・マッピングが削除されます。

その他にドロップされるオブジェクトはありません。

VIEW *view-name*
ドロップするビューを指定します。 *view-name* (ビュー名) は、カタログに記述されているビューを指定していなければなりません (SQLSTATE 42704)。型付き表のサブビューは、それぞれスーパービューに従属しています。したがって、スーパービューをドロップする前に、サブビューをすべてドロップする必要があります (SQLSTATE 42893)。

指定したビューは削除されます。直接的または間接的にそのビューに従属するビューまたはトリガーの定義は、作動不能としてマークされます。作動不能というマークが付いた表に従属するマテリアライズ照会表はすべてドロップされます。ドロップされたビューまたは作動不能としてマークされたビューに従属するパッケージはいずれも無効になります。これには、サブビューよりも上位の階層であるスーパービューに従属するパッケージが含まれます。参照列の中で、ドロップされたビューを参照の有効範囲として定義したものがあれば、参照範囲は無効になります。

VIEW HIERARCHY *root-view-name*
ドロップする型付きビュー階層を指定します。 *root-view-name* で指定する型付きビューは、型付きビュー階層のルート・ビューでなければなりません (SQLSTATE 428DR)。 *root-view-name* で指定する型付きビューとそのビューのすべてのサブビューが、データベースから削除されます。

直接的または間接的にドロップされたビューに従属するビューまたはトリガーの定義は、作動不能としてマークされます。ドロップされたビューやトリガー、または作動不能としてマークされたビューやトリガーに従属するパッケージはいずれも、無効になります。参照列の中で、ドロップされたビューや作動不能とマークされたビューを参照の有効範囲として定義したものがあれば、参照範囲は無効になります。

WRAPPER *wrapper-name*
ドロップするラッパーを指定します。 *wrapper-name* (ラッパー名) は、カタログに記述されているラッパーを指定していなければなりません (SQLSTATE 42704)。 そのラッパーは削除されます。

そのラッパーに従属するすべてのサーバー定義、ユーザー定義関数マッピング、およびユーザー定義データ・タイプ・マッピングはドロップされます。 ドロップされたサーバー定義に従属するユーザー定義関数マッピング、ニックネーム、ユーザー定義データ・タイプ・マッピング、およびユーザー・マッピングもすべてドロップされます。 ドロップされたニックネームに従属する索引指定はすべてドロップされ、こうしたニックネームに従属するビューはすべて、作動不能と

してマークが付けられます。 ドロップされたオブジェクトと作動不能ビューに従属するすべてのパッケージは無効になります。

規則:

従属関係: 538 ページの表 10 は、オブジェクト相互間従属関係を示します。カタログには明示的に記録されない従属関係があります。たとえば、パッケージが従属している制約の記録はありません。このリストには、以下の 4 つの異なるタイプの従属関係が示されています。

R 制限 (Restrict) を意味します。従属オブジェクトが存在する限り、その基礎となるオブジェクトはドロップできません。

C カスケード (Cascade) を意味します。基礎となるオブジェクトをドロップすると、その従属オブジェクトも同時にドロップされます。ただし、その従属オブジェクトにさらに他のオブジェクトに対する制限 (R) 従属関係があり、それによってその従属オブジェクトをドロップできない場合には、基礎となるオブジェクトのドロップは失敗します。

X 作動不能 (Inoperative) を意味します。基礎となるオブジェクトをドロップすると、その従属オブジェクトは作動不能になります。ユーザーが何らかの明示的な処置を取るまで、それは作動不能のままになります。

A 自動無効化 / 再有効化 (Automatic Invalidation/Revalidation) を意味します。基礎となるオブジェクトをドロップすると、従属オブジェクトは無効になります。データベース・マネージャーは、無効になったオブジェクトを再度有効にしようとします。

関数またはメソッドによって使用されるか、関数またはメソッドによって直接あるいは間接に呼び出されるプロシージャーによって使用されるパッケージは、ルーチンが MODIFIES SQL DATA として定義される場合にのみ、自動的に再度有効にされます。ルーチンが MODIFIES SQL DATA でなければ、エラーが戻されます (SQLSTATE 56098)。

DROP ステートメントのパラメーターおよびオブジェクトには、結果的にブランク行または列になるため、 538 ページの表 10 に示されていないものもあります。

- EVENT MONITOR、PACKAGE、PROCEDURE、SCHEMA、TYPE MAPPING、および USER MAPPING DROP ステートメントには、オブジェクトの従属関係はありません。
- 別名、バッファー・プール、パーティション・キー、特権、およびプロシージャーのオブジェクト・タイプには、 DROP ステートメントの従属関係はありません。
- 指定した作業単位 (UOW) の内側にある A DROP SERVER、 DROP FUNCTION MAPPING、または DROP TYPE MAPPING ステートメントは、以下に示すいずれかの条件下で処理することはできません。
  - ステートメントが単一のデータ・ソースを参照し、このデータ・ソース内の表またはビューのニックネームを参照する SELECT ステートメントが、 UOW にすでに含まれている場合 (SQLSTATE 55006)。
  - ステートメントがデータ・ソースの区分 (たとえば、特定のタイプおよびバージョンのすべてのデータ・ソース) を参照し、こうしたデータ・ソースの 1 つ

の内側にある表またはビューのニックネームを参照する SELECT ステートメントが、 UOW にすでに含まれている場合 (SQLSTATE 55006)。

表 10. 従属関係

オブジェクト・タイプ → / ステートメント ↓	CONSTRAINT	FUNCTION	FUNCMAPPING	INDEX	INDEXEXTENSION	METHOD	NICKNAME	NODEGROUP	PACKAGE[31]	SERVER	TABLE	TABLESPACE	TRIGGER	TYPE	TYPEMAPPING	USERMAPPING	VIEW
ALTER FUNCTION	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER NICKNAME (ローカル名またはローカル・タイプを変更する)	R[33]	R	-	-	-	R	-	-	A	-	R	-	-	-	-	-	R
ALTER NICKNAME (列オプションまたはニックネーム・オプションを変更する)	-	-	-	-	-	-	-	-	A	-	R	-	-	-	-	-	-
ALTER NICKNAME (制約を追加、変更、またはドロップする)	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER PROCEDURE	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER SERVER	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	R[20]	A[1]	-	-	-	-	-	-	-	-
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	R[27]	-	-	-	-	-	-	-	-	-	-	-
CREATE METHOD	-	-	-	-	-	-	-	-	A[28]	-	-	-	-	-	-	-	-
CREATE TYPE	-	-	-	-	-	-	-	-	A[29]	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	-	-	-	-	-	A[3]	-	R[3]	-	X[3]	-	-	-	X[3]
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-
DROP DATABASE PARTITION GROUP	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-

表 10. 従属関係 (続き)

オブジェクト・タイプ → / ステートメント ↓	CONSTRAINT	FUNCTION	FUNCMAPPING	INDEX	INDEXEXTENSION	METHOD	NICKNAME	NODEGROUP	PACKAGE[31]	SERVER	TABLE	TABLESPACE	TRIGGER	TYPE	TYPEMAPPING	USERMAPPING	VIEW
DROP FUNCTION	R	R[7]	R	-	R	R[7]	-	-	X	-	R	-	R	-	-	-	R
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	R[17]
DROP INDEX EXTENSION	-	R	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R[7]	R	-	R	R	-	-	X/A[30]	-	R	-	R	-	-	-	R
DROP NICKNAME	-	R	-	C	-	R	-	-	A	-	C[11]	-	-	-	-	-	X[16]
DROP PROCEDURE	-	R[7]	-	-	-	R[7]	-	-	A	-	-	-	R	-	-	-	-
DROP SEQUENCE	-	R	-	-	-	R	-	-	A	-	-	-	R	-	-	-	-
DROP SERVER	-	C[21]	C[19]	-	-	-	C	-	A	-	-	-	-	-	C[19]	C	-
DROP TABLE[32]	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLE HIERARCHY	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLESPACE	-	-	-	C[6]	-	-	-	-	-	-	CR[6]	-	-	-	-	-	-
DROP TRANSFORM	-	R	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-
DROP TRIGGER	-	-	-	-	-	-	-	-	A[1]	-	-	-	X[26]	-	-	-	-
DROP TYPE	R[13]	R[5]	-	-	R	-	-	-	A[12]	-	R[18]	-	R[13]	R[4]	-	-	R[14]
DROP VIEW	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[15]
DROP VIEW HIERARCHY	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[16]
DROP WRAPPER	-	-	C	-	-	-	-	-	-	C	-	-	-	-	C	-	-
REVOKE 特権 [10]	-	CR[25]	-	-	-	CR[25]	-	-	A[1]	-	CX[8]	-	X	-	-	-	X[8]

1 この従属関係は、これらの制約、トリガー、またはパーティション・キーを持つ表に従属することによって、暗黙的に決まります。

2 パッケージに、ビューに影響を与える INSERT、UPDATE、または DELETE ステートメントが含まれている場合、そのパッケージはビューの基礎となる基本表に対して挿入、更新、または削除の操作を行うことになります。 UPDATE の場合、パッケージは UPDATE によって修正される基本表の各列ごとに更新操作を行います。

型付きビューに対して操作を行うステートメントがパッケージに含まれている場合、同じビュー階層内でビューを作成したりドロップしたりすると、パッケージが無効になります。

3 パッケージ、マテリアライズ照会表、ステージング表、ビュー、トリガーが

別名を使用する場合、その別名と、その別名が参照するオブジェクトの両方に従属することになります。別名がチェーニングしている場合、そのチェーンの中の別名ごとに従属関係が作成されます。

別名自体は、どのような従属関係も持ちません。存在していないオブジェクトに対しても、別名を定義できます。

4 あるユーザー定義タイプ T を別のユーザー定義タイプ B に従属させるには、 T が以下の条件を満たしていなければなりません。

- 属性のデータ・タイプとして B を指定している
- REF(B) の属性を持っている
- スーパータイプとして B を持っている

5 データ・タイプをドロップすると、その効果がカスケードして、パラメーターや結果タイプとしてそのデータ・タイプを使用する関数やメソッド、そしてそのデータ・タイプで定義されているメソッドもドロップされることになります。それらの関数やメソッドが互いに依存していても、そのことがそれらの関数やメソッドのドロップを防ぐことにはなりません。ただし、本体でそのデータ・タイプを使用している関数やメソッドに対しては、制約セマンティクスが適用されます。

6 表スペースまたは表スペースのリストをドロップすると、指定した表スペース内に完全に含まれているすべての表やリストがドロップされることになります。ただし、表が複数の表スペース (異なる表スペース内の索引または長形式列) にわたり、そうした表スペースがドロップされるリストにない場合、これらの表スペースは表が存在する限りはドロップできません。

7 従属関数が SOURCE 文節内の基本関数の名前である場合、その関数は別の特定の関数に従属します。また、従属のルーチンが SQL で書かれており、その本体で基本のルーチンを使用する場合も、関数やメソッドは別の特定の関数やメソッドに従属することができます。加えて、構造タイプのパラメーターや戻りタイプをもつ外部のメソッドや関数も、 1 つまたは複数のトランスフォーム関数に従属することができます。

8 マテリアライズ照会表がドロップされたり、ビューが作動不能になるのは、SELECT 特権がない場合だけです。作動不能にされたビューが型付きビュー階層に含まれていれば、そのサブビューもすべて作動不能になります。

9 パッケージに、表 T に影響を与える INSERT、UPDATE、または DELETE ステートメントが含まれている場合、そのパッケージは T に対して挿入、更新、または削除の操作を行うことになります。 UPDATE の場合、パッケージは UPDATE によって修正される T の各列ごとに更新操作を行います。

型付き表に対して操作を行うステートメントがパッケージに含まれている場合、同じ表階層内で表を作成したりドロップしたりすると、パッケージが無効になります。

10 列に対する特権を個々に取り消すことはできないので、列レベルでの従属関係は存在しません。

パッケージ、トリガー、またはビューの FROM 文節で OUTER(Z) が使用されている場合、 Z のすべての副表またはサブビューで SELECT 特権に対する従属関係が存在します。同じように、パッケージ、トリガー、または

ビューで DEREF(*Y*) が使用されていて、 *Y* が *Z* という表またはビューをターゲットとする参照タイプである場合、*Z* のすべての副表またはサブビューで SELECT 特権に対する従属関係が存在します。

11 マテリアライズ照会表は、基礎表、あるいは表定義の全選択で指定されたニックネームに従属しています。

カスケードのセマンティクスが、従属するマテリアライズ照会表に適用されます。

副表はスーパー表に従属しており、この従属関係はルート表にまで及びます。従属するすべての副表がドロップされるまで、スーパー表はドロップできません。

12 TYPE 述部またはサブタイプ処理の式 (TREAT *expression* AS *data-type*) を使用した結果、パッケージは構造タイプに従属することができます。パッケージは、TYPE 述部の右辺、または TREAT 式の右辺で指定した各構造タイプのサブタイプすべてと従属関係にあります。構造タイプをドロップしたり作成したりして、パッケージと従属関係にあるサブタイプを変更すると、ステートメントが無効になる場合があります。

ドロップするタイプのスーパータイプで定義されるメソッドに従属し、オーバーライドに適したパッケージはすべて、無効になります。

13 あるタイプがチェック制約またはトリガーで使用されている場合、チェック制約またはトリガーはこのタイプに従属する関係にあります。チェック制約またはトリガーの TYPE 述部で使用される、構造タイプのサブタイプに従属しません。

14 あるタイプがビュー定義で使用されている場合、ビューはこのタイプに従属する関係にあります (型付きビューのタイプも含まれます)。ビュー定義内の TYPE 述部で使用される、構造タイプのサブタイプに従属しません。

15 サブビューはスーパービューに従属しており、この従属関係はルート・ビューにまで及びます。従属するすべてのサブビューがドロップされるまで、スーパービューはドロップできません。ビューの従属関係の詳細については、16 を参照してください。

16 トリガーまたはビューは逆参照操作または DEREF 関数のターゲット表やターゲット・ビューにも従属しています。 FROM 文節のトリガーまたはビューで OUTER(Z) を含むものは、トリガーまたはビューが作成された時点に存在した *Z* の副表またはサブビューすべてに対して従属関係にあります。

17 型付きビューはユニーク索引が存在しているかどうかに依存していることがあり、それによってオブジェクト ID 列がユニークなものにすることができます。

18 表はユーザー定義データ・タイプ (特殊タイプまたは構造タイプ) に従属している場合があり、それには以下の理由があります。

- そのタイプが列のタイプとして使用されている
- そのタイプが表のタイプとして使用されている
- そのタイプが表のタイプの属性として使用されている
- そのタイプが、表の列タイプまたは表のタイプの属性を表す、参照タイプのターゲット・タイプとして使用されている

- そのタイプが、表の列のタイプによって直接または間接的に使用されている

19 サーバーをドロップすると、カスケード的に、そのネーム・サーバーに作成した関数マッピングとタイプ・マッピングがドロップされます。

20 複数パーティションのデータベース・パーティション・グループにある表に対してパーティション・キーが定義されている場合、このパーティション・キーは必須です。

21 従属している OLE DB 表関数に ″R″ 従属オブジェクト (DROP FUNCTION を参照) が含まれている場合は、サーバーをドロップできません。

22 SQL 関数またはメソッドは、その本体によって参照されるオブジェクトに従属することができます。

23 *type-name* T の タイプ TA の属性 A がドロップされると、以下の DROP ステートメントが実際に実行されます。

```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
  DROP METHOD A(TA)
  DROP METHOD A()
```

24 次のような場合に、表はユーザー定義による構造データ・タイプの属性に従属することがあります。

1. 表が、*type-name* またはそのサブタイプのいずれかに基づく型付き表である。
2. 表に、*type-name* を直接または間接的に参照するタイプの列が含まれている。

25 定義された関数またはメソッド本体に SELECT 特権がなくなると、 SQL 関数の本体またはメソッド本体で使用される表またはビューに対する SELECT 特権の REVOKE により、特権を失った関数またはメソッド本体のドロップが試行されます。これらの関数またはメソッド本体がビュー、トリガー、関数、またはメソッド本体で使用されている場合は、これをドロップすることはできないので、結果として REVOKE が制約されます。それ以外の場合は、REVOKE がカスケードしてそれらの関数はドロップされます。

26 トリガーは、INSTEAD OF トリガーが定義されるビューを変更して、INSTEAD OF トリガーが実行される場合、INSTEAD OF トリガーに従属します。

27 他のメソッドによってオーバーライドされた元のメソッドのメソッド宣言は、ドロップすることができません (SQLSTATE -2)。

28 作成されるメソッド本体のメソッドが、別のメソッドをオーバーライドするものと宣言される場合、オーバーライドされるメソッド (および、作成されるメソッドのスーパータイプでこのメソッドをオーバーライドするメソッド) に従属したパッケージはすべて無効になります。

29 既存のタイプの新しいサブタイプが作成されると、作成されるタイプのスー

パータイプで定義されるメソッド (および、オーバーライドに適しているメソッド (たとえば、 no mutator や observer)) に従属するパッケージはすべて無効になります。

30 ドロップされるメソッド本体の特定メソッドが、別のメソッドをオーバーライドするものと宣言される場合、オーバーライドされるメソッド (および、ドロップされる特定メソッドのスーパータイプでこのメソッドをオーバーライドするメソッド) に従属したパッケージはすべて無効になります。

31 キャッシュに入れられた動的 SQL には、パッケージと同じセマンティクスがあります。

32 DROP TABLE ステートメントを使ってリモート基本表をドロップするときには、ニックネームとリモート基本表の両方がドロップされます。

33 外部キーが参照していない主キーまたはユニーク・キーは、ニックネームのローカル名やローカル・タイプの変更を制限しません。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DATABASE PARTITION GROUP の代わりに NODEGROUP を指定できます。
  - DB2 UDB for OS/390 and z/OS との互換性:
    - ALIAS の代わりに SYNONYM を指定できます。
    - PACKAGE の代わりに PROGRAM を指定できます。
- ユーザー定義関数を使用中に、そのユーザー定義関数をドロップすることは有効です。また、ユーザー定義関数への参照を含むステートメントでカーソルがオープンされているようにすることができます。そのカーソルがオープンされている間に、カーソルのフェッチがエラーになることなくその関数をドロップすることができます。
- ユーザー定義関数に従属しているパッケージが実行されている場合、そのパッケージが現行の作業単位を完了するまで、別の許可 ID からその関数をドロップすることはできません。その時点で、関数はドロップされ、パッケージは作動不能になります。このパッケージの次の要求はエラーになり、パッケージの明示再バインドが必要であることが示されます。
- 関数本体を必要とするアプリケーションが実行されている時に、関数本体が除去される場合があります (これは関数のドロップとは異なります)。ステートメントの代わりにデータベース・マネージャーが関数本体をストレージにロードする必要があるかどうかに応じて、ステートメントはエラーになる場合もあれば、エラーにならない場合もあります。
- ドロップされた表の中に DATALINK 列によってリンクされているファイルが含まれている場合には、それらのファイルはリンク解除されてから、データ・リンク列の定義に応じてリストアされたり削除されたりします。
- データベースに対して構成された DB2 Data Links Manager を使用できないときに、 DROP TABLE または DROP TABLESPACE を使って DATALINK 列を含んだ表がドロップされると、操作は失敗します (SQLSTATE 57050)。

- 明示的に指定された UDF に記録されている従属関係に加えて、トランスフォームが暗黙的に必要な場合には以下の従属関係が記録されます。
  1. 構造タイプのパラメーターや関数またはメソッドの結果にトランスフォームが必要な場合は、その関数またはメソッドに、 TO SQL か FROM SQL の必要なトランスフォーム関数に対する従属関係が記録されます。
  2. パッケージに含まれている SQL ステートメントでトランスフォーム関数が必要になる場合は、そのパッケージに、TO SQL か FROM SQL の指定されたトランスフォーム関数に対する従属関係が記録されます。

  上記の部分では、トランスフォームを暗黙的に呼び出すことによって従属関係が記録される場合のみを扱っているため、関数、メソッド、あるいはパッケージ以外のオブジェクトが、暗黙的に呼び出されたトランスフォーム関数に従属することはありません。一方、トランスフォーム関数を明示的に呼び出した場合 (たとえば、ビューやトリガーなどで) は、これらの他のタイプのオブジェクトが通常どおりトランスフォーム関数に従属します。したがって、トランスフォームに対するこれらの「明示的な」タイプの従属がドロップされることによって、 DROP TRANSFORM が失敗する場合もあります (SQLSTATE 42893)。
- 従属関係カタログでは、暗黙的なトランスフォームによる関数への従属と明示的に関数を呼び出すことによって生じる従属とを区別していません。したがって、トランスフォーム関数に対する明示的な呼び出しは書かないよう勧められています。このようなインスタンスでは、単に SQL の式に明示的な呼び出しが含まれているという理由で、関数上のトランスフォーム・プロパティーがドロップされなかったり、パッケージが作動不能としてマークされてしまいます。
- ID 列のシーケンスを作成したシステムを、DROP SEQUENCE ステートメントでドロップすることはできません。
- シーケンスがドロップされると、シーケンスに関する特権もすべてドロップされ、そのシーケンスを参照するパッケージはすべて無効になります。
- リレーショナル・ニックネームの場合、所定の作業単位 (UOW) 内の DROP NICKNAME ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - このステートメントで参照されているニックネームには、同じ UOW 内でオープンされているカーソルがある。
  - このステートメントで参照されているニックネームに対して、同じ UOW 内ですでに INSERT、DELETE、または UPDATE ステートメントのいずれかが出されている。
- 非リレーショナル・ニックネームの場合、所定の作業単位 (UOW) 内の DROP NICKNAME ステートメントは、以下のいずれかの条件の下では処理できません (SQLSTATE 55007)。
  - このステートメントで参照されているニックネームには、同じ UOW 内でオープンされているカーソルがある。
  - このステートメントで参照されているニックネームは、同じ UOW 内の SELECT ステートメントですでに参照されている。
  - このステートメントで参照されているニックネームに対して、同じ UOW 内ですでに INSERT、DELETE、または UPDATE ステートメントのいずれかが出されている。

- 所定の作業単位 (UOW) 内の DROP SERVER ステートメント (SQLSTATE 55006)、または DROP FUNCTION MAPPING あるいは DROP TYPE MAPPING ステートメント (SQLSTATE 55007) は、以下のいずれかの条件の下では処理できません。
  - ステートメントが 1 つのデータ・ソースを参照していて、次のいずれかがすでに UOW に含まれている。
    - このデータ・ソース内の表またはビューのニックネームを参照する SELECT ステートメント。
    - このデータ・ソース内の表またはビューのニックネーム上のオープン・カーソル。
    - このデータ・ソース内の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。
  - ステートメントがデータ・ソースのカテゴリー (例えば、特定のタイプおよびバージョンのすべてのデータ・ソースなど) を参照しており、次のいずれかがすでに UOW に含まれている。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームを参照する SELECT ステートメント。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネーム上のオープン・カーソル。
    - それらのデータ・ソースのいずれかの中の表またはビューのニックネームに対して発行された INSERT、DELETE、または UPDATE ステートメント。

例:

例 1: 表 TDEPT をドロップします。

```
DROP TABLE TDEPT
```

例 2: ビュー VDEPT をドロップします。

```
DROP VIEW VDEPT
```

例 3: 許可 ID HEDGES が別名をドロップします。

```
DROP ALIAS A1
```

別名 HEDGES.A1 がカタログから除去されます。

例 4: Hedges は別名のドロップを試みますが、既存の表の名前である (別名でない) T1 を別名として指定しています。

```
DROP ALIAS T1
```

このステートメントはエラーになります (SQLSTATE 42809)。

例 5:

BUSINESS_OPS データベース・パーティション・グループをドロップします。このデータベース・パーティション・グループをドロップするには、まずデータベース・パーティション・グループ内の表スペース (ACCOUNTING と PLANS) をドロップする必要があります。

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP DATABASE PARTITION GROUP BUSINESS_OPS
```

例 6: Pellow は CENTRE 関数をドロップします。この関数は、ドロップする関数インスタンスであることを示すためにシグニチャーを使用して、 PELLOW スキーマに作成したものです。

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

例 7: McBride は FOCUS92 関数をドロップします。この関数は、ドロップする関数インスタンスであることを示すために特定名を使用して、 PELLOW スキーマに作成したものです。

```
DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

例 8: CHEM スキーマから関数 ATOMIC_WEIGHT をドロップします。このスキーマには、この名前の関数は 1 つしかないことが分かっています。

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

例 9: トリガー SALARY_BONUS をドロップします。このトリガーにより、従業員は指定の条件で給与に加えてボーナスを受け取ります。

```
DROP TRIGGER SALARY_BONUS
```

例 10: 現在使用していない SHOESIZE という名前の特殊データ・タイプをドロップします。

```
DROP DISTINCT TYPE SHOESIZE
```

例 11: SMITHPAY イベント・モニターをドロップします。

```
DROP EVENT MONITOR SMITHPAY
```

例 12: CREATE SCHEMA の例 2 で RESTRICT を使用して作成したスキーマをドロップします。 PART という名前の表をまずドロップする必要があることに注意してください。

```
  DROP TABLE PART
DROP SCHEMA INVENTRY RESTRICT
```

例 13: Macdonald は DESTROY プロシージャーをドロップします。このプロシージャーは、ドロップするプロシージャー・インスタンスであることを示すために特定名を使用して、 EIGLER スキーマに作成したものです。

```
DROP SPECIFIC PROCEDURE  EIGLER.DESTROY
```

例 14: BIOLOGY スキーマからプロシージャー OSMOSIS をドロップします。このスキーマには、この名前のプロシージャーは 1 つしかないことが分かっています。

```
DROP PROCEDURE BIOLOGY.OSMOSIS
```

例 15: ユーザー SHAWN は、フェデレーテッド・データベースにアクセスするときと、 ORACLE1 という Oracle データ・ソースのデータベースにアクセスするときでは、異なる許可 ID を使用しました。 2 つの許可でマッピングが作成されましたが、 SHAWN がそのデータ・ソースにアクセスする必要はなくなりました。マッピングをドロップします。

```
DROP USER MAPPING FOR  SHAWN SERVER ORACLE1
```

例 16: ニックネームが参照するデータ・ソース表の索引が削除されました。 オプティマイザーにこの索引を認識させるために作成した索引指定をドロップします。

```
DROP INDEX INDEXSPEC
```

例 17: トランスフォーム・グループ MYSTRUCT1 をドロップします。

```
DROP TRANSFORM MYSTRUCT1 FOR POLYGON
```

例 18: PERSONNEL スキーマで EMP データ・タイプからメソッド BONUS をドロップします。

```
DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

例 19: 制限を使用して ORG_SEQ からシーケンスをドロップします。

```
DROP SEQUENCE ORG_SEQ
```

例 20: リモート表 EMPLOYEE が、フェデレーテッド・システムに透過性 DDL を使用して作成されました。この表へのアクセスは、もう必要ありません。リモート表 EMPLOYEE をドロップします。

```
DROP TABLE EMPLOYEE
```

例 21: 関数マッピング BONUS_CALC をドロップし、デフォルトの関数マッピングがあればそれを復元します。

```
    DROP FUNCTION MAPPING BONUS_CALC
```

関連タスク:

- フェデレーテッド・システム・ガイド の『デフォルトの関数マッピングを使用不可にする』
- フェデレーテッド・システム・ガイド の『ユーザー定義関数マッピングのドロップ』
- フェデレーテッド・システム・ガイド の『透過 DDL を使用したリモート表のドロップ』

関連資料:

- 428 ページの『CREATE TRIGGER』
- 474 ページの『CREATE VIEW』
- 272 ページの『CREATE FUNCTION MAPPING』

関連サンプル:

- 『dbstat.sqb -- Reorganize table and run statistics (MF COBOL)』
- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『tbcreate.sqC -- How to create and drop tables (C++)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)』
- 『TbConstr.java -- How to create, use and drop constraints (JDBC)』
- 『TbCreate.java -- How to create and drop tables (JDBC)』

- 『TbTemp.java -- How to use Declared Temporary Table (JDBC)』
- 『TbTrig.java -- How to use triggers (JDBC)』
- 『UDFDrop.db2 -- How to uncatalog the Java UDFs contained in UDFsrv.java 』
- 『spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『tbcreate.sqc -- How to create and drop tables (C)』
- 『tbtemp.sqc -- How to use a declared temporary table (C)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『TbConstr.sqlj -- How to create, use and drop constraints (SQLj)』
- 『TbCreate.sqlj -- How to create and drop tables (SQLj)』
- 『TbTrig.sqlj -- How to use triggers (SQLj)』

END DECLARE SECTION

END DECLARE SECTION ステートメントは、ホスト変数宣言セクションの終わりを示します。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、実行可能ステートメントではありません。また、REXX に指定することはできません。

許可:

必要ありません。

構文:

説明:

END DECLARE SECTION ステートメントは、ホスト言語の規則に従って宣言を指定できる個所であれば、アプリケーション・プログラムのどこにでもコーディングすることができます。これは、ホスト変数の宣言セクションの終了を示します。 ホスト変数セクションは、 BEGIN DECLARE SECTION ステートメントで開始されます。

BEGIN DECLARE SECTION と END DECLARE SECTION ステートメントは、対にして使用する必要があり、ネストすることはできません。

ホスト変数の宣言は、SQL INCLUDE ステートメントを使用して指定することができます。それ以外の場合、ホスト変数の宣言セクションに、ホスト変数の宣言以外のステートメントを含めることはできません。

REXX 以外のホスト言語では、 SQL ステートメントで参照されるホスト変数をホスト変数宣言セクションで宣言しなければなりません。また、各変数の宣言は、その変数を最初に参照する個所よりも前にある必要があります。

宣言セクションの外部で宣言される変数の名前を、宣言セクションで宣言されている変数と同じ名前にすることはできません。

関連資料:

- 104 ページの『BEGIN DECLARE SECTION』

関連サンプル:

- 『advsql.sqb -- How to read table data using CASE (MF COBOL)』
- 『prepbind.sqb -- Precompile and bind an embedded SQL program to a database (MF COBOL)』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『spclient.sqc -- Call various stored procedures (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』

END DECLARE SECTION

- 『spclient.sqC -- Call various stored procedures (C++)』
- 『tut_read.sqC -- How to read tables (C++)』

EXECUTE

EXECUTE ステートメントは、準備済み SQL ステートメントを実行します。

呼び出し:

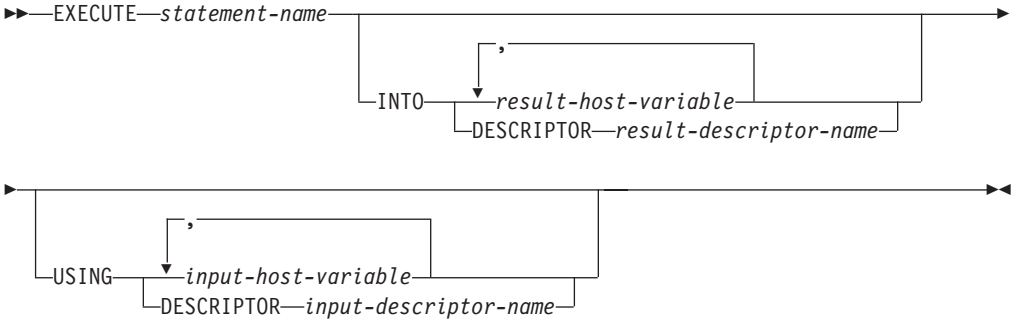
このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

ステートメントの実行時に許可検査が行われるステートメント (DDL、GRANT、および REVOKE ステートメント) の場合、このステートメントの許可 ID の特権には、 PREPARE ステートメントで指定されている SQL ステートメントを実行するための特権が含まれていなければなりません。ステートメントの許可 ID は、BIND オプション DYNAMICRULES の影響を受けることがあります。

許可検査がステートメントの準備の時点で行われるステートメント (DML) の場合、このステートメントを使用するために必要な権限はありません。

構文:



説明:

statement-name

実行する準備済みのステートメントを指定します。 *statement-name* (ステートメント名) はすでに準備済みのステートメントを指定していなければならず、またそのステートメントが SELECT ステートメントであってはなりません。

INTO

この後に、準備済みステートメントの出力パラメーター・マーカー (?) から値を受け取るために使用される、ホスト変数のリストを指定します。

動的 CALL ステートメントの場合は、ストアード・プロシージャーに対する OUT および INOUT 引き数に使用されるパラメーター・マーカーは、出力パラメーター・マーカーです。ステートメントに出力パラメーター・マーカーを使用する場合は、 INTO 文節を指定する必要があります (SQLSTATE 07007)。

result-host-variable, ...

ホスト変数の宣言規則に従って、該当プログラムで宣言されているホスト変数を指定します。変数の数は、準備されるステートメントの出力パラメーター・マーカーの数と同じでなければなりません。 *n* 番目の変数は、準備済

みステートメントの *n* 番目のパラメーター・マーカーに対応します。場合によっては、ロケーター変数とファイル参照変数も、パラメーター・マーカーの宛先として指定できます。

DESCRIPTOR *result-descriptor-name*
出力 SQLDA を指定します。その内容は、ホスト変数についての有効な記述でなければなりません。

EXECUTE ステートメントが処理される前に、ユーザーは、入力 SQLDA の以下のフィールドを設定する必要があります。

- SQLDA に用意する SQLVAR のエレメント数を示す SQLN
- SQLDA に割り振るストレージのバイト数を示す SQLDABC
- ステートメントの処理時にその SQLDA の使用される変数の数を示す SQLD
- 変数の属性を示す SQLVAR のオカレンス

SQLDA には、すべての SQLVAR オカレンスが入るだけの十分なストレージがなければなりません。

LOB または構造化データ・タイプの出力データを入れる必要がある場合には、各パラメーター・マーカーごとに 2 つの SQLVAR 項目が必要になります。

SQLD に設定する値は、ゼロ以上で SQLN 以下でなければなりません。

USING
この後に、準備済みステートメントの入力パラメーター・マーカー (?) に置き換わる値を含むホスト変数のリストを指定します。

動的 CALL ステートメントの場合、ストアード・プロシージャーに対する IN および INOUT 引き数に使用されるパラメーター・マーカーは、入力パラメーター・マーカーです。その他のすべての動的ステートメントの場合、すべてのパラメーター・マーカーは入力パラメーター・マーカーです。ステートメントに出力パラメーター・マーカーを使用する場合は、 USING 文節を指定する必要があります (SQLSTATE 07004)。

input-host-variable, ...
ホスト変数の宣言規則に従って、該当プログラムで宣言されているホスト変数を指定します。変数の数は、準備されるステートメントの入力パラメーター・マーカーの数と同じでなければなりません。 *n* 番目の変数は、準備済みステートメントの *n* 番目のパラメーター・マーカーに対応します。 場合によっては、ロケーター変数とファイル参照変数も、パラメーター・マーカーの値のソースとして指定できます。

DESCRIPTOR *input-descriptor-name*
入力 SQLDA を指定します。その内容は、ホスト変数についての有効な記述でなければなりません。

EXECUTE ステートメントが処理される前に、ユーザーは、入力 SQLDA の以下のフィールドを設定する必要があります。

- SQLDA に用意する SQLVAR のエレメント数を示す SQLN
- SQLDA に割り振るストレージのバイト数を示す SQLDABC

- ステートメントの処理時にその SQLDA の使用される変数の数を示す SQLD
- 変数の属性を示す SQLVAR のオカレンス

SQLDA には、すべての SQLVAR オカレンスが入るだけの十分なストレージがなければなりません。したがって、SQLDABC の値は 16 + SQLN*(N) 以上でなければなりません (N は 1 つの SQLVAR オカレンスの長さ)。

LOB または構造化データ・タイプの入力データを入れる必要がある場合には、各パラメーター・マーカーごとに 2 つの SQLVAR 項目が必要になります。

SQLD に設定する値は、ゼロ以上で SQLN 以下でなければなりません。

注:

- 準備済みステートメントを実行する前に、各入力パラメーター・マーカーはそれに対応するホスト変数の値によって置き換えられます。型付きパラメーター・マーカーの場合、ターゲット変数の属性は CAST 指定によって指定されます。 タイプなしパラメーター・マーカーの場合、ターゲット変数の属性はパラメーター・マーカーのコンテキストに従って決定されます。

  V は、パラメーター・マーカー P に対応するホスト変数を表します。 V の値は、列への値の割り振り規則に従って、P のターゲット変数に割り当てられます。したがって、
  - V はターゲットと互換でなければなりません。
  - V がストリングの場合、その長さはターゲットの長さ属性を超えることはできません。
  - V が数値の場合、 V の整数部分の絶対値はターゲットの整数部分の絶対値の最大を超えることはできません。
  - V の属性がターゲットの属性と同一でない場合、その値はターゲットの属性に合うように変換されます。

  準備済みステートメントを実行すると、P の代わりに使用される値は P のターゲット変数になります。たとえば、V が CHAR(6) でターゲットが CHAR(8) の場合、 P の代わりに使用される値は V の値にブランクを 2 個付加したものになります。
- 動的 CALL ステートメントの場合は、準備済みステートメントの実行後は、OUT および INOUT の各引き数の戻り値は、引き数に使用された出力パラメーター・マーカーに対応するホスト変数に割り当てられます。型付きパラメーター・マーカーの場合、ターゲット変数の属性は CAST 指定によって指定されます。タイプなしパラメーター・マーカーの場合、ターゲット変数の属性は、ストアード・プロシージャーのパラメーターの定義によって指定されます。

  V は、パラメーター・マーカー P に対応する出力ホスト変数を表し、ストアード・プロシージャーの引き数 A に使用されます。 A の値は、列から値を検索するための規則に従って V に割り当てられます。したがって、
  - V は A と互換でなければなりません。

- V がストリングの場合、その長さは A の長さより短いものであってはなりません。そうでないと A の値は切り捨てられます。
- V が数値の場合、 V の整数部分の最大絶対値は A の整数部分の絶対値より小さいものであってはなりません。
- V の属性が A の属性と同一でない場合、A の値は V の属性に合うように変換されます。

- ***動的 SQL ステートメント・キャッシング:*** 動的および静的 SQL ステートメントの実行に必要な情報は、静的 SQL ステートメントが最初に参照された時点、または動的 SQL ステートメントが最初に準備された時点で、データベース・パッケージ・キャッシュに入れられます。 この情報は、無効になるか、キャッシュ・スペースが他のステートメントで必要になるか、またはデータベースがシャットダウンされるまでは、パッケージ・キャッシュに存続します。

  SQL ステートメントが実行または準備される場合に、要求を出したアプリケーションに関連するパッケージ情報が、システム・カタログからパッケージ・キャッシュにロードされます。 個々の SQL ステートメントの実際の実行可能セクションもキャッシュに入れられます。静的 SQL セクションは、該当のステートメントが最初に参照された時点で、システム・カタログから読み取られてパッケージ・キャッシュに入れられ、動的 SQL セクションは作成後にキャッシュに直接入れられます。動的 SQL セクションは、 PREPARE や EXECUTE IMMEDIATE などの明示的なステートメントによって作成されます。一度作成された動的 SQL ステートメントのセクションが、スペース管理のために削除された場合や、環境の変化によって無効になった場合に、システムによるステートメントの暗黙的な準備によって、再作成されることがあります。

  各 SQL ステートメントは、データベース・レベルでキャッシュされ、アプリケーション間で共有できます。静的 SQL ステートメントは、同じパッケージを使用してアプリケーション間で共有されます。動的 SQL ステートメントは、同じコンパイル環境と、厳密に同じステートメント・テキストを使用してアプリケーション間で共有されます。アプリケーションによって発行される各 SQL ステートメントのテキストは、アプリケーションにローカルにキャッシュされ、暗黙的な準備が必要な場合に使用されます。アプリケーション・プログラム中の各 PREPARE ステートメントは、1 つのステートメントをキャッシュできます。アプリケーション・プログラム中のすべての EXECUTE IMMEDIATE ステートメントは、同じスペースを共用し、これらの EXECUTE IMMEDIATE ステートメントに対しては、キャッシュされるステートメントは同時に 1 つしか存在しません。 それぞれ異なる SQL ステートメントに対して、同じ PREPARE またはいずれかの EXECUTE IMMEDIATE ステートメントが何度も発行される場合は、最後のステートメントだけがキャッシュに入れられ、再使用の対象になります。キャッシュの使用を最適化するには、アプリケーションの開始時に多くの異なる PREPARE ステートメントを一度に発行し、その後必要に応じて EXECUTE または OPEN ステートメントを発行することです。

  動的 SQL ステートメントのキャッシングを使用すると、ステートメントを一度作成すれば、ステートメントを再度準備しなくても複数の作業単位にわたってステートメントを再使用できます。 環境が変わった場合には、必要に応じてシステムはステートメントを再コンパイルします。

以下のイベントは、次の PREPARE、EXECUTE、EXECUTE IMMEDIATE、または OPEN の要求時に、キャッシュされた動的ステートメントが暗黙的に準備される原因となる環境またはデータ・オブジェクトの変更の例です。

- ALTER FUNCTION
- ALTER METHOD
- ALTER NICKNAME
- ALTER PROCEDURE
- ALTER SERVER
- ALTER TABLE
- ALTER TABLESPACE
- ALTER TYPE
- CREATE FUNCTION
- CREATE FUNCTION MAPPING
- CREATE INDEX
- CREATE METHOD
- CREATE PROCEDURE
- CREATE TABLE
- CREATE TEMPORARY TABLESPACE
- CREATE TRIGGER
- CREATE TYPE
- DROP (すべてのオブジェクト)
- 表または索引の RUNSTATS
- ビューが作動不能になる原因となるすべてのアクション
- システム・カタログ表の統計の UPDATE
- SET CURRENT DEGREE
- SET PATH
- SET QUERY OPTIMIZATION
- SET SCHEMA
- SET SERVER OPTION

キャッシュに入れられる動的 SQL ステートメントから予想される動作の概略は、以下のようになります。

- *PREPARE 要求:* 以後同じステートメントの準備に、セクションが有効であればステートメントのコンパイルのコストがかかりません。現在キャッシュに入れられているセクションのコストとカーディナリティーの見積もりが戻されます。それらの値は、同じ SQL ステートメントに対するそれより前の PREPARE から戻される値とは違っている場合があります。 COMMIT または ROLLBACK ステートメントの後に PREPARE ステートメントを発行する必要はありません。
- *EXECUTE 要求:* 元の PREPARE 以後にステートメントが無効になった場合に、ステートメントを暗黙的に準備するコストが EXECUTE ステートメント

にかかることがあります。セクションが暗黙的に準備される場合、当初の PREPARE ステートメントの環境でなく、現行の環境が使用されます。

- *EXECUTE IMMEDIATE 要求:* 以後同じステートメントに対して EXECUTE IMMEDIATE ステートメントを出す際に、セクションが有効であればステートメントのコンパイルのコストがかかりません。
- *OPEN 要求:* 当初の PREPARE ステートメント以後にステートメントが無効になった場合、ステートメントを暗黙的に準備するコストが動的に定義されたカーソルに対する OPEN 要求にかかることがあります。セクションが暗黙的に準備される場合、当初の PREPARE ステートメントの環境でなく、現行の環境が使用されます。
- *FETCH 要求:* 予想される動作の変化はありません。
- *ROLLBACK:* ロールバック操作の影響を受ける作業単位で準備されたか暗黙的に準備された動的 SQL ステートメントだけが無効になります。
- *COMMIT:* 動的 SQL ステートメントは無効になりませんが、確立されたロックは解放されます。 WITH HOLD カーソルとして定義されていないカーソルはクローズされ、そのロックは解放されます。オープンされている WITH HOLD カーソルは、そのパッケージとセクション・ロックを保持し、コミット処理中およびコミット処理後にアクティブなセクションを保護します。

暗黙の準備の過程でエラーが生じると、その暗黙の準備の原因となった要求にエラーが戻されます (SQLSTATE 56098)。

例:

例 1: この C の例では、パラメーター・マーカーを伴う INSERT ステートメントが準備され、実行されます。ホスト変数 h1 - h4 は、TDEPT の形式に対応します。

```
strcpy (s,"INSERT INTO TDEPT VALUES(?,?,?,?)");
EXEC SQL PREPARE DEPT_INSERT FROM :s;
  .
  .
(Check for successful execution and put values into :h1, :h2, :h3, :h4)
  .
  .
EXEC SQL EXECUTE DEPT_INSERT USING :h1, :h2,
:h3, :h4;
```

例 2: この EXECUTE ステートメントは SQLDA を使用します。

```
EXECUTE S3 USING DESCRIPTOR :sqlda3
```

例 3: 従業員に賞与を与えるための以下のストアード・プロシージャーを考慮します。

```
CREATE PROCEDURE GIVE_BONUS (IN EMPNO INTEGER,
                             IN DEPTNO INTEGER,
                             OUT CHEQUE INTEGER,
                             INOUT BONUS DEC(6,0))
...
```

ストアード・プロシージャーを C アプリケーションから動的に呼び出します。ストアード・プロシージャーは、以下のホスト変数を入力として取ります。

- *employee*。従業員の ID 番号。
- *dept*。部門番号。

- *bonus*。従業員の賞与。

ストアード・プロシージャーは、以下の値をホスト変数に戻します。

- *cheque_no*。小切手の ID 番号。
- *bonus*。実際の賞与額 (調整後の)

```
strcpy (s, "CALL GIVE_BONUS(?, ?, ?, ?)");
EXEC SQL PREPARE DO_BONUS FROM :s;
  .
  .
/* Check for successful execution and put values into
   :employee, :dept, and :bonus */
  .
  .
EXEC SQL EXECUTE DO_BONUS INTO :cheque_no, :bonus
                         USING :employee, :dept, :bonus;
  .
  .
/* Check for successful execution and process the
   values returned in :cheque_no and :bonus */
```

関連資料:

- *SQL リファレンス 第 1 巻* の『ID』
- 647 ページの『PREPARE』
- *SQL リファレンス 第 1 巻* の『SQLDA (SQL 記述子域)』

関連サンプル:

- 『dbuse.sqc -- How to use a database (C)』
- 『fnuse.sqc -- How to use built-in SQL functions (C)』
- 『tut_use.sqc -- How to modify a database (C)』
- 『udfcli.sqc -- Call a variety of types of user-defined functions (C)』
- 『dbuse.sqC -- How to use a database (C++)』
- 『fnuse.sqC -- How to use built-in SQL functions (C++)』
- 『tut_use.sqC -- How to modify a database (C++)』
- 『udfcli.sqC -- Call a variety of types of user-defined functions (C++)』
- 『inpsrv.sqb -- A stored procedure using the GENERAL parameter style (MF COBOL)』

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE ステートメントは、以下のことを行います。

- 文字ストリング形式の SQL ステートメントから、実行可能形式の SQL ステートメントを準備します。
- その SQL ステートメントを実行します。

EXECUTE IMMEDIATE の機能は、 PREPARE ステートメントと EXECUTE ステートメントの基本的な機能の組み合わせです。このステートメントは、ホスト変数もパラメーター・マーカーも含まれていない SQL ステートメントを準備し実行する場合に使用することができます。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

指定された SQL ステートメントに定義されているのと同じ許可規則が適用されます。

ステートメントの許可 ID は、DYNAMICRULES BIND オプションの影響を受けることがあります。

構文:

説明:

host-variable

ホスト変数の指定は必須であり、文字ストリング変数の宣言規則に従ってプログラムに記述されたホスト変数を指定していなければなりません。これは、最大のステートメント・サイズの 65 535 より小さい文字ストリング変数でなければなりません。 CLOB(65535) には最大のステートメント・サイズを含めることができますが、 VARCHAR には含めることができませんので注意してください。指定するホスト変数の値は、ステートメント・ストリングと呼ばれます。

ステートメント・ストリングは、以下のいずれかの SQL ステートメントでなければなりません。

- ALTER
- CALL
- COMMENT
- COMMIT
- CREATE
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- GRANT

- INSERT
- LOCK TABLE
- REFRESH TABLE
- RELEASE SAVEPOINT
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- ROLLBACK
- SAVEPOINT
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET INTEGRITY
- SET PASSTHRU
- SET PATH
- SET SCHEMA
- SET SERVER OPTION
- UPDATE

ステートメント・ストリングには、パラメーター・マーカーやホスト変数への参照を含めてはなりません。また EXEC SQL で始まっていてはなりません。ステートメント終止符を含めることはできません。ただし、CREATE TRIGGER および CREATE PROCEDURE ステートメントは例外です。 CREATE TRIGGER ステートメントには、トリガーによって実行される SQL ステートメントを区切るために、セミコロン (;) を含めることができます。 CREATE PROCEDURE ステートメントには、 SQL プロシージャーの本体で SQL ステートメントを区切るためにセミコロンを含めることができます。 CALL ステートメントで指定されるストアード・プロシージャーには、 OUT または INOUT パラメーターがあってはなりません (SQLSTATE 07007)。

EXECUTE IMMEDIATE ステートメントを実行すると、指定したステートメント・ストリングの構文解析が行われ、エラーの有無が検査されます。その SQL ステートメントが無効である場合それは実行されず、実行を妨げているエラー条件が SQLCA に報告されます。 SQL ステートメントが有効で、その実行の過程でエラーが発生した場合、エラー条件が SQLCA に報告されます。

注:

- ステートメントのキャッシュは、EXECUTE IMMEDIATE ステートメントの動作に影響を与えます。

例:

C プログラム・ステートメントを使用して SQL ステートメントをホスト変数 qstring (char[80]) に入れ、そのホスト変数 qstring に入れられた SQL ステートメントを作成および実行します。

```
if ( strcmp(accounts,"BIG") == 0 )
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO < 100");
else
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO >= 100");
  .
  .
  .
EXEC SQL  EXECUTE IMMEDIATE :qstring;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『ID』
- 551 ページの『EXECUTE』

関連サンプル:

- 『dbuse.sqc -- How to use a database (C)』
- 『dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)』
- 『fnuse.sqc -- How to use built-in SQL functions (C)』
- 『tbconstr.sqc -- How to create, use, and drop constraints (C)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『tscreate.sqc -- How to create and drop buffer pools and table spaces (C)』
- 『dbuse.sqC -- How to use a database (C++)』
- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』
- 『dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)』
- 『fnuse.sqC -- How to use built-in SQL functions (C++)』
- 『tbconstr.sqC -- How to create, use, and drop constraints (C++)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)』
- 『DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)』
- 『expsamp.sqb -- Export and import tables with table data to a DRDA database (MF COBOL)』

EXPLAIN

EXPLAIN ステートメントは、指定された EXPLAIN 可能ステートメントに関して選択されたアクセス・プランについての情報をキャプチャーするとともに、この情報を Explain 表に入れます。

EXPLAIN 可能ステートメント とは、 DELETE、INSERT、SELECT、SELECT INTO、 UPDATE、 VALUES、および VALUES INTO SQL ステートメントのことです。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

Explain 情報を取り込むステートメントは実行されません。

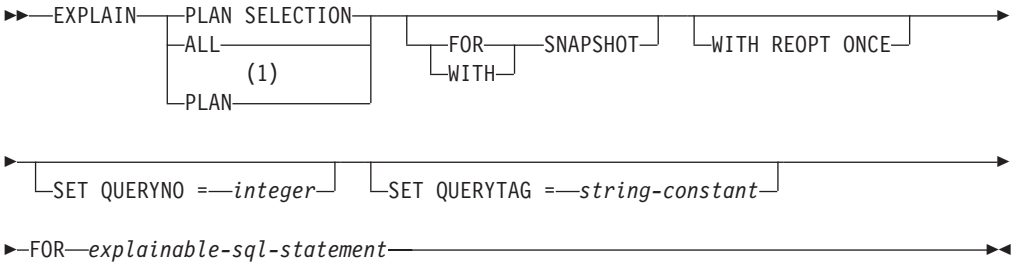
許可:

EXPLAIN ステートメントに指定された SQL ステートメントに定義されているのと同じ許可規則が適用されます。 たとえば、*explainable-sql-statement* (次ページのステートメント構文を参照) として DELETE ステートメントが使用された場合、DELETE ステートメントに定義されている許可規則が、DELETE ステートメントの Explain 情報を取り出す場合にも適用されます。

静的 EXPLAIN ステートメントの場合の許可規則は、 *explainable-sql-statement* として渡されるステートメントの静的バージョンに適用される規則と同じです。 動的に準備された EXPLAIN ステートメントに関しては、 *explainable-sql-statement* パラメーターに指定されたステートメントの動的準備の際に適用された許可規則が使用されます。

現行の許可 ID には、Explain 表に対する挿入特権が必要になります。

構文:



注:

1 PLAN オプションは、 DB2 for MVS の既存の EXPLAIN ステートメントの構文を許容する目的でのみサポートされます。 PLAN 表は存在しません。 PLAN を指定することは、PLAN SELECTION を指定するのと同等です。

説明:

PLAN SELECTION
SQL コンパイルのプラン選択フェーズからの情報を Explain 表に挿入することを示します。

ALL
ALL を指定することは、PLAN SELECTION を指定するのと同等です。

PLAN
PLAN オプションの指定によって、他のシステムからの既存のデータベース・アプリケーションの構文の差異が許容されます。 PLAN を指定することは、PLAN SELECTION を指定するのと同等です。

FOR SNAPSHOT
この文節は、Explain スナップショットだけを取り、それを EXPLAIN_STATEMENT 表の SNAPSHOT 列に入れることを示します。 EXPLAIN_INSTANCE および EXPLAIN_STATEMENT 表に存在するもの以外の、Explain 情報はキャプチャーされません。

Explain スナップショット情報は、Visual Explain での使用を意図しています。

WITH SNAPSHOT
この文節は、通常の Explain 情報に加えて、Explain スナップショットも取ることを示します。

デフォルトでは、EXPLAIN ステートメントは通常の Explain 情報だけを収集し、 Explain スナップショットは取りません。

Explain スナップショット情報は、Visual Explain での使用を意図しています。

デフォルト (FOR SNAPSHOT も WITH SNAPSHOT も指定しない場合)
Explain 情報を Explain 表に入れます。 Visual Explain での使用を目的としたスナップショットは取られません。

WITH REOPT ONCE
REOPT ONCE を使ってこのステートメントを再最適化するために以前に使用された、ホスト変数、パラメーター・マーカー、または特殊レジスターの値を使用して、指定した EXPLAIN 可能 SQL ステートメントを再最適化することを、この文節は示しています。 Explain 表には、新しいアクセス・プランが取り込まれます。ユーザーが DBADM 権限を持っているか、データベース・レジストリー変数 DB2_VIEW_REOPT_VALUES が YES に設定されている場合は、EXPLAIN_PREDICATE 表にも値が取り込まれます (それらの値を使ってステートメントを再最適化する場合)。

SET QUERYNO = *integer*
integer (整数) を、 EXPLAIN_STATEMENT 表の QUERYNO 列を介して *explainable-sql-statement* に関連付けます。 指定する整数値は、正の値でなければなりません。

動的 EXPLAIN ステートメントにこの文節を指定しなかった場合は、デフォルト値 (1) が割り当てられます。 静的 EXPLAIN ステートメントの場合には、プリコンパイラーによって割り当てられるステートメント番号がデフォルト値として割り当てられます。

SET QUERYTAG = *string-constant*
string-constant (ストリング定数) を、 EXPLAIN_STATEMENT 表の QUERYTAG 列を介して *explainable-sql-statement* に関連付けます。

string-constant には、長さ 20 バイトまでの任意の文字ストリングを指定できます。 指定された値が 20 バイトに満たない場合は必要な長さに達するまで右側がブランクで埋め込まれます。

EXPLAIN ステートメントにこの文節を指定しなかった場合はデフォルト値としてブランクが使用されます。

FOR *explainable-sql-statement*

Explain 情報を取り出す SQL ステートメントを指定します。 このステートメントとして、有効な DELETE、 INSERT、 SELECT、 SELECT INTO、 UPDATE、 VALUES、または VALUES INTO SQL ステートメントを指定できます。 EXPLAIN ステートメントがプログラムに組み込まれている場合には、 *explainable-sql-statement* にホスト変数に対する参照を含めることができます (ただし、これらのホスト変数がプログラム内で定義されている必要があります)。 同様に、EXPLAIN が動的に準備される場合には、 *explainable-sql-statement* にパラメーター・マーカーを含めることができます。

explainable-sql-statement には、 EXPLAIN ステートメントによってそれぞれ個別に準備および実行された有効な SQL ステートメントを指定する必要があります。ステートメント名やホスト変数を指定することはできません。 CLP を使用して定義されたカーソルを参照する SQL ステートメントを、このステートメントで使用することはできません。

アプリケーション内の動的 SQL に関する Explain 情報を取り出すためには、 EXPLAIN ステートメント全体を動的に準備する必要があります。

注:

次の表は、スナップショット・キーワードと Explain 情報の相互の関係を示しています。

指定したキーワード	Explain 情報をキャプチャーするか?	Visual Explain 用にスナップショットを取るか?
なし	Yes	No
FOR SNAPSHOT	No	Yes
WITH SNAPSHOT	Yes	Yes

FOR SNAPSHOT 文節と WITH SNAPSHOT 文節のどちらも指定しなかった場合は、 Explain スナップショットは取られません。

EXPLAIN ステートメントを呼び出す前に、Explain 表を作成しておく必要があります。このステートメントが生成した情報は、ステートメントをコンパイルした時点で指定されたスキーマにあるそれぞれの Explain 表に保管されます。

指定した *explainable-sql-statement* のコンパイル中に何らかのエラーが発生すると、Explain 表に情報は取り込まれません。

explainable-sql-statement について生成されたアクセス・プランは保管されません。したがって、後から呼び出すということはできません。 *explainable-sql-statement* についての Explain 情報が挿入されるのは、 EXPLAIN ステートメント自体のコンパイルが正常に完了した場合です。

静的 EXPLAIN SQL ステートメントの場合、情報はバインド時および明示的な再バインド時に Explain 表に挿入されます。 プリコンパイル中、静的 EXPLAIN ステートメントはコメント化され、修正済みのアプリケーション・ソース・ファイルに書き込まれます。バインド時に、EXPLAIN ステートメントは SYSCAT.STATEMENTS カタログに保管されます。 パッケージが実行される時は、EXPLAIN ステートメントは実行されません。 アプリケーション内にあるすべてのステートメントのセクション番号は連続した順序に並べられます。その中には EXPLAIN ステートメントも含まれることに注意してください。静的 EXPLAIN ステートメントを使用する代わりに、 EXPLAIN と EXPLSNAP BIND/PREP オプションを組み合わせることもできます。 静的 EXPLAIN ステートメントを使用することにより、数多くある SQL ステートメントの中からただ 1 つだけ静的 SQL ステートメントを選び出し、そのステートメントに関する情報を Explain 表に入れることもできます。そのことは、適切な EXPLAIN ステートメント構文を指定したターゲット・ステートメントに簡単な接頭部を付け、 Explain BIND/PREP オプションのどちらも使用せずにアプリケーションをバインドすることによって行えます。実際の Explain の呼び出し時に QUERYNO もしくは QUERYTAG フィールドを設定することが有利な場合にも、 EXPLAIN ステートメントを使用することができます。

追加バインド EXPLAIN ステートメントの場合、Explain 表に情報が入れられるのは、 EXPLAIN ステートメントのコンパイルがサブミットされるときです。パッケージが実行される時は、EXPLAIN ステートメントは実行されません (ただし、ステートメントは正常終了します)。 Explain 表にデータを取り込む際、Explain 表の修飾子と許可 ID には、パッケージ所有者の修飾子と許可 ID が使用されます。実際の Explain の呼び出し時に QUERYNO もしくは QUERYTAG フィールドを設定することが有利な場合にも、 EXPLAIN ステートメントを使用することができます。

動的 EXPLAIN ステートメントの場合、Explain 表に情報が入れられるのは、EXPLAIN ステートメントのコンパイルがサブミットされるときです。 PREPARE ステートメントを指定して Explain ステートメントを準備することもできますが、そのようにして実行しても処理は行われません (ステートメントは正常終了します)。動的 EXPLAIN ステートメントを発行する代わりに、 CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスターを組み合わせて使用することによっても、動的 SQL ステートメントの Explain 情報を取り出すことができます。実際の Explain の呼び出し時に QUERYNO もしくは QUERYTAG フィールドを設定することが有利な場合には、 EXPLAIN ステートメントを使用してください。

REOPT BIND オプションが ONCE に設定されていて、 CURRENT EXPLAIN MODE または CURRENT EXPLAIN SNAPSHOT 特殊レジスターのいずれかが REOPT に設定されている場合、ホスト変数、特殊レジスター、またはパラメーター・マーカーを含む静的または動的 SQL ステートメントを実行すると、ステートメントが再最適化されるときだけに、ステートメントの Explain 情報がキャプチャーされます。一方、REOPT BIND オプションが ALWAYS に設定されている場合は、それらのステートメントが実行されるたびに Explain 情報がキャプチャーされます。

例:

例 1: 単純な SELECT ステートメントの Explain 情報を取り出し、 QUERYNO = 13 のタグを付けます。

```
EXPLAIN PLAN SET QUERYNO = 13
FOR SELECT C1
FROM T1
```

例 2: 単純な SELECT ステートメントの Explain 情報を取り出し、 QUERYTAG = 'TEST13' のタグを付けます。

```
EXPLAIN PLAN SELECTION SET QUERYTAG = 'TEST13'
  FOR SELECT C1
  FROM T1
```

例 3: 単純な SELECT ステートメントの Explain 情報を取り出し、 QUERYNO = 13 および QUERYTAG = 'TEST13' のタグを付けます。

```
EXPLAIN PLAN SELECTION SET QUERYNO = 13 SET QUERYTAG = 'TEST13'
  FOR SELECT C1
  FROM T1
```

例 4: Explain 表が存在しない場合に、Explain 情報の入手を試みます。

```
EXPLAIN ALL FOR SELECT C1
FROM T1
```

このステートメントは失敗します。Explain 表が定義されていないからです (SQLSTATE 42704)。

例 5: 以下のステートメントがパッケージ・キャッシュ内に見出され、 REOPT ONCE を使ってすでにコンパイルされている場合に、ステートメントは成功します。

```
EXPLAIN ALL WITH REOPT ONCE FOR SELECT C1
FROM T1
WHERE C1 = :<host variable>
```

関連資料:

- *SQL リファレンス 第 1 巻* の『EXPLAIN_ARGUMENT 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_OBJECT 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_OPERATOR 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_PREDICATE 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_STREAM 表』
- *SQL リファレンス 第 1 巻* の『ADVISE_INDEX 表』
- *SQL リファレンス 第 1 巻* の『ADVISE_WORKLOAD 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_INSTANCE 表』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_STATEMENT 表』
- *SQL リファレンス 第 1 巻* の『Explain 表』

FETCH

FETCH ステートメントは、カーソルの位置を結果表の次の行に移し、その行の値をホスト変数に割り当てます。

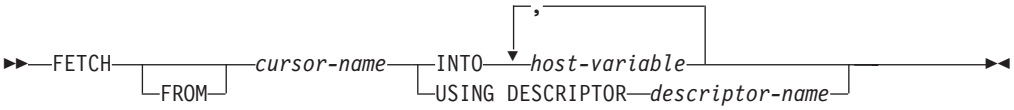
呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

カーソルを使用するために必要な権限については、『DECLARE CURSOR』 を参照してください。

構文:



説明:

cursor-name
フェッチ操作で使用するカーソルを指定します。 『DECLARE CURSOR』の項で説明されているように、 *cursor-name* は、宣言済みカーソルを指定しなければなりません。ソース・プログラムにおいて、 FETCH ステートメントより前に DECLARE CURSOR ステートメントがなければなりません。 FETCH ステートメントを実行する場合、該当のカーソルはオープン状態でなければなりません。

そのカーソルの位置が、現在その結果表の最終行またはそれ以降にある場合、

- SQLCODE は +100 に設定され、SQLSTATE は '02000' に設定されます。
- カーソルは最終行の後に位置づけられます。
- ホスト変数に値は割り当てられません。

ある行より前に現在カーソルが位置している場合、カーソルはその行に再配置され、 INTO または USING で指定されたホスト変数に値が割り当てられます。

最終行以外の行に現在カーソルが位置している場合、カーソルは次の行に再配置され、その行の値は INTO または USING で指定されたホスト変数に割り当てられます。

INTO *host-variable*, **...**
1 つまたは複数のホスト変数を指定します。そのホスト変数は、ホスト変数の宣言規則に従って記述されていなければなりません。 結果行の最初の値はリスト中の最初のホスト変数、その次の値は 2 番目のホスト変数、以下同様に割り当てられます。 選択リストの LOB 値は、正規のホスト変数 (十分な大きさの場合)、ロケーター変数、またはファイル参照変数に割り当てることができます。

USING DESCRIPTOR *descriptor-name*

ゼロ個以上のホスト変数の有効な記述を含む SQLDA を識別します。

FETCH ステートメントが処理される前に、ユーザーは次に示す SQLDA 内のフィールドを設定する必要があります。

- SQLDA に用意する SQLVAR のエレメント数を示す SQLN
- SQLDA に割り振るストレージのバイト数を示す SQLDABC
- ステートメントの処理時にその SQLDA の使用される変数の数を示す SQLD
- 変数の属性を示す SQLVAR のオカレンス

SQLDA には、すべての SQLVAR オカレンスが入るだけの十分なストレージがなければなりません。したがって、SQLDABC の値は 16 + SQLN*(N) 以上でなければなりません (N は 1 つの SQLVAR オカレンスの長さ)。

LOB または構造タイプの結果列を入れるには、各選択リスト項目 (または結果表の列) ごとに 2 つの SQLVAR 項目が必要です。

SQLD に設定する値は、ゼロ以上で SQLN 以下でなければなりません。

INTO 文節で指定されるか、または SQLDA に記述される *n* 番目の変数は、カーソルの結果表の *n* 番目の列に対応します。 各変数のデータ・タイプは、それに対応する列と互換性がなければなりません。

各変数には、特定の規則に従って値が割り当てられます。変数の数がその行の値の数よりも少ない場合、 SQLDA の SQLWARN3 フィールドが 'W' に設定されます。 変数の数が結果表の列の数よりも多い場合、警告は出されません。 割り当てエラーが発生すると、値は変数に割り当てられず、値はそれ以上変数に割り当てられません。それまでにすでに変数に割り当てられていた値はそのままになります。

注:

- オープン・カーソルの位置として、3 つの位置が考えられます。
  - 行の前
  - 行の上
  - 最終行のあと
- カーソル位置が行にある場合、その行はカーソルの現在行と呼ばれます。UPDATE ステートメントまたは DELETE ステートメントでカーソルを参照する場合、そのカーソル位置は行でなければなりません。カーソルの現在行となるのは、FETCH ステートメントの結果としての行だけです。
- 複数の FETCH を通じてロケーターを維持する必要がない場合、 LOB ロケーターへの取り出しを行う場合には、ロケーター・リソースの限度を考慮して、その次の FETCH ステートメントを発行する前に FREE LOCATOR ステートメントを発行しておくとよいでしょう。
- エラーが発生したことによって、カーソルの状態が予測できないものになることがあります。
- 警告が FETCH に戻されなかったり、以前取り出された行に対する警告が戻されたりする場合があります。これらの問題は、システム一時表やプッシュダウン演算子を使用するような最適化によって生じる場合があります。

- ステートメントのキャッシュは、EXECUTE IMMEDIATE ステートメントの動作に影響を与えます。
- DB2 CLI は追加の取り出し機能をサポートしています。たとえば、カーソルの結果表が読み取り専用の場合に、 SQLFetchScroll() 関数を使用してその結果表の中の任意のスポットにカーソルを位置づけることができます。
- 更新可能カーソルの場合は、行が取り出される時に行でロックが取得されます。
- カーソル定義に SQL データ変更ステートメントが含まれている場合や、 SQL データを変更するルーチンの呼び出しが関係している場合は、フェッチ操作中にエラーが発生し、エラーによってカーソルがクローズされることがあっても、変更された行がロールバックされることはありません。

例:

例 1: この C の例では、 FETCH ステートメントは SELECT ステートメントの結果を取り出して、プログラム変数 dnum、dname、および mnum に入れます。取り出す行がなくなった場合、見つからないことを示す状態が戻されます。

```
EXEC SQL  DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
  WHERE ADMRDEPT = 'A00';

EXEC SQL  OPEN C1;

while (SQLCODE==0) {
    EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;
}

EXEC SQL  CLOSE C1;
```

例 2: この FETCH ステートメントは SQLDA を使用しています。

```
FETCH CURS USING DESCRIPTOR :sqlda3
```

関連資料:

- 491 ページの『DECLARE CURSOR』
- 551 ページの『EXECUTE』
- *SQL リファレンス 第 1 巻* の『SQLDA (SQL 記述子域)』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

関連サンプル:

- 『cursor.sqb -- How to update table data with cursor statically (MF COBOL)』
- 『tbread.sqc -- How to read tables (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『tbread.sqC -- How to read tables (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『TutMod.sqlj -- Modify data in a table (SQLj)』

FLUSH EVENT MONITOR

FLUSH EVENT MONITOR ステートメントは、イベント・モニター *event-monitor-name* に関連付けられたすべてのアクティブ・モニター・タイプの現行のデータベース・モニター値を、イベント・モニターの I/O ターゲットに書き込みます。このため、レコードの生成頻度が低いイベント・モニター (データベース・イベント・モニターなど) で、いつでも部分イベント・レコードを使用することができます。 こうしたレコードには、イベント・モニターのログで、*部分レコード ID* が付けられます。

イベント・モニターがフラッシュされると、そのモニターのアクティブな内部バッファーが、イベント・モニターの出力オブジェクトに書き込まれます。

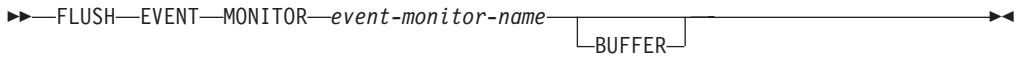
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

許可 ID の特権には、 SYSADM 権限または DBADM 権限のいずれかが含まれていなければなりません (SQLSTATE 42502)。

構文:



説明:

event-monitor-name
イベント・モニターの名前。 これは、1 つの部分からなる名前です。 これは、SQL ID です。

BUFFER
イベント・モニターのバッファーを書き出すことを示します。 BUFFER を指定すると、部分レコードは生成されません。 イベント・モニターのバッファーにすでに入っているデータだけが書き出されます。

注:

- イベント・モニターをフラッシュアウトしても、イベント・モニター値はリセットされません。 これはつまり、フラッシュが実行されない場合に生成されていたイベント・モニターのレコードが、通常のモニター・イベントが起動されるときにもやはり生成されるということです。

FLUSH PACKAGE CACHE

FLUSH PACKAGE CACHE ステートメントは、現在パッケージ・キャッシュ内に存在する、キャッシュされたすべての動的 SQL を除去します。このステートメントは、キャッシュされたすべての動的 SQL ステートメントを論理的に無効化し、同じ SQL ステートメントに対する次の要求が強制的に DB2 によって暗黙的にコンパイルされるようにします。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの許可 ID が持つ特権には、 SYSADM 権限か DBADM 権限のいずれかが含まれている必要があります (SQLSTATE 42502)。

構文:

注:

- このステートメントは、すべてのアクティブ・データベース・パーティション上のパッケージ・キャッシュ内の、キャッシュされたすべての動的 SQL 項目に影響を与えます。
- キャッシュされた動的 SQL ステートメントが無効にされると、キャッシュされた項目に使用されたパッケージ・キャッシュ・メモリーは、 FLUSH PACKAGE CACHE ステートメントの実行時に項目が使用中でなければ解放されます。
- 現在使用中の、キャッシュされた動的 SQL ステートメントは、現在のユーザーが必要としなくなるまでパッケージ・キャッシュ内に存在することができます。同じステートメントの次の新しいユーザーは、DB2 によるステートメントの暗黙作成を強制するので、そのユーザーはキャッシュされた動的 SQL ステートメントの新しいバージョンを実行することになります。

FOR

FOR ステートメントは、表の行ごとに、ステートメントまたはステートメントのグループを実行します。

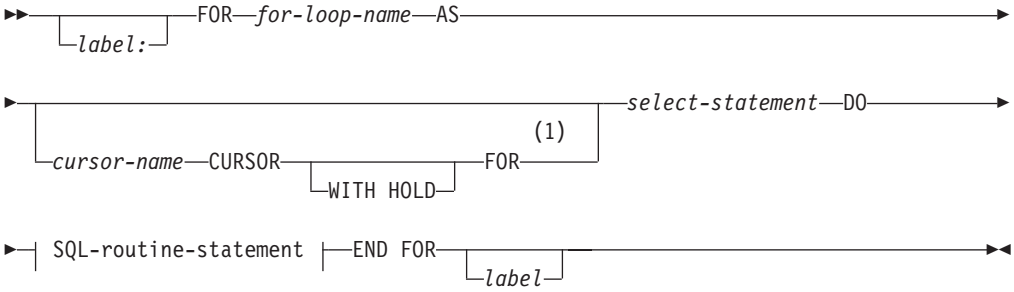
呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

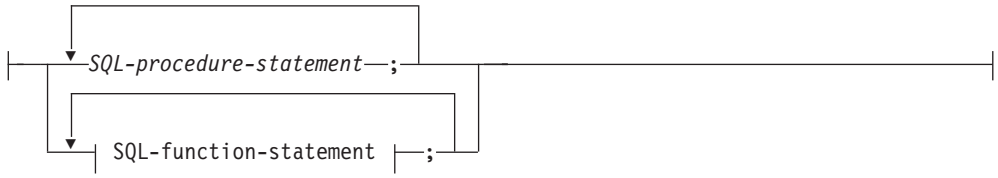
許可:

FOR ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 FOR ステートメントに組み込まれている SQL ステートメントを呼び出すために必要な特権がなければなりません。カーソルの使用に必要な許可については、DECLARE CURSOR ステートメントの説明を参照してください。

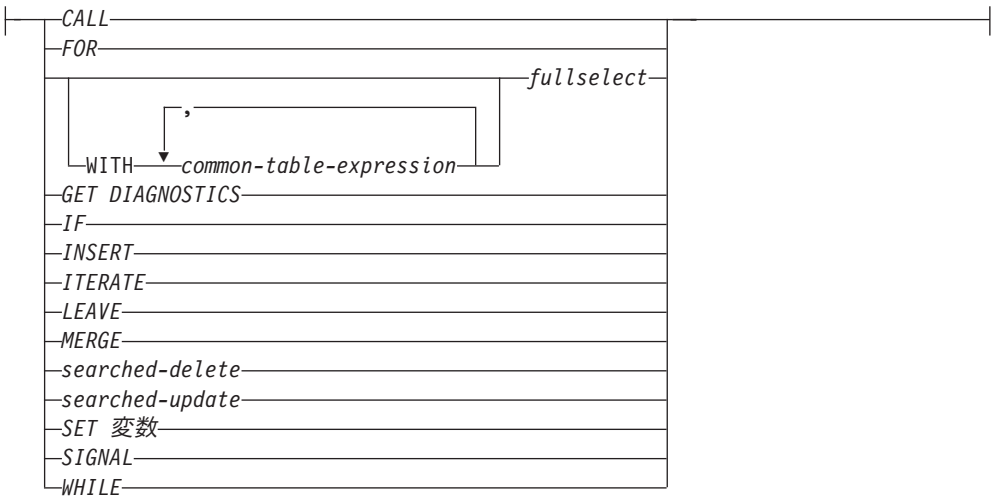
構文:



SQL-routine-statement:



SQL-function-statement:



注:

1 このオプションは、SQL プロシージャーのコンテキスト内でのみ使用できます。

説明:

label
FOR ステートメントのラベルを指定します。開始ラベルが指定された場合、そのラベルは LEAVE および ITERATE ステートメントで使用できます。終了ラベルを指定する場合、そのラベルは開始ラベルと同じでなければなりません。

for-loop-name
FOR ステートメントをインプリメントするために生成された暗黙的コンパウンド・ステートメントのラベルを指定します。 FOR ステートメント内の ITERATE および LEAVE ステートメントで使用できないことを除いては、コンパウンド・ステートメントのラベルの規則に従います。 *for-loop-name* は、指定された *select-statement* によって返された列名を修飾するために使用します。

cursor-name
SELECT ステートメントの結果表から行を選択するために使用されるカーソルを指定します。指定しない場合は、DB2 がユニークなカーソル名を生成します。WITH HOLD 文節の説明については、 『DECLARE CURSOR』 ステートメントを参照してください。

select-statement
カーソルの SELECT ステートメントを指定します。選択リスト内の列にはすべて名前がなければならず、同じ名前の列が 2 つあってはいけません。

トリガー、関数、メソッド、または動的コンパウンド・ステートメントでは、*select-statement* はオプションで共通表式を持つ *fullselect* のみから構成されていなければなりません。

SQL-procedure-statement
表の各行に対して呼び出すステートメントを 1 つ以上指定します。 *SQL-procedure-statement* は、SQL プロシージャーのコンテキスト内でのみ使用できます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

SQL-function-statement
表の各行に対して呼び出すステートメントを 1 つ以上指定します。ニックネームに対する searched-update (検索済み更新)、searched-delete (検索済み削除)、または INSERT 操作はサポートされていません。 *SQL-function-statement* は、SQL 関数または SQL メソッドのコンテキスト内でのみ使用できます。

規則:

- 選択リストはユニークな列名から構成されていることが必要で、選択リストで指定された表はプロシージャーが作成されたときには存在していなければなりません。そうでなければ、これは前の SQL プロシージャー・ステートメントで作成された表でなければなりません。
- for-statement で指定されたカーソルは、for-statement の外側では参照できず、OPEN、FETCH、または CLOSE ステートメントでは指定できません。

例:

下の例では、for-statement は `employee` 表全体を繰り返すために使用されています。表の中の行ごとに、SQL 変数 `fullname` は、従業員のラストネーム (姓)、コンマ、ファーストネーム (名)、ブランク・スペース、そしてミドルネームのイニシャルという順序で設定されます。 `fullname` の各値は、表 `tnames` に挿入されます。

```
BEGIN ATOMIC
  DECLARE fullname CHAR(40);
  FOR v1 AS
    SELECT firstnme, midinit, lastname FROM employee
      DO
      SET fullname = lastname || ',' || firstnme ||' ' || midinit;
      INSERT INTO tnames VALUES (fullname);
   END FOR
END
```

関連資料:

- 491 ページの『DECLARE CURSOR』
- 140 ページの『コンパウンド SQL (プロシージャー)』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』

FREE LOCATOR

FREE LOCATOR ステートメントは、ロケーター変数とその値との間の関連を除去します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

必要ありません。

構文:

説明:

LOCATOR *variable-name,* **...**

1 つまたは複数のロケーター変数を指定します。それらは、ロケーター変数の宣言規則に従って宣言されていなければなりません。

ロケーター変数には、現在ロケーターが割り当てられていなければなりません。つまり、ロケーターはこの作業単位で割り当てられている (CALL、FETCH、SELECT INTO、または VALUES INTO ステートメントによって) 必要があり、またその後解放されて (FREE LOCATOR ステートメントによって) いないことが必要です。これに違反する場合には、エラーが戻されます (SQLSTATE 0F001)。

複数のロケーターを指定すると、リスト中の他のロケーターにエラーがあるか否かには関係なく、解放可能なすべてのロケーターは解放されることになります。

例:

COBOL プログラムで、BLOB ロケーター変数 TKN-VIDEO と TKN-BUF、および CLOB ロケーター変数 LIFE-STORY-LOCATOR を解放します。

```
EXEC SQL
FREE LOCATOR :TKN-VIDEO, :TKN-BUF, :LIFE-STORY-LOCATOR
END-EXEC.
```

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『spserver.c -- Definition of various types of stored procedures』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』
- 『lobeval.sqb -- Demonstrates how to use a Large Object (LOB) (MF COBOL)』

GET DIAGNOSTICS

GET DIAGNOSTICS ステートメントは、以前に実行した SQL ステートメントについての情報を得るために使用されます。

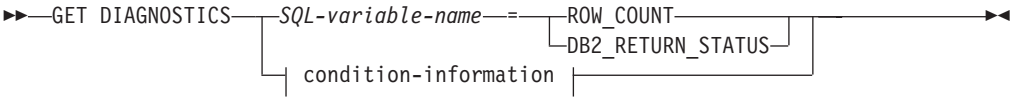
呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

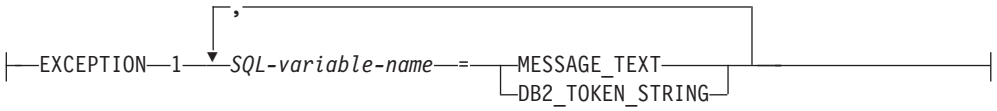
許可:

必要ありません。

構文:



condition-information:



説明:

SQL-variable-name
割り当てのターゲットとなる変数を識別します。 ROW_COUNT または DB2_RETURN_STATUS が指定される場合、この変数は整変数でなければなりません。それ以外の場合は、この変数は CHAR か VARCHAR でなければなりません。 SQL 変数はコンパウンド・ステートメントで定義できます。

ROW_COUNT
直前の SQL ステートメントに関連する行数を識別します。直前の SQL ステートメントが DELETE、INSERT、または UPDATE ステートメントである場合、ROW_COUNT は、操作のために選ばれた行数を識別します。直前のステートメントが PREPARE ステートメントの場合、ROW_COUNT は、準備済みステートメントの結果行の見積もり 数を識別します。

DB2_RETURN_STATUS
直前に実行された SQL ステートメントが、状況を戻すプロシージャーを呼び出す CALL ステートメントの場合に、そのステートメントに関連するストアード・プロシージャーから戻される状況値を識別します。直前のステートメントがそのようなステートメントでなければ、戻される値は特に意味のない、何らかの整数です。

condition-information
以前に実行した SQL ステートメントのエラー情報または警告情報が戻されることを指定します。エラーについての情報が必要であれば、GET DIAGNOSTICS ステートメントは、エラーを処理するハンドラーに指定された最初のステートメントでなければなりません。警告についての情報が必要で、ハンドラーが警告条

件を制御している場合は、 GET DIAGNOSTICS ステートメントは、そのハンドラーに指定された最初のステートメントでなければなりません。ハンドラーが警告条件を制御していない 場合、 GET DIAGNOSTICS ステートメントは、実行される次のステートメントでなければなりません。このオプションは、SQL プロシージャーのコンテキスト内にのみ指定できます (SQLSTATE 42601)。

MESSAGE_TEXT
以前に実行された SQL ステートメントから戻されたエラーまたは警告メッセージ・テキストを示します。このメッセージ・テキストは、そのステートメントが処理されたデータベース・サーバーの言語で戻されます。ステートメントがゼロの SQLCODE で完了した場合、VARCHAR 変数には空ストリングが戻され、 CHAR 変数にはブランクが戻されます。

DB2_TOKEN_STRING
以前に実行された SQL ステートメントから戻されたエラーまたは警告メッセージ・トークンを示します。ステートメントがゼロの SQLCODE で完了したか、SQLCODE にトークンがない場合、 VARCHAR 変数には空ストリングが戻され、CHAR 変数にはブランクが戻されます。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - DB2_RETURN_STATUS の代わりに RETURN_STATUS を指定できます。
- GET DIAGNOSTICS ステートメントは、診断域の内容の変更は行いません (SQLCA)。 SQLSTATE または SQLCODE 特殊変数が SQL プロシージャーで宣言されている場合、 GET DIAGNOSTICS ステートメントの発行によって戻される SQLSTATE または SQLCODE に設定されます。

例:

SQL プロシージャーで GET DIAGNOSTICS ステートメントを実行し、更新された行数を判別します。

```
CREATE PROCEDURE sqlprocg (IN deptnbr VARCHAR(3))
   LANGUAGE SQL
     BEGIN
      DECLARE SQLSTATE CHAR(5);
      DECLARE rcount INTEGER;
      UPDATE CORPDATA.PROJECT
         SET PRSTAFF = PRSTAFF + 1.5
         WHERE DEPTNO = deptnbr;
      GET DIAGNOSTICS rcount = ROW_COUNT;
-- この時点で、rcount には更新済みの行数が入っています。
...
   END
```

SQL プロシージャー内で TRYIT というストアード・プロシージャーの呼び出しから戻される状況値を処理します。この呼び出しでは、ユーザー障害を示す正の値が明示的に戻されるか、あるいは SQL エラーが発生して負の戻り状況値が戻されます。プロシージャーが成功すると、ゼロの値が戻されます。

```
CREATE PROCEDURE TESTIT ()
   LANGUAGE SQL
   A1:BEGIN
        DECLARE RETVAL INTEGER DEFAULT 0;
      ...
      CALL TRYIT;
```

```
   GET DIAGNOSTICS RETVAL = DB2_RETURN_STATUS;
   IF RETVAL <> 0 THEN
      ...
      LEAVE A1;
   ELSE
      ...
   END IF;
END A1
```

GOTO

GOTO ステートメントは、SQL プロシージャー内のユーザー定義ラベルに分岐させます。

呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

必要ありません。

構文:

説明:

label

処理を続行するラベル付きステートメントを指定します。このラベル付きステートメントと GOTO ステートメントの有効範囲は同じでなければなりません。

- GOTO ステートメントが FOR ステートメントで定義されている場合、 *label* は同じ FOR ステートメントの内側で定義しなければなりません。ただし、ネストされている FOR ステートメントまたはネストされているコンパウンド・ステートメントは除きます。
- GOTO ステートメントがコンパウンド・ステートメントで定義されている場合、 *label* は同じコンパウンド・ステートメントの内側で定義しなければなりません。ただし、ネストされている FOR ステートメントまたはネストされているコンパウンド・ステートメントは除きます。
- GOTO ステートメントがハンドラーで定義されている場合、 *label* は他の有効範囲の規則に従って、同じハンドラーで定義しなければなりません。
- GOTO ステートメントがハンドラーの外側で定義されている場合、 *label* をハンドラーの内側で定義してはなりません。

label が、GOTO ステートメントが到達できる有効範囲内で定義されていない場合、エラーが戻されます (SQLSTATE 42736)。

注:

- GOTO ステートメントは使い過ぎないようにお勧めします。このステートメントは通常の処理シーケンスを妨げるので、ルーチンの読み取りおよび保守が困難になります。なるべく GOTO ステートメントを使用しなくて済むように、GOTO ステートメントを使用する前に、他のステートメント (IF や LEAVE など) を代わりに使用できるかどうか判別してください。

例:

以下のコンパウンド・ステートメントでは、パラメーター *rating* および *v_empno* がプロシージャーに渡されます。そして、日付期間として出力パラメーター

return_parm が戻されます。従業員のその会社での就労期間が 6 か月未満の場合、GOTO ステートメントは制御をプロシージャーの最後に移動させ、 *new_salary* は未変更のままになります。

```
CREATE PROCEDURE adjust_salary
  (IN v_empno CHAR(6),
  IN rating INTEGER)
  OUT return_parm DECIMAL (8,2))
  MODIFIES SQL DATA
  LANGUAGE SQL
  BEGIN
    DECLARE new_salary DECIMAL (9,2)
    DECLARE service DECIMAL (8,2)
      SELECT SALARY, CURRENT_DATE - HIREDATE
        INTO new_salary, service
        FROM EMPLOYEE
        WHERE EMPNO = v_empno
      IF service < 600
        THEN GOTO EXIT
      END IF
      IF rating = 1
        THEN SET new_salary = new_salary + (new_salary * .10)
      ELSE IF rating = 2
        THEN SET new_salary = new_salary + (new_salary * .05)
      END IF
      UPDATE EMPLOYEE
        SET SALARY = new_salary
        WHERE EMPNO = v_empno
      EXIT: SET return_parm = service
END
```

GRANT (データベース権限)

この形式の GRANT ステートメントは、データベース全体に適用される権限 (データベース内の特定のオブジェクトに適用される特権ではなく) を付与します。

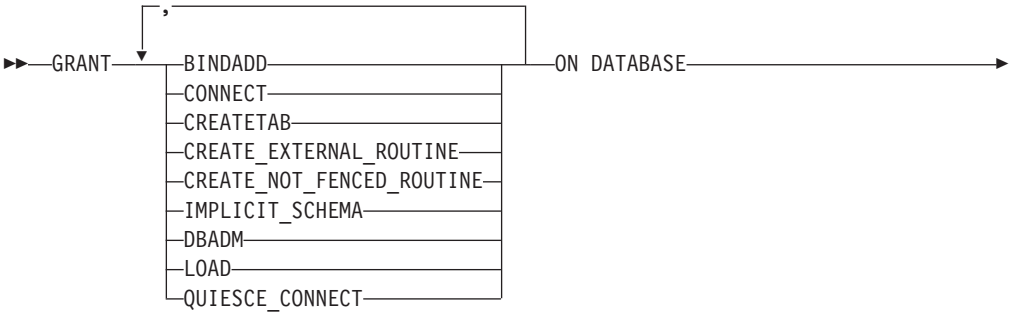
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

DBADM 権限を付与するには、SYSADM 権限が必要です。その他の権限を付与するには、DBADM 権限、または SYSADM 権限のいずれかが必要です。

構文:



説明:

BINDADD
パッケージを作成する権限を付与します。パッケージの作成者には自動的にそのパッケージに対する CONTROL 特権が与えられ、後で BINDADD 権限が取り消されたとしてもその特権はそのまま保持されます。

CONNECT
データベースにアクセスする権限を与えます。

CREATETAB
基本表を作成する権限を付与します。基本表の作成者には、自動的にその表に対する CONTROL 特権が与えられます。後で CREATETAB 権限が取り消されたとしても、作成者はこの特権を保持したままになります。

ビュー作成に必要な明示的な権限は特にありません。ビューの作成に使用するステートメントの許可 ID に各ビューの基本表に対する CONTROL 特権または SELECT 特権のいずれかが与えられている場合には、いつでもビューを作成できます。

CREATE_EXTERNAL_ROUTINE
外部ルーチンを登録する権限を付与します。そのようにして登録されたルーチンが不利な副作用を引き起こすことがないように注意してください。 (詳しくは、CREATE または ALTER ステートメント上の THREADSAFE 文節の説明を参照してください。)

外部ルーチンを登録し終えると、後で CREATE_EXTERNAL_ROUTINE が取り消されたとしてもそのまま保持されます。

CREATE_NOT_FENCED_ROUTINE
データベース・マネージャーの処理の中で実行するルーチンを登録する権限を与えます。そのようにして登録されたルーチンが不利な副作用を引き起こすことがないように注意してください。 (詳しくは、CREATE または ALTER ステートメント上の FENCED 文節の説明を参照してください。)

ルーチンが非 fenced として登録された場合は、それ以降に CREATE_NOT_FENCED が取り消されたとしてもその方式での実行が続けられます。

CREATE_NOT_FENCED_ROUTINE 権限を付与される *authorization-name* には、自動的に CREATE_EXTERNAL_ROUTINE が付与されます。

IMPLICIT_SCHEMA
スキーマを暗黙的に作成する権限を与えます。

DBADM
データベース管理者の権限や、他のすべてのデータベース権限を付与します。データベース管理者にはデータベース中のすべてのオブジェクトに対する特権が与えられ、また他のユーザーにそれらの特権を与えることができます。

注: 他のすべてのデータベース権限は、 DBADM 権限を与えられている *authorization-name* に暗黙的かつ自動的に与えられます。

LOAD
このデータベースでロードを実行する権限を付与します。この権限を付与されたユーザーは、このデータベースにおいて LOAD ユーティリティーを使用する権利を持ちます。この権限は、デフォルトで SYSADM と DBADM にも付与されます。ただし、LOAD 権限しか付与されていないユーザー (SYSADM と DBADM 以外) の場合は、表レベルでの特権が別に必要になります。 LOAD 特権に加えて、ユーザーは以下の特権を付与されていなければなりません。

- モード INSERT、TERMINATE (直前の LOAD INSERT を終了するため)、または RESTART (直前の LOAD INSERT を再び開始するため) で LOAD を実行する場合は、その表に対する INSERT 特権。
- モード REPLACE、TERMINATE (直前の LOAD REPLACE を終了するため)、または RESTART (直前の LOAD REPLACE を再び開始するため) で LOAD を実行する場合は、その表に対する INSERT および DELETE 特権。
- LOAD の一部として例外表を使用する場合は、その表に対する INSERT 特権。

QUIESCE_CONNECT
静止中のデータベースにアクセスする権限を与えます。

TO

権限を誰に与えるかを指定します。

USER

authorization-name がユーザーであることを指定します。

GROUP

authorization-name がグループ名であることを指定します。

authorization-name,...

1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC

すべてのユーザーに対して権限を付与します。 DBADM は、PUBLIC に付与することはできません。

規則:

- USER も GROUP も指定しない場合には、
  - authorization-name がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - authorization-name がオペレーティング・システムで USER としてのみ定義されている場合には、 USER であると見なされます。
  - オペレーティング・システムで authorization-name が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - CREATE_NOT_FENCED_ROUTINE の代わりに CREATE_NOT_FENCED を指定できます。

例:

例 1: ユーザー WINKEN、BLINKEN、および NOD に、データベースに接続する権限を与えます。

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

例 2: データベースに対する BINDADD 権限を、D024 という名前のグループに与えます。システムには、D024 と呼ばれるグループとユーザーの両方が存在しています。

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

GROUP キーワードの指定は必須です。この指定がない場合、D024 という名前のユーザーとグループが両方とも存在しているのでエラーになります。 D024 グループのメンバーは、いずれもデータベースのパッケージをバインドできるようになります。しかし、D024 というユーザーはそれは許されません (ただし、このユーザーがグループ D024 のメンバーでもある場合、または以前に BINDADD 権限を与えられていた場合、または BINDADD 権限がユーザー D024 がメンバーとして属している別のグループに与えられていた場合を除きます)。

関連資料:

- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

関連サンプル:

- 『dbauth.sqb -- How to grant and display authorities on a database (MF COBOL)』
- 『dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)』
- 『dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)』
- 『DbAuth.java -- Grant, display or revoke privileges on database (JDBC)』
- 『DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)』

GRANT (索引特権)

この形式の GRANT ステートメントは、索引に対する CONTROL 特権を付与します。

呼び出し:

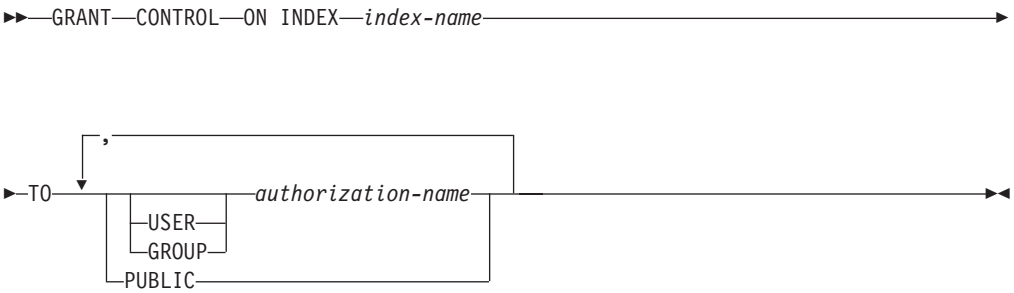
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- DBADM 権限
- SYSADM 権限

構文:



説明:

CONTROL
索引をドロップする特権を付与します。 これは、索引の作成者に自動的に与えられるその索引に対する CONTROL 権限です。

ON INDEX *index-name*
CONTROL 特権を付与する対象となる索引の名前を指定します。

TO
特権を誰に与えるかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーに特権を付与します。

規則:

- USER も GROUP も指定しない場合には、
  - authorization-name がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - authorization-name がオペレーティング・システムで USER としてのみ定義されている場合には、 USER であると見なされます。
  - オペレーティング・システムで authorization-name が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。

例:

```
GRANT CONTROL ON INDEX DEPTIDX TO USER USER4
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

GRANT (パッケージ特権)

この形式の GRANT ステートメントは、パッケージに対する特権を付与します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

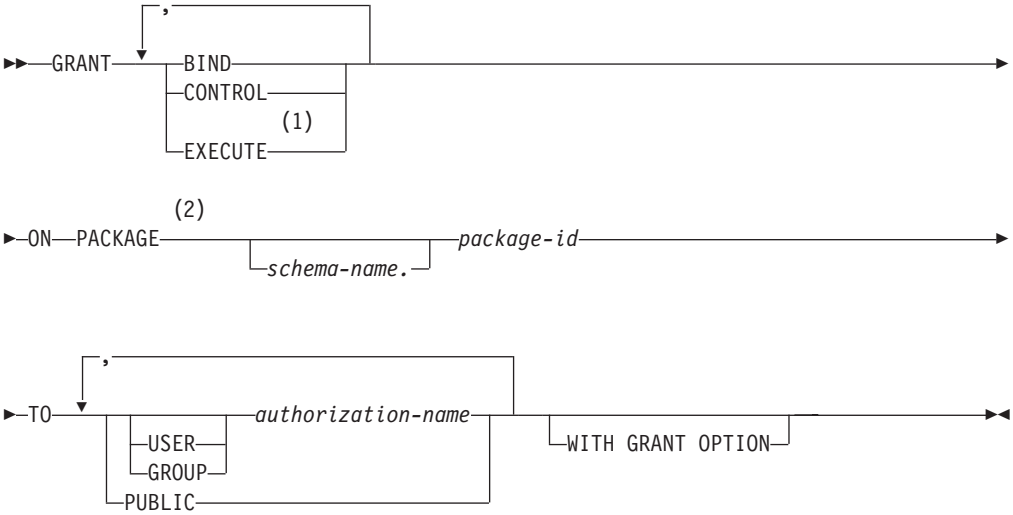
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 参照されるパッケージに対する CONTROL 特権
- *package-name* 上の指定した各特権に対する WITH GRANT OPTION
- SYSADM または DBADM 権限

CONTROL 特権を付与するには、SYSADM または DBADM の権限が必要です。

構文:



注:

1 EXECUTE の同義語として RUN を使用できます。

2 PACKAGE の同義語として PROGRAM を使用できます。

説明:

BIND

パッケージをバインドする特権を付与します。 BIND 特権を使用すると、ユーザーが BIND コマンドをパッケージに対して再発行したり、 REBIND コマンドを発行したりできます。また、ユーザーが既存のパッケージの新しいバージョンを作成することもできます。

ユーザーには、BIND 特権に加えて、プログラムに含まれている静的 DML ステートメントによって参照される表ごとに必要な特権が与えられていなければなりません。これは、静的 DML ステートメントに対する許可がバインド時に検査されるので必要になります。

CONTROL
パッケージを再バインド、ドロップ、または実行するための特権、およびパッケージ特権を他のユーザーに与える特権を付与します。パッケージの作成者には、自動的にパッケージに対する CONTROL 特権が与えられます。パッケージ所有者は、パッケージ・バインド・プログラムか、またはバインド/プリコンパイル時に OWNER オプションを使って指定した ID です。

CONTROL 権限を付与される *authorization-name* には、自動的に BIND と EXECUTE が付与されます。

CONTROL は、他のユーザーに上記の特権 (CONTROL を除く) を付与する特権を付与します。

EXECUTE
パッケージを実行する特権を与えます。

ON PACKAGE *schema-name.package-id*
特権の対象となるパッケージの名前を指定します。スキーマ名が指定されていない場合、パッケージ ID は暗黙的にデフォルト・スキーマで修飾されます。パッケージ特権の付与は、そのパッケージのすべてのバージョン (つまり、同一のパッケージ ID とパッケージ・スキーマを共用するすべてのパッケージ) に適用されます。

TO
特権を誰に与えるかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーに特権を付与します。

WITH GRANT OPTION
指定した *authorization-name* に対し、特権を他のユーザーに与えることを許可します。

指定した特権に CONTROL が含まれる場合、 WITH GRANT OPTION は CONTROL を除くすべての適用可能な特権に適用されます (SQLSTATE 01516)。

規則:

- USER も GROUP も指定しない場合には、

- *authorization-name* がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
- *authorization-name* がオペレーティング・システムで USER としてのみ定義されているか、未定義の場合には、USER であると見なされます。
- オペレーティング・システムで *authorization-name* が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。

注:

- パッケージ特権は、パッケージのすべてのバージョン (つまり、同一のパッケージ ID とパッケージ・スキーマを共用するすべてのパッケージ) に適用されます。アクセスを 1 つのバージョンだけに制約することはできません。
  CONTROL 特権はパッケージのバインド・プログラムに暗黙的に付与されるので、 2 人のユーザーが 2 つのバージョンのパッケージをバインドすると、両者とも互いのパッケージに対するアクセス権を暗黙的に付与されます。

例:

例 1: PACKAGE CORPDATA.PKGA に対する EXECUTE 権限を PUBLIC に与えます。

```
GRANT EXECUTE
  ON PACKAGE CORPDATA.PKGA
  TO PUBLIC
```

例 2: パッケージ CORPDATA.PKGA に対する EXECUTE 権限を EMPLOYEE という名前のユーザーに与えます。 EMPLOYEE と呼ばれるグループもユーザーも存在していません。

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO EMPLOYEE
```

または

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO USER EMPLOYEE
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

GRANT (ルーチン特権)

この形式の GRANT ステートメントは、ルーチン (関数、メソッド、またはプロシージャー) に対する特権を付与します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

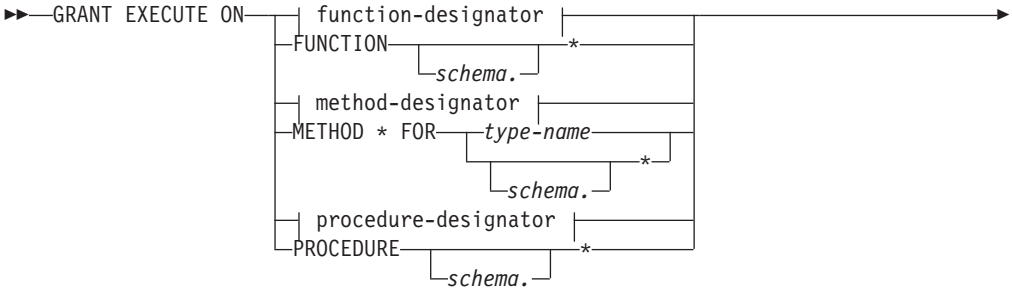
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

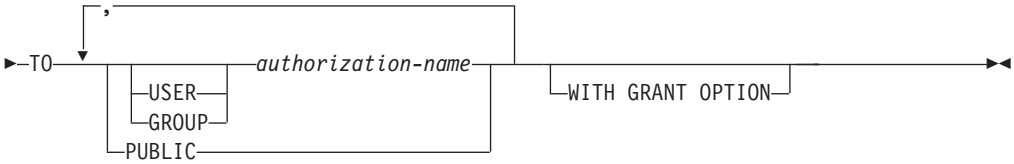
- ルーチンに対する EXECUTE の WITH GRANT OPTION
- SYSADM または DBADM 権限

特定のスキーマ中または特定のタイプのすべてのルーチン EXECUTE 特権を付与するには、許可 ID によって保持されている特権に少なくとも以下のいずれかが含まれていなければなりません。

- 指定されたスキーマ中の (指定されたタイプの)、すべての既存のルーチンと将来作成するルーチンに対する EXECUTE の WITH GRANT OPTION
- SYSADM または DBADM 権限

構文:





説明:

EXECUTE
識別されたユーザー定義の関数、メソッド、またはストアード・プロシージャーを実行する特権を付与します。

function-designator
関数を一意的に識別します。

FUNCTION *schema*.*
スキーマ中のすべての関数を識別します。将来作成される予定の関数も含まれます。動的 SQL ステートメント中でスキーマが指定されていない場合は、CURRENT SCHEMA 特殊レジスター中のスキーマが使用されます。静的 SQL ステートメント中でスキーマが指定されていない場合は、 QUALIFIER プリコンパイル/ BIND オプション中のスキーマが使用されます。

method-designator
メソッドを一意的に識別します。

METHOD *
タイプ *type-name* のすべてのメソッドを識別します。将来作成される予定のメソッドも含まれます。

FOR *type-name*
指定されたメソッドを検索する際のタイプを指定します。ここで指定される名前は、カタログにすでに記述されているタイプを示すものでなければなりません (SQLSTATE 42704)。動的 SQL ステートメントでは、 CURRENT SCHEMA 特殊レジスターの値が、修飾子のないタイプ名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないタイプ名に修飾子が暗黙指定されます。 *type-name* の代わりにアスタリスク (*) を使用して、スキーマ中のすべてのタイプを識別することもできます。これには将来作成される予定のタイプも含まれます。

procedure-designator
プロシージャーを一意的に識別します。

PROCEDURE *schema*.*
スキーマ中のすべてのプロシージャーを識別します。将来作成される予定のプロシージャーも含まれます。動的 SQL ステートメント中でスキーマが指定されていない場合は、 CURRENT SCHEMA 特殊レジスター中のスキーマが使用されます。静的 SQL ステートメント中でスキーマが指定されていない場合は、QUALIFIER プリコンパイル/ BIND オプション中のスキーマが使用されます。

TO
EXECUTE 特権を誰に付与するかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

PUBLIC
すべてのユーザーに EXECUTE 特権を付与します。

WITH GRANT OPTION
指定した *authorization-name* に対し、EXECUTE 特権を他のユーザーに与えることを許可します。

WITH GRANT OPTION を省略すると、指定した *authorization-name* は以下のいずれかの場合にのみ、 EXECUTE 特権を他のユーザーに与えることができます。

- SYSADM または DBADM 権限を持っている。
- 他のソースから EXECUTE 特権を与える許可を受けた。

規則:

- スキーマ 'SYSIBM' または 'SYSFUN' を使って定義された関数やメソッドに対する EXECUTE 特権を付与することはできません (SQLSTATE 42832)。
- USER も GROUP も指定しない場合には、
  - *authorization-name* がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - *authorization-name* がオペレーティング・システムで USER としてのみ定義されているか、未定義の場合には、USER であると見なされます。
  - オペレーティング・システムで *authorization-name* が USER と GROUP の両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。
- 一般に、GRANT ステートメントはステートメントの許可 ID が与えることを許されている特権の GRANT のみを処理し、 1 つまたは複数の特権が与えられなかった場合は警告 (SQLSTATE 01007) を戻します。ステートメントの処理に使用されるパッケージが、 LANGLEVEL を SQL92E または MIA に設定してプリコンパイルされていた場合、特権が付与されない場合には、警告が戻されます (SQLSTATE 01007)。付与者が GRANT 操作の対象に対して特権を持っていない場合、エラーが戻されます (SQLSTATE 42501)。

例:

例 1: 関数 CALC_SALARY に対する EXECUTE 特権をユーザー JONES に与えます。スキーマ中に CALC_SALARY という名前の関数が 1 つだけ含まれていると想定しています。

```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES
```

例 2: プロシージャー VACATION_ACCR に対する EXECUTE 特権を、現行サーバー上のすべてのユーザーに与えます。

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC
```

例 3: 関数 DEPT_TOTALS に対する EXECUTE 特権を管理部門のアシスタントに与え、この関数に対する EXECUTE 特権を他者に付与する特権をこのアシスタントに与えます。この関数には DEPT85_TOT という特定の名前があります。スキーマに DEPT_TOTALS という名前の関数が複数あることを想定しています。

```
GRANT EXECUTE ON SPECIFIC FUNCTION DEPT85_TOT
  TO ADMIN_A WITH GRANT OPTION
```

例 4: 関数 NEW_DEPT_HIRES に対する EXECUTE 特権を HR (Human Resources) に与えます。この関数には、2 つの入力パラメーターがあり、それぞれのパラメーターのタイプは INTEGER および CHAR(10) です。スキーマに NEW_DEPT_HIRES という名前の関数が複数あることを想定しています。

```
GRANT EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10)) TO HR
```

例 5: タイプ EMPLOYEE のメソッド SET_SALARY に対する EXECUTE 特権をユーザー JONES に与えます。

```
GRANT EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE TO JONES
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- x ページの『共通の構文エレメント』

GRANT (スキーマ特権)

この形式の GRANT ステートメントは、スキーマに対する特権を付与します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

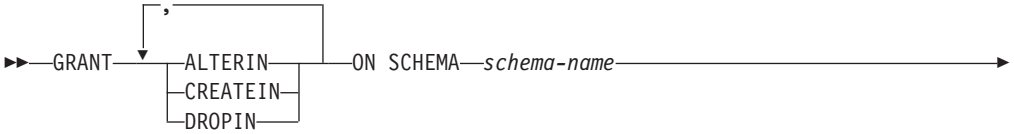
許可:

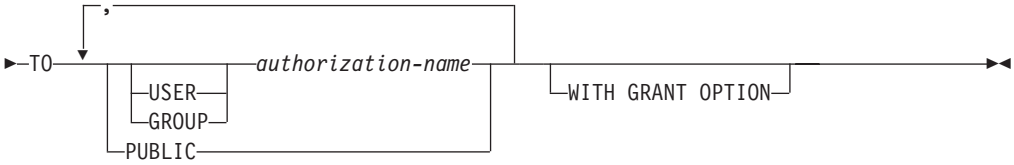
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- *schema-name* 上の指定した各特権に対する WITH GRANT OPTION
- SYSADM または DBADM 権限

スキーマ名 SYSIBM、SYSCAT、SYSFUN および SYSSTAT に対する特権は、ユーザーが与えることはできません (SQLSTATE 42501)。

構文:





説明:

ALTERIN
スキーマ内のすべてのオブジェクトの変更、またはコメント付けのための特権を与えます。明示的にスキーマを作成した所有者は、ALTERIN 特権が自動的に与えられます。

CREATEIN
スキーマにオブジェクトを作成する特権を与えます。 オブジェクトの作成に必要なその他の権限または特権 (CREATETAB など) は、これを指定しても必要です。明示的に作成されたスキーマの所有者には、自動的に CREATEIN 特権が付与されます。暗黙的に作成されたスキーマの CREATEIN 特権は、PUBLIC に自動的に付与されます。

DROPIN
スキーマ内のオブジェクトをドロップする特権を与えます。 明示的に作成されたスキーマの所有者は、DROPIN 特権を自動的に与えられます。

ON SCHEMA *schema-name*
特権を与える対象となるスキーマを指定します。

TO
特権を誰に与えるかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーに特権を付与します。

WITH GRANT OPTION
指定した *authorization-name* に対し、特権を他のユーザーに与えることを許可します。

WITH GRANT OPTION を省略すると、指定した *authorization-name* は以下のいずれかの場合にのみ、特権を他のユーザーに与えることができます。

- DBADM 権限を持っている
- 他のソースから特権を与える許可を受けた

規則:

- USER も GROUP も指定しない場合には、
  - authorization-name がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - authorization-name がオペレーティング・システムで USER としてのみ定義されている場合には、 USER であると見なされます。
  - オペレーティング・システムで authorization-name が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。
- 一般に、GRANT ステートメントはステートメントの許可 ID が与えることを許されている特権の GRANT のみを処理し、 1 つまたは複数の特権が与えられなかった場合は警告 (SQLSTATE 01007) を戻します。どのような特権も与えられなかった場合は、エラーが戻されます (SQLSTATE 42501)。ステートメントの処理に使用されるパッケージが、 LANGLEVEL を SQL92E または MIA に設定してプリコンパイルされていた場合、付与者が GRANT 操作の対象に対して特権を持っていない場合以外は警告が戻されます (SQLSTATE 01007)。

例:

例 1: スキーマ CORPDATA にオブジェクトを作成する特権を、ユーザー JSINGLETON に与えます。

```
GRANT CREATEIN ON SCHEMA CORPDATA TO JSINGLETON
```

例 2: スキーマ CORPDATA のオブジェクトを作成およびドロップする特権を、ユーザー IHAKES に与えます。

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO IHAKES
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

GRANT (シーケンス特権)

この GRANT ステートメントのフォームは、シーケンスでの特権を付与します。

呼び出し:

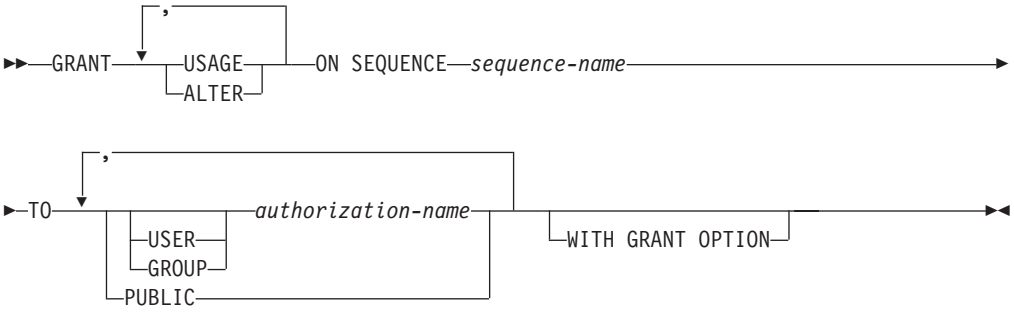
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- *sequence-name* に対して指定された各特権ごとに WITH GRANT OPTION
- SYSADM または DBADM 権限

構文:



説明:

USAGE
nextval-expression または *prevval-expression* を使用してシーケンスを参照する特権を付与します。

ALTER
ALTER SEQUENCE ステートメントを使用してシーケンス・プロパティーを変更する特権を付与します。

ON SEQUENCE *sequence-name*
指定された特権が付与されるシーケンスを識別します。暗黙的または明示的スキーマ修飾子を含むシーケンス名は、現在のサーバーに存在するシーケンスを固有に識別していなければなりません。この名前によるシーケンスが存在しない場合、エラー (SQLSTATE 42704) が戻されます。

TO
指定された特権を誰に与えるかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループであることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

PUBLIC
すべてのユーザーに指定された特権を付与します。

WITH GRANT OPTION
指定した *authorization-name* に対して、指定した特権を他のユーザーに与えることを許可します。

WITH GRANT OPTION を省略すると、指定した *authorization-name* は以下のいずれかの場合にのみ、指定された特権を他のユーザーに与えることができます。

- SYSADM または DBADM 権限を持っている。
- 他のソースから、指定された特権を与える許可を受けた。

規則:

- USER も GROUP も指定しない場合には、
  - *authorization-name* がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - *authorization-name* がオペレーティング・システムで USER としてのみ定義されているか、未定義の場合には、USER であると見なされます。
  - オペレーティング・システムで *authorization-name* が USER と GROUP の両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。
- 一般に、GRANT ステートメントはステートメントの許可 ID が与えることを許されている特権の GRANT のみを処理し、 1 つまたは複数の特権が与えられていない場合は警告 (SQLSTATE 01007) を戻します。どの特権も与えられていない場合は、エラーが戻されます (SQLSTATE 42501)。ステートメントの処理に使用されるパッケージが、 LANGLEVEL を SQL92E または MIA に設定してプリコンパイルされていた場合、付与者が GRANT 操作の対象に対して特権を持っていない場合以外は警告が戻されます (SQLSTATE 01007)。

例:

例 1: シーケンス ORG_SEQ での USAGE 特権をユーザーに付与します。

```
GRANT USAGE ON SEQUENCE ORG_SEQ TO PUBLIC
```

例 2: ユーザー BOBBY に、GENERATE_ID というシーケンスを変更する許可と、この特権を他のユーザーに付与する許可を与えます。

```
GRANT ALTER ON SEQUENCE GENERATE_ID TO BOBBY WITH GRANT OPTION
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』

- 589 ページの『GRANT (ルーチン特権)』

GRANT (サーバー特権)

この形式の GRANT ステートメントは、指定したデータ・ソースにパススルー・モードでアクセスおよび使用する特権を付与します。

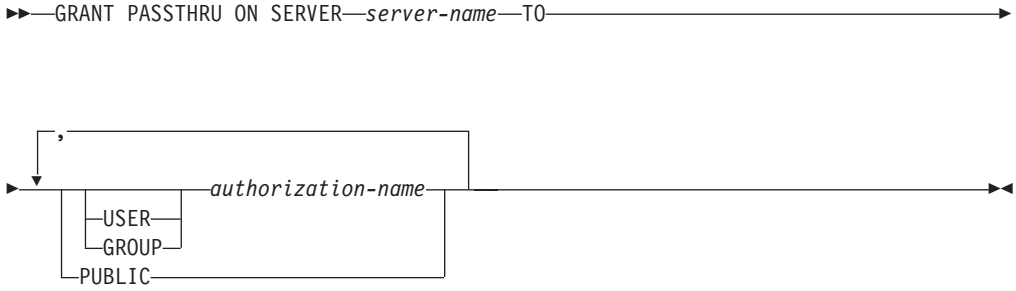
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID は、 SYSADM または DBADM のいずれかの権限を持っている必要があります。

構文:



説明:

server-name
パススルー・モードで使用する特権が与えられるデータ・ソースを指定します。 *server-name* (サーバー名) は、カタログに記述されているデータ・ソースを指定していなければなりません。

TO
特権を誰に付与するかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
server-name にパススルーする特権をすべてのユーザーに付与します。

例:

例 1: R. Smith および J. Jones に、データ・ソース SERVALL にパススルーする特権を付与します。この 2 人の許可 ID は RSMITH および JJONES です。

```
   GRANT PASSTHRU ON SERVER SERVALL
TO USER RSMITH,
USER JJONES
```

例 2: データ・ソース EASTWING にパススルーする特権を、許可 ID が D024 のグループに付与します。許可 ID が D024 であるユーザーも存在しています。

```
   GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

GROUP キーワードの指定は必須です。この指定がない場合、D024 という名前のユーザーとグループが両方とも存在しているので、エラーになります (SQLSTATE 56092)。グループ D024 のメンバーはすべて、EASTWING にパススルーすることができます。また、ユーザー D024 がこのグループに所属する場合、このユーザーは EASTWING にパススルーすることができます。

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 601 ページの『GRANT (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

GRANT (表スペース特権)

この形式の GRANT ステートメントは、表スペースに対する特権を付与します。

呼び出し:

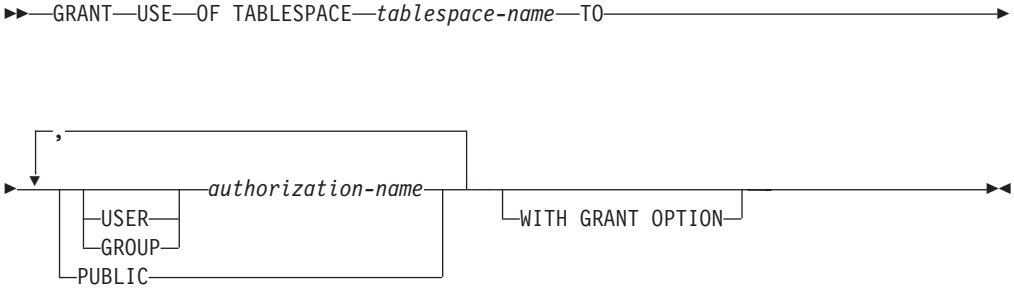
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 表スペースを使用するための WITH GRANT OPTION
- SYSADM、SYSCTRL、または DBADM 権限

構文:



説明:

USE
表を作成する際に表スペースを指定したり、デフォルトの表スペースを使用したりするための特権を付与します。表スペースの作成者には、USE 特権と GRANT オプションが自動的に GRANT されます。

OF TABLESPACE *tablespace-name*
どの表スペースに対する USE 特権を付与するかを指定します。ここで、SYSCATSPACE (SQLSTATE 42838) や SYSTEM TEMPORARY 表スペース (SQLSTATE 42809) を指定することはできません。

TO
USE 特権を誰に付与するかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name
1 人または複数のユーザーまたはグループの許可 ID をリストします。

この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーに USE 特権を付与します。

WITH GRANT OPTION
指定した *authorization-name* に対し、 USE 特権を他のユーザーに与えることを許可します。

WITH GRANT OPTION が省略された場合、指定された *authorization-name* は、以下のいずれかの場合にのみ、USE 特権を他のユーザーに GRANT することができます。

- SYSADM または DBADM 権限を持っている。
- 他のソースから、USE 特権を付与する許可を得ている。

注:

USER も GROUP も指定しない場合には、

- *authorization-name* がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
- *authorization-name* がオペレーティング・システムで USER としてのみ定義されているか、未定義の場合には、USER であると見なされます。
- オペレーティング・システムで *authorization-name* が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。

例:

例 1: ユーザー BOBBY に、表スペース PLANS に表を作成する許可と、この特権を他のユーザーに付与する許可を与えます。

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

関連資料:

- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 603 ページの『GRANT (表、ビュー、またはニックネーム特権)』
- 599 ページの『GRANT (サーバー特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

GRANT (表、ビュー、またはニックネーム特権)

この形式の GRANT ステートメントは、表、ビュー、またはニックネームに対する特権を付与します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

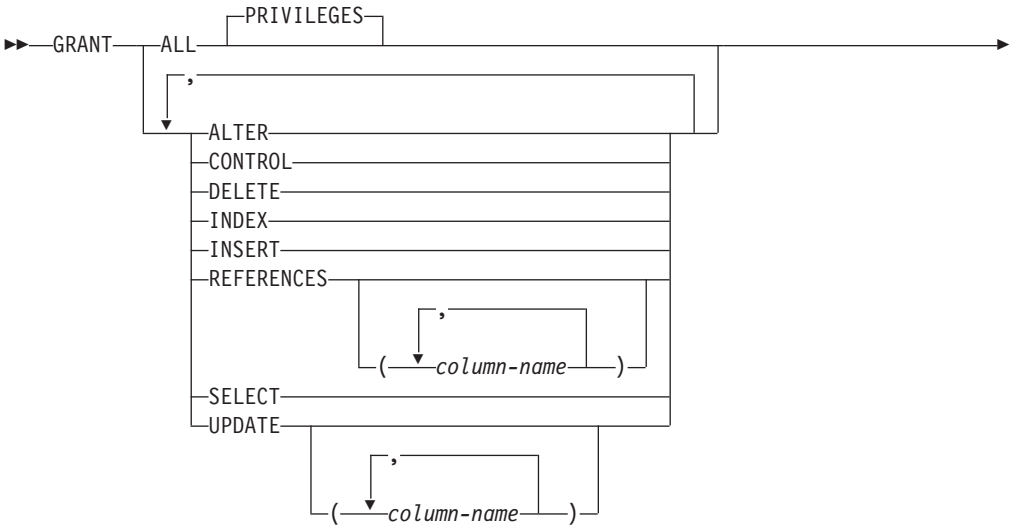
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

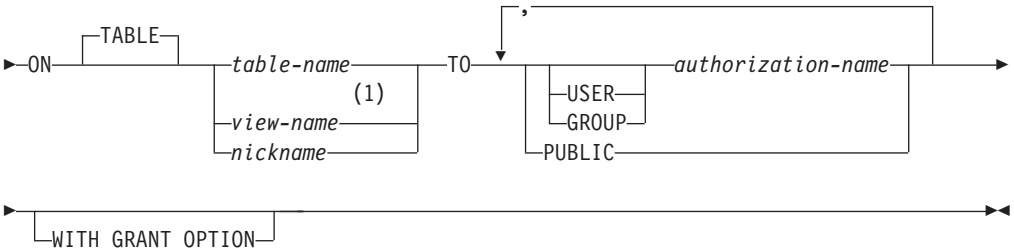
- 参照されている表、ビュー、またはニックネームに対する CONTROL 特権
- 指定したそれぞれの特権に対する WITH GRANT OPTION。 ALL を指定する場合、許可 ID は指定した表、ビュー、またはニックネームに対して何らかの付与可能な特権を持っている必要があります。
- SYSADM または DBADM 権限

CONTROL 特権を付与するには、SYSADM または DBADM の権限が必要です。

カタログ表とカタログ・ビューに対する特権を付与するには、 SYSADM 権限または DBADM 権限のいずれかが必要です。

構文:





注:

1 ALTER、INDEX、および REFERENCES 特権は、ビューには適用されません。

説明:

ALL または **ALL PRIVILEGES**

ON 文節で指定する基本表、ビュー、またはニックネームについて、該当するすべての特権 (CONTROL を除く) を付与します。

ステートメントの許可 ID が表、ビュー、またはニックネームに対して CONTROL 特権を持っている場合、あるいは DBADM 権限または SYSADM 権限を持っている場合には、オブジェクトに適用できる特権のすべて (CONTROL を除く) が与えられます。 そうでない場合、与えられる特権は、ステートメントの許可 ID が指定の表、ビュー、またはニックネームに対して持っているすべての付与可能な特権です。

ALL の指定がない場合、特権のリストに示されているキーワードの 1 つまたは複数を指定する必要があります。

ALTER

以下のことを行うための特権を付与します。

- 基本表の定義に列を追加する。
- 基本表の主キー制約またはユニーク制約を作成またはドロップする。
- 基本表の外部キーを作成またはドロップする。

  親表のそれぞれの列に対する REFERENCES 特権も必要です。
- 基本表のチェック制約を作成またはドロップする。
- 基本表のトリガーを作成する。
- ニックネームの列オプションを追加、リセット、またはドロップする。
- ニックネームの列名またはデータ・タイプを変更する。
- 基本表、またはニックネームのコメントを追加または変更する。

CONTROL

以下を付与します。

- リストに示されているすべての特権。すなわち、
  - 基本表に対する ALTER、CONTROL、DELETE、INSERT、INDEX、REFERENCES、SELECT、および UPDATE
  - ビューに対する CONTROL、DELETE、INSERT、SELECT、および UPDATE

  - ニックネームに対する ALTER、CONTROL、INDEX、および REFERENCES
- 他のユーザーに上記の特権 (CONTROL を除く) を付与する特権。
- 基本表、ビュー、またはニックネームをドロップする特権。

  CONTROL 特権があっても、この特権を他のユーザーに拡張することはできません。拡張するための唯一の方法は、CONTROL 特権を付与することであり、それは SYSADM または DBADM の権限を持つユーザーのみが行うことができます。
- 表と索引に対して RUNSTATS ユーティリティーを実行する特権。
- 表に対して REORG ユーティリティーを実行する特権。
- 基本表、マテリアライズ照会表、またはステージング表に対して SET INTEGRITY ステートメントを発行する特権。

基本表、マテリアライズ照会表、ステージング表、またはニックネームの定義者には、自動的に CONTROL 特権が付与されます。

ビューの定義者に全選択で指定されているすべての表、ビュー、およびニックネームに対する CONTROL 特権が与えられている場合、その定義者には自動的に CONTROL 特権が付与されます。

DELETE
表または更新可能なビューから行を削除する特権を付与します。

INDEX
表の索引、またはニックネームの索引指定を作成する特権を付与します。 この特権は、ビューに対して付与することはできません。索引または索引指定の作成者には、その索引または索引指定に対する CONTROL 特権が自動的に与えられます (これにより、作成者は索引または索引指定をドロップできます)。 さらに、INDEX 特権が取り消されても、作成者は CONTROL 特権をそのまま保持します。

INSERT
表または更新可能なビューに行を挿入し、 IMPORT ユーティリティーを実行する特権を与えます。

REFERENCES
親表を参照するための外部キーの作成やドロップを行う特権を付与します。

ステートメントの許可 ID が以下のいずれかを持っている場合、

- DBADM 権限または SYSADM 権限
- その表に対する CONTROL 特権
- 表に対する REFERENCES WITH GRANT OPTION

特権を与えられたユーザーは、表のすべての列を親キーとして使用して参照制約を作成できます (ALTER TABLE ステートメントを使用して後で追加された列であっても)。そうでない場合、付与される特権はステートメントの許可 ID が指定の表に対して持っているすべての列の付与可能な REFERENCE 特権です。

この特権はニックネームに付与することができますが、ニックネームを参照するために外部キーは定義できません。

REFERENCES (*column-name*,...)
列のリストに指定された列のみを親キーとして使用して外部キーを作成およびドロップする特権を与えます。各 *column-name* (列名) は、 ON 文節で指定される表の列を指定する非修飾名でなければなりません。型付き表、型付きビュー、またはニックネームに対する列レベルの REFERENCES 特権は付与できません (SQLSTATE 42997)。

SELECT
以下のことを行うための特権を付与します。

- 表またはビューから列を検索する特権。
- 表にビューを作成する特権。
- 表またはビューに対して EXPORT ユーティリティーを実行する特権。

UPDATE
ON 文節で指定される表または更新可能なビューに対して UPDATE ステートメントを使用する特権を付与します。

ステートメントの許可 ID が以下のいずれかを持っている場合、

- DBADM 権限または SYSADM 権限
- その表またはビューに対する CONTROL 特権
- その表またはビューに対する UPDATE WITH GRANT OPTION

特権を与えられたユーザーは、付与者が GRANT 特権を持っている表またはビューのすべての更新可能な列を更新できます (ALTER TABLE ステートメントを使用して後で追加された列であっても)。そうでない場合、与えられる特権はステートメントの許可 ID が指定の表またはビューに対して持っているすべての列の付与可能な UPDATE 特権です。

UPDATE (*column-name*,...)
列のリストに指定した列だけを、 UPDATE ステートメントを使用して更新する特権を付与します。各 *column-name* は、 ON 文節で指定される表またはビューの列を指定する非修飾名でなければなりません。 型付き表、型付きビュー、またはニックネームに対する列レベルの UPDATE 特権は付与できません (SQLSTATE 42997)。

ON TABLE *table-name* または *view-name* または *nickname*
特権を付与する表、ビュー、またはニックネームを指定します。

作動不能なビューまたは作動不能なマテリアライズ照会表に対する特権を付与することはできません (SQLSTATE 51024)。宣言済み一時表に対する特権を付与することはできません (SQLSTATE 42995)。

TO
特権を誰に与えるかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。ステートメントを発行しているユーザーの許可 ID への付与に関する、以前の制約はなくなりました。

グループに付与された特権は、次のような許可検査では使用されません。

- パッケージ内の静的 DML ステートメントに対する許可検査
- CREATE VIEW ステートメントの処理過程での基本表に対する許可検査
- マテリアライズ照会表の CREATE TABLE ステートメントの処理過程での基本表に対する許可検査

DB2 Universal Database の場合、グループに付与される表特権は、動的に準備されるステートメントにのみ適用されます。たとえば、PROJECT 表に対する INSERT 特権がグループ D204 に与えられ、 UBIQUITY (D204 のメンバー) には与えられていない場合、 UBIQUITY は以下のステートメントを出すことができます。

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

ここで、ストリングの内容は次のとおりです。

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

ただし、以下のステートメントを含むプログラムをプリコンパイルまたはバインドすることはできません。

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

PUBLIC
すべてのユーザーに特権を付与します。静的 SQL ステートメントおよび CREATE VIEW ステートメントに対して PUBLIC に与えられた特権の使用に関する、以前の制約は除かれました。

WITH GRANT OPTION
指定した *authorization-name* に対し、特権を他のユーザーに与えることを許可します。

指定した特権に CONTROL が含まれる場合、 WITH GRANT OPTION は CONTROL を除くすべての適用可能な特権に適用されます (SQLSTATE 01516)。

規則:

- USER も GROUP も指定しない場合には、
  - *authorization-name* がオペレーティング・システムで GROUP としてのみ定義されている場合には、 GROUP であると見なされます。
  - *authorization-name* がオペレーティング・システムで USER としてのみ定義されているか、未定義の場合には、USER であると見なされます。
  - オペレーティング・システムで *authorization-name* が両方として定義されている場合、エラー (SQLSTATE 56092) が戻されます。
- 一般に、GRANT ステートメントはステートメントの許可 ID が与えることを許されている特権の GRANT のみを処理し、 1 つまたは複数の特権が与えられな

かった場合は警告 (SQLSTATE 01007) を戻します。どのような特権も与えられなかった場合は、エラーが戻されます (SQLSTATE 42501)。ステートメントの処理に使用されるパッケージが、 LANGLEVEL を SQL92E または MIA に設定してプリコンパイルされていた場合、付与者が GRANT 操作の対象に対して特権を持っていない場合以外は警告が戻されます (SQLSTATE 01007)。 CONTROL 特権を指定する場合、特権が与えられるのは、ステートメントの許可 ID に SYSADM または DBADM 権限を持っているときだけです (SQLSTATE 42501)。

注:

- ***互換性***
  - DB2 UDB for OS/390 and z/OS との互換性:
    - 以下の構文は許容されますが、無視されます。
      - PUBLIC AT ALL LOCATIONS
- 特権は表階層のどのレベルにも個別に付与できます。スーパー表に対する特権を持つユーザーは、その副表にも影響を及ぼす場合があります。たとえば、スーパー表 *T* に対する UPDATE 特権は持っているものの、そのスーパー表の副表である *S* に対する UPDATE 特権は持っていないユーザーが *T* を指定して更新を要求すると、 *T* の副表 *S* 内にある行に対して変更を要求したかのようになる場合があります。ユーザーが副表を直接操作できるのは、その副表に対して必要な特権を持っている場合だけです。
- ニックネーム特権を付与しても、データ・ソース・オブジェクト (表またはビュー) の特権に与える影響はありません。 通常、データ・ソースの特権は、データの検索を試行する際、ニックネームが参照する表またはビューで必要とされます。

例:

例 1: 表 WESTERN_CR に対するすべての特権を PUBLIC に与えます。

```
GRANT ALL ON WESTERN_CR
   TO PUBLIC
```

例 2: ユーザー PHIL と CLAIRE が CALENDAR 表を読み取り、また新しい項目を挿入することができるように、 CALENDAR 表に対する適切な特権を付与します。既存の項目の変更や削除を行うことは許可しません。

```
GRANT SELECT, INSERT ON CALENDAR
   TO USER  PHIL, USER CLAIRE
```

例 3: COUNCIL 表に対するすべての特権と、その特権を他のユーザーに与える特権をユーザー FRANK に付与します。

```
GRANT ALL ON COUNCIL
   TO USER FRANK WITH GRANT OPTION
```

例 4: 表 CORPDATA.EMPLOYEE に対する SELECT 特権を JOHN という名前のユーザーに付与します。 JOHN と呼ばれるユーザーは存在していますが、JOHN と呼ばれるグループは存在していません。

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

または

```
GRANT SELECT
  ON CORPDATA.EMPLOYEE TO USER JOHN
```

例 5: 表 CORPDATA.EMPLOYEE に対する SELECT 特権を JOHN という名前のグループに付与します。 JOHN と呼ばれるグループは存在していますが、 JOHN と呼ばれるユーザーは存在していません。

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

または

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN
```

例 6: D024 という名前のグループと、D024 という名前のユーザーの両方に、表 T1 に対する INSERT および SELECT 特権を付与します。

```
GRANT INSERT, SELECT ON TABLE T1
  TO GROUP D024, USER D024
```

この場合、D024 グループのメンバーとユーザー D024 はいずれも、表 T1 に対する INSERT と SELECT を使用できるようになります。また、SYSCAT.TABAUTH カタログ・ビューには 2 つの行が追加されることになります。

例 7: ユーザー FRANK に、CALENDAR 表に対する INSERT、SELECT、および CONTROL 特権を付与します。 FRANK は特権を他のユーザーに渡すことが可能である必要があります。

```
GRANT CONTROL ON TABLE CALENDAR
  TO FRANK WITH GRANT OPTION
```

このステートメントの結果、 CONTROL に WITH GRANT OPTION が与えられなかったことを示す警告 (SQLSTATE 01516) が出されます。 Frank は、INSERT と SELECT を含む CALENDAR に対する特権を必要に応じて付与することが可能になります。 FRANK は、SYSADM 権限または DBADM 権限を持っていない限り、他のユーザーに CALENDAR に対する CONTROL 特権を付与することはできません。

例 8: ユーザー JON が、索引のない Oracle 表のニックネームを作成しました。ニックネームは ORAREM1 です。その後、Oracle DBA がこの表の索引を定義しました。そのため、ユーザー SHAWN は、さらに効率よく表にアクセスするための戦略をオプティマイザーが立てられるようにするため、この索引の存在を DB2 に認識させたいと思っています。 SHAWN は、ORAREM1 の索引指定を作成することにより、索引を DB2 に認識させることができます。 SHAWN が索引指定を作成できるようにするため、このニックネームに対する索引特権を SHAWN に与えます。

```
GRANT INDEX ON NICKNAME ORAREM1
  TO USER SHAWN
```

関連資料:

- 45 ページの『ALTER TABLE』
- 580 ページの『GRANT (データベース権限)』
- 584 ページの『GRANT (索引特権)』
- 586 ページの『GRANT (パッケージ特権)』
- 593 ページの『GRANT (スキーマ特権)』
- 599 ページの『GRANT (サーバー特権)』
- 601 ページの『GRANT (表スペース特権)』

- 596 ページの『GRANT (シーケンス特権)』
- 589 ページの『GRANT (ルーチン特権)』

関連サンプル:

- 『tbpriv.sqc -- How to grant, display, and revoke privileges (C)』
- 『tbpriv.sqC -- How to grant, display, and revoke privileges (C++)』
- 『TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)』
- 『TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)』

IF

IF ステートメントは、条件の評価に基づいて実行パスを選択します。

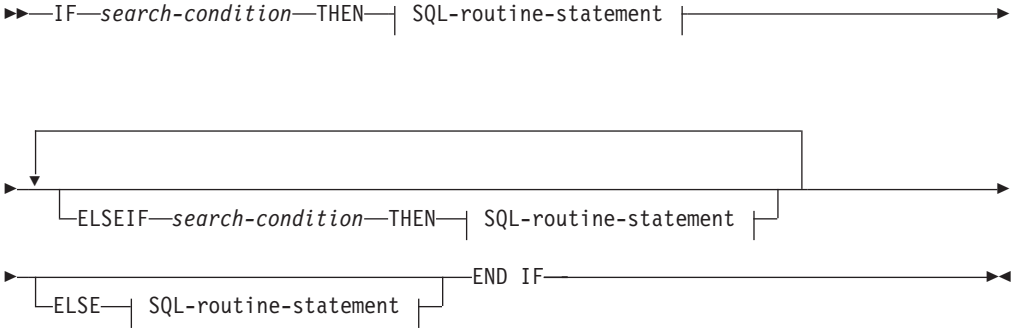
呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

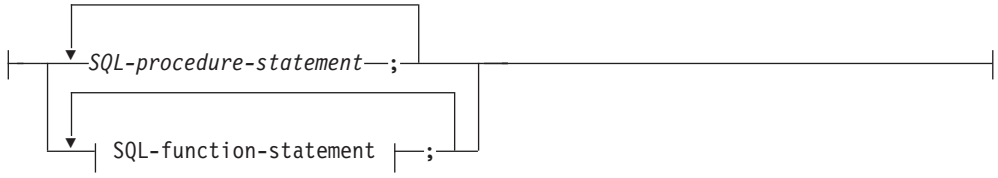
許可:

IF ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 IF ステートメントに組み込まれている SQL ステートメントおよび検索条件を呼び出すために必要な特権がなければなりません。

構文:



SQL-routine-statement:



説明:

search-condition
SQL ステートメントを呼び出す条件を指定します。条件が不明または偽の場合、条件が真になるか、または処理が ELSE 文節に到達するまで、処理は次の検索条件に継続されます。

SQL-procedure-statement
前の *search-condition* が真の場合に呼び出されるステートメントを指定します。*SQL-procedure-statement* は、SQL プロシージャーのコンテキスト内でのみ使用できます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

SQL-function-statement
前の *search-condition* が真の場合に呼び出されるステートメントを指定します。

SQL-function-statement は、SQL 関数または SQL メソッドのコンテキスト内でのみ使用できます。 FOR ステートメントの説明にある SQL-function-statement の項目を参照してください。

例:

以下の SQL プロシージャーでは、 2 つの IN パラメーター (従業員番号 *employee_number* および従業員評定 *rating*) を使用します。 *rating* の値によっては、 employee 表の salary および bonus 列が、新しい値に更新されます。

```
CREATE PROCEDURE UPDATE_SALARY_IF
  (IN employee_number CHAR(6), INOUT rating SMALLINT)
  LANGUAGE SQL
  BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
      SET rating = -1;
    IF rating = 1
      THEN UPDATE employee
      SET salary = salary * 1.10, bonus = 1000
      WHERE empno = employee_number;
    ELSEIF rating = 2
      THEN UPDATE employee
      SET salary = salary * 1.05, bonus = 500
      WHERE empno = employee_number;
    ELSE UPDATE employee
      SET salary = salary * 1.03, bonus = 0
      WHERE empno = employee_number;
      END IF;
   END
```

関連資料:

- 140 ページの『コンパウンド SQL (プロシージャー)』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』

INCLUDE

INCLUDE ステートメントは、宣言をソース・プログラムに挿入します。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、実行可能ステートメントではありません。

許可:

必要ありません。

構文:

説明:

SQLCA
SQL 連絡域 (SQLCA) の記述を組み込むことを指定します。

SQLDA
SQL 記述子域 (SQLDA) の記述を組み込むことを指定します。

name
プリコンパイルするソース・プログラムに組み込むテキストが入っている外部ファイルを指定します。 ファイル名拡張子のない SQL ID、または一重引用符で囲んだ (' ') リテラルを指定することができます。 SQL ID は、そのファイル名拡張子として、プリコンパイルするソース・ファイルのファイル名拡張子が想定されます。 引用符で囲んだリテラルにファイル名拡張子の指定がない場合には、拡張子はないものと想定されます。

注:

- プログラムをプリコンパイルすると、 INCLUDE ステートメントはソース・ステートメントで置き換えられます。 したがって、ソース・プログラム中での INCLUDE ステートメントの位置は、展開結果のソース・ステートメントがコンパイラーに受け入れられる位置でなければなりません。
- 外部ソース・ファイルは、 *name* に指定されているホスト言語で作成しなければなりません。 名前が 18 文字を超える場合、または SQL ID としては使用できない文字が含まれている場合は、一重引用符で囲む必要があります。 INCLUDE *name* ステートメントは、ネスト可能ですが、循環が発生してはなりません (たとえば、A と B というモジュールがあり、 A の中に INCLUDE *name* ステートメントが含まれている場合、 A が B を呼び出し、その B が A を呼び出すようにするのは有効ではありません)。
- LANGLEVEL プリコンパイル・オプションに SQL92E 値が指定されている場合、 INCLUDE SQLCA を指定してはなりません。 SQLSTATE と SQLCODE 変数は、ホスト変数宣言セクションで定義できます。

例:

C プログラムに SQLCA を組み込みます。

```
EXEC SQL INCLUDE SQLCA;

EXEC SQL  DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
    WHERE ADMRDEPT = 'A00';

EXEC SQL  OPEN C1;

while (SQLCODE==0) {
    EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;

(結果の印刷)

}

EXEC SQL  CLOSE C1;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『SQLDA (SQL 記述子域)』
- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』

関連サンプル:

- 『dbcfg.sqc -- Configure database and database manager configuration parameters (C)』
- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』
- 『dtformat.sqc -- Load and import data format extensions (C)』
- 『tbcreate.sqc -- How to create and drop tables (C)』
- 『tbident.sqc -- How to use identity columns (C)』
- 『dbcfg.sqC -- Configure database and database manager configuration parameters (C++)』
- 『tbcreate.sqC -- How to create and drop tables (C++)』

INSERT

INSERT ステートメントは、表、ニックネーム、またはビュー、あるいは指定された全選択の基礎になる表、ニックネーム、またはビューに、行を挿入します。 行をニックネームに挿入することは、その行をそのニックネームが参照するデータ・ソース・オブジェクトに挿入することでもあります。このビューに対する挿入操作用に INSTEAD OF トリガーが定義されていない場合、行をビューに挿入することは、その行をそのビューの基本となる表に挿入することでもあります。このようなトリガーが定義されている場合は、トリガーが代わりに実行されます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントを実行するには、ステートメントの許可 ID に、以下の特権の少なくとも 1 つが含まれている必要があります。

- 更新する行を含む表、ビュー、またはニックネームに対する INSERT 特権
- 更新する行を含む表、ビュー、またはニックネームに対する CONTROL 特権
- SYSADM または DBADM 権限

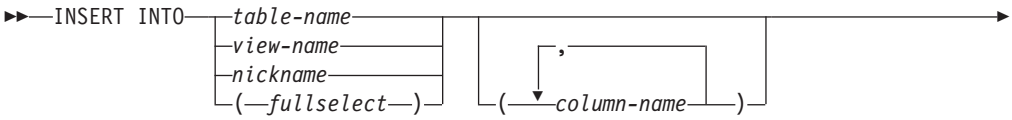
さらに、ステートメントの許可 ID には、 INSERT ステートメントで使用する全選択で参照される表、ビュー、またはニックネームのそれぞれに対して、以下の特権の少なくとも 1 つが含まれている必要があります。

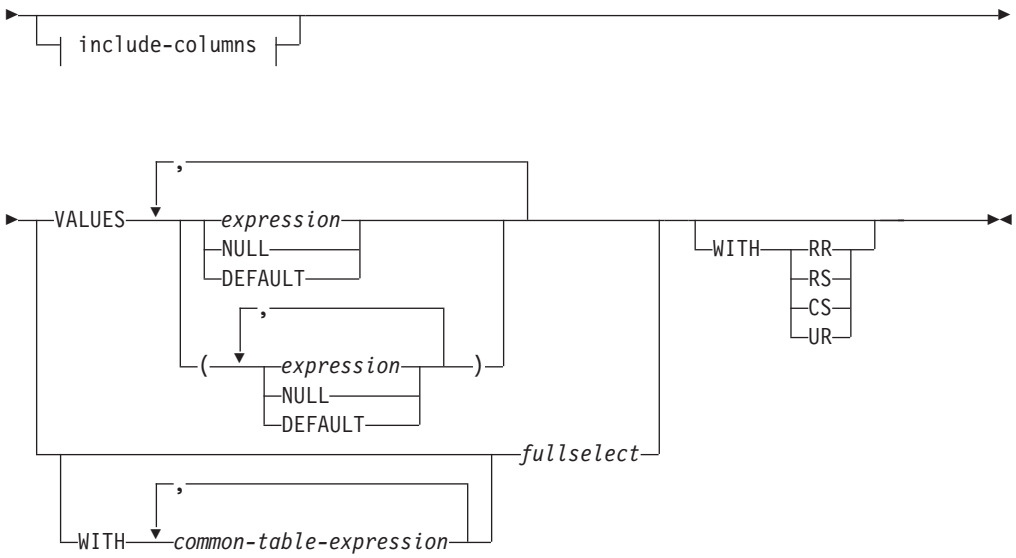
- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

静的 INSERT ステートメントの場合、GROUP 特権はチェックされません。

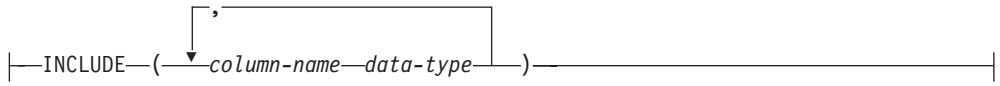
挿入操作の対象がニックネームの場合は、データ・ソースでステートメントが実行されないうちは、そのデータ・ソース上のオブジェクトに対する特権は考慮されません。この時点で、データ・ソースに接続するために使用される許可 ID は、データ・ソースのオブジェクトに対して操作を行うのに必要な特権を持っている必要があります。ステートメントの許可 ID は、データ・ソースの別の許可 ID へマップできます。

構文:





include-columns:



説明:

INTO *table-name*、*view-name*、 *nickname*、**または** *(fullselect)*
挿入操作の対象のオブジェクトを指定します。 *table-name* (表名)、*view-name* (ビュー名)、または *nickname* (ニックネーム) は、それぞれアプリケーション・サーバーに存在する表、ビュー、またはニックネームを指定していなければならず、カタログ表、システムで保守されているマテリアライズ照会表、カタログ表のビュー、または読み取り専用のビュー (対象となるビューに対する挿入操作用に INSTEAD OF トリガーが定義されていない場合) は指定できません。行をニックネームに挿入することは、その行をそのニックネームが参照するデータ・ソース・オブジェクトに挿入することでもあります。

挿入操作のオブジェクトが全選択である場合、全選択は、 CREATE VIEW ステートメントの説明の『注』にある『挿入可能ビュー』という項目で定義されているように、挿入可能になっている必要があります。

このビューに対する挿入操作用に INSTEAD OF トリガーがない場合、ビューの以下のような列には、値を挿入することはできません。

- 定数、式、またはスカラー関数から得られる列。
- そのビューの他の列と同じ基本表の列から得られる列。

挿入操作の対象となるビューにこのような列がある場合は、列名のリストを指定する必要があり、そのリストに上記の列を指定してはなりません。

行が基礎となる基本表のうち 1 つだけのチェック制約を満たしている場合、UNION ALL を使用して定義されているビューまたは全選択にその行を挿入できます。行が複数の表のチェック制約を満たしている場合や、どの表のチェック制約も満たしていない場合は、エラーが戻されます (SQLSTATE 23513)。

(*column-name***,...)**
挿入する値の対象となる列を、各 *column-name* に指定します。それぞれの名前は、表またはビューの列、あるいは全選択の列を指定する非修飾名でなければなりません。同じ列を重複して指定することはできません。挿入値を受け入れることのできない列 (たとえば、式を基にした列) を指定することはできません。

列のリストを省略すると、その表またはビューのすべての列、あるいは全選択の選択リストのすべての項目を左から右に指定したリストが暗黙に指定されます。このリストはステートメントが準備される時点で確立されます。したがって、ステートメントの準備後に表に追加された列は含まれません。

include-columns
全選択の FROM 文節にネストされているとき、 *table-name* や *view-name* などの列と一緒に INSERT ステートメントの中間結果表に組み込まれている列セットを指定します。 *include-columns* は、 *table-name* や *view-name* で指定されている列のリストの最後に付加されます。

INCLUDE
INSERT ステートメントの中間結果表に組み込まれる列のリストを指定します。この文節は、INSERT ステートメントが全選択の FROM 文節にネストされている場合にのみ指定できます。

column-name
INSERT ステートメントの中間結果表の列を指定します。名前は、他の組み込み列や、 *table-name* または *view-name* の列と同じ名前であってはなりません (SQLSTATE 42711)。

data-type
組み込み列のデータ・タイプを指定します。データ・タイプは、CREATE TABLE ステートメントでサポートされているものでなければなりません。

VALUES
挿入したい 1 行または複数行の値を、この後に指定します。

ホスト変数を指定する場合、それらのホスト変数は、ホスト変数の宣言規則に従ってそのプログラムで記述されていなければなりません。

各行の値の数は、暗黙的または明示的な列のリスト、および INCLUDE 文節で識別された列にある名前の数と一致していなければなりません。最初の値はリストの最初の列に挿入され、2 番目の値は 2 番目の列に挿入されます。以下同様です。

expression
『Expressions』で定義されている *expression* (式) を使用できます。

NULL
NULL 値を指定します。これは NULL 可能の列に対してのみ指定できます。

DEFAULT
デフォルト値を使用することを指定します。 DEFAULT を指定したときに使用される値は、該当の列がどのように定義されているかによって決まります。次のとおりです。

- 式に基づいて生成される列として列が定義されている場合は、その式に基づいた列の値がシステムによって生成されます。

- IDENTITY 文節が使用されている場合は、データベース・マネージャーによって値が生成されます。
- WITH DEFAULT 文節が使用されている場合は、その列に対して定義された値が挿入されます (『CREATE TABLE』の *default-clause* を参照してください)。
- NOT NULL 文節が使用されているが GENERATED 文節は使用されていない場合、または WITH DEFAULT 文節は使用されていないか DEFAULT NULL が使用されている場合は、その列に対して DEFAULT キーワードを指定することができません (SQLSTATE 23502)。
- ニックネームに挿入する場合、データ・ソースが照会言語構文中の DEFAULT キーワードをサポートしている場合に限り、 DEFAULT キーワードはそのデータ・ソースに対して INSERT ステートメントをパススルーします。

WITH *common-table-expression*
後続の全選択で使用する共通表式を定義します。

fullselect
新しい行の集合を、全選択の結果表の形式で指定します。行の数は、1 つか、複数か、またはゼロのいずれかです。 結果表が空の場合、SQLCODE は +100 に設定され、 SQLSTATE は '02000' に設定されます。

INSERT の基本オブジェクトおよび全選択の基本オブジェクトまたは全選択の副照会のいずれかが同一の表である場合、行挿入の前に、全選択が完全に評価されます。

結果表の列の数は、列リストの名前の数と同じでなければなりません。 結果の最初の列の値はリストの最初の列に挿入され、 2 番目の値は 2 番目の列に挿入されます。以下同様です。

WITH
fullselect が実行される分離レベルを指定します。

RR
反復可能読み取り

RS
読み取り固定

CS
カーソル固定

UR
非コミット読み取り

ステートメントのデフォルト分離レベルは、ステートメントがバインドされているパッケージの分離レベルです。

規則:

- ***トリガー:*** INSERT ステートメントによってトリガーの実行が引き起こされる場合があります。トリガーが他のステートメントの実行を引き起こす場合や、挿入値に起因するエラーが発生する場合があります。ビューに挿入操作を行うと INSTEAD OF トリガーが起動する場合は、そのトリガーによって実行される更新

に対して妥当性、参照保全、および制約がチェックされます。トリガーを起動させたビューやその基本表に対するチェックは行われません。

- ***デフォルト値:*** 列リストにない列に挿入される値は、列のデフォルト値または NULL 値のいずれかになります。 NULL 値が許されない列で NOT NULL WITH DEFAULT として定義されていない列は、列リストに含める必要があります。同様に、ビューへの挿入の場合、基本表の列で、ビューにはない列に挿入される値は、その列のデフォルト値か、または NULL 値のいずれかになります。したがって、基本表に存在し、ビューにはない列はすべて、デフォルト値があるか、または NULL 可能であるかのいずれかでなければなりません。生成される列が GENERATED ALWAYS 文節で定義されている場合は、 DEFAULT 以外の値を挿入することはできません (SQLSTATE 428C9)。
- ***長さ:*** 列の挿入値が数値の場合、列はその数の整数部分を入れる容量を持つ数値列でなければなりません。 列の挿入値がストリングの場合、列は、長さ属性がそのストリングの長さ以上である列であるか、またはストリングが日付、時刻、またはタイム・スタンプを表す場合は、日付/時刻列でなければなりません。
- ***割り当て:*** 挿入値は、特定の割り当ての規則に従って列に割り当てられます。
- ***妥当性:*** 指定された表または指定されたビューの基本表に 1 つまたは複数のユニーク索引がある場合、表に挿入される各行は、それらの索引の制約に適合していなければなりません。 その定義に WITH CHECK OPTION を伴うビューが指定された場合、そのビューに挿入する各行は、そのビューの定義に適合していなければなりません。 この状況に関連する規則については、『CREATE VIEW』 を参照してください。
- ***参照保全:*** 表に対して定義されている制約ごとに、外部キーの挿入値のうち NULL 以外の値は、それぞれ親表の主キーの値に等しくなければなりません。
- ***チェック制約:*** 挿入値は、表に定義されているチェック制約のチェック条件を満たしていなければなりません。チェック制約が定義されている表に対する INSERT では、挿入される各行ごとに一度、制約条件が評価されます。
- ***データ・リンク:*** DATALINK 値を含む挿入ステートメントは、該当するファイルに URL 値 (空ストリングまたはブランクを除く) が組み込まれていて、なおかつその列が FILE LINK CONTROL として定義されている場合、そのファイルへリンクしようとします。 DATALINK 値にエラーがあるか、ファイルへのリンクがエラーになった場合、挿入は失敗します (SQLSTATE 428D1 または 57050)。

注:

- INSERT ステートメントの実行後、SQLCA の SQLERRD の 3 番目の変数 (SQLERRD(3)) の値は、挿入操作に渡された行の数を示します。 SQL プロシージャー・ステートメントでは、値は GET DIAGNOSTICS ステートメントの ROW_COUNT 変数を使用して検索できます。 SQLERRD(5) には、トリガーによって実行された挿入、更新、および削除操作の数が入られます。
- 適切なロックがすでに存在しない限り、 1 つまたは複数の排他ロックが正常な INSERT ステートメントの実行時に獲得されます。 それらのロックが解放されるまで、挿入された行は以下によってのみアクセス可能です。
  - その挿入を行ったアプリケーション・プロセス
  - 読み取り専用カーソル、SELECT INTO ステートメント、または副照会で使用されている副選択を介して分離レベル UR を使用する他のアプリケーション・プロセス

- ロッキングについての詳細は、COMMIT、ROLLBACK、および LOCK TABLE のステートメントの説明を参照してください。
- パーティション・データベースに対してアプリケーションが実行されており、INSERT BUF オプションを指定してアプリケーションがバインドされている場合、 EXECUTE IMMEDIATE を使用して処理されない VALUES を伴う INSERT はバッファーに入れられます。 DB2 は、このような INSERT ステートメントがアプリケーションの論理においてループ中で処理されるものと想定します。ステートメントをその完了まで実行する代わりに、 DB2 は新しい行の値を 1 つまたは複数のバッファーに入れることを試みます。 その結果として、表に対する行の実際の挿入は後で行われ、アプリケーションの INSERT の論理とは非同期になります。 この非同期の挿入が原因で、アプリケーションでその INSERT に続く他の SQL ステートメントに INSERT が戻されることに関連してエラーが生じる場合がある点に注意してください。

  この方法は、INSERT のパフォーマンスを大幅に向上させる可能性を持っていますが、エラー処理が非同期であるために、クリーン・データに対して使用するのが最適です。
- ID 列が含まれている表に行が挿入されると、DB2 は ID 列の値を生成します。
  - GENERATED ALWAYS の ID 列に対しては、常に DB2 が値を生成します。
  - GENERATED BY DEFAULT 列に対しては、値が明示的に指定されていない (VALUES 文節や副選択によって) 場合にのみ、DB2 が値を生成します。

  DB2 は、その ID 列に対して START WITH で指定された値を最初の値として生成します。
- ユーザー定義特殊タイプの ID 列に値が挿入されるときは、まずすべての計算がソース・タイプで行われます。そして計算された値は、値が列に実際に割り当てられる前に、ソース・タイプから定義された特殊タイプにキャストされます。(計算に先立って、元の値がソース・タイプにキャストされることはありません。)
- GENERATED ALWAYS の ID 列に挿入するときは、常に DB2 がその列の値を生成します。挿入の際にユーザーが値を指定することはできません。列のリストに GENERATED ALWAYS という ID 列がリストされている INSERT ステートメントで、 VALUES 文節に DEFAULT 以外の値が指定された場合は、エラーが発生します (SQLSTATE 428C9)。

  たとえば、EMPID という列が GENERATED ALWAYS の ID 列として定義されているとします。そこで、次のコマンドを入力します。

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
  ```

  すると、エラーが戻されます。
- GENERATED BY DEFAULT 列に挿入するときは、 DB2 では VALUES 文節で、または副選択からその列に実際の値を指定することができます。ただし、VALUES 文節に値を指定するとき、DB2 は指定された値を一切検査しません。したがって、指定された値が必ずユニークなものとなるよう、ID 列でユニーク索引を作成する必要があります。

列のリストを指定せずに、GENERATED BY DEFAULT の ID 列のある表に挿入するときは、ID 列の値を表す DEFAULT キーワードを VALUES 文節で指定することができます。 DB2 は、指定された値を ID 列に生成します。

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
  VALUES (DEFAULT, :hv_name, :hv_addr)
```

この例では、EMPID が ID 列として定義され、この列に挿入される値は DB2 によって生成されます。

- 副選択を使用して ID 列に値を挿入する場合も、 VALUES 文節を使用する場合と同様の規則が適用されます。 ID 列に値を指定できるのは、ID 列が GENERATED BY DEFAULT として定義されている場合だけです。

  たとえば、同じ定義を持つ、T1 と T2 という 2 つの表があるとします。これらの表にはいずれも列 *intcol1* および *identcol2* (これらはどちらもタイプ INTEGER の列で、 2 番目の列には識別属性がある) が含まれています。次のような挿入について考慮します。

  ```
  INSERT INTO T2
     SELECT *
     FROM T1
  ```

  この例は、論理的には以下と同じ意味になります。

  ```
  INSERT INTO T2 (intcol1,identcol2)
     SELECT intcol1, identcol2
     FROM T1
  ```

  このどちらの場合においても、 INSERT ステートメントには T2 の ID 列を表す明示的な値が指定されています。このように明示的な値を指定した場合は、ID 列の値を指定することができます。しかしこれは、T2 の ID 列が GENERATED BY DEFAULT として定義されている場合に限られます。それ以外の場合は、ID 列に値を指定するとエラーが戻されます (SQLSTATE 428C9)。

  表に GENERATED ALWAYS の ID 列として定義された列がある場合でも、同じ定義を持つ表から、他のすべての列に伝搬することができます。たとえば、先に取り上げた例の表 T1 と T2 であれば、 T1 と T2 に含まれている intcol1 の値を以下の SQL で伝搬することができます。

  ```
  INSERT INTO T2 (intcol1)
     SELECT intcol1
     FROM T1
  ```

  なお、identcol2 は列のリストで指定されていないため、この列にはデフォルトの (生成) 値が使用されます。

- GENERATED ALWAYS の ID 列として定義された単一列の表に行を挿入するときは、 VALUES 文節に DEFAULT キーワードを指定することができます。この場合は、アプリケーションによって表に提供される値はありません。ID 列の値は DB2 によって生成されます。

  ```
  INSERT INTO IDTABLE
     VALUES(DEFAULT)
  ```

  識別属性をもつ列が含まれているこの同じ単一列の表に、 1 つの INSERT ステートメントを使用して複数の行を挿入するとします。その場合は、次のような INSERT ステートメントを使用できます。

```
INSERT INTO IDTABLE
  VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
```

- DB2 によって生成される ID 列の値は一時的なものです。次に値が必要な時には、また新しい値が生成されます。これは、ID 列に関連した INSERT ステートメントが失敗した場合やロールバックされた場合も同様です。

  たとえば、ID 列にユニーク索引が作成されていると想定します。 ID 列に対する値の生成で重複キーの違反が検出されると、エラーが戻され (SQLSTATE 23505)、その ID 列に対して生成される値は破棄されることになります。このエラーが生じる可能性があるのは、ID 列が GENERATED BY DEFAULT として定義されており、システムが新しい値を生成しようとしたものの、ユーザーが以前の INSERT ステートメントで ID 列に明示的な値を指定していた場合です。このような場合は、同じ INSERT ステートメントをもう一度発行すればうまくいきます。 DB2 は ID 列に対して次の値を生成します。次に生成された値が重複していなければ、INSERT ステートメントは正常に完了します。
- ID 列に対して生成される値が ID 列の最大値 (降順で値が生成される場合は最小値) を超えると、エラーが発生します (SQLSTATE 23522)。この場合、ユーザーは、表を DROP して、より広い範囲を持つ ID 列 (より広い値の範囲で、列のデータ・タイプを変更したり、値を増分したりできるようにするため) を指定して、新しい表の CREATE を実行する必要があります。

  たとえば、データ・タイプ SMALLINT で定義されている ID 列があり、この列で割り当てられる値を使い切ってしまったとします。 ID の列を INTEGER として再定義するには、データをアンロードし、表をドロップし、新しい定義の列で表を再作成して、それからデータを再ロードしなければなりません。そして、表を再定義する際は、 DB2 によって生成される次の値が元の順序で次の値になるように、ID 列の START WITH 値を指定しなければなりません。最後の値を確認するには、データをアンロードする前に、ID 列の MAX (昇順で値を生成している場合) または MIN (降順で値を生成している場合) を使用して照会を発行します。

例:

例 1: DEPARTMENT 表に、以下の指定で新しい部門を挿入します。

- 部門番号 (DEPTNO) は 'E31'
- 部門名 (DEPTNAME) は 'ARCHITECTURE'
- その管理者の社員番号 (MGRNO) は '00390'
- 報告先の部門 (ADMRDEPT) は 'E01'

```
INSERT INTO DEPARTMENT
  VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

例 2: 例 1 と同様に DEPARTMENT 表に新しい部門を挿入しますが、新しい部門に管理者は割り当てません。

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
  VALUES ('E31', 'ARCHITECTURE', 'E01')
```

例 3: 例 2 と同様の DEPARTMENT 表に 2 つの新しい部門を 1 つのステートメントを使用して挿入しますが、新しい部門に管理者は割り当てません。

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
  VALUES ('B11', 'PURCHASING', 'B01'),
         ('E41', 'DATABASE ADMINISTRATION', 'E01')
```

例 4: EMP_ACT 表と同じ列を持つ一時表 MA_EMP_ACT を作成します。EMP_ACT 表から、 'MA' で始まるプロジェクト番号 (PROJNO) を持つ行を MA_EMP_ACT 表にロードします。

```
CREATE TABLE MA_EMP_ACT
         ( EMPNO CHAR(6)  NOT NULL,
           PROJNO CHAR(6)  NOT NULL,
           ACTNO SMALLINT  NOT NULL,
           EMPTIME DEC(5,2),
           EMSTDATE DATE,
           EMENDATE  DATE )
INSERT INTO MA_EMP_ACT
  SELECT * FROM EMP_ACT
    WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

例 5: C プログラムのステートメントを使用して、 PROJECT 表に骨組みとなるプロジェクトを追加します。プロジェクト番号 (PROJNO)、プロジェクト名 (PROJNAME)、部門番号 (DEPTNO)、および責任者 (RESPEMP) は、ホスト変数から入手します。プロジェクトの開始日 (PRSTDATE) として、現在の日付を使用します。表のその他の列には、NULL (NULL) 値を割り当てます。

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
  VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```

例 6: SELECT ステートメントで、INSERT ステートメントを *data-change-table-reference* として指定します。 VALUE 文節で値が指定されている組み込み列を余分に定義し、それを、挿入される行の配列用の列として使用します。

```
SELECT inorder.ordernum
  FROM (INSERT INTO orders(custno)INCLUDE (insertnum integer)
    VALUES(:cnum1, 1), (:cnum2, 2)) InsertedOrders
  ORDER BY insertnum;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『式』
- 347 ページの『CREATE TABLE』
- 474 ページの『CREATE VIEW』
- *SQL リファレンス 第 1 巻* の『SQL 照会』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

関連サンプル:

- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『tbident.sqc -- How to use identity columns (C)』
- 『tbmod.sqc -- How to modify table data (C)』
- 『tbtrig.sqc -- How to use a trigger on a table (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『tbmod.sqC -- How to modify table data (C++)』
- 『tbtrig.sqC -- How to use a trigger on a table (C++)』
- 『DtLob.java -- How to use LOB data type (JDBC)』

- 『TbIdent.java -- How to use Identity Columns (JDBC)』
- 『TbMod.java -- How to modify table data (JDBC)』
- 『TbTrig.java -- How to use triggers (JDBC)』
- 『TbIdent.sqlj -- How to use Identity Columns (SQLj)』
- 『TbMod.sqlj -- How to modify table data (SQLj)』
- 『TbTrig.sqlj -- How to use triggers (SQLj)』
- 『updat.sqb -- How to update, delete and insert table data (MF COBOL)』

ITERATE

ITERATE ステートメントを使用すると、制御のフローがラベル付きループの最初に戻ります。

呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

必要ありません。

構文:

説明:

label
DB2 が制御のフローを渡す先の FOR、LOOP、REPEAT、または WHILE ステートメントのラベルを指定します。

例:

この例では、カーソルを使用して新しい部門の情報を戻します。 *not_found* 条件ハンドラーが呼び出されると、制御のフローがループの外側に渡されます。 *v_dept* の値が 'D11' の場合、 ITERATE ステートメントは制御のフローを LOOP ステートメントの先頭に戻します。それ以外の場合は、新しい行が DEPARTMENT 表に挿入されます。

```
CREATE PROCEDURE ITERATOR()
  LANGUAGE SQL
  BEGIN
    DECLARE v_dept CHAR(3);
    DECLARE v_deptname VARCHAR(29);
    DECLARE v_admdept CHAR(3);
    DECLARE at_end INTEGER DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT deptno, deptname, admrdept
      FROM department
      ORDER BY deptno;
    DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
      OPEN c1;
    ins_loop:
    LOOP
      FETCH c1 INTO v_dept, v_deptname, v_admdept;
      IF at_end = 1 THEN
        LEAVE ins_loop;
      ELSEIF v_dept = 'D11' THEN
        ITERATE ins_loop;
      END IF;
      INSERT INTO department (deptno, deptname, admrdept)
```

```
    VALUES ('NEW', v_deptname, v_admdept);
  END LOOP;
  CLOSE c1;
END
```

LEAVE

LEAVE ステートメントは、プログラム制御をループまたはコンパウンド・ステートメントの外側に移動させます。

呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

必要ありません。

構文:

説明:

label
終了するコンパウンド、FOR、LOOP、REPEAT、または WHILE ステートメントのラベルを指定します。

注:

- LEAVE ステートメントがコンパウンド・ステートメントの外側に制御を移動すると、そのコンパウンド・ステートメント内のすべてのオープン・カーソル (結果セットを戻すのに使用されているカーソルを除く) がクローズされます。

例:

以下の例には、カーソル *c1* のデータを取り出すループが含まれています。 SQL 変数 *at_end* の値がゼロでなければ、 LEAVE ステートメントは制御をループの外側に移動させます。

```
CREATE PROCEDURE LEAVE_LOOP(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE at_end SMALLINT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER for not_found
      SET at_end = 1;
    SET v_counter = 0;
      OPEN c1;
    fetch_loop:
    LOOP
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
      IF at_end <> 0 THEN LEAVE fetch_loop;
      END IF;
      SET v_counter = v_counter + 1;
```

```
  END LOOP fetch_loop;
  SET counter = v_counter;
  CLOSE c1;
END
```

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』

LOCK TABLE

LOCK TABLE ステートメントを使用すると、並行アプリケーション・プロセスが表を変更したり表を使用したりできなくなります。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- その表に対する SELECT 特権
- その表に対する CONTROL 特権
- SYSADM または DBADM 権限

構文:

説明:

table-name **または** *nickname*
該当の表またはニックネームを指定します。 *table-name* は、アプリケーション・サーバーに存在する表を指定していなければならず、カタログ表は指定できません。宣言済み一時表にすることはできません (SQLSTATE 42995)。
table-name が型付き表である場合、その表は表階層のルート表でなければなりません (SQLSTATE 428DR)。ニックネームを指定すると、DB2 は、そのニックネームが参照するデータ・ソースの基礎オブジェクト (つまり表かビュー) をロックします。

IN SHARE MODE
複数の並行するアプリケーション・プロセスが、その表に対して読み取り専用以外の操作を実行するのを防止します。

IN EXCLUSIVE MODE
複数の並行するアプリケーション・プロセスが、その表に対してどのような操作も実行できないようにします。ただし、EXCLUSIVE MODE は、非コミット読み取り分離レベル (UR) で実行している並行アプリケーション・プロセスが、その表に対して読み取り専用操作を実行することは妨げない点に注意してください。

注:

- ロッキングは、複数の操作が並行して行われるのを防止するのに使用されます。すでに適切なロックが存在している場合には、LOCK TABLE ステートメントを実行しても、必ずしもロックが獲得されるとは限りません。並行操作を防止するロックは、少なくともその作業単位の終了まで保持されます。

- パーティション・データベースでは、表ロックはデータベース・パーティション・グループ内の最初のパーティション (最も番号の小さいパーティション) で最初に獲得され、その後他のパーティションで獲得されます。 LOCK TABLE ステートメントが割り込まれると、表は一部のパーティションではロックされ、その他ではロックされないことになります。このような場合、他の LOCK TABLE ステートメントを出してすべてのパーティションに対してロッキングを完了するか、 COMMIT または ROLLBACK ステートメントを出して現在のロッキングを解放します。
- このステートメントは、データベース・パーティション・グループ内のすべてのパーティションに影響を与えます。

例:

表 EMP に対するロックを入手します。他のプログラムは、その表の読み取りや更新を行うことができなくなります。

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

LOOP

LOOP ステートメントは、ステートメント、またはステートメントのグループの実行を繰り返します。

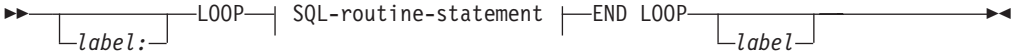
呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

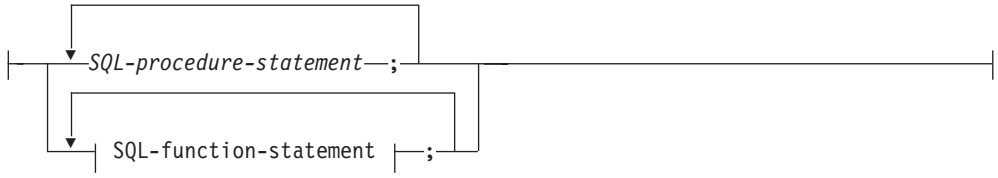
許可:

LOOP ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 LOOP ステートメントに組み込まれている SQL ステートメントを呼び出すために必要な特権がなければなりません。

構文:



SQL-routine-statement:



説明:

label
LOOP ステートメントのラベルを指定します。開始ラベルを指定した場合、そのラベルを LEAVE および ITERATE ステートメントで指定することができます。終了ラベルを指定する場合、一致する開始ラベルを指定しなければなりません。

SQL-procedure-statement
ループ内で呼び出される SQL ステートメントを指定します。
SQL-procedure-statement は、SQL プロシージャーのコンテキスト内でのみ使用できます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

SQL-function-statement
ループ内で呼び出される SQL ステートメントを指定します。
SQL-function-statement は、SQL 関数または SQL メソッドのコンテキスト内でのみ使用できます。 FOR ステートメントの説明にある SQL-function-statement の項目を参照してください。

例:

以下のプロシージャーでは、LOOP ステートメントを使用して、 `employee` 表から値を取り出します。ループが繰り返されるたびに、OUT パラメーター *counter* が増

加し、 *v_midinit* が検査されて、値が単一スペース (' ') でないことを確認します。 *v_midinit* が単一スペースの場合、 LEAVE ステートメントは制御のフローをループの外側に渡します。

```
CREATE PROCEDURE LOOP_UNTIL_SPACE(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET counter = -1;
    OPEN c1;
    fetch_loop:
    LOOP
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
      IF v_midinit = ' ' THEN
        LEAVE fetch_loop;
      END IF;
      SET v_counter = v_counter + 1;
    END LOOP fetch_loop;
    SET counter = v_counter;
    CLOSE c1;
  END
```

関連資料:

- 140 ページの『コンパウンド SQL (プロシージャー)』

MERGE

MERGE ステートメントは、ソース (表参照の結果) からのデータを使ってターゲット (表またはビュー、 あるいは全選択の基礎表またはビュー) を更新します。ターゲット内にあるソースと一致する行を削除または更新するよう指定でき、ターゲットに存在しない行を挿入することができます。ビュー内の行を更新、削除、または挿入すると、ビューの元になっている表の行が更新、削除、または挿入されます。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 挿入操作が指定されている場合、表またはビューに対する INSERT 特権。削除挿入操作が指定されている場合、表またはビューに対する DELETE 特権。更新操作が指定されている場合には、以下のいずれか。
  - 表またはビューに対する UPDATE 特権
  - 更新されるそれぞれの列に対する UPDATE 特権
- 表に対する CONTROL 特権
- SYSADM または DBADM 権限

このステートメントの許可 ID が持つ特権には、少なくとも次のいずれかも含まれている必要があります。

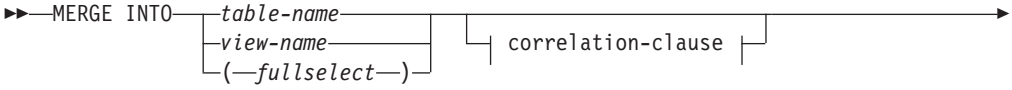
- *table-reference* で指定されたすべての表またはビューに対する SELECT 特権
- *table-reference* で指定された表またはビューに対する CONTROL 特権
- SYSADM または DBADM 権限

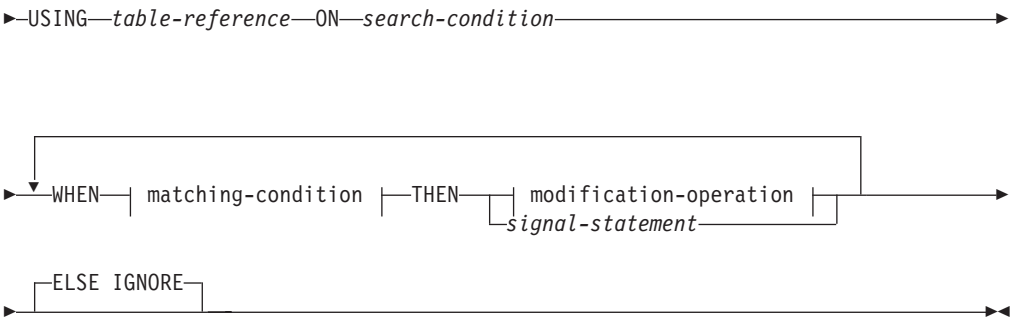
search-condition、*insert-operation*、または *assignment-clause* が副照会を持つ場合、このステートメントの許可 ID が持つ特権には、少なくとも次のいずれかも含まれている必要があります。

- 副照会で指定されたすべての表またはビューに対する SELECT 特権
- 副照会で指定された表またはビューに対する CONTROL 特権
- SYSADM または DBADM 権限

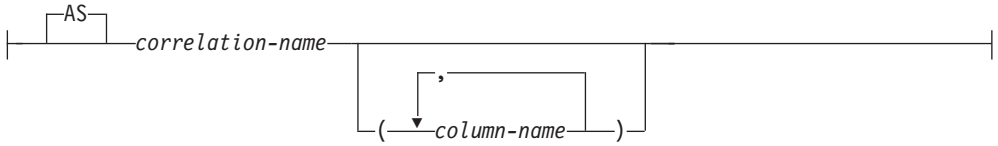
関数を参照する式が指定されている場合、特権セットにはその関数を実行するのに必要な権限が含まれていなければなりません。

構文:

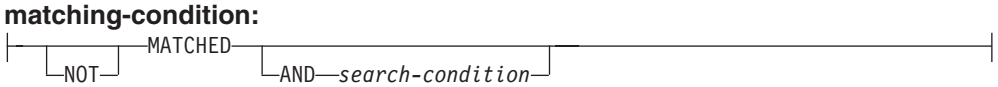




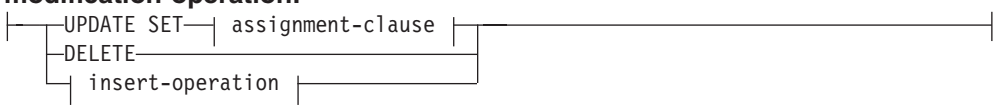
correlation-clause:



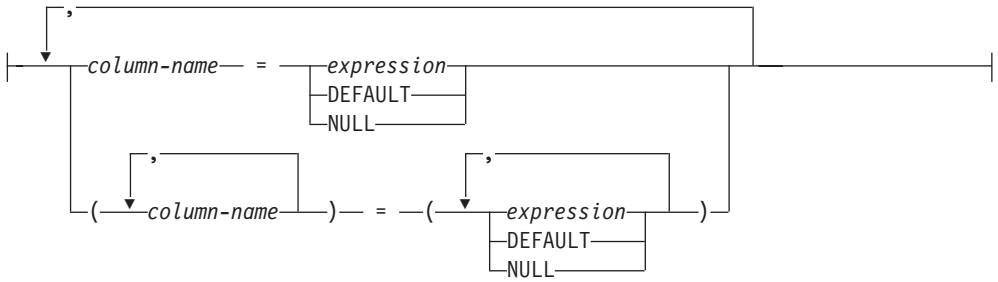
matching-condition:



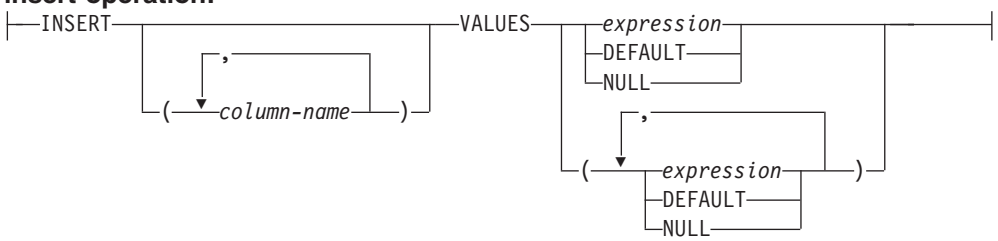
modification-operation:



assignment-clause:



insert-operation:



説明:

table-name、*view-name*、**または** *(fullselect)*

マージの更新、削除、または挿入操作のターゲットを識別します。この名前は、

現在のサーバーに存在する表またはビューを識別する必要があります。ただし、カタログ表、システムで保守されているマテリアライズ照会表、カタログ表のビュー、読み取り専用のビューを参照することはできません。さらに、 NOT DETERMINISTIC または EXTERNAL ACTION を使って定義されている副照会またはルーチンを参照する WHERE 文節を直接または間接的に含むようなビューを参照することもできません (SQLSTATE 42807)。

マージ操作のターゲットが全選択である場合、全選択は、CREATE VIEW ステートメントの説明の『注』にある、『更新可能ビュー』、『削除可能ビュー』、または『挿入可能ビュー』という項目で定義されているように、更新可能、削除可能、または挿入可能になっている必要があります。

correlation-clause

search-condition 内や *assignment-clause* の右側で使用して、表、ビュー、または全選択を指定できます。 *correlation-clause* についての説明は、『副選択』の説明にある『table-reference』を参照してください。

USING *table-reference*

ターゲットにマージされる結果表として、行のセットを指定します。結果表が空の場合、警告が戻されます (SQLSTATE 02000)。

ON *search-condition*

マージの更新または削除操作のために *table-reference* のどの行が使用されるか、およびマージの挿入操作のためにどの行が使用されるかを指定します。

副照会以外の検索条件に含まれる各 *column-name* は、ターゲット表、ビュー、または *table-reference* の列を指定していなければなりません。検索条件の中に、同じ表が MERGE と副照会の両方の基本オブジェクトであるような副照会が含まれている場合、行が更新または挿入される前に、その副照会が完全に評価されます。

search-condition は、ターゲット表および *table-reference* の結果表の各行に適用されます。 *table-reference* の結果表の中で、 *search-condition* が真である行に対して、指定された更新または削除操作が実行されます。 *table-reference* の結果表の中で、 *search-condition* が真でない行に対しては、指定された挿入操作が実行されます。

search-condition に副照会が含まれる場合、 *table-reference* の結果表の行、および検索条件の適用に使われた結果に対して検索条件が適用されるときには常に、副照会が効果的に実行されます。実際、相関参照が含まれない副照会は 1 度だけ実行されるのに対し、相関参照を含む副照会は、 *table-reference* の結果表の各行ごとに 1 度以上実行される必要があります。

WHEN *matching-condition*

modification-operation または *signal-statement* が実行される条件を指定します。それぞれの *matching-condition* は、指定された順序で評価されます。*matching-condition* が真と評価された行は、後続の一致条件では無視されます。

MATCHED

ON 検索条件が真である行に対して実行される操作を示します。 THEN の後には、UPDATE、DELETE、または *signal-statement* のみを指定できます。

AND *search-condition*
ON 検索条件に一致する行に対して THEN 後の操作を実行するための、さらに適用される追加の検索条件を指定します。

NOT MATCHED
ON 検索条件が偽または不明である行に対して実行される操作を示します。THEN の後には、INSERT または *signal-statement* のみを指定できます。

AND *search-condition*
ON 検索条件に一致しなかった行に対して THEN 後の操作を実行するための、さらに適用される追加の検索条件を指定します。

THEN *modification-operation*
matching-condition が真と評価された場合に実行される操作を指定します。

UPDATE SET
matching-condition が真と評価された行に対して実行される更新操作を指定します。

assignment-clause
列更新のリストを指定します。

column-name
更新したい列を指定します。 *column-name* は指定された表またはビューの列を識別する必要がありますが、スカラー関数、定数、または式から得られたビュー列を識別することはできません。同じ列を 2 度以上指定することはできません (SQLSTATE 42701)。

1 つのビュー列から得られた 2 つのビュー列を更新するとき、両方の列を 1 つの MERGE ステートメントで更新することはできません (SQLSTATE 42701)。

expression
列の新しい値を指定します。 *expression* には列関数を含めることはできません (SQLSTATE 42903)。

expression には、 *table-name* または *view-name* の列への参照を含めることができます。更新対象の行ごとに、式の中のそのような列の値は、行の更新前のその行の列の値になります。

DEFAULT
列に割り当てられるデフォルト値。デフォルト値を持つ列に関してのみ、DEFAULT を指定できます。データ・タイプのデフォルト値については、 『CREATE TABLE』 ステートメントの DEFAULT 文節に関する説明を参照してください。

GENERATED ALWAYS として定義された列に関しては、DEFAULT を指定する必要があります。 GENERATED BY DEFAULT として定義された列に関しては、有効な値を指定することができます。

NULL
列の新しい値として NULL 値を指定します。 NULL は、NULL 可能列にのみ指定できます (SQLSTATE 23502)。

DELETE
matching-condition が真と評価された行に対して実行される削除操作を指定します。

insert-operation
matching-condition が真と評価された行に対して実行される挿入操作を指定します。

INSERT
挿入操作に使われる、列名と行値の式からなるリストを指定します。

行値の式における行の値の数は、挿入列リストにおける名前の数と同じでなければなりません。最初の値はリストの最初の列に挿入され、2 番目の値は 2 番目の列に挿入されます。以下同様です。

(*column-name,...***)**
挿入値が提供される列を指定します。それぞれの名前は、表またはビューの列を識別する必要があります。同じ列を 2 度以上指定することはできません (SQLSTATE 42701)。挿入値を受け入れることのできないビューの列を指定することはできません。以下のようなビュー列には、値を挿入できません。

- 定数、式、またはスカラー関数から得られる列。
- そのビューの他の列と同じ基本表の列から得られる列。

操作の対象となるビューにこのような列がある場合は、列名のリストを指定する必要があり、そのリストに上記の列を指定してはなりません。

列のリストを省略すると、その表またはビューのすべての列を左から右に指定したリストが暗黙に指定されます。このリストはステートメントが準備される時点で確立されます。したがって、ステートメントの準備後に表に追加された列は含まれません。

VALUES
挿入したい 1 行または複数行の値を、この後に指定します。

expression
列名を含まない任意の式 (SQLSTATE 42703)。

DEFAULT
列に割り当てられるデフォルト値。デフォルト値を持つ列に関してのみ、DEFAULT を指定できます。データ・タイプのデフォルト値については、 『CREATE TABLE』 ステートメントの DEFAULT 文節に関する説明を参照してください。

GENERATED ALWAYS として定義された列に関しては、DEFAULT を指定する必要があります。 GENERATED BY DEFAULT として定義された列に関しては、有効な値を指定することができます。

NULL
列の値として NULL 値を指定します。 NULL は、NULL 可能列にのみ指定できます (SQLSTATE 23502)。

signal-statement
matching-condition が真と評価された場合にエラーを戻すために実行される SIGNAL ステートメントを指定します。

ELSE IGNORE
どの *matching-condition* も真と評価されない場合に、行に対してアクションが実行されないことを指定します。

規則:

- 複数の *modification-operation* (UPDATE SET、DELETE 、または *insert-operation*) あるいは *signal-statement* を、単一の MERGE ステートメントの中で指定できます。
- ターゲット内の各行は、1 度だけ操作できます。ターゲット内の各行は、*table-reference* の結果表のただ 1 つの行とのみ MATCHED として識別されます (SQLSTATE 21506)。ネストした SQL 操作 (RI、または INSTEAD OF トリガーを除くトリガー) では、ターゲット表 (または同じ表階層内の表) を UPDATE、DELETE、INSERT、または MERGE ステートメントのターゲットとして指定することはできません (SQLSTATE 27000)。

MERGE ステートメントにおける更新、挿入、または削除操作に関連した他の規則については、該当するステートメントの『規則』セクションを参照してください。

注:

- ***処理の順序***:
  1. ソースからターゲットにかけて処理される行のセットを判別します。このステートメントで CURRENT TIMESTAMP が使用される場合、ステートメント全体でただ 1 度だけクロックが読み取られます。
  2. ON 文節を使用して、これらの行が MATCHED または NOT MATCHED のいずれであるかを分類します。
  3. WHEN 文節内に *matching-condition* があれば評価します。
  4. *assignment-clause* および *insert-operation* 内に *expression* があれば評価します。
  5. それぞれの *signal-statement* を実行します。
  6. 指定された順序に従って、それぞれの *modification-operation* を該当する行に適用します。それぞれの *modification-operation* によって活動化される制約およびトリガーが、その *modification-operation* に関して実行されます。ステートメント・レベルのトリガーは、 *modification-operation* を満たす行が存在しない場合でも活動化されます。それぞれの *modification-operation* は、後続の各 *modification-operation* のトリガーや参照制約に影響する可能性があります。
- ***ステートメント・レベルの原子性***: MERGE ステートメントの実行中にエラーが発生した場合、ステートメント全体がロールバックされます。
- ***更新される行数***: MERGE ステートメントの実行が完了すると、 SQLCA の GET DIAGNOSTICS および SQLERRD(3) の ROW_COUNT 項目の値は、 MERGE ステートメントによって処理された行数になります (ELSE IGNORE 文節によって

識別された行を除く)。 SQLERRD(3) の値には、制約またはトリガーの結果として処理された行数は含まれません。 SQLERRD(5) の値には、このような行の数が含まれます。

- ***挿入された行を更新することはできない***: ターゲット内で、 MERGE ステートメントの実行前に存在しなかった行の更新操作は一切行われません。つまり、MERGE ステートメントによって挿入された行は更新されません。
- ***INSTEAD OF トリガー***: MERGE ステートメントのターゲットとしてビューが指定される場合、そのビューに対して、まったく INSTEAD OF トリガーを定義しないか、更新、削除、挿入の各 INSTEAD OF トリガーを定義する必要があります (SQLSTATE 428FZ)。

例:

例 1: 記述 (description) が変更されたアクティビティーに関して、アーカイブ表内の記述を更新します。新しいアクティビティーについては、アーカイブ表に挿入します。アーカイブ表とアクティビティー表にはどちらも、主キーとしてアクティビティーが含まれます。

```
MERGE INTO archive ar
USING (SELECT activity, description FROM activities) ac
ON (ar.activity = ac.activity)
WHEN MATCHED THEN
   UPDATE SET
      description = ac.description
WHEN NOT MATCHED THEN
  INSERT
      (activity, description)
      VALUES (ac.activity, ac.description)
```

例 2: 出荷 (shipment) 表を使って、在庫 (inventory) 表に行をマージします。その際、出荷表のマッチした行の部品カウント (part count) ごとに数量を増分します。そうでない場合は、新しい *partno* を在庫表に挿入します。

```
MERGE INTO inventory AS in
USING (SELECT partno, description, count FROM shipment
       WHERE shipment.partno IS NOT NULL) AS sh
ON (in.partno = sh.partno)
WHEN MATCHED THEN
   UPDATE SET
      description = sh.description,
      quantity = in.quantity + sh.count
WHEN NOT MATCHED THEN
   INSERT
      (partno, description, quantity)
      VALUES (sh.partno, sh.description, sh.count)
```

例 3: 決済 (transaction) 表を使って、アカウント (account) 表に行をマージします。その際、いくつかのトランザクションによるアカウント ID の収支を更新し、アカウントがまだ存在しない場合には、統合トランザクションからアカウントを新しく挿入します。

```
MERGE INTO account AS a
USING (SELECT id, sum(amount) sum_amount FROM transaction
       GROUP BY id) AS t
ON a.id = t.id
WHEN MATCHED THEN
   UPDATE SET
      balance = a.balance + t.sum_amount
```

```
  WHEN NOT MATCHED THEN
    INSERT
        (id, balance)
        VALUES (t.id, t.sum_amount)
```

例 4: トランザクション・ログ (transaction_log) 表を使って、 employee_file (従業員ファイル) 表に行をマージします。その際、トランザクション時間に基づいて、最新の トランザクション・ログ (transaction_log) 行の内容で電話 (phone) および部署 (office) を更新し、まだ存在しない場合には、最新の新しい 従業員ファイル (employee_file) 行を挿入します。

```
MERGE INTO employee_file AS e
USING (SELECT empid, phone, office
       FROM (SELECT empid, phone, office,
             ROW_NUMBER() OVER (PARTITION BY empid
             ORDER BY transaction_time DESC) rn
             FROM transaction_log) AS nt
             WHERE rn = 1) AS t
ON e.empid = t.empid
WHEN MATCHED THEN
   UPDATE SET
      (phone, office) =
      (t.phone, t.office)
WHEN NOT MATCHED THEN
  INSERT
      (empid, phone, office)
      VALUES (t.empid, t.phone, t.office)
```

例 5: 従業員 (employee) 行に動的に提供される値を使って、既存の従業員に該当するデータの場合はマスター従業員 (employee) 表を更新します。データが新しい従業員に関するものである場合は、行を挿入します。次の例は、C プログラムのコードの断片です。

```
hv1 =
"MERGE INTO employee AS t
USING TABLE(VALUES(CAST (? AS CHAR(6)), CAST (? AS VARCHAR(12)),
                   CAST (? AS CHAR(1)), CAST (? AS VARCHAR(15)),
                   CAST (? AS SMALLINT), CAST (? AS INTEGER)))
                   s(empno, firstnme, midinit, lastname, edlevel, salary)
ON t.empno = s.empno
WHEN MATCHED THEN
   UPDATE SET
      salary = s.salary
WHEN NOT MATCHED THEN
  INSERT
      (empno, firstnme, midinit, lastname, edlevel, salary)
      VALUES (s.empno, s.firstnme, s.midinit, s.lastname, s.edlevel,
              s.salary)";
EXEC SQL PREPARE s1 FROM :hv1;
EXEC SQL EXECUTE s1 USING '000420', 'SERGE', 'K', 'FIELDING', 18, 39580;
```

例 6: Group A によって編成されたアクティビティーのリストをアーカイブ表内で更新します。期限切れのアクティビティーをすべて削除し、アーカイブ表のアクティビティー情報 (日付と記述) が変更されていれば、それを更新します。新規の着信アクティビティーについては、アーカイブ表に挿入します。アクティビティーの日付が不明の場合、エラーを発します。アーカイブ表内のアクティビティー日付の指定は必須です。アクティビティー表は、それぞれのグループごとに存在します。たとえば、activities_groupA にはこのグループが編成したすべてのアクティビティーが含まれ、アーカイブ表には企業のさまざまなグループによって編成された将来のアクティビティーがすべて含まれます。アーカイブ表には主キーとして (group,

activity) が含まれ、日付を NULL にすることはできません。すべてのアクティビティー表には、主キーとして activity が含まれます。アーカイブ内の最終更新 (last_modified) 列は、デフォルト値として CURRENT TIMESTAMP を使って定義されます。

```
MERGE INTO archive ar
USING (SELECT activity, description, date, last_modified
       FROM activities_groupA) ac
ON (ar.activity = ac.activity) AND ar.group = 'A'
WHEN MATCHED AND ac.date IS NULL THEN
   SIGNAL SQLSTATE '70001'
      SET MESSAGE_TEXT =
         ac.activity CONCAT ' cannot be modified. Reason: Date is not known'
WHEN MATCHED AND ac.date < CURRENT DATE THEN
   DELETE
WHEN MATCHED AND ar.last_modified < ac.last_modified THEN
  UPDATE SET
      (description, date, last_modified) = (ac.description, ac.date, DEFAULT)
WHEN NOT MATCHED AND ac.date IS NULL THEN
   SIGNAL SQLSTATE '70002'
      SET MESSAGE_TEXT =
         ac.activity CONCAT ' cannot be inserted. Reason: Date is not known'
WHEN NOT MATCHED AND ac.date >= CURRENT DATE THEN
  INSERT
      (group, activity, description, date)
      VALUES ('A', ac.activity, ac.description, ac.date)
ELSE IGNORE
```

関連資料:

- *SQL リファレンス 第 1 巻* の『式』
- *SQL リファレンス 第 1 巻* の『検索条件』
- *SQL リファレンス 第 1 巻* の『副選択』
- 347 ページの『CREATE TABLE』
- 505 ページの『DELETE』
- 615 ページの『INSERT』
- 779 ページの『UPDATE』
- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』

OPEN

OPEN ステートメントは、カーソルをオープンして、そのカーソルを結果表からの行の取り出しに使用できるようにします。

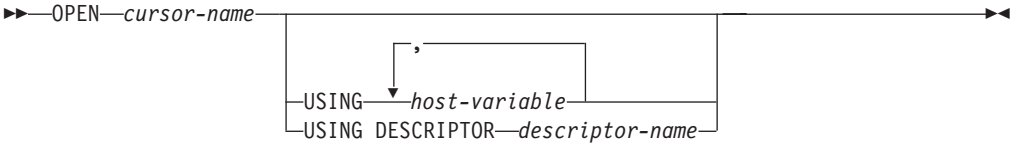
呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

カーソルの使用に必要な許可については、 『DECLARE CURSOR』を参照してください。

構文:



説明:

cursor-name
プログラムのそれ以前の個所にある DECLARE CURSOR ステートメントで定義されているカーソルの名前を指定します。 この OPEN ステートメントが実行される時点で、該当のカーソルはクローズ状態でなければなりません。

該当の DECLARE CURSOR ステートメントは、次のいずれかの方法により、SELECT ステートメントを指定していなければなりません。

- その DECLARE CURSOR ステートメントに SELECT ステートメントを組み込む。
- 準備済み SELECT ステートメントを指定する *statement-name* を組み込む。

該当のカーソルの結果表は、その SELECT ステートメントを評価することによって得られます。評価の際には、その SELECT ステートメントに指定されている特殊レジスターまたは PREVIOUS VALUE 式の現行値、またはその SELECT ステートメントか OPEN ステートメントの USING 文節に指定されたホスト変数の現行値が使用されます。結果表の行は、OPEN ステートメントの実行の過程で得られ、それらを入れる一時表が作成されるか、または後続の FETCH ステートメントの実行によって得られます。いずれの場合でも、カーソルはオープン状態になり、その位置はその結果表の最初の行の前になります。表が空の場合、カーソルの状態は「最終行の後」になります。

USING
この後に、準備済みステートメントのパラメーター・マーカー (?) に代入する値が入っているホスト変数のリストを指定します。 DECLARE CURSOR ステートメントでパラメーター・マーカーを含む準備済みステートメントを指定した

場合、 USING の使用は必須です。 準備済みステートメントにパラメーター・マーカーが含まれていない場合、USING は無視されます。

host-variable
ホスト変数の宣言規則に従って、そのプログラムで記述されている変数を指定します。変数の数は、準備済みステートメントのパラメーター・マーカーの数と同じでなければなりません。 *n* 番目の変数は、準備済みステートメントの *n* 番目のパラメーター・マーカーに対応します。 場合によっては、ロケーター変数とファイル参照変数も、パラメーター・マーカーの値のソースとして指定できます。

DESCRIPTOR *descriptor-name*
ホスト変数の有効な記述を含む SQLDA を指定します。

OPEN ステートメントが処理される前に、ユーザーは次に示す SQLDA 内のフィールドを設定する必要があります。

- SQLDA に用意する SQLVAR のエレメント数を示す SQLN
- SQLDA に割り振るストレージのバイト数を示す SQLDABC
- ステートメントの処理時にその SQLDA の使用される変数の数を示す SQLD
- 変数の属性を示す SQLVAR のオカレンス

SQLDA には、すべての SQLVAR オカレンスが入るだけの十分なストレージがなければなりません。したがって、SQLDABC の値は 16 + SQLN*(N) 以上でなければなりません (N は 1 つの SQLVAR オカレンスの長さ)。

LOB の結果列を入れるには、各選択リスト項目 (または結果表の列) ごとに 2 つの SQLVAR 項目が必要です。

SQLD に設定する値は、ゼロ以上で SQLN 以下でなければなりません。

規則:

- カーソルの SELECT ステートメントが評価される場合に、そのステートメント中の各パラメーター・マーカーは、対応するホスト変数によって置き換えられます。 型付きパラメーター・マーカーの場合、ターゲット変数の属性は CAST 指定によって指定されます。 タイプなしパラメーター・マーカーの場合、ターゲット変数の属性はパラメーター・マーカーのコンテキストに従って決定されます。
- V は、パラメーター・マーカー P に対応するホスト変数を表します。 V の値は、列への値の割り振り規則に従って、P のターゲット変数に割り当てられます。したがって、
  - V はターゲットと互換でなければなりません。
  - V がストリングの場合、その長さ (ただし、 LONG ストリングでないストリングの末尾ブランクは含まない) はターゲットの長さ属性を超えることはできません。
  - V が数値の場合、 V の整数部分の絶対値はターゲットの整数部分の絶対値の最大を超えることはできません。
  - V の属性がターゲットの属性と同一でない場合、その値はターゲットの属性に合うように変換されます。

カーソルの SELECT ステートメントが評価されると、 P の代わりに使用される値は P のターゲット変数になります。たとえば、V が CHAR(6) でターゲットが CHAR(8) の場合、 P の代わりに使用される値は V の値にブランクを 2 個付加したものになります。

- USING 文節は、パラメーター・マーカーを含む準備済み SELECT ステートメントのために用意されています。ただし、これは、カーソルの SELECT ステートメントが DECLARE CURSOR ステートメントの一部である場合にも使用できます。このような場合、OPEN ステートメントは、あたかも SELECT ステートメントの各ホスト変数がパラメーター・マーカーであるように実行されます。ただし、ターゲット変数の属性は SELECT ステートメントのホスト変数と同じになります。その結果、USING 文節に指定するホスト変数の値によって、カーソルの SELECT ステートメントの中のホスト変数の値がオーバーライドされることになります。
- カーソル定義に組み込まれている SQL データを変更する SQL データ変更ステートメントとルーチンは完全に実行され、結果セットは、カーソルのオープン時に一時表に保管されます。ステートメントの実行が正常に完了すると、SQLERRD(3) フィールドには、挿入、更新、および削除操作が可能な行の数の合計が入ります。全選択内にデータ変更ステートメントを含むカーソルが関係する OPEN ステートメントの実行中にエラーが発生した場合は、そのデータ変更ステートメントがロールバックされます。

  OPEN ステートメントの明示的なロールバック、つまり OPEN ステートメント以前のセーブポイントまでのロールバックにより、カーソルはクローズします。カーソルの定義で、全選択の FROM 文節内にデータ変更ステートメントが含まれている場合、データ変更ステートメントの結果はロールバックされます。

  SELECT ステートメントや SELECT INTO ステートメントにネストされていたデータ変更ステートメントで、ターゲット表の行に対して行われた変更は、カーソルがオープンされるときに処理されるため、カーソルに対するフェッチ操作の途中でエラーが発生しても、変更が元に戻ることはありません。

注:

- ***クローズ状態のカーソル:*** プログラムが開始された時点、およびプログラムが ROLLBACK ステートメントを開始した時点では、そのプログラム中のすべてのカーソルはクローズ状態になります。

  WITH HOLD として宣言されたオープン・カーソル以外のすべてのカーソルは、プログラムが COMMIT ステートメントを発行する際にクローズ状態になります。

  また、 CLOSE ステートメントを実行した場合、またはカーソル位置が予期できなくなるようなエラーが検出された場合にも、カーソルはクローズ状態になることがあります。
- カーソルの結果表から行を取り出すには、カーソルがオープンされている時に FETCH ステートメントを実行します。 カーソルの状態をクローズからオープンに変更する唯一の方法は、 OPEN ステートメントを実行することです。
- ***一時表の効果:*** 場合によっては、FETCH ステートメントの実行の過程でカーソルの結果表が得られます。また、一時表メソッドが使用される場合もあります。

このメソッドでは、結果表全体が OPEN ステートメントの実行中に一時表に転送されます。 一時表が使用される場合、プログラムの結果は、以下の点で異なる可能性があります。

- 以後の FETCH ステートメントまでは起こることのないエラーが、 OPEN の過程で起こる可能性があります。
- カーソルがオープン状態の間、同じトランザクションで実行された INSERT、UPDATE、および DELETE ステートメントは結果表に影響を与えません。
- OPEN の実行中に、結果表の行ごとに SELECT ステートメント中の NEXT VALUE 式が評価されます。

逆に、一時表を使用しない場合、カーソルがオープン状態の間に実行される INSERT、 UPDATE、および DELETE ステートメントが、同じ作業単位から発行される場合には結果表に影響を与えることがあり、個々の行が取り出されるたびに SELECT ステートメント中の NEXT VALUE 式が評価されます。この結果表は、同じ作業単位で実行される操作による影響を受けることがあり、そのような操作の影響は、必ずしも予測可能であるとは限りません。たとえば、カーソル C の位置が SELECT * FROM T と定義された結果表の 1 つの行である場合に、 T に新しい行を挿入すると、行の順序が整っていないために、その挿入が結果表に与える影響は予測できません。したがって、後続する FETCH C で T の新しい行が取り出される場合もあれば、取り出されない場合もあります。

- ステートメントのキャッシュは、 OPEN ステートメントによってオープンと宣言されているカーソルに影響を与えます。

例:

例 1: COBOL プログラムで、以下を行う組み込みステートメントを作成します。

1. カーソル C1 を定義します。このカーソルは、 DEPARTMENT 表から管理部門 (ADMRDEPT) 'A00' によって管理される部門の行すべてを検索するために使用します。
2. 最初に取り出す行の前に、カーソル C1 を置きます。

```
EXEC SQL  DECLARE C1 CURSOR FOR
                SELECT DEPTNO, DEPTNAME, MGRNO
                  FROM DEPARTMENT
                  WHERE ADMRDEPT = 'A00'
END-EXEC.


EXEC SQL  OPEN C1
END-EXEC.
```

例 2: C プログラムで動的に定義される 選択ステートメントにカーソル DYN_CURSOR を関連付ける OPEN ステートメントをコーディングします。選択ステートメントの述部には 2 つのパラメーター・マーカーが使用されており、 2 つのホスト参照変数をその OPEN ステートメントに指定して、アプリケーションとデータベースとの間で整数と VARCHAR(64) の値を渡すために使用します。 (関連するホスト変数の定義、PREPARE ステートメント、および DECLARE CURSOR ステートメントも以下の例に示しています。)

```
EXEC SQL  BEGIN DECLARE SECTION;
  static short    hv_int;
  char            hv_vchar64[65];
  char    stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
```

```
EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING :hv_int, :hv_vchar64;
```

例 3: 例 2 と同様に OPEN ステートメントをコーディングしますが、この例では WHERE 文節のパラメーター・マーカーの数とデータ・タイプは未知です。

```
EXEC SQL  BEGIN DECLARE SECTION;
  char    stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;

EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING DESCRIPTOR :sqlda;
```

関連資料:

- 491 ページの『DECLARE CURSOR』
- 551 ページの『EXECUTE』
- 647 ページの『PREPARE』
- *SQL リファレンス 第 1 巻* の『SQLDA (SQL 記述子域)』

関連サンプル:

- 『dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』
- 『tut_read.sqC -- How to read tables (C++)』
- 『udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)』

PREPARE

PREPARE ステートメントは、SQL ステートメントの動的な実行を準備するために、アプリケーション・プログラムによって使用されます。 PREPARE ステートメントは、*ステートメント・ストリング* と呼ばれる文字ストリング形式のステートメントから、*準備済みステートメント* と呼ばれる実行可能な SQL ステートメントを作成します。

呼び出し:

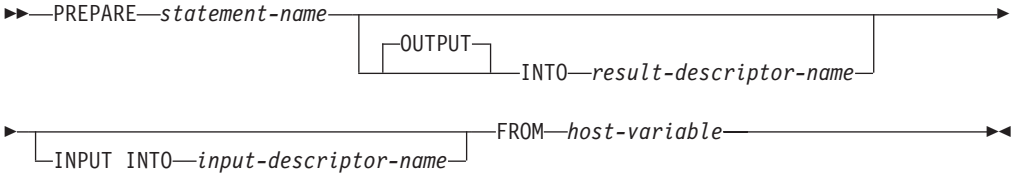
このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

ステートメントの準備時に許可検査が行われるステートメント (DML) の場合、ステートメントの許可 ID の特権には、 PREPARE ステートメントで指定されている SQL ステートメントの実行に必要な特権が含まれていなければなりません。ステートメントの許可 ID は、BIND オプション DYNAMICRULES の影響を受けることがあります。

ステートメントの実行時に許可検査が行われるステートメント (DDL、GRANT、および REVOKE ステートメント) の場合、このステートメントを使用するために必要な許可は特にありません。ただし、準備済みステートメントの実行時にその許可が検査されます。

構文:



説明:

statement-name
準備したいステートメントの名前を指定します。名前として既存の準備済みのステートメントを指定した場合、前もって準備されたそのステートメントは破棄されます。名前として、オープン・カーソルの SELECT ステートメントである準備済みステートメントを指定することはできません。

OUTPUT INTO
OUTPUT INTO を使用すると、PREPARE ステートメントを正常に実行した場合に、準備済みステートメント中の出力パラメーター・マーカーについての情報が、 *result-descriptor-name* で指定する SQLDA に入れられます。

result-descriptor-name
SQLDA の名前を指定します。 (この文節の代わりに、DESCRIBE ステートメントを使用できます。)

INPUT INTO
INPUT INTO を使用すると、PREPARE ステートメントを正常に実行した場合に、準備済みステートメント中の入力パラメーター・マーカーについての情報

が、 *input-descriptor-name* で指定する SQLDA に入れられます。入力パラメーター・マーカーは、使用法に関係なく常に NULL 可能と見なされます。

input-descriptor-name
SQLDA の名前を指定します。 (この文節の代わりに、DESCRIBE ステートメントを使用できます。)

FROM
この後に、ステートメント・ストリングを指定します。ステートメント・ストリングは、指定するホスト変数の値です。

host-variable
文字ストリング変数の宣言規則に従ってそのプログラムで記述されているホスト変数を指定します。固定長または可変長の文字ストリング変数でなければなりません。

規則:

- ***ステートメント・ストリングの規則:*** ステートメント・ストリングは、動的に準備可能な実行可能ステートメントでなければなりません。以下のいずれかの SQL ステートメントでなければなりません。
  - ALTER
  - CALL
  - COMMENT
  - COMMIT
  - CREATE
  - DECLARE GLOBAL TEMPORARY TABLE
  - DELETE
  - DROP
  - EXPLAIN
  - FLUSH EVENT MONITOR
  - FLUSH PACKAGE CACHE
  - GRANT
  - INSERT
  - LOCK TABLE
  - REFRESH TABLE
  - RELEASE SAVEPOINT
  - RENAME TABLE
  - RENAME TABLESPACE
  - REVOKE
  - ROLLBACK
  - SAVEPOINT
  - SELECT ステートメント
  - SET CURRENT DEFAULT TRANSFORM GROUP
  - SET CURRENT DEGREE
  - SET CURRENT EXPLAIN MODE

- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET INTEGRITY
- SET PASSTHRU
- SET PATH
- SET SCHEMA
- SET SERVER OPTION
- UPDATE

- ***パラメーター・マーカー:*** ステートメント・ストリングにホスト変数への参照を組み込むことはできませんが、パラメーター・マーカー を組み込むことはできます。準備済みステートメントの実行時に、パラメーター・マーカーはホスト変数の値に置き換えることができます。 CALL ステートメントの場合、ストアード・プロシージャーに対する OUT 引き数や INOUT 引き数に、パラメーター・マーカーを使用することもできます。 CALL の実行後に、引き数の戻り値は、パラメーター・マーカーに対応するホスト変数に割り当てられます。

  パラメーター・マーカーは疑問符 (?) で表されます。ステートメント・ストリングが静的 SQL ステートメントであれば、パラメーター・マーカーは、ホスト変数を使用できる場所に使用できます。 パラメーター・マーカーがどのように値で置き換えられるかについては、 『OPEN』 と 『EXECUTE』 を参照してください。

  パラメーター・マーカーには、以下の 2 つのタイプがあります。

  型付きパラメーター・マーカー
  ターゲットのデータ・タイプと一緒に指定するパラメーター・マーカー。一般的な形式は、次のとおりです。

  ```
  CAST(? AS data-type)
  ```

  この表記は関数呼び出しではなく、ランタイムのパラメーター・タイプが指定のデータ・タイプであること、または指定のデータ・タイプに変換できるデータ・タイプであることを「保証」するものです。たとえば、

  ```
  UPDATE EMPLOYEE
  SET LASTNAME = TRANSLATE(CAST(? AS VARCHAR(12)))
  WHERE EMPNO = ?
  ```

  TRANSLATE 関数の引き数の値は、ランタイムに与えられます。その値のデータ・タイプは、VARCHAR(12)、または VARCHAR(12) に変換可能なタイプになるはずです。

  タイプなしパラメーター・マーカー
  ターゲットのデータ・タイプを指定しないで指定するパラメーター・マーカー。これは、1 つの疑問符の形式です。タイプなしパラメーター・マーカーのデータ・タイプは、そのコンテキストによって決まります。たとえば、上

記の UPDATE ステートメントの述部にあるタイプの指定がないパラメーター・マーカーは、 EMPNO 列のデータ・タイプと同じになります。

型付きパラメーター・マーカーは、動的 SQL ステートメントで、ホスト変数がサポートされている場所であれば、どこにでも使用でき、そのデータ・タイプは CAST 関数で行った保証に基づきます。

タイプなしパラメーター・マーカーは、動的 SQL ステートメントで、ホスト変数がサポートされている位置の中から選択された位置で使用できます。 それらの位置と結果データ・タイプを、表 11 に示しています。この表で、位置は、式、述部、組み込み関数、およびユーザー定義ルーチンに類別されており、タイプなしパラメーター・マーカーが使用可能かどうかを容易に調べることができます。非修飾関数名の関数 (算術演算子、CONCAT、および日付/時刻演算子を含む) にタイプなしパラメーター・マーカーを使用する場合、関数解決の目的で、その修飾子は 'SYSIBM' に設定されます。

表 11. タイプなしパラメーター・マーカーの使用法

タイプなしパラメーター・マーカーの位置	データ・タイプ
式 (選択リスト、CASE、VALUES を含む)	
選択リストに単独で	エラー
演算子の優先順位と演算順序規則の分析後に、単一算術演算子のオペランドの両方となる位置 例: `? + ? + 10`	エラー
日付/時刻の式以外の算術式の単一演算子のオペランドのいずれか一方 例: `? + ? * 10`	もう一方のオペランドのデータ・タイプ。
日付/時刻の式のラベル付き期間 (ラベル付き期間の単位のタイプを示す部分には、パラメーター・マーカーを使用できません。)	DECIMAL(15,0)
日付/時刻の式のその他のオペランド ('timecol + ?' または '? - datecol')	エラー
CONCAT 演算子の 2 つのオペランド	エラー
CONCAT 演算子の 1 つのオペランド (もう一方のオペランドが CLOB 以外の文字データ・タイプである場合)	一方のオペランドが CHAR(n) または VARCHAR(n) (n は 128 より小さい) の場合、もう一方のオペランドは VARCHAR(254 - n)。その他のすべての場合のデータ・タイプは VARCHAR(254)。
CONCAT 演算子の 1 つのオペランド (もう一方のオペランドが DBCLOB 以外のグラフィック・データ・タイプである場合)	一方のオペランドが GRAPHIC(n) または VARGRAPHIC(n) の場合 (n は 64 より小さい)、もう一方のオペランドは VARCHAR(127 - n)。その他のすべての場合のデータ・タイプは VARCHAR(127)。

表 11. タイプなしパラメーター・マーカーの使用法 (続き)

タイプなしパラメーター・マーカーの位置	データ・タイプ
CONCAT 演算子の 1 つのオペランド (もう一方のオペランドがラージ・オブジェクト・ストリングである場合)	もう一方のオペランドと同じ。
UPDATE ステートメントの SET 文節の右側の値	列のデータ・タイプ。その列がユーザー定義特殊タイプとして定義されている場合は、そのユーザー定義特殊タイプのソース・データ・タイプ。その列がユーザー定義の構造タイプとして定義されている場合は、構造タイプ。これはトランスフォーム関数の戻りタイプも示している。
単純な CASE 式の CASE キーワードに続く式	エラー
結果式の残りがタイプなしパラメーター・マーカーまたは NULL のいずれかである CASE 式 (単純および探索) の結果式の少なくとも 1 つ	エラー
単純 CASE 式の WHEN の後のいずれかまたはすべての式	タイプなしパラメーター・マーカー以外の CASE の後の式および WHEN の後の式に、結果データ・タイプに関する規則を適用した結果。
NULL でもタイプなしパラメーター・マーカーでもない結果式が少なくとも 1 つある CASE 式 (単純および探索) の結果式	結果式のうち NULL でもタイプなしパラメーター・マーカーでもないものすべてに、結果データ・タイプに関する規則を適用した結果。
INSERT ステートメント内にない単一行 VALUES 文節の列式として単独で	エラー
INSERT ステートメント内になく、他のすべての行式での同じ位置にある列式がタイプなしパラメーター・マーカーである複数行 VALUES 文節の列式として単独で	エラー
INSERT ステートメント内になく、他の行式のうちの少なくとも 1 つで同じ位置にある式がタイプなしパラメーター・マーカーでも NULL でもない複数行 VALUES 文節の列式として単独で	タイプなしパラメーター・マーカー以外のすべてのオペランドに結果データ・タイプに関する規則を適用した結果。
INSERT ステートメント内にある単一行 VALUES 文節の列式として単独で	列のデータ・タイプ。その列がユーザー定義特殊タイプとして定義されている場合は、そのユーザー定義特殊タイプのソース・データ・タイプ。その列がユーザー定義の構造タイプとして定義されている場合は、構造タイプ。これはトランスフォーム関数の戻りタイプも示している。

表 11. タイプなしパラメーター・マーカーの使用法 (続き)

タイプなしパラメーター・マーカーの位置	データ・タイプ
INSERT ステートメント内にある複数行 VALUES 文節の列式として単独で	列のデータ・タイプ。その列がユーザー定義特殊タイプとして定義されている場合は、そのユーザー定義特殊タイプのソース・データ・タイプ。その列がユーザー定義の構造タイプとして定義されている場合は、構造タイプ。これはトランスフォーム関数の戻りタイプも示している。
SET 特殊レジスター・ステートメントの右側にある値として	特殊レジスターのデータ・タイプ。
述部	
比較演算子の両方のオペランド	エラー
比較演算子の一方のオペランド (もう一方のオペランドはタイプなしパラメーター・マーカー)	もう一方のオペランドのデータ・タイプ
BETWEEN 述部のすべてのオペランド	エラー
BETWEEN 述部の 第 1 と第 2 オペランド、または 第 1 と第 3 オペランド BETWEEN 述部のオペランド	もう一方の非パラメーター・マーカーと同じ。
その他の BETWEEN の場合 (タイプなしパラメーター・マーカーが 1 個だけの場合など)	タイプなしパラメーター・マーカー以外のすべてのオペランドに結果データ・タイプに関する規則を適用した結果。
IN 述部のすべてのオペランド	エラー
IN 述部の第 1 オペランド (右側が副選択でない場合: ? IN (?,A,B) や ? IN (A,?,B,?) など)	IN リストのオペランド (IN キーワードの右側のオペランド) のうち、タイプなしパラメーター・マーカー以外のすべてのオペランドに、結果データ・タイプに関する規則を適用した結果。
IN 述部の第 1 オペランド (右側が全選択の場合)	選択した列のデータ・タイプ。
IN 述部の IN リストのいずれかまたはすべてのオペランド	IN 述部のオペランド (IN 述部の左右のオペランド) のうち、タイプなしパラメーター・マーカー以外のすべてのオペランドに結果データ・タイプに関する規則を適用した結果。
LIKE 述部の 3 つのオペランドすべて	一致式 (オペランド 1) とパターン式 (オペランド 2) は VARCHAR(32672)。 エスケープ式 (オペランド 3) は VARCHAR(2)。
LIKE 述部の一致式 (パターン式またはエスケープ式のいずれかが、タイプなしパラメーター・マーカー以外である場合)。	第 1 オペランドのデータ・タイプ (タイプなしパラメーター・マーカー) に応じて、VARCHAR(32672) または VARGRAPHIC(16336) のいずれか。

表 11. タイプなしパラメーター・マーカーの使用法 (続き)

タイプなしパラメーター・マーカーの位置	データ・タイプ
LIKE 述部のパターン式 (一致式またはエスケープ式のいずれかが、タイプなしパラメーター・マーカー以外である場合)。	第 1 オペランドのデータ・タイプ (タイプなしパラメーター・マーカー) に応じて、VARCHAR(32672) または VARGRAPHIC(16336) のいずれか。一致式のデータ・タイプが BLOB の場合、パターン式のデータ・タイプは BLOB(32672) と見なされます。
LIKE 述部のエスケープ式 (一致式またはパターン式のいずれかが、タイプなしパラメーター・マーカー以外である場合)	第 1 オペランドのデータ・タイプ (タイプなしパラメーター・マーカー) に応じて、VARCHAR(2) または VARGRAPHIC(1) のいずれか。一致式またはパターン式のデータ・タイプが BLOB の場合、エスケープ式のデータ・タイプは BLOB(1) と見なされます。
NULL 述部のオペランド	エラー
組み込み関数	
COALESCE (VALUE とも呼ばれる) または NULLIF のすべてのオペランド	エラー
少なくとも最初のオペランドがタイプなしパラメーター・マーカー以外である COALESCE または NULLIF のオペランド。	タイプなしパラメーター・マーカー以外のすべてのオペランドに結果データ・タイプに関する規則を適用した結果。
POSSTR (両方のオペランド)	両方のオペランドは共に VARCHAR(32672)。
POSSTR の一方のオペランド (もう一方のオペランドが文字データ・タイプである場合)	VARCHAR(32672)。
POSSTR の 1 つのオペランド (もう一方のオペランドが GRAPHIC データ・タイプである場合)	VARGRAPHIC(16336)。
POSSTR の探索ストリング・オペランド (もう一方のオペランドが BLOB である場合)	BLOB(32672)。
SUBSTR (第 1 オペランド)	VARCHAR(32672)
SUBSTR (第 2 と第 3 のオペランド)	INTEGER
TRANSLATE スカラー関数の第 1 オペランド	エラー
TRANSLATE スカラー関数の第 2 と第 3 のオペランド	第 1 オペランドが文字タイプの場合、VARCHAR(32672)。第 1 オペランドが GRAPHIC タイプの場合、VARGRAPHIC(16336)。
TRANSLATE スカラー関数の第 4 オペランド	第 1 オペランドが文字タイプの場合、VARCHAR(1)。第 1 オペランドが GRAPHIC タイプの場合、VARGRAPHIC(1)。
TIMESTAMP スカラー関数の第 2 オペランド	TIME
単項マイナス	DOUBLE-PRECISION
単項プラス	DOUBLE-PRECISION
VARCHAR_FORMAT 関数の第 1 オペランド	TIMESTAMP

表 11. タイプなしパラメーター・マーカーの使用法 (続き)

タイプなしパラメーター・マーカーの位置	データ・タイプ
TIMESTAMP_FORMAT 関数の第 1 オペランド	VARCHAR (短ストリングの長さ)
その他のすべてのスカラー関数のその他のすべてのオペランド	エラー
列関数のオペランド。	エラー
ユーザー定義ルーチン	
関数の引き数	エラー
メソッドの引き数	エラー
プロシージャーの引き数	プロシージャーの作成時に定義された、パラメーターのデータ・タイプ

注:

- PREPARE ステートメントが実行される時点で、ステートメント・ストリングの構文解析が行われ、エラーの有無が検査されます。ステートメント・ストリングが無効な場合には、エラー条件が SQLCA に報告されます。 エラーが訂正されない限り、そのステートメントを参照するそれ以降の EXECUTE または OPEN ステートメントも同じエラーになります (システムにより行われる暗黙の準備によって)。
- 準備済みステートメントは、以下の種類のステートメントで、示された制限付きで参照できます。

ステートメントの種類	準備済みステートメント...
DESCRIBE	任意のステートメント
DECLARE CURSOR	SELECT でなければならない
EXECUTE	SELECT であってはならない

- 準備済みステートメントは、何回でも実行できます。実際に、準備済みステートメントが 1 回しか実行されず、しかもパラメーター・マーカーが含まれていない場合には、 PREPARE と EXECUTE ステートメントを使用するよりも、EXECUTE IMMEDIATE ステートメントを使用する方が効率が良くなります。
- ステートメントのキャッシュは、準備の繰り返しに影響します。

例:

例 1: select ステートメント以外のステートメントを COBOL プログラムで準備して実行します。そのステートメントはホスト変数 HOLDER に含まれ、ユーザーによる何らかの指示に基づいて、プログラムはそのステートメント・ストリングをそのホスト変数に入れるものと想定します。準備するステートメントには、パラメーター・マーカーは含まれていません。

```
EXEC SQL  PREPARE STMT_NAME FROM :HOLDER
END-EXEC.
EXEC SQL  EXECUTE STMT_NAME
END-EXEC.
```

例 2: 例 1 と同様に select ステートメント以外のステートメントを準備しますが、この例では、C プログラムにコーディングします。また、準備するステートメントには、いくつかのパラメーター・マーカーが含まれていると想定します。

```
EXEC SQL  PREPARE STMT_NAME FROM :holder;
EXEC SQL  EXECUTE STMT_NAME USING DESCRIPTOR :insert_da;
```

以下のステートメントを準備するものと想定します。

```
INSERT INTO DEPT VALUES(?, ?, ?, ?)
```

DEPT 表の列は、以下のように定義されています。

```
DEPT_NO   CHAR(3) NOT NULL, -- department number
DEPTNAME  VARCHAR(29), -- department name
MGRNO     CHAR(6), -- manager number
ADMRDEPT  CHAR(3)  -- admin department number
```

部門長が存在せず、部門 A00 に報告を行う COMPLAINTS という名前の部門番号 G01 を挿入するには、 EXECUTE ステートメントを実行する前に、構造体 INSERT_DA は表 12 中の値を持っていなければなりません。

表 12.

SQLDA フィールド	**値**
SQLDAID	SQLDA
SQLDABC	192 (注 1 を参照。)
SQLN	4
SQLD	4
SQLTYPE	452
SQLLEN	3
SQLDATA	G01 を指すポインター
SQLIND	(注 2 を参照。)
SQLNAME	
SQLTYPE	449
SQLLEN	29
SQLDATA	COMPLAINTS を指すポインター
SQLIND	0 を指すポインター
SQLNAME	
SQLTYPE	453
SQLLEN	6
SQLDATA	(注 3 を参照。)
SQLIND	-1 を指すポインター
SQLNAME	
SQLTYPE	453
SQLLEN	3
SQLDATA	A00 を指すポインター
SQLIND	0 を指すポインター
SQLNAME	

表 12. (続き)

SQLDA フィールド	値
注: 1. この値は、32 ビット・アプリケーションで PREPARE が実行される場合を想定しています。 64 ビット・アプリケーションで PREPARE が実行される場合は、SQLDABC の値は 240 になります。 2. SQLTYPE は NULL 不可データ・タイプを識別するので、SQLIND 中のこの SQLVAR の値は無視されます。 3. SQLIND の値は、SQLDATA 中のこの SQLVAR の値が NULL 値であることを識別するので、この値は無視されます。	

関連資料:

- *SQL リファレンス 第 1 巻* の『ID』
- *SQL リファレンス 第 1 巻* の『LIKE 述部』
- 512 ページの『DESCRIBE』
- 551 ページの『EXECUTE』
- 642 ページの『OPEN』
- *SQL リファレンス 第 1 巻* の『結果データ・タイプの規則』

関連サンプル:

- 『inpsrv.sqb -- A stored procedure using the GENERAL parameter style (MF COBOL)』
- 『trigsql.sqb -- How to use a trigger on a table (MF COBOL)』
- 『tbread.sqc -- How to read tables (C)』
- 『tut_use.sqc -- How to modify a database (C)』
- 『tbread.sqC -- How to read tables (C++)』
- 『tut_use.sqC -- How to modify a database (C++)』

REFRESH TABLE

REFRESH TABLE ステートメントは、マテリアライズ照会表内のデータをリフレッシュします。

呼び出し:

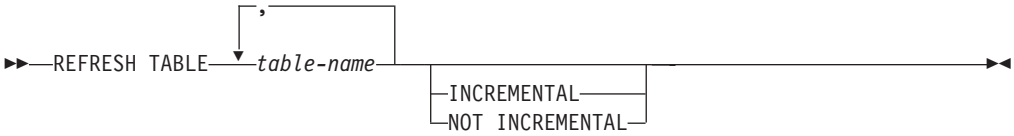
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- 表に対する CONTROL 特権

構文:



説明:

table-name
リフレッシュする表を指定します。

名前 (暗黙的または明示的なスキーマ名を含む) は、現行サーバーにすでに存在する表を指定していなければなりません。表は、REFRESH TABLE ステートメントを許可していなければなりません (SQLSTATE 42809)。これには、次のステートメントで定義したマテリアライズ照会表が含まれます。

- REFRESH IMMEDIATE
- REFRESH DEFERRED

INCREMENTAL
基礎表の追加部分 (ある場合) か、関連したステージング表の内容 (この表があり、内容が一貫している場合) だけを考慮する方法での、表の増分リフレッシュを指定します。この要求が満たされない場合 (たとえば、システムがマテリアライズ照会表定義を完全に再計算する必要があると判断する場合)、エラー (SQLSTATE 55019) が戻されます。

NOT INCREMENTAL
マテリアライズ照会表の定義を再計算する方法での、表のフル・リフレッシュを指定します。

INCREMENTAL と NOT INCREMENTAL をどちらも指定しない場合、システムは増分処理が可能かどうかを判断します。それが可能でなければ、フル・リフレッシュが実行されます。リフレッシュ対象のマテリアライズ照会表に関するステージング表がある場合に、ステージング表がペンディング状態のため増分処理ができないと、エラーが戻されます (SQLSTATE 428A8)。ステージング表かマテリアライズ照

会表が不整合な状態の場合は、フル・リフレッシュが実行されます。不整合でない場合は、ステージング表の内容を使用して増分処理が行われます。

注:

- 1 つ以上のニックネームを参照するマテリアライズ照会表に対して REFRESH TABLE を発行する場合は、リモート表から選択する権限が発行者になければなりません (SQLSTATE 42501)。
- このステートメントを使用して、基礎表がロード済みの REFRESH IMMEDIATE マテリアライズ照会表をリフレッシュする場合には、基礎表の追加部分を使用してマテリアライズ照会表の増分リフレッシュを行うようシステムが選択する場合があります。このステートメントを使用して、ステージング表をサポートしている REFRESH DEFERRED マテリアライズ照会表をリフレッシュする場合には、ステージング表にキャプチャーされた基礎表の追加部分を使用してマテリアライズ照会表の増分リフレッシュを行うようシステムが選択する場合があります。ただし、データの保全性を保証するために、この最適化を実行できずにフル・リフレッシュ (つまり、マテリアライズ照会表の定義の再計算) を行う必要が生じる場合もあります。ユーザーが INCREMENTAL オプションを指定して増分保守を明示的に要求することもできます。この最適化を実行できない場合は、システムはエラーを戻します。システムが増分処理を使用し、一貫性に関する責任はユーザーに負わせるというシナリオもあります。
- マテリアライズ照会表にステージング表が関連付けられている場合、リフレッシュが正常に実行されるとそのステージング表は整理されます。

関連資料:

- 743 ページの『SET INTEGRITY』
- 472 ページの『CREATE USER MAPPING』

RELEASE (接続)

RELEASE (接続) ステートメントは、1 つまたは複数の接続を解放ペンディング状態にします。

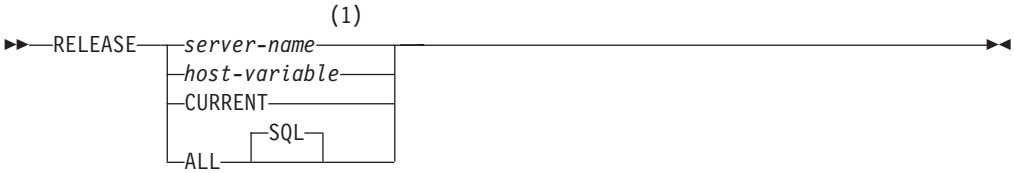
呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。

構文:



注:

1 CURRENT または ALL という名前のアプリケーション・サーバーは、ホスト変数または区切り ID を使用してのみ指定することができます。

説明:

server-name または *host-variable*

server-name (サーバー名) またはその *server-name* を含む *host-variable* (ホスト変数) によって、アプリケーション・サーバーを指定します。

host-variable (ホスト変数) を指定する場合、それは、長さ属性が 8 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。その *host-variable* に入る *server-name* は、左寄せする必要があり、引用符で区切ることはできません。

server-name は、アプリケーション・サーバーを指定するデータベース別名である点に注意してください。 この名前は、アプリケーション・リクエスターのローカル・ディレクトリーにリストされている必要があります。

指定されたデータベース別名、またはホスト変数に含まれているデータベース別名は、そのアプリケーション・プロセスの既存の接続を指定するものでなければなりません。 データベース別名が既存の接続を指定していない場合、エラー (SQLSTATE 08003) になります。

CURRENT

アプリケーション・プロセスの現行接続を指定します。 アプリケーション・プロセスは、接続された状態でなければなりません。 接続されていない場合、エラー (SQLSTATE 08003) になります。

ALL または ALL SQL

アプリケーション・プロセスの既存のすべての接続を指定します。 この形式の RELEASE ステートメントの使用により、アプリケーション・プロセスの既存のすべての接続が解放ペンディング状態になります。そのような場合、すべての接続は、次回のコミット操作の過程で破棄されることになります。 ステートメント実行時に接続が存在していない場合でも、エラーまたは警告のメッセージは出されません。

例:

例 1: IBMSTHDB への SQL 接続は、アプリケーションではもはや必要でなくなりました。以下のステートメントを実行すると、次のコミット操作の過程でその接続が破棄されることになります。

```
EXEC SQL RELEASE IBMSTHDB;
```

例 2: 現行の接続は、アプリケーションでもはや必要でなくなりました。以下のステートメントを実行すると、次のコミット操作の過程でその接続が破棄されることになります。

```
EXEC SQL RELEASE CURRENT;
```

例 3: アプリケーションがコミット後にデータベースにアクセスする必要がなく、実行はしばらく継続する場合、不必要に接続を続けないようにした方が得策です。コミット時にすべての接続が破棄されるようにするために、コミット前に次のステートメントを実行できます。

```
EXEC SQL RELEASE ALL;
```

RELEASE SAVEPOINT

RELEASE SAVEPOINT ステートメントは、指定されたセーブポイントの保持を、アプリケーションがもはや必要としなくなったことを指示するために使用されます。このステートメントが呼び出されると、そのセーブポイントまでロールバックすることはできなくなります。

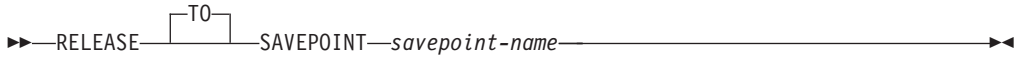
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

必要ありません。

構文:



説明:

savepoint-name

解放するセーブポイントを指定します。指名されたセーブポイント内でネストされているセーブポイントもすべて解放されます。そのセーブポイントおよびその内部でネストされているセーブポイントへのロールバックは不可能になります。現行のセーブポイント・レベルに名前付きセーブポイントがない場合 (SAVEPOINT ステートメントの『規則』 の節を参照) は、エラーが戻されます (SQLSTATE 3B001)。 *savepoint-name* を指定する際に、'SYS' で始めることはできません (SQLSTATE 42939)。

注:

- 同じセーブポイントの名前を指定した以前の SAVEPOINT ステートメントで UNIQUE キーワードが指定されたかどうかに関係なく、 1 度解放されたセーブポイントの名前は他の SAVEPOINT ステートメントでも再使用できるようになります。

例:

例 1: SAVEPOINT1 という名前のセーブポイントを解放します。

```
RELEASE SAVEPOINT SAVEPOINT1
```

関連資料:

- 704 ページの『SAVEPOINT』

RENAME

RENAME ステートメントは、既存の表または索引の名前を変更します。

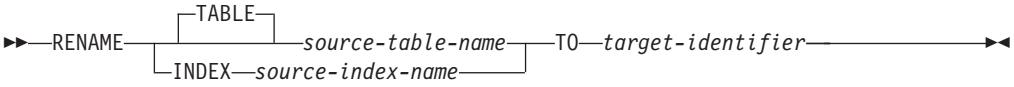
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID が持つ特権には、 SYSADM 権限か DBADM 権限、表か索引に対する CONTROL 特権、またはスキーマに対する ALTERIN のいずれかの特権が含まれている必要があります。

構文:



説明:

TABLE *source-table-name*
名前を変更したい既存の表を指定します。名前 (スキーマ名を含む) は、データベースにすでに存在する表を指定していなければなりません (SQLSTATE 42704)。この名前に、カタログ表 (SQLSTATE 42832)、マテリアライズ照会表、型付き表 (SQLSTATE 42997)、宣言されたグローバル一時表 (SQLSTATE 42995)、ニックネーム、または表や別名以外のオブジェクト (SQLSTATE 42809) を指定することはできません。 TABLE キーワードはオプションです。

INDEX *source-index-name*
名前を変更したい既存の索引を指定します。 名前 (スキーマ名を含む) は、データベースにすでに存在する索引を指定していなければなりません (SQLSTATE 42704)。宣言されたグローバル一時表上の索引の名前は指定できません (SQLSTATE 42995)。スキーマ名は SYSIBM、SYSCAT、SYSFUN、または SYSSTAT であってはなりません (SQLSTATE 42832)。

target-identifier
表または索引の新しい名前をスキーマ名を付けずに指定します。 ソース・オブジェクトのスキーマ名が、オブジェクトの新しい名前の修飾に使用されます。修飾された名前が、データベースにすでに存在する表、ビュー、別名、または索引を指定するものであってはなりません (SQLSTATE 42710)。

規則:

表の名前を変更する場合、ソース表は以下に該当していてはなりません。

- 既存のビュー定義またはマテリアライズ照会表定義で参照されている
- 既存のトリガーのトリガー SQL ステートメントで参照されているか、既存のトリガーの対象の表である
- SQL 関数で参照されている

- チェック制約がある
- ID 列以外に生成列がある
- 参照保全制約における親表または従属表である
- 既存の参照列の有効範囲内である

ソース表が上記の条件の 1 つまたは複数に違反している場合、エラー (SQLSTATE 42986) が戻されます。

索引の名前を変更する場合:

- 型付き表の基になっているインプリメンテーション表のシステム生成索引を、ソース索引にすることはできません (SQLSTATE 42858)。

注:

- カタログ項目が更新され、新しい表名または索引名が反映されます。
- ソース表名または索引名に関連するすべての 許可は、新しい表名または索引名に転送 されます (許可カタログ表が適切に更新されます)。
- ソース表に対して定義された索引は、新しい表に転送 されます (索引カタログ表が適切に更新されます)。
- RENAME TABLE を行うと、ソース表に従属するパッケージはいずれも無効になります。 RENAME INDEX を行うと、ソース索引に従属するパッケージはいずれも無効になります。
- *source-table-name* として別名を使用する場合、その別名は表名に解決されなければなりません。表の名前は、その表のスキーマの中で変更されます。別名は RENAME ステートメントによって変更されず、従来の表名を引き続き指します。
- 主キー制約またはユニーク制約のある表の名前は、主キーまたはユニーク制約がいずれも外部キーによって参照されていない場合に変更できます。

例:

EMP 表の名前を EMPLOYEE に変更します。

```
RENAME TABLE EMP TO EMPLOYEE
RENAME TABLE ABC.EMP TO EMPLOYEE
```

索引 NEW-IND の名前を IND に変更します。

```
RENAME INDEX NEW-IND TO IND
RENAME INDEX ABC.NEW-IND TO IND
```

RENAME TABLESPACE

RENAME TABLESPACE ステートメントは、既存の表スペースの名前を変更します。

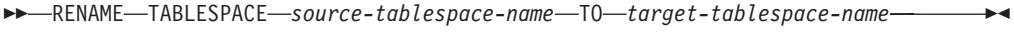
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID が持つ特権には、 SYSADM 権限か SYSCTRL 権限のいずれかが含まれている必要があります。

構文:



説明:

source-tablespace-name
1 つの部分からなる名前で、名前を変更する既存の表スペースを指定します。これは、SQL ID です (通常 ID または区切り ID)。表スペース名は、カタログ内にすでに存在している表スペースを識別するものでなければなりません (SQLSTATE 42704)。

target-tablespace-name
表スペースに 1 つの部分からなる新しい名前を指定します。これは、SQL ID です (通常 ID または区切り ID)。新しく指定する表スペース名は、カタログ内にすでに存在する表スペースを識別するものであってはならず (SQLSTATE 42710)、また、'SYS' から始まる名前を指定することもできません (SQLSTATE 42939)。

規則:

- SYSCATSPACE 表スペースの名前を変更することはできません (SQLSTATE 42832)。
- 「ロールフォワード・ペンディング」状態または「ロールフォワード進行中」状態にある表スペースの名前は変更できません (SQLSTATE 55039)。

注:

- 表スペースの名前を変更すると、表スペースの最短のリカバリー時間が、名前変更の行われた時点に更新されます。これにより、表スペースのレベルでロールフォワードを実行すると、最低でもこの時点までロールフォワードされることになります。
- バックアップの作成後にバックアップ・イメージで名前の変更を行った場合は、バックアップ・イメージから表スペースをリストアするときに、新しい表スペース名を使用する必要があります。

例:

表スペースの名前 USERSPACE1 を DATA2000 に変更します。

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

REPEAT

REPEAT ステートメントは、検索条件が真になるまでステートメントまたはステートメントのグループを実行します。

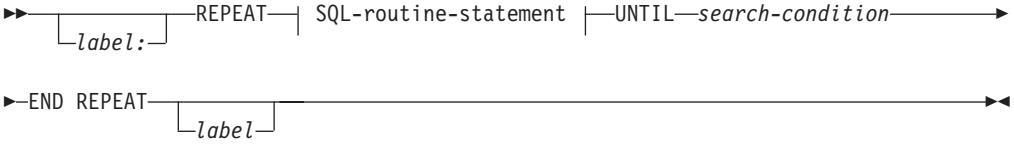
呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

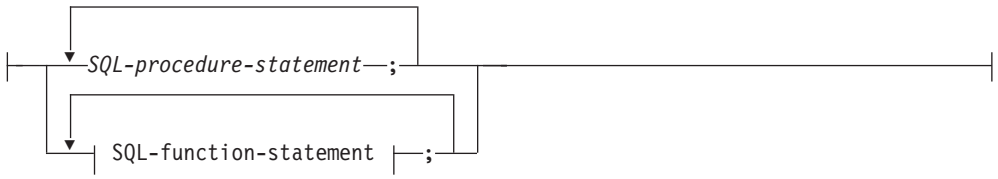
許可:

REPEAT ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 REPEAT ステートメントに組み込まれている SQL ステートメントおよび検索条件を呼び出すために必要な特権がなければなりません。

構文:



SQL-routine-statement:



説明:

label
REPEAT ステートメントのラベルを指定します。開始ラベルを指定した場合、そのラベルを LEAVE および ITERATE ステートメントで指定することができます。終了ラベルを指定する場合、一致する開始ラベルも指定しなければなりません。

SQL-procedure-statement
ループ内で実行する SQL ステートメントを指定します。
SQL-procedure-statement は、SQL プロシージャーのコンテキスト内でのみ使用できます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

SQL-function-statement
ループ内で実行する SQL ステートメントを指定します。 *SQL-function-statement* は、SQL 関数または SQL メソッドのコンテキスト内でのみ使用できます。
FOR ステートメントの説明にある SQL-function-statement の項目を参照してください。

search-condition

search-condition は、毎回、REPEAT ループの実行後に評価されます。条件が真であれば、ループは終了します。条件が不明または偽であれば、ループは続行されます。

例:

REPEAT ステートメントは、*not_found* 条件ハンドラーが呼び出されるまで、表から行を取り出します。

```
CREATE PROCEDURE REPEAT_STMT(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE at_end SMALLINT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
      OPEN c1;
    fetch_loop:
    REPEAT
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
      SET v_counter = v_counter + 1;
      UNTIL at_end > 0
    END REPEAT fetch_loop;
    SET counter = v_counter;
    CLOSE c1;
  END
```

関連資料:

• 140 ページの『コンパウンド SQL (プロシージャー)』

RESIGNAL

RESIGNAL ステートメントは、エラーまたは警告条件を再通知するのに使用します。これを使用すると、指定した SQLSTATE とオプションのメッセージ・テキストが、エラーまたは警告とともに戻されます。

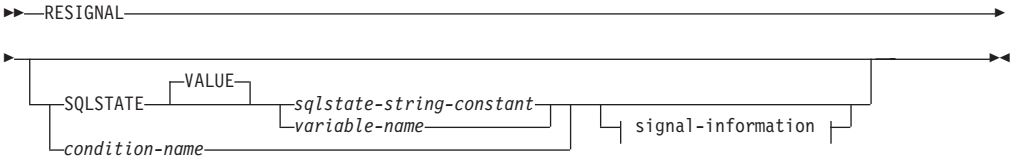
呼び出し:

このステートメントは、SQL プロシージャーに組み込む方法でのみ使用可能です。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

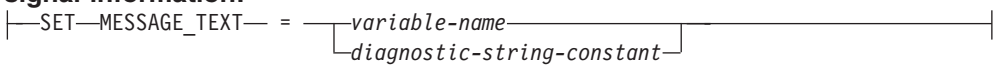
許可:

必要ありません。

構文:



signal-information:



説明:

SQLSTATE VALUE *sqlstate-string-constant*
指定されたストリング定数が SQLSTATE を表します。この定数は、正確に 5 文字の文字ストリング定数でなければならず、 SQLSTATE の規則に従っていなければなりません。

- 各文字は、数字 ('0'〜'9')、またはアクセントのない大文字の英字 ('A'〜'Z') でなければなりません。
- SQLSTATE クラス (最初の 2 文字) は '00' にはできません。これは正常な完了を示します。

SQLSTATE がこれらの規則に従っていない場合には、エラーになります (SQLSTATE 428B3)。

SQLSTATE VALUE *variable-name*
指定する変数名はタイプ CHAR(5) でなければなりません。ステートメント実行時のこの変数の値が、*sqlstate-string-constant* に記述されているものと同じ規則に準拠している必要があります。 SQLSTATE がこれらの規則に従っていない場合、エラーが戻されます (SQLSTATE 428B3)。

condition-name
条件の名前を指定します。

SET MESSAGE_TEXT =
エラーまたは警告を記述するストリングを指定します。ストリングは SQLCA

の sqlerrmc フィールドに返されます。実際のストリングが 70 バイトを超えている場合は、警告なしで切り捨てられます。

variable-name
コンパウンド・ステートメント内で宣言される必要のある SQL 変数を識別します。 SQL 変数は CHAR または VARCHAR データ・タイプとして定義されていなければなりません。

diagnostic-string-constant
メッセージ・テキストを含む文字ストリング定数を指定します。

注:

- SQLSTATE 文節または *condition-name* を使用せずに RESIGNAL ステートメントを指定すると、ハンドラーを呼び出したのと同じ条件が戻されます。この条件と関連付けられた SQLSTATE、SQLCODE および SQLCA は変更されません。
- RESIGNAL ステートメントが発行され、SQLSTATE または *condition-name* が指定された場合、以下のように、戻される SQLCODE は SQLSTATE 値に基づいています。
  - 指定した SQLSTATE クラスが '01' か '02' のいずれかである場合、警告か、見つからないことを示す条件が戻され、SQLCODE は +438 に設定されます。
  - それ以外の場合、例外条件が戻され、SQLCODE は -438 に設定されます。

  SQLCA の他のフィールドは、以下のように設定されます。
  - sqlerrd フィールドはゼロに設定されます
  - sqlwarn フィールドはブランクに設定されます
  - sqlerrmc は MESSAGE_TEXT の先頭の 70 バイトに設定されます
  - sqlerrml は sqlerrmc の長さか、 SET MESSAGE_TEXT 文節が指定されていない場合にはゼロに設定されます
  - sqlerrp は ROUTINE に設定されます
- SQLSTATE 値の詳細は、『SIGNAL ステートメント』の『注』を参照してください。

例:

以下の例では、ゼロ除算によるエラーを検出します。 IF ステートメントは、SIGNAL ステートメントを使用して *overflow* 条件ハンドラーを呼び出します。その条件ハンドラーは、 RESIGNAL ステートメントを使用して別の SQLSTATE 値をクライアント・アプリケーションに戻します。

```
CREATE PROCEDURE divide ( IN numerator INTEGER,
                          IN denominator INTEGER,
                          OUT result INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE overflow CONDITION FOR SQLSTATE '22003';
  DECLARE CONTINUE HANDLER FOR overflow
    RESIGNAL SQLSTATE '22375';
  IF denominator = 0 THEN
    SIGNAL overflow;
  ELSE
    SET result = numerator / denominator;
  END IF;
END
```

関連資料:

- 776 ページの『SIGNAL』

RETURN

RETURN ステートメントはルーチンから戻るために使用されます。 SQL 関数またはメソッドの場合、関数またはメソッドの結果を返します。 SQL プロシージャーの場合、オプションで整数状況値が戻されます。

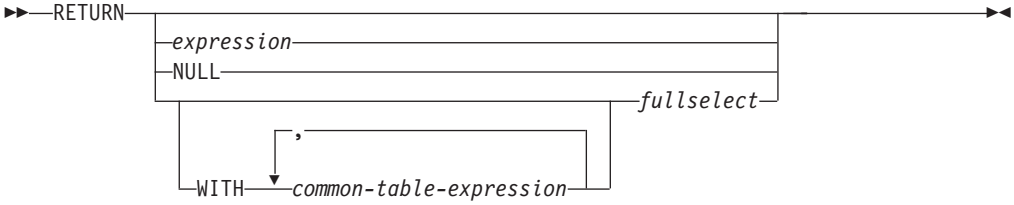
呼び出し:

このステートメントは、SQL 関数、SQL メソッド、または SQL プロシージャーに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

RETURN ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 RETURN ステートメントに組み込まれている式または全選択を呼び出すために必要な特権がなければなりません。

構文:



説明:

expression
ルーチンから戻される値を指定します。

- ルーチンが関数またはメソッドの場合は、*expression*、NULL または *fullselect* の指定が必要 (SQLSTATE 42630) であり、結果のデータ・タイプはルーチンの RETURNS タイプに割り当て可能でなければなりません (SQLSTATE 42866)。
- ルーチンが表関数の場合は、スカラー式 (スカラー fullselect 以外) は定義できません (SQLSTATE 428F1)。
- ルーチンがプロシージャーの場合は、*expression* のデータ・タイプは INTEGER でなければなりません (SQLSTATE 428E2)。プロシージャーは NULL または *fullselect* を返すことができません。

NULL
関数またはメソッドが、RETURNS 文節で定義されたデータ・タイプの NULL 値を返すことを指定します。 NULL はプロシージャーからの RETURN には指定できません。

WITH *common-table-expression*
後続の *fullselect* で使用する共通表式を定義します。

fullselect
関数に対して返される行を指定します。 *fullselect* 内の列数は、関数の結果の列数に一致していなければなりません (SQLSTATE 42811)。さらに、

fullselect の静的列タイプが、列に対する割り当て規則を使用して関数結果について宣言された列タイプに割り当てられていなければなりません (SQLSTATE 42866)。

fullselect はプロシージャーからの RETURN には指定できません。

ルーチンがスカラー関数またはメソッドの場合、*fullselect* は 1 つの列 (SQLSTATE 42823) と、 1 つの行 (SQLSTATE 21000) を返さなければなりません。

ルーチンが行関数の場合は、1 つの行 (SQLSTATE 21505) を返さなければなりません。ただし、1 つ以上の列が戻されることがあります。

ルーチンが表関数の場合は、1 つまたは複数の列を持つゼロ以上の行を返すことができます。

規則:

- SQL 関数またはメソッドの実行は RETURN ステートメントで終わっていなければなりません (SQLSTATE 42632)。
- *dynamic-compound-statement* を使用する SQL 表または行関数では、 RETURN ステートメントのみがコンパウンド・ステートメントの終わりで指定できます (SQLSTATE 429BD)。

注:

- プロシージャーから値が返されると、呼び出し元は以下のようにして値にアクセスすることができます。
  - SQL プロシージャーが他の SQL プロシージャーから呼び出されたときに RETURN_STATUS を検索する GET DIAGNOSTICS ステートメントを使用する。
  - CLI アプリケーションでエスケープ文節 CALL 構文 (?=CALL...) にある戻り値パラメーター・マーカーに結合されたパラメーターを使用する。
  - SQL プロシージャーの CALL の処理後に SQLCA の sqlerrd[0] フィールドから直接。このフィールドは、SQLCODE がゼロまたは正の場合にのみ有効です (これ以外の場合は -1 の値と見なされます)。

例:

RETURN ステートメントを使用して、SQL ストアード・プロシージャーから状況値を戻します。成功した場合は値ゼロが、失敗した場合は -200 が戻されます。

```
BEGIN
...
   GOTO FAIL
...
   SUCCESS: RETURN 0
   FAIL: RETURN -200
END
```

REVOKE (データベース権限)

この形式の REVOKE ステートメントは、データベース全体に適用される権限を取り消します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

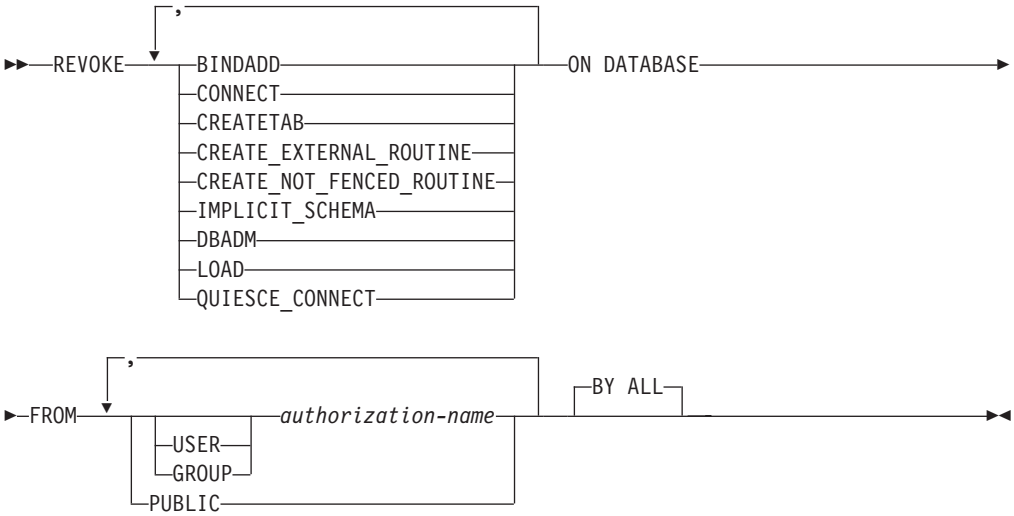
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- DBADM 権限
- SYSADM 権限

DBADM 権限を取り消すには、SYSADM 権限が必要です。

構文:



説明:

BINDADD
パッケージを作成する権限を取り消します。 パッケージの作成者には自動的にそのパッケージに対する CONTROL 特権が与えられ、後でその BINDADD 権限が取り消されたとしてもその特権はそのまま保持されます。

DBADM 権限も取り消すのでない限り、 DBADM 権限を与えられている *authorization-name* から BINDADD 権限を取り消すことはできません。

CONNECT
データベースにアクセスする権限を取り消します。

ユーザーから CONNECT 権限を取り消しても、そのユーザーに付与されていたデータベースのオブジェクトに対する特権には影響しません。 後で再度そのユ

ーザーに CONNECT 権限が付与された場合でも、以前に持っていた特権は、明示的に取り消されたのでない限り、依然としてすべて有効です。

DBADM 権限も取り消すのでない限り、 DBADM 権限を与えられている *authorization-name* から CONNECT 権限を取り消すことはできません (SQLSTATE 42504)。

CREATETAB
表を作成する権限を取り消します。 表の作成者には自動的にその表に対する CONTROL 特権が与えられ、後で CREATETAB 権限が取り消されたとしても、その特権はそのまま保持します。

DBADM 権限も取り消すのでない限り、 DBADM 権限を与えられている *authorization-name* から CREATETAB 権限を取り消すことはできません (SQLSTATE 42504)。

CREATE_EXTERNAL_ROUTINE
外部ルーチンを登録する権限を取り消します。外部ルーチンを登録し終えると、それ以降にそのルーチンを登録した許可 ID から CREATE_EXTERNAL_ROUTINE が取り消されたとしてもそのまま保持されます。

DBADM または CREATE_NOT_FENCED_ROUTINE 権限も取り消すのでない限り、 DBADM または CREATE_NOT_FENCED_ROUTINE 権限を与えられている*許可 ID* から CREATE_EXTERNAL_ROUTINE 権限を取り消すことはできません (SQLSTATE 42504)。

CREATE_NOT_FENCED_ROUTINE
データベース・マネージャーの処理の中で実行するルーチンを登録する権限を取り消します。ルーチンが非 fenced としていったん登録されると、それ以降にそのルーチンを登録した許可 ID から CREATE_NOT_FENCED_ROUTINE が取り消されるとしても、そのまま実行が続けられます。

DBADM 権限も取り消すのでない限り、 DBADM 権限を与えられている *authorization-name* から CREATE_NOT_FENCED_ROUTINE 権限を取り消すことはできません (SQLSTATE 42504)。

IMPLICIT_SCHEMA
スキーマを暗黙的に作成する権限を取り消します。既存のスキーマにオブジェクトを作成する権限、または CREATE SCHEMA ステートメントを処理する権限には影響しません。

DBADM
DBADM 権限を取り消します。

DBADM 権限を PUBLIC から取り消すことはできません (PUBLIC に対して与えることができないので、当然取り消しもできません)。

注意:
DBADM 権限の取り消しによって、データベース内のオブジェクトに対して *authorization-name* が持っていた特権が自動的に取り消されることはなく、また元々 DBADM 権限が付与された際に暗黙的および自動的に付与された他のデータベース権限も取り消されることもありません。

LOAD
このデータベースで LOAD を実行する権限を取り消します。

QUIESCE_CONNECT
静止中のデータベースにアクセスする権限を取り消します。

FROM
権限を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から権限を取り消すことはできません。

PUBLIC
PUBLIC から該当の権限を取り消します。

BY ALL
指定された個々の特権を、その付与者にかかわらず、それらの特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.DBAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合には、 USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

注:

- ***互換性***

  バージョン 8 より前のバージョンとの互換性を保つ場合は、オプション CREATE_NOT_FENCED を CREATE_NOT_FENCED_ROUTINE に置き換えることができます。
- 特定の特権の取り消しにより、アクションを実行する権限が取り消されるとは限りません。 PUBLIC またはグループが他の特権を持っている場合、またはユーザーが DBADM などのより上位の権限を持っている場合は、ユーザーは作業を続行できます。

例:

例 1: USER6 はユーザーであり、グループではない場合に、ユーザー USER6 の表を作成する特権を取り消します。

```
REVOKE CREATETAB ON DATABASE FROM USER6
```

例 2: D024 という名前のグループのデータベースに対する BINDADD 権限を取り消します。 SYSCAT.DBAUTH カタログ・ビューには、このグループの行として 2 つの行があります。その 1 つでは GRANTEETYPE が U、もう 1 つでは GRANTEETYPE が G になっています。

```
REVOKE BINDADD ON DATABASE FROM GROUP D024
```

この場合、GROUP キーワードの指定は必須です。そうしないとエラーになります (SQLSTATE 56092)。

関連資料:

- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

関連サンプル:

- 『dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)』
- 『dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)』
- 『DbAuth.java -- Grant, display or revoke privileges on database (JDBC)』
- 『DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)』

REVOKE (索引特権)

この形式の REVOKE ステートメントは、索引に対する CONTROL 特権を取り消します。

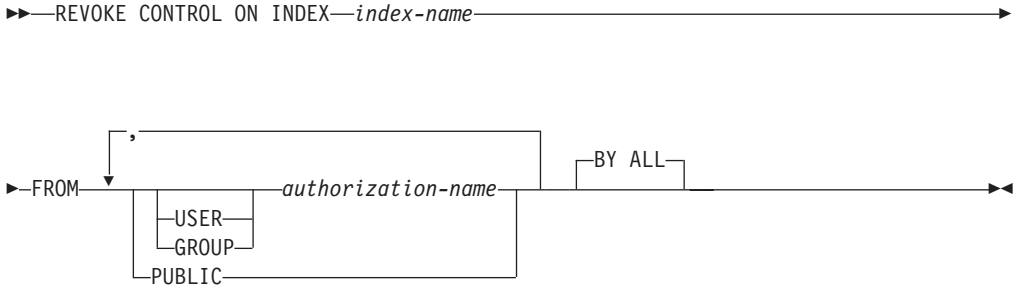
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID は、 SYSADM または DBADM のいずれかの権限を持っている必要があります (SQLSTATE 42501)。

構文:



説明:

CONTROL
索引をドロップする特権を取り消します。これは、索引に対する CONTROL 特権であり、この特権は索引の作成者に自動的に付与されます。

ON INDEX *index-name*
その CONTROL 特権を取り消す索引の名前を指定します。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
PUBLIC から特権を取り消します。

BY ALL
特権の付与者にかかわらず、その特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.INDEXAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合には、 USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

注:

- 特定の特権の取り消しにより、アクションを実行する権限が取り消されるとは限りません。 PUBLIC またはグループが他の特権を持っている場合、またはユーザーが索引のスキーマに対する ALTERIN などの権限を持っている場合は、ユーザーは作業を続行できます。

例:

例 1: USER4 はユーザーであり、グループではない場合に、ユーザー USER4 から索引 DEPTIDX をドロップする特権を取り消します。

```
REVOKE CONTROL ON INDEX DEPTIDX FROM USER4
```

例 2: ユーザー CHEF とグループ WAITERS から、索引 LUNCHITEMS をドロップする特権を取り消します。

```
REVOKE CONTROL ON INDEX LUNCHITEMS
  FROM USER CHEF, GROUP WAITERS
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (パッケージ特権)

この形式の REVOKE ステートメントは、パッケージに対する CONTROL、BIND、および EXECUTE 特権を取り消します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

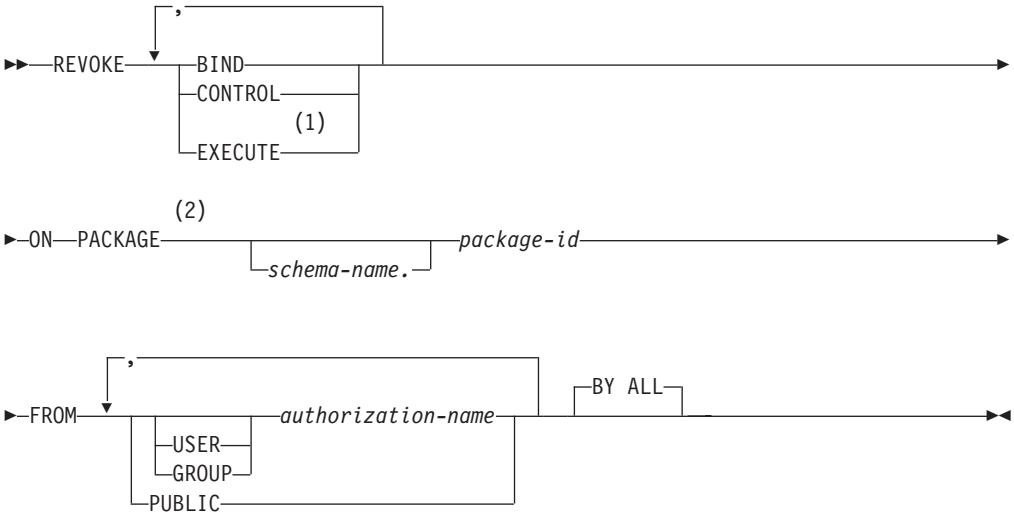
許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 参照されるパッケージに対する CONTROL 特権
- SYSADM または DBADM 権限

CONTROL 特権を取り消すには、SYSADM または DBADM の権限が必要です。

構文:



注:

1 EXECUTE の同義語として RUN を使用できます。

2 PACKAGE の同義語として PROGRAM を使用できます。

説明:

BIND

指定されたパッケージに対する BIND または REBIND を実行する特権か、または指定されたパッケージの新しいバージョンを追加する特権を取り消します。

CONTROL 特権も取り消すのでない限り、パッケージに対する CONTROL 特権を与えられている *authorization-name* から BIND 特権を取り消すことはできません。

CONTROL
パッケージをドロップする特権、および他のユーザーに対してパッケージの特権を拡張する特権を取り消します。

CONTROL を取り消しても、他のパッケージ特権は取り消されません。

EXECUTE
パッケージを実行する特権を取り消します。

CONTROL 特権も取り消すのでない限り、パッケージに対する CONTROL 特権を与えられている *authorization-name* から EXECUTE 特権を取り消すことはできません。

ON PACKAGE *schema-name.package-id*
特権を取り消す対象のパッケージの名前を指定します。スキーマ名が指定されていない場合、パッケージ ID は暗黙的にデフォルト・スキーマで修飾されます。パッケージ特権の取り消しは、すべてのバージョンのパッケージに適用されます。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
PUBLIC から特権を取り消します。

BY ALL
指定された個々の特権を、その付与者にかかわらず、それらの特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.PACKAGEAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合、USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

注:

- 特定の特権の取り消しにより、アクションを実行する権限が取り消されるとは限りません。 PUBLIC またはグループが他の特権を持っている場合、またはユーザーがパッケージのスキーマに対する ALTERIN などの特権を持っている場合は、ユーザーは作業を続行できます。

例:

例 1: PUBLIC から、パッケージ CORPDATA.PKGA に対する EXECUTE 権限を取り消します。

```
REVOKE EXECUTE
  ON PACKAGE CORPDATA.PKGA
  FROM PUBLIC
```

例 2: ユーザー FRANK および PUBLIC から、 RRSP_PKG パッケージに対する CONTROL 権限を取り消します。

```
REVOKE CONTROL
  ON PACKAGE RRSP_PKG
  FROM USER FRANK, PUBLIC
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (ルーチン特権)

この形式の REVOKE ステートメントは、ルーチン (関数、メソッド、またはプロシージャー) に対する特権を取り消します。

呼び出し:

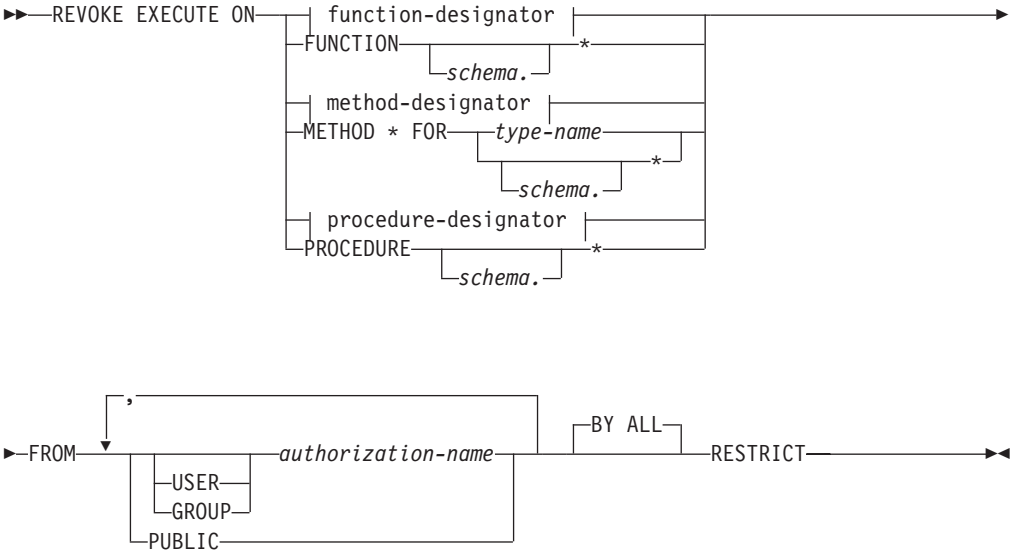
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限

構文:



説明:

EXECUTE
識別されたユーザー定義の関数、メソッド、またはストアード・プロシージャーを実行する特権を取り消します。

function-designator
関数を一意的に識別します。

FUNCTION *schema.**
スキーマ中の既存の関数と将来作成される関数に関する明示的な権限付与を識別します。 *schema.** 特権を取り消しても、特定の関数に付与された特権は取り消されません。動的 SQL ステートメント中でスキーマが指定されていない場合は、 CURRENT SCHEMA 特殊レジスター中のスキーマが使用されます。静的 SQL ステートメント中でスキーマが指定されていない場合は、 QUALIFIER プリコンパイル/ BIND オプション中のスキーマが使用されます。

method-designator
メソッドを一意的に識別します。

METHOD *
タイプ *type-name* の既存のメソッドと将来作成されるメソッドに関する明示的な権限付与を識別します。 * 特権を取り消しても、特定のメソッドに付与された特権は取り消されません。

FOR *type-name*
指定されたメソッドを検索する際のタイプを指定する。ここで指定される名前は、カタログにすでに記述されているタイプを示すものでなければなりません (SQLSTATE 42704)。動的 SQL ステートメントでは、 CURRENT SCHEMA 特殊レジスターの値が、修飾子のないタイプ名の修飾子として使用されます。静的 SQL ステートメントでは、QUALIFIER プリコンパイル/ BIND オプションによって、修飾子のないタイプ名に修飾子が暗黙指定されます。 *type-name* の代わりにアスタリスク (*) を使用して、スキーマ中のすべての既存のタイプと将来作成されるタイプの、すべての既存のメソッドと将来作成されるメソッドに対する明示的な権限付与を識別することもできます。アスタリスクを使用したメソッドおよび *type-name* に関する特権を取り消しても、特定のメソッドまたは特定のタイプのすべてのメソッドに付与された特権は取り消されません。

procedure-designator
プロシージャーを一意的に識別します。

PROCEDURE *schema.**
スキーマ中の既存のプロシージャーと将来作成されるプロシージャーに関する明示的な権限付与を識別します。 *schema.** 特権を取り消しても、特定のプロシージャーに付与された特権は取り消されません。動的 SQL ステートメント中でスキーマが指定されていない場合は、 CURRENT SCHEMA 特殊レジスター中のスキーマが使用されます。静的 SQL ステートメント中でスキーマが指定されていない場合は、 QUALIFIER プリコンパイル/ BIND オプション中のスキーマが使用されます。

FROM
EXECUTE 特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。この許可 ID のリストに、このステートメントを出すユーザーの許可 ID を含めることはできません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーから EXECUTE 特権を取り消します。

BY ALL
EXECUTE 特権の付与者にかかわらず、 EXECUTE 特権を明示的に付与されたユーザーのうち、指定された人から取り消します。これはデフォルトの動作です。

RESTRICT
以下の両方が該当する場合に、EXECUTE 特権を取り消せないことを指定します (SQLSTATE 42893)。

- 指定されたルーチンがビュー、トリガー、制約、索引拡張、SQL 関数、SQL メソッド、またはトランスフォーム・グループ中で使用されているか、または指定されたルーチンがソース関数の SOURCE として参照されている。
- EXECUTE 特権がなくなると、ビュー、トリガー、制約、索引拡張、SQL 関数、SQL メソッド、トランスフォーム・グループ、またはソース関数の定義者が、指定されたルーチンを実行できなくなる。

規則:

- スキーマ 'SYSIBM' または 'SYSFUN' を使って定義された関数やメソッドに対する EXECUTE 特権を取り消すことはできません (SQLSTATE 42832)。
- USER も GROUP も指定しない場合には、
  - SYSCAT.ROUTINEAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合、 USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

例:

例 1: ユーザー JONES から、関数 CALC_SALARY に対する EXECUTE 特権を取り消します。スキーマ中に CALC_SALARY という名前の関数が 1 つだけ含まれていると想定しています。

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES RESTRICT
```

例 2: 現行サーバー上のすべてのユーザーから、プロシージャー VACATION_ACCR に対する EXECUTE 特権を取り消します。

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC RESTRICT
```

例 3: HR (Human Resources) から、関数 NEW_DEPT_HIRES に対する EXECUTE 特権を取り消します。この関数には、2 つの入力パラメーターがあり、それぞれのパラメーターのタイプは INTEGER および CHAR(10) です。スキーマに NEW_DEPT_HIRES という名前の関数が複数あることを想定しています。

```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
  FROM HR RESTRICT
```

例 4: ユーザー Jones から、タイプ EMPLOYEE のメソッド SET_SALARY に対する EXECUTE 特権を取り消します。

```
REVOKE EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE FROM JONES RESTRICT
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- x ページの『共通の構文エレメント』

REVOKE (スキーマ特権)

この形式の REVOKE ステートメントは、スキーマに対する特権を取り消します。

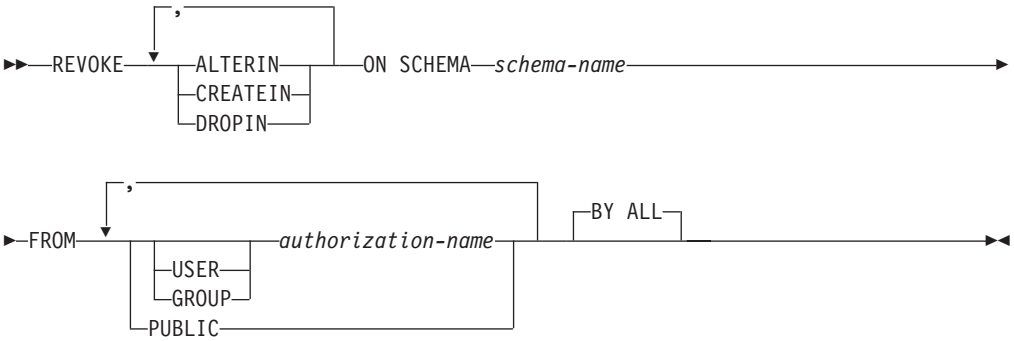
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID は、 SYSADM または DBADM のいずれかの権限を持っている必要があります (SQLSTATE 42501)。

構文:



説明:

ALTERIN
スキーマ中のオブジェクトの変更、またはコメント付けを行う特権を取り消します。

CREATEIN
スキーマにオブジェクトを作成する特権を取り消します。

DROPIN
スキーマのオブジェクトをドロップする特権を取り消します。

ON SCHEMA *schema-name*
特権を取り消す対象のスキーマの名前を指定します。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
PUBLIC から特権を取り消します。

BY ALL
指定された個々の特権を、その付与者にかかわらず、それらの特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.SCHEMAAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合、 USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

注:

- 特定の特権の取り消しにより、アクションを実行する権限が取り消されるとは限りません。 PUBLIC またはグループが他の特権を持っている場合、またはユーザーが DBADM などのより上位の権限を持っている場合は、ユーザーは作業を続行できます。

例:

例 1: USER4 がユーザーで、グループではない場合に、ユーザー USER4 からスキーマ DEPTIDX にオブジェクトを作成する特権を取り消します。

```
REVOKE CREATEIN ON SCHEMA DEPTIDX FROM USER4
```

例 2: ユーザー CHEF とグループ WAITERS から、スキーマ LUNCH のオブジェクトをドロップする特権を取り消します。

```
REVOKE DROPIN ON SCHEMA LUNCH
  FROM USER CHEF, GROUP WAITERS
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (シーケンス特権)

この形式の REVOKE ステートメントは、シーケンスに対する特権を取り消します。

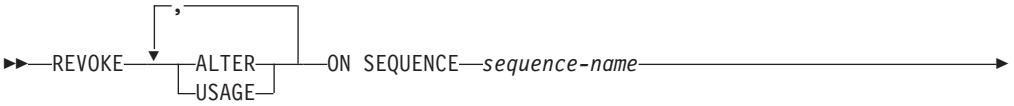
呼び出し:

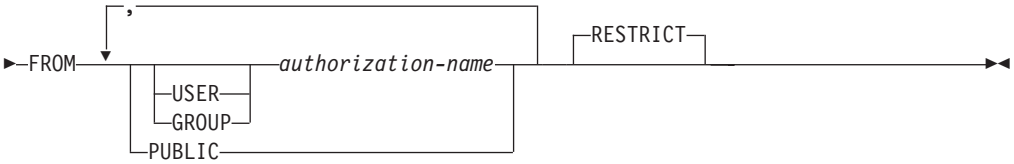
このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。ただし、BIND オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可:

このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文:





説明:

ALTER
ALTER SEQUENCE ステートメントを使用して、シーケンスのプロパティーを変更する特権、またはシーケンス番号の生成を再始動する特権を取り消します。

USAGE
nextval-expression または *prevval-expression* を使用してシーケンスを参照する特権を取り消します。

ON SEQUENCE *sequence-name*
指定された特権が取り消されるシーケンスを識別します。暗黙的または明示的スキーマ修飾子を含むシーケンス名は、現在のサーバーに存在するシーケンスを固有に識別していなければなりません。この名前によるシーケンスが存在しない場合、エラーが戻されます (SQLSTATE 42704)。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループであることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。 REVOKE ステートメント自体の許可 ID は指定できません (SQLSTATE 42502)。

PUBLIC
すべてのユーザーから指定された特権を取り消します。

RESTRICT
このオプショナル・キーワードは、取り消される特権に依存するオブジェクトがある場合、ステートメントが失敗することを示します。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.SEQUENCEAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合、 USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と G の行が混在している場合は、エラーが戻されます (SQLSTATE 56092)。

注:

- 特定の特権を取り消しても、アクションを実行する権限が必ずしも取り除かれるとは限りません。 PUBLIC またはユーザーの属するグループに他の特権が与えられている場合、あるいは DBADM などのより上位の権限をユーザーが持っている場合には、ユーザーは作業を続行できます。

例:

例 1: シーケンス GENERATE_ID に対する USAGE 特権をユーザー ENGLES から取り消します。 SYSCAT.SEQUENCEAUTH カタログ・ビューにはこのシーケンスとユーザーについての行が 1 つあり、その GRANTEETYPE の値は U です。

```
REVOKE USAGE ON SEQUENCE GENERATE_ID FROM ENGLES
```

例 2: 以前にすべてのローカル・ユーザーに与えられたシーケンス GENERATE_ID に対する更新特権を取り消します。 (特定のユーザーに対する特権付与は、影響を受けません。)

```
REVOKE ALTER ON SEQUENCE GENERATE_ID FROM PUBLIC
```

例 3: シーケンス GENERATE_ID に対するすべての特権を、ユーザー PELLOW と MLI、およびグループ PLANNERS から取り消します。

```
REVOKE ALTER, USAGE ON SEQUENCE GENERATE_ID
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』

- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 596 ページの『GRANT (シーケンス特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (サーバー特権)

この形式の REVOKE ステートメントは、指定したデータ・ソースにパススルー・モードでアクセスおよび使用する特権を取り消します。

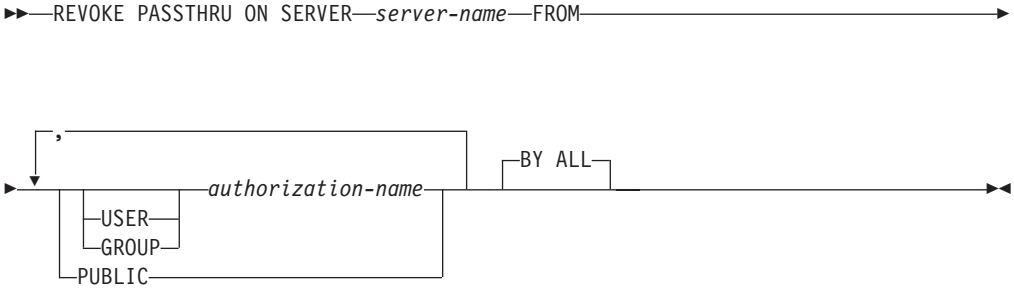
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID には、SYSADM 権限または DBADM 権限がなければなりません。

構文:



説明:

SERVER *server-name*
パススルー・モードで使用する特権が取り消されるデータ・ソースを指定します。 *server-name* (サーバー名) は、カタログに記述されているデータ・ソースを指定していなければなりません。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 人または複数のユーザーまたはグループの許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
server-name にパススルーする特権をすべてのユーザーから取り消します。

BY ALL
特権の付与者にかかわらず、その特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

例:

例 1: USER6 が持っているデータ・ソース MOUNTAIN にパススルーする特権を取り消します。

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

例 2: グループ D024 が持っている、データ・ソース EASTWING にパススルーする特権を取り消します。

```
REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024
```

グループ D024 のメンバーは、このグループ ID を使って EASTWING にパススルーすることはできなくなります。しかし、EASTWING にパススルーする特権をユーザー ID に持っているメンバーがいれば、それらのメンバーはこの特権を保持することができます。

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (表スペース特権)

この形式の REVOKE ステートメントは、表スペースに対する USE 特権を取り消します。

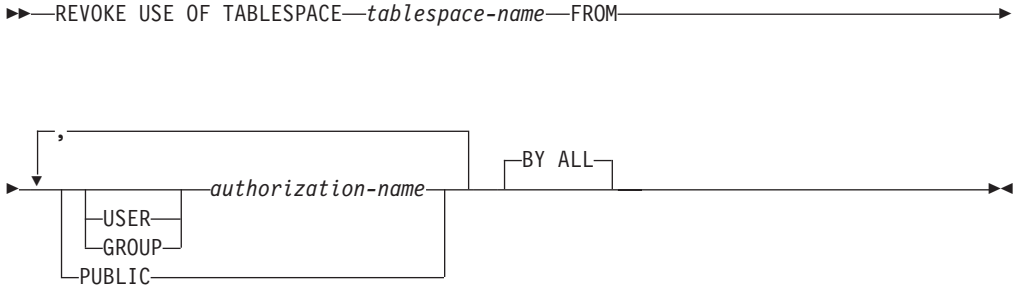
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID には、SYSADM、SYSCTRL、または DBADM のいずれかの権限がなければなりません (SQLSTATE 42501)。

構文:



説明:

USE
表を作成する際に表スペースを指定したり、デフォルトの表スペースを使用したりするための特権を取り消します。

OF TABLESPACE *tablespace-name*
どの表スペースに対する USE 特権を取り消すかを指定します。ここで、SYSCATSPACE (SQLSTATE 42838) や SYSTEM TEMPORARY 表スペース (SQLSTATE 42809) を指定することはできません。

FROM
USE 特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の許可 ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
PUBLIC から USE 特権を取り消します。

BY ALL
特権の付与者にかかわらず、その特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.TBSPACEAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合、USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と G の行が混在している場合は、エラーが戻されます (SQLSTATE 56092)。

注:

- USE 特権が取り消されたからといって、必ずしもその表スペースに表を作成する権限が取り消されるとは限りません。 PUBLIC またはグループが USE 特権を保持している場合、またはユーザーが DBADM などのより上位の権限を持っている場合は、引き続きその表スペースに表を作成することができます。

例:

例 1: ユーザー BOBBY から、表スペース PLANS で表を作成する特権を取り消します。

```
REVOKE USE OF TABLESPACE PLANS FROM USER BOBBY
```

関連資料:

- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 695 ページの『REVOKE (表、ビュー、またはニックネーム特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 682 ページの『REVOKE (ルーチン特権)』

REVOKE (表、ビュー、またはニックネーム特権)

この形式の REVOKE ステートメントは、表、ビュー、またはニックネームに対する特権を取り消します。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

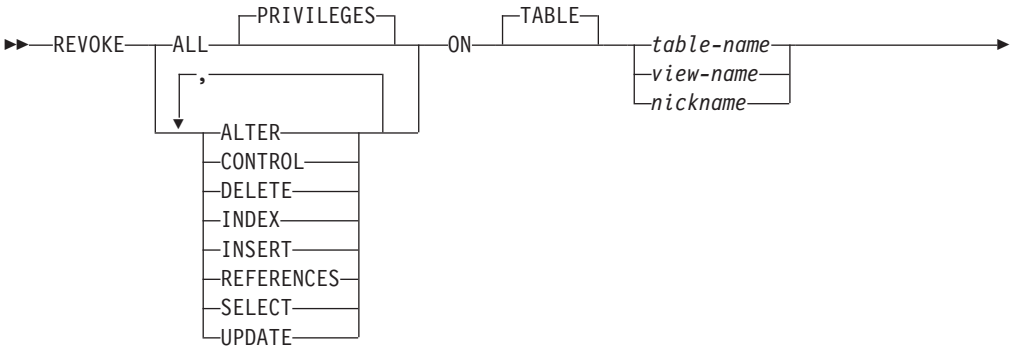
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- SYSADM または DBADM 権限
- 参照されている表、ビュー、またはニックネームに対する CONTROL 特権

CONTROL 特権を取り消すには、SYSADM または DBADM のいずれかの権限が必要です。

カタログの表やビューに対する特権を取り消すには、 SYSADM または DBADM のいずれかの権限が必要です。

構文:



説明:

ALL または **ALL PRIVILEGES**

指定された表、ビュー、またはニックネームに対して *authorization-name* に与えられている特権をすべて (CONTROL を除く) 取り消します。

ALL を指定しない場合は、以下に示すキーワードのうち 1 つまたは複数を指定する必要があります。 各キーワードは、それぞれ説明されている特権を取り消

しますが、その取り消しは ON 文節に指定する表、ビュー、またはニックネームに当てはまる場合にのみ行われます。同じキーワードを複数回指定することはできません。

ALTER
基本表の定義への列の追加、表の主キーまたはユニーク制約の作成またはドロップ、表の外部キーの作成またはドロップ、表、ビュー、またはニックネームに対するコメントの追加や変更、チェック制約の作成またはドロップ、トリガーの作成、ニックネームに対する列オプションの追加、リセット、ドロップ、またはニックネームの列名やデータ・タイプの変更を行うための特権を取り消します。

CONTROL
表、ビュー、またはニックネームをドロップする権限、および表または索引に対して RUNSTATS ユーティリティーを実行する権限を取り消します。

authorization-name から CONTROL 特権を取り消しても、そのオブジェクトに対してそのユーザーに付与されているその他の特権は取り消されません。

DELETE
表、更新可能なビュー、またはニックネームから行を削除する特権を取り消します。

INDEX
表の索引、またはニックネームの索引指定を作成する特権を取り消します。 索引または索引指定の作成者には、その索引または索引指定に対する CONTROL 特権が自動的に与えられます (これにより、作成者は索引または索引指定をドロップできます)。 さらに、INDEX 特権が取り消されても、作成者は CONTROL 特権をそのまま保持します。

INSERT
表、更新可能なビュー、またニックネームに行を挿入したり、 IMPORT ユーティリティーを実行したりする特権を取り消します。

REFERENCES
親として表を参照する外部キーの作成、またはドロップを行う特権を取り消します。列レベルの REFERENCES 特権もすべて取り消されます。

SELECT
表またはビューからの行の検索、表に対するビューの作成、および表またはビューに対して EXPORT ユーティリティーを実行する特権を取り消します。

SELECT 特権を取り消すと、ビューによっては作動不能になるものがあります。 (作動不能なビューについては、『CREATE VIEW』 を参照してください。)

UPDATE
表、更新可能なビュー、またはニックネームの行を更新する特権を取り消します。列レベルの UPDATE 特権もすべて取り消されます。

ON TABLE *table-name* または *view-name* または *nickname*
特権を取り消す表、ビュー、またはニックネームを指定します。 *table-name* を宣言済み一時表にすることはできません (SQLSTATE 42995)。

FROM
特権を誰から取り消すかを指定します。

USER
authorization-name がユーザーであることを指定します。

GROUP
authorization-name がグループ名であることを指定します。

authorization-name,...
1 つまたは複数の許可 ID をリストします。

REVOKE ステートメント自体の ID は使用できません (SQLSTATE 42502)。 REVOKE ステートメントの許可 ID と同じである *authorization-name* から特権を取り消すことはできません。

PUBLIC
PUBLIC から特権を取り消します。

BY ALL
指定された個々の特権を、その付与者にかかわらず、それらの特権を明示的に付与されたユーザーのうち指定された人から取り消します。これはデフォルトの動作です。

規則:

- USER も GROUP も指定しない場合には、
  - SYSCAT.TABAUTH および SYSCAT.COLAUTH カタログ・ビューの権限保持者のすべての行の GRANTEETYPE が U の場合には、USER であると見なされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であると見なされます。
  - U の行と、G の行が混在する場合には、エラーになります (SQLSTATE 56092)。

注:

- ビューの作成に使用された *authorization-name* (これは、 SYSCAT.VIEWS におけるビューの DEFINER と呼ばれる) から特権が取り消されると、従属するビューの特権も取り消されます。
- ビューの DEFINER が、そのビュー定義が従属しているオブジェクトに対する SELECT 特権を失った場合、またはそのビュー定義が従属するオブジェクトがドロップされるか、または別のビューのために作動不能になった場合、そのビューは作動不能になります。

  ただし、DBADM または SYSADM が明示的に DEFINER からビューの特権すべてを取り消した場合、 SYSCAT.TABAUTH にはその DEFINER についてのレコードが表示されませんが、ビューには何も影響がなく作動可能のままになります。
- 作動不能なビューに対する特権は取り消すことはできません。
- 特権を取り消すオブジェクトに従属するすべてのパッケージは、無効になります。そのようなパッケージは、そのアプリケーションでバインド操作または再バインド操作が正常に実行されるか、またはそのアプリケーションが実行され、そのアプリケーションを (カタログに保管されている情報を使用して) データベー

ス・マネージャーが正常に再バインドするまで、無効のままです。取り消しによって無効としてマークされたパッケージは、追加の付与操作なしで正常に再バインドできます。

たとえば、USER1 が所有するパッケージに表 T1 からの SELECT が含まれ、その表 T1 に対する SELECT 特権が USER1 から取り消された場合、パッケージは無効としてマークされます。 SELECT 権限が再び付与された場合、またはそのユーザーに DBADM 権限が与えられている場合には、パッケージは実行時に正常に再バインドされます。

- パッケージ、トリガー、またはビューの FROM 文節で OUTER(*Z*) が使用されている場合、 *Z* のすべての副表またはサブビューで SELECT 特権に対する従属関係が存在します。同じように、パッケージ、トリガー、またはビューで DEREF(*Y*) が使用されていて、 *Y* が *Z* という表またはビューをターゲットとする参照タイプである場合、 *Z* のすべての副表またはサブビューで SELECT 特権に対する従属関係が存在します。こうした SELECT 特権の 1 つが取り消されると、そのパッケージは無効になり、そのトリガーまたはビューは作動不能になります。
- CONTROL 特権も取り消すのでない限り、そのオブジェクトに対する CONTROL が与えられている *authorization-name* から表、ビューまたはニックネームの特権を取り消すことはできません (SQLSTATE 42504)。
- 特定の特権の取り消しにより、アクションを実行する権限が取り消されるとは限りません。 PUBLIC またはグループが他の特権を持っている場合、またはユーザーが表またはビューのスキーマに対する ALTERIN などの特権を持っている場合は、ユーザーは作業を続行できます。
- マテリアライズ照会表の DEFINER が、マテリアライズ照会表定義が従属している表に対する SELECT 特権を失った場合、 (またはマテリアライズ照会表定義が従属する表がドロップされる場合)、マテリアライズ照会表は作動不能になります。

  ただし、DBADM または SYSADM が明示的に DEFINER から表の特権すべてを取り消した場合には、 SYSTABAUTH のその DEFINER についてのレコードは削除されますが、マテリアライズ照会表には何も影響がなく作動可能のままになります。
- ニックネーム特権を取り消しても、データ・ソース・オブジェクト (表またはビュー) の特権に影響を与えることはありません。
- オブジェクトが従属しているためにドロップできない SQL 関数またはメソッド本体がある場合は、その SQL 関数またはメソッド本体で直接または間接的に参照される表やビューに対する SELECT 特権も取り消せない場合があります (SQLSTATE 42893)。
- SQL 関数またはメソッド本体の DEFINER が、その関数またはメソッド本体の定義が従属するオブジェクトに対する SELECT 特権を失う (または、その関数またはメソッド本体の定義が従属するオブジェクトがドロップされる) と、別のオブジェクトがその関数またはメソッドが従属していなければ、その関数またはメソッド本体はドロップされます (SQLSTATE 42893)。

例:

例 1: ユーザー ENGLES から、表 EMPLOYEE に対する SELECT 特権を取り消します。 SYSCAT.TABAUTH カタログ・ビューにはこの表とユーザーについての行が 1 行あり、その GRANTEETYPE の値は U です。

```
REVOKE SELECT
  ON TABLE EMPLOYEE
  FROM ENGLES
```

例 2: 以前にすべてのローカル・ユーザーに与えられた表 EMPLOYEE に対する更新特権を取り消します。特定のユーザーに対する特権付与には影響を与えない点に注意してください。

```
REVOKE UPDATE
  ON EMPLOYEE
  FROM PUBLIC
```

例 3: ユーザー PELLOW と MLI、およびグループ PLANNERS から、表 EMPLOYEE に対する特権をすべて取り消します。

```
REVOKE ALL
  ON EMPLOYEE
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

例 4: JOHN という名前のユーザーから、表 CORPDATA.EMPLOYEE に対する SELECT 特権を取り消します。 SYSCAT.TABAUTH カタログ・ビューにはこの表とユーザーについての行が 1 行あり、その GRANTEETYPE の値は U です。

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

または

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM USER JOHN
```

GROUP JOHN には特権が与えられていないので、 GROUP JOHN から特権を取り消そうとしてもエラーになります。

例 5: JOHN という名前のグループから、表 CORPDATA.EMPLOYEE に対する SELECT 特権を取り消します。 SYSCAT.TABAUTH カタログ・ビューにはこの表とユーザーについての行が 1 行あり、その GRANTEETYPE の値は G です。

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

または

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM GROUP JOHN
```

例 6: ユーザー SHAWN から、ニックネーム ORAREM1 の索引指定を作成する特権を取り消します。

```
REVOKE INDEX
  ON ORAREM1 FROM USER SHAWN
```

関連資料:

- 347 ページの『CREATE TABLE』
- 474 ページの『CREATE VIEW』

- 520 ページの『DROP』
- 673 ページの『REVOKE (データベース権限)』
- 677 ページの『REVOKE (索引特権)』
- 679 ページの『REVOKE (パッケージ特権)』
- 686 ページの『REVOKE (スキーマ特権)』
- 691 ページの『REVOKE (サーバー特権)』
- 693 ページの『REVOKE (表スペース特権)』
- 682 ページの『REVOKE (ルーチン特権)』

関連サンプル:

- 『tbpriv.sqc -- How to grant, display, and revoke privileges (C)』
- 『tbpriv.sqC -- How to grant, display, and revoke privileges (C++)』
- 『TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)』
- 『TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)』

ROLLBACK

ROLLBACK ステートメントは、作業単位またはセーブポイントにおいてデータベースに加えられた変更を撤回するために使用します。

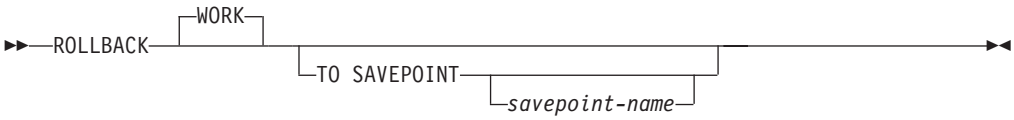
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

必要ありません。

構文:



説明:

ROLLBACK ステートメントが実行される作業単位は終了し、新しい作業単位が開始されます。その作業単位の過程でデータベースに対して行われた変更はすべて取り消されます。

ただし、以下のステートメントはトランザクションによって制御されず、これらのステートメントによって行われた変更は ROLLBACK ステートメントとは独立しています。

- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET PASSTHRU

  注: SET PASSTHRU ステートメントはトランザクションによって制御されませんが、ステートメントによって開始されたパススルー・セッションはトランザクションにより制御されます。
- SET PATH
- SET SCHEMA

- SET SERVER OPTION

シーケンスと ID 値の生成は、トランザクションの制御下にはありません。ROLLBACK ステートメントを発行しても、*nextval-expression* によって生成されて使用される値や、 ID 列のある表に行を挿入することによって生成されて使用される値には影響を与えません。また、ROLLBACK ステートメントを発行しても、*prevval-expression* によって戻される値と IDENTITY_VAL_LOCAL 関数のどちらにも影響を与えません。

TO SAVEPOINT

部分的なロールバック (ROLLBACK TO SAVEPOINT) を実行することを指定します。現行のセーブポイント・レベルのアクティブなセーブポイントがない場合 (SAVEPOINT ステートメントの『規則』 の節を参照) は、エラーが戻されます (SQLSTATE 3B502)。セーブポイントは、ロールバックが正常に完了した後もそのまま存続しますが、ネストされたセーブポイントはすべて解放され、存在しなくなります。ネストされたセーブポイントがある場合、それらはロールバックされたものとみなされ、その後、現行セーブポイントへのロールバックへの一部として解放されます。 *savepoint-name* が指定されない場合は、現行セーブポイント・レベルでの最新セットのセーブポイントへのロールバックが行われます。

この文節を省略して ROLLBACK ステートメントを実行すると、トランザクション全体がロールバックされます。また、トランザクション内のセーブポイントは解放されます。

savepoint-name

ロールバック操作に使用されるセーブポイントを指定します。 *savepoint-name* を指定する際に、'SYS' で始めることはできません (SQLSTATE 42939)。 ロールバックが正常に完了した後も、その名前のセーブポイントはそのまま存続します。指定された名前のセーブポイントが存在しない場合は、エラーが戻されます (SQLSTATE 3B001)。セーブポイントが設定された後に加えられたデータおよびスキーマの変更が取り消されます。

注:

- ROLLBACK が実行された作業単位では、保持されていたロックがすべて解放されます。オープン・カーソルはすべてクローズされます。 LOB ロケーターはすべて解放されます。
- ROLLBACK ステートメントの実行により、特殊レジスターの値を変更する SET ステートメントまたは RELEASE ステートメントは影響を受けません。
- プログラムが異常終了した場合は、暗黙的にその作業単位がロールバックされます。
- ステートメントのキャッシュは、ロールバック操作の影響を受けます。
- ROLLBACK TO SAVEPOINT がカーソルに与える影響は、セーブポイントに含まれているステートメントによって異なります。
  - セーブポイントに DDL が含まれており、この DDL にカーソルが従属している場合、カーソルは無効としてマークされます。これらのカーソルを使おうとすると、エラーが戻されます (SQLSTATE 57007)。
  - それ以外の場合は、次のとおりです。

  - セーブポイントで参照されているカーソルは、オープンされたままになり、結果表の次の論理行の前に置かれます。 (位置による UPDATE ステートメントまたは DELETE ステートメントが出される前に、 FETCH を実行する必要があります。)
  - セーブポイントで参照されていないカーソルは、 ROLLBACK TO SAVEPOINT の影響を受けません (元の位置でオープンされたままになります)。
- 動的に準備されたステートメントの名前は依然として有効ですが、セーブポイント内でロールバックされた DDL 操作の結果として、ステートメントが暗黙的に再び準備されることがあります。
- ROLLBACK TO SAVEPOINT 操作が行われると、セーブポイントの中で指定されていた宣言済み一時表はすべてドロップされます。宣言済み一時表をセーブポイントの中で変更していた場合は、すべての行が表から削除されます。
- すべてのロックは、 ROLLBACK TO SAVEPOINT ステートメントの後にも保持されます。
- すべての LOB ロケーターは、 ROLLBACK TO SAVEPOINT 操作の後にも保持されます。

例:

最後のコミット・ポイントまたはロールバック以後に行われた変更を削除します。

```
ROLLBACK WORK
```

関連資料:

- 551 ページの『EXECUTE』
- 704 ページの『SAVEPOINT』

関連サンプル:

- 『delet.sqb -- How to delete table data (MF COBOL)』
- 『spclient.sqc -- Call various stored procedures (C)』
- 『tut_use.sqc -- How to modify a database (C)』
- 『spclient.sqC -- Call various stored procedures (C++)』
- 『tut_use.sqC -- How to modify a database (C++)』

SAVEPOINT

SAVEPOINT ステートメントを使用して、トランザクション内にセーブポイントを設定します。

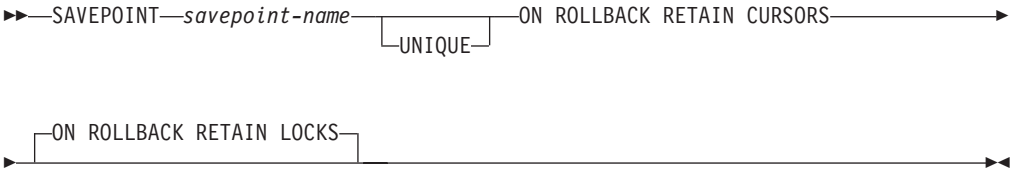
呼び出し:

このステートメントは、アプリケーション・プログラム (ストアード・プロシージャーを含む) に組み込むこともでき、対話式に発行することもできます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

必要ありません。

構文:



説明:

savepoint-name
セーブポイントの名前を指定します。 *savepoint-name* を指定する際に、'SYS' で始めることはできません (SQLSTATE 42939)。 同じ名前のセーブポイントがこのセーブポイント・レベル内で UNIQUE としてすでに定義されている場合、エラーが戻されます (SQLSTATE 3B501)。

UNIQUE
セーブポイントが現行セーブポイント・レベル内でアクティブな間、このセーブポイントの名前がアプリケーションによって再使用されないことを指定します。*savepoint-name* がこのセーブポイント・レベル内にすでに存在していると、エラーが戻されます (SQLSTATE 3B501)。

ON ROLLBACK RETAIN CURSORS
SAVEPOINT ステートメントの後に処理されるオープン・カーソルのステートメントに関して、このセーブポイントへのロールバックでのシステムの動作を指定します。この文節は可能な限り、セーブポイントへのロールバックによる影響を受けないことを示します。どのような場合にカーソルがセーブポイントへのロールバックから影響を受けるかについては、 『ROLLBACK』を参照してください。

ON ROLLBACK RETAIN LOCKS
セーブポイントの設定後にかけられるロックに関して、このセーブポイントへのロールバックでのシステムの動作を指定します。このセーブポイント以降に獲得したロックは追跡されず、このセーブポイントへのロールバック時にはロールバック (解放) されません。

規則:

- セーブポイント関連のステートメントをトリガー定義内で使用することはできません (SQLSTATE 42987)。
- 以下のいずれかの状態になると、新規のセーブポイント・レベルが開始します。
  - 新規の作業単位 (UOW) が開始する。
  - NEW SAVEPOINT LEVEL 文節で定義されたプロシージャーが呼び出される。
  - アトミック・コンパウンド SQL ステートメントが開始する。
- セーブポイント・レベルの作成の原因となったイベントが終了されるか削除されると、セーブポイント・レベルは終了します。セーブポイント・レベルが終了すると、その中に含まれるすべてのセーブポイントは解放されます。オープン・カーソル、DDL アクション、またはデータ変更すべてはその親セーブポイント・レベル (すなわち、今終了したセーブポイント・レベルがその内部で作成されたセーブポイント・レベル) によって継承され、親セーブポイント・レベルに対して出されたセーブポイント関連のステートメントが適用されます。
- セーブポイント・レベル内のアクションには、以下の規則が適用されます。
  - セーブポイントは、それが設定されているセーブポイント・レベル内でのみ参照可能です。現行のセーブポイント・レベルの外で設定されたセーブポイントを解放、破棄、またはロールバックすることはできません。
  - 現行のセーブポイント・レベル内で設定されているすべてのアクティブなセーブポイントは、セーブポイント・レベルが終了すると自動的に解放されます。
  - 現行のセーブポイント・レベル内でのみ、セーブポイント名の固有性が強制されます。他のセーブポイント・レベルでアクティブであるセーブポイントの名前に影響がなければ、そのセーブポイントの名前を現行のセーブポイント・レベルで再利用できます。

注:

- SAVEPOINT ステートメントを発行し終えると、ニックネームに対する挿入、更新、または削除操作は行えなくなります。
- UNIQUE 文節を省略した場合、別のセーブポイントが *savepoint-name* を同じ保管レベル内で再使用してもよいと指定したことになります。同じ名前のセーブポイントが保管レベル内にすでに存在しているときには、既存のセーブポイントが破棄され、新規のセーブポイントがその名前で現在処理中のポイントに作成されます。この新規のセーブポイントが、アプリケーションによって最後に設定されたセーブポイントであるとみなされます。同じ名前の別のセーブポイントを再利用したことによって既存のセーブポイントが破棄されたとしても、それはそのセーブポイントだけが破棄されたのであり、破棄されたセーブポイント以降に設定されたセーブポイントが解放されることはないので注意してください。これら後で設定されたセーブポイントは RELEASE SAVEPOINT ステートメントでのみ解放できます。このステートメントは、指名されたセーブポイントと、そのセーブポイント以降に設定されたすべてのセーブポイントを解放します。
- UNIQUE 文節を指定した場合、 *savepoint-name* は同じ名前の既存のセーブポイントを解放した後でのみ再利用できます。
- あるセーブポイントにおいて、処理の途中でユーティリティー、SQL ステートメント、または DB2 コマンドが断続的にコミットを実行した場合、そのセーブポイントは暗黙的に解放されます。

- あるセーブポイントで SET INTEGRITY ステートメントがロールバックされた場合、動的に準備されたステートメントの名前は依然として有効ですが、そのステートメントが暗黙的に再び準備されることがあります。
- 挿入がバッファーに入れられることになっている場合 (すなわち、アプリケーションが INSERT BUF オプションを指定してプリコンパイルされた場合)、バッファーは、SAVEPOINT、ROLLBACK、または RELEASE TO SAVEPOINT ステートメントが出されるとフラッシュされます。

例:

例 1: ネストされたセーブポイントに対してロールバック操作を実行します。まず、DEPARTMENT という名前の表を作成します。 SAVEPOINT1 を開始する前に 1 行挿入し、 SAVEPOINT2 を開始する前にもう 1 行挿入し、 SAVEPOINT3 を開始する前にさらにもう 1 行挿入します。

```
CREATE TABLE DEPARTMENT (
  DEPTNO   CHAR(6),
  DEPTNAME VARCHAR(20),
  MGRNO    INTEGER)

  INSERT INTO DEPARTMENT VALUES ('A20', 'MARKETING', 301)

  SAVEPOINT SAVEPOINT1 ON ROLLBACK RETAIN CURSORS

  INSERT INTO DEPARTMENT VALUES ('B30', 'FINANCE', 520)

  SAVEPOINT SAVEPOINT2 ON ROLLBACK RETAIN CURSORS

  INSERT INTO DEPARTMENT VALUES ('C40', 'IT SUPPORT', 430)

  SAVEPOINT SAVEPOINT3 ON ROLLBACK RETAIN CURSORS

  INSERT INTO DEPARTMENT VALUES ('R50', 'RESEARCH', 150)
```

この時点で、DEPARTMENT 表には A20、B30、C40、および R50 という行があります。そして、次のステートメントを出すと、

```
ROLLBACK TO SAVEPOINT SAVEPOINT3
```

行 R50 が DEPARTMENT 表からなくなります。そして、次のステートメントを出すと、

```
ROLLBACK TO SAVEPOINT SAVEPOINT1
```

DEPARTMENT 表は残っていますが、 SAVEPOINT1 を設定した後に挿入した行 (B30 と C40) が表からなくなります。

関連資料:

- 701 ページの『ROLLBACK』

SELECT

SELECT ステートメントは、照会の 1 つの形式です。 これは、アプリケーション・プログラムに組み込むことも、または対話式に発行することも可能です。

関連資料:

- *SQL リファレンス 第 1 巻* の『副選択』
- *SQL リファレンス 第 1 巻* の『Select-statement』

関連サンプル:

- 『dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)』
- 『static.sqb -- Get table data using static SQL statement (MF COBOL)』
- 『tbread.c -- How to read data from tables』
- 『tut_read.c -- How to read data from tables』
- 『tbread.sqc -- How to read tables (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『tbread.sqC -- How to read tables (C++)』
- 『tut_read.sqC -- How to read tables (C++)』
- 『TbRead.java -- How to read table data (JDBC)』
- 『TutRead.java -- Read data in a table (JDBC)』
- 『TbRead.sqlj -- How to read table data (SQLj)』
- 『TutRead.sqlj -- Read data in a table (SQLj)』

SELECT INTO

SELECT INTO ステートメントは、0 行あるいは 1 行から成る結果表を作成し、その行の値をホスト変数に割り当てます。その表が空の場合、ステートメントは、SQLCODE に +100、SQLSTATE に '02000' を割り当て、ホスト変数には値を割り当てません。 複数の行が検索条件を満たしている場合、ステートメントの処理は終了し、エラーが発生します (SQLSTATE 21000)。

呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

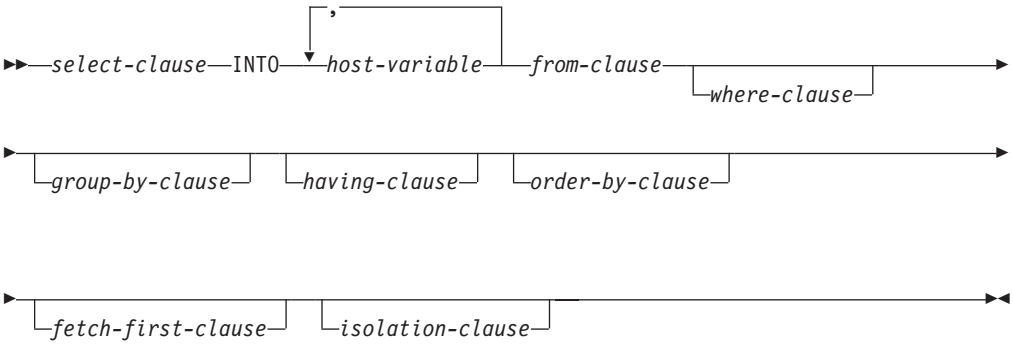
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 表、ビュー、またはニックネームに対する SELECT 特権
- 表、ビュー、またはニックネームに対する CONTROL 特権
- SYSADM または DBADM 権限

静的 SELECT INTO ステートメントの場合、GROUP 特権は検査されません。

SELECT INTO ステートメントの対象がニックネームの場合は、データ・ソースでステートメントが実行されないうちは、そのデータ・ソース上のオブジェクトに対する特権は考慮されません。この時点で、データ・ソースに接続するために使用される許可 ID は、データ・ソースのオブジェクトに対して操作を行うのに必要な特権を持っている必要があります。ステートメントの許可 ID は、データ・ソースの別の許可 ID へマップできます。

構文:



説明:

select-clause、 *from-clause*、 *where-clause*、 *group-by-clause*、 *having-clause*、 *order-by-clause*、 *fetch-first-clause*、および *isolation-clause* についての説明は、 *SQL リファレンス 第 2 巻* の 『照会』 の項を参照してください。

INTO
この後にホスト変数のリストを指定します。

host-variable
ホスト変数の宣言規則に従ってプログラムに記述されている変数を指定します。

結果行の最初の値はリストの最初の変数、 2 番目の値は 2 番目の変数に割り当てられます。以下同様です。ホスト変数の数が列の値の数より少ない場合は、 SQLCA の SQLWARN3 フィールドに値 'W' が割り当てられます。

変数への個々の割り当ては、リストに指定された順序で行われます。エラーが発生すると、値はホスト変数に割り当てられません。

例:

例 1: この C の例では、EMP 表における給与の最高額をホスト変数 MAXSALARY に割り当てています。

```
EXEC SQL SELECT MAX(SALARY)
  INTO :MAXSALARY
  FROM EMP;
```

例 2: この C の例では、EMP 表にある従業員 528671 の行をホスト変数に割り当てています。

```
EXEC SQL SELECT * INTO :h1, :h2, :h3, :h4
  FROM EMP
  WHERE EMPNO = '528671';
```

例 3: この SQLJ の例では、EMP 表にある従業員 528671 の行をホスト変数に割り当てています。その後、その行は検索更新を使用して更新されますが、照会の実行時にはロックされることになります。

```
#sql { SELECT * INTO :FIRSTNAME, :LASTNAME, :EMPNO, :SALARY
  FROM EMP
  WHERE EMPNO = '528671'
  WITH RS USE AND KEEP EXCLUSIVE LOCKS };
```

関連資料:

- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』
- *SQL リファレンス 第 1 巻* の『SQL 照会』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

関連サンプル:

- 『dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『spclient.sqc -- Call various stored procedures (C)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『tbreorg.sqc -- How to reorganize a table and update its statistics (C)』
- 『dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『spclient.sqC -- Call various stored procedures (C++)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』

- 『tbreorg.sqC -- How to reorganize a table and update its statistics (C++)』

SET CONNECTION

SET CONNECTION ステートメントは、接続の状態を休止状態から現行状態に変更して、指定された位置を現行サーバーにします。このステートメントは、トランザクションの制御下にはありません。

呼び出し:

対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。

構文:

説明:

server-name または *host-variable*

server-name (サーバー名) またはその *server-name* を含む *host-variable* (ホスト変数) によって、アプリケーション・サーバーを指定します。

host-variable (ホスト変数) を指定する場合、それは、長さ属性が 8 以下の文字ストリング変数でなければならず、標識変数を含めることはできません。その *host-variable* に入る *server-name* は、左寄せする必要があり、引用符で区切ることはできません。

server-name は、アプリケーション・サーバーを指定するデータベース別名である点に注意してください。 この名前は、アプリケーション・リクエスターのローカル・ディレクトリーにリストされている必要があります。

server-name または *host-variable* は、アプリケーション・プロセスの既存の接続を指定していなければなりません。 既存の接続を指定していない場合には、エラー (SQLSTATE 08003) になります。

現行接続に対する SET CONNECTION の場合、アプリケーション・プロセスのすべての接続の状態は変更されません。

正常に接続された場合

SET CONNECTION ステートメントが正常に実行された場合、

- 作成される接続はありません。 CURRENT SERVER 特殊レジスターは、指定した *server-name* で更新されます。
- それ以前の現行接続がある場合、それは休止状態になります (別の *server-name* を指定した場合)。
- CURRENT SERVER 特殊レジスターと SQLCA は、 『CONNECT (タイプ 1)』 で説明した方法と同じ方法で更新されます。

接続が正常に実行されなかった場合

SET CONNECTION ステートメントが失敗した場合、

- エラーの理由に関係なく、アプリケーション・プロセスの接続状態とその接続の状態は変更されません。
- エラーになったタイプ 1 の CONNECT の場合と同様に、 SQLCA の SQLERRP フィールドは、エラーを検出したモジュール名に設定されます。

注:

- タイプ 1 CONNECT ステートメントの使用は、 SET CONNECTION の使用を排除するわけではありませんが、休止状態の接続は存在し得ないので、 SET CONNECTION ステートメントに現行接続を指定するのでない限り、このステートメントは常にエラーになります (SQLSTATE 08003)。
- SQLRULES(DB2) 接続オプション (『分散作業単位のセマンティクスを制御するオプション』 を参照) を使用した場合、 SET CONNECTION の使用を排除するわけではありませんが、タイプ 2 CONNECT ステートメントが使用できるので、このステートメントは不要です。
- 同じ作業単位で接続が使用され、休止状態になり、次に現行状態にリストアすると、ロック、カーソル、および準備済みステートメントの状況に関して、その接続はアプリケーション・プロセスでの最後の使用を反映したものになります。

例:

IBMSTHDB で SQL ステートメントを実行し、次に IBMTOKDB で SQL ステートメントを実行し、その後、IBMSTHDB で SQL ステートメントを実行します。

```
EXEC SQL CONNECT TO IBMSTHDB;
/* Execute statements referencing objects at IBMSTHDB */

EXEC SQL CONNECT TO IBMTOKDB;
/* Execute statements referencing objects at IBMTOKDB */

EXEC SQL SET CONNECTION IBMSTHDB;
/* Execute statements referencing objects at IBMSTHDB */
```

最初の CONNECT ステートメントでは IBMSTHDB の接続が作成され、 2 番目の CONNECT ステートメントでその接続は休止状態になり、 SET CONNECTION ステートメントによってその接続は現行状態に戻ります。

関連概念:

- *SQL リファレンス 第 1 巻* の『分散リレーショナル・データベース』

関連資料:

- 149 ページの『CONNECT (タイプ 1)』

関連サンプル:

- 『dbmcon.sqc -- How to use multiple databases (C)』
- 『dbmcon.sqC -- How to use multiple databases (C++)』

SET CURRENT DEFAULT TRANSFORM GROUP

SET CURRENT DEFAULT TRANSFORM GROUP ステートメントは、 CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にありません。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:

説明:

group-name

トランスフォーム・グループを識別する名前を 1 つの部分からなる名前で指定します。このグループ名はすべての構造タイプに定義されます。ここで指定された名前は、このステートメントに続く他のステートメントでも (つまり、別の SET CURRENT DEFAULT TRANSFORM GROUP ステートメントによって特殊レジスターの値が再び変更されるまで) 参照することができます。

名前は、長さが 18 文字以下の SQL ID でなければなりません (SQLSTATE 42815)。特殊レジスターが設定される際に、構造タイプに定義されている *group-name* の妥当性が検査されることはありません。特定の構造タイプを指定して参照するときにのみ、指定されたトランスフォーム・グループの定義が妥当であるかどうかが検査されます。

規則:

- 指定された値が *group-name* の規則に準拠していない場合は、エラーが発生します (SQLSTATE 42815)。
- トランスフォーム・グループ *group-name* に定義されている TO SQL 関数と FROM SQL 関数は、ユーザー定義構造タイプのデータをホスト・プログラムとの間で交換するために使用されます。

注:

- CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターの初期値は空ストリングです。

例:

例 1: デフォルトのトランスフォーム・グループを MYSTRUCT1 に設定します。トランスフォーム・グループ MYSTRUCT1 に定義されている TO SQL 関数と FROM SQL 関数は、ユーザー定義構造タイプの変数を現在のホスト・プログラムとの間で交換するために使用されます。

```
SET CURRENT DEFAULT TRANSFORM GROUP = MYSTRUCT1
```

関連資料:

- *SQL リファレンス 第 1 巻* の『CURRENT DEFAULT TRANSFORM GROUP 特殊レジスター』

SET CURRENT DEGREE

SET CURRENT DEGREE ステートメントは、 CURRENT DEGREE 特殊レジスターに値を割り当てます。このステートメントは、トランザクションの制御下にありません。

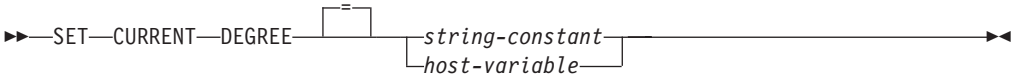
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

CURRENT DEGREE の値は、ストリング定数またはホスト変数の値によって置き換えられます。 値は 5 文字を超えない文字ストリングでなければなりません。その値は、1 ～ 32 767 (両端を含む) の整数の文字ストリング表現、または 'ANY' でなければなりません。

SQL ステートメントが動的に準備される時点で、整数として表現される CURRENT DEGREE の値が 1 である場合には、そのステートメントの実行にパーティション内並列処理は使用されません。

SQL ステートメントが動的に準備される時点で、CURRENT DEGREE の値が 1 以外の数値である場合には、そのステートメントの実行には、指定した度合いのパーティション内並列処理を使用できます。

SQL ステートメントが動的に準備される時点で、 CURRENT DEGREE の値が 'ANY' である場合、そのステートメントの実行には、データベース・マネージャーによって決定された度合いを用いたパーティション内並列処理を使用できます。

host-variable
host-variable (ホスト変数) は、そのデータ・タイプが CHAR または VARCHAR で、5 文字を超えない長さでなければなりません。それより長いフィールドを指定すると、エラーになります (SQLSTATE 42815)。実際に指定する値が、指定した置換値より大きい場合は、入力の右側にブランクを入れる必要があります。先行ブランクは使用できません (SQLSTATE 42815)。すべての入力値は、大文字小文字を区別しないものとして処理されます。 *host-variable* が標識変数を伴う場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

string-constant
string-constant (ストリング定数) の長さは 5 を超えてはなりません。

注:

静的 SQL ステートメントのパーティション内並列処理の度合いは、 PREP または BIND コマンドの DEGREE オプションを使用して制御できます。

パーティション内並列処理の実際の実行時の度合いは、以下のものより小さい値になります。
- 最大照会度合 (max_querydegree) 構成パラメーター
- アプリケーション実行時の度合い
- SQL ステートメントのコンパイルの度合い

パーティション内並列処理を使用するには、 intra_parallel データベース・マネージャー構成をオンにする必要があります。 オフに設定されている場合、このレジスターの値は無視され、ステートメントは最適化にパーティション内並列処理を使用しません (SQLSTATE 01623)。

SQL ステートメントによっては、パーティション内並列処理を使用できません。

例:

例 1: 以下のステートメントは、パーティション内並列処理を禁止する CURRENT DEGREE を設定します。

```
SET CURRENT DEGREE = '1'
```

例 2: 以下のステートメントは、パーティション内並列処理を許可する CURRENT DEGREE を設定します。

```
SET CURRENT DEGREE = 'ANY'
```

SET CURRENT EXPLAIN MODE

SET CURRENT EXPLAIN MODE ステートメントは、 CURRENT EXPLAIN MODE 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

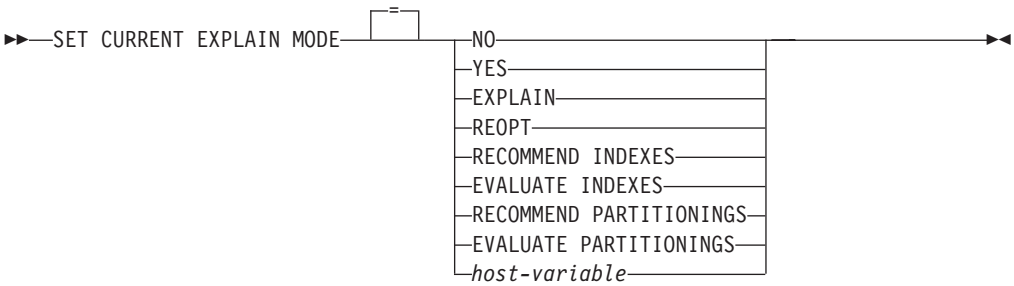
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特別な権限は必要ありません。

構文:



説明:

NO
Explain 機能を使用不可にします。 Explain 情報はキャプチャーされません。 NO は、特殊レジスターの初期値です。

YES
Explain 機能を使用可能にし、適格な動的 SQL ステートメントについての Explain 情報を Explain 表に挿入します。 すべての動的 SQL ステートメントが、通常どおりにコンパイルおよび実行されます。

EXPLAIN
Explain 機能を使用可能にし、準備される適切な動的 SQL ステートメントについての Explain 情報をキャプチャーします。 ただし、動的ステートメントは実行されません。

REOPT
Explain 機能を使用可能にし、実行時のステートメント再最適化の際 (すなわち、ホスト変数、特殊レジスター、またはパラメーター・マーカーの実際の値が使用可能になるとき) に、静的または動的 SQL ステートメントのために Explain 情報がキャプチャーされるようにします。

RECOMMEND INDEXES
SQL コンパイラーが索引を推奨できるようにします。この Explain モードで実行される照会はすべて、推奨された索引を ADVISE_INDEX 表に埋め込みま

す。 さらに、推奨された索引を使用する方法を示すため、 Explain 表に Explain 情報がキャプチャーされますが、そのステートメントのコンパイルや実行は行われません。

EVALUATE INDEXES
SQL コンパイラーが索引を評価できるようにします。評価される索引は ADVISE_INDEX 表から読み取られ、 EVALUATE = Y というマークが付けられる必要があります。 オプティマイザーは、カタログの値に基づく仮想索引を生成します。この Explain モードで実行される照会はすべて、仮想索引に基づいて見積もられた統計を使用してコンパイルされ、最適化されます。 ステートメントは実行されません。

RECOMMEND PARTITIONINGS
特定の照会がアクセスするそれぞれの表ごとに、コンパイラーが最良のパーティションを推奨するように指定します。それから、最良のパーティションは ADVISE_PARTITION 表に書き込まれます。照会は実行されません。

EVALUATE PARTITIONINGS
ADVISE_PARTITION 表に指定された仮想パーティションを使って、コンパイラーが照会の推定パフォーマンスを取得するように指定します。

host-variable
host-variable (ホスト変数) のデータ・タイプは CHAR または VARCHAR でなければならず、その内容の長さは 254 を超えてはなりません。それより長いフィールドを指定すると、エラーになります (SQLSTATE 42815)。指定する値は、NO、 YES、EXPLAIN、RECOMMEND INDEXES、または EVALUATE INDEXES でなければなりません。実際に指定する値が、指定した置換値より大きい場合は、入力の右側にブランクを入れる必要があります。先行ブランクは使用できません (SQLSTATE 42815)。すべての入力値は、大文字小文字を区別しないものとして処理されます。 *host-variable* が標識変数を伴う場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

注:

- 静的 SQL ステートメントの Explain 情報は、 PREP または BIND コマンドの EXPLAIN オプションの使用によってキャプチャーすることができます。 EXPLAIN オプションの ALL の値が指定され、 CURRENT EXPLAIN MODE のレジスター値が NO の場合には、ランタイムに動的 SQL ステートメントの Explain 情報がキャプチャーされます。 CURRENT EXPLAIN MODE レジスターの値が NO 以外の場合、 EXPLAIN BIND オプションの値は無視されます。
- RECOMMEND INDEXES と EVALUATE INDEXES は特殊モードで、それらを設定するために使えるのは SET CURRENT EXPLAIN MODE ステートメントだけです。 これらのモードは PREP または BIND オプションを使って設定することはできません。また、SET CURRENT EXPLAIN SNAPSHOT ステートメントを使用しても動作しません。
- Explain 機能が活動化される場合、現行の許可 ID に Explain 表に対する INSERT 特権が必要です。この特権がない場合には、エラー (SQLSTATE 42501) が発生します。
- ルーチンから SQL ステートメントの Explain 情報を取り出す場合は、MODIFIES SQL DATA の SQL データ・アクセス標識を指定して、ルーチンを定義しなければなりません (SQLSTATE 42985)。

- 特殊レジスターが REOPT に設定され、実行時の再最適化のために SQL ステートメントが修飾されない場合 (すなわち、ステートメントが入力変数を持っていないか、 REOPT BIND オプションが NONE に設定されている場合) は、Explain 情報はキャプチャーされません。 REOPT BIND オプションが ONCE に設定されている場合、Explain 情報は、ステートメントが最初に再最適化されるときの 1 回だけキャプチャーされます。ステートメントがキャッシュに入れられた後は、後続の実行では、それ以上の Explain 情報はこのステートメントに関して獲得されません。
- Explain 機能が使用可能で、REOPT BIND オプションが ONCE に設定されていて、すでにキャッシュに入れられている SQL ステートメントを実行しようとした場合は、入力変数の現行値を使ってステートメントがコンパイルおよび再最適化され、それにしたがって Explain 表にデータが取り込まれます。このステートメントのために新たに生成されるアクセス・プランは、キャッシュに入れられず、実行されません。このキャッシュ・ステートメントを並行して実行する他のアプリケーションは引き続き稼働し、このステートメントを実行するための新しい要求は、すでにキャッシュに入れられたアクセス・プランを採用します。
- 静的または動的 SQL ステートメントが入力変数を持っていて、 REOPT BIND オプションが ONCE または ALWAYS に設定されている場合は、 CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスターの REOPT という値は、 EXPLAIN および EXPLSNAP BIND オプションの値をバインド時にオーバーライドします。

例:

次のステートメントでは、以降の適格な動的 SQL ステートメントの Explain 情報を取り込み、そのステートメントが実行されないように、 CURRENT EXPLAIN MODE 特殊レジスターを設定しています。

```
SET CURRENT EXPLAIN MODE = EXPLAIN
```

関連資料:

- *SQL リファレンス 第 1 巻* の『EXPLAIN レジスター値』

SET CURRENT EXPLAIN SNAPSHOT

SET CURRENT EXPLAIN SNAPSHOT ステートメントは、 CURRENT EXPLAIN SNAPSHOT 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

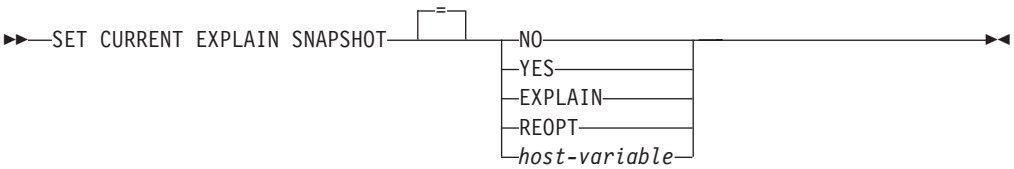
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

NO
Explain スナップショット機能を使用不可にします。 スナップショットは取られません。 NO は、特殊レジスターの初期値です。

YES
Explain スナップショット機能を使用可能にし、適格な動的 SQL ステートメントに対して内部表記のスナップショットを作成します。この情報は、EXPLAIN_STATEMENT 表の SNAPSHOT 列に挿入されます。

EXPLAIN SNAPSHOT 機能は、Visual Explain での使用を意図しています。

EXPLAIN
Explain スナップショット機能を使用可能にし、準備済みの適格な動的 SQL ステートメントごとに内部表記のスナップショットを作成します。 ただし、動的ステートメントは実行されません。

REOPT
Explain 機能を使用可能にし、実行時のステートメント再最適化の際 (すなわち、ホスト変数、特殊レジスター、またはパラメーター・マーカーの実際の値が使用可能になるとき) に、静的または動的 SQL ステートメントのために Explain 情報がキャプチャーされるようにします。

host-variable
host-variable (ホスト変数) のデータ・タイプは CHAR または VARCHAR でなければならず、その内容の長さは 8 を超えてはなりません。それより長いフィールドを指定すると、エラーになります (SQLSTATE 42815)。このレジスターの値は、NO、YES、または EXPLAIN でなければなりません。実際に指定する値が、指定した置換値より大きい場合は、入力の右側にブランクを入れる必要があります。先行ブランクは使用できません (SQLSTATE 42815)。すべての入力

値は、大文字小文字を区別しないものとして処理されます。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

注:

- 静的 SQL ステートメントの Explain スナップショットは、 PREP または BIND コマンドの EXPLSNAP オプションの使用によって取ることができます。 EXPLSNAP オプションの ALL の値を指定し、 CURRENT EXPLAIN SNAPSHOT のレジスター値が NO の場合には、ランタイムに動的 SQL ステートメントの Explain スナップショットが取られます。 CURRENT EXPLAIN SNAPSHOT レジスターの値が NO 以外の場合、 EXPLSNAP オプションは無視されます。
- Explain スナップショット機能が活動化される場合、現行の許可 ID には、Explain 表に対する INSERT 特権が必要です。この特権がないと、エラー (SQLSTATE 42501) になります。
- ルーチンから SQL ステートメントの Explain 情報を取り出す場合は、MODIFIES SQL DATA の SQL データ・アクセス標識を指定して、ルーチンを定義しなければなりません (SQLSTATE 42985)。
- 特殊レジスターが REOPT に設定され、実行時の再最適化のために SQL ステートメントが修飾されない場合 (すなわち、ステートメントが入力変数を持っていないか、 REOPT BIND オプションが NONE に設定されている場合) は、Explain 情報はキャプチャーされません。 REOPT BIND オプションが ONCE に設定されている場合、Explain スナップショット情報は、ステートメントが最初に再最適化されるときの 1 回だけキャプチャーされます。ステートメントがキャッシュに入れられた後は、後続の実行では、それ以上の Explain 情報はこのステートメントに関して獲得されません。
- Explain 機能が使用可能で、REOPT BIND オプションが ONCE に設定されていて、すでにキャッシュに入れられている再最適化可能な SQL ステートメントを実行しようとした場合は、入力変数の現行値を使ってステートメントがコンパイルおよび再最適化され、それにしたがって Explain スナップショットがキャプチャーされます。このステートメントのために新たに生成されるアクセス・プランは、キャッシュに入れられず、実行されません。このキャッシュ・ステートメントを並行して実行する他のアプリケーションは引き続き稼働し、このステートメントを実行するための新しい要求は、すでにキャッシュに入れられたアクセス・プランを採用します。
- 静的または動的 SQL ステートメントが入力変数を持っていて、 REOPT BIND オプションが ONCE または ALWAYS に設定されている場合は、 CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスターの REOPT という値は、 EXPLAIN および EXPLSNAP BIND オプションの値をバインド時にオーバーライドします。

例:

例 1: 以下のステートメントは、 CURRENT EXPLAIN SNAPSHOT 特殊レジスターを設定して、以降の適格な動的 SQL ステートメントの Explain スナップショットを取り、そのステートメントを実行します。

```
SET CURRENT EXPLAIN SNAPSHOT = YES
```

例 2: 以下の例では、 SNAP という名前のホスト変数に CURRENT EXPLAIN SNAPSHOT 特殊レジスターの現行値を入れます。

```
EXEC SQL VALUES (CURRENT EXPLAIN SNAPSHOT) INTO :SNAP;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『EXPLAIN レジスター値』
- *SQL リファレンス 第 1 巻* の『EXPLAIN_STATEMENT 表』

SET CURRENT ISOLATION

SET CURRENT ISOLATION ステートメントは、 CURRENT ISOLATION 特殊レジスターに値を割り当てます。このステートメントは、トランザクションの制御下にありません。

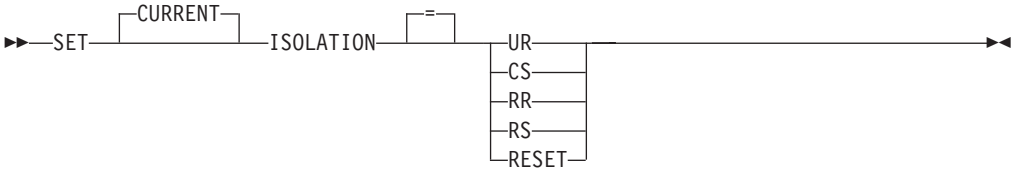
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

CURRENT ISOLATION 特殊レジスターの値は、RESET が指定されている場合、指定した値で置き換えられるかまたはブランクに設定されます。

注意:

- ***互換性***
  - 以下の構文もサポートされています。
    - 等号 (=) の代わりに TO を指定できます。
    - UR の代わりに DIRTY READ を指定できます。
    - UR の代わりに READ UNCOMMITTED を指定できます。
    - READ COMMITTED が認識され、CS に更新されます。
    - CS の代わりに CURSOR STABILITY を指定できます。
    - RR の代わりに REPEATABLE READ を指定できます。
    - RR の代わりに SERIALIZABLE を指定できます。

関連概念:

- *SQL リファレンス 第 1 巻* の『分離レベル』

関連資料:

- *SQL リファレンス 第 1 巻* の『CURRENT ISOLATION 特殊レジスター』

SET CURRENT LOCK TIMEOUT

SET CURRENT LOCK TIMEOUT ステートメントは、 CURRENT LOCK TIMEOUT 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

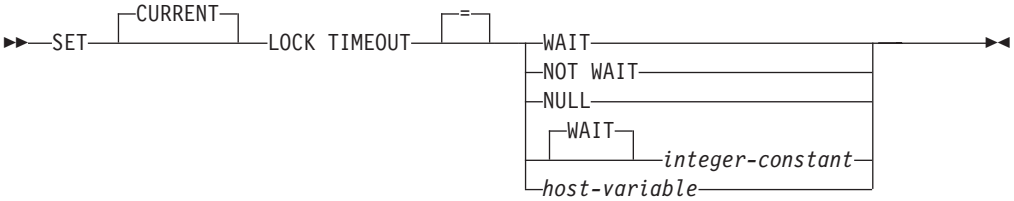
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また対話式に出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

指定値は -1 から 32767 までの整数 (両端を含む) (SQLSTATE 428B7)、または NULL 値でなければなりません。

WAIT
CURRENT LOCK TIMEOUT の値を -1 に設定します。この値は、ロックが解除されるか、デッドロックが検出されるまで、データベース・マネージャーが待機することを意味します (SQLSTATE 40001 または SQLSTATE 57033)。

NOT WAIT
CURRENT LOCK TIMEOUT の値を 0 に設定します。この値は、獲得できないロックをデータベース・マネージャーが待機せず、エラーが戻されることを意味します (SQLSTATE 40001 または SQLSTATE 57033)。

NULL
CURRENT LOCK TIMEOUT の値を設定解除するように指定します。ロックの待機の際には、*locktimeout* データベース構成パラメーターの値が使用されます。特殊レジスターに戻される値は、*locktimeout* の値が変更されると変化します。

WAIT *integer-constant*
-1 から 32767 までの整数を指定します。 -1 の値は、整数値なしで WAIT キーワードを指定することと等価です。 0 の値は、NOT WAIT 文節を指定するのと等価です。値が 1 から 32767 までの場合は、(ロックを獲得できない場合に) エラーが戻される前に、データベース・マネージャーはその秒数だけ待機します (SQLSTATE 40001 または SQLSTATE 57033)。

host-variable
タイプが INTEGER の変数です。値は -1 から 32767 までの範囲内である必要

があります。 *host-variable* が関連した標識変数を伴っていて、その標識変数の値が NULL 値を指定している場合、CURRENT LOCK TIMEOUT の値は設定解除されます。これは NULL キーワードを指定するのと等価です。

注:

- ***互換性***
  - Informix との互換性:
    - TIMEOUT の代わりに MODE を指定できます。
    - 等号 (=) 演算子の代わりに TO を指定できます。
    - SET CURRENT LOCK TIMEOUT WAIT の代わりに SET LOCK WAIT を指定できます。
    - SET CURRENT LOCK TIMEOUT NOT WAIT の代わりに SET LOCK NO WAIT を指定できます。
- 特殊レジスターの更新された値は、このステートメントが正常実行されると即時に有効になります。ステートメントの実行中に使用される特殊レジスター値はステートメント実行の初めに固定されるため、実行を開始したステートメントによって CURRENT LOCK TIMEOUT 特殊レジスターの更新された値が戻されるのは、 SET LOCK TIMEOUT ステートメントが正常に完了した後になります。

例:

例 1: エラーを戻す前に 30 秒間待つよう、ロック・タイムアウト値を設定します。

```
SET CURRENT LOCK TIMEOUT 30
```

例 2: locktimeout データベース構成パラメーター値が代わりに使用されるよう、ロック・タイムアウト値を設定解除します。

```
SET CURRENT LOCK TIMEOUT NULL
```

関連資料:

- *SQL リファレンス 第 1 巻* の『CURRENT LOCK TIMEOUT 特殊レジスター』

SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION

SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION ステートメントは、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

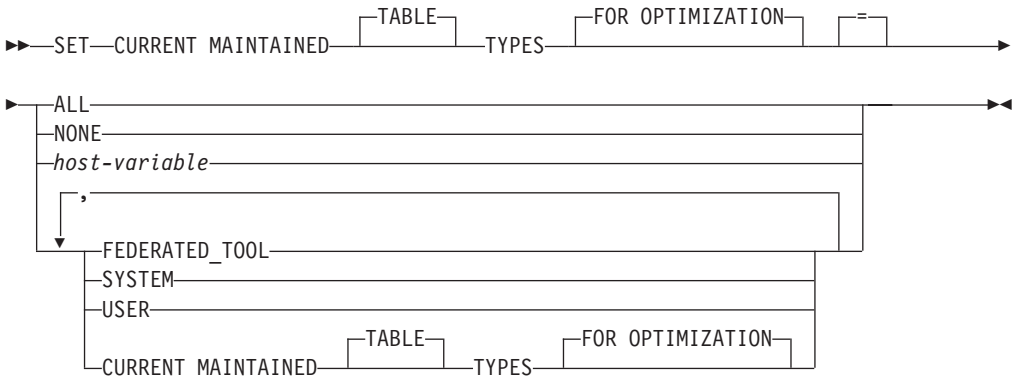
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

ALL
動的 SQL 照会の処理を最適化する際に、この特殊レジスターによって制御されるすべての有効なタイプの保守されている表が、現在および将来に考慮されるよう指定します。

NONE
動的 SQL 照会の処理を最適化する際に、この特殊レジスターによって制御されるオブジェクト・タイプが考慮されないよう指定します。

FEDERATED_TOOL
CURRENT QUERY OPTIMIZATION 特殊レジスターの値が 2 であるかまたは 5 より大きいときに、フェデレーテッド・ツールによって保守されているリフレッシュ据え置きマテリアライズ照会表が動的 SQL 照会の処理を最適化すると見なすことを指定します。

SYSTEM
動的 SQL 照会の処理を最適化する際に、システムによって保守されているリフレッシュ据え置きマテリアライズ照会表が考慮されるよう指定します。 (即時マテリアライズ照会表は常に使用できます。)

USER
動的 SQL 照会の処理を最適化する際に、ユーザーが保守しているリフレッシュ据え置きマテリアライズ照会表が考慮されるよう指定します。

CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
このステートメントを実行する前の CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値。

host-variable
タイプ CHAR または VARCHAR の変数です。ホスト変数の内容の長さは、254 バイトを超えてはなりません (SQLSTATE 42815)。 NULL に設定することはできません。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

host-variable の文字は左寄せされていなければなりません。 *host-variable* の内容は、特殊レジスターのキーワードとして指定できるキーワードをコンマで区切ってリストにしたストリングです。大文字変換は行われないため、これらのキーワードはすべて大文字小文字を区別して指定しなければなりません。値の長さがホスト変数の長さ未満の場合は、値の右側をブランクで埋め込まなければなりません。

注:

- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの初期値は SYSTEM です。
- 動的 SQL 照会の処理を最適化する際に、指定した表タイプが考慮されるようにするには、 CURRENT REFRESH AGE 特殊レジスターをゼロ以外の値に設定しなければなりません。

例:

例 1: CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターを設定します。

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION SYSTEM = USER
```

例 2: CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの現行値を検索して CURMAINTYPES という名前のホスト変数に入れます。

```
EXEC SQL VALUES (CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION)
  INTO :CURMAINTYPES
```

例 3: CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターを値なしに設定します。

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = NONE
```

SET CURRENT PACKAGE PATH

SET CURRENT PACKAGE PATH ステートメントは、 CURRENT PACKAGE PATH 特殊レジスターに値を割り当てます。このステートメントは、トランザクションの制御下にはありません。

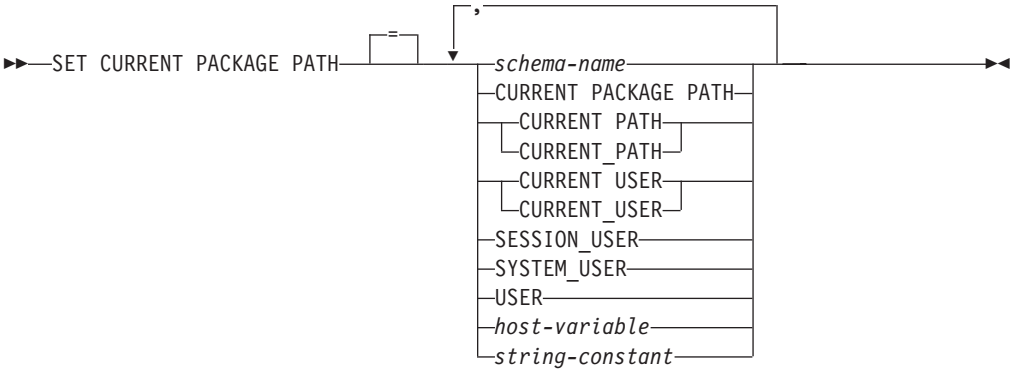
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

schema-name
スキーマを識別します。その名前は、空またはブランクだけの区切り ID であってはなりません (SQLSTATE 42815)。

CURRENT PACKAGE PATH
このステートメントが実行される前の CURRENT PACKAGE PATH 特殊レジスターの値。

CURRENT PATH
CURRENT PATH 特殊レジスターの値。

CURRENT USER
CURRENT USER 特殊レジスターの値。

SESSION_USER
SESSION_USER 特殊レジスターの値。

SYSTEM_USER
SYSTEM_USER 特殊レジスターの値。

USER
USER 特殊レジスターの値。

host-variable
1 つ以上のスキーマ名をコンマで区切って指定します。ホスト変数は次の条件を満たしていなければなりません。

- 文字ストリング変数であること (CHAR または VARCHAR)。ホスト変数の内容の実際の長さは、CURRENT PACKAGE PATH 特殊レジスターの長さを超えてはなりません。
- NULL 値でないこと。標識変数が提供される場合、その値が NULL 値を示していてはなりません。
- 空またはブランクのストリング、または 1 つ以上のスキーマ名をコンマで区切って指定すること。
- ホスト変数の実際の長さが内容よりも大きい場合は、右側をブランクで埋めること。
- CURRENT PACKAGE PATH、CURRENT PATH、CURRENT_PATH、CURRENT
  USER、CURRENT_USER、SESSION_USER、SYSTEM_USER、PATH、または USER を含まないこと。
- 空またはブランクだけを含む区切り ID を指定しないこと。

string-constant
コンマで区切ったゼロ個以上のスキーマ名を含む文字ストリング定数を指定します。ストリング定数は次の条件を満たしていなければなりません。

- CURRENT PACKAGE PATH 特殊レジスターの最大長を超えない長さであること。
- CURRENT PACKAGE PATH、CURRENT PATH、CURRENT_PATH、CURRENT
  USER、CURRENT_USER、SESSION_USER、SYSTEM_USER、PATH、または USER を含まないこと。
- 空またはブランクだけを含む区切り ID を指定しないこと。

規則:

- 複数の同じスキーマがリストに現れた場合は、最初に現れるスキーマが使用されます (SQLSTATE 01625)。
- 指定できるスキーマの数は、CURRENT PACKAGE PATH 特殊レジスターの合計長によって限定されます。特殊レジスター・ストリングは、指定されたそれぞれのスキーマ名を採って末尾ブランクを除去し、名前を二重引用符で囲み、そしてスキーマ名をコンマで区切ることによって構築されます。結果リストの長さが、特殊レジスターの最大長を超えることはできません (SQLSTATE 0E000)。
- スキーマ名が通常 ID の規則に準拠していない場合 (例えば、小文字を含むスキーマ名や、通常 ID に指定できない文字を含むスキーマ名など) には、区切り文字で区切られているスキーマ名として指定する必要があり、ホスト変数内またはストリング定数内に指定することはできません。
- 特殊レジスター (単一キーワードとして指定したもの) の現行値がパッケージ・パス内で使用されることを指示するには、特殊レジスターの名前をキーワードとして指定します。その代わりに、特殊レジスターの名前が区切り ID として指定される場合 (例えば ″USER″) は、その値のスキーマ名 ('USER') として解釈されます。

- SET CURRENT PACKAGE PATH ステートメントに指定された値が変数であるか、スキーマ名であるかを判別するために、次の規則が使用されます。
  - *name* が SQL プロシージャー内のパラメーターまたは SQL 変数と同じ場合は、 *name* はパラメーターまたは SQL 変数として解釈され、 *name* の値がパッケージ・パスに割り当てられます。
  - *name* が SQL プロシージャー内のパラメーターまたは SQL 変数と同じでない場合は、 *name* はスキーマ名として解釈され、 *name* の値がパッケージ・パスに割り当てられます。

注:

- ***トランザクションの考慮事項:*** SET CURRENT PACKAGE PATH ステートメントはコミット可能な操作ではありません。 ROLLBACK は CURRENT PACKAGE PATH 特殊レジスターに影響を及ぼしません。
- ***スキーマの存在検査:*** CURRENT PACKAGE PATH 特殊レジスターがセットされる時点では、指定されたスキーマが存在することの検証は行われません。例えば、つづりを誤ったスキーマが検出されない場合、それは後続の SQL の作動の仕方に影響する可能性があります。パッケージの実行時には、合致するパッケージへの許可が検査され、この許可検査に失敗した場合はエラーが戻されます (SQLSTATE 42501)。
- ***ホスト変数またはストリング定数の内容:*** ホスト変数またはストリング定数の内容は、スキーマ名のリストとして解釈されます。複数のスキーマ名を指定する場合は、それぞれの名前をコンマで区切る必要があります。リスト内の各スキーマ名は、通常 ID を形成するための規則に準拠するか、区切り ID として指定する必要があります。ホスト変数またはストリング定数の内容は大文字変換されません。
- ***COBOL アプリケーション用の組み込み SQL に特有の制限:*** SET CURRENT PACKAGE PATH ステートメントの右辺には、最大で 10 個のリテラル (非ホスト変数) を指定できます。そのような値の最大長は 130 (区切りなし) または 128 (区切りあり) です。

例:

例 1: CURRENT PACKAGE PATH 特殊レジスターを、以下のスキーマのリストに設定します: MYPKGS, 'ABC E', SYSIBM

```
SET CURRENT PACKAGE PATH = MYPKGS, 'ABC E', SYSIBM
```

次のステートメントは、ホスト変数を結果リストの値に設定します。

```
SET :hvpklist = CURRENT PACKAGE PATH
```

ホスト変数の値は: "MYPKGS", "ABC E", "SYSIBM"

例 2: CURRENT PACKAGE PATH 特殊レジスターを、以下のスキーマのリストに設定します: "SCH4","SCH5" (ただし、:hvar1 は 'SCH4,SCH5' を含みます)

```
SET CURRENT PACKAGE PATH :hvar1
```

このステートメントの実行後の CURRENT PACKAGE PATH 特殊レジスターの値は: "SCH4","SCH5"

例 3: CURRENT PACKAGE PATH 特殊レジスターを、以下のスキーマのリストに設定します: ″SCH1″,″SCH#2″,″SCH3″,″SCH4″,″SCH5″ (ただし、 :hvar1 は 'SCH4,SCH5' を含みます)

```
SET CURRENT PACKAGE PATH = SCH1,'SCH#2',"SCH3",:hvar1
```

このステートメントの実行後の CURRENT PACKAGE PATH 特殊レジスターの値は: ″SCH1″,″SCH#2″,″SCH3″,″SCH4″,″SCH5″

例 4: CURRENT PACKAGE PATH 特殊レジスターをクリアします。

```
SET CURRENT PACKAGE PATH = ''
```

例 5: SUMMARIZE プロシージャーの実行のために、 ″SCH_PROD″ スキーマ (:prodschema ホスト変数に含まれる) および ″SCH_PROD2″ スキーマ (:prod2schema ホスト変数に含まれる) を、 CURRENT PACKAGE PATH 特殊レジスターの末尾に一時的に付加します。それから、CURRENT PACKAGE PATH 特殊レジスターを以前の値に戻します。

```
SET :oldCPP = CURRENT PACKAGE PATH

SET CURRENT PACKAGE PATH = CURRENT PACKAGE PATH,:prodschema,:prod2schema

CALL SUMMARIZE(:V1,:V2)

SET CURRENT PACKAGE PATH = :oldCPP
```

例 6: CURRENT PACKAGE PATH 特殊レジスターを、区切り文字で区切られているスキーマ名のリストに設定します: ″MY.SCHEMA″ (組み込みピリオド)、″OLD SCHEMA″ (組み込みブランク)。両方の区切り ID を含む単一のホスト変数を使用します。

```
hv = '"MY.SCHEMA", "OLD SCHEMA"'

SET CURRENT PACKAGE PATH = :hv
```

あるいは、両方の区切り ID を含む単一のストリング定数を使用します。

```
SET CURRENT PACKAGE PATH = '"MY.SCHEMA", "OLD SCHEMA"'
```

あるいは、区切り文字で区切られているスキーマのリストを使用します。

```
SET CURRENT PACKAGE PATH = 'MY.SCHEMA', 'OLD SCHEMA'
```

関連資料:

- *SQL リファレンス 第 1 巻* の『CURRENT PACKAGE PATH 特殊レジスター』

SET CURRENT PACKAGESET

SET CURRENT PACKAGESET ステートメントは、それ以降の SQL ステートメントで使用するパッケージの選択に使用されるスキーマ名 (コレクション ID) を設定します。このステートメントは、トランザクションの制御下にありません。

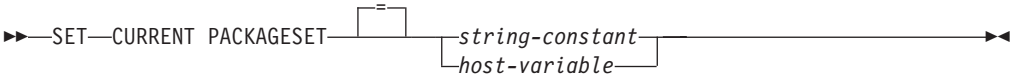
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。このステートメントは REXX ではサポートされません。

許可:

必要ありません。

構文:



説明:

string-constant
文字ストリング定数です。値が 30 バイトを超える場合は、先頭の 30 バイトだけが使用されます。

host-variable
タイプ CHAR または VARCHAR の変数です。 NULL に設定することはできません。値が 30 バイトを超える場合は、先頭の 30 バイトだけが使用されます。

注:

- このステートメントを使用して、アプリケーションは、実行可能な SQL ステートメントのパッケージを選択する際に使用するスキーマ名を指定することができます。このステートメントは、クライアントで処理され、アプリケーション・サーバーへの流れはありません。
- COLLECTION BIND オプションを使用して、指定したスキーマ名を伴うパッケージを作成できます。
- DB2 UDB for OS/390 and z/OS とは異なり、SET CURRENT PACKAGESET ステートメントは、 CURRENT PACKAGESET 特殊レジスターのサポートなしでインプリメントされています。

例:

TRYIT というアプリケーションがユーザー ID PRODUSA によってプリコンパイルされ、バインド・ファイルのデフォルトのスキーマ名は 'PRODUSA' になっていると想定します。 その後、このアプリケーションは、異なる BIND オプションを使用して 2 回バインドされます。以下のコマンド行プロセッサーのコマンドが使用されました。

```
DB2 CONNECT TO SAMPLE USER PRODUSA
DB2 BIND TRYIT.BND DATETIME USA
DB2 CONNECT TO SAMPLE USER PRODEUR
DB2 BIND TRYIT.BND DATETIME EUR COLLECTION 'PRODEUR'
```

これにより、TRYIT というパッケージが 2 つ作成されます。最初の BIND コマンドでは、'PRODUSA' というスキーマにパッケージが作成されます。 2 番目 BIND コマンドでは、COLLECTION オプションに基づいて 'PRODEUR' というスキーマにパッケージが作成されます。

ここで、アプリケーション TRYIT に、次のステートメントが含まれていると想定します。

```
EXEC SQL CONNECT TO SAMPLE;
.
.
EXEC SQL SELECT HIREDATE INTO :HD FROM EMPLOYEE WHERE EMPNO='000010'; 1
.
.
EXEC SQL SET CURRENT PACKAGESET 'PRODEUR';                            2
.
.
EXEC SQL SELECT HIREDATE INTO :HD FROM EMPLOYEE WHERE EMPNO='000010'; 3
```

1 このステートメントは、アプリケーションのデフォルトのパッケージである PRODUSA.TRYIT パッケージを使って実行されます。日付は、USA 形式で戻されます。

2 このステートメントは、パッケージ選択のスキーマ名を 'PRODEUR' に設定します。

3 SET CURRENT PACKAGESET ステートメントの結果として、このステートメントは PRODEUR.TRYIT パッケージを使用して実行されます。日付は、EUR 形式で戻されます。

SET CURRENT QUERY OPTIMIZATION

SET CURRENT QUERY OPTIMIZATION ステートメントは、 CURRENT QUERY OPTIMIZATION 特殊レジスターに値を割り当てます。この値は、動的 SQL ステートメントの準備の時点で使用される最適化手法の現行クラスを指定します。このステートメントは、トランザクションの制御下にはありません。

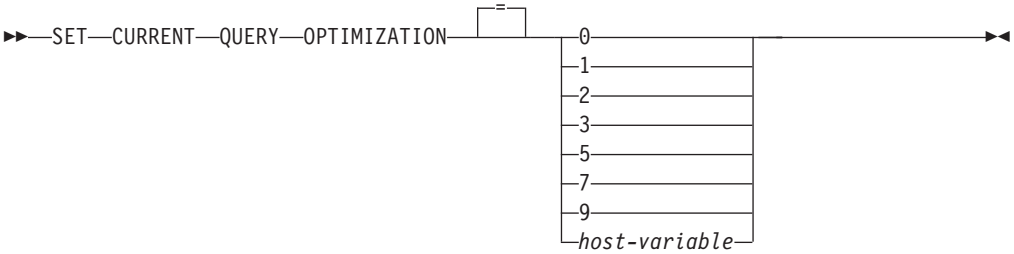
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

optimization-class

optimization-class (最適化クラス) は、整数定数、またはランタイムに適切な値が入れられるホスト変数の名前として指定することができます。 クラスの概要を以下に示します。

0 アクセス・プランの生成に、最低限の最適化が行われることを指定します。このクラスは、適切な索引が付けられた表にアクセスする単純な動的 SQL の場合に最も適しています。

1 アクセス・プランの生成に、 DB2 バージョン 1 に匹敵する最適化を行うことを指定します。

2 DB2 バージョン 1 よりも高度な最適化を指定します。ただし、特にきわめて複雑な照会の場合、レベル 3 やそれ以降よりも最適化コストは大幅に低くなります。

3 アクセス・プランの生成に、中程度の最適化を行うことを指定します。

5 アクセス・プランの生成に、かなり高度な最適化を行うことを指定します。動的 SQL 照会が複雑な場合には、アクセス・プランの選択にかかる時間を制限するのに発見的手法の規則が使用されます。 可能な場合、照会では基礎となる基本表ではなくマテリアライズ照会表が使用されます。

7 アクセス・プランの生成に、かなり高度な最適化を行うことを指定します。 5 とほとんど同じですが、発見的手法の規則が使用されない点が異なります。

9 アクセス・プランの生成に、最大限の最適化を行うことを指定します。これにより、評価対象のアクセス・プランの数は大幅に増大します。 このクラスは、大規模な表を使用するきわめて複雑で、実行に長時間を要する照会に対して、より良いアクセス・プランを生成することを判別する場合に使うようにしてください。 Explain とパフォーマンス測定値を使用することにより、効率的なプランが生成されたかどうかを検証することができます。

host-variable データ・タイプは INTEGER です。値は、0 ～ 9 の範囲内である必要があります (SQLSTATE 42815)。ただし、値は、0、1、2、3、5、7、または 9 のいずれかでなければなりません (SQLSTATE 01608)。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

注:

- CURRENT QUERY OPTIMIZATION レジスターを特定の値に設定すると、一連の照会書き直し規則が有効になり、特定の最適化変数が特定の値になります。該当のクラスの最適化手法が、動的 SQL ステートメントの準備の過程で使用されます。
- 一般に、最適化クラスの変更は、アプリケーションの実行時間、コンパイル時間、および必要なリソースに影響を与えます。多くのステートメントは、デフォルトの照会最適化クラスを用いて適切な最適化が行われます。 動的 SQL ステートメントに対して、動的 *PREPARE* が消費するリソースが、照会の実行に必要なリソースのかなりの部分を占める場合には、低い照会最適化クラス (特にクラス 1 と 2) が動的 SQL ステートメントに適している場合があります。より高いレベルの最適化クラスは、消費するリソースがどれだけ増えるかを検討し、より良いアクセス・プランが生成されたことを確認して初めて、選択するようにしてください。
- 照会最適化クラスは、0 ～ 9 の範囲でなければなりません。この範囲外のクラスは、エラーになります (SQLSTATE 42815)。 この範囲内でサポートされていないクラスを指定すると、警告 (SQLSTATE 01608) が戻され、より低い次の照会最適化クラスで置き換えられます。 たとえば、照会最適化クラス 6 は 5 に置き換えられます。
- 動的に準備されるステートメントは、最近、実行された SET CURRENT QUERY OPTIMIZATION ステートメントによって設定された最適化クラスを使用します。SET CURRENT QUERY OPTIMIZATION ステートメントがまだ実行されていない場合、照会最適化クラスはデータベース構成パラメーター dft_queryopt によって決まります。
- 静的にバインドされたステートメントでは、CURRENT QUERY OPTIMIZATION 特殊レジスターを使用しません。したがって、このレジスターはそれらのステートメントに影響を与えません。静的にバインドされたステートメントに対する必要な最適化クラスの指定には、プリプロセスまたはバインドの過程で

QUERYOPT オプションが使用されます。 QUERYOPT の指定がない場合は、データベース構成パラメーター dft_queryopt によって指定されたデフォルト値が使用されます。

- SET CURRENT QUERY OPTIMIZATION ステートメントが実行される作業単位がロールバックされても、このステートメントの実行結果はロールバックされません。

例:

例 1: この例は、最も程度の高い最適化を選択する方法を示しています。

```
SET CURRENT QUERY OPTIMIZATION 9
```

例 2: 以下の例は、照会の中で CURRENT QUERY OPTIMIZATION 特殊レジスターを使用する方法を示しています。

以下の例は、SYSCAT.PACKAGES カタログ・ビューを使用して、 CURRENT QUERY OPTIMIZATION 特殊レジスターの現行値と同じ設定でバインドされたすべてのプランを検索しています。

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT PKGNAME, PKGSCHEMA FROM SYSCAT.PACKAGES
  WHERE QUERYOPT = CURRENT QUERY OPTIMIZATION
```

SET CURRENT REFRESH AGE

SET CURRENT REFRESH AGE ステートメントは、 CURRENT REFRESH AGE 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

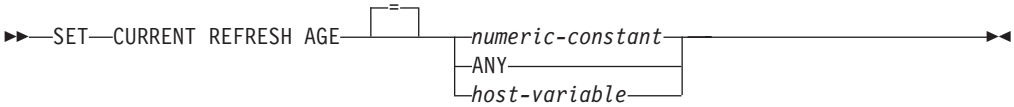
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

numeric-constant
タイム・スタンプ期間を表す DECIMAL(20,6) 値。 この値には、0 または 99 999 999 999 999 を指定しなければなりません (値のマイクロ秒部分は無視されるので、任意の値を指定できます)。

ANY
これは、99 999 999 999 999 を簡略化したものです。

host-variable
タイプ DECIMAL(20,6) の変数、または DECIMAL(20,6) に割り当て可能な他のタイプ。 NULL に設定することはできません。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。 *host-variable* の値には、 0 または 99 999 999 999 999 を指定しなければなりません。

注:

- CURRENT REFRESH AGE 特殊レジスターの初期値はゼロです。
- CURRENT REFRESH AGE は、指定した値によって置き換えられます。値は 0 または 99 999 999 999 999 でなければなりません。 99 999 999 999 999 という値は、 9999 年、99 か月、99 日、99 時間、99 分、99 秒を表します。

  CURRENT REFRESH AGE の値が 0 の場合は、この特殊レジスターから影響を受けるマテリアライズ照会表は、照会の処理を最適化するために使用されません。 CURRENT REFRESH AGE の値が 99 999 999 999 999 の場合は、この特殊レジスターから影響を受けるマテリアライズ照会表は、照会の処理を最適化するために使用できますが、それが可能なのは、CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値がそれらを含んでおり、CURRENT QUERY OPTIMIZATION 特殊レジスターが 2 か、5 以上の値に設定

されている場合だけです。この特殊レジスターから影響を受けるマテリアライズ照会表は、 REFRESH DEFERRED MAINTAINED BY USER および REFRESH DEFERRED MAINTAINED BY SYSTEM です。

CURRENT QUERY OPTIMIZATION 特殊レジスターが 2 か、5 以上の値に設定されている場合、 REFRESH IMMEDIATE MAINTAINED BY SYSTEM マテリアライズ照会表は常に、照会の処理を最適化するために使用できます。

CURRENT QUERY OPTIMIZATION 特殊レジスターが 2 か、5 以上の値に設定されていて、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値が ALL に設定されているか FEDERATED_TOOL を含む場合に、 REFRESH DEFERRED MAINTAINED BY FEDERATED_TOOL マテリアライズ照会表は照会の処理を最適化するために使用できます。

- CURRENT REFRESH AGE 特殊レジスターをゼロ以外の値に設定する場合は、注意が必要です。 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターによって指定される表タイプは、基礎となる基本表の値を表していない可能性があります。そのような表を使用して照会の処理を最適化する場合、照会結果は基礎表内のデータを正確に表していない可能性があります。とはいえ、基礎データが変化していないことが分かっている場合、あるいはキャッシュに入れられたデータに関する知識に基づいて照会結果のエラーの度合いを受け入れるつもりである場合、これはさほど気になる問題にならないことがあります。
- タイム・スタンプの算術演算では、CURRENT REFRESH AGE の値として 99 999 999 999 999 を使用することはできません。その結果が、日付の有効範囲外になるからです (SQLSTATE 22008)。

例:

例 1: 以下のステートメントは、CURRENT REFRESH AGE 特殊レジスターを設定します。

```
SET CURRENT REFRESH AGE ANY
```

例 2: 以下の例では、 CURRENT REFRESH AGE 特殊レジスターの現行値を検索して CURMAXAGE という名前のホスト変数に入れます。前の例で設定された値は 99999999999999.000000 です。

```
EXEC SQL VALUES (CURRENT REFRESH AGE) INTO  :CURMAXAGE;
```

SET ENCRYPTION PASSWORD

SET ENCRYPTION PASSWORD ステートメントは、
ENCRYPT、DECRYPT_BIN、および DECRYPT_CHAR 関数によって使用されるパスワードを設定します。このパスワードは DB2 認証には関連付けられず、データの暗号化および暗号化解除にのみ使用されます。

このステートメントは、トランザクションの制御下にありません。

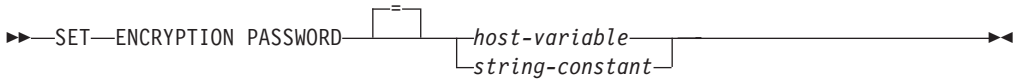
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また対話式に出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:
ENCRYPTION PASSWORD は、パスワード・ベースの暗号化のための
ENCRYPT、 DECRYPT_BIN、および DECRYPT_CHAR 組み込み関数によって使用できます。長さは 6 ～ 127 バイトの間でなければなりません。自動大文字変換は行われないため、文字はすべて大文字小文字を区別して指定しなければなりません。

host-variable
タイプ CHAR または VARCHAR の変数です。 *host-variable* の長さは 6 ～ 127 バイトの間でなければなりません (SQLSTATE 428FC)。 NULL に設定することはできません。大文字変換は行われないため、文字はすべて大文字小文字を区別して指定します。

string-constant
文字ストリング定数です。長さは 6 ～ 127 バイトの間でなければなりません (SQLSTATE 428FC)。

注:

- ENCRYPTION PASSWORD の初期値は空ストリング ('') です。
- *host-variable* または *string-constant* は、標準 DB2 メカニズムでデータベース・サーバーに送信されます。

例:

例 1: 次のステートメントは ENCRYPTION PASSWORD を設定します。

```
SET ENCRYPTION PASSWORD = 'Gre89Ea'
```

関連資料:

- *SQL リファレンス 第 1 巻* の『DECRYPT_BIN および DECRYPT_CHAR スカラー関数』
- *SQL リファレンス 第 1 巻* の『ENCRYPT スカラー関数』

SET EVENT MONITOR STATE

SET EVENT MONITOR STATE ステートメントは、イベント・モニターの活動化、または非活動化を行います。 イベント・モニターの現在の状態 (アクティブまたは非アクティブ) は、 EVENT_MON_STATE 組み込み関数によって判別することができます。 SET EVENT MONITOR STATE ステートメントは、トランザクションの制御下にありません。

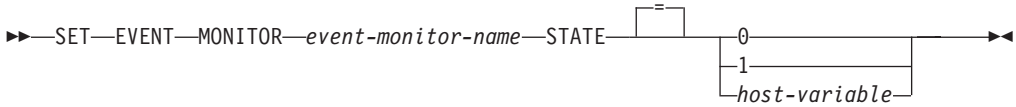
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

このステートメントの許可 ID には、 SYSADM 権限または DBADM 権限のいずれかがなければなりません (SQLSTATE 42815)。

構文:



説明:

event-monitor-name
活動化または非活動化するイベント・モニターを指定します。この名前は、カタログに存在しているイベント・モニターを指定していなければなりません (SQLSTATE 42704)。

new-state
new-state (新しい状態) は、整数定数として、またはランタイムに適切な値が入れられるホスト変数の名前として指定することができます。 指定可能な値は、次のとおりです。

0 指定したイベント・モニターを非活動化することを指定します。

1 指定したイベント・モニターを活動化することを指定します。そのイベント・モニターはすでにアクティブであってはなりません。そうでない場合、警告 (SQLSTATE 01598) が出されます。

host-variable データ・タイプは INTEGER です。指定する値は、0 または 1 でなければなりません (SQLSTATE 42815)。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

規則:

- 定義できるイベント・モニターの数には制限はありませんが、同時にアクティブ化できるイベント・モニターの数は 32 までです (SQLSTATE 54030)。

- イベント・モニターを活動化するには、そのイベント・モニターが作成されたトランザクションはコミットされていなければなりません (SQLSTATE 55033)。 この規則は、(1 つの作業単位内で) イベント・モニターを作成し、そのモニターを活動化し、その後で、トランザクションをロールバックするのを防止します。
- イベント・モニター・ファイルの数またはサイズが、 CREATE EVENT MONITOR ステートメントの MAXFILES または MAXFILESIZE に指定された値を超える場合には、エラー (SQLSTATE 54031) になります。
- イベント・モニターのターゲット・パス (CREATE EVENT MONITOR ステートメントにより指定) が、他のイベント・モニターですでに使用されている場合、エラー (SQLSTATE 51026) になります。

注:

- イベント・モニターを活動化すると、それに対応するカウンターはいずれもリセットされます。
- WRITE TO TABLE イベント・モニターは、SET EVENT MONITOR STATE を使用して開始されると、 SYSCAT.EVENTMONITORS カタログ・ビューの EVMON_ACTIVATES 列を更新します。セット演算が実行された作業単位が何らかの理由でロールバックされると、そのカタログ更新は失われます。イベント・モニターが再開したときに、ロールバックされた EVMON_ACTIVATES 値が再使用されます。

例:

次の例では、SMITHPAY というイベント・モニターを活動化しています。

```
SET EVENT MONITOR SMITHPAY STATE = 1
```

SET INTEGRITY

SET INTEGRITY ステートメントは、以下の目的で使用されます。

- 1 つまたは複数の表で保全性検査をオフにする。これには、チェック制約と参照制約の検査、データ・リンクの保全性検査、および生成された列に対する値の生成が含まれます。表が REFRESH IMMEDIATE で定義したマテリアライズ照会表であるか、 PROPAGATE IMMEDIATE 属性を指定したステージング表である場合、データの即時リフレッシュはオフになります。これにより、表はチェック・ペンディング状態 になります。この場合、限定された一連のステートメントとコマンドによる限定されたアクセスしかできません。主キー制約とユニーク制約は引き続き検査されます。
- 1 つ以上の表で、保全性検査をオンに戻し、すべての据え置き検査を実行する。表がシステムによって保守されるマテリアライズ照会表またはステージング表である場合、データは必要に応じてリフレッシュされます。マテリアライズ照会表が REFRESH IMMEDIATE 属性で定義されている場合、またはステージング表が PROPAGATE IMMEDIATE 属性で定義されている場合、データの即時リフレッシュはオンになります。 CASCADE DEFERRED オプションを使用してチェック・ペンディング状態にされた表では、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表は、(必要であれば) チェック・ペンディング・アクセスなし状態にされます。
- 1 つまたは複数の表に対して、据え置かれている保全性検査を実行することなく、保全性検査をオンにする。表が REFRESH IMMEDIATE 属性で定義したマテリアライズ照会表であるか、 PROPAGATE IMMEDIATE 属性で定義したステージング表である場合、データの即時リフレッシュはオンになります。 CASCADE DEFERRED オプションを使用してチェック・ペンディング状態にされた表では、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表は、(必要であれば) チェック・ペンディング・アクセスなし状態にされます。
- 表がすでにデータ・リンク調整ペンディング (DRP) またはデータ・リンク調整不能 (DRNP) 状態である場合に、表をチェック・ペンディング状態にする。表がどちらの状態でもない場合には、表は暫定的に DRP 状態にされ、チェック・ペンディング状態になります。
- 1 つ以上の表をデータ移動なしモードから完全アクセス・モードに戻す。
- 1 つ以上のステージング表の内容を整理する。

表がロードされた後でその保全性を検査するためにステートメントを使用する場合、システムは、制約に違反する追加部分だけを検査するという、増分的な表の処理を行います。サブジェクト表がマテリアライズ照会表かステージング表であり、その基礎表でロード操作が実行される場合、システムは、増分的にマテリアライズ照会表をリフレッシュするか、基礎表の追加部分だけを使用して、増分的にステージング表を伝搬します。ただし、システムでは、そのような最適化を実行できなかったりする場合や、データ保全性を保証するために、完全処理を行うことにする場合もあります。完全検査は、制約違反がないか表全体を検査すること、マテリアライズ照会表の定義を再計算すること、またはステージング表を矛盾しているとしてマーク付けすることで行われます。後者の方法の場合、関連するマテリアライズ照会表のフル・リフレッシュが必要であることを意味します。また、ユーザーが INCREMENTAL オプションを指定して、増分処理を明示的に要求できる場合もあります。

SET INTEGRITY ステートメントは、トランザクションの制御下にありません。

呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。 DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可:

SET INTEGRITY の実行に必要な特権は、ステートメントの使用法によって以下のように異なります。

- 保全性検査をオフにする場合。

  ステートメントの許可 ID の特権には、少なくとも次のいずれかが含まれている必要があります。
  - 以下に対する CONTROL 特権
    - 指定された表
    - ステートメントによって保全性検査がオフにされた下層外部キー表
    - ステートメントによって保全性検査がオフにされた下層即時マテリアライズ照会表
    - ステートメントによって保全性検査がオフにされた下層即時ステージング表
  - SYSADM または DBADM 権限
  - LOAD 権限
- 保全性検査をオンにして、検査を実行する場合。

  ステートメントの許可 ID の特権には、少なくとも次のいずれかが含まれている必要があります。
  - SYSADM または DBADM 権限
  - 検査する表に対する CONTROL 特権、および 1 つまたは複数の表に対して例外がポストされる場合は例外表に対する INSERT 特権。ステートメントによって暗黙的にチェック・ペンディング状態にされた、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表すべてに対する CONTROL 特権。
  - LOAD 権限。 1 つまたは複数の表に対して例外が通知される場合は、以下の特権も必要。
    - 検査されるそれぞれの表に対する SELECT および DELETE 特権、および
    - 例外表に対する INSERT 特権
- 最初に検査を実行することなく保全性検査をオンにする場合。

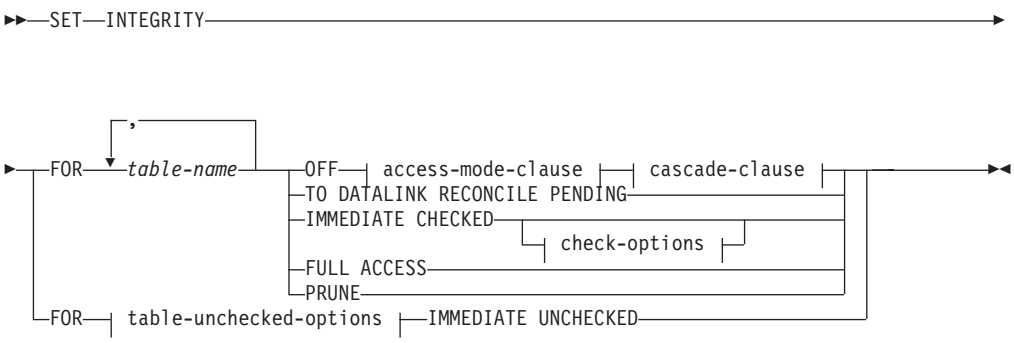
  このステートメントの許可 ID には、少なくとも次のいずれかが必要です。
  - SYSADM または DBADM 権限
  - 検査する表に対する CONTROL 特権。ステートメントによって暗黙的にチェック・ペンディング状態にされた、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表それぞれに対する CONTROL 特権

  - LOAD 権限
- 表をデータ移動なしモードから完全アクセス・モードにする場合。

  このステートメントの許可 ID には、少なくとも次のいずれかが必要です。
  - SYSADM または DBADM 権限
  - データ移動なしモードから完全アクセス・モードにされる表に対する CONTROL 特権
  - LOAD 権限
- ステージング表を整理する場合。

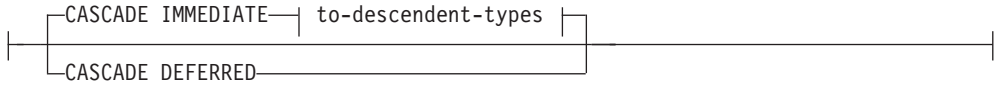
  このステートメントの許可 ID には、少なくとも次のいずれかが必要です。
  - SYSADM または DBADM 権限
  - 整理する表に対する CONTROL 特権

構文:

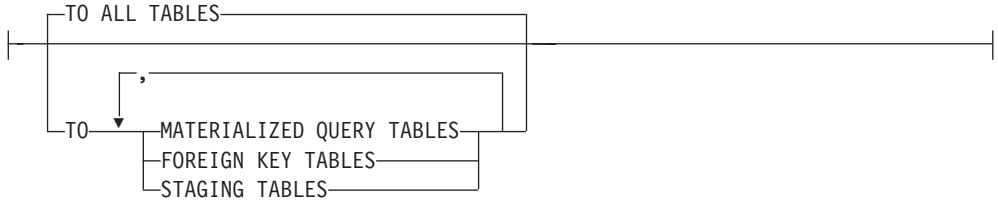


access-mode-clause:

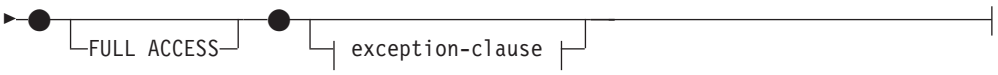
cascade-clause:



to-descendent-types:

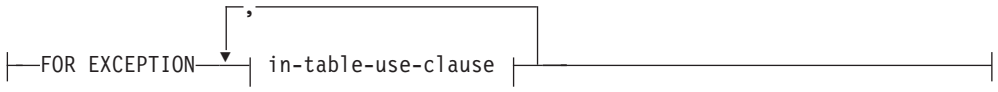


check-options:



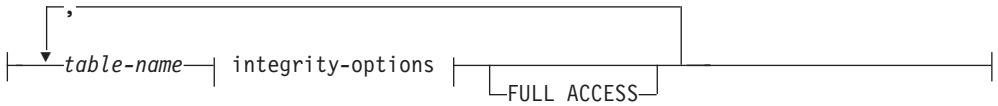
incremental-options:

exception-clause:

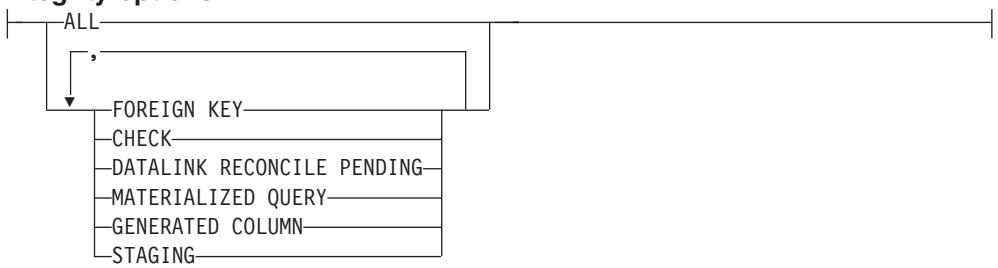


in-table-use-clause:

table-unchecked-options:



integrity-options:



説明:

FOR *table-name*

保全性処理を行う表 (複数可) を指定します。 これは、カタログに記述されている表でなければならず、ビュー、カタログ表、または型付き表を対象にすることはできません。

OFF

表の外部キー制約、チェック制約、および列の生成をオフにして、表をチェック・ペンディング状態にすることを指定します。マテリアライズ照会表かステージング表の場合、即時リフレッシュはオフになり (該当する場合)、そのマテリアライズ照会表またはステージング表はチェック・ペンディング状態になります。

表がすでに、保全性検査の 1 つのタイプだけがオフで、チェック・ペンディング状態になっている可能性がある点に注意してください。そのような場合、他のタイプの保全性検査もオフになります。

チェック・ペンディング状態にある表に対しては、極めて限定されたアクティビティーのみが許されます。

access-mode-clause
チェック・ペンディング状態のときの表の読み取り可否を指定します。

NO ACCESS
表をチェック・ペンディング・アクセスなし状態にすることを指定します。この状態では、表への読み取りまたは書き込みアクセスは許可されません。

READ ACCESS
表をチェック・ペンディング読み取り状態にすることを指定します。この状態では、表の追加部分以外への読み取りアクセスが許可されています。このオプションは、チェック・ペンディング・アクセスなし状態の表に対しては許可されていません (SQLSTATE 428FH)。

access-mode-clause を指定しない場合、表はチェック・ペンディング・アクセスなし状態になります。

cascade-clause
SET INTEGRITY ステートメントで参照される表のチェック・ペンディング状態が、すべての下層外部キー表、すべての下層即時マテリアライズ照会表、およびすべての下層即時ステージング表に対して、すぐにカスケードされるかどうかを指定します。

CASCADE IMMEDIATE
外部キー制約のチェック・ペンディング状態が、すべての下層外部キー表に拡大されることを指定します。表に下層即時マテリアライズ照会表か下層即時ステージング表が含まれる場合、チェック・ペンディング状態は、すぐにマテリアライズ照会表およびステージング表に拡大されます。

表が、保全性違反について後で検査され、チェック・ペンディング状態から抜け出すと、チェック・ペンディング読み取り状態の下層表はすべて、チェック・ペンディング・アクセスなし状態になります。

to-descendent-types

TO ALL TABLES
チェック・ペンディング状態が、呼び出しリストにある表のすべての下層表に対して、すぐにカスケードされることを指定します。下層表には、呼び出しリストの表の下層である、あるいは下層外部キー表の下層である、すべての下層外部キー表、即時ステージング表、および即時マテリアライズ照会表が含まれます。

TO ALL TABLES を指定することは、同じステートメントに、 TO FOREIGN KEY TABLES、TO MATERIALIZED QUERY TABLES、および TO STAGING TABLES すべてを指定することと同じです。

TO MATERIALIZED QUERY TABLES
TO MATERIALIZED QUERY TABLES だけを指定する場合、チェック・ペンディング状態は、すぐに、下層即時マテリアライズ照会表に対してだけカスケードされます。他の下層表は、表がチェック・ペンディング状態でなくなるような場合には、必要であれば後でチェック・ペンディング状態にすることができます。 TO FOREIGN KEY TABLES と TO MATERIALIZED QUERY TABLES の両方が指定される場合、チェック・ペンディング状態はすぐに、すべての下層外部キー表、呼び出し

リストにある表のすべての下層即時マテリアライズ照会表、そして、下層外部キー表の下層であるすべての即時マテリアライズ照会表にカスケードされます。

TO FOREIGN KEY TABLES
チェック・ペンディング状態が、下層外部キー表に対して、すぐにカスケードされることを指定します。他の下層表は、表がチェック・ペンディング状態でなくなるような場合には、必要であれば後でチェック・ペンディング状態にすることができます。

TO STAGING TABLES
チェック・ペンディング状態が、下層ステージング表に対して、すぐにカスケードされることを指定します。他の下層表は、表がチェック・ペンディング状態でなくなるような場合には、必要であれば後でチェック・ペンディング状態にすることができます。 TO FOREIGN KEY TABLES と TO STAGING TABLES の両方が指定される場合、チェック・ペンディング状態はすぐに、すべての下層外部キー表、呼び出しリストにある表のすべての下層即時ステージング表、そして、下層外部キー表の下層であるすべての即時ステージング表にカスケードされます。

CASCADE DEFERRED
呼び出しリストに含まれる表だけがチェック・ペンディング状態になることを指定します。下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表の状態は、変更されないままです。下層外部キー表は、(SET INTEGRITY ステートメントの IMMEDIATE CHECKED オプションを使用して) その親表の制約違反を検査するときに、後で暗黙的にチェック・ペンディング・アクセスなし状態にすることができます。下層即時マテリアライズ照会表および下層即時ステージング表は、基礎表のいずれかで保全性違反を検査するときに、暗黙的にチェック・ペンディング・アクセスなし状態になる場合があります。下層表がチェック・ペンディング状態になったことを示す警告 (SQLSTATE 01586) が発行されます。

cascade-clause を指定しない場合、チェック・ペンディング状態は、すぐに従属表および下層表にカスケードされます。

TO DATALINK RECONCILE PENDING
表の DATALINK 保全性検査をオフにすることと、表をチェック・ペンディング・アクセスなし状態にすることを指定します。表がすでにデータ・リンク調整不能 (DRNP) 状態である場合は、そのままチェック・ペンディング状態になります。 そうでない場合には、表はデータ・リンク調整ペンディング (DRP) 状態に設定されます。

このオプションを指定しても、従属表および下層表には影響がありません。

IMMEDIATE CHECKED
表の保全性検査をオンにし、据え置かれている保全性検査を行うことを指定します。 これは、SYSCAT.TABLES カタログの STATUS 列と CONST_CHECKED 列の情報に基づいて行われます。 すなわち、

- 表が、リストで指定され、チェック・ペンディング状態にあり、さらに中間上層もリストに含まれる表の下層外部キー表、下層マテリアライズ照会表、または下層ステージング表でない限り、 STATUS の値は C (表はチェック・

ペンディング状態にある) でなければなりません。そうでない場合には、エラー (SQLSTATE 51027) が戻されます。

- 検査する表がチェック・ペンディング状態にある場合、CONST_CHECKED の値は、どの保全性オプションを検査するかを示します。

表が、CASCADE DEFERRED オプションを使用してチェック・ペンディング状態にされた場合、その下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表は、必要であれば、チェック・ペンディング・アクセスなし状態にされます。下層表がチェック・ペンディング状態になったことを示す警告 (SQLSTATE 01586) が発行されます。

表がシステムによって保守されるマテリアライズ照会表であれば、データは必要に応じて照会で検査され、リフレッシュされます。 (このステートメントは、ユーザーが保守するマテリアライズ照会表には使用できません。) 表がステージング表であれば、データは必要に応じてその照会定義に対して検査されて伝搬されます。

子表の保全性を検査する場合、以下のようになります。

- 親をチェック・ペンディング状態にすることはできません
- それぞれの親は、同じ SET INTEGRITY ステートメントで制約違反を検査される必要があります

即時マテリアライズ照会表がリフレッシュされる場合、あるいは差分がステージング表に伝搬される場合、以下のようになります。

- 基礎表をチェック・ペンディング状態にすることはできません
- それぞれの基礎表は、同じ SET INTEGRITY ステートメントで検査される必要があります

これ以外の場合には、エラーになります (SQLSTATE 428A8)。

表が DRP または DRNP 状態であっても、DATALINK 値は検査されません。DATALINK 値の調整を実行するには、RECONCILE コマンドまたは API を使用してください。表はチェック・ペンディング状態ではなくなりますが、DRP または DRNP フラグはセットされたままとなります。 DATALINK 値の調整が据え置かれているため、表を使用できます。

check-options

incremental-options

INCREMENTAL
表の追加部分 (もしあれば) に対して保全性検査を適用することを指定します。 この要求が満たされない場合 (たとえば、システムが表全体でデータ保全性検査を実行する必要があると判断する場合)、エラー (SQLSTATE 55019) が戻されます。

NOT INCREMENTAL
表全体に対して保全性検査を適用することを指定します。表がマテリアライズ照会表である場合、マテリアライズ照会表定義が再計算されます。表に少なくとも 1 つの制約が定義されている場合、このオプションを指定すると、下層外部キー表と下層即時マテリアライズ照会表が完全処理されます。表がステージング表である場合、矛盾状態に設定されます。

incremental-options 文節を指定しない場合、システムは増分処理が可能かどうかを判断します。それが可能でなければ、表全体が検査されます。

FORCE GENERATED
表に生成された列が含まれている場合は、式に基づいてその値が計算され、列に保管されます。この文節が指定されない場合は、等価検査の制約があるかのように、現行値が式の算出値と比較されます。表の保全性が増分的に検査される場合、生成された列は追加部分についてのみ計算されます。

PRUNE
このオプションは、ステージング表の場合にのみ指定できます。ステージング表の内容を整理すること、ステージング表を矛盾状態にすることを指定します。 *table-name* リストに含まれている表がステージング表でなければ、エラーが戻されます (SQLSTATE 428FH)。
INCREMENTAL 検査オプションも指定されている場合、エラーが戻されます (SQLSTATE 428FH)。

FULL ACCESS
SET INTEGRITY ステートメントの実行後に表が完全にアクセス可能になることを指定します。このオプションは、IMMEDIATE CHECKED 文節と IMMEDIATE UNCHECKED 文節の両方で指定できます。

呼び出しリストの基礎表が増分的に処理され、従属の即時マテリアライズ照会表か、従属の即時ステージング表を持つ場合、基礎表は、SET INTEGRITY ステートメントの後に、必要に応じてデータ移動なしモードにされます。増分的にリフレッシュ可能な従属の即時マテリアライズ照会表およびステージング表がチェック・ペンディング状態でなくなると、基礎表は、自動的にデータ移動なしモードから完全アクセス・モードになります。 IMMEDIATE CHECKED 文節または IMMEDIATE UNCHECKED 文節に FULL ACCESS オプションが指定される場合、基礎表は、データ移動なしモードにならずに、直接に完全アクセス・モードになります。リフレッシュされたことのない従属の即時マテリアライズ照会表は、後続の REFRESH ステートメントですべてが再計算される可能性があり、伝搬される表の追加部分を持たない従属の即時ステージング表は、矛盾状態であるというフラグが立てられる可能性があります。

呼び出しリストの基礎表が、完全処理を必要とするか、従属の即時マテリアライズ照会表、あるいは従属の即時ステージング表を持たない場合には、その基礎表は、FULL ACCESS オプションが指定されているかどうかに関係なく、 SET INTEGRITY ステートメントの後に直接に完全アクセス・モードにされます。

IMMEDIATE UNCHECKED 文節に FULL ACCESS オプションを指定して、そのステートメントが表をチェック・ペンディング状態から解放しない場合、エラーになります (SQLSTATE 428FH)。

exception-clause

FOR EXCEPTION
外部キー制約またはチェック制約に違反している行があれば、その

行を例外表にコピーして、元の表から削除することを指定します。エラーが検出されても、制約は再びオンに戻り、表はチェック・ペンディング状態ではなくなります。 1 行以上の行が例外表に移されたことを示す警告 (SQLSTATE 01603) が出されます。

FOR EXCEPTION 文節の指定がない場合に、制約違反が生じると、検出された最初の違反だけがユーザーに戻されます (SQLSTATE 23514)。 表のいずれかに違反がある場合、すべての表は、ステートメント実行前と同じチェック・ペンディング状態のままになります。

IN *table-name*
制約に違反している行のコピー元である表を *table-name* に指定します。検査される各表ごとに、1 つの例外表を指定する必要があります。この文節は、表がマテリアライズ照会表かステージング表の場合、指定できません (SQLSTATE 428A7)。

USE *table-name*
エラー行のコピー先である例外表を *table-name* に指定します。

FULL ACCESS
ステートメントの唯一の操作として FULL ACCESS オプションが指定される場合、表はデータ移動なしモードから完全アクセス・モードにされます。表の保全性違反について再検査されることはありません。しかし、リフレッシュされたことのない従属の即時マテリアライズ照会表は、後続の REFRESH ステートメントですべてを再計算しなければならない可能性があり、伝搬される表の追加部分を持たない従属の即時ステージング表は、未完了状態に変更される可能性があります。このオプションは、データ移動なしモードの表にのみ指定できます (SQLSTATE 428FH)。

PRUNE
このオプションは、ステージング表の場合にのみ指定できます。ステージング表の内容を整理すること、ステージング表を矛盾状態にすることを指定します。*table-name* リストに含まれている表がステージング表でなければ、エラーが戻されます (SQLSTATE 428FH)。

table-unchecked-options

table-name
保全性処理を行う表 (複数可) を指定します。 これは、カタログに記述されている表でなければならず、ビュー、カタログ表、または型付き表を対象にすることはできません。

integrity-options
IMMEDIATE UNCHECKED に設定される保全性オプションを定義するのに使用します。

ALL
保全性オプションすべてがオンにされ、表がチェック・ペンディング状態ではなくなります。

FOREIGN KEY
表がチェック・ペンディング状態になると、外部キー制約がオンになります。

CHECK
表がチェック・ペンディング状態になると、チェック制約がオンになります。

DATALINK RECONCILE PENDING
表がチェック・ペンディング状態になると、DATALINK 保全性制約がオンになります。

MATERIALIZED QUERY
REFRESH IMMEDIATE 属性を指定したマテリアライズ照会表で、即時リフレッシュがオンになります。

GENERATED COLUMN
表がチェック・ペンディング状態になると、生成された列がオンになります。

STAGING
ステージング表で、即時伝搬がオンになります。

FULL ACCESS
SET INTEGRITY ステートメントの実行後に表が完全にアクセス可能になることを指定します。このオプションは、IMMEDIATE CHECKED 文節と IMMEDIATE UNCHECKED 文節の両方で指定できます。

呼び出しリストの基礎表が増分的に処理され、従属の即時マテリアライズ照会表か、従属の即時ステージング表を持つ場合、基礎表は、SET INTEGRITY ステートメントの後に、必要に応じてデータ移動なしモードにされます。増分的にリフレッシュ可能な従属の即時マテリアライズ照会表およびステージング表がチェック・ペンディング状態でなくなると、基礎表は、自動的にデータ移動なしモードから完全アクセス・モードになります。IMMEDIATE CHECKED 文節または IMMEDIATE UNCHECKED 文節に FULL ACCESS オプションが指定される場合、基礎表は、データ移動なしモードにならずに、直接に完全アクセス・モードになります。リフレッシュされたことのない従属の即時マテリアライズ照会表は、後続の REFRESH ステートメントですべてが再計算される可能性があり、伝搬される表の追加部分を持たない従属の即時ステージング表は、矛盾状態であるというフラグが立てられる可能性があります。

呼び出しリストの基礎表が、完全処理を必要とするか、従属の即時マテリアライズ照会表、あるいは従属の即時ステージング表を持たない場合には、その基礎表は、FULL ACCESS オプションが指定されているかどうかに関係なく、 SET INTEGRITY ステートメントの後に直接に完全アクセス・モードにされます。

IMMEDIATE UNCHECKED 文節に FULL ACCESS オプションを指定して、そのステートメントが表をチェック・ペンディング状態から解放しない場合、エラーになります (SQLSTATE 428FH)。

IMMEDIATE UNCHECKED
以下のいずれかを指定します。

- 保全性違反についての表の検査は行わずに、表の保全性検査をオンにします (その結果、チェック・ペンディング状態ではなくなります)。

これは、ALL を指定するか、該当する表に対してオフになっている保全性オプションごとに、他の *integrity-options* を 1 つ以上指定することによって指定されます。

- 表の 1 つのタイプの保全性検査だけをオンにして、このタイプの保全性違反は検査しませんが、表はチェック・ペンディング状態のままにします。

  これは、該当する表に対して少なくとも 1 つの保全性オプションがオフのままにされている場合に、 1 つ以上の CHECK、FOREIGN KEY、MATERIALIZED QUERY、STAGING、GENERATED COLUMN、または DATALINK RECONCILE PENDING を指定することによって指定されます。

このオプションの使用に先立って、データ保全性に関連する影響について検討する必要があります。このステートメントの『注』の節を参照してください。

注:

- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - SET INTEGRITY の代わりに SET CONSTRAINTS を指定できます。
    - MATERIALIZED QUERY の代わりに SUMMARY を指定できます。
- チェック・ペンディング状態のいずれかの表への影響 :
  - INSERT、UPDATE、または DELETE は、チェック・ペンディング読み取り状態またはチェック・ペンディング・アクセスなし状態の表に対しては実行できません。さらに、チェック・ペンディング状態の表にそのような変更を加える必要のあるステートメントはリジェクトされます。

    たとえば、チェック・ペンディング状態にある従属表にカスケードする親表の行の削除は実行できません。
  - SELECT は、チェック・ペンディング・アクセスなし状態の表に対しては実行できません。さらに、チェック・ペンディング・アクセスなし状態の表への読み取りアクセスが必要なステートメントはリジェクトされます。
  - 表に新しく追加される制約は、通常、ただちに適用されます。 ただし、表がチェック・ペンディング状態の場合は、表がチェック・ペンディング状態でなくなるまで、新しい制約の検査は据え置かれます。表がチェック・ペンディング読み取り状態にある場合、データの検査にはリスクが伴うため、新しい制約を追加すると、表はチェック・ペンディング・アクセスなし状態になります。
  - CREATE INDEX ステートメントでは、チェック・ペンディング状態にある表を参照できません。 同様に、主キー制約またはユニーク制約を追加する ALTER TABLE ステートメントでは、チェック・ペンディング状態にある表を参照できません。
  - IMPORT ユーティリティーは、チェック・ペンディング状態にある表に対しては許可されていません。 (IMPORT ユーティリティーは、LOAD ユーティリティーとは異なり、常に制約検査をただちに実行します。)
  - EXPORT ユーティリティーは、チェック・ペンディング・アクセスなし状態の表に対する操作が許可されていませんが、チェック・ペンディング読み取り状

態の表に対する操作は許可されています。表がチェック・ペンディング読み取り状態の場合、EXPORT ユーティリティーは、追加部分以外にあるデータだけをエクスポートします。

– 表の中でのデータ移動が関係することのある操作 (REORG、REDISTRIBUTE、パーティション・キーの更新、クラスタリング・キーの更新など) では、いずれかのチェック・ペンディング状態にある、あるいはデータ移動なしモードにある表に対する操作は許可されていません。

– ユーティリティー LOAD、 BACKUP、 RESTORE、 UPDATE STATISTICS、 RUNSTATS、 REORGCHK、 LIST HISTORY、および ROLLFORWARD は、チェック・ペンディング状態の表に対して実行可能です。

– ステートメント ALTER TABLE、 COMMENT、 DROP TABLE、 CREATE ALIAS、 CREATE TRIGGER、 CREATE VIEW、 GRANT、 REVOKE、および SET INTEGRITY では、チェック・ペンディング状態の表を参照することができます。しかしこれにより、表がアクセスなしモードにされる可能性があります。

– チェック・ペンディング・アクセスなし状態の表に従属しているパッケージ、ビュー、およびその他のオブジェクトは、ランタイムにその表がアクセスされると、エラーを戻します。チェック・ペンディング状態の表に従属しているパッケージは、ランタイムにその表に対して挿入、更新、または削除操作が試行されると、エラーを戻します。

SET INTEGRITY ステートメントによる違反行の除去は、削除イベントではありません。したがって、SET INTEGRITY ステートメントではトリガーは活動化されません。同様に、FORCE GENERATED オプションを使用して生成された列を更新しても、トリガーは活動化されません。

- 状態が許すときには、増分処理が使用されます。増分処理はより効率的です。INCREMENTAL オプションは多くの場合必要ありません。しかし、保全性検査が確実に増分的に行われることを保証するため、このオプションが必要になります。システムが、データ保全性を保証するために完全処理が必要だと判断すると、エラー (SQLSTATE 55019) が戻されます。
- IMMEDIATE UNCHECKED 文節の使用に関する警告 :

  – この文節は、ユーティリティー・プログラムで使用することを意図しているので、アプリケーション・プログラムによる使用はお勧めしません。その表に定義された保全性仕様を満たさない表にデータが存在する場合で、 IMMEDIATE UNCHECKED 文節が使用される場合、不正確な照会結果が戻されることがあります。

    据え置き状態の検査を行うことなく保全性検査がオンになったという事実は、カタログに記録されます (SYSCAT.TABLES ビューの CONST_CHECKED 列の値が 'U' に設定されます)。これは、特定の制約に関するデータ保全の責任はユーザーにあることを示しています。この値は、以下のいずれかの条件が満たされるまで変更されません。

    - CONST_CHECKED 列にある 'U' 値が 'W' 値に変更になると、 OFF 文節を指定した SET INTEGRITY ステートメントで表を参照することによって、表がチェック・ペンディング状態に戻る。これは、データ保全性の責任が以前はユーザーにあったと見なされていて、システムがデータを検査する必要があることを示します。

- 検査されていないすべての表の制約がドロップされる。

'W' 状態は 'N' 状態と違って、保全性が以前はシステムではなくユーザーによって検査されていたことを記録しています。ユーザーが NOT INCREMENTAL オプションを指定した SET INTEGRITY ... IMMEDIATE CHECKED ステートメントを発行すると、システムは、表全体のデータ保全性を再検査 (または、マテリアライズ照会表で完全リフレッシュを実行) してから、 'W' 状態を 'Y' 状態に変更します。 IMMEDIATE UNCHECKED が指定されるか、NOT INCREMENTAL が指定されない場合、 'W' 状態は変更されて 'U' 状態に戻され、一部のデータがまだシステムで検査されていないことを記録されます。後者の場合 (NOT INCREMENTAL が指定されない場合)、警告 (SQLSTATE 01636) が戻されます。

基礎表の保全性が IMMEDIATE UNCHECKED 文節を使用して検査された場合、基礎表の CONST_CHECKED 列にある 'U' の値は、以下の表の対応する CONST_CHECKED 列に伝搬されます。

- 従属即時マテリアライズ照会表
- 従属据え置きマテリアライズ照会表
- 従属ステージング表

従属即時マテリアライズ照会表の場合、この伝搬は、基礎表がチェック・ペンディング状態でなくなるとき、およびマテリアライズ照会表がリフレッシュされるときに必ず行われます。従属据え置きマテリアライズ照会表の場合、この伝搬は、マテリアライズ照会表がリフレッシュされるときに必ず行われます。従属ステージング表の場合、この伝搬は、基礎表がチェック・ペンディング状態でなくなるときに必ず行われます。従属マテリアライズ照会表およびステージング表の CONST_CHECKED 列に示される、これらの伝搬された 'U' の値では、これらのマテリアライズ照会表およびステージング表は、保全性が IMMEDIATE UNCHECKED 文節を使用して検査された基礎表に従属していることを記録します。

マテリアライズ照会表の場合、基礎表によって伝搬された CONST_CHECKED 列の 'U' の値は、マテリアライズ照会表が完全にリフレッシュされ、すべての基礎表の対応する CONST_CHECKED 列に 'U' の値がなくなるまで、そのまま変わりません。リフレッシュが行われたら、マテリアライズ照会表の CONST_CHECKED 列にある 'U' の値は、 'Y' に変更されます。

ステージング表の場合、基礎表によって伝搬された CONST_CHECKED 列の 'U' の値は、ステージング表の対応する据え置きマテリアライズ照会表がリフレッシュされるまで、そのまま変わりません。リフレッシュが行われたら、ステージング表の CONST_CHECKED 列にある 'U' の値は、'Y' に変更されます。

– 子表とその親表が同じ SET INTEGRITY ... IMMEDIATE CHECKED ステートメントで検査され、親表で制約を完全に検査する必要がある場合、子表の外部キー制約の CONST_CHECKED 列に 'U' の値があるかどうかに関係なく、子表では外部キー制約が検査されます。

• LOAD INSERT を使用してデータを追加した後、SET INTEGRITY ... IMMEDIATE CHECKED ステートメントは、表の制約違反を検査します。表に対

する増分処理が可能かどうかは、システムが判断します。可能な場合には、追加部分だけが保全性違反を検査されます。不可能な場合には、システムは、表全体の保全性違反を検査します。

- 次のステートメントについて考慮します。

  ```
  SET INTEGRITY FOR T IMMEDIATE CHECKED
  ```

  システムが完全なリフレッシュを必要とする状況、または表全体の保全性 (INCREMENTAL オプションは指定できない) を検査する状況は、以下のとおりです。

  – T そのものに新しい制約が追加された場合。
  – T、その親、またはその基礎表に対する LOAD REPLACE 操作が生じた場合。
  – T、その親、またはその基礎表に対する最後の保全性検査の後に、 NOT LOGGED INITIALLY WITH EMPTY TABLE オプションが活動化された場合。
  – 完全処理のカスケード効果により、T の親 (T がマテリアライズ照会表かステージング表である場合には、基礎表) について、増分的ではない方法で保全性が検査された場合。
  – チェック・ペンディング状態の表を移行したため、移行後初めて表の保全性検査を行う際に完全処理が必要な場合。
  – 表またはその親 (またはマテリアライズ照会表またはステージング表の基礎表) を含む表スペースが、ある時点までロールフォワードされ、表およびその親 (表がマテリアライズ照会表またはステージング表の場合は基礎表) が別の表スペースに存在する場合。
  – T がマテリアライズ照会表で、最後のリフレッシュ後に、 T に対する直接の LOAD REPLACE または LOAD INSERT 操作が行われる場合。

- 上記の完全処理の条件が満たされない場合、システムは、追加部分の保全性だけを検査しようとするか、ユーザーがステートメント `SET INTEGRITY FOR T IMMEDIATE CHECKED` に NOT INCREMENTAL オプションを指定していなければ、増分リフレッシュを実行します (マテリアライズ照会表の場合)。
- データ・リンク調整不能 (DRNP) 状態の表は、 (おそらくデータベースの外部で) 訂正処置が必要です。 訂正処置が完了すれば、IMMEDIATE UNCHECKED を使用することにより、表は DRNP 状態から脱することができます。 DATALINK 保全性制約を検査するには、RECONCILE コマンドまたは API を使用してください。
- 保全性が検査されている間、 SET INTEGRITY の呼び出し中に指定された各表に対して排他ロックがかけられます。 SET INTEGRITY 呼び出し時に指定されなかったが、検査対象であるいずれかの従属表の親表または基礎表である表については、共用ロックがかけられます。
- 保全性検査の過程でエラーが発生すると、 (元の表からの削除や例外表への挿入を含め) すべての検査結果がロールバックされます。
- FORCE GENERATED オプションを指定して発行された SET INTEGRITY ステートメントが、ログ・スペースの不足のために失敗する場合、使用できるアクティブなログ・スペースを増やし、 SET INTEGRITY ステートメントを再発行します。別の方法としては、SET INTEGRITY GENERATED COLUMN IMMEDIATE UNCHECKED ステートメントを使用し、表の生成された列検査を回避します。そ

の後、FORCE GENERATED オプションを指定せずに SET INTEGRITY IMMEDIATE CHECKED ステートメントを発行し、他の保全性違反 (該当する場合) があるかどうか表を検査し、チェック・ペンディング状態から解放します。表がチェック・ペンディング状態でなくなると、UPDATE ステートメントのキーワード DEFAULT に生成された列を割り当てることにより、生成された列をそのデフォルトの (生成された) 値に更新できます。このことは、範囲に基づいて複数の検索済み更新ステートメントを使用する方法 (それぞれの後にコミットされる) と、断続的なコミットを使用したカーソル・ベースによる方法のいずれかを使用することで、実現されます。カーソル・ベースによる方法を使用した断続的なコミットの後で、ロックを保存する場合には、 『WITH HOLD』 カーソルを使用する必要があります。

- (SET INTEGRITY ステートメントか LOAD コマンドの) CASCADE DEFERRED オプションを使用してチェック・ペンディング状態にされ、 SET INTEGRITY ステートメントの IMMEDIATE CHECKED オプションを使用して保全性違反を検査される表には、必要に応じてチェック・ペンディング・アクセスなし状態にされる、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表があります。
  - 保全性違反について表全体が検査される場合、その下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表は、チェック・ペンディング・アクセスなし状態にされます。
  - 保全性違反について表が増分的に検査される場合、その下層即時マテリアライズ照会表とステージング表は、チェック・ペンディング・アクセスなし状態にされ、その下層外部キー表は、元の状態のままにされます。
  - 表を検査する必要がまったくない場合、その下層即時マテリアライズ照会表、下層ステージング表、および下層外部キー表は、元の状態のままにされます。
- (SET INTEGRITY ステートメントか LOAD コマンドの) CASCADE DEFERRED オプションを使用してチェック・ペンディング状態にされ、 SET INTEGRITY ステートメントの IMMEDIATE UNCHECKED オプションを使用してチェック・ペンディング状態から解放される表には、必要に応じてチェック・ペンディング・アクセスなし状態にされる、下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表があります。
  - 表が REPLACE モードを使用してロードされた場合、その下層外部キー表、下層即時マテリアライズ照会表、および下層即時ステージング表は、チェック・ペンディング・アクセスなし状態にされます。
  - 表が INSERT モードを使用してロードされた場合、その下層即時マテリアライズ照会表とステージング表は、チェック・ペンディング・アクセスなし状態にされ、その下層外部キー表は、元の状態のままにされます。
  - 表がロードされていない場合、その下層即時マテリアライズ照会表、下層ステージング表、および下層外部キー表は、元の状態のままにされます。

例:

例 1: 以下は、表のチェック・ペンディング状態に関する情報を提供する照会の例です。 SUBSTR を使用して、SYSCAT.TABLES の CONST_CHECKED 列の最初の 2 バイトを抽出しています。 最初の 1 バイトは外部キー制約、2 番目バイトはチェック制約を表します。 STATUS ではチェック・ペンディング状態になり、ACCESS_MODE ではアクセス・モードになります。

```
SELECT TABNAME, STATUS, ACCESS_MODE,
  SUBSTR( CONST_CHECKED, 1, 1 ) AS FK_CHECKED,
  SUBSTR( CONST_CHECKED, 2, 1 ) AS CC_CHECKED
  FROM SYSCAT.TABLES
```

例 2: 表 T1 と T2 をチェック・ペンディング・アクセスなし状態に設定し、チェック・ペンディング状態をすぐに下層表へカスケードします。

```
SET INTEGRITY FOR T1, T2 OFF NO ACCESS CASCADE IMMEDIATE
```

例 3: 親表 T1 をチェック・ペンディング読み取り状態に設定し、チェック・ペンディング状態をすぐに子表 T2 にカスケードすることはしません。

```
SET INTEGRITY FOR T1 OFF READ ACCESS CASCADE DEFERRED
```

例 4: T1 の保全性を検査して、最初の違反のみを入手します。

```
SET INTEGRITY FOR T1 IMMEDIATE CHECKED
```

例 5: T1 と T2 の保全性を検査し、違反する行を例外表 E1 と E2 に入れます。

```
SET INTEGRITY FOR T1, T2 IMMEDIATE CHECKED
  FOR EXCEPTION IN T1 USE E1,
                IN T2 USE E2
```

例 6: T1 の FOREIGN KEY 制約検査を使用可能にし、 T2 の CHECK 制約検査を IMMEDIATE UNCHECKED オプションを指定してバイパスします。

```
SET INTEGRITY FOR T1 FOREIGN KEY,
                                    T2 CHECK IMMEDIATE UNCHECKED
```

例 7: 2 つの ALTER TABLE ステートメントを使用して、チェック制約と外部キーを EMP_ACT 表に追加します。表を単一パスで制約検査を実行するため、保全性検査は、 ALTER ステートメントの呼び出し前はオフにされ、ステートメントの実行後にオンにされます。

```
SET INTEGRITY FOR EMP_ACT OFF;
ALTER TABLE EMP_ACT ADD CHECK (EMSTDATE <= EMENDATE);
ALTER TABLE EMP_ACT ADD FOREIGN KEY (EMPNO) REFERENCES EMPLOYEE;
SET INTEGRITY FOR EMP_ACT IMMEDIATE CHECKED
```

例 8: 生成された列に保全性を設定します。

```
SET INTEGRITY FOR T1 IMMEDIATE CHECKED
  FORCE GENERATED
```

例 9: (LOAD INSERT を使用して) 別のソースから REFRESH IMMEDIATE マテリアライズ照会表 AST1 の基礎表 UT1 に追加してから、 UT1 のデータ保全性を (増分的に) 検査し、AST1 を (増分的に) リフレッシュします。このシナリオでは、UT1 の保全性検査および AST1 のリフレッシュは、システム側で増分処理を選択しているため、増分的になります。

```
LOAD FROM IMTFILE1.IXF OF IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF OF IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1 IMMEDIATE CHECKED;
REFRESH TABLE AST1;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『例外表』

関連サンプル:

- 『TbGenCol.java -- How to use generated columns (JDBC)』

SET PASSTHRU

SET PASSTHRU ステートメントは、データ・ソースのネイティブ SQL を、直接そのデータ・ソースに送信するセッションをオープンおよびクローズします。 このステートメントは、トランザクションの制御下にはありません。

呼び出し:

このステートメントは、対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの許可 ID が持つ特権には、以下の事柄を行う許可がなければなりません。

- データ・ソースにパススルーする。
- データ・ソースでのセキュリティーの制限を満たす。

構文:

説明:

server-name
パススルー・セッションをオープンするデータ・ソースを指定します。 *server-name* (サーバー名) は、カタログに記述されているデータ・ソースを指定していなければなりません。

RESET
パススルー・セッションをクローズします。

注:

- 以下の制約事項が Microsoft SQL Server、Sybase、および Oracle のデータ・ソースに適用されます。
  - パススルー・モードでは、ユーザー定義のトランザクションを Microsoft SQL Server や Sybase のデータ・ソースに対して使用できません。なぜなら、Microsoft SQL Server および Sybase では、ユーザー定義のトランザクションで指定できる SQL ステートメントが制約されるからです。パススルー・モードで処理される SQL ステートメントはグローバル・オプティマイザーによって構文解析されないので、ユーザーが指定した SQL ステートメントがユーザー定義のトランザクションで使用できるかどうかを検出できません。
  - Microsoft SQL Server および Sybase データ・ソース上では、COMPUTE 文節はサポートされていません。
  - Microsoft SQL Server、Oracle、および Sybase データ・ソース上では、 DDL ステートメントはトランザクション・セマンティクスの対象外です。操作が完了した時点で Microsoft SQL Server、Oracle または Sybase によって自動的にコミットされます。ロールバックが行われても、DDL はロールバックされません。

例:

例 1: データ・ソース BACKEND に対するパススルー・セッションを開始します。

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
```

例 2: PREPARE ステートメントを使ってパススルー・セッションを開始します。

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL PREPARE STMT FROM :PASS_THRU;
EXEC SQL EXECUTE STMT;
```

例 3: パススルー・セッション終了します。

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

例 4: PREPARE および EXECUTE ステートメントを使って、パススルー・セッションを終了します。

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL PREPARE STMT FROM :PASS_THRU_RESET;
EXEC SQL EXECUTE STMT;
```

例 5: データ・ソースに移動するセッションをオープンし、このデータ・ソースにある表のクラスター索引を作成し、それからパススルー・セッションを終了します。

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
EXEC SQL PREPARE STMT                 pass-through mode
  FROM "CREATE UNIQUE
        CLUSTERED INDEX TABLE_INDEX
        ON USER2.TABLE               table is not an
        WITH IGNORE DUP KEY";         alias
EXEC SQL EXECUTE STMT;
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

SET PATH

SET PATH ステートメントは、CURRENT PATH 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。

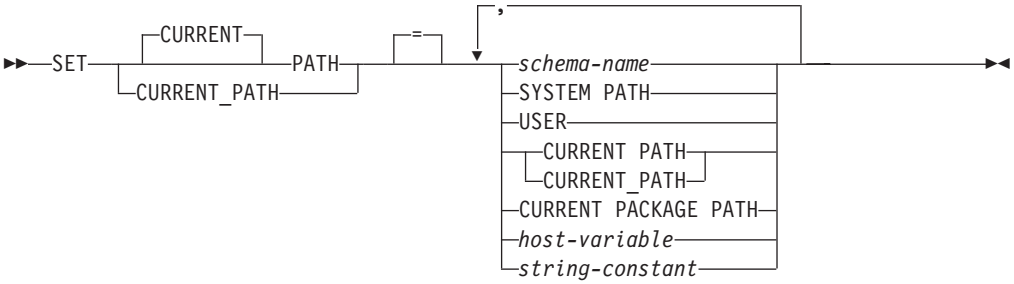
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

schema-name
これは、1 つの部分だけからなる名前で、アプリケーション・サーバーに存在するスキーマを指定します。そのスキーマの存否の検査は、パス設定時には行われません。 たとえば *schema-name* (スキーマ名) が間違っていると、エラーを捉えることができず、以降の SQL 操作に影響を及ぼします。

SYSTEM PATH
この値を指定すると、スキーマ名として *"SYSIBM"*、 *"SYSFUN"*、*"SYSPROC"* を指定したのと同じことになります。

USER
USER 特殊レジスターの値。

CURRENT PATH
このステートメントを実行する前の CURRENT PATH 特殊レジスターの値。

CURRENT PACKAGE PATH
CURRENT PACKAGE PATH 特殊レジスターの値。

host-variable
タイプ CHAR または VARCHAR の変数です。 *host-variable* の内容の長さは、30 バイトを超えてはなりません (SQLSTATE 42815)。 NULL に設定することはできません。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

host-variable の文字は左寄せされていなければなりません。 *host-variable* にスキーマ名 を指定する場合は、英大文字への変換はなされないので、すべての文字を大文字小文字の区別も含めて正確に指定する必要があります。

string-constant
30 バイトを超えない文字ストリング定数。

規則:

- 関数パスの中に 1 つのスキーマ名を 2 回以上指定することはできません (SQLSTATE 42732)。
- 指定できるスキーマの数は、CURRENT PATH 特殊レジスターの合計長によって限定されます。 特殊レジスターのストリングは、指定した各スキーマの名前から後続ブランクを除き、二重引用符で区切り、必要に応じてスキーマ名の中で使われている引用符を反復させ、スキーマ名をコンマで区切ったものになります。 結果のストリングの長さが 254 バイトを超えてはなりません (SQLSTATE 42907)。

注:

- 互換性
  - 以前のバージョンの DB2 との互換性:
    - CURRENT PATH の代わりに CURRENT FUNCTION PATH を指定できます。
- CURRENT PATH 特殊レジスターの初期値は、 ″SYSIBM″、 ″SYSFUN″、 ″SYSPROC″、 ″X″ です (X は USER 特殊レジスターの値)。
- SYSIBM スキーマを指定する必要はありません。それが SQL パスに含まれていない場合、暗黙のうちに最初のスキーマであると見なされます (この場合 CURRENT PATH 特殊レジスターには入れられません)。
- CURRENT PATH 特殊レジスターは、動的 SQL ステートメント内のユーザー定義データ・タイプ、プロシージャー、および関数を解決するために使用する SQL パスを指定します。動的 SQL ステートメント内のユーザー定義データ・タイプおよび関数の解決に使用する SQL パスは、 FUNCPATH BIND オプションによって指定されます。

例:

例 1: 以下のステートメントは、CURRENT PATH 特殊レジスターを設定します。

```
SET PATH = FERMAT, "McDrw #8", SYSIBM
```

例 2: 以下の例では、 CURRENT PATH 特殊レジスターの現行値を検索して CURPATH という名前のホスト変数に入れます。

```
EXEC SQL VALUES (CURRENT PATH) INTO :CURPATH;
```

例 1 での設定を使った場合、値は ″FERMAT″、″McDrw #8″、″SYSIBM″ になります。

SET SCHEMA

SET SCHEMA ステートメントは、CURRENT SCHEMA 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。DYNAMICRULES BIND オプションを使ってパッケージがバインドされている場合、このステートメントは、修飾されていないデータベース・オブジェクト参照に使用される修飾子に影響を与えません。

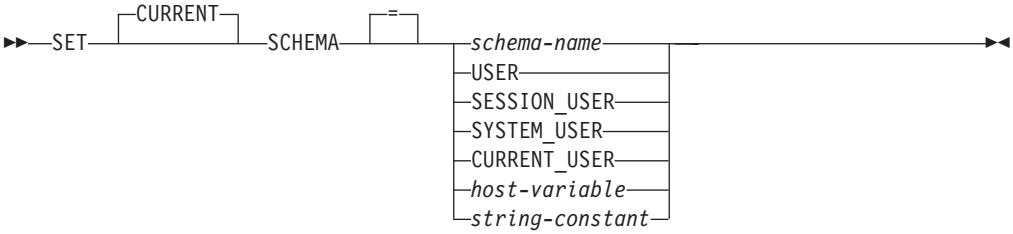
呼び出し:

このステートメントはアプリケーション・プログラムに組み込むことができ、また対話式に出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの実行には、特に権限は必要ありません。

構文:



説明:

schema-name
これは、1 つの部分だけからなる名前で、アプリケーション・サーバーに存在するスキーマを指定します。名前の長さは、30 バイトを超えてはなりません (SQLSTATE 42815)。そのスキーマの存否の検査は、スキーマ設定時には行われません。 *schema-name* (スキーマ名) が間違っていると、取り込むことができず、以降の SQL 操作に影響を及ぼします。

USER
USER 特殊レジスターの値。

SESSION_USER
SESSION_USER 特殊レジスターの値。

SYSTEM_USER
SYSTEM_USER 特殊レジスターの値。

CURRENT_USER
CURRENT_USER 特殊レジスターの値。

host-variable
タイプ CHAR または VARCHAR の変数です。 *host-variable* の内容の長さは、30 バイトを超えてはなりません (SQLSTATE 42815)。 NULL に設定することはできません。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 42815)。

host-variable の文字は左寄せされていなければなりません。 *host-variable* にスキーマ名 を指定する場合は、英大文字への変換はなされないので、すべての文字を大文字小文字の区別も含めて正確に指定する必要があります。

string-constant
最大長の 30 文字を超えない文字ストリング定数。

規則:

- 指定した値が *schema-name* の規則に適合しない場合、エラー (SQLSTATE 3F000) が発生します。
- CURRENT SCHEMA 特殊レジスターの値は、すべての動的 SQL ステートメント (データベース・オブジェクトへの非修飾参照がある CREATE SCHEMA ステートメントを除く) でスキーマ名として使用されます。
- QUALIFIER BIND オプションは、静的 SQL ステートメントで非修飾データベース・オブジェクト名の修飾子として使用するスキーマ名を指定します。

注:

- CURRENT SCHEMA 特殊レジスターの初期値は USER と同じです。
- CURRENT SCHEMA 特殊レジスターを設定しても、CURRENT PATH 特殊レジスターには影響しません。 したがって、CURRENT SCHEMA は SQL パスおよび関数に含まれず、プロシージャーおよびユーザー定義タイプの解決ではこれらのオブジェクトを見つけることができない場合があります。 SQL パスに現行スキーマ値を含めるには、SET SCHEMA ステートメントを発行するときに、 SET SCHEMA ステートメントからスキーマ名を含む SET PATH ステートメントも発行してください。
- CURRENT SQLID は CURRENT SCHEMA の同義語として受け入れられ、 SET CURRENT SQLID ステートメントは SET CURRENT SCHEMA ステートメントと同じ影響を及ぼします。 ステートメント許可変更など、他の影響はありません。

例:

例 1: 以下のステートメントは、CURRENT SCHEMA 特殊レジスターを設定します。

```
SET SCHEMA RICK
```

例 2: 以下の例では、 CURRENT SCHEMA 特殊レジスターの現行値を検索して CURSCHEMA という名前のホスト変数に入れます。

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

値は、前の例で設定された RICK になります。

SET SERVER OPTION

SET SERVER OPTION ステートメントは、ユーザーまたはアプリケーションがフェデレーテッド・データベースに接続中に有効となる、サーバー・オプションの設定値を指定します。 接続が終了すると、このサーバー・オプションの以前の設定値が復元されます。このステートメントは、トランザクションの制御下にありません。

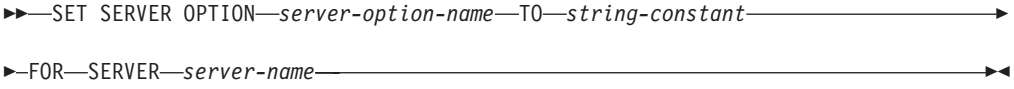
呼び出し:

このステートメントは、対話式に発行することができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

このステートメントの許可 ID には、フェデレーテッド・データベースに対する SYSADM 権限または DBADM 権限がなければなりません。

構文:



説明:

server-option-name
設定するサーバー・オプションを指名します。

TO *string-constant*
server-option-name の設定を、文字ストリング定数として指定します。

SERVER *server-name*
server-option-name が適用されるデータ・ソースを指定します。これは、カタログに記述されているサーバーでなければなりません。

注:

- サーバー・オプション名は、大文字または小文字で入力することができます。
- ユーザーまたはアプリケーションがフェデレーテッド・データベースに接続する際、 1 つまたは複数の SET SERVER OPTION ステートメントをサブミットすることができます。 このステートメント (複数可) は、接続が確立した後、最初に処理される作業単位の初めに指定する必要があります。
- SYSCAT.SERVEROPTIONS は SET SERVER OPTION ステートメントに基づいては更新されません。この変更内容は現行接続だけに影響を与えるからです。

例:

例 1: DJDB というフェデレーテッド・データベースに、 ORASERV という Oracle データ・ソースを定義します。 ORASERV は、プランのヒントを使用できないように構成されます。 しかし、DBA は新しいアプリケーションを試験的に実行するため、プランのヒントを使用できるようにすることを希望しています。実行を終了すると、プランのヒントは再度使用不可になります。

```
CONNECT TO DJDB;
strcpy(stmt,"set server option plan_hints to 'Y' for server oraserv");
EXEC SQL EXECUTE IMMEDIATE :stmt;
strcpy(stmt,"select c1 from ora_t1 where c1 > 100"); /*Generate plan hints*/
```

```
EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :hv;
```

例 2: すべての Oracle 8 データ・ソースで、サーバー・オプション PASSWORD を 'Y' (データ・ソースでパスワードを妥当性検査する) に設定しました。しかし、特定の Oracle 8 データ・ソース (フェデレーテッド・データベース DJDB に ORA8A と定義されているデータ・ソース) にアクセスするために、アプリケーションがフェデレーテッド・データベースに接続するセッションの場合、パスワードを妥当性検査する必要はありません。

```
CONNECT TO DJDB;
strcpy(stmt,"set server option password to 'N' for server ora8a");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL EXECUTE STMT_NAME FROM :stmt;
strcpy(stmt,"select max(c1) from ora8a_t1");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR STMT_NAME;
EXEC SQL OPEN c1; /*Does not validate password at ora8a*/
EXEC SQL FETCH c1 INTO :hv;
```

関連資料:

- フェデレーテッド・システム・ガイド の『フェデレーテッド・システムのサーバー・オプション』

SET SESSION AUTHORIZATION

SET SESSION AUTHORIZATION ステートメントは、SESSION_USER 特殊レジスターの値を変更します。このステートメントは、トランザクションの制御下にはありません。このステートメントは、同一の接続に複数の異なる許可 ID を見こむ単一ユーザーをサポートすることを目的としているため、複数の異なるユーザーが同じ接続を再利用するシナリオ (一般に、接続プールと呼ばれる) には使用しないでください。

呼び出し:

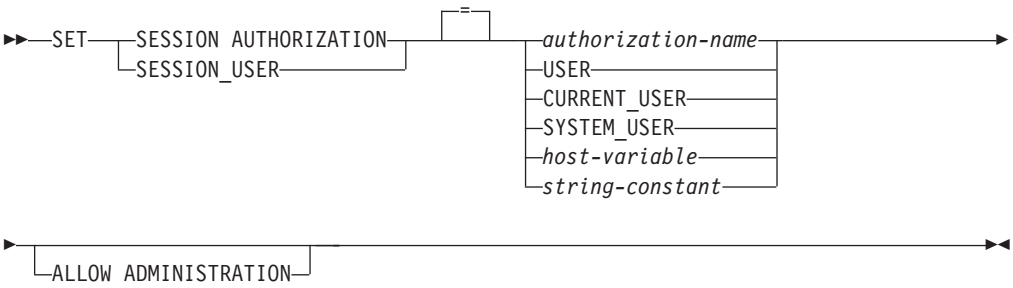
このステートメントはアプリケーション・プログラムに組み込むことができ、また対話式に出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

接続を確立する際に使用された許可 ID (SYSTEM_USER 特殊レジスターに反映されている値) を別の許可 ID に変更するためには、ステートメントの許可 ID に含まれている特権に、以下のうち少なくとも 1 つが含まれていなければなりません (SQLSTATE 28000)。

- SYSADM または DBADM 権限

構文:



説明:

authorization-name
SESSION_USER 特殊レジスターに新しい値として使用する許可 ID。

USER
USER 特殊レジスターの値。

CURRENT_USER
CURRENT USER 特殊レジスターの値。

SYSTEM_USER
SYSTEM_USER 特殊レジスターの値。

host-variable
タイプ CHAR または VARCHAR の変数です。 *host-variable* の内容の長さは、30 バイトを超えてはなりません (SQLSTATE 28000)。 NULL に設定することはできません。 *host-variable* が標識変数を伴っている場合、その標識変数の値は NULL 値以外でなければなりません (SQLSTATE 28000)。

host-variable の文字は左寄せされていなければなりません。ホスト変数に *authorization-name* を指定する場合は、大文字への変換が行われないため、すべての文字の大文字小文字を、示されているとおりに正確に指定する必要があります。

string-constant
最大長の 30 文字を超えない文字ストリング定数。

ALLOW ADMINISTRATION
同じ作業単位内でこのステートメントよりも前に SQL スキーマ・ステートメントを指定できることを示します。

規則:

- SESSION_USER 特殊レジスターで指定される値は、タイプ USER の許可 ID の規則に従わなければなりません (SQLSTATE 42602)。
- OWNER BIND オプションは、静的 SQL ステートメントに使用される許可 ID を指定します。
- このステートメントは、開かれている WITH HOLD カーソルがない、新しい作業単位の最初のステートメント (SET 特殊レジスター・ステートメントを除く) として発行します (SQLSTATE 25001)。この制限には、SET 特殊レジスター・ステートメント以外のステートメントに対するすべての PREPARE 要求も含まれます。
- SESSION_USER 特殊レジスターの値は、 DYNAMICRULES(RUN) BIND オプションでバインドされたパッケージのすべての動的 SQL ステートメントに対する許可 ID として使用されます (これには、パッケージがルーチンで使用されていない場合の INVOKERUN および DEFINERUN も含まれます)。パッケージが DYNAMICRULES オプションに基づいて所有者、起動側、または定義元の許可に使用される場合、このステートメントには、パッケージ内から発行される動的 SQL ステートメントに対して効果がありません。

注:

- 新しい接続における SESSION_USER 特殊レジスターの初期値は、SYSTEM_USER 特殊レジスターの値と同じです。
- このステートメントで指定されているセッション許可 ID のグループ情報は、ステートメントの実行時に獲得されます。
- SESSION_USER 特殊レジスターの設定は、 CURRENT SCHEMA または CURRENT PATH 特殊レジスターには影響を与えません。
- SESSION_USER 特殊レジスターの設定中にエラーが発生した場合、レジスターは直前の値に復帰します。
- このステートメントを使用して、複数の異なるユーザーによる同一接続の再利用を許可しないでください。 SESSION_USER 特殊レジスター値の変更を可能にする許可は、各ユーザーに継承されます。このステートメントは、特権のチェックを SYSTEM_USER の値に依存しており、最初の接続許可 ID は SET SESSION AUTHORIZATION ステートメントによって変更されません。さらに、このステートメントでは、接続の再利用に影響を与える以下の振る舞いに対処できません。
  - CONNECT 特権には、新しい許可 ID のチェックが行われません。

  - 更新可能特殊レジスターの内容がリセットされません。特に、ENCRYPTION PASSWORD 特殊レジスターの内容が変更されず、新しい許可 ID で暗号化または暗号化解除できてしまいます。
  - 宣言済み一時表の内容が影響を受けず、新しい許可 ID からアクセスできてしまいます。
  - リモート・サーバーへの既存のリンクがリセットされません。
- ALLOW ADMINISTRATION 文節が指定される場合、以下のタイプのステートメントまたは操作を SET SESSION AUTHORIZATION ステートメントよりも先行させることができます。
  - データ定義言語 (DDL)。これにはセーブポイントの定義やグローバル一時表の宣言が含まれますが、SET INTEGRITY は含まれません。
  - GRANT および REVOKE ステートメント
  - LOCK TABLE ステートメント
  - COMMIT および ROLLBACK ステートメント
  - 特殊レジスターの SET

例:

例 1: 以下のステートメントは、SESSION_USER 特殊レジスターを設定します。

```
SET SESSION_USER = RAJIV
```

例 2: セッション許可 ID (SESSION_USER 特殊レジスター) を、ステートメント発行元の接続を確立する際に使用されたシステム許可 ID の値にします。

```
SET SESSION AUTHORIZATION SYSTEM_USER
```

関連資料:

- *コマンド・リファレンス* の『BIND コマンド』

SET 変数

SET 変数ステートメントは、ローカル変数、出力パラメーター、または新しい遷移変数に値を割り当てます。これは、トランザクションによる制御下にあります。

呼び出し:

このステートメントは、動的コンパウンド・ステートメント、トリガー、SQL 関数、SQL メソッド、または SQL プロシージャーのいずれかで SQL ステートメントとしてのみ使用できます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

許可:

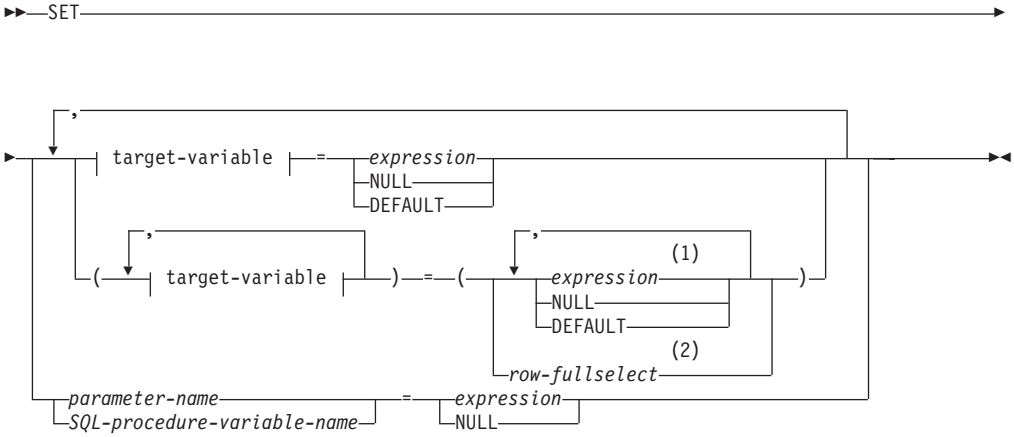
遷移変数を参照するには、トリガー作成者の許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 割り当ての左辺で参照されている列に対する UPDATE 特権、および右辺で参照されているすべての列に対する SELECT 特権。
- 表 (トリガーのサブジェクト表) での CONTROL 特権
- SYSADM または DBADM 権限

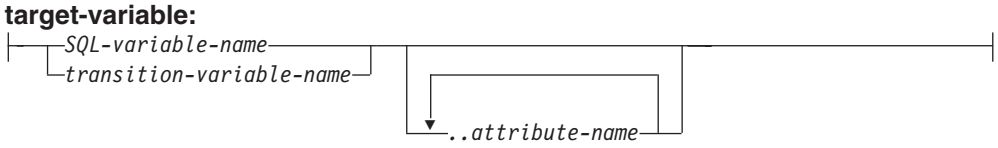
割り当ての右辺として *row-fullselect* を持っているこのステートメントを実行するには、トリガー定義者または動的コンパウンド・ステートメント所有者のいずれかの許可 ID によって保持されている特権には、参照されている表またはビューそれぞれについて、少なくとも以下のいずれかも含まれていなければなりません。

- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

構文:



target-variable:



注:

1 式、NULL、および DEFAULT の数は、 *target-variable* の指定数と一致している必要があります。

2 選択リスト内の列の数は、 *target-variable* の指定数と一致している必要があります。

説明:

target-variable
割り当てのターゲット変数を識別します。同じ変数を表す *target-variable* を複数指定することはできません (SQLSTATE 42701)。

SQL-variable-name
割り当てターゲットである SQL 変数を識別します。 SQL 変数は、使用する前に宣言しておかなければなりません。 SQL 変数は、動的コンパウンド・ステートメントとプロシージャー・コンパウンド・ステートメントの両方に定義することができます。

transition-variable-name
遷移行で更新する列を識別します。 *transition-variable-name* は、新しい値を識別する相関名によってオプションで修飾されている、トリガーのサブジェクト表にある列を識別していなければなりません (SQLSTATE 42703)。

..attribute name
設定されている構造化タイプの属性 (*属性割り当て* という) を指定します。指定する *SQL-variable-name* または *transition-variable-name* は、ユーザー定義の構造化タイプで定義されていなければなりません (SQLSTATE 428DP)。 *attribute-name* は、構造化タイプの属性でなければなりません (SQLSTATE 42703)。 *..attribute name* 文節が関係しない割り当ては、従来型割り当てと呼ばれます。

parameter-name
割り当てのターゲットとなるパラメーターを識別します。そのパラメーターは、CREATE PROCEDURE ステートメントの *parameter-declaration* で指定しなければならず、さらに OUT または INOUT パラメーターとして定義しなければなりません。

SQL-procedure-variable-name
SQL プロシージャー内の割り当てターゲットである SQL 変数を識別します。SQL 変数は、使用する前に宣言しておかなければなりません。 SQL 変数はプロシージャー・コンパウンド・ステートメントに定義することができます。

expression
割り当てのターゲットの新しい値を指定します。この expression (式) として、『式』 で説明されているタイプの式はいずれも使用することができます。スカラー fullselect で使用される場合を除き、列関数を組み込むことはできません (SQLSTATE 42903)。 CREATE TRIGGER ステートメントのコンテキストにおいて、 *expression* は OLD および NEW 遷移変数への参照を含むことができます。遷移変数は、*correlation-name* で修飾されていなければなりません (SQLSTATE 42702)。

NULL
NULL 値を指定します。属性のデータ・タイプに特定してキャストされていない限り、NULL を属性割り当ての値にすることはできません (SQLSTATE 429B9)。

DEFAULT
デフォルト値が使用されることを指定します。

target-variable が列であれば、挿入される値は、どのように列が表に定義されているかによって異なります。

- 列が WITH DEFAULT 文節で定義されている場合、値は、その列に定義されたデフォルトに設定されます (『ALTER TABLE』 の *default-clause* を参照してください)。
- 列が IDENTITY 文節で定義されている場合、値はデータベース・マネージャーによって生成されます。
- 列が WITH DEFAULT 文節、IDENTITY 文節、または NOT NULL 文節のいずれも指定せずに定義されている場合、値は NULL になります。
- 列が NOT NULL 文節で定義されている場合で、次のいずれかに該当する場合には、列に DEFAULT キーワードを指定できません (SQLSTATE 23502)。
  - IDENTITY 文節が使用されていない
  - WITH DEFAULT 文節が使用されていない
  - DEFAULT NULL が使用されている

target-variable が SQL 変数であれば、挿入される値は、変数宣言に指定または暗黙指定されているデフォルトになります。

row-fullselect
割り当てに指定されているターゲット変数の数に対応する列数とともに、単一行を返す全選択です。値は、対応するターゲット変数それぞれに割り当てられます。 row-fullselect の結果が行なしであれば、NULL 値が割り当てられます。CREATE TRIGGER ステートメントのコンテキストにおいて、 *row-fullselect* には OLD および NEW 遷移変数への参照を含めることができます。その際、どの遷移変数が使用されるかを指定するために *correlation-name* で修飾する必要があります (SQLSTATE 42702)。結果に行が複数ある場合、エラーが返されます (SQLSTATE 21000)。

規則:

- 式から割り当てる値、NULL、DEFAULT、または *row-fullselect* の数は、割り当てに指定されている *target-variables* の数に一致していなければなりません (SQLSTATE 42802)。
- SET 変数ステートメントは、1 つのステートメントに SQL 変数と遷移変数を割り当てることができません (SQLSTATE 42997)。
- 特定の割り当て規則に従って、値がターゲット変数に割り当てられます。
- SQL プロシージャーの割り当てステートメントは、 SQL 割り当て規則に準拠していなければなりません。ストリング割り当てでは、ストレージ割り当て規則が使用されます。

- 特殊レジスターの名前 (PATH など) と一致した ID で変数が宣言されている場合には、意図せずに特殊レジスターに割り当てられてしまわないように、その変数を引用符で区切ってください (例えば、 PATH という変数が整数として宣言されている場合は SET "PATH" = 1;)。

注:

- 複数の割り当てが組み込まれている場合、それぞれの *expression* および *row-fullselect* は、割り当てが実行される前に評価されます。そのため、式または行の全選択でのターゲット変数への参照は常に、単一 SET ステートメントでの割り当ての前のターゲット変数の値となります。
- 特殊タイプとして定義された ID 列が更新された場合は、まずすべての計算がソース・タイプで行われます。その結果は、値が列に実際に割り当てられる前に、ソース・タイプから定義された特殊タイプにキャストされます。 (計算に先立って、元の値がソース・タイプにキャストされることはありません。)
- ID 列に対する SET ステートメントで DB2 によって値が生成されるようにするには、 DEFAULT キーワードを使用します。

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  この例では、NEW.EMPNO が ID 列として定義されており、この列の更新に使用される値は DB2 によって生成されます。
- ID 列に生成されるシーケンス値の使用に関する詳細、および ID 列で値が最大値を超えた場合の詳細は、『INSERT』 を参照してください。

例:

例 1: 現在トリガー・アクションが実行されている行の給与の列を 50000 に設定します。

```
SET NEW_VAR.SALARY = 50000;
```

または、

```
SET (NEW_VAR.SALARY) = (50000);
```

例 2: 現在トリガー・アクションが実行されている行の給与と歩合の列を、それぞれ 50000 および 8000 に設定します。

```
SET NEW_VAR.SALARY = 50000, NEW_VAR.COMM = 8000;
```

または、

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (50000, 8000);
```

例 3: 現在トリガー・アクションが実行されている行の給与と歩合の列を、更新される行に関連した部門の従業員の平均給与および平均歩合にそれぞれ設定します。

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM)
   = (SELECT AVG(SALARY), AVG(COMM)
        FROM EMPLOYEE E
        WHERE E.WORKDEPT = NEW_VAR.WORKDEPT);
```

例 4: 現在トリガー・アクションが実行されている行の給与と歩合の列を、それぞれ 10000、および元の (つまり SET ステートメントの実行前の) 給与値に設定します。

```
SET NEW_VAR.SALARY = 10000, NEW_VAR.COMM = NEW_VAR.SALARY;
```

または、

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (10000, NEW_VAR.SALARY);
```

例 5: SQL 変数 p_salary を 10 % ずつ増加させます。

```
SET p_salary = p_salary + (p_salary * .10)
```

例 6: SQL 変数 p_salary を NULL 値に設定します。

```
SET p_salary = NULL
```

関連資料:

- *SQL リファレンス 第 1 巻* の『式』
- 45 ページの『ALTER TABLE』
- 615 ページの『INSERT』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

SIGNAL

SIGNAL ステートメントは、エラーまたは警告条件を通知するために使用されます。これを使用すると、指定した SQLSTATE とオプションのメッセージ・テキストが、エラーまたは警告とともに戻されます。

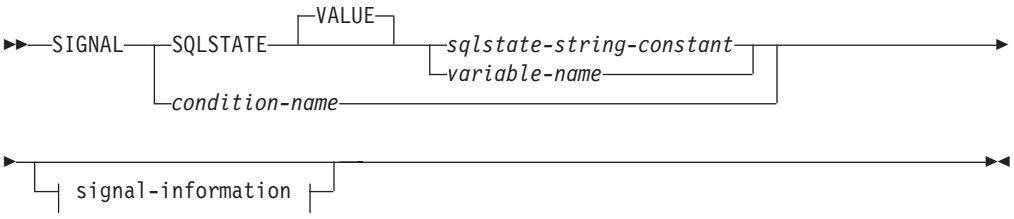
呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

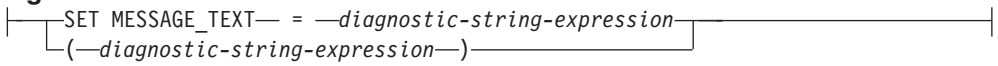
許可:

必要ありません。

構文:



signal-information:



説明:

SQLSTATE VALUE *sqlstate-string-constant*

指定されたストリング定数が SQLSTATE を表します。この定数は、正確に 5 文字の文字ストリング定数でなければならず、 SQLSTATE の規則に従っていなければなりません。

- 各文字は、数字 ('0'〜'9')、またはアクセントのない大文字の英字 ('A'〜'Z') でなければなりません。
- SQLSTATE クラス (最初の 2 文字) は '00' にはできません。これは正常な完了を示します。

動的コンパウンド・ステートメント、トリガー、SQL 関数、または SQL メソッドのコンテキストでは、以下の規則も適用されます。

- SQLSTATE クラス (最初の 2 文字) はエラー・クラスではないため、 '01' または '02' にはできません。
- SQLSTATE クラスが数字 '0'〜'6' または文字 'A'〜'H' で始まっている場合、サブクラス (最後の 3 文字) は 'I'〜'Z' の範囲の文字で始まっていなければなりません。
- SQLSTATE クラスが数字 '7'、'8'、'9'、または文字 'I'〜'Z' で始まっている場合、サブクラスは '0'〜'9' または 'A'〜'Z' のいずれかになります。

SQLSTATE がこれらの規則に従っていない場合、エラーが戻されます (SQLSTATE 428B3)。

SQLSTATE VALUE *variable-name*
指定する変数名はタイプ CHAR(5) でなければなりません。ステートメント実行時のこの変数の値が、*sqlstate-string-constant* に記述されているものと同じ規則に準拠している必要があります。 SQLSTATE がこれらの規則に従っていない場合、エラーが戻されます (SQLSTATE 428B3)。

condition-name
条件の名前を指定します。条件名はプロシージャー内でユニークでなければならず、宣言されたコンパウンド・ステートメント内でのみ参照が可能です。

SET MESSAGE_TEXT =
エラーまたは警告を記述するストリングを指定します。ストリングは SQLCA の SQLERRMC フィールドに返されます。実際のストリングが 70 バイトを超えている場合は、警告なしで切り捨てられます。

diagnostic-string-expression
エラー条件を記述する最高 70 バイトの文字ストリングを戻すタイプ CHAR または VARCHAR の式。ストリングは 70 バイトを超えると切り捨てられます。

(*diagnostic-string-expression***)**
エラー条件を記述する最高 70 バイトの文字ストリングを戻すタイプ CHAR または VARCHAR の式。ストリングは 70 バイトを超えると切り捨てられます。このオプションは、以前のバージョンの DB2 との互換性のために、 CREATE TRIGGER ステートメントの有効範囲内でのみ提供されます。通常は使用しないでください。

注:

- SIGNAL ステートメントが発行される場合、戻される SQLCODE は以下の SQLSTATE に基づいています。
  - 指定した SQLSTATE クラスが '01' か '02' のいずれかである場合、警告か、見つからないことを示す条件が戻され、SQLCODE は +438 に設定されます。
  - それ以外の場合、例外条件が戻され、SQLCODE は -438 に設定されます。

  SQLCA の他のフィールドは、以下のように設定されます。
  - sqlerrd フィールドはゼロに設定されます
  - sqlwarn フィールドはブランクに設定されます
  - sqlerrmc は MESSAGE_TEXT の先頭の 70 バイトに設定されます
  - sqlerrml は sqlerrmc の長さか、 SET MESSAGE_TEXT 文節が指定されていない場合にはゼロに設定されます
  - sqlerrp は ROUTINE に設定されます
- SQLSTATE 値は 2 文字のクラス・コード値からなり、その後に 3 文字のサブクラス・コード値が続きます。クラス・コード値は実行の成功状態または不成功状態のクラスを表します。

  有効な SQLSTATE 値はいずれも SIGNAL ステートメントで使用できます。ただし、プログラマーがアプリケーション用に予約された範囲に基づいて新しい SQLSTATE を定義することをお勧めします。これにより、将来のリリースでデー

タベース・マネージャーによって定義される可能性のある SQLSTATE 値を誤って使用してしまうのを避けることができます。

– 文字 '7'〜'9'、または 'I'〜'Z' で始まる SQLSTATE クラスは定義可能です。これらのクラス内では、サブクラスを定義することができます。
– 文字 '0'〜'6'、または 'A'〜'H' で始まる SQLSTATE クラスはデータベース・マネージャー用に予約されています。これらのクラス内では、文字 '0'〜'H' で始まるサブクラスはデータベース・マネージャー用に予約されています。文字 'I'〜'Z' で始まるサブクラスは定義可能です。

例:

顧客番号がアプリケーションに認識されていないときにアプリケーション・エラーを通知する、オーダー・システムの SQL プロシージャーです。ORDERS 表には CUSTOMER 表に対する外部キーが含まれており、オーダーを入れるためには CUSTNO が存在していることが必要になります。

```
CREATE PROCEDURE SUBMIT_ORDER
   (IN ONUM INTEGER, IN CNUM INTEGER,
    IN PNUM INTEGER, IN QNUM INTEGER)
    SPECIFIC SUBMIT_ORDER
    MODIFIES SQL DATA
   LANGUAGE SQL
   BEGIN
     DECLARE EXIT HANDLER FOR SQLSTATE VALUE '23503'
       SIGNAL SQLSTATE '75002'
         SET MESSAGE_TEXT = 'Customer number is not known';
     INSERT INTO ORDERS (ORDERNO, CUSTNO, PARTNO, QUANTITY)
       VALUES (ONUM, CNUM, PNUM, QNUM);
   END
```

UPDATE

UPDATE ステートメントは、表、ビュー、またはニックネームの行で、あるいは指定された全選択の基礎になる表、ニックネーム、またはビューの行で、指定された列の値を更新します。 ビューに対する更新操作用に INSTEAD OF トリガーが定義されていない場合、ビューの行を更新することは、そのビューの基本表の行を更新することでもあります。このようなトリガーが定義されている場合は、トリガーが代わりに実行されます。ニックネームを使用して行を更新することは、そのニックネームが参照するデータ・ソース・オブジェクト中の行を更新することでもあります。

このステートメントの形式は以下のとおりです。

- *検索条件付き* UPDATE 形式は、 1 つまたは複数の行 (任意指定の検索条件によって決まる) を更新する場合に使用されます。
- *位置指定* UPDATE 形式は、 1 行 (カーソルの現在位置によって決まる) だけを更新する場合に使用されます。

呼び出し:

UPDATE ステートメントはアプリケーション・プログラムに組み込むことができ、また動的 SQL ステートメントを使用して出すことができます。このステートメントは、動的に作成できる実行可能ステートメントです。

許可:

ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかが含まれていなければなりません。

- 更新する行を含む表、ビュー、またはニックネームに対する UPDATE 特権
- 更新するそれぞれの列に対する UPDATE 特権
- 更新する行を含む表、ビュー、またはニックネームに対する CONTROL 特権
- SYSADM または DBADM 権限
- 割り当て式に *row-fullselect* (行全選択) を含める場合には、参照される表、ビュー、またはニックネームのそれぞれに対して、少なくとも次のいずれかが必要です。
  - SELECT 特権
  - CONTROL 特権
  - SYSADM または DBADM 権限

副照会によって参照される表、ビュー、またはニックネームのそれぞれに対して、このステートメントの許可 ID が持つ特権には以下の少なくとも 1 つが含まれている必要があります。

- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

ステートメントの処理に使用されるパッケージが SQL92 規則を使用してプリコンパイルされており (オプション LANGLEVEL の値を SQL92E または MIA と指定)、 UPDATE ステートメントの検索条件付き形式で *assignment-clause* の右側また

は *search-condition* のいずれかの個所に表、ビュー、またはニックネームの列への参照が含まれている場合、このステートメント許可 ID が持つ特権には、さらに以下の少なくとも 1 つが含まれている必要があります。

- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

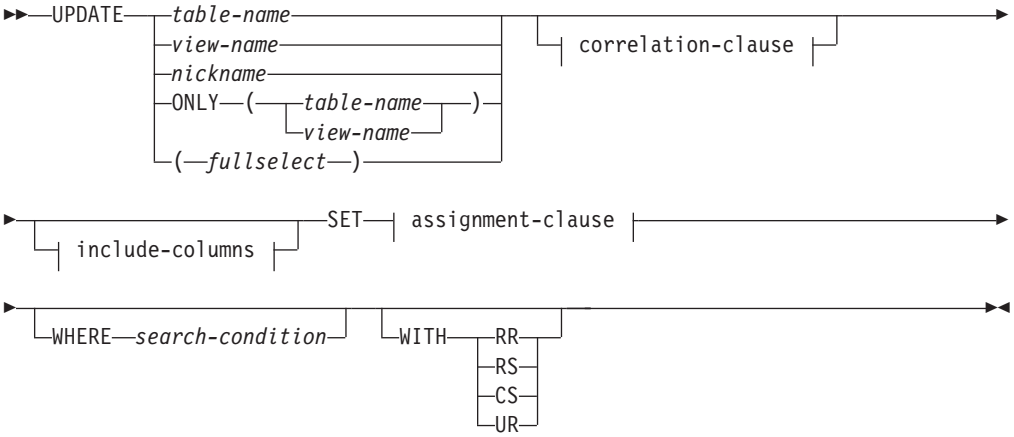
指定した表またはビューが ONLY キーワードの後にくる場合、ステートメントの許可 ID が持つ特権にも、指定した表またはビューの副表またはサブビューごとに SELECT 特権が含まれている必要があります。

静的 UPDATE ステートメントの場合、GROUP 特権はチェックされません。

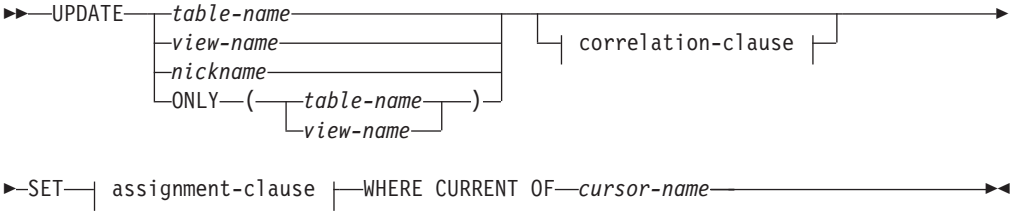
更新操作の対象がニックネームの場合は、データ・ソースでステートメントが実行されないうちは、そのデータ・ソース上のオブジェクトに対する特権は考慮されません。この時点で、データ・ソースに接続するために使用される許可 ID は、データ・ソースのオブジェクトに対して操作を行うのに必要な特権を持っている必要があります。ステートメントの許可 ID は、データ・ソースの別の許可 ID へマップできます。

構文:

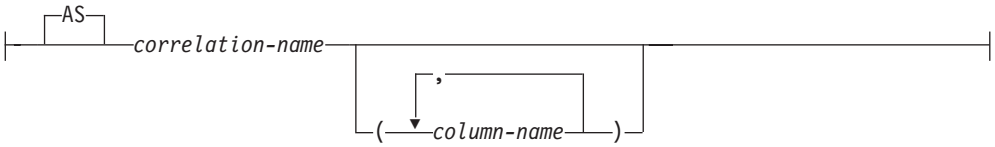
検索更新:



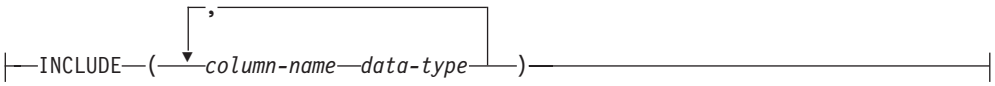
位置指定更新:



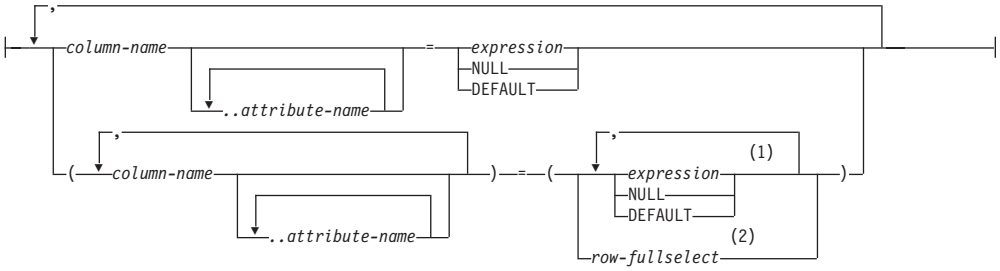
correlation-clause:



include-columns:



assignment-clause:



注:

1 式、NULL、および DEFAULT の数は、列名の数と一致している必要があります。

2 選択リストの列の数は、列名の数と一致している必要があります。

説明:

table-name、*view-name*、 *nickname*、**または** *(fullselect)*

更新操作の対象のオブジェクトを指定します。 この名前は、カタログに記述されている表、ビュー、またはニックネームを指定する名前でなければならず、カタログ表、カタログ表のビュー (更新可能な SYSSTAT ビューを除く)、システム保守のマテリアライズ照会表、または更新操作用に INSTEAD OF トリガーが定義されていない読み取り専用のビューを指定することはできません。

table-name が型付き表である場合は、このステートメントを使用して、その表またはそれに関係する副表の行を更新できます。 WHERE 文節で設定または参照できるのは、指定した表の列だけです。 位置指定 UPDATE の場合は、FROM 文節に指定されているのと同じ表、ビュー、またはニックネームを、関連するカーソルにも ONLY を使用せずに指定しなければなりません。

更新操作のオブジェクトが全選択である場合、全選択は、CREATE VIEW ステートメントの説明の『注』にある、『更新可能ビュー』の項目で定義されているように、更新可能になっている必要があります。

ONLY (*table-name***)**

型付き表の場合に適用できます。 ONLY キーワードは、指定した表のデータだけにステートメントを適用し、その表に関係する副表の行は更新できないことを指定します。 位置指定 UPDATE の場合は、FROM 文節に指定されているのと同じ表を、関連するカーソルにも ONLY キーワードを使用して指定しなければなりません。 *table-name* が型付き表でない場合は、このステートメントに ONLY キーワードを使用しても効果はありません。

ONLY (*view-name*)
型付きビューの場合に適用できます。 ONLY キーワードは、指定されたビューのデータだけにステートメントを適用し、その表に関係するサブビューの行は更新できないことを指定します。 位置指定 UPDATE の場合は、FROM 文節に指定されているのと同じビューを、関連するカーソルにも ONLY を指定して指定しなければなりません。 *view-name* が型付きビューでない場合は、このステートメントに ONLY キーワードを使用しても効果はありません。

correlation-clause
search-condition や *assignment-clause* で、表、ビュー、ニックネーム、または全選択の指定に使用できます。 *correlation-clause* についての説明は、『副選択』の説明にある『table-reference』を参照してください。

include-columns
全選択の FROM 文節にネストされているとき、 *table-name* や *view-name* などの列と一緒に UPDATE ステートメントの中間結果表に組み込まれている列セットを指定します。 *include-columns* は、 *table-name* や *view-name* で指定されている列のリストの最後に付加されます。

INCLUDE
UPDATE ステートメントの中間結果表に組み込まれる列のリストを指定します。

column-name
UPDATE ステートメントの中間結果表の列を指定します。名前は、他の組み込み列や、 *table-name* または *view-name* の列と同じ名前であってはなりません (SQLSTATE 42711)。

data-type
組み込み列のデータ・タイプを指定します。データ・タイプは、CREATE TABLE ステートメントでサポートされているものでなければなりません。

SET
この跡に、列名への値の割り当てを指定します。

assignment-clause

column-name
更新したい列を指定します。 *column-name* は、指定された表、ビュー、またはニックネームの更新可能列か、INCLUDE 列を識別しなければなりません。 型付き表のオブジェクト ID 列は更新できません (SQLSTATE 428DZ)。 *..attribute-name* を付けて指定しない限り、 1 つの列を 2 回以上指定することはできません (SQLSTATE 42701)。

INCLUDE 列を指定した場合、列名は修飾できません。

位置指定 UPDATE の場合 :

- カーソルの *select-statement* に *update-clause* を指定した場合、この *assignment-clause* の各列名は、その *update-clause* にも指定されていなければなりません。
- カーソルの *select-statement* に *update-clause* を指定せず、アプリケーションのプリコンパイル時に LANGLEVEL MIA または SQL92E が指定されていた場合には、更新可能な列の名前はいずれも指定することができます。

- カーソルの *select-statement* に *update-clause* 文節を指定せず、アプリケーションのプリコンパイル時に LANGLEVEL SAA1 を明示的にまたはデフォルト値として指定していた場合には、列は更新できません。

..attribute-name
設定されている構造タイプの属性 (*属性割り当て* という) を指定します。指定される *column-name* は、ユーザー定義構造タイプで定義されているものでなければなりません (SQLSTATE 428DP)。 attribute-name は、*column-name* の構造タイプの属性でなければなりません (SQLSTATE 42703)。 *..attribute-name* 文節と関係のない割り当ては、*通常の割り当て* と見なされます。

expression
列の新しい値を指定します。この expression (式) として、『式』 で説明されているタイプの式はいずれも使用することができます。スカラー fullselect で使用される場合を除き、列関数を組み込むことはできません (SQLSTATE 42903)。

expression には、 UPDATE ステートメントのターゲット表の列への参照を含めることができます。更新対象の行ごとに、式の中のそのような列の値は、行の更新前のその行の列の値になります。

式に、INCLUDE 列への参照を含めることはできません。

NULL
NULL 値を指定します。NULL 可能列にのみ指定することができます (SQLSTATE 23502)。 NULL が特に属性のデータ・タイプにキャストされたのでない限り、属性割り当ての値として NULL を使用することはできません (SQLSTATE 429B9)。

DEFAULT
対応する列の表における定義方法に基づくデフォルト値を使用することを指定します。 挿入される値は、その列の定義方法によって異なります。

- 式に基づいて生成された列として列が定義されている場合は、その式に基づいた列の値がシステムによって生成されます。
- 列が IDENTITY 文節で定義されている場合、値はデータベース・マネージャーによって生成されます。
- 列が WITH DEFAULT 文節で定義されている場合、値は、その列に定義されたデフォルトに設定されます (『ALTER TABLE』 の *default-clause* を参照してください)。
- 列の定義に NOT NULL 文節が使用されたが GENERATED 文節が使用されなかった場合、また WITH DEFAULT 文節が使用されていない場合や DEFAULT NULL が使用されている場合は、その列に DEFAULT キーワードを指定することはできません (SQLSTATE 23502)。

生成された列が GENERATED ALWAYS 文節で定義されている場合は、DEFAULT 以外の値を挿入することはできません (SQLSTATE 428C9)。

属性割り当てでは、DEFAULT キーワードを値として使用することはできません (SQLSTATE 429B9)。

データ・ソースが DEFAULT 構文をサポートしていない場合に、割り当てで DEFAULT キーワードを値として使用してニックネームに対する更新を行うことはできません 。

row-fullselect
割り当て式に指定した*列名* の数と同じ数の列を含む 1 つの行を戻す全選択です。 値は、それぞれ対応する*列名* に割り当てられます。 この *row-fullselect* の結果の行がない場合は、NULL 値が割り当てられます。

row-fullselect (行全選択) には、 UPDATE ステートメントのターゲット表の列に対する参照を含めることができます。 更新対象の行ごとに、式の中のそのような列の値は、行の更新前のその行の列の値になります。結果に行が複数ある場合、エラーが返されます (SQLSTATE 21000)。

WHERE
この後に、更新したい行を識別する条件を指定します。 この文節は、省略することも、検索条件を指定することも、またはカーソル名を指定することもできます。 この文節を省略すると、表、ビュー、またはニックネームのすべての行が更新されます。

search-condition
副照会以外の検索条件の各*列名* は、表、ビュー、またはニックネームの列を指定していなければなりません。 検索条件に、同じ表が UPDATE と副照会の両方の基本オブジェクトである副照会が含まれている場合、行が更新される前に、その副照会が完全に評価されます。

検索条件は、表、ビュー、またはニックネームの各行に適用され、検索条件の結果が「真」の行が更新されます。

検索条件に副照会が含まれる場合、その副照会は、検索条件が 1 つの行に適用されるたびに実行され、その結果は検索条件の適用に使用されるものと見なされます。 実際には、相関参照が含まれていない副照会は一度実行されるのに対し、相関参照を含む副照会は各行ごとに一度ずつ実行しなければならない場合があります。

CURRENT OF *cursor-name*
更新操作で使用するカーソルを指定します。 『DECLARE CURSOR』 で説明されているように、 *cursor-name* は、宣言済みカーソルを指定しなければなりません。プログラムで、UPDATE ステートメントよりも前に、該当の DECLARE CURSOR ステートメントがなければなりません。

指定する表、ビュー、またはニックネームは、そのカーソルの SELECT ステートメントの FROM 文節でも指定されていなければならず、またそのカーソルの結果表が読み取り専用であってはなりません。 (読み取り専用の結果表については、『DECLARE CURSOR』 を参照してください。)

UPDATE ステートメントが実行される時点で、そのカーソルは行に位置づけられていなければなりません。その行が更新されます。

この書式の UPDATE は、カーソルが次のものを参照している場合は使用できません (SQLSTATE 42828)。

- INSTEAD OF UPDATE トリガーが定義されているビュー
- ビューを定義する全選択の選択リストに OLAP 関数が含まれているビュー

- WITH ROW MOVEMENT 文節を使用して直接または間接的に定義されたビュー

WITH
UPDATE ステートメントが実行される分離レベルを指定します。

RR
反復可能読み取り

RS
読み取り固定

CS
カーソル固定

UR
非コミット読み取り

ステートメントのデフォルト分離レベルは、ステートメントがバインドされているパッケージの分離レベルです。

規則:

- ***トリガー:*** UPDATE ステートメントによってトリガーの実行が引き起こされる場合があります。トリガーが他のステートメントの実行を引き起こす場合や、更新値に起因するエラーが発生する場合があります。ビューに対する更新操作を行うと INSTEAD OF トリガーが起動する場合は、そのトリガーによって実行される更新に対して妥当性、参照保全、および制約が検査されます。トリガーを起動させたビューやその基礎表に対する検査は行われません。
- ***割り当て:*** 更新値は、特定の割り当て規則に従って列に割り当てられます。
- ***妥当性:*** 更新される列のユニーク索引がある場合には、その表 (またはビューの基本表) に適用される制約に更新された行は、適合していなければなりません。

  WITH CHECK OPTION を使用して定義されていないビューが使用される場合、行が変更され、その結果、それらの行がそのビューの定義に適合しないことになる場合があります。 そのような行は、ビューの基本表で更新され、そのビューには現れなくなります。

  WITH CHECK OPTION を用いて定義されたビューを使用する場合、更新された行は、そのビューの定義に従っていなければなりません。 この状況に関連する規則については、『CREATE VIEW』 を参照してください。
- ***チェック制約:*** 更新値は、表に定義されているチェック制約の検査条件を満たしていなければなりません。

  チェック制約が定義されている表に対する UPDATE では、更新される各行ごとに一度、更新される各列に対して制約条件が評価されます。 UPDATE ステートメントが処理される時点で、更新される列を参照しているチェック制約だけが検査されます。
- ***参照保全:*** 更新規則が RESTRICT で、従属行が存在する場合には、親のユニーク・キーの値は変更できません。ただし、NO ACTION の更新規則では、更新ステートメントの完了時にすべての子が親キーを持つ場合、親のユニーク・キーを更新することができます。 NULL 以外の外部キーの更新値は、関連する親表の主キーの値に等しくなければなりません。

注:

- 更新値が制約のいずれかに違反している場合、または UPDATE ステートメントの実行時に他のエラーが発生した場合、行は更新されません。 複数の行が更新される順序は、決められていません。
- WITH ROW MOVEMENT 文節を使用して定義されたビューへの更新は、ビューの基礎表に対する削除操作および挿入操作を引き起こす可能性があります。詳細は、CREATE VIEW ステートメントの説明を参照してください。
- UPDATE ステートメントの実行が完了すると、 SQLCA の SQLERRD(3) の値は更新操作用に修飾された行の数を示します。 SQL プロシージャー・ステートメントでは、値は GET DIAGNOSTICS ステートメントの ROW_COUNT 変数を使用して検索できます。 SQLERRD(5) フィールドには、活動化されたすべてのトリガーによって挿入、削除、または更新された行の数が入れられます。
- 適切なロックがすでに存在している場合を除き、正常な UPDATE ステートメントの実行によって、1 つまたは複数の排他ロックが獲得されます。 そのようなロックが解放されるまで、更新された行には、その更新を行ったアプリケーション・プロセス以外はアクセスできません (非コミット読み取り分離レベルを使用するアプリケーションを除く)。 ロッキングについては、COMMIT、ROLLBACK、および LOCK TABLE の各ステートメントの説明を参照してください。
- DATALINK 列の URL 値を更新する場合、それは古い DATALINK 値を削除してから新しい DATALINK 値を挿入するのと同じです。 最初に、古い値があるファイルにリンクされていれば、そのファイルをリンク解除されます。 次に、DATALINK 値のリンケージ属性が空でなければ、指定されたファイルがその列にリンクされます。 ただし例外が 1 つだけあり、それは新しい DATALINK 値の URL と既存の DATALINK の URL が*同一*の場合です。この場合は、関連した Data Links Manager と通信して同一のファイルをリンク解除したり再リンクしたりする必要はありません。この状態の場合、オーバーヘッドは完全に取り除かれます。

  DATALINK 列のコメント値は、空のストリングを URL パスとして指定することによりファイルを再リンクさせなくても (たとえば、 DLVALUE スカラー関数の *data-location* 引き数を指定したり、古い値と同じ値を新しい値として指定したりしなくても)、更新することができます。

  DATALINK 列を更新して NULL にすることは、既存の DATALINK 値を削除することと同じです。

  DATALINK 値を更新しようとしたときに、既存の値または新しい値のファイル・サーバーのいずれかがデータベースに登録されていないと、エラーになる場合があります (SQLSTATE 55022)。
- 型付き表の列分布統計を更新する場合は、列を最初に生成した副表を指定しなければなりません。
- 同じ構造タイプの列で複数の属性割り当てが行われる場合は、 SET 文節で (括弧付きで挿入された SET 文節では左から右の順番で) 指定された順に属性が割り当てられます。

- 属性割り当てでは、ユーザー定義構造タイプの属性に対して mutator メソッドが呼び出されます。たとえば、割り当て `st..a1=x` は、割り当て `st = st..a1(x)` で mutator メソッドを使用した場合と同じ働きをします。
- 通常の割り当ての場合、指定された列に対しては 1 つの割り当てしか行われませんが、属性割り当てでは、1 つの列が複数の割り当てのターゲット列になることができます (ただし、通常の割り当てでターゲット列として指定されていない場合)。
- 特殊タイプとして定義された ID 列が更新された場合は、まずすべての計算がソース・タイプで行われます。その結果は、値が列に実際に割り当てられる前に、ソース・タイプから定義された特殊タイプにキャストされます。 (計算に先立って、元の値がソース・タイプにキャストされることはありません。)
- ID 列に対する SET ステートメントで DB2 によって値が生成されるようにするには、 DEFAULT キーワードを使用します。

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  この例では、NEW.EMPNO が ID 列として定義されており、この列の更新に使用される値は DB2 によって生成されます。
- ID 列に生成されるシーケンス値の使用に関する詳細、または ID 列で値が最大値を超えた場合の詳細は、『INSERT』 を参照してください。

例:

- *例 1:* EMPLOYEE 表において、従業員番号 (EMPNO) '000290' のジョブ (JOB) を 'LABORER' に変更します。

  ```
  UPDATE EMPLOYEE
     SET JOB = 'LABORER'
     WHERE EMPNO = '000290'
  ```

- *例 2:* PROJECT 表において、部門 (DEPTNO) 'D21' が担当しているすべてのプロジェクトについて、プロジェクトのスタッフ・レベル (PRSTAFF) を 1.5 増やします。

  ```
  UPDATE PROJECT
     SET PRSTAFF = PRSTAFF + 1.5
     WHERE DEPTNO = 'D21'
  ```

- *例 3:* 部門 (WORKDEPT) 'E21' の管理者以外の全従業員が一時的に配置替えになったとします。このことは、EMPLOYEE 表において、そのジョブ (JOB) を NULL 値に、給与額 (SALARY、BONUS、COMM) をゼロに変更することにより示されます。

  ```
  UPDATE EMPLOYEE
    SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

  このステートメントは、次のように書き換えることもできます。

  ```
  UPDATE EMPLOYEE
    SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

- *例 4:* 従業員番号 000120 の従業員の給与と歩合の列を、それぞれ更新後の行の部門の従業員の平均給与と平均歩合に更新します。

  ```
  UPDATE (SELECT SALARY,
   COMM,
   AVG(SALARY) OVER (PARTITION BY WORKDEPT),
   AVG(COMM) OVER (PARTITION BY WORKDEPT)
  ```

```
  FROM EMPLOYEE)
  AS E(SALARY, COMM, AVGSAL, AVGCOMM)
  SET (SALARY, COMM)
    = (AVGSAL, AVGCOMM)
  WHERE EU.EMPNO = '000120'
```

上のステートメントは、意味的には次のステートメントと同等ですが、EMPLOYEE 表へのアクセスを一度しか必要としません。それに対し、次のステートメントでは、 EMPLOYEE 表を二度指定します。

```
UPDATE EMPLOYEE EU
  SET (EU.SALARY, EU.COMM)
  =
(SELECT AVG(ES.SALARY), AVG(ES.COMM)
  FROM EMPLOYEE ES
  WHERE ES.WORKDEPT = EU.WORKDEPT)
  WHERE EU.EMPNO = '000120'
```

- *例 5:* C プログラムにおいて、 EMPLOYEE 表の行を表示し、必要に応じて、特定の従業員のジョブ (JOB) を、キーボードから入力した新しいジョブに変更します。

```
EXEC SQL  DECLARE C1 CURSOR FOR
               SELECT *
                 FROM EMPLOYEE
                 FOR UPDATE OF JOB;

EXEC SQL  OPEN C1;

EXEC SQL  FETCH C1 INTO ...     ;
if ( strcmp (change, "YES") == 0 )
  EXEC SQL  UPDATE EMPLOYEE
              SET JOB = :newjob
              WHERE CURRENT OF C1;

EXEC SQL  CLOSE C1;
```

- *例 6:* これらの例では、列オブジェクトの属性を変化させます。

  以下のタイプと表が存在すると想定します。

```
CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE
```

  以下の例では、CIRCLES 表を更新して、OWNER 列と、ID が 999 の CIRCLE 列の RADIUS 属性を変更します。

```
UPDATE CIRCLES
  SET OWNER = 'Bruce'
    C..RADIUS = 5
  WHERE ID = 999
```

  以下の例では、999 で識別される円の中心の X 座標と Y 座標を転置します。

```
UPDATE CIRCLES
  SET C..CENTER..X = C..CENTER..Y,
    C..CENTER..Y = C..CENTER..X
  WHERE ID = 999
```

以下は、上の 2 つのステートメントを別の方法で書いた例です。この例では、上の例に示した 2 つのステートメントの働きを結合させています。

```
UPDATE CIRCLES
  SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
    ('Bruce',5,C..CENTER..Y,C..CENTER..X)
  WHERE ID = 999
```

関連資料:

- *SQL リファレンス 第 1 巻* の『式』
- *SQL リファレンス 第 1 巻* の『検索条件』
- *SQL リファレンス 第 1 巻* の『副選択』
- 45 ページの『ALTER TABLE』
- 474 ページの『CREATE VIEW』
- 491 ページの『DECLARE CURSOR』
- 615 ページの『INSERT』
- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『tbmod.sqc -- How to modify table data (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』
- 『tbmod.sqC -- How to modify table data (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)』
- 『TbMod.java -- How to modify table data (JDBC)』
- 『TutMod.java -- Modify data in a table (JDBC)』
- 『SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)』
- 『TbMod.sqlj -- How to modify table data (SQLj)』
- 『TutMod.sqlj -- Modify data in a table (SQLj)』
- 『tbmod.c -- How to modify table data』
- 『tut_mod.c -- How to modify table data』
- 『updat.sqb -- How to update, delete and insert table data (MF COBOL)』
- 『varinp.sqb -- How to update table data using parameter markers (MF COBOL)』

VALUES

VALUES ステートメントは、照会の 1 つの形式です。 これは、アプリケーション・プログラムに組み込むことも、または対話式に発行することも可能です。

関連資料:

- *SQL リファレンス 第 1 巻* の『全選択』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『fnuse.sqc -- How to use built-in SQL functions (C)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『fnuse.sqC -- How to use built-in SQL functions (C++)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』
- 『lobloc.sqb -- Demonstrates the use of LOB locators (MF COBOL)』

VALUES INTO

VALUES INTO ステートメントは、 0 行か 1 行から成る結果表を作成して、その行の値をホスト変数に割り当てます。

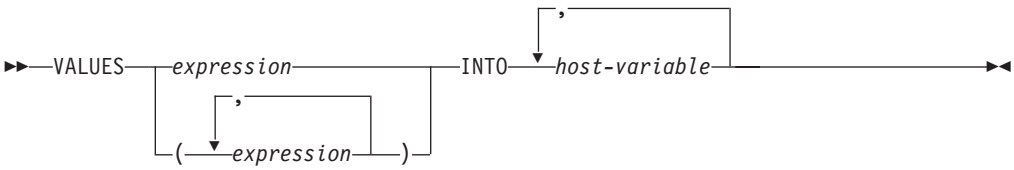
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、動的に作成できない実行可能ステートメントです。

許可:

必要ありません。

構文:



説明:

VALUES

1 つまたは複数の列からなる単一行をこの後に指定します。

expression

1 つの列からなる結果表の単一値を定義する式。

(expression,...)

1 つまたは複数の列からなる結果表の値を定義する 1 つまたは複数の式。

INTO

この後にホスト変数のリストを指定します。

host-variable

ホスト変数の宣言規則に従ってプログラムに記述されている変数を指定します。

結果行の最初の値はリストの最初の変数、 2 番目の値は 2 番目の変数に割り当てられます。以下同様です。ホスト変数の数が列の値の数より少ない場合は、 SQLCA の SQLWARN3 フィールドに値 'W' が割り当てられます

変数への個々の割り当ては、リストに指定された順序で行われます。エラーが発生すると、値はホスト変数に割り当てられません。

例:

例 1: この C の例では、CURRENT PATH 特殊レジスターの値を検索してホスト変数に入れます。

```
EXEC SQL VALUES(CURRENT PATH)
         INTO :hvl;
```

例 2: この C の例では、LOB フィールドの一部を検索してホスト変数に入れます。 LOB ロケーターを使用して、据え置き検索を実行します。

```
EXEC SQL VALUES (substr(:locator1,35))
         INTO :details;
```

関連資料:

- *SQL リファレンス 第 1 巻* の『SQLCA (SQL 連絡域)』
- *SQL リファレンス 第 1 巻* の『割り当てと比較』

WHENEVER

WHENEVER ステートメントは、指定した例外条件が発生した時点で実行するアクションを指定します。

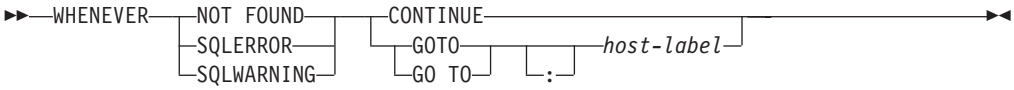
呼び出し:

このステートメントは、アプリケーション・プログラムに組み込む方法でのみ使用可能です。これは、実行可能ステートメントではありません。このステートメントは REXX ではサポートされません。

許可:

必要ありません。

構文:



説明:

NOT FOUND、SQLERROR、または SQLWARNING の各文節は、例外条件のタイプの指定に使用されます。

NOT FOUND
SQLCODE が +100、または SQLSTATE が '02000' になる条件を指定します。

SQLERROR
SQLCODE が負になる条件を指定します。

SQLWARNING
警告状態 (SQLWARN0 が 'W') または SQL 戻りコードが +100 以外の正の値になる条件を指定します。

CONTINUE または GO TO の各文節は、指定したタイプの例外条件が生じた場合に行うアクションを指定します。

CONTINUE
ソース・プログラムの次に続く命令を実行します。

GOTO または **GO TO** *host-label*
host-label で識別されるステートメントに制御を渡します。 *host-label* には、単一のトークンを指定します。オプションとして、その先頭にコロンを付けることができます。 トークンの形式は、ホスト言語によって異なります。

注:

WHENEVER ステートメントには、以下の 3 つのタイプがあります。

- WHENEVER NOT FOUND
- WHENEVER SQLERROR
- WHENEVER SQLWARNING

プログラムの実行可能な SQL ステートメントはいずれも、各タイプの暗黙のまたは明示的な WHENEVER ステートメントの有効範囲内にあります。 WHENEVER

ステートメントの有効範囲は、プログラムのステートメントの実行順序ではなく、ステートメントのリスト順序に関連しています。

SQL ステートメントは、ソース・プログラムでその SQL ステートメントよりも前に指定されている各タイプの最後の WHENEVER ステートメントの有効範囲内にあります。いずれかのタイプの WHENEVER ステートメントが SQL ステートメントよりも前に指定されていない場合、その SQL ステートメントは、CONTINUE が指定されたそのタイプの暗黙の WHENEVER ステートメントの有効範囲内にあります。

例:

次の C の例では、エラーが発生した場合に HANDLERR へ進みます。警告コードを生成された場合は、プログラムの通常フローを続行します。データが戻されない場合には、ENDDATA に進みます。

```
EXEC SQL WHENEVER SQLERROR GOTO HANDLERR;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND GO TO ENDDATA;
```

関連サンプル:

- 『outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (MF COBOL)』
- 『spserver.sqc -- Definition of various types of stored procedures (C)』
- 『spserver.sqC -- Definition of various types of stored procedures (C++)』

WHILE

WHILE ステートメントは、指定した条件が真である間、ステートメント、またはステートメントのグループの実行を繰り返します。

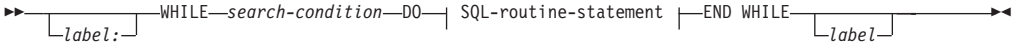
呼び出し:

このステートメントは、SQL プロシージャーまたは動的コンパウンド・ステートメントに組み込むことができます。このステートメントは実行可能ステートメントではなく、動的に準備することはできません。

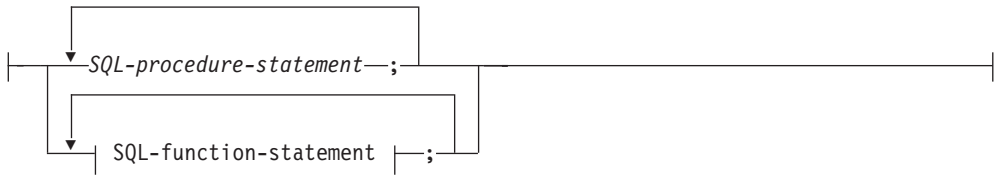
許可:

WHILE ステートメントを呼び出すために、特権は必要ありません。ただし、ステートメントの許可 ID には、 WHILE ステートメントに組み込まれている SQL ステートメントおよび検索条件を呼び出すために必要な特権がなければなりません。

構文:



SQL-routine-statement:



説明:

label
WHILE ステートメントのラベルを指定します。開始ラベルを指定した場合、それを LEAVE および ITERATE ステートメントで指定することができます。終了ラベルを指定する場合、そのラベルは開始ラベルと同じでなければなりません。

search-condition
ループが実行される前に評価される条件を指定します。条件が真であれば、ループ内の SQL-procedure-statement が処理されます。

SQL-procedure-statement
ループ内で実行する SQL ステートメントを指定します。
SQL-procedure-statement は、SQL プロシージャーのコンテキスト内でのみ使用できます。コンパウンド SQL (プロシージャー) ステートメントの説明については、 SQL-procedure-statement の項目を参照してください。

SQL-function-statement
ループ内で実行する SQL ステートメントを指定します。 *SQL-function-statement* は、SQL 関数または SQL メソッドのコンテキスト内でのみ使用できます。
FOR ステートメントの説明にある SQL-function-statement の項目を参照してください。

例:

以下の例では、WHILE ステートメントを使用して、 FETCH から SET ステートメントまでを繰り返します。 SQL 変数 *v_counter* の値が、 IN パラメーター *deptNumber* で識別される部門内の従業員数の半分より少ない間は、 WHILE ステートメントは FETCH および SET ステートメントを引き続き実行します。条件が真でなくなれば、WHILE ステートメントは制御のフローを渡し、カーソルがクローズされます。

```
CREATE PROCEDURE DEPT_MEDIAN
  (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INTEGER DEFAULT 1;
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE c1 CURSOR FOR
      SELECT CAST(salary AS DOUBLE)
      FROM staff
        WHERE DEPT = deptNumber
        ORDER BY salary;
    DECLARE EXIT HANDLER FOR NOT FOUND
      SET medianSalary = 6666;
    SET medianSalary = 0;
    SELECT COUNT(*) INTO v_numRecords
      FROM staff
      WHERE DEPT = deptNumber;
    OPEN c1;
    WHILE v_counter < (v_numRecords/2 + 1) DO
      FETCH c1 INTO medianSalary;
      SET v_counter = v_counter + 1;
    END WHILE;
    CLOSE c1;
  END
```

関連資料:

- 140 ページの『コンパウンド SQL (プロシージャー)』

関連サンプル:

- 『dbinline.sqc -- How to use inline SQL Procedure Language (C)』

付録 A. DB2 Universal Database テクニカル情報

DB2 資料とヘルプ

DB2® 技術情報は、以下のツールと方法を介して利用できます。

- DB2 インフォメーション・センター
  - トピック
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- ダウンロード可能な PDF ファイル、CD 上の PDF ファイル、および印刷された資料
  - ガイド
  - リファレンス・マニュアル
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ
  - SQL 状態ヘルプ
- インストール済みソース・コード
  - サンプル・プログラム

ibm.com® にある技術資料、白書、 Redbooks™ その他の DB2 Universal Database™ 技術情報にオンラインでアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (www.ibm.com/software/data/pubs/) にアクセスしてください。

DB2 資料の更新

IBM® は、 DB2 インフォメーション・センターの資料のフィックスパックやその他の資料更新を定期的に発行しています。 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) にアクセスすれば、常に最新の情報が掲載されます。 DB2 インフォメーション・センターをローカル・インストールしている場合、更新記事を表示するには、まず手動で更新をインストールしてください。新しい情報が発表されたときに資料を更新することにより、 *DB2 インフォメーション・センター CD* からインストールした情報を更新することができます。

インフォメーション・センターの方が、PDF 資料やハードコピー資料よりも頻繁に更新されます。 DB2 の最新の技術情報を入手するには、資料更新が発行されたときにそれをインストールするか、または www.ibm.com サイトの DB2 インフォメーション・センターにアクセスしてください。

関連概念:

- *コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻* の『CLI サンプル・プログラム』

- *アプリケーション開発ガイド アプリケーションの構築および実行* の『Java サンプル・プログラム』
- 798 ページの『DB2 インフォメーション・センター』

関連タスク:

- 819 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 820 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 820 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 821 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 811 ページの『DB2 PDF 資料および印刷された資料』

DB2 インフォメーション・センター

DB2® インフォメーション・センターを使用すると、 DB2 Universal Database™、 DB2 Connect™、 DB2 Information Integrator および DB2 Query Patroller™ などの DB2 ファミリー製品を最大限に活用するのに必要なすべての情報にアクセスできます。また、DB2 インフォメーション・センターは、 DB2 の主な機能とコンポーネントに関する情報を提供します (レプリケーション、データウェアハウジング、および DB2 の種々の Extender など)。

Mozilla 1.0 以上または Microsoft® Internet Explorer 5.5 以上で表示する場合、 DB2 インフォメーション・センターには以下の機能があります。以下のいくつかの機能では、 JavaScript™ のサポートを使用可能にする必要があります:

柔軟なインストール・オプション
以下の中から、 ご使用の環境に最も適したオプションを使って DB2 資料を表示できます。

- 最新の資料を常に自動的に利用できるようにするには、 IBM® の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターからすべての資料に直接アクセスします。
- 更新処理を最小化し、イントラネット内のネットワーク・トラフィックだけに制限するには、イントラネット上の 1 つのサーバーに DB2 資料をインストールします。
- 柔軟性を改善し、ネットワーク接続への依存を軽減するには、個々のコンピューターに DB2 資料をインストールします。

検索 「**検索**」テキスト・フィールドに検索語を入力することにより、 DB2 インフォメーション・センターのすべてのトピックを検索できます。複数の語句を引用符で囲めば、完全一致を検索できます。また、ワイルドカード演算子 (*、?) とブール演算子 (AND、 NOT、OR) を使用して検索を絞り込むことができます。

タスク指向の目次
単一の目次の中から、DB2 資料のトピックを見付けることができます。目

次は、主に実行するタスクの種類に従って編成されていますが、そのほかに製品概要、特定のゴール (目的) の情報、参照情報、索引、および用語集も含まれます。

- 製品概要では、DB2 ファミリーで使用可能な製品間の関係、そうした各製品で提供される機能、および各製品の最新リリース情報について説明されています。
- インストール、管理および開発などのゴール・カテゴリーには、タスクを迅速に完了し、そのための背景情報をよく理解できるようにするトピックが含まれています。
- 「参照」トピックでは、その対象に関する詳細な情報 (ステートメントとコマンドの構文、メッセージ・ヘルプ、構成パラメーターなど) が説明されています。

現在のトピックを目次に表示する

現在のトピックが目次のどの部分に該当するかを表示するには、目次フレーム内の**「リフレッシュ/現在のトピックの表示 (Refresh/Show Current Topic)」**ボタンをクリックするか、コンテンツ・フレーム内の**「目次に表示 (Show in Table of Contents)」**ボタンをクリックします。幾つかのファイルで関連トピックへの複数のリンクをたどった場合、または検索結果からトピックにアクセスした場合には、この機能が役立ちます。

索引 索引から、すべての資料にアクセスすることができます。索引では、用語が 50 音順に編成されています。

用語集 用語集を見れば、DB2 資料で使われているさまざまな用語の定義を調べることができます。用語集では、用語が 50 音順に編成されています。

組み込まれているローカライズ情報

DB2 インフォメーション・センターは、ブラウザーで設定された言語でトピックを表示します。設定された言語のトピックが利用できない場合、DB2 インフォメーション・センターにはそのトピックの英語版が表示されます。

iSeries™ 技術情報については、 IBM eServer™ iSeries Information Center (www.ibm.com/eserver/iseries/infocenter/) を参照してください。

関連概念:

- 800 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 810 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 802 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 805 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターのインストール・シナリオ

さまざまに異なる業務環境のもとでは、 DB2® 情報にどのようにアクセスするかの要件もそれぞれ異なります。 DB2 インフォメーション・センターにアクセスするには、 IBM® の Web サイト、サーバーまたは組織のネットワーク、あるいはコンピューターへのインストールという 3 つの方法が可能です。この 3 つのケースのいずれも、資料は DB2 インフォメーション・センター内に置かれます。インフォメーション・センターは、ブラウザーを使って表示できるように設計されたトピック・ベースの情報の Web サイトです。デフォルトでは、DB2 製品から、 IBM Web サイト上の DB2 インフォメーション・センターにアクセスします。これに対して、 イントラネット・サーバーまたはご自分のコンピューターから DB2 インフォメーション・センターにアクセスしたい場合、製品メディア・パック内にある DB2 インフォメーション・センター CD から DB2 インフォメーション・センターをインストールする必要があります。以下では、DB2 資料へのアクセス・オプションの要約、および 3 つのインストール・シナリオを示します。これを参考にして、お客様の業務環境で DB2 インフォメーション・センターにアクセスするにはどの方法が最適か、どのようなインストール上の問題に配慮する必要があるかを判別してください。

DB2 資料にアクセスするオプションの要約:

以下の表は、お客様の実際の業務環境で、 DB2 インフォメーション・センターの DB2 製品情報にアクセスする方法としてどんなオプションが推奨されるかを示します。

インターネット・アクセス	イントラネット・アクセス	推奨されるアクション
はい	はい	IBM Web サイト上の DB2 インフォメーション・センターへのアクセス、またはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターへのアクセス
はい	いいえ	IBM Web サイト上の DB2 インフォメーション・センターへのアクセス
いいえ	はい	イントラネット・サーバーにインストール済みの DB2 インフォメーション・センターへのアクセス
いいえ	いいえ	ローカル・コンピューター上の DB2 インフォメーション・センターへのアクセス

シナリオ: コンピューター上の DB2 インフォメーション・センターへのアクセス:

Tsu-Chen 氏は小さな町で工場を経営していますが、その町には、インターネット・アクセスを提供する地元のインターネット・サービス・プロバイダーがありません。彼は、在庫、製品オーダー、銀行口座情報、および営業経費を管理するために DB2 Universal Database™ を購入しました。 Tsu-Chen 氏は以前に DB2 製品を利用したことがないので、 DB2 の使用方法を習得するために、DB2 製品資料を参照する必要があります。

Tsu-Chen 氏は 標準インストール・オプションを使って DB2 Universal Database を自分のコンピューターにインストールした後、 DB2 資料にアクセスしようとします。しかし、開こうとしているページが見つからないというエラー・メッセージがブラウザーから通知されました。 Tsu-Chen 氏は DB2 製品のインストール・マニュアルを調べた結果、 DB2 資料を自分のコンピューター上で利用するには、 DB2 インフォメーション・センターをインストールしなければならないことに気付きます。そしてメディア・パックの中にあった *DB2 インフォメーション・センター CD* を見つけ出して、インストールしました。

これで、Tsu-Chen 氏はオペレーティング・システムのアプリケーション・ランチャーから DB2 インフォメーション・センターにアクセスできるようになり、より良い業務成果をあげるために DB2 製品を利用する方法を習得できます。

シナリオ: IBM Web サイト上の DB2 インフォメーション・センターへのアクセス:

Colin は、あるセミナー企業に所属する情報技術コンサルタントです。彼の専門はデータベース・テクノロジーおよび SQL で、 DB2 Universal Database を使って北米一帯の企業を対象にこれらの科目のセミナーを開催しています。 Colin のセミナーでは、教材として DB2 資料も使用されます。たとえば、SQL の講習コースでは、データベース照会の基本構文と拡張構文を教えるために SQL に関する DB2 資料が使用されます。

Colin が教えている企業の大半はインターネット・アクセスを配備しています。このような状況から判断して、Colin は、最新バージョンの DB2 Universal Database を自分のモバイル・コンピューターにインストールしたとき、 IBM Web サイト上の DB2 インフォメーション・センターにアクセスするよう構成しました。この構成によって、Colin はセミナーで教えるときに最新の DB2 資料にオンライン・アクセスすることができます。

しかし、時折、Colin は移動中にインターネット・アクセスを利用できないことがあります。これは問題となります。担任するセミナーの準備のために DB2 資料にアクセスする必要のある場合には、とくにそうです。このような事態が起きないようにするために、 Colin は自分のモバイル・コンピューターに DB2 インフォメーション・センターのコピーをインストールしました。

こうして、Colin は常に DB2 資料のコピーを自在に活用できるようになりました。**db2set** コマンドを使って自分のモバイル・コンピューターのレジストリー変数を簡単に構成し、どこにいるかに応じて、IBM Web サイトまたは自分のモバイル・コンピューターから DB2 インフォメーション・センターにアクセスできます。

シナリオ: イントラネット・サーバー上の DB2 インフォメーション・センターへのアクセス:

Eva は、生命保険会社のデータベース上級管理者です。彼女は管理業務の一環として、会社の UNIX® データベース・サーバーに最新バージョンの DB2 Universal Database をインストールおよび構成します。彼女の会社は最近、セキュリティー上の理由から、インターネット・アクセスをもはや業務で利用できないようにすると社員に通知しました。同社はネットワーク環境を装備しているため、 Eva は DB2 インフォメーション・センターのコピーをイントラネット・サーバー上にインスト

ールして、社内のデータウェアハウスを定期的に利用するすべての社員 (営業担当者、営業部長、および業務分析担当者) から DB2 資料へのアクセスを可能にすることにしました。

Eva は、応答ファイルを使って全社員のコンピューター上に最新バージョンの DB2 Universal Database をインストールするようデータベース・チームに指示します。その際、イントラネット・サーバーのホスト名とポート番号を使って DB2 インフォメーション・センターにアクセスできるよう、確実に各コンピューターを構成します。

しかし、Eva のチームの下級データベース管理者である Migual の誤解によって、数人の社員のコンピューター上で、イントラネット・サーバーの DB2 インフォメーション・センターにアクセスするよう DB2 Universal Database を構成する代わりに、 DB2 インフォメーション・センターのコピーをそれらのコンピューターにインストールしてしまいました。これを訂正するために、Eva は、 **db2set** コマンドを使ってこれらのコンピューター上の DB2 インフォメーション・センターのレジストリー変数 (ホスト名は DB2_DOCHOST、ポート番号は DB2_DOCPORT) を変更するよう Migual に指示しました。これで、ネットワーク上の適切なすべてのコンピューターが DB2 インフォメーション・センターにアクセスできるようになり、社員は DB2 に関する質問の答えを DB2 資料から見つけることができます。

関連概念:

- 798 ページの『DB2 インフォメーション・センター』

関連タスク:

- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 802 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 805 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』

関連資料:

- コマンド・リファレンス の『db2set - DB2 プロファイル・レジストリー・コマンド』

DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)

DB2 製品資料にアクセスする方法として、 IBM Web サイト、イントラネット・サーバー、またはコンピューターにインストールしたバージョンの 3 つがあります。デフォルトでは、DB2 製品は IBM Web サイト上の DB2 資料にアクセスします。イントラネット・サーバーまたはコンピューター上の DB2 資料にアクセスしたい場合には、 *DB2 インフォメーション・センター CD* から資料をインストールする必要があります。 DB2 セットアップ・ウィザードを使用すれば、インストール設

定を定義し、 UNIX オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールできます。

前提条件:

このセクションでは、UNIX コンピューターに DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の諸要件を一覧で示します。

- **ハードウェア要件**

  以下のいずれかのプロセッサーが必要です。

  - PowerPC (AIX)
  - HP 9000 (HP-UX)
  - Intel 32 ビット (Linux)
  - Solaris UltraSPARC コンピューター (Solaris オペレーティング環境)

- **オペレーティング・システム要件**

  以下のいずれかのオペレーティング・システムが必要です。

  - IBM AIX 5.1 (PowerPC 上)
  - HP-UX 11i (HP 9000 上)
  - Red Hat Linux 8.0 (Intel 32 ビット上)
  - SuSE Linux 8.1 (Intel 32 ビット上)
  - Sun Solaris バージョン 8 (Solaris オペレーティング環境の UltraSPARC コンピューター上)

  注: DB2 インフォメーション・センターは、 DB2 クライアントをサポートする UNIX オペレーティング・システム上で稼動します。このため、IBM Web サイトから DB2 インフォメーション・センターにアクセスするか、イントラネット・サーバーに DB2 インフォメーション・センターをインストールしてそれにアクセスすることをお勧めします。

- **ソフトウェア要件**

  - 以下のブラウザーがサポートされています。
    - Mozilla バージョン 1.0 以上

- DB2 セットアップ・ウィザードは、グラフィック・インストーラーです。ご使用のマシンで DB2 セットアップ・ウィザードのグラフィカル・ユーザー・インターフェースを表示可能にする X Window システム・ソフトウェアをインプリメントする必要があります。 DB2 セットアップ・ウィザードを実行する前に、ディスプレイを正しくエクスポートしたことを確認してください。たとえば、コマンド・プロンプトで

  ```
  export DISPLAY=9.26.163.144:0.
  ```

  というコマンドを入力します。

- **通信要件**

  - TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、以下のようにします。

1. システムにログオンします。
2. DB2 インフォメーション・センター製品 CD を挿入してシステムにマウントします。
3. 次のコマンドを入力して、CD がマウントされているディレクトリーに移動します。

   ```
   cd /cd
   ```

   /cd は、CD のマウント・ポイントを表します。
4. **./db2setup** コマンドを入力して、 DB2 セットアップ・ウィザードを開始します。
5. IBM DB2 セットアップ・ランチパッドが開きます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを呼び出すには、**「ヘルプ」**をクリックします。**「キャンセル」**をクリックすれば、いつでもインストールを終了できます。
6. **「インストールしたい製品を選択します」**ページでは、**「次へ」**をクリックします。
7. **「DB2 セットアップ・ウィザードにようこそ (Welcome to the DB2 Setup wizard)」**ページで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードは、プログラムのセットアップ操作を案内します。
8. インストールを続行するには、使用許諾条件に同意する必要があります。**「ご使用条件」**ページで、**「ご使用条件に同意します (I accept the terms in the license agreement)」** をクリックして、**「次へ」**をクリックします。
9. **「インストール・アクションの選択」**で、**「このコンピューターに DB2 インフォメーション・センターをインストールする (Install DB2 Information Center on this computer)」**を選択します。応答ファイルを使用して、このコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをあとでインストールしたい場合には、**「設定を応答ファイルに保管する」**を選択します。**「次へ」**をクリックします。
10. **「インストールする言語の選択」**ページでは、 DB2 インフォメーション・センターをインストールする言語を選択します。**「次へ」**をクリックします。
11. **「DB2 インフォメーション・センター・ポートの指定」**ページでは、 DB2 インフォメーション・センターへの着信通信を構成します。**「次へ」**をクリックしてインストールを続けます。
12. **「ファイルのコピーの開始」**ページでは、インストールの選択項目を確認します。設定を変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

このほか、応答ファイルを使って DB2 インフォメーション・センターをインストールすることもできます。

インストール・ログ db2setup.his、 db2setup.log、および db2setup.err は、デフォルトでは /tmp ディレクトリーに置かれます。

db2setup.log ファイルは、エラーも含めた DB2 製品のインストール情報をすべてキャプチャーします。 db2setup.his ファイルは、コンピューター上の DB2 製品インストール内容をすべて記録します。 DB2 は、db2setup.log ファイルを db2setup.his に付加します。 db2setup.err ファイルは、 Java から戻されるすべてのエラー出力 (例外やトラップの情報など) をキャプチャーします。

インストールが完了したら、ご使用の UNIX オペレーティング・システムに応じて、 DB2 は以下のいずれかのディレクトリーにインストールされます。

- AIX: /usr/opt/db2_08_01
- HP-UX: /opt/IBM/db2/V8.1
- Linux: /opt/IBM/db2/V8.1
- Solaris オペレーティング環境: /opt/IBM/db2/V8.1

関連概念:

- 798 ページの『DB2 インフォメーション・センター』
- 800 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- *インストールおよび構成 補足* の『応答ファイルによる DB2 のインストール (UNIX)』
- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 810 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 805 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)

DB2 製品資料にアクセスする方法として、 IBM Web サイト、イントラネット・サーバー、またはコンピューターにインストールしたバージョンの 3 つがあります。デフォルトでは、DB2 製品は IBM Web サイト上の DB2 資料にアクセスします。イントラネット・サーバーまたはコンピューター上の DB2 資料にアクセスしたい場合には、 *DB2 インフォメーション・センター CD* から DB2 資料をインストールする必要があります。 DB2 セットアップ・ウィザードを使用すれば、インストール設定を定義し、 Windows オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールできます。

前提条件:

このセクションでは、Windows に DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の諸要件を一覧で示します。

- **ハードウェア要件**

  以下のいずれかのプロセッサーが必要です。

  – 32 ビット・コンピューター: Pentium または Pentium 互換の CPU

- **オペレーティング・システム要件**

  以下のいずれかのオペレーティング・システムが必要です。

  – Windows 2000

  – Windows XP

  注: DB2 インフォメーション・センターは、 DB2 クライアントをサポートする Windows オペレーティング・システム上で稼動します。このため、IBM Web サイトの DB2 インフォメーション・センターにアクセスするか、イントラネット・サーバーに DB2 インフォメーション・センターをインストールしてそれにアクセスすることをお勧めします。

- **ソフトウェア要件**

  – 以下のブラウザーがサポートされています。

    - Mozilla 1.0 以上

    - Internet Explorer バージョン 5.5 または 6.0 (Windows XP の場合はバージョン 6.0)

- **通信要件**

  – TCP/IP

制約事項:

- DB2 インフォメーション・センターをインストールするには、管理権限をもつアカウントが必要です。

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、以下のようにします。

1. DB2 インフォメーション・センターのインストールで定義したアカウントで、システムにログオンします。
2. CD をドライブに挿入します。自動実行機能が使用可能になっていれば、IBM DB2 セットアップ・ランチパッドが起動します。
3. DB2 セットアップ・ウィザードは、システム言語を判別して、その言語用のセットアップ・プログラムを立ち上げます。英語以外の言語でセットアップ・プログラムを実行したい場合、またはセットアップ・プログラムの自動始動が失敗した場合には、 DB2 セットアップ・ウィザードを手動で開始できます。

   次のようにして、DB2 セットアップ・ウィザードを手動で開始します。

   a. **「スタート」**をクリックし、**「ファイル名を指定して実行」**を選択します。

   b. **「開く」**フィールドで、以下のコマンドを入力します。

      ```
      x:¥setup.exe /i 2-letter language identifier
      ```

ここで、*x:* は CD ドライブ、 *2-letter language identifier* (2 文字の言語識別子) はセットアップ・プログラムを実行する言語を表します。

c. **「OK」**をクリックします。

4. IBM DB2 セットアップ・ランチパッドが開きます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを呼び出すには、**「ヘルプ」**をクリックします。**「キャンセル」**をクリックすれば、いつでもインストールを終了できます。
5. **「インストールしたい製品を選択します」**ページでは、**「次へ」**をクリックします。
6. **「DB2 セットアップ・ウィザードにようこそ (Welcome to the DB2 Setup wizard)」**ページで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードは、プログラムのセットアップ操作を案内します。
7. インストールを続行するには、使用許諾条件に同意する必要があります。**「ご使用条件」**ページで、**「ご使用条件に同意します (I accept the terms in the license agreement)」** をクリックして、**「次へ」**をクリックします。
8. **「インストール・アクションの選択」**で、**「このコンピューターに DB2 インフォメーション・センターをインストールする (Install DB2 Information Center on this computer)」**を選択します。応答ファイルを使用して、このコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをあとでインストールしたい場合には、**「設定を応答ファイルに保管する」**を選択します。**「次へ」**をクリックします。
9. **「インストールする言語の選択」**ページでは、 DB2 インフォメーション・センターをインストールする言語を選択します。**「次へ」**をクリックします。
10. **「DB2 インフォメーション・センター・ポートの指定」**ページでは、 DB2 インフォメーション・センターへの着信通信を構成します。**「次へ」**をクリックしてインストールを続けます。
11. **「ファイルのコピーの開始」**ページでは、インストールの選択項目を確認します。設定を変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

応答ファイルを使って DB2 インフォメーション・センターをインストールすることができます。また、**db2rspgn** コマンドを使って、既存のインストール内容に基づく応答ファイルを生成することもできます。

インストール時に検出されるエラーの詳細については、「マイ ドキュメント」¥DB2LOG¥ ディレクトリー内の db2.log ファイルと db2wi.log ファイルを参照してください。「マイ ドキュメント」 ディレクトリーの場所は、ご使用のコンピューターの設定によって異なります。

db2wi.log ファイルは、 DB2 の最新のインストール情報をキャプチャーします。db2.log は、DB2 製品のインストールの履歴をキャプチャーします。

関連概念:

• 798 ページの『DB2 インフォメーション・センター』

- 800 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- *インストールおよび構成 補足* の『応答ファイルによる DB2 製品のインストール (Windows)』
- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 810 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 802 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』

関連資料:

- コマンド・リファレンス の『db2rspgn - 応答ファイル生成プログラム・コマンド』

DB2 インフォメーション・センターの呼び出し

DB2 インフォメーション・センターは、Linux、UNIX、および Windows オペレーティング・システム用の DB2 製品 (DB2 Universal Database、 DB2 Connect、DB2 Information Integrator、 DB2 Query Patroller など) を使用するために必要なすべての情報を提供します。

DB2 インフォメーション・センターは、以下の場所から呼び出すことができます。

- DB2 UDB クライアントまたはサーバーがインストールされているコンピューター
- DB2 インフォメーション・センターがインストールされているイントラネット・サーバーまたはローカル・コンピューター
- IBM の Web サイト

前提条件:

DB2 インフォメーション・センターを呼び出すための要件は、以下のとおりです。

- *オプショナル*: 希望する言語でトピックを表示するようブラウザーを構成する
- *オプショナル*: コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを使用するよう DB2 クライアントを構成する

手順:

DB2 UDB クライアントまたはサーバーがインストールされているコンピューターから DB2 インフォメーション・センターを呼び出すには、以下のようにします。

- (Windows オペレーティング・システムの)「スタート」メニューから: **「スタート」→「プログラム」→「IBM DB2」→「情報」→「インフォメーション・センター」**をクリックします。
- コマンド行プロンプトから:
  - Linux および UNIX オペレーティング・システムの場合、 **db2icdocs** コマンドを発行します。

– Windows オペレーティング・システムの場合、 **db2icdocs.exe** コマンドを発行します。

イントラネット・サーバーまたはローカル・コンピューターにインストール済みの DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ http://<host-name>:<port-number>/ を開きます (<host-name> はホスト名、 <port-number> は DB2 インフォメーション・センターを利用可能なポート番号)。

IBM Web サイトにある DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ publib.boulder.ibm.com/infocenter/db2help/ を開きます。

関連概念:

- 798 ページの『DB2 インフォメーション・センター』
- 800 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 819 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 809 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 820 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』

関連資料:

- *コマンド・リファレンス* の『HELP コマンド』

コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール

http://publib.boulder.ibm.com/infocenter/db2help/ から利用できる DB2 インフォメーション・センターは、資料の新規追加または変更によって定期的に更新されます。さらに、更新された DB2 インフォメーション・センターをコンピューターまたはイントラネット・サーバーにダウンロードしてインストールできる場合もあります。DB2 インフォメーション・センターを更新しても、 DB2 クライアント製品またはサーバー製品は更新されません。

前提条件:

インターネットに接続されたコンピューターへのアクセスが必要です。

手順:

DB2 インフォメーション・センターの更新をコンピューターまたはイントラネット・サーバーにインストールするには、以下のようにします。

1. IBM の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターを開きます。

2. 「DB2 インフォメーション・センターにようこそ」ページの見出し「サービスおよびサポート」の「ダウンロード」セクションで、**「DB2 資料」**リンクをクリックします。
3. 最新のドキュメンテーション・イメージのレベルと、インストール済みのドキュメンテーション・レベルを比較して、 DB2 インフォメーション・センターを更新する必要があるかどうかを確認します。「DB2 インフォメーション・センターにようこそ」ページに、インストール済みのドキュメンテーションのレベルがリストされます。
4. より新しいバージョンの DB2 インフォメーション・センターが存在する場合、ご使用のオペレーティング・システムに対応する最新の *DB2 インフォメーション・センター・イメージをダウンロードします。*
5. 最新の *DB2 インフォメーション・センター・イメージをインストールするには、* Web ページの指示に従ってください。

関連概念:

- 800 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 802 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 805 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターにおける特定の言語でのトピックの表示

DB2 インフォメーション・センターでは、ブラウザーの設定で指定した言語でのトピックの表示が試みられます。トピックがその指定言語に翻訳されていない場合は、 DB2 インフォメーション・センターでは英語でトピックが表示されます。

手順:

Internet Explorer Web ブラウザーで、指定どおりの言語でトピックを表示するには、以下のようにします。

1. Internet Explorer の**「ツール」**—> **「インターネット オプション」**—> **「言語...」**ボタンをクリックします。「言語の優先順位」ウィンドウがオープンします。
2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
   - リストに新しい言語を追加するには、**「追加...」**ボタンをクリックします。

     注: 言語を追加しても、特定の言語でトピックを表示するのに必要なフォントがコンピューターに備えられているとはかぎりません。
   - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで**「上へ」**ボタンをクリックします。
3. 使いたい言語で DB2 インフォメーション・センターを表示するには、ページをリフレッシュします。

Mozilla Web ブラウザーの場合に、使いたい言語でトピックを表示するには、以下のようにします。

1. Mozilla の**「編集」**—> **「設定」**—> **「言語」**ボタンをクリックします。「設定」ウィンドウに「言語」パネルが表示されます。
2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
   - リストに新しい言語を追加するには、**「追加...」**ボタンをクリックしてから、「言語を追加」ウィンドウで言語を選択します。
   - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで**「上に移動」**ボタンをクリックします。
3. 使いたい言語で DB2 インフォメーション・センターを表示するには、ページをリフレッシュします。

関連概念:

- 798 ページの『DB2 インフォメーション・センター』

DB2 PDF 資料および印刷された資料

以下の表は、正式な資料名、資料番号、および PDF ファイル名を示しています。ハードコピー版の資料を注文するには、正式な資料名を知っておく必要があります。 PDF ファイルを印刷するには、PDF ファイル名を知っておく必要があります。

DB2 資料は、以下のカテゴリーに分類されています。

- DB2 中核情報
- 管理情報
- アプリケーション開発情報
- ビジネス・インテリジェンス情報
- DB2 Connect 情報
- 入門情報
- チュートリアル情報
- オプショナル・コンポーネント情報
- リリース・ノート

以下の表は、DB2 ライブラリー内の各資料について、その資料のハードコピー版を注文したり、PDF 版を印刷または表示したりするのに必要な情報を示しています。DB2 ライブラリー内の各資料に関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。

DB2 の基本情報

こうした資料の情報は、すべての DB2 ユーザーに基本的なもので、プログラマーおよびデータベース管理者にとって役立つ情報であるとともに、DB2 Connect、DB2 Warehouse Manager、または他の DB2 製品を使用するユーザーにとっても役立つ内容です。

表 13. DB2 の基本情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database コマンド・リファレンス*」	SC88-9140	db2n0j81
「*IBM DB2 Universal Database 用語集*」	資料番号なし	db2t0j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 1 巻*」	GC88-9152 (ハードコピーなし)	db2m1j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 2 巻*」	GC88-9153 (ハードコピーなし)	db2m2j81
「*IBM DB2 Universal Database 新機能*」	SC88-9158	db2q0j81

管理情報

これらの資料の情報は、DB2 データベース、データウェアハウス、およびフェデレーテッド・システムを効果的に設計し、インプリメントし、保守するために必要なトピックを扱っています。

表 14. 管理情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database 管理ガイド: プランニング*」	SC88-9135	db2d1j81
「*IBM DB2 Universal Database 管理ガイド: インプリメンテーション*」	SC88-9133	db2d2j81
「*IBM DB2 Universal Database 管理ガイド: パフォーマンス*」	SC88-9134	db2d3j81
「*IBM DB2 Universal Database 管理 API リファレンス*」	SC88-9136	db2b0j81
「*IBM DB2 Universal Database データ移動ユーティリティー ガイドおよびリファレンス*」	SC88-9142	db2dmj81
「*IBM DB2 Universal Database データ・リカバリーと高可用性 ガイドおよびリファレンス*」	SC88-9143	db2haj81
「*IBM DB2 Universal Database データウェアハウス・センター 管理ガイド*」	SC88-9165	db2ddj81
「*IBM DB2 Universal Database SQL リファレンス 第 1 巻*」	SC88-9155	db2s1j81
「*IBM DB2 Universal Database SQL リファレンス 第 2 巻*」	SC88-9156	db2s2j81

表 14. 管理情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database システム・モニター ガイドおよびリファレンス*」	SC88-9157	db2f0j81

アプリケーション開発情報

これらの資料の情報は、 DB2 Universal Database (DB2 UDB) のアプリケーション開発者またはプログラマーが特に関心を持つ内容です。サポートされるさまざまなプログラミング・インターフェース (組み込み SQL、ODBC、JDBC、SQLJ、CLI など) を使用して DB2 UDB にアクセスするのに必要な資料とともに、サポートされる言語およびコンパイラーについても紹介されています。また、DB2 インフォメーション・センターをご使用の場合には、サンプル・プログラムのソース・コードの HTML バージョンにアクセスすることもできます。

表 15. アプリケーション開発情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database アプリケーション開発ガイド アプリケーションの構築および実行*」	SC88-9137	db2axj81
「*IBM DB2 Universal Database アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」	SC88-9138	db2a1j81
「*IBM DB2 Universal Database アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」	SC88-9139	db2a2j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」	SC88-9159	db2l1j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」	SC88-9160	db2l2j81
「*IBM DB2 Universal Database データウェアハウス・センター アプリケーション統合ガイド*」	SC88-9166	db2adj81
「*IBM DB2 Universal Database XML Extender 管理およびプログラミングのガイド*」	SC88-9172	db2sxj81

ビジネス・インテリジェンス情報

これらの資料の情報は、さまざまなコンポーネントを使用して、 DB2 Universal Database のデータウェアハウジング機能および分析機能を拡張する方法を説明しています。

表 16. ビジネス・インテリジェンス情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Warehouse Manager Standard Edition インフォメーション・カタログ・センター 管理ガイド*」	SC88-9167	db2dij81
「*IBM DB2 Warehouse Manager Standard Edition インストール・ガイド*」	GC88-9164	db2idj81
「*IBM DB2 Warehouse Manager Standard Edition DB2 Warehouse Manager を使用時の ETI ソリューション・コンバージョン・プログラムの管理*」	SC88-9894	iwhe1mstx80

DB2 Connect 情報

このカテゴリーの情報は、 DB2 Connect Enterprise Edition または DB2 Connect Personal Edition を使用して、メインフレーム・サーバーおよびミッドレンジ・サーバー上のデータにアクセスする方法を説明しています。

表 17. DB2 Connect 情報

資料名	資料番号	PDF ファイル名
「*IBM コネクティビティー 補足*」	資料番号なし	db2h1j81
「*IBM DB2 Connect Enterprise Edition 概説およびインストール*」	GC88-9145	db2c6j81
「*IBM DB2 Connect Personal Edition 概説およびインストール*」	GC88-9146	db2c1j81
「*IBM DB2 Connect ユーザーズ・ガイド*」	SC88-9147	db2c0j81

入門情報

このカテゴリーの情報は、サーバー、クライアント、および他の DB2 製品をインストールして構成する場合に役立ちます。

表 18. 入門情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database DB2 クライアント機能 概説およびインストール*」	GC88-9144 (ハードコピーなし)	db2itj81
「*IBM DB2 Universal Database DB2 サーバー機能 概説およびインストール*」	GC88-9148	db2isj81
「*IBM DB2 Universal Database DB2 Personal Edition 概説およびインストール*」	GC88-9150	db2i1j81
「*IBM DB2 Universal Database インストールおよび構成 補足*」	GC88-9149 (ハードコピーなし)	db2iyj81
「*IBM DB2 Universal Database DB2 Data Links Manager 概説およびインストール*」	GC88-9141	db2z6j81

チュートリアル情報

チュートリアル情報は、DB2 機能を紹介し、さまざまなタスクを実行する方法を示します。

表 19. チュートリアル情報

資料名	資料番号	PDF ファイル名
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介*」	資料番号なし	db2tuj81
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド*」	資料番号なし	db2taj81
「*インフォメーション・カタログ・センター チュートリアル*」	資料番号なし	db2aij81
「*Video Central for e-business チュートリアル*」	資料番号なし	db2twj81
「*Visual Explain チュートリアル*」	資料番号なし	db2tvj81

オプショナル・コンポーネント情報

このカテゴリーの情報は、 DB2 のオプショナル・コンポーネントを使用する方法について説明しています。

表 20. オプショナル・コンポーネント情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Cube Views Guide and Reference*」	SC18–7298	db2aax81
「*IBM DB2 Query Patroller インストール、管理、使用法のガイド*」	GC88–9154	db2dwj81
「*IBM DB2 Spatial Extender and Geodetic Extender ユーザーズ・ガイドおよびリファレンス*」	SC88-9171	db2sbj81
「*IBM DB2 Universal Database Data Links Manager 管理ガイドおよびリファレンス*」	SC88-9169	db2z0x82
「*DB2 Net Search Extender 管理およびユーザーズ・ガイド*」 **注:** この資料の HTML 版は、HTML ドキュメンテーション CD からインストールされません。	SH88-8546	N/A

リリース・ノート

リリース・ノートは、ご使用の製品のリリースおよびフィックスパック・レベルに特有の追加情報を紹介します。また、リリース・ノートには、各リリース、アップデート、およびフィックスパックで組み込まれた資料上の更新の要約も含まれています。

表 21. リリース・ノート

資料名	資料番号	PDF ファイル名
「*DB2 リリース・ノート*」	「注」を参照。	「注」を参照。
「*DB2 インストール情報*」	製品 CD-ROM でのみ参照可能。	使用できません。

注: リリース・ノートは以下の形式で入手できます。

- XHTML およびテキスト形式 (製品 CD 内)
- PDF 形式 (PDF ドキュメンテーション CD 内)

さらに、リリース・ノートの中で、『*既知の問題と予備手段*』および『*リリース間の非互換性*』に関する部分は DB2 インフォメーション・センターにも表示されます。

UNIX ベースのプラットフォームでテキスト形式でリリース・ノートを確認するには、 Release.Notes ファイルを参照してください。このファイルは、DB2DIR/Readme/*%L* ディレクトリーに収録されています。 *%L* はロケール名を表しています。 DB2DIR は以下になります。

- AIX オペレーティング・システムの場合: /usr/opt/db2_08_01
- その他のすべての UNIX ベースのオペレーティング・システムの場合: /opt/IBM/db2/V8.1

関連概念:

- 797 ページの『DB2 資料とヘルプ』

関連タスク:

- 817 ページの『PDF ファイルからの DB2 資料の印刷方法』
- 818 ページの『DB2 の印刷資料の注文方法』
- 819 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』

PDF ファイルからの DB2 資料の印刷方法

DB2 PDF ドキュメンテーション CD に収録されている DB2 資料を印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷できます。

前提条件:

Adobe Acrobat Reader がインストールされていることを確認してください。 Adobe Acrobat Reader をインストールする必要がある場合、 Adobe Web サイト (www.adobe.com) から入手できます。

手順:

PDF ファイルから DB2 資料を印刷するには以下のようにします。

1. *DB2 PDF* ドキュメンテーション CD をドライブに挿入します。 UNIX オペレーティング・システムの場合、 DB2 PDF ドキュメンテーション CD をマウントします。 UNIX オペレーティング・システムで CD をマウントする方法については、「*概説およびインストール*」を参照してください。
2. index.htm を開きます。ブラウザー・ウィンドウにファイルが開きます。
3. 参照したい PDF のタイトルをクリックします。 Acrobat Reader で PDF が開きます。
4. **「ファイル」 → 「印刷」**を選択して、所要の資料の任意の部分を印刷します。

関連概念:

- 798 ページの『DB2 インフォメーション・センター』

関連タスク:

- *DB2 Universal Database サーバー機能 概説およびインストール* の『CD-ROM のマウント (AIX)』
- *DB2 Universal Database サーバー機能 概説およびインストール* の『HP-UX 上での CD-ROM のマウント』
- *DB2 Universal Database サーバー機能 概説およびインストール* の『CD-ROM のマウント (Linux)』
- 818 ページの『DB2 の印刷資料の注文方法』

- *DB2 Universal Database サーバー機能 概説およびインストール* の『CD-ROM のマウント (Solaris)』

関連資料:

- 811 ページの『DB2 PDF 資料および印刷された資料』

DB2 の印刷資料の注文方法

ハードコピー版の資料を望む場合には、以下のいずれかの方法で注文できます。

印刷資料の注文方法:

一部の国または地域では、印刷された資料を注文することもできます。お客様がお住まいの国または地域でこのサービスが利用可能かどうかを確認するには、お住まいの国または地域の IBM Publications Web サイトをご覧ください。資料のご注文が可能な場合、以下のようにすることができます。

- 正規の IBM 製品販売業者または営業担当員に連絡してください。お客様がお住まいの地域の IBM 担当員の情報については、お手数ですが IBM の Web サイト (www.ibm.com/planetwide) の IBM Worldwide Directory of Contacts で確認してください。
- IBM Publications Center (http://www.ibm.com/shop/publications/order) にアクセスしてください。なお、IBM Publications Center から資料を注文できない国もあります。

DB2 製品がご利用可能になった時点で、印刷された資料は *DB2 PDF* ドキュメンテーション *CD* にある PDF 形式の資料と同じものです。さらに、 *DB2* インフォメーション・センター *CD* に収録されている印刷された資料の内容もまた、これらと同じです。ただし、DB2 インフォメーション・センター CD には、 PDF 資料にない追加情報も含まれます (たとえば、SQL 管理作業や HTML サンプル)。 DB2 PDF ドキュメンテーション CD に収録されている資料の中には、ハードコピーとしてご注文できない資料もあります。

注: DB2 インフォメーション・センターは、 PDF またはハードコピー の資料よりも頻繁に更新されます。ドキュメンテーションの更新が入手可能になった時点でインストールするか、 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) を参照して最新の情報を入手してください。

関連タスク:

- 817 ページの『PDF ファイルからの DB2 資料の印刷方法』

関連資料:

- 811 ページの『DB2 PDF 資料および印刷された資料』

DB2 ツールからコンテキスト・ヘルプを呼び出す

コンテキスト・ヘルプは、特定のウィンドウ、ノートブック、ウィザード、またはアドバイザーに関連したタスクまたはコントロールの情報を提供します。コンテキスト・ヘルプは、グラフィカル・ユーザー・インターフェースのある DB2 管理ツールおよび開発ツールから利用できます。コンテキスト・ヘルプには、以下の 2 種類があります。

- それぞれのウィンドウまたはノートブックにある**「ヘルプ」**ボタンからアクセス可能なヘルプ
- infopop (ポップアップ情報ウィンドウ)。これは、マウス・カーソルを特定のフィールドまたはコントロール上に置いたとき、またはウィンドウ、ノートブック、ウィザード、アドバイザー内でフィールドまたはコントロールを選択して F1 を押すと表示されます。

「ヘルプ」ボタンを押すと、概説、前提条件、およびタスク情報が表示されます。infopop は、それぞれのフィールドおよびコントロールについて説明します。

手順:

コンテキスト・ヘルプを呼び出すには、以下のようにします。

- ウィンドウおよびノートブックのヘルプを表示するには、いずれかの DB2 ツールを開始して、任意のウィンドウまたはノートブックを開きます。ウィンドウまたはノートブックの右下隅にある**「ヘルプ」**ボタンをクリックして、コンテキスト・ヘルプを呼び出します。

  また、それぞれの DB2 ツール・センターの上部にある**「ヘルプ」**メニュー項目からコンテキスト・ヘルプにアクセスすることもできます。

  ウィザードおよびアドバイザーでは、最初のページの「タスクの概要」リンクをクリックすると、コンテキスト・ヘルプを表示できます。
- ウィンドウまたはノートブック上の各コントロールの infopop ヘルプを表示するには、コントロールをクリックしてから、**F1** を押します。コントロールの詳細情報を示すポップアップ情報が、黄色いウィンドウに表示されます。

  注: フィールドまたはコントロールにマウス・カーソルを置いておくだけで infopops が表示されるようにするには、「ツール設定」ノートブックの**「文書 (Documentation)」**ページの**「infopops の自動表示」**チェック・ボックスを選択します。

  infopop に似た別のコンテキスト・ヘルプに、診断ポップアップ情報があります。これにはデータ入力規則が示されます。診断ポップアップ情報は、無効または不十分なデータが入力されたとき、紫色のウィンドウに表示されます。診断ポップアップ情報は、以下に関して表示されます。
  - 必須フィールド。
  - 日付フィールドのように、正確なフォーマットを必要とするデータのフィールド。

関連タスク:

- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 820 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』

- 820 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 821 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』
- 『DB2 インフォメーション・センターへのアクセス: Concepts help』
- 『DB2 UDB ヘルプの使用法: Common GUI help』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』
- 『DB2 コンテキスト・ヘルプと資料へのアクセスを設定する: Common GUI help』

コマンド行プロセッサーからメッセージ・ヘルプを呼び出す

メッセージ・ヘルプは、メッセージが出された原因と、エラーへの応答として実行すべきアクションを説明します。

手順:

メッセージ・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? XXXnnnnn
```

ここで、*XXXnnnnn* は有効なメッセージ ID を表します。

たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。

関連概念:

- *メッセージ・リファレンス 第 1 巻* の『メッセージの概要』

関連資料:

- *コマンド・リファレンス* の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーからコマンド・ヘルプを呼び出す

コマンド・ヘルプは、コマンド行プロセッサーでのコマンドの構文を説明します。

手順:

コマンド・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? command
```

ここで *command* はキーワードまたはコマンド全体を表します。

たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、 `? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプだけが表示されます。

関連タスク:

- 819 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 820 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 821 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- コマンド・リファレンス の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーから SQL 状態ヘルプを呼び出す

DB2 Univerrsal Database は、 SQL ステートメントの結果の原因となったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順:

SQL 状態ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。

たとえば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

関連タスク:

- 808 ページの『DB2 インフォメーション・センターの呼び出し』
- 820 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 820 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』

DB2 チュートリアル

DB2® チュートリアルは、 DB2 Universal Database のさまざまな機能について学習するのを支援します。このチュートリアルでは、アプリケーションの開発、SQL 照会のパフォーマンス調整、データウェアハウスの処理、メタデータの管理、および DB2 を使用した Web サービスの開発の各分野で、段階的なレッスンが用意されています。

はじめに:

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

チュートリアルの中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、それぞれのチュートリアルを参照してください。

DB2 Universal Database チュートリアル:

以下に示すチュートリアルのタイトルをクリックすると、そのチュートリアルを表示できます。

ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介
データウェアハウス・センターを使用して簡単なデータウェアハウジング・タスクを実行します。

ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド
データウェアハウス・センターを使用して高度なデータウェアハウジング・タスクを実行します。

インフォメーション・カタログ・センター・チュートリアル
インフォメーション・カタログを作成および管理して、インフォメーション・カタログ・センターを使用してメタデータを配置し使用します。

Visual Explain チュートリアル
Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2® 製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション
トラブルシューティング情報は、DB2 インフォメーション・センター、および DB2 ライブラリーに含まれる PDF 資料の中でご利用いただけます。DB2 インフォメーション・センターで、 (ブラウザー・ウィンドウの左側の) ナビゲーション・ツリーの「サポートおよびトラブルシューティング (Support and troubleshooting)」ブランチを参照すると、 DB2 トラブルシューティング・ドキュメンテーションの詳細なリストが見つかります。

DB2 Technical Support の Web サイト
現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、 DB2 Technical Support の Web サイトを参照してください。Technical Support サイトには、最新の DB2 出版物、TechNotes、プログラム診断依頼書 (APAR)、フィックスパック、DB2 内部エラー・コードの最新リスト、その他のリソースが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

DB2 Technical Support の Web サイト (http://www.ibm.com/software/data/db2/udb/winos2unix/support) にアクセスしてください。

DB2 Problem Determination Tutorial Series
DB2 製品で作業中に直面するかもしれない問題を素早く識別し、解決する方法に関する情報を見つけるには、 DB2 Problem Determination Tutorial Series の Web サイトを参照してください。あるチュートリアルでは、使用可能な DB2 問題判別機能およびツールを紹介し、それらをいつ使用すべきかを判断する助けを与えます。別のチュートリアルは、『データベース・エ

ンジン問題判別 (Database Engine Problem Determination)』、『パフォーマンス問題判別 (Performance Problem Determination)』、『アプリケーション問題判別 (Application Problem Determination)』などの関連トピックを扱っています。

DB2 Technical Support (http://www.ibm.com/software/data/support/pdm/db2tutorials.html) には、 DB2 問題判別チュートリアルがすべて揃っています。

関連概念:

- 798 ページの『DB2 インフォメーション・センター』
- *問題判別の手引き* の『Introduction to Problem Determination - DB2 テクニカル・サポートのチュートリアル』

アクセス支援

アクセス支援機能は、身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーがソフトウェア製品を十分活用できるように支援します。 DB2® バージョン 8 製品に備わっている主なアクセス支援機能は、以下のとおりです。

- すべての DB2 機能は、マウスの代わりにキーボードを使ってナビゲーションできます。詳細については、『キーボードによる入力およびナビゲーション』を参照してください。
- DB2 ユーザー・インターフェースのフォント・サイズおよび色をカスタマイズすることができます。詳細については、 824 ページの『アクセスしやすい表示』を参照してください。
- DB2 製品は、Java™ Accessibility API を使用するアクセス支援アプリケーションをサポートします。詳細については、 824 ページの『支援テクノロジーとの互換性』を参照してください。
- DB2 資料は、アクセスしやすい形式で提供されています。詳細については、 824 ページの『アクセスしやすい資料』を参照してください。

キーボードによる入力およびナビゲーション

キーボード入力

キーボードだけを使用して DB2 ツールを操作できます。マウスを使って実行できる操作は、キーまたはキーの組み合わせによっても実行できます。標準のオペレーティング・システム・キー・ストロークを使用して、標準のオペレーティング・システム操作を実行できます。

キーまたはキーの組み合わせによって操作を実行する方法について、詳しくは キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・ナビゲーション

キーまたはキーの組み合わせを使用して、 DB2 ツールのユーザー・インターフェースをナビゲートできます。

キーまたはキーの組み合わせによって DB2 ツールをナビゲートする方法の詳細については、キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・フォーカス

UNIX® オペレーティング・システムでは、アクティブ・ウィンドウの中で、キー・ストロークによって操作できる領域が強調表示されます。

アクセスしやすい表示

DB2 ツールには、視力の弱いユーザー、その他の視力障害をもつユーザーのためにアクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

「ツール設定」ノートブックを使用して、メニューおよびダイアログ・ウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

フォント設定に関する詳細情報は、メニューおよびテキストのフォントを変更する: Common GUI help を参照してください。

色に依存しない

本製品のすべての機能を使用するために、ユーザーは必ずしも色を識別する必要はありません。

支援テクノロジーとの互換性

DB2 ツールのインターフェースは、Java Accessibility API をサポートします。これによって、スクリーン・リーダーその他の支援テクノロジーを DB2 製品で利用できるようになります。

アクセスしやすい資料

DB2 形式は、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で提供されています。 XHTML により、ご使用のブラウザーに設定されている表示設定に従って資料を表示できます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

シンタックス・ダイアグラムはドット 10 進形式で提供されます。この形式は、スクリーン・リーダーを使用してオンライン・ドキュメンテーションにアクセスする場合にのみ使用できます。

関連概念:

- 825 ページの『ドット 10 進シンタックス・ダイアグラム』

関連タスク:

- 『キーボード・ショートカットおよびアクセラレーター: Common GUI help』
- 『メニューおよびテキストのフォントを変更する: Common GUI help』

ドット 10 進シンタックス・ダイアグラム

スクリーン・リーダーを使用してインフォメーション・センターを利用するユーザーのために、シンタックス・ダイアグラムがドット 10 進形式で提供されます。

ドット 10 進形式では、各シンタックス・エレメントは別々の行に書き込まれます。複数のシンタックス・エレメントが常に同時に存在する (または常に同時に不在の) 場合、単一のコンパウンド・シンタックス・エレメントとみなせるので同一行に表示できます。

各行は、ドット 10 進数で開始します。たとえば、3 または 3.1 ないしは 3.1.1 です。こうした数を適切に聞き取るには、スクリーン・リーダーが句読点を読み取るように設定されていることを確認してください。同じドット 10 進数を持つすべてのシンタックス・エレメント (たとえば、 3.1 という数値を持つすべてのシンタックス・エレメント) は、相互に排他的な代替エレメントです。 3.1 USERID および 3.1 SYSTEMID という行を聞き取る場合、シンタックスには両方ではなく USERID または SYSTEMID のどちらかが含まれることが分かります。

ドット 10 進レベルは、ネストのレベルを表示します。たとえば、ドット 10 進数 3 のシンタックス・エレメントの後に、一連のドット 10 進数 3.1 のシンタックス・エレメントが続きます。 3.1 の番号が付されたシンタックス・エレメントすべては、番号 3 の付されたシンタックス・エレメントに従属します。

シンタックス・エレメントに関する情報を追加するため、ドット 10 進数の次に特定のワードおよびシンボルが使用されます。 時折、こうしたワードおよびシンボルはエレメントの最初に表示される場合もあります。 簡単に識別するため、ワードやシンボルがシンタックス・エレメントの一部である場合には、円記号 (¥) 文字が先頭に付きます。 * シンボルはドット 10 進数の次に使用でき、シンタックス・エレメントが反復することを示します。 たとえば、ドット 10 進数 3 のシンタックス・エレメント *FILE は、3 ¥* FILE という形式になります。 3* FILE という形式は、シンタックス・エレメント FILE が反復されることを示します。 3* ¥* FILE という形式は、シンタックス・エレメント * FILE が反復されることを示します。

シンタックス・エレメントのストリングを分離するのに使用されるコンマなどの文字は、シンタックス内の分離する項目の直前に表示されます。 こうした文字は、それぞれの項目と同一行に表示するか、同じドット 10 進数を持つ関連する項目のある別の行に表示できます。 またその行には、シンタックス・エレメントに関する情報を提供する別のシンボルを表示することも可能です。 たとえば、複数の LASTRUN および DELETE シンタックス・エレメントを使用している場合には、5.1*、5.1 LASTRUN、および 5.1 DELETE という行は、エレメントをコンマで区切る必要があります。区切り文字が指定されないと、各シンタックス・エレメントを区切るのにブランクが使用されると想定されます。

シンタックス・エレメントの前に % シンボルが付く場合、他の箇所で定義されている参照であることを示します。 % シンボルの後のストリングは、リテラルではなくシンタックス・フラグメントの名前です。 たとえば、2.1 %OP1 という行は別のシンタックス・フラグメント OP1 を参照すべきことを意味します。

以下のワードおよびシンボルが、ドット 10 進数の次に使用されます。

- ? は、オプションのシンタックス・エレメントであることを表します。? シンボルが後に続くドット 10 進数は、対応するドット 10 進数のシンタックス・エレメント、および任意の従属のシンタックス・エレメントがオプションであることを示します。ドット 10 進数の付いたシンタックス・エレメントが 1 つしかない場合、 ? シンボルはそのシンタックス・エレメントと同じ行に表示されます (たとえば、5? NOTIFY)。ドット 10 進数の付いたシンタックス・エレメントが複数ある場合、 ? シンボルだけで行に表示され、その後にオプションのシンタックス・エレメントが続きます。たとえば、「5 ?、5 NOTIFY、および 5 UPDATE」という行を聞き取る場合、シンタックス・エレメント NOTIFY および UPDATE がオプションである、つまりそのいずれかを選択でき、どちらも選択しないこともできることが分かります。 ? シンボルは、線路型ダイアグラムのバイパス線に相当します。
- ! は、デフォルトのシンタックス・エレメントであることを表します。! シンボルおよびシンタックス・エレメントが後に続くドット 10 進数は、そのシンタックス・エレメントが、同じドット 10 進数を共用するシンタックス・エレメントすべてのデフォルト・オプションであることを示します。同じドット 10 進数を共用するシンタックス・エレメントのうち 1 つだけに、 ! シンボルを指定できます。たとえば、「2? FILE、2.1! (KEEP)、および 2.1 (DELETE)」という行を聞き取る場合、 FILE キーワードのデフォルト・オプションは (KEEP) になります。この例では、FILE キーワードを含めてもオプションを指定しない場合には、デフォルト・オプション KEEP が適用されます。デフォルト・オプションは、次に高位のドット 10 進数にも適用されます。 この例の場合、FILE キーワードが省略されると、デフォルトの FILE(KEEP) が使用されます。しかし、「2? FILE、2.1、2.1.1! (KEEP)、および 2.1.1 (DELETE)」という行を聞き取る場合、デフォルト・オプション KEEP は次に高位のドット 10 進数 2.1 (関連キーワードを持っていない) にのみ適用され、 2? FILE には適用されません。キーワード FILE が省略されると、どれも使用されません。
- * は、0 回以上反復できるシンタックス・エレメントを示します。 * シンボルが後に続くドット 10 進数は、このシンタックス・エレメントが 0 回以上使用できること、つまりオプションであり、なおかつ反復できることを表します。たとえば、5.1* データ域という行を聞き取る場合、 1 つまたは複数のデータ域を含めるか、またはデータ域を全く含めないことが可能です。「3*、3 HOST、および 3 STATE」という行を聞き取る場合、 HOST、STATE をどちらか一方または両方同時に含めるか、どちらも含めないことができます。

  注:

  1. ドット 10 進数の後にアスタリスク (*) が付き、ドット 10 進数の付いた項目が 1 つしかない場合には、同じ項目を複数回反復できます。
  2. ドット 10 進数の後にアスタリスクが付き、ドット 10 進数の付いた項目が複数ある場合、リストから複数の項目を使用できますが、各項目を複数回使用することはできません。 前述の例では、HOST STATE と書くことはできますが、HOST HOST とは書けません。
  3. * シンボルは、線路型シンタックス・ダイアグラムのループバック線に相当します。
- + は、1 回以上含める必要のあるシンタックス・エレメントであることを示します。 + シンボルが後に続くドット 10 進数は、このシンタックス・エレメントを 1 回以上含める必要があること、つまり少なくとも 1 回は含める必要があり、反

復できることを表します。たとえば、「6.1+ データ域」という行を聞き取る場合、データ域を少なくとも 1 回は含めなければなりません。「2+、2 HOST、および 2 STATE」という行を聞き取る場合には、 HOST、STATE、またはその両方を含める必要があります。 * シンボルと同様に、+ シンボルは、ドット 10 進数の付いた項目が 1 つしかない場合に限り、その特定の項目のみを反復できます。 * シンボルと同様、 + シンボルは線路型シンタックス・ダイアグラムのループバック線に相当します。

関連概念:

- 823 ページの『アクセス支援』

関連タスク:

- 『キーボード・ショートカットおよびアクセラレーター: Common GUI help』

関連資料:

- viii ページの『構文図の見方』

DB2 Universal Database 製品の共通基準認証

DB2 Universal Database は、 Common Criteria の評価検定レベル 4 (EAL4) で認証の評価を受けています。 Common Criteria の詳細については、以下の Common Criteria の Web サイトを参照してください。 http://niap.nist.gov/cc-scheme/

付録 B. 特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム(本プログラムを含む)との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

[特殊文字]

IBM と連絡をとる

技術上の問題がある場合は、お客様サポートにご連絡ください。

製品情報

DB2 Universal Database 製品に関する情報は、
http://www.ibm.com/software/data/db2/udb から入手できます。

このサイトには、技術ライブラリー、資料の注文方法、製品のダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

米国以外の国で IBM に連絡する方法については、 IBM Worldwide ページ (www.ibm.com/planetwide) にアクセスしてください。

Printed in Japan

SC88-9156-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12