DB2® ユニバーサル・データベース

SQL 解説書 (第 1 巻)

バージョン 7

SC88-8540-00
(英文原典：SC09-2974-00)

DB2® ユニバーサル・データベース

SQL 解説書 (第 1 巻)

バージョン 7

SC88-8540-00
(英文原典：SC09-2974-00)

ご注意！

本書、および本書がサポートする製品をご使用になる前に、 1581ページの『付録S. 特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

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)

原 典：	SC09-2974-00 IBM® DB2® Universal Database SQL Reference, Volume 1 Version 7
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 2000.6

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

第1章 はじめに

この序章では、以下について説明します。

- 本書の目的と対象読者
- 本書の使用法と構成
- 本書全体で使用する構文図の表記法、命名規則、および強調表記の規則
- 関連資料
- 製品ファミリーの概要

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を使う一般ユーザーの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。

本書の使用法

本書は、DB2 ユニバーサル・データベース バージョン 7 によって使用される SQL 言語を定義しています。本書は、リレーショナル・データベースの概念、言語要素、関数、照会の形式、および SQL ステートメントの構文や意味に関する情報についての解説書としてご使用ください。重要な構成要素に関する情報や制限事項については、付録を参照してください。

本書の構成

この解説書は、2 巻に分かれています。第 1 巻には、以下のセクションが含まれています。

- 『第1章 はじめに』では、本書の目的、対象読者、および使用法を説明します。
- 9ページの『第2章 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。

- 71ページの『第3章 言語要素』では、多くの SQL ステートメントに共通する SQL および言語要素の基本的な構文を説明します。
- 231ページの『第4章 関数』では、SQL の列およびスカラー関数の構文図、意味の説明、規則、および使用例について説明します。
- 421ページの『第5章 照会』では、様々な形式の照会について説明します。
- 第 1 巻の付録には、以下の情報が収められています。
  - 1203ページの『付録A. SQL の制限値』では、SQL の制限事項を示します。
  - 1211ページの『付録B. SQL 連絡 (SQLCA)』では、SQLCA の構造を示します。
  - 1217ページの『付録C. SQL 記述子域 (SQLDA)』では、SQLDA の構造を示します。
  - 1231ページの『付録D. カタログ視点』では、データベースのカタログ視点について説明します。
  - 1337ページの『付録E. 構造タイプで使用するカタログ視点』では、データベースの構造タイプのカタログ視点について説明します。
  - 1351ページの『付録F. 連合システム』では、連合システムのオプションとタイプ割り当てについて説明します。
  - 1367ページの『付録G. サンプル・データベース表』には、例として使用するサンプル表を示します。
  - 1389ページの『付録H. 予約スキーマ名と予約語』では、 IBM SQL および ISO/ANS SQL92 標準規格の予約スキーマ名と予約語について示します。
  - 1395ページの『付録I. 分離レベルの比較』では、分離レベルについて要約しています。
  - 1397ページの『付録J. 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。
  - 1401ページの『付録K. Explain 表と定義』では、Explain 表とその定義について説明します。
  - 1437ページの『付録L. 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 コマンドの相互作用について説明します。
  - 1441ページの『付録M. 再帰の例: 部品構成表』では、再帰的照会の例を示します。

– 1447ページの『付録N. 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。
– 1453ページの『付録O. 日本語および繁体字中国語 EUC についての考慮事項』には、EUC 文字セット使用時の考慮事項を示します。
– 1463ページの『付録P. DATALINK での BNF 指定』では、DATALINK の BNF 仕様について説明します。

第 2 巻には、以下のセクションが含まれています。

- 481ページの『第6章 SQL ステートメント』では、すべての SQL ステートメントの構文図、意味の説明、規則、および例について説明します。
- 1157ページの『第7章 SQL プロシージャー』では、SQL プロシージャー・ステートメントの構文図、意味の説明、規則、および例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。

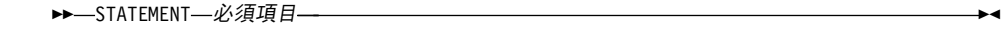
記号 ▸▸── は、ステートメントの始まりを示します。

記号 ──▸ は、ステートメントの構文が次の行に続くことを示します。

記号 ▸── は、ステートメントが前の行から続いていることを示します。

記号 ──▸◂ は、ステートメントの終わりを示します。

必須項目は、横線 (メイン・パス) 上に示されます。



任意選択 (オプション) 項目は、メイン・パスの下に示されます。

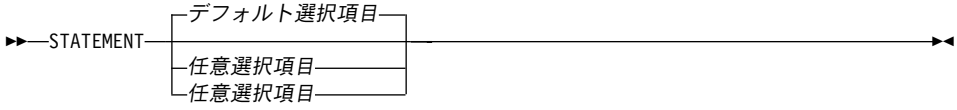
任意選択項目をメイン・パスの上に示すこともありますが、それは構文図を見やすくするためであり、ステートメントの実行には関係しません。

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。

項目も何も選択しなくてもよい場合、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。

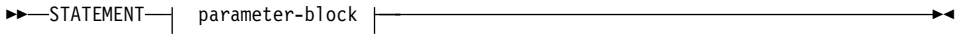
繰り返しの矢印にコンマが示されている場合は、繰り返し項目をコンマで区切らなければなりません。

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。

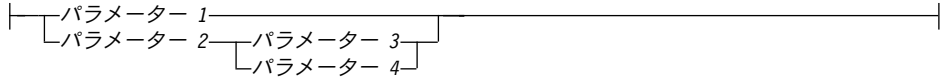
キーワードは英大文字で示してあります (例: FROM)。示されているとおりに入力することが必要です。変数は英小文字で示しています (例: column-name)。このような変数は、構文にユーザーが指定する名前や値を示しています。

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。

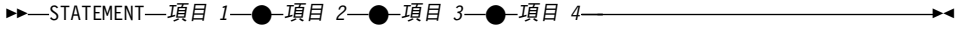
場合によっては、1 つの変数が一連の複数のパラメーターを表すことがあります。たとえば次の図で、変数 parameter-block は、 **parameter-block** という見出しの図の当てはまるもので置き換えることができます。



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、項目2 と 項目3 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。

```
STATEMENT 項目1 項目2 項目3 項目4
STATEMENT 項目1 項目3 項目2 項目4
```

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。たとえば、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 (Boldface)	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。

- *管理の手引き*
  - 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、実現、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の手引き*
  - 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。
- *地理情報エクステンダー 使用者の手引きおよび解説書*
  - 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、および分布などの情報を入手することが関係しています。
- *IBM SQL 解説書*
  - この資料は、データベース製品の IBM ライブラリー全般に共通するすべての SQL 要素について説明しています。 IBM のデータベースを使用し

て可搬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。

第2章 概念

この章では、構造化照会言語 (SQL) で使用される共通の概念について説明します。この章の目的は、高いレベルの視点から諸概念について説明することです。より詳細な内容については、次章以降で説明します。

リレーショナル・データベース

リレーショナル・データベース は、表の集まりとして理解することができるデータベースで、データのリレーショナル・モデルに従って操作することができます。リレーショナル・データベースには、データの保管、管理、およびアクセスに使用される一連のオブジェクトが含まれます。そのようなオブジェクトの例として、表、視点、索引、関数、トリガー、パッケージなどがあります。

*区分 リレーショナル・データベース*とは、複数の区分 (ノードともいう) にわたってデータが管理されるリレーショナル・データベースです。このように複数の区分にまたがってデータを区分化しても、大部分の SQL ステートメントのユーザーには影響ありません。ただし、DDL ステートメントによっては、区分情報が関係する場合があります (たとえば、CREATE NODEGROUP)。

*連合 データベース*とは、データが複数のデータ・ソース (別個のリレーショナル・データベースなど) に格納されるリレーショナル・データベースです。データは、すべて 1 つの大規模なデータベースに含まれているかのように扱うことができ、従来の SQL 照会でアクセスすることができます。データに対する変更は、該当するデータ・ソースへ明示的に送られます。詳細は、46ページの『DB2 連合システム』を参照してください。

構造化照会言語 (SQL)

SQL は、リレーショナル・データベースのデータの定義と操作を行うための標準化された言語です。データのリレーショナル・モデルに従って、データベースは表の集まりとして理解することができ、関係は表の中の各値で表され、データは 1 つまたは複数の基礎表から派生する結果表を指定することによって検索されます。

SQL ステートメントは、データベース・マネージャーによって実行されます。データベース・マネージャーの機能の 1 つは、結果表の仕様を、データ検索を

最適化する一連の内部命令に変換することです。この変換は、準備処理およびバインドの 2 つのフェーズで行われます。

実行可能な SQL ステートメントはすべて、その実行に先立って準備しておく必要があります。その準備の結果は、ステートメントの実行可能形式または操作可能形式です。 SQL ステートメントを準備する方式とその操作可能形式の持続の程度の違いによって、静的 SQL と動的 SQL とがあります。

組み込み SQL

組み込み SQL ステートメントとは、 C などのアプリケーション・プログラミング言語の中に記述され、 SQL プリプロセッサーで前処理される SQL ステートメントのことです。組み込み SQL には、静的と動的の 2 つのタイプがあります。

静的 SQL

静的 SQL ステートメントのソース形式は、 COBOL などのホスト言語で記述されたアプリケーション・プログラムに組み込まれます。そのようなステートメントは、プログラムの実行前に準備され、ステートメントの操作可能形式はプログラムの実行後も存続します。

静的 SQL ステートメントを含むソース・プログラムは、そのコンパイルに先立って SQL プリコンパイラーによって処理する必要があります。このプリコンパイラーは、SQL ステートメントをホスト言語のコメントに変換し、データベース・マネージャーを呼び出すホスト言語ステートメントを生成します。SQL ステートメントの構文は、プリコンパイル処理の過程で検査されます。

SQL アプリケーション・プログラムの準備には、プリコンパイル、宛先データベースへの静的 SQL ステートメントのバインド、修正されたソース・プログラムのコンパイルが含まれます。これらのステップについては、*アプリケーション開発の手引き* で説明されています。

動的 SQL

組み込み動的 SQL ステートメントを含むプログラムも、静的 SQL を含むプログラムと同様にプリコンパイルの必要があります。ただし、静的 SQL とは異なり、動的 SQL ステートメントは実行時に構築され、準備されます。 SQL ステートメントのテキストは、PREPARE と EXECUTE ステートメント、または EXECUTE IMMEDIATE ステートメントのいずれかの使用により、準備され、実行されます。また、SELECT ステートメントの場合は、カーソル操作でも実行可能です。

DB2 コール・レベル・インターフェース (CLI) とオープン・データベース・コネクティビティー (ODBC)

DB2 コール・レベル・インターフェースは、アプリケーション・プログラムに動的 SQL ステートメントを処理する機能を提供するアプリケーション・プログラミング・インターフェースです。 CLI プログラムは、Microsoft または他のベンダーから出されている、オープン・データベース・コネクティビティー (ODBC) ソフトウェア開発者キットを使用してコンパイルすることもできます。これにより、ODBC データ・ソースへアクセスできるようになります。組み込み SQL とは異なり、プリコンパイルは必要ありません。このインターフェースを使用して開発されたアプリケーションは、データベースごとにコンパイルし直すことなく、種々のデータベースに対して実行することができます。このインターフェースにより、アプリケーションは実行時に手続き呼び出しを使用して、データベースへの接続、SQL ステートメントの発行、および戻りデータや状況情報の入手を行うことができます。

DB2 CLI インターフェースには、組み込み SQL では使用できない多くの機能があります。次のような機能が含まれます。

- CLI では、データベース管理システムの DB2 ファミリーを通じて、データベース・システム・カタログ情報を一貫した方法で照会し検索する関数呼び出しが用意されています。これにより、データベース・サーバーごとに合わせたカタログ照会を作成する必要が少なくなります。
- CLI には、カーソルを使った、次のようなスクロール機能があります。
  - 1 行以上の順方向スクロール
  - 1 行以上の逆方向スクロール
  - 最初の行からの 1 行以上の順方向スクロール
  - 最後の行からの 1 行以上の逆方向スクロール
  - カーソルの直前保管位置からのスクロール
- CLI を使用して作成されたアプリケーション・プログラムから呼び出されるストアード・プロシージャーからは、結果セットをプログラムに戻すことができます。

コール・レベル・インターフェースの手引きおよび解説書 には、このインターフェースでサポートされる API が説明されています。

Java データベース・コネクティビティー (JDBC) と Java Embedded SQL (SQLJ) プログラム

DB2 ユニバーサル・データベースは、Java データベース・コネクティビティー (JDBC) と Java Embedded SQL (SQLJ) という、 2 つの標準 Java プログラミング API を実装します。どちらを使用しても、 DB2 にアクセスする Java アプリケーションおよびアプレットを作成することができます。

JDBC 呼び出しは、Java 固有の方式によって DB2 CLI への呼び出しに変換されます。 JDBC は、DB2 CLI を介した DB2 クライアントから DB2 サーバーへの流れを要求します。 JDBC で静的 SQL を使用することはできません。

SQLJ アプリケーションは、データベースへの接続や SQL エラーの処理といったタスクの基盤として JDBC を使用しますが、 SQLJ ソース・ファイルに組み込み静的 SQL ステートメントを含めることもできます。 SQLJ 変換プログラムを使って SQLJ ソース・ファイルを変換してからでないと、生成される Java ソース・コードをコンパイルすることはできません。

JDBC および SQLJ アプリケーションの詳細については、*アプリケーション開発の手引き* を参照してください。

対話式 SQL

対話式 SQL ステートメントは、コマンド行プロセッサーまたはコマンド・センターなどのインターフェースを介して、ユーザーによって入力されます。このようなステートメントは、動的 SQL ステートメントとして処理されます。たとえば、対話式 SELECT ステートメントは、 DECLARE CURSOR、 PREPARE、 DESCRIBE、 OPEN、FETCH、および CLOSE ステートメントを使用して動的に処理することができます。

コマンド行プロセッサーまたはそれに類似する機能や製品を使用して発行できるコマンドについては、 *コマンド解説書* を参照してください。

スキーマ

スキーマ とは、名前を持つオブジェクトの集合です。スキーマは、データベースのオブジェクトを論理的に区分します。スキーマに入れることができるオブジェクトには、表、視点、ニックネーム、トリガー、関数、およびパッケージが含まれます。

スキーマはデータベースのオブジェクトでもあります。スキーマは、ユーザーを所有者として記録する CREATE SCHEMA ステートメントを使用して明示的に作成されます。また、他のオブジェクトを作成する際に、ユーザーが IMPLICIT_SCHEMA 権限を持っている場合には、暗黙的に作成することもできます。

スキーマ名 は、 2 つの部分から成るオブジェクト名の高位の部分として使用されます。スキーマに含まれるオブジェクトは、その作成時点で、スキーマに割り当てられます。オブジェクトが割り当てられるスキーマは、特定のスキーマ名で修飾されている場合にはそのオブジェクトの名前によって、あるいは修飾されていない場合はデフォルトのスキーマ名によって判別されます。

たとえば、DBADM 権限を有するユーザーが、ユーザー A に対して C と呼ばれるスキーマを作成するとします。

```
CREATE SCHEMA C AUTHORIZATION A
```

ユーザー A は、以下のステートメントを出して、スキーマ C に X という名前の表を作成することができます。

```
CREATE TABLE C.X (COL1 INT)
```

スキーマの使用の制御

データベースが作成される場合に、すべてのユーザーが IMPLICIT_SCHEMA 権限を持ちます。これにより、すべてのユーザーが、まだ存在しない任意のスキーマにオブジェクトを作成することができます。暗黙的に作成されたスキーマに対して、どのようなユーザーも、そのスキーマに他のオブジェクトを作成することができます。[1]

IMPLICIT_SCHEMA 権限が PUBLIC から取り消される場合、スキーマは、CREATE SCHEMA ステートメントを使用して明示的に作成されるか、または IMPLICIT_SCHEMA 権限が与えられているユーザー (たとえば、 DBADM 権限のあるユーザー) によって暗黙的に作成されます。 PUBLIC から IMPLICIT_SCHEMA 権限を取り消すことは、スキーマ名の使用に対する制御が増すと同時に、オブジェクトの作成を試みる時に既存のアプリケーションで許可エラーが生じる可能性があります。

また、スキーマに関連する特権には、どのようなユーザーがスキーマ中のオブジェクトを作成、更新、および除去する権限をもつかを制御するものもありま

1. 暗黙的に作成されるスキーマについてのデフォルトの特権には、旧バージョンとの上位互換性があります。別名、特殊タイプ、関数、およびトリガーの作成は、暗黙的に作成されるスキーマにまで拡張されます。

す。当初、スキーマの所有者にはスキーマに関するこれらのすべての特権が与えられ、それらの特権を他のユーザーに付与することもできます。暗黙的に作成されたスキーマはシステムによって所有され、当初、そのようなスキーマにオブジェクトを作成する権限がすべてのユーザーに与えられます。 DBADM または SYSADM 権限を有するユーザーは、任意のスキーマでユーザーが保持する特権を変更することができます。したがって、任意のスキーマ (暗黙的に作成されたスキーマであっても) のオブジェクトを作成、更新、および除去するためのアクセスを制御することができます。

表

表は、データベース・マネージャーが管理する論理構造です。表は列と行で構成されます。表の内部で行の順序を整えることは必要ではありません (行の順序はアプリケーション・プログラムが決定します)。 1 つの列と 1 つの行が交差する箇所は、*値* と呼ばれる特定のデータ項目です。 *列* は、同じタイプ、あるいはそのサブタイプの 1 つの値の集まりです。*行* は、その *n* 番目の値が表の *n* 番目の列の値であるような値の並びです。

基礎表 は、 CREATE TABLE ステートメントを使用して作成され、永続的なユーザー・データを保持するのに使用されます。*結果表* は、照会の要求に応じるためにデータベース・マネージャーが 1 つ以上の基礎表から選択または生成した行の集まりです。

要約表は、照会によって定義される表です。この照会は、要約表のデータを判別するのにも使用されます。要約表を使用すると、照会のパフォーマンスが向上します。要約表を使用して照会の一部を解決することができると判別した場合、データベース・マネージャーは、要約表を使用するようその照会を書き換えることがあります。この決定は、 CURRENT REFRESH AGE および CURRENT QUERY OPTIMIZATION 特殊レジスターなどの特定の設定に基づいてなされます。

表では、列ごとにデータ・タイプを個別に定義する (つまり、列のタイプをユーザー定義の構造タイプの属性に基づいたものにする) ことができます。このような表は、*タイプ付き表* と呼ばれます。ユーザー定義の構造タイプは、タイプ階層の一部にすることができます。*サブタイプ* は、*スーパータイプ* から属性を継承します。同様に、タイプ付き表はタイプ階層の一部にすることができます。*副表* は、*スーパー表* から列を継承します。*サブタイプ* という用語は、タイプ階層において 1 つのユーザー定義の構造タイプおよびその下にあるすべてのユーザー定義の構造タイプを指して用いられることに注意してください。構造タイプ T の*厳密な意味でのサブタイプ* とは、タイプ階層で T の下にある構造タイプのことです。同様に*副表* という用語は、表階層において 1 つの

タイプ付き表およびその下にあるすべてのタイプ付き表を指して用いられます。表 T の*厳密な意味での副表* とは、表階層において T の下にある表のことです。

宣言済み一時表 は、DECLARE GLOBAL TEMPORARY TABLE ステートメントで作成され、 1 つのアプリケーションのために一時データを保持するときに使います。この表は、アプリケーションがデータベースから切断されるときに、暗黙的に除去されます。

視点

視点 は、 1 つまたは複数の表にあるデータを見るためのもう 1 つの方法を提供するものです。

視点は、結果表に名前を付けて指定したものです。その指定は、SQL ステートメントで視点が参照されるたびに実行される SELECT ステートメントです。したがって、視点は基礎表と同じく列と行をもつものとして考えることができます。検索作業では、すべての視点を基礎表とまったく同じように使用することができます。 CREATE VIEW の説明の中で示されているように、挿入、更新、または削除の操作に視点を使用できるかどうかは、その定義によって異なります。 (詳細については、895ページの『CREATE VIEW』を参照してください。)

視点の列が基礎表の列から直接に派生している場合、その列は基礎表の列に適用されるあらゆる制約を継承します。たとえば、視点にその基礎表の外部キーが含まれている場合は、その視点を使用する INSERT および UPDATE 操作は基礎表と同じ参照制約に従います。また、視点の基礎表が親表である場合、その視点を使用する DELETE および UPDATE 操作は、基礎表の DELETE および UPDATE 操作と同じ規則に従います。

視点では、列ごとにデータ・タイプを結果表から派生させる (つまり、列のタイプをユーザー定義の構造タイプの属性に基づいたものにする) ことができます。このような視点を、*タイプ付き視点* といいます。タイプ付き表と同様に、タイプ付き視点は視点階層の一部にすることができます。*副視点* は、*スーパー視点* から列を継承します。*副視点* という用語は、視点階層において 1 つのタイプ付き視点およびその下にあるすべてのタイプ付き視点を指して用いられます。視点 V の*厳密な意味での副視点* とは、タイプ付き視点階層で V の下にある視点のことです。

視点は操作不能になることがあります。その場合、それ以後 SQL ステートメントで使用することはできません。

別名

別名 とは、表または視点の代替名のことです。別名は、既存の表や視点を参照できる場所で、表や視点を参照するために使用できます。[2] 表や視点と同様に、別名は作成や除去が可能であり、コメントを付けることもできます。ニックネームに別名を作成することもできます。ただし、表とは異なり、連鎖と呼ばれるプロセスの中で相互に参照し合うことが可能です。別名は広く参照される名前であり、このため別名の使用には特別な許可や特権などは必要ありません。しかし、別名が参照する表や視点にアクセスするには、やはり現行の文脈に応じた適切な権限が必要です。

表の別名のほかにも、データベース別名やネットワーク別名など各種の別名があります。

別名の詳細については、79ページの『別名』および 608ページの『CREATE ALIAS』を参照してください。

索引

索引 とは、基礎表の行へのポインターに順序を付けた集合のことです。それぞれの索引は、1 つ以上の表列のデータ値に基づいています。索引は、表のデータとは別個の 1 つのオブジェクトです。索引を作成すると、データベース・マネージャーによって、自動的に索引の構造が作成されて保守されます。

索引は、以下の目的でデータベース・マネージャーが使用します。

- パフォーマンスの改善。ほとんどの場合、索引のある方がデータへのアクセスが速くなります。

  視点に対して索引を作成することはできません。しかし、視点の元の表に索引を作成すると、視点の操作のパフォーマンスが向上する可能性があります。
- 固有性の保証。固有索引をもつ表では、複数の行のキーを同じにすることができません。

キー

キー は、特定の行または行の集まりの識別やアクセスのために使用する列の集合です。キーは、表、索引、または参照制約の記述において指定されます。同じ列が複数のキーを構成することも可能です。

2. 別名はあらゆる文脈で使用できるわけではありません。たとえば、検査制約の検査条件の中では使用できません。同様に、別名で宣言済み一時表を参照することはできません。

2 つ以上の列から成るキーは、*複合キー* と呼ばれます。複合キーのある表では、複合キー内の列の順序は表内の列の順序の制約を受けません。複合キーと関連して使われるとき、*値* という用語は、複合値を指します。したがって、「外部キーの値は基本キーの値に等しくなければならない」という規則の場合、外部キーの値を構成する各要素が、基本キーの値を構成する要素のうちのそれぞれ対応するものと等しくなければならないことを意味します。

固有キー

固有キー は、キーのどのような 2 つの値も等しい値であってはならないという制約のあるキーです。固有キーの列の内容を NULL 値にすることはできません。この制約は、データ値を変更する任意の操作 (たとえば、 INSERT や UPDATE など) の実行の過程でデータベース・マネージャーによって課せられます。制約を課すために使用されるメカニズムは、*固有索引* と呼ばれます。つまり、固有キーは固有索引のキーであるということです。このような索引は UNIQUE 属性があるとも言われます。詳細については、18ページの『固有制約』を参照してください。

基本キー

基本キー は、固有キーの特殊な場合です。表は、複数の基本キーを持つことはできません。詳細については、『固有キー』を参照してください。

外部キー

外部キー とは、参照制約の定義で指定されているキーです。詳細については、19ページの『参照制約』を参照してください。

区分化キー

区分化キー とは、区分データベースの表の定義の一部であるキーです。区分化キーは、データの行が保管される区分を判別するのに使用されます。区分化キーが定義される場合、固有キーおよび基本キーには、区分化キーと同じ列が含まれていなければなりません (固有キーと基本キーにはこの他にも列が含まれている場合があります)。表は、複数の区分化キーを持つことはできません。

制約

制約 とは、データベース・マネージャーによって課せられる規則のことです。

制約には次の 3 つのタイプがあります。

- *固有制約*。これは、表の 1 つまたは複数の列に重複する値を指定することを禁止する規則です。固有キーおよび基本キーは、サポートされている固有制

約です。たとえば、固有制約を納入業者の表の納入業者識別子に定義すると、同一の納入業者識別子を 2 つの納入業者に指定することを避けることができます。

- *参照制約*。これは 1 つまたは複数の表の 1 つまたは複数の列の値に関する論理的な規則です。たとえば、ある表の集合で、納入業者に関する情報を共用している場合に、納入業者の名前が変わることがあります。参照制約を定義して、表の納入業者の ID が納入業者情報の納入業者 ID と一致するように指定することができます。この制約は、納入業者情報の脱落を招く挿入、更新、または削除を防止します。
- *表検査制約* は、特定の表に追加されるデータに対して制限を設定するものです。たとえば、従業員情報を含む表に給与データを追加もしくは更新する際に、従業員の給与レベルが $20,000.00 以上であることを定義することができます。

参照制約および表検査制約はオン / オフの切り換えが可能です。一般に、大量のデータをデータベースにロードする場合には、制約の実施検査をオフにします。制約の設定の切り換えについては、1113ページの『SET INTEGRITY』 を参照してください。

固有制約

固有制約 は、表内でその値が固有である場合にのみキーの値を有効とする規則です。固有制約は任意選択であり、 PRIMARY KEY 文節または UNIQUE 文節を使用して、 CREATE TABLE または ALTER TABLE ステートメントで定義することができます。固有制約に指定される列は、NOT NULL として定義する必要があります。固有索引は、固有制約の列の変更時にキーの固有性を実現するためにデータベース・マネージャーによって使用されます。

表には任意の数の固有制約を指定することができますが、基本キーとして定義されるのは最大 1 つの固有制約です。表には、同一の一連の列に複数の固有制約を指定することはできません。

参照制約の外部キーによって参照される固有制約は、*親キー* と呼ばれます。

CREATE TABLE ステートメントで固有制約が定義される場合に、固有索引がデータベース・マネージャーによって自動的に作成され、システムに必須の基本索引または固有索引として指定されます。

固有制約が ALTER TABLE ステートメントで定義され、しかも同一列に索引が存在する場合、その索引は固有かつシステム必須として指定されます。この

ような索引が存在しない場合、固有索引がデータベース・マネージャーによって自動的に作成され、システムに必須の基本索引または固有索引として指定されます。

固有制約の定義と固有索引の作成とは明らかに異なることに注意してください。いずれも固有性を実現しますが、固有索引はヌル可能列を使用することができ、通常、親キーとして使用することはできません。

参照制約

参照保全 とは、すべての外部キーのすべての値が有効であるデータベースの状態のことです。*外部キー* とは、その値が親表の行の基本キーまたは固有キーの少なくとも 1 つの値と一致する必要がある表内の列または列の集合です。*参照制約* は、外部キーの値を以下の場合にのみ有効とする規則です。

- 外部キーの値が親キーの値として現れる場合、または
- 外部キーの構成要素の一部がヌル値の場合

親キーを含む表は参照制約の *親表* と呼ばれ、外部キーを含む表はその表の *従属表* と呼ばれます。

参照制約はオプションであり、 CREATE TABLE ステートメントおよび ALTER TABLE ステートメントで定義することができます。参照制約は、データベース・マネージャーによって、 INSERT、UPDATE、DELETE、ALTER TABLE ADD CONSTRAINT、および SET INTEGRITY ステートメントの実行の過程で課せられます。これが有効に実施されるのは、ステートメントの完了時です。

RESTRICT の削除または更新規則を伴う参照制約は、他のすべての参照制約より先に課せられます。 NO ACTION の削除または更新規則を伴う参照制約は、ほとんどの場合、RESTRICT と同じように動作します。ただし、特定の SQL ステートメントでは異なる場合があります。

参照保全、検査制約、およびトリガーは実行時に結合されることがあります。これらの要素の結合については、1397ページの『付録J. トリガーと制約の相互作用』を参照してください。

参照保全の規則に関係した概念および用語には、次のものがあります。

親キー 参照制約の基本キーまたは固有キー。

親行 少なくとも 1 つの従属行をもつ行。

親表 参照制約の親キーを含む表。表は、任意の数の参照制約に関して親として定義することができ

ます。ある参照制約で親となっている表が、他の参照制約の従属となる場合もあります。

従属表
最低限 1 つの参照制約が定義に含まれている表。表は、任意の数の参照制約に関して従属として定義することができます。ある参照制約で従属となっている表が、他の参照制約の親となる場合もあります。

下層表
ある表が T の従属であるか、または T の従属の下層である場合、その表は表 T の下層です。

従属行
少なくとも 1 つの親行をもつ行。

下層行
ある行が行 p の子孫であるのは、その行が p の従属行である場合、または p の従属行の子孫である場合です。

参照サイクル
参照制約の集合で、その集合内のそれぞれの表がそれ自身の下層であるもの。

自己参照行
自分自身の親である行。

自己参照表
同じ参照制約の中で親であり従属でもある表。この場合の制約は*自己参照制約* と呼ばれます。

挿入規則

参照制約の挿入規則は、外部キーのヌル値以外の挿入値が親表の親キーの何らかの値と一致していなければならない、というものです。複合外部キーの場合、値がヌル値であるのは、その構成要素のどれかがヌル値である場合です。この規則が暗黙的であるのは、外部キーが指定されている場合です。

更新規則

参照制約の更新規則は、参照制約の定義時に指定されます。選択項目には NO ACTION と RESTRICT があります。更新規則は、親の行または従属表の行が更新されるときに適用されます。

親行の場合、親キーの列の値が更新されると、次のようになります。

- 従属表の行がキーの元の値と一致し、しかも更新規則が RESTRICT である場合には、その更新は拒否されます。

- 更新ステートメントの完了時に (トリガー後を除く)、従属表の行に対応する親キーがなく、しかも更新規則が NO ACTION である場合には、その更新は拒否されます。

従属行の場合、外部キーが指定されると、暗黙的な更新規則は NO ACTION です。 NO ACTION は、更新ステートメントの完了時に、外部キーのヌル値以外の更新値が親表の親キーの何らかの値に一致していなければならないことを意味します。

複合外部キーの場合、値がヌル値であるのは、その構成要素のどれかがヌル値である場合です。

削除規則

参照制約の削除規則は、参照制約の定義時に指定されます。選択項目としては、NO ACTION、RESTRICT、CASCADE、または SET NULL があります。SET NULL を指定できるのは、外部キーの列の中にヌル値が可能なものがある場合だけです。

参照制約の削除規則は、親表の行が削除されるときに適用されます。もっと正確に言い換えれば、この規則は、親表の行が削除または (後で定義する) 伝送した削除操作の対象となっており、参照制約の従属表の中にその行の従属行がある場合に適用されます。たとえば、P が親表、D が従属表、そして p が削除または伝送削除操作の対象である親行を指すものとします。この場合の削除規則は以下のとおりです。

- 削除規則が RESTRICT または NO ACTION の場合、エラーとなり行は削除されません。
- CASCADE の場合、削除規則は D 内の p の従属行へ伝送します。
- 削除規則が SET NULL の場合、D にある p の各従属行の外部キーのうち、ヌル値可の各列はヌル値に設定されます。

表が親となっている参照制約にはそれぞれ独自の削除規則があり、適用可能な削除規則のすべてを使用して削除操作の結果が決められます。したがって、ある行の従属行が削除規則 RESTRICT か NO ACTION の参照制約になっている場合、または、削除規則 RESTRICT か NO ACTION の参照制約の従属である子孫のどれかに対して削除がカスケードしている場合、その行を削除することはできません。

親表 P から行を削除すると、他の表もこの操作に関係し、その表の行に以下のような影響が出る場合があります。

- D が P の従属表で削除規則が RESTRICT か NO ACTION の場合、 D はこの操作に関係しますが、操作の影響は受けません。
- D が P の従属表で削除規則が SET NULL の場合、D はこの操作に関係し、操作中に D の行が更新される場合があります。
- D が P の従属表で削除規則が CASCADE の場合、D はこの操作に関係し、操作中に D の行が削除される可能性があります。

  D の行が削除される場合、P の削除操作が D に伝送する、といいます。 D が親表でもある場合は、このリストで説明した動作は D の従属表にも順に適用されていきます。

P での削除操作に関係する可能性のある表は、 P に*連結削除* されている、と言われます。 したがって、ある表が P の従属表であるか、または P からの削除操作のカスケード先の表の従属表である場合、この表は表 P に対して連結削除されることになります。

表検査制約

表検査制約 とは、表の各行の 1 つ以上の列について可能な値を指定する規則のことです。これらは任意選択であり、 SQL ステートメントの CREATE TABLE および ALTER TABLE を使用して定義されます。表検査制約は、ある制限された形の探索条件で指定されます。その制限の 1 つは、表 *T* の表検査制約に含まれる列名は *T* の列を指定したものでなければならない、ということです。

表には、任意の数の表検査制約を指定することができます。表検査制約が実施されるのは以下の場合です。
- 表に行が挿入されるとき
- 表の行が更新されるとき

表検査制約は、挿入または更新される各行に対してその探索条件を適用することによって実施されます。探索条件の結果が行のどれかに対して偽となった場合は、エラーになります。

既存データの含まれている表に対する ALTER TABLE ステートメントの中に表検査制約が定義されている場合、その ALTER TABLE ステートメントの実行に先立って、その既存のデータが新しい条件に基づいて検査されます。表を*検査保留状態* にすることができます。検査保留状態では、データを検査しなくても ALTER TABLE ステートメントを実行することができます。表を検査保

留状態にするには、SET INTEGRITY ステートメントを使用します。 SET CONSTRAINT ステートメントは、制約に基づく各行の検査を再開するときにも使用されます。

トリガー

トリガー とは、特定の表での削除、挿入、または更新操作時に実行される処置、またはそれらの操作によって引き起こされる処置の集まりを定義するものです。このような SQL 操作が実行されるとき、トリガーが*起動される* といいます。

トリガーは、データ保全規則を実施するために、参照制約および検査制約と共に使用することができます。また、他の表を更新したり、挿入行または更新行の値を自動的に生成または変換したり、警告を出すなどのタスクを実行する関数を呼び出したりする場合にも使用できます。

トリガーは、データの異なる様々な状態が関係した規則である*遷移* 業務規則を定義し実施するのに便利なメカニズムです (たとえば、給与は 10 パーセントを超えて増加することはない、など)。データの複数の状態が関係しない規則については、検査制約と参照保全制約を考慮してください。

トリガーを使用すると、業務規則を実施する論理をデータベースの中に組み込むことができ、表を使用するアプリケーション側で業務規則を実施する必要がなくなります。すべての表に対して実施する論理を集中管理することによって、論理が変わってもアプリケーション・プログラムを変更する必要がないため、保守が容易になります。

トリガーは任意選択であり、CREATE TRIGGER ステートメントで定義されます。

トリガーの作成時には、トリガーをいつ起動するかを決めるために使われる基準を定義します。

- *対象表* は、トリガーが定義される表を定義します。
- *トリガー事象* は、対象表を変更する特定の SQL 操作を定義します。そのような操作として、削除、挿入、更新があります。
- *トリガー起動時* は、トリガー事象を起動するのが、トリガー事象が対象表で実行される前か後かを定義します。

トリガーを起動するステートメントには、*影響を受ける行の集合* を含めます。それらは、対象表の中で削除、挿入、または更新される行です。 *トリガー細分*

性 は、トリガーの処置がステートメントに対して 1 回実行されるのか、影響を受ける行集合の中の行ごとに 1 回ずつ実行されるのかを定義します。

トリガー・アクション は、任意選択の探索条件、およびトリガーが起動されると必ず実行される SQL ステートメントの集合で構成されます。この SQL ステートメントは、探索条件が真の場合にのみ実行されます。トリガー起動時がトリガー事象の前である場合、トリガー・アクションには、選択、変換変数の設定、SQLSTATE の通知を行うステートメントを組み込むことができます。トリガー起動時がトリガー事象の後である場合、トリガー・アクションには、選択、更新、挿入、削除、SQLSTATE の通知を行うステートメントを組み込むことができます。

トリガー・アクションでは、影響を受ける行の集合の中の値を参照することもできます。これは、*変換変数* の使用によりサポートされます。変換変数では、対象表の列の名前を、参照が (更新の前の) 古い値に対するものか (更新の後の) 新しい値に対するものかを識別する、所定の名前によって修飾したものを使用します。更新トリガーまたは挿入トリガーの前に、 SET 変換変数ステートメントを使って新しい値を変更することもできます。影響を受ける行の集合内の値を参照するもう 1 つの方法は、*変換表* を使用することです。変換表も対象表の列の名前を使用しますが、影響を受ける行の集合全体が 1 つの表として扱われるように名前が指定されます。変換表は、トリガー後に限り使用することができます。さらに、古い値と新しい値について、個別の変換表を定義することができます。

表、事象、または起動時の 1 つの組み合わせに対して、複数のトリガーを指定することができます。トリガーは作成順に起動されます。このため、最後に作成されたトリガーは最後に起動されることになります。

トリガーの起動により、*トリガー・カスケード* が発生する場合があります。これは、1 つのトリガーの起動によって、他のトリガーや同じトリガーを再度起動する SQL ステートメントが実行される場合に発生します。トリガー・アクションは、元々の変更の結果として更新操作を引き起こしたり、別のトリガーを起動する可能性のある参照保全削除規則の結果として、更新操作を引き起こしたりする場合もあります。トリガー・カスケードが発生すると、トリガーと参照保全削除規則とのかなり大きな連鎖が起動されて、 1 回の削除、挿入、または更新ステートメントの結果として、データベースにかなりの変更が加えられる場合があります。

事象モニター

事象モニター は、ある事象の結果としての特定のデータを追跡します。たとえば、データベースを始動するという事象によって、事象モニターに、そのデータベースを使用する許可 ID の定期スナップショットを取って、システム上のユーザー数を追跡させることができます。

事象モニターの起動と非活動化の切り換えは、 1 つのステートメント (SET EVENT MONITOR STATE) によって行われます。事象モニターの現在の状態、すなわち活動状態かどうかを調べるには、関数 (EVENT_MON_STATE) を使います。

照会

照会 は、 (一時的な) 結果表を指定するための特定の SQL ステートメントからなる構成要素です。

表式

表式 は、単純な照会から (一時的な) 結果表を作成します。文節を使うと、その結果表がさらに詳細なものになります。たとえば、表式は、複数の部門からすべての管理者を選択して、さらに管理者が 15 年以上の実務経験があり、ニューヨーク支社に配属されていることを指定する照会にすることができます。

共通表式

共通表式 は、複雑な照会内の一時視点のようなものであり、照会内の他の場所で参照することができます。たとえば、視点を作成しないように視点の代わりに参照することができます。複雑な照会の中で特定の共通表式を使用する場合、それぞれが同じ一時視点を共用することになります。

1 つの照会の中で 1 つの共通表式を再帰的に使用することにより、部品構成表 (BOM)、航空便予約システム、ネットワーク・プランニングなどのアプリケーションのサポートのために利用できます。 1441ページの『付録M. 再帰の例: 部品構成表』に、BOM アプリケーションの事例集が載せられています。

パッケージ

パッケージ とは、単一のソース・ファイル内にあるすべてのセクション が入ったオブジェクトのことです。セクションは、SQL ステートメントの形式でコンパイルされます。セクションは必ず 1 つのステートメントに対応しますが、ステートメントにはセクションが含まれていなくても構いません。静的 SQL

用に作成されたセクションは、バインド形式または操作可能形式の SQL ステートメントとして扱うことができます。動的 SQL 用に作成されたセクションは、実行時に使用されるプレースホルダー制御構造として扱うことができます。パッケージは、プログラムの作成処理中に生成されます。パッケージの詳細については、*アプリケーション開発の手引き* を参照してください。

カタログ視点

データベース・マネージャーは、その制御下のデータに関する情報の含まれる一連の視点と基礎表を管理しています。このような視点と基礎表を、まとめて *カタログ* と呼びます。これには、表、視点、索引、パッケージ、関数などの、データベース中のオブジェクトの情報が納められています。

カタログ視点は、データベースの他の視点とよく似たものです。カタログ視点のデータを参照するには、システムの他の視点からデータを検索するのと同じ方法で SQL ステートメントを使用することができます。データベース・マネージャーは、データベース中のオブジェクトの正確な内容を常にカタログに記録しています。カタログの特定の値を変更する場合には、更新可能なカタログ視点の集合を使用できます (1232ページの『更新可能なカタログ視点』 を参照)。

カタログには統計的な情報も含まれています。この統計的な情報は、管理者の実行するユーティリティー、または適切な権限を与えられたユーザーの実行する更新ステートメントによって更新されます。

カタログ視点のリストが、1231ページの『付録D. カタログ視点』に示されています。

アプリケーションのプロセス、並行性、および回復

すべての SQL プログラムは、*アプリケーション・プロセス* またはエージェントの一部として実行されます。 アプリケーション・プロセスには、1 つ以上のプログラムの実行が関係しており、データベース・マネージャーが資源を割り当てたりロックしたりする場合の単位となります。異なるいくつかのプログラムの実行、または同じプログラムの複数の異なる実行には、異なる複数のアプリケーション・プロセスが関係しています。

同時に複数のアプリケーション・プロセスが同じデータへのアクセスを要求することがあります。 このような状況でデータ保全性を維持するためのメカニズムとして*ロック* があります。これは、たとえば 2 つのアプリケーション・プロセスが同時にデータの同じ行を更新するのを防ぐ、などの処理を行います。

データベース・マネージャーは、あるアプリケーション・プログラムが行った変更でまだコミットされていないものが、誤って他に認識されることのないよう、ロックを獲得します。プロセスが終了すると、データベース・マネージャーは、アプリケーション・プロセスのためにデータベース・マネージャーが獲得し保持していたロックをすべて解放します。もっと早い時期にロックを解放するには、アプリケーション・プロセス自体で明示的に要求する必要があります。この操作はコミット と呼ばれ、これにより作業単位中に獲得していたロックが解放され、作業単位中に加えられた変更がデータベースにコミットされます。

データベース・マネージャーには、アプリケーション・プロセスが行った変更で、まだコミットされていないものを*取り消す* 手段が用意されています。これは、アプリケーション・プロセスの一部に障害が発生したとき、またはデッド*ロック* やロック・タイムアウト状態などで必要になります。ただし、アプリケーション・プロセス自体でも、自分の行ったデータベースへの変更を取り消すように明示的に要求することができます。この操作は*ロールバック* と呼ばれます。

作業単位 とは、アプリケーション・プロセス内の、回復可能な一連の操作のことです。作業単位は、アプリケーション・プロセスの開始時に開始されます。[3] また、アプリケーションの終了以外の何らかの理由によって直前の作業単位が終了したときにも、作業単位が開始します。作業単位は、コミット操作、ロールバック操作、またはアプリケーション・プロセスの終了によって終了します。コミットまたはロールバック操作は、それによって終了する作業単位の中で行われたデータベースへの変更内容にしか影響しません。

このような変更がコミットされないまま残っている間は、他のアプリケーション・プロセスはそれらの変更を認識することはできません。コミットせずに変更をバックアウトすることも可能です。[4] データベースの変更内容がコミットされると、他のアプリケーション・プロセスからその変更内容にアクセスできるようになり、ロールバックによってバックアウトすることはできなくなります。

データベース・マネージャーがアプリケーション・プロセスのために獲得したロックは、作業単位が終了するまで保持されます。この規則の例外は、カーソ

---

3. DB2 CLI および組み込み SQL は、*並行トランザクション* と呼ばれる接続モードをサポートします。これは、それぞれが独立したトランザクションである複数の接続をサポートします。 1 つのアプリケーションが同じデータベースに対して複数の接続を並行して行うことができます。複数のスレッドによるデータベース・アクセスの詳細については、 *アプリケーション開発の手引き* を参照してください。

4. 32ページの『非コミット読み取り (UR)』で説明されている未非コミット読み取りの分離レベルの場合を除きます。

ル固定分離レベル (この場合は、カーソルが行から行へ移動するときにロックが解放されます) と、非コミット読み取りレベル (この場合は、ロックは獲得されません) です (29ページの『分離レベル』 を参照)。

作業単位の開始と終了によって、アプリケーション・プロセス内の一貫性ポイントが定義されます。たとえば、銀行業務のトランザクションで、ある口座から別の口座へ資金を移動することがあるかもしれません。

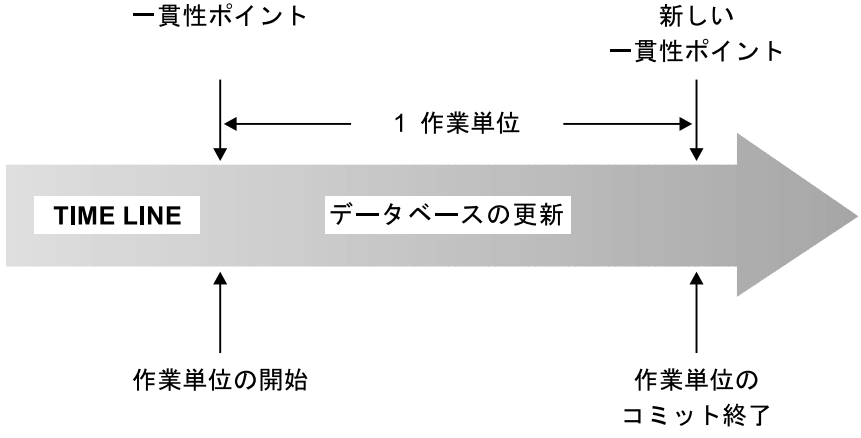


図 1. コミット・ステートメントの作業単位

このようなトランザクションでは、まずその資金を初めの口座から減算してから、第 2 の口座にそれを加算する、という処理が必要です。

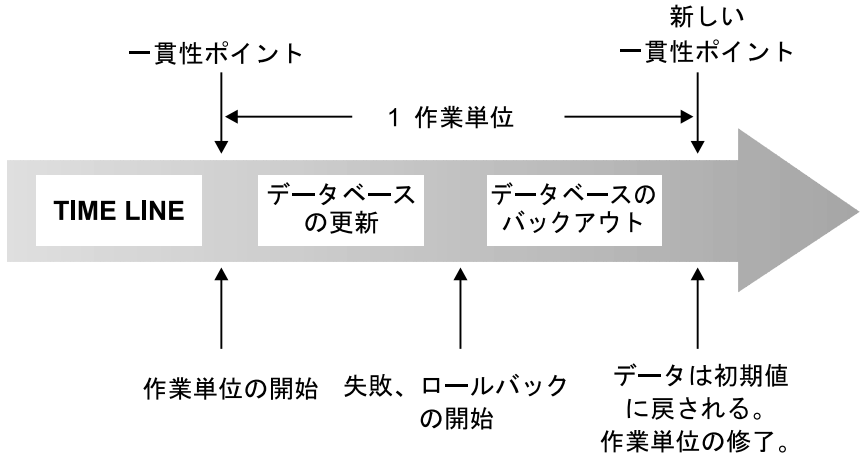


図2. ロールバック・ステートメントの作業単位

減算のステップの直後の段階では、データに矛盾が生じています。資金が第 2 の口座に加算して初めて、一貫性が取り戻されるわけです。両方のステップが完了したときに、コミット操作を実行して作業単位を終了させるなら、他のアプリケーション・プロセスが変更内容を利用できるようになります。

1 つの作業単位が終わる前に障害が発生すると、データベース・マネージャーはコミットされていない変更内容をロールバックし、その作業単位の開始時点でのデータ一貫性を復元します。

注: アプリケーション・プロセスが自分自身のロックのために操作できなくなるということは決してありません。[5]

分離レベル

アプリケーション・プロセスに関連する*分離レベル* は、並行して実行している他のアプリケーション・プロセスからそのアプリケーション・プロセスを分離する度合いを定義します。したがって、アプリケーション・プロセスの分離レベル P は、以下を指定します。

- P によって読み取られ更新される行を、並行して実行される他のアプリケーション・プロセスから使用できる度合い。

5. アプリケーションが並行してトランザクションを使用している場合、一方のトランザクションによるロックのために、他方のトランザクションの運用が影響を受ける可能性があります。詳細については、*アプリケーション開発の手引き* を参照してください。

- 並行して実行される他のアプリケーション・プロセスの更新活動によって P が影響を受ける度合い。

分離レベルは、パッケージの属性として指定され、そのパッケージを使用するアプリケーション・プロセスに適用されます。分離レベルは、プログラム準備処理で指定されます。これによって、並行アプリケーション・プロセスによるデータ・アクセスは、ロックの種類に応じて制限または禁止されます。各種ロックのタイプおよび属性の詳細については、*管理の手引き* を参照してください。宣言済み一時表と宣言済み一時表の行は、一時表を宣言したアプリケーションによってのみアクセス可能なものなので、ロックされることはありません。したがって、ロックおよび分離レベルに関する以下の説明は、宣言済み一時表には当てはまりません。

データベース・マネージャーでは、大きく分けて次の 3 つのロックがサポートされています。

共用
並行アプリケーション・プロセスの操作を、読み取り専用のデータ操作だけに制限します。

更新
並行アプリケーション・プロセスが行の更新を宣言したのでない限り、データへの読み取り専用操作に限定されます。データベース・マネージャーは、現在その行を見ているプロセスは行を更新するものとみなします。

排他
並行アプリケーション・プロセスがどのような方法であってもデータにアクセスしないようにします。ただし、データの読み取りは可能であるが変更はできない*非コミット読み取り* の分離レベルのアプリケーション・プロセスを除きます。 (32ページの『非コミット読み取り (UR)』を参照。)

ロックは基礎表の行について行われます。しかし、データベース・マネージャーが複数の行ロックを単一の表ロックに置き換えることがあります。これを*ロック・エスカレーション* といいます。アプリケーション・プロセスには、少なくとも最低要求されるロック・レベルが保証されます。

DB2 ユニバーサル・データベース データベース・マネージャーは、4 つの分離レベルをサポートしています。分離レベルとは関係なく、データベース・マネージャーは、挿入、更新、または削除の対象となる行のすべてに排他ロックをかけます。このため、どの分離レベルでも、このアプリケーション・プロセ

スが 1 作業単位中に変更する行は、その作業単位が終了するまで他のアプリケーション・プロセスから変更されないことが保証されます。分離レベルには、以下のものがあります。

反復可能読み取り (RR)

レベル RR では次のようになります。

- 作業単位中に読み取られる行は[6]、その作業単位が完了するまで他のアプリケーションによって変更されることがありません。[7]
- 他のアプリケーション・プロセスによって変更される行は、そのアプリケーション・プロセスがコミットするまで読み取ることができません。

RR では、単独読み取り行 (「読み取り固定」の説明を参照) の出現は許されません。

RR レベルで実行しているアプリケーション・プロセスは、排他ロック以外に、参照するすべての行に対して共有ロックを獲得します。さらに、アプリケーション・プロセスが並行アプリケーション・プロセスの影響から完全に分離されるようにロックが実行されます。

読み取り固定 (RS)

RR レベルと同様に、RS レベルでは、以下が保証されます。

- 作業単位中に読み取られる行は[6]、その作業単位が完了するまで他のアプリケーションによって変更されることがありません。[8]
- 他のアプリケーション・プロセスによって変更される行は、そのアプリケーション・プロセスがコミットするまで読み取ることができません。

RR とは異なり RS では、アプリケーション・プロセスは、他の並行アプリケーション・プロセスの影響から完全には分離されません。 RS レベルでは、同じ照会を複数回発行するアプリケーション・プロセスで行が追加されていくという場合があります。このような付加行は*単独読み取り行* と呼ばれます。

単独読み取り行は、たとえば次のような状況で発生します。

---

6. 行は、対応する OPEN ステートメントと同じ作業単位で読み取られなければなりません。 914ページの『DECLARE CURSOR』 の WITH HOLD の説明を参照してください。

7. CLOSE ステートメントに任意選択の WITH RELEASE 文節を使用すると、反復不能読み取りおよび単独読み取りに対する保証は、カーソルが再オープンされる場合、以前にアクセスされた行には適用されなくなります。

8. CLOSE ステートメントに任意選択の WITH RELEASE 文節を使用すると、反復不能読み取りに対する保証は、カーソルが再オープンされる場合、以前にアクセスした行には適用されなくなります。

1. アプリケーションのプロセス P1 が、探索条件を満たす一連の *n* 個の行を読み取る。
2. 次にアプリケーション・プロセス P2 が、探索条件を満たす 1 つまたは複数の行を挿入 (INSERT) し、それらの挿入行をコミット (COMMIT) する。
3. P1 が同じ探索条件で一連の行を再度読み取り、元の行と P2 によって挿入された行を両方とも獲得する。

RS レベルで実行しているアプリケーション・プロセスは、排他ロック以外に、条件に合うすべての行に対して少なくとも共用ロックを獲得します。

カーソル固定 (CS)

CS レベルは RR レベルと次の点で類似しています。

- CS では、他のアプリケーション・プロセスによって変更された行は、そのアプリケーション・プロセスによってコミットされるまで読み取ることができないように保証されます。

CS レベルは RR レベルと次の点で異なっています。

- CS では、すべての更新可能なカーソルの現在行が、他のアプリケーション・プロセスによって変更されない、ということしか保証されません。このため、作業単位の中で読み取られた行が、他のアプリケーション・プロセスによって変更される可能性があります。

CS レベルで実行しているアプリケーション・プロセスは、排他ロックに加えて、すべてのカーソルの現在行に対して少なくとも共用ロックを獲得します。

非コミット読み取り (UR)

SELECT INTO、読み取り専用カーソルによる FETCH、INSERT で使用される全選択、 UPDATE での行の全選択、またはスカラーの全選択が使用されているなら、UR レベルでは以下のことが可能です。

- 作業単位の中で読み取られた行は、すべて他のアプリケーション・プロセスから変更できます。
- 他のアプリケーション・プロセスで変更された行は、そのアプリケーション・プロセスで変更をコミットしていなくても、すべて読み取ることができます。

これ以外の点では、CS レベルの規則が適用されます。

分離レベルの比較

4 つの分離レベルの比較については、1395ページの『付録I. 分離レベルの比較』で説明されています。

分散リレーショナル・データベース

分散リレーショナル・データベース は、別々のものであるが相互に接続されている複数のコンピューター・システムにわたる、一連の表およびその他のオブジェクトで構成されています。コンピューター・システムごとにリレーショナル・データベース・マネージャーがあり、それぞれの環境で表を管理しています。データベース・マネージャーは、相互に情報をやり取りして協同で作業することにより、あるデータベース・マネージャーが別のコンピューター・システム上で SQL ステートメントを実行できるようになっています。

分散リレーショナル・データベースは、正式なリクエスター / サーバー・プロトコルと機能に基づいて構築されます。*アプリケーション・リクエスター* は、接続の両端のうち、アプリケーション側をサポートするものです。アプリケーション・リクエスターは、アプリケーションからのデータベース要求を分散データベース・ネットワークに適した通信プロトコルに変換します。このような要求は、接続のもう一方の側の*アプリケーション・サーバー* によって受信され、処理されます。アプリケーション・リクエスターとアプリケーション・サーバーが、協同で通信や位置についての考慮事項を処理することによって、アプリケーションでは通信や場所に関する考慮事項を意識することなく、ローカル・データベースをアクセスするかのようにして操作できるようになります。分散リレーショナル・データベースの 1 つの簡単な環境を図3 に示します。

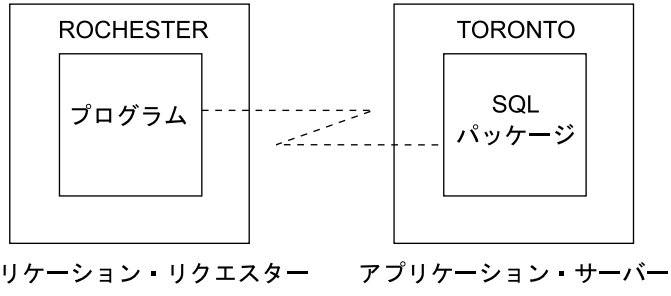


図3. 分散リレーショナル・データベース環境

分散リレーショナル・データベース体系 (DRDA) の通信プロトコルの詳細については、 *分散データ・ライブラリー 分散リレーショナル・データベース体系解説書* (N:SC26-4651) を参照してください。

アプリケーション・サーバー

アプリケーション・プロセスが表や視点を参照する SQL ステートメントを実行するためには、その前にデータベース・マネージャーのアプリケーション・サーバーに接続する必要があります。アプリケーション・プロセスとそのサーバーとの接続を確立するには、 CONNECT ステートメントが使われます。[9] サーバー側で CONNECT ステートメントを実行してサーバーを変更することができます。

アプリケーション・サーバーは、プロセスが開始される環境に対してローカルな位置にあっても、リモートの位置にあってもかまいません。 (アプリケーション・サーバーは、分散リレーショナル・データベースを使用していない場合でも存在しています。) この環境には、 CONNECT ステートメントに指定されるアプリケーション・サーバーを記述するローカル・ディレクトリーが含まれています。ローカル・ディレクトリーについては、*管理の手引き* を参照してください。

表や視点を参照する静的 SQL ステートメントを実行するため、アプリケーション・サーバーではバインドした形式のステートメントを使用します。このバインド・ステートメントは、データベース・マネージャーでバインド操作によってそれ以前に作成されているパッケージから取り出されます。

ほとんどの部分で、アプリケーションは、現在接続されているアプリケーション・サーバーのデータベース・マネージャーがサポートするステートメントおよび文節を使用できます。このことは、一部のステートメントおよび文節がサポートされていないデータベース・マネージャーのアプリケーション・リクエスターによってアプリケーションが実行されている場合でも同じです。

アプリケーション・サーバーを使って分散データ・ソースのシステムで照会を実行要求する方法については、 49ページの『サーバー定義とサーバー・オプション』を参照してください。

CONNECT (タイプ 1) と CONNECT (タイプ 2)

CONNECT ステートメントには、次の 2 つのタイプがあります。

---

9. DB2 CLI および組み込み SQL は、*並行トランザクション* と呼ばれる接続モードをサポートします。これは、それぞれが独立したトランザクションである複数の接続をサポートします。 1 つのアプリケーションが同じデータベースに対して複数の接続を並行して行うことができます。複数のスレッドによるデータベース・アクセスの詳細については、 *アプリケーション開発の手引き* を参照してください。

- CONNECT (タイプ 1) では、作業単位 (リモート作業単位) ごとに 1 つのデータベースがサポートされます。 589ページの『CONNECT (タイプ 1)』を参照してください。
- CONNECT (タイプ 2) では、作業単位 (アプリケーション制御の分散作業単位) ごとに複数のデータベースがサポートされます。 599ページの『CONNECT (タイプ 2)』 を参照してください。

リモート作業単位

リモート作業単位 機能は、 SQL ステートメントをリモートで作成および実行するためのものです。コンピューター・システム A のアプリケーション・プロセスは、コンピューター・システム B のアプリケーション・プロセスのアプリケーション・サーバーに接続し、 1 つ以上の作業単位の中で、 B のオブジェクトを参照する任意の数の静的または動的 SQL ステートメントを実行できます。 B での作業単位が終了したら、このアプリケーション・プロセスはコンピューター・システム C のアプリケーション・サーバーに接続する、というようなことが可能です。

以下の制限付きで、ほとんどの SQL ステートメントをリモートで準備、実行することができます。

- 単一の SQL ステートメントで参照されるオブジェクトは、すべて同一のアプリケーション・サーバーによって管理される必要があります。
- ある作業単位内の SQL ステートメントは、すべて同一のアプリケーション・サーバーによって実行される必要があります。

リモート作業単位の接続管理

ここでは、アプリケーション・プロセスの接続状態について概要を説明します。

接続状態

アプリケーション・プロセスは、常に次の 4 つのいずれかの状態にあります。

接続可能 / 接続状態
接続不能で接続済み
接続可能 / 未接続状態
暗黙に接続可能 (暗黙接続が利用可能の場合)

暗黙の接続が可能な場合 (38ページの図4 を参照)、アプリケーション・プロセスは当初、*暗黙接続可能* 状態になっています。暗黙の接続が使用不能の場合 (39ページの図5 を参照)、アプリケーション・プロセスは当初、*接続可能・未接続* 状態になっています。

暗黙接続の有効性は、導入オプション、環境変数と認証設定値によって決まります。導入時の暗黙接続の設定については*概説およびインストール* を、環境変数および認証設定値については*管理の手引き* を参照してください。

暗黙接続可能状態

暗黙の接続が可能な場合は、これがアプリケーション・プロセスの初期状態です。 CONNECT RESET ステートメントを使うと、この状態に移行します。接続不可・接続済み状態で COMMIT または ROLLBACK ステートメントを発行した後、接続可能・接続済み状態で DISCONNECT ステートメントを発行することによっても、この暗黙接続可能状態に移行できます。

接続可能・接続済み状態

アプリケーション・プロセスがアプリケーション・サーバーに接続されていて、 CONNECT ステートメントの実行が可能です。

暗黙の接続が可能な場合は以下のようになります。

- 接続可能・未接続状態で CONNECT TO ステートメントまたはオペランドなしの CONNECT ステートメントが正常に実行されると、アプリケーション・プロセスは接続可能・接続済み状態になります。
- また、CONNECT RESET、DISCONNECT、SET CONNECTION、または RELEASE ステートメント以外のステートメントが発行された場合も、アプリケーション・プロセスは暗黙接続可能状態からこの接続可能・接続済み状態に移行することがあります。

暗黙の接続が可能か否かに関係なく、以下の場合にはこの接続可能・接続済み状態になります。

- 接続可能・未接続状態から CONNECT TO ステートメントが正常に実行された場合。
- 接続不可・接続済み状態から、COMMIT または ROLLBACK ステートメントが正常に発行されるか、または強制的なロールバックが発生した場合。

接続不可・接続済み状態

アプリケーション・プロセスはアプリケーション・サーバーに接続されていますが、アプリケーション・サーバーを変更する CONNECT TO ステートメントを正常に実行することはできません。アプリケーション・プロセスが、CONNECT TO、オペランドなしの CONNECT、 CONNECT RESET、DISCONNECT、SET CONNECTION、RELEASE、COMMIT、ROLLBACK 以外の SQL ステートメントを実行すると、接続可能・接続済み状態からこの接続不可・接続済み状態に移行します。

接続可能・未接続状態

アプリケーション・プロセスはアプリケーション・サーバーに接続していません。実行できる SQL ステートメントは CONNECT TO だけで、それ以外を実行するとエラー (SQLSTATE 08003) になります。

暗黙の接続が可能かどうかにかかわらず、以下のようになります。

– アプリケーション・プロセスは、CONNECT TO ステートメントの発行時にエラーが発生するか、または接続の消失とロールバックを引き起こすエラーが作業単位の中で発生した場合に、この接続可能・未接続状態になります。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発生したエラーや、サーバー名がローカル・ディレクトリーのリストにないために発生したエラーでは、この状態へは移行しません。

暗黙の接続が使用不能の場合は、以下のようになります。

– CONNECT RESET および DISCONNECT ステートメントが実行されると、この接続可能・未接続状態へ移行します。

状態遷移 の様子を次の図に示します。

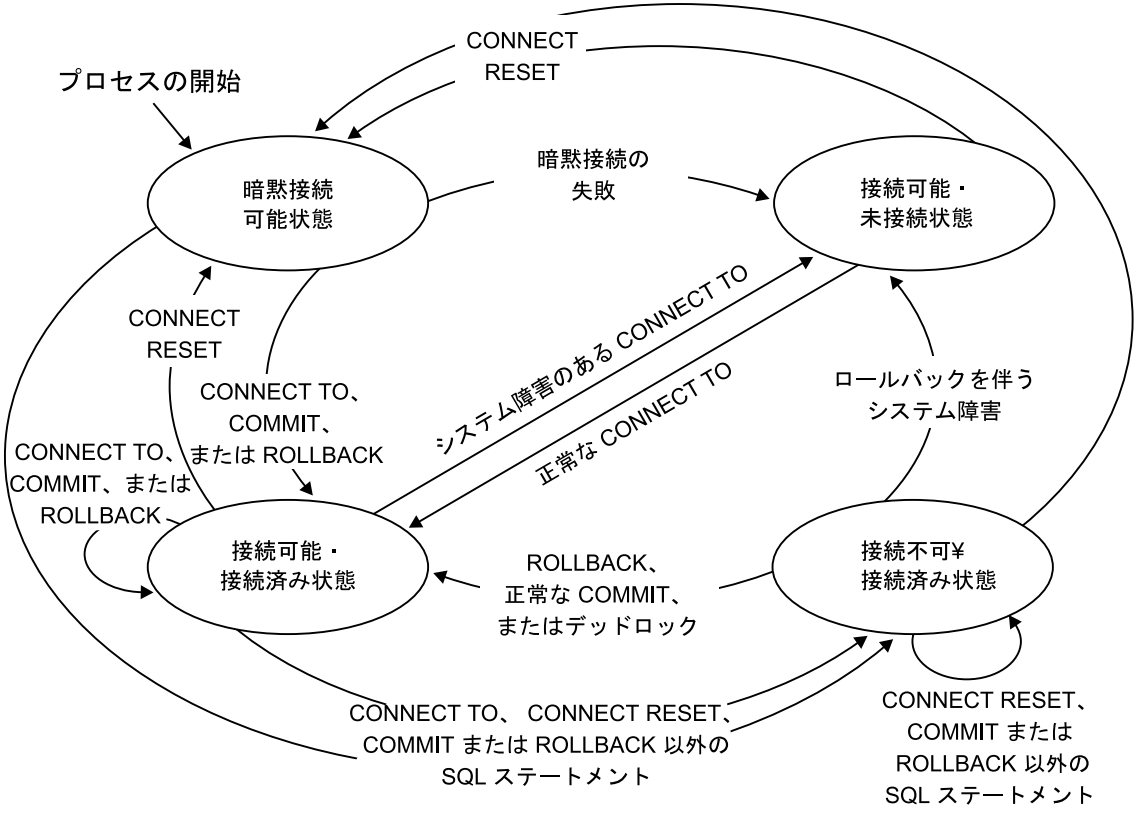


図 4. 暗黙的接続が使用可能な場合の接続状態の遷移

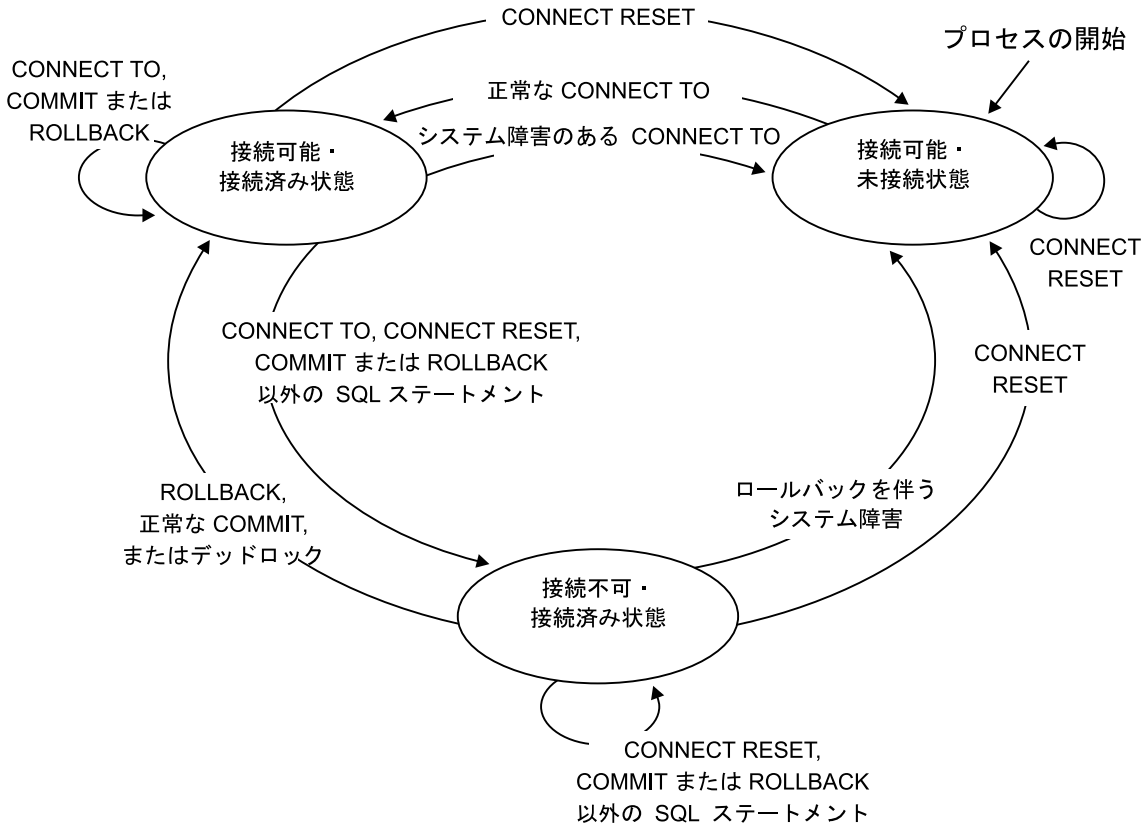


図 5. 暗黙の接続が使用不能な場合の接続状態の遷移

その他の規則

- 連続して複数の CONNECT ステートメントを実行することはエラーではありません。これは、CONNECT 自体ではアプリケーション・プロセスの接続可能状態は終了しないためです。
- 連続的な CONNECT RESET ステートメントを実行するのはエラーです。
- CONNECT TO、CONNECT RESET、オペランドなしの CONNECT、 SET CONNECTION、RELEASE、COMMIT、ROLLBACK 以外の SQL ステートメントを実行した後、 CONNECT TO ステートメントを実行すると、エラーになります。このエラーを回避するには、CONNECT TO の実行前に、CONNECT RESET、DISCONNECT (COMMIT または ROLLBACK ステートメントの後)、 COMMIT、または ROLLBACK ステートメントを実行する必要があります。

アプリケーション制御の分散作業単位

アプリケーション制御の分散作業単位機能 により、リモート作業単位と同じように、SQL ステートメントをリモートで作成し、実行することができます。コンピューター・システム A のアプリケーション・プロセスは、 CONNECT または SET CONNECTION ステートメントを発行することにより、コンピューター・システム B のアプリケーション・サーバーに接続できます。作業単位の終了までの間に、アプリケーション・プロセスは、 B のオブジェクトを参照する任意の数の静的または動的 SQL ステートメントを実行することができます。単一の SQL ステートメントで参照されるオブジェクトは、すべて同一のアプリケーション・サーバーによって管理される必要があります。しかし、リモート作業単位とは異なり、複数のアプリケーション・サーバーが同じ作業単位に加わることができます。コミットまたはロールバック操作により、作業単位が終了します。

アプリケーション制御の分散作業単位の接続管理

アプリケーション制御の分散作業単位では、タイプ 2 の接続を使用します。タイプ 2 の接続は、指定され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ための規則を確立します。

アプリケーション・プロセスと接続状態の概要

タイプ 2 のアプリケーション・プロセスは、常に以下の状態になっています。

- 常に接続可能です。
- *接続済み* 状態か、*未接続* 状態です。
- ゼロ個以上の接続があります。

アプリケーション・プロセスの各接続は、その接続のアプリケーション・サーバーのデータベース別名によって、一意に識別されます。

個々の接続の状態は、常に以下の中のいずれか 1 つになっています。

- *現行 で保留*
- *現行 で解放保留*
- *休止 で保留*
- *休止 で解放保留*

初期状態および状態遷移: タイプ 2 のアプリケーション・プロセスは当初、*未接続状態* であり、接続はありません。

初期の接続は、*現行 で保留状態* です。

状態の移行する様子を次の図に示します。

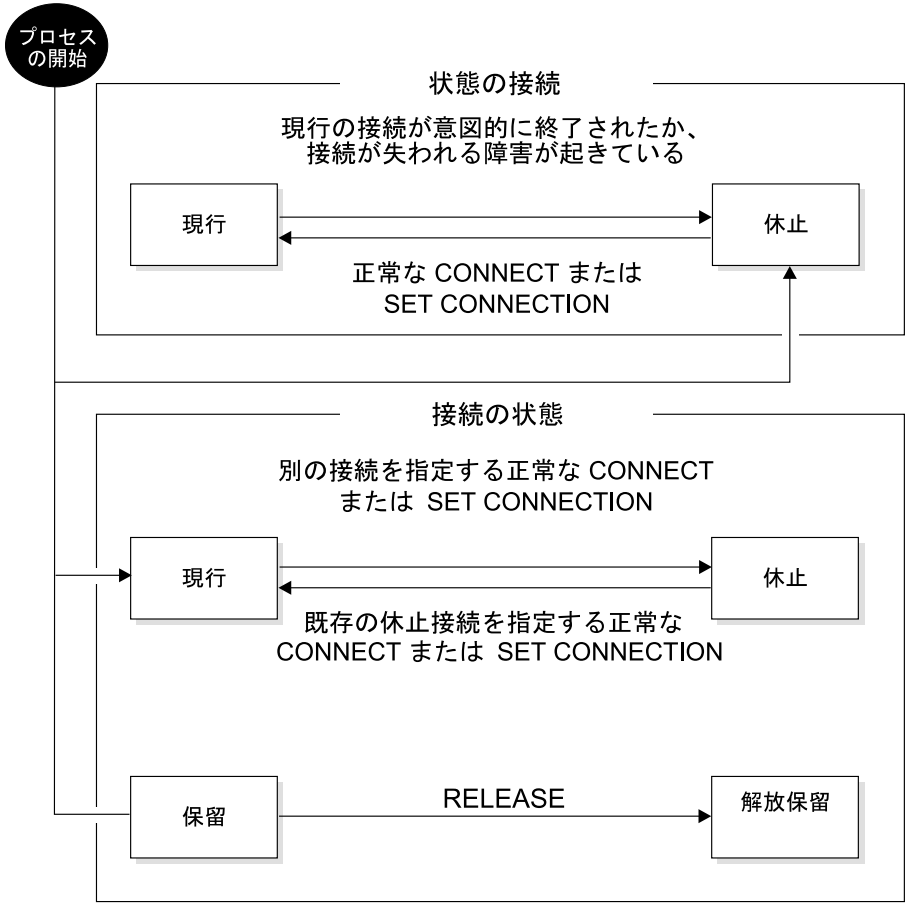


図 6. アプリケーション制御の分散作業単位とアプリケーション・プロセスの接続状態遷移

アプリケーション・プロセスの接続状態: 明示的または暗黙の CONNECT ステートメントの実行により、種々のアプリケーション・サーバーを確立することができます。[10] 以下の規則が適用されます。

- 文脈は、 1 つのアプリケーション・サーバーに対して同時に 2 つ以上の接続を持つことはできません。同時に同じ DB2 ユニバーサル・データベースに対して複数の接続を行う場合のサポートについては、 *管理の手引き* およびアプリケーション*開発の手引き* を参照してください。

10. タイプ 2 の暗黙接続は、タイプ 1 よりも制限が厳しくなります。詳細については、599ページの『CONNECT (タイプ 2)』を参照してください。

- アプリケーション・プロセスが SET CONNECTION ステートメントを実行する場合、指定する位置名は、そのアプリケーション・プロセスの接続の集合に含まれている既存の接続でなければなりません。
- アプリケーション・プロセスが CONNECT ステートメントを実行する場合、 SQLRULES(STD) オプションが有効なら、指定するサーバー名は、そのアプリケーション・プロセスの接続の集合に含まれている既存の接続であってはなりません。 SQLRULES オプションについては、44ページの『分散作業単位の意味を制御するオプション』 を参照してください。

アプリケーション・プロセスに現行の接続がある場合、アプリケーション・プロセスは*接続済み* 状態です。 CURRENT SERVER 特殊レジスターに、現行の接続のアプリケーション・サーバーの名前が入れられています。アプリケーション・プロセスは、アプリケーション・サーバーによって管理されるオブジェクトを参照する SQL ステートメントを実行できます。

未接続状態のアプリケーション・プロセスは、 CONNECT または SET CONNECTION ステートメントが正常に実行されたときに、接続済み状態に移行します。アプリケーションに接続がなく、SQL ステートメントが発行された場合に、 DB2DBDFT 環境変数がデフォルトのデータベースで定義されているなら、暗黙の接続が行われます。

アプリケーション・プロセスに現行の接続がない場合、アプリケーション・プロセスは*未接続* 状態です。実行できる SQL ステートメントは、CONNECT、DISCONNECT ALL、データベースを指定した DISCONNECT、 SET CONNECTION、 RELEASE、COMMIT、および ROLLBACK だけです。

接続状態 のアプリケーション・プロセスは、現行の接続が意図的に終了された場合、またはアプリケーション・サーバーでのロールバック操作を引き起こして、接続が失われることになるような障害のために SQL ステートメントが正常に実行されなかった場合、*未接続状態* に移行します。接続が*解放保留状態* のときに、 DISCONNECT ステートメントが正常に実行された場合、またはコミット操作が正常に実行された場合、接続は意図的に終了させられます。意図的な接続終了の動作は、 DISCONNECT プリコンパイラー・オプションで指定される各種のオプションの影響を受けます。それが AUTOMATIC に設定されている場合は、すべての接続が終了します。 CONDITIONAL に設定されている場合は、オープンされている WITH HOLD カーソルのない接続がすべて終了します。

接続状態: アプリケーション・プロセスが CONNECT ステートメントを実行し、サーバー名がアプリケーション・リクエスターに認識され、かつ、そのサーバー名がそのアプリケーション・プロセスの既存の一連の接続にない場合は、以下のようになります。

- 現行の接続は*休止状態* に移行します。また、
- そのサーバー名が接続の集合に追加されます。また、
- 新しい接続が*現行状態* かつ*保留状態* になります。

サーバー名がすでにアプリケーション・プロセスの既存の接続の集合に含まれている場合に、アプリケーションを SQLRULES(STD) オプション付きでプリコンパイルすると、エラー (SQLSTATE 08002) になります。

- ***保留状態および解放保留状態:*** RELEASE ステートメントは、接続が保留状態か解放保留状態かを制御します。*解放保留状態* とは、その次の正常なコミット操作で接続が切断される状態のことです (ロールバックは接続に影響しません)。*保留状態* とは、その次の操作で接続が切断されない状態のことです。すべての接続は、最初は保持状態であり、 RELEASE ステートメントを使うと解放保留状態に移行します。接続が、いったん解放保留状態になると、保留状態に戻ることはありません。 ROLLBACK ステートメントが発行されるか、またはコミット操作の異常によりロールバック操作が発生した場合、接続は作業単位の境界を超えて解放保留状態のまま残ることになります。

  接続に対して解放が明示されていなくても、コミット操作が DISCONNECT プリコンパイラー・オプションの条件を満たしていれば、コミット操作によって切断可能です。

- ***現行状態および休止状態:*** 接続が*保留状態* であるか*解放保留状態* であるかに関係なく、接続は、*現行状態* もしくは*休止状態* でもあります。*現行状態* とは、接続が、この状態の間に実行される SQL ステートメントに対して使用される接続であることです。*休止状態* とは、接続が現行でない状態のことです。休止接続に対して発行可能な SQL ステートメントは、 COMMIT と ROLLBACK、または DISCONNECT と RELEASE です。 DISCONNECT と RELEASE では、 ALL (すべての接続) または特定のデータベース名のいずれかを指定できます。 SET CONNECTION および CONNECT ステートメントを使うと、指定したサーバーとの接続が*現行状態* に変更になり、既存の接続は*休止状態* に移行するかまたは休止状態のままになります。どの時点でも、*現行状態* にある接続は 1 つだけです。同じ作業単位の中で休止接続が現行状態に変わると、すべてのロック、カーソル、作成済みステートメントの状態は同じままで残り、接続が現行であったときに最後に使用されたときの状態が反映されます。

接続の終了時: 接続が終了すると、アプリケーション・プロセスが接続によって獲得していたすべての資源、および接続を確立し維持するために使用されたすべての資源が割り振り解除されます。たとえば、アプリケーション・プロセスが RELEASE ステートメントを実行すると、その次のコミット操作で接続が終了するときにオープンしているカーソルがすべてクローズされます。

接続は、通信の障害によっても終了することがあります。終了する接続が現行の接続であった場合、アプリケーション・プロセスは未接続状態に移行します。

アプリケーション・プロセスが終了すると、そのプロセスのすべての接続が終了します。

分散作業単位の意味を制御するオプション

タイプ 2 接続管理の意味は、プリコンパイラーのオプション群によって決定されます。これらのオプションについて以下に要約します。デフォルト値はボールド体に下線を付けたテキストで示します。詳細については、*コマンド解説書* または *管理 API 解説書* を参照してください。

- CONNECT (**1** | 2)

  CONNECT ステートメントが、タイプ 1 とタイプ 2 のどちらとして処理されるかを指定します。

- SQLRULES (**DB2** | STD)

  タイプ 2 の CONNECT が、 CONNECT による休止接続への切り替えを認める DB2 規則に従って処理されるのか、それともその切り替えを認めない SQL92 標準 (STD) 規則に従って処理されるのかを指定します。

- DISCONNECT (**EXPLICIT** | CONDITIONAL | AUTOMATIC)

  コミット操作の発生時に切断されるデータベース接続を指定します。これは以下のいずれかです。
  - SQL の RELEASE ステートメントによって解放するよう明示的に指定されているデータベース接続 (EXPLICIT)、または
  - 解放するように指定されているデータベース接続、およびオープンしている WITH HOLD カーソルのないデータベース接続 (CONDITIONAL) [11] または
  - すべての接続 (AUTOMATIC)

- SYNCPOINT (**ONEPHASE** | TWOPHASE | NONE)

11. CONDITIONAL オプションは、バージョン 2 より前の下位レベルのサーバーでは正しく機能しません。このような場合、WITH HOLD カーソルの有無に関係なく、切断が発生します。

複数のデータベース接続にまたがってコミットまたはロールバックを調整する仕方を指定します。

ONEPHASE 作業単位の中で更新を行えるのは 1 つのデータベースに対してだけです。他のデータベースはすべて読み取り専用です。他のデータベースに対して更新しようとすると、エラー (SQLSTATE 25000) になります。

TWOPHASE 実行時にトランザクション・マネージャー (TM) を使って、このプロトコルをサポートするデータベースの間で 2 フェーズ・コミットを調整します。

NONE 2 フェーズ・コミットを実行する TM を一切使用せず、単一更新・複数読み取りを強制しません。 COMMIT または ROLLBACK ステートメントが実行されると、個々の COMMIT または ROLLBACK がすべてのデータベースに通知されます。 1 つまたは複数のロールバックが失敗すると、エラー (SQLSTATE 58005) になります。 1 つまたは複数のコミットが失敗すると、エラー (SQLSTATE 40003) になります。

上記のどのオプションについても、特殊な SET CLIENT アプリケーション・プログラミング・インターフェース (API) を使用することによって、実行時に指定変更することができます。これらの現行の設定値は、特殊な QUERY CLIENT API を使用して調べることができます。これらは SQL ステートメントではなく、さまざまなホスト言語およびコマンド行プロセッサーで定義されている API であることに注意してください。これらは、*コマンド解説書* および*管理 API 解説書* で定義されています。

データの表現についての考慮事項

システムが異なると、データを表現する方式も異なります。データをあるシステムから別のシステムへ移動する場合、データ変換が必要なことがあります。DRDA をサポートする製品は、データを受け取る側のシステムで必要な変換を自動的に実行します。 数値データの変換では、データ・タイプと、そのデータ・タイプが送り側システムでどのように表現されるかという情報が必要です。文字データの場合、文字ストリングの変換のためにさらに情報が必要になります。ストリングの変換は、データのコード・ページと、そのデータに対して実行する操作の両方に応じて変わります。文字変換は、IBM の文字データ表現体系 (CDRA) に従って実行されます。文字変換の詳細については、*Character Data Representation Architecture Reference* (SC09-1390) を参照してください。

DB2 連合システム

この節では、DB2 連合システムに関連した各要素の概要と、このシステムの管理者およびユーザーが実行するタスクの概要とを示します。また、それらのタスクに関連した概念についても説明します。

連合サーバー、連合データベース、およびデータ・ソース

DB2 連合システム は、以下のもので構成される分散コンピューティング・システムです。

- *連合サーバー* と呼ばれる DB2 サーバー。

  DB2 インストール・システムでは、 DB2 インスタンスをいくつでも連合サーバーとして構成することができます。

- 連合サーバーからの照会の送信先になる複数のデータ・ソース。

  各データ・ソースは、関係データベース管理システムのインスタンス 1 つと、そのインスタンスがサポートするデータベース (複数可) とで構成されます。 DB2 連合システム内のデータ・ソースには、 Oracle インスタンスや DB2 ファミリー・メンバーのインスタンスを含めることができます。

  データ・ソースは半自立的です。たとえば、Oracle アプリケーションが Oracle データ・ソースにアクセスしている最中でも、連合サーバーはそれらの Oracle データ・ソースに照会を送信できます。 DB2 連合システムは、(保全性やロックの制約を無視してまでも) Oracle などのデータ・ソースへのアクセスを独占したり制限することはありません。

エンド・ユーザーやクライアント・アプリケーションの側から見ると、データ・ソースは単一の集合的なデータベースです。実際、ユーザーやアプリケーションとのインターフェースには、*連合データベース* と呼ばれるデータベースが使用されます。このデータベースは連合サーバーの中に置かれます。データ・ソースからデータを取得するには、連合データベースに対して DB2 SQL 形式の照会を出します。それから DB2 は、出された照会を適切なデータ・ソースに配布します。 DB2 は照会を最適化するためのアクセス・プランも提供します。 (これらのプランは、データ・ソースではなく、連合サーバーで照会を処理するよう求める場合もあります。) 最後に、DB2 は要求データを収集し、それをユーザーやアプリケーションに渡します。

連合サーバーからデータ・ソースに出される照会は、読み取り専用でなければなりません。データ・ソースに書き込みを行う必要がある場合 (たとえば、データ・ソース表を更新する場合)、ユーザーやアプリケーションはそのデータ・ソース専用の SQL を、*パススルー* と呼ばれる特殊モードで使用しなければなりません。

DB2 連合システムで実行するタスク

この節では、連合システムを確立および使用する際にユーザーが実行するタスクに関連した概念を紹介します。 (この節と以降の節では、ユーザーという用語は、連合システムに関連した作業を行うすべての人を指します。たとえば、データベース管理者、アプリケーション・プログラマー、エンド・ユーザーはすべてユーザーです。)

これから紹介する一連のタスクでは、それらのタスクを一般に実行するのはどのタイプのユーザーであるかが明らかにされています。とはいえ、他のタイプに属するユーザーもそれらのタスクを実行できることに注意してください。たとえば、連合データベースへのアクセス許可とデータ・ソースへのアクセス許可との間のマッピングを行うのは、一般的に DBA であることが示されています。しかし、アプリケーション・プログラマーやエンド・ユーザーが、このタスクを実行できないわけではありません。

DB2 連合システムを確立し、使用する方法:

1. 連合サーバーとして使用する DB2 サーバーを DBA が指定します。このことを行う方法については、*インストールおよび構成 補足* を参照してください。
2. アクセスするデータ・ソースをセットアップします:
   a. DBA が、連合データベースに接続します。
   b. DBA が、連合システムに含めるデータ・ソースの区分ごとにラッパーを 1 つずつ作成します。 (ラッパー とは、データ・ソースとの対話を行うために連合サーバーが使用する機構のことです。詳細については、49ページの『ラッパーおよびラッパー・モジュール』を参照してください。)
   c. DBA が、各データ・ソースの記述を連合サーバーに提供します。 (この記述のことを*サーバー定義* といいます。詳細については、49ページの『サーバー定義とサーバー・オプション』を参照してください。)
   d. 連合データベースへアクセスするときに使う許可 ID が、データ・ソースへアクセスするときに使う許可 ID とは異なる場合は、 DBA がこれら 2 つの許可 ID の関連を定義します。 (この関連のことをユーザー・マッピング といいます。詳細については、52ページの『ユーザー・マッピングとユーザー・オプション』を参照してください。)
   e. DB2 のデータ・タイプとデータ・ソースのデータ・タイプとの間のデフォルト・マッピングがユーザー要件と異なる場合は、 DBA が必要に応じてマッピングを修正します。 (データ・タイプ・マッピング とは、 2 つの互換性のあるデータ・タイプ (1 つは連合データベースがサポート

するデータ・タイプ、もう 1 つはデータ・ソースがサポートするデータ・タイプ) の間であらかじめ定義された関係のことです。詳細については、53ページの『データ・タイプ・マッピング』を参照してください。)

f. DB2 の関数とデータ・ソースの関数との間のデフォルト・マッピングがユーザー要件と異なる場合は、 DBA が必要に応じてマッピングを修正します。 (*関数マッピング* とは、 2 つの互換性のあるデータ・タイプ (1 つは連合データベースがサポートする関数、もう 1 つはデータ・ソースがサポートする関数) の間であらかじめ定義された関係のことです。詳細については、54ページの『関数マッピング、関数テンプレート、および関数マッピング・オプション』を参照してください。)

g. DBA とアプリケーション・プログラマーは、アクセスする予定のデータ・ソース表および視点にニックネームを作成します。 (ニックネームとは、連合システムがデータ・ソース表や視点を参照する際に使用する識別子のことです。詳細については、55ページの『ニックネームと列オプション』を参照してください。)

h. 任意選択: データ・ソース表に索引がない場合は、実際に索引があったならばその定義に含めたであろう情報を DBA は連合サーバーに提供できます。データ・ソース表に連合サーバーの認識できない索引がある場合は、 DBA はその索引の存在をサーバーに通知できます。どちらの場合も、DBA が提供する情報は DB2 が表データの照会を最適化するのに役立ちます。 (この情報のことを *索引指定* といいます。詳細については、56ページの『索引指定』を参照してください。)

3. アプリケーション・プログラマーやエンド・ユーザーがデータ・ソースの情報を検索します:

   - DB2 SQL を使用して、アプリケーション・プログラマーやエンド・ユーザーが、ニックネームによって参照されている表や視点を照会します。 (複数のデータ・ソースに送信される照会のことを、*分散要求* といいます。詳細については、57ページの『分散要求』を参照してください。)

     照会を処理するときに、連合サーバーは DB2 SQL ではサポートされているものの、データ・ソースの SQL ではサポートされていない操作を実行することができます。 (この能力のことを *補償* といいます。詳細については、58ページの『補償』を参照してください。)

   - アプリケーション・プログラマーやエンド・ユーザーは、データ・ソース独自の SQL の形式でデータ・ソースへの照会、 DML ステートメント、および DDL ステートメントを出すことがあります。プログラマーやユーザーはこの操作をパススルー・モードで実行できます。 (詳細については、59ページの『パススルー』を参照してください。)

以降の節では、ここまで紹介してきたタスクの概念について、紹介したのと同じ順序で説明します。節によっては、関連する他の概念についても説明します。

ラッパーおよびラッパー・モジュール

ラッパーとは、データ・ソースと通信したり、データ・ソースのデータを検索するために、連合サーバーが使用する機構のことです。ラッパーを実装するために、サーバーはラッパー・モジュール と呼ばれる、ライブラリーに含まれているルーチンを使用します。これらのルーチンを使用することにより、サーバーはデータ・ソースへの接続やデータ・ソースのデータの検索といった繰り返しの操作を実行します。

次の 3 つのラッパーがあります。

- デフォルト名が DRDA のラッパー。このラッパーはすべての DB2 ファミリー・データ・ソースに使用されます。
- デフォルト名が SQLNET のラッパー。このラッパーは Oracle の SQL*Net クライアント・ソフトウェアがサポートするすべての Oracle データ・ソースに使用されます。
- デフォルト名が NET8 のラッパー。このラッパーは Oracle の Net8 クライアント・ソフトウェアがサポートするすべての Oracle データ・ソースに使用されます。

ラッパーは、CREATE WRAPPER ステートメントを使用して連合サーバーに登録します。詳細については、912ページの『CREATE WRAPPER』を参照してください。

サーバー定義とサーバー・オプション

DBA がラッパーを登録し、連合サーバーがデータ・ソースと対話できるようになると、 DBA は次にそれらのデータ・ソースを連合データベースに定義します。この節の内容は以下のとおりです。

- DBA が提供する定義について説明します。
- *サーバー・オプション* と呼ばれるパラメーターを指定するための SQL について説明します。これらのパラメーターにはサーバー定義の部分が含まれます。
- 『サーバー』 という用語の 3 通りの意味を明らかにします。

サーバー定義について

連合データベースにデータ・ソースを定義する際、 DBA はそのデータ・ソースの名前と情報を提供します。この情報には、そのデータ・ソースがインスタ

ンスとなっている RDBMS のタイプとバージョン、およびそのデータ・ソース用の RDBMS の名前が含まれます。その RDBMS に特有のメタデータも含まれます。たとえば、DB2 ファミリー・データ・ソースには複数のデータベースを含めることができますが、データ・ソースの定義には、連合サーバーがそれらのデータベースのうちどれに接続できるのかを指定しなければなりません。一方、Oracle データ・ソースは 1 つのデータベースにだけ関連付けられます。連合サーバーはそのデータベースの名前が分からなくても接続することができます。したがって、Oracle データ・ソースの連合サーバー定義に名前を含める必要はありません。

DBA が提供する名前と情報のことを一括してサーバー定義といいます。この用語は、データ要求に応答するという役割ゆえにデータ・ソースが実質的にサーバーであるという事実を反映しています。他の用語もこの事実を反映しています。たとえば、次のとおりです。

- サーバー定義の情報の一部は、サーバー・オプションとして格納されます。たとえば、データ・ソースの名前は、NODE と呼ばれるサーバー・オプションの値として格納されます。 DB2 ファミリー・データ・ソースの場合、連合サーバーが接続するデータベースの名前は、 DBNAME と呼ばれるサーバー・オプションの値として格納されます。
- サーバー定義を作成するための SQL ステートメントは CREATE SERVER、サーバー定義を変更するための SQL ステートメントは ALTER SERVER です。

サーバー・オプションを設定するための SQL ステートメント

サーバー・オプションに値を割り当てるには、 CREATE SERVER、ALTER SERVER、および SET SERVER OPTION ステートメントを使用します。

CREATE SERVER ステートメントと ALTER SERVER ステートメントは、データ・ソースに正常に接続されている間は持続する値をサーバー・オプションに設定します。これらの値はカタログに保管されます。次のようなシナリオを考えてみましょう。連合システムの DBA が、CREATE SERVER ステートメントを使用して、新しい Oracle データ・ソースをその連合システムに定義しようとしています。この新しいデータ・ソースのデータベースには、連合データベースが使用しているのと同じ照合順序を使用させます。パフォーマンスを向上させるため、DBA は最適化プログラムに、この一致について認識させたいと考えています。そのために DBA は、 CREATE SERVER ステートメントで COLLATING_SEQUENCE というサーバー・オプションを 'Y' (はい、データ・ソースと連合データベースの照合順序は同じです) に設定します。 'Y' という設定がカタログに記録され、ユーザーやアプリケーションが Oracle データ・ソースにアクセスしている間は有効です。

数か月後、Oracle の DBA は Oracle データ・ソースの照合順序を変更することにしました。それに応じて、連合システムの DBA は COLLATING_SEQUENCE を 'N' (いいえ、データ・ソースと連合データベースの照合順序は異なります) にリセットします。 DBA は ALTER SERVER ステートメントを使用してこの更新を行います。カタログも更新されます。新しい設定はユーザーやアプリケーションがデータ・ソースにアクセスし続けている限り有効です。

SET SERVER OPTION ステートメントは、連合データベースへの単一接続が行われている間、サーバー・オプションの値を一時的にオーバーライドします。オーバーライドしている値がカタログに保管されることはありません。

例を挙げて説明しましょう。PLAN_HINTS というサーバー・オプションを設定すれば、 DB2 は Oracle データ・ソースにプラン・ヒントと呼ばれるステートメントの断片を提供することができます。このプラン・ヒントは、Oracle 最適化プログラムのジョブを援助します。たとえば、表へのアクセスにどの索引を使用したらよいか、また結果セットを生成するためにデータを検索するときにどの表結合順序を使用したらよいかを決定する上で、このプラン・ヒントは最適化プログラムを援助します。

データ・ソース ORACLE1 と ORACLE2 について、 PLAN_HINTS サーバー・オプションがデフォルトの 'N' (いいえ、これらのデータ・ソースにプラン・ヒントを提供しません) に設定されているものとします。その後、プログラマーは ORACLE1 と ORACLE2 のデータに対する分散要求を作成することになりましたが、そのときにプラン・ヒントに最適化プログラムを援助させることにより、このデータにアクセスするための戦略を向上させたいと考えました。そのため、このプログラマーは SET SERVER OPTION ステートメントを使用して、 'N' を 'Y' (はい、プラン・ヒントを提供します) でオーバーライドしました。要求を含んでいるアプリケーションが連合データベースに接続している間だけ、この 'Y' 設定は有効です。この設定はカタログには保管されません。

詳細については、766ページの『CREATE SERVER』、503ページの『ALTER SERVER』、および 1132ページの『SET SERVER OPTION』を参照してください。すべてのサーバー・オプションとその設定値の説明については、 1355ページの『サーバー・オプション』を参照してください。

“サーバー” という用語の 3 つの意味

サーバー定義 および サーバー・オプション という用語において、また前節で説明した SQL ステートメントにおいては、サーバー という用語はデータ・ソースだけを指します。連合サーバーや DB2 アプリケーション・サーバーのことではありません。

とはいえ、DB2 アプリケーション・サーバーの概念と連合サーバーの概念とは重なり合う部分があります。 33ページの『分散リレーショナル・データベース』でも説明したとおり、アプリケーション・サーバーとは、アプリケーション・プロセスが接続し、要求を出す、データベース・マネージャー・インスタンスのことです。このことは連合サーバーにもあてはまります。したがって、連合サーバーはアプリケーション・サーバーの一種といえます。しかし、次の 2 つの点で連合サーバーは他のアプリケーション・サーバーと大きく異なります。

- 連合サーバーは、データ・ソースを最終的に意図している要求を受信するために構成されます。したがって、これらの要求はデータ・ソースに配布されます。
- 他のアプリケーション・サーバーと同様、連合サーバーは DRDA 通信プロトコルを使用して、 DB2 ファミリー・インスタンスとの通信を行います。他のアプリケーション・サーバーと異なるのは、Oracle インスタンスとの通信を行うときには、 sqlnet および net8 通信プロトコルを使用する点です。

ユーザー・マッピングとユーザー・オプション

以下のどちらかの条件が存在する場合、連合サーバーは許可ユーザーまたは許可アプリケーションの分散要求をデータ・ソースに送信できます。

- ユーザーまたはアプリケーションが、連合データベースでもデータ・ソースでも同じユーザー ID を使用している。さらに、データ・ソースがパスワードを必要とする場合には、ユーザーまたはアプリケーションが、連合データベースでもデータ・ソースでも同じパスワードを使用している。
- ユーザーまたはアプリケーションの所有している連合データベースへのアクセス許可が、データ・ソースへのアクセス許可とはいくらか異なる。ユーザーまたはアプリケーションがデータ・ソースへのアクセスを要求するときは、連合データベース許可がデータ・ソース許可に変更されるため、アクセスは問題なく付与されます。この変更が行われるのは、ユーザー・マッピングと呼ばれるあらかじめ定義された関係が 2 つの許可の間に存在するときだけです。

ユーザー・マッピングを定義するには、CREATE USER MAPPING ステートメントを使用します。ユーザー・マッピングを変更するには、ALTER USER

MAPPING ステートメントを使用します。この 2 つのステートメントには、ユーザー・オプション と呼ばれるパラメーターが含まれています。ユーザー・オプションには、許可に関連した値が割り当てられます。たとえば、あるユーザーが連合データベースとデータ・ソースについて、ユーザー ID は同じものを持っているものの、パスワードは別々であるとします。そのユーザーがデータ・ソースにアクセスするためには、パスワードを相互にマッピングする必要があります。これを行うには CREATE USER MAPPING ステートメントを使用して、データ・ソースのパスワードを、REMOTE_PASSWORD というユーザー・オプションに値として割り当てます。

詳細については、893ページの『CREATE USER MAPPING』、550ページの『ALTER USER MAPPING』、および *管理の手引き* を参照してください。ユーザー・オプションとその設定値の説明については、 1361ページの『ユーザー・オプション』を参照してください。

データ・タイプ・マッピング

連合サーバーがデータ・ソース表や視点の列からデータを検索するためには、そのデータ・ソースの列のデータ・タイプを、連合データベースにあらかじめ定義されている対応するデータ・タイプにマッピングしなければなりません。DB2 はデフォルトのマッピングを用意しており、それらは多くのデータ・タイプをサポートしています。たとえば、Oracle のタイプ FLOAT は、デフォルトでは DB2 のタイプ DOUBLE によってマッピングされます。また、DB2 ユニバーサル・データベース (OS/390 版) のタイプ DATE は、デフォルトでは DB2 のタイプ DATE によってマッピングされます。 DB2 連合サーバーがマッピングをサポートしないデータ・タイプは、 LONG VARCHAR、LONG VARGRAPHIC、DATALINK、ラージ・オブジェクト (LOB) タイプ、およびユーザー定義タイプです。

デフォルトのデータ・タイプ・マッピングのリストについては、 1362ページの『デフォルト・データ・タイプ・マッピング』を参照してください。

データ・ソース列から値が戻されるとき、その列に対して実施されるマッピングによって、それらの値は DB2 のタイプに完全に規格合致します。このマッピングがデフォルトのものである場合、それらの値はデータ・ソース・タイプにも、マッピングによって完全に規格合致します。たとえば、FLOAT 列を含む Oracle 表を連合データベースに定義すると、 Oracle FLOAT から DB2 DOUBLE へのデフォルト・マッピングが自動的にその列に対して実施されます (オーバーライドされていない限り)。結果として、列から戻される値は FLOAT にも DOUBLE にも完全に規格合致します。

戻される値の形式や長さを変更することは、それらの値が規格合致していなければならないと定める DB2 タイプを変更すれば可能です。たとえば、Oracle のタイプ DATE は、タイム・スタンプ用のタイプです。このタイプは、デフォルトでは DB2 のタイプ TIMESTAMP にマッピングされます。いくつかの Oracle 表列にデータ・タイプ DATE が含まれており、ユーザーがそれらの列を照会することによって時刻部分だけを戻したいと考えているとしましょう。そうするためには、デフォルトをオーバーライドして、 Oracle のタイプ DATE を DB2 のタイプ TIME にマッピングします。このようにしてから列を照会すれば、タイム・スタンプの時刻部分が戻されます。

CREATE TYPE MAPPING ステートメントを使用すれば、1 つ以上のデータ・ソースだけに適用される、独自のデータ・タイプ・マッピングを作成することができます。 ALTER NICKNAME ステートメントを使用すれば、特定の表の特定の列についてのみ、データ・タイプ・マッピングを変更することができます。

詳細については、887ページの『CREATE TYPE MAPPING』、495ページの『ALTER NICKNAME』、および*アプリケーション開発の手引き* を参照してください。

関数マッピング、関数テンプレート、および関数マッピング・オプション

連合サーバーがデータ・ソース関数を認識するためには、その関数と、サーバーにすでに存在しているそれに対応する DB2 関数との間でのマッピングが必要となります。 DB2 は、既存の組み込みデータ・ソース関数と、組み込み DB2 関数との間でデフォルト・マッピングを提供します。連合サーバーが認識していないデータ・ソース関数 (たとえば、新しい組み込み関数やユーザー定義関数) を使用したいと考えている場合、ユーザーはその関数と、連合データベースでそれに対応する関数との間で行われるマッピングを作成する必要があります。対応する関数が存在しない場合には、以下の要件を満たす関数をユーザーが作成しなければなりません。

- データ・ソース関数に入力パラメーターが含まれている場合には、それに対応する関数の側にもデータ・ソース関数と同じ数の入力パラメーターが必要になります。データ・ソース関数に入力パラメーターが含まれていない場合には、それに対応する関数の側に入力パラメーターを含めてはなりません。
- 対応する関数の入力パラメーター (含まれている場合) に適用されるデータ・タイプは、データ・ソース関数の対応するデータ・タイプと互換性がなければなりません。

対応する側の関数は、完全な関数でも関数テンプレートでも構いません。*関数テンプレート* とは、実行可能コードを含まない部分的な関数のことです。関数

テンプレートを単独で呼び出すことはできません。関数テンプレートの目的はデータ・ソース関数とのマッピングに参加することだけなので、連合サーバーからデータ・ソース関数を呼び出すことが可能になります。

関数マッピングを作成するには、CREATE FUNCTION MAPPING ステートメントを作成します。このステートメントには、*関数マッピング・オプション* と呼ばれるパラメーターが含まれています。このパラメーターには、新しく作成するマッピングに適用する値、またはマッピング内のデータ・ソース関数に適用する値をユーザーが割り当てます。たとえば、データ・ソース関数を呼び出したときに生じるかもしれないオーバーヘッドについての見積統計を、それらの値に含めることができます。関数を呼び出すときの戦略を開発する際に、最適化プログラムはそれらの見積もりを参考にします。

関数テンプレートの作成について詳しくは、690ページの『CREATE FUNCTION (ソースまたはテンプレート)』を参照してください。関数マッピングの作成について詳しくは、710ページの『CREATE FUNCTION MAPPING』を参照してください。関数マッピングとその設定値の説明については、1354ページの『関数マッピング・オプション』を参照してください。データ・ソース関数の呼び出しを最適化するためのガイドラインについては、 *アプリケーション開発の手引き* を参照してください。

ニックネームと列オプション

クライアント・アプリケーションが連合サーバーに分散要求を出すと、サーバーはその要求を適切なデータ・ソースに分配します。要求がデータ・ソースを指定する必要はありません。代わりに、要求はニックネームによってデータ・ソース表や視点を参照します。ニックネームは、データ・ソースでの表名および視点名にマッピングされます。このマッピングのおかげで、ニックネームをデータ・ソース名によって修飾する手間が省けます。表および視点がある場所はクライアント・アプリケーションには分かりません。

別名とは異なり、ニックネームは表や視点の代替名ではありません。ニックネームは、連合サーバーがそれに関連しているオブジェクトを参照するときのポインターです。ニックネームを定義するには、CREATE NICKNAME ステートメントを使用します。詳細については、737ページの『CREATE NICKNAME』を参照してください。

表または視点にニックネームを作成すると、最適化プログラムが表や視点へ簡単にアクセスできるようにするためのメタデータがカタログに提供されます。たとえば、表または視点の列のデータ・タイプがマッピングされた後の DB2

データ・タイプの名前がカタログに提供されます。索引付きの表にニックネームを作成した場合は、索引に関連した情報もカタログに提供されます。たとえば、索引キーの各列の名前などです。

ニックネームを作成すると、ユーザーは最適化プログラムが使用するためのメタデータをカタログにさらに提供できるようになります。たとえば、ニックネームが参照している表または視点の特定の列に含まれている値を記述するメタデータなどを提供できます。ユーザーは、*列オプション* と呼ばれるパラメーターにこのメタデータを割り当てます。例を挙げて説明しましょう。表列に数値ストリングだけが含まれている場合、 NUMERIC_STRING という列オプションに値 'Y' を割り当てると、ユーザーはそのことを示すことができます。結果として、最適化プログラムはそれらのストリングがデータ・ソースでソートされる戦略を確立できるようになるため、それらのストリングを連合サーバーに移植してそこでソートするというオーバーヘッドを減らすことができます。特に、値を含むデータベースの照合順序が連合データベースの照合順序とは異なる場合には、オーバーヘッドを大きく減らすことができます。

列オプションを定義するには、ALTER NICKNAME ステートメントを使用します。このステートメントの詳細については、495ページの『ALTER NICKNAME』を参照してください。列オプションとその設定値の説明については、1353ページの『列オプション』を参照してください。

索引指定

データ・ソース表のニックネームを作成すると、連合サーバーはその表に含まれている索引についての情報をカタログに提供します。最適化プログラムはこの情報を活用することにより、表データを容易に検索できるようにします。表に索引がない場合でも、索引定義に一般に含まれている情報をユーザーが提供することは可能です。たとえば、情報を短時間で検索できるようにするために、表のどの列を検索すべきかを指定したりできます。これを行うためには、情報を含み、表のニックネームを参照している CREATE INDEX ステートメントを実行します。

連合サーバーが認識できない索引を含んでいる表についても、ユーザーは最適化プログラムに上記と同じような情報を提供できます。たとえば、現時点では索引がないものの、あとで取得する予定のある表に対して、ニックネーム NICK1 を作成するとしましょう。また、索引を含む表を参照している視点に対して、ニックネーム NICK2 を作成するとしましょう。どちらの場合にも、連合サーバーは索引を認識できません。しかし、ユーザーが CREATE INDEX ステートメントを使用すれば、索引が存在していることをサーバーに通知することができます。 1 つのステートメントには NICK1 を参照させ、 NICK1 が参

照している表の索引に関する情報を含めます。もう 1 つのステートメントには NICK2 を参照させ、 NICK2 が指し示している視点の背後にある基礎表の索引に関する情報を含めます。

ここで説明したような状況の場合、CREATE INDEX ステートメントの情報は、索引指定と呼ばれるメタデータのセットとしてカタログに提供されます。ステートメントがニックネームを参照しているときは、索引指定だけが生成され、実際の索引は生成されないことに注意してください。詳細については、715 ページの『CREATE INDEX』および*管理の手引き: パフォーマンス* を参照してください。

分散要求

分散要求では副照会や結合副選択などの装置を使用できるため、どの表または視点の列にアクセスするかを指定したり、どのデータを検索するか指定することができます。

この節では、以下に示すシナリオの文脈に含まれている例を示します。このシナリオにおいて、連合サーバーは DB2 ユニバーサル・データベース (OS/390 版) データ・ソース、DB2 ユニバーサル・データベース (AS/400 版) データ・ソース、および Oracle データ・ソースにアクセスすることを念頭に置いて構成されています。各データ・ソースには、従業員情報を含む表が格納されています。連合サーバーは、それらの表をニックネームによって参照します。ニックネームは、表がどこに存在しているかを指し示します。それぞれのニックネームは UDB390_EMPLOYEES、AS400_EMPLOYEES、および ORA_EMPLOYEES です。 Oracle データ・ソースには、従業員情報を含む表だけでなく、各従業員が在住している国に関する情報を含む表もあります。この 2 番目の表のニックネームは ORA_COUNTRIES です。

副照会を含む要求

表 AS400_EMPLOYEES の中には、アジア地区に在住している従業員の電話番号が含まれています。この表の中には、それらの電話番号に関連した国別コードも含まれていますが、そのコードが表す国がどこであるかは示されていません。表 ORA_COUNTRIES には、コードとそれが表す国の両方が示されています。以下に示す照会は、中国を表している国別コードを調べるための副照会を使用しています。この照会は SELECT 文節と WHERE 文節を使用することによって、 AS400_EMPLOYEES に登録されている従業員のうち、中国を表す国別コードが必要な電話番号を持っている従業員をリストします。

```
SELECT NAME, TELEPHONE
  FROM DJADMIN.AS400_EMPLOYEES
  WHERE COUNTRY_CODE IN
  (SELECT COUNTRY_CODE
     FROM DJADMIN.ORA_COUNTRIES
     WHERE COUNTRY_NAME = 'CHINA')
```

上記のような分散要求をコンパイルすると、その分散要求はコンパイラーの書き直し機能によって、より簡単に最適化できるような形式へと変換されます。

結合を行うための要求

関係結合を行うと、結果セットには複数の表から取り出された列が含まれます。結果セットの行サイズを制限するための条件を必ず指定してください。

下記の照会は、2 つの表に示されている国別コードを比較することによって、従業員名とそれに対応する国名とを結合します。各表は、別々のデータ・ソースに存在しています。

```
SELECT T1.NAME, T2.COUNTRY_NAME
  FROM DJADMIN.UDB390_EMPLOYEES T1, DJADMIN.ORA_COUNTRIES T2
  WHERE T1.COUNTRY_CODE = T2.COUNTRY_CODE
```

補償

補償とは、RDBMS が SQL ステートメントをサポートしていない場合の SQL ステートメントの処理のことです。各タイプの RDBMS (DB2 ユニバーサル・データベース (AS/400 版) 、DB2 ユニバーサル・データベース (OS/390 版)、Oracle など) は、 SQL の国際規格のサブセットをサポートしています。また、一部のタイプの RDBMS は、この規格外の SQL 構成もサポートしています。 RDBMS がサポートするタイプの SQL のことを一括して *SQL ダイアレクト* といいます。 SQL 構成が DB2 の SQL ダイアレクトには含まれているものの、データ・ソースのダイアレクトには含まれていない場合は、連合サーバーがデータ・ソースの代わりにこの構成を実装できます。

例 1: DB2 の SQL には共通表式が含まれています。この文節には、全選択内のすべての FROM 文節が結果セットを参照できるようにするための名前を指定できます。連合サーバーは、Oracle の SQL ダイアレクトに共通表式が含まれていない場合でも、 Oracle データベースの共通表式を処理します。

例 2: アプリケーション内に複数のオープン・カーソルがあることをサポートしないデータ・ソースに接続する場合、連合サーバーはそのデータ・ソースへの同時接続を 1 つずつ確立することによって、この関数をシミュレートできるようにします。この関数を提供しないデータ・ソースについても、連合サーバーは同様の方法で CURSOR WITH HOLD 機能をシミュレートできるようにします。

補償を適用すれば DB2 の SQL ダイアレクトだけを使用して、連合サーバーがすべての照会をサポートするようにできます。 DB2 以外の RDBMS に固有のダイアレクトを使用する必要はありません。

パススルー

パススルー関数を使用すれば、ユーザーは、各データ・ソースに固有の SQL ダイアレクトの中のデータ・ソースとも通信できるようになります。パススルーでは、照会だけでなく、DML ステートメントや DDL ステートメントを出すこともできます。パススルー・セッション中に出されたステートメントの処理を、 DB2 やデータ・ソースがどのように管理するかについては、 1364ページの『パススルー・セッションにおける SQL 処理』を参照してください。

連合サーバーは、パススルー・セッションを管理するために次の SQL ステートメントを提供します。

SET PASSTHRU
: パススルー・セッションをオープンおよび終了します。

GRANT (サーバー特権)
: 特定のデータ・ソースに対してパススルー・セッションを開始するための権限を、ユーザー、グループ、許可 ID のリスト、または PUBLIC に付与します。

REVOKE (サーバー特権)
: パススルー・セッションを開始するための権限を取り消します。

パススルーの使用には、いくつかの制限があります。たとえば、パススルー・セッション内では、データ・ソース・オブジェクトに対してカーソルを直接オープンすることはできません。すべての制限が示されたリストについては、1365ページの『考慮事項と制約事項』を参照してください。

文字変換

ストリング は、文字を表す一連のバイトです。ストリングの中では、すべての文字が共通のコーディング表現によって表されます。場合によっては、このような文字を別のコーディング表現に変換しなければならないことがあります。この変換プロセスは*文字変換* と呼ばれます。 [12]

12. 文字変換が必要な場合は自動的に実行され、正常に終了されたならその実行はアプリケーションからは認識されません。そのため、あるステートメントの実行に関係するすべてのストリングが同じ方式で表現されている場合は、変換に関する知識は必要ありません。これには、*スタンドアロン* のインストールや、同じ国内のネットワークなどが該当します。したがって、読者の多くにとっては文字変換は無関係なものです。

文字変換は、SQL ステートメントがリモートで実行される場合に発生する可能性があります。たとえば、次の 2 つの場合を考えてみます。

- ホスト変数の値が、アプリケーション・リクエスターからアプリケーション・サーバーに送信された。
- 結果列の値が、アプリケーション・サーバーからアプリケーション・リクエスターに送信された。

どちらの場合も、送信側システムと受信側システムとでストリングの表現が違う可能性があります。また、同じシステムでのストリング操作でも変換が発生することがあります。

以下のリストは、文字変換の説明でよく使用される用語の定義です。

文字セット
定義済みの文字の集まり。 たとえば、いくつかのコード・ページには次の文字セットが出現します。

- A〜Z の 26 個の文字 (アクセント記号なし)
- a〜z の 26 個の文字 (アクセント記号なし)
- 0〜9 の数字
- . , : ; ? ( ) ' " / - _ & + % * = < >

コード・ページ
コード・ポイントに対する一連の文字の割り当て。 たとえば、コード・ページ 850 の ASCII コード化スキーマでは、 "A" にはコード・ポイント X'41' が割り当てられ、 "B" にはコード・ポイント X'42' が割り当てられています。1 つのコード・ページの中では、それぞれのコード・ポイントはただ 1 つの特定の意味をもちます。コード・ページはデータベースの 1 つの属性です。アプリケーション・プログラムがデータベースに接続している場合、データベース・マネージャーがそのアプリケーションのコード・ページを判別します。

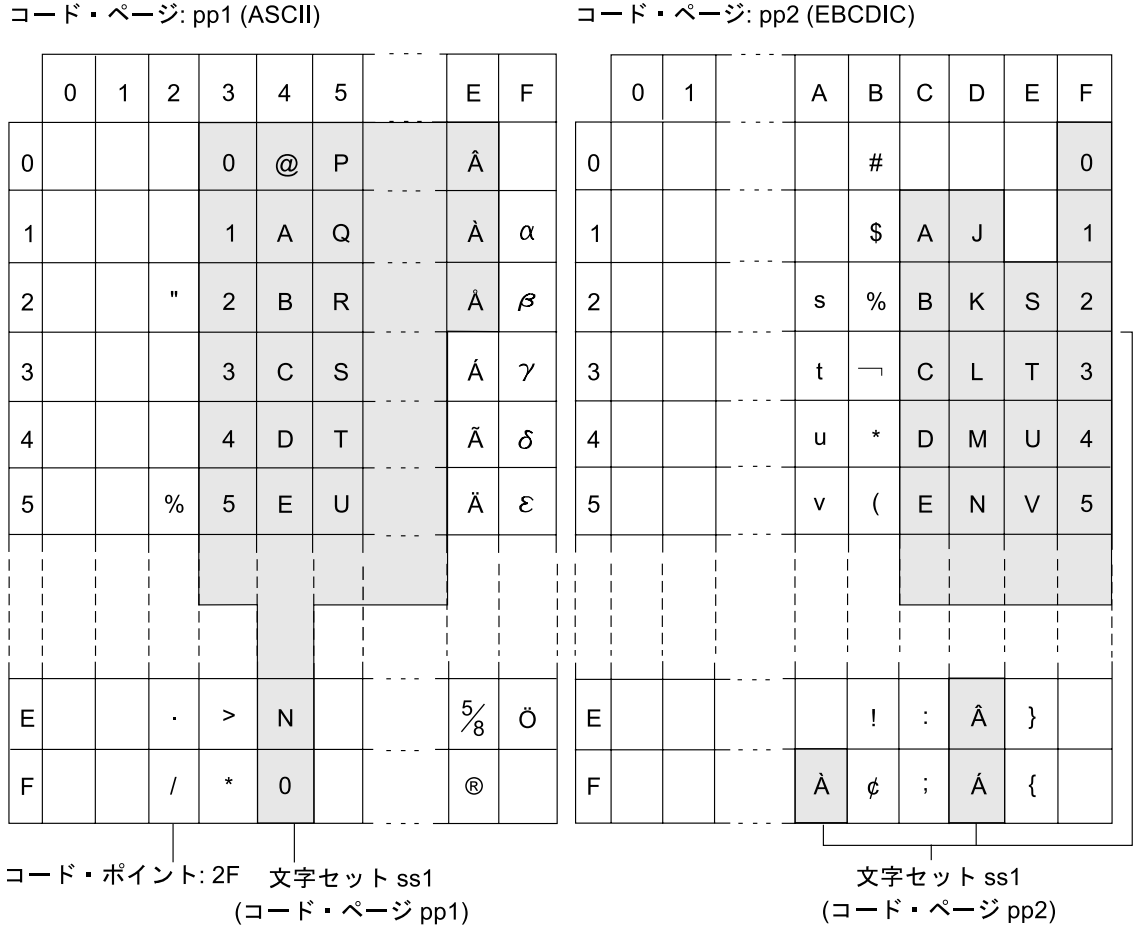
コード・ポイント
文字を表す固有のビット・パターン。

コード化スキーマ
文字データを表現するために使用する規則の集まり。 たとえば、次のとおりです。

- 1 バイト ASCII
- 1 バイト EBCDIC
- 2 バイト ASCII
- 1 バイト / 2 バイト混合 ASCII

文字セットとコード・ページ

以下の例は、典型的な文字セットが、 2 つの異なるコード・ページの異なるコード・ポイントにどのようにマップされるかを示しています。



コード化スキーマが同じでも多くの異なるコード・ページがあり、同じコード・ポイントであってもコード・ページが違うなら違う文字を表す場合があります。さらに、文字ストリングの中の 1 バイトは、 1 バイト文字セット (SBCS) の文字を表すとは限りません。文字ストリングは、混合およびビット・データにも使用されます。*混合データ* は、1 バイト文字、2 バイト文字、または多重バイト文字の混合です。ビット・データ (FOR BIT DATA か BLOB、または 2 進ストリングとして定義されている列) は、どの文字セットにも関連していません。

コード・ページ属性

データベース・マネージャーは、アプリケーションがデータベースにバインドされるときに、すべての文字ストリングのコード・ページ属性を判別します。潜在的なコード・ページ属性には、以下のものがあります。

データベース・コード・ページ
: データベース・コード・ページは、データベース構成ファイルに保管されています。このコード・ページの値はデータベースの作成時に決定され、その後の変更は不可能です。

アプリケーション・コード・ページ
: このコード・ページの下でアプリケーションが実行されます。これはアプリケーションがバインドされたときのコード・ページと同じであるとは限りません。 (アプリケーション・プログラムのバインドおよび実行の詳細については、*アプリケーション開発の手引き* を参照してください。)

コード・ページ 0
: これは、FOR BIT DATA または BLOB の値を含む式から派生したストリングを表すものです。

ストリング・コード・ページ属性

文字ストリング・コード・ページ属性は次のとおりです。

- 列は、データベース・コード・ページかコード・ページ 0 (FOR BIT DATA または BLOB と定義されている場合) です。
- 定数と特殊レジスター (USER や CURRENT SERVER など) は、データベース・コード・ページです。定数は、SQL ステートメントがデータベースにバインドされるときに、データベース・コード・ページに変換されます。
- 入力ホスト変数は、アプリケーション・コード・ページです。

スカラー操作、連結、または集合演算の結果のように、ストリング・オブジェクトを結合する操作のコード・ページ属性は、一連の規則を使用して判別されます。実行時には、コード・ページ属性を使用して、ストリングのコード・ページ変換の要件が判別されます。

文字変換の詳細は、以下を参照してください。

- ストリングの割り当てに関する規則は、 109ページの『文字ストリング割り当ての変換規則』

- 文字ストリングの比較または結合時の変換時の規則については、126ページの『ストリング変換に関する規則』

許可と特権

許可 とは、あるユーザーまたはグループに対し、データベースへの接続、表の作成、システムの管理などの一般的なタスクを実行することを認めるものです。*特権* とは、あるユーザーまたはグループに対し、ある特定のデータベース・オブジェクトに指定した方法でアクセスする権利を与えるものです。

データベース・マネージャーでは、ユーザーが特定のタスクの実行に必要な個々のデータベース機能を使用するための権限を、暗黙にまたは明示的に[13] 与えられていることが必要です。このため、ユーザーには、表を作成するには表を作成するための権限が、表を変更するには表を変更するための権限が与えられていなければなりません。他の処理についても同様です。

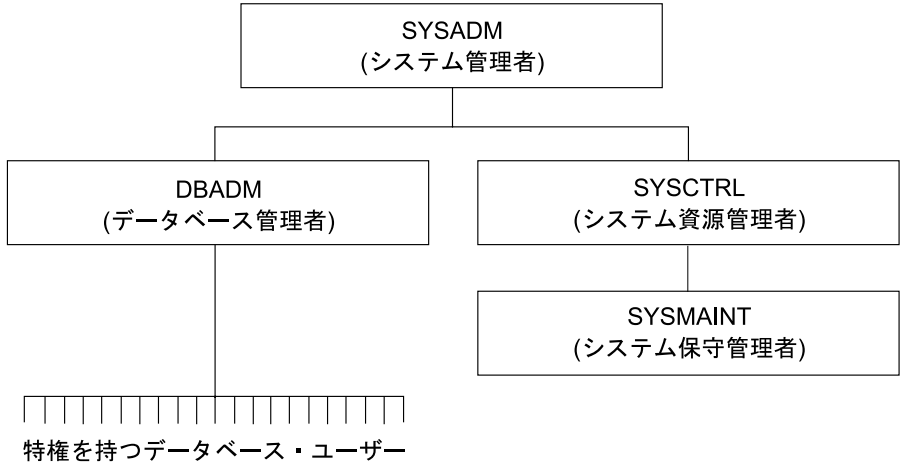


図 7. 権限と特権の階層

管理権限を有するユーザーは、データベース・マネージャーの制御を担当し、データの安全性や保全性に関する責任を負います。このようなユーザーは、誰がデータベース・マネージャーへアクセスできるか、また各ユーザーがどの程度アクセスできるかを制御します。

データベース・マネージャーは、2 つの管理権限を提供しています。

13. 明示的な権限または特権はユーザーに付与されます (GRANTEETYPE は U)。暗黙的な権限または特権は、ユーザーが属しているグループに付与されます (GRANTEETYPE は G)。

SYSADM
システム管理者権限

DBADM
データベース管理者権限

また、システム制御権限には、以下の 2 つがあります。

SYSCTRL
システム制御権限

SYSMAINT
システム保守権限

SYSADM 権限は、最高位の権限であり、データベース・マネージャーによって作成および管理されるすべての資源を制御します。 SYSADM 権限には、DBADM、 SYSCTRL、および SYSMAINT のすべての特権と、 DBADM 権限の付与または取り消しを行う権限が含まれています。

DBADM 権限は、単一のデータベースに固有の管理権限です。この権限には、オブジェクトを作成したり、データベース・コマンドを発行したり、データベースの表に含まれるデータに SQL ステートメントによってアクセスしたりする特権が含まれています。また DBADM 権限には、CONTROL および個々の特権を付与したり、取り消したりする権限も含まれています。

SYSCTRL 権限は、高水準のシステム制御権限であり、システム資源に影響を与える操作にのみ適用されます。データへの直接アクセスは許されていません。この権限には、データベースの作成、更新、または除去を行う特権、インスタンスまたはデータベースを静止させる特権、および表スペースの除去または作成を行う特権が含まれています。

SYSMAINT 権限は、第 2 レベルのシステム制御権限です。 SYSMAINT 権限を与えられたユーザーは、あるインスタンスに関連したすべてのデータベースに対して保守操作を実行することができます。データへの直接アクセスは許されていません。この権限には、データベース構成ファイルの更新、データベースまたは表スペースのバックアップ、既存のデータベースの復元、データベースのモニターを行う特権が含まれています。

データベース権限は、管理者がデータベース内でユーザーが実行することを許可する活動で、データベース・オブジェクトの特定のインスタンスに適用されない活動に適用されます。たとえば、パッケージの作成はできるが表の作成はできない権限がユーザーに与えられることがあります。

特権は、管理者またはオブジェクト所有者によって、ユーザーがデータベース・オブジェクトで実行を許されている活動に適用されます。特権が付与されたユーザーは、種々のオブジェクトを作成できますが、 SYSADM または DBADM などの権限をもつユーザーとは異なり、いくつかの制約があります。たとえば、ユーザーは表の視点は作成できるが、同じ表でトリガーは作成できない特権を与えられることがあります。特権を与えられたユーザーは、自分の所有するオブジェクトにアクセスでき、自分の所有するオブジェクトに対する特権を GRANT ステートメントによって他のユーザーに付与することができます。

CONTROL 特権の与えられたユーザーは、必要に応じて特定のデータベース・オブジェクトにアクセスしたり、そのオブジェクトに関する特権を他のユーザーに付与したり (GRANT)、取り消したり (REVOKE) できます。 CONTROL 特権を付与するには、DBADM 権限が必須です。

個々の特権およびデータベース権限は、特定の機能を許可するものですが、これと同じ特権もしくは権限を他のユーザーに付与する権利は含まれていません。表、視点、またはスキーマの特権を他のユーザーに付与する権利は、GRANT ステートメントの WITH GRANT OPTION を使って他のユーザーに広げることができます。

表スペースおよび他の記憶域構造

記憶域構造には、データベースのオブジェクトが入れられます。データベース・マネージャーによって管理される基本的な記憶域構造は、表スペースです。*表スペース* は、表、索引、大型オブジェクト、および LONG データ・タイプで定義されたデータが入っている記憶域構造です。表スペースには、次の 2 つの種類があります。

データベース管理スペース (DMS) の表スペース
: スペースがデータベース・マネージャーによって管理される表スペース。

システム管理スペース (SMS) の表スペース
: スペースがオペレーティング・システムによって管理される表スペース。

表スペースはすべて、いくつかのコンテナーで構成されます。 *コンテナー* とは、表などのオブジェクトが保管されている場所を記述するものです。たとえば、ファイル・システムのサブディレクトリーは 1 つのコンテナーです。

表スペースおよびコンテナーの詳細については、 828ページの『CREATE TABLESPACE』 または*管理の手引き* を参照してください。

表スペース・コンテナーから読み取られるデータは、*バッファー・プール* と呼ばれる記憶域に入れられます。 バッファー・プールは表スペースに関連付けられるので、どのデータがデータのバッファリングに同一記憶域を共用するかを制御することができます。バッファー・プールの詳細については、 612ページの『CREATE BUFFERPOOL』または*管理の手引き* を参照してください。

区分データベースの使用により、別々のデータベース区画にデータを拡散させることができます。どの区分が含まれるかは、表スペースに割り当てられる*ノード・グループ* によって決まります。 ノード・グループとは、データベースの一部として定義される 1 つまたは複数の区分のグループです。表スペースには、ノード・グループの各区分ごとに 1 つまたは複数のコンテナーが含まれます。*区分化マップ* が、各ノード・グループに関連付けられます。区分化マップは、特定の行のデータがノード・グループのどの区分に保管されているかを判別するのに、データベース・マネージャーによって使用されます。ノード・グループおよびデータ区分化の詳細については、次の 67ページの『複数の区分にわたるデータの区分化』、740ページの『CREATE NODEGROUP』、または*管理の手引き* を参照してください。

外部ファイルに保管されているデータへのリンクを登録している列も、表に含めることができます。このメカニズムは、DATALINK データ・タイプと呼ばれます。正規表に記録されている DATALINK 値は、外部ファイル・サーバーに保管されているファイルを指します。

DB2 データ・リンク・マネージャー (ファイル・サーバーにインストールされる) は、 DB2 と連動して、以下の任意選択機能を提供します。

- DB2 に現在リンクされているファイルが削除または名前変更されないようにする参照保全
- DATALINK 列に対して適切な SQL 特権を持っているユーザーだけが、その列にリンクされているファイルを読み取ることができるようにするセキュリティー
- ファイルのバックアップおよび回復の調整

DB2 データ・リンク・マネージャー製品は、以下の機能によって構成されています。

データ・リンク・ファイル・マネージャー
特定のファイル・サーバーにあり、DB2 にリンクされているすべてのファイルを登録します。

データ・リンク・ファイル・システム・フィルター
ファイル・システム・コマンドをフィルター処理し、登録されているファイルが削除または名前変更されないようにします。任意選択で、適正なアクセス権限があるかどうかコマンドをフィルター処理することもできます。

DB2 データ・リンク・マネージャーの詳細は、*管理の手引き* を参照してください。

複数の区分にわたるデータの区分化

DB2 では、区分化データベースの複数の区分 (ノード) にわたってデータを柔軟に拡散させることができます。ユーザーは、区分化キーを宣言することによってデータを区分する方法を選択することや、データを保管するノード・グループおよび表スペースを選択することによって、表データをスプレッドする区分およびその数を判別することができます。さらに、区分化マップ (ユーザーが更新可能) は、区分化キー値の区分へのマッピングを指定します。これにより、大きな表の区分化データベース全体について作業負荷を柔軟に均等化することができ、一方で、アプリケーションの設計者が選択すれば、 1 つまたは少数の区分に小さな表を保管することができます。ハイパフォーマンスのローカル・データ・アクセスを提供するために、各ローカル区分は、保管しているデータのローカル索引を持つことができます。

区分化データベースは、区分化記憶モデルをサポートしています。このモデルでは、区分化キーを使用して一連のデータベース区画に表データを区分化します。索引データも、それに対応する表とともに区分化され、各区分にローカルに保管されます。

区分を使用してデータベース・データを保管する前に、区分をデータベース・マネージャーに対して定義する必要があります。区分は、`db2nodes.cfg` というファイルに定義されます。区分の定義の詳細については、*管理の手引き* を参照してください。

区分ノード・グループの表スペースの表の区分化キーは、 CREATE TABLE ステートメント (または ALTER TABLE ステートメント) に指定されます。指定されていない場合、デフォルト解釈によって、表の区分化キーは、基本キーの最初の列から作成されます。基本キーが指定されていない場合、デフォルトの区分化キーは、 LONG または LOB データ・タイプ以外のデータ・タイプを持つ表に定義されている最初の列になります。区分化された表には、データ・タイプが LONG でも LOB でもない列が少なくとも 1 つは必要になりま

す。明示的に指定されている場合には、単一区分ノード・グループの表スペースの表は区分化キーだけを持つことになります。

ハッシュ区分化 は、以下のように区分に行を入れるのに使用されます。

1. ハッシュ・アルゴリズム (区分化関数) が区分化キーのすべての列に適用され、区分化マップの索引が生成されます。
2. この区分化マップ索引は、区分化マップへの索引として使用されます。区分化マップの索引の区分番号は、行が保管される区分です。
3. 区分化マップはノード・グループに関連付けられ、表はノード・グループの表スペースに作成されます。

DB2 は、*部分非クラスター化* をサポートします。これは、システム内の区分のサブセット (つまりノード・グループ) 全体で表を区分化できることを意味しています。システム内のすべての区分にわたって表を区分化する必要がありません。

区分化マップ

各ノード・グループは、 4 096 の区分番号の配列である*区分化マップ* に関連付けられています。表の行ごとに区分化関数によって生成された区分化マップの索引は、行が保管される区分を判別するために区分化マップへの索引として使用されます。

69ページの図8 は、区分化キー値 (c1、c2、c3) のある行が区分化マップの索引 2 にマップされ、この索引 2 が区分 p5 を参照する様子を示しています。

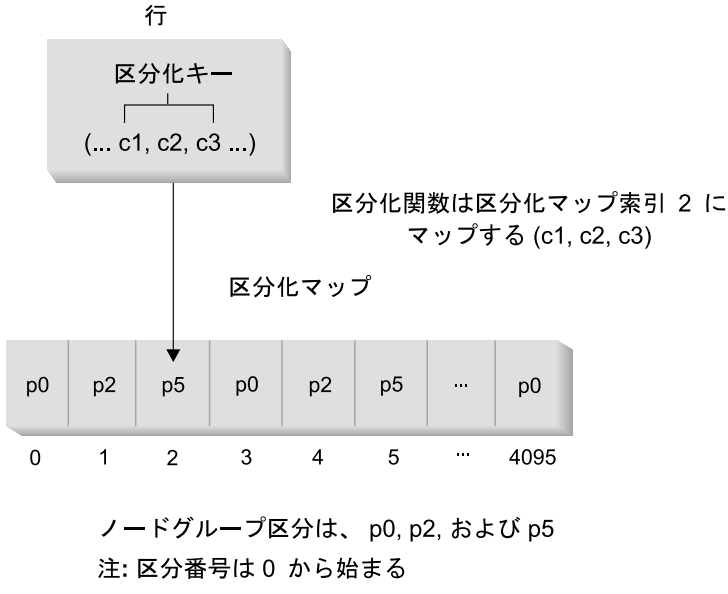


図 8. データ配分

区分化マップを変更することができ、それにより、区分化キーまたは実際のデータを変更することなくデータ配分を変更することができます。新しい区分化マップは、この区分化マップを使用してノード・グループの表を再配分する REDISTRIBUTE NODEGROUP コマンドまたは API の一部として指定されます。詳細については、*コマンド解説書* または *管理 API 解説書* を参照してください。

表の連結

DB2 は、結合または副照会でアクセスしたデータが同一ノード・グループの同一区分にある場合、それを認識することができます。これが起きると、DB2 は、データが保管されている区分で結合または副照会の処理を実行することができます。多くの場合、これはパフォーマンスの面で大きな利点があります。このような状況のことを、表の連結といいます。表が連結されるには、以下の条件が必要です。

- 表が同一ノード・グループになければなりません。 (すなわち、再配分されていてはいけません。[14])
- 表の列の区分化キーが同数でなければなりません。

14. ノード・グループを再配分する間、ノード・グループ内の表が別々の区分化マップを使用する、つまり連結されない場合があります。

- 区分化キーの対応する列に、区分の面で互換性がなければなりません (128ページの『区分の互換性』 を参照)。

または

- 表が、同一区分に定義されている単一の区分ノード・グループになければなりません。

同一の区分化キー値を使用して連結されている表の行は、同一区分に置かれます。

第3章 言語要素

この章では、SQL と多くの SQL ステートメントに共通する言語要素の基本構文を定義します。

文字

SQL 言語のキーワードと演算子で使う基本的な記号は、すべての IBM 文字セットの一部である 1 バイト文字です。言語の文字は、英字、数字、または特殊文字に分類されます。

文字

文字 とは、26 個の大文字 (A〜Z) および 26 個の小文字 (a〜z)、さらに 3 個の文字 ($、#、および @) のいずれかです。これらの特殊文字は、ホスト・データベース製品との互換性を保つために含まれています (たとえば、コード・ページ 850 では、$ は X'24'、# は X'23'、 @ は X'40' にあります)。英字には、拡張文字セットのアルファベット文字も含まれます。拡張文字セットには、追加のアルファベット文字が含まれています。たとえば、分音符号 (´ は分音符号の一例です) の付いたアルファベット文字です。使用可能な文字は、使用するコード・ページによって異なります。

数字 は、0 〜 9 のいずれかの文字です。

特殊文字 は、以下のいずれかの文字です。

	ブランク	-	負符号
"	引用符または二重引用符	.	ピリオド
%	パーセント	/	スラッシュ
&	アンパーサンド	:	コロン
'	アポストロフィ (') または一重引用符	;	セミコロン
(	左括弧	<	不等号 (より小さい)
)	右括弧	=	等号
*	アスタリスク	>	不等号 (より大きい)
+	正符号	?	疑問符
,	コンマ	_	下線
|	縦線	^	脱字記号
!	感嘆符		

MBCS についての考慮事項

マルチバイト文字はすべて文字として扱われます。ただし、特殊文字である 2 バイト・ブランクは例外です。

トークン

言語の基本的な構文単位は、*トークン* と呼ばれます。トークンは、1 つまたは複数の一連の文字です。ブランク文字を含めることのできるストリング定数または区切り識別子の場合を除いて、トークンにブランク文字を含めることはできません。 (これらの用語については後で定義します。)

トークンは、*通常トークン* と *区切り文字トークン* に分類されます。

- *通常トークン* とは、数値定数、通常識別子、ホスト識別子、またはキーワードです。

例

```
1       .1       +2       SELECT       E       3
```

- *区切りトークン* とは、ストリング定数、区切り識別子、演算子記号、または構文図に示される特殊文字です。 1042ページの『PREPARE』で説明されているように、パラメーター・マーカーとして機能する場合は疑問符も区切りトークンです。

  例

```
,       'string'       "fld1"       =       .
```

スペース: スペースは、1 つまたは複数の一連のブランク文字です。ストリング定数と区切り識別子以外のトークンには、スペースを含めることができません。トークンの後にはスペースを続けることができます。すべての通常トークンの後には、スペースか、または構文で許されているなら区切りトークンを付ける必要があります。

コメント: 静的 SQL ステートメントには、ホスト言語のコメントまたは SQL コメントが含まれている場合があります。どちらのタイプのコメントも、スペースを指定できる場所であればどこにでも指定可能ですが、区切りトークンの中または EXEC と SQL キーワードの間には指定できません。 SQL のコメントは、2 つの連続するハイフン (--) で始まり、行末で終わります。詳しくは、492ページの『SQL コメント』を参照してください。

大文字と小文字: どのトークンにも小文字を含めることができますが、通常トークンでの小文字は大文字に変換されます。ただし、識別子の大文字と小文字を区別する C 言語のホスト変数は例外です。区切りトークンが大文字に変換されることはありません。したがって、次のステートメントは、

```
select * from EMPLOYEE where lastname = 'Smith';
```

大文字に変換した後は、以下のステートメントと同等になります。

```
SELECT * FROM EMPLOYEE WHERE LASTNAME = 'Smith';
```

MBCS についての考慮事項

マルチバイトの英文字は大文字変換されません。 a ～ z の 1 バイト文字は大文字に変換されます。

識別子

識別子 とは、名前の形成に使用されるトークンです。 SQL ステートメントの識別子は、SQL 識別子かホスト識別子のいずれかです。

SQL 識別子

SQL 識別子には、*通常識別子* と*区切り識別子* の 2 つのタイプがあります。

- *通常識別子* は、英字の後にゼロ個またはそれ以上の文字が続いている識別子です。各文字は、英大文字、数字、または下線文字です。通常識別子は予約語と同一であってはなりません。 (予約語については、1389ページの『付録 H. 予約スキーマ名と予約語』 を参照してください。)
- *区切り識別子* は、 1 文字以上の文字を引用符 (″) で囲んだ識別子です。区切り識別子の中で 1 つの引用符を表す場合には、 2 つの連続した引用符を使用します。この方法で、識別子に小文字を含めることができます。

通常識別子と区切り識別子の例

```
WKLYSAL    WKLY_SAL    "WKLY_SAL"    "WKLY SAL"    "UNION"    "wkly_sal"
```

2 バイトのコード・ページで生成され、マルチバイトのコードのアプリケーションやデータベースにより使用される識別子間の文字変換に関して特殊な考慮事項が存在します。マルチバイトへ変換した後に、そのような識別子が識別子の制限長を超えることがあります (詳細については、 1453ページの『付録O. 日本語および繁体字中国語 EUC についての考慮事項』を参照してください)。

ホスト識別子

ホスト識別子 は、ホスト・プログラムで宣言されている名前です。ホスト識別子の規則は、ホスト言語の規則に従います。ホスト識別子は 255 文字以下でなければならず、また、大文字小文字を問わず 'SQL' または 'DB2' で始まっていてはなりません。

命名規則と暗黙オブジェクト名修飾

名前を形成するための規則は、その名前で指定されるオブジェクトの種類によって異なります。データベース・オブジェクト名は、1 つの識別子で構成されているか、 2 つの識別子で構成されるスキーマ修飾オブジェクトとなっています。スキーマ修飾オブジェクト名は、スキーマ名なしで指定することができます。この場合、スキーマ名は暗黙指定になります。

動的 SQL ステートメントでは、スキーマ修飾オブジェクト名は、修飾子のないオブジェクト名参照の修飾子として CURRENT SCHEMA 特殊レジスター値を暗黙的に使用します。デフォルトでは、この値は現行の許可 ID に設定されています。詳細については、 1129ページの『SET SCHEMA』を参照してください。 SQL ステートメントが DYNAMICRULES BIND オプションを使用してバインドされたパッケージからのものである場合、修飾に CURRENT SCHEMA 特殊レジスターは使用されません。 DYNAMICRULES BIND パッケ

ージでは、動的 SQL ステートメント内にある修飾されていないオブジェクト参照子を修飾するための値として、パッケージのデフォルト修飾子が使用されます。

静的 SQL ステートメントでは、 QUALIFIER プリコンパイル / バインド・オプションにより、未修飾オブジェクト名の修飾子が暗黙指定されます。デフォルトでは、この値はパッケージの許可 ID に設定されています。詳細については、*コマンド解説書* を参照してください。

構文図では、異なる種類の名前には異なる用語を使用しています。以下のリストでそのような用語を定義します。各種の識別子の最大長については、1203ページの『付録A. SQL の制限値』を参照してください。

alias-name
別名を指定するスキーマ修飾名。

attribute-name
構造化データ・タイプを指定する識別子。

authorization-name
ユーザーまたはグループを指定する識別子。文字に関しては以下の制限が適用されます。

- 有効な文字は、A〜Z、a〜z、0〜9、#、@、$、および _ です。
- 名前は、文字 'SYS'、'IBM'、または 'SQL' で始めることはできません。
- 名前は、ADMINS、GUESTS、LOCAL、PUBLIC、または USERS であってはなりません。
- 区切り許可 ID に小文字を含めることはできません。

bufferpool-name
バッファー・プールを指定する識別子。

column-name
表または視点の列を指定する修飾子付きまたは修飾子のない名前。修飾子は、表名、視点名、ニックネーム、または相関名です。

condition-name
SQL プロシージャーの条件を指定する識別子。

constraint-name
参照制約、基本キー制約、固有制約、または表検査制約を指定する識別子。

correlation-name
結果表を指定する識別子。

cursor-name
SQL カーソルを指定する識別子。ホストとの

	互換性を保つため、この名前にハイフン文字を使用することもできます。
data-source-name	データ・ソースを指定する識別子。この識別子は、リモート・オブジェクト名を構成する 3 部分のうちの 1 番目です。
descriptor-name	コロンの後に、SQL 記述子域 (SQLDA) を指定するホスト識別子を付けたもの。ホスト識別子の説明については、 151ページの『ホスト変数の参照』を参照してください。記述子名には標識変数は含まれません。
distinct-type-name	特殊タイプ名を指定する修飾子付きまたは修飾子のない名前。 SQL ステートメントでの修飾子のない特殊タイプ名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。
event-monitor-name	事象モニターを指定する識別子。
function-mapping-name	関数マッピングを指定する識別子。
function-name	関数を指定する、修飾子付きまたは修飾子のない名前。 SQL ステートメントにおける修飾子のない関数名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。
group-name	構造タイプ用に定義した変換グループを指定する、修飾子のない識別子。
host-variable	ホスト変数を指定するトークンを連結したもの。 151ページの『ホスト変数の参照』に説明されているように、ホスト変数には少なくとも 1 つのホスト識別子が含まれます。
index-name	索引または索引指定を指定するスキーマによって修飾された名前。
label	SQL プロシージャーのラベルを指定する識別子。
method-name	メソッドを指定する識別子。メソッドのスキーマ・コンテキストは、そのメソッドのサブジェ

	クト・タイプ (または、サブジェクト・タイプのスーパータイプ) のスキーマによって決まります。
nickname	連合サーバーが表または視点に参照することを指定する、スキーマによって修飾された名前。
nodegroup-name	ノードグループを指定する識別子。
package-name	パッケージを指定するスキーマ修飾名。
parameter-name	プロシージャー、ユーザー定義関数、メソッド、または索引拡張機能で参照できるパラメーターを指定する識別子。
procedure-name	プロシージャーを指定する修飾子付きまたは修飾子のない名前。 SQL ステートメントにおける修飾子のないプロシージャー名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。
remote-authorization-name	データ・ソースのユーザーを指定する識別子。許可名に関する規則は、データ・ソースごとに異なります。
remote-function-name	データ・ソース・データベースに登録されている関数を指定する名前。
remote-object-name	データ・ソース表または視点を指定するとともに、その表または視点が存在しているデータ・ソースを識別する 3 部分名。この名前は、データ・ソース名、リモート・スキーマ名、およびリモート表名の 3 部分で構成されます。
remote-schema-name	データ・ソース表または視点が属するスキーマを指定する名前。この名前は、リモート・オブジェクト名を構成する 3 部分のうちの 2 番目です。
remote-table-name	データ・ソースにある表または視点を指定する名前。この名前は、リモート・オブジェクト名を構成する 3 部分のうちの 3 番目です。
remote-type-name	データ・ソース・データベースがサポートするデータ・タイプ。システムに組み込まれたタイ

プの場合は、長形式を使用しないでください (たとえば、 CHARACTER ではなく CHAR を使用してください)。

savepoint-name
保管点を指定する識別子。

schema-name
SQL オブジェクトを論理的にグループ化するための識別子。オブジェクト名の修飾子として使用されるスキーマ名は、以下のものから暗黙的に決定されます。

- CURRENT SCHEMA 特殊レジスターの値
- QUALIFIER プリコンパイル / バインド・オプションの値
- CURRENT PATH 特殊レジスターを使用する解決アルゴリズムに基づいて
- 同一の SQL ステートメントにある別のオブジェクトのスキーマ名に基づいて

混乱を避けるために、スキーマとしてスキーマ名「SESSION」を使わないようお勧めします。ただし、宣言済みのグローバル一時表のスキーマは除きます (この場合は、スキーマ名「SESSION」を使用する必要があります)。

server-name
アプリケーション・サーバーを指定する識別子。連合システムでは、server-name によってデータ・ソースのローカル名も指定します。

specific-name
固有名を指定する、修飾子付きまたは修飾子のない名前。 SQL ステートメントにおける修飾子のない固有名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。

SQL-variable-name
SQL プロシージャー・ステートメントのローカル変数名を定義します。 SQL 変数名は、ホスト変数名が許可されている他の SQL ステートメントで使うこともできます。この名前に対しては、SQL 変数を宣言した複合ステートメントのラベルで修飾できます。

statement-name
作成済み SQL ステートメントを指定する識別子。

supertype-name	タイプ名のスーパータイプを指定する修飾子付きまたは修飾子のない名前。 SQL ステートメントにおける修飾子のないスーパータイプ名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。
table-name	表を指定するスキーマによって修飾された名前。
tablespace-name	表スペースを指定する識別子。
trigger-name	トリガーを指定するスキーマによって修飾された名前。
type-mapping-name	データ・タイプ・マッピングを指定する識別子。
type-name	タイプ名を指定する修飾子付きまたは修飾子のない名前。 SQL ステートメントにおける修飾子のないタイプ名は、文脈に応じてデータベース・マネージャーによって暗黙のうちに修飾されます。
typed-table-name	タイプ付き表を指定するスキーマによって修飾された名前。
typed-view-name	タイプ付き視点を指定するスキーマによって修飾された名前。
view-name	視点を指定するスキーマによって修飾された名前。
wrapper-name	ラッパーを指定する識別子。

別名

表の別名は、表または視点の代替名とみなすことができます。このため、SQL ステートメントの中で表または視点は、名前か表別名のどちらかで参照できます。

別名は、表名または視点名を使用できる場所であればどこででも使用できます。別名は、オブジェクトが存在していなくても作成することができます (ただし、オブジェクトを参照するステートメントがコンパイルされる時点では存在している必要があります)。別名連鎖の中に循環参照または反復参照がない限り、他の別名を別名によって参照することができます。別名で参照できるのは、同じデータベース内の表、視点、別名だけです。別名は、CREATE

TABLE や CREATE VIEW ステートメントのような、新しい表名または視点名の指定が予測される場所では使用できません。たとえば、 PERSONNEL という別名を作成した後で CREATE TABLE PERSONNEL... のような使い方をするとエラーとなります。

構文図や SQL ステートメントの説明では、別名によって表または視点を参照するというオプションは明示的には示されません。

修飾子なしの新しい別名に、既存の表、視点、または別名と同じ完全修飾名を付けることはできません。

SQL ステートメントで別名を使用する効果は、テキスト置換の効果に似ています。別名は SQL ステートメントのコンパイル時には定義されている必要があり、ステートメントのコンパイル時には修飾子付きの基礎表名または視点名によって置換されます。たとえば、PBIRD.SALES が DSPN014.DIST4_SALES_148 の別名である場合に、コンパイル時には、

```
SELECT * FROM PBIRD.SALES
```

は、実際には次のようになります。

```
SELECT * FROM DSPN014.DIST4_SALES_148
```

連合システムでは、上記のような使用法と制限は、表の別名だけでなくニックネームが表す別名にも適用されます。したがって、ニックネームの代わりにニックネームの別名を SQL ステートメントで使用することも可能です。別名を参照するステートメントをコンパイルする前にニックネームを作成する場合、まだ存在していないニックネームに別名をあらかじめ作成しておくこともできます。あるニックネームの別名に、そのニックネームの他の別名を参照させることもできます。

他のリレーショナル・データベース管理システム・アプリケーションの構文を受け入れるために、 CREATE ALIAS ステートメントと DROP ALIAS ステートメントにおいては、 ALIAS の代わりに SYNONYM を使用できるようになっています。

別名の詳細については、608ページの『CREATE ALIAS』を参照してください。

許可 ID と許可名

許可 ID とは、データベース・マネージャーとアプリケーション・プロセスとの間、またはデータベース・マネージャーとプログラム作成処理との間の接続が確立されるときに、データベース・マネージャーが獲得する文字ストリングのことです。これは、特権の集合を指定するものです。ユーザーやユーザー・グループを指す場合もありますが、この特性はデータベース・マネージャーからは制御されません。

許可 ID は、データベース・マネージャーにより以下の目的で使用されます。

- SQL ステートメントの許可検査
- QUALIFIER プリコンパイル / バインド・オプションと CURRENT SCHEMA 特殊レジスターのデフォルト値。許可 ID は、デフォルトの CURRENT PATH 特殊レジスターと FUNCPATH プリコンパイル / バインド・オプションにも含まれています。

許可 ID はすべての SQL ステートメントに適用されます。静的 SQL ステートメントに適用される許可 ID は、プログラムのバインディングの過程で使用される許可 ID です。動的 SQL ステートメントに適用される許可 ID は、その動的 SQL ステートメントを発行したパッケージに対してバインド時に提供された DYNAMICRULES オプションによって決まります。 DYNAMICRULES RUN を使用してバインドされたパッケージの場合は、パッケージを実行するユーザーの許可 ID が許可 ID として使用されます。 DYNAMICRULES BIND を使用してバインドされたパッケージの場合は、パッケージの許可 ID が許可 ID として使用されます。これは*実行時許可 ID* と呼ばれます。

SQL ステートメントで指定される*許可名* を、そのステートメントの許可 ID と混同してはなりません。許可名は、種々の SQL ステートメントで使用される識別子です。許可名は、スキーマの所有者を指定するために CREATE SCHEMA ステートメントで使用されます。許可名は、付与または取り消しの対象を指定するために GRANT および REVOKE ステートメントで使用されます。 *X* に特権を付与するということは、それ以降、その特権を必要とするステートメントの許可 ID が、 *X* またはグループ X のメンバーになることが前提となっていることに注意してください。

例:

- ユーザー ID が SMITH であり、またデータベース・マネージャーがアプリケーション・プロセスとの接続を確立したときに獲得した許可 ID も SMITH であるとします。以下のステートメントは対話的に実行されます。

```
GRANT SELECT ON TDEPT TO KEENE
```

SMITH はこのステートメントの許可 ID です。したがって、動的 SQL ステートメントでの CURRENT SCHEMA 特殊レジスターのデフォルト値と、静的 SQL でのデフォルトの QUALIFIER プリコンパイル / バインド・オプションは SMITH です。このため、このステートメントを実行できる権限は、SMITH に対して検査され、 SMITH が 74ページの『命名規則と暗黙オブジェクト名修飾』で説明されている修飾規則に基づく *table-name* 暗黙修飾子となります。

KEENE はこのステートメントでの許可名です。 KEENE には SMITH.TDEPT に対する SELECT 特権を付与されます。

- SMITH が管理権限を持っており、セッション中に SET SCHEMA ステートメントが発行されない、以下の動的 SQL ステートメントの許可 ID であるとします。

  ```
  DROP TABLE TDEPT
  ```

  は、SMITH.TDEPT 表を削除します。

  ```
  DROP TABLE SMITH.TDEPT
  ```

  は、SMITH.TDEPT 表を削除します。

  ```
  DROP TABLE KEENE.TDEPT
  ```

  これは、KEENE.TDEPT 表を削除することになります。 KEENE.TDEPT と SMITH.TDEPT は別の表であることに注意してください。

  ```
  CREATE SCHEMA PAYROLL AUTHORIZATION KEENE
  ```

  KEENE は、PAYROLL と呼ばれるスキーマを作成するステートメントで指定されている許可名です。 KEENE は、スキーマ PAYROLL の所有者であり、 CREATEIN、ALTERIN、および DROPIN 特権が与えられ、このような特権を他のユーザーに付与することができます。

実行時における動的 SQL の特性

BIND コマンドと PRECOMPILE コマンドの OWNER オプションは、パッケージの許可 ID を定義します。

BIND コマンドと PRECOMPILE コマンドの QUALIFIER オプションは、パッケージに含まれる未修飾オブジェクトの暗黙修飾子を指定します。

BIND コマンドと PRECOMPILE コマンドの DYNAMICRULES オプションは、動的 SQL ステートメントを実行時に処理するときに、実行時の規則である DYNAMICRULES RUN とバインド時の規則である DYNAMICRULES BIND のどちらを適用するかを決定します。これらの規則は、許可 ID として

使用する値の設定と、動的 SQL ステートメントに含まれる未修飾オブジェクト参照の暗黙修飾子として使用する値の設定を示します。 DYNAMICRULES オプションには、以下の表に示されている作用があります。

表 1. OWNER および QUALIFIER が影響を与える静的 SQL の特性

機能	OWNER だけが指定された場合	QUALIFIER だけが指定された場合	QUALIFIER と OWNER の両方が指定された場合
使用される許可 ID	BIND コマンドの OWNER オプションに指定されたユーザーの ID	パッケージをバインドするユーザーの ID	BIND コマンドの OWNER オプションに指定されたユーザーの ID
使用される未修飾オブジェクトの修飾値	BIND コマンドの OWNER オプションに指定されたユーザーの ID	BIND コマンドの QUALIFIER オプションに指定されたユーザーの ID	BIND コマンドの QUALIFIER オプションに指定されたユーザーの ID

表 2. DYNAMICRULES、OWNER、および QUALIFIER が影響を与える動的 SQL の特性

機能	RUN	BIND
使用される許可 ID	パッケージを実行するユーザーの ID	パッケージに適用される許可 ID
使用される未修飾オブジェクトの修飾値	CURRENT SCHEMA 特殊レジスター	パッケージに適用される許可 ID

DYNAMICRULES オプションを使用するときは、以下の点を考慮してください。

- DYNAMICRULES BIND によってバインドされたパッケージから実行される動的 SQL ステートメントに含まれている未修飾オブジェクト参照を修飾するために、 CURRENT SCHEMA 特殊レジスターが使用されることはありません。そのような未修飾オブジェクト参照については、上記の表に示されている規則に従ってパッケージのデフォルト修飾子が使用されます。このことは、 CURRENT SCHEMA 特殊レジスターを変更するために SET CURRENT SCHEMA ステートメントを発行した後にもあてはまります。レジスター値は変更されますが、使用されることはありません。
- DYNAMICRULES BIND オプションを使用してバインドされたパッケージ内では、動的に準備された SQL ステートメントを使用できません。つまり、そのようなパッケージの中では、GRANT、REVOKE、ALTER、CREATE、DROP、COMMENT ON、RENAME、SET INTEGRITY、 SET EVENT MONITOR STATE、およびニックネームを参照する照会は使用できません。

- 1 回の接続で複数のパッケージが参照される場合、動的 SQL は、ステートメントがバインドされているパッケージに指定された BIND オプションに従って機能します。
- DYNAMICRULES BIND を使用してパッケージをバインドするときは、ユーザーに付与したくない許可をバインダーが付与してしまわないように注意してください。動的ステートメントはパッケージ所有者の許可 ID を使用します。

許可 ID とステートメントの準備

BIND 時に VALIDATE BIND を指定する場合、バインド実行時に、表や視点を扱うときに必要な特権が存在していなければなりません。特権または参照オブジェクトが存在せず、SQLERROR NOPACKAGE が有効である場合、バインド操作は失敗します。 SQLERROR CONTINUE を指定する場合、バインドは成功し、エラーのあるステートメントにはフラグが立てられます。エラーのフラグが立てられたステートメントを実行しようとすると、アプリケーションでエラーが生じます。

VALIDATE RUN でパッケージがバインドされると、通常の BIND 処理はすべて完了しますが、この時点では、アプリケーションで参照する表や視点を使うときに必要な特権は、存在していなくてもかまいません。ステートメントに必要な特権がバインド実行時に存在しない場合、アプリケーション内でステートメントを初めて実行するときに、必ず増分バインドが実行されます。このときには、ステートメントに必要なすべての特権が存在していなければなりません。何らかの特権が存在しない場合、ステートメントを実行しても失敗します。実行時に許可検査が行われるときには、パッケージ所有者の許可 ID を使用して行われます。

データ・タイプ

列のデータ・タイプの指定については、771ページの『CREATE TABLE』 を参照してください。

SQL で扱うことのできる一番小さいデータの単位は*値* です。 値の解釈方法は、値の出所 (ソース) のデータ・タイプによって異なります。値の出所は、以下のとおりです。

- 定数
- 列
- ホスト変数
- 関数

- 式
- 特殊レジスター

DB2 は、この項で説明されている複数の組み込みデータ・タイプをサポートしています。また、ユーザー定義のデータ・タイプもサポートします。ユーザー定義のデータ・タイプについては、97ページの『ユーザー定義タイプ』を参照してください。

図9 は、サポートされる組み込みデータ・タイプを示しています。

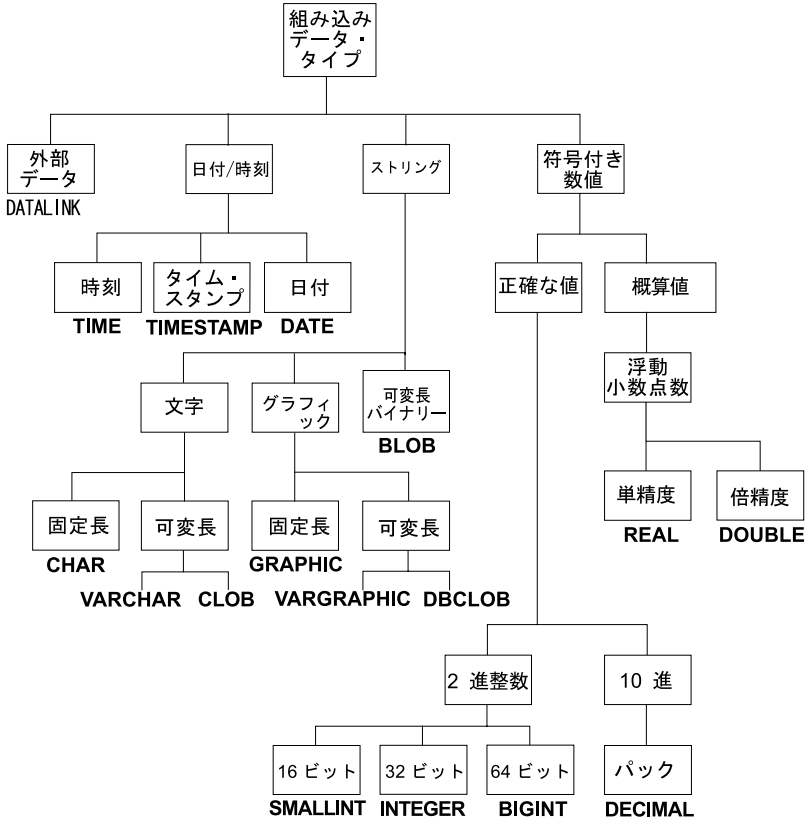


図9. サポートされる組み込みデータ・タイプ

ヌル値

すべてのデータ・タイプには、ヌル値が含まれています。ヌル値とは、すべての非ヌル値と区別されていて、それによって (非ヌル) 値がないことを指し示すための特殊値のことです。すべてのデータ・タイプにはヌル値が含まれますが、 NOT NULL として定義されている列にヌル値を入れることはできません。

ラージ・オブジェクト (LOB)

ラージ・オブジェクト および総称頭字語である *LOB* は、 BLOB、CLOB、または DBCLOB のデータ・タイプを参照するときに使用されます。 LOB 値は、 89ページの『可変長文字ストリングの使用制限』で指定されている LONG VARCHAR 値に適用される制限に従います。 LOB ストリングに対するこのような制限は、ストリングの長さ属性が 254 バイト以下であっても適用されます。

文字ラージ・オブジェクト (CLOB) ストリング

文字ラージ・オブジェクト *(CLOB)* は、最長 2 ギガバイト (2 147 483 647 バイト) のバイト単位の可変長ストリングです。 *CLOB* は、単一文字セットで記述された文書などのラージ SBCS、または混合 (SBCS および MBCS) 文字ベース・データを保管するのに使用されます (したがって、それに関連する SBCS または混合コード・ページがあります)。 CLOB は文字ストリングとして扱われることに注意してください。

2 バイト文字ラージ・オブジェクト (DBCLOB) ストリング

2 バイト文字ラージ・オブジェクト *(DBCLOB)* は、最長 1 073 741 823 文字の 2 バイト文字の可変長ストリングです。 *DBCLOB* は、単一文字セットで記述された文書のような、大規模な DBCS 文字ベースのデータの保管に使用されます (したがって、 DBCLOB にはそれに関連する DBCS の CCSID があります)。 DBCLOB は漢字ストリングとして扱われることに注意してください。

2 進ラージ・オブジェクト (BLOB)

2 進ラージ・オブジェクト *(BLOB)* は、最長 2 ギガバイト (2 147 483 647 バイト) のバイト単位の可変長ストリングです。 *BLOB* は、主に、ピクチャー、音声、マルチメディアなど、従来はあまりなかったデータの保持を意図しています。また、ユーザー定義タイプおよびユーザー定義関数で活用するために、構造化データを保持する目的にも使用されます。 FOR BIT DATA 文字ストリングと同じように、 BLOB ストリングに対応する文字セットはありません。

ロケーターによるラージ・オブジェクト (LOB) の操作

LOB 値は非常に大きいため、この値をデータベース・サーバーからクライアント・アプリケーション・プログラムのホスト変数に転送するには多くの時間がかかります。しかし、アプリケーションが一度に処理するのは通常は LOB 値の全体ではなく、小さな部分だけであるのがほとんどです。アプリケーションが LOB 値全体をアプリケーション・メモリーに保管することを必要としない (または保管したくない) 場合、アプリケーションはラージ・オブジェクト・ロケーター (LOB ロケーター) によって 1 つの LOB 値を参照することができます。

ラージ・オブジェクト・*ロケーター* つまり LOB ロケーターは、データベース・サーバーの単一 LOB 値を表す値を伴うホスト変数です。 LOB ロケーターは、ユーザーが、アプリケーション・プログラムの動作しているクライアント・マシンに LOB 値全体を保管することなく、アプリケーション・プログラムで非常に大規模なオブジェクトを簡単に操作できるようにするためのメカニズムとして開発されました。

たとえば 1 つの LOB 値を選択する場合、アプリケーション・プログラムは LOB 値全体を選択し、それを同じように大きなホスト変数に入れることもできますが (これは、アプリケーション・プログラムが一度に LOB 値全体を処理しようとする場合に適しています)、 LOB ロケーターに入る LOB 値だけを選択することもできます。その後、アプリケーション・プログラムは LOB ロケーターを使用して、そのロケーター値を入力として指定することによって、その LOB 値に対する以降のデータベース操作 (スカラー関数 SUBSTR、CONCAT、 VALUE、 LENGTH の適用、割り当ての実行、 LIKE または POSSTR による LOB の探索、LOB に対する UDF の適用など) を行うことができます。クライアントのホスト変数に割り当てられるデータ量など、ロケーター操作による出力結果は、多くの場合、入力 LOB 値の小さいサブセットとなります。

LOB ロケーターは、基本値以外のものを表現する場合もあり、 LOB 式に対応する値を表現することもできます。たとえば、LOB ロケーターで、次の式に対応する値を表現できます。

```
SUBSTR( <lob 1> CONCAT <lob 2> CONCAT <lob 3>, <start>, <length> )
```

アプリケーション・プログラムの通常のホスト変数では、そのホスト変数にヌル値が選択されている場合、標識変数は値がヌル値であることを示す -1 に設定されます。しかし、LOB ロケーターの場合は、標識変数の意味が少し違います。ロケーター・ホスト変数自体はヌル値にすることができないので、標識変数の負の値は、その LOB ロケーターが表す LOB 値がヌル値であることを示

します。標識変数の値により、ヌル値情報はクライアントにとってローカルに保持されます。サーバー側では有効なロケーターによってヌル値を追跡しません。

LOB ロケーターが表すのは 1 つの値であって、データベースの行や場所を表すわけではない、ということは重要です。値がロケーターに選択されると、ロケーターが参照する値に影響を及ぼすような操作を、元の行や表に対して実行することはできません。ロケーターに対応する値は、トランザクションが終了するか、ロケーターが明示的に解放されるかする時まで有効です。ロケーターでは、この機能を実現するためにデータのコピーなどを行ったりはしません。その代わりに、ロケーター・メカニズムに基本 LOB 値の内容が保管されます。 LOB 値 (または、上記のように式) の具体化は、 LOB 値が実際に何らかの位置を割り当てられるまで延期されます。すなわち、ホスト変数の形式でユーザー・バッファーに入れられるか、もしくはデータベースの別のレコードのフィールド値に入れられるまでです。

LOB ロケーターは、トランザクションの中で LOB 値を参照するための唯一のメカニズムです。 LOB ロケーターはそれが作成されたトランザクションを超えて存続することはありません。また、LOB ロケーターはデータベース・タイプではなく、データベースに保管されることはありません。したがって、視点や検査制約には加わりません。しかし、ロケーターは LOB タイプのクライアント側の表現なので、 FETCH、OPEN、および EXECUTE ステートメントで使用される SQLDA 構造の中で記述されるよう、 LOB ロケーターの SQLTYPE が用意されています。

文字ストリング

文字ストリング は、一連のバイトです。ストリングの長さは、その一連のバイトのバイト数です。長さがゼロの場合、その値は*空ストリング* と呼ばれます。この値をヌル値と混同しないようにしてください。

固定長文字ストリング

固定長ストリングの列の値は、すべて同じ長さです。この長さは、その列の長さ属性によって決定されます。 長さ属性は、1 以上 254 以下でなければなりません。

可変長文字ストリング

可変長文字ストリングには、VARCHAR、LONG VARCHAR、および CLOB の 3 つのタイプがあります。

- VARCHAR タイプは、最長 32 672 バイトの可変長ストリングです。
- LONG VARCHAR タイプは、最長 32 700 バイトの可変長ストリングです。

- CLOB タイプは、最長 2 ギガバイトの可変長ストリングです。

可変長文字ストリングの使用制限: 最大長が 255 バイトを超える可変長ストリングのデータ・タイプが結果となる式には、特別な制限が適用されます。このような式は以下の場所には使用できません。

- DISTINCT が先行している SELECT リスト
- GROUP BY 文節
- ORDER BY 文節
- DISTINCT を伴う列関数
- UNION ALL 以外のセット演算子の副選択

上記の制限に加えて、LONG VARCHAR、CLOB のデータ・タイプ、または構造タイプ列が結果となる式は、以下の場所には使用できません。

- 基本述部、多値比較述部、BETWEEN 述部、または IN 述部
- 列関数
- VARGRAPHIC、TRANSLATE、および日付 / 時刻スカラー関数
- LIKE 述部のパターン・オペランドまたは POSSTR 関数の探索ストリング・オペランド
- 日付 / 時刻値のストリング表記

VARCHAR を引き数として取る SYSFUN スキーマの関数は、 4 000 バイトよりも長い VARCHAR を引き数として受け入れません。しかし、そのような関数の多くには、 CLOB(1M) を受け入れるための代替シグニチャーが用意されています。そのような代替シグニチャーが用意されている関数の場合は、ユーザーが 4 000 バイトよりも長い VARCHAR ストリングを明示的に CLOB へキャストし、結果が戻されたら希望する長さの VARCHAR へ再びキャストしなおすという手順を取ります。

NULL 終了文字ストリング

C の NULL 終了文字ストリングは、プリコンパイル・オプションの標準レベルに応じて、異なった方式で処理されます。 NULL 終了文字ストリングの処理については、 *アプリケーション開発の手引き* の C 言語に関する部分を参照してください。

文字サブタイプ

それぞれの文字ストリングは、さらに次のいずれかとして定義されます。

ビット・データ
コード・ページに対応していないデータ。

SBCS データ それぞれの文字が 1 バイトで表現されるデータ。

混合データ　1 バイト文字セット (SBCS) とマルチバイト文字セット (MBCS) の文字が混合して含まれるデータ。

SBCS と MBCS についての考慮事項: SBCS データは、SBCS データベースでのみサポートされます。混合データは、DBCS データベースでのみサポートされます。

漢字ストリング

漢字ストリング は、 2 バイト文字データを表す一連のバイトです。ストリングの長さは、その一連のバイトの 2 バイト文字の数です。長さがゼロの場合、その値は空ストリングと呼ばれます。この値をヌル値と混同しないようにしてください。

漢字ストリングについて、それらの値に 2 バイト文字コード・ポイント以外の値が含まれていないかどうかを調べる妥当性検査は行われません。[15] データベース・マネージャーは、 2 バイト文字データが漢字データ・フィールドに含まれていることを想定しています。データベース・マネージャーは、漢字ストリング値の長さが偶数バイトであることを検査します。

漢字ストリング・データ・タイプは、固定長でも可変長でもかまいません。固定長と可変長の意味は、文字ストリング・データ・タイプでの定義と同様です。

固定長漢字ストリング

固定長漢字ストリングの列の値は、すべて同じ長さです。この長さは、その列の長さ属性によって決定されます。長さ属性は、1 以上 127 以下でなければなりません。

可変長漢字ストリング

可変長漢字ストリングには、VARGRAPHIC、LONG VARGRAPHIC、および DBCLOB の 3 つのタイプがあります。

- VARGRAPHIC タイプは、最長 16 336 個の 2 バイト文字からなる可変長ストリングです。
- LONG VARGRAPHIC タイプは、最長 16 350 個の 2 バイト文字からなる可変長ストリングです。

15. この規則の例外は、 WCHARTYPE CONVERT オプションを指定してプリコンパイルされたアプリケーションです。このオプションを指定した場合、妥当性検査が行われます。詳細については、 *アプリケーション開発の手引き* の 『C および C++ のプログラミング』に関する項を参照してください。

- DBCLOB タイプは、最長 1 073 741 823 個の 2 バイト文字からなる可変長ストリングです。

最大長が 127 を超える可変長漢字ストリング・データ・タイプが結果となる式には、特別な制限が適用されます。この制限は、89ページの『可変長文字ストリングの使用制限』で指定されているものと同じです。

NULL 終了漢字ストリング

C の NULL 終了漢字ストリングは、プリコンパイル・オプションの標準レベルに応じて、異なった方式で処理されます。 NULL 終了漢字ストリングの処理については、 *アプリケーション開発の手引き* の C 言語に関する部分を参照してください。

このデータ・タイプは表内に作成することはできません。データをデータベースに挿入するときやデータベースから検索するときにのみ使用可能です。

2 進ストリング

2 進ストリング は、一連のバイトです。通常、テキスト・データを含む文字ストリングとは異なり、 2 進ストリングは、ピクチャーなどの従来はあまりなかったデータの保持に使用されます。ビット・データ・サブタイプの文字ストリングも似たような目的で使用されることがありますが、この 2 つのデータ・タイプは互換ではありません。 BLOB スカラー関数を使用すると、ビット・ストリングの文字を 2 進ストリングにキャスト (タイプ変換) することができます。 2 進ストリングの長さは、バイトの数です。コード・ページは対応していません。 2 進ストリングにも文字ストリングと同じ制限があります (詳細については、 89ページの『可変長文字ストリングの使用制限』を参照)。

数値

すべての数値には、符号と精度があります。*精度* とは、符号を除いたビット数または桁数です。数値の値がゼロのときは、符号は正であるとみなされます。

短精度整数 (SMALLINT)

短精度整数 は、精度が 5 桁の 2 バイト整数です。短精度整数の範囲は -32 768 ～ 32 767 です。

長精度整数 (INTEGER)

長精度整数 は、精度が 10 桁の 4 バイトの整数です。長精度整数の範囲は -2 147 483 648 ～ +2 147 483 647 です。

大整数 (BIGINT)

大整数 は、精度が 19 桁の 8 バイトの整数です。大整数の範囲は、-9 223 372 036 854 775 808 ～ +9 223 372 036 854 775 807 です。

単精度浮動小数点 (REAL)

単精度浮動小数点 数は、実数の 32 ビット近似値です。 この数はゼロ、または -3.402E+38 ～ -1.175E-37、もしくは 1.175E-37 ～ 3.402E+38 の範囲にすることができます。

倍精度浮動小数点 (DOUBLE または FLOAT)

倍精度浮動小数点 数は、実数の 64 ビットの近似値です。 値は、ゼロ、または -1.79769E+308 ～ -2.225E-307、または 2.225E-307 ～ 1.79769E+308 の範囲です。

10 進数 (DECIMAL または NUMERIC)

10 進数 値は、暗黙の小数点を伴うパック 10 進数です。小数点の位置は、その数値の精度と位取りによって決定されます。 数値の小数部分の桁数である位取りが、負になったり精度数よりも大きくなったりすることはありません。最大精度は 31 桁です。パック 10 進数の表現については、1229ページの『パック 10 進数』を参照してください。

10 進数の列の値は、すべて同じ精度と位取りの値です。 10 進数の変数または 10 進数の列の数値の範囲は、 *-n* ～ *+n* です (絶対値 *n* は、適切な精度および 10 進数で表現できる最も大きな数値)。最大の範囲は -10**31+1 ～ 10**31-1 です。

日付 / 時刻の値

日付 / 時刻データ・タイプについて以下に説明します。 日付 / 時刻の値は、特定の算術演算およびストリング操作で使用することができ、特定のストリングとは互換性がありますが、これはストリングでも数字でもありません。

日付

日付 (date) は、年、月、日の 3 つの部分からなる値です。年の部分の範囲は 0001 ～ 9999 です。月の部分の範囲は 1 ～ 12 です。日の部分の範囲は 1 から *x* です (*x* は月によって違います)。

日付の内部表示は 4 バイトのストリングです。各バイトは、2 桁のパック 10 進数からなります。最初の 2 バイトは年、3 番目のバイトは月、最後のバイトは日です。

DATE 列の長さは、SQLDA の項で説明するように、10 バイトです。これは、日付の値を文字ストリングで表記するために適した長さになっています。

時刻

時刻 (time) は、時、分、秒の 3 つの部分からなる値であり、 24 時間制の時刻を表します。時の部分の範囲は 0 ～ 24、それ以外の部分の範囲は 0 ～ 59 です。時が 24 の場合、分と秒の指定はゼロになります。

時刻の内部表示は 3 バイトのストリングです。各バイトは 2 桁のパック 10 進数からなります。最初のバイトは時、2 番目のバイトは分、最後のバイトは秒です。

TIME 列の長さは、SQLDA の項で説明するように、8 バイトです。これは、時刻の値を文字ストリングで表記するために適した長さになっています。

タイム・スタンプ

タイム・スタンプ *(timestamp)* は、 7 つの部分 (年、月、日、時、分、秒、マイクロ秒) から成っており、時刻が小数部分でマイクロ秒を指定する以外は、上記の定義と同様に日時を示します。

タイム・スタンプの内部表示は 10 バイトのストリングであり、各バイトは 2 桁のパック 10 進数からなります。最初の 4 バイトは日付、次の 3 バイトは時刻、最後の 3 バイトはマイクロ秒です。

SQLDA に記述されている TIMESTAMP 列の長さは 26 バイトです。これは、値の文字ストリング表示に適した長さです。

日付 / 時刻の値のストリング表記

データ・タイプが DATE、TIME、または TIMESTAMP の値は、 SQL ユーザーには見えない内部形式で表されます。ただし、日付、時刻、およびタイム・スタンプは文字ストリングで表すこともできます。データ・タイプが DATE、TIME、または TIMESTAMP である定数や変数がないため、これらの表示は SQL ユーザーに直接かかわってきます。したがって、日付 / 時刻の値を取り出すには、この値を文字ストリング変数に割り当てる必要があります。CHAR 関数を使用すると、日付 / 時刻値をストリング表記に変更することができます。通常、文字ストリング表記は、プログラムがプリコンパイルされるとき、またはデータベースにバインドされるときに、 *DATETIME* オプションの指定によってオーバーライドされるのでない限り、データベースの国別コードに関連する日付 / 時刻の値のデフォルトの形式になります。

ラージ・オブジェクト・ストリングや LONG VARCHAR は、その長さに関係なく、日付 / 時刻値を表すストリングとして使用することはできません。そのように使用した場合はエラー (SQLSTATE 42884) になります。

日付 / 時刻の値の有効なストリング表記が内部の日付 / 時刻の値の操作に使用される場合、ストリング表記が日付、時刻、またはタイム・スタンプの内部形式に変換されてから、操作が実行されます。次に、日付 / 時刻の値の有効なストリング表記の定義について説明します。

日付ストリング

日付のストリング表示は、数字で始まり、長さが 8 バイト以上の文字ストリングです。後書きブランクを含めることができます。月と日の部分の先行ゼロは省略可能です。

日付を示す有効な文字列形式を、表 3 に示します。各形式の名前を示し、対応する省略形とその使用例も示します。

表 3. 日付のストリング表記形式

形式名	省略形	日付形式	例
国際標準化機構	ISO	yyyy-mm-dd	1991-10-27
IBM USA 標準規格	USA	mm/dd/yyyy	10/27/1991
IBM 欧州標準規格	EUR	dd.mm.yyyy	27.10.1991
日本工業規格西暦	JIS	yyyy-mm-dd	1991-10-27
地域別定義 (*DB2 データ・リンク・マネージャー 概説およびインストール* を参照)	LOC	データベースの国コードに依存	–

時刻ストリング

時刻のストリング表記は、数字で始まり、長さが 4 バイト以上の文字ストリングです。後書きブランクを含めることができます。時刻の時部分の先行ゼロは省略可能であり、秒は完全に省略することができます。秒が省略されている場合は、暗黙のうちに 0 秒とみなされます。したがって、13.30 は 13.30.00 に等しくなります。

時刻を示す有効なストリング形式を、表4 に示します。各形式の名前を示し、対応する省略形とその使用例も示します。

表 4. 時刻のストリング表記形式

形式名	省略形	時刻形式	例
国際標準化機構[2]	ISO	hh.mm.ss	13.30.05

表 4. 時刻のストリング表記形式 (続き)

形式名	省略形	時刻形式	例
IBM USA 標準規格	USA	hh:mm AM または PM	1:30 PM
IBM 欧州標準規格	EUR	hh.mm.ss	13.30.05
日本工業規格西暦	JIS	hh:mm:ss	13:30:05
地域別定義 (*DB2 データ・リンク・マネージャー 概説およびインストール* を参照)	LOC	データベースの国コードに依存	―

注:

1. ISO、EUR および JIS 形式では、.ss (もしくは :ss) は省略可能です。
2. 最近、国際標準化機構は、時刻形式を日本工業規格 (西暦) と同じ形式に変更しました。このため、アプリケーションで現行の国際標準化機構形式が必要な場合は、 JIS を使用してください。
3. USA 時刻ストリング形式の場合、分の指定を省略できます。その場合、00 分とみなされます。したがって、1 PM は 1:00 PM に等しくなります。
4. USA 時刻形式では、時を 13 以上にすることはできず、 00:00 AM という特殊な場合を除いて、0 にすることはできません。 AM および PM の前にはスペースが 1 個入れられます。 24 時間制の ISO 形式を使用した場合、 USA 形式と 24 時間制との対応は次のようになります。
   12:01 AM から 12:59 AM は、00.01.00 から 00.59.00 に対応します。
   01:00 AM から 11:59 AM は、01.00.00 から 11.59.00 に対応します。
   12:00 PM (正午) から 11:59 PM は、12.00.00 から 23.59.00 に対応します。
   12:00 AM (深夜) は 24.00.00 に対応し、00:00 AM (深夜) は 00.00.00 に対応します。

タイム・スタンプ・ストリング

タイム・スタンプのストリング表記は、数字で始まり、長さが 16 バイト以上の文字ストリングです。タイム・スタンプの完全なストリング表示は、*yyyy-mm-dd-hh.mm.ss.nnnnnn* という形式です。後書きブランクを含めることができます。タイム・スタンプの月、日、および時の部分の先行ゼロは省略でき、マイクロ秒は切り捨てたり、完全に省略したりできます。マイクロ秒の部分で後続ゼロが省略されている場合、抜けている数字は暗黙のうちに 0 であるとみなされます。したがって、 1991-3-2-8.30.00 は 1991-03-02-08.30.00.000000 に等しくなります。

また SQL ステートメントは、タイム・スタンプの ODBC ストリング表示を入力値としてのみサポートします。タイム・スタンプの ODBC ストリング表示の形式は、 *yyyy-mm-dd hh:mm:ss.nnnnnn* です。 ODBC の詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

MBCS についての考慮事項

日付、時刻、およびタイム・スタンプには、 1 バイト文字と数字だけしか含めることができません。

DATALINK 値

DATALINK 値はカプセル化された値で、データベース以外の場所に保管されているファイルへのデータベースからの論理参照を含んでいます。このカプセル化された値の属性は以下のとおりです。

リンク・タイプ
現在サポートされているリンクのタイプは 'URL' です。

データ・ロケーション
DB2 内でリンクで参照されているファイルのロケーション。これは、URL 形式になります。この URL に許可されているスキーム名は、以下のとおりです。

- HTTP
- FILE
- UNC
- DFS

URL の他の部分は、以下のとおりです。

- HTTP、FILE、および UNC スキームのファイル・サーバー名
- DFS スキームのセル名
- ファイル・サーバーまたはセル内の完全ファイル・パス名

DATALINK の正確な BNF (バッカス正規形式) 仕様についての詳細は、1463ページの『付録P. DATALINK での BNF 指定』を参照してください。

コメント
最大 254 バイトの記述情報。これは、アプリケーションの特定の使用 (データの場所を詳細に識別する、または別の方法で識別するなど) のためのものです。

URL によるデータ・ロケーション属性を解析しているときに、前後のブランク文字は切り詰められます。さらに、スキーム名 ('http'、'file'、'unc'、 'dfs') およ

びホストは、大文字小文字を区別せず、常に大文字でデータベース内に格納されます。データベースから DATALINK 値を取り出すと、適切な場合には、URL 属性内にアクセス・トークンが組み込まれます。これは動的に生成され、データベースに保管される DATALINK 値に永続的に組み込まれているわけではありません。詳細については、310ページの『DLCOMMENT』から始まる、DATALINK に関連したスカラー関数の部分を参照してください。

DATALINK 値に指定されているのがコメント属性だけで、データ・ロケーション属性が空である場合もあります。そのような値が列に保管されていても、当然そのような列にリンクされているファイルはありません。 DATALINK 値のコメントおよびデータ・ロケーション属性の全長は、今のところ、200 バイトまでに制限されています。

DATALINK は DRDA サーバーとの間でやり取りすることができないことに注意してください。

ファイルへのこれらの DATALINK 参照と、 154ページの『BLOB、CLOB、および DBCLOB のホスト変数の参照』という項で説明されている LOB ファイル参照変数との違いを理解することは大切です。似ている点は、それらが両方ともファイルの表記を含んでいるということです。しかし、次の点で異なっています。

- DATALINK はデータベースに保存されるので、リンクおよびそのリンクされたファイルにあるデータは両方とも、データベース内のデータの自然な拡張とみなすことができます。
- ファイル参照変数はクライアント上に一時的に存在するものなので、ホスト・プログラムのバッファーの代わりとみなすことができます。

組み込みスカラー関数は、DATALINK 値 (DLVALUE) の作成と、DATALINK 値 (DLCOMMENT、 DLLINKTYPE、 DLURLCOMPLETE、 DLURLPATH、 DLURLPATHONLY、 DLURLSCHEME、 DLURLSERVER) からのカプセル化された値の抽出を目的として提供されています。

ユーザー定義タイプ

特殊タイプ

特殊タイプ とは、内部表記を既存のタイプ (その 『ソース』・タイプ) と共用するユーザー定義のデータ・タイプです。しかし、特殊タイプはほとんどの操作で、非互換の別個のタイプとみなされます。たとえば、ピクチャー・タイ

プ、テキスト・タイプ、音声タイプを定義しようとする場合、これらのタイプの意味はどれも異なりますが、内部表記としては組み込みデータ・タイプ BLOB を使用します。

次に、AUDIO という名前の特殊タイプを作成する例を示します。

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M)
```

AUDIO は組み込みデータ・タイプの BLOB と内部表記は同じですが、 BLOB や他のどのタイプとも互換でない、別個のタイプとみなされます。これにより、AUDIO 用に特別に関数を設定できるようになり、そのような関数が他のどのタイプ (ピクチャー、テキストなど) にも適用されないことが保証されます。

特殊タイプは、修飾子付き識別子によって識別されます。 CREATE DISTINCT TYPE、DROP DISTINCT TYPE、または COMMENT ON DISTINCT TYPE ステートメント以外で特殊タイプ名が使用されるとき、スキーマ名によってそれが修飾されていない場合は、 *SQL パス* を順に調べて、特殊タイプの一致する最初のスキーマが探索されます。 SQL パスについては、138ページの『CURRENT PATH』 を参照してください。

特殊タイプを使うと、そのインスタンスに対しては、明示的に特殊タイプに基づいて定義された関数や演算子しか適用されないようになるため、強力なタイプ識別機能が実現されます。そのため、特殊タイプはそのソース・タイプの関数や演算子を自動的には獲得しません。そのようなものは無意味である可能性があるためです。 (たとえば、AUDIO タイプの LENGTH 関数は、そのオブジェクトの長さをバイト単位ではなく秒単位で戻します。)

LONG VARCHAR、LONG VARGRAPHIC、LOB の各タイプ、または DATALINK をソースとする特殊タイプは、ソース・タイプと同じ制限に従います。

しかし、特殊タイプのソース・タイプに対して定義された関数をソースとするユーザー定義関数を定義することによって、ソース・タイプの特定の関数と演算子が特殊タイプに適用されるように明示的に指定することは可能です (例については 120ページの『ユーザー定義タイプの比較』を参照)。ソース・タイプとして LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、または DATALINK を使用しているもの以外のユーザー定義特殊タイプについては、自動的に比較演算子が生成されます。さらに、ソース・タイプから特殊タイプへ、また特殊タイプからソース・タイプへのキャストをサポートする関数も生成されます。

構造タイプ

構造タイプ とは、データベースに定義されている構造を持つユーザー定義のデータ・タイプのことです。これには、名前が付けられている一連の*属性* が含まれており、それぞれにデータ・タイプがあります。構造タイプには、一連のメソッド仕様も含まれています。

構造タイプは表、視点、または列のタイプとして使用することができます。表または視点のタイプとして使用する場合、その表または視点は、それぞれ*タイプ付き表* または*タイプ付き視点* となります。タイプ付き表およびタイプ付き視点の場合、構造タイプの属性の名前およびデータ・タイプは、タイプ付き表またはタイプ付き視点の列の名前およびデータ・タイプになります。タイプ付き表またはタイプ付き視点の行は、構造タイプのインスタンスの表示と考えることができます。列のデータ・タイプとして使用する場合、その列には該当構造タイプの値 (または、下記のように、そのタイプにサブタイプがあれば、その値) が含まれます。構造列オブジェクトの属性を取り出して処理するときには、メソッドを使います。

用語: *スーパータイプ* とは、*サブタイプ* という、他の構造タイプが定義されている構造タイプのことです。サブタイプはスーパータイプのすべての属性およびメソッドを継承し、さらに属性およびメソッドを定義することもできます。共通のスーパータイプに関連する構造タイプのセットは*タイプ階層* と呼ばれ、それより上位のスーパータイプを持たないタイプをそのタイプ階層の*ルート・タイプ* と呼びます。

サブタイプという用語は、タイプ階層において 1 つのユーザー定義の構造タイプおよびその下にあるすべてのユーザー定義の構造タイプを指して用いられます。したがって、階層内における構造タイプ T のサブタイプは、 T と、T の下にあるすべての構造タイプになります。構造タイプ T の*厳密な意味*でのサブタイプ とは、タイプ階層で T の下にある構造タイプのことです。

タイプ階層内に再帰的タイプ定義を含めることには、いくつかの制限があります。このため、許可されている特定タイプの再帰的定義を参照するために、簡単な方法を考える必要があります。以下の定義が使われます。

- *直接的な使用*: 以下のいずれか 1 つが当てはまる場合のみ、タイプ **A** は、他のタイプ **B** を直接に使用します。
  1. タイプ **A** に、タイプ **B** の属性がある場合
  2. タイプ **B** が、**A** のサブタイプ、または **A** のスーパータイプである場合
- *間接的な使用*: 以下のいずれかが当てはまる場合、タイプ **A** は、タイプ **B** を間接的に使用します。
  1. タイプ **A** がタイプ **B** を直接に使う場合

2. タイプ **A** がなんらかのタイプ **C** を直接に使用し、タイプ **C** がタイプ **B** を間接的に使う場合

いずれかの属性タイプがそれ自体を直接にまたは間接的に使うように、タイプを定義することはできません。そのような構成を作成する必要がある場合、参照を属性として使うことを考慮してください。たとえば、構造タイプ属性では、「管理職」が属性タイプ「従業員」である場合に、「管理職」の属性を持つ「従業員」のインスタンスというものはあり得ません。しかし、REF (従業員) のタイプを持つ「管理職」の属性はあり得ます。

他の特定のオブジェクトが、あるタイプを直接または間接的に使っている場合、そのタイプを除去することはできません。たとえば、表または視点の列が、タイプを直接または間接的に使っている場合、タイプを除去することはできません。タイプの除去を制限する可能性のあるオブジェクトすべてに関しては、 964ページの表27 を参照してください。

構造タイプ列の値は、 89ページの『可変長文字ストリングの使用制限』で指定されている CLOB 値に適用される制限に従います。

参照 (REF) タイプ

参照タイプ は構造タイプと対になっているタイプです。特殊タイプに似て、参照タイプは組み込みデータ・タイプの 1 つと共通の表記を使用するスカラー・タイプです。この同じ表記はタイプ階層のすべてのタイプで共用されます。参照タイプ表記は、タイプ階層のルート・タイプの作成時に定義されます。参照タイプを使用する場合、構造タイプはタイプのパラメーターとして指定されます。このパラメーターを、参照のターゲット・タイプ といいます。

参照のターゲットは、通常、タイプ付き表またはタイプ付き視点の行です。参照タイプを使用する場合、*効力範囲* を定義することができます。 効力範囲は、参照値のターゲット行がある表 (*ターゲット表* と呼ばれる) または視点 (*ターゲット視点* と呼ばれる) を指定します。ターゲット表またはターゲット視点は、参照タイプのターゲット・タイプと同じタイプでなければなりません。効力範囲を持つ参照タイプのインスタンスは、タイプ付き表またはタイプ付き視点の行 (*ターゲット行* と呼ばれる) を一意的に識別します。

データ・タイプのプロモーション

データ・タイプは、関連するいくつかのデータ・タイプからなるグループに分類されます。そのようなグループの中では、あるデータ・タイプを他のデータ・タイプより優先するとみなす優先順位が存在します。この優先順位を使用すると、あるデータ・タイプを、優先順位がそれより上のデータ・タイプにプロモートすること (*プロモーション*) が可能になります。たとえば、CHAR データ・タイプは VARCHAR に、 INTEGER は DOUBLE PRECISION にプロモートできますが、 CLOB を VARCHAR にプロモートすることはできません。

データ・タイプのプロモーションは、以下の場合に使用されます。

- 関数解決を実行する場合 (162ページの『関数解決』を参照)
- ユーザー定義タイプをキャストする場合 (102ページの『データ・タイプ間のキャスト』を参照)
- ユーザー定義タイプを組み込みデータ・タイプに割り当てる場合 (114ページの『ユーザー定義タイプの割り当て』を参照)

次の表5 に、各データ・タイプの優先順位順のリストを示します。特定のデータ・タイプのプロモート先として可能なデータ・タイプを調べたいとき、この表を使うことができます。この表に示されているとおり、最適の選択は、別のデータ・タイプへプロモートすることではなく、常に同じデータ・タイプです。

表 5. データ・タイプの優先順位表

データ・タイプ	データ・タイプ優先順位リスト (高いものから順に)
CHAR	CHAR、VARCHAR、LONG VARCHAR、CLOB
VARCHAR	VARCHAR、LONG VARCHAR、CLOB
L O N G VARCHAR	LONG VARCHAR、CLOB
GRAPHIC	GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC、DBCLOB
VARGRAPHIC	VARGRAPHIC、LONG VARGRAPHIC、DBCLOB
L O N G VARGRAPHIC	LONG VARGRAPHIC、DBCLOB
BLOB	BLOB
CLOB	CLOB
DBCLOB	DBCLOB
SMALLINT	SMALLINT、INTEGER、BIGINT、decimal、real、double

表 5. データ・タイプの優先順位表 (続き)

データ・タイプ	データ・タイプ優先順位リスト (高いものから順に)
INTEGER	INTEGER、BIGINT、decimal、real、double
BIGINT	BIGINT、decimal、real、double
decimal	decimal、real、double
real	real、double
double	double
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
DATALINK	DATALINK
udt	udt (同じ名前) または udt のスーパータイプ
REF(T)	REF(S) (S が T のスーパータイプの場合)

注:

上記の小文字で示したタイプは、以下のように定義されます。

decimal
= DECIMAL(p,s) または NUMERIC(p,s)

real = REAL または FLOAT(*n*)。ここで、 *n* は 24 を超えない値。

double = DOUBLE、DOUBLE PRECISION、FLOAT または FLOAT(*n*)。ここで、*n* は 25 以上。

udt = ユーザー定義タイプ

リストの中のデータ・タイプの短形式および長形式の同義語は、リストの中の同義語と同じであるとみなされます。

データ・タイプ間のキャスト

特定のデータ・タイプの値を別のデータ・タイプへ、またはデータ・タイプは同じでも長さ、精度、または位取りが異なるデータ・タイプへキャスト する必要がよく生じます。データ・タイプのプロモーション (101ページの『データ・タイプのプロモーション』で定義されている) は、あるデータ・タイプから別のデータ・タイプへのプロモーションにおいて、値を新しいデータ・タイプへキャストすることが必要になる 1 つの例です。別のデータ・タイプへキャストできるデータ・タイプは、ソース・データ・タイプから宛先データ・タイプへ *キャスト可能* であるといいます。

データ・タイプ間でのキャストは、 CAST 仕様 (193ページの『CAST 指定』を参照) を使用して明示的に行うことができますが、ユーザー定義タイプを含む割り当て (114ページの『ユーザー定義タイプの割り当て』を参照) の過程で暗黙的に行われる場合もあります。また、ソース関数から派生するユーザー定義関数を作成するとき (634ページの『CREATE FUNCTION』を参照) は、ソース関数のパラメーターのデータ・タイプが、作成しようとしている関数のデータ・タイプにキャスト可能なものでなければなりません。

組み込みデータ・タイプの間でサポートされているキャストを、 105ページの表6 に示します。

特殊タイプに関する以下のキャストがサポートされています。

- 特殊タイプ *DT* から、そのソース・データ・タイプ *S* へのキャスト
- 特殊タイプ *DT* のソース・データ・タイプ *S* から、特殊タイプ *DT* へのキャスト
- 特殊タイプ *DT* から、それと同じソース・データ・タイプ *DT* へのキャスト
- データ・タイプ *A* から、特殊タイプ *DT* へのキャスト。ただし、 *A* は特殊タイプ *DT* のソース・データ・タイプ *S* へプロモート可能なもの (101ページの『データ・タイプのプロモーション』を参照)
- INTEGER から、ソース・データ・タイプが SMALLINT である特殊タイプ *DT* へのキャスト
- DOUBLE から、ソース・データ・タイプが REAL である特殊タイプ *DT* へのキャスト
- VARCHAR から、ソース・データ・タイプが CHAR である特殊タイプ *DT* へのキャスト
- VARGRAPHIC から、ソース・データ・タイプが GRAPHIC である特殊タイプ *DT* へのキャスト

構造タイプの値を何か別のものにキャストすることはできません。 *ST* のスーパータイプに対するすべてのメソッドは、 *ST* に当てはまるので、構造タイプ *ST* を、そのスーパータイプのいずれかにキャストすべきではありません。必要な操作が *ST* のサブタイプだけに当てはまる場合、サブタイプ処理式を使用して、*ST* をサブタイプの 1 つとして扱います。詳しくは、206ページの『サブタイプの扱い』を参照してください。

キャストに含まれるユーザー定義データ・タイプがスキーマ名によって修飾されていない場合、 *SQL パス* が、ユーザー定義データ・タイプを含む最初のス

キーマをその名前で検出するために使用されます。 SQL パスについては、138 ページの『CURRENT PATH』 を参照してください。

参照タイプに関して、以下のキャストがサポートされています。

- 参照タイプ *RT* から、表記データ・タイプ *S* へのキャスト
- 参照タイプ *RT* の表記データ・タイプ *S* から、参照タイプ *RT* へのキャスト
- ターゲット・タイプが *T* である参照タイプ *RT* から、ターゲット・タイプが *S* である参照タイプ *RS* へのキャスト (*S* は *T* のスーパータイプ)
- データ・タイプ *A* から、参照タイプ *RT* へのキャスト (ただし *A* は、参照タイプ *RT* の表記データ・タイプ *S* へプロモート可能なもの) (101ページの『データ・タイプのプロモーション』を参照)

キャストに含まれる参照データ・タイプのターゲット・タイプがスキーマ名によって修飾されていない場合、 *SQL パス* が、ユーザー定義データ・タイプを含む最初のスキーマをその名前で検出するために使用されます。 SQL パスについては、138ページの『CURRENT PATH』 を参照してください。

表 6. 組み込みデータ・タイプ間のサポートされるキャスト

ターゲット・データ・タイプ → / ソース・データ・タイプ ↓	SMALLINT	INTEGER	BIGINT	DECIMAL	REAL	DOUBLE	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARG	DBCLOB	DATE	TIME	TIMESTAMP	BLOB
SMALLINT	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-
INTEGER	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-
BIGINT	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-
DECIMAL	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-
REAL	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-	-
DOUBLE	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	-	-
CHAR	Y	Y	Y	Y	-	-	Y	Y	Y	Y	-	Y	-	-	Y	Y	Y	Y
VARCHAR	Y	Y	Y	Y	-	-	Y	Y	Y	Y	-	Y	-	-	Y	Y	Y	Y
LONG VARCHAR	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	-	-	-	-	Y
CLOB	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	-	-	-	-	Y
GRAPHIC	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	Y
VARGRAPHIC	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	Y
LONG VARG	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	Y
DBCLOB	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	Y
DATE	-	-	-	-	-	-	Y	Y	-	-	-	-	-	-	Y	-	-	-
TIME	-	-	-	-	-	-	Y	Y	-	-	-	-	-	-	-	Y	-	-
TIMESTAMP	-	-	-	-	-	-	Y	Y	-	-	-	-	-	-	Y	Y	Y	-
BLOB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y

注

- ユーザー定義タイプおよび参照タイプに関してサポートされているキャストについては、この表の前にある説明を参照してください。
- DATALINK タイプにキャストできるのは DATALINK タイプだけです。
- 構造タイプの値を何か別のものにキャストすることはできません。

割り当てと比較

SQL の基本的な演算は、割り当てと比較です。割り当て操作は、INSERT、UPDATE、FETCH、SELECT INTO、VALUES INTO および SET 変換変数ステートメントの実行時に行われます。関数の引き数も、関数の呼び出し時に割り当てられます。比較演算は、MAX、MIN、DISTINCT、GROUP BY、ORDER BY のような述部およびその他の言語要素を含むステートメントの実行時に行われます。

両方の演算に適用される基本的な規則は、関係するオペランドのデータ・タイプは互換でなければならないということです。この互換性規則は集合演算にも適用されます (121ページの『結果のデータ・タイプに関する規則』を参照)。互換性について一覧表にまとめると以下のようになります。

表 7. 割り当てと比較におけるデータ・タイプの互換性

オペランド	2 進整数	10 進数	浮動小数点数	文字ストリング	漢字ストリング	日付	時刻	タイム・スタンプ	2 進ストリング	UDT
2 進整数	Yes	Yes	Yes	No	No	No	No	No	No	[2]
10 進数	Yes	Yes	Yes	No	No	No	No	No	No	[2]
浮動小数点数	Yes	Yes	Yes	No	No	No	No	No	No	[2]
文字ストリング	No	No	No	Yes	No	[1]	[1]	[1]	No [3]	[2]
漢字ストリング	No	No	No	No	Yes	No	No	No	No	[2]
日付	No	No	No	[1]	No	Yes	No	No	No	[2]
時刻	No	No	No	[1]	No	No	Yes	No	No	[2]
タイム・スタンプ	No	No	No	[1]	No	No	No	Yes	No	[2]
2 進ストリング	No	No	No	No [3]	No	No	No	No	Yes	[2]
UDT	[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]	Yes

表 7. 割り当てと比較におけるデータ・タイプの互換性 (続き)

オペランド	2 進整数	10 進数	浮動小数点数	文字ストリング	漢字ストリング	日付	時刻	タイム・スタンプ	2 進ストリング	UDT

注:

1. 日付 / 時刻値と文字ストリングの互換性は、割り当てと比較に限定されています。
   - 日付 / 時刻値は、111ページの『日付 / 時刻の割り当て』の項で説明するように、文字ストリング列と文字ストリング変数に割り当てることができます。
   - 日付の有効なストリング表記は、日付列に割り当てるか、または日付と比較できます。
   - 時刻の有効なストリング表記は、時刻列に割り当てるか、または時刻と比較できます。
   - タイム・スタンプの有効なストリング表記は、タイム・スタンプ列に割り当てるか、またはタイム・スタンプと比較できます。
2. ユーザー定義特殊タイプ (UDDT) の値は、同じ UDDT で定義された値とのみ比較できます。一般に、特殊タイプの値とそのソース・データ・タイプとの間では割り当てがサポートされます。ユーザー定義構造タイプは比較することができません。また、同じ構造タイプまたはそのスーパータイプのいずれかのオペランドにのみ、割り当てることができます。さらに詳しい情報については、114ページの『ユーザー定義タイプの割り当て』を参照してください。
3. これは、 FOR BIT DATA 属性で定義された文字ストリングも 2 進ストリングと互換でないということです。
4. DATALINK オペランドはほかの DATALINK オペランドにしか割り当てることができません。列が NO LINK CONTROL と定義されている場合、またはファイルは存在しているがまだファイル・リンク制御されていない場合、 DATALINK 値はその列にしか割り当てることができません。
5. 参照タイプの割り当ておよび比較については、 115ページの『参照タイプの割り当て』および121ページの『参照タイプの比較』を参照してください。

割り当て操作の基本的な規則は、ヌル値を入れることができない列や、関連する標識変数がないホスト変数に、ヌル値を割り当てることができないという規則です。 (標識変数については、151ページの『ホスト変数の参照』を参照。)

数値の割り当て

数値割り当ての基本的な規則は、 10 進数または整数の整数部分は決して切り捨てられることがないということです。宛先数値の位取りが割り当て元の数値の位取りより小さい場合は、 10 進数の小数部の超過桁が切り捨てられます。

10 進数または整数から浮動小数点への割り当て

浮動小数点数は、実数の近似値です。したがって、10 進数または整数が浮動小数点の列または変数に割り当てられた場合、その結果が元の数値と異なってくる可能性があります。

浮動小数点または 10 進数の整数への割り当て

浮動小数点または 10 進数が、整数の列または整数変数に割り当てられた場合、その数の小数部分が失われます。

10 進数の 10 進数への割り当て

10 進数が 10 進数の列または変数に割り当てられる場合、その数値は、必要に応じて宛先の精度および位取りに変換されます。必要な数だけ先行ゼロが追加または除去されます。また、数値の小数部分では、必要な数の後続ゼロが追加されるか、または必要な数だけ後続ブランクが除去されます。

10 進数への整数の割り当て

整数を 10 進数の列または変数に割り当てるときは、まず数値が一時的な 10 進数へ変換された後、必要なら、宛先の精度と位取りに変換されます。一時的な 10 進数の精度と位取りは、短精度整数では 5,0、長精度整数では 11,0、大整数では 19,0 です。

10 進数への浮動小数点の割り当て

浮動小数点数を 10 進数に割り当てるときは、まず数値が精度 31 の一時的な 10 進数へ変換された後、必要なら、宛先の精度と位取りまで切り捨てられます。この変換では、数値は精度 31 の 10 進数に (浮動小数点数算術演算を使って) 丸められます。その結果、$0.5*10^{-31}$ 未満の数値は 0 になります。位取りは、有効数字を消失させることなく数値全体を表現できるような値のうち可能な最大のものになります。

ストリングの割り当て

割り当てには次の 2 つのタイプがあります。

- 値を列または関数パラメーターに割り当てる場合の*記憶域割り当て*
- 値をホスト変数に割り当てる場合の*検索割り当て*

ストリング割り当てに関する規則は、割り当てのタイプによって異なります。

記憶域割り当て

基本的な規則は、列または関数パラメーターに割り当てられるストリングの長さが、列または関数パラメーターの長さ属性を超えてはならないということです。ストリングの長さが列または関数パラメーターの長さ属性を超えた場合は、以下の処置が取られます。

- ストリングは、宛先の列または関数パラメーターの長さ属性に適合するように、 (長ストリングを除くすべてのストリング・タイプから）後続ブランクが切り捨てられた上で割り当てられます。
- 以下の場合にはエラー (SQLSTATE 22001) になります。
  - 長ストリング以外のストリングからブランク以外の文字が切り捨てられるとき
  - 長ストリングから任意の文字 (またはバイト) が切り捨てられるとき

ストリングが固定長列に割り当てられ、ストリングの長さが割り当て先の長さ属性より短いときは、そのストリングの右に必要な数のブランクが埋め込まれます。埋め込み文字は、FOR BIT DATA 属性で定義されている列の場合も含めて、常にブランクです。

検索割り当て

ホスト変数に割り当てられるストリングの長さは、ホスト変数の長さ属性より長くてもかまいません。 ストリングがホスト変数に割り当てられるときに、ストリングの長さが変数の長さ属性より長ければ、ストリングの右側から文字 (またはバイト) が必要な数だけ切り捨てられます。この場合には警告 (SQLSTATE 01004) が戻され、 SQLCA の SQLWARN1 フィールドに値 'W' が割り当てられます。

さらに、標識変数があってその値のソースが LOB でない場合は、その標識変数はストリングの元の長さに設定されます。

文字ストリングが固定長変数に割り当てられる場合に、ストリングの長さが割り当て先の長さ属性よりも短いなら、ストリングの右端に必要な数のブランクが埋め込まれます。埋め込み文字は、FOR BIT DATA 属性で定義されているストリングの場合も含めて、常にブランクです。

C の NUL 終止符ホスト変数の検索割り当ては、 PREP または BIND コマンドに指定されたオプションに基づいて処理されます。詳細については、 *アプリケーション開発の手引き* の C および C++ のプログラミングに関する項を参照してください。

文字ストリング割り当ての変換規則

列またはホスト変数に割り当てられる文字ストリングまたは漢字ストリングは、必要なら、まず割り当て先のコード・ページに変換されます。文字変換が必要になるのは、次の条件がすべて真の場合だけです。

- コード・ページが異なる。

- ストリングがヌルでも空でもない。
- どちらのストリングのコード・ページ値も 0 (FOR BIT DATA) でない。[16]

文字ストリング割り当てについての MBCS の考慮事項

1 バイト文字とマルチバイト文字の両方を入れることのできる文字ストリングを割り当てる場合には、いくつかの考慮事項があります。このような考慮事項は、FOR BIT DATA として定義されているものを含めて、すべての文字ストリングに適用されます。

- ブランクの埋め込みは、常に 1 バイトのブランク文字 (X'20') を使用して行われます。
- ブランクの切り捨ては、常に 1 バイトのブランク文字 (X'20') に基づいて行われます。切り捨てに関しては、2 バイトのブランク文字はその他の文字と同様に扱われます。
- 文字ストリングをホスト変数に割り当てる場合に、割り当て先のホスト変数にソース・ストリング全体を納めるだけの長さがないなら、 MBCS 文字の断片化が発生します。 MBCS 文字がこのように断片化される場合は、断片化された MBCS 文字の各バイトが割り当て先で 1 バイトのブランク文字 (X'20') に設定されます。それ以外のバイトに関してはソースからの移動は行われず、 SQLWARN1 が 'W' に設定されて切り捨ての発生を示します。 MBCS 文字の断片化に関するこの処理は、文字ストリングが FOR BIT DATA として定義されている場合にも同じであることに注意してください。

漢字ストリング割り当てについての DBCS の考慮事項

漢字ストリングの割り当ては、文字ストリングの場合と同じように処理されます。漢字ストリング・データ・タイプが互換であるのは他の漢字ストリング・データ・タイプとだけであり、数値、文字ストリング、日付 / 時刻データ・タイプとは互換ではありません。

漢字ストリング値が漢字ストリング列に割り当てられる場合、その値の長さがその列の長さを超えてはなりません。

漢字ストリング値 ('ソース'・ストリング) を固定長漢字ストリング・データ・タイプ ('割り当て先'、列またはホスト変数) に割り当てる場合に、ソース・ストリングの長さが割り当て先より短いなら、割り当て先には、ソース・ストリ

16. DRDA アプリケーション・サーバーとして動作する場合、 FOR BIT DATA 列との割り当て、比較、もしくは結合が行われる場合であっても、入力されたホスト変数はアプリケーション・サーバーのコード・ページに変換されます。 SQLDA が入力ホスト変数を FOR BIT DATA として識別するよう変更されている場合は、変換は行われません。

ングのコピーの右端に、値の長さが割り当て先の長さに等しくなるために必要な数の 2 バイト・ブランク文字を埋め込んだものが入れられます。

漢字ストリング値を漢字ストリングのホスト変数に割り当てる場合に、ソース・ストリングの長さがホスト変数の長さよりも長いなら、ホスト変数には、ソース・ストリングのコピーの右端から、値の長さがホスト変数の長さと等しくなるために必要な数の 2 バイト・ブランク文字を切り捨てたものが入れられます。 (このシナリオでは、切り捨てにおいて 2 バイト文字の二分化を考慮する必要はありません。二分化が発生するとすれば、それはソース値または割り当て先のホスト変数のどちらかで漢字ストリング・データ・タイプの定義に異常があるということです。) SQLCA の警告フラグ SQLWARN1 が 'W' に設定されます。標識変数が指定されていれば、標識変数にはソース・ストリングの元の長さ (2 バイト文字の文字数) が入れられます。しかし、DBCLOB の場合は、標識変数に元の長さは入れられません。

C の NUL 終止符ホスト変数 (wchar_t を使って宣言されたもの) の検索割り当ては、 PREP または BIND コマンドに指定されたオプションに基づいて処理されます。詳細については、 *アプリケーション開発の手引き* の C および C++ のプログラミングに関する項を参照してください。

日付 / 時刻の割り当て

日付 / 時刻の割り当てに関する基本的な規則は、DATE、TIME、または TIMESTAMP 値は、データ・タイプの一致する列 (DATE、TIME、TIMESTAMP のいずれか)、または、固定長または可変長の文字ストリング列変数かストリング列にしか割り当てできないということです。 LONG VARCHAR、BLOB、または CLOB の変数または列に割り当てることはできません。

日付 / 時刻値を文字ストリング変数またはストリング列に割り当てるときは、ストリング表記に自動的に変換されます。日付、時刻、タイム・スタンプのどの部分からも先行ゼロが省略されることはありません。割り当て先で必要な長さは、ストリング表記の形式によって異なります。割り当て先の長さが必要よりも長く、割り当て先が固定長ストリングである場合は、割り当て先の右端にブランクが埋め込まれます。割り当て先の長さが必要よりも短い場合は、関係する日付 / 時刻値のタイプと割り当て先のタイプによって結果が異なります。

宛先がホスト変数である場合、以下の規則が適用されます。

- **DATE の場合:** 変数の長さが 10 バイト未満であればエラーが起こります。
- **TIME の場合:** USA 形式では 8 未満の長さの変数を使用できません。その他の形式では 5 未満にすることはできません。

ISO または JIS 形式を使用しホスト変数の長さが 8 未満の場合、時刻の 2 番目の部分は結果から省略され、標識変数があれば、その変数に割り当てられます。 SQLCA の SQLWARN1 フィールドに、省略処理を示す値が設定されます。

- **TIMESTAMP の場合:** ホスト変数が 19 バイト未満であればエラーになります。長さが 19 バイト以上 26 バイト未満の場合、値のマイクロ秒部分の後続桁が省略されます。 SQLCA の SQLWARN1 フィールドに、省略処理を示す値が設定されます。

日付 / 時刻値のストリング長については、 92ページの『日付 / 時刻の値』を参照してください。

DATALINK の割り当て

値を DATALINK 列に割り当てると、値のリンケージ属性が指定されていないか、あるいはその列に NO LINK CONTROL が定義されていない限り、ファイルへのリンクが確立されます。リンクされている値がその列にすでにある場合、そのファイルへのリンクは解除されます。リンクされている値がすでにある場合に、ヌル値を割り当てることによっても、その元の値に関連付けられているファイルへのリンクを解除することができます。

すでに列にあるデータの位置と同じ位置をアプリケーションが指定している場合、そのリンクが保存されます。このようになる理由には、次の 2 つがあります。

- コメントが変更されるため。
- 表がデータ・リンク調整不能 (DRNP) 状態の場合、その表でのリンクは列と同じリンケージ属性を指定することにより、元に戻すことができます。

DATALINK 値は、以下のいずれかの方法で列に割り当てることができます。

- DLVALUE スカラー関数を使用して、新しい DATALINK 値を作成し、その値を列に割り当てることができます。値に含まれているのがコメントだけか、あるいは URL がまったく同じでない場合には、この割り当てを行うことによりファイルがリンクされます。
- CLI 関数 SQLBuildDataLink を使用して、 CLI パラメーターで DATALINK 値を構成することができます。その後、この値を列に割り当てることができます。値に含まれているのがコメントだけか、あるいは URL がまったく同じでない場合には、この割り当てを行うことによりファイルがリンクされます。

値を DATALINK 列に割り当てる場合、以下のエラー条件では SQLSTATE 428D1 が戻されます。

- データ位置 (URL) 形式が無効です (理由コード 21)
- ファイル・サーバーがこのデータベースに登録されていません (理由コード 22)
- 無効なリンク・タイプが指定されています (理由コード 23)
- コメントまたは URL の長さが無効です (理由コード 27)

  URL パラメーターまたは関数結果のサイズは、入力または出力の両方で同じであること、また DATALINK 列の長さによって束縛されるということに注意してください。ただし、戻される URL 値にアクセス・トークンが付け加えられている場合もあります。このことが起こりうる状態では、アクセス・トークンと、DATALINK 列の長さのための十分の記憶スペースが出力位置になければなりません。したがって、完全に展開された書式の注釈と URL (入力で提供されるすべてのデフォルト URL 方式またはデフォルト・ホスト名を含む) の実際の長さには、出力記憶スペースを合わせるための制限が加えられます。制限された長さを超えると、このエラーが生じます。

割り当てによりリンクの作成も行う場合、以下のエラーが発生する場合があります。

- ファイル・サーバーは現在使用できません (SQLSTATE 57050)
- ファイルがありません (SQLSTATE 428D1、理由コード 24)
- 参照されているファイルにアクセスできず、リンクすることができません (理由コード 26)
- ファイルはすでにほかの列にリンクされています (SQLSTATE 428D1、理由コード 25)

  ほかのデータベースへのリンクでも、このエラーは発生することに注意してください。

さらに、割り当てにより既存のリンクを除去する場合、以下のエラーが発生する場合があります。

- ファイル・サーバーは現在使用できません (SQLSTATE 57050)
- 参照保全制御が指定されているファイルが、データ・リンク・ファイル・マネージャーに従った正しい状態ではありません (SQLSTATE 58004)

DATALINK 値は以下のいずれかの方法でデータベースから検索することができます。

- DATALINK 値の一部分を、スカラー関数 (DLLINKTYPE または DLURLPATH など) を使用してホスト変数に割り当てることができます。

通常、検索時にファイル・サーバーへのアクセスは試みられないことに注意してください。[17] そのため、それ以降にファイル・システム・コマンドに使ってファイル・サーバーにアクセスを試みても、失敗する可能性があります。

DATALINK を検索する場合、データベース・サーバーでのファイル・サーバーの登録が検査され、そのファイル・サーバーがまだそのデータベース・サーバーに登録されていることが確認されます (SQLSTATE 55022)。さらに、DATALINK 値を検索する場合に警告が戻される場合がありますが、それはその表が調整保留状態または調整不能状態のためです (SQLSTATE 01627)。

ユーザー定義タイプの割り当て

ユーザー定義タイプをホスト変数へ割り当てる場合には、他の割り当てで使用される規則とは異なる規則が適用されます。

特殊タイプ: ホスト変数への割り当ては、特殊タイプのソース・タイプに基づいて行われます。つまり、次の規則に従います。

- 割り当ての右辺に指定する特殊タイプの値は、その特殊タイプのソース・タイプが割り当ての左辺に指定するホスト変数に割り当て可能な場合にのみ、ホスト変数へ割り当てられます。

割り当ての宛先が、特殊タイプに基づく列である場合、ソース・データ・タイプは、ユーザー定義タイプについて 102ページの『データ・タイプ間のキャスト』で説明されているように、ターゲット・データ・タイプへキャスト可能でなければなりません。

構造タイプ: ホスト変数に対する割り当ては、ホスト変数の宣言済みタイプに基づきます。つまり、次の規則に従います。

割り当ての右辺に指定する構造タイプの値は、宣言済みタイプのホスト変数が構造タイプ、または構造タイプのスーパータイプである場合にのみ、左辺のホスト変数へ割り当てられます。

割り当ての宛先が、構造タイプの列である場合、ソース・データ・タイプは、ターゲット・データ・タイプ、またはターゲット・データ・タイプのサブタイプでなければなりません。

---

17. パスに関連した接頭部名を決定するのに、ファイル・サーバーにアクセスすることが必要になる場合があります。これはファイル・システムのマウント・ポイントを移動する場合に、ファイル・サーバーで変更することができます。サーバー上のファイルに始めてアクセスしたときに、必要な値がファイル・サーバーから検索されてデータベース・サーバーのキャッシュに置かれます。そして、それ以降の DATALINK 値の検索はそのファイル・サーバーで行われます。ファイル・サーバーにアクセスできない場合には、エラーが戻されます (SQLSTATE 57050)。

参照タイプの割り当て

ターゲット・タイプ *T* を指定している参照タイプは、ターゲット・タイプ *S* を指定している参照タイプでもある参照タイプ列に割り当てることができます (*S* は *T* のスーパータイプ)。効力範囲が指定されている参照列または変数に割り当てが行われる場合、割り当てられている実際の値が、効力範囲で定義されているターゲット表またはターゲット視点に確実に存在するようにするための検査は行われません。

ホスト変数への割り当ては、参照タイプの表示タイプに基づいて行われます。つまり、次の規則に従います。

- 割り当ての右側に指定する参照タイプの値は、ホスト参照変数の左側にも割り当て可能ですが、それはこの参照タイプの表示タイプがこのホスト変数に割り当て可能な場合だけです。

割り当てのターゲットが列で、その割り当ての右側にホスト変数が指定されている場合、そのホスト変数はそのターゲット列の参照タイプに明示的にキャストされなければなりません。

数値の比較

数値は代数的に、つまり符号を考慮して比較されます。たとえば、-2 は +1 より小さい値として扱われます。

一方が整数で、もう一方が 10 進数の場合、 10 進数に変換された整数の一時コピーが比較に使用されます。

位取りの異なる 10 進数を比較する場合、比較は、一方の数値の小数部分が、他方の数値の小数部分と同じ桁数になるように、後続ゼロを使って拡張されたその数値の一時コピーを使用して行われます。

一方が浮動小数点数で、他方が整数か 10 進数の場合、この後者の数値を倍精度浮動小数点数に変換したものの一時コピーが比較に使用されます。

一方が浮動小数点でもう一方が整数または 10 進数の場合、浮動小数点に変換された一時コピーが比較に使用されます。

ストリングの比較

文字ストリングは、データベースの作成時に指定された照合順序に従って比較されます。ただし、FOR BIT DATA 属性の文字ストリングは例外で、そのような文字ストリングは常にビット値に従って比較されます。

長さの異なる文字ストリングを比較する場合、短い方のストリングの右端に、長い方のストリングの長さになるまで 1 バイト・ブランクを埋め込んだものの論理コピーが比較に使用されます。この論理的な拡張は、FOR BIT DATA のものも含め、すべての文字ストリングに対して行われます。

文字ストリング (FOR BIT DATA のタグが付けられた文字ストリングを除く) は、データベースの作成時に指定された照合順序にしたがって比較されます (データベースの作成時に指定される照合順序の詳細については、 *管理の手引き* を参照してください)。たとえば、データベース・マネージャーによって指定されるデフォルトの照合順序は、同じ文字の小文字と大文字に同じ重みを与えています。データベース・マネージャーは、完全に同一のストリングだけが相互に等しいものとして扱われるようにするために、 2 パス比較を実行します。第 1 のパスでは、ストリングがデータベースの照合順序に従って比較されます。ストリングの文字の重みが等しい場合、2 番目の ″同点決勝戦″ パスを実行して、実際のコード・ポイント値に基づいてストリングを比較します。

2 つのストリングは、両方が空であるか、または対応するすべてのバイト数が等しい場合には、等しくなります。どちらかのオペランドがヌル値の場合の結果は不定です。

基本比較演算子 (=、<>、<、>、<=、および >=) を使用する比較演算では、長ストリングおよび LOB ストリングはサポートされません。このようなストリングの比較は、 LIKE 述部と POSSTR 関数を使用した比較でサポートされています。詳細については、219ページの『LIKE 述部』と 360ページの『POSSTR』を参照してください。

長ストリングおよび LOB ストリングのうち 4 000 バイト以下の部分は、SUBSTR と VARCHAR のスカラー関数を使用して比較できます。たとえば、以下のような列を考えてみます。

```
MY_SHORT_CLOB   CLOB(300)
MY_LONG_VAR     LONG VARCHAR
```

この場合、次の演算は有効です。

```
WHERE VARCHAR(MY_SHORT_CLOB) > VARCHAR(SUBSTR(MY_LONG_VAR,1,300))
```

例:

以下の例で、'A'、'Á'、'a'、および 'á' のコード・ポイント値はそれぞれ、X'41'、X'C1'、X'61'、および X'E1' です。

'A'、'Á'、'a'、'á' という文字の重みが 136、139、135、138 である照合順序を考えてみます。このような場合は以下のようになります。

```
'a' < 'A' < 'á' < 'Á'
```

今度は D1、D2、D3、および D4 という 4 つの DBCS 文字を例にとって考えてみましょう。これらの文字はそれぞれ 0xC141、0xC161、0xE141、および 0xE161 というコード・ポイントを持っています。これらの DBCS 文字が CHAR 列に含まれている場合、各文字のバイトが持っている照合重みに従った順序でソートされます。最初の 2 つのバイトの重みは 138 と 139 であるため、 D3 と D4 は D2 と D1 よりも前に来ます。続く 2 つのバイトの重みは 135 と 136 であるため、順序は以下のようになります。

```
D4 < D3 < D2 < D1
```

ただし、比較する値に FOR BIT DATA 属性がある場合や、これらの DBCS 文字が GRAPHIC 列に格納された場合は、照合順序は無視され、これらの文字が持っているコード・ポイントに従って文字が比較されます。以下のようになります。

```
'A' < 'a' < 'Á' < 'á'
```

DBCS 文字はコード・ポイントの順序でソートされます。以下のようになります。

```
D1 < D2 < D3 < D4
```

次に 'A'、'Á'、'a'、'á' という文字が、 74、75、74、および 75 の (固有でない) 重みを持つ照合順序を考えてみましょう。照合重みだけに注目すると (第 1 のパス)、 'a' は 'A' に等しく、'á' は 'Á' に等しいですが、決着を付けるために文字のコード・ポイントを使用すると (第 2 のパス)、以下のようになります。

```
'A' < 'a' < 'Á' < 'á'
```

CHAR 列に含まれている DBCS 文字は、最初は重みに従ったバイトの順序 (第 1 パス) でソートされます。それでも決着がつかない場合は、コード・ポイントに従ったバイトの順序 (第 2 パス) でソートされます。最初の 2 つのバイトは重みが同じであるため、コード・ポイント (0xC1 と 0xE1) で決着を付けることになります。結果として、文字 D1 と D2 は文字 D3 と D4 の前にソートされます。続く 2 つのバイトもこれと同じように比較されます。最終的な結果は以下のようになります。

```
D1 < D2 < D3 < D4
```

ここでも、比較する値に FOR BIT DATA 属性がある場合や、これらの DBCS 文字が GRAPHIC 列に格納された場合は、照合順序は無視され、これらの文字が持っているコード・ポイントに従って文字が比較されます。以下のようになります。

```
D1 < D2 < D3 < D4
```

この例では、照合重みが使用されたときと同じ結果が戻されていますが、実際の場面でいつもそのようになるとは限りません。

比較の際の変換規則

2 つのストリングを比較する場合、必要なら、一方のストリングがまずもう一方のストリングのコード・ページに変換されます。詳細については、126ページの『ストリング変換に関する規則』を参照してください。

結果の順序付け

結果の分類が必要な場合、 115ページの『ストリングの比較』で説明されているストリング比較規則に基づいて順序付けが行われます。比較はデータベース・サーバー側で実行されます。クライアント・アプリケーションに結果が戻される時点で、コード・ページ変換が実行されることがあります。後から行われるこのようなコード・ページ変換は、サーバーの決定した結果セットの順序には影響しません。

ストリング比較に関する MBCS の考慮事項

SBCS/MBCS 混合文字ストリングは、データベースの作成時に指定された照合順序に従って比較されます。デフォルト (SYSTEM) 照合順序で作成されたデータベースの場合、 1 バイトの ASCII 文字はすべて正しい順序で保管されますが、 2 バイト文字は必ずしもコード・ポイントの順序になっているとは限りません。 IDENTITY 順序で作成されたデータベースの場合、 2 バイト文字はすべてコード・ポイントの順序で保管され、 1 バイトの ASCII 文字も同様にコード・ポイントの順序で保管されます。 COMPATIBILITY 順序で作成されたデータベースの場合、ほとんどの 2 バイト文字について正しくソートを行い、 ASCII についてもほぼ正しい、中間的な順序が使用されます。これは、DB2 バージョン 2 ではデフォルトの照合表でした。

混合文字ストリングはバイトごとに比較されます。混合ストリングに含まれる多重バイト文字について通常と異なる結果になる場合がありますが、これは個々のバイトが別個に扱われるためです。

例:

この例で、'A'、'B'、'a'、および 'b' の 2 バイト文字のコード・ポイント値はそれぞれ、 X'8260'、X'8261'、X'8281'、および X'8282' です。

コード・ポイント X'8260'、X'8261'、X'8281'、および X'8282' の重みがそれぞれ 96、65、193、および 194 である照合順序を考えてみます。この場合は以下のようになります。

```
'B' < 'A' < 'a' < 'b'
```

および

```
'AB' < 'AA' < 'Aa' < 'Ab' < 'aB' < 'aA' < 'aa' < 'ab'
```

漢字ストリングの比較は、文字ストリングの場合と同じように処理されます。

漢字ストリングの比較は、 LONG VARGRAPHIC を除くすべての漢字ストリング・データ・タイプの間で有効です。 LONG VARGRAPHIC および DBCLOB データ・タイプは、比較演算では使用できません。

漢字ストリングに対しては、データベースの照合順序は使用されません。その代わりに、漢字ストリングは、常に対応するバイトの数値 (2 進値) に基づいて比較されます。

前の例で、リテラルが漢字ストリングの場合、以下のような結果になります。

```
'A' < 'B' < 'a' < 'b'
```

および

```
'AA' < 'AB' < 'Aa' < 'Ab' < 'aA' < 'aB' < 'aa' < 'ab'
```

長さの異なる漢字ストリングを比較する場合、短い方のストリングの右端に長い方のストリングの長さになるまで、 2 バイト・ブランクを埋め込んだものの論理コピーが比較に使用されます。

2 つの漢字ストリングの値が等しくなるのは、両方が空であるか、または対応する漢字がすべて等しい場合です。どちらかのオペランドがヌル値の場合の結果は不定です。 2 つの値が等しくない場合は、両者の関係は単純な 2 進ストリング比較によって決定されます。

この節で説明してきたとおり、バイトに基づくストリングの比較は誤った結果をもたらす場合があります。つまり、文字比較で得られる文字とは異なる結果が生じる場合があります。ここで示した一連の例は、同じ MBCS コード・ページであることを前提にしていますが、実際には、同じ言語を使用していても異なる多重バイトのコード・ページを使用することがあるので、状況はもっと複雑であるといえます。たとえば、日本語 DBCS コード・ページと日本語 EUC コード・ページからのストリングを比較するというような場合が考えられます。

日付 / 時刻の比較

DATE、TIME、または TIMESTAMP 値は、同じデータ・タイプの別の値か、そのデータ・タイプのストリング表記と比較することができます。すべての比較は日時順に行われます。つまり、0001 年 1 月 1 日からの時間の経過の大きい方が値が大きいということです。

TIME 値と、時刻値のストリング表記とが関係する比較では、常に秒数が含められます。ストリング表記で秒数を省略しているときは、暗黙のうちにゼロ秒が補われます。

TIMESTAMP 値に関する比較は、等しいとみなしてもよいような表示の考慮はしません。日時順に行われます。

例:

```
TIMESTAMP('1990-02-23-00.00.00') > '1990-02-22-24.00.00'
```

ユーザー定義タイプの比較

ユーザー定義特殊タイプの値は、完全に同じユーザー定義特殊タイプの値とのみ比較することができます。ユーザー定義特殊タイプは、WITH COMPARISONS 文節を使用して定義されていなければなりません。

例:

以下の YOUTH 特殊タイプおよび CAMP_DB2_ROSTER 表を想定します。

```
CREATE DISTINCT TYPE YOUTH AS INTEGER WITH COMPARISONS

 CREATE TABLE CAMP_DB2_ROSTER
   ( NAME                VARCHAR(20),
     ATTENDEE_NUMBER     INTEGER NOT NULL,
     AGE                 YOUTH,
     HIGH_SCHOOL_LEVEL   YOUTH)
```

以下の比較は有効です。

```
SELECT * FROM CAMP_DB2_ROSTER
   WHERE AGE > HIGH_SCHOOL_LEVEL
```

以下の比較は無効です。

```
SELECT * FROM CAMP_DB2_ROSTER
   WHERE AGE > ATTENDEE_NUMBER
```

ただし、特殊タイプとそのソース・タイプとの間では、キャストのための関数または CAST 指定を使用することによって、 AGE と ATTENDEE_NUMBER とを比較することができます。以下の比較はすべて有効です。

```
SELECT * FROM CAMP_DB2_ROSTER
   WHERE INTEGER(AGE) > ATTENDEE_NUMBER

SELECT * FROM CAMP_DB2_ROSTER
   WHERE CAST( AGE AS INTEGER) > ATTENDEE_NUMBER

SELECT * FROM CAMP_DB2_ROSTER
   WHERE AGE > YOUTH(ATTENDEE_NUMBER)

SELECT * FROM CAMP_DB2_ROSTER
   WHERE AGE > CAST(ATTENDEE_NUMBER AS YOUTH)
```

ユーザー定義構造タイプの値を、他の値と比較することはできません (NULL 述部および TYPE 述部が使えます)。

参照タイプの比較

参照タイプ値を比較できるのは、それらのターゲット・タイプが共通のスーパータイプを持っている場合だけです。その共通のスーパータイプのスキーマ名が関数パスに組み込まれている場合、適切な比較関数は検索されるだけです。比較は参照タイプの表示タイプを使用して行われます。参照の効力範囲は、比較では考慮されません。

結果のデータ・タイプに関する規則

結果のデータ・タイプは、演算のオペランドに適用される規則によって決定されます。ここでは、そのような規則について説明します。

これらの規則は以下に適用されます。

- 集合演算 (UNION、INTERSECT、および EXCEPT) の全選択における対応する列
- CASE 式の結果式
- スカラー関数 COALESCE (または VALUE) の引き数
- IN 述部のリストの式値
- 複数行の VALUES 文節の対応する式

これらの規則は、さまざまな演算での長ストリングに関するその他の制限にも従って、適用されます。

さまざまなデータ・タイプに関係する規則を以下に示します。一部については、考えられる結果データ・タイプを表に示します。

それらの表では、適用される長さまたは精度と位取りも含めて、結果データ・タイプを示します。結果タイプは、オペランドを考慮して決定されます。オペ

ランドの対が複数の場合は、まず最初の対から検討します。それによる結果タイプとその次のオペランドとの組み合わせが検討されて、次の結果タイプが決定される、というようになります。最後の中間結果タイプと最後のオペランドによって、その演算の最終的な結果タイプが決定されます。演算処理は左から右へ行われます。このため、演算が繰り返されるときは、中間結果タイプが重要になります。たとえば、以下のような演算を考えてみます。

```
CHAR(2) UNION CHAR(4) UNION VARCHAR(3)
```

最初の対の結果のタイプは CHAR(4) です。この結果の値は常に 4 文字になります。最終的な結果タイプは VARCHAR(4) です。最初の UNION 演算の結果の値は、常に長さが 4 になります。

文字ストリング

文字ストリングは他の文字ストリングと互換性があります。文字ストリングには、CHAR、VARCHAR、LONG VARCHAR、および CLOB データ・タイプが含まれます。

一方のオペランド	他方のオペランド	結果のデータ・タイプ
CHAR(x)	CHAR(y)	CHAR(z)、ただし z = max(x,y)
CHAR(x)	VARCHAR(y)	VARCHAR(z)、ただし z = max(x,y)
VARCHAR(x)	CHAR(y) または VARCHAR(y)	VARCHAR(z)、ただし z = max(x,y)
LONG VARCHAR	CHAR(y)、VARCHAR(y)、または LONG VARCHAR	LONG VARCHAR
CLOB(x)	CHAR(y)、VARCHAR(y)、または CLOB(y)	CLOB(z)、ただし z = max(x,y)
CLOB(x)	LONG VARCHAR	CLOB(z)、ただし z = max(x,32700)

結果の文字ストリングのコード・ページは、 126ページの『ストリング変換に関する規則』に基づいて導き出されます。

漢字ストリング

漢字ストリングは他の漢字ストリングと互換性があります。漢字ストリングには、GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC、および DBCLOB データ・タイプが含まれます。

一方のオペランド	他方のオペランド	結果のデータ・タイプ
GRAPHIC(x)	GRAPHIC(y)	GRAPHIC(z)、ただし z = max(x,y)
VARGRAPHIC(x)	GRAPHIC(y) または VARGRAPHIC(y)	VARGRAPHIC(z)、ただし z = max(x,y)
LONG VARGRAPHIC	GRAPHIC(y)、VARGRAPHIC(y)、または LONG VARGRAPHIC	LONG VARGRAPHIC
DBCLOB(x)	GRAPHIC(y)、VARGRAPHIC(y)、または DBCLOB(y)	DBCLOB(z)、ただし z = max(x,y)
DBCLOB(x)	LONG VARGRAPHIC	DBCLOB(z)、ただし z = max(x,16350)

結果の漢字ストリングのコード・ページは、126ページの『ストリング変換に関する規則』に基づいて導き出されます。

2 進ラージ・オブジェクト (BLOB)

BLOB は別の BLOB とのみ互換であり、結果は BLOB になります。 BLOB タイプとして扱う必要がある場合 (281ページの『BLOB』を参照)、 BLOB スカラー関数を使用して他のタイプからキャストしなければなりません。結果の BLOB の長さは、すべてのデータ・タイプの中で最大の長さです。

数値

数値タイプは他の数値タイプと互換性があります。数値タイプには、SMALLINT、INTEGER、BIGINT、DECIMAL、REAL および DOUBLE が含まれます。

一方のオペランド	他方のオペランド	結果のデータ・タイプ
SMALLINT	SMALLINT	SMALLINT
INTEGER	INTEGER	INTEGER
INTEGER	SMALLINT	INTEGER
BIGINT	BIGINT	BIGINT
BIGINT	INTEGER	BIGINT
BIGINT	SMALLINT	BIGINT
DECIMAL(w,x)	SMALLINT	DECIMAL(p,x)、ただし p = x+max(w-x,5)[1]

一方のオペランド	他方のオペランド	結果のデータ・タイプ
DECIMAL(w,x)	INTEGER	DECIMAL(p,x)、ただし p = x+max(w-x,11)[1]
DECIMAL(w,x)	BIGINT	DECIMAL(p,x)、ただし p = x+max(w-x,19)[1]
DECIMAL(w,x)	DECIMAL(y,z)	DECIMAL(p,s)、ただし p = max(x,z)+max(w-x,y-z)[1]s = max(x,z)
REAL	REAL	REAL
REAL	DECIMAL、BIGINT、INTEGER、または SMALLINT	DOUBLE
DOUBLE	任意の数値	DOUBLE

注: 1. 精度は 31 以下でなければなりません。

DATE (日付)

日付は、別の日付、または日付の有効なストリング表記を値とする任意の CHAR または VARCHAR 式と互換性があります。結果のデータ・タイプは DATE です。

TIME (時刻)

時刻は、別の時刻、または時刻の有効なストリング表記を値とする任意の CHAR または VARCHAR 式と互換性があります。結果のデータ・タイプは TIME です。

TIMESTAMP (タイム・スタンプ)

タイム・スタンプは、別のタイム・スタンプ、またはタイム・スタンプの有効なストリング表記を値とする任意の CHAR または VARCHAR 式と互換性があります。結果のデータ・タイプは TIMESTAMP です。

DATALINK (データ・リンク)

データ・リンクはほかのデータ・リンクと互換性があります。結果のデータ・タイプは DATALINK です。結果の DATALINK の長さは、すべてのデータ・タイプの中で最大の長さです。

ユーザー定義タイプ

特殊タイプ

ユーザー定義特殊タイプは同じユーザー定義特殊タイプとしか互換性がありません。結果のデータ・タイプはそのユーザー定義特殊タイプです。

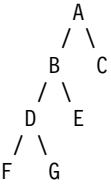
参照タイプ

参照タイプは、ほかの参照タイプと互換性がありますが、それは両方のターゲット・タイプが共通のスーパータイプを持っている場合に限ります。結果のデータ・タイプは、共通のスーパータイプをターゲット・タイプとして持っている参照タイプです。すべてのオペランドに同じ効力範囲の表がある場合、結果は効力範囲の表になります。それ以外の場合、結果では効力範囲は指定されません。

構造タイプ

構造タイプは、ほかの構造タイプと互換性がありますが、それは両方が共通のスーパータイプを持っている場合に限ります。結果の構造タイプ列の静的データ・タイプは、いずれかの列の最小限の共通スーパータイプである構造タイプです。

たとえば、以下の構造タイプ階層について考えてみます。



静的タイプ E と F の構造タイプは、結果の静的タイプ B と互換性があります。ただし、E および F の最小限の共通スーパータイプです。

結果のヌル可能属性

INTERSECT と EXCEPT を除き、 2 つのオペランドの両方ともヌル値が使用できないのでない限り、結果でヌル値が可能です。

- INTERSECT で、どちらかのオペランドでヌル値を使えない場合、結果でのヌル値の使用は認められません (論理積がヌル値になることはありません)。
- EXCEPT では、最初のオペランドでヌル値を使えない場合、結果でのヌル値の使用は認められません (結果は最初のオペランドの値しか取れないためです)。

ストリング変換に関する規則

演算の実行に使用されるコード・ページは、その演算のオペランドに適用される規則によって決定されます。ここでは、そのような規則について説明します。

これらの規則は以下に適用されます。

- 集合演算 (UNION、INTERSECT、および EXCEPT) の全選択における対応するストリング列
- 連結のオペランド
- 述部のオペランド (LIKE を除く)
- CASE 式の結果式
- スカラー関数 COALESCE (および VALUE) の引き数
- IN 述部のリストの式値
- 複数行の VALUES 文節の対応する式

それぞれの場合で結果コード・ページはバインド実行時に決定されます。演算の実行時に、ストリングがそのコード・ページで識別されるコード・ページに変換されることがあります。有効な変換がなされていない文字は、置換文字にマップされ、文字セットと SQLWARN10 が SQLCA で 'W' に設定されます。

結果コード・ページは、オペランドのコード・ページによって決定されます。初めの 2 つのオペランドのコード・ページが中間結果コード・ページを決定し、 (該当する場合には) そのコード・ページと次のオペランドのコード・ページが新たな中間結果コード・ページを決定します。以下、同様になります。最後の中間結果コード・ページと最後のオペランドのコード・ページが、最終結果のストリングまたは列のコード・ページを決定します。コード・ページのそれぞれの対では、以下の規則を順次適用することにより結果が決定されます。

- コード・ページが等しい場合、結果はそのコード・ページになります。
- いずれかのコード・ページが BIT DATA (コード・ページ 0) の場合、結果コード・ページは BIT DATA となります。
- それ以外の場合、結果コード・ページは127ページの表8 に従って決定されます。表の中の「第 1」の項目は第 1 オペランドのコード・ページが選択されることを意味し、「第 2」という項目は第 2 オペランドのコード・ページが選択されることを意味しています。

表 8. 中間結果のコード・ページの選択

第 1 オペランド	第 2 オペランド				
	列の値	派生値	定数	特殊レジスター	ホスト変数
列の値	第 1	第 1	第 1	第 1	第 1
派生値	第 2	第 1	第 1	第 1	第 1
定数	第 2	第 2	第 1	第 1	第 1
特殊レジスター	第 2	第 2	第 1	第 1	第 1
ホスト変数	第 2	第 2	第 2	第 2	第 1

中間結果は、派生値オペランドとみなされます。単一の列値、定数、特殊レジスター、またはホスト変数以外の式も、派生値オペランドであるとみなされます。式が CAST 指定の場合 (またはそれに相当する関数への呼び出しの場合) は、この規則に例外があります。この場合、第 1 オペランドの *種別* は、CAST 指定の第 1 引き数に基づいて決められます。

視点の列は、その最初の源となっているオブジェクトのオペランド・タイプをもつとみなされます。たとえば、表列に基づいて定義された視点列は列値とみなされますが、ストリング式 (A CONCAT B など) に基づく視点列は派生値とみなされます。

以下については、必要なら結果のコード・ページへの変換が行われます。

- 連結演算子のオペランド
- スカラー関数 COALESCE (または VALUE) から選択された引き数
- CASE 式から選択された結果式
- IN 述部のリストの式
- 複数行の VALUES 文節の対応する式
- 集合演算に関係した対応する列

文字変換は、次の条件のすべてに該当する場合に必要になります。

- コード・ページが異なる
- いずれのストリングも BIT DATA ではない
- ストリングがヌルでも空でもない
- コード・ページ変換選択表が、変換の必要性を示している

例

例 1: 以下の条件がある場合は、次のようになります。

式	タイプ	コード・ページ
COL_1	列	850
HV_2	ホスト変数	437

ここで、以下の述部を評価すると、

```
COL_1 CONCAT :HV_2
```

優先されるオペランドは COL_1 列であるため、 2 つのオペランドの結果コード・ページは 850 になります。

例 2: 上記の例からの情報を使用して、述部を評価すると、

```
COALESCE(COL_1, :HV_2:NULLIND,)
```

結果のコード・ページは 850 になります。したがって、スカラー関数 COALESCE の結果のコード・ページは、コード・ページ 850 になります。

区分の互換性

区分の互換性 は、区分化キーの対応する列の基本データ・タイプ相互間で定義されます。区分互換データ・タイプには、同じ値をもつ 2 つの変数 (タイプごとに 1 つずつ) が、同じ区分化関数によって同じ区分化マップ索引にマップされるという特性があります。

129ページの表9 は、区分のデータ・タイプの互換性を示しています。

区分の互換性には、次の特性があります。

- DATE、TIME、および TIMESTAMP には内部形式が使用されます。内部形式は相互に互換性がなく、CHAR との互換性がありません。
- 区分の互換性は、NOT NULL または FOR BIT DATA 定義を伴う列の影響を受けません。
- 互換データ・タイプの NULL 値は同じように取り扱われます。互換性のないデータ・タイプの NULL の場合は異なる結果が生じることがあります。
- UDT の基本データ・タイプは、区分の互換性を分析する場合に使用されます。
- 区分化キーの同一値の小数部は、位取りおよび精度が異なっている場合であっても、同一として取り扱われます。

- 文字ストリング (CHAR、VARCHAR、GRAPHIC または VARGRAPHIC) の後書きブランクは、システムにより提供されるハッシュ関数によって無視されます。
- 長さが異なる CHAR または VARCHAR は、互換データ・タイプです。
- 等しい REAL または DOUBLE の値は、精度が異なっていても同一として取り扱われます。

表 9. 区分の互換性

オペランド	2 進整数	10 進数	浮動小数点数	文字ストリング	漢字ストリング	日付	時刻	タイム・スタンプ	特殊タイプ	構造タイプ
2 進整数	Yes	No	No	No	No	No	No	No	[1]	No
10 進数	No	Yes	No	No	No	No	No	No	[1]	No
浮動小数点数	No	No	Yes	No	No	No	No	No	[1]	No
文字ストリング[3]	No	No	No	Yes[2]	No	No	No	No	[1]	No
漢字ストリング[3]	No	No	No	No	Yes	No	No	No	[1]	No
日付	No	No	No	No	No	Yes	No	No	[1]	No
時刻	No	No	No	No	No	No	Yes	No	[1]	No
タイム・スタンプ	No	No	No	No	No	No	No	Yes	[1]	No
特殊タイプ	[1]	[1]	[1]	[1]	[1]	[1]	[1]	[1]	[1]	No
構造タイプ[3]	No	No	No	No	No	No	No	No	No	No

注:

1 ユーザー定義特殊タイプ (UDT) の値は、UDT のソース・タイプ、もしくは区分互換ソース・タイプのその他の UDT との互換性がある区分です。

2 FOR BIT DATA 属性は、区分の互換性に影響を与えません。

3 ユーザー定義構造タイプとデータ・タイプ LONG VARCHAR、LONG VARGRAPHIC、CLOB、DBCLOB、および BLOB は、区分化キーでサポートされていないので、区分互換性には適用されないことに注意してください。

定数

定数 (リテラル と呼ばれるときもあります) は、値を指定するものです。定数は、ストリング定数か数値定数かに分類されます。数値定数はさらに、整数、浮動小数点数、または 10 進数に分類されます。

定数は、すべて NOT NULL の属性を持ちます。

数値定数では、負のゼロ値 (-0) は符号のないゼロ (0) と同じ値です。

整数定数

整数定数 は、小数点を除き最大 19 桁の符号付きまたは符号なしの整数を指定します。整数定数の値が長精度整数の範囲内である場合、その整数定数のデータ・タイプは長精度整数 (large integer) です。整数定数の値が長精度整数の範囲外であるが、大整数の範囲内にある場合、その整数定数のデータ・タイプは大整数 (big integer) です。大整数値の範囲外で定義された定数は、10 進定数とみなされます。

長精度整数定数の最小のリテラル表現は -2 147 483 647 であり、整数値の限界である -2 147 483 648 ではありません。同様に、大整数定数の最小のリテラル表現は、 -9 223 372 036 854 775 807 であり、 -9 223 372 036 854 775 808 (大整数値の限界) ではありません。

例

```
64     -15     +100     32767     720176      12345678901
```

構文図で、'integer' (整数) という用語は、符号を含んではならない長精度整数定数を指すものとして使用されます。

浮動小数点定数

浮動小数点定数 は、 E で区切られた 2 つの数値で浮動小数点数を指定します。最初の数値には符号と小数点を指定することができます。 2 番目の数値には符号を指定できますが、小数点を含めることはできません。浮動小数点定数のデータ・タイプは倍精度です。 定数の値は、最初の数値と、2 番目の数値で指定される 10 の累乗との積であり、浮動小数点数の範囲内になければなりません。定数の文字数は 30 以下でなければなりません。

例

```
15E1     2.E5     2.2E-1     +5.E+2
```

10 進定数

10 進定数 は、 31 桁以内の数字で構成される符号付きまたは符号なしの数値です。小数点を含むものか、または、2 進整数の範囲に収まらないもののどちらかです。 10 進定数の値は 10 進数の範囲内になければなりません。精度は桁数の合計数 (前後のゼロを含む)、位取りは小数点の右側の桁数 (後続ゼロを含む) です。

例

```
25.5     1000.     -15.     +37589.3333333333
```

文字ストリング定数

文字ストリング定数 は、可変長の文字ストリングを指定するもので、アポストロフィ (') で始まりアポストロフィで終わる文字の並びで構成されます。ストリング定数のこの形式では、ストリング区切り文字の間に文字ストリングを指定します。文字ストリングの長さは、32 672 バイト以下でなければなりません。ストリング区切り文字の間で 1 つのストリング区切り文字を表したいときは、ストリング区切り文字を 2 つ連続して使用します。

例

```
'12/14/1985'
'32'
'DON''T CHANGE'
```

等しくないコード・ページに関する考慮事項

定数値は、データベースにバインドされるときに、必ずデータベース・コード・ページに変換されます。それは、データベース・コード・ページのものとみなされます。したがって、定数を FOR BIT DATA 列と結合して、その結果が FOR BIT DATA となる式で使用される場合、定数値は使用時にそのデータベース・コード・ページ表記から変換されません。

16 進定数

16 進定数 は、アプリケーション・サーバーのコード・ページを使って可変長の文字ストリングを指定します。

16 進数ストリング定数の形式は、X の後に、アポストロフィ (一重引用符) で囲んだ文字の並びを続けたものです。アポストロフィの間にある文字は、偶数個の 16 進数字でなければなりません。 16 進数字の数は 16 336 以下でなければなりません。これを超えると、エラー (SQLSTATE -54002) になります。1 桁の 16 進数字は 4 ビットを表します。これは、数字、または英字 A 〜 F (大文字か小文字に関係なく) のいずれかとして指定されます。この場合、たと

えば A はビット・パターン '1010' を表し、B はビット・パターン '1011' を表します。以下同様です。16 進定数の形式に誤りがある (たとえば、無効な 16 進数もしくは奇数の 16 進数が入っている) 場合、エラーが生じます (SQLSTATE 42606)。

例

```
X'FFFF'       representing the bit pattern '1111111111111111'

X'4672616E6B' representing the VARCHAR pattern of the ASCII string 'Frank'
```

漢字ストリング定数

漢字ストリング定数 は、可変長の漢字ストリングを指定するもので、先頭の単一バイトの G または N の後に、単一バイトのアポストロフィ (') で囲んだ 2 バイト文字の並びを続けます。この形式のストリング定数では、ストリング区切り文字の間に漢字ストリングを指定します。漢字ストリングの長さは必ず偶数バイトであり、 16 336 バイト以下でなければなりません。

例:

```
G'double-byte character string'
N'double-byte character string'
```

MBCS についての考慮事項

MBCS 文字の一部としては、単引用符 (') を使用しないでください。区切り文字とみなされてしまいます。

ユーザー定義タイプを伴う定数の使用

ユーザー定義タイプは強力なタイプ指定です。つまり、ユーザー定義タイプはそれ自体のタイプとしか互換性がありません。一方、定数には組み込みタイプがあります。このため、ユーザー定義タイプと定数の関係する演算は、ユーザー定義タイプがその定数の組み込みタイプにキャストされている場合、または定数がそのユーザー定義タイプにキャストされている場合にのみ実行可能です (キャストについては、 193ページの『CAST 指定』を参照)。たとえば、120ページの『ユーザー定義タイプの比較』にある表と特殊タイプを使用する場合、定数 14 との以下の比較が有効です。

```
SELECT * FROM CAMP_DB2_ROSTER
   WHERE AGE > CAST(14 AS YOUTH)

SELECT * FROM CAMP_DB2_ROSTER
   WHERE CAST(AGE AS INTEGER) > 14
```

以下の比較は無効です。

```
SELECT * FROM CAMP_DB2_ROSTER
  WHERE AGE > 14
```

特殊レジスター

特殊レジスター は、データベース・マネージャーによってアプリケーション・プロセスに対して定義される記憶域であり、 SQL ステートメントで参照可能な情報を保管するのに使用されます。特殊レジスターは、データベース・コード・ページにあります。

CURRENT DATE

CURRENT DATE 特殊レジスターは、アプリケーション・サーバーで SQL ステートメントが実行される時点の、時刻機構の読み取り値にもとづく日付を指定します。この特殊レジスターが単一の SQL ステートメントで何度も使用される場合、または単一のステートメントで CURRENT TIME または CURRENT TIMESTAMP と共に使用される場合、その値はすべて時刻機構の 1 回の読み取りに基づきます。

連合システムでは、データ・ソースでの使用を目的とした照会で CURRENT DATE を使用できます。この照会が処理されて戻される日付は、連合サーバーの CURRENT DATE レジスターから取得されたもので、データ・ソースから取得されたものではありません。

例

以下の例は、PROJECT 表を使用して、 MA2111 プロジェクト (PROJNO) のプロジェクト終了日付 (PRENDATE) に CURRENT DATE を設定しています。

```
UPDATE PROJECT
  SET PRENDATE = CURRENT DATE
  WHERE PROJNO = 'MA2111'
```

CURRENT DEFAULT TRANSFORM GROUP

CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターは、VARCHAR (18) 値を指定します。ここには、ユーザー定義構造タイプの値をホスト・プログラムと交換するときに、動的 SQL ステートメントで使用する変形グループの名前を指定します。この特殊レジスターでは、静的 SQL ステートメントで使用する、またはメソッドの外部関数を使ったパラメーターと結果の交換で使用する変形グループを指定しません。

その値は SET CURRENT DEFAULT TRANSFORM GROUP ステートメントによって設定することができます。値を設定しない場合、特殊レジスターの初期値は、空ストリングになります (長さがゼロの VARCHAR)。

動的 SQL ステートメント (つまり、ホスト変数と相互作用するもの) では、値を交換するときに使用する変形グループの名前は、このレジスターに空ストリングが含まれていない限り、この特殊レジスターの値と同じになります。レジスターに空ストリングが含まれる場合 (SET CURRENT DEFAULT TRANSFORM GROUP ステートメントを使用して、値が設定されていない場合)、変形のために、DB2_PROGRAM 変形グループが使われます。構造タイプ・サブジェクト用に DB2_PROGRAM 変形グループが定義されていない場合、実行時にエラーが生じます (SQLSTATE 42741)。

例

省略時の変形グループを MYSTRUCT1 に設定します。 MYSTRUCT1 変形で定義される TO SQL および FROM SQL 関数は、ユーザー定義構造タイプ変数とホスト・プログラムを交換するときに使います。

```
SET CURRENT DEFAULT TRANSFORM GROUP = MYSTRUCT1
```

この特殊レジスターに割り当てられた、省略時の変形グループの名前を検索します。

```
VALUES (CURRENT DEFAULT TRANSFORM GROUP)
```

CURRENT DEGREE

CURRENT DEGREE 特殊レジスターは、動的 SQL ステートメントを実行するときの、区画内並行性の度合いを指定します。[18] このレジスターのデータ・タイプは CHAR(5) です。有効な値は、'ANY'、または 1 ～ 32 767 の範囲 (両端の値を含む) の整数のストリング表記です。

SQL ステートメントが動的に準備されるときに、整数として表される CURRENT DEGREE の値が 1 である場合、そのステートメントの実行には区画内並行性は使用されません。

SQL ステートメントが動的に準備されるときに、整数として表される CURRENT DEGREE の値が 2 以上 32 767 以下である場合、そのステートメントの実行には、指定された度合いの区画内並行性を伴う場合があります。

18. 静的 SQL の場合、DEGREE バインド・オプションで同じ制御を行えます。

SQL ステートメントが動的に準備されるときに、 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判別される度合いを使用する区画内並行性を伴う場合があります。

実際の実行時の並列性の度合いは、以下の低い方になります。
- 最大照会度 (max_querydegree) 構成パラメーター
- アプリケーションの実行時の度合い
- SQL ステートメントのコンパイルの度合い

intra_parallel のデータベース・マネージャー構成パラメーターが NO に設定される場合、最適化のために CURRENT DEGREE 特殊レジスターの値は無視され、ステートメントは区画内並行性を使用しません。

並列性および制約事項の説明については、*管理の手引き* を参照してください。

値の変更は、 SET CURRENT DEGREE ステートメントを実行して行うことができます (このステートメントについては、 1096ページの『SET CURRENT DEGREE』を参照してください)。

CURRENT DEGREE の初期値は、dft_degree データベース構成パラメーターによって判別されます。この構成パラメーターについては、*管理の手引き* を参照してください。

CURRENT EXPLAIN MODE

CURRENT EXPLAIN MODE 特殊レジスターには、適格な動的 SQL ステートメントに関連のある Explain 機能の動作を制御するための VARCHAR(254) の値が入れられます。この機能は、Explain 情報を生成し、その情報を Explain 表に挿入します (詳細については、 *管理の手引き* を参照してください)。この情報には、Explain スナップショットは含まれません。

可能な値は YES、NO、EXPLAIN、RECOMMEND INDEXES、および EVALUATE INDEXES です。 [19]

YES Explain 機能を使用可能にし、動的 SQL ステートメントについての Explain 情報をそのステートメントのコンパイル時に取り込みます。

19. 静的 SQL の場合、EXPLAIN バインド・オプションで同じ制御を行えます。 PREP および BIND コマンドの場合、EXPLAIN オプション値は YES、NO、および ALL です。

EXPLAIN
YES と同様にスナップショット機能を使用可能にしますが、動的ステートメントの実行は許可しません。

NO Explain 機能を使用不可にします。

RECOMMEND INDEXES
動的照会ごとに、索引のセットが推奨されます。 ADVISE_INDEX 表の中に索引のセットが入れられます。

EVALUATE INDEXES
推奨される索引があたかも存在しているかのように、動的照会の Explain が実行されます。使用される索引は ADVISE_INDEX 表から選出されます。

初期値は NO です。

その値は SET CURRENT EXPLAIN MODE ステートメントによって変更することができます (このステートメントについては、 1098ページの『SET CURRENT EXPLAIN MODE』を参照してください)。

CURRENT EXPLAIN MODE と CURRENT EXPLAIN SNAPSHOT 特殊レジスター値は、 Explain 機能が呼び出されている場合に相互に作用します (詳細については、 1437ページの表142 を参照してください)。 CURRENT EXPLAIN MODE 特殊レジスター値の方は、 EXPLAIN バインド・オプションとも相互に作用します (詳細については、 1438ページの表143 を参照してください)。 RECOMMEND INDEXES 値と EVALUATE INDEXES 値を設定できるのは、CURRENT EXPLAIN MODE レジスターの場合だけです。これらの値を設定するには、 SET CURRENT EXPLAIN MODE ステートメントを使用します。

例: ホスト変数 EXPL_MODE (VARCHAR(254)) を CURRENT EXPLAIN MODE 特殊レジスターの現在の値に設定します。

```
VALUES CURRENT EXPLAIN MODE
  INTO :EXPL_MODE
```

CURRENT EXPLAIN SNAPSHOT

CURRENT EXPLAIN SNAPSHOT 特殊レジスターには、 Explain スナップショット機能の動作を制御するための CHAR(8) の値が入れられます。この機能は、アクセス・プラン情報、操作員コスト、バインド実行時の統計などに関する情報を圧縮して生成するものです (詳細については、 *管理の手引き* を参照)。

次に挙げるステートメントだけがこのレジスターの値として認められます。すなわち、DELETE、INSERT、SELECT、SELECT INTO、UPDATE、VALUES、および VALUES INTO です。

可能な値は YES、NO、および EXPLAIN です。 [20]

YES スナップショット機能を使用可能にし、動的 SQL ステートメントがコンパイルされるときの、そのステートメントの内部表記のスナップショットを取り出します。

EXPLAIN
YES と同様にスナップショット機能を使用可能にしますが、動的ステートメントの実行は許可しません。

NO Explain スナップショット機能を使用不可にします。

初期値は NO です。

その値は SET CURRENT EXPLAIN SNAPSHOT ステートメントによって変更することができます (このステートメントについては、 1101ページの『SET CURRENT EXPLAIN SNAPSHOT』を参照してください)。

CURRENT EXPLAIN SNAPSHOT と CURRENT EXPLAIN MODE 特殊レジスター値は、 Explain 機能が呼び出されている場合に相互に作用します (詳細については、 1437ページの表142 を参照してください)。 CURRENT EXPLAIN SNAPSHOT 特殊レジスター値の方は、 EXPLSNAP バインド・オプションとも相互に作用します (詳細については、 1439ページの表144 を参照してください)。

例
以下の例は、ホスト変数 EXPL_SNAP (char(8)) に、 CURRENT EXPLAIN SNAPSHOT 特殊レジスターの現在の値を設定するものです。

```
VALUES CURRENT EXPLAIN SNAPSHOT
  INTO :EXPL_SNAP
```

CURRENT NODE

CURRENT NODE 特殊レジスターは、調整プログラム・ノード番号 (アプリケーションが接続する区分) を識別する INTEGER 値を指定します。

20. 静的 SQL の場合、EXPLSNAP バインド・オプションで同じ制御を行えます。 PREP および BIND コマンドの場合、EXPLSNAP オプション値は YES、NO、および ALL です。

CURRENT NODE は、データベース・インスタンスが区分化をサポートするように定義されていない (db2nodes.cfg ファイルがない) 場合には 0 を戻します。[21]

CURRENT NODE は、一定の条件に該当する場合にのみ CONNECT ステートメントによって変更できます (589ページの『CONNECT (タイプ 1)』を参照してください)。

例

以下の例では、アプリケーションが接続している区分の番号をホスト変数 APPL_NODE (整数) に設定しています。

```
VALUES CURRENT NODE
  INTO :APPL_NODE
```

CURRENT PATH

CURRENT PATH 特殊レジスターには、 *SQL パス* を指定する VARCHAR(254) の値が入れられます。これは、動的に作成される SQL ステートメントでの関数参照およびデータ・タイプ参照を解決するために使用されるものです。[22] CURRENT PATH は、 CALL ステートメントのストアード・プロシージャー参照を解決するのにも使用されます。初期値は、後述するデフォルト値です。静的 SQL の場合は、FUNCPATH バインド・オプションで関数およびデータ・タイプの解決のための SQL パスを指定できます (FUNCPATH バインド・オプションについては、 *コマンド解説書* を参照)。

CURRENT PATH 特殊レジスターの内容は、 1 つ以上のスキーマ名をコンマで囲んだリストを二重引用符で囲んだものです (ストリング内で引用符を表現するときは、区切り識別子の場合と同様に引用符を繰り返します)。

たとえば、データベース・マネージャーが最初に FERMAT スキーマで、次に XGRAPHIC スキーマで、その後 SYSIBM スキーマで参照するように指定する SQL パスは、 CURRENT PATH 特殊レジスターに以下のように返されます。

```
"FERMAT","XGRAPHIC","SYSIBM"
```

デフォルト値は ″SYSIBM″、″SYSFUN″、X です。ただし、この X は USER 特殊レジスターの値を二重引用符で囲んだものです。

---

21. 区分化データベースの場合、db2nodes.cfg ファイルが存在し、区分 (またはノード) の定義が含まれています。詳細については、*管理の手引き* を参照してください。

22. CURRENT FUNCTION PATH は、CURRENT PATH の同義語です。

その値は SET CURRENT FUNCTION PATH ステートメントによって変更することができます (1126ページの『SET PATH』 を参照してください)。SYSIBM スキーマを指定する必要はありません。 SQL パスにそれが含まれていない場合は、1 番目のスキーマであるとみなされます。暗黙のうちに SYSIBM が想定されている場合、254 文字の中には入りません。

SQL パスを使って関数を解決することについては、 159ページの『関数』を参照してください。スキーマ名で修飾されていないデータ・タイプは、SQL パスのスキーマ名で、修飾子なしの同じ名前が指定されたデータ・タイプのうち最も早く出現するものによって、暗黙のうちに修飾されます。この規則には、CREATE DISTINCT TYPE、CREATE FUNCTION、COMMENT ON、および DROP ステートメントの部分で説明されているように例外があります。

例

以下の例は、SYSCAT.VIEWS カタログ視点を使用して、 CURRENT PATH 特殊レジスターの現行値と同じ設定で作成されたすべての視点を検索しています。

```
SELECT VIEWNAME, VIEWSCHEMA FROM SYSCAT.VIEWS
  WHERE FUNC_PATH = CURRENT PATH
```

CURRENT QUERY OPTIMIZATION

CURRENT QUERY OPTIMIZATION 特殊レジスターには、動的 SQL ステートメントのバインド時に、データベース・マネージャーによって行われる照会最適化のクラスを制御する INTEGER 値が入れられます。 QUERYOPT バインド・オプションは、静的 SQL ステートメントの照会クラスの最適化を制御します (QUERYOPT バインド・オプションの詳細については、 *コマンド解説書* を参照してください)。可能な値の範囲は 0 ～ 9 です。たとえば、照会最適化クラスが最適化の最小クラス (0) に設定されている場合、この特殊レジスターの値は 0 です。デフォルト値は、dft_queryopt データベース構成パラメーターによって決められます。

その値は SET CURRENT QUERY OPTIMIZATION ステートメントによって変更することができます (1105ページの『SET CURRENT QUERY OPTIMIZATION』を参照してください)。

例

以下の例は、SYSCAT.PACKAGES カタログ視点を使って、 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するものです。

```
SELECT PKGNAME, PKGSCHEMA FROM SYSCAT.PACKAGES
  WHERE QUERYOPT = CURRENT QUERY OPTIMIZATION
```

CURRENT REFRESH AGE

CURRENT REFRESH AGE 特殊レジスターは、データ・タイプが DECIMAL(20,6) のタイム・スタンプ期間値を指定します。この期間は、REFRESH DEFERRED 要約表で REFRESH TABLE ステートメントが処理された時からの、照会の処理を最適化するために要約表を使用できる最大期間です。 CURRENT REFRESH AGE の値が 99 999 999 999 999 (ANY) で、QUERY OPTIMIZATION クラスが 5 以上の場合は、 REFRESH DEFERRED 要約表が動的 SQL 照会の処理を最適化するとみなされます。 REFRESH IMMEDIATE 属性が指定されていて、検査保留状態になっていない要約表の最新表示時間はゼロであるとみなされます。

その値は SET CURRENT REFRESH AGE ステートメントによって変更することができます (1109ページの『SET CURRENT REFRESH AGE』を参照してください)。 REFRESH DEFERRED によって定義された要約表は、静的組み込み SQL 照会で使用されることは決してありません。

CURRENT REFRESH AGE の初期値はゼロです。

CURRENT SCHEMA

CURRENT SCHEMA 特殊レジスターは、動的に作成された SQL ステートメントで可能な場合に、修飾子のないデータベース・オブジェクト参照を修飾するのに使用されるスキーマ名を識別する VARCHAR(128) 値を指定します。[23]

CURRENT SCHEMA の初期値は、現行セッション・ユーザーの許可 ID です。

その値は SET SCHEMA ステートメントによって変更することができます (1129ページの『SET SCHEMA』を参照してください)。

QUALIFIER バインド・オプションは、動的に作成された SQL ステートメントについて可能な場合に、修飾子のないデータベース・オブジェクト参照を修飾するのに使用されるスキーマ名を制御します (詳細については、 *コマンド解説書* を参照してください)。

例

オブジェクト修飾のスキーマを 'D123' に設定します。

```
SET CURRENT SCHEMA = 'D123'
```

23. DB2 (OS/390 版) との互換性を保つため、特殊レジスター CURRENT SQLID は CURRENT SCHEMA の同義語として扱われます。

CURRENT SERVER

CURRENT SERVER 特殊レジスターには、現在のアプリケーション・サーバーを識別する VARCHAR(18) の値が入れられます。アプリケーション・サーバーの実際の名前 (別名ではない) が、このレジスターに含まれます。

CURRENT SERVER は、一定の条件に該当する場合にのみ CONNECT ステートメントによって変更できます (589ページの『CONNECT (タイプ 1)』を参照)。

例

以下の例は、アプリケーションが接続されているアプリケーション・サーバーの名前をホスト変数 APPL_SERVE (VARCHAR(18)) に設定しています。

```
VALUES CURRENT SERVER
  INTO :APPL_SERVE
```

CURRENT TIME

CURRENT TIME 特殊レジスターは、アプリケーション・サーバーで SQL ステートメントが実行される時点の時刻機構の読み取り値に基づく時刻を指定します。この特殊レジスターが単一の SQL ステートメントで何度も使用される場合、または単一のステートメントで CURRENT DATE または CURRENT TIMESTAMP と共に使用される場合、その値はすべて時刻機構の 1 回の読み取りに基づく値です。

連合システムでは、データ・ソースでの使用を目的とした照会で CURRENT TIME を使用できます。この照会が処理されて戻される時刻は、連合サーバーの CURRENT TIME レジスターから取得されたもので、データ・ソースから取得されたものではありません。

例

以下の例は、CL_SCHED 表を使って、その日のそれ以降に開始される (STARTING) すべてのクラス (CLASS_CODE) を選択するものです。現在のクラスの DAY 列の値は 3 です。

```
SELECT CLASS_CODE FROM CL_SCHED
  WHERE STARTING > CURRENT TIME  AND DAY = 3
```

CURRENT TIMESTAMP

CURRENT TIMESTAMP 特殊レジスターは、アプリケーション・サーバーで SQL ステートメントが実行される時点の、時刻機構の読み取り値にもとづくタイム・スタンプを指定します。この特殊レジスターが単一の SQL ステートメントで何度も使用される場合、または単一のステートメントで CURRENT

DATE または CURRENT TIME と共に使用される場合、その値はすべて時刻機構の 1 回の読み取りに基づく値です。

連合システムでは、データ・ソースでの使用を目的とした照会で CURRENT TIMESTAMP を使用できます。この照会が処理されて戻されるタイム・スタンプは、連合サーバーの CURRENT TIMESTAMP レジスターから取得されたもので、データ・ソースから取得されたものではありません。

例

以下の例は、1 つの行を IN_TRAY 表に挿入するものです。 RECEIVED 列の値は、その列の挿入時点を示すタイム・スタンプです。他の 3 つの列の値は、ホスト変数 SRC (char(8))、SUB (char(64))、および TXT (VARCHAR(200)) から取られたものです。

```
INSERT INTO IN_TRAY
  VALUES (CURRENT TIMESTAMP, :SRC, :SUB, :TXT)
```

CURRENT TIMEZONE

CURRENT TIMEZONE 特殊レジスターには、 UTC[24] とアプリケーション・サーバーのローカル時との差が入れられます。この差は、時刻期間 (最初の 2 桁が時間数、次の 2 桁が分数、最後の 2 桁が秒数である 10 進数) によって表現されます。時間数の数値は -24 と 24 を除く -24 と 24 の間です。ローカル時刻から CURRENT TIMEZONE を減算すると、ローカル時刻が UTC に変換されます。時刻は、SQL ステートメントが実行されるときに、オペレーティング・システムの時刻から計算されます。[25]

CURRENT TIMEZONE 特殊レジスターは、時刻やタイム・スタンプの算術演算など、 DECIMAL(6,0) のデータ・タイプの式が使用されるところならどこででも使用できます。

例

以下の例は、RECEIVED 列の UTC タイム・スタンプを使って、 IN_TRAY 表にレコードを挿入します。

```
INSERT INTO IN_TRAY VALUES (
       CURRENT TIMESTAMP - CURRENT TIMEZONE,
       :source,
       :subject,
       :notetext )
```

24. 世界標準時 (Coordinated Universal Time)。旧 GMT。

25. CURRENT TIMEZONE の値は、C の実行時関数によって決まります。時間帯に関連したインストール要件については、 *概説およびインストール* を参照してください。

USER

USER 特殊レジスターは、データベースでアプリケーションが始動するときにデータベース・マネージャーに渡される実行時許可 ID を指定します。このレジスターのデータ・タイプは VARCHAR(128) です。

例

以下の例は、ユーザー自身が入れたすべてのメモを IN_TRAY 表から選択します。

```
SELECT * FROM IN_TRAY
  WHERE SOURCE = USER
```

列名

列名 の意味は文脈によって異なります。 列名は以下の目的に使用できます。

- 列の名前を宣言する (CREATE TABLE ステートメントなどで)。
- 列を識別する (CREATE INDEX ステートメントなどで)。
- 列の値を指定する (以下に示すような文脈で)。
  - 列関数においては、列名によって、その関数が適用されるグループまたは中間結果表の列のすべての値が指定されます。 (グループおよび中間結果表については、421ページの『第5章 照会』を参照。) たとえば、MAX(SALARY) は、あるグループの SALARY 列の中のすべての値に関数 MAX が適用されることを表します。
  - GROUP BY または ORDER BY 文節の中では、その文節が適用される中間結果表の中のすべての値を、列名によって指定します。たとえば、ORDER BY DEPT と指定すると、 DEPT 列の値によって中間結果表が順序付けられます。
  - 式、探索条件、またはスカラー関数においては、列名によって、その構成項目が適用されるそれぞれの行またはグループの値が指定されます。たとえば、探索条件 CODE = 20 が何らかの行に適用される場合、列名 CODE によって指定される値は、その行の CODE 列の値です。
- FROM 文節における *table-reference* の *correlation-clause* のように、列の名前を一時的に変更する。

修飾子付き列名

列名の修飾子としては、表名、視点名、ニックネーム、別名、または相関名が使用されます。

列名が修飾されるかどうかは、文脈によって異なります。

- COMMENT ON ステートメントの形式に応じ、単一の列名に修飾の必要な場合があります。複数の列名は修飾してはなりません。
- 列名が列の値を指定している場合、修飾することができます。
- 上記以外の文脈では、列名を修飾してはいけません。

修飾子がオプションである場合、修飾子には 2 つの目的があります。これについては、146ページの『あいまいさを避けるための列名修飾子』および 148ページの『相関参照における列名修飾子』で説明されています。

相関名

相関名 は、照会の FROM 文節、および UPDATE または DELETE ステートメントの第 1 文節で指定されます。たとえば、FROM X.MYTABLE Z という文節では、Z が X.MYTABLE の相関名として確立されます。

```
FROM X.MYTABLE Z
```

X.MYTABLE の相関名として Z が定義されると、その SELECT ステートメントにおいては Z だけが X.MYTABLE インスタンスの列を参照する修飾子として使用できます。

相関名は、それが定義されている文脈内でのみ、表、視点、ニックネーム、別名、ネストされた表の式または表関数に関連付けられます。したがって、別の目的に使用するために、異なるステートメントの中や同じステートメントの異なる文節の中で同じ相関名を定義できます。

修飾子としての相関名は、あいまいさの回避や、相関参照の確立に利用できます。また、表、視点、ニックネーム、別名の単なる短縮名として利用することもできます。ネストされた表の式または表関数の場合、相関名は結果表を識別するのに必要になります。例では、X.MYTABLE と何度も入力するのを避けるためだけに使用されています。

相関名を表名、視点名、ニックネーム、または別名に対して指定する場合、その表、視点、ニックネーム、または別名のそのインスタンスでの列への修飾子付き参照は、その表名、視点名、ニックネーム、別名ではなく、相関名を使用しなければなりません。たとえば、以下の例の EMPLOYEE.PROJECT への参照は、 EMPLOYEE に対する相関名がすでに指定されているため誤りです。

例:

```
FROM EMPLOYEE E
  WHERE EMPLOYEE.PROJECT='ABC'      * incorrect*
```

PROJECT に対する修飾子付き参照では、以下のように、相関名 “E” を EMPLOYEE の代わりに使用する必要があります。

```
FROM EMPLOYEE E
  WHERE E.PROJECT='ABC'
```

FROM 文節で指定する名前は、*直接的* か*間接的* のどちらかです。相関名が指定されていない場合の表名、視点名、ニックネーム、または別名は FROM 文節の直接的な名前です。相関名は常に直接的な名前です。たとえば、以下の FROM 文節では、EMPLOYEE には相関名が指定され、 DEPARTMENT には指定されていません。このため、DEPARTMENT は直接的な名前、EMPLOYEE は間接的な名前になります。

```
FROM EMPLOYEE E, DEPARTMENT
```

FROM 文節の直接的な表名、視点名、ニックネーム、または別名は、その FROM 文節での直接的なその他の表名、視点名、ニックネーム、または FROM 文節の相関名のどれかと同じになる場合があります。これにより列名参照があいまいとなり、エラー (SQLSTATE 42702) となる可能性があります。

以下に示す最初の 2 つの FROM 文節は、直接的な名前である EMPLOYEE をそれぞれ 2 回以上参照するような参照を行っていないので正しい FROM 文節です。

1. 次の FROM 文節が与えられているものとします。

   ```
   FROM EMPLOYEE E1, EMPLOYEE
   ```

   EMPLOYEE.PROJECT のような修飾子付き参照は、 FROM 文節での EMPLOYEE の 2 番目のインスタンスの列を指すことになります。EMPLOYEE の 1 番目のインスタンスに対する修飾子付き参照では、相関名 “E1” を使用する (E1.PROJECT) 必要があります。
2. 次の FROM 文節が与えられているものとします。

   ```
   FROM EMPLOYEE, EMPLOYEE E2
   ```

   EMPLOYEE.PROJECT のような修飾子付き参照は、 FROM 文節での EMPLOYEE の 1 番目のインスタンスの列を指すことになります。EMPLOYEE の 2 番目のインスタンスに対する修飾子付き参照では、相関名 “E2” を使用する (E2.PROJECT) 必要があります。
3. 次の FROM 文節が与えられているものとします。

   ```
   FROM EMPLOYEE, EMPLOYEE
   ```

この文節では、直接的な 2 つの表名 (EMPLOYEE と EMPLOYEE) が同じになっています。これ自体は可能ですが、特定の列名への参照があいまいになってしまいます (SQLSTATE 42702)。

4. 次のステートメントが与えられているものとします。

```
SELECT *
  FROM EMPLOYEE E1, EMPLOYEE E2             * incorrect *
  WHERE EMPLOYEE.PROJECT = 'ABC'
```

修飾子付き参照 EMPLOYEE.PROJECT は誤りです。これは、FROM 文節の EMPLOYEE の 2 つのインスタンスの両方に相関名があるためです。そうするのではなく、PROJECT を参照するときは、どちらかの相関名 (E1.PROJECT または E2.PROJECT) で修飾する必要があります。

5. 次の FROM 文節が与えられているものとします。

```
  FROM EMPLOYEE, X.EMPLOYEE
```

EMPLOYEE の 2 番目のインスタンスの列を参照するときは、X.EMPLOYEE を使用する (X.EMPLOYEE.PROJECT) 必要があります。 X が、動的 SQL では CURRENT SCHEMA 特殊レジスター値、静的 SQL では QUALIFIER プリコンパイル / バインド・オプションである場合、そのような参照はあいまいなので列を参照することはできません。

FROM 文節で相関名を使用することにより、結果表の列に関連付けられる列名のリストを指定することもできます。相関名の場合と同じように、このようにリストされた列名は、照会時に列の参照に使用する必要がある列の*直接的な* 名前になります。列名のリストを指定する場合、基礎表の列名は*間接的な名前* になります。

次の FROM 文節が与えられているものとします。

```
  FROM DEPARTMENT D (NUM,NAME,MGR,ANUM,LOC)
```

D.NUM などの修飾子のついた参照は、 DEPTNO として表に定義されている DEPARTMENT 表の最初の列を表します。この FROM 文節を使用した D.DEPTNO の参照は、列名 DEPTNO が間接的な列名であるため誤りです。

あいまいさを避けるための列名修飾子

関数、GROUP BY 文節、ORDER BY 文節、式、または探索条件の文脈では、列名は、何らかの表、視点、ニックネーム、ネストされた表の式あるいは表関数の列の値を指します。列を含む可能性のある表、視点、ニックネーム、ネストされた表の式および表関数は、その文脈のオブジェクト表 と呼ばれます。複数の表に同じ名前の列が含まれている場合があります。列名を修飾する理由の

1 つは、列がどの表のものかを指定することです。列名の修飾子は、SQL プロシージャーにおいて、列名と SQL ステートメントで使われる SQL 変数名を区別するときにも役立ちます。

ネストされた表の式または表関数は、 FROM 文節で先行する *table-references* をオブジェクト表と見なします。後続の *table-references* はオブジェクト表とは見なされません。

表指定子

特定のオブジェクト表を指定する修飾子は、*表指定子* と呼ばれます。オブジェクト表を指定する文節では、そのオブジェクト表に対する表指定子も設定します。以下の例は、SELECT 文節の式のオブジェクト表を、直後の FROM 文節で指定しています。

```
SELECT CORZ.COLA, OWNY.MYTABLE.COLA
  FROM OWNX.MYTABLE CORZ, OWNY.MYTABLE
```

FROM 文節の表指定子は次のように設定されます。

- 表、視点、ニックネーム、別名、ネストされた表の式または表関数の後に続く名前は、相関名でもあり表指定子でもあります。したがって、CORZ は表指定子です。選択リストの中で、最初の列名を修飾するために CORZ が使用されています。
- 直接的な表名、視点名、ニックネーム、または別名は、表指定子です。したがって、OWNY.MYTABLE は表指定子です。選択リストの中で、第 2 の列名を修飾するために OWNY.MYTABLE が使用されています。

列へのあいまいな参照が生じる可能性を避けるため、各表指定子は、特定の FROM 文節の中では固有でなければなりません。

未定義またはあいまいな参照の回避

列名が列の値を参照する場合、その名前はただ 1 つのオブジェクト表の中に含まれているものでなければなりません。以下の状態はエラーとみなされます。

- 指定された名前の列を含むオブジェクト表がない。この参照は未定義になります。
- 列名が表記述文字によって修飾されているが、指定された表に指定された名前の列が入っていない。この参照も未定義になります。
- 名前が修飾なしで、2 つ以上のオブジェクト表の中にその名前の列がある。この参照はあいまいです。
- 列名が表指定子で修飾されているが、その指定されている表が FROM 文節の中で固有でなく、指定されている表のどちらのオカレンスにもその列がある。この参照はあいまいです。

- 列名は、TABLE キーワードが先行しないネストされた表の式、もしくは右外部結合または全外部結合の右側のオペランドである表関数またはネストされた表の式にあります。列名は、ネストされた表の式の全選択内の *table-reference* の列を指しません。この参照は未定義になります。

固有に定義された表指定子で列名を修飾することによって、あいまいな参照を避けてください。列が名前の異なる複数のオブジェクト表の中に含まれている場合、その表名を指定子として使用することができます。また、相関名の後に続いて列名のリストを使用してオブジェクト表のいずれかの列に固有名を指定することによって、表指定子を使用しなくてもあいまいな参照を避けることができます。

列を直接的な表名形式の表指定子で修飾するとき、直接的な表名は修飾の付いた形式でも付かない形式でも使用できます。しかし、表名、視点名、またはニックネームと、表指定子を完全に修飾した後は、使用される修飾子と表が同じものでなければなりません。

1. たとえば、ステートメントの許可 ID が CORPDATA とすると、以下のステートメントは有効です。

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE
   ```

   は有効なステートメントです。

2. ステートメントの許可 ID が REGION の場合、以下は無効です。

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE                          * incorrect *
   ```

   これは、EMPLOYEE が表 REGION.EMPLOYEE を表しているのに対し、WORKDEPT の修飾子が別の表 CORPDATA.EMPLOYEE を表しているためです。

相関参照における列名修飾子

全選択 とは、種々の SQL ステートメントの構成要素として使用される照会の 1 つの形式です。全選択については、421ページの『第5章 照会』を参照してください。任意のステートメントの探索条件で使用される全選択は、*副照会* と呼ばれます。ステートメントで式として単一値を検索するのに使用される全選択は、*スカラー全選択* または*スカラー副照会* と呼ばれます。照会の FROM 文節で使用される全選択は、*ネストされた表の式* と呼びます。探索条件、スカラー副照会、およびネストされた表の式の副照会を、このトピックのこれ以降の部分では副照会と呼びます。

副照会にはそれ自身の副照会を含めることができます。その副照会の中に、また副照会が含まれていてもかまいません。したがって、SQL ステートメントに副照会の階層が含まれることになる場合があります。副照会を含む階層の要素は、それに含まれている副照会よりも高いレベルとされます。

階層のあらゆる要素には、1 つ以上の表指定子が含まれています。副照会は、階層中の自分のレベルで指定されている表の列だけでなく、階層中のそれより前のレベルで指定されている表の列から階層の最上位で識別される表の列まで参照できます。上位のレベルで指定される表の列への参照は、*相関参照* と呼ばれます。

既存の SQL 標準規格との互換性のため、修飾子付きと修飾子なしのどちらの列名も相関参照として認められています。ただし、副照会で使用されるすべての列参照を修飾することをお奨めします。そうしないと、同一の列名により予期しない結果が生じることがあります。たとえば、ある階層の表が相関参照として同じ列名を含むように変更され、ステートメントが再度作成処理された場合、新たな参照は変更された表に対して適用されます。

副照会に含まれる列名が修飾されているときは、修飾されているその列名が出現するのと同じ副照会から探索が始まり、修飾子に一致する表指定子が見つかるまで、階層の上位へ向かって階層の各レベルの探索が続けられます。該当するものが見つかると、その表に指定の列があるかどうかが調べられます。列名の含まれているレベルより高いレベルで表が見つかった場合、これは表指定子が見つかったレベルに対する相関参照となります。ネストされた表の式の全選択より上の階層を探索するためには、ネストされた表の式の前にオプションの TABLE キーワードを指定しなければなりません。

副照会に含まれる列名が修飾されていないときは、その列名が出現するのと同じ副照会から始めて、階層の各レベルで参照されている表が探索され、一致する列名が見つかるまで、階層の上位へ向かって探索が続けられます。列名を含むレベルより高いレベルの表で列が見つかった場合は、その列を含む表が見つかったレベルに対する相関参照となります。列名が、特定のレベルの 2 つ以上の表で見つかった場合は、参照はあいまいになり、エラーとみなされます。

以下の例の T は、どの場合も、列 C を含む表指定子を参照しています。列名 T.C は、以下の条件がすべて満たされているときにのみ相関参照となります (この T は暗黙の修飾子か明示的な修飾子のいずれかを表します)。

- T.C は副照会の式で使用される。
- T が、その副照会の from 文節で使用されている表を指していない。
- T が、副照会を含む上位の階層レベルで使用されている表を示している。

同じ表、視点、またはニックネームが、多くのレベルで指定されていることがあるため、表指定子としては固有の相関名を使用するようお勧めします。 T が 2 つ以上のレベルで表の指定に使用される場合 (T は表名自体か重複の相関名)、 T.C は、T.C を含む副照会を最も直接的に含むように T が使用されているレベルを参照することになります。上位レベルへの相関が必要な場合、固有な相関名を使用する必要があります。

相関参照 T.C は、2 つの探索条件が、探索条件 1 が副照会で、探索条件 2 が上位のレベルでそれぞれ適用されている T の行またはグループでの C の値を識別します。条件 2 が WHERE 文節で使用される場合、副照会は条件 2 が適用される行ごとに評価されます。条件 2 が HAVING 文節で使用される場合、副照会は条件 2 が適用されるグループごとに評価されます。 (副照会の評価については、421ページの『第5章 照会』の WHERE 文節と HAVING 文節の部分を参照。)

たとえば、次のステートメントにおいて、(最後の行の) 相関参照 X.WORKDEPT は、最初の FROM 文節のレベルにある表 EMPLOYEE の WORKDEPT の値を指します。 (この文節は X を EMPLOYEE の相関名として設定します。) このステートメントは、その部署の平均給与を下回る社員のリストを作成するものです。

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM EMPLOYEE X
  WHERE SALARY < (SELECT AVG(SALARY)
                  FROM EMPLOYEE
                  WHERE WORKDEPT = X.WORKDEPT)
```

次の例は、THIS を相関名として使用しています。このステートメントは、社員のいない部門の行を削除します。

```
DELETE FROM DEPARTMENT THIS
   WHERE NOT EXISTS(SELECT *
                    FROM EMPLOYEE
                    WHERE WORKDEPT = THIS.DEPTNO)
```

ホスト変数の参照

ホスト変数 とは、以下のいずれかです。

- C の変数、C++ の変数、COBOL のデータ項目、FORTRAN の変数、または Java の変数など、ホスト言語の変数

または

- SQL 拡張機能を使って宣言された変数から SQL のプリコンパイラーによって生成されたホスト言語構成

これらは、SQL ステートメントで参照されています。ホスト変数はホスト言語のステートメントによって直接定義されるか、または SQL 拡張機能を使って間接的に定義されます。

SQL ステートメント内のホスト変数は、ホスト変数宣言規則に従ってプログラム内に記述されたホスト変数を識別する必要があります。

SQL ステートメントで使用されるホスト変数はすべて、 REXX を除くすべてのホスト言語の SQL DECLARE セクションで宣言する必要があります (アプリケーション・プログラムで SQL ステートメントのホスト変数を宣言する場合の詳細については、*アプリケーション開発の手引き* を参照してください)。 SQL DECLARE セクションで宣言されている変数と同じ名前の変数を、 SQL DECLARE セクションの外部で宣言することはできません。 SQL DECLARE セクションは、BEGIN DECLARE SECTION で始まり、 END DECLARE SECTION で終わります。

メタ変数の *host-variable* (ホスト変数) が構文図の中で使われる場合、それはホスト変数への参照を示します。 VALUES INTO 文節または、FETCH か SELECT INTO ステートメントの INTO 文節のホスト変数は、行の中の列の値または式の値が割り当てられるホスト変数を識別するものです。その他の文脈でのホスト変数は、アプリケーション・プログラムからデータベース・マネージャーに渡される値を指定します。

動的 SQL におけるホスト変数

動的 SQL ステートメントにおいては、ホスト変数の代わりにパラメーター・マーカーが使用されます。 パラメーター・マーカーは疑問符 (?) で表されます。これは、動的 SQL ステートメントにおいてアプリケーションが値を提供する位置、すなわち、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るとすれば、ホスト変数が来ることになる位置を示します。以下に、ホスト変数を使った静的 SQL ステートメントの例を示します。

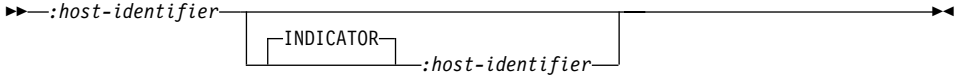
```
INSERT INTO DEPARTMENT
 VALUES (:hv_deptno, :hv_deptname, :hv_mgrno, :hv_admrdept)
```

次に、パラメーター・マーカーを使った動的 SQL ステートメントの例を示します。

```
INSERT INTO DEPARTMENT VALUES (?, ?, ?, ?)
```

パラメーター・マーカーについては、 1042ページの『PREPARE』の『パラメーター・マーカー』の項を参照してください。

構文図におけるメタ変数 *host-variable* (ホスト変数) は、一般に以下のように展開されます。



各 *host-identifier* (ホスト識別子) は、ソース・プログラムの中で宣言される必要があります。 2 番目のホスト識別子で指定される変数は、データ・タイプが短精度整数のものでなければなりません。

最初のホスト識別子 (host-identifier) は、*メイン変数* を指定します。演算に応じて、このホスト識別子はデータベース・マネージャーに値を提供したり、またはデータベース・マネージャーから提供される値を受け取ったりします。入力ホスト変数は、実行時アプリケーション・コード・ページの値を提供します。出力ホスト変数には、データが出力アプリケーション変数にコピーされるときに、必要に応じて実行時アプリケーション・コード・ページに変換される値が提供されます。指定されるホスト変数は、同じプログラム内で入力変数と出力変数の両方として機能できます。

2 番目の host-identifier (ホスト識別子) は、その*標識変数* を示します。標識変数の目的は以下のとおりです。

- ヌル値を指定する。標識変数の負の値は、ヌル値を指定するものとなります。 -2 の値は、結果を求める際に数値変換または演算式のエラーが発生したことを示します。
- 切り捨てられたストリングの元の長さを記録する (値のソースがラージ・オブジェクト・タイプでない場合)。
- ホスト変数に割り当てたときに時刻が切り捨てられた場合、その時刻の秒の部分を記録します。

たとえば、:HV1:HV2 を使用して挿入値または更新値を指定する場合に、 HV2 が負であると、指定される値はヌル値になります。 HV2 が負でない場合、指定される値は HV1 の値です。

同様に、:HV1:HV2 が VALUES INTO 文節、または FETCH あるいは SELECT INTO ステートメントに指定され、しかも戻された値がヌル値である場合には、HV1 は変更されず、HV2 は負の値に設定されます。[26] 戻された値がヌル値でない場合は、その値が HV1 に割り当てられ、 HV2 はゼロに設定されます。(ただし、 HV1 への割り当てに非 LOB ストリングのストリング切り捨てが必要になる場合を除きます。この場合 HV2 はストリングの元の長さに設定されます。) 割り当て時に時刻の秒の部分の切り捨てが必要な場合、HV2 は秒数に設定されます。

2 番目のホスト識別子が省略されている場合は、ホスト変数は標識変数を持たないことになります。ホスト変数参照 :HV1 によって指定される値は、常に HV1 の値であり、変数にヌル値を割り当てることはできません。したがって、この形式は、対応する列でヌル値を使えない場合以外は、INTO 文節では使用しないでください。この形式が使用された場合に、列にヌル値が含まれていると、データベース・マネージャーは実行時にエラーを生成します。

ホスト変数を参照する SQL ステートメントは、対象のホスト変数の宣言の範囲内にある必要があります。カーソルの SELECT ステートメントで参照されるホスト変数の場合、この規則は DECLARE CURSOR ステートメントではなく、OPEN ステートメントに適用されます。

例

PROJECT 表を使用し、プロジェクト (PROJNO) 'IF1000' について、ホスト変数 PNAME (VARCHAR(26)) はプロジェクト名 (PROJNAME) に、ホスト変数 STAFF (dec(5,2)) はスタッフ配置の平均レベル (PRSTAFF) に、ホスト変数 MAJPROJ (char(6)) は主要プロジェクト (MAJPROJ) に設定します。 PRSTAFF と MAJPROJ 列はヌル値である可能性があるため、標識変数 STAFF_IND (短精度整数) と MAJPROJ_IND (短精度整数) を使用します。

```
SELECT PROJNAME, PRSTAFF, MAJPROJ
  INTO :PNAME, :STAFF :STAFF_IND, :MAJPROJ :MAJPROJ_IND
  FROM PROJECT
  WHERE PROJNO = 'IF1000'
```

26. DFT_SQLMATHWARN を yes にしてデータベースが構成されている場合 (または静的 SQL ステートメントのバインドの過程である場合)、 HV2 を -2 にすることができます。 HV2 が -2 である場合、HV1 の数値タイプへの変換エラー、または HV1 の値を判別するために使用される演算式の評価エラーにより、 HV1 の値を戻すことができません。 DB2 ユニバーサル・データベースのバージョン 5 より前のクライアント・バージョンを使用してデータベースにアクセスする場合、 HV2 は算術例外に対して -1 になります。

MBCS の考慮事項: ホスト変数名にマルチバイト文字を使用できるかどうかは、ホスト言語によって決まります。

BLOB、CLOB、および DBCLOB のホスト変数の参照

通常の BLOB、CLOB、および DBCLOB の変数、 LOB のロケーター変数 (『ロケーター変数の参照』を参照)、および LOB ファイル参照変数 (155ページの『BLOB、CLOB、および DBCLOB ファイル参照変数の参照』を参照) は、すべてのホスト言語の中で定義可能です。 LOB が可能な場所では、構文図の *host-variable* (ホスト変数) という用語は、通常のホスト変数、ロケーター変数、またはファイル参照変数を指します。これらはネイティブのデータ・タイプではないため、SQL 拡張機能が使用され、それぞれの変数を表現するのに必要なホスト言語構成をプリコンパイラーが生成します。 REXX の場合、LOB はストリングにマップされます。

ラージ・オブジェクト値全体を保持できるほど大きい変数を定義することのできる場合もあります。このような場合で、サーバーからのデータ転送を据え置いてもパフォーマンス上のメリットが期待できない場合は、ロケーターを使用する必要はありません。しかし、ホスト言語やスペースの制限により、ラージ・オブジェクト全体を一度に一時記憶に保管するのが難しい場合がよくありますし、パフォーマンス上のメリットを考え合わせた上で、ラージ・オブジェクトはロケーターによって参照し、一度にラージ・オブジェクトの一部分だけを保持するホスト変数にオブジェクトの一部を選択して割り当て、そこで更新するという方法を採用することもできるかもしれません。

他のすべてのホスト変数と同様に、ラージ・オブジェクトのロケーター変数にも標識変数を対応させることができます。ラージ・オブジェクトのロケーター・ホスト変数に対応する標識変数は、他のデータ・タイプの標識変数と同じように動作します。データベースからヌル値が戻されると、標識変数が設定され、ロケーター・ホスト変数は変更されません。つまり、ロケーターがヌル値を指すことはないということです。

ロケーター変数の参照

ロケーター変数 は、アプリケーション・サーバーで LOB 値を表すロケーターを含むホスト変数です。 (ロケーターを使用して LOB 値を操作する方法については、87ページの『ロケーターによるラージ・オブジェクト (LOB) の操作』を参照。)

SQL ステートメントにおけるロケーター変数は、ロケーター変数の宣言規則に従ってプログラムに記述されたロケーター変数を識別したものでなければなりません。これは常に SQL ステートメントによって間接的に行われます。

構文図で locator-variable (ロケーター変数) の語が使用される場合、それはロケーター変数への参照を表します。メタ変数 *locator-variable* (ロケーター変数) は、 *host-variable* (ホスト変数) の場合と同じく、 *host-identifier* (ホスト識別子) を含めるように拡張されています。

ロケーターに対応する標識変数がヌル値のときは、参照される LOB の値はヌル値です。

現時点で何の値も表していないロケーター変数が参照されると、エラー (SQLSTATE 0F001) になります。

トランザクションのコミット時、またはトランザクションの終了時に、そのトランザクションが獲得していたロケーターはすべて解放されます。

BLOB、CLOB、および DBCLOB ファイル参照変数の参照

BLOB、CLOB、および DBCLOB のファイル参照変数は、 LOB の直接のファイル入出力に使用されるもので、すべてのホスト言語で定義可能です。 これらはネイティブのデータ・タイプではないため、 SQL 拡張機能が使用され、それぞれの変数を表現するのに必要なホスト言語構成をプリコンパイラーが生成します。 REXX の場合、LOB はストリングにマップされます。

LOB ロケーターは LOB バイトを含むものではなく LOB バイトを表すものであるのと同じように、ファイル参照変数はファイルを含むものではなく間接的に指し示します。データベースの照会、更新、および挿入では、ファイル参照変数を使用して 1 つの列値を保管したり検索したりすることができます。

ファイル参照変数には以下の特性があります。

データ・タイプ
BLOB、CLOB、または DBCLOB。この特性は、変数の宣言時に指定されます。

方向
これはアプリケーション・プログラムによって実行時に指定される必要があります (ファイル・オプション値の一部として)。方向は以下のどちらかです。

- 入力 (EXECUTE ステートメント、OPEN ステートメント、 UPDATE ステートメント、INSERT ステートメント、または DELETE ステートメントでのデータのソースとして使用されます)。

- 出力 (FETCH ステートメントまたは SELECT INTO ステートメントのデータの宛先として使用されます)。

ファイル名

これはアプリケーション・プログラムによって実行時に指定される必要があります。以下のいずれかです。

- ファイルの完全パス名 (こちらをお勧めします)。
- 相対ファイル名。相対ファイル名を指定した場合、それはクライアント・プロセスの現行パスに追加されます。

アプリケーション内では、ファイルは 1 つのファイル参照変数でのみ参照する必要があります。

ファイル名の長さ

これはアプリケーション・プログラムによって実行時に指定される必要があります。ファイル名の長さをバイト単位で表したものです。

ファイル・オプション

アプリケーションがファイル参照変数を使用するには、事前にいくつかのオプションの中の 1 つをその変数に割り当てる必要があります。オプションの設定は、ファイル参照変数構造の中のフィールドの INTEGER 値によって行います。ファイル参照変数ごとに、以下の値の 1 つを指定する必要があります。

- 入力 (クライアントからサーバーへ)

  SQL_FILE_READ [27]

  これは、オープン、読み取り、クローズの対象となる通常のファイルです。

- 出力 (サーバーからクライアントへ)

  SQL_FILE_CREATE [28]

  新規ファイルを作成しま

27. COBOL では SQL-FILE-READ、FORTRAN では sql_file_read、REXX では READ です。

28. COBOL では SQL-FILE-CREATE、FORTRAN では sql_file_create、REXX では CREATE です。

す。該当のファイルがすでに存在していると、エラーになります。

SQL_FILE_OVERWRITE (上書き) [29]
: 指定した名前のファイルが存在している場合には上書きされ、存在していない場合には新たなファイルが作成されます。

SQL_FILE_APPEND [30]
: 指定した名前のファイルが存在している場合には出力がそれに追加されます。存在していない場合には新たなファイルが作成されます。

データ長
: これは入力では使用されません。出力のとき、ファイルに書き込まれる新規データの長さがこのデータ長に設定されます。このデータ長はバイト単位です。

他のすべてのホスト変数と同様に、ファイル参照変数にも標識変数を対応させることができます。

出力ファイル参照変数の例 (C の場合)

- 宣言部が以下のようにコーディングされているとします。

```
EXEC SQL BEGIN DECLARE SECTION
   SQL TYPE IS CLOB_FILE  hv_text_file;
   char  hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

これを前処理した後は以下のようになります。

29. COBOL では SQL-FILE-OVERWRITE、FORTRAN では sql_file_overwrite、REXX では OVERWRITE です。

30. COBOL では SQL-FILE-APPEND、FORTRAN では sql_file_append、REXX では APPEND です。

```
EXEC SQL BEGIN DECLARE SECTION
   /* SQL TYPE IS CLOB_FILE  hv_text_file; */
   struct {
       unsigned long  name_length; //  File Name Length
       unsigned long  data_length; //  Data Length
       unsigned long  file_options; // File Options
       char           name[255];   // File Name
   } hv_text_file;
   char  hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

その後、以下のコードを使って、データベースの CLOB の列から選択し、:hv_text_file で参照される新規ファイルに書き込むことができます。

```
strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
hv_text_file.file_options = SQL_FILE_CREATE;

EXEC SQL SELECT content INTO :hv_text_file from papers
     WHERE TITLE = 'The Relational Theory behind Juggling';
```

入力ファイル参照変数の例 (C の場合)

- 前出の例と同じ宣言部を考えます。以下のコードは、:hv_text_file によって参照される通常ファイルからのデータを CLOB の列へ挿入するものです。

```
strcpy(hv_text_file.name, "/u/gainer/patents/chips.13");
  hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
  hv_text_file.file_options = SQL_FILE_READ:
 strcpy(:hv_patent_title, "A Method for Pipelining Chip Consumption");

  EXEC SQL INSERT INTO patents( title, text )
           VALUES(:hv_patent_title, :hv_text_file);
```

構造タイプ・ホスト変数の参照

構造タイプ変数は、FORTRAN、REXX、および Java を除く、すべてのホスト言語で定義できます。これらはネイティブのデータ・タイプではないため、SQL 拡張機能が使用され、それぞれの変数を表現するのに必要なホスト言語構成をプリコンパイラーが生成します。

他のすべてのホスト変数と同様に、構造タイプ変数にも標識変数を対応させることができます。構造タイプ・ホスト変数に対応する標識変数は、他のデータ・タイプの標識変数と同じように動作します。データベースからヌル値が戻されると、標識変数が設定され、構造タイプ・ホスト変数は変更されません。

構造タイプ用の実際のホスト変数は、組み込みデータ・タイプとして定義されます。構造タイプと関連した組み込みデータ・タイプは、以下のようなアクセスが可能でなければなりません。

- プリコンパイル・コマンドで指定した TRANSFORM GROUP オプションによって定義したように、構造タイプの FROM SQL 変形関数の結果に基づくアクセス。
- プリコンパイル・コマンドで指定した TRANSFORM GROUP オプションによって定義したように、構造タイプの TO SQL 変形関数のパラメーターへのアクセス。

ホスト変数の代わりにパラメーター・マーカーを使用している場合、 SQLDA に適切なパラメーター・タイプの特性を指定する必要があります。この場合、SQLDA には SQLVAR 構造のセットが「2 つ」必要です。また、2 番目の SQLVAR の SQLDATATYPE_NAME フィールドには、構造タイプのスキーマおよびタイプ名を入れなければなりません。 SQLDA 構造でスキーマを省略すると、エラーが発生します (SQLSTATE 07002)。このトピックの詳細は、1217 ページの『付録C. SQL 記述子域 (SQLDA)』を参照してください。

例

C プログラムで、(組み込みタイプの BLOB(1048576) を使い、タイプ POLYGON の) ホスト変数 *hv_poly* と *hv_point* を定義します。

```
EXEC SQL BEGIN DECLARE SECTION;
      static SQL
         TYPE IS POLYGON AS BLOB(1M)
         hv_poly, hv_point;
EXEC SQL END DECLARE SECTION;
```

関数

データベース関数 とは、一連の入力データの値と一連の結果の値との間の関係です。たとえば TIMESTAMP 関数は、入力データ値として DATE および TIME データ・タイプを受け取り、 TIMESTAMP の結果を生成します。関数には組み込み関数とユーザー定義関数があります。

- *組み込み関数* はデータベース・マネージャーに用意されている関数で、 1 つの結果値を返します。組み込み関数は SYSIBM スキーマの一部として識別されます。このような関数の例としては、AVG などの列関数、 ″+″ などの演算関数、DECIMAL などのキャスト関数、 SUBSTR などのその他の関数があります。
- *ユーザー定義関数* は、 SYSCAT.FUNCTIONS のデータベースに登録されている (CREATE FUNCTION ステートメントを使って) 関数です。ユーザー定義関数は SYSIBM スキーマの一部ではありません。このような関数の集合の 1 つは、 SYSFUN という名前のスキーマでデータベース・マネージャーに提供されています。

ユーザー定義関数の使用により、DB2 ではユーザーおよびアプリケーションの開発者は、ユーザーもしくは他社によって提供された関数定義を追加することによってデータベース・システムの関数を拡張して、データベース・エンジン自体に適用することができます。これにより、データベースから行を検索し、検索されたデータに関数を適用してさらに修飾するかまたはデータ整理を実行する場合よりも、パフォーマンスを高めることができます。また、データベース関数を拡張することにより、データベースはアプリケーションが使用するのと同じ関数をエンジンで活用することができ、アプリケーションとデータベースとの間の共同作用が高まり、またオブジェクト指向性が高まるためにアプリケーション開発者にとって生産性向上に役立ちます。

SYSIBM および SYSFUN スキーマの関数のリストは、232ページの表15 に示されています。

外部、SQL およびソース・ユーザー定義関数

ユーザー定義関数は、*外部関数*、*SQL 関数*、または*ソース関数* とすることができます。*外部関数* は、オブジェクト・コード・ライブラリーと関数呼び出し時に実行されるそのライブラリー内の関数によってデータベースに対して定義されます。列関数を外部関数にすることはできません。*ソース関数からの派生関数* は、データベースがすでに認識している別の組み込み関数またはユーザー定義関数への参照によって、データベースに対して定義されます。ソース関数からの派生関数はスカラー関数または列関数です。これらは、ユーザー定義タイプの既存の関数を使用する場合のサポートにきわめて有用です。 *SQL 関数* は、SQL RETURN ステートメントだけを使用して、データベースに対して定義されます。スカラー値、行、または表のいずれかを戻すことができます。SQL 関数を、列関数とすることはできません。

スカラー、列、行および表ユーザー定義関数

各ユーザー定義関数は、*スカラー関数*、*列関数*、または*表関数* として類別することもできます。

スカラー関数 は、呼び出されるたびに単一値の応答を戻す関数です。たとえば、組み込み関数 SUBSTR() はスカラー関数です。スカラー UDF は、外部関数またはソースからの派生関数のいずれであってもかまいません。

列関数 は、概念上、類似値の集合 (列) を渡され、単一値の応答を戻す関数です。 DB2 では、場合によっては*総計関数* と呼ばれることもあります。列関数の例として、組み込み関数 AVG() があります。外部列 UDF を DB2 に対して定義することはできませんが、組み込み列関数のいずれかをソースとして派生する列 UDF を定義することができます。これは、特殊タイプに対して有用

です。たとえば、特殊タイプ SHOESIZE が基本タイプ INTEGER を使用して定義されている場合、組み込み関数 AVG(INTEGER) をソースとする UDF AVG(SHOESIZE) を定義することができ、これは列関数になります。

行関数 とは、値を一行で戻す関数です。これは、構造タイプの属性値を行の値に割り当てる変形関数としてのみ、使うことができます。行関数は、SQL 関数として定義する必要があります。

表関数 は、その関数を参照する SQL ステートメントに表を戻す関数です。SELECT の FROM 文節でのみ参照することができます。このような関数を使用して、DB2 データ以外のデータに SQL 言語処理能力を適用することや、このようなデータを DB2 表に変換することができます。たとえば、ファイルを取り出してそれを表に変換することや、 WWW からデータをサンプリングしてそれを表にすること、あるいは Lotus Notes データベースにアクセスして、日付、送信元、メッセージのテキストなどのすべてのメール・メッセージに関する情報を戻すこと、などを行うことができます。このような情報は、データベースの他の表と結合することができます。表関数は、外部関数または SQL 関数として定義できます (表関数を、ソースから派生される関数にすることはできません)。

関数シグニチャー

関数は、そのスキーマ、関数名、パラメーター数、およびそのパラメーターのデータ・タイプによって識別されます。これは*関数シグニチャー* と呼ばれ、データベース内で固有である必要があります。パラメーターの数やパラメーターのデータ・タイプが違っていれば、 1 つのスキーマに同じ名前の関数が複数存在してもかまいません。複数の関数インスタンスのある関数名は、*多重定義* 関数と呼ばれます。ある関数名があるスキーマ内で多重定義される場合、そのスキーマにはその名前で 2 つ以上の関数があるということです (パラメーター・タイプが異なっているものでなければなりません)。関数名は SQL パスにおいても多重定義可能です。その場合、そのパスにその名前の関数が 2 つ以上ありますが、必ずしもパラメーター・タイプが異なる必要はありません。

SQL パス

関数は、使用可能な文脈で、その後に括弧に入った引き数のリストを伴うその修飾名 (スキーマ名および関数名) を参照することによって、呼び出すことができます。また、スキーマ名を指定せずに関数を呼び出すことも可能であり、その場合は、異なるスキーマの関数のうち、同じパラメーターまたは許容可能なパラメーターをもつ可能な関数を選択できることになります。このような場合、*関数解決* 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、および受け入れ可能なデータ・タイプを持つ関数を

識別するために探索されるスキーマのリストです。静的 SQL ステートメントに対する SQL パスは、 FUNCPATH バインド・オプションを使って指定されます (詳細については、 *コマンド解説書* を参照)。動的 SQL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 (138ページの『CURRENT PATH』を参照してください)。

関数解決

特定の関数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関数の中でどれが『最適』かを判別する必要があります。これには、組み込み関数とユーザー定義関数の中から関数を解決する処理が含まれます。

引き数 とは、呼び出し時に関数に渡される値です。 SQL の中で呼び出されるとき、関数にはゼロ個以上の引き数のリストが渡されます。このような引き数は、引き数が引き数リストの位置によって決定されるという意味で定位置と言えます。パラメーター は、関数への入力の形式的な定義です。関数がデータベースに対して内部的に (組み込み関数) またはユーザーによって (ユーザー定義関数) 定義されるとき、関数のパラメーターが (ゼロ個以上) 指定されます。パラメーターの定義の順序がパラメーターの位置、およびその結果としてパラメーターの意味を定義することになります。したがって、どのパラメーターも関数の特定の定位置入力です。呼び出し時に、引き数リスト中のその位置によって、引き数は特定のパラメーターに対応します。

データベース・マネージャーは、呼び出しで指定される関数名、引き数の数とデータ・タイプ、SQL パスの中で同じ名前をもつすべての関数、対応するパラメーターのデータ・タイプを使用して、ある関数を選択するかどうかの判断基準とします。関数を決定する過程で発生する可能性のある結果について以下に示します。

1. 特定の関数が最適であると判断される場合。たとえば、名前が RISK、スキーマが TEST、シグニチャーが以下のように定義された関数を考えてみます。

   ```
   TEST.RISK(INTEGER)
   TEST.RISK(DOUBLE)
   ```

   SQL パスには TEST スキーマが含まれており、以下のように関数が参照されたとします (DB は DOUBLE 列)。

   ```
   SELECT ... RISK(DB) ...
   ```

   この場合は、2 番目の RISK が選択されます。

   以下のように関数が参照される場合は (SI は SMALLINT 列)、

```
SELECT ... RISK(SI) ...
```

最初の RISK が選択されます。これは、SMALLINT は INTEGER にプロモート可能であり、優先順位リストでの順位がさらに下になる DOUBLE よりも一致性が高いためです (101ページの表5 を参照)。

構造タイプである引き数について考慮する場合、優先順位リストには、静的タイプの引き数のスーパータイプが含まれます。最適なのは、構造タイプ階層の中で、静的タイプの関数引き数に最も近いスーパータイプ・パラメーターで定義する関数です。

2. 許容できる適合性をもつ関数がないと判断される場合。たとえば、前出の例と同じ 2 つの関数が指定され、以下のように関数が参照されたとします (C は CHAR(5) 列)。

   ```
   SELECT ... RISK(C) ...
   ```

   この場合、引き数はどちらの RISK 関数のパラメーターとも整合性がありません。

3. SQL パス、および呼び出し時に渡される引き数の数とデータ・タイプに基づいて特定の関数が選択される場合。たとえば、名前が RANDOM で、シグニチャーが以下のように定義されている関数を考えてみます。

   ```
   TEST.RANDOM(INTEGER)
   PROD.RANDOM(INTEGER)
   ```

   SQL パスは以下のとおりです。

   ```
   "TEST","PROD"
   ```

   この場合、次の関数参照では、

   ```
   SELECT ... RANDOM(432) ...
   ```

   TEST.RANDOM が選択されます。これは、どちらの RANDOM 関数も同程度に高い一致性を示し (この例の場合は完全一致)、どちらのスキーマも関数パスにありますが、 SQL パスにおいて TEST の方が PROD より先に指定されているためです。

最適な関数を選択する方式

引き数のデータ・タイプと、対象とする関数のパラメーターに定義されているデータ・タイプとの比較は、似たような名前の関数の中でどれが『最適』かを決定する基準となります。関数の結果のデータ・タイプまたは関数のタイプ (列、スカラー、または表) は、この決定には関係し**ない**ことに注意してください。

関数解決は、以下の手順で行われます。

1. まず、カタログ (SYSCAT.FUNCTIONS) と組み込み関数から、以下のすべての条件が真となるすべての関数を探します。
   a. 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、スキーマ名と関数名が呼び出し名に一致する。
   b. 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、関数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。
   c. 定義済みパラメーターの数が呼び出しと一致している。
   d. 呼び出しの各引き数のデータ・タイプが、関数の対応する定義済みパラメーターのデータ・タイプに一致するか、またはそのデータ・タイプに『プロモート可能』である (101ページの『データ・タイプのプロモーション』を参照)。
2. 次に、関数呼び出しの個々の引き数を左から右に検討していきます。引き数ごとに、その引き数に対して*最適な一致* ではない関数をすべて除去していきます。ある引き数の*最適な一致* とは、 101ページの表5 の引き数データ・タイプに対応する優先順位リストの中で、そのデータ・タイプのパラメーターをもつ関数が存在するデータ・タイプのうち、最初に記述されているデータ・タイプです。長さ、精度、位取り、および ″FOR BIT DATA″ 属性は、この比較では考慮されません。たとえば、DECIMAL(9,1) の引き数は DECIMAL(6,5) のパラメーターと完全に一致するとみなされ、また VARCHAR(19) の引き数は VARCHAR(6) のパラメーターと完全に一致するとみなされます。

   ユーザー定義構造タイプ引き数に最適なのは、それ自体です。次に適しているのは、すぐ上のスーパータイプです。このことは、引き数の各スーパータイプに当てはまります。ここで考慮しているのは、静的タイプ (宣言済みタイプ) の構造タイプ引き数であり、動的タイプ (最も特定的なタイプ) ではありません。
3. ステップ 2 の後でも 2 つ以上の関数が候補として残っているときは、残りの関数候補のすべてが同一のシグニチャーを持ち、しかも異なるスキーマに存在するという状況であるはずです (アルゴリズムがそうなっています)。スキーマがそのユーザーの SQL パスで最初に出現する関数が選択されます。
4. ステップ 2 の後で候補となる関数が残らなかった場合は、エラー (SQLSTATE 42884) になります。

組み込み関数の場合の関数パスについての考慮事項

組み込み関数は SYSIBM という名前の特殊なスキーマに含まれています。SYSFUN スキーマにさらに別の使用可能な関数がありますが、それらは組み込み関数とはみなされません。それらはユーザー定義関数として開発されたものであり、処理上の特別な考慮事項がないためです。 SYSIBM または SYSFUN スキーマに (あるいは 『SYS』 の文字で始まる名前のスキーマに)、ユーザーがさらに関数を定義することはできません。

すでに説明したように、関数解決処理の中で、組み込み関数はユーザー定義関数とまったく同じように扱われます。関数解決の点で組み込み関数とユーザー定義関数の違いの 1 つは、組み込み関数は関数解決で常に検討される必要があるということです。したがって、パスから SYSIBM を省いても、関数およびデータ・タイプの解決では SYSIBM がパスの最初のスキーマであることが想定されることになります。

たとえば、ユーザーの SQL パスが以下のように定義されているとします。

```
"SHAREFUN","SYSIBM","SYSFUN"
```

また、引き数の数とタイプが SYSIBM.LENGTH と同じである LENGTH 関数が SHAREFUN スキーマに定義されているとします。この場合、このユーザーの SQL ステートメントで、修飾なしで LENGTH を参照すると、SHAREFUN.LENGTH が選択されます。一方、ユーザーの SQL パスが以下のように定義されている場合には、

```
"SHAREFUN","SYSFUN"
```

同じ SHAREFUN.LENGTH 関数が存在していたとしても、このユーザーの SQL ステートメントで修飾せずに LENGTH を参照すると、SYSIBM.LENGTH が選択されることになります。これは、SYSIBM がパスに指定されていないので、暗黙のうちに SYSIBM が 1 番目のパスとなるためです。

このような状況では、以下の手段により、発生の予想される問題を最小限にすることができます。

- ユーザー定義関数には組み込み関数の名前を決して使用しないようにするか、または、
- 何らかの理由で組み込み関数と同じ名前のユーザー定義関数を作成する必要のある場合は、そのような関数への参照に必ず修飾子を付ける。

関数解決の例

以下は、正常な関数解決の例を示しています。

3 つの異なるスキーマに 7 個の FOO 関数があり、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 FUNCTION AUGUSTUS.FOO (CHAR(5), INT, DOUBLE)  SPECIFIC FOO_1 ...
CREATE FUNCTION AUGUSTUS.FOO (INT, INT, DOUBLE)      SPECIFIC FOO_2 ...
CREATE FUNCTION AUGUSTUS.FOO (INT, INT, DOUBLE, INT) SPECIFIC FOO_3 ...
CREATE FUNCTION JULIUS.FOO   (INT, DOUBLE, DOUBLE)   SPECIFIC FOO_4 ...
CREATE FUNCTION JULIUS.FOO   (INT, INT, DOUBLE)      SPECIFIC FOO_5 ...
CREATE FUNCTION JULIUS.FOO   (SMALLINT, INT, DOUBLE) SPECIFIC FOO_6 ...
CREATE FUNCTION NERO.FOO     (INT, INT, DEC(7,2))    SPECIFIC FOO_7 ...
```

以下のように関数が参照されるとします (I1 および I2 は INTEGER 列、 D は DECIMAL 列です)。

```
SELECT ... FOO(I1, I2, D) ...
```

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"
```

この場合、アルゴリズムは次のようになります。

″NERO″ が SQL パスに含まれていないため、FOO_7 は候補から除かれます。

パラメーターの数が違っているため、FOO_3 は候補から除かれます。第 1 引き数が第 1 パラメーターのデータ・タイプにプロモートできないため、FOO_1 と FOO_6 はどちらも候補から除かれます。

この時点で複数の候補が残っているため、次に引き数が順に検討されます。最初の引き数については、残りのすべての関数 FOO_2、FOO_4 および FOO_5 がその引き数タイプと完全に一致します。この検討ではどの関数も検討の対象から除かれないため、次の引き数を検討する必要があります。

2 番目の引き数では、FOO_2 および FOO_5 が完全に一致しているのに対し、 FOO_4 は一致していないため、FOO_4 が検討の対象から除かれます。 FOO_2 と FOO_5 の間の何らかの差異を判別するために、さらに次の引き数が検討されます。

第 3 の最後の引き数では、 FOO_2 も FOO_5 も引き数のタイプと完全には一致していませんが、両方とも適合度は同程度です。

この時点で、パラメーター・シグニチャーが同じである関数として FOO_2 と FOO_5 の 2 つが残っています。最終的な決着を付けるのは、どちらの関数のスキーマが SQL パスで先にくるかであり、この基準によって FOO_5 が最終的に選択されます。

関数の呼び出し

関数が選択された後も、いくつかの理由でその関数の使用が許可されない場合があります。 個々の関数は特定のデータ・タイプの結果を返すように定義されています。この結果のデータ・タイプが、関数が呼び出される文脈と互換性がない場合、エラーが生じます。たとえば、今度は結果のデータ・タイプが異なる STEP という名前の関数が次のように定義されているとします。

```
STEP(SMALLINT) returns CHAR(5)
STEP(DOUBLE)   returns INTEGER
```

以下のように関数が参照されると (S は SMALLINT 列)、

```
SELECT ... 3 + STEP(S) ...
```

引き数タイプが完全に一致しているため、最初の STEP が選択されます。しかし、結果のタイプが加法演算子の引き数として求められる数値タイプではなく CHAR(5) であるため、ステートメントではエラーになります。

この他にこの状態が起きる場合の例は次のとおりです。いずれの例もステートメントのエラーが生じます。

1. 関数が FROM 文節で参照されたが、関数解決ステップで選択された関数がスカラー関数または列関数であった場合。
2. その逆の場合。つまり文脈がスカラー関数または列関数を要求し、関数解決が表関数を選択する場合。

関数呼び出しの引き数が、選択された関数のパラメーターのデータ・タイプと完全一致でない場合、列への割り当てと同じ規則を適用して、実行時に引き数がパラメーターのデータ・タイプに変換されます (106ページの『割り当てと比較』を参照)。これには、引き数とパラメーターの間で精度、位取り、または長さが異なる場合も含まれます。

メソッド

構造タイプのデータベース・メソッドは、一連の入力データ値と、一連の結果値との関連のことです。ここで、最初の入力値 (またはサブジェクト*引き数*) は、メソッドと同じ値になるか、サブジェクト・タイプ (サブジェクト・パラメーター ともいう) のサブタイプになります。たとえば、タイプが ADDRESS である CITY というメソッドは、タイプが VARCHAR の入力データ値に渡すことができます。結果は ADDRESS (または ADDRESS のサブタイプ) になります。

メソッドは、ユーザー定義構造タイプの定義の一部として、暗黙的にあるいは明示的に定義されます。

暗黙的に定義されたメソッドは、構造タイプごとに作成されます。また、構造タイプの属性ごとに、監視用メソッドが定義されます。監視用メソッドを使うと、アプリケーション側は、該当タイプのインスタンスの属性値を知ることができます。変更メソッドも属性ごとに定義されます。これにより、アプリケーション側では、タイプ・インスタンスの属性の値を変更することによってタイプ・インスタンスを変更できます。上記の CITY メソッドは、タイプ ADDRESS の変更メソッドの一例です。

明示的に定義したメソッド、すなわちユーザー定義メソッド は、 CREATE TYPE (または ALTER TYPE ADD METHOD) および CREATE METHOD ステートメントを組み合わせて使用して、 SYSCAT.FUNCTIONS のデータベースに登録されるメソッドです。特定構造タイプ用に定義されたメソッドはすべて、そのタイプと同じスキーマで定義されます。

構造タイプのユーザー定義メソッドの使用により、 DB2 ではユーザーおよびアプリケーションの開発者は、データベース・システムの関数を拡張することができます。このことは、ユーザーもしくは他社によって提供されたメソッド定義を追加し、データベース・エンジンの構造タイプ・インスタンスに適用することにより実現します。追加したメソッド定義により、データベースから行を取り出したり、取り出したデータに対して関数を適用するときの場合とは逆に、パフォーマンスが著しく向上します。さらに、データベース・メソッドを定義することにより、データベースは、アプリケーションで使われるエンジン内にある同じメソッドを活用できます。このようにすると、アプリケーションとデータベースとの間のやり取りの効率が、著しく向上します。これは、さらにオブジェクト指向になっているため、アプリケーション開発者は、生産性を高めることができます。

外部および SQL ユーザー定義メソッド

ユーザー定義メソッドは、外部メソッドとするか、 SQL 式に基づくものとすることができます。外部メソッドは、オブジェクト・コード・ライブラリーと、メソッド呼び出し時に実行されるそのライブラリー内の関数によって、データベースに対して定義されます。 SQL 式に基づいたメソッドは、メソッドの呼び出し時に、その SQL 式の結果を戻します。そのようなメソッドでは、すべてが SQL で作成されているので、オブジェクト・コード・ライブラリーが必要ありません。

ユーザー定義メソッドでは、呼び出されるたびに単一値の応答を戻すことができます。この値は、構造タイプとすることができます。また、メソッドを (SELF AS RESULT を使用して) *タイプ保持* として定義し、メソッドの戻され

るタイプとして、動的タイプのサブジェクト引き数を戻すようにすることが可能です。暗黙的に定義した変更メソッドは、タイプ保持されます。

メソッド・シグニチャー

メソッドは、そのサブジェクト・タイプ、メソッド名、パラメーター数、およびそのパラメーターのデータ・タイプによって識別されます。これはメソッド・シグニチャーと呼ばれ、データベース内で固有である必要があります。

以下の場合に、同じ名前を付けられた構造タイプのメソッドが、複数存在する可能性があります。

- パラメーターの数やパラメーターのデータ・タイプが違う場合。
- メソッドのサブジェクト・タイプのサブタイプまたはスーパータイプのために、同じシグニチャーが存在していない場合。
- (最初のパラメーターとして、サブジェクト・タイプ、またはそのサブタイプかスーパータイプを使用した) 同じ関数シグニチャーが存在しない場合。

複数のメソッド・インスタンスを持つメソッド名のことを、多重定義メソッドといいます。あるメソッド名があるタイプ内で多重定義される場合、そのタイプには、その名前で 2 つ以上のメソッドがあるということです (それぞれのタイプにはすべて、異なるパラメーター・タイプがあります)。メソッド名はサブジェクト・タイプ階層においても多重定義可能です。その場合、そのタイプ階層にその名前のメソッドが 2 つ以上ありますが、それぞれのメソッドには異なるパラメーター・タイプがなければなりません。

メソッドの呼び出し

メソッドを呼び出すときには、許可されている文脈で、構造タイプ・インスタンス (サブジェクト引き数) への参照と二重ドット演算子の両方が先頭に記されているメソッド名を参照します。その後に、括弧で囲まれた引き数のリストが続きます。実際に呼び出されるメソッドは、次の項で説明されているメソッド解決法により、静的タイプのサブジェクト・タイプに基づいて決められます。関数呼び出しを使い、WITH FUNCTION ACCESS で定義されているメソッドを呼び出すこともできます。その場合、関数解決のための通常のルールが適用されます。

メソッド解決

特定のメソッド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メソッドの中でどれが「最適」かを判別する必要があります。メソッド解決時には、関数 (組み込みまたはユーザー定義) は考慮されません。

引き数とは、呼び出し時にメソッドに渡される値です。 SQL の中で呼び出されるとき、メソッドには (特定構造タイプの) サブジェクト引き数、およびゼロ個以上の引き数のリストが渡されます。このような引き数は、引き数が引き数リストの位置によって決定されるという意味で定位置と言えます。このサブジェクト引き数は、最初の引き数であると見なされます。パラメーターは、メソッドへの入力の形式上の定義です。

メソッドがデータベースに対して暗黙的に (特定タイプ用にシステム生成される)、またはユーザーによって (ユーザー定義メソッド) 定義されるとき、メソッドのパラメーターが (最初のパラメーターとしてのサブジェクト・パラメーター付きで) 指定されます。パラメーターの定義の順序がパラメーターの位置、およびその結果としてパラメーターの意味を定義することになります。したがって、どのパラメーターもメソッドの特定の定位置入力です。呼び出し時に、引き数リスト中のその位置によって、引き数は特定のパラメーターに対応します。

データベース・マネージャーは、呼び出しで指定されるメソッド名、引き数の数とデータ・タイプ、サブジェクト引き数の静的タイプ (およびそのスーパータイプ) 用に同じ名前をもつすべてのメソッド、対応するパラメーターのデータ・タイプを使用して、あるメソッドを選択するかどうかの判断基準とします。

メソッドを決定する過程で発生する可能性のある結果について以下に示します。

1. 特定のメソッドが最適であると判断される場合。たとえば、シグニチャーが以下のように定義されたタイプ SITE の RISK というメソッドを考えてみます。

   ```
   PROXIMITY(INTEGER) FOR SITE
   PROXIMITY(DOUBLE) FOR SITE
   ```

   続くメソッドの呼び出しは次のようになります (ここで、ST は SITE 列、DB は DOUBLE 列です)。

   ```
   SELECT ST..PROXIMITY(DB) ...
   ```

   この場合は、2 番目の PROXIMITY が選択されます。

   以下のようにメソッドが呼び出される場合は (SI は SMALLINT 列)、

   ```
   SELECT ST..PROXIMITY(SI) ...
   ```

最初の PROXIMITY が選択されます。これは、SMALLINT は INTEGER にプロモート可能であり、優先順位リストでの順位がさらに下になる DOUBLE よりも一致性が高いためです。

構造タイプである引き数について考慮する場合、優先順位リストには、静的タイプの引き数のスーパータイプが含まれます。最適なのは、構造タイプ階層の中で、静的タイプの関数引き数に最も近いスーパータイプ・パラメーターで定義する関数です。

2. 許容できる適合性をもつメソッドがないと判断される場合。たとえば、前出の例と同じ 2 つの関数が指定され、以下のように関数が参照されたとします (C は CHAR(5) 列)。

```
SELECT ST..PROXIMITY(C) ...
```

この場合、引き数はどちらの PROXIMITY 関数のパラメーターとも整合性がありません。

3. タイプ階層のメソッド、および呼び出し時に渡される引き数の数とデータ・タイプに基づいて特定のメソッドが選択される場合。たとえば、シグニチャーが以下のように定義された、タイプ SITE および DRILLSITE (SITE のサブタイプ) の RISK というメソッドを考えてみます。

```
RISK(INTEGER) FOR DRILLSITE
RISK(DOUBLE) FOR SITE
```

続くメソッドの呼び出しは次のようになります (ここで、DRST は DRILLSITE 列、DB は DOUBLE 列です)。

```
SELECT DRST..RISK(DB) ...
```

DRILLSITE は SITE へプロモートできるので、この場合は、2 番目の RISK が選択されます。

以下のようにメソッドが参照される場合は (SI は SMALLINT 列)、

```
SELECT DRST..RISK(SI) ...
```

最初の RISK が選択されます。これは、 SMALLINT は INTEGER にプロモート可能 (優先順位リストでの順位が DOUBLE よりも近い) であり、DRILLSITE は、SITE よりも一致性が高いスーパータイプであるためです。

同じタイプ階層内のメソッドで同じシグニチャーを使い、サブジェクト・パラメーター以外のパラメーターで利用することはできません。

最適な選択をするための方式

引き数のデータ・タイプと、対象とするメソッドのパラメーターに定義されているデータ・タイプとの比較は、似たような名前のメソッドの中でどれが『最適』かを決定する基準となります。考慮しているメソッドの結果のデータ・タイプは、この決定には関係しないことに注意してください。

メソッド解決は、以下の手順で行われます。

1. まず、カタログ (SYSCAT.FUNCTIONS) から、以下のすべての条件が真となるすべてのメソッドを探します。
   - メソッド名が呼び出し名と同じで、サブジェクト・パラメーターが、サブジェクト引き数の静的タイプと同じであるか、そのスーパータイプである。
   - 定義済みパラメーターの数が呼び出しと一致している。
   - 呼び出しの各引き数のデータ・タイプが、メソッドの対応する定義済みパラメーターのデータ・タイプに一致するか、またはそのデータ・タイプに「プロモート可能」である (101ページの『データ・タイプのプロモーション』を参照)。
2. 次に、メソッド呼び出しの個々の引き数を左から右に検討していきます。左端の引き数 (すなわち最初の引き数) は、暗黙的な SELF パラメーターです。たとえば、タイプ ADDRESS_T に定義したメソッドには、暗黙的な最初のパラメーターとしてタイプ ADDRESS_T があります。

   引き数ごとに、その引き数に対して最適な一致ではない関数をすべて除去していきます。ある引き数の最適な一致とは、 101ページの表5 の引き数データ・タイプに対応する優先順位リストの中で、そのデータ・タイプのパラメーターをもつ関数が存在するデータ・タイプのうち、最初に記述されているデータ・タイプです。

   長さ、精度、位取り、および ″FOR BIT DATA″ 属性は、この比較では考慮されません。たとえば、DECIMAL(9,1) の引き数は DECIMAL(6,5) のパラメーターと完全に一致するとみなされ、また VARCHAR(19) の引き数は VARCHAR(6) のパラメーターと完全に一致するとみなされます。

   ユーザー定義構造タイプ引き数に最適なのは、それ自体です。次に適しているのは、すぐ上のスーパータイプです。このことは、引き数の各スーパータイプに当てはまります。ここで考慮しているのは、静的タイプ (宣言済みタイプ) の構造タイプ引き数であり、動的タイプ (最も特定的なタイプ) ではありません。
3. ほとんどの場合、ステップ 2 の実行後に候補メソッドが 1 つ残ります。このメソッドを選択します。

4. ステップ 2 の後で候補となるメソッドが残らなかった場合は、エラー (SQLSTATE 42884) になります。

メソッド解決の例

以下は、正常なメソッド解決の例を示しています。

GOVERNOR の階層内で定義された3 つの構造タイプに関し、HEADOFSTATE のサブタイプとしての EMPEROR のサブタイプとして、 7 つの FOO メソッドがあります。それぞれは、シグニチャーで登録されています。

```
CREATE METHOD FOO (CHAR(5), INT, DOUBLE)  FOR HEADOFSTATE SPECIFIC FOO_1 ...
CREATE METHOD FOO (INT, INT, DOUBLE)      FOR HEADOFSTATE SPECIFIC FOO_2 ...
CREATE METHOD FOO (INT, INT, DOUBLE, INT) FOR HEADOFSTATE SPECIFIC FOO_3 ...
CREATE METHOD FOO (INT, DOUBLE, DOUBLE)   FOR EMPEROR     SPECIFIC FOO_4 ...
CREATE METHOD FOO (INT, INT, DOUBLE)      FOR EMPEROR     SPECIFIC FOO_5 ...
CREATE METHOD FOO (SMALLINT, INT, DOUBLE) FOR EMPEROR     SPECIFIC FOO_6 ...
CREATE METHOD FOO (INT, INT, DEC(7,2))    FOR GOVERNOR    SPECIFIC FOO_7 ...
```

以下のようにメソッドが参照されるとします (I1 および I2 は INTEGER 列、D は DECIMAL 列、そして E は EMPEROR 列です)。

```
SELECT E..FOO(I1, I2, D) ...
```

アルゴリズムに従ってゆきます。

タイプ GOVERNOR は EMPEROR のサブタイプなので (スーパータイプではない)、 FOO_7 は候補から除かれます。

パラメーターの数が違っているため、FOO_3 は候補から除かれます。

第 1 引き数 (サブジェクト引き数ではない) が第 1 パラメーターのデータ・タイプにプロモートできないため、 FOO_1 と FOO_6 はどちらも候補から除かれます。この時点で複数の候補が残っているため、次に引き数が順に検討されます。

サブジェクト引き数の場合、FOO_2 はスーパータイプですが、 FOO_4 と FOO_5 はサブジェクト引き数に一致しています。

最初の引き数については、残りのメソッド FOO_4 および FOO_5 がその引き数タイプと完全に一致します。この検討ではどのメソッドも検討の対象から除かれないため、次の引き数を検討する必要があります。

2 番目の引き数では、FOO_5 が完全に一致しているのに対し、 FOO_4 は一致していないため、FOO_4 が検討の対象から除かれます。これにより、メソッド FOO_5 が選ばれます。

メソッドの呼び出し

メソッドが選択された後も、いくつかの理由でそのメソッドの使用が許可されない場合があります。

個々のメソッドは特定のデータ・タイプの結果を戻すように定義されています。この結果のデータ・タイプが、メソッドが呼び出される文脈と互換性がない場合、エラーが生じます。たとえば、STEP というメソッドが定義されていて、結果としてそれぞれが別々のデータ・タイプを持っているとします。

```
STEP(SMALLINT) FOR TYPEA RETURNS CHAR(5)
STEP(DOUBLE) FOR TYPEA RETURNS INTEGER
```

以下のようにメソッドが参照されると (S は SMALLINT 列で TA は TYPEA の列)、

```
SELECT 3 + TA..STEP(S) ...
```

引き数タイプが完全に一致しているため、最初の STEP が選択されます。しかし、結果のタイプが加法演算子の引き数として求められる数値タイプではなく CHAR(5) であるため、ステートメントではエラーになります。

選択したメソッドがタイプ保持メソッドである場合、以下の点に注意してください。

- 関数解決に続く静的結果タイプは、メソッド呼び出しのサブジェクト引き数の静的タイプと同じです。
- メソッドを呼び出すときの動的結果タイプは、メソッド呼び出しのサブジェクト引き数の動的タイプと同じです。

これは、タイプ保持メソッド定義で指定した結果タイプのサブタイプになる可能性がありますが、逆に、メソッドの処理時に実際に戻される動的タイプのスーパータイプになる場合もあります。

メソッド呼び出しの引き数が、選択されたメソッドのパラメーターのデータ・タイプと完全一致でない場合、列への割り当てと同じ規則を適用して、実行時に引き数がパラメーターのデータ・タイプに変換されます (106ページの『割り当てと比較』を参照)。これには、引き数とパラメーターの間で精度、位取り、または長さが異なる場合も含まれます。ただし、引き数の動的タイプが、パラメーターの静的タイプのサブタイプである場合を除きます。

保守的バインド・セマンティクス

ステートメントの処理時にデータベースで関数、メソッド、およびデータ・タイプを解決する場合がありますが、データベース・マネージャーはこの解決を繰り返すことができなければなりません。これは、以下の場合に当てはまります。

- パッケージ内の静的 DML ステートメント
- 視点
- トリガー
- 検査制約
- SQL ルーチン

パッケージ内の静的 DML ステートメントの場合、関数、メソッド、およびデータ・タイプの参照はバインド操作時に解決されます。視点、トリガー、検査制約の中にある関数、メソッド、およびデータ・タイプの参照は、データベース・オブジェクトの作成時に解決されます。

これらのオブジェクトに含まれている関数またはメソッド参照のいずれかで、2 度目の関数またはメソッド解決が実行される場合、一致の度合いは高くても、実際の実行可能プログラムには異なる操作を実行させるシグニチャーを持った新しい関数またはメソッドが追加されていると、動作が変わってしまうおそれがあります。同様に、これらのオブジェクトに含まれているデータ・タイプのいずれかで 2 度目の関数解決が実行される場合、他のスキーマにあるのと同じ名前 (SQL パスにも存在している) で新しいデータ・タイプが追加されていると、動作が変わってしまうおそれがあります。この問題を回避するため、データベース・マネージャーは*保守的バインド・セマンティクス* という概念を適用します。この概念は、関数やデータ・タイプの参照を解決するときに、バインド時と同じ SQL パスが使用されるようにします。また、解決時に考慮される関数[31]、メソッドおよびデータ・タイプの作成タイム・スタンプが、ステートメントをバインドした時刻よりも遅くならないようにします。[32] このようにして、関数、メソッド、およびデータ・タイプのうち、ステートメントを最初に処理したときの解決時に考慮されたものだけが考慮されるようにします。

31. バージョン 6.1 から追加された組み込み関数には、データベースの作成または移行時を基準にした作成タイム・スタンプがあります。

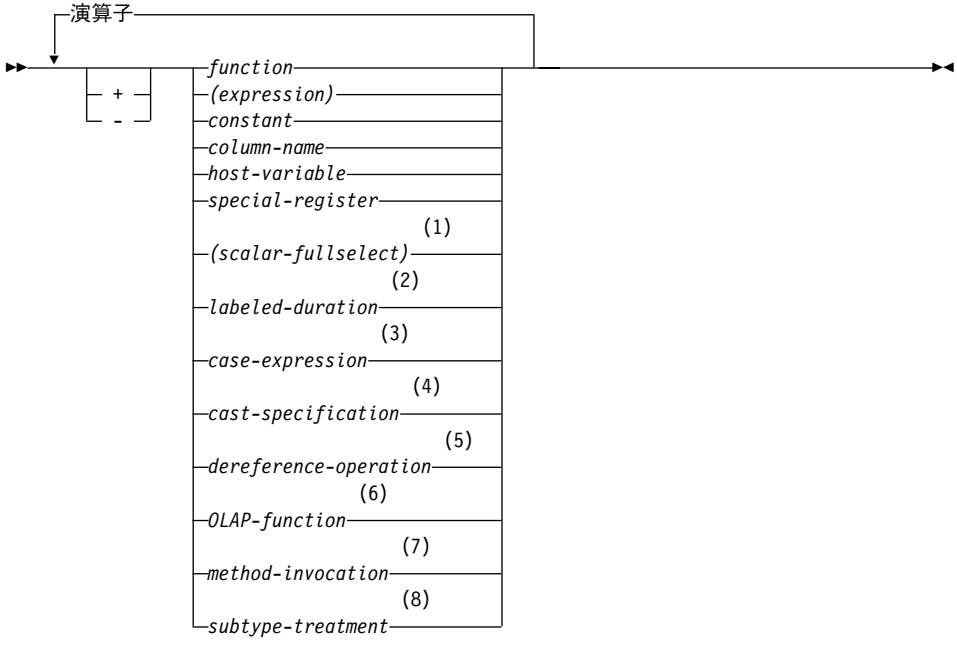
32. 視点の場合、保守的バインドによって、視点が作成時に存在していたのと同じ列で構成されるようにします。たとえば、選択リストでアスタリスクを使用して定義された視点は、それが作成された後に基礎表に追加された列については考慮しません。

したがって、保守的バインド・セマンティクスが適用される場合、後から作成された関数、メソッド、およびデータ・タイプは考慮されません。

パッケージ内の静的 DML の場合、パッケージの再バインドは暗黙的に行われるか、 REBIND コマンド (またはこれに対応する API) または BIND コマンド (またはこれに対応する API) を明示的に発行することによって行われます。暗黙的な再バインドでは、常に保守的バインド・セマンティクスを適用して、関数、メソッド、およびデータ・タイプの解決が実行されます。 REBIND コマンドを使用する場合は、保守的バインド・セマンティクスを適用して解決するか (RESOLVE CONSERVATIVE オプション)、それとも新しい関数、メソッド、およびデータ・タイプを考慮して解決するか (デフォルトまたは RESOLVE ANY オプションを使用) を選択できます。

式

式は値を指定します。これは、定数や列名だけで構成される簡単な値にすることもでき、さらに複雑な値にすることも可能です。同じような複雑な式を繰り返し使用しているのであれば、 SQL 関数を使用して共通の式をカプセル化することを考慮できます。 701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』を参照してください。



演算子:

注:

1 184ページの『スカラー全選択』を参照。

2 184ページの『ラベル付き期間』を参照。

3 191ページの『CASE 式』を参照。

4 193ページの『CAST 指定』を参照。

5 197ページの『参照解除操作』を参照。

6 198ページの『OLAP 関数』を参照。

7 204ページの『メソッドの呼び出し』を参照。

8 206ページの『サブタイプの扱い』を参照。

9 CONCAT の同義語として || を使用できます。

演算子を使用しない式

演算子を使用しない式では、指定した値が式の結果になります。

例: `SALARY` `:SALARY` `'SALARY'` `MAX(SALARY)`

連結演算子を使用する式

連結演算子 (CONCAT) は、2 つのストリング・オペランドを連結して、 1 つのストリング式 にします。

連結するオペランドは、互換性のあるストリングでなければなりません。 FOR BIT DATA として定義されている文字ストリングも含め、 2 進ストリングを文字ストリングと連結することはできません (SQLSTATE 42884)。互換性の詳細については、 106ページの表7 の互換性マトリックスを参照してください。

いずれかのオペランドがヌル値になる可能性がある場合は、結果もヌル値になる可能性があり、いずれかがヌル値なら結果はヌル値になります。そうでない場合、結果は第 1 オペランドの後に第 2 オペランドが続いた形式となります。連結時に混合データが不正に形成されても、それに対する検査は行われません。

結果ストリングの長さは、オペランドの長さの合計になります。

結果のデータ・タイプと長さ属性は、以下の表に示すように、オペランドのデータ・タイプと長さ属性によって決まります。

表 10. 連結するオペランドのデータ・タイプと長さ

オペランド	連結後の長さ属性	結果
CHAR(A) CHAR(B)	<255	CHAR(A+B)
CHAR(A) CHAR(B)	>254	VARCHAR(A+B)
CHAR(A) VARCHAR(B)	<4001	VARCHAR(A+B)
CHAR(A) VARCHAR(B)	>4000	LONG VARCHAR
CHAR(A) LONG VARCHAR	-	LONG VARCHAR
VARCHAR(A) VARCHAR(B)	<4001	VARCHAR(A+B)
VARCHAR(A) VARCHAR(B)	>4000	LONG VARCHAR
VARCHAR(A) LONG VARCHAR	-	LONG VARCHAR
LONG VARCHAR LONG VARCHAR	-	LONG VARCHAR
CLOB(A) CHAR(B)	-	CLOB(MIN(A+B, 2G))
CLOB(A) VARCHAR(B)	-	CLOB(MIN(A+B, 2G))
CLOB(A) LONG VARCHAR	-	CLOB(MIN(A+32K, 2G))
CLOB(A) CLOB(B)	-	CLOB(MIN(A+B, 2G))
GRAPHIC(A) GRAPHIC(B)	<128	GRAPHIC(A+B)
GRAPHIC(A) GRAPHIC(B)	>127	VARGRAPHIC(A+B)
GRAPHIC(A) VARGRAPHIC(B)	<2001	VARGRAPHIC(A+B)
GRAPHIC(A) VARGRAPHIC(B)	>2000	LONG VARGRAPHIC
GRAPHIC(A) LONG VARGRAPHIC	-	LONG VARGRAPHIC
VARGRAPHIC(A) VARGRAPHIC(B)	<2001	VARGRAPHIC(A+B)
VARGRAPHIC(A) VARGRAPHIC(B)	>2000	LONG VARGRAPHIC
VARGRAPHIC(A) LONG VARGRAPHIC	-	LONG VARGRAPHIC

表 10. 連結するオペランドのデータ・タイプと長さ (続き)

オペランド	連結後の長さ属性	結果
LONG VARGRAPHIC LONG VARGRAPHIC	-	LONG VARGRAPHIC
DBCLOB(A) GRAPHIC(B)	-	DBCLOB(MIN(A+B, 1G))
DBCLOB(A) VARGRAPHIC(B)	-	DBCLOB(MIN(A+B, 1G))
DBCLOB(A) LONG VARGRAPHIC	-	DBCLOB(MIN(A+16K, 1G))
DBCLOB(A) DBCLOB(B)	-	DBCLOB(MIN(A+B, 1G))
BLOB(A) BLOB(B)	-	BLOB(MIN(A+B, 2G))

直前のバージョンとの互換性を保つために、 LONG データ・タイプを含む結果を LOB データ・タイプに自動調整しないことに注意してください。たとえば、CHAR(200) の値と、完全に文字の詰まった LONG VARCHAR の値とを連結した場合、 CLOB データ・タイプへプロモートされるのではなくエラーになります。

結果のコード・ページは派生コード・ページとみなされ、 126ページの『ストリング変換に関する規則』で説明されているようにして、そのオペランドのコード・ページによって決定されます。

一方のオペランドはパラメーター・マーカーにすることができます。パラメーター・マーカーが使用されている場合、そのオペランドのデータ・タイプと長さ属性は、パラメーター・マーカーでないオペランドと同じであるとみなされます。ネストした連結の場合、これらの属性を決定できるように演算の順序を考慮する必要があります。

例 1: FIRSTNME が Pierre で LASTNAME が Fermat である場合、以下のようになります。

```
FIRSTNME CONCAT ' ' CONCAT LASTNAME
```

`Pierre Fermat` の値が戻されます。

例 2: 以下を条件とします。

- `COLA` は、 `'AA'` の値を持つ VARCHAR(5) として定義されている。
- `:host_var` は、長さが 5 で値が `'BB  '` である文字ホスト変数として定義されている。

- COLC は、値が 'CC' の CHAR(5) として定義されている。
- COLD は、値が 'DDDDD' の CHAR(5) として定義されている。

この場合、 COLA **CONCAT** :host_var **CONCAT** COLC **CONCAT** COLD の値は次のとおりです。

```
'AABB   CC   DDDDD'
```

データ・タイプが VARCHAR で、長さ属性は 17、結果コード・ページはデータベース・コード・ページとなります。

例 3: 以下を条件とします。
COLA は、CHAR(10) として定義されている。
COLB は、VARCHAR(5) として定義されている。

次の式の中のパラメーター・マーカーは、

```
COLA CONCAT COLB CONCAT ?
```

VARCHAR(15) とみなされます。これは、COLA CONCAT COLB が最初に評価され、その結果が 2 番目の CONCAT 演算の第 1 オペランドとなるためです。

ユーザー定義タイプ

ユーザー定義タイプは、ストリング・タイプのソース・データ・タイプがある特殊タイプであっても、連結演算子は使用できません。 連結するためには、そのソースとしての CONCAT 演算子を使った関数を作成する必要があります。たとえば、TITLE と TITLE_DESCRIPTION という特殊タイプがあり、どちらも VARCHAR(25) データ・タイプである場合は、以下に示すユーザー定義関数 ATTACH でそれらを連結することができます。

```
CREATE FUNCTION ATTACH (TITLE, TITLE_DESCRIPTION)
  RETURNS VARCHAR(50) SOURCE CONCAT (VARCHAR(), VARCHAR())
```

別の方法として、新規のデータ・タイプを追加するユーザー定義関数を使用し、連結演算子を多重定義することもできます。

```
CREATE FUNCTION CONCAT (TITLE, TITLE_DESCRIPTION)
  RETURNS VARCHAR(50) SOURCE CONCAT (VARCHAR(), VARCHAR())
```

算術演算子を使用する式

算術演算子が使用されている場合、式の結果は、演算子をオペランドの値に適用して導かれた値となります。

いずれかのオペランドがヌル値になる可能性がある場合、またはデータベースが DFT_SQLMATHWARN を yes に設定して構成されている場合、結果もヌル値になる可能性があります。

どちらか一方のオペランドがヌル値ならば、式の結果はヌル値になります。

算術演算子は、符号付き数値タイプと日時タイプに適用できます (185ページの『SQL における日付 / 時刻の算術演算』を参照)。たとえば、USER+2 は無効です。ソース関数については、符号付き数値タイプであるソース・タイプを持つ特殊タイプ上の算術演算子に定義できます。

接頭演算子、+ (単項加算) はそのオペランドを変更しません。接頭演算子、- (単項減算) は、ゼロ以外のオペランドの符号を逆にします。 A のデータ・タイプが短精度整数である場合、 -A のデータ・タイプは長精度整数になります。接頭演算子の後に続くトークンの先頭の文字は、正または負の符号であってはなりません。

挿入演算子 +、-、*、および / はそれぞれ、加算、減算、乗算、および除算を指定します。除算の第 2 オペランドの値はゼロにすることはできません。これらの演算子は関数としても扱われます。したがって、式 *"+"(a,b)* は、式 *a+b* の『演算子』の機能と同じ意味になります。

算術演算エラー

ゼロによる除算や数値の桁あふれなどの算術演算エラーが、式の処理の過程で生じると、エラーが戻され、その式を処理する SQL ステートメントは失敗し、エラー (SQLSTATE 22003 または 22012) が出されます。

データベースは、算術演算エラーが生じた場合に式の値としてヌル値を戻すように構成することが可能で (DFT_SQLMATHWARN を yes に設定して)、警告 (SQLSTATE 01519 または 01564) を出して、その SQL ステートメントの処理を続けることができます。算術計算エラーがヌル値として扱われる場合、SQL ステートメントの結果に影響があります。以下は、このような影響の例を示しています。

- 列関数の引き数の式で算術演算エラーが起きると、その列関数の結果を判別する際に行が無視されます。算術演算エラーが桁あふれである場合、結果の値に大きな影響を与える場合があります。
- WHERE 文節の述部の式で算術演算エラーが起きると、結果に行が含まれない場合があります。
- 検査制約の述部の式で算術演算エラーが起きても、制約には誤りがないため更新または挿入は続行されます。

このようなタイプの影響が受け入れられない場合、算術演算エラーを処理するのに必要な他の処置を行って、受け入れ可能な結果を生成する必要があります。たとえば次のような処置です。

- ゼロによる除算の有無を検査するために CASE 式を追加して、このような状態に対応する必要な値を設定する。
- ヌル値を処理する述部を追加する (ヌル値が不能な列の検査制約は次のようになります。

  ```
  check (c1*c2 is not null and c1*c2>5000)
  ```

  これにより、桁あふれの制約に違反する場合があります)。

2 つの整数オペランド

算術演算子のオペランドが両方とも整数の場合、その演算は 2 進数で実行され、いずれかの (または両方の) オペランドが大整数 (big integer) でない限り、その結果は*長精度整数 (large integer)* になります。いずれかの (または両方の) オペランドが大整数である場合は、結果は大整数になります。除算の剰余は失われます。整数算術演算 (単項減算符号を含む) の結果は、結果タイプの範囲内でなければなりません。

整数と 10 進数オペランド

一方のオペランドが整数で、もう一方のオペランドが 10 進数の場合、その演算は、精度 *p* および位取り 0 の 10 進数に変換されたその整数の一時コピーを使用して、 10 進数で行われます。 *p* は、大整数 (big integer) の場合 19 であり、長精度整数 (large integer) の場合 11 であり、短精度整数 (small integer) の場合 5 です。

2 つの 10 進数オペランド

オペランドが両方とも 10 進数の場合、その演算は 10 進数で行われます。 10 進数の算術演算の結果は 10 進数であり、その結果の精度と位取りは、演算の種類およびオペランドの精度と位取りによって異なります。 演算が加算または減算で、オペランドの位取りが同じでない場合は、オペランドの一方の一時コピーを使用して演算が行われます。短い方のオペランドの小数部分が、長い方のオペランドと同じ桁数になるように、短い方のオペランドのコピーに後続ゼロを加えて拡張されます。

10 進数演算の結果は、精度が 31 以下でなければなりません。 10 進数の加算、減算、および乗算の結果は、精度が 31 を超える一時結果から求められることがあります。一時結果の精度が 31 を超えない場合、最終結果は一時結果と同じです。

SQL での 10 進数演算

以下の公式により、SQL における 10 進数演算の結果の精度および位取りが決まります。記号 *p* と *s* は第 1 オペランドの精度と位取りを表し、記号 *p'* と *s'* は第 2 オペランドの精度と位取りを表します。

加算および減算

精度は min(31,max(*p*-*s*,*p'*-*s'*) +max(*s*,*s'*)+1) になります。位取りは max(*s*,*s'*) です。

乗算

乗算結果の精度は (31,*p*+*p'*)、位取りは min(31,*s*+*s'*) です。

除算

除算結果の精度は 31 です。位取りは *31-p+s-s'* です。位取りは負であってはなりません。

浮動小数点オペランド

算術演算子のいずれかのオペランドが浮動小数点の場合、演算は浮動小数点で行われ、必要に応じてオペランドが最初に倍精度の浮動小数点数に変換されます。したがって、式の要素のいずれかが浮動小数点数の場合、その式の結果は倍精度浮動小数点数になります。

浮動小数点数と整数を含む演算は、整数を倍精度浮動小数点に変換したものの一時コピーを使って実行されます。浮動小数点数と 10 進数を含む演算は、 10 進数を倍精度浮動小数点に変換したものの一時コピーを使って実行されます。浮動小数点数演算の結果は、浮動小数点数の範囲内でなければなりません。

オペランドとしてのユーザー定義タイプ

ユーザー定義タイプは、そのソース・データ・タイプが数値であっても算術演算子には使用できません。 算術演算を実行するには、そのソースとしての算術演算子を使用する関数を作成する必要があります。たとえば、INCOME と EXPENSES という特殊タイプがあり、どちらも DECIMAL(8,2) データ・タイプである場合は、以下に示すユーザー定義関数 REVENUE を使って一方からもう一方を減算することができます。

```
CREATE FUNCTION REVENUE (INCOME, EXPENSES)
  RETURNS DECIMAL(8,2) SOURCE "-" (DECIMAL, DECIMAL)
```

別の方法として、新規のデータ・タイプを減算するユーザー定義関数を使って - (マイナス) 演算子を多重定義することも可能です。

```
CREATE FUNCTION "-" (INCOME, EXPENSES)
  RETURNS DECIMAL(8,2) SOURCE "-" (DECIMAL, DECIMAL)
```

スカラー全選択

式でサポートされる *scalar fullselect (スカラー全選択)* は、括弧で囲まれる全選択であり、1 つの列値で構成される 1 つの行を戻します。全選択が行を戻さない場合、式の結果はヌル値になります。選択リスト要素が単なる列名か別の演算の式である場合、その列の名前に基づいて結果列の名前が付けられます。詳細については、461ページの『全選択』を参照してください。

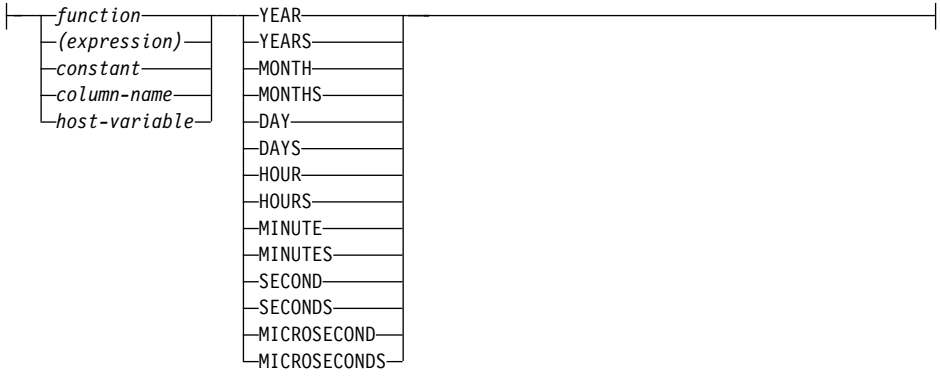
日付 / 時刻演算と期間

日付 / 時刻の値は、増分、減分、および減算を行うことができます。このような演算には、*期間* と呼ばれる 10 進数を伴う場合があります。期間の定義と、日付 / 時刻の算術演算に関する規則の仕様について以下に説明します。

期間とは、時間の間隔を表す数値です。期間には以下の 4 つのタイプがあります。

ラベル付き期間

labeled-duration:



ラベル付き期間 (labeled-duration) は、特定の時間単位を表すもので、数値 (式の結果でも可) の後に 7 つの期間キーワード YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、または MICROSECONDS[33] のうちの 1 つを付けたものです。指定した値は、DECIMAL(15,0) の数値へ割り当てられる場合と同様に変換されます。ラベル付き期間は、算術演算子の 1 つのオペランドとしてのみ使用でき、このときの他方のオペランドは DATE、TIME、または TIMESTAMP です。したがって、式 HIREDATE + 2 MONTHS + 14 DAYS

33. これらのキーワードの単数形 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、および MICROSECOND も可能です。

は有効ですが、式 HIREDATE + (2 MONTHS + 14 DAYS) は有効ではありません。どちらの式でも 2 MONTHS と 14 DAYS がラベル付き期間です。

日付期間

日付期間 は、DECIMAL(8,0) の数値として表現される年数、月数、および日数を表します。 適切に解釈されるために、この数値は *yyyymmdd.* という形式にする必要があります (*yyyy* は年数、 *mm* は月数、*dd* は日数を表します)。[34] 式 HIREDATE - BRTHDATE のように、ある日付値から別の日付値を減算した結果が日付期間です。

時刻期間

時刻期間 は、DECIMAL(6,0) の数値として表現される時間数、分数、および秒数を表します。適切に解釈されるために、この数値は *hhmmss.* という形式にする必要があります (*hh* は時間数、 *mm* は分数、*ss* は秒数を表します)。[34] ある時刻値から別の時刻値を減算した結果が時刻期間です。

タイム・スタンプ期間

タイム・スタンプ期間 は、DECIMAL(20,6) の数値として表現され、年数、月数、日数、時間数、分数、秒数、およびマイクロ秒数を表します。適切に解釈されるために、この数値は *yyyymmddhhmmss.zzzzzz* という形式にする必要があります (*yyyy*、 *mm*、*dd*、 *hh*、*mm*、 *ss*、および *zzzzzz* はそれぞれ、年数、月数、日数、時間数、分数、秒数、およびマイクロ秒数を表します)。あるタイム・スタンプ値から別のタイム・スタンプ値を減算した結果が、タイム・スタンプ期間です。

SQL における日付 / 時刻の算術演算

日付 / 時刻値に関して実行できる算術演算は加算と減算だけです。日付 / 時刻値が加算のオペランドである場合、他方のオペランドは期間でなければなりません。日付 / 時刻の値を使う加算演算子を使用するときには、次のような特有の規則があります。

- 一方のオペランドが日付である場合、もう一方のオペランドは日付期間、または YEARS、MONTHS、DAYS のラベル付き期間であることが必要です。
- 一方のオペランドが時刻である場合、もう一方のオペランドは時刻期間、または HOURS、MINUTES、SECONDS のラベル付き期間であることが必要です。
- 一方のオペランドがタイム・スタンプである場合、もう一方のオペランドは期間でなければなりません。この場合、期間のどのタイプでも有効です。

34. この形式の期間は、DECIMAL データ・タイプを示します。

- 加算演算子のどちらのオペランドにも、パラメーター・マーカーは使用できません。

日付 / 時刻の値に減算演算子を使用する際の規則は、加算演算子の場合とは異なります。これは、日付 / 時刻の値を期間から引くことができないため、さらに 2 つの日付 / 時刻の値を差し引くことと期間を日付 / 時刻の値から差し引くこととは異なるためです。日付 / 時刻の値を使う減算演算子を使用するときには、次のような特有の規則があります。

- 第 1 オペランドが日付の場合、第 2 オペランドは日付、日付期間、日付のストリング表記、または YEARS、MONTHS、DAYS のラベル付き期間であることが必要です。
- 第 2 オペランドが日付の場合、第 1 オペランドは、日付または日付のストリング表記であることが必要です。
- 第 1 オペランドが時刻の場合、第 2 オペランドは、時刻、時刻期間、時刻のストリング表記、または HOURS、MINUTES、SECONDS のラベル付き期間であることが必要です。
- 第 2 オペランドが時刻の場合、第 1 オペランドは、時刻または時刻のストリング表記であることが必要です。
- 第 1 オペランドがタイム・スタンプの場合、第 2 オペランドは、タイム・スタンプまたはタイム・スタンプのストリング表記、または期間であることが必要です。
- 第 2 オペランドがタイム・スタンプの場合、第 1 オペランドは、タイム・スタンプまたはタイム・スタンプのストリング表記であることが必要です。
- 減算演算子のどちらのオペランドにも、パラメーター・マーカーは使用できません。

日付の算術演算

日付は、減算、増分、および減分を行うことができます。

日付の減算: ある日付 (DATE2) を別の日付 (DATE1) から減算した結果は、これら 2 つの日付の間の年数、月数、日数を示す日付期間です。結果のデータ・タイプは DECIMAL(8,0) です。 DATE1 が DATE2 以上の場合、DATE1 から DATE2 が減算されます。これに対し、DATE1 が DATE2 より小さい場合は、 DATE2 から DATE1 が減算され、結果の符号が負になります。演算 RESULT = DATE1 - DATE2 の実行ステップを、以下に順に示します。

```
If DAY(DATE2) <= DAY(DATE1)
     then DAY(RESULT) = DAY(DATE1) - DAY(DATE2).
```

```
If DAY(DATE2) > DAY(DATE1)
     then DAY(RESULT) = N + DAY(DATE1) - DAY(DATE2)
        where N = the last day of MONTH(DATE2).
           MONTH(DATE2) is then incremented by 1.

If MONTH(DATE2) <= MONTH(DATE1)
     then MONTH(RESULT) = MONTH(DATE1) - MONTH(DATE2).

If MONTH(DATE2) > MONTH(DATE1)
     then MONTH(RESULT) = 12 + MONTH(DATE1) - MONTH(DATE2).
        YEAR(DATE2) is then incremented by 1.

YEAR(RESULT) = YEAR(DATE1) - YEAR(DATE2).
```

たとえば、DATE('3/15/2000') - '12/31/1999' の結果は 00000215 になります。(すなわち、0 年 2 か月 15 日の期間です。)

日付の増分と減分: 日付に期間を加算したり、日付から期間を減算したりすると、結果自体は日付となります。 (この演算では、月はカレンダーのページに相当します。つまり、日付に月を加算することは、その日付のページから順にカレンダーをめくっていくようなものです。) 結果は、0001 年 1 月 1 日以後 9999 年 12 月 31 日以前の日付となる必要があります。

年の期間を加算または減算する場合、影響を受けるのは日付の年の部分だけです。月も日も変更されませんが、その結果がうるう年でない年の 2 月 29 日となった場合は別です。その場合は日が 28 に変更され、 SQLCA の警告標識が日付調整の発生を示すように設定されます。

同様に、月の期間を加算または減算する場合、影響を受けるのは月の部分だけです。ただし、必要に応じて年の部分にも影響が及びます。日付の日の部分は変更されませんが、結果が無効な場合 (たとえば 9 月 31 日など) は別です。その場合は日がその月の最後の日に設定され、 SQLCA の警告標識が日付調整の発生を示すように設定されます。

日の期間を加算または減算すると、日付の中の日の部分は当然影響を受けますが、月および年も影響を受ける可能性があります。

日付期間も、正負にかかわらず、日付に対して加減算が行えます。ラベル付き期間の場合と同じように、結果は有効な日付となり、月末の調整が必要になれば SQLCA の警告標識が設定されます。

正の日付期間が日付に加算されるとき、または負の日付期間が日付から減算されるときは、日付は、指定した年数、月数、日数の順で増分されます。したがって、X が正の DECIMAL(8,0) の数値であるとき、 DATE1 + X は以下の式と同等です。

```
DATE1 + YEAR(X) YEARS + MONTH(X) MONTHS + DAY(X) DAYS.
```

正の日付期間を日付から減算するとき、または負の日付期間を日付に加算するとき、日付は、指定した日数、月数、年数の順で減分されます。したがって、X が正の DECIMAL(8,0) の数値であるとき、 DATE1 - X は以下の式と同等です。

```
DATE1 - DAY(X) DAYS - MONTH(X) MONTHS - YEAR(X) YEARS.
```

期間を日付に加算するとき、特定の日付に 1 か月を加算すると、1 か月後の同じ日付になります。ただし、1 か月後にその日付が存在しない場合は扱いが異なります。その場合、日付は 1 か月後の最後の日に設定されます。たとえば、1 月 28 日に 1 か月を加えると 2 月 28 日になります。 1 月 29、30、または 31 日に 1 か月を加えると通常の年では 2 月 28 日、うるう年では 2 月 29 日になります。

注: 特定の日付に 1 か月以上の月数を加算し、その結果から同じ月数を減算した場合、最終的な日付が元の日付と同じになるとは限りません。

時刻の算術演算

時刻は、減算、増分、または減分を行うことができます。

時刻の減算: ある時刻 (TIME2) を別の時刻 (TIME1) から減算した結果は、それら 2 つの時刻の間の時間数、分数、秒数を示す時刻期間です。結果のデータ・タイプは DECIMAL(6,0) です。

TIME1 が TIME2 と同じかまたはそれより大きい場合、 TIME1 から TIME2 が引かれます。

これに対し、TIME1 が TIME2 より小さい場合は、TIME2 から TIME1 が減算され、結果の符号が負になります。演算 RESULT = TIME1 - TIME2 の実行ステップを、以下に順に示します。

```
If SECOND(TIME2) <= SECOND(TIME1)
     then SECOND(RESULT) = SECOND(TIME1) - SECOND(TIME2).

If SECOND(TIME2) > SECOND(TIME1)
     then SECOND(RESULT) = 60 + SECOND(TIME1) - SECOND(TIME2).
        MINUTE(TIME2) is then incremented by 1.

If MINUTE(TIME2) <= MINUTE(TIME1)
     then MINUTE(RESULT) = MINUTE(TIME1) - MINUTE(TIME2).

If MINUTE(TIME1) > MINUTE(TIME1)
     then MINUTE(RESULT) = 60 + MINUTE(TIME1) - MINUTE(TIME2).
        HOUR(TIME2) is then incremented by 1.

HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).
```

たとえば、TIME('11:02:26') - '00:32:56' の結果は 102930 になります。 (10 時間 29 分、30 秒の期間です。)

時刻の増分と減分: 時刻に期間を加算したり、時刻から期間を減算したりすると、結果自体は時刻となります。時間数の桁あふれや下位桁あふれは捨てられ、これにより常に結果が時刻となります。時間数で指定する期間を加算または減算する場合、影響を受けるのは時間数の部分だけです。分数と秒数は変更されません。

同様に、分数で指定する期間を加算または減算する場合、影響を受けるのは分の部分だけです。ただし、必要に応じて時間数の部分にも影響が及びます。時刻の秒の部分は変更されません。

秒の期間を加算または減算すると、時刻の中の秒の部分は当然影響を受けますが、分および時も影響を受ける可能性があります。

時刻期間も、正負にかかわらず、時刻との加減算を行えます。結果は、指定した時間数、分数、秒数の順に増分または減分された時刻となります。TIME1 + X ("X" は DECIMAL(6,0)) は次の式と同等です。

```
TIME1 + HOUR(X) HOURS + MINUTE(X) MINUTES + SECOND(X) SECONDS
```

注: 時刻 '24:00:00' は有効な値として受け付けられますが、時刻の加減算の結果として戻されることはありません。これは、期間オペランドがゼロであっても同じです (たとえば、 time('24:00:00')±0 秒 ='00:00:00' となります)。

タイム・スタンプの算術演算

タイム・スタンプは、減算、増分、または減分を行うことができます。

タイム・スタンプの減算: あるタイム・スタンプ (TS2) を別のタイム・スタンプ (TS1) から減算した結果は、それら 2 つのタイム・スタンプの間の年数、月数、日数、時間数、分数、秒数、およびマイクロ秒数を示すタイム・スタンプ期間です。結果のデータ・タイプは DECIMAL(20,6) です。

TS1 が TS2 以上の場合、TS1 から TS2 が減算されます。これに対し、TS1 が TS2 より小さい場合は、TS2 から TS1 が減算され、結果の符号が負になります。演算 RESULT = TS1 - TS2 の実行ステップを、以下に順に示します。

```
If MICROSECOND(TS2) <= MICROSECOND(TS1)
     then MICROSECOND(RESULT) = MICROSECOND(TS1) -
        MICROSECOND(TS2).
```

```
If MICROSECOND(TS2) > MICROSECOND(TS1)
     then MICROSECOND(RESULT) = 1000000 +
        MICROSECOND(TS1) - MICROSECOND(TS2)
           and SECOND(TS2) is incremented by 1.
```

タイム・スタンプの秒および分の部分は、時刻の減算規則で指定されたように減算されます。

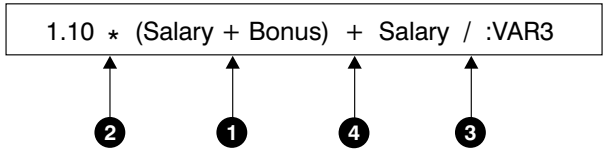
```
If HOUR(TS2) <= HOUR(TS1)
     then HOUR(RESULT) = HOUR(TS1) - HOUR(TS2).

If HOUR(TS2) > HOUR(TS1)
     then HOUR(RESULT) = 24 + HOUR(TS1) - HOUR(TS2)
        and DAY(TS2) is incremented by 1.
```

タイム・スタンプの日付の部分は、日付の減算規則での説明と同じようにして減算されます。

タイム・スタンプの増分と減分: タイム・スタンプに期間を加算したり、タイム・スタンプから期間を減算したりすると、結果自体はタイム・スタンプとなります。日付と時刻の算術演算はすでに説明したとおりに実行されますが、時間数の桁あふれと下位桁あふれは結果の日付の部分に繰り上げまたは繰り下げられ、有効な日付の範囲内に収められます。マイクロ秒の桁あふれは秒に繰り上げられます。

演算の優先順位

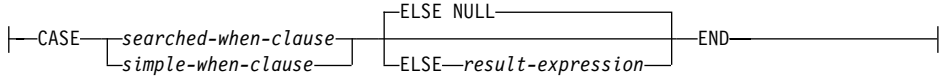
括弧の中の式および参照解除操作が、最初に左から右へと評価されます。[35] 評価の順序が括弧で指定されていない場合は、まず接頭演算子が乗算および除算に先立って行われ、次に乗算と除算が加算および減算に先立って行われます。同じ優先順位の演算子は左から右に行われます。



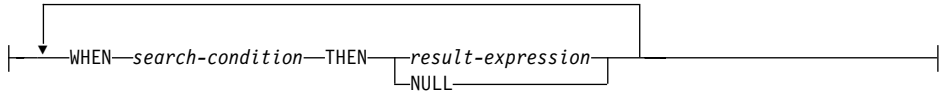
35. 括弧は、副選択ステートメントや、探索条件、関数でも使用される点に注意してください。ただし、SQL ステートメント内でセクションを任意にグループ分けするのに使用することはできません。

CASE 式

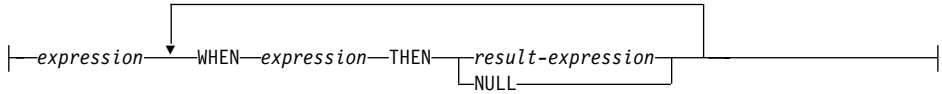
case-expression:



searched-when-clause:



simple-when-clause:



CASE 式は、1 つ以上の条件の評価に基づいて式を選択するためのものです。一般に、CASE 式の値は、評価が「真」である最初の (左端の) ケースの後に来る *result-expression* (結果式) の値になります。評価が「真」であるケースがなく、ELSE キーワードが指定されている場合、結果は ELSE の *result-expression* (結果式) または NULL になります。評価が「真」であるケースがなく、ELSE キーワードが指定されていない場合、結果は NULL になります。あるケースの評価が「不定」の場合 (NULL のため)、そのケースは「真」ではなく、したがって評価が「偽」であるケースと同じように扱われます。

CASE 式が VALUES 文節、IN 述部、GROUP BY 文節、または ORDER BY 文節中にある場合、 searched-when-clause の *search-condition* は、比較述部、全選択を使用する IN 述部、または EXISTS 述部 (SQLSTATE 42625) にすることはできません。

simple-when-clause (単純 WHEN 文節) を使用する場合は、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 *expression* の値と等しいかどうかが検査されます。このため、最初の *WHEN* キーワードの前の *expression* は、 WHEN キーワードの後に来るそれぞれの *expression* のデータ・タイプと互換である必要があります。 *simple-when-clause* の中の最初の *WHEN* キーワードの前にある *expression* に、可変の関数または外部処理を伴う関数を含めることはできません (SQLSTATE 42845)。

result-expression (結果式) は、 THEN または ELSE キーワードの後に指定する式 です。 CASE 式では、少なくとも 1 つの結果式 を指定する必要があります (すべてのケースに NULL を指定することはできません) (SQLSTATE 42625)。すべての結果式 のデータ・タイプは互換でなければならず (SQLSTATE 42804)、結果の属性は 121ページの『結果のデータ・タイプに関する規則』に基づいて決定されます。

例:

- 以下の例では、部門番号の先頭文字が組織内の部を示すものとし、 CASE 式を使用して、各社員が属する部の正式名称を取り出します。

```
SELECT EMPNO, LASTNAME,
       CASE SUBSTR(WORKDEPT,1,1)
       WHEN 'A' THEN 'Administration'
       WHEN 'B' THEN 'Human Resources'
       WHEN 'C' THEN 'Accounting'
       WHEN 'D' THEN 'Design'
       WHEN 'E' THEN 'Operations'
       END
FROM EMPLOYEE;
```

- 研修年数は、研修レベルを示す目的で EMPLOYEE 表で使用されています。 CASE 式を使用して、これらを分類し、研修レベルを示します。

```
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
       CASE
       WHEN EDLEVEL < 15 THEN 'SECONDARY'
       WHEN EDLEVEL < 19 THEN 'COLLEGE'
       ELSE 'POST GRADUATE'
       END
FROM EMPLOYEE
```

- CASE ステートメントの別の有効な使い方として、ゼロ除算によるエラーを防止することができます。たとえば以下のコードは、収入のすべてではないが 25% より多くを歩合で得ている社員を検索しています。

```
SELECT EMPNO, WORKDEPT, SALARY+COMM FROM EMPLOYEE
WHERE (CASE WHEN SALARY=0 THEN NULL
            ELSE COMM/SALARY
            END) > 0.25;
```

- 以下の 2 つの CASE 式は同じものです。

```
SELECT LASTNAME,
 CASE
 WHEN LASTNAME = 'Haas' THEN 'President'
 ...

SELECT LASTNAME,
 CASE LASTNAME
 WHEN 'Haas' THEN 'President'
 ...
```

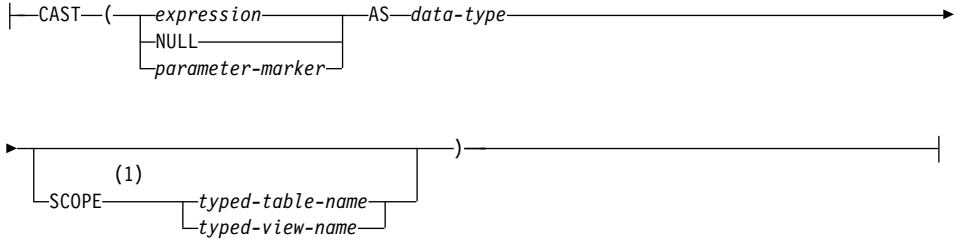
CASE の機能の一部を処理する目的で、スカラー関数の NULLIF と COALESCE が特別に用意されています。 表11 に、 CASE を使用した場合とそれらの関数を使用した場合とで同等の式を示します。

表 11. 同等の CASE 式

式	同等の式
CASE WHEN e1=e2 THEN NULL ELSE e1 END	NULLIF(e1,e2)
CASE WHEN e1 IS NOT NULL THEN e1 ELSE e2 END	COALESCE(e1,e2)
CASE WHEN e1 IS NOT NULL THEN e1 ELSE COALESCE(e2,...,eN) END	COALESCE(e1,e2,...,eN)

CAST 指定

cast-specification:



注:

1 SCOPE 文節が適用されるのは、REF データ・タイプだけです。

CAST 指定は、データ・タイプ によって指定されたタイプにキャストされたキャスト・オペランド (第 1 オペランド) を戻します。

expression

キャスト・オペランドが式 (パラメーター・マーカーまたは NULL ではなく) である場合、結果は、指定された目的データ・タイプ に変換された引き数値です。

サポートされるキャストについては 105ページの表6 に記載されています。この表では、第 1 列がキャスト・オペランドのデータ・タイプ (ソース・データ・タイプ) を表し、上部に横方向に示したデータ・タイプが CAST 指定の目的データ・タイプを表します。サポートされていないキャストを実行すると、エラー (SQLSTATE 42846) になります。

文字ストリング (CLOB 以外) を長さの異なる文字ストリングにキャストするとき、後続ブランク以外の文字が切り捨てられると、警告

(SQLSTATE 01004) が戻されます。漢字ストリング (DBCLOB 以外) を長さの異なる漢字ストリングにキャストするとき、後続ブランク以外の文字が切り捨てられると、警告 (SQLSTATE 01004) が戻されます。キャスト・オペランドが BLOB、CLOB、および DBCLOB の場合、何らかの文字が切り捨てられると警告が発行されます。

NULL
キャスト・オペランドがキーワード NULL である場合、結果は、指定されたデータ・タイプ のヌル値です。

parameter-marker
パラメーター・マーカー (疑問符で指定されるもの) は通常は式としてみなされますが、ここでは特別な意味をもつため別個に説明します。キャスト・オペランドがパラメーター・マーカー である場合、指定されたデータ・タイプ は、指定されたデータ・タイプに置き換えが割り当て可能である (ストリングの記憶割り当てを使用して) ことを示す合意であるとみなされます。 このようなパラメーター・マーカーは、*タイプ付きパラメーター・マーカー* とみなされます。タイプ付きパラメーター・マーカーは、関数解決、選択リストの DESCRIBE、または列割り当てを行う目的で、他のタイプ付き値と同じように扱われます。

data type
既存のデータ・タイプの名前。このタイプ名が修飾されていない場合は、SQL パスを使用してデータ・タイプが解決されます。長さ、精度、および位取りなどの関連する属性を伴うデータ・タイプには、*データ・タイプ* の指定時にこのような属性を組み込む必要があります (指定されていない場合、 CHAR は長さ 1 にデフォルト解釈され、 DECIMAL は精度 5 および位取り 0 にデフォルト解釈されます)。サポートされるデータ・タイプに関する制限は、指定したキャスト・オペランドに基づいて適用されます。

- キャスト・オペランドが式 の場合、キャスト・オペランドのデータ・タイプ (ソース・データ・タイプ) に応じてサポートされる目的データ・タイプについては、 102ページの『データ・タイプ間のキャスト』を参照してください。
- キャスト・オペランドがキーワード NULL の場合、既存のどのデータ・タイプでも指定できます。
- キャスト・オペランドがパラメーター・マーカーの場合、ターゲット・データ・タイプは、既存の任意のデータ・タイプとすることができます。データ・タイプがユーザー定義特殊タイプの場合、パラメーター・マーカーを使用するアプリケーションは、そのユーザー定義特殊タイプのソース・データ・タイプを使用します。データ・タイプがユーザー定

義構造タイプの場合、パラメーター・マーカーを使用するアプリケーションは、そのユーザー定義構造タイプの TO SQL 変形関数の入力パラメーター・タイプを使用します。

SCOPE

データ・タイプが参照タイプの場合、効力範囲は参照のターゲット表またはターゲット視点を識別するように定義することができます。

typed-table-name

タイプ付き表の名前。表名はすでに指定されていなければなりません (SQLSTATE 42704)。キャストは *data-type* REF(*S*) にするものでなければなりません。ここでの *S* は *typed-table-name* (SQLSTATE 428DM) のタイプを表しています。

typed-view-name

タイプ付き視点の名前。その視点は存在しているか、あるいは視点定義の一部としてキャストを含むように作成されている視点と同じ名前でなければなりません。キャストは *data-type* REF(*S*) にするものでなければなりません。ここでの *S* は *typed-view-name* (SQLSTATE 428DM) のタイプを表しています。

数値データを文字にキャストする場合、結果のデータ・タイプは固定長文字ストリングです (283ページの『CHAR』を参照)。文字データを数値にキャストする場合、結果のデータ・タイプは指定した数値のタイプによって異なります。たとえば整数へのキャストの場合、結果のデータ・タイプは長精度整数になります (333ページの『INTEGER』を参照)。

例:

- アプリケーションが、 EMPLOYEE 表の SALARY (decimal(9,2) として定義) の整数部だけを使用するとします。社員番号や SALARY の整数値を含んだ、以下のような照会が考えられます。

```
SELECT EMPNO, CAST(SALARY AS INTEGER) FROM EMPLOYEE
```

- SMALLINT に基づいて定義された T_AGE という名前の特殊タイプがあり、 PERSONNEL 表に AGE 列を作成するために使用されるとします。さらに INTEGER に基づいて定義された R_YEAR という名前の特殊タイプがあり、 PERSONNEL 表に RETIRE_YEAR 列を作成するために使用されるとします。以下のような更新ステートメントが考えられます。

```
UPDATE PERSONNEL SET RETIRE_YEAR =?
                 WHERE AGE = CAST( ? AS T_AGE)
```

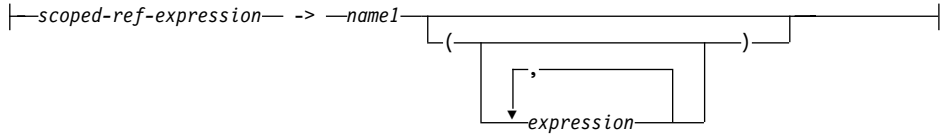
第 1 パラメーターは、データ・タイプ R_YEAR のタイプなしパラメーター・マーカーです。一方、アプリケーションはこのパラメーター・マーカーの整数部を使用します。この場合、これは割り当てなので、明示的な CAST 指定をする必要はありません。

2 番目のパラメーター・マーカーは、特殊タイプ T_AGE としてキャストされるタイプ付きパラメーター・マーカーです。これにより、比較は互換データ・タイプとの間でなければならない、という要件が満たされます。アプリケーションは、ソース・データ・タイプ (SMALLINT) を使用してこのパラメーター・マーカーを処理します。

このステートメントの正常な処理は、関数パスには、 2 つの特殊タイプが定義されているスキーマ (複数の場合あり) のスキーマ名が含まれていることを前提としています。

参照解除操作

dereference-operation:



効力範囲を指定した参照式の効力範囲は、ターゲット 表または視点という表または視点になります。効力範囲を指定した参照式は、*ターゲット行* を識別します。*ターゲット行* は、ターゲット表または視点の (あるいは、副表か副視点のいずれかの) 行です。この行のオブジェクト ID (OID) 列値は、参照式と一致しています。 OID の詳細は、771ページの『CREATE TABLE』を参照してください。参照解除操作を使い、ターゲット行の列にアクセスしたり、そのターゲット行をメソッドのサブジェクトとして使用してメソッドを呼び出すことができます。参照解除操作の結果は、常にヌルになり得ます。参照解除操作は、他のすべての操作よりも優先されます。

scoped-ref-expression

効力範囲を持っている参照タイプである式 (SQLSTATE 428DT)。式がホスト変数、パラメーター・マーカー、またはほかの効力範囲がない参照タイプ値の場合、 SCOPE 文節での CAST 指定で効力範囲の参照を指定する必要があります。

name1

修飾なしの識別子を指定します。

name1 の後に括弧がなく、 *name1* がターゲット・タイプの属性名と一致している場合、参照解除操作の値は、ターゲット行の名前付き列の値になります。その場合、列のデータ・タイプ (ヌル可能) の値により、参照解除操作の結果タイプが決まります。オブジェクト ID が参照式と一致するターゲット行がない場合、参照解除操作の結果はヌルになります。参照解除操作が選択リストで使用され、式の一部分としては組み込まれていない場合、 *name1* が結果の列名になります。

name1 の後に括弧があるか、*name1* がターゲット・タイプの属性名と一致しない場合、参照解除操作はメソッド呼び出しとして扱われます。呼び出されるメソッドの名前は *name1* です。メソッドのサブジェクトは、ターゲット行であり、その構造タイプのインスタンスと見なされます。オブジェクト ID が参照式と一致するターゲット行がない場合、メソッドのサブジェクトは、ターゲット・タイプのヌル値になります。括弧内に式があれば、それはメソッド呼び出しの残りのパラメーターを指定するものです。

メソッド呼び出しの解決には、通常の処理が使われます。選択したメソッドの結果タイプ (ヌル可能) の値により、参照解除操作の結果タイプが決まります。

参照解除操作を使用するステートメントの許可 ID は、 *scoped-ref-expression* のターゲット表での SELECT 特権を持っていなければなりません (SQLSTATE 42501)。

参照解除操作では、データベース内の値を変更できません。参照解除操作を使用して変更メソッドを呼び出す場合、その変更メソッドはターゲット行のコピーを変更してコピーを戻しますが、データベースは未変更のままです。

例:

- DEPTREF という列がある EMPLOYEE 表 (属性 DEPTNAME を持っているタイプに基づくタイプ付き表を効力範囲に含む参照タイプ) があるとします。表 EMPLOYEE の DEPTREF の値は、 DEPTREF 列のターゲット表にある OID 列値と対応していなければなりません。

```
SELECT EMPNO, DEPTREF->DEPTNAME
  FROM EMPLOYEE
```

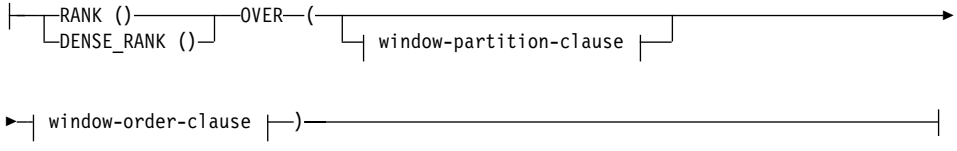
- 前の例と同じ表を使用し、参照解除操作を使って BUDGET というメソッドを呼び出します。その際に、ターゲット行をサブジェクト・パラメーターとして、そして '1997' を追加パラメーターとして指定します。

```
SELECT EMPNO, DEPTREF->BUDGET('1997')
  AS DEPTBUDGET97
  FROM EMPLOYEE
```

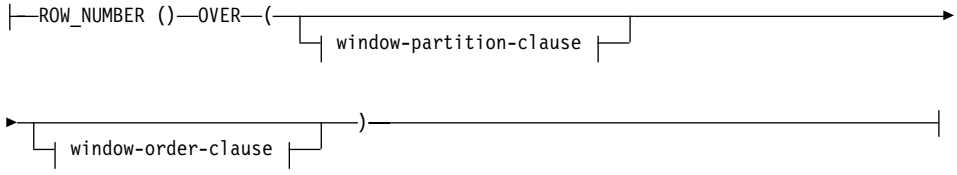
OLAP 関数

OLAP-function:

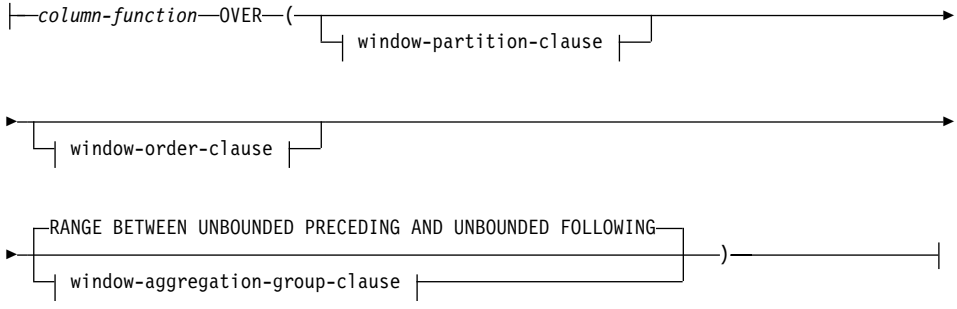
ranking-function:



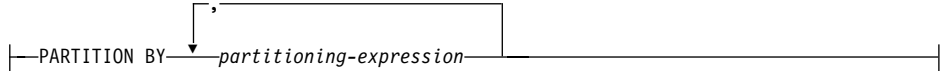
numbering-function:



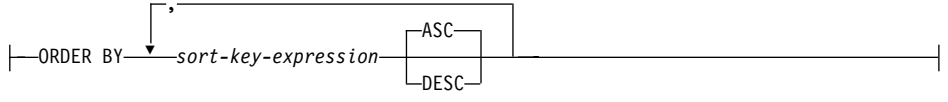
aggregation-function:



window-partition-clause:



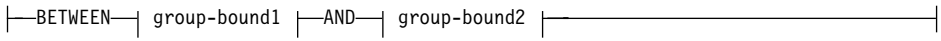
window-order-clause:



window-aggregation-group-clause:

group-start:

group-between:



group-bound1:

group-bound2:

OLAP (On-Line Analytical Processing) 関数には、照会の結果の中で、ランキング、行番号、および既存の列関数情報をスカラー値で戻す機能があります。OLAP 関数は、select-list の式、または select-statement の ORDER BY 文節に組み込むことができます (SQLSTATE 42903)。 OLAP 関数を列関数の引き数として使うことはできません (SQLSTATE 42607)。 OLAP 関数を適用したときの照会の結果は、その OLAP 関数が含まれる、最も内側の副選択の結果表です。

OLAP 関数を指定するときには、関数を適用する行を定義したり、その順序を定義する枠が指定されます。列関数とともに使用すると、該当する行をさらに洗練して、現在行との相対関係で、その前後の行範囲または行数として扱うことができます。たとえば、月単位の区分では、直前の四半期の平均を計算することができます。

ランキング関数は、枠内の行の序数ランクを計算します。それぞれの枠内での順序がはっきりしていない行は、同位に割り当てられます。ランキングの結果については、重複する値の結果の数値にギャップがあってもなくても定義できます。

RANK を指定すると、該当行に先行する行数に 1 を足した数で、行のランクが定義されます。したがって、順序がはっきりしていない行が 2 行以上あると、通しランク番号には、1 つ以上のギャップができます。

DENSE_RANK[36] を指定する場合、順序のはっきりしている先行する行数に 1 を足した数で、行のランクが定義されます。したがって、通しランク番号にはギャップはありません。

ROW_NUMBER[37] 関数は、最初の行を 1 行目としてみなす順序付けに基づいて定義される、枠内の行の通し行番号を計算します。枠内で ORDER BY 文節を指定していない場合、 (SELECT ステートメントの ORDER BY 文節に基づくのではなく) 副選択で戻されたとおりに、任意の順番で行に行番号が割り当てられます。

RANK、DENSE_RANK、または ROW_NUMBER の結果のデータ・タイプは BIGINT です。結果がヌル値になることはありません。

PARTITION BY (*partitioning-expression***,...)**

関数を適用するときの区分を定義します。 *partitioning-expression* は、結果セットの区分化を定義するときに使う式です。 partitioning-expression で参照されている各 *column-name* は、 OLAP 関数副選択ステートメントの結果セット列をはっきり参照するものでなければなりません (SQLSTATE 42702 または 42703)。各 partitioning-expression の長さは、255 バイトを超えてはなりません (SQLSTATE 42907)。 partitioning-expression には、scalar-fullselect (SQLSTATE 42822) や、可変の関数または外部処理を伴う関数 (SQLSTATE 42845) を含めることはできません。

ORDER BY (*sort-key-expression***,...)**

OLAP 関数の値、または window-aggregation-group-clause の ROW 値の意味を決める、区分内の行の順序を定義します (照会結果セットの順序を定義するものではありません)。 *sort-key-expression* は、枠の区分に含まれる行の順序を定義するときに使う式です。 sort-key-expression で参照されている各 column-name は、 OLAP 関数を含む副選択の結果セットの列をはっきり参照するものでなければなりません (SQLSTATE 42702 または 42703)。各 sort-key-expression の長さは、255 バイトを超えてはなりません (SQLSTATE 42907)。 sort-key-expression には、scalar-fullselect (SQLSTATE 42822) や、可変の関数または外部処理を伴う関数 (SQLSTATE 42845) を含めることはできません。この文節は、RANK および DENSE_RANK 関数 (SQLSTATE 42601) で必要になります。

ASC

sort-key-expression の値を昇順に使用します。ヌル値は順序の最後になります。

---

36. DENSE_RANK と DENSERANK は同義です。

37. ROW_NUMBER と ROWNUMBER は同義です。

DESC
sort-key-expression の値を降順に使用します。ヌル値は順序の最初になります。

window-aggregation-group-clause
行 R の集約グループは一連の行であり、 R の区分の行の順序で R に対して定義されます。この文節は、集約グループを指定します。

ROWS
カウント行ごとに集約グループが定義されることを示します。

RANGE
分類キーからの相対位置によって集約グループが定義されることを示します。

group-start
集約グループの開始点を指定します。集約グループが終了するのは、現在行です。 group-start 文節を指定することは、 ″BETWEEN group-start AND CURRENT ROW″ の形式で group-between 文節を指定することと同じです。

group-between
ROWS か RANGE に基づいて集約グループの開始点と終了点を指定します。

UNBOUNDED PRECEDING
現在行より前の区分全体を組み込みます。これは、ROWS または RANGE に基づいて指定できます。また、window-order-clause 内で複数の sort-key-expression を使って、指定することも可能です。

UNBOUNDED FOLLOWING
現在行より後の区分全体を組み込みます。これは、ROWS または RANGE に基づいて指定できます。また、window-order-clause 内で複数の sort-key-expression を使って、指定することも可能です。

CURRENT ROW
集約グループの開始点または終了点を現在行として指定します。 *group-bound1* で *value* FOLLOWING を指定している場合には、 *group-bound2* にこの文節を指定することはできません。

value **PRECEDING**
現在行より前の行の範囲か行数を指定します。 ROWS が指定されている場合、*value* は行数を示す正の整数になります。 RANGE が指定されている場合、*value* のデータ・タイプは、 window-order-clause の sort-key-expression のタイプと比較できるものでなければなりません。

sort-key-expression は 1 つだけ指定でき、 sort-key-expression のデータ・タイプは減算できるものでなければなりません。 *group-bound1* が CURRENT ROW または *value* FOLLOWING の場合には、*group-bound2* にこの文節を指定することはできません。

value **FOLLOWING**
現在行より後の行の範囲か行数を指定します。 ROWS が指定されている場合、*value* は行数を示す正の整数になります。 RANGE が指定されている場合、*value* のデータ・タイプは、 window-order-clause の sort-key-expression のタイプと比較できるものでなければなりません。 sort-key-expression は 1 つだけ指定でき、 sort-key-expression のデータ・タイプは加算できるものでなければなりません。

例:

- 給与合計 (給与 + ボーナス) が $30,000 を超えている従業員のランクを、それぞれの給与合計に基づいて、姓の順に表示します。

```
SELECT EMPNO, LASTNAME, FIRSTNME, SALARY+BONUS AS TOTAL_SALARY,
       RANK() OVER (ORDER BY SALARY+BONUS DESC) AS RANK_SALARY
  FROM EMPLOYEE WHERE SALARY+BONUS > 30000
  ORDER BY LASTNAME
```

結果をランキング順に並べる場合、ORDER BY LASTNAME を以下のように置き換えます。

```
ORDER BY RANK_SALARY
```

または

```
ORDER BY RANK() OVER (ORDER BY SALARY+BONUS DESC)
```

- それぞれの給与合計の平均に基づいて部門をランク付けします。

```
SELECT WORKDEPT, AVG(SALARY+BONUS) AS AVG_TOTAL_SALARY,
       RANK() OVER (ORDER BY AVG(SALARY+BONUS) DESC) AS RANK_AVG_SAL
  FROM EMPLOYEE
  GROUP BY WORKDEPT
  ORDER BY RANK_AVG_SAL
```

- それぞれの学歴に基づいて部門内で従業員をランク付けします。部門内で同じランクの従業員が複数いたとしても、次のランキング値を増やさないようにします。

```
SELECT WORKDEPT, EMPNO, LASTNAME, FIRSTNME, EDLEVEL
       DENSE_RANK() OVER
         (PARTITION BY WORKDEPT ORDER BY EDLEVEL DESC) AS RANK_EDLEVEL
  FROM EMPLOYEE
  ORDER BY WORKDEPT, LASTNAME
```

- 照会の結果に行番号を含めます。

```
SELECT ROW_NUMBER() OVER (ORDER BY WORKDEPT, LASTNAME) AS NUMBER,
       LASTNAME, SALARY
  FROM EMPLOYEE
  ORDER BY WORKDEPT, LASTNAME
```

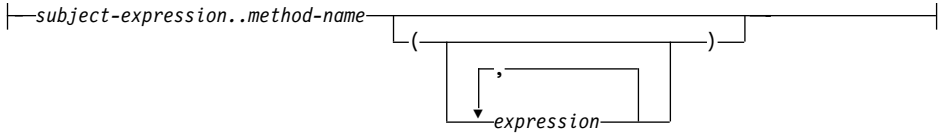
- 収入の多い上位 5 人をリストします。

```
SELECT EMPNO, LASTNAME, FIRSTNME, TOTAL_SALARY, RANK_SALARY
   FROM (SELECT EMPNO, LASTNAME, FIRSTNME, SALARY+BONUS AS TOTAL_SALARY,
                RANK() OVER (ORDER BY SALARY+BONUS DESC) AS RANK_SALARY
                FROM EMPLOYEE) AS RANKED_EMPLOYEE
   WHERE RANK_SALARY < 6
   ORDER BY RANK_SALARY
```

結果を最初に計算する際、ランクを WHERE 文節で使う前に、そのランクが含まれた、ネストされた表の式が使われていることに注意してください。共通表式も使われています。

メソッドの呼び出し

method-invocation:



システム生成による監視および変更メソッドの両方、さらにユーザー定義メソッドも、二重ドット演算子を使って呼び出されます。

subject-expression
ユーザー定義構造タイプである静的結果タイプを持つ式。

method-name
修飾なしのメソッド名。 *subject-expression* の静的タイプまたはそのスーパータイプのいずれかに、指定した名前を持つメソッドが含まれている必要があります。

(*expression***,...)**
括弧内に *method-name* の引き数を指定します。引き数がないことを示すときには、括弧内を空にしておくことができます。特定のメソッドを解決するときに、*subject-expression* の静的タイプに基づき、 *method-name* と、指定した引き数の式のデータ・タイプを使用します (詳細は、 169ページの『メソッド解決』を参照してください)。

メソッド呼び出しに使う二重ドット演算子は、優先順位が高い順に左から右へ列挙される挿入演算子です。たとえば、以下の 2 つの式は同じことを意味します。

```
a..b..c + x..y..z
```

および

```
((a..b)..c) + ((x..y)..z)
```

メソッドにサブジェクト以外のパラメーターがない場合、括弧はあってもなくても呼び出すことができます。たとえば、以下の 2 つの式は同じことを意味します。

```
point1..x
```

および

```
point1..x()
```

メソッド呼び出しのヌル・サブジェクトは、次のように扱われます。

1. システム生成の変更メソッドがヌル・サブジェクトで呼び出される場合、エラーになります (SQLSTATE 2202D)。
2. システム生成の変更メソッド以外のメソッドがヌル・サブジェクトで呼び出される場合、そのメソッドは実行されず、結果はヌルになります。この規則は、SELF AS RESULT を指定したユーザー定義メソッドにも当てはまります。

データベース・オブジェクト (パッケージ、視点、またはトリガーなど) を作成する場合、それぞれのメソッド呼び出しのための最適な方法は、 169ページの『メソッド解決』で指定した規則を使用して見つけられます。

注:

- 定義した WITH FUNCTION ACCESS タイプのメソッドは、通常の関数表記を使用して呼び出すこともできます。関数解決では、候補となる関数として、すべての関数だけでなく、関数アクセスのあるメソッドも考慮します。ただし、メソッド呼び出しを使用して関数を呼び出すことはできません。メソッド解決では、候補となるメソッドとして、すべてのメソッドを考慮しますが、関数については考慮しません。適切な関数またはメソッドの解決に失敗すると、エラーになります (SQLSTATE 42884)。

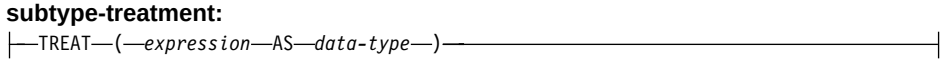
例:

- 二重ドット演算子を使用して、AREA というメソッドを呼び出します。構造タイプ CIRCLE の列 CIRCLE_COL がある、RINGS という表が存在するとします。また、CIRCLE タイプのために、メソッド AREA が、AREA() RETURNS DOUBLE としてあらかじめ定義されているとします。

```
SELECT CIRCLE_COL..AREA()
   FROM RINGS
```

サブタイプの扱い

subtype-treatment:



subtype-treatment は、構造タイプの式を、そのサブタイプのいずれかへキャストするときに使います。 *expression* の静的タイプは、ユーザー定義構造タイプでなければなりません。このタイプは、*data-type* と同じタイプであるか、またはそのスーパータイプでなければなりません。 *data-type* のタイプ名が修飾されていない場合は、 SQL パスを使用してタイプ参照を解決します。subtype-treatment の結果の静的タイプは *data-type* であり、 subtype-treatment の値は expression の値になります。実行時に、expression の動的タイプが *data-type* ではないか、 *data-type* のサブタイプでない場合、エラーが戻されます (SQLSTATE 0D000)。

例:

- 列 CIRCLE_COL のすべての列オブジェクト・インスタンスに、動的タイプ COLOREDCIRCLE があることを、アプリケーション側が認識している場合、次の照会を使って、そのようなオブジェクト上でメソッド RGB を呼び出します。構造タイプ CIRCLE の列 CIRCLE_COL がある、RINGS という表が存在するとします。また、COLOREDCIRCLE は CIRCLE のサブタイプであり、COLOREDCIRCLE のために、メソッド RGB が RGB() RETURNS DOUBLE としてあらかじめ定義されているとします。

```
SELECT TREAT (CIRCLE_COL AS COLOREDCIRCLE)..RGB()
   FROM RINGS
```

  実行時に、動的タイプ CIRCLE のインスタンスが存在する場合、エラーになります (SQLSTATE 0D000)。このエラーは、次に示すように、CASE 式の中で TYPE 述部を使うことで避けることができます。

```
SELECT (CASE
   WHEN CIRCLE_COL IS OF (COLOREDCIRCLE)
      THEN TREAT (CIRCLE_COL AS COLOREDCIRCLE)..RGB()
      ELSE NULL
   END)
   FROM RINGS
```

詳細については、225ページの『TYPE 述部』を参照してください。

述部

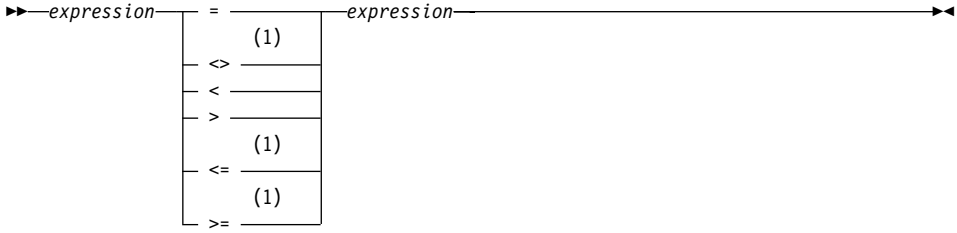
述部 とは、特定の行またはグループに対して「真」、「偽」、または「不定」の条件を指定するものです。

以下の規則は、すべてのタイプの述部に適用されます。

- 述部の中で指定される値は、すべて互換であること。
- 基本、多値比較、IN、または BETWEEN 述部の結果が、長さ属性が 4 000 を超える文字ストリング、長さ属性が 2 000 を超える漢字ストリング、任意のサイズの LOB ストリングになってはなりません。
- ホスト変数の値は、ヌル値となる可能性があること (つまり、変数が負の標識変数を持つことがある)。
- LIKE を除き、2 つ以上のオペランドを伴う述部のオペランドのコード・ページ変換は、 126ページの『ストリング変換に関する規則』に従って行われます。
- DATALINK 値の使用は、NULL 述部に限定されています。
- 構造タイプ値の使用は、NULL 述部と TYPE 述部に限定されています。

全選択は、421ページの『第5章 照会』で説明される SELECT ステートメントの 1 つの形式です。述部で使用される全選択は*副照会* とも呼ばれます。

基本述部



注:

1 その他の比較演算子もサポートされています。[38]

基本述部 は 2 つの値を比較します。

一方のオペランドの値がヌル値の場合、述部の結果は不定です。それ以外の場合の結果は、真または偽のいずれかになります。

値 *x* および *y* について、次のような関係が成り立ちます。

述部	次の場合にのみ「真」になります。
x **=** *y*	*x* は *y* に等しい
x **<>** *y*	*x* は *y* に等しくない
x **<** *y*	*x* は *y* より小さい
x **>** *y*	*x* は *y* より大きい
x **>=** *y*	*x* は *y* より大きいか等しい
x **<=** *y*	*x* は *y* より小さいか等しい

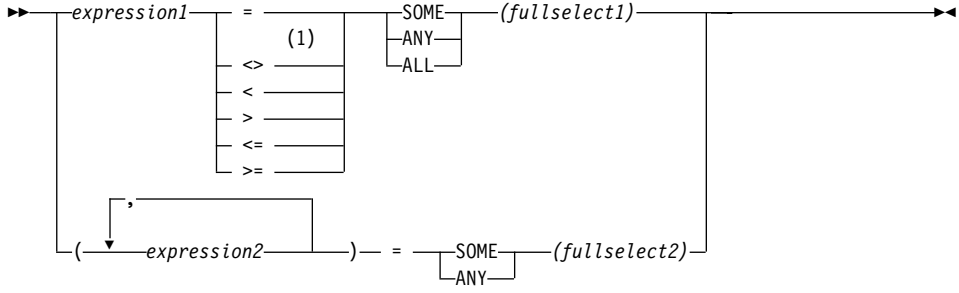
例:

```
EMPNO='528671'
SALARY < 20000
PRSTAFF <> :VAR1
SALARY > (SELECT AVG(SALARY) FROM EMPLOYEE)
```

38. 基本述部および比較述部では、 ^=、^<、^>、!=、!< および !> の形式の比較演算子もサポートされています。さらに、コード・ページ 437、819、および 850 では、 ¬=、¬<、および ¬> の形式もサポートされています。

このような製品固有の比較演算子の形式は、このような演算子を使用する既存の SQL をサポートすることのみを目的としており、新たに SQL ステートメントを書く場合には使用しないようお勧めします。

比較述部



注:

1 その他の比較演算子もサポートされています。[38]

比較述部 は、 1 つの値もしくは複数の値と、値の集合との間で比較を行います。

全選択は、述部演算子の左側に指定されている式の数と同じ数の列を識別しなければなりません (SQLSTATE 428C4)。全選択は、任意の行数を戻すことができます。

ALL を指定した場合、

- 全選択が値をまったく戻さない場合、または指定した関係が、全選択によって戻される値のすべてに対して「真」である場合、述部の結果は「真」となります。
- 指定した関係が、全選択によって戻される値の少なくとも 1 つに対して「偽」である場合、述部の結果は「偽」となります。
- 指定した関係が、全選択によって戻されるどの値に対しても「偽」でなく、少なくとも 1 つの比較がヌル値のために「不定」である場合、述部の結果は「不定」となります。

SOME または ANY を指定した場合、

- 指定した関係が、全選択によって戻される少なくとも 1 つの行の各値に対して「真」である場合、述部の結果は「真」になります。
- 全選択が行を戻さない場合、または指定した関係が、全選択によって戻されるすべての行の少なくとも 1 つの値に対して「偽」である場合、述部の結果は「偽」になります。
- 指定した関係がどの行に対しても「真」でなく、少なくとも 1 つの比較がヌル値のために「不定」である場合、述部の結果は「不定」になります。

例: 以降の例を参照する場合、次の表を使用してください。

TBL***AB***:

COL***A***	COL***B***
1	12
2	12
3	13
4	14
-	-

TBL***XY***:

COL***X***	COL***Y***
2	22
3	23

図 10.

例 1

```
SELECT COLA FROM TBLAB
   WHERE COLA = ANY(SELECT COLX FROM TBLXY)
```

結果は 2、3 です。副選択は (2,3) を戻します。行 2 と 3 の COLA は、それらの値の少なくとも 1 つに等しくなっています。

例 2

```
SELECT COLA FROM TBLAB
   WHERE COLA > ANY(SELECT COLX FROM TBLXY)
```

結果は 3、4 です。副選択は (2,3) を戻します。行 3 と 4 の COLA は、それらの値の少なくとも 1 つより大きくなっています。

例 3

```
SELECT COLA FROM TBLAB
   WHERE COLA > ALL(SELECT COLX FROM TBLXY)
```

結果は 4 です。副選択は (2,3) を戻します。それらの値の両方より大きいものは、行 4 の COLA しかありません。

例 4

```
SELECT COLA FROM TBLAB
   WHERE COLA > ALL(SELECT COLX FROM TBLXY
                    WHERE COLX<0)
```

結果は 1、2、3、4、ヌル値です。副選択は値を戻しません。したがって、述部は TBLAB のすべての行に対して真です。

例 5

```
SELECT * FROM TBLAB
   WHERE (COLA,COLB+10) = SOME (SELECT COLX, COLY FROM TBLXY)
```

副選択は TBLXY からすべての項目を戻します。述部は副選択に対して真であるため、結果は次のようになります。

```
COLA        COLB
----------- -----------
          2          12
          3          13
```

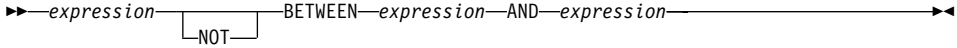
例 6

```
SELECT * FROM TBLAB
   WHERE (COLA,COLB) = ANY (SELECT COLX,COLY-10 FROM TBLXY)
```

副選択は TBLXY から COLX および COLY-10 を戻します。述部は副選択に対して真であるため、結果は次のようになります。

```
COLA        COLB
----------- -----------
          2          12
          3          13
```

BETWEEN 述部



BETWEEN 述部は、ある値を値の範囲と比較します。

次の BETWEEN 述部は、

```
value1 BETWEEN value2 AND value3
```

次の探索条件と同等です。

```
value1 >= value2 AND value1 <= value3
```

次の BETWEEN 述部は、

```
value1 NOT BETWEEN value2 AND value3
```

次の探索条件と同等です。

```
NOT(value1 BETWEEN value2 AND value3); that is,
value1 < value2 OR value1 > value3.
```

BETWEEN 述部の式の値ごとにコード・ページが異なる場合があります。この場合、オペランドは、上記の同等な探索条件が指定された場合と同じようにして変換されます。

第 1 オペランド (expression) に、可変の関数または外部処理を伴う関数を含めることはできません (SQLSTATE 426804)。

日付 / 時刻値と日付 / 時刻値のストリング表記が混在している場合、すべての値は日付 / 時刻オペランドのデータ・タイプに変換されます。

例:

例 1

```
EMPLOYEE.SALARY BETWEEN 20000 AND 40000
```

結果は $20,000.00 と $40,000.00 の間のすべての給与となります。

例 2

```
SALARY NOT BETWEEN 20000 + :HV1 AND 40000
```

:HV1 が 5000 であるとすると、結果は $25,000.00 より低いか $40,000.00 より高いすべての給与となります。

例 3

次の条件がある場合に、

表 12.

式	タイプ	コード・ページ
HV_1	ホスト変数	437
HV_2	ホスト変数	437
Col_1	列	850

ここで、以下の述部を評価すると、

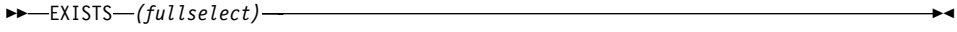
```
  :HV_1 BETWEEN :HV_2 AND COL_1
```

次のように解釈されます。

```
  :HV_1 >= :HV_2
AND :HV_1 <= COL_1
```

:HV_1 の最初のオカレンスは、やはりアプリケーションのコード・ページのままである :HV_2 と比較されるため、アプリケーションのコード・ページのままになります。 :HV_1 の 2 番目のオカレンスは、列値と比較されるため、データベースのコード・ページに変換されます。

EXISTS 述部



EXISTS 述部は、特定の行の存在を調べるためのものです。

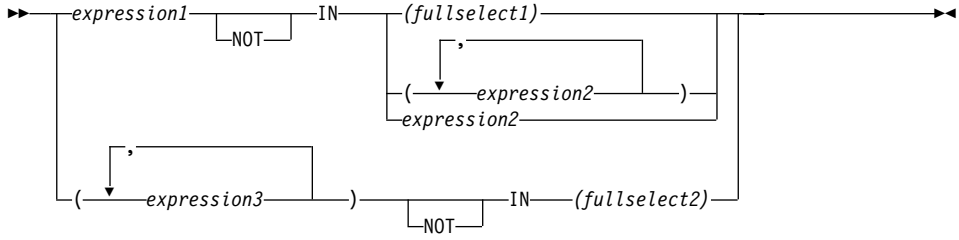
fullselect (全選択) には、必要な数の列を指定できます。

- fullselect に指定された行数がゼロでない場合にのみ、結果が「真」になります。
- 指定された行数がゼロの場合にのみ、結果が「偽」になります。
- 結果が「不定」になることはありません。

例:

```
EXISTS (SELECT * FROM TEMPL WHERE SALARY < 10000)
```

IN 述部



IN 述部は、1 つの値または複数の値を値の集合と比較します。

全選択は、 IN キーワードの左側に指定されている式の数と同じ数の列を識別しなければなりません (SQLSTATE 428C4)。全選択は、任意の行数を戻すことができます。

- 次の形式の IN 述部は、

  ```
  expression IN expression
  ```

  以下の形式の基本述部と同等です。

  ```
  expression = expression
  ```

- 次の形式の IN 述部は、

  ```
  expression IN (fullselect)
  ```

  以下の形式の比較述部と同等です。

  ```
  expression = ANY (fullselect)
  ```

- 次の形式の IN 述部は、

  ```
  expression NOT IN (fullselect)
  ```

  以下の形式の比較述部と同等です。

  ```
  expression <> ALL (fullselect)
  ```

- 次の形式の IN 述部は、

  ```
  expression IN (expressiona, expressionb, ..., expressionk)
  ```

  以下と同等です。

  ```
  expression = ANY (fullselect)
  ```

  この fullselect は、values 文節形式では次のようになります。

  ```
  VALUES (expressiona), (expressionb), ..., (expressionk)
  ```

- 次の形式の IN 述部は、

```
(expressiona, expressionb,..., expressionk) IN (fullselect)
```

以下の形式の比較述部と同等です。

```
(expressiona, expressionb,..., expressionk) = ANY (fullselect)
```

IN 述部の *expression1* および *expression2* の値、または *fullselect1* の列には、互換性が必要です。 IN 述部の *expression3* の各値、およびそれに対応する *fullselect2* の列にも互換性が必要です。比較で使用される結果の属性については、121ページの『結果のデータ・タイプに関する規則』を参照してください。

IN 述部の式の値 (全選択の対応する列を含めて) のコード・ページが異なっていても構いません。変換が必要な場合は、126ページの『ストリング変換に関する規則』をまず IN リストに適用し、次に第 2 オペランドとして IN リストの導出コード・ページを使って同じ規則を述部に適用することによって、コード・ページが決定されます。

例:

例 1: DEPTNO 列で評価の対象となる行の値に D01、B01、または C01 が含まれている場合、以下は真であると評価されます。

```
DEPTNO IN ('D01', 'B01', 'C01')
```

例 2: 左側の EMPNO (従業員番号) が部門 E11 の従業員の EMPNO と一致する場合のみ、以下は真であると評価されます。

```
EMPNO IN (SELECT EMPNO FROM EMPLOYEE WHERE WORKDEPT = 'E11')
```

例 3: 以下の情報に基づき、COL_1 列の行の特定の値が、リスト内のいずれかの値と一致する場合、この例は真であると評価されます。

表 13. IN 述部の例

式	タイプ	コード・ページ
COL_1	列	850
HV_2	ホスト変数	437
HV_3	ホスト変数	437
CON_1	定数	850

ここで、以下の述部を評価すると、

```
COL_1 IN (:HV_2, :HV_3, CON_4)
```

126ページの『ストリング変換に関する規則』に基づいて、2 個のホスト変数がコード・ページ 850 に変換されます。

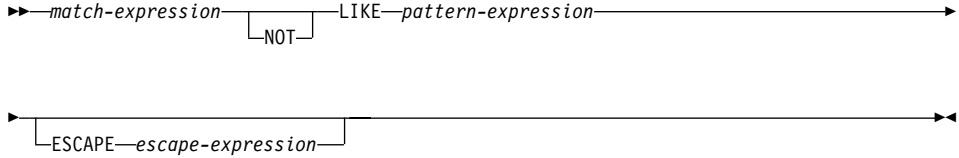
例 4: EMENDATE に指定された年 (プロジェクトの従業員の活動が終了した日付) が、リストに指定された値のいずれか (現在の年または過去 2 年) と一致する場合、以下は真として評価されます。

```
YEAR(EMENDATE) IN (YEAR(CURRENT DATE),
                   YEAR(CURRENT DATE - 1 YEAR),
                   YEAR(CURRENT DATE - 2 YEARS))
```

例 5: 左側の ID と DEPT の両方が、 ORG 表の任意の行の MANAGER と DEPTNUMB にそれぞれ一致する場合、以下は真として評価されます。

```
(ID, DEPT) IN (SELECT MANAGER, DEPTNUMB FROM ORG)
```

LIKE 述部



LIKE 述部は、ある一定のパターンを含むストリングを探索するものです。パターンは、特殊な意味のある下線とパーセント記号を使ったストリングによって指定されます。パターンでは後続ブランクもパターンの一部です。

引き数の中に値がヌル値の引き数のものがある場合、 LIKE 述部の結果は不定になります。

match-expression (一致式)、 *pattern-expression* (パターン式)、および *escape-expression* (エスケープ式) の値は、互換性のある文字ストリング式です。サポートされる文字ストリング式のタイプは、各引き数ごとに少しずつ違います。式の有効なタイプは、各引き数の説明の中に示します。

どの式も特殊タイプを生成するものであってはなりません。ただし、特殊タイプをそのソース・タイプへキャストする関数は可能です。

match-expression
特定の文字パターンに適合するかどうか調べる対象のストリングを指定する式。

この式は、以下のいずれかを使って指定できます。

- 定数
- 特殊レジスター
- ホスト変数 (ロケーター変数またはファイル参照変数を含む)
- スカラー関数
- ラージ・オブジェクトのロケーター
- 列名
- 上記のいずれかを連結する式

pattern-expression
一致すべき基準となるストリングを指定する式。

この式は、以下のいずれかを使って指定できます。

- 定数

- 特殊レジスター
- ホスト変数
- 上記のいずれかをオペランドとするスカラー関数
- 上記のいずれかを連結する式

以下の制約があります。

- 式の要素に、LONG VARCHAR、CLOB、LONG VARGRAPHIC、または DBCLOB のタイプを使うことはできません。また、BLOB ファイル参照変数は使えません。
- *pattern-expression* の実際の長さは、 32 672 バイト以下でなければなりません。

LIKE パターンについて**簡単に説明**するとすれば、これは *match-expression* の値のための適合基準を指定するために使用されるパターンです。これには以下の規則があります。

- 下線文字 (_) は、任意の 1 文字を表します。
- パーセント記号 (%) は、ゼロ個以上の文字から構成されるストリングを表します。
- その他の文字は、その文字自身を表します。

pattern-expression に下線またはパーセント文字を含める必要がある場合は、 *escape-expression* によって、パターンの中で下線またはパーセント文字に先行させる文字を指定します。

LIKE パターンについて**厳密に説明**するとすれば、以下のようにします。この説明では、*escape-expression* の使用は無視されています。それについては後で説明します。

- *m* が *match-expression* の値を、 *p* が *pattern-expression* の値を表すとします。ストリング *p* は、最小の数のサブストリング指定子の並びとして解釈されるので、 *p* の各文字はただ 1 つのサブストリング指定子の一部となります。サブストリング指定子とは、下線、パーセント記号、または下線およびパーセント記号以外の任意の文字の空でない並びです。

  m または *p* がヌル値の場合は、述部の結果が不定になります。それ以外の場合の結果は、真か偽のどちらかになります。 *m* と *p* の両方が空ストリングの場合、または以下のようにして *m* をサブストリングに区分化したものが存在する場合、結果は真になります。

  – *m* のサブストリングがゼロ個以上の連続する文字の並びで、 *m* の各文字がちょうど 1 つのサブストリングの一部である。

- *n* 番目のサブストリング指定子が下線の場合、 *m* の *n* 番目のサブストリング指定子は任意の 1 文字である。
- *n* 番目のサブストリング指定子がパーセント記号の場合、 *m* の *n* 番目のサブストリング指定子は 0 個以上の文字の並びである。
- *n* 番目のサブストリング指定子が下線でもパーセント記号でもない場合、 *m* の *n* 番目のサブストリングは、対応するサブストリング指定子と等しく、同じ長さである。
- *m* のサブストリングの数は、サブストリング指定子の数と同じである。

p が空ストリングで、 *m* が空ストリングでない場合、結果は偽になります。同様に、*m* が空ストリングで、 *p* が空ストリングでない場合、結果は偽になります。

述部 *m* NOT LIKE *p* は、探索条件 NOT (*m* LIKE *p*) と同等です。

escape-expression が指定されている場合、直後にエスケープ文字、下線文字、またはパーセント記号文字が続くのでない限り、 *pattern-expression* の中に、 *escape-expression* で指定されるエスケープ文字が含まれていてはなりません (SQLSTATE 22025)。

match-expression が MBCS データベースの文字ストリングの場合、それには**混合データ**を含めることができます。この場合は、パターンに SBCS 文字と MBCS 文字の両方を含めることができます。パターンの中の特殊文字は、以下のようにして解釈されます。

- SBCS の下線は、1 つの SBCS 文字を指します。
- DBCS の下線は、1 個の MBCS 文字を指します。
- パーセント (SBCS または DBCS) は、 SBCS または MBCS のゼロ個以上の文字からなるストリングを指します。

escape-expression

これはオプションの引き数であり、 *pattern-expression* の中の下線 (_) 文字とパーセント (%) 文字の特別な意味を変更するための文字を指定する式です。これにより、実際にパーセントや下線文字を含む値との一致を調べるために LIKE 述部を使うことができます。

この式は、以下のいずれかを使って指定できます。

- 定数
- 特殊レジスター
- ホスト変数
- 上記のいずれかをオペランドとするスカラー関数

- 上記のいずれかを連結する式

以下の制約があります。

- 式の要素に、LONG VARCHAR、CLOB、LONG VARGRAPHIC、または DBCLOB のタイプを使うことはできません。また、BLOB ファイル参照変数は使えません。
- 式の結果は、SBCS または DBCS の 1 文字、または 1 バイトだけを含む 2 進ストリングでなければなりません (SQLSTATE 22019)。

パターン・ストリングにエスケープ文字が含まれている場合、下線、パーセント記号、またはエスケープ文字は、それ自体のリテラル・オカレンスを表すことができます。これは、その文字が奇数個の連続したエスケープ文字に先行されている場合です。そうでない場合は当てはまりません。

パターンの中で、連続するエスケープ文字の並びは以下のように扱われます。

- S がそのような並びであり、さらに長く連続するエスケープ文字の並びの一部となってはいないものとします。また、S が合計で n 文字を含むものとします。このとき、S に適用される規則は、n の値により以下のように異なります。
  - n が奇数の場合、S の後には下線またはパーセント記号がなければなりません (SQLSTATE 22025)。 S とその後に続く文字は、エスケープ文字の (n-1)/2 個のリテラル・オカレンスの後に下線記号またはパーセント記号のリテラル・オカレンスが続くことを表します。
  - n が偶数の場合、S はエスケープ文字の n/2 個のリテラル・オカレンスを表します。 n が奇数の場合とは異なり、S でパターンが終了する場合もあります。 S でパターンが終了しない場合、S の後にはどんな文字が続いても構いません (ただし、 S がさらに長く連続したエスケープ文字の並びの一部ではないとしているので、当然エスケープ文字は除外されます)。 S の後に下線記号またはパーセント記号が続く場合、その文字には特別な意味があります。

以下は、エスケープ文字 (この場合は、円記号 (¥)) の連続したオカレンスの影響を示しています。

パターン・ストリング	実際のパターン
¥%	パーセント記号
¥¥%	1 つの円記号の後にゼロ個以上の任意の文字が続く
¥¥¥%	1 つの円記号の後に 1 つのパーセント記号が続く

比較で使用されるコード・ページは、 *match-expression* の値のコード・ページに基づいて決定されます。

- *match-expression* の値が変換されることはありません。
- *pattern-expression* のコード・ページが、 *match-expression* のコード・ページと異なる場合、 *pattern-expression* の値が *match-expression* のコード・ページに変換されます。ただし、どちらかのオペランドが FOR BIT DATA で定義されている場合は除きます (その場合は変換されません)。
- *escape-expression* のコード・ページが、 *match-expression* のコード・ページと異なる場合、 *escape-expression* の値が *match-expression* のコード・ページに変換されます。ただし、どちらかのオペランドが FOR BIT DATA で定義されている場合は除きます (その場合は変換されません)。

例

- PROJECT 表の PROJNAME 列で 'SYSTEMS' というストリングを探索します。

```
SELECT PROJNAME FROM PROJECT
  WHERE PROJECT.PROJNAME LIKE '%SYSTEMS%'
```

- EMPLOYEE 表の FIRSTNME 列で、先頭の文字が 'J'で、長さがちょうど 2 文字のストリングを探索します。

```
SELECT FIRSTNME FROM EMPLOYEE
  WHERE EMPLOYEE.FIRSTNME LIKE 'J_'
```

- EMPLOYEE 表の FIRSTNME 列で、先頭の文字が 'J'で、任意の長さのストリングを探索します。

```
SELECT FIRSTNME FROM EMPLOYEE
  WHERE EMPLOYEE.FIRSTNME LIKE 'J%'
```

- CORP_SERVERS 表で、LA_SERVERS 列のストリングのうち、 CURRENT SERVER 特殊レジスターの値と一致するものを探索します。

```
SELECT LA_SERVERS FROM CORP_SERVERS
  WHERE CORP_SERVERS.LA_SERVERS LIKE CURRENT SERVER
```

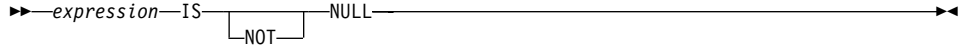
- 表 T の列 A で '%_¥' で始まるすべてのストリングを検索します。

```
SELECT A FROM T WHERE T.A LIKE
'¥%¥_¥¥%' ESCAPE '¥'
```

- 一致ストリングとパターン・ストリングのデータ・タイプ (どちらも BLOB) と互換である 1 バイトのエスケープ文字を獲得するには、次のように BLOB スカラー関数を使用します。

```
SELECT COLBLOB FROM TABLET
  WHERE COLBLOB LIKE :pattern_var ESCAPE BLOB(X'0E')
```

NULL 述部



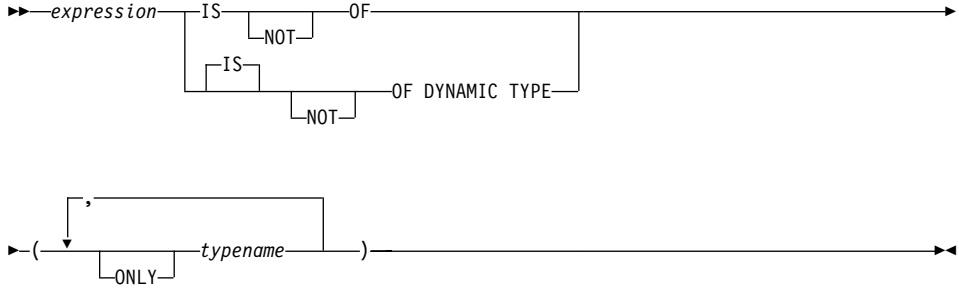
NULL 述部は、ヌル値かどうかを検査するものです。

NULL 述部の結果が不定になることはありません。式の値がヌル値の場合、結果は真になります。値がヌル値でない場合、結果は偽になります。 NOT が指定されている場合、結果は逆になります。

例:

```
PHONENO IS NULL
SALARY IS NOT NULL
```

TYPE 述部



TYPE 述部 は、式のタイプと 1 つまたは複数のユーザー定義構造タイプとを比較します。

参照タイプの参照解除を含んでいる式の動的タイプは、ターゲットのタイプ付き表または視点にある参照される行の実際のタイプです。これは、その参照を含んでいる式のターゲット・タイプ (式の静的タイプと呼ばれる) とは異なる場合があります。

expression の値がヌルの場合、述部の結果は不明です。 *expression* の動的タイプが *typename* で指定された構造タイプの 1 つのサブタイプの場合、述部の結果は「真」になり、そうでない場合は「偽」になります。 ONLY のあとに *typename* がある場合、そのタイプの適切なサブタイプは考慮されません。

typename が修飾されていない場合、SQL パスを使用して解決されます。 *typename* は、 *expression* の静的タイプのタイプ階層にあるユーザー定義タイプを識別しなければなりません (SQLSTATE 428DU)。

DEREF 関数は、参照タイプの値を含んでいる式が TYPE 述部にある場合はいつでも、使用されなければなりません。 *expression* がこの形式の場合の静的タイプは、参照のターゲット・タイプです。 DEREF 関数の詳細については、307ページの『DEREF』を参照してください。

構文上の IS OF と OF DYNAMIC TYPE は、TYPE 述部では同じ働きをします。同様に、IS NOT OF と NOT OF DYNAMIC TYPE も TYPE 述部では同じ働きをします。

例:

ある表階層には、タイプ EMP のルート表 EMPLOYEE と、タイプ MGR の副表 MANAGER があります。別の表 ACTIVITIES は、 REF(EMP) SCOPE

EMPLOYEE として定義されている WHO_RESPONSIBLE という列を含んでいます。 WHO_RESPONSIBLE と対応する行が管理者の場合に、結果が「真」となるタイプ述部を以下に示します。

```
DEREF (WHO_RESPONSIBLE) IS OF (MGR)
```

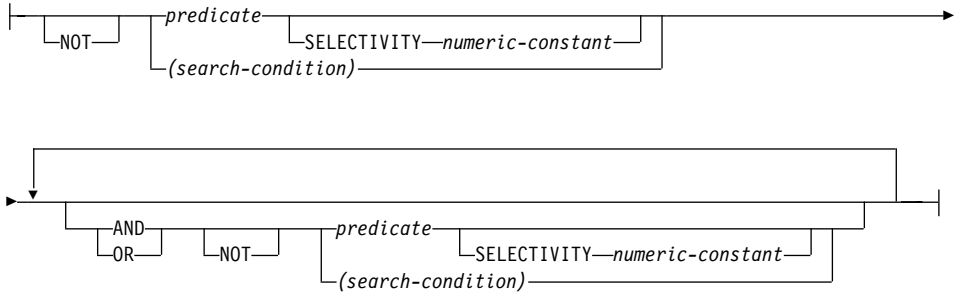
表にタイプ EMP の列 EMPLOYEE が含まれている場合、EMPLOYEE には、タイプ EMP の値だけではなく、MGR のようなサブタイプの値を含めることができます。次のような述部は、

```
EMPL IS OF (MGR)
```

EMPL がヌルではなく、実際に管理職である場合に、「真」を戻します。

探索条件

search-condition:



search-condition (探索条件) は、特定の行について「真」、「偽」、または「不定」となる条件を指定します。

探索条件の結果は、指定した各述部の結果に、指定した*論理演算子* (AND、OR、NOT) を適用することによって求められます。論理演算子の指定がない場合、探索条件の結果は指定された述部の結果になります。

AND と OR は、表14 で定義されています。表中の P と Q は任意の述部です。

表 14. AND と OR の真理値表

P	Q	P AND Q	P OR Q
真	真	真	真
真	偽	偽	真
真	不定	不定	真
偽	真	偽	真
偽	偽	偽	偽
偽	不定	偽	不定
不定	真	不定	真
不定	偽	偽	不定
不定	不定	不定	不定

NOT(true) は偽、NOT(false) は真、NOT(unknown) は不定です。

括弧の中の探索条件が最初に評価されます。評価の順序を括弧によって指定していない場合、 NOT が AND の前に適用され、AND が OR の前に適用され

ます。同じ優先順位の演算子が評価される順序は、探索条件の最適化を図るために定義されていません。

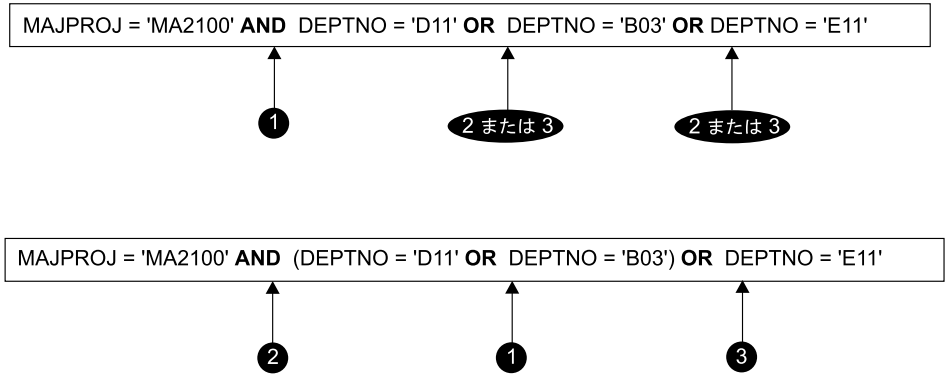


図 11. 探索条件の評価順序

SELECTIVITY *value*

SELECTIVITY 文節は、述部に指定する選択の予想パーセントを、 DB2 に知らせるときに使用します。 SELECTIVITY を指定できるのは、述部がユーザー定義述部である場合だけです。

ユーザー定義述部とは、述部が指定されている文脈の中で、ユーザー定義関数呼び出しで構成される述部のことです。これは、CREATE FUNCTION の PREDICATES 文節で指定した述部と一致します。たとえば、PREDICATES WHEN=1... で関数 foo が定義される場合、 SELECTIVITY を次のように使うことができます。

```
SELECT *
     FROM STORES
     WHERE foo(parm,parm) = 1 SELECTIVITY 0.004
```

この SELECTIVITY の値は、 0 ～ 1 の範囲の数値リテラル値でなければなりません (SQLSTATE 42615)。 SELECTIVITY を指定しない場合、省略時値は 0.01 になります (つまり、ユーザー定義述部は、表に含まれるすべての行の 1 パーセントを除いて、すべての行をフィルターではじくことになります)。この SELECTIVITY 省略時値については、SYSSTAT.FUNCTIONS 視点の SELECTIVITY 列を更新することにより、特定の関数用に変更することができます。ユーザー定義述部以外の述部に SELECTIVITY 文節を指定すると、エラーが戻されます (SQLSTATE 428E5)。

ユーザー定義関数 (UDF) はユーザー定義述部として使うことができるので、以下の場合、索引を利用するときにも使える可能性があります。

- CREATE FUNCTION ステートメントに述部の指定がある場合
- WHERE 文節で UDF が呼び出されていて、述部を指定したときの指定方法で (文法的に) 比較される場合
- 「否定」(NOT 演算子) がない場合

例

次の照会では、WHERE 文節に within UDF が指定されていて、 3 つの条件がすべて満たされているので、ユーザー定義述部であると見なされます。(within および distance UDF の詳細は、 635ページの『CREATE FUNCTION (外部スカラー)』の『例』の項を参照してください。)

```
SELECT *
   FROM customers
   WHERE within(location, :sanJose) = 1 SELECTIVITY  0.2
```

ただし、次の照会に within を指定しても、「否定」が含まれているため、索引を利用できません。これは、ユーザー定義述部とは見なされません。

```
SELECT *
   FROM customers
   WHERE NOT(within(location, :sanJose) = 1) SELECTIVITY  0.3
```

次の例では、相互が特定の距離内に含まれている顧客と店を識別します。特定の店から別の店の距離は、顧客が居住している都市の半径の範囲に基づいて計算されます。

```
SELECT *
   FROM customers, stores
   WHERE distance(customers.loc, stores.loc) < CityRadius(stores.loc) SELECTIVITY 0.02
```

上記の照会では、WHERE 文節の述部は、ユーザー定義述部であると見なされます。 CityRadius による結果は、範囲を生成する関数に対する検索引き数として使われます。

しかし、CityRadius によって生成される結果は、範囲を生成する関数として使われるため、上記のユーザー定義述部では、 stores.loc 列で定義した索引の拡張機能を十分に利用することができません。したがって、UDF は customers.loc 列で定義した索引だけを利用します。

第4章 関数

関数とは、関数名の後に 1 対の括弧で囲んだ引き数の指定を伴う演算です (引き数がない場合もあります)。

関数は、*列関数*、*スカラー関数*、*行関数*、または*表関数* として分類されます。

- 列関数の引き数は、類似する値の集合です。それは、単一値を戻し (NULL の場合もある)、式 を使用することができる SQL ステートメントに指定することができます。列関数の使用については、251ページの『列関数』で示されているように、他の制約事項があります。
- スカラー関数の引き数 (複数の場合あり) は個々のスカラー値で、タイプや意味が異なっていても構いません。スカラー関数は単一値を戻し (ヌルの場合もある)、式 を使用できる場合はいつでも、 SQL ステートメントに指定することができます。
- 行関数の引き数は、構造タイプです。これは、組み込みデータ・タイプの行を戻すもので、構造タイプの変換関数としてだけ指定できます。
- 表関数の引き数 (複数の場合あり) は個々のスカラー値で、タイプや意味が異なっていても構いません。表関数は、SQL ステートメントへ表を戻し、SELECT の FROM 文節にのみ指定することができます。表関数の使用については、428ページの『FROM 文節』で示されているように、他の制約事項があります。

232ページの表15 に、サポートされている関数を示します。「関数名」と「スキーマ」を組み合わせることによって、関数の完全修飾名となります。「説明」は、関数の動作の簡単な説明です。「入力パラメーター」には、関数呼び出し中に各引き数に予期されるデータ・タイプを示します。関数の多くには、さまざまな入力パラメーターが含まれており、異なるデータ・タイプまたは異なる数の引き数を使用することができます。スキーマ、関数名、および入力パラメーターの組み合わせが関数シグニチャーを構成します。関数シグニチャーごとに異なるタイプの値を戻すことがあります。この値については「戻り値」の欄に示してあります。

入力パラメーターのタイプについて、理解しておくべきいくつかの点があります。タイプは、特定の組み込みデータ・タイプとして指定される場合と、*any-numeric-type* (任意の数値タイプの意) のような総称的変数を使って示されている場合とがあります。特定のデータ・タイプが示されている場合は、指定されたデータ・タイプに対してのみ正しい一致が成立します。一般的な変数が

使用される場合、その変数に対応する各データ・タイプに対して正確な一致が成立します。 162ページの『関数解決』で説明したように、この違いが関数選択に影響します。

これらの関数シグニチャーのいずれかをソースとして使用してユーザー定義関数が作成されていたり (634ページの『CREATE FUNCTION』を参照)、ユーザーが独自のプログラムを使用して外部関数を作成したりすることがあるため、下記に示すもの以外にも使用可能な関数が存在する場合もあります。

注:

- 組み込み関数はデータベース・マネージャーに用意されている関数で、1つの結果値を戻します。組み込み関数は SYSIBM スキーマの一部として識別されます。このような関数の例としては、AVG などの列関数、″+″ などの演算関数、DECIMAL などのキャスト関数、 SUBSTR などのその他の関数があります。
- ユーザー定義関数は、 SYSCAT.FUNCTIONS のデータベースに登録されている (CREATE FUNCTION ステートメントを使って) 関数です。ユーザー定義関数は SYSIBM スキーマの一部ではありません。このような関数の集合の 1 つは、 SYSFUN という名前のスキーマでデータベース・マネージャーに提供されています。

表 15. サポートされている関数

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
ABS または ABSVAL	SYSFUN	引き数の絶対値を戻します。	
	SMALLINT		SMALLINT
	INTEGER		INTEGER
	BIGINT		BIGINT
	DOUBLE		DOUBLE
ACOS	SYSFUN	引き数のアークコサイン (逆余弦) の値をラジアン単位で戻します。	
	DOUBLE		DOUBLE
ASCII	SYSFUN	引き数の左端の文字の ASCII コード値を整数として戻します。	
	CHAR		INTEGER
	VARCHAR(4000)		INTEGER
	CLOB(1M)		INTEGER
ASIN	SYSFUN	引き数のアークサイン (逆正弦) の値をラジアン単位で戻します。	
	DOUBLE		DOUBLE
ATAN	SYSFUN	引き数のアークタンジェント (逆正接) の値をラジアン単位で戻します。	
	DOUBLE		DOUBLE

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
ATAN2	SYSFUN	最初と 2 番目の引き数によってそれぞれ指定される *x* 座標と、 *y* 座標に基づくアークタンジェント (逆正接) の値をラジアン単位で戻します。
	DOUBLE, DOUBLE	DOUBLE
AVG	SYSIBM	(列関数) 一連の数値の平均値を戻します。
	numeric-type [4]	*numeric-type* [1]
BIGINT	SYSIBM	数値または文字ストリングを 64 ビットで表した整数を、整数定数の形で戻します。
	numeric-type	BIGINT
	VARCHAR	BIGINT
BLOB	SYSIBM	ソース・タイプから BLOB にキャストします (長さの指定は任意)。
	string-type	BLOB
	string-type, INTEGER	BLOB
CEIL または CEILING	SYSFUN	引き数よりも大きいか等しい整数で、最小の整数を戻します。
	SMALLINT	SMALLINT
	INTEGER	INTEGER
	BIGINT	BIGINT
	DOUBLE	DOUBLE
CHAR	SYSIBM	ソース・タイプのストリング表記を戻します。
	character-type	CHAR
	character-type, INTEGER	CHAR(*integer*)
	datetime-type	CHAR
	datetime-type, *keyword* [2]	CHAR
	SMALLINT	CHAR(6)
	INTEGER	CHAR(11)
	BIGINT	CHAR(20)
	DECIMAL	CHAR(2+*precision*)
	DECIMAL, VARCHAR	CHAR(2+*precision*)
CHAR	SYSFUN	浮動小数点数の文字ストリング表記を戻します。
	DOUBLE	CHAR(24)
CHR	SYSFUN	引き数で指定される ASCII コード値を持つ文字を戻します。引き数の値は 0 ～ 255 の範囲でなければなりません。そうでない場合、戻り値はヌル値になります。
	INTEGER	CHAR(1)

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
CLOB	SYSIBM	ソース・タイプから CLOB にキャストします (長さは任意)。	
	character-type		CLOB
	character-type, INTEGER		CLOB
COALESCE [3]	SYSIBM	一連の引き数のうち、ヌル値以外の最初の引き数を戻します。	
	any-type, *any-union-compatible-type*, ...		*any-type*
CONCAT または ||	SYSIBM	2 つのストリング引き数を連結した結果を戻します。	
	string-type, *compatible-string-type*		*max string-type*
CORRELATION または CORR	SYSIBM	数値の組の集合に関する相関係数を戻します。	
	numeric-type, *numeric-type*		DOUBLE
COS	SYSFUN	引き数のコサイン (余弦) の値を戻します。引き数は、ラジアン単位の角度です。	
	DOUBLE		DOUBLE
COT	SYSFUN	引き数に対するコタンジェント (余接) の値を戻します。引き数はラジアン単位の角度です。	
	DOUBLE		DOUBLE
COUNT	SYSIBM	(列関数) 行または値の集合内の行数を戻します。	
	any-builtin-type [4]		INTEGER
COUNT_BIG	SYSIBM	(列関数) 一連の行の行または値の数、あるいは値を戻します。結果は整数の最大値より大きくなる場合があります。	
	any-builtin-type [4]		DECIMAL(31,0)
COVARIANCE または COVAR	SYSIBM	数値の組の集合に見られる共分散を戻します。	
	numeric-type, *numeric-type*		DOUBLE
DATE	SYSIBM	単一の入力値から日付を戻します。	
	DATE		DATE
	TIMESTAMP		DATE
	DOUBLE		DATE
	VARCHAR		DATE
DAY	SYSIBM	値の日の部分を戻します。	
	VARCHAR		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
DAYNAME	SYSFUN	db2start が 出された時点のロケールに基づいて、引き数の日の部分の曜日名を含む大文字小文字混合文字ストリング (たとえば、 Friday) を戻します。
	VARCHAR(26)	VARCHAR(100)
	DATE	VARCHAR(100)
	TIMESTAMP	VARCHAR(100)
DAYOFWEEK	SYSFUN	引き数の曜日を 1 〜 7 の範囲の整数値として戻します。1 は日曜を表します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYOFWEEK_ISO	SYSFUN	引き数の曜日を 1 〜 7 の範囲の整数値として戻します。1 は月曜を表します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYOFYEAR	SYSFUN	引き数の年間通算日を、1 〜 366 の範囲の整数値として戻します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYS	SYSIBM	日付の整数表記を戻します。
	VARCHAR	INTEGER
	TIMESTAMP	INTEGER
	DATE	INTEGER
DBCLOB	SYSIBM	ソース・タイプから DBCLOB にキャストします (長さは任意)。
	graphic-type	DBCLOB
	graphic-type, INTEGER	DBCLOB
DECIMAL または DEC	SYSIBM	数値の 10 進表記を戻します (精度と位取りの指定は任意)。
	numeric-type	DECIMAL
	numeric-type, INTEGER	DECIMAL
	numeric-type INTEGER, INTEGER	DECIMAL

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
DECIMAL または DEC	SYSIBM	文字ストリングの 10 進数表記を戻します (精度、位取り、および小数点文字の指定は任意)。	
	VARCHAR		DECIMAL
	VARCHAR, INTEGER		DECIMAL
	VARCHAR, INTEGER, INTEGER		DECIMAL
	VARCHAR, INTEGER, INTEGER, VARCHAR		DECIMAL
DEGREES	SYSFUN	ラジアン単位の引き数を度単位の角度に変換して戻します。	
	DOUBLE		DOUBLE
DEREF	SYSIBM	参照タイプ引き数のターゲット・タイプのインスタンスを戻します。	
	効力範囲が定義されている REF(*any-structured-type*)		*any-structured-type (入力ターゲット・タイプと同じ)*
DIFFERENCE	SYSFUN	SOUNDEX 関数によって判別される 2 つの引き数ストリングの語の音の差を戻します。 4 の値は、それらのストリングが同じ音であることを意味します。	
	VARCHAR(4000), VARCHAR(4000)		INTEGER
DIGITS	SYSIBM	数値の文字ストリング表記を戻します。	
	DECIMAL		CHAR
DLCOMMENT	SYSIBM	データ・リンク値のコメント属性を戻します。	
	DATALINK		VARCHAR(254)
DLLINKTYPE	SYSIBM	データ・リンク値のリンク・タイプ属性を戻します。	
	DATALINK		VARCHAR(4)
DLURLCOMPLETE	SYSIBM	データ・リンク値の完全 URL (アクセス・トークンを含む) を戻します。	
	DATALINK		VARCHAR
DLURLPATH	SYSIBM	データ・リンク値のパスおよびファイル名 (アクセス・トークンを含む) を戻します。	
	DATALINK		VARCHAR
DLURLPATHONLY	SYSIBM	データ・リンク値のパスおよびファイル名 (アクセス・トークンを含まない) を戻します。	
	DATALINK		VARCHAR
DLURLSCHEME	SYSIBM	データ・リンク値の URL 属性から方式を戻します。	
	DATALINK		VARCHAR
DLURLSERVER	SYSIBM	データ・リンク値の URL 属性からサーバーを戻します。	
	DATALINK		VARCHAR

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
DLVALUE	SYSIBM	データ位置付け引き数、リンク・タイプ引き数、および任意選択のコメント・ストリング引き数からデータ・リンク値を作成します。
	VARCHAR	DATALINK
	VARCHAR, VARCHAR	DATALINK
	VARCHAR, VARCHAR, VARCHAR	DATALINK
DOUBLE または DOUBLE_PRECISION	SYSIBM	数値の浮動小数点表記を戻します。
	numeric-type	DOUBLE
DOUBLE	SYSFUN	数値の文字ストリング表記に対応する浮動小数点数を戻します。*引き数*に先行ブランクや後続ブランクがあっても、それは無視されます。
	VARCHAR	DOUBLE
EVENT_MON_STATE	SYSIBM	特定のイベント・モニターの作動状態を戻します。
	VARCHAR	INTEGER
EXP	SYSFUN	引き数に対する指数関数値を戻します。
	DOUBLE	DOUBLE
FLOAT	SYSIBM	DOUBLE と同じ。
FLOOR	SYSFUN	引き数よりも小さいか等しい整数で、最大の整数値を戻します。
	SMALLINT	SMALLINT
	INTEGER	INTEGER
	BIGINT	BIGINT
	DOUBLE	DOUBLE
GENERATE_UNIQUE	SYSIBM	同じ関数の他の実行とは異なる固有なビット・データ文字ストリングを戻します。
	引き数はない	CHAR(13) FOR BIT DATA
GRAPHIC	SYSIBM	ソース・タイプから GRAPHIC にキャストします (長さは任意)。
	graphic-type	GRAPHIC
	graphic-type, INTEGER	GRAPHIC
GROUPING	SYSIBM	(列関数) グループ化集合によって生成される小計行を示すために、グループ化集合およびスーパー・グループで使用されます。戻される値は以下のとおりです。 **1** 戻される行の引き数の値はヌル値であり、この行はグループ化集合用に生成されました。生成されたこの行は、グループ化集合の小計を提供します。 **0** それ以外の場合。
	any-type	SMALLINT

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
HEX	SYSIBM	16 進数表記の値を戻します。
	any-builtin-type	VARCHAR
HOUR	SYSIBM	値の時間の部分を戻します。
	VARCHAR	INTEGER
	TIME	INTEGER
	TIMESTAMP	INTEGER
	DECIMAL	INTEGER
INSERT	SYSFUN	*argument1* の *argument2* から始まる *argument3* バイトの部分を削除し、その *argument2* から始まる *argument1* に *argument4* を挿入した結果のストリングを戻します。
	VARCHAR(4000), INTEGER, INTEGER, VARCHAR(4000)	VARCHAR(4000)
	CLOB(1M), INTEGER, INTEGER, CLOB(1M)	CLOB(1M)
	BLOB(1M), INTEGER, INTEGER, BLOB(1M)	BLOB(1M)
INTEGER または INT	SYSIBM	数値の整数表記を戻します。
	numeric-type	INTEGER
	VARCHAR	INTEGER
JULIAN_DAY	SYSFUN	紀元前 4712 年 1 月 1 日 (ユリウス暦の起点) からの経過日数を表す整数値を *argument* で指定された日付値に戻します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
LCASE または LOWER	SYSIBM	すべての文字を小文字に変換した結果のストリングを戻します。
	CHAR	CHAR
	VARCHAR	VARCHAR
LCASE	SYSFUN	すべての文字を小文字に変換した結果のストリングを戻します。 LCASE は、不変セットの文字だけを処理します。したがって、LCASE(UCASE(string)) は LCASE(string) と同じ結果を戻すとはかぎりません。
	VARCHAR(4000)	VARCHAR(4000)
	CLOB(1M)	CLOB(1M)
LEFT	SYSFUN	*argument1* の左端の *argument2* バイトからなるストリングを戻します。
	VARCHAR(4000), INTEGER	VARCHAR(4000)
	CLOB(1M), INTEGER	CLOB(1M)
	BLOB(1M), INTEGER	BLOB(1M)
LENGTH	SYSIBM	オペランドの長さをバイト数で戻します (長さを文字数で戻す 2 バイト・ストリング・タイプを除きます)。
	any-builtin-type	INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
LN	SUSFUN	引き数の自然対数を戻します (LOG と同じ)。
	DOUBLE	DOUBLE
LOCATE	SYSFUN	*argument2* の中の *argument1* の最初のオカレンスの開始位置を戻します。任意指定の 3 番目の引き数が指定されている場合、その引き数は *argument2* の中で探索を開始する文字位置を示します。 *argument1* が *argument2* 内にない場合は、値 0 が戻されます。
	VARCHAR(4000), VARCHAR(4000)	INTEGER
	VARCHAR(4000), VARCHAR(4000), INTEGER	INTEGER
	CLOB(1M), CLOB(1M)	INTEGER
	CLOB(1M), CLOB(1M), INTEGER	INTEGER
	BLOB(1M), BLOB(1M)	INTEGER
	BLOB(1M), BLOB(1M), INTEGER	INTEGER
LOG	SYSFUN	引き数の自然対数を戻します (LN と同じ)。
	DOUBLE	DOUBLE
LOG10		引き数の 10 を底とする対数を戻します。
	DOUBLE	DOUBLE
LONG_VARCHAR	SYSIBM	長ストリングを戻します。
	character-type	LONG VARCHAR
LONG_VARGRAPHIC	SYSIBM	ソース・タイプから LONG_VARGRAPHIC にキャストします。
	graphic-type	LONG VARGRAPHIC
LTRIM	SYSIBM	先行ブランクを除いた引き数の文字を戻します。
	CHAR	VARCHAR
	VARCHAR	VARCHAR
	GRAPHIC	VARGRAPHIC
	VARGRAPHIC	VARGRAPHIC
LTRIM	SYSFUN	先行ブランクを除いた引き数の文字を戻します。
	VARCHAR(4000)	VARCHAR(4000)
	CLOB(1M)	CLOB(1M)
MAX	SYSIBM	(列関数) 一連の値の最大値を戻します。
	any-builtin-type [5]	*入力タイプと同じ*
MICROSECOND	SYSIBM	値のマイクロ秒 (時間の単位) の部分を戻します。
	VARCHAR	INTEGER
	TIMESTAMP	INTEGER
	DECIMAL	INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
MIDNIGHT_SECONDS	SYSFUN	午前 0 時から *argument* で指定した時刻値までの秒数を表す 0 ～ 86 400 の範囲の整数値を戻します。	
	VARCHAR(26)		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
MIN	SYSIBM	(列関数) 一連の値の最小値を戻します。	
	any-builtin-type [5]		*入力タイプと同じ*
MINUTE	SYSIBM	値の分の部分を戻します。	
	VARCHAR		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
MOD	SYSFUN	*argument1* を *argument2* で除算して、その剰余 (モジュラス) を戻します。結果は、*argument1* が負の場合にのみ負になります。	
	SMALLINT, SMALLINT		SMALLINT
	INTEGER, INTEGER		INTEGER
	BIGINT, BIGINT		BIGINT
MONTH	SYSIBM	値の月の部分を戻します。	
	VARCHAR		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
MONTHNAME	SYSFUN	データベース開始時のロケールに基づいて、日付またはタイム・スタンプである引き数の月の部分の月名 (January など) を含む大文字小文字混合文字ストリングを戻します。	
	VARCHAR(26)		VARCHAR(100)
	DATE		VARCHAR(100)
	TIMESTAMP		VARCHAR(100)
NODENUMBER [3]	SYSIBM	行のノード番号を戻します。引き数は表内の列名です。	
	any-type		INTEGER
NULLIF [3]	SYSIBM	引き数が等しい場合はヌル値、等しくない場合は最初の引き数を戻します。	
	any-type [5], *any-comparable-type*[5]		*any-type*
PARTITION [3]	SYSIBM	行の区分化マップ索引 (0 ～ 4095) を戻します。引き数は表内の列名です。	
	any-type		INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
POSSTR	SYSIBM	あるストリングが他のストリングに含まれている位置を戻します。
	string-type, compatible-string-type	INTEGER
POWER	SYSFUN	*argument1* の *argument2* 乗の値を戻します。
	INTEGER, INTEGER	INTEGER
	BIGINT, BIGINT	BIGINT
	DOUBLE, INTEGER	DOUBLE
	DOUBLE, DOUBLE	DOUBLE
QUARTER	SYSFUN	引き数で指定された日付がどの 4 半期かを示す 1 ～ 4 の範囲の整数値を戻します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
RADIANS	SYSFUN	度単位の引き数をラジアン単位の角度に変換して戻します。
	DOUBLE	DOUBLE
RAISE_ERROR[3]	SYSIBM	SQLCA にエラーを発生させます。戻される sqlstate は *argument1* により指定します。 2 番目の引き数には、戻されるテキストが入れられます。
	VARCHAR, VARCHAR	*any-type* [6]
RAND	SYSFUN	引き数を任意選択のシード値として使用して、 0 と 1 の間のランダムな浮動小数点数値を戻します。
	引き数を必要としない	DOUBLE
	INTEGER	DOUBLE
REAL	SYSIBM	数値の単精度浮動小数点表記を戻します。
	numeric-type	REAL
REGR_AVGX	SYSIBM	診断統計を計算するために使用される数量を戻します。
	numeric-type, numeric-type	DOUBLE
REGR_AVGY	SYSIBM	診断統計を計算するために使用される数量を戻します。
	numeric-type, numeric-type	DOUBLE
REGR_COUNT	SYSIBM	回帰直線をフィッティングするために使用される非ヌル文字の対の数を戻します。
	numeric-type, numeric-type	INTEGER
REGR_INTERCEPT または REGR_ICPT	SYSIBM	回帰直線の y 切片を戻します。
	numeric-type, numeric-type	DOUBLE
REGR_R2	SYSIBM	回帰に関する判別の係数を戻します。
	numeric-type, numeric-type	DOUBLE

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
REGR_SLOPE	SYSIBM	直線の傾きを戻します。
	numeric-type, numeric-type	DOUBLE
REGR_SXX	SYSIBM	診断統計を計算するために使用される数量を戻します。
	numeric-type, numeric-type	DOUBLE
REGR_SXY	SYSIBM	診断統計を計算するために使用される数量を戻します。
	numeric-type, numeric-type	DOUBLE
REGR_SYY	SYSIBM	診断統計を計算するために使用される数量を戻します。
	numeric-type, numeric-type	DOUBLE
REPEAT	SYSFUN	*argument1* を *argument2* 回繰り返した結果の文字ストリングを戻します。
	VARCHAR(4000), INTEGER	VARCHAR(4000)
	CLOB(1M), INTEGER	CLOB(1M)
	BLOB(1M), INTEGER	BLOB(1M)
REPLACE	SYSFUN	*argument1* に存在する *argument2* をすべて *argument3* に置き換えます。
	VARCHAR(4000), VARCHAR(4000), VARCHAR(4000)	VARCHAR(4000)
	CLOB(1M), CLOB(1M), CLOB(1M)	CLOB(1M)
	BLOB(1M), BLOB(1M), BLOB(1M)	BLOB(1M)
RIGHT	SYSFUN	*argument1* の右端の *argument2* バイトからなるストリングを戻します。
	VARCHAR(4000), INTEGER	VARCHAR(4000)
	CLOB(1M), INTEGER	CLOB(1M)
	BLOB(1M), INTEGER	BLOB(1M)
ROUND	SYSFUN	最初の引き数を小数点**以下** *argument2* 桁目で丸めた結果を戻します。*argument2* が負の場合、 *argument1* の小数点の**左側**の「*argument2* の絶対値」桁未満が切り捨てられます。
	INTEGER, INTEGER	INTEGER
	BIGINT, INTEGER	BIGINT
	DOUBLE, INTEGER	DOUBLE
RTRIM	SYSIBM	後続ブランクを除いた引き数の文字を戻します。
	CHAR	VARCHAR
	VARCHAR	VARCHAR
	GRAPHIC	VARGRAPHIC
	VARGRAPHIC	VARGRAPHIC
RTRIM	SYSFUN	後続ブランクを除いた引き数の文字を戻します。
	VARCHAR(4000)	VARCHAR(4000)
	CLOB(1M)	CLOB(1M)

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
SECOND	SYSIBM	値の秒 (時間の単位) の部分を戻します。
	VARCHAR	INTEGER
	TIME	INTEGER
	TIMESTAMP	INTEGER
	DECIMAL	INTEGER
SIGN	SYSFUN	引き数の符号の標識を戻します。引き数が負の場合は、-1 が戻されます。引き数がゼロの場合は、0 が戻されます。引き数が正の場合には、1 が戻されます。
	SMALLINT	SMALLINT
	INTEGER	INTEGER
	BIGINT	BIGINT
	DOUBLE	DOUBLE
SIN	SYSFUN	引き数のサイン (正弦) の値を戻します。引き数は、ラジアン単位の角度です。
	DOUBLE	DOUBLE
SMALLINT	SYSIBM	数値の短精度整数表記を戻します。
	numeric-type	SMALLINT
	VARCHAR	SMALLINT
SOUNDEX	SYSFUN	引き数の語の音を表す 4 文字コードを戻します。この結果は、他のストリングの音との比較に使用することができます。 DIFFERENCE も参照。
	VARCHAR(4000)	CHAR(4)
SPACE	SYSFUN	*argument1* 個のブランクからなる文字ストリングを戻します。
	INTEGER	VARCHAR(4000)
SQLCACHE_SNAPSHOT	SYSFUN	DB2 動的 SQL ステートメント・キャッシュのスナップショットの表を戻します。
	417ページの『SQLCACHE_SNAPSHOT』を参照。	
SQRT	SYSFUN	引き数の平方根を戻します。
	DOUBLE	DOUBLE
STDDEV	SYSIBM	(列関数) 一連の数値の標準偏差を戻します。
	DOUBLE	DOUBLE
SUBSTR	SYSIBM	*argument1* の位置 *argument2* から始まる *argument3* 文字のサブストリングを戻します。 *argument3* を指定しない場合は、ストリングの残りすべてが戻されます。
	string-type, INTEGER	*string-type*
	string-type, INTEGER, INTEGER	*string-type*

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
SUM	SYSIBM	(列関数) 一連の数値の合計値を戻します。	
	numeric-type [4]		*max-numeric-type* [1]
TABLE_NAME	SYSIBM	*argument1* で指定するオブジェクト名、および *argument2* で指定する任意選択のスキーマ名に基づく表または視点の非修飾名を戻します。この関数は、別名の解決に使用されます。	
	VARCHAR		VARCHAR(128)
	VARCHAR, VARCHAR		VARCHAR(128)
TABLE_SCHEMA	SYSIBM	*argument1* で指定するオブジェクト名、および *argument2* で指定する任意選択のスキーマ名によって指定される、 2 つの部分からなる表名または視点名のスキーマ名の部分を戻します。この関数は、別名の解決に使用されます。	
	VARCHAR		VARCHAR(128)
	VARCHAR, VARCHAR		VARCHAR(128)
TAN	SYSFUN	引き数のタンジェント (正接) の値を戻します。引き数は、ラジアン単位の角度です。	
	DOUBLE		DOUBLE
TIME	SYSIBM	値から時刻を戻します。	
	TIME		TIME
	TIMESTAMP		TIME
	VARCHAR		TIME
TIMESTAMP	SYSIBM	値または一対の値からタイム・スタンプを戻します。	
	TIMESTAMP		TIMESTAMP
	VARCHAR		TIMESTAMP
	VARCHAR, VARCHAR		TIMESTAMP
	VARCHAR, TIME		TIMESTAMP
	DATE, VARCHAR		TIMESTAMP
	DATE, TIME		TIMESTAMP
TIMESTAMP_ISO	SYSFUN	日付、時刻、またはタイム・スタンプの引き数に基づいてタイム・スタンプ値を戻します。引き数が日付の場合は、時間要素のすべてにゼロが入れられます。引き数が時刻の場合、日付要素には CURRENT DATE の値、時刻の小数要素にはゼロが入れられます。	
	DATE		TIMESTAMP
	TIME		TIMESTAMP
	TIMESTAMP		TIMESTAMP
	VARCHAR(26)		TIMESTAMP

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
TIMESTAMPDIFF	SYSFUN	2 つのタイム・スタンプの間の差に従って、タイプ *argument1* での間隔数の見積もりが戻されます。 2 番目の引き数は、2 つのタイム・スタンプ・タイプの減算を行い、その結果を CHAR に変換した結果です。間隔 (*argument1*) の有効な値は次のとおりです。 **1** 秒の小数部 **2** 秒数 **4** 分数 **8** 時間数 **16** 日数 **32** 週数 **64** 月数 **128** 四半期数 **256** 年数
	INTEGER, CHAR(22)	INTEGER
TRANSLATE	SYSIBM	1 つまたは複数の文字を他の文字に変換したストリングを戻します。
	CHAR	CHAR
	VARCHAR	VARCHAR
	CHAR, VARCHAR, VARCHAR	CHAR
	VARCHAR, VARCHAR, VARCHAR	VARCHAR
	CHAR, VARCHAR, VARCHAR, VARCHAR	CHAR
	VARCHAR, VARCHAR, VARCHAR, VARCHAR	VARCHAR
	GRAPHIC, VARGRAPHIC, VARGRAPHIC	GRAPHIC
	VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	VARGRAPHIC
	GRAPHIC, VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	GRAPHIC
	VARGRAPHIC, VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	VARGRAPHIC
TRUNC または TRUNCATE	SYSFUN	*argument1* の小数点**以下** *argument2* 桁までを切り捨てた結果を戻します。 *argument2* が負の場合、 *argument1* の小数点の**左側**の「*argument2* の絶対値」桁未満が切り捨てられます。
	INTEGER, INTEGER	INTEGER
	BIGINT, INTEGER	BIGINT
	DOUBLE, INTEGER	DOUBLE
TYPE_ID [3]	SYSIBM	引き数の動的データ・タイプの内部データ・タイプ識別子を戻します。この関数の結果はデータベース間で移行できないことに注意してください。
	any-structured-type	INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
TYPE_NAME [3]	SYSIBM	引き数の動的データ・タイプの非修飾名を戻します。
	any-structured-type	VARCHAR(18)
TYPE_SCHEMA [3]	SYSIBM	引き数の動的タイプのスキーマ名を戻します。
	any-structured-type	VARCHAR(128)
UCASE または UPPER	SYSIBM	すべての文字を大文字に変換したストリングを戻します。
	CHAR	CHAR
	VARCHAR	VARCHAR
UCASE	SYSFUN	すべての文字を大文字に変換したストリングを戻します。
	VARCHAR	VARCHAR
VALUE [3]	SYSIBM	COALESCE と同じ。
VARCHAR	SYSIBM	最初の引き数の VARCHAR 表記を戻します。 2 番目の引き数を指定した場合、その値は結果の長さを指定します。
	character-type	VARCHAR
	character-type, INTEGER	VARCHAR
	datetime-type	VARCHAR
VARGRAPHIC	SYSIBM	最初の引き数の VARGRAPHIC 表記を戻します。 2 番目の引き数を指定した場合、その値は結果の長さを指定します。
	graphic-type	VARGRAPHIC
	graphic-type, INTEGER	VARGRAPHIC
	VARCHAR	VARGRAPHIC
VARIANCE または VAR	SYSIBM	(列関数) 一連の数値の差異を戻します。
	DOUBLE	DOUBLE
WEEK	SYSFUN	引き数の年間通算週番号を、1 ～ 54 の範囲の整数値として戻します。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
WEEK_ISO	SYSFUN	引き数の年間通算週番号を、1 ～ 53 の範囲の整数値として戻します。週の最初の日は月曜日です。第 1 週は、年の第 1 週目で木曜日を含みます。
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明
	入力パラメーター	戻り値のタイプ
YEAR	SYSIBM	値の年の部分を戻します。
	VARCHAR	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
	DECIMAL	INTEGER
“+”	SYSIBM	2 つの数値オペランドを加算します。
	numeric-type, *numeric-type*	*max numeric-type*
“+”	SYSIBM	単項加算演算子。
	numeric-type	*numeric-type*
“+”	SYSIBM	日付 / 時刻加算演算子
	DATE, DECIMAL(8,0)	DATE
	TIME, DECIMAL(6,0)	TIME
	TIMESTAMP, DECIMAL(20,6)	TIMESTAMP
	DECIMAL(8,0), DATE	DATE
	DECIMAL(6,0), TIME	TIME
	DECIMAL(20,6), TIMESTAMP	TIMESTAMP
	datetime-type, DOUBLE, *labeled-duration-code*	*datetime-type*
“-”	SYSIBM	2 つの数値オペランドを減算します。
	numeric-type, *numeric-type*	*max numeric-type*
“-”	SYSIBM	単項減算演算子。
	numeric-type	*numeric-type* [1]
“-”	SYSIBM	日付 / 時刻減算演算子。
	DATE, DATE	DECIMAL(8,0)
	TIME, TIME	DECIMAL(6,0)
	TIMESTAMP, TIMESTAMP	DECIMAL(20,6)
	DATE, VARCHAR	DECIMAL(8,0)
	TIME, VARCHAR	DECIMAL(6,0)
	TIMESTAMP, VARCHAR	DECIMAL(20,6)
	VARCHAR, DATE	DECIMAL(8,0)
	VARCHAR, TIME	DECIMAL(6,0)
	VARCHAR, TIMESTAMP	DECIMAL(20,6)
	DATE, DECIMAL(8,0)	DATE
	TIME, DECIMAL(6,0)	TIME
	TIMESTAMP, DECIMAL(20,6)	TIMESTAMP
	datetime-type, DOUBLE, *labeled-duration-code*	*datetime-type*

表 15. サポートされている関数 (続き)

関数名	スキーマ	説明	
	入力パラメーター		戻り値のタイプ
“*”	SYSIBM	2 つの数値オペランドを乗算します。	
	numeric-type, numeric-type		*max numeric-type*
“/”	SYSIBM	2 つの数値オペランドを除算します。	
	numeric-type, numeric-type		*max numeric-type*
“‖”	SYSIBM	CONCAT と同じ。	

注

- 長さで修飾されていないストリング・データ・タイプへの参照は、最大長のデータ・タイプをサポートするものとみなされます。
- 精度と位取りを指定していない DECIMAL データ・タイプへの参照は、サポートされているすべての精度と位取りが可能であるとみなされます。

凡例

any-builtin-type 特殊タイプ以外の任意のデータ・タイプ。

any-type データベースに定義されている任意のタイプ。

any-structured-type
データベースに定義されているユーザー定義の構造タイプ。

any-comparable-type
106ページの『割り当てと比較』 の定義に従って、他の引き数タイプと比較可能なタイプ。

any-union-compatible-type
121ページの『結果のデータ・タイプに関する規則』の定義に従って、他の引き数タイプと互換性のあるタイプ。

character-type 文字ストリング・タイプ CHAR、VARCHAR、LONG VARCHAR、CLOB のいずれか。

compatible-string-type
他の引き数と同じグループのストリング・タイプ (たとえば、引き数の一方が *character-type* なら、もう一方の引き数も *character-type* でなければなりません)。

datetime-type 日付 / 時刻タイプ DATE、TIME、TIMESTAMP のいずれか。

graphic-type 2 バイト文字ストリング・タイプ GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC、 DBCLOB のいずれか。

labeled-duration-code
これは、タイプとしては SMALLINT です。関数が、プラスまたはマイナス演算子の挿入形式を使用して呼び出される場合は、 184ページの『ラベル付き期間』で定義されているラベル付き期間を使用することができます。プラスまたはマイナスの演算子文字を名前として使用しないソース関数の場合、関数を呼び出すときに labeled-duration-code 引き数に次の値を使用する必要があります。

1 YEAR または YEARS
2 MONTH または MONTHS
3 DAY または DAYS
4 HOUR または HOURS
5 MINUTE または MINUTES
6 SECOND または SECONDS
7 MICROSECOND または MICROSECONDS

LOB-type ラージ・オブジェクト・タイプ BLOB、CLOB、DBCLOB のいずれか。

max-numeric-type
引き数の最大数値タイプ。最大値は右端の *numeric-type* として定義されます。

max-string-type 引き数の最大文字列タイプ。最大値は右端の *character-type* または *graphic-type* として定義されます。引き数が BLOB の場合、*max-string-type* は BLOB です。

numeric-type 数値タイプ SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE のいずれか。

string-type *character-type*、 *graphic-type*、または BLOB のいずれか。

表の脚注

1. 入力パラメーターが SMALLINT の場合、結果タイプは INTEGER です。入力パラメーターが REAL の場合、結果タイプは DOUBLE です。
2. 使用できる keyword (キーワード) は、ISO、USA、EUR、JIS、および LOCAL です。この関数シグニチャーは、ソース関数からの派生関数としてはサポートされていません。
3. この関数は、ソース関数としては使用できません。
4. 第 1 パラメーターの前にキーワード ALL または DISTINCT を使用できます。 DISTINCT を指定した場合、ユーザー定義構造タイプ、長ストリング・タイプまたは DATALINK タイプは使用できません。
5. ユーザー定義構造タイプ、長ストリング・タイプまたは DATALINK タイプは使用できません。
6. RAISE_ERROR によって戻されるタイプは、それを使う文脈によって異なります。特定のタイプにキャストしない場合、 RAISE_ERROR は、CASE 式内でのその呼び出しに適したタイプを戻します。

列関数

列関数の引き数は、1 つの式から派生する一連の値の集合です。式に列を含めることができますが、*スカラー全選択*、または他の列関数を含めることはできません (SQLSTATE 42607)。その集合の効力範囲は、グループ、または 421ページの『第5章 照会』で説明されている中間結果表です。

GROUP BY 文節が照会に指定され、 FROM、WHERE、GROUP BY および HAVING 文節からの中間結果が空集合の場合、列関数は適用されず、照会の結果は空集合であり、 SQLCODE が +100 に設定され、SQLSTATE が '02000' に設定されます。

GROUP BY 文節が照会の中に指定されておらず、FROM、WHERE、および HAVING の文節の中間結果が空集合の場合、列関数は空集合に適用されます。

たとえば、次の SELECT ステートメントの結果は、部門 D01 の社員に対して重複しない JOBCODE 値の数となります。

```
SELECT COUNT(DISTINCT JOBCODE)
  FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT = 'D01'
```

キーワード DISTINCT は、関数の引き数ではなく、関数が適用される以前に実行される演算の指定と見なされます。 DISTINCT を指定すると、重複する値は除去されます。暗黙のうちにまたは明示的に ALL を指定すると、重複する値は削除されません。

列関数で、たとえば次のようにして、式を使用することができます。

```
SELECT MAX(BONUS + 1000)
  INTO :TOP_SALESREP_BONUS
  FROM EMPLOYEE
  WHERE COMM > 5000
```

以下に示す列関数は、SYSIBM スキーマのものであり、スキーマ名で修飾できます (たとえば、SYSIBM.COUNT(*))。

AVG

スキーマは SYSIBM です。

AVG 関数は、一連の数値の平均値を戻します。

引き数の値は数値でなければならず、その合計は、結果のデータ・タイプの範囲内になければなりません。結果はヌル値の場合もあります。

結果のデータ・タイプは、引き数値のデータ・タイプと同じです。ただし、以下の場合を除きます。

- 引き数値が短精度整数 (small integer) の場合、結果は長精度整数 (large integer) になります。
- 引き数値が単精度浮動小数点の場合、結果は倍精度浮動小数点になります。

引き数値のデータ・タイプが精度 *p* で位取りが *s* の 10 進数の場合、結果の精度は 31、位取りは 31-*p*+*s* となります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。 DISTINCT を指定すると、重複する値は除去されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の平均値になります。

結果のデータ・タイプが整数であれば、平均値の小数部分は失われます。

例:

- PROJECT 表を使用して、部門 (DEPTNO) 'D11' におけるプロジェクトの平均スタッフ数 (PRSTAFF) を、ホスト変数 AVERAGE(decimal(5,2)) に設定します。

  ```
  SELECT AVG(PRSTAFF)
    INTO :AVERAGE
    FROM PROJECT
    WHERE DEPTNO = 'D11'
  ```

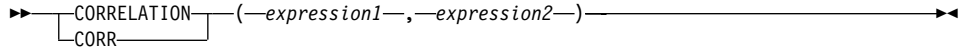
  サンプル表を使用してこの例を実行すると、結果として AVERAGE には、4.25 (つまり、17/4) が設定されます。

- PROJECT 表を使用して、ホスト変数 ANY_CALC (decimal(5,2)) を、部門 (DEPTNO) 'D11' の中でのプロジェクトのスタッフ・レベル (PRSTAFF) の固有の値についての平均値に設定します。

```
SELECT AVG(DISTINCT PRSTAFF)
  INTO :ANY_CALC
  FROM PROJECT
  WHERE DEPTNO = 'D11'
```

  サンプル表を使用してこの例を実行すると、結果として ANY_CALC は 4.66 (つまり 14/3) に設定されます。

CORRELATION



スキーマは SYSIBM です。

CORRELATION 関数は、数値の組の集合に関する相関係数を戻します。

引き数には、数値を指定しなければなりません。

結果のデータ・タイプは、倍精度の浮動小数点です。結果はヌル値の場合もあります。値がヌル値でない場合、結果は 0 ～ 1 になります。

この関数は、引き数の値から導出された対の集合 (*expression1*, *expression2*) から、 *expression1* または *expression2* のどちらかがヌル値の対を除外したものに対して適用されます。

この関数が空集合に適用された場合や、 STDDEV(*expression1*) または STDDEV(*expression2*) のどちらかがゼロに等しい場合、結果はヌル値になります。それ以外の場合、結果は集合内にある値の組の相関係数になります。結果は、以下の式と等しくなります。

```
COVARIANCE(expression1,expression2)/(STDDEV(expression1)*STDDEV(expression2))
```

値を集計する順序は定義されていませんが、すべての中間結果は結果のデータ・タイプの範囲内になければなりません。

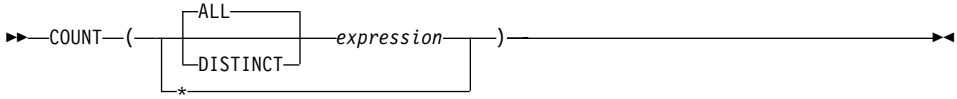
例:

- EMPLOYEE 表を使用して、ホスト変数 CORRLN (倍精度の浮動小数点数) を、部署 (WORKDEPT) 'A00' の従業員の給与と賞与の間に見られる相関に設定します。

  ```
  SELECT CORRELATION(SALARY, BONUS)
    INTO :CORRLN
    FROM EMPLOYEE
    WHERE WORKDEPT = 'A00'
  ```

  サンプル表を使用した場合、CORRLN はおよそ 9.99853953399538E-001 に設定されます。

COUNT



スキーマは SYSIBM です。

COUNT 関数は、行の集合または値の集合の中に含まれる行または値の数を戻します。

DISTINCT を使用する場合、*expression* の結果データ・タイプの文字ストリングとしての長さは 255 を超えてはならず、漢字ストリング列としての長さは 127 を超えてはなりません。 *expression* のデータ・タイプは、LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK、これらのタイプの特殊タイプ、または構造タイプとすることはできません (SQLSTATE 42907)。

この関数の結果は長精度整数 (large integer) です。結果がヌル値になることはありません。

COUNT(*) の引き数は行の集合になります。 結果は、集合の行の数です。NULL 値 (ヌル値) しか含まれない行は、カウントに含められます。

COUNT(DISTINCT *expression*) の引き数は、値の集合です。この関数は、引き数の値からヌル値と重複値を除いた値の集合に対して適用されます。結果は、その集合の中のヌルでない異なる値の数です。

COUNT(*expression*) または COUNT(ALL *expression*) の引き数は、値の集合です。この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。結果は、その集合の中のヌルでない値の数です (重複値も含む)。

例:

- EMPLOYEE 表を使用して、 SEX 列の値が 'F' である行数をホスト変数 FEMALE(int) に設定します。

```
SELECT COUNT(*)
  INTO :FEMALE
  FROM EMPLOYEE
  WHERE SEX = 'F'
```

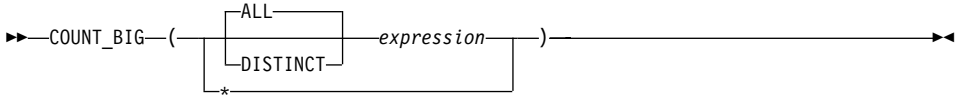
  サンプル表を使用してこの例を実行すると、結果として FEMALE に 13 が設定されます。

- EMPLOYEE 表を使用して、女性社員が少なくとも 1 人所属している部門 (WORKDEPT) の数を、ホスト変数 FEMALE_IN_DEPT (int) に設定します。

```
SELECT COUNT(DISTINCT WORKDEPT)
  INTO :FEMALE_IN_DEPT
  FROM EMPLOYEE
  WHERE SEX = 'F'
```

  サンプル表を使用した場合、結果として FEMALE は 5 に設定されます。(少なくとも 1 人の女性がいる部門は、A00、C01、D11、D21、および E11 です。)

COUNT_BIG



スキーマは SYSIBM です。

COUNT_BIG 関数は、一連の行または値の行の数または値の数を戻します。これは COUNT とほぼ同じですが、結果が整数の最大値より大きくなる場合がある点が異なります。

DISTINCT を使用する場合、*expression* の結果データ・タイプの文字ストリングとしての長さは 255 を超えてはならず、漢字ストリング列としての長さは 127 を超えてはなりません。 *expression* の結果データ・タイプは、LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK、これらのタイプの特殊タイプ、または構造タイプとすることはできません (SQLSTATE 42907)。

関数の結果は、精度 31 および位取り 0 の 10 進数です。結果がヌル値になることはありません。

COUNT_BIG(*) の引き数は、行の集合です。結果は、集合の行の数です。NULL 値しか含まれない行は、カウントに含められます。

COUNT_BIG(DISTINCT *expression*) の引き数は、値の集合です。この関数は、引き数の値からヌル値と重複値を除いた値の集合に対して適用されます。結果は、その集合の中のヌル以外の異なる値の数です。

COUNT_BIG(*expression*) または COUNT_BIG(ALL *expression*) の引き数は、値の集合です。この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。結果は、その集合の中のヌルでない値の数です (重複値も含む)。

例:

- COUNT の例を参照して、COUNT を COUNT_BIG に置き換えてください。結果のデータ・タイプ以外は、同じ結果になります。
- アプリケーションによっては、COUNT を使用する必要があるにもかかわらず、最大整数よりも大きい値をサポートする必要がある場合があります。これは、ソースから派生されたユーザー定義関数を使用し、 SQL パスを設定して行うことができます。以下の一連のステートメントは、 COUNT_BIG

に基づいて COUNT(*) をサポートするソースからの派生関数を作成して、精度 15 の 10 進数を戻す方法を示しています。 COUNT_BIG に基づくソースからの派生関数が、ここに示されている照会などの後続のステートメントで使用されるように、 SQL パスが設定されます。

```
CREATE FUNCTION RICK.COUNT() RETURNS DECIMAL(15,0)
       SOURCE SYSIBM.COUNT_BIG();
SET CURRENT FUNCTION PATH RICK, SYSTEM PATH;
SELECT COUNT(*) FROM EMPLOYEE;
```

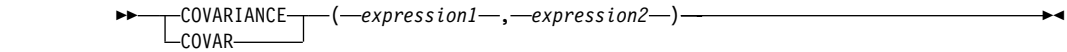
ソースからの派生関数は、 COUNT(*) をサポートするパラメーターを指定せずに定義されていることに注意してください。これが有効なのは、関数 COUNT を指定し、関数の使用時に関数をスキーマ名で修飾しない場合だけです。 COUNT 以外の名前を使用して COUNT(*) と同じ結果を得るためには、パラメーターを指定せずに関数を呼び出します。したがって、RICK.COUNT が RICK.MYCOUNT として定義されている場合、照会は以下のように書く必要があります。

```
SELECT MYCOUNT() FROM EMPLOYEE;
```

特定の列についてカウントを取る場合、ソースからの派生関数はその列のタイプを指定する必要があります。以下のステートメントによって作成されたソースからの派生関数は、 CHAR 列を引き数として取り、COUNT_BIG を使用してカウントを実行します。

```
CREATE FUNCTION RICK.COUNT(CHAR()) RETURNS DOUBLE
       SOURCE SYSIBM.COUNT_BIG(CHAR());
SELECT COUNT(DISTINCT WORKDEPT) FROM EMPLOYEE;
```

COVARIANCE



スキーマは SYSIBM です。

COVARIANCE 関数は、数値の組の集合に関する (集団) 共分散を戻します。

引き数には、数値を指定しなければなりません。

結果のデータ・タイプは、倍精度の浮動小数点です。結果はヌル値の場合もあります。

この関数は、引き数の値から導出された対の集合 (*expression1*, *expression2*) から、 *expression1* または *expression2* のどちらかがヌル値の対を除外したものに対して適用されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の値の組の共分散になります。結果は、次のようにして割り出されます。

1. avgexp1 を AVG(*expression1*) の結果に、 avgexp2 を AVG(*expression2*) の結果にします。
2. COVARIANCE(*expression1*, *expression2*) の結果は、 AVG(*expression1* - avgexp1) * (*expression2* - avgexp2) になります。

値を集計する順序は定義されていませんが、すべての中間結果は結果のデータ・タイプの範囲内になければなりません。

例:

- EMPLOYEE 表を使用して、ホスト変数 COVARNCE (倍精度の浮動小数点) を、部署 (WORKDEPT) 'A00' の従業員の給与と賞与の間に見られる共分散に設定します。

```
SELECT COVARIANCE(SALARY, BONUS)
  INTO :COVARNCE
  FROM EMPLOYEE
  WHERE WORKDEPT = 'A00'
```

  サンプル表を使用した場合、COVARNCE はおよそ 1.68888888888889E+006 に設定されます。

GROUPING

スキーマは SYSIBM です。

GROUPING 関数は、グループ化集合およびスーパー・グループ (詳細については、 437ページの『GROUP BY 文節』を参照) に関連して使用され、 GROUP BY 応答セットで戻された行が、 *expression* によって表される列を除外し、グループ化集合によって生成された行であるか否かを示す値を戻します。

引き数はどのようなタイプでも構いませんが、 GROUP BY 文節の項目でなければなりません。

この関数の結果は短精度整数 (small integer) です。結果は以下のいずれかの値に設定されます。

1 戻された行の *expression* の値はヌル値であり、しかもその行はスーパー・グループによって生成されました。生成されたその行は、GROUP BY 式の小計の値を求めるのに使用することができます。

0 値は上記以外です。

例:

以下の照会を行うと、

```
SELECT SALES_DATE,
       SALES_PERSON,
       SUM(SALES) AS UNITS_SOLD,
       GROUPING(SALES_DATE) AS DATE_GROUP,
       GROUPING(SALES_PERSON) AS SALES_GROUP
  FROM SALES
  GROUP BY CUBE (SALES_DATE, SALES_PERSON)
  ORDER BY SALES_DATE, SALES_PERSON
```

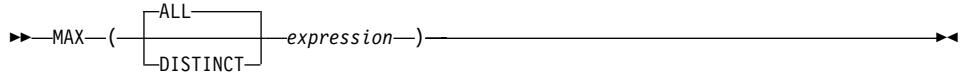
結果は次のようになります。

```
SALES_DATE SALES_PERSON    UNITS_SOLD  DATE_GROUP  SALES_GROUP
---------- --------------- ----------- ----------- -----------
12/31/1995 GOUNOT                    1           0           0
12/31/1995 LEE                       6           0           0
12/31/1995 LUCCHESSI                 1           0           0
12/31/1995 -                         8           0           1
03/29/1996 GOUNOT                   11           0           0
03/29/1996 LEE                      12           0           0
03/29/1996 LUCCHESSI                 4           0           0
03/29/1996 -                        27           0           1
```

```
03/30/1996 GOUNOT                21          0          0
03/30/1996 LEE                   21          0          0
03/30/1996 LUCCHESSI              4          0          0
03/30/1996 -                     46          0          1
03/31/1996 GOUNOT                 3          0          0
03/31/1996 LEE                   27          0          0
03/31/1996 LUCCHESSI              1          0          0
03/31/1996 -                     31          0          1
04/01/1996 GOUNOT                14          0          0
04/01/1996 LEE                   25          0          0
04/01/1996 LUCCHESSI              4          0          0
04/01/1996 -                     43          0          1
-          GOUNOT                50          1          0
-          LEE                   91          1          0
-          LUCCHESSI             14          1          0
-          -                    155          1          1
```

アプリケーションは、DATE_GROUP の値が 0 であり、 SALES_GROUP の値が 1 であることによって、 SALES_DATE の小計行を認識することができます。 SALES_PERSON の小計行は、DATE_GROUP の値が 1 であり、SALES_GROUP の値が 0 であることによって認識することができます。合計行は、DATE_GROUP と SALES_GROUP の両方の値が 1 であることによって認識することができます。

MAX



スキーマは SYSIBM です。

MAX 関数は、値の集合の最大値を戻します。

引き数値は、長ストリングまたは DATALINK 以外の任意の組み込みタイプにすることができます。

DISTINCT を使用する場合、*expression* の結果データ・タイプの文字ストリングとしての長さは 255 を超えてはならず、漢字ストリング列としての長さは 127 を超えてはなりません。 *expression* の結果データ・タイプは、LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK、これらのタイプの特殊タイプ、または構造タイプとすることはできません (SQLSTATE 42907)。

結果のデータ・タイプ、長さ、およびコード・ページは、引き数値のデータ・タイプ、長さ、およびコード・ページと同じです。結果は、派生値とみなされ、ヌル値の場合もあります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の中の最大値になります。

DISTINCT を指定しても結果に影響しないので、指定しないようにしてください。これは、他の関係システムとの互換性の目的で組み込まれています。

例:

- EMPLOYEE 表を使用して、月額の給与の最高額 (SALARY/12) の値をホスト変数 MAX_SALARY (decimal(7,2)) に設定します。

```
SELECT MAX(SALARY) / 12
  INTO :MAX_SALARY
  FROM EMPLOYEE
```

  サンプル表を使用すると、結果として MAX_SALARY は 4395.83 に設定されます。

- PROJECT 表を使用して、照合順序の最後にくるプロジェクト名 (PROJNAME) をホスト変数 LAST_PROJ (char(24)) に設定します。

```
SELECT MAX(PROJNAME)
  INTO :LAST_PROJ
  FROM PROJECT
```

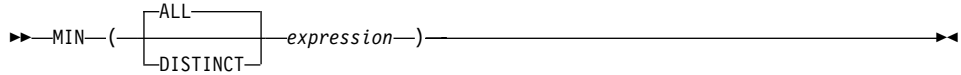
  サンプル表を使用すると、結果として LAST_PROJ は ’WELD LINE PLANNING’ に設定されます。

- 前の例と同様に、プロジェクト名にホスト変数 PROJSUPP を連結した場合に照合順序で最後になるプロジェクト名を、ホスト変数 LAST_PROJ (char(40)) に設定します。 PROJSUPP は '_Support' であり、データ・タイプは char(8) です。

```
SELECT MAX(PROJNAME CONCAT PROJSUPP)
  INTO :LAST_PROJ
  FROM PROJECT
```

  サンプル表を使用した場合、結果として LAST_PROJ は 'WELD LINE PLANNING_SUPPORT' に設定されます。

MIN



スキーマは SYSIBM です。

MIN 関数は、値の集合の最小値を戻します。

引き数値は、長ストリングまたは DATALINK 以外の任意の組み込みタイプにすることができます。

DISTINCT を使用する場合、*expression* の結果データ・タイプの文字ストリングとしての長さは 255 を超えてはならず、漢字ストリング列としての長さは 127 を超えてはなりません。 *expression* の結果データ・タイプは、LONG VARCHAR、LONG VARGRAPHIC、
BLOB、CLOB、DBCLOB、DATALINK、これらのタイプの特殊タイプ、または構造タイプとすることはできません (SQLSTATE 42907)。

結果のデータ・タイプ、長さ、およびコード・ページは、引き数値のデータ・タイプ、長さ、およびコード・ページと同じです。結果は、派生値とみなされ、ヌル値の場合もあります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の最小値になります。

DISTINCT を指定しても結果に影響しないので、指定しないようにしてください。これは、他の関係システムとの互換性の目的で組み込まれています。

例:

- EMPLOYEE 表を使用して、部門 (WORKDEPT) 'D11' の社員に対する最高と最低歩合 (COMM) の差をホスト変数 COMM_SPREAD (decimal(7,2)) に設定します。

```
SELECT MAX(COMM) - MIN(COMM)
  INTO :COMM_SPREAD
  FROM EMPLOYEE
  WHERE WORKDEPT = 'D11'
```

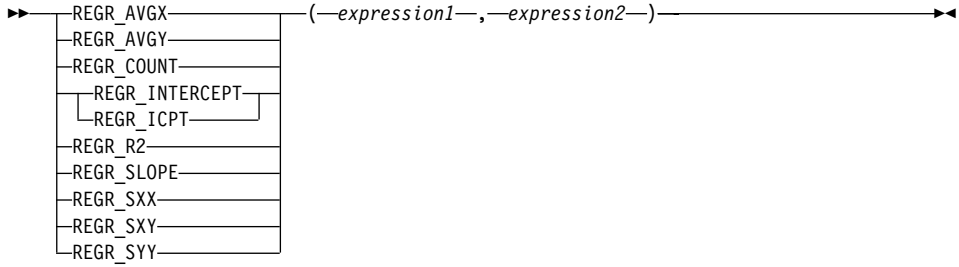
サンプル表を使用すると、結果として COMM_SPREAD は 1118 (つまり 2580 - 1462) に設定されます。

- PROJECT 表を使用して、最初に完了するように予定されたプロジェクトの予定終了日 (PRENDATE) をホスト変数 (FIRST_FINISHED (char(10))) に設定します。

```
SELECT MIN(PRENDATE)
  INTO :FIRST_FINISHED
  FROM PROJECT
```

サンプル表を使用すると、結果として FIRST_FINISHED は '1982-09-15' に設定されます。

REGRESSION 関数



スキーマは SYSIBM です。

回帰関数は、通常の最小 2 乗法による回帰直線 (形式 y = a * x + b) を数値の組の集合に当てはめることをサポートします。各対の最初の要素 (*expression1*) は、従属変数の値 (つまり、″y 値″) として解釈されます。各対の 2 番目の要素 (*expression2*) は、独立変数の値 (つまり、″x 値″) として解釈されます。

関数 REGR_COUNT は、回帰直線をフィッティングするために使用された非ヌル文字の対の数を戻します (下記参照)。

関数 REGR_INTERCEPT (省略形は REGR_ICPT) は、回帰直線の y 切片 (前述の式では ″b″) を戻します。

関数 REGR_R2 は、回帰に関する判別の係数 (″R 2 乗″ または ″適合度″ ともいう) を戻します。

関数 REGR_SLOPE は、直線の傾き (前述の式ではパラメーター ″a″) を戻します。

関数 REGR_AVGX、REGR_AVGY、REGR_SXX、REGR_SYY、および REGR_SXY が戻す数量データを使用すれば、回帰モデルの質と統計としての有効性を評価するために必要な各種の診断統計を計算できます (下記参照)。

引き数には、数値を指定しなければなりません。

REGR_COUNT の結果のデータ・タイプは整数です。その他の関数については、結果のデータ・タイプは倍精度の浮動小数点です。結果はヌル値の場合もあります。値がヌル値でない場合、REGR_R2 の結果は 0 ～ 1 になります。REGR_SXX と REGR_SYY の結果はどちらも非負になります。

各関数は、引き数の値から導出された対の集合 (*expression1, expression2*) から、 *expression1* または *expression2* のどちらかがヌル値の対を除外したものに対して適用されます。

集合が空ではなく、かつ VARIANCE(*expression2*) が正の場合、REGR_COUNT は非ヌルの対の数を集合に戻し、その他の関数は次のように定義された結果を戻します。

```
REGR_SLOPE(expression1,expression2) =
 COVARIANCE(expression1,expression2)/VARIANCE(expression2)

REGR_INTERCEPT(expression1, expression2) =
  AVG(expression1) - REGR_SLOPE(expression1, expression2) * AVG(expression2)

REGR_R2(expression1, expression2) =
  POWER(CORRELATION(expression1, expression2), 2) if VARIANCE(expression1)>0
  REGR_R2(expression1, expression2) = 1 if VARIANCE(expression1)=0

REGR_AVGX(expression1, expression2) = AVG(expression2)

REGR_AVGY(expression1, expression2) = AVG(expression1)

REGR_SXX(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * VARIANCE(expression2)

REGR_SYY(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * VARIANCE(expression1)

REGR_SXY(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * COVARIANCE(expression1, expression2)
```

集合が空ではなく、かつ VARIANCE(*expression2*) がゼロに等しい場合、回帰直線は無限の傾きになるか、未定義の状態になります。その場合、関数 REGR_SLOPE、REGR_INTERCEPT、および REGR_R2 はそれぞれヌル値を戻し、その他の関数は上記のように定義された戻り値を戻します。集合が空の場合は、REGR_COUNT はゼロを戻し、その他の関数はヌル値を戻します。

値を集計する順序は定義されていませんが、すべての中間結果は結果のデータ・タイプの範囲内になければなりません。

回帰関数はすべて、1 回のデータ・パススルーでまとめて計算されます。一般に、回帰関数を使用して回帰分析に必要な統計を計算した方が、AVERAGE、VARIANCE、COVARIANCE などの通常の列関数を使用して同じ計算を実行するよりも効率的です。

線形回帰分析に関係する一般的な診断統計についても、上記の関数を使用して計算できます。たとえば、次のようにします。

調整を加えた R2

```
1 - ( (1 - REGR_R2) * ((REGR_COUNT - 1) / (REGR_COUNT - 2)) )
```

標準誤差

SQRT((REGR_SYY-
(POWER(REGR_SXY,2)/REGR_SXX))/(REGR_COUNT-2))

2 乗和の合計

REGR_SYY

回帰 2 乗和

POWER(REGR_SXY,2) / REGR_SXX

残差 2 乗和

(2 乗和の合計) - (回帰 2 乗和)

傾きについての t 統計

REGR_SLOPE * SQRT(REGR_SXX) / (標準誤差)

y 切片についての t 統計

REGR_INTERCEPT/(標準
誤差)*SQRT((1/REGR_COUNT)+(POWER(REGR_AVGX,2)/REGR_SXX))

例:

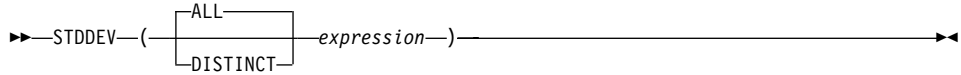
- EMPLOYEE 表を使用して、部門 (WORKDEPT) 'A00' の従業員の賞与を表す回帰直線 (通常の最小 2 乗法による) を、従業員の給与の線形関数として計算します。ホスト変数 SLOPE、ICPT、RSQR (いずれも倍精度浮動小数点数) をそれぞれ、回帰直線の判別の傾き、y 切片、係数に設定します。また、ホスト変数 AVGSAL を部門 'A00' の従業員の平均給与、ホスト変数 AVGBONUS を部門 'A00' の従業員の平均賞与に設定します。さらに、ホスト変数 CNT (整数) を、部門 'A00' の従業員のうち、給与データと賞与データが両方とも存在している従業員の数に設定します。その他の回帰統計はホスト変数 SXX、SYY、および SXY に格納します。

```
SELECT REGR_SLOPE(BONUS,SALARY), REGR_INTERCEPT(BONUS,SALARY),
   REGR_R2(BONUS,SALARY), REGR_COUNT(BONUS,SALARY),
   REGR_AVGX(BONUS,SALARY), REGR_AVGY(BONUS,SALARY),
   REGR_SXX(BONUS,SALARY), REGR_SYY(BONUS,SALARY),
   REGR_SXY(BONUS,SALARY)
   INTO :SLOPE, :ICPT,
        :RSQR, :CNT,
        :AVGSAL, :AVGBONUS,
        :SXX, :SYY,
        :SXY
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
```

サンプル表を使用する場合、ホスト変数は以下の概数値に設定されます。

```
SLOPE:  +1.71002671916749E-002
ICPT:  +1.00871888623260E+002
RSQR:  +9.99707928128685E-001
CNT:  3
AVGSAL:  +4.28333333333333E+004
AVGBONUS:  +8.33333333333333E+002
SXX:  +2.96291666666667E+008
SYY:  +8.66666666666667E+004
SXY:  +5.06666666666667E+006
```

STDDEV



スキーマは SYSIBM です。

STDDEV 関数は、一連の数値の標準偏差を戻します。

引き数には、数値を指定しなければなりません。

結果のデータ・タイプは、倍精度の浮動小数点です。結果はヌル値の場合もあります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。 DISTINCT を指定すると、重複する値は除去されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の値の標準偏差になります。

値を集計する順序は定義されていませんが、すべての中間結果は結果のデータ・タイプの範囲内になければなりません。

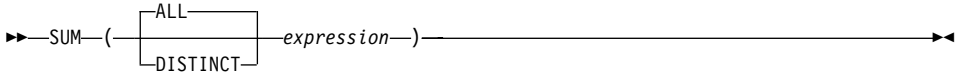
例:

- EMPLOYEE 表を使用して、ホスト変数 DEV (倍精度の浮動小数点) を部署 (WORKDEPT) 'A00' の従業員の給与の標準偏差に設定します。

```
SELECT STDDEV(SALARY)
  INTO :DEV
  FROM EMPLOYEE
  WHERE WORKDEPT = 'A00'
```

  サンプル表を使用した場合、結果として DEV はおよそ 9938.00 に設定されます。

SUM



スキーマは SYSIBM です。

SUM 関数は、一連の数値の合計値を戻します。

引き数の値は数値 (組み込みタイプのみ) でなければならず、その合計は、結果のデータ・タイプの範囲内になければなりません。

結果のデータ・タイプは、引き数値のデータ・タイプと同じです。ただし、以下の点が異なります。

- 引き数値が短精度整数 (small integer) の場合、結果は長精度整数 (large integer) になります。
- 引き数値が単精度浮動小数点の場合、結果は倍精度浮動小数点になります。

引き数値のデータ・タイプが 10 進数の場合、結果の精度は 31、位取りは引き数値の位取りと同じになります。結果はヌル値の場合もあります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。 DISTINCT を指定すると、重複する値も除去されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の値の合計になります。

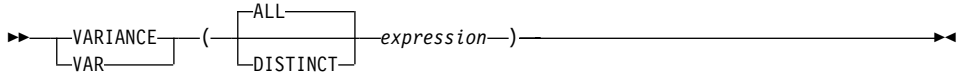
例:

- EMPLOYEE 表を使用して、事務職員 (JOB='CLERK') に支払われるボーナス (BONUS) の総額をホスト変数 JOB_BONUS (decimal(9,2)) に設定します。

```
SELECT SUM(BONUS)
  INTO :JOB_BONUS
  FROM EMPLOYEE
  WHERE JOB = 'CLERK'
```

  サンプル表を使用すると、結果として JOB_BONUS は 2800 に設定されます。

VARIANCE



スキーマは SYSIBM です。

VARIANCE 関数は、一連の数値の差異を戻します。

引き数には、数値を指定しなければなりません。

結果のデータ・タイプは、倍精度の浮動小数点です。結果はヌル値の場合もあります。

この関数は、引き数の値からヌル値を除いて求めた値の集合に対して適用されます。 DISTINCT を指定すると、重複する値は除去されます。

この関数が空集合に適用されると、結果はヌル値になります。それ以外の場合、結果は集合の値の差異になります。

値が加算される順序は定義されていませんが、すべての中間結果は結果のデータ・タイプの範囲内になければなりません。

例:

- EMPLOYEE 表を使用して、ホスト変数 VARNCE (倍精度の浮動小数点) を部署 (WORKDEPT) 'A00' の従業員の給与の差異に設定します。

```
SELECT VARIANCE(SALARY)
  INTO :VARNCE
  FROM EMPLOYEE
  WHERE WORKDEPT = 'A00'
```

  サンプル表を使用した場合、結果として VARNCE はおよそ 98763888.88 に設定されます。

スカラー関数

スカラー関数は、式を使用できる個所であればどこにでも使用することができます。ただし、式と列関数の使用に適用される制約事項は、スカラー関数の中で式または列関数を使用する場合にも適用されます。たとえば、スカラー関数の引き数を列関数にすることができるのは、スカラー関数が使用される文脈で列関数の使用が許されている場合だけです。

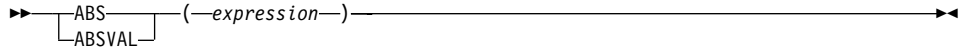
列関数の使用法に関する制約事項がスカラー関数に適用されないのは、スカラー関数が、値の集合ではなく、単一の値を対象にするからです。

例: 次の SELECT ステートメントの結果には、部門 D01 の従業員の数と同じ数の行が含まれます。

```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BRTHDATE)
  FROM EMPLOYEE
  WHERE WORKDEPT = 'D01'
```

以下に示すスカラー関数は、スキーマ名によって修飾することができます (たとえば、SYSIBM.CHAR(123))。

ABS または ABSVAL



スキーマは SYSFUN です。

引き数の絶対値を戻します。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数のタイプが DECIMAL または REAL である場合、関数による処理に必要な倍精度浮動小数点数に変換されます。

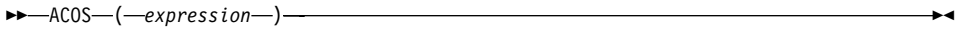
関数の結果は次のとおりです。

- 引き数が SMALLINT の場合は SMALLINT になります。
- 引き数が INTEGER の場合は INTEGER になります。
- 引き数が BIGINT の場合は BIGINT になります。
- 引き数が DOUBLE、DECIMAL または REAL の場合は DOUBLE になります。[39]

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

39. 引き数が BIGINT の最小値である -9 223 372 036 854 775 808 の場合も、 DOUBLE が戻されます。

ACOS



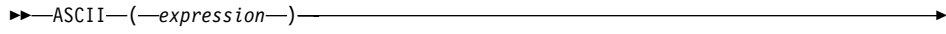
スキーマは SYSFUN です。

引き数のアークコサイン (逆余弦) の角度を戻します (ラジアン単位)。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

ASCII



スキーマは SYSFUN です。

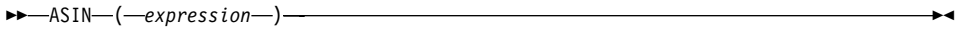
引き数の左端の文字の ASCII コード値を整数として戻します。

引き数は、任意の組み込み文字ストリング・タイプにすることができます。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB の場合、最大長は 1 048 576 バイトです。 LONG VARCHAR は、関数による処理に必要な CLOB に変換されます。

関数の結果は常に INTEGER になります。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

ASIN



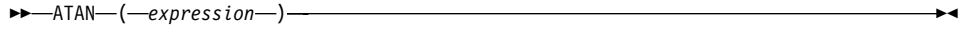
スキーマは SYSFUN です。

引き数のアークサイン (逆正弦) の角度を戻します (ラジアン単位)。

引き数は、任意の組み込み数値タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

ATAN



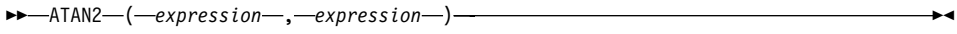
スキーマは SYSFUN です。

引き数のアークタンジェント (逆正接) の角度を戻します (ラジアン単位)。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

ATAN2



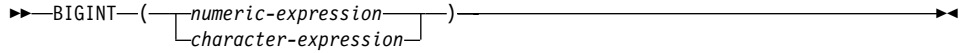
スキーマは SYSFUN です。

x 座標および y 座標のアークタンジェント (逆正接) の角度を戻します (ラジアン単位)。 x 座標および y 座標はそれぞれ、最初と 2 番目の引き数によって指定されます。

最初と 2 番目の引き数は、任意の組み込み数値データ・タイプにすることができます。いずれの引き数も、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

BIGINT



スキーマは SYSIBM です。

BIGINT 関数は、数値または文字ストリングを 64 ビットで表した整数を、整数定数の形で戻します。

numeric-expression
組み込み数値データ・タイプの値を戻す式。

引き数が*数値式* の場合、結果は、引き数を大整数 (big integer) の列または変数に割り当てた場合と同じ数値になります。引き数の整数部分が整数の範囲内にない場合、エラーになります。引き数に小数部分がある場合は、切り捨てられます。

character-expression
文字定数の最大長以下の長さの文字ストリング値を戻す式。先行ブランクと後書きブランクは除去されます。その結果のストリングは、 SQL 整数定数を形成するための規則に従うものでなければなりません (SQLSTATE 22018)。文字ストリングとして、長ストリングを使うことはできません。

引き数が文字式 の場合、結果は、対応する整数定数を大整数 (big integer) の列または変数に割り当てた場合の数値と同じになります。

関数の結果は大整数です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例:

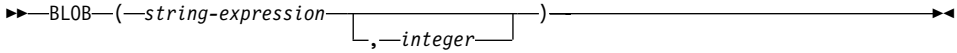
- ORDERS_HISTORY 表から、注文の数を数えて結果を大整数値として戻します。

```
SELECT BIGINT (COUNT_BIG(*) )
  FROM ORDERS_HISTORY
```

- EMPLOYEE 表を使用して、アプリケーションでさらに処理を行うために、EMPNO を大整数形式として選択します。

```
SELECT BIGINT(EMPNO) FROM EMPLOYEE
```

BLOB



スキーマは SYSIBM です。

BLOB 関数は、任意のタイプのストリングの BLOB 表記を戻します。

string-expression
その値が文字ストリング、漢字ストリング、または 2 進ストリングのストリング式。

integer
結果の BLOB データ・タイプの長さ属性を指定する整数値。 *integer* を指定しない場合、結果の長さ属性は入力の長さと同じになります。ただし、入力が漢字ストリングの場合は除きます。その場合、結果の長さ属性は入力の長さの 2 倍になります。

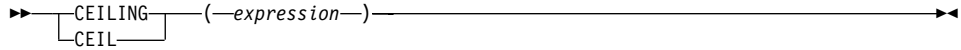
関数の結果は BLOB です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例

- TOPOGRAPHIC_MAP という名前の BLOB 列と、 MAP_NAME という名前の VARCHAR 列の含まれている表を使用して、 'Pellow Island' というストリングを含むマップ (MAP) を探し出し、実際のマップの先頭にマップ名を連結した単一の 2 進ストリングを戻します。

```
SELECT BLOB(MAP_NAME  || ':  ') || TOPOGRAPHIC_MAP
  FROM ONTARIO_SERIES_4
  WHERE TOPOGRAPIC_MAP LIKE BLOB('%Pellow Island%')
```

CEILING または CEIL



スキーマは SYSFUN です。

引き数よりも大きいか、または等しい整数で、最小の値を戻します。

引き数は、任意の組み込み数値タイプにすることができます。引き数のタイプが DECIMAL または REAL である場合、関数による処理に必要な倍精度浮動小数点数に変換されます。引き数のタイプが SMALLINT または INTEGER である場合、その引き数値が戻されます。

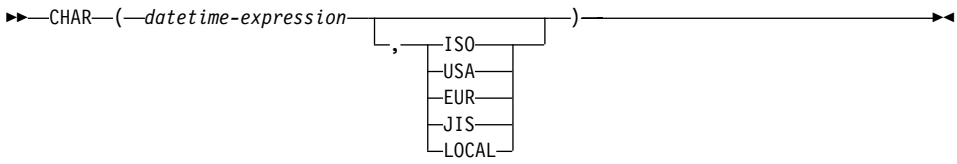
関数の結果は次のとおりです。

- 引き数が SMALLINT の場合は SMALLINT になります。
- 引き数が INTEGER の場合は INTEGER になります。
- 引き数が BIGINT の場合は BIGINT になります。
- 引き数が DECIMAL、REAL または DOUBLE の場合は DOUBLE になります。小数部の左側にある桁数が 15 桁を超える 10 進数値は、 DOUBLE への変換での精度の低下により、必要な整数値は戻されません。

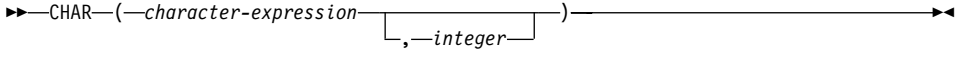
結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

CHAR

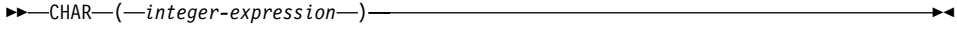
日付 / 時刻→文字:



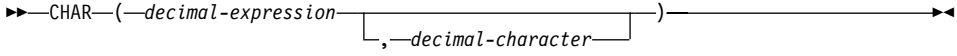
文字→文字:



整数→文字:



10 進数→文字:



浮動小数点数→文字:

スキーマは SYSIBM です。 ただし、CHAR(floating-point-expression) のスキーマは SYSFUN です。

CHAR 関数は、以下の文字ストリング表記を戻します。

- 日付 / 時刻 (最初の引き数が日付、時刻、またはタイム・スタンプの場合)
- 文字ストリング (最初の引き数がいずれかのタイプの文字ストリングの場合)
- 整数 (最初の引き数が SMALLINT、INTEGER または BIGINT の場合)
- 10 進数 (最初の引き数が 10 進数の場合)
- 倍精度浮動小数点 (最初の引き数が DOUBLE または REAL の場合)

関数の結果は、固定長文字ストリングです。最初の引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。最初の引き数がヌル値の場合には、結果もヌル値です。

日付 / 時刻→文字

datetime-expression
次の 3 つのデータ・タイプのいずれかの式。

日付
結果は、2 番目の引き数によって指定された形式の日付の文字ストリング表記になります。結果の長さは 10 文字です。 2 番目の引き数が指定され、その値が有効な値でない場合には、エラーが生じます (SQLSTATE 42703)。

時刻 (time)
結果は、2 番目の引き数によって指定された形式の時刻の文字ストリング表記になります。結果の長さは 8 になります。2 番目の引き数が指定され、その値が有効な値でない場合には、エラーが生じます (SQLSTATE 42703)。

タイム・スタンプ
2 番目の引き数は適用されないので、指定してはなりません。 SQLSTATE 42815。結果は、タイム・スタンプの文字ストリング表記になります。結果の長さは 26 文字です。

ストリングのコード・ページは、アプリケーション・サーバーのデータベースのコード・ページになります。

文字→文字

character-expression
CHAR、VARCHAR、LONG VARCHAR、または CLOB のいずれかのデータ・タイプの値を戻す式。

integer
結果の固定長文字ストリングの長さ属性。値は 0 〜 254 の範囲でなければなりません。

文字式の長さが結果の長さ属性より短い場合、結果の長さになるまでブランクが結果に埋められます。文字式の長さが結果の長さ属性より長い場合、切り捨てが行われます。その場合、切り捨てられた文字がすべてブランクで、文字式が長ストリング (LONG VARCHAR または CLOB) でない限り、警告 (SQLSTATE 01004) が戻されます。

整数→文字

integer-expression
整数データ・タイプの値 (SMALLINT、INTEGER または BIGINT のいずれか) を戻す式。

結果は、SQL 整数定数の形式による引き数の文字ストリング表記になります。結果は、引き数の値を表す有効数字 n 桁で構成されます。引き数が負の場合は、先行負符号 (-) も含みます。これは、左そろえになります。

- 最初の引き数が短精度整数 (small integer) の場合、

  結果の長さは 6 になります。結果の文字数が 6 未満の場合、 6 文字になるまで結果の右側にブランクが埋められます。

- 最初の引き数が長精度整数 (large integer) の場合、

  結果の長さは 11 文字です。結果の文字数が 11 未満の場合、 11 文字になるまで結果の右側にブランクが埋められます。

- 最初の引き数が大整数 (big integer) の場合、

  結果の長さは 20 文字です。結果の文字数が 20 未満の場合、 20 文字になるまで結果の右側にブランクが埋められます。

ストリングのコード・ページは、アプリケーション・サーバーのデータベースのコード・ページになります。

10 進数→文字

decimal-expression
10 進数データ・タイプの値を戻す式。異なる精度と位取りが必要になる場合、まず DECIMAL スカラー関数を使用して変更を行うことができます。

decimal-character
結果文字ストリングの中で 10 進数を区切るために使用する 1 バイト文字定数を指定します。文字を数字、正記号 ('+')、負記号 ('-')、またはブランクにすることはできません (SQLSTATE 42815)。デフォルト値はピリオド ('.') 文字です。

結果は、引き数の固定長文字ストリング表記になります。結果は *p* 桁の数字になります (*p* は *decimal-expression* の精度)。引き数が負の場合は、前に負符号が付けられます。結果の長さは 2+*p* です (*p* は *decimal-expression* の精度)。つまり、正の値には常に 1 個の後書きブランクが含められます。

ストリングのコード・ページは、アプリケーション・サーバーのデータベースのコード・ページになります。

浮動小数点数→文字

floating-point-expression
浮動小数点データ・タイプ (DOUBLE または REAL) である値を戻す式。

結果は、浮動小数点定数形式による引き数の固定長文字ストリング表記になります。結果の長さは 24 文字です。引き数が負である場合、結果の先頭の文字は負符号 (-) になります。それ以外の場合、先頭の文字は数字になります。引き数値がゼロである場合、結果は 0E0 です。それ以外の場合、結果は、小数部がゼロ以外の単一数字の後にピリオドおよび一連の数字が続いたもので構成され、引き数の値を表すことができる最小の文字数になります。結果の文字数が 24 未満の場合、 24 文字になるまで結果の右側にブランクが埋められます。

ストリングのコード・ページは、アプリケーション・サーバーのデータベースのコード・ページになります。

例:

- 列 PRSTDATE には、1988-12-25 に相当する内部値が入っているとします。

```
CHAR(PRSTDATE, USA)
```

結果は ‘12/25/1988’ の値になります。

- 列 STARTING には 17.12.30 に相当する内部値が入っており、ホスト変数 HOUR_DUR (decimal(6,0)) は、値が 050000 (つまり 5 時間) の時刻期間とします。

```
CHAR(STARTING, USA)
```

結果の値は ’5:12 PM’ になります。

```
CHAR(STARTING + :HOUR_DUR, USA)
```

結果の値は ’10:12 PM’ になります。

- 列 RECEIVED (タイム・スタンプ) には、列 PRSTDATE と列 STARTING を組み合わせたものに相当する内部値が入っているとします。

```
CHAR(RECEIVED)
```

結果は ‘1988-12-25-17.12.30.000000’ の値になります。

- CHAR 関数を使用して、 EMPLOYEE 表の LASTNAME 列 (VARCHAR(15) として定義されているもの) を、固定長文字ストリングにし、表示結果の長さを 10 文字にします。

```
SELECT CHAR(LASTNAME,10) FROM EMPLOYEE
```

後書きブランクを除いて長さが 10 文字を超える LASTNAME を含む行の場合は、値が切り捨てられたことを示す警告が戻されます。

- CHAR 関数を使用して、EDLEVEL (smallint として定義されているもの) の値を、固定長文字ストリングとして戻します。

```
SELECT CHAR(EDLEVEL) FROM EMPLOYEE
```

EDLEVEL が 18 なら、CHAR(6) 値の '18 ' (18 と 4 文字のブランク) として戻されます。

- STAFF の中に、精度 9、位取り 2 の 10 進数として定義された SALARY 列が含まれているとします。その現行値は 18357.50 であり、小数点文字をコンマにしてそれを表示するとします (18357,50)。

```
CHAR(SALARY, ',')
```

値 '00018357,50 ' が戻されます。

- 同じ SALARY 列を 20000.25 から引いて、デフォルトの小数点文字を使って表示するとします。

```
CHAR(20000.25 - SALARY)
```

値 '-0001642.75' が戻されます。

- ホスト変数 SEASONS_TICKETS のデータ・タイプが整数で、その値が 10000 であるとします。

```
CHAR(DECIMAL(:SEASONS_TICKETS,7,2))
```

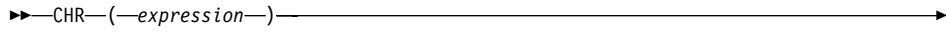
結果は、'10000.00 ' という文字値になります。

- ホスト変数 DOUBLE_NUM のデータ・タイプが DOUBLE であり、値が -987.654321E-35 であるとします。

```
CHAR(:DOUBLE_NUM)
```

結果は、'-9.87654321E-33 ' の文字値になります。結果のデータ・タイプが CHAR(24) であるため、結果には 9 つの後書きブランクがあります。

CHR



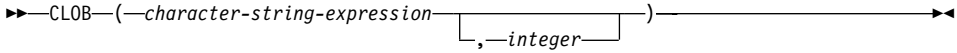
スキーマは SYSFUN です。

引き数で指定される ASCII コード値の文字を戻します。

引き数は INTEGER または SMALLINT のいずれかです。引き数の値は 0 ～ 255 の範囲でなければなりません。そうでない場合、戻り値はヌル値になります。

関数の結果は CHAR(1) です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

CLOB



スキーマは SYSIBM です。

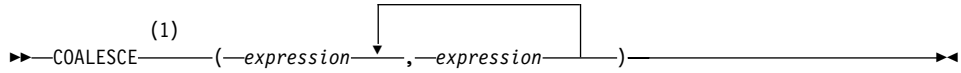
CLOB 関数は、文字ストリング・タイプの CLOB 表記を戻します。

character-string-expression
文字ストリングである値を戻す式。

integer
結果の CLOB データ・タイプの長さ属性を指定する整数値。値は 0 ～ 2 147 483 647 の範囲でなければなりません。 *integer* を指定しない場合、結果の長さは、最初の引き数の長さと同じになります。

関数の結果は CLOB です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

COALESCE



注:

1 VALUE は COALESCE の同義語です。

スキーマは SYSIBM です。

COALESCE は、その値がヌル値以外の最初の引き数を戻します。

引き数は指定された順序で評価され、関数の結果はヌル値以外の最初の引き数になります。結果は、すべての引き数がヌル値の場合にのみヌル値になります。選択された引き数は、必要に応じて結果の属性に変換されます。

引き数は互いに互換性がなければなりません。互換性のあるデータ・タイプと結果の属性については、 121ページの『結果のデータ・タイプに関する規則』を参照してください。 データ・タイプは、組み込みデータ・タイプか、ユーザー定義データ・タイプのいずれかです。[40]

例:

- DEPARTMENT 表のすべての行のすべての値を選択する場合に、部門の管理者 (MGRNO) が欠落しているときには (つまりヌル値なら)、 'ABSENT' という値を戻すようにします。

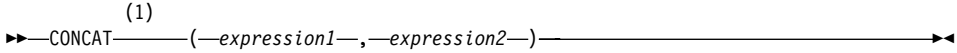
```
SELECT DEPTNO, DEPTNAME, COALESCE(MGRNO, 'ABSENT'), ADMRDEPT
  FROM DEPARTMENT
```

- EMPLOYEE 表のすべての行から従業員番号 (EMPNO) と給与 (SALARY) を選択する場合に、給与が欠落しているなら (つまりヌル値なら)、値としてゼロを戻すようにします。

```
SELECT EMPNO, COALESCE(SALARY, 0)
  FROM EMPLOYEE
```

40. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべての互換データ・タイプを引き数として受け入れるので、ユーザー定義特殊タイプをサポートするための追加のシグニチャーを作成する必要はありません。

CONCAT



注:

1 CONCAT の同義語として || を使用できます。

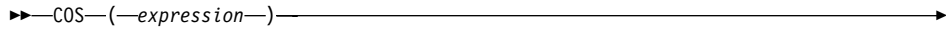
スキーマは SYSIBM です。

2 つのストリング引き数を連結した値を戻します。この 2 つの引き数のタイプには互換性がなければなりません。

関数の結果はストリングです。結果の長さは、2 つの引き数の長さの合計です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

177ページの『連結演算子を使用する式』を参照してください。

COS



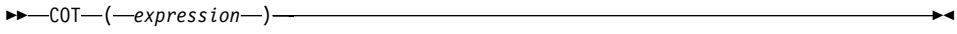
スキーマは SYSFUN です。

引き数に対するコサイン (余弦) の値を戻します。引き数はラジアン単位の角度です。

引き数は、任意の組み込み数値タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

COT



スキーマは SYSFUN です。

引き数に対するコタンジェント (余接) の値を戻します。引き数はラジアン単位の角度です。

引き数は、任意の組み込み数値タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DATE

スキーマは SYSIBM です。

DATE 関数は、値から日付を戻します。

引き数は、日付、タイム・スタンプ、3 652 059 以下の正の整数、日付またはタイム・スタンプの有効な文字ストリング表記、または CLOB でも LONG VARCHAR でもない長さ 7 文字の文字ストリングのいずれかでなければなりません。

引き数が長さ 7 の文字ストリングの場合、 *yyyynnn* という形式の有効な日付を表したものであることが必要です。ここで、*yyyy* は年を示す数字、 *nnn* は年間通算日を示す 001 ～ 366 までの数字です。

関数の結果は日付です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

- 引き数が日付、タイム・スタンプ、または日付やタイム・スタンプの有効なストリング表記の場合
  - 結果はその値の日付部分です。
- 引き数が数値の場合
  - 結果は、0001 (1 月 1 日) から数えて *n*-1 日後の日付です (*n* は数字の整数部分)。
- 引き数が長さ 7 の文字ストリングの場合
  - 結果は、その文字ストリングで表された日付になります。

例:

- 列 RECEIVED (タイム・スタンプ) には、 '1988-12-25-17.12.30.000000' に相当する内部値が入っているものとします。

  ```
  DATE(RECEIVED)
  ```

  結果は、'1988-12-25' の内部表記になります。
- 以下の例の結果は、'1988-12-25' の内部表記になります。

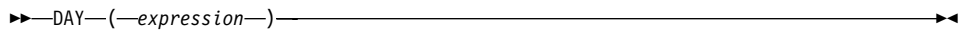
  ```
  DATE('1988-12-25')
  ```
- 以下の例の結果は、'1988-12-25' の内部表記になります。

```
DATE('25.12.1988')
```

- 以下の例の結果は、‘0001-02-04’ の内部表記になります。

```
DATE(35)
```

DAY



スキーマは SYSIBM です。

DAY 関数は、値の日の部分を戻します。

引き数は、日付、タイム・スタンプ、日付期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

- 引き数が日付、タイム・スタンプ、または日付やタイム・スタンプの有効なストリング表記の場合
  - 結果は、値の日の部分 (1 ～ 31 の整数) になります。
- 引き数が日付期間またはタイム・スタンプ期間の場合
  - 結果は、値の日の部分 (-99 ～ 99 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- PROJECT 表を使用して、 WELD LINE PLANNING プロジェクト (PROJNAME) の終了予定日 (PRENDATE) をホスト変数 END_DAY (短精度整数) に設定します。

```
SELECT DAY(PRENDATE)
  INTO :END_DAY
  FROM PROJECT
  WHERE PROJNAME = 'WELD LINE PLANNING'
```

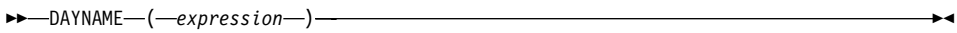
  サンプル表を使用した場合、結果として END_DAY は 15 に設定されます。

- 列 DATE1 (日付) には、2000-03-15 に相当する内部値が入っていて、列 DATE2 (日付) には、1999-12-31 に相当する内部値が入っているとします。

```
DAY(DATE1 - DATE2)
```

  結果は、15 の値になります。

DAYNAME



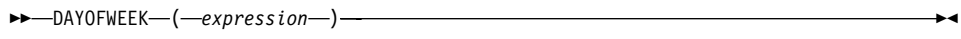
スキーマは SYSFUN です。

データベースが始動された時点のロケールに基づいて、引き数の日の部分の曜日名 (例: Friday) を含む大文字小文字混合文字ストリングを戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は VARCHAR(100) です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DAYOFWEEK



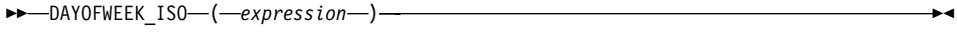
スキーマは SYSFUN です。

引き数の曜日を 1 ～ 7 の範囲の整数値として戻します。1 は日曜日を表します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DAYOFWEEK_ISO



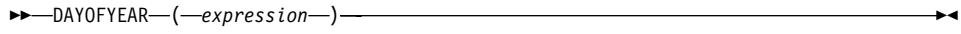
スキーマは SYSFUN です。

引き数の曜日を 1 ～ 7 の範囲の整数値として戻します。1 は月曜を表します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DAYOFYEAR



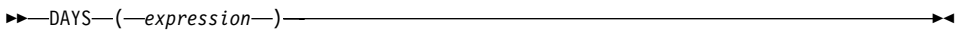
スキーマは SYSFUN です。

引き数の年間通算日を、1 ～ 366 の範囲の整数値として戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DAYS



スキーマは SYSIBM です。

DAYS 関数は、日付の整数表記を戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

結果は、1 月 1 日 (0001) から *D* までの日数に、 1 を加えた数になります (*D* は、引き数に DATE 関数を適用した場合の結果となる日付)。

例:

- PROJECT 表を使用して、プロジェクト (PROJNO) 'IF2000' に要する見積日数 (PRENDATE - PRSTDATE) をホスト変数 EDUCATION_DAYS (整数) に設定します。

  ```
  SELECT DAYS(PRENDATE) - DAYS(PRSTDATE)
    INTO :EDUCATION_DAYS
    FROM PROJECT
    WHERE PROJNO = 'IF2000'
  ```

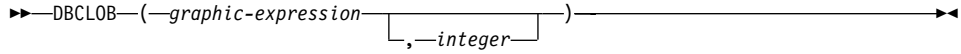
  サンプル表を使用した場合、結果として EDUCATION_DAYS は 396 に設定されます。

- PROJECT 表を使用して、ホスト変数 TOTAL_DAYS (int) に、部署 (DEPTNO) 'E21' のすべてのプロジェクトについての経過日数見積もり (PRENDATE - PRSTDATE) の合計を設定します。

  ```
  SELECT SUM(DAYS(PRENDATE) - DAYS(PRSTDATE))
    INTO :TOTAL_DAYS
    FROM PROJECT
    WHERE DEPTNO = 'E21'
  ```

  サンプル表を使用した場合、結果として TOTAL_DAYS は 1584 に設定されます。

DBCLOB



スキーマは SYSIBM です。

DBCLOB 関数は、漢字ストリング・タイプの DBCLOB 表記を戻します。

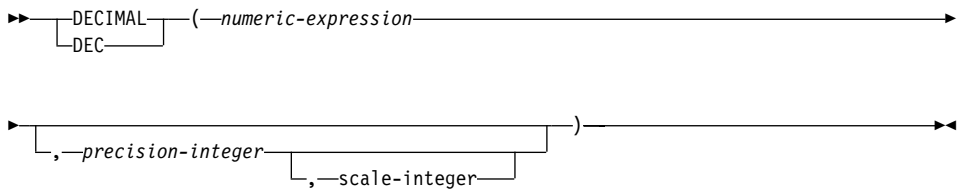
graphic-expression
漢字ストリング値を戻す式。

integer
結果の DBCLOB データ・タイプの長さ属性を指定する整数値。値は 0 ～ 1 073 741 823 の範囲でなければなりません。 *integer* を指定しない場合、結果の長さは、最初の引き数の長さと同じになります。

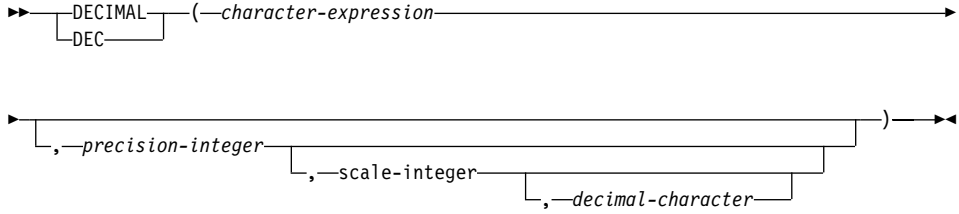
関数の結果は DBCLOB です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DECIMAL

数値→ 10 進数:



文字→ 10 進数:



スキーマは SYSIBM です。

DECIMAL 関数は、以下の 10 進表記を戻します。

- 数値
- 10 進数の文字ストリング表記
- 整数の文字ストリング表記

関数の結果は、精度 *p*、位取り *s* の 10 進数になります (*p* と *s* はそれぞれ 2 番目と 3 番目の引き数)。最初の引き数がヌル値の可能性がある場合、結果もヌル値の可能性があります。最初の引き数がヌル値である場合、その結果もヌル値になります。

数値→ 10 進数

numeric-expression
数値データ・タイプの値を戻す式。

precision-integer
1 ～ 31 の範囲の値の整数定数。

precision-integer のデフォルト値は、 *numeric-expression* のデータ・タイプによって異なります。

- 浮動小数点および 10 進数の場合は 15

- 大整数の場合は 19
- 長精度整数の場合は 11
- 短精度整数の場合は 5

scale-integer
0 から *precision-integer* の値までの範囲の整数定数。デフォルト値はゼロです。

結果は、最初の引き数が精度 *p*、位取り *s* の 10 進数の列または変数に割り当てられた場合と同じ数になります (*p* と *s* は、それぞれ 2 番目と 3 番目の引き数)。数値の整数部分を表すために必要な有効 10 進桁数が、 *p* - *s* より大きい場合はエラーになります。

文字→ 10 進数

character-expression
長さが文字定数の最大長 (4 000 バイト) 以下の文字ストリングである値を戻す式。 CLOB または LONG VARCHAR データ・タイプにすることはできません。先行ブランクと後書きブランクは、ストリングから除去されます。この結果のサブストリングは、 SQL 整数または 10 進定数を形成するための規則に準拠していなければなりません (SQLSTATE 22018)。

定数 *decimal-character* のコード・ページに一致させるために必要なら、 *character-expression* はデータベース・コード・ページに変換されます。

precision-integer
結果の精度を指定する整数定数 (値の範囲は 1 ～ 31)。この指定がない場合のデフォルト値は 15 です。

scale-integer
結果の位取りを指定する整数定数 (値の範囲は 0 ～ *precision-integer*)。この指定がない場合のデフォルト値は 0 です。

decimal-character
character-expression の小数部分と整数部分とを区切るために使用する 1 バイト文字定数を指定します。この文字には、数字のプラス ('+')、マイナス ('-')、またはブランクは使用できず、 *character-expression* の中に最高 1 回しか使用することができません (SQLSTATE 42815)。

結果は、精度 *p*、位取り *s* の 10 進数になります (*p* と *s* は、それぞれ 2 番目と 3 番目の引き数)。小数点以下の数字の桁数が、位取り *s*

より多い場合、終わりから数字が切り捨てられます。
character-expression の小数点文字の左側にある有効数字 (数値の整数部分) の桁数が *p - s* よりも多い場合は、エラーになります (SQLSTATE 22003)。 *decimal-character* 引き数が指定されている場合、サブストリングに使われているデフォルトの小数点文字は無効になります (SQLSTATE 22018)。

例:

- EMPLOYEE 表の EDLEVEL 列 (データ・タイプ = SMALLINT) の選択リストに (精度が 5 で、位取りが 2 の) DECIMAL データ・タイプが必ず戻されるように DECIMAL 関数を使用します。選択リストには、EMPNO 列も入れておいてください。

```
SELECT EMPNO, DECIMAL(EDLEVEL,5,2)
  FROM EMPLOYEE
```

- ホスト変数 PERIOD のタイプが INTEGER であるとします。その値を日付期間として使用するためには、それを decimal(8,0) として「キャスト」する必要があります。

```
SELECTPRSTDATE + DECIMAL(:PERIOD,8)
  FROM PROJECT
```

- SALARY 列の更新内容が、小数点文字にコンマを使用した文字ストリングとして、ウィンドウから入力されるものとします (ユーザーは、たとえば 21400,50 と入力します)。アプリケーションが妥当性検査を行った後、これを、CHAR(10) として定義されたホスト変数 newsalary に設定します。

```
UPDATE STAFF
  SET SALARY = DECIMAL(:newsalary, 9, 2, ',')
  WHERE ID = :empid;
```

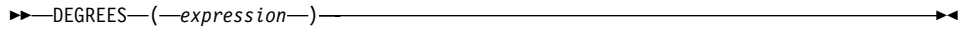
  newsalary の値は 21400.50 になります。

- 値にデフォルト値の小数点文字 (.) を追加します。

```
DECIMAL('21400,50', 9, 2, '.')
```

  この例では、小数点文字としてピリオド (.) を指定しているのに、第 1 引き数の中で区切り文字としてコンマ (,) が使われているため、エラーになります。

DEGREES



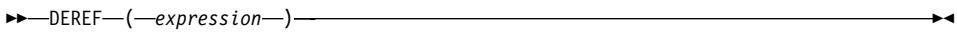
スキーマは SYSFUN です。

ラジアン単位の引き数を度単位の角度に変換して戻します。

引き数は、任意の組み込み数値タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

DEREF



スキーマは SYSIBM です。

DEREF 関数は引き数のターゲット・タイプのインスタンスを戻します。

引き数は、効力範囲が定義された参照データ・タイプが指定されている任意の値にすることができます (SQLSTATE 428DT)。

結果の静的データ・タイプは、引き数のターゲット・タイプです。結果の動的データ・タイプは、引き数のターゲット・タイプのサブタイプです。結果はヌル値の場合もあります。 *expression* がヌル値か、または *expression* が突き合わせする OID がターゲット表にない参照の場合、結果はヌル値になります。

結果は参照のターゲット・タイプのサブタイプのインスタンスです。結果は参照値と突き合わせするオブジェクト識別子がある参照の、ターゲット表またはターゲット視点の行を検出することにより決定されます。この行のタイプによって結果の動的タイプが決定されます。結果のタイプがターゲット表の副表の行またはターゲット視点の副視点の行に基づく場合があるため、ステートメントの許可 ID にはターゲット表とその副表のすべて、またはターゲット視点とその副視点のすべての SELECT 特権が必要です (SQLSTATE 42501)。

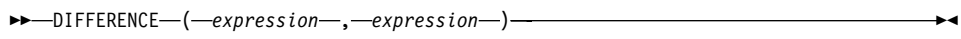
例:

EMPLOYEE はタイプ EMP の表であり、そのオブジェクト ID 列は EMPID であるとします。次の照会は、EMPLOYEE 表 (およびその副表) の行ごとに、タイプ EMP (またはそのサブタイプのいずれか) のオブジェクトを戻します。この照会では、EMPLOYEE およびその副表すべてに対する SELECT 権限が必要です。

```
SELECT DEREF(EMPID) FROM EMPLOYEE
```

別の例については、405ページの『TYPE_NAME』を参照してください。

DIFFERENCE



スキーマは SYSFUN です。

ストリングへの SOUNDEX 関数の適用に基づいて、 2 つのストリングの音の差を示す 0 ～ 4 の値を戻します。 4 の値は可能な限り最良の音の一致を示します。

引き数は、CHAR または VARCHAR いずれかの文字ストリング (最大長 4 000 バイト) にすることができます。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

例:

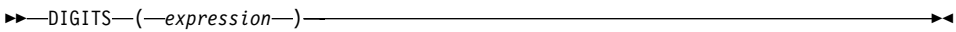
```
VALUES (DIFFERENCE('CONSTRAINT','CONSTANT'),SOUNDEX('CONSTRAINT'),
        SOUNDEX('CONSTANT')),
        (DIFFERENCE('CONSTRAINT','CONTRITE'),SOUNDEX('CONSTRAINT'),
        SOUNDEX('CONTRITE'))
```

この例では、以下を戻します。

```
1           2    3
----------- ---- ----
          4 C523 C523
          2 C523 C536
```

最初の行で、SOUNDEX の語は同じ結果になりますが、2 行目の語は類似性があるにすぎません。

DIGITS



スキーマは SYSIBM です。

DIGITS 関数は、数値の文字ストリング表記を戻します。

引き数は、タイプが SMALLINT、INTEGER、 BIGINT または DECIMAL の値を戻す式でなければなりません。

引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

この関数の結果は、引き数の位取りに関係なく、引き数の絶対値を表す固定長文字ストリングになります。結果には、符号も小数点文字も含まれません。結果は、必要に応じてストリングを埋めるための先行ゼロを含めて数字だけから構成されます。ストリングの長さは次のとおりです。

- 引き数が短精度整数 (small integer) の場合は 5
- 引き数が長精度整数 (large integer) の場合は 10
- 引き数が大整数 (big integer) の場合は 19
- 引き数が精度 *p* の 10 進数の場合は *p*

例:

- 表 TABLEX に、INTCOL という INTEGER 列があり、その値が 10 桁の数値であるとします。列 INTCOL に含まれる最初の 4 桁の数字からなる、異なる 4 文字の組み合わせすべてのリストを作成します。

```
SELECT DISTINCT SUBSTR(DIGITS(INTCOL),1,4)
  FROM TABLEX
```

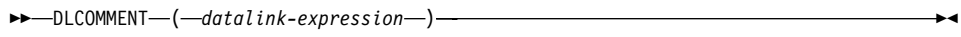
- COLUMNX のデータ・タイプが DECIMAL(6,2) であり、その値の 1 つが -6.28 であると想定します。この値に対して次の関数を実行すると、

```
DIGITS(COLUMNX)
```

  値 '000628' が戻されます。

  この結果は、ストリングをこの長さまで埋めるための先行ゼロを含む、長さ 6 (列の精度) のストリングになります。符号も小数点文字も結果には示されません。

DLCOMMENT



スキーマは SYSIBM です。

DLCOMMENT 関数は、DATALINK 値からコメント値を戻します (コメント値がある場合)。

引き数はデータ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(254) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例:

- HOCKEY_GOALS 表の ARTICLES 列へのリンクから日付、記述およびコメントを選択するステートメントを準備します。選択する行は、Richard 兄弟 (Maurice または Henri) のどちらかが得点したゴールの行です。

```
stmtvar = "SELECT DATE_OF_GOAL, DESCRIPTION, DLCOMMENT(ARTICLES)
             FROM HOCKEY_GOALS
             WHERE BY_PLAYER = 'Maurice Richard'
             OR BY_PLAYER = 'Henri Richard' ";
EXEC SQL PREPARE HOCKEY_STMT FROM :stmtvar;
```

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

```
DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','A comment')
```

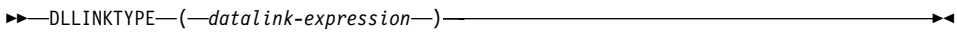
そして、その値を次の関数で処理します。

```
DLCOMMENT(COLA)
```

次の値が戻されます。

```
A comment
```

DLLINKTYPE



スキーマは SYSIBM です。

DLLINKTYPE 関数は、DATALINK 値からリンク・タイプ値を戻します。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(4) になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

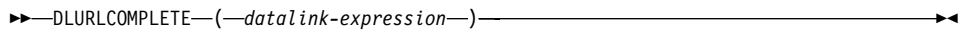
  そして、その値を次の関数で処理します。

  ```
  DLLINKTYPE(COLA)
  ```

  次の値が戻されます。

  ```
  URL
  ```

DLURLCOMPLETE



スキーマは SYSIBM です。

DLURLCOMPLETE 関数は、リンク・タイプの URL を持つ DATALINK 値から、データ位置属性を戻します。適切な場合には、この値にはファイル・アクセス・トークンが組み込まれます。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(254) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DATALINK 値にコメントしか組み込まれていない場合、長さがゼロのストリングが戻されます。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

  そして、その値を次の関数で処理します。

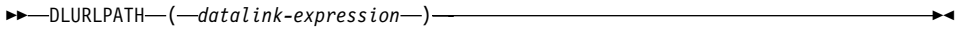
  ```
  DLURLCOMPLETE(COLA)
  ```

  次の値が戻されます。

  ```
  HTTP://DLFS.ALMADEN.IBM.COM/x/y/****************;a.b
  ```

  (ここで ********** はアクセス・トークンを表します)

DLURLPATH



スキーマは SYSIBM です。

DLURLPATH 関数は、特定サーバー内のファイルにアクセスするために必要なパスおよびファイル名を、リンク・タイプが URL の DATALINK 値から戻します。適切な場合には、この値にはファイル・アクセス・トークンが組み込まれます。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(254) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DATALINK 値にコメントしか組み込まれていない場合、長さがゼロのストリングが戻されます。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

  そして、その値を次の関数で処理します。

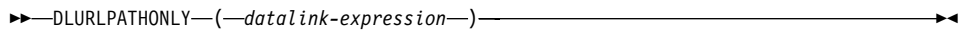
  ```
  DLURLPATH(COLA)
  ```

  次の値が戻されます。

  ```
  /x/y/****************;a.b
  ```

  (ここで ********** はアクセス・トークンを表します)

DLURLPATHONLY



スキーマは SYSIBM です。

DLURLPATHONLY 関数は、特定サーバー内のファイルにアクセスするために必要なパスおよびファイル名を、リンク・タイプが URL の DATALINK 値から戻します。戻される値にはファイル・アクセス・トークンは含まれません。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(254) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DATALINK 値にコメントしか組み込まれていない場合、長さがゼロのストリングが戻されます。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

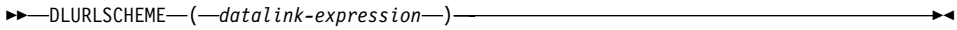
  そして、その値を次の関数で処理します。

  ```
  DLURLPATHONLY(COLA)
  ```

  次の値が戻されます。

  ```
  /x/y/a.b
  ```

DLURLSCHEME



スキーマは SYSIBM です。

DLURLSCHEME 関数は、リンク・タイプが URL の DATALINK 値から方式を戻します。この値は必ず大文字になります。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(20) になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DATALINK 値にコメントしか組み込まれていない場合、長さがゼロのストリングが戻されます。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

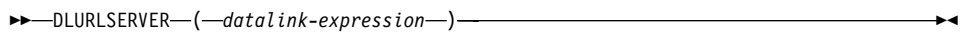
  そして、その値を次の関数で処理します。

  ```
  DLURLSCHEME(COLA)
  ```

  次の値が戻されます。

  ```
  HTTP
  ```

DLURLSERVER



スキーマは SYSIBM です。

DLURLSERVER 関数は、リンク・タイプが URL の DATALINK 値からファイル・サーバーを戻します。この値は必ず大文字になります。

引き数は、データ・タイプが DATALINK の値を戻す式でなければなりません。

関数の結果は VARCHAR(254) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

DATALINK 値にコメントしか組み込まれていない場合、長さがゼロのストリングが戻されます。

例:

- スカラー関数を使用して、表 TBLA にある行の列 COLA に DATALINK 値を挿入したとします。

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

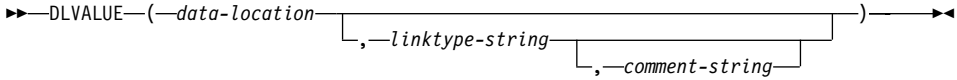
  そして、その値を次の関数で処理します。

  ```
  DLURLSERVER(COLA)
  ```

  次の値が戻されます。

  ```
  DLFS.ALMADEN.IBM.COM
  ```

DLVALUE



スキーマは SYSIBM です。

DLVALUE 関数は DATALINK 値を戻します。この関数は、UPDATE ステートメント内の SET 文節の右辺にあるか、または INSERT ステートメント内の VALUES 文節にあるとき、通常ファイルへのリンクも作成します。ただし、コメントだけが指定されている (data-location がゼロ長ストリングである) 場合、DATALINK 値はリンケージ属性を指定せずに作成されるため、ファイル・リンクはありません。

data-location
リンク・タイプが URL である場合、これは完全 URL 値を含む可変長文字ストリングになる式です。

linktype-string
DATALINK 値のリンク・タイプを指定する任意指定の VARCHAR 式。'URL' (SQLSTATE 428D1) だけが有効な値です。

comment-string
コメントまたは追加の位置情報を提供する任意指定の VARCHAR(254) 値。

この関数の結果は DATALINK 値となります。 DLVALUE 関数のいずれかの引き数がヌル値の可能性がある場合、結果もヌル値になる可能性があります。*data-location* がヌル値の場合、その結果はヌル値になります。

この関数を使用して DATALINK 値を定義する際には、ターゲット値の最大長を考慮してください。たとえば、列が DATALINK(200) で定義されている場合、 *data-location* に *comment* を加えた最大長は 200 バイトとなります。

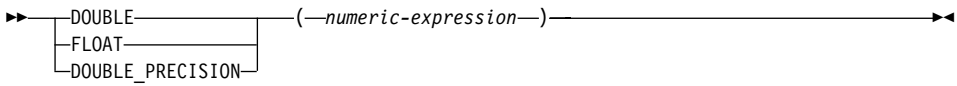
例:

- 1 つの行を表に挿入します。最初の 2 つのリンクの URL 値は、 url_article および url_snapshot という名前の変数内に入っています。url_snapshot_comment という名前の変数には、スナップショット・リンクに付随するコメントが含まれています。現時点では、ムービー用のリンクはなく、 url_movie_comment という名前の変数にコメントだけを含めることができます。

```
EXEC SQL  INSERT INTO HOCKEY_GOALS
            VALUES('Maurice Richard',
                   'Montreal Canadien',
                   '?',
                   'Boston Bruins,
                   '1952-04-24',
                   'Winning goal in game 7 of Stanley Cup final',
                    DLVALUE(:url_article),
                    DLVALUE(:url_snapshot, 'URL', :url_snapshot_comment),
                    DLVALUE('', 'URL', :url_movie_comment) );
```

DOUBLE

数値 → 倍精度:



文字ストリング → 倍精度:

スキーマは SYSIBM です。 ただし、DOUBLE(*string-expression*) のスキーマは SYSFUN です。

DOUBLE 関数は、以下に対応する浮動小数点数を戻します。

- 引き数が数式の場合は、数値。
- 引き数が文字ストリング式の場合は、数値の文字ストリング表記。

数値 → 倍精度

numeric-expression

引き数は、任意の組み込み数値データ・タイプの値を戻す式です。

関数の結果は倍精度浮動小数点数になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

結果は、引き数が倍精度浮動小数点の列、または変数に割り当てられた場合の結果と同じ数値になります。

文字ストリング → 倍精度

string-expression

引き数のタイプは、数値定数形式の CHAR または VARCHAR にすることができます。引き数に先行ブランクや後続ブランクがあっても、それは無視されます。

関数の結果は倍精度浮動小数点数になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

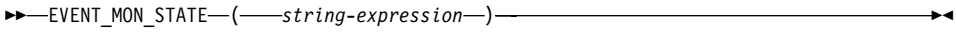
結果は、ストリングが定数とみなされ、倍精度浮動小数点の列または変数に割り当てられる場合の結果と同じ数値になります。

例:

EMPLOYEE 表を使用して、歩合がゼロではない従業員の給与と歩合の比率を計算します。関係する列 (SALARY と COMM) のデータ・タイプは DECIMAL です。結果が範囲外にならないようにするため、DOUBLE を SALARY に適用して、除算が浮動小数点数で実行されるようにします。

```
SELECT EMPNO, DOUBLE(SALARY)/COMM
  FROM EMPLOYEE
  WHERE COMM > 0
```

EVENT_MON_STATE



スキーマは SYSIBM です。

EVENT_MON_STATE 関数は、イベント・モニターの現在の状態を戻します。

引き数は、結果のタイプが CHAR または VARCHAR で、値がイベント・モニターの名前であるストリング式です。指定したイベント・モニターが SYSCAT.EVENTMONITORS カタログ表にない場合は、 SQLSTATE 42704 が戻されます。

結果は、次のいずれかの値の整数になります。

•

0 イベント・モニターは非活動状態です。

1 イベント・モニターは活動状態です。

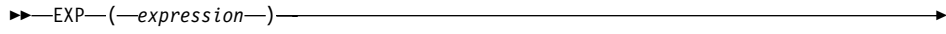
引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例:

- 次の例は、定義済みのイベント・モニターすべてを選択して、各モニターが活動状態か非活動状態かを示すものです。

```
SELECT EVMONNAME,
   CASE
      WHEN EVENT_MON_STATE(EVMONNAME) = 0 THEN 'Inactive'
      WHEN EVENT_MON_STATE(EVMONNAME) = 1 THEN 'Active'
   END
FROM SYSCAT.EVENTMONITORS
```

EXP



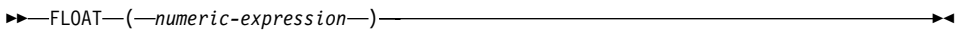
スキーマは SYSFUN です。

引き数に対する指数関数値を戻します。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

FLOAT

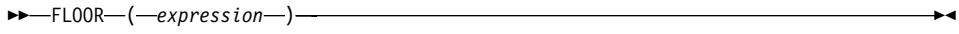


スキーマは SYSIBM です。

FLOAT 関数は、数値の浮動小数点数表記を戻します。

FLOAT は DOUBLE の同義語です。詳細については、319ページの『DOUBLE』 を参照してください。

FLOOR



スキーマは SYSFUN です。

引き数よりも小さいか等しい整数で、最大の整数値を戻します。

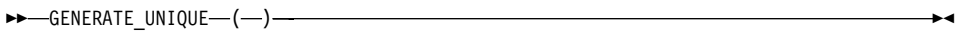
引き数は、任意の組み込み数値タイプにすることができます。引き数のタイプが DECIMAL または REAL である場合、関数による処理に必要な倍精度浮動小数点数に変換されます。引き数のタイプが SMALLINT、INTEGER または BIGINT である場合、その引き数値が戻されます。

関数の結果は次のとおりです。

- 引き数が SMALLINT の場合は SMALLINT になります。
- 引き数が INTEGER の場合は INTEGER になります。
- 引き数が BIGINT の場合は BIGINT になります。
- 引き数が DOUBLE、DECIMAL または REAL の場合は DOUBLE になります。小数部の左側にある桁数が 15 桁を超える 10 進数値は、 DOUBLE への変換での精度の低下により、必要な整数値は戻されません。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

GENERATE_UNIQUE



スキーマは SYSIBM です。

GENERATE_UNIQUE 関数は、同一関数の他の実行と比較して固有である、13 バイト長のビット・データ文字ストリング (CHAR(13) FOR BIT DATA) を戻します。[41] この関数は、deterministic 関数ではないものとして定義されます。

この関数には引き数がありません (空の括弧を指定する必要があります)。

関数の結果は、内部形式の世界標準時 (UTC)、および関数が処理された区分番号を含む固有な値になります。結果がヌル値になることはありません。

この関数の結果を使用して、表の固有値を用意することができます。後続の各値は直前の値より大きくなり、表で使用できる順序列を提供します。 値には、関数が実行された区分の番号が含まれ、それにより、複数の区分にわたって区分化された表もある順序列の固有値を持つことになります。この順序列は、関数が実行された時刻に基づいています。

この関数は、特殊レジスター CURRENT TIMESTAMP を使用する場合とは異なります。この特殊レジスターの場合、複数行の挿入ステートメントまたは全選択を伴う挿入ステートメントの各行について固有値が生成されます。

この関数の結果の一部であるタイム・スタンプ値は、 GENERATE_UNIQUE の結果を引き数にする TIMESTAMP スカラー関数を使用して決定することができます。

例:

- 行ごとに固有な列を含む表を作成します。 GENERATE_UNIQUE 関数を使用してこの列を移植します。 UNIQUE_ID 列には、列をビット・データ文字ストリングとして識別するために ″FOR BIT DATA″ が指定されていることに注意してください。

```
CREATE TABLE EMP_UPDATE
(UNIQUE_ID CHAR(13) FOR BIT DATA,
 EMPNO CHAR(6),
 TEXT  VARCHAR(1000))
```

41. システムの刻時機構は、関数が実行される区分番号とともに、内部の世界標準時 (UTC) タイム・スタンプを生成するのに使用されます。実際のシステム刻時機構を逆方向へ動かす調整を行うと、値が重複する場合があります。

```
INSERT INTO EMP_UPDATE
          VALUES (GENERATE_UNIQUE(), '000020', 'Update entry...'),
                 (GENERATE_UNIQUE(), '000050', 'Update entry...')
```

この表には、行ごとに固有の識別子があります。ただし、UNIQUE_ID 列が、常に GENERATE_UNIQUE を使用して設定されている場合です。これは、表にトリガーを導入することによって行うことができます。

```
CREATE TRIGGER EMP_UPDATE_UNIQUE
NO CASCADE BEFORE INSERT ON EMP_UPDATE
REFERENCING NEW AS NEW_UPD
FOR EACH ROW MODE DB2SQL
SET NEW_UPD.UNIQUE_ID = GENERATE_UNIQUE()
```

このトリガーを定義すると、以下のように最初の列を指定せずに上記の INSERT ステートメントを出すことができます。

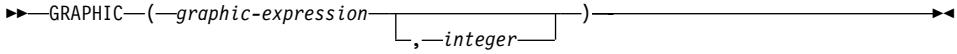
```
INSERT INTO EMP_UPDATE (EMPNO, TEXT)
           VALUES ('000020', 'Update entry 1...'),
                  ('000050', 'Update entry 2...')
```

行が EMP_UPDATE に追加された時点のタイム・スタンプ (UTC における) は、以下を使用して戻すことができます。

```
SELECT TIMESTAMP (UNIQUE_ID), EMPNO, TEXT  FROM EMP_UPDATE
```

したがって、表内のタイム・スタンプ列を行の挿入時に記録する必要はありません。

GRAPHIC



スキーマは SYSIBM です。

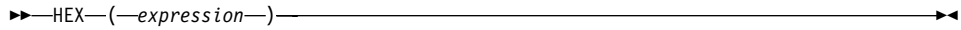
GRAPHIC 関数は、漢字ストリング・タイプの GRAPHIC 表記を戻します。

graphic-expression
漢字ストリング値を戻す式。

integer
結果の GRAPHIC データ・タイプの長さ属性を指定する整数値。値は 1 ～ 127 の範囲でなければなりません。 *integer* を指定しない場合、結果の長さは、最初の引き数の長さと同じになります。

関数の結果は GRAPHIC になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

HEX



スキーマは SYSIBM です。

HEX 関数は、値の 16 進表記を文字ストリングとして戻します。

引き数には、最大長 16 336 バイトの任意の組み込みデータ・タイプの値である式を使うことかできます。

この関数の結果は文字ストリングです。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値である場合、結果はヌル値です。

コード・ページはデータベース・コード・ページになります。

結果は、16 進数のストリングです。最初の 2 つは引き数の最初のバイト、次の 2 つは引き数の 2 番目のバイトを表します。以下同様です。引き数が日付 / 時刻値または数値である場合、結果は引き数の内部形式の 16 進表記になります。戻される 16 進表記は、関数が実行されるアプリケーション・サーバーによって異なる場合があります。違いが生じる場合としては、次のような場合があります。

- EBCDIC サーバーに対する ASCII クライアントまたは ASCII サーバーに対する EBCDIC クライアントで、文字ストリング引き数を指定して HEX 関数を実行したとき。
- クライアント・システムとサーバー・システムとで数値のバイト配列が異なる場合に、 HEX 関数に数値引き数を指定したとき (場合によります)。

結果のタイプと長さは、文字ストリング引き数のタイプと長さによって異なります。

- 文字ストリング
  - 127 以下の固定長
    - 結果は、引き数について定義されている長さの 2 倍の長さの固定長文字ストリングになります。
  - 127 を超える固定長
    - 結果は、引き数について定義されている長さの 2 倍の長さの可変長文字ストリングになります。
  - 可変長

  - 結果は、引き数について定義されている最大長さの 2 倍を最大長とする可変長文字ストリングになります。
- 漢字ストリング
  - 63 以下の固定長
    - 結果は、引き数について定義されている長さの 4 倍の長さの固定長文字ストリングになります。
- 63 を超える固定長
  - 結果は、引き数について定義されている長さの 4 倍の長さの可変長文字ストリングになります。
- 可変長
  - 結果は、引き数について定義されている最大の長さの 4 倍を最大長とする可変長文字ストリングになります。

例:

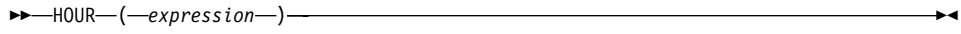
以下の例では、DB2 (AIX 版) アプリケーション・サーバーを使用することを前提にしています。

- DEPARTMENT 表を使用して、ホスト変数 HEX_MGRNO (char(12)) に、'PLANNING' 部門 (DEPTNAME) の管理者番号 (MGRNO) の 16 進表記を設定します。

```
SELECT HEX(MGRNO)
  INTO :HEX_MGRNO
  FROM DEPARTMENT
  WHERE DEPTNAME = 'PLANNING'
```

  サンプル表を使用した場合、HEX_MGRNO は '303030303230' に設定されます (文字値は '000020')。
- COL_1 が、データ・タイプ char(1)、値 'B' の列であるとします。英字 'B' の 16 進数表記は X'42' です。 HEX(COL_1) は 2 文字のストリング '42' を戻します。
- COL_3 が、データ・タイプ decimal(6,2)、値 40.1 の列であるとします。 10 進数値 40.1 の内部表記に HEX 関数を適用した結果は、 8 文字のストリング '0004010C' になります。

HOUR



スキーマは SYSIBM です。

HOUR 関数は、値の時の部分を戻します。

引き数は、時刻、タイム・スタンプ、時刻期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない時刻またはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

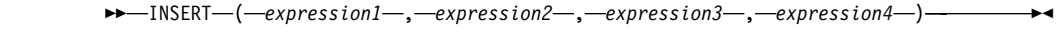
- 引き数が、時刻、タイム・スタンプ、または時刻やタイム・スタンプの有効なストリング表記の場合
  - 結果は、値の時間の部分 (0 ～ 24 の整数) になります。
- 引き数が時刻期間またはタイム・スタンプ期間の場合
  - 結果は、値の時の部分 (-99 ～ 99 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

CL_SCHED サンプル表を使用して、午後に始まるすべてのクラスを選択します。

```
SELECT * FROM CL_SCHED
  WHERE HOUR(STARTING) BETWEEN 12 AND 17
```

INSERT



スキーマは SYSFUN です。

expression2 から始まる *expression1* から *expression3* に等しいバイト数が削除され、 *expression2* から始まる *expression1* に *expression4* が挿入されたストリングが戻されます。結果のストリングの長さが戻りタイプの最大値を超える場合、エラーが生じます (SQLSTATE 38552)。

最初の引き数は、文字ストリングまたは 2 進ストリングです。 2 番目および 3 番目の引き数は、データ・タイプが SMALLINT または INTEGER の数値でなければなりません。最初の引き数は、文字ストリングまたは 2 進ストリングです。最初の引き数が 2 進ストリングである場合、 4 番目の引き数は 2 進ストリングでなければなりません。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB または 2 進ストリングの場合、最大長は 1 048 576 バイトです。最初の引き数および 4 番目の引き数については、CHAR が VARCHAR に変換され、 LONG VARCHAR が CLOB(1M) に変換されます。 2 番目および 3 番目の引き数の場合、この関数による処理に必要なので、 SMALLINT が INTEGER に変換されます。

結果は、以下のように引き数のタイプに基づきます。

- 最初および 4 番目の引き数がいずれも (4 000 バイトを超えない) VARCHAR または CHAR である場合、 VARCHAR(4000) になります。
- 最初または 4 番目の引き数のいずれかが CLOB または LONG VARCHAR である場合、 CLOB(1M) になります。
- 最初および 4 番目の引き数がいずれも BLOB である場合、BLOB(1M) になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

例:

- 語 'DINING' から 1 文字を削除して、'VID' を挿入します。いずれも 3 番目の文字から始まります。

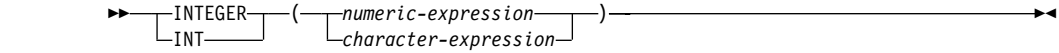
```
VALUES CHAR(INSERT('DINING', 3, 1, 'VID'), 10)
```

この例では、以下を戻します。

```
1
----------
DIVIDING
```

前述のように、INSERT 関数の出力は VARCHAR(4000) になります。上記の例の場合、関数 CHAR が、INSERT の出力を 10 バイトに制限するために使用されています。特定のストリングの開始位置は、LOCATE を使用して見つけることができます。詳細については、341ページの『LOCATE』を参照してください。

INTEGER



スキーマは SYSIBM です。

INTEGER 関数は、数値または文字ストリングを表した整数を、整数定数の形で戻します。

numeric-expression
組み込み数値データ・タイプの値を戻す式。

引き数が*数値式* の場合、結果は、引き数を長精度整数 (large integer) の列または変数に割り当てた場合と同じ数値になります。引き数の整数部分が整数の範囲内にない場合、エラーになります。引き数に小数部分がある場合は、切り捨てられます。

character-expression
文字定数の最大長以下の長さの文字ストリング値を戻す式。先行ブランクと後書きブランクは削除されます。その結果のストリングは、 SQL 整数定数を形成するための規則に従うものでなければなりません (SQLSTATE 22018)。文字ストリングとして、長ストリングを使うことはできません。

引き数が*文字式* の場合、結果は、対応する整数定数を長精度整数の列または変数に割り当てた場合の数値と同じになります。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例:

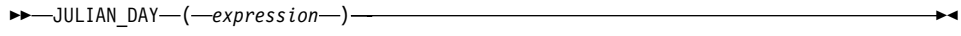
- EMPLOYEE 表を使用して、給与 (SALARY) を学歴 (EDLEVEL) 値で除算した値を含むリストを選択します。計算では小数部をすべて切り捨てます。リストには、計算に使用される値と従業員番号 (EMPNO) も含めます。リストは、計算値の降順に配列します。

```
SELECT INTEGER (SALARY / EDLEVEL), SALARY, EDLEVEL, EMPNO
  FROM EMPLOYEE
  ORDER BY 1 DESC
```

- EMPLOYEE 表を使用して、アプリケーションでさらに処理を行うために、EMPNO を整数形式として選択します。

```
SELECT INTEGER(EMPNO) FROM EMPLOYEE
```

JULIAN_DAY



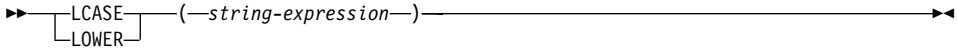
スキーマは SYSFUN です。

紀元前 4712 年 1 月 1 日 (ユリウス暦の起点) からの経過日数を示す整数値を、引き数で指定した日付値に戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

LCASE または LOWER



スキーマは SYSIBM です。[42]

LCASE または LOWER 関数は、すべての SBCS 文字が小文字に変換されたストリングを戻します。 (つまり、文字 A ～ Z は文字 a ～ z に変換され、発音符付きの文字は小文字に相当するものがあればその文字に変換されます。たとえば、コード・ページ 850 では、É は é に変換されます。) 必ずすべての文字が変換されるわけではないので、 LCASE(UCASE(*string-expression*)) の結果が LCASE(*string-expression*) と同じになるとは限りません。

引き数は、値が CHAR または VARCHAR データ・タイプの式でなければなりません。

関数の結果のデータ・タイプと長さ属性は、引き数と同じになります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

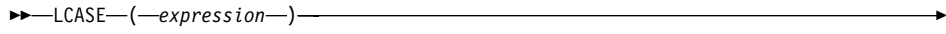
例: EMPLOYEE 表の列 JOB の値で使用されている文字を小文字に変換します。

```
SELECT LCASE(JOB)
  FROM EMPLOYEE WHERE EMPNO = '000020';
```

結果は、値 'manager' になります。

42. この関数の SYSFUN バージョンでは、 LONG VARCHAR 引き数と CLOB 引き数のサポートが引き続き有効です。説明については、336ページの『LCASE (SYSFUN スキーマ)』を参照してください。

LCASE (SYSFUN スキーマ)



スキーマは SYSFUN です。

文字 A ～ Z のすべてが文字 a ～ z に変換されたストリングを戻します (分音符号のついた文字は変換されません)。したがって、LCASE(UCASE(string)) の結果が LCASE(string) と同じになるとは限らない点に注意してください。

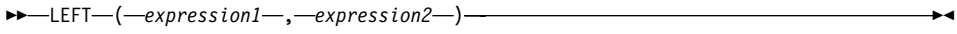
引き数は、任意の組み込み文字ストリング・タイプにすることができます。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB の場合、最大長は 1 048 576 バイトです。

関数の結果は次のとおりです。

- 引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

LEFT



スキーマは SYSFUN です。

expression1 の左端 *expression2* バイトからなるストリングを戻します。*expression1* 値の右側には必要な数のブランク文字が効率的に付加されて、*expression1* のうちの指定したサブストリングが常に存在するようにされます。

最初の引き数は、文字ストリングまたは 2 進ストリングです。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB または 2 進ストリングの場合、最大長は 1 048 576 バイトです。 2 番目の引き数のデータ・タイプは INTEGER または SMALLINT でなければなりません。

関数の結果は次のとおりです。

- 引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。
- 引き数が BLOB の場合は BLOB(1M) になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

LENGTH

スキーマは SYSIBM です。

LENGTH 関数は、値の長さを戻します。

引き数としては、任意の組み込みデータ・タイプの値を戻す式を使えます。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

結果は引き数の長さです。その長さには、ヌル値が可能な列引き数のヌル値標識バイトは含まれません。ストリングの長さにブランクは含まれますが、可変長ストリングの長さ制御フィールドは含まれません。可変長ストリングの長さは、最大長ではなく実際の長さです。

漢字ストリングの長さは、DBCS 文字の数になります。それ以外のすべての値の場合の長さは、その値を表すために使用されるバイトの数になります。

- 短精度整数の場合は 2
- 長精度整数の場合は 4
- 精度 *p* の 10 進数の場合は (*p*/2)+1
- 2 進ストリングの場合は、そのストリングの長さ
- 文字ストリングの場合は、そのストリングの長さ
- 単精度浮動小数点の場合は 4
- 倍精度浮動小数点の場合は 8
- 日付の場合は 4
- 時刻の場合は 3
- タイム・スタンプの場合は 10

例:

- ホスト変数 ADDRESS が、 '895 Don Mills Road' という値を持つ可変長文字ストリングであると想定します。

  ```
  LENGTH(:ADDRESS)
  ```

  戻り値は 18 です。
- START_DATE が DATE タイプの列であると想定します。

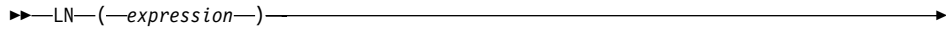
```
LENGTH(START_DATE)
```

この例では、4 の値を戻します。

- 次の例は、10 の値を戻します。

```
LENGTH(CHAR(START_DATE, EUR))
```

LN



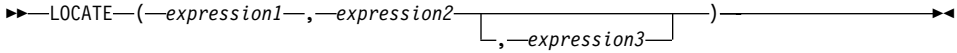
スキーマは SYSFUN です。

引き数の自然対数値を戻します (LOG と同じ)。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

LOCATE



スキーマは SYSFUN です。

expression2 内の *expression1* の最初のオカレンスの開始位置を戻します。オプションの *expression3* が指定されている場合、その値は、探索を開始する *expression2* の文字位置を示します。 *expression1* が *expression2* 内にない場合、値 0 が戻されます。

最初の引き数が文字ストリングである場合、 2 番目の引き数は文字ストリングでなければなりません。 VARCHAR の場合、最大長は 4 000 バイトです。CLOB の場合、最大長は 1 048 576 バイトです。最初の引き数が 2 進ストリングである場合、2 番目の引き数は、最大長が 1 048 576 バイトの 2 進ストリングでなければなりません。 3 番目の引き数は INTEGER または SMALLINT でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

例:

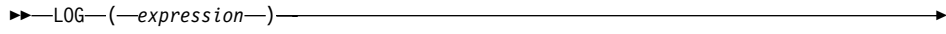
- 語 'DINING' 内の英字 'N' (最初のオカレンス) の位置を検出します。

  ```
  VALUES LOCATE ('N', 'DINING')
  ```

  この例では、以下を戻します。

  ```
  1
  -----------
  3
  ```

LOG



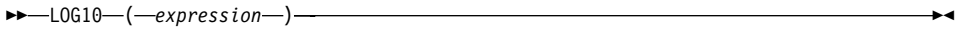
スキーマは SYSFUN です。

引き数の自然対数値を戻します (LN と同じ)。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

LOG10



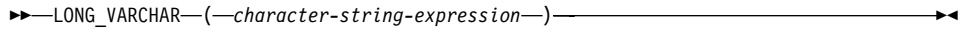
スキーマは SYSFUN です。

引き数に対する、10 を底とする対数 (常用対数) を戻します。

引き数は、任意の組み込み数値タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

LONG_VARCHAR



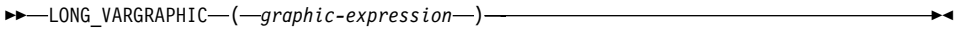
スキーマは SYSIBM です。

LONG_VARCHAR 関数は、文字ストリング・データ・タイプの LONG VARCHAR 表記を戻します。

character-string-expression
最大長が 32 700 バイトの文字ストリング値を戻す式。

関数の結果は LONG VARCHAR になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

LONG_VARGRAPHIC



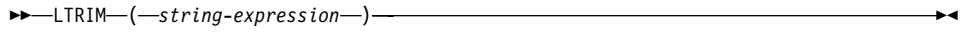
スキーマは SYSIBM です。

LONG_VARGRAPHIC 関数は、2 バイト文字ストリングの LONG VARGRAPHIC 表記を戻します。

graphic-expression
最大長が 16 350 個の 2 バイト文字である漢字ストリング値を戻す式。

関数の結果は LONG VARGRAPHIC になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

LTRIM



スキーマは SYSIBM です。[43]

LTRIM 関数は、*string-expression* の先頭からブランクを除去します。

引き数には CHAR、VARCHAR、GRAPHIC、または VARGRAPHIC データ・タイプを使用できます。

- 引き数が DBCS または EUC データベースの漢字ストリングである場合は、先行 2 バイト・ブランクが除去されます。
- 引き数が Unicode データベースの漢字ストリングである場合は、先行 UCS-2 ブランクが除去されます。
- それ以外の場合は、先行 1 バイト・ブランクが除去されます。

この関数の結果のデータ・タイプは次のとおりです。

- *string-expression* のデータ・タイプが VARCHAR または CHAR の場合は VARCHAR になります。
- *string-expression* のデータ・タイプが VARGRAPHIC または GRAPHIC の場合は VARGRAPHIC になります。

戻されるタイプの長さパラメーターは、引き数のデータ・タイプの長さパラメーターと同じになります。

結果が文字ストリングである場合の実際の長さは、除去されるブランク文字のバイト数を *string-expression* から引いた値になります。結果が漢字ストリングである場合の実際の長さは、除去される 2 バイト・ブランク文字の数を *string-expression* から引いた値 (2 バイト文字単位) になります。すべての文字が除去された場合、結果は空になり、可変長ストリング (長さゼロ) が戻されます。

引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例: ホスト変数 HELLO が CHAR(9) として定義されており、値は 'Hello ' であるものとします。

---

43. この関数の SYSFUN バージョンでは、 LONG VARCHAR 引き数と CLOB 引き数のサポートが引き続き有効です。説明については、348ページの『LTRIM (SYSFUN スキーマ)』を参照してください。

```
VALUES LTRIM(:HELLO)
```

結果は ’Hello’ になります。

LTRIM (SYSFUN スキーマ)

スキーマは SYSFUN です。

引き数の文字から先行ブランクを除去して戻します。

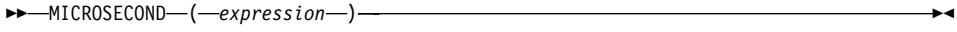
引き数は、任意の組み込み文字ストリング・タイプにすることができます。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB の場合、最大長は 1 048 576 バイトです。

関数の結果は次のとおりです。

- 引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

MICROSECOND



スキーマは SYSIBM です。

MICROSECOND 関数は、値のマイクロ秒の部分を戻します。

引き数は、タイム・スタンプ、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもないタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

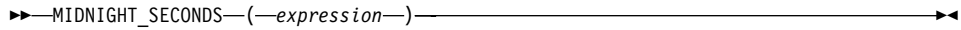
- 引き数がタイム・スタンプまたはタイム・スタンプの有効なストリング表記の場合
  - 整数の範囲は 0 ～ 999 999 となります。
- 引き数が期間の場合
  - 結果には、 -999 999 ～ 999 999 の間の整数値としてのマイクロ秒部分が反映されます。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- 表 TABLEA に、タイプが TIMESTAMP の TS1 および TS2 という 2 つの列が含まれるものとします。 TS1 のマイクロ秒部分がゼロではなく、TS1 と TS2 の秒部分が同じである行すべてを選択します。

```
SELECT * FROM TABLEA
  WHERE MICROSECOND(TS1) <> 0 AND
  SECOND(TS1) = SECOND(TS2)
```

MIDNIGHT_SECONDS



スキーマは SYSFUN です。

午前 0 時から引き数で指定した時刻値までの秒数を表す 0 ～ 86 400 の範囲の整数値を戻します。

引き数は、時刻、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない時刻またはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

例:

- 午前 0 時から 00:10:10 までの秒数、および午前 0 時から 13:10:10 までの秒数を求めます。

  ```
  VALUES (MIDNIGHT_SECONDS('00:10:10'), MIDNIGHT_SECONDS('13:10:10'))
  ```

  この例では、以下を戻します。

  ```
  1           2
  ----------- -----------
          610       47410
  ```

  1 分は 60 秒なので、午前 0 時から指定された時刻までは 610 秒です。 2 番目の例でも同じです。 1 時間は 3600 秒であり、1 分は 60 秒なので、指定された時刻から午前 0 時までは 47410 秒です。
- 午前 0 時から 24:00:00 までの秒数、および午前 0 時から 00:00:00 までの秒数を求めます。

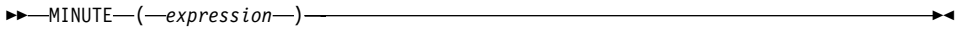
  ```
  VALUES (MIDNIGHT_SECONDS('24:00:00'), MIDNIGHT_SECONDS('00:00:00'))
  ```

  この例では、以下を戻します。

  ```
  1           2
  ----------- -----------
        86400           0
  ```

  これらの 2 つの値は、同じ時刻を表しているにもかかわらず、MIDNIGHT_SECONDS 値が異なっていることに注意してください。

MINUTE



スキーマは SYSIBM です。

MINUTE 関数は、値の分の部分を戻します。

引き数は、時刻、タイム・スタンプ、時刻期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない時刻またはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

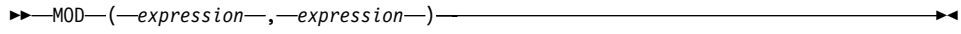
- 引き数が、時刻、タイム・スタンプ、または時刻やタイム・スタンプの有効なストリング表記の場合
  - 結果は、値の分の部分 (0 ～ 59 の整数) になります。
- 引き数が時刻期間またはタイム・スタンプ期間の場合
  - 結果は、値の分の部分 (-99 ～ 99 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- CL_SCHED サンプル表を使用して、授業時間が 50 分未満のクラスをすべて選択します。

```
SELECT * FROM CL_SCHED
  WHERE HOUR(ENDING - STARTING) = 0 AND
  MINUTE(ENDING - STARTING) < 50
```

MOD



スキーマは SYSFUN です。

最初の引き数を 2 番目の引き数で割った剰余を戻します。結果は、最初の引き数が負である場合にのみ負になります。

関数の結果は次のとおりです。

- 両方の引き数が SMALLINT の場合は SMALLINT になります
- 一方の引き数が INTEGER で、他方が INTEGER または SMALLINT の場合は INTEGER になります
- 一方の引き数が BIGINT で、他方が BIGINT、INTEGER または SMALLINT の場合は BIGINT になります

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

MONTH

スキーマは SYSIBM です。

MONTH 関数は、値の月の部分を戻します。

引き数は、日付、タイム・スタンプ、日付期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

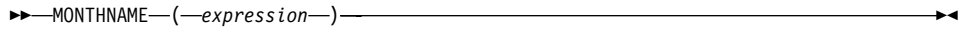
- 引き数が日付、タイム・スタンプ、または日付やタイム・スタンプの有効なストリング表記の場合
  - 結果は、値の月の部分 (1 ～ 12 の整数) になります。
- 引き数が日付期間またはタイム・スタンプ期間の場合
  - 結果は、値の月の部分 (-99 ～ 99 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- EMPLOYEE 表から、12 月に生まれた (BIRTHDATE) 社員の行をすべて選択します。

```
SELECT * FROM EMPLOYEE
  WHERE MONTH(BIRTHDATE) = 12
```

MONTHNAME



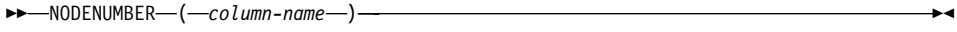
スキーマは SYSFUN です。

データベースが始動された時点のロケールに基づいて、引き数の月の部分の名前 (例: January) を含む大文字小文字混合文字ストリングを戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は VARCHAR(100) です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

NODENUMBER



スキーマは SYSIBM です。

NODENUMBER 関数は、行の区分番号を戻します。たとえば、SELECT 文節で使用すると、 SELECT ステートメントの結果の生成に使用された表の各行の区分番号を戻します。

変換変数および表に戻される区分番号は、区分化キー列の現行変換値から得られます。たとえば、挿入前トリガーにおいて、新しい変換変数の現行値があれば、関数は予定区分番号を戻します。ただし、区分化キー列の値はそれ以後の挿入前トリガーによって変更される場合があります。したがって、データベースに挿入される時点での行の最終区分番号は、予定値とは異なるかもしれません。

引き数は、表の列の修飾された名前または無修飾の名前でなければなりません。該当の列は、どのようなデータ・タイプであっても構いません。[44] *column-name* が視点の列を参照する場合、その列の視点の式は基礎表の列を参照する必要があり、その視点は削除可能でなければなりません。ネストされているか、または共通の表式は、視点と同じ規則に従います。削除可能な視点の定義については、904ページの『注』を参照してください。

NODENUMBER 関数によって区分番号が戻される特定の行 (または表) は、この関数を使用する SQL ステートメントの文脈から判別されます。

結果のデータ・タイプは INTEGER であり、ヌル値にはなりません。行レベルの情報が戻されるので、どの列が表に指定されるかに関係なく、結果は同じです。 `db2nodes.cfg` ファイルがない場合、結果は 0 になります。

NODENUMBER 関数は、複製された表、検査制約内、または生成された列の定義で使用することはできません (SQLSTATE 42881)。

例:

- EMPLOYEE の行が DEPARTMENT の従業員の部門記述とは異なる区分上にある行の数を数えます。

44. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべてのデータ・タイプを引き数として受け入れるので、ユーザー定義特殊タイプをサポートするための追加のシグニチャーを作成する必要はありません。

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
        WHERE D.DEPTNO=E.WORKDEPT
        AND NODENUMBER(E.LASTNAME) <> NODENUMBER(D.DEPTNO)
```

- 2 つの表の行が同じ区分にある EMPLOYEE および DEPARTMENT の表を結合します。

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
        WHERE NODENUMBER(E.LASTNAME) = NODENUMBER(D.DEPTNO)
```

- 表 EMPLOYEE で前トリガーを作成して、従業員の挿入の際には必ず、EMPINSERTLOG1 という表に従業員番号と新しい行の予定区分番号を記録します。

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH MODE ROW MODE DB2SQL
INSERT INTO EMPINSERTLOG1
VALUES(NEWTABLE.EMPNO, NODENUMBER(NEWTABLE.EMPNO))
```

NULLIF

スキーマは SYSIBM です。

NULLIF 関数は、引き数が等しい場合はヌル値を戻し、それ以外の場合には最初の引き数の値を戻します。

2 つの引き数は比較可能でなければなりません (106ページの『割り当てと比較』 を参照)。それらは、組み込みデータ・タイプ (長ストリングまたは DATALINK 以外) か、まったく異なるデータ・タイプ (長ストリングまたは DATALINK に基づくもの以外) のいずれかにすることができます。 [45] 結果の属性は、最初の引き数の属性になります。

NULLIF(e1,e2) を使用した結果は、次の式を使用した結果と同じになります。

```
CASE WHEN e1=e2 THEN NULL ELSE e1 END
```

一方または両方の引き数が NULL で、e1=e2 が未知として評価されると、CASE 式はこれを真ではないとみなします。したがって、この場合、NULLIF は最初の引き数の値を戻します。

例:

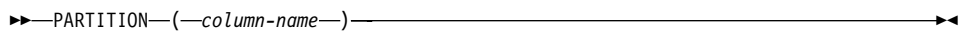
- ホスト変数 PROFIT、CASH、および LOSSES のデータ・タイプが DECIMAL で、値がそれぞれ 4500.00、500.00、および 5000.00 であるとします。

  ```
  NULLIF (:PROFIT + :CASH , :LOSSES )
  ```

  ヌル値が戻されます。

45. この関数は、ユーザー定義関数の作成時にソース関数として使用しなくてもかまいません。この関数は、すべての互換データ・タイプを引き数として受け入れるので、ユーザー定義特殊タイプをサポートするための追加のシグニチャーを作成する必要はありません。

PARTITION



スキーマは SYSIBM です。

PARTITION 関数は、区分化関数を行の区分化キー値に適用することによって入手された行の区分化マップ索引を戻します。たとえば、SELECT 文節で使用すると、その SELECT ステートメントの結果の生成に使用された表の各行の区分化マップ索引を戻します。

変換変数および表に戻される区分化マップ索引は、区分化キー列の現行変換値から派生します。たとえば、挿入前トリガーにおいて、新しい変換変数の現行値があれば、関数は予定区分化マップ索引を戻します。ただし、区分化キー列の値はそれ以後の挿入前トリガーによって変更される場合があります。したがって、データベースに挿入される時点で、行の最終区分化マップ索引は予定値と異なるかもしれません。

引き数は、表の列の修飾された名前または無修飾の名前でなければなりません。該当の列は、どのようなデータ・タイプであっても構いません。[46] *column-name* が視点の列を参照する場合、その列の視点の式は基礎表の列を参照する必要があり、その視点は削除可能でなければなりません。ネストされているか、または共通の表式は、視点と同じ規則に従います。削除可能な視点の定義については、904ページの『注』を参照してください。

PARTITION 関数によって区分化マップ索引が戻される特定の行 (または表) は、この関数を使用する SQL ステートメントの文脈から判別されます。

結果のデータ・タイプは、0 ～ 4095 の範囲の INTEGER です。区分化キーのない表の場合、結果は常に 0 になります。ヌル値が戻されることはありません。行レベルの情報が戻されるので、どの列が表に指定されるかに関係なく、結果は同じです。

PARTITION 関数は、複製された表、検査制約内、または生成された列の定義で使用することはできません (SQLSTATE 42881)。

例:

---

46. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべてのデータ・タイプを引き数として受け入れるので、ユーザー定義特殊タイプをサポートするための追加のシグニチャーを作成する必要はありません。

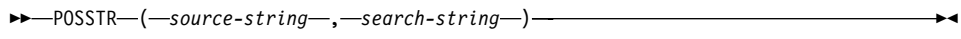
- 区分化マップ索引が 100 であるすべての行について、 EMPLOYEE 表から従業員番号 (EMPNO) をリストします。

```
SELECT EMPNO FROM EMPLOYEE
  WHERE PARTITION(PHONENO) = 100
```

- 表 EMPLOYEE で前トリガーを作成して、従業員の挿入の際には必ず、EMPINSERTLOG2 という表に従業員番号と新しい行の予定区分化マップ索引を記録します。

```
CREATE TRIGGER EMPINSLOGTRIG2
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH MODE ROW MODE DB2SQL
INSERT INTO EMPINSERTLOG2
  VALUES(NEWTABLE.EMPNO, PARTITION(NEWTABLE.EMPNO))
```

POSSTR



スキーマは SYSIBM です。

POSSTR 関数は、あるストリング (*source-string*、ソース・ストリングと呼ばれる) の中で、別のストリング (*search-string*、探索ストリングと呼ばれる) の最初の出現箇所の開始位置を戻します。 *search-string* の位置を示す数値は、1 から始まります (0 ではない)。

この関数の結果は長精度整数 (large integer) です。引き数のいずれかがヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数のいずれかがヌル値の場合、その結果はヌル値です。

source-string
探索が行われる場所としてのソース・ストリングを指定します。

この式は、以下のいずれかによって指定できます。

- 定数
- 特殊レジスター
- ホスト変数 (ロケーター変数またはファイル参照変数を含む)
- スカラー関数
- ラージ・オブジェクトのロケーター
- 列名
- 上記のいずれかを連結する式

search-string
ソース・ストリングから探索するストリングを指定する式。

この式は、以下のいずれかによって指定できます。

- 定数
- 特殊レジスター
- ホスト変数
- 上記のいずれかをオペランドとするスカラー関数
- 上記のいずれかを連結する式

以下の制約があります。

- 式の要素に、LONG VARCHAR、CLOB、LONG VARGRAPHIC、または DBCLOB のタイプを使うことはできません。また、BLOB ファイル参照変数は使えません。
- *search-string* の実際の長さは 4 000 バイト以下でなければなりません。

これらの規則は、 219ページの『LIKE 述部』で説明されている *pattern-expression* の規則と同じです。

search-string と *source-string* には、いずれも、ゼロ個以上の連続した位置があります。ストリングが文字ストリングまたは 2 進ストリングの場合、 1 つの位置は 1 バイトを表します。ストリングが漢字ストリングの場合、位置は図形 (DBCS) 文字になります。

POSSTR 関数は混合データ・ストリングを受け入れます。ただし、 POSSTR は、厳密にバイト・カウント単位で計算し、 1 バイト文字と多重バイト文字の間の変更は感知しません。

以下の規則が適用されます。

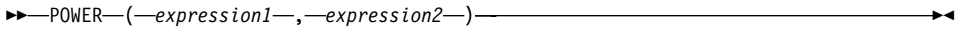
- *source-string* と *search-string* のデータ・タイプは比較可能でなければなりません。そうでない場合、エラーになります (SQLSTATE 42884)。
  - *source-string* が文字ストリングの場合、 *search-string* は CLOB または LONG VARCHAR 以外の文字ストリングでなければならず、実際の長さが 32 672 バイト以下でなければなりません。
  - *source-string* が漢字ストリングの場合、 *search-string* は DBCLOB または LONG VARGRAPHIC 以外の漢字ストリングでなければならず、実際の長さが 16 336 以下でなければなりません。
  - *source-string* が 2 進ストリングである場合、*source-string* は、実際の長さが 32 672 バイト以下の 2 進ストリングでなければなりません。
- *search-string* の長さがゼロの場合、この関数によって戻される結果は 1 です。
- それ以外の場合は、次のとおりです。
  - *source-string* の長さがゼロの場合、関数によって戻される結果はゼロです。
  - それ以外の場合は、次のとおりです。
    - *search-string* が *source-string* の値のうち、連続する複数の位置の同じ長さのサブストリングに等しい場合、この関数によって戻される結果は、*source-string* 値の中でそのようなサブストリングのうち最初の開始位置になります。
    - それ以外の場合、この関数によって戻される結果は 0 です。

例:

- IN_TRAY 表の項目のうち、 NOTE_TEXT 列に 'GOOD BEER' という語句が含まれている項目について、 RECEIVED 列と SUBJECT 列、およびその語句の開始位置を選択します。

```
SELECT RECEIVED, SUBJECT, POSSTR(NOTE_TEXT, 'GOOD BEER')
  FROM IN_TRAY
  WHERE POSSTR(NOTE_TEXT, 'GOOD BEER') <> 0
```

POWER



スキーマは SYSFUN です。

expression1 の *expression2* 乗の値を戻します。

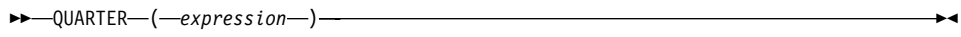
引き数は、任意の組み込み数値データ・タイプにすることができます。DECIMAL および REAL の引き数は倍精度浮動小数点数に変換されます。

関数の結果は次のとおりです。

- 両方の引き数が INTEGER または SMALLINT の場合は INTEGER になります。
- 一方の引き数が BIGINT で、他方が BIGINT、INTEGER または SMALLINT の場合は BIGINT になります。
- それ以外の場合は DOUBLE になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

QUARTER



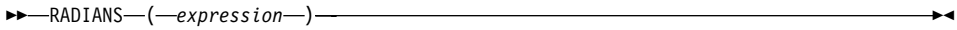
スキーマは SYSFUN です。

引き数に指定した日付が属する 4 半期を示す 1 〜 4 の範囲の整数値を戻します。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

RADIANS



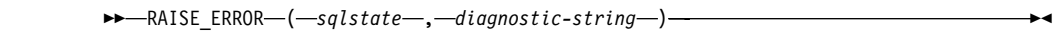
スキーマは SYSFUN です。

度単位の引き数をラジアン単位の角度に変換して戻します。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

RAISE_ERROR



スキーマは SYSIBM です。

RAISE_ERROR 関数は、この関数を含むステートメントが、指定した SQLSTATE、 SQLCODE -438、および *diagnostic-string* のエラーを戻すようにします。 RAISE_ERROR 関数は、未定義データ・タイプでは常に NULL を戻します。

sqlstate
厳密に 5 文字の文字ストリング。これは、長さ 5 として定義された CHAR タイプ、または長さ 5 以上として定義された VARCHAR タイプでなければなりません。 *sqlstate* の値は、次のように、アプリケーション定義の 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 ('A' ～ 'Z') でなければなりません。
- SQLSTATE クラス (最初の 2 文字) は、エラー・クラスではない '00'、'01'、または '02' にすることはできません。
- SQLSTATE クラス (最初の 2 文字) が文字 '0' ～ '6' または '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' の文字で始まっていなければなりません。
- SQLSTATE クラス (最初の 2 文字) が文字 '7'、'8'、'9'、または 'I' ～ 'Z' で始まっている場合、サブクラス (最後の 3 文字) には '0' ～ '9' または 'A' ～ 'Z' を使えます。

SQLSTATE がこれらの規則に従っていない場合は、エラーになります (SQLSTATE 428B3)。

diagnostic-string
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。ストリングが 70 バイトより長い場合は切り捨てられます。

この関数を、結果データ・タイプの規則が適用されない文脈 (選択リストで単独の場合など) で使用するには、 Cast 指定を使用して、データ・タイプにヌル値の戻される値を割り当てる必要があります。 CASE 式は、RAISE_ERROR 関数を使う最適の場所といえます。

例:

従業員番号と学歴のリストを、学歴を Post Graduate、Graduate、および Diploma として示します。学歴が 20 を超える場合は、エラーになります。

```
SELECT EMPNO,
       CASE WHEN EDUCLVL < 16 THEN 'Diploma'
            WHEN EDUCLVL < 18 THEN 'Graduate'
            WHEN EDUCLVL < 21 THEN 'Post Graduate'
            ELSE RAISE_ERROR('70001',
                             'EDUCLVL has a value greater than 20')
       END
     FROM EMPLOYEE
```

RAND

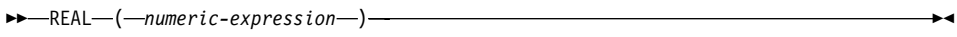
スキーマは SYSFUN です。

引き数を任意選択のシード値として使用して、0 と 1 の間のランダムな浮動小数点数値を戻します。この関数は、deterministic 関数ではないものとして定義されます。

引き数は必須ではありませんが、引き数を指定する場合には INTEGER か SMALLINT のいずれかにすることができます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

REAL



スキーマは SYSIBM です。

REAL 関数は、数値の単精度浮動小数点表記を戻します。

引き数は、任意の組み込み数値データ・タイプの値を戻す式です。

関数の結果は単精度浮動小数点数になります。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

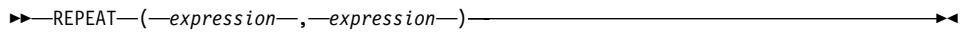
結果は、引き数が単精度浮動小数点の列または変数に割り当てられた場合の結果と同じ数値になります。

例:

EMPLOYEE 表を使用して、歩合がゼロではない従業員の給与と歩合の比率を計算します。関係する列 (SALARY と COMM) のデータ・タイプは DECIMAL です。単精度浮動小数点の結果が必要です。したがって、除算が浮動小数点 (実際には倍精度) で実行されるように REAL が SALARY に適用され、次に単精度の浮動小数点で結果を戻すために REAL が式全体に適用されます。

```
SELECT EMPNO, REAL(REAL(SALARY)/COMM)
  FROM EMPLOYEE
  WHERE COMM > 0
```

REPEAT



スキーマは SYSFUN です。

2 番目の引き数によって指定された回数だけ繰り返した最初の引き数で構成される文字ストリングを戻します。

最初の引き数は、文字ストリングまたは 2 進ストリングです。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB または 2 進ストリングの場合、最大長は 1 048 576 バイトです。 2 番目の引き数は SMALLINT または INTEGER にすることができます。

関数の結果は次のとおりです。

- 最初の引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 最初の引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。
- 最初の引き数が BLOB の場合は BLOB(1M) になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

例:

- 句 'REPEAT THIS' を 5 回リストします。

```
VALUES CHAR(REPEAT('REPEAT THIS', 5), 60)
```

この例では以下が戻されます。

```
1
------------------------------------------------------------
REPEAT THISREPEAT THISREPEAT THISREPEAT THISREPEAT THIS
```

前述のように、REPEAT 関数の出力は VARCHAR(4000) になります。上記の例の場合、関数 CHAR が、 REPEAT の出力を 60 バイトに制限するために使用されています。

REPLACE

スキーマは SYSFUN です。

expression1 における *expression2* のすべてのオカレンスを *expression3* に置き換えます。

最初の引き数は、任意の組み込み文字ストリングまたは 2 進ストリング・タイプにすることができます。 VARCHAR の場合、最大長は 4 000 バイトです。CLOB または 2 進ストリングの場合、最大長は 1 048 576 バイトです。CHAR は VARCHAR に変換され、LONG VARCHAR は CLOB(1M) に変換されます。 2 番目および 3 番目の引き数は、最初の引き数と同一です。

関数の結果は次のとおりです。

- 最初、2 番目、および 3 番目の引き数が VARCHAR または CHAR の場合は VARCHAR(4000) になります。
- 最初、2 番目、および 3 番目の引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。
- 最初、2 番目、および 3 番目の引き数が BLOB の場合は BLOB(1M) になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

例:

- 語 'DINING' 内の英字 'N' のすべてのオカレンスを 'VID' に置き換えます。

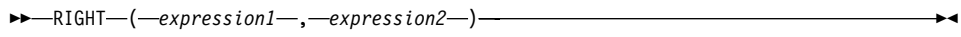
```
VALUES CHAR (REPLACE ('DINING', 'N', 'VID'), 10)
```

この例では、以下を戻します。

```
1
----------
DIVIDIVIDG
```

前述のように、REPLACE 関数の出力は VARCHAR(4000) になります。上記の例の場合、関数 CHAR が、 REPLACE の出力を 10 バイトに制限するために使用されています。

RIGHT



スキーマは SYSFUN です。

expression1 の右端の *expression2* バイトからなるストリングを戻します。 *expression1* 値の右側には必要な数のブランク文字が効率的に付加されて、 *expression1* のうちの指定したサブストリングが常に存在するようにされます。

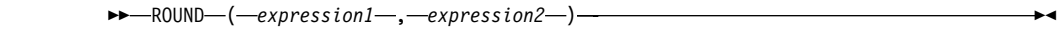
最初の引き数は、文字ストリングまたは 2 進ストリングです。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB または 2 進ストリングの場合、最大長は 1 048 576 バイトです。 2 番目の引き数は INTEGER または SMALLINT にすることができます。

関数の結果は次のとおりです。

- 最初の引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 最初の引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。
- 最初の引き数が BLOB の場合は BLOB(1M) になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

ROUND



スキーマは SYSFUN です。

expression1 を小数点以下 *expression2* 桁までに丸めた結果を戻します。*expression2* が負の場合、 *expression1* の小数点の左側の「*expression2* の絶対値」桁に丸められます。

最初の引き数は、任意の組み込み数値データ・タイプにすることができます。2 番目の引き数は INTEGER または SMALLINT にすることができます。DECIMAL および REAL は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は次のとおりです。

- 最初の引き数が INTEGER または SMALLINT の場合は INTEGER になります。
- 最初の引き数が BIGINT の場合は BIGINT になります。
- 最初の引き数が DOUBLE、DECIMAL または REAL の場合は DOUBLE になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

例:

- 数値 873.726 を小数点以下 2 桁、1 桁、0 桁、-1 桁および -2 桁にそれぞれ丸めた値を表示します。

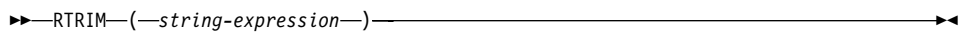
```
VALUES (DECIMAL(ROUND(873.726,2),6,3), DECIMAL(ROUND(873.726,1),6,3),
        DECIMAL(ROUND(873.726,0),6,3), DECIMAL(ROUND(873.726,-1),6,3),
        DECIMAL(ROUND(873.726,-2),6,3))
```

  上記の例では以下が戻されます。

```
1        2        3        4        5
-------- -------- -------- -------- ---------
 873.730  873.700  874.000  870.000   900.000
```

  前述のように、ROUND 関数の出力は DOUBLE になります。上記の例で、関数 DECIMAL は ROUND の出力を限定するのに使用されています。

RTRIM



スキーマは SYSIBM です。[47]

RTRIM 関数は、*string-expression* の末尾からブランクを除去します。

引き数には CHAR、VARCHAR、GRAPHIC、または VARGRAPHIC データ・タイプを使用できます。

- 引き数が DBCS または EUC データベースの漢字ストリングである場合は、後続 2 バイト・ブランクが除去されます。
- 引き数が Unicode データベースの漢字ストリングである場合は、後続 UCS-2 ブランクが除去されます。
- それ以外の場合は、後続 1 バイト・ブランクが除去されます。

この関数の結果のデータ・タイプは次のとおりです。

- *string-expression* のデータ・タイプが VARCHAR または CHAR の場合は VARCHAR になります。
- *string-expression* のデータ・タイプが VARGRAPHIC または GRAPHIC の場合は VARGRAPHIC になります。

戻されるタイプの長さパラメーターは、引き数のデータ・タイプの長さパラメーターと同じになります。

結果が文字ストリングである場合の実際の長さは、除去されるブランク文字のバイト数を *string-expression* から引いた値になります。結果が漢字ストリングである場合の実際の長さは、除去される 2 バイト・ブランク文字の数を *string-expression* から引いた値 (2 バイト文字単位) になります。すべての文字が除去された場合、結果は空になり、可変長ストリング (長さゼロ) が戻されます。

引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

例: ホスト変数 HELLO が CHAR(9) として定義されており、値は 'Hello ' であるものとします。

---

47. この関数の SYSFUN バージョンでは、 LONG VARCHAR 引き数と CLOB 引き数のサポートが引き続き有効です。説明については、376ページの『RTRIM (SYSFUN スキーマ)』を参照してください。

```
VALUES RTRIM(:HELLO)
```

結果は ’Hello’ になります。

RTRIM (SYSFUN スキーマ)

スキーマは SYSFUN です。

引き数の文字から後続ブランクを除去した文字を戻します。

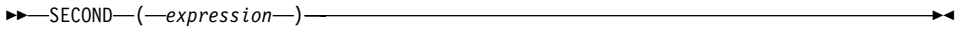
引き数は、任意の組み込み文字ストリング・データ・タイプにすることができます。 VARCHAR の場合、最大長は 4 000 バイトです。 CLOB の場合、最大長は 1 048 576 バイトです。

関数の結果は次のとおりです。

- 引き数が VARCHAR (4 000 バイトを超えない) または CHAR である場合、 VARCHAR(4000) になります。
- 引き数が CLOB または LONG VARCHAR の場合は CLOB(1M) になります。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SECOND



スキーマは SYSIBM です。

SECOND 関数は、値の秒の部分を戻します。

引き数は、時刻、タイム・スタンプ、時刻期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない時刻またはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

- 引き数が、時刻、タイム・スタンプ、または時刻やタイム・スタンプの有効なストリング表記の場合
  - 結果は、値の秒の部分 (0 ～ 59 の整数) になります。
- 引き数が時刻期間またはタイム・スタンプ期間の場合
  - 結果は、値の秒の部分 (-99 ～ 99 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- ホスト変数 TIME_DUR (decimal(6,0)) の値が 153045 と想定します。

  ```
  SECOND(:TIME_DUR)
  ```

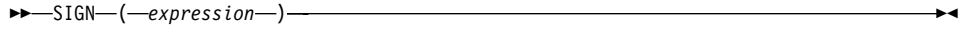
  戻り値は 45 です。

- RECEIVED (timestamp) 列に、 1988-12-25-17.12.30.000000 に相当する内部値が入っていると想定します。

  ```
  SECOND(RECEIVED)
  ```

  この例では 30 の値を戻します。

SIGN



スキーマは SYSFUN です。

引き数の符号の標識を戻します。引き数が負の場合は、-1 が戻されます。引き数がゼロの場合は、0 が戻されます。引き数が正の場合には、1 が戻されます。

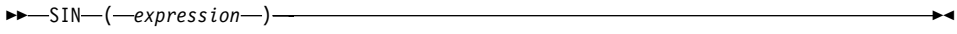
引き数は、任意の組み込み数値データ・タイプにすることができます。DECIMAL および REAL は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は次のとおりです。

- 引き数が SMALLINT の場合は SMALLINT になります。
- 引き数が INTEGER の場合は INTEGER になります。
- 引き数が BIGINT の場合は BIGINT になります。
- それ以外の場合は DOUBLE になります。

結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SIN



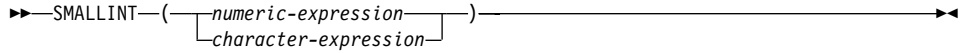
スキーマは SYSFUN です。

引き数のサイン (正弦) の値を戻します。引き数は、ラジアン単位の角度です。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SMALLINT



スキーマは SYSIBM です。

SMALLINT 関数は、短精度整数 (small integer) 定数の形式の数値または文字ストリングの短精度整数 (small integer) 表記を戻します。

numeric-expression
組み込み数値データ・タイプの値を戻す式。

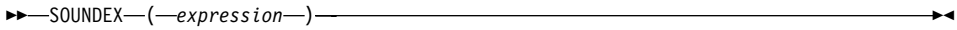
引き数が*数値式* の場合、結果は、引き数を短精度整数 (small integer) の列または変数に割り当てた場合と同じ数値になります。引き数の整数部分が短精度整数 (small integer) の範囲内にない場合、エラーになります。引き数に小数部分がある場合は、切り捨てられます。

character-expression
文字定数の最大長以下の長さの文字ストリング値を戻す式。先行ブランクと後書きブランクは除去されます。その結果のストリングは、 SQL 整数定数を形成するための規則に従うものでなければなりません (SQLSTATE 22018)。ただし、定数の値は短精度整数 (small integer) の範囲内になければなりません (SQLSTATE 22003)。文字ストリングとして、長ストリングを使うことはできません。

引き数が*文字式* の場合、結果は、対応する整数定数を短精度整数 (small integer) の列または変数に割り当てた場合の数値と同じになります。

この関数の結果は短精度整数 (small integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

SOUNDEX



スキーマは SYSFUN です。

引き数内の語の音を示す 4 文字コードを戻します。この結果は、他のストリングの音との比較に使用することができます。

引き数は、 CHAR または VARCHAR (4 000 バイトを超えない) のいずれかの文字ストリングです。

関数の結果は CHAR(4) です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SOUNDEX 関数は、音が分かっているが、正確なつづりが不明なストリングを検出する場合に有用です。文字および文字の組合わせがどのように聞こえるかを前提とするものであり、類似する音の語の探索に役立ちます。比較は、直接、またはストリングを引き数として DIFFERENCE 関数へ渡すことによって行うことができます (308ページの『DIFFERENCE』を参照)。

例:

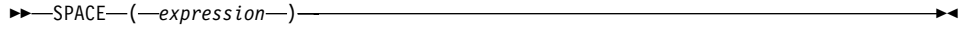
EMPLOYEE 表を使って、 'Loucesy' に類似する発音する姓をもつ従業員の EMPNO および LASTNAME を見つけます。

```
SELECT EMPNO, LASTNAME FROM EMPLOYEE
  WHERE SOUNDEX(LASTNAME) = SOUNDEX('Loucesy')
```

この例では、以下を戻します。

```
EMPNO  LASTNAME
------ ---------------
000110 LUCCHESSI
```

SPACE



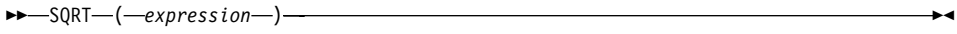
スキーマは SYSFUN です。

2 番目の引き数によって指定された長さのブランクで構成される文字ストリングを戻します。

引き数は SMALLINT または INTEGER にすることができます。

関数の結果は VARCHAR(4000) になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SQRT



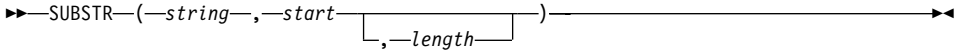
スキーマは SYSFUN です。

引き数の平方根を戻します。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換する必要があります。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

SUBSTR



スキーマは SYSIBM です。

SUBSTR 関数は、ストリングのサブストリングを戻します。

string が文字ストリングの場合、関数の結果は、その最初の引き数のコード・ページで示された文字ストリングになります。 2 進ストリングの場合には、関数の結果も 2 進ストリングになります。漢字ストリングの場合は、関数の結果も最初の引き数のコード・ページで示された漢字ストリングになります。
SUBSTR 関数のいずれかの引き数がヌル値の可能性がある場合、結果もヌル値になる可能性があります。いずれかの引き数がヌル値の場合、結果はヌル値になります。

string
結果を取り出すストリングを指定する式。

string が文字ストリングまたは 2 進ストリングの場合、 *string* のサブストリングは、ゼロ個以上の連続したバイトからなるストリング になります。 *string* が漢字ストリングの場合、*string* のサブストリングは、ゼロ個以上の連続する 2 バイト文字からなるストリング になります。

start
結果の最初のバイト位置 (文字ストリングまたは 2 進ストリングの場合)、あるいは結果の最初の文字の位置 (漢字ストリングの場合) を指定する式。 *start* は、*string* が固定長か可変長かに応じて、 1 から *string* の最大長までの整数でなければなりません (この範囲外の場合は SQLSTATE 22011)。データベース・コード・ページの文脈内にあるバイト数として指定しなければなりません。アプリケーション・コード・ページの文脈で指定してはなりません。

length
結果の長さを指定する式。この式を指定する場合、 *length* は、0 ～ *n* の範囲の 2 進整数でなければなりません。ただし、*n* は、 (*string* の長さ属性) - *start* + 1 です (その範囲外の場合は SQLSTATE 22011)。

length を明示的に指定した場合、 *string* の右側には必要な数のブランク文字 (文字ストリングの場合は 1 バイト、漢字ストリングの場合は 2 バイト) が効率的に付加されて、 *string* のうちの指定したサブストリングが常に存在するようにされます。 *length* のデフォルト値は、文字ストリングまたは 2 進ストリングの場合は、 *start* で指定されたバイト位置から *string*

の最後のバイト位置までのバイト数、漢字ストリングの場合には、 *start* で指定された文字位置から *string* の最後の文字位置までの 2 バイト文字の数です。ただし、*string* が可変長ストリングで、その長さが *start* 未満の場合、デフォルト値はゼロになり、結果は空ストリングになります。データベース・コード・ページの文脈内にあるバイト数として指定しなければなりません。アプリケーション・コード・ページの文脈で指定してはなりません。 (たとえば、データ・タイプ VARCHAR(18)、値 'MCKNIGHT' の列 NAME の場合、 `SUBSTR(NAME,10)` では空ストリングが戻されます) 。

表16 に、入力のタイプと属性ごとに、 SUBSTR 関数の結果タイプと長さがどうなるかを示しています。

表 16. SUBSTR の結果のデータ・タイプと長さ

ストリング引き数のデータ・タイプ	長さ引き数	結果のデータ・タイプ
CHAR(A)	定数 (*l*<255)	CHAR(*l*)
CHAR(A)	指定しない。*start* 引き数は定数	CHAR(A-*start*+1)
CHAR(A)	定数以外	VARCHAR(A)
VARCHAR(A)	定数 (*l*<255)	CHAR(*l*)
VARCHAR(A)	定数 (254<*l*<32673)	VARCHAR(*l*)
VARCHAR(A)	定数以外、または指定しない。	VARCHAR(A)
LONG VARCHAR	定数 (*l*<255)	CHAR(*l*)
LONG VARCHAR	定数 (254<*l*<4001)	VARCHAR(*l*)
LONG VARCHAR	定数 (*l*>4000)	LONG VARCHAR
LONG VARCHAR	定数以外、または指定しない。	LONG VARCHAR
CLOB(A)	定数 (*l*)	CLOB(*l*)
CLOB(A)	定数以外、または指定しない。	CLOB(A)
GRAPHIC(A)	定数 (*l*<128)	GRAPHIC(*l*)
GRAPHIC(A)	指定しない。*start* 引き数は定数	GRAPHIC(A-*start*+1)
GRAPHIC(A)	定数以外	VARGRAPHIC(A)

表 16. SUBSTR の結果のデータ・タイプと長さ (続き)

ストリング引き数のデータ・タイプ	長さ引き数	結果のデータ・タイプ
VARGRAPHIC(A)	定数 (*l*<128)	GRAPHIC(*l*)
VARGRAPHIC(A)	定数 (127<*l*<16337)	VARGRAPHIC(*l*)
VARGRAPHIC(A)	定数以外	VARGRAPHIC(A)
LONG VARGRAPHIC	定数 (*l*<128)	GRAPHIC(*l*)
LONG VARGRAPHIC	定数 (127<*l*<2001)	VARGRAPHIC(*l*)
LONG VARGRAPHIC	定数 (*l*>2000)	LONG VARGRAPHIC
LONG VARGRAPHIC	定数以外、または指定しない。	LONG VARGRAPHIC
DBCLOB(A)	定数 (*l*)	DBCLOB(*l*)
DBCLOB(A)	定数以外、または指定しない。	DBCLOB(A)
BLOB(A)	定数 (*l*)	BLOB(*l*)
BLOB(A)	定数以外、または指定しない。	BLOB(A)

string が固定長ストリングの場合に *length* を省略すると、暗黙に LENGTH(*string*) - *start* + 1 が指定されます。 *string* が可変長ストリングの場合に *length* を省略すると、暗黙にゼロまたは LENGTH(*string*) - *start* + 1 のいずれか大きい方が指定されます。

例:

- ホスト変数 NAME (VARCHAR(50)) の値が 'BLUE JAY' で、ホスト変数 SURNAME_POS (int) の値が 6 と想定します。

```
SUBSTR(:NAME, :SURNAME_POS):ehp2s
```

  値 'JAY' が戻されます。

```
SUBSTR(:NAME, :SURNAME_POS,1)
```

  値 'J' が戻されます。

- PROJECT 表から、プロジェクト名 (PROJNAME) が語 'OPERATION ' で始まるすべての行を選択します。

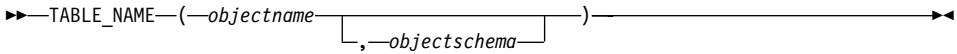
```
SELECT * FROM PROJECT
  WHERE SUBSTR(PROJNAME,1,10) = 'OPERATION '
```

定数の最後にあるスペースは、 'OPERATIONS' などの語で始まるものを除外するために必要です。

注:

1. 動的 SQL では、*string*、*start*、および *length* が、パラメーター・マーカー (?) によって表される場合があります。 *string* にパラメーター・マーカーが使用されると、オペランドのデータ・タイプは VARCHAR になり、オペランドはヌル値可になります。
2. 上記の結果定義には明確には述べられていませんが、 *string* が 1 バイト文字 / 多重バイト文字混合ストリングの場合、 *start* と *length* の値によっては、結果に多重バイト文字の断片が含まれることになる場合があります。つまり、結果が 2 バイト文字の 2 番目のバイトから始まったり、 2 バイト文字の最初のバイトで終わったりする可能性があるということです。
   SUBSTR 関数は、このような断片化の検出を行わず、またこのような断片化があっても特別な処理は何も行われません。

TABLE_NAME



スキーマは SYSIBM です。

TABLE_NAME 関数は、別名連鎖が解決された後で検出されるオブジェクトの非修飾名を戻します。指定された *objectname* (および *objectschema*) が、解決の開始点として使用されます。開始点が別名を参照していない場合は、開始点の非修飾名が戻されます。結果の名前は、表、視点、または未定義オブジェクトのいずれかになります。

objectname
解決しようとする非修飾名 (通常は既存の別名) を表す文字式。 *objectname* は、データ・タイプが CHAR または VARCHAR、長さが 1 文字以上 129 文字未満でなければなりません。

objectschema
指定された *objectname* の解決前の値を修飾するのに使うスキーマを表す文字式。 *objectschema* は、データ・タイプが CHAR または VARCHAR、長さが 1 文字以上 129 文字未満でなければなりません。

objectschema を指定しない場合は、修飾子にデフォルトのスキーマが使用されます。

この関数の結果のデータ・タイプは VARCHAR(128) です。 *objectname* がヌル値の可能性がある場合は、結果もヌル値になる可能性があります。 *objectname* がヌル値であれば、結果もヌル値になります。 *objectschema* がヌル値の場合は、デフォルトのスキーマ名が使用されます。結果は、非修飾名を表す文字ストリングになります。結果の名前は、次のいずれかを表す可能性があります。

表 *objectname* の値が、表名 (入力値が戻される) であったか、あるいは解決結果が表となり、その表名が戻されることになる別名であった。

視点 *objectname* の値が、視点名 (入力値が戻される) であったか、あるいは解決結果が視点となり、その視点名が戻されることになる別名であった。

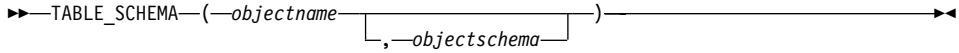
未定義オブジェクト
objectname の値が、未定義オブジェクト (入力値が戻される) であったか、あるいは解決結果が未定義オブジェクトとなり、その名前が戻されることになる別名であった。

したがって、ヌル値以外の値がこの関数に指定された場合、結果名のオブジェクトが存在していなくても、常にその値が戻されます。

例:

390ページの『TABLE_SCHEMA』の例を参照してください。

TABLE_SCHEMA



スキーマは SYSIBM です。

TABLE_SCHEMA 関数は、別名連鎖が解決された後で検出されるオブジェクトのスキーマ名を戻します。指定された *objectname* (および *objectschema*) が、解決の開始点として使用されます。開始点が別名を参照していない場合は、開始点のスキーマ名が戻されます。結果のスキーマ名は、表、視点、または未定義オブジェクトのいずれかになります。

objectname
解決しようとする非修飾名 (通常は既存の別名) を表す文字式。 *objectname* は、データ・タイプが CHAR または VARCHAR、長さが 1 文字以上 129 文字未満でなければなりません。

objectschema
指定された *objectname* の解決前の値を修飾するのに使うスキーマを表す文字式。 *objectschema* は、データ・タイプが CHAR または VARCHAR、長さが 1 文字以上 129 文字未満でなければなりません。

objectschema を指定しない場合は、修飾子にデフォルトのスキーマが使用されます。

この関数の結果のデータ・タイプは VARCHAR(128) です。 *objectname* がヌル値の可能性がある場合は、結果もヌル値になる可能性があります。
objectname がヌル値であれば、結果もヌル値になります。 *objectschema* がヌル値の場合は、デフォルトのスキーマ名が使用されます。結果は、スキーマ名を表す文字ストリングになります。結果のスキーマは、次のいずれかのスキーマ名を表します。

表 *objectname* の値が、表名 (*objectschema* の入力値またはデフォルト値が戻される) であったか、あるいは解決結果が表となり、そのスキーマ名が戻されることになる別名であった。

視点 *objectname* の値が、視点名 (*objectschema* の入力値またはデフォルト値が戻される) であったか、あるいは解決結果が視点となり、そのスキーマ名が戻されることになる別名であった。

未定義オブジェクト

objectname の値が、未定義オブジェクト名 (*objectschema* の入力値またはデフォルト値が戻される) であったか、あるいは解決結果が未定義オブジェクトとなり、そのスキーマ名が戻されることになる別名であった。

したがって、ヌル値以外の *objectname* 値がこの関数に指定された場合、結果名のスキーマ名でのオブジェクトが存在していなくても、常に値が戻されます。たとえば、TABLE_SCHEMA('DEPT', 'PEOPLE') は、カタログ項目が見つからない場合には、'PEOPLE ' を戻します。

例:

- PBIRD は、表 HEDGES.T1 に定義されている別名 PBIRD.A1 を使用して、SYSCAT.TABLES から指定した表の統計値を選択しようとしています。

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('A1')
  AND TABSCHEMA = TABLE_SCHEMA ('A1')
```

  HEDGES.T1 について要求された統計値が、カタログから取り出されます。

- HEDGES.X1 というオブジェクトの統計値を、 HEDGES.X1 を使用して SYSCAT.TABLES から選択します。 HEDGES.X1 が別名か表かが分からないため、 TABLE_NAME と TABLE_SCHEMA を使用します。

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('X1','HEDGES')
  AND TABSCHEMA = TABLE_SCHEMA ('X1','HEDGES')
```

  HEDGES.X1 が表であるとすると、 HEDGES.X1 について要求された統計がカタログから取り出されます。

- HEDGES.T2 に対して定義された別名 PBIRD.A2 を使用して、SYSCAT.TABLES から指定した表の統計を選択しますが、HEDGES.T2 は存在していません。

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('A2','PBIRD')
  AND TABSCHEMA = TABLE_SCHEMA ('A2',PBIRD')
```

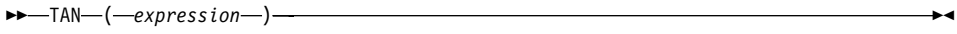
  TABNAME = 'T2' および TABSCHEMA = 'HEDGES' である項目が SYSCAT.TABLES の中に見つからないため、このステートメントからは 0 個のレコードが戻されます。

- SYSCAT.TABLES 内の各項目の修飾名、および別名項目については最終参照名を選択します。

```
SELECT TABSCHEMA AS SCHEMA,  TABNAME AS NAME,
  TABLE_SCHEMA (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_SCHEMA,
  TABLE_NAME (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_NAME
  FROM SYSCAT.TABLES
```

このステートメントは、カタログ内の各オブジェクトの修飾名と、別名項目については最終参照名 (別名が解決された後の名前) を戻します。別名でないすべての項目については、 BASE_TABNAME および BASE_TABSCHEMA がヌル値であるため、 REAL_SCHEMA 列と REAL_NAME 列はヌル値になります。

TAN



スキーマは SYSFUN です。

引き数のタンジェント (正接) の値を戻します。引き数は、ラジアン単位の角度です。

引き数は、任意の組み込み数値データ・タイプにすることができます。引き数は、関数による処理に必要な倍精度浮動小数点数に変換する必要があります。

関数の結果は倍精度浮動小数点数です。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

TIME

スキーマは SYSIBM です。

TIME 関数は、値から時刻を戻します。

引き数は、時刻、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない時刻またはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は時刻です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、引き数のデータ・タイプに応じて以下のように異なります。

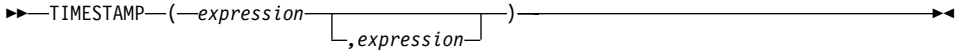
- 引き数が時刻の場合
  - 結果は、指定した時刻になります。
- 引き数がタイム・スタンプの場合
  - 結果はタイム・スタンプの時刻の部分になります。
- 引き数が文字ストリングの場合
  - 結果は、その文字ストリングによって表される時刻になります。

例:

- IN_TRAY サンプル表から、 (任意の日の) 現在の時刻よりも 1 時間後以降に受け取ったすべての注を選択します。

```
SELECT * FROM IN_TRAY
  WHERE TIME(RECEIVED) >= CURRENT TIME + 1 HOUR
```

TIMESTAMP



スキーマは SYSIBM です。

TIMESTAMP 関数は、1 つの値または 2 つの値からタイム・スタンプを戻します。

引き数に関する規則は、2 番目の引き数を指定するか否かによって異なります。

- 引き数を 1 つだけ指定した場合
  - タイム・スタンプ、タイム・スタンプの有効な文字ストリング表記、あるいは CLOB でも LONG VARCHAR でもない長さ 14 の文字ストリングのいずれかでなければなりません。

    長さ 14 の文字ストリングは、有効な日付と時刻を *yyyyxxddhhmmss* という形式で表した数字のストリングであることが必要です (ここで、 *yyyy* は年、 *xx* は月、 *dd* は日、 *hh* は時、 *mm* は分、そして *ss* は秒を表します) 。
- 引き数を 2 つとも指定する場合
  - 最初の引き数は日付または日付の有効な文字ストリング表記でなければならず、 2 番目の引き数は時刻または時刻の有効なストリング表記でなければなりません。

関数の結果はタイム・スタンプです。引き数のいずれかがヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数のいずれかがヌル値の場合、その結果はヌル値です。

その他の規則は、2 番目の引き数を指定するか否かによって以下のように異なります。

- 引き数を 2 つとも指定する場合
  - 結果は、最初の引き数によって日付が指定され、 2 番目の引き数によって時刻が指定されたタイム・スタンプです。タイム・スタンプのマイクロ秒部分はゼロです。
- 引き数が 1 つだけ指定され、それがタイム・スタンプの場合
  - 結果は、指定したタイム・スタンプになります。
- 引き数が 1 つだけ指定され、それが文字ストリングの場合

– 結果は、文字ストリングによって表されるタイム・スタンプになります。引き数が長さ 14 の文字ストリングの場合、タイム・スタンプのマイクロ秒部分はゼロになります。

例:

- START_DATE (日付) 列が 1988-12-25 に等しい値、 START_TIME (時刻) 列が 17.12.30 に等しい値であると想定します。

```
TIMESTAMP(START_DATE, START_TIME)
```

この例は、値 '1988-12-25-17.12.30.000000' を戻します。

TIMESTAMP_ISO

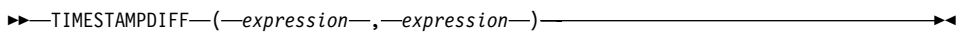
スキーマは SYSFUN です。

日付、時刻、またはタイム・スタンプの引き数に基づくタイム・スタンプ値を戻します。引き数が日付の場合は、時間要素のすべてにゼロが入れられます。引き数が時刻の場合、日付要素には CURRENT DATE の値、時刻の小数要素にはゼロが入れられます。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は TIMESTAMP になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

TIMESTAMPDIFF



スキーマは SYSFUN です。

2 つのタイム・スタンプ間の差に基づいて、最初の引き数によって定義されたタイプの間隔数の見積もりが戻されます。

最初の引き数は INTEGER または SMALLINT のいずれかです。間隔 (最初の引き数) の有効な値は次のとおりです。

1 秒の小数部

2 秒

4 分

8 時間

16 日

32 週

64 月

128 四半期

256 年

2 番目の引き数は、2 つのタイム・スタンプ・タイプの減算を行い、その結果を CHAR(22) に変換した結果です。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

以下の前提事項が、差の見積もりに使用されます。

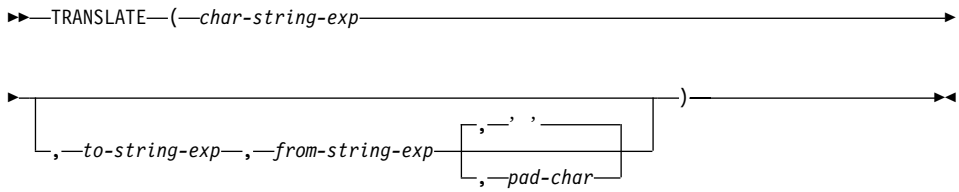
- 1 年は 365 日である
- 1 か月は 30 日である
- 1 日は 24 時間である
- 1 時間は 60 分である
- 1 分は 60 秒である

上記の前提は、タイム・スタンプ期間である 2 番目の引き数の情報を最初の引き数で指定された間隔タイプに変換する際に使用されます。戻される見積もりが、日数によって異なる場合があります。たとえば、 '1997-03-01-00.00.00' と

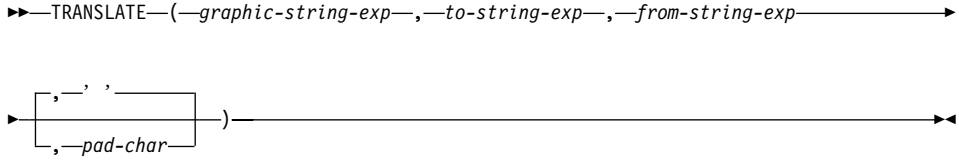
’1997-02-01-00.00.00’ のタイム・スタンプの差の日数 (間隔 16) が要求された場合、結果は 30 になります。これは、タイム・スタンプ相互間の差は 1 か月であり、 1 か月は 30 日であることを前提としているからです。

TRANSLATE

文字ストリング式:



漢字ストリング式:



スキーマは SYSIBM です。

TRANSLATE 関数は、ストリング式内の 1 つまたは複数の文字が他の文字に変換された値を戻します。

関数の結果のデータ・タイプとコード・ページは、最初の引き数と同じです。結果の長さ属性は、最初の引き数と同じになります。指定した式のいずれかがヌル値 (NULL) の可能性がある場合、結果も NULL の可能性があります。指定した式のいずれかがヌル値 (NULL) の場合、結果も NULL になります。

char-string-exp または *graphic-string-exp*
変換される文字または漢字ストリング。

to-string-exp
char-string-exp の特定文字の変換後の文字ストリング。

to-string-exp を指定せず、データ・タイプが漢字でない場合、*char-string-exp* 内の文字すべてが大文字変換されます (つまり、文字 a ～ z は文字 A ～ Z に変換され、発音符付きの文字は大文字に相当するものがあればその文字に変換されます。たとえば、コード・ページ 850 では、é は É に変換されますが、 ÿ は変換されません。これは、コード・ページ 850 に Ÿ が含まれていないためです) 。

from-string-exp
このストリングの文字が *char-string-exp* の中にある場合、それは *to-string-exp* の中の対応する文字に変換されることになります。

from-string-exp に重複する文字が含まれている場合は、最初に見つかった文字が使用され、重複は無視されます。 *to-string-exp* が *from-string-exp* より長い場合、余分な文字は無視されます。 *to-string-exp* を指定する場合は、*from-string-exp* も指定する必要があります。

pad-char-exp
これは、*to-string-exp* が *from-string-exp* より短い場合に、 *to-string-exp* への埋め込みに使用する単一の文字です。 *pad-char-exp* は長さ属性が 1 でなければなりません。そうでない場合には、エラーになります。これを指定しない場合、1 バイトのブランクとみなされます。

引き数には、データ・タイプ CHAR または VARCHAR のいずれかのストリングか、あるいはデータ・タイプ GRAPHIC または VARGRAPHIC の漢字ストリングを指定できます。データ・タイプ LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、または DBCLOB は使用できません。

graphic-string-exp を指定する場合、 *pad-char-exp* だけが任意選択です (これを指定しない場合、 2 バイトのブランクが使用されます)。埋め込み文字を含め、各引き数は漢字データ・タイプでなければなりません。

結果は、*from-string-exp* の中で出現する *char-string-exp* または *graphic-string-exp* 内の文字すべてを、 *to-string-exp* 内の対応する文字に変換した結果のストリング、あるいは対応する文字がない場合は *pad-char-exp* で指定される埋め込み文字に変換した結果のストリングになります。

TRANSLATE の結果のコード・ページは、常に最初のオペランドのコード・ページと同じになります。最初のオペランドのコード・ページは、決して変換されません。それ以外の各オペランドは、最初のオペランドのコード・ページに変換されます。ただし、そのオペランドまたは最初のオペランドが FOR BIT DATA として定義されている場合は、変換されません。

引き数のデータ・タイプが CHAR または VARCHAR の場合、 *to-string-exp* と *from-string-exp* に対応する文字は、同じバイト数でなければなりません。たとえば、1 バイト文字をマルチバイト文字に変換することや、マルチバイト文字を 1 バイト文字に変換することは無効です。そのような変換を行うと、エラーになります。 *pad-char-exp* として有効なマルチバイト文字の第 1 バイトを指定することはできません。そのように指定すると、SQLSTATE 42815 が戻されます。 *pad-char-exp* を指定しない場合は、 1 バイトのブランクが使用されます。

char-string-exp だけを指定した場合、 1 バイト文字は大文字変換され、マルチバイト文字はそのままになります。

例:

- ホスト変数 SITE (VARCHAR(30)) の値が 'Hanauma Bay' であると想定します。

```
TRANSLATE(:SITE)
```

この例では、値 'HANAUMA BAY' を戻します。

```
TRANSLATE(:SITE 'j','B')
```

この例では、'Hanauma jay' が戻されます。

```
TRANSLATE(:SITE,'ei','aa')
```

この例では、値 'Heneume Bey' が戻されます。

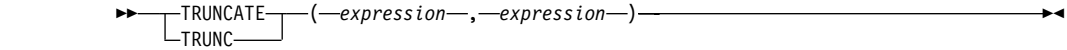
```
TRANSLATE(:SITE,'bA','Bay','%')
```

この例では、値 'HAnAumA bA%' が戻されます。

```
TRANSLATE(:SITE,'r','Bu')
```

この例では、値 'Hana ma ray' が戻されます。

TRUNCATE または TRUNC



スキーマは SYSFUN です。

argument1 を小数点以下 argument2 桁で切り捨てた結果を戻します。argument2 が負の場合、 argument1 の小数点の左側の「argument2 の絶対値」桁が切り捨てられます。

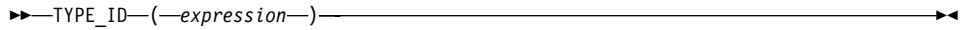
最初の引き数は、任意の組み込み数値データ・タイプにすることができます。2 番目の引き数は、INTEGER か SMALLINT でなければなりません。DECIMAL および REAL は、関数による処理に必要な倍精度浮動小数点数に変換されます。

関数の結果は次のとおりです。

- 最初の引き数が INTEGER または SMALLINT の場合は INTEGER になります。
- 最初の引き数が BIGINT の場合は BIGINT になります。
- 最初の引き数が DOUBLE、DECIMAL または DOUBLE の場合は DOUBLE になります。

結果はヌル値になることがあります。いずれかの引き数がヌル値である場合、結果はヌル値になります。

TYPE_ID



スキーマは SYSIBM です。

TYPE_ID 関数は、*expression* の動的データ・タイプの内部タイプ識別子を戻します。

引き数はユーザー定義の構造タイプでなければなりません。[48]

この関数の結果のデータ・タイプは INTEGER です。 *expression* がヌル値の可能性がある場合は、結果もヌル値になる可能性があります。 *expression* がヌル値であれば、結果もヌル値になります。

TYPE_ID 関数が戻した値は、データベース間で移行することはできません。動的データ・タイプのタイプ・スキーマおよびタイプ名が同じであっても、値が異なる可能性があります。移行性を考慮してコーディングを行う場合、タイプ・スキーマおよびタイプ名の判別には TYPE_SCHEMA および TYPE_NAME 関数を使用してください。

例:

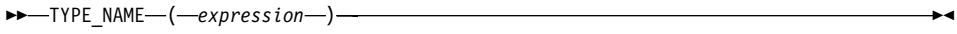
- ある表階層には、タイプ EMP のルート表 EMPLOYEE と、タイプ MGR の副表 MANAGER があります。別の表 ACTIVITIES は、 REF(EMP) SCOPE EMPLOYEE として定義されている WHO_RESPONSIBLE という列を含んでいます。 ACTIVITIES を参照するたびに、参照に対応する行の内部タイプ識別子を表示します。

```
SELECT TASK, WHO_RESPONSIBLE->NAME,
       TYPE_ID(DEREF(WHO_RESPONSIBLE))
  FROM ACTIVITIES
```

  DEREF 関数は、行に対応するオブジェクトを戻すときに使用します。

48. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべての構造化データ・タイプを引き数として受け入れるので、別のユーザー定義タイプをサポートするための追加のシグニチャーを作成する必要はありません。

TYPE_NAME



スキーマは SYSIBM です。

TYPE_ID 関数は、*expression* の動的データ・タイプの非修飾名を戻します。

引き数はユーザー定義の構造タイプでなければなりません。[49]

この関数の結果のデータ・タイプは VARCHAR(18) です。 *expression* がヌル値の可能性がある場合は、結果もヌル値になる可能性があります。 *expression* がヌル値であれば、結果もヌル値になります。 TYPE_SCHEMA 関数を使用して、 TYPE_NAME が戻したタイプ名のスキーマ名を判別してください。

例:

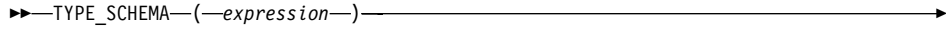
- ある表階層には、タイプ EMP のルート表 EMPLOYEE と、タイプ MGR の副表 MANAGER があります。別の表 ACTIVITIES は、 REF(EMP) SCOPE EMPLOYEE として定義されている WHO_RESPONSIBLE という列を含んでいます。 ACTIVITIES を参照するたびに、参照に対応する行のタイプを表示します。

```
SELECT TASK, WHO_RESPONSIBLE->NAME,
       TYPE_NAME(DEREF(WHO_RESPONSIBLE)),
       TYPE_SCHEMA(DEREF(WHO_RESPONSIBLE))
  FROM ACTIVITIES
```

  DEREF 関数は、行に対応するオブジェクトを戻すときに使用します。

49. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべての構造化データ・タイプを引き数として受け入れるので、別のユーザー定義タイプをサポートするための追加のシグニチャーを作成する必要はありません。

TYPE_SCHEMA



スキーマは SYSIBM です。

TYPE_SCHEMA 関数は、*expression* の動的データ・タイプのスキーマ名を戻します。

引き数はユーザー定義の構造タイプでなければなりません。[50]

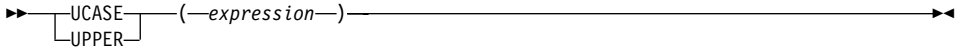
この関数の結果のデータ・タイプは VARCHAR(128) です。 *expression* がヌル値の可能性がある場合は、結果もヌル値になる可能性があります。 *expression* がヌル値であれば、結果もヌル値になります。 TYPE_NAME 関数を使用して、 TYPE_SCHEMA が戻したスキーマ名に関連するタイプ名を判別してください。

例:

405ページの『TYPE_NAME』の例を参照してください。

50. この関数は、ユーザー定義関数を作成するソース関数として使用されません。この関数は、すべての構造化データ・タイプを引き数として受け入れるので、別のユーザー定義タイプをサポートするための追加のシグニチャーを作成する必要はありません。

UCASE または UPPER

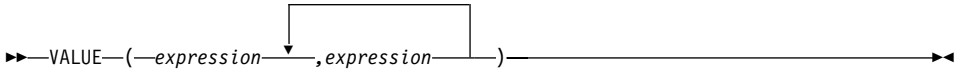


スキーマは SYSIBM です。[51]

UCASE または UPPER 関数は、最初の引き数 (*char-string-exp*) だけが指定されるという点を除けば、 TRANSLATE 関数と同じです。詳細については、400 ページの『TRANSLATE』を参照してください。

51. この関数の SYSFUN バージョンでは、上向き互換性が引き続き有効です。この点の説明については、バージョン 5 の資料を参照してください。

VALUE



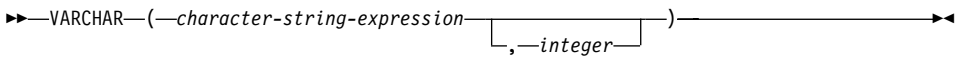
スキーマは SYSIBM です。

VALUE 関数は、ヌル値以外の最初の引き数を戻します。

VALUE は COALESCE の同義語です。詳細については、290ページの『COALESCE』を参照してください。

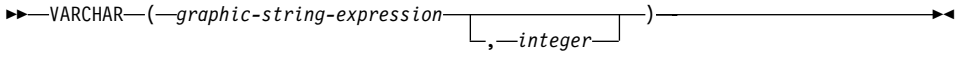
VARCHAR

文字 → VARCHAR:



日時 → VARCHAR:

GRAPHIC → VARCHAR:



スキーマは SYSIBM です。

VARCHAR 関数は、文字ストリング、日時値、または漢字ストリング (UCS-2 の場合のみ) の可変長文字ストリング表記を戻します。

この関数の結果は、可変長ストリング (VARCHAR データ・タイプ) です。最初の引き数がヌル値の可能性がある場合、結果もヌル値の可能性があります。最初の引き数がヌル値である場合、その結果もヌル値になります。

GRAPHIC → VARCHAR は UCS-2 データベースの場合のみ有効です。Unicode 以外のデータベースでは、GRAPHIC → VARCHAR は実行できません。

文字 → VARCHAR

character-string-expression
値が LONG VARGRAPHIC および DBCLOB 以外の文字ストリング・データ・タイプで、最大長が 32 672 バイトを超えない式。

integer
結果の可変長文字ストリングの長さ属性。値は 0 ～ 32 672 の範囲でなければなりません。この引き数を指定しない場合、結果の長さは引き数の長さと同じになります。

日時 → VARCHAR

datetime-expression
値のデータ・タイプが日付、時刻、またはタイム・スタンプのいずれかである式。

GRAPHIC → VARCHAR

graphic-string-expression
値が LONG VARGRAPHIC および DBCLOB 以外の文字ストリング・データ・タイプで、最大長が 16 336 バイトを超えない式。

integer
結果の可変長文字ストリングの長さ属性。値は 0 ～ 32 672 の範囲でなければなりません。この引き数を指定しない場合、結果の長さは引き数の長さと同じになります。

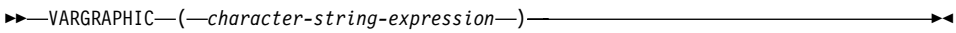
例:

- EMPLOYEE 表を使用して、ホスト変数 JOB_DESC (VARCHAR(8)) に、従業員 Delores Quintana のジョブ記述 (CHAR(8) として定義された JOB) に相当する VARCHAR を設定します。

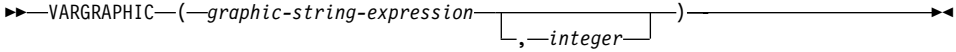
```
SELECT VARCHAR(JOB)
  INTO :JOB_DESC
  FROM EMPLOYEE
  WHERE LASTNAME = 'QUINTANA'
```

VARGRAPHIC

文字→ VARGRAPHIC:



GRAPHIC → VARGRAPHIC:



スキーマは SYSIBM です。

VARGRAPHIC 関数は、以下の漢字ストリング表記を戻します。

- 文字ストリング (1 バイト文字を 2 バイト文字に変換する)
- 漢字ストリング (最初の引き数がいずれかのタイプの漢字ストリングである場合)

この関数の結果は、可変長漢字ストリング (VARGRAPHIC データ・タイプ) です。最初の引き数がヌル値の可能性がある場合、結果もヌル値の可能性があります。最初の引き数がヌル値である場合、その結果もヌル値になります。

文字 → VARGRAPHIC

character-string-expression
値のデータ・タイプが LONG VARCHAR または CLOB 以外である文字ストリングで、その最大長が 16 336 バイトを超えない式。

結果の長さ属性は、引き数の長さ属性と同じになります。

S を *character-string-expression* の値とします。 S 内の各 1 バイト文字は、結果においては、それに相当する 2 バイト表記または 2 バイト置換文字に変換されます。 S 内の各 2 バイト文字はそのままになります。 S の最後のバイトが 2 バイト文字の第 1 バイトである場合、それは 2 バイトの置換文字に変換されます。 S 内の文字の順序はそのまま維持されます。

変換に際しては、さらに次の考慮事項があります。

- Unicode データベースの場合、この関数は文字ストリングをオペランドのコード・ページから UCS-2 へと変換します。 DBCS 文字を含む、オペランドのすべての文字が変換されます。 2 番目の引き数を使用すれば、戻される UCS-2 の長さ (UCS-2 文字の数) を指定できます。

- VARGRAPHIC 関数による 2 バイト・コード・ポイントへの変換は、オペランドのコード・ページに基づいています。
- オペランドの 2 バイト文字は変換されません (例外については、 1453ページの『付録O. 日本語および繁体字中国語 EUC についての考慮事項』を参照)。その他の文字はすべて、それに対応する 2 バイトの表記に変換されます。対応する 2 バイト表記が存在しない場合には、コード・ページ用の 2 バイト置換文字が使用されます。
- 1 つ以上の 2 バイト置換文字が結果内に戻されても警告やエラー・コードは生成されません。

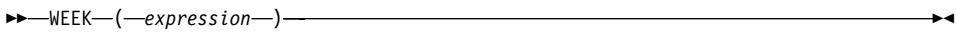
GRAPHIC → VARGRAPHIC

graphic-string-expression
漢字ストリング値を戻す式。

integer
結果の可変長漢字ストリングの長さ属性。値は 0 ～ 16 336 の範囲でなければなりません。この引き数を指定しない場合、結果の長さは引き数の長さと同じになります。

graphic-string-expression の長さが結果の長さ属性より長い場合は、切り捨てが行われて、警告が戻されます (SQLSTATE 01004)。ただし、切り捨てられた文字がすべてブランクであり、 *graphic-string-expression* が長ストリング (LONG VARGRAPHIC または DBCLOB) ではない場合は除きます。

WEEK



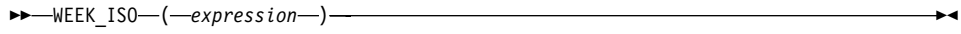
スキーマは SYSFUN です。

引き数の年間通算週番号を 1 ～ 54 の範囲の整数値として戻します。週は日曜日から始まります。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

WEEK_ISO



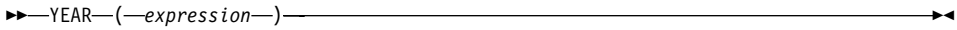
スキーマは SYSFUN です。

引き数の年間通算週番号を 1 ～ 53 の範囲の整数値として戻します。週は月曜日から始まります。第 1 週は、年の第 1 週目で木曜日を含みます。これは、1 月 4 日を含む一週目になります。

引き数は、日付、タイム・スタンプ、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

関数の結果は INTEGER になります。結果はヌル値になることがあります。引き数がヌル値である場合、その結果はヌル値になります。

YEAR



スキーマは SYSIBM です。

YEAR 関数は、指定された値の年の部分を戻します。

引き数は、日付、タイム・スタンプ、日付期間、タイム・スタンプ期間、または CLOB でも LONG VARCHAR でもない日付あるいはタイム・スタンプの有効な文字ストリング表記でなければなりません。

この関数の結果は長精度整数 (large integer) です。引き数がヌル値になる可能性がある場合、結果もヌル値になる可能性があります。引き数がヌル値であれば、結果はヌル値です。

その他の規則は、指定した引き数のデータ・タイプに応じて以下のように異なります。

- 引き数が日付、タイム・スタンプ、または日付やタイム・スタンプの有効なストリング表記の場合
  - 結果は、指定した値の年の部分 (1 ～ 9 999 の整数) になります。
- 引き数が日付期間またはタイム・スタンプ期間の場合
  - 結果は、指定した値の年の部分 (-9 999 ～ 9 999 の整数) になります。ゼロ以外の結果の符号は、引き数と同じになります。

例:

- PROJECT 表から、同一暦年内に開始 (PRSTDATE) および終了 (PRENDATE) が予定されているプロジェクトをすべて選択します。

```
SELECT * FROM PROJECT
  WHERE YEAR(PRSTDATE) = YEAR(PRENDATE)
```

- PROJECT 表から、1 年未満での完了が予定されているプロジェクトをすべて選択します。

```
SELECT * FROM PROJECT
  WHERE YEAR(PRENDATE - PRSTDATE) < 1
```

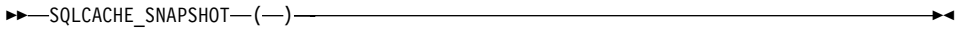
表関数

表関数は、ステートメントの FROM 文節でしか使用できません。さらに、表関数の中では式や列関数を使用できません。

表関数は、表の列を戻します。これは、単純な CREATE TABLE ステートメントが作成する表に似ています。

後続の表関数については、スキーマ名で修飾することができます。

SQLCACHE_SNAPSHOT



スキーマは SYSFUN です。

SQLCACHE_SNAPSHOT は、 DB2 動的 SQL ステートメント・キャッシュのスナップショットの結果を戻します。

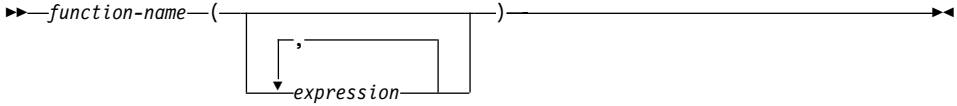
この関数は引き数を取りません。

この関数は、下記に示されている表を戻します。各列の詳細については、システム・モニター *手引きおよび解説書* を参照してください。

表 17. SQLCACHE_SNAPSHOT 表関数が戻す表の列名とデータ・タイプ

列名	データ・タイプ
NUM_EXECUTIONS	INTEGER
NUM_COMPILATIONS	INTEGER
PREP_TIME_WORST	INTEGER
PREP_TIME_BEST	INTEGER
INT_ROWS_DELETED	INTEGER
INT_ROWS_INSERTED	INTEGER
ROWS_READ	INTEGER
INT_ROWS_UPDATED	INTEGER
ROWS_WRITE	INTEGER
STMT_SORTS	INTEGER
TOTAL_EXEC_TIME_S	INTEGER
TOTAL_EXEC_TIME_MS	INTEGER
TOT_U_CPU_TIME_S	INTEGER
TOT_U_CPU_TIME_MS	INTEGER
TOT_S_CPU_TIME_S	INTEGER
TOT_S_CPU_TIME_MS	INTEGER
DB_NAME	VARCHAR(8)
STMT_TEXT	CLOB(64K)

ユーザー定義関数



ユーザー定義関数は、SQL 言語の既存の組み込み関数に対する拡張または追加です。ユーザー定義関数は、呼び出されるたびに単一値を戻すスカラー関数、類似の値の集合を渡され、その集合の単一値を戻す列関数、1 行を戻す行関数、または表を戻す表関数のいずれでかです。 UDF が既存の列関数をソースとして派生される場合にのみ、 UDF は列関数になる点に注意してください。

データベースに登録されたユーザー定義のスカラー関数または列関数は、組み込み関数を使用できるのと同じ文脈で参照することができます。

データベースに登録されたユーザー定義の表関数は、 428ページの『FROM 文節』で説明されているように、 SELECT の FROM 文節でのみ参照することができます。

ユーザー定義の行関数は、ユーザー定義タイプの変形関数として登録しておく場合に限り、暗黙的に参照できます。

ユーザー定義関数は、修飾または無修飾の関数名およびその後に括弧で囲んだその関数の引き数を指定することによって参照します。

関数の引き数の数および位置は、データベースに登録された時点のユーザー定義関数に指定されたパラメーターと対応していなければなりません。さらに、引き数は、対応する定義済みパラメーターのデータ・タイプにプロモート可能なデータ・タイプでなければなりません (634ページの『CREATE FUNCTION』を参照)。

関数の結果は、そのユーザー定義関数が登録された時点で指定された RETURNS 文節に指定されたとおりです。 RETURNS 文節は、関数が表関数であるか否かを決定します。

関数登録時に RETURNS NULL ON NULL INPUT 文節が指定されていると (あるいはデフォルト解釈されると)、引き数のいずれかがヌル値の場合には、結果はヌル値になります。表関数の場合、これは、戻される表が行を含まない (空表) という意味に解釈されます。

SYSFUN スキーマには、一群のユーザー定義関数が用意されています (232ページの表15 を参照)。

例:

- ADDRESS と呼ばれるカラー関数が、スクリプト形式の履歴書から自宅の住所を抽出するように作成されていると想定します。この ADDRESS 関数には、CLOB 引き数を指定し、VARCHAR(4000) が戻されます。次の例は、ADDRESS 関数を呼び出す例です。

```
SELECT EMPNO, ADDRESS(RESUME) FROM EMP_RESUME
  WHERE RESUME_FORMAT = 'SCRIPT'
```

- 数値列 A を含む表 T2 と、前の例で説明した ADDRESS 関数を想定します。次の例は、誤った引き数を使用して ADDRESS 関数を呼び出しています。

```
SELECT ADDRESS(A) FROM T2
```

  名前が一致し、引き数からプロモート可能なパラメーターを持つ関数がないので、エラー (SQLSTATE 42884) が生じます。

- ステートメントの実行時に活動状態であった、サーバー・マシンのセッションについての情報を戻す表関数 WHO が作成されていると想定します。次の例は、FROM 文節における WHO の呼び出しを示しています (必須相関変数のある TABLE キーワード)。

```
SELECT ID, START_DATE, ORIG_MACHINE
FROM TABLE( WHO() ) AS QQ
WHERE START_DATE LIKE 'MAY%'
```

  WHO() 表の列名は、CREATE FUNCTION ステートメントで定義されます。

第5章 照会

照会 は結果表を指定します。

照会は、いくつかの特定の SQL ステートメントの構成要素です。照会には、次の 3 つの形式があります。

- 副選択
- 全選択
- 選択ステートメント

1089ページの『SELECT INTO』 で説明されているように、単一行を検索するのに使用できる SQL ステートメントもあります。

許可

照会で参照される表または視点のそれぞれに対して、ステートメントの許可 ID は少なくとも以下の 1 つを持っている必要があります。

- SYSADM または DBADM 権限
- CONTROL 特権
- SELECT 特権

静的 SQL ステートメントに含まれている照会について、グループ特権の検査は行われません。

照会の中で参照されているニックネームの場合、連合データベースでは、考慮すべき特権はありません。このニックネームで示されている表または視点のデータ・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。

副選択

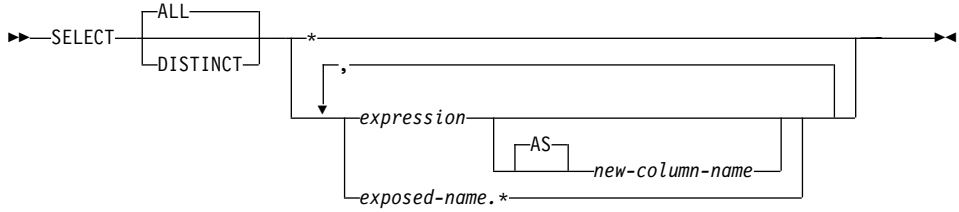
副選択 は、全選択の構成要素の 1 つです。

副選択は、FROM 文節で指定される表、視点、またはニックネームから派生する結果表を指定します。この派生の方法は、各操作の結果が次の演算の入力になるような、一連の操作として記述することができます。 (これは、副選択を記述する 1 つの方法にすぎません。派生操作を実行するために使用される方式は、この記述とはまったく異なる場合があります。)

副選択の文節は以下の順序で処理されます。

1. FROM 文節
2. WHERE 文節
3. GROUP BY 文節
4. HAVING 文節
5. SELECT 文節

SELECT 文節



SELECT 文節は、最終結果表の列を指定します。列値は、*選択リスト* を R に適用することによって作成されます。選択リストとは、SELECT 文節に指定された名前または式であり、 R は副選択のうち直前の操作の結果です。たとえば、指定されている文節が SELECT、FROM、および WHERE だけの場合、R は WHERE 文節の結果になります。

ALL

最終結果表の行すべてをそのまま保持し、冗長な重複を削除しません。これはデフォルト値です。

DISTINCT

最終結果表の中に重複行があれば、その中の 1 つを除き、それ以外のすべてを削除します。 DISTINCT を使用した場合、結果表のストリング列の最大長を 255 バイトより大きくすることはできません。さらに、列を LONG VARCHAR、LONG VARGRAPHIC、DATALINK、LOB といったタイプにすることはできず、それらのタイプのいずれかを基にした特殊タイプ、または構造タイプにすることもできません。 DISTINCT は、1 つの副選択で複数回使用することができます。これには、SELECT DISTINCT、選択リストまたは HAVING 文節の列関数での DISTINCT の使用、および副選択の副照会が含まれます。

2 つの行が互いに重複していると言えるのは、最初の行の各値が 2 番目の行の対応する値に等しい場合だけです。重複を判別する場合、2 つのヌル値は等しいものとみなされます。

選択リストの表記法

* 表 R の列を識別する名前のリストを表します。リスト内の最初の名前は R の最初の列、2 番目の名前は R の 2 番目の列、というようになります。

名前のリストは、その SELECT 文節を含むプログラムのバインド時に確立されます。したがって、* (アスタリスク) では、表参照を含むステートメントのバインド後に表に追加された列は識別されません。

expression

結果列の値を指定します。 『第3章 言語要素』 に説明されているタイプの式でもかまいませんが、通常、使用される式には列名が入っています。選択リストで使用される各列名は、 R の列をあいまいなところなく識別するものでなければなりません。

new-column-name または **AS** *new-column-name*

列名の名前を指定または変更します。この名前は修飾してはならず、固有である必要もありません。列名の後の使用方法は、次のように限定されています。

- AS 文節に指定された新しい列名 (new-column-name) は、その名前が固有であれば、ORDER BY 文節で使用することができます。
- 選択リストの AS 文節に指定された新しい列名を、副選択の他の文節 (WHERE 文節、GROUP BY 文節、または HAVING 文節) で使用することはできません。
- AS 文節に指定された新しい列名を、UPDATE 文節で使用することはできません。
- AS 文節に指定された新しい列名は、ネストした表式、共通表式、および CREATE VIEW の全選択の外部で認識されます。

*name.**

結果表の列を指定する名前のリストを表します。この名前は *exposed-name* によって示されます。 *exposed-name* は、表名、視点名、ニックネーム、または相関名のいずれかにすることができ、FROM 文節で指定された表、視点、またはニックネームを指定するものでなければなりません。リスト内の最初の名前は表、視点、あるいはニックネームの最初の列、 2 番目の名前は表、視点、またはニックネームの 2 番目の列を識別する、というようになります。

名前のリストは、その SELECT 文節を含むステートメントのバインド時に確立されます。したがって、ステートメントのバインド後に表に追加された列は、 * によって識別されません。

SELECT の結果の列の数は、演算形式の選択リスト (つまり、ステートメントの準備時に設定されたリスト) の式の数と同じであり、 500 を超えることはできません。

ストリング列に関する制限

選択リストの制限については、89ページの『可変長文字ストリングの使用制限』 を参照してください。

選択リストの適用
選択リストを R に適用した結果は、 GROUP BY または HAVING が使用されているかどうかによって異なる場合があります。その結果について、次の 2 つのリストで説明します。

GROUP BY または HAVING が使用されている場合:

- 選択リストで使用される式 *X* (列関数ではない) には、以下を指定した GROUP BY 文節が必要です。
  - 各列名が列 R を明確に識別するグループ化式 (437ページの『GROUP BY 文節』を参照) または
  - 個別のグループ化式 として *X* で参照される R の各列
- 選択リストは R のそれぞれのグループに対して適用され、その結果には、R にあるグループと同数の行が含まれます。選択リストが R の 1 つのグループに適用されるとき、そのグループは、選択リストの列関数の引き数のソースになります。

GROUP BY または HAVING のどちらも使用されていない場合:

- 選択リストに列関数が含まれていないか、または選択リスト内の各 *column-name* が列関数の中に指定されているか、あるいは相関列参照であるかのいずれかでなければなりません。
- 選択リストが列関数を含まない場合、選択リストは R のそれぞれの行に対して適用され、その結果には R にある行と同数の行が含まれます。
- 選択リストが列関数のリストである場合、関数の引き数は R から与えられ、選択リストを適用した結果は 1 行となります。

どちらの場合も、結果の *n* 番目の列には、命令形式の選択リストにある *n* 番目の式を適用することによって指定された値が入ります。

結果列の NULL 属性: 結果列は、以下から取り出された場合には、ヌル値を使用できません。

- ヌル値をもたない列
- 定数値
- COUNT または COUNT_BIG 関数
- 標識変数を持たないホスト変数
- ヌルを使用できるオペランドを含まないスカラー関数または式

結果列が以下から得られた場合は、ヌル値を使用できます。

- COUNT または COUNT_BIG 以外の列関数

- ヌル値が可能な列
- ヌル値が可能なオペランドの含まれるスカラー関数または式
- 等しい値を引き数とする NULLIF 関数
- 標識変数を持つホスト変数
- 選択リスト内の対応項目の少なくとも 1 つがヌル値をとれる場合のセット演算の結果
- 演算式から得られた演算式または視点の列で、そのデータベースが DFT_SQLMATHWARN を yes に 設定して構成されている。
- 参照解除操作

結果列の名前:

- AS 文節が指定されている場合、結果列の名前は、AS 文節に指定された名前になります。
- AS 文節が指定されておらず、結果列が列から派生している場合、結果列の列名は、その列の非修飾名になります。
- AS 文節が指定されておらず、結果列が参照解除操作を使用して派生している場合、結果列の列名がその参照解除操作のターゲット列の非修飾名になります。
- それ以外の結果列には、名前が付けられません。[52]

結果列のデータ・タイプ: SELECT の結果の各列は、その列の派生元の式のデータ・タイプとなります。

式	結果列のデータ・タイプ
数値列の名前	列のデータ・タイプと同じ。DECIMAL 列の場合は精度と位取りも同じ。
整数定数	INTEGER
10 進定数	定数の精度および位取りを持つ DECIMAL
浮動小数点定数	DOUBLE
任意の数値変数の名前	変数のデータ・タイプと同じで、10 進変数については精度と位取りも同じ。
式	データ・タイプ属性の説明は、176ページの『式』の項を参照。
関数	(結果のデータ・タイプを判別するには、『第4章 関数』 を参照。)

52. システムは、これらの列に対して一時的な数字を (文字ストリングとして) 割り当てます。

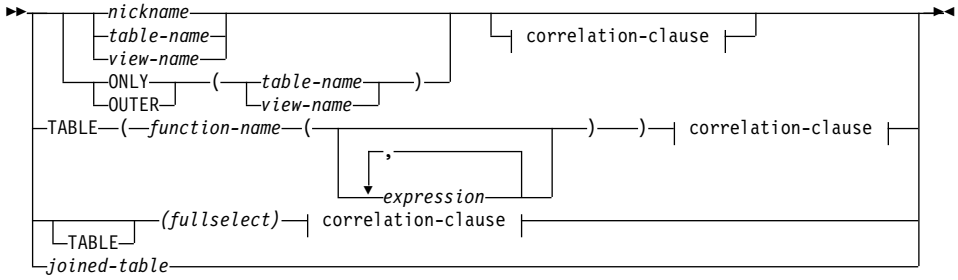
式	結果列のデータ・タイプ
n バイトを表す 16 進定数	VARCHAR(n)。コード・ページはデータベース・コード・ページです。
ストリング列の名前	列のデータ・タイプと同じで、長さ属性も同じ
ストリング変数の名前	変数のデータ・タイプと同じで、長さ属性も同じ。変数のデータ・タイプが SQL データ・タイプと同じではない場合 (たとえば、C において NUL 文字で終了するストリング)、結果列は可変長ストリングになります。
長さ *n* の文字ストリング	VARCHAR(*n*)
長さ *n* の漢字ストリング	VARGRAPHIC(*n*)
日付 / 時刻の列の名前	列のデータ・タイプと同じ。
ユーザー定義タイプ列の名前	列のデータ・タイプと同じ
参照タイプ列の名前	列のデータ・タイプと同じ

FROM 文節

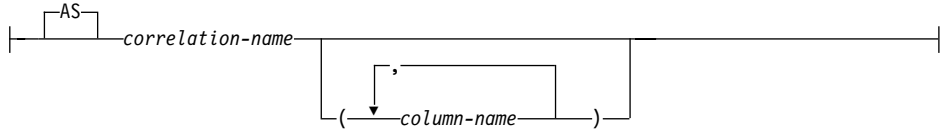
FROM 文節は、中間結果表を指定します。

表参照 (table-reference) が 1 つ指定されている場合、中間結果表は単に、その表参照の結果です。複数の表参照が指定されている場合、中間結果表は、指定された表参照の行の可能なすべての組み合わせ (カルテシアン積) からなります。結果の各行は、最初の表参照の行を 2 番目の表参照の行に連結し、それを 3 番目の表参照の行に連結し、以下同様の手順で連結した行です。結果の行数は、すべての表参照の行数の積です。*表参照* については、429ページの『表参照』 を参照してください。

表参照



correlation-clause:



表参照として指定する各 *table-name* (表名)、*view-name* (視点名)、または *nickname* (ニックネーム) は、アプリケーション・サーバーの既存の表、視点、またはニックネーム、あるいは表参照を含む全選択の前に定義された共通表式 (468ページの『共通表式』 を参照) の *table-name* を指定するものでなければなりません。 *table-name* (表名) がタイプ付き表を参照する場合、その名前はその表およびその表の副表 (*table-name* の列のみ) の UNION ALL (全合併) を表します。同様に、*view-name* (視点名) がタイプ付き視点を参照する場合、その名前はその視点およびその視点の副視点 (*view-name* の列のみ) の UNION ALL を表します。

ONLY(*table-name*) または ONLY(*view-name*) を使用した場合は、適正な副表または副視点の行は含まれません。 ONLY に指定した *table-name* に副表がない場合、 ONLY(*table-name*) は *table-name* を指定することと同じになります。 ONLY に指定した *view-name* に副視点がない場合、 ONLY(*view-name*) は *view-name* を指定することと同じになります。

OUTER(*table-name*) または OUTER(*view-name*)を指定した場合、それは仮想表を表します。 OUTER に指定した *table-name* または *view-name* に副表または副視点がない場合は、 OUTER を指定してもしなくても同じです。 OUTER(*table-name*) は、次のように *table-name* から派生します。

- 列には、*table-name* の列に続いて、副表によって導入された追加列 (副表がある場合) が組み込まれます。副表階層を階層深度の大きい順に走査し、追加列は右側に追加されます。共通の親を持つ副表の場合は、タイプの作成順に走査します。
- 行には、*table-name* のすべての行、およびその表の副表のすべての行が組み込まれます。その行の副表にない列には、ヌル値が戻されます。

上記の点は OUTER(*view-name*) にも当てはまります。その場合、*table-name* を *view-name* に、副表を副視点に読み替えてください。

ONLY または OUTER を使用するときには、 *table-name* の副表または *view-name* の副視点ごとに、 SELECT 権限が必要です。

表参照として指定された各 *function-name* (関数名) およびその引き数のタイプは、アプリケーション・サーバーの既存の表関数に解決されなければなりません。

括弧内の全選択 (fullselect) とその後に続く相関名 (correlation name) は、*ネストされた表式* と呼ばれます。

joined-table (結合表) は、 1 つまたは複数の結合演算の結果である中間結果セットを指定します。詳細については、433ページの『結合表』を参照してください。

すべての表参照の直接的な名前は固有でなければなりません。直接的な名前とは、以下の名前です。

- *相関名*
- *表名* で、その後に *相関名* のないもの
- *視点名* で、その後に *相関名* のないもの
- ニックネーム で、その後に *相関名* のないもの
- *別名* で、その後に *相関名* のないもの

各 *correlation-name* (相関名) は、直前の *表名*、*視点名*、ニックネーム、*関数名* の参照またはネストした表式の指定子として定義されます。表、視点、表関数、またはネストした表式の列に対する修飾参照では、必ず直接的な名前を使用しなければなりません。同じ表名、視点名、またはニックネームを 2 回指定する場合は、その少なくとも 1 回の指定の後には *相関名* を付ける必要があります。*相関名* は、表、視点、またはニックネームの列に対する参照を修飾するのに使用されます。*相関名* が指定されている場合、*表名*、*視点名*、ニックネー

ム、*関数名* の参照、あるいはネストした表式の列に名前を指定するために、*列名* を指定することもできます。詳しくは、144ページの『相関名』を参照してください。

通常、表関数、およびネストされた表の式は、FROM 文節にのみ指定することができます。表関数およびネストされた表の式からの列は、選択リストの中および残りの副選択で、指定する必要のある相関名を使用して参照することができます。この相関名の効力範囲は、FROM 文節の他の表名、視点名、またはニックネームの相関名と同じです。ネストされた表式は、次の場合に使用することができます。

- 視点の代わりに使用して、視点が作成されないようにする場合 (視点を一般的に使用する必要がない場合)
- 必要な結果表がホスト変数に基づいている場合

表関数参照

一般的に、表関数とその引き数値は、表や視点とまったく同じ方法で SELECT の FROM 文節で参照することができます。ただし、特殊な考慮事項が適用されます。

- 表関数の列名

  相関名 の後に代替列名を指定する場合を除いて、表関数の列名は、CREATE FUNCTION ステートメントの RETURNS 文節に指定された列名になります。これは、CREATE TABLE ステートメントに定義されている表の列名に類似したものです。表関数作成の詳細については、 663ページの『CREATE FUNCTION (外部表)』または 701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』を参照してください。

- 表関数解決

  表関数参照に指定された引き数は関数名と共に、*関数解決* と呼ばれるアルゴリズムによって、使用する正確な関数を判別するのに用いられます。 これは、ステートメントで使用される他の関数 (たとえば、スカラー関数) の場合に行われるのと同じです。関数解決は、162ページの『関数解決』 で説明されています。

- 表関数の引き数

  スカラー関数の引き数の場合と同じように、通常、表関数の引き数は有効な SQL 式です。したがって、次の例は正しい構文です。

```
例 1:       SELECT c1
            FROM TABLE( tf1('Zachary') ) AS z
            WHERE c2 = 'FLORIDA';
例 2:       SELECT c1
            FROM TABLE( tf2 (:hostvar1, CURRENT DATE) ) AS z;
例 3:       SELECT c1
```

```
FROM t
WHERE c2 IN
        (SELECT c3 FROM
         TABLE( tf5(t.c4) ) AS z  -- correlated reference
        )                         -- to previous FROM clause
```

表参照における相関参照

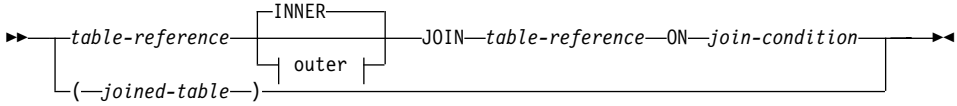
相関参照は、ネストされた表の式や、表関数の引き数として使用することができます。両方の場合に適用される基本的な規則は、相関参照は、副照会の階層のより高いレベルにある*表参照* から行う必要があるということです。この階層には、FROM 文節の左から右への処理により、すでに解決されている表参照が含まれています。ネストされた表の式の場合、TABLE キーワードが全選択の前に指定されなければなりません。したがって、次の例は正しい構文です。

```
例 1:   SELECT t.c1, z.c5
        FROM t, TABLE( tf3(t.c2) ) AS z      -- t precedes tf3 in FROM
        WHERE t.c3 = z.c4;                   -- so t.c2 is known
例 2:   SELECT t.c1, z.c5
        FROM t, TABLE( tf4(2 * t.c2) ) AS z  -- t precedes tf3 in FROM
        WHERE t.c3 = z.c4;                   -- so t.c2 is known
例 3:   SELECT d.deptno, d.deptname,
               empinfo.avgsal, empinfo.empcount
        FROM department d,
             TABLE (SELECT AVG(e.salary) AS avgsal,
                           COUNT(*) AS empcount
                    FROM employee e           -- department precedes and
                    WHERE e.workdept=d.deptno -- TABLE is specified
                   ) AS empinfo;              -- so d.deptno is known
```

しかし、以下は正しくない例です。

```
例 4:   SELECT t.c1, z.c5
        FROM TABLE( tf6(t.c2) ) AS z, t     -- cannot resolve t in t.c2!
        WHERE t.c3 = z.c4;                  -- compare to Example 1 above.

例 5:   SELECT a.c1, b.c5
        FROM TABLE( tf7a(b.c2) ) AS a, TABLE( tf7b(a.c6) ) AS b
        WHERE a.c3 = b.c4;                   -- cannot resolve b in b.c2!
例 6:   SELECT d.deptno, d.deptname,
               empinfo.avgsal, empinfo.empcount
        FROM department d,
             (SELECT AVG(e.salary) AS avgsal,
                           COUNT(*) AS empcount
                    FROM employee e           -- department precedes but
                    WHERE e.workdept=d.deptno -- TABLE is not specified
                   ) AS empinfo;              -- so d.deptno is unknown
```

結合表



outer:

結合表 (joined table) は、内部結合または外部結合のいずれかの結果である中間結果の表を指定します。この表は、結合演算子 INNER、LEFT OUTER、RIGHT OUTER、または FULL OUTER のいずれかをそのオペランドに適用することによって求められます。

内部結合は、表のクロス・プロダクト (左側の表の各行を右側の表の各行に結合する)と見なすことができ、結合条件が真である行のみを保持します。結果表では、結合された表の一方または両方からの行が欠落している場合があります。外部結合には、内部結合が含まれ、このような欠落行を保持します。外部結合には、次の 3 種類のものがあります。

1. ***左外部結合***。内部結合から欠落している左側の表の行を含みます。
2. ***右外部結合***。内部結合から欠落している右側の表の行を含みます。
3. ***全外部結合***。内部結合から欠落している左側および右側の表の行を含みます。

結合演算子の指定がない場合、INNER が暗黙の指定になります。複数の結合が行われる順序は、結果に影響を与えます。結合は、他の結合内にネストすることができます。結合の処理順序は、通常、左から右方向ですが、必要な結合条件の位置に基づきます。ネストされた結合の順序を読みやすくするために、括弧の使用をお勧めします。たとえば、

```
tb1 left join  tb2 on tb1.c1=tb2.c1
    right join tb3 left join tb4 on tb3.c1=tb4.c1
                                 on tb1.c1=tb3.c1
```

これは、以下と同等です。

```
(tb1 left join tb2 on tb1.c1=tb2.c1)
    right join (tb3 left join tb4 on tb3.c1=tb4.c1)
                                 on tb1.c1=tb3.c1
```

結合表は、SELECT ステートメントの形式のいずれかが使用される文脈であれば、どのような文脈でも使用することができます。その SELECT ステートメントに結合表が含まれている場合には、視点またはカーソルは読み取り専用です。

join-condition (結合条件) は *search-condition* (検索条件) です。ただし、以下の点で異なっています。

- 結合条件には、副照会 (スカラーなど) を含むことができません。
- 参照値がオブジェクト識別子列以外の場合、参照解除操作または DEREF 関数を含めることができません。
- SQL 関数を含めることができません。
- *結合条件* の式で参照される列は、関連する結合のオペランド表のいずれかの列でなければなりません (同じ結合表の文節の範囲内)。
- 全外部結合の *結合条件* の式で参照される関数は、決定的なものでなければならず、外部アクションは持ちません。

結合条件が上記の規則にしたがっていない場合、エラーが生じます (SQLSTATE 42972)。

列参照は、列名の修飾子を解決するための規則を使用して解決されます。述部に適用されるのと同じ規則が、*結合条件* にも適用されます (208ページの『述部』 を参照)。

結合演算

結合条件 は、T1 と T2 のペアを指定します。ここで、T1 および T2 は、*結合条件* の JOIN 演算子の左と右のオペランド表です。 T1 および T2 の行のすべての組合わせについて、*結合条件* が真であれば、T1 の行は T2 の行とペアになります。 T1 の行が T2 の行に結合する場合、結果の行は、T1 の行の値が T2 の行の値と連結された値で構成されます。この実行には、NULL 行の生成が含まれる場合があります。表の NULL 行は、列がヌル値を許すか否かに関係なく、表の各列のヌル値で構成されます。

以下に、結合演算の結果を要約します。

- T1 INNER JOIN T2 の結果は、結合条件が真であるペアの行で構成されます。
- T1 LEFT OUTER JOIN T2 の結果は、結合条件が真であるペアの行、およびペアになっていない T1 の行ごとに、その行を T2 のヌル行に連結したもので構成されます。 T2 から得られるすべての列にはヌル値を使用することができます。

- T1 RIGHT OUTER JOIN T2 の結果は、結合条件が真であるペアの行、およびペアになっていない T2 の行ごとに、その行を T1 のヌル行に連結したもので構成されます。 T1 から得られるすべての列にはヌル値を使用することができます。
- T1 FULL OUTER JOIN T2 の結果は、ペアの行、およびペアになっていない T2 の行ごとにその行を T1 のヌル行に連結したもの、およびペアになっていない T1 の行ごとにその行を T2 のヌル行に連結したもので構成されます。 T1 および T2 から得られるすべての列にはヌル値を使用することができます。

WHERE 文節

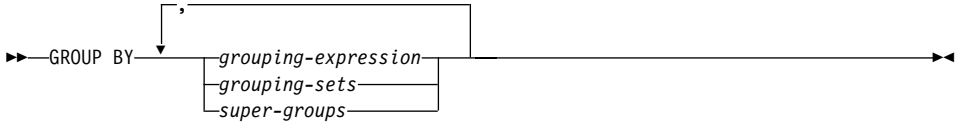
WHERE 文節は、 *search-condition (検索条件)* が真である R の行で構成される中間結果表を指定します。 R は、その副選択の FROM 文節の結果です。

search-condition は、以下の規則に適合していなければなりません。

- 各 *列名* は、R の列をあいまいなところなしに指定するか、あるいは相関参照でなければなりません。副選択外の*表参照*の列を識別している*列名* は、相関参照となります。
- WHERE 文節が HAVING 文節の副照会に指定されていて、関数の引き数がグループに対する相関参照であるのでない限り、列関数を指定することはできません。

search-condition 内の副照会は、R の各行に対して実際に実行され、*search-condition* を R のその行に適用するときにその結果が使用されます。副照会が実際に R の各行に対して実行されるのは、その中に相関参照が含まれる場合だけです。実際、相関参照のない副照会は 1 回しか実行されませんが、相関参照のある副照会は、各行ごとに 1 回ずつ実行されます。

GROUP BY 文節



GROUP BY 文節は、R の行のグループ化で構成される中間結果表を指定します。 R は、副選択のそれ以前の文節の結果です。

もっとも単純な形式では、 GROUP BY 文節に *grouping expression* (グループ化式) が含まれます。グループ化式とは、R のグループ化の定義に使用される式 です。グループ化式に含まれている各*列名* は、 R の列を明確に識別しなければなりません (SQLSTATE 42702 または 42703)。各グループ化式の長さ属性は、 255 バイトを超えてはなりません (SQLSTATE 42907)。グループ化式には、スカラー全選択 (SQLSTATE 42822)、または可変の関数または外部処理を伴う関数 (SQLSTATE 42845) を含めることはできません。

複雑な形式の GROUP BY 文節には、 *grouping-sets* (グループ化集合) および *super-groups* (スーパー・グループ) が含まれます。これらの形式の説明については、それぞれ 438ページの『グループ化集合』 と 439ページの『スーパー・グループ』 を参照してください。

GROUP BY の結果は、いくつかの行グループの集まりです。この結果の各行は、*グループ化式* が等しい行の集合を表します。グループ化では、*グループ化式* のヌル値はすべて等しいものとみなされます。

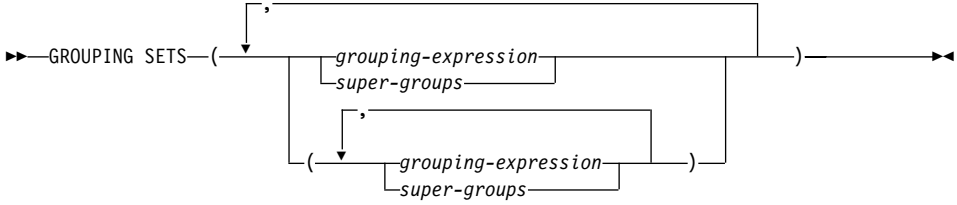
グループ化式 は、HAVING 文節の検索条件、SELECT 文節の式、または ORDER BY 文節の *sort-key-expression* (分類キー式) (詳細については、 471ページの『ORDER BY 文節』 を参照) で使用することができます。いずれの場合も、その参照は各グループの 1 つの値だけを指定します。たとえば、*グループ化式* が *col1+col2* である場合、選択リストで使用できる式は、*col1+col2+3* になります。式の関連性規則では、類似した式 *3+col1+col2* を使用することを許しません。ただし、対応する式を同じ順序で評価するために括弧を使用する場合を除きます。したがって、選択リストでは *3+(col1+col2)* も使用することができます。連結演算子を使用する場合、*グループ化式* は、選択リストで指定された式とまったく同じように使用しなければなりません。

グループ化式 に後書きブランクを含む可変長ストリングが含まれている場合、そのグループの値は後書きブランクの数が異なる場合があり、必ずしも同じ長さになるとはかぎりません。そのような場合でも、*グループ化式* への参照は各

グループに 1 つの値だけを指定しますが、グループの値は使用可能な値の集合の中から任意に選択されます。したがって、結果値の実際の長さは予測できません。

前述のように、GROUP BY 文節が、 SELECT 文節で指定された列を式 (スカラー全選択、可変または外部処理関数) として直接参照することができない場合があります。そのような式を使用してグループ化を行うには、ネストした表式または共通表式を使用して、まず結果の列が式となる結果表を指定します。ネストした表式の使用例については、447ページの例 A9 を参照してください。

グループ化集合



grouping-sets (グループ化集合) の指定により、複数のグループ化文節を単一ステートメントで指定することができます。これは、行の複数のグループを単一の結果セットにまとめたものとみなすことができます。これは、1 つのグループ化集合に対応する各副選択ごとに GROUP BY 文節を持つ複数の副選択を合併したものと論理的に等しくなります。グループ化集合は、単一のエレメント、もしくは括弧によって区切られる複数のエレメントのリストになります。この場合、エレメントはグループ化式、またはスーパー・グループのいずれかです。*グループ化集合* を使用して、基礎表の単一パスを使用してグループを計算することができます。

グループ化集合 の指定により、単純な*グループ化式* を使用するか、またはより複雑な形式の*スーパー・グループ* を使用することができます。*スーパー・グループ* については、 439ページの『スーパー・グループ』 を参照してください。

グループ化集合は、GROUP BY 演算の基礎的な構築ブロックであることに注意してください。単純な列を使用する単純な GROUP BY は、 1 つのエレメントを持つグループ化集合とみなすことができます。たとえば、

```
GROUP BY a
```

これは、以下と同じです。

```
GROUP BY GROUPING SET((a))
```

および

```
GROUP BY a,b,c
```

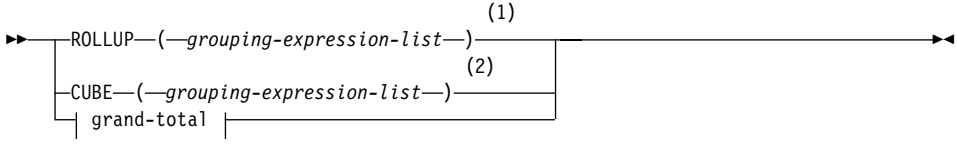
これは、以下と同じです。

```
GROUP BY GROUPING SET((a,b,c))
```

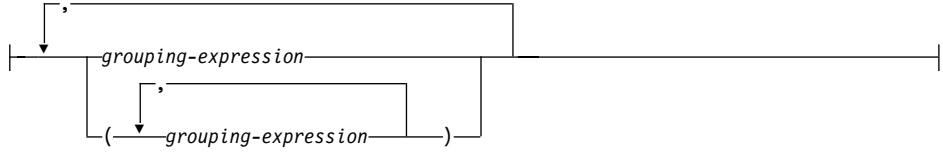
グループ化集合から除外される副選択の選択リストの非集約の列は、そのグループ化集合用に生成される各行の列にヌル値を戻します。これは、集約が列の値を考慮せずに行われたことを表しています。実際のデータのヌル値の行と、グループ化集合から生成されたヌル値の行とを区別する方法については、 260ページの『GROUPING』 を参照してください。

453ページの例 C2 ～ 457ページの例 C7 は、グループ化集合の使用法を示しています。

スーパー・グループ



grouping-expression-list:



grand-total:

注:

1 GROUP BY 文節で単独で使用される代替仕様は、 grouping-expression-list WITH ROLLUP です。

2 GROUP BY 文節で単独で使用される代替仕様は、 grouping-expression-list WITH CUBE です。

ROLLUP (*grouping-expression-list*)

ROLLUP grouping (ROLLUP グループ化) は GROUP BY 文節の拡張であり、「通常の」グループ化された行に加えて*小計* 行を含む結果セットを生成します。*小計* 行[53] は、グループ化行を入手するのに使用されたのと同じ列関数を適用して値が得られる集合体が入っている「スーパー集約」行です。

ROLLUP グループ化は、一連の*グループ化集合* です。 *n* 個のエレメントを持つ ROLLUP の一般的な仕様は、次のとおりです。

```
GROUP BY ROLLUP(C1,C2,...,Cn-1,Cn)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS(
(C1,C2,...,Cn-1,Cn)
                       (C1,C2,...,Cn-1)
                       ...
                       (C1,C2)
                       (C1)
                       () )
```

ROLLUP の *n* のエレメントは、 *n* +1 のグループ化集合に変換される点に注意してください。

グループ化式 が指定されている順序が、 ROLLUP にとって重要である点に注意してください。たとえば、

```
GROUP BY ROLLUP(a,b)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b)
                       (a)
                       () )
```

一方、

```
GROUP BY ROLLUP(b,a)
```

これは、以下と同じです。

```
GROUP BY GROUPING SETS((b,a)
                       (b)
                       () )
```

53. 小計がもっとも一般的な用途なので小計行と呼ばれますが、集約には任意の列関数を使用することができます。たとえば、MAX および AVG は 459ページの例 C8 で使用されます。

ORDER BY 文節は、結果セットの行の順序を保証する唯一の方法です。453ページの例 C3 は ROLLUP の使用法を示しています。

CUBE (*grouping-expression-list*)

CUBE grouping (CUBE グループ化)は GROUP BY 文節の拡張であり、すべての ROLLUP 集約行に加えて、「クロス集計」行を含む結果セットを生成します。*クロス集計* 行は、小計による集約の一部ではない「スーパー集約行」です。

ROLLUP と同じように、CUBE グループ化も、一連の*グループ化集合* と見なすことができます。 CUBE の場合、*グループ化式リスト* のすべての順列が総計とともに計算されます。したがって、CUBE の *n* 個のエレメントは、 2**n* (2 の *n* 乗) の*グループ化集合* に変換されます。たとえば、次のように指定するとします。

```
GROUP BY CUBE(a,b,c)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (b,c)
                       (a)
                       (b)
                       (c)
                       () )
```

CUBE の 3 個のエレメントは、8 個のグループ化集合に変換されることに注意してください。

エレメントの指定の順序は、CUBE の場合、重要ではありません。 CUBE (DayOfYear, Sales_Person)' と 'CUBE (Sales_Person, DayOfYear)' は同じ結果セットを生成します。「同じ」とは、結果セットの順序ではなく、結果セットの内容を指します。 ORDER BY 文節は、結果セットの行の順序を保証する唯一の方法です。 454ページの例 C4 に、CUBE の使用例が示されています。

grouping-expression-list

grouping-expression-list (グループ化式リスト) は、 CUBE または ROLLUP 演算のエレメント数を定義するために、 CUBE または ROLLUP 文節で使用されます。これは、複数の*グループ化式* を持つエレメントを区切るために括弧を使用して制御されます。

グループ化式 の規則については、 437ページの『GROUP BY 文節』 で説明しています。たとえば、照会が、County ではなく Province 内の City の ROLLUP について合計費用を戻すものと想定します。しかし、以下の文節の場合、

```
GROUP BY ROLLUP(Province, County, City)
```

County についての不要な小計行が生じます。 以下の文節では、

```
GROUP BY ROLLUP(Province, (County, City))
```

複合 (County, City) が ROLLUP の 1 つのエレメントを形成するので、この文節を使用する照会は、必要な結果を生じます。つまり、次のように 2 つのエレメントからなる ROLLUP の場合、

```
GROUP BY ROLLUP(Province, (County, City))
```

以下を生成します。

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province)
                       () )
```

一方、3 つのエレメントからなる ROLLUP の場合は、次のようになります。

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province, County)
                       (Province)
                       () )
```

453ページの例 C2 でも、複合列値が使用されています。

grand-total

CUBE および ROLLUP は、全体の集約 (総計) である行を戻します。これは、GROUPING SET 文節に空の括弧を使用して別々に指定することができます。また、GROUP BY 文節に直接指定することもできます。これは照会の結果には影響しません。 454ページの例 C4 では、総計構文を使用しています。

グループ化集合の組み合わせ

これは、任意のタイプの GROUP BY 文節を組み合わせるのに使用することができます。単純な*グループ化式* のフィールドを他のグループと組み合わせる場合、結果の*グループ化集合* の始めに「追加」されます。 ROLLUP 式または CUBE 式を組み合わせる場合、それらの式は、残りの式では「乗数」のように動作し、 ROLLUP または CUBE の定義に応じて追加のグループ化集合項目を生成します。

たとえば、*グループ化式* のエレメントを組み合わせると、次のようになります。

```
GROUP BY a, ROLLUP(b,c)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a) )
```

もしくは

```
GROUP BY a, b, ROLLUP(c,d)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b) )
```

ROLLUP エレメントを組み合わせると、次のようになります。

```
GROUP BY ROLLUP(a), ROLLUP(b,c)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a)
                       (b,c)
                       (b)
                       () )
```

同様に、

```
GROUP BY ROLLUP(a), CUBE(b,c)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (a)
                       (b,c)
                       (b)
                       (c)
                       () )
```

CUBE と *ROLLUP* のエレメントの組み合わせは次のようになります。

```
GROUP BY CUBE(a,b), ROLLUP(c,d)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b)
                       (a,c,d)
                       (a,c)
                       (a)
                       (b,c,d)
                       (b,c)
                       (b)
                       (c,d)
                       (c)
                       () )
```

単純なグループ化式 の場合のように、グループ化集合を組み合わせると、各グループ化集合内で重複したものが除去されます。たとえば、次のようになります。

```
GROUP BY a, ROLLUP(a,b)
```

これは、以下と同じ意味になります。

```
GROUP BY GROUPING SETS((a,b)
                       (a) )
```

グループ化集合を組み合わせる場合の詳しい例は、完全な CUBE 集約について戻される特定行を除去する結果セットを構成する場合です。

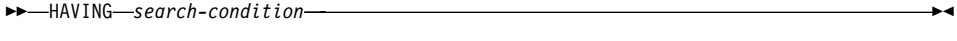
たとえば、以下の GROUP BY 文節について考えてみます。

```
GROUP BY Region,
         ROLLUP(Sales_Person, WEEK(Sales_Date)),
         CUBE(YEAR(Sales_Date), MONTH (Sales_Date))
```

GROUP BY のすぐ右側にリストされている列は単純にグループ化され、ROLLUP の後の括弧内の列がロールアップされ、 CUBE の後の括弧内の列は 3 乗されます。したがって、上記の文節の結果は、YEAR 内の MONTH のキューブが生成されてから、 REGION 内の Sales_Person 内の WEEK の集約内でロールアップが行われます。この結果は、Region、 Sales_Person または WEEK(Sales_Date) の総計行にもクロス集計行にもならないため、生成される行は、以下の文節より少なくなります。

```
GROUP BY ROLLUP (Region, Sales_Person, WEEK(Sales_Date),
                 YEAR(Sales_Date), MONTH(Sales_Date) )
```

HAVING 文節



HAVING 文節は、 *search-condition (検索条件)* が真である R のグループで構成される中間結果表を指定します。 R は、副選択のそれ以前の文節の結果です。その文節が GROUP BY ではない場合、R はグループ化列のない単一のグループとみなされます。

検索条件内の各*列名* は、以下のいずれかを満たすものであることが必要です。

- R のグループ化列を明確に識別すること。
- 列関数内で指定されていること。
- 相関参照であること。副選択外の*表参照*の列を識別している*列名* は、相関参照となります。

検索条件が適用される R のグループは、検索条件内の各列関数 (引き数が相関参照である関数を除く) の引き数を提供するものとなります。

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 R のグループ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おいて使用されるもの、とみなすことができます。実際には、副照会が各グループごとに実行されるのは、その中に相関参照が含まれる場合だけです。この違いについては、447ページの例 A6 および 447ページの例 A7 を参照してください。

R のグループに対する相関参照は、グループ化列を指定するものであるか、あるいは列関数に含まれるものでなければなりません。

HAVING が GROUP BY なしで使用される場合、選択リストには、列関数内の列名、相関列参照、リテラル、または特殊レジスターしか使えません。

副選択の例

例 A1: EMPLOYEE 表の列および行をすべて選択します。

```
SELECT * FROM EMPLOYEE
```

例 A2: EMP_ACT 表および EMPLOYEE 表を結合し、 EMP_ACT 表からすべての列を選択して、 EMPLOYEE 表の従業員の姓 (LASTNAME) を結果の各行に追加します。

```
 SELECT EMP_ACT.*, LASTNAME
FROM EMP_ACT, EMPLOYEE
WHERE EMP_ACT.EMPNO = EMPLOYEE.EMPNO
```

例 A3: EMPLOYEE 表と DEPARTMENT 表を結合し、 1930 年よりも前に生まれた (BIRTHDATE) 従業員すべての従業員番号 (EMPNO)、従業員の姓 (LASTNAME)、部門番号 (EMPLOYEE 表の WORKDEPT と DEPARTMENT 表の DEPTNO)、および部門名 (DEPTNAME) を選択します。

```
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
 FROM EMPLOYEE, DEPARTMENT
 WHERE WORKDEPT = DEPTNO
 AND YEAR(BIRTHDATE) < 1930
```

例 A4: EMPLOYEE 表の中で同じジョブ・コードをもつ行のグループごとに、ジョブ (JOB) と給与 (SALARY) の最低額と最高額を選択します。ただし、グループの中でも、複数の行が含まれ、給与の最高額が 27000 以上のグループについてのみ選択を行います。

```
SELECT JOB, MIN(SALARY), MAX(SALARY)
  FROM EMPLOYEE
  GROUP BY JOB
  HAVING COUNT(*) > 1
  AND MAX(SALARY) >= 27000
```

例 A5: EMP_ACT 表の中から、部門 (WORKDEPT) 'E11' の従業員 (EMPNO) についてのすべての行を選択します。 (従業員部門番号は、EMPLOYEE 表に示されています。)

```
SELECT *
  FROM EMP_ACT
  WHERE EMPNO IN
           (SELECT EMPNO
               FROM EMPLOYEE
               WHERE WORKDEPT = 'E11')
```

例 A6: EMPLOYEE 表から、給与の最高額が従業員全体の平均給与に満たないすべての部門について、部門番号 (WORKDEPT) と部門別給与 (SALARY) の最高額を選択します。

```
SELECT WORKDEPT, MAX(SALARY)
  FROM EMPLOYEE
  GROUP BY WORKDEPT
  HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                        FROM EMPLOYEE)
```

この例では、HAVING 文節の副照会は 1 度だけ実行されることになります。

例 A7: EMPLOYEE 表を使用して、部門別給与の最高額が他のすべての部門の平均給与より少ない部門の部門番号 (WORKDEPT) とその部門別給与 (SALARY) の最高額を選択します。

```
SELECT WORKDEPT, MAX(SALARY)
   FROM EMPLOYEE EMP_COR
    GROUP BY WORKDEPT
    HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                          FROM EMPLOYEE
                          WHERE NOT WORKDEPT = EMP_COR.WORKDEPT)
```

例 A6 とは反対に、HAVING 文節の副照会は、各グループごとに実行する必要があります。

例 A8: セールス担当員の従業員番号と給与、およびその部門の給与平均額と人数とを調べます。

この照会では、まずネストされた表式 (DINFO) を作成して、AVGSALARY 列と EMPCOUNT 列、また WHERE 文節で使用される DEPTNO 列を入手する必要があります。

```
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY, DINFO.AVGSALARY, DINFO.EMPCOUNT
FROM EMPLOYEE THIS_EMP,
      (SELECT OTHERS.WORKDEPT AS DEPTNO,
                        AVG(OTHERS.SALARY) AS AVGSALARY,
                        COUNT(*) AS EMPCOUNT
        FROM EMPLOYEE OTHERS
         GROUP BY OTHERS.WORKDEPT
       ) AS DINFO
WHERE THIS_EMP.JOB = 'SALESREP'
AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

このような場合には、ネストした表式を使用することによって、 DINFO 視点を正規の視点として作成することによりオーバーヘッドを軽減することができます。ステートメントの準備中に、視点のカタログにはアクセスされません。これは、照会の残りの部分の文脈により、視点によって考慮する必要があるのはセールス担当の部門の行だけだからです。

例 A9: 5 つの従業員のランダム・グループについて平均的な教育レベルと給与を表示します。

この照会では、各従業員のランダム値を GROUP BY 文節で使用できるようにするために、ネストした表式を使用してこのランダム値を設定する必要があります。

```
SELECT RANDID , AVG(EDLEVEL), AVG(SALARY)
FROM ( SELECT EDLEVEL, SALARY, INTEGER(RAND()*5) AS RANDID
         FROM EMPLOYEE
     ) AS EMPRAND
GROUP BY RANDID
```

結合の例

例 B1: この例では、表 J1 および J2 を使用した種々の結合の結果を示しています。これらの表には、以下の行が含まれています。

```
   SELECT * FROM J1
W   X
--- ------
A       11
B       12
C       13
SELECT * FROM J2
Y   Z
--- ------
A       21
C       22
D       23
```

以下の照会では、両方の表の最初の列が一致する J1 および J2 の内部結合を行っています。

```
   SELECT * FROM J1 INNER JOIN J2 ON W=Y
W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
```

この内部結合の例では、J1 の列 W='C' の行および J2 の列 Y='D' の行が、結果に含まれていません。これは、これらの行がもう一方の表に一致するものがないからです。次のような代替形式の内部結合照会で同じ結果が生成されることに注意してください。

```
   SELECT * FROM J1, J2 WHERE W=Y
```

以下の左外部結合では、J2 の列がヌル値である J1 の欠落行を戻します。 J1 のすべての行が含まれます。

```
   SELECT * FROM J1 LEFT OUTER JOIN J2 ON W=Y
W   X      Y   Z
--- ------ --- ------
A       11 A       21
B       12 -        -
C       13 C       22
```

以下の右外部結合では、J1 の列がヌル値である J2 の欠落行を戻します。 J2 のすべての行が含まれます。

```
   SELECT * FROM J1 RIGHT OUTER JOIN J2 ON W=Y
W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
```

以下の全外部結合では、ヌル値である J1 と J2 の両方の欠落行を戻します。J1 と J2 の両方のすべての行が含まれます。

```
   SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
B       12 -        -
```

例 B2: 上記の例の表 J1 および J2 を使用して、述部が検索条件に追加されるとどうなるかを調べます。

```
   SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=13
W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

条件を追加すると、内部結合は、 449ページの例 B1 の内部結合と比較して 1 行のみを選択します。

全外部結合に対するこの影響に注意してください。

```
   SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=13
W   X      Y   Z
--- ------ --- ------
-        - A       21
C       13 C       22
-        - D       23
A       11 -        -
B       12 -        -
```

内部結合には 1 行のみがあり、両方の表のすべての行を戻す必要があるため、結果は 5 行になります (追加の述部がない場合の 4 行と比較して)。

以下の照会では、同じ述部を WHERE 文節に追加することにより、まったく異なる結果を生成される場合を示しています。

```
   SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=13
W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

WHERE 文節は、全外部結合の中間結果の後に適用されます。この中間結果は、449ページの例 B1 の全外部結合照会の結果と同じになります。 WHERE 文節は、この中間結果に適用され、X=13 の行を除くすべての行を除去します。外部結合を行う場合に、述部の位置の選択によって、結果に大きな影響を与える可能性があります。述部が X=13 ではなく X=12 であるとどうなるかを考えてみます。以下の内部結合は行を戻しません。

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=12
```

したがって、全外部結合は 6 行を返します。つまり、J2 の列がヌル値である J1 の 3 行と、J1 の列がヌル値である J2 の 3 行です。

```
   SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=12
W   X      Y   Z
--- ------ --- ------
-        - A       21
-        - C       22
-        - D       23
A       11 -        -
B       12 -        -
C       13 -        -
```

追加の述部が WHERE 文節にある場合には、1 行が戻されます。

```
   SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=12
W   X      Y   Z
--- ------ --- ------
B       12 -        -
```

例 B3: 管理者のいない部門も含めて、すべての部門を従業員番号と管理者の姓と共にリストします。

```
SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
     ON MGRNO = EMPNO
```

例 B4: 管理者のいない従業員も含めて、すべての従業員の番号と姓を管理者の従業員番号と姓と共にリストします。

```
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E LEFT OUTER JOIN
                                    DEPARTMENT INNER JOIN EMPLOYEE M
     ON MGRNO = M.EMPNO
     ON E.WORKDEPT = DEPTNO
```

内部結合は、DEPARTMENT 表で識別されるすべての管理者の姓を判別し、左外部結合により、対応する部門が DEPARTMENT にない場合であっても各従業員がリストされていることが保証されます。

グループ化集合、CUBE、および ROLLUP の例

例 C1 ～ 454ページの例 C4 の照会では、述部 'WEEK(SALES_DATE) = 13' に基づいて SALES 表の行のサブセットを使用しています。

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SALES AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
```

これは次の結果になります。

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 LUCCHESSI                 3
         13           6 LUCCHESSI                 1
         13           6 LEE                       2
         13           6 LEE                       2
         13           6 LEE                       3
         13           6 LEE                       5
         13           6 GOUNOT                    3
         13           6 GOUNOT                    1
         13           6 GOUNOT                    7
         13           7 LUCCHESSI                 1
         13           7 LUCCHESSI                 2
         13           7 LUCCHESSI                 1
         13           7 LEE                       7
         13           7 LEE                       3
         13           7 LEE                       7
         13           7 LEE                       4
         13           7 GOUNOT                    2
         13           7 GOUNOT                   18
         13           7 GOUNOT                    1
```

例 C1: これは、3 つの列に対して基本の GROUP BY 文節を使用している照会です。

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
      SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

これは次のような結果になります。

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                   11
         13           6 LEE                      12
```

```
          13           6 LUCCHESSI                4
          13           7 GOUNOT                  21
          13           7 LEE                     21
          13           7 LUCCHESSI                4
```

例 C2: SALES 表の行の 2 つのグループ化集合に基づいて結果を生成します。

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
      SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY GROUPING SETS ( (WEEK(SALES_DATE), SALES_PERSON),
                         (DAYOFWEEK(SALES_DATE), SALES_PERSON))
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

これは次のような結果になります。

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
          13           - GOUNOT                  32
          13           - LEE                     33
          13           - LUCCHESSI                8
           -           6 GOUNOT                  11
           -           6 LEE                     12
           -           6 LUCCHESSI                4
           -           7 GOUNOT                  21
           -           7 LEE                     21
           -           7 LUCCHESSI                4
```

WEEK 13 の行は、最初のグループ化集合のものであり、それ以外の行は 2 番目のグループ化集合のものです。

例 C3: 例 C2 のグループ化集合に含まれる 3 つの特殊な列を使用して、ROLLUP を実行する場合、(WEEK、DAY_WEEK、SALES_PERSON)、(WEEK、DAY_WEEK)、(WEEK) および総計のグループ化集合を見ることができます。

```
     SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
      SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

これは次のような結果になります。

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
          13           6 GOUNOT                  11
          13           6 LEE                     12
```

```
          13           6 LUCCHESSI                4
          13           6 -                       27
          13           7 GOUNOT                  21
          13           7 LEE                     21
          13           7 LUCCHESSI                4
          13           7 -                       46
          13           - -                       73
           -           - -                       73
```

例 C4: 453ページの例 C3 と同じ照会を実行して ROLLUP を CUBE に置き換える場合、結果に (WEEK、SALES_PERSON)、(DAY_WEEK、SALES_PERSON)、 (DAY_WEEK)、(SALES_PERSON) の追加のグループ化集合を見ることができます。

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
      SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY CUBE ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

これは次のような結果になります。

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
          13           6 GOUNOT                  11
          13           6 LEE                     12
          13           6 LUCCHESSI                4
          13           6 -                       27
          13           7 GOUNOT                  21
          13           7 LEE                     21
          13           7 LUCCHESSI                4
          13           7 -                       46
          13           - GOUNOT                  32
          13           - LEE                     33
          13           - LUCCHESSI                8
          13           - -                       73
           -           6 GOUNOT                  11
           -           6 LEE                     12
           -           6 LUCCHESSI                4
           -           6 -                       27
           -           7 GOUNOT                  21
           -           7 LEE                     21
           -           7 LUCCHESSI                4
           -           7 -                       46
           -           - GOUNOT                  32
           -           - LEE                     33
           -           - LUCCHESSI                8
           -           - -                       73
```

例 C5: SALES 表から選択された行の総計と、 SALES_PERSON および MONTH によって集計された行のグループを含む結果セットを入手します。

```
SELECT SALES_PERSON,
       MONTH(SALES_DATE) AS MONTH,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS ( (SALES_PERSON, MONTH(SALES_DATE)),
                         ()
                       )
ORDER BY SALES_PERSON, MONTH
```

これは次のような結果になります。

```
SALES_PERSON    MONTH       UNITS_SOLD
--------------- ----------- -----------
GOUNOT                    3          35
GOUNOT                    4          14
GOUNOT                   12           1
LEE                       3          60
LEE                       4          25
LEE                      12           6
LUCCHESSI                 3           9
LUCCHESSI                 4           4
LUCCHESSI                12           1
          - -                       155
```

例 C6: この例では、2 つの単純な ROLLUP 照会を示し、その後に、2 つの ROLLUP を単一結果セットのグループ化集合として扱い、グループ化集合に含まれている列ごとに行の順序を指定する照会を示しています。

例 C6-1:

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) )
ORDER BY WEEK, DAY_WEEK
```

結果は次のようになります。

```
WEEK        DAY_WEEK    UNITS_SOLD
----------- ----------- -----------
         13           6          27
         13           7          46
         13           -          73
         14           1          31
         14           2          43
         14           -          74
         53           1           8
         53           -           8
          - -                     155
```

例 C6-2:

```
SELECT MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( MONTH(SALES_DATE), REGION );
ORDER BY MONTH, REGION
```

結果は次のようになります。

```
MONTH       REGION          UNITS_SOLD
----------- --------------- -----------
          3 Manitoba                 22
          3 Ontario-North             8
          3 Ontario-South            34
          3 Quebec                   40
          3 -                       104
          4 Manitoba                 17
          4 Ontario-North             1
          4 Ontario-South            14
          4 Quebec                   11
          4 -                        43
         12 Manitoba                  2
         12 Ontario-South             4
         12 Quebec                    2
         12 -                         8
          - -                       155
```

例 C6-3:

```
SELECT WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS ( ROLLUP( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) ),
                         ROLLUP( MONTH(SALES_DATE), REGION )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

結果は次のようになります。

```
WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
----------- ----------- ----------- --------------- -----------
         13           6           - -                        27
         13           7           - -                        46
         13           -           - -                        73
         14           1           - -                        31
         14           2           - -                        43
         14           -           - -                        74
         53           1           - -                         8
         53           -           - -                         8
          -           -           3 Manitoba                 22
          -           -           3 Ontario-North             8
          -           -           3 Ontario-South            34
          -           -           3 Quebec                   40
```

```
-          -          3 -                    104
-          -          4 Manitoba              17
-          -          4 Ontario-North          1
-          -          4 Ontario-South         14
-          -          4 Quebec                11
-          -          4 -                     43
-          -         12 Manitoba               2
-          -         12 Ontario-South          4
-          -         12 Quebec                 2
-          -         12 -                      8
-          -          - -                    155
-          -          - -                    155
```

2 つの ROLLUP をグループ化集合として使用すると、結果に重複した行が含まれます。総計行も 2 つになります。

ORDER BY を使用すると結果にどのような影響があるかを調べてみます。

- 最初のグループ化集合では、week 53 が最後に位置変更されています。
- 2 番目のグループ化集合では、month 12 が最後に位置付けられ、地域がアルファベット順になっています。
- ヌル値は上位に分類されます。

例 C7: 単一のパスで複数の ROLLUP を実行する照会 (たとえば 456ページの例 C6-3) では、各行を生成したのがどのグループ化集合であるかを示すことができます。以下のステップは、結果セット内の各行の発生点を示す列 (GROUP と呼ばれます) を提供する方法を示しています。結果セットの行を生成したのが、2 つのグループ化集合のいずれであるかということです。

ステップ *1:* VALUES 文節から選択する照会 (代替形式の全選択) を使用して、新しいデータ値を「生成する」方法を導入します。この照会は、2 つの列 ″R1″ と ″R2″、および 1 行のデータがある、 ″X″ と呼ばれる表を得る方法を示しています。

```
SELECT R1,R2
FROM (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2);
```

結果は次のようになります。

```
R1      R2
------- -------
GROUP 1 GROUP 2
```

ステップ *2:* SALES 表を使ってこの表 ″X″ のクロス・プロダクトを生成します。これにより、各行に列 ″R1″ および ″R2″ が追加されます。

```
SELECT R1, R2, WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
```

```
       REGION,
       SALES AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
```

これにより、各行に列 ″R1″ および ″R2″ が追加されます。

ステップ *3:* これで、これらの列をグループ化集合と組み合わせて、 ROLLUP 分析にこれらの列を含めることができるようになりました。

```
SELECT R1, R2,
       WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE))),
                        (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

結果は次のようになります。

```
R1      R2      WEEK     DAY_WEEK  MONTH     REGION          UNITS_SOLD
------- ------- -------- --------- --------- --------------- -----------
GROUP 1 -             13         6         - -                          27
GROUP 1 -             13         7         - -                          46
GROUP 1 -             13         -         - -                          73
GROUP 1 -             14         1         - -                          31
GROUP 1 -             14         2         - -                          43
GROUP 1 -             14         -         - -                          74
GROUP 1 -             53         1         - -                           8
GROUP 1 -             53         -         - -                           8
-       GROUP 2        -         -         3 Manitoba                   22
-       GROUP 2        -         -         3 Ontario-North               8
-       GROUP 2        -         -         3 Ontario-South              34
-       GROUP 2        -         -         3 Quebec                     40
-       GROUP 2        -         -         3 -                         104
-       GROUP 2        -         -         4 Manitoba                   17
-       GROUP 2        -         -         4 Ontario-North               1
-       GROUP 2        -         -         4 Ontario-South              14
-       GROUP 2        -         -         4 Quebec                     11
-       GROUP 2        -         -         4 -                          43
-       GROUP 2        -         -        12 Manitoba                    2
-       GROUP 2        -         -        12 Ontario-South               4
-       GROUP 2        -         -        12 Quebec                      2
-       GROUP 2        -         -        12 -                           8
-       GROUP 2        -         -         - -                         155
GROUP 1 -              -         -         - -                         155
```

ステップ *4:* R1 および R2 が異なるグループ化集合で使用されるため、 R1 の結果が非ヌル値である場合は常に R2 はヌル値であり、 R2 の結果が非ヌル値である場合は常に R1 はヌル値になることに注意してください。つまり、 COALESCE 関数を使用すれば、これらの列を単一列に統合することができる

ということです。また、ORDER BY 文節でこの列を使用すれば、 2 つのグループ化集合の結果をまとめることもできます。

```
SELECT COALESCE(R1,R2) AS GROUP,
       WEEK(SALES_DATE) AS WEEK,
      DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE))),
                        (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY GROUP, WEEK, DAY_WEEK, MONTH, REGION;
```

結果は次のようになります。

```
GROUP   WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
------- ----------- ----------- ----------- --------------- -----------
GROUP 1          13           6           - -                          27
GROUP 1          13           7           - -                          46
GROUP 1          13           -           - -                          73
GROUP 1          14           1           - -                          31
GROUP 1          14           2           - -                          43
GROUP 1          14           -           - -                          74
GROUP 1          53           1           - -                           8
GROUP 1          53           -           - -                           8
GROUP 1           -           -           - -                         155
GROUP 2           -           -           3 Manitoba                   22
GROUP 2           -           -           3 Ontario-North               8
GROUP 2           -           -           3 Ontario-South              34
GROUP 2           -           -           3 Quebec                     40
GROUP 2           -           -           3 -                         104
GROUP 2           -           -           4 Manitoba                   17
GROUP 2           -           -           4 Ontario-North               1
GROUP 2           -           -           4 Ontario-South              14
GROUP 2           -           -           4 Quebec                     11
GROUP 2           -           -           4 -                          43
GROUP 2           -           -          12 Manitoba                    2
GROUP 2           -           -          12 Ontario-South               4
GROUP 2           -           -          12 Quebec                      2
GROUP 2           -           -          12 -                           8
GROUP 2           -           -           - -                         155
```

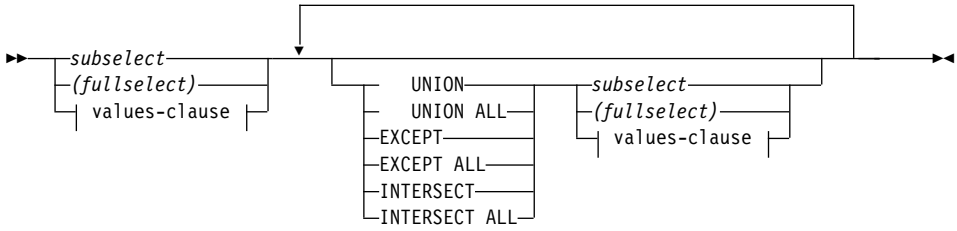
例 C8: 以下の例は、CUBE を実行する場合の種々の列関数の使用例を示しています。また、この例は、cast 関数および丸めを利用して、妥当な精度と位取りで 10 進数の結果を生成します。

```
SELECT MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD,
       MAX(SALES) AS BEST_SALE,
       CAST(ROUND(AVG(DECIMAL(SALES)),2) AS DECIMAL(5,2)) AS AVG_UNITS_SOLD
FROM SALES
GROUP BY CUBE(MONTH(SALES_DATE),REGION)
ORDER BY MONTH, REGION
```

これは次のような結果になります。

```
MONTH       REGION          UNITS_SOLD  BEST_SALE   AVG_UNITS_SOLD
----------- --------------- ----------- ----------- --------------
          3 Manitoba                 22           7           3.14
          3 Ontario-North             8           3           2.67
          3 Ontario-South            34          14           4.25
          3 Quebec                   40          18           5.00
          3 -                       104          18           4.00
          4 Manitoba                 17           9           5.67
          4 Ontario-North             1           1           1.00
          4 Ontario-South            14           8           4.67
          4 Quebec                   11           8           5.50
          4 -                        43           9           4.78
         12 Manitoba                  2           2           2.00
         12 Ontario-South             4           3           2.00
         12 Quebec                    2           1           1.00
         12 -                         8           3           1.60
          - Manitoba                 41           9           3.73
          - Ontario-North             9           3           2.25
          - Ontario-South            52          14           4.00
          - Quebec                   53          18           4.42
          - -                       155          18           3.87
```

全選択



values-clause:

Values-row:

全選択 (fullselect) は、選択ステートメント、 INSERT ステートメント、および CREATE VIEW ステートメントの構成要素の 1 つです。また、これはステートメントの構成要素である特定の述部の構成要素ともなります。述部の構成要素である*全選択* は、*副照会 (subquery)* と呼ばれます。括弧で囲んだ*全選択 (fullselect)* は、副照会と呼ばれることがあります。

セット演算子である UNION、EXCEPT、および INTERSECT は、関係演算子の合併、差、積に対応しています。

全選択 は結果表を指定します。集合演算子を使用しない*全選択* の結果は、指定した副選択または VALUES 文節の結果になります。

Values-clause

結果表の行の各列ごとに式を使用して実際の値を指定することによって、結果表を派生させます。複数の行を指定することができます。

NULL は、複数の *Values-row* でのみ使用することができ、同一列の少なくとも 1 行は NULL 以外でなければなりません (SQLSTATE 42826)。

Values-row は以下によって指定されます。

- 結果表の単一の列についての単一の式
- コンマで区切った n 個の式 (または NULL) を括弧で囲んだもの (n は結果表の列の数)

複数行からなる Values-clause には、各 *Values-row* に同数の式が必要です (SQLSTATE 42826)。

次に、Values-clause の例とその意味を示します。

```
VALUES (1),(2),(3)          - 3 rows of 1 column
VALUES 1, 2, 3              - 3 rows of 1 column
VALUES (1, 2, 3)            - 1 row of 3 columns
VALUES (1,21),(2,22),(3,23) - 3 rows of 2 columns
```

Values-clause は、 n 個の *Values-row* RE_1 ～ RE_n (n は 2 以上) で構成され、以下と同等です。

RE_1 UNION ALL RE_2 ... UNION ALL RE_n

これは、各 *Values-row* に対応する式は比較可能でなければならず (SQLSTATE 42825)、結果のデータ・タイプは 121ページの『結果のデータ・タイプに関する規則』 に基づくことを意味しています。

UNION または **UNION ALL**
2 つの結果表 (R1 と R2) を組み合わせて、新たな結果表を導きます。 UNION ALL を指定すると、結果は R1 と R2 のすべての行から構成されるものになります。 ALL オプションなしで UNION を指定すると、結果は R1 または R2 のいずれかの行すべての集合から、重複行を除去したものになります。しかしいずれにしても、UNION 表の各行は R1 か R2 のどちらかから取られた行です。

EXCEPT または **EXCEPT ALL**
2 つの結果表 (R1 と R2) を組み合わせて、新たな結果表を導きます。 EXCEPT ALL を指定すると、結果は、重複行の数を勘定に入れつつ、 R2 の中に対応する行のないすべての行で構成されるものになります。 ALL オプションなしで EXCEPT を指定すると、結果は、それぞれの重複行を除去してから R1 にのみ存在する行を取り出したもので構成されます。

INTERSECT または **INTERSECT ALL**
2 つの結果表 (R1 と R2) を組み合わせて、新たな結果表を導きます。 INTERSECT ALL を指定すると、結果は R1 と R2 の両方に含まれる行すべてで構成されるものになります。 ALL オプションなしで INTERSECT を指定すると、結果は、R1 と R2 の両方にある行すべての集合から重複行を除去したものになります。

結果表 R1 の中の列の数と R2 の中の列の数は、同じでなければなりません (SQLSTATE 42826)。 ALL キーワードを指定しない場合、R1 および R2 には、 255 バイトよりも大きいと宣言されたストリング列を含めてはなりません。また、データ・タイプが LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK であるストリング列や、それらのタイプのいずれかを基にした特殊タイプ、または構造タイプであるストリング列を含めることもできません (SQLSTATE 42907)。

結果の列の名前は、次のようになります。

- R1 の *n* 番目の列と R2 の *n* 番目の列の結果列の名前が同じ場合、 R の *n* 番目の列が結果列の名前になります。
- R1 の *n* 番目の列と R2 の *n* 番目の列の結果列の名前が異なる場合は、名前が生成されます。この名前を、ORDER BY 文節または UPDATE 文節の列名として使用することはできません。

生成された名前を調べるには、SQL ステートメントの DESCRIBE を実行して、 SQLNAME フィールドを参照します。

2 つの行が互いに重複していると言えるのは、最初の行の各値が 2 番目の行の対応する値に等しい場合です。 (重複を判別する場合、2 つのヌル値は等しいものとみなされます)。

複数の演算を 1 つの式の中に結合した場合は、括弧内の演算が先に実行されます。括弧がない場合、演算は左から右に実行されますが、例外として、すべての INTERSECT 演算は UNION または EXCEPT の演算の前に実行されます。

次の例では、表 R1 と R2 の値を左端に示しています。他にリストされている見出しは、R1 と R2 の種々のセット演算の結果の値を示しています。

R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTER-SECT ALL	INTER-SECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					

R1	R2	UNION	UNION ALL	EXCEPT	EXCEPT ALL	INTERSECT	INTERSECT ALL
4		2					
4		3					
5		3					
		3					
		3					
		3					
		4					
		4					
		4					
		5					

結果列のデータ・タイプの判別の規則については、 121ページの『結果のデータ・タイプに関する規則』 を参照してください。

ストリング列の変換の処理方法の規則については、 126ページの『ストリング変換に関する規則』 を参照してください。

全選択の例

例 1: EMPLOYEE 表からすべての列と行を選択します。

```
SELECT * FROM EMPLOYEE
```

例 2: EMPLOYEE 表の従業員で、その部門番号 (WORKDEPT) が 'E' で始まる部門に属しているか、**または**プロジェクト番号 (PROJNO) が 'MA2100'、'MA2110'、または 'MA2112' である EMP_ACT 表のプロジェクトに割り当てられている従業員すべての従業員番号 (EMPNO) をリストします。

```
SELECT EMPNO
   FROM EMPLOYEE
   WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

例 3: 例 2 と同じ照会を行い、さらに EMPLOYEE 表の行には 'emp'、EMP_ACT 表の行には 'emp_act' という “タグ” を付けます。例 2 の結果とは

異なり、この照会では、同じ EMPNO が複数回戻され、付加される “タグ” によりどの表からとられたかが示されます。

```
SELECT EMPNO, 'emp'
   FROM EMPLOYEE
   WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO, 'emp_act' FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

例 4: 例 2 と同じの照会を行いますが、重複行が除去されないように UNION ALL を使用します。

```
   SELECT EMPNO
     FROM EMPLOYEE
    WHEREWORKDEPTLIKE 'E%'
UNION ALL
SELECT EMPNO
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

例 5: 例 3 と同じ照会を行いますが、現在どの表にもない 2 人の従業員を追加して、それらの行に ″new″ というタグを付けます。

```
   SELECT EMPNO, 'emp'
    FROM EMPLOYEE
    WHEREWORKDEPTLIKE 'E%'
UNION
SELECT EMPNO, 'emp_act'
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
UNION
   VALUES ('NEWAAA', 'new'), ('NEWBBB', 'new')
```

例 6: この EXCEPT の例は、T1 に存在し、T2 に存在しない行をすべて生成します。

```
(SELECT * FROM T1)
EXCEPT ALL
(SELECT * FROM T2)
```

NULL 値が含まれていない場合、この例は次の例と同じ結果を戻します。

```
SELECT ALL *
FROM T1
WHERE NOT EXISTS (SELECT * FROM T2
                  WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

例 7: この INTERSECT の例は、表 T1 と T2 の両方にあるすべての行を生成し、重複した行を除去します。

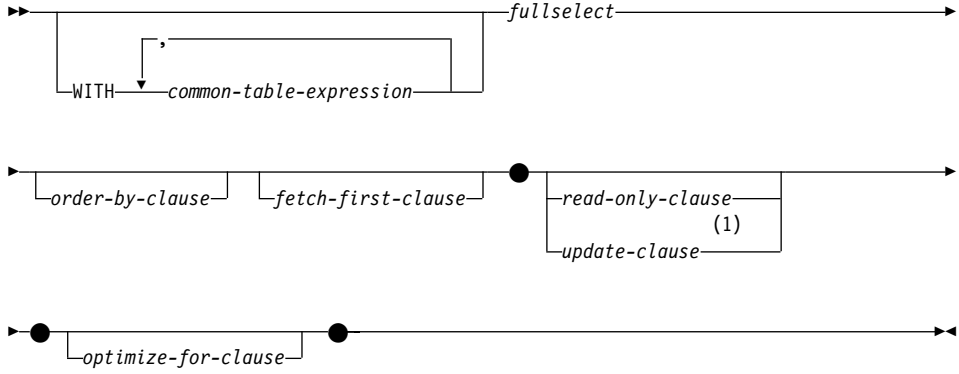
```
(SELECT * FROM T1)
INTERSECT
(SELECT * FROM T2)
```

NULL 値が含まれていない場合、この例は次の例と同じ結果を戻します。

```
SELECT DISTINCT * FROM T1
WHERE EXISTS (SELECT * FROM T2
                    WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

ここで、C1、C2、などは T1 と T2 の列を表します。

選択ステートメント



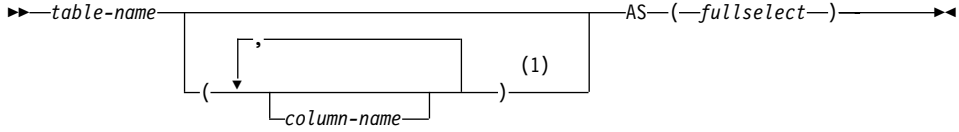
注:

1 UPDATE 文節 (update-clause) と ORDER BY 文節 (order-by-clause) の両方を、 同一の選択ステートメント内で指定することはできません。

選択ステートメント は、DECLARE CURSOR ステートメントに直接指定することや、準備した後で DECLARE CURSOR ステートメントで参照することができる照会の形式です。また、選択ステートメントはコマンド行プロセッサー (または同種のツール) を使用して動的 SQL ステートメントの使用によって出すことが可能で、それにより、結果表を画面に表示することもできます。いずれの場合も、*選択ステートメント* によって指定される表は、全選択 (fullselect) の結果です。

ニックネームを参照する*選択ステートメント* については、 DECLARE CURSOR ステートメントに直接に指定することはできません。

共通表式



注:

1 共通表式 (common table expression) が再帰的である場合、 あるいは全選択の結果として列名が重複する場合は、 列名を指定する必要があります。

共通表式 を使用すると、結果表を *table-name* (表名) によって定義して、それをその後に続く全選択の任意の FROM 文節に指定できるようにすることができます。単一の WITH キーワードの後に、複数の共通表式を指定することができます。指定する各共通表式は、それ以降の共通表式の FROM 文節の中でも名前によって参照することができます。

列のリストを指定する場合、その中の列の名前の数は、全選択の結果表内の列数と同じ数でなければなりません。各 *column-name* (列名) は、固有かつ非修飾でなければなりません。これらの列名を指定しない場合、共通表式の定義に使用された全選択の選択リストから名前が取られます。

*共通表式*の *table-name* は、同じステートメントの他の共通表式の *table-name* すべてと異なるものでなければなりません (SQLSTATE 42726)。共通表式が INSERT ステートメントに指定されている場合、*table-name* を、その挿入の対象である表または視点の名前にすることはできません (SQLSTATE 42726)。共通表式の *table-name* は、その全選択を通じて、どの FROM 文節の中でも表名として指定することができます。共通表式の *table-name* は、(カタログの中で) 同じ修飾名の既存の表、視点、または別名を指定変更するものとなります。

同じステートメントの中に複数の共通表式が定義されている場合、共通表式相互間の循環参照があってはなりません (SQLSTATE 42835)。*循環参照* が生じるのは、 2 つの共通表式 *dt1* と *dt2* が作成された場合に、 *dt1* が *dt2* を参照し、 *dt2* が *dt1* を参照するようになる場合です。

共通表式 は、 CREATE VIEW および INSERT の各ステートメントの前でもオプションとして使用できます。

共通表式 は、次のような場合に使用することができます。

- 視点の代わりに使用して、視点が作成されないようにするため (視点を一般的に使用する必要がなく、定位置の更新や削除を使わない場合)
- 決定的でなく外部処理を伴うスカラー副選択または関数から得られる列によりグループ化できるようにする場合
- 必要な結果表がホスト変数に基づいたものである場合
- 同じ結果表を *全選択* で共用する必要がある場合
- 結果表を再帰的に派生させる必要がある場合

全選択 の FROM 文節の中にそれ自体への参照が含まれる場合、その共通表式は、*再帰的共通表式* です。再帰処理を使用した照会は、部品構成表 (BOM)、予約システム、およびネットワーク計画などのアプリケーションをサポートする上で役立ちます。例については、1441ページの『付録M. 再帰の例: 部品構成表』 を参照してください。

再帰的共通表式では、次のことが成り立っていなければなりません。

- 再帰サイクルの一部をなす各全選択は、 SELECT または SELECT ALL で始まっていなければなりません。 SELECT DISTINCT は使用できません (SQLSTATE 42925)。また、集合の和を求める場合には UNION ALL を使用する必要があります (SQLSTATE 42925)。
- 共通表式の *table-name* (表名) の後には、必ず列名を指定する必要があります (SQLSTATE 42908)。
- 最初の UNION の最初の全選択 (初期化全選択) には、どの FROM 文節の共通表式のどの列に対する参照も含まれていてはなりません (SQLSTATE 42836)。
- 共通表式の列名が反復全選択において参照される場合、その列のデータ・タイプ、長さ、およびコード・ページは、初期化全選択に基づいて決められます。反復全選択の中の対応する列のデータ・タイプと長さは、初期化全選択に基づいて決められたデータ・タイプと長さと同じでなければならず、コード・ページは一致していなければなりません (SQLSTATE 42825)。ただし、文字ストリング・タイプの場合は、 2 つのデータ・タイプの長さが違っても構いません。この場合、反復全選択の列の長さは、初期化全選択から決められた長さに常に割り当て可能な長さでなければなりません。
- 再帰サイクルの一部をなす各全選択には、列関数、GROUP BY 文節、または HAVING 文節が含まれていてはなりません (SQLSTATE 42836)。

  これらの全選択の FROM 文節には、再帰サイクルの一部である共通表式に対する参照を多くても 1 つまで含めることができます (SQLSTATE 42836)。

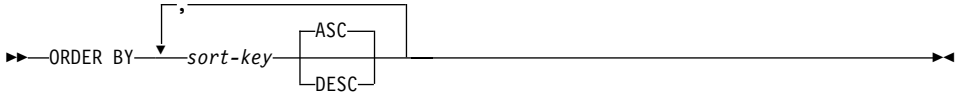
- 副照会 (スカラーまたは多値) が再帰サイクルの一部であってはなりません (SQLSTATE 42836)。

再帰的共通表式を開発するときには、無限再帰サイクル (ループ) が作成される恐れについて常に注意してください。再帰サイクルは、必ず終了するようにしてください。これは、関係しているデータが循環している場合に特に重要です。再帰的共通表式には、無限ループを防止する述部を含めるようにしてください。再帰的共通表式には、次のものを含めるようにしてください。

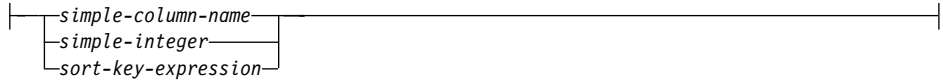
- 反復全選択の中に、定数ずつ増分される整数列。
- ″counter_col < constant″ または ″counter _col < :hostvar″ の形式の反復全選択の WHERE 文節の述部。

この構文が再帰的共通表式に見つからないなら、警告が出されます (SQLSTATE 01605)。

ORDER BY 文節



sort-key:



ORDER BY 文節は、結果表の行の順序を指定します。単一の分類指定 (方向が関連した 1 つの *sort-key* (分類キー)) が指定された場合、行は、その分類指定の値によって順序付けられます。複数の分類指定を指定すると、行の順序は、最初に指定された分類指定の値によって分類され、次に 2 番目に指定された分類指定の値によって分類されます。各分類キー の長さ属性は、文字列の場合は 255 文字以下、漢字列の場合は 127 文字以下でなければならず (SQLSTATE 42907)、データ・タイプを LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、DBCLOB、DATALINK にしたり、それらのタイプのいずれかを基にした特殊タイプ、または構造タイプにすることもできません (SQLSTATE 42907)。

選択リストに指定された列は、 *simple-integer* または *simple-column-name* である分類キー によって識別することができます。選択リストに指定されていない列は、*simple-integer*、もしくは場合によっては、選択リストの式と一致する *sort-key-expression* (*sort-key-expression* の詳細を参照) によって識別されなければなりません。列は、AS 文節の指定がなく、しかもその列が定数、演算子を含む式、または関数から派生した列の場合には無名です。[54]

順序付けは、『第3章 言語要素』 で説明した比較規則に従って行われます。ヌル値は、他のどのような値よりも高位として扱われます。 ORDER BY 文節で行が完全に順序付けされない場合、指定されたすべての列の値が重複する複数の行は、任意の順序で表示されます。

simple-column-name
通常、結果表の列を識別します。この場合、*simple-column-name (単純列名)* は、選択リストに指定された列の列名でなければなりません。

54. セット演算子 (UNION、INTERSECT、または EXCEPT) を伴う全選択の結果列の名前を判別する規則については、461ページの『全選択』を参照してください。

また、照会が副選択である場合、*simple-column-name* として、 FROM 文節で識別される表、視点、またはネストされた表の列名も指定することができます。副選択が以下の場合は、エラーが生じます。

- SELECT 文節に DISTINCT を指定する場合 (SQLSTATE 42822)
- グループ化された結果を生成する場合に、*単純列名* が*グループ化式* ではない場合 (SQLSTATE 42803)

結果の順序付けにどの列を使用するかの決定については、「分類キーの列名」で説明されています (473ページの『注意』 を参照)。

simple-integer
0 より大きく、結果表の列の数以下でなければなりません (SQLSTATE 42805)。整数 *n* は、結果表の *n* 番目の列を指定します。

sort-key-expression
単なる列名または無符号の整数定数ではない式。順序付けが適用される照会は、この形式の分類キーを使用するためには*副選択* でなければなりません。 *sort-key-expression (分類*キー式*)* には、相関スカラー全選択 (SQLSTATE 42703)、または外部処理を伴う関数 (SQLSTATE 42845) を含めることはできません。

*分類*キー式 内の列名は、「分類キーの列名」(473ページの『注意』 を参照) で説明されている規則に従っていなければなりません。

指定可能な式にさらに制約が加わる特殊な場合があります。

- DISTINCT が、副選択の SELECT 文節に指定されている (SQLSTATE 42822)。

  分類キー式は、副選択の選択リスト内の式と完全に一致しなければなりません (スカラー全選択は一致しません)。
- 副選択がグループ化されている (SQLSTATE 42803)。

  分類キー式は以下が可能です。
  - 副選択の選択リスト内の式である。
  - 副選択の GROUP BY 文節の*グループ化式* を含む。
  - 列関数、定数、またはホスト変数を含む。

ASC
列の値を昇順に使用します。これはデフォルト値です。

DESC
列の値を降順に使用します。

注意

- **分類キーの列名**
  - 列名が修飾されている場合

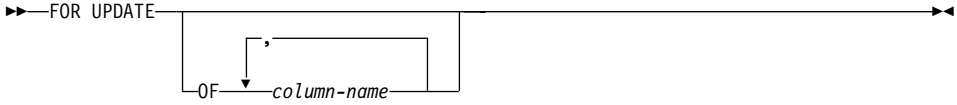
    照会は*副選択* でなければなりません (SQLSTATE 42877)。列名は、副選択の FROM 文節の表、視点、またはネストされた表の列を明確に識別する必要があります (SQLSTATE 42702)。列の値は、分類指定の値を計算するのに使用されます。
  - 列名が無修飾の場合
    - 照会は副選択です。

      列名が結果表の複数の列の名前と同一である場合、この列名は、順序付け副選択の FROM 文節内の表、視点、またはネストされた表の列を明確に識別する必要があります (SQLSTATE 42702)。列名が 1 つの列と同一である場合、その列は、分類指定の値を計算するのに使用されます。列名が結果表の列と同一でない場合、この列名は、選択ステートメントの全選択の FROM 文節の表、視点、またはネストされた表の列を明確に識別する必要があります (SQLSTATE 42702)。
    - 照会は、副選択ではありません (UNION、EXCEPT、または INTERSECT などの SET 演算が含まれます)。

      列名は、結果表の複数の列の名前と同一にすることはできません (SQLSTATE 42702)。列名は、結果表のちょうど 1 つの列と同一でなければなりません (SQLSTATE 42707)。この列は、分類指定の値を計算するのに使用されます。

    修飾された列名の詳細については、146ページの『あいまいさを避けるための列名修飾子』 を参照してください。
- **制限**: *分類キー式* またはその列が選択リストにない*単純列名* を使用すると、分類に使用される一時表にその列または式が追加される場合があります。これにより、表の列の数の制限、または表の行のサイズの制限に到達する場合があります。分類操作を行うのに一時表が必要になる場合、このような制限を超えると、エラーが生じます。

UPDATE 文節

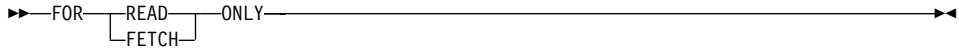


FOR UPDATE 文節は、それ以降の定位置 UPDATE ステートメントで更新可能となる列を指定します。各 *column-name* (列名) は非修飾でなければならず、全選択の最初の FROM 文節で指定された表または視点の列を指定するものでなければなりません。列名のない FOR UPDATE 文節を指定すると、全選択の最初の FROM 文節に指定された表または視点の列のうち更新可能な列すべてが含められます。

次のいずれかに該当する場合、FOR UPDATE 文節は使用できません。

- 選択ステートメントに関連付けられているカーソルが削除不能である (917ページの『注』 を参照)。
- 選択された列のいずれかがカタログ表の更新不能な列であり、 FOR UPDATE 列がその列を除外するのに使用されていない。

READ ONLY 文節



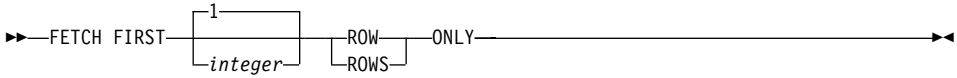
FOR READ ONLY 文節は、結果表が読み取り専用であり、カーソルを定位置 UPDATE ステートメントおよび定位置 DELETE ステートメントで参照できないことを指定します。 FOR FETCH ONLY も同じ意味です。

結果表の中には、最初から読み取り専用のものがあります。 (読み取り専用視点に基づく表など。) FOR READ ONLY は、このような表にも指定できますが、指定しても効果はありません。

更新と削除ができない結果表の場合、FOR READ ONLY (または FOR FETCH ONLY) を指定すると、データベース・マネージャーが、ブロック化を行い、排他ロックを回避することができるため、 FETCH 操作のパフォーマンスが向上する可能性があります。たとえば、FOR READ ONLY 文節または ORDER BY 文節のない動的 SQL ステートメントを含むプログラムでは、 FOR UPDATE 文節が指定されたかのようにして、データベース・マネージャーがカーソルをオープンする場合があります。したがって、定位置 UPDATE または DELETE ステートメントで照会を使用する場合以外は、パフォーマンスを向上させるために FOR READ ONLY 文節を使用するようにしてください。

読み取り専用結果表は、それが最初から読み取り専用であるか、それとも FOR READ ONLY (FOR FETCH ONLY) として指定されたのかには関係なく、定位置 UPDATE または DELETE ステートメントで参照することはできません。読み取り専用カーソルと更新可能カーソルについては、 914ページの『DECLARE CURSOR』 を参照してください。

FETCH FIRST 文節

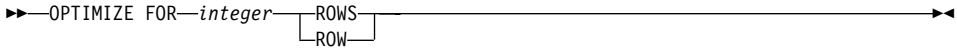


FETCH FIRST 文節 は、検索できる最大行数を設定します。この文節は、アプリケーションが最大 *integer* 行数までしか検索しないことを、データベース・マネージャーに認識させます。これは、この文節が指定されていない場合に、結果表にある行数に影響されません。 *integer* 行より多くの行を取り出そうとすると、通常のデータの終わりと同じように処理されます (SQLSTATE 02000)。 *integer* の値は、正の整数 (ゼロを除く) でなければなりません。

結果表を最初の *integer* 行に限定することにより、パフォーマンスが向上します。データベース・マネージャーは、一度最初の *integer* 行を判別すると、照会処理を停止します。 *fetch-first-clause* と *optimize-for-clause* の両方が指定されている場合には、これらの文節の *integer* 値のうちの小さい方を使用して通信バッファー・サイズが決定されます。これらの値は最適化処理専用です。

SELECT ステートメントに FETCH FIRST 文節を指定すると、カーソルを削除できなくなります (読み取り専用)。この文節は、FOR UPDATE 文節とともに指定することはできません。

OPTIMIZE FOR 文節



OPTIMIZE FOR 文節は、*選択ステートメント* の特殊な処理を要求します。この文節が省略されると、結果表のすべての行が検索されることが想定されます。指定されている場合には、検索される行数はおそらく *n* を超えないことを前提としています。ここで、*n* は整数 の値です。 *n* の値は、正の整数でなければなりません。 OPTIMIZE FOR 文節を使用すると、 *n* 個の行が検索されることを前提とする照会の最適化に影響を与えます。さらに、ブロックされているカーソルの場合、この文節は、各ブロックで戻される行の数に影響を与えます (すなわち、各ブロックで戻される行の数は *n* 行以下になります)。 *fetch-first-clause* と *optimize-for-clause* の両方が指定されている場合には、これらの文節の integer 値のうちの小さい方を使用して通信バッファー・サイズが決定されます。これらの値は最適化処理専用です。

この文節を指定しても、取り出される行の数が制限されることはなく、パフォーマンス以外ではどんな点でも結果に影響を与えることはありません。 OPTIMIZE FOR *n* ROWS を使用した場合、 *n* 個以下の行を取り出す場合にはパフォーマンスが向上することがありますが、 *n* 個を超える行を取り出す場合にはパフォーマンスが低下する可能性があります。

n に行のサイズを乗算した値が通信バッファーのサイズを超えると[55] 、 OPTIMIZE FOR 文節はデータ・バッファーに影響を与えません。

55. 通信バッファーのサイズは、RQRIOBLK または ASLHEAPSZ 構成パラメーターによって定義されます。詳細については、*管理の手引き* を参照してください。

選択ステートメントの例

例 1: EMPLOYEE 表からすべての列と行を選択します。

```
SELECT * FROM EMPLOYEE
```

例 2: PROJECT 表からプロジェクト名 (PROJNAME)、開始日 (PRSTDATE)、および終了日 (PRENDATE) を選択します。その日付が最新の終了日から順に結果表を配列します。

```
SELECT PROJNAME, PRSTDATE, PRENDATE
          FROM PROJECT
ORDER BY PRENDATE DESC
```

例 3: EMPLOYEE 表のすべての部門の部門番号 (WORKDEPT) と部門別給与 (SALARY) の平均額を選択します。結果表は、部門別給与の平均額の昇順に配列します。

```
SELECT WORKDEPT, AVG(SALARY)
 FROM EMPLOYEE
 GROUP BY WORKDEPT
 ORDER BY 2
```

例 4: C プログラムで使用する UP_CUR という名前のカーソルを宣言して、PROJECT 表の開始日 (PRSTDATE) と終了日 (PRENDATE) の列を更新します。プログラムは、各行のこれらの 2 つの値と、プロジェクト番号 (PROJNO) とを受け取る必要があります。

```
EXEC SQL  DECLARE UP_CUR CURSOR FOR
            SELECT PROJNO, PRSTDATE, PRENDATE
          FROM PROJECT
               FOR UPDATE OF PRSTDATE, PRENDATE;
```

例 5: この例では、SAL+BONUS+COMM に TOTAL_PAY という名前を付けます。

```
 SELECT SALARY+BONUS+COMM AS TOTAL_PAY
FROM EMPLOYEE
ORDER BY TOTAL_PAY
```

例 6: セールス担当員の従業員番号と給与、およびその部門の給与平均額と人数とを調べます。また、部門別給与平均額と、平均額の最高値も調べます。

ここでは、共通表式を使用することによって、 DINFO 視点を正規の視点として作成したときのオーバーヘッドを軽減します。ステートメントの作成中に、視点のカタログにはアクセスされません。これは、全選択の残りの部分の文脈により、視点によって考慮する必要があるのはセールス担当の部門の行だけだからです。

```
 WITH
   DINFO (DEPTNO, AVGSALARY, EMPCOUNT) AS
       (SELECT OTHERS.WORKDEPT, AVG(OTHERS.SALARY), COUNT(*)
         FROM EMPLOYEE OTHERS
          GROUP BY OTHERS.WORKDEPT
       ),
   DINFOMAX AS
       (SELECT MAX(AVGSALARY) AS AVGMAX FROM DINFO)
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY,
       DINFO.AVGSALARY, DINFO.EMPCOUNT, DINFOMAX.AVGMAX
 FROM EMPLOYEE THIS_EMP, DINFO, DINFOMAX
 WHERE THIS_EMP.JOB = 'SALESREP'
 AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

第6章 SQL ステートメント

この章では、SQL ステートメントの構文図、意味の説明、規則、および使用例を示しています。

表 18. SQL ステートメント

SQL ステートメント	機能	ページ
ALTER BUFFERPOOL	バッファー・プールの定義を変更する。	493
ALTER NICKNAME	ニックネームの定義を変更する。	495
ALTER NODEGROUP	ノード・グループの定義を変更する。	499
ALTER SERVER	サーバーの定義を変更する。	503
ALTER TABLE	表の定義を変更する。	507
ALTER TABLESPACE	表スペースの定義を変更する。	535
ALTER TYPE (構造化)	構造型の定義を変更する。	542
ALTER USER MAPPING	ユーザー許可マッピングの定義を変更する。	550
ALTER VIEW	参照タイプ列を変更して効力範囲を追加することによって、視点の定義を変更する。	553
BEGIN DECLARE SECTION	ホスト変数宣言セクションの始まりを示す。	555
CALL	ストアード・プロシージャーを呼び出す。	558
CLOSE	カーソルをクローズする。	567
COMMENT ON	オブジェクトの記述のコメントを置換または追加する。	569
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。	582
複合 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。	584
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。	589
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。	599
CREATE ALIAS	表、視点、または別の別名に対する別名を定義する。	608
CREATE BUFFERPOOL	新しいバッファー・プールを作成する。	612
CREATE DISTINCT TYPE	特殊データ・タイプを定義する。	616
CREATE EVENT MONITOR	モニターしたいデータベースの事象を指定する。	623
CREATE FUNCTION	ユーザー定義関数を登録する。	634

表 18. SQL ステートメント (続き)

SQL ステートメント	機能	ページ
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。	635
CREATE FUNCTION (外部表)	ユーザー定義外部表関数を登録する。	663
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。	681
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。	690
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。	701
CREATE FUNCTION MAPPING	関数マッピングを定義する。	710
CREATE INDEX	表の索引を定義する。	715
CREATE INDEX EXTENSION	構造タイプまたは特殊タイプを使用して、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。	723
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。	731
CREATE NICKNAME	ニックネームを定義する。	737
CREATE NODEGROUP	ノードグループを定義する。	740
CREATE PROCEDURE	ストアード・プロシージャーを登録する。	743
CREATE SCHEMA	スキーマを定義する。	762
CREATE SERVER	データ・ソースを連合データベースへ定義する。	766
CREATE TABLE	表を定義する。	771
CREATE TABLESPACE	表スペースを定義する。	828
CREATE TRANSFORM	変換関数を定義する。	839
CREATE TRIGGER	トリガーを定義する。	846
CREATE TYPE (構造化)	構造化データ・タイプを定義する。	859
CREATE TYPE MAPPING	データ・タイプ間でのマッピングを定義する。	887
CREATE USER MAPPING	ユーザー許可間でのマッピングを定義する。	893
CREATE VIEW	1 つまたは複数の表、視点、あるいはニックネームの視点を定義する。	895
CREATE WRAPPER	ラッパーを登録する。	912
DECLARE CURSOR	SQL カーソルを定義する。	914
DECLARE GLOBAL TEMPORARY TABLE	グローバル一時表を定義する。	920

表 18. SQL ステートメント (続き)

SQL ステートメント	機能	ページ
DELETE	1 つまたは複数の行を表から削除する。	930
DESCRIBE	準備済みの SELECT ステートメントの結果列を記述する。	936
DISCONNECT	活動状態の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。	941
DROP	データベース中のオブジェクトを削除する。	944
END DECLARE SECTION	ホスト変数宣言セクションの終わりを示す。	973
EXECUTE	準備済み SQL ステートメントを実行する。	975
EXECUTE IMMEDIATE	SQL ステートメントを準備して実行する。	981
EXPLAIN	選択したアクセス・プランについての情報を取り込む。	984
FETCH	行の値をホスト変数に割り当てる。	990
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。	994
FREE LOCATOR	ロケーター変数とその値の間の関連を除去する。	996
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。	997
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。	1001
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。	1003
GRANT (スキーマ特権)	スキーマについての特権を付与する。	1006
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。	1009
GRANT (表、視点、またはニックネーム特権)	表、視点、およびニックネームについての特権を付与する。	1011
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。	1020
INCLUDE	ソース・プログラムにコードまたは宣言を挿入する。	1023
INSERT	1 つまたは複数の行を表に挿入する。	1025
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。	1034
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。	1036
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。	1042
REFRESH TABLE	要約表のデータを最新表示する。	1053
RELEASE (接続)	1 つまたは複数の接続を解放保留状態にする。	1054
RELEASE SAVEPOINT	トランザクション内の保管点を解放する。	1056
RENAME TABLE	既存の表の名前を変更する。	1057

表 18. SQL ステートメント (続き)

SQL ステートメント	機能	ページ
RENAME TABLESPACE	既存の表スペースの名前を変更する。	1059
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。	1061
REVOKE (索引特権)	所定の索引に対する CONTROL 特権を取り消す。	1065
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。	1067
REVOKE (スキーマ特権)	スキーマの特権を取り消す。	1070
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。	1073
REVOKE (表、視点、またはニックネーム特権)	所定の表、視点、またはニックネームから特権を取り消す。	1075
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。	1081
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。	1083
SAVEPOINT	トランザクション内に保管点を設定する。	1086
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。	1089
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。	1091
SET CURRENT DEFAULT TRANSFORM GROUP	CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターの値を変更する。	1094
SET CURRENT DEGREE	CURRENT DEGREE 特殊レジスターの値を変更する。	1096
SET CURRENT EXPLAIN MODE	CURRENT EXPLAIN MODE 特殊レジスターの値を変更する。	1098
SET CURRENT EXPLAIN SNAPSHOT	CURRENT EXPLAIN SNAPSHOT 特殊レジスターの値を変更する。	1101
SET CURRENT PACKAGESET	パッケージ選択のためのスキーマ名を設定する。	1103
SET CURRENT QUERY OPTIMIZATION	CURRENT QUERY OPTIMIZATION 特殊レジスターの値を変更する。	1105
SET CURRENT REFRESH AGE	CURRENT REFRESH AGE 特殊レジスターの値を変更する。	1109
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。	1111
SET INTEGRITY	検査保留状態を設定し、制約違反の有無についてデータを検査する。	1113
SET PASSTHRU	データ・ソースのネイティブな SQL を、対象となるデータ・ソースへ直接に実行依頼するため、セッションをオープンする。	1124

表 18. SQL ステートメント (続き)

SQL ステートメント	機能	ページ
SET PATH	CURRENT PATH 特殊レジスターの値を変更する。	1126
SET SCHEMA	CURRENT SCHEMA 特殊レジスターの値を変更する。	1129
SET SERVER OPTION	サーバーのオプション設定をセットする。	1132
SET transition-variable	NEW 変換変数に値を割り当てる。	1134
SIGNAL SQLSTATE	エラーを通知する。	1139
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。	1141
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。	1153
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。	1155

SQL ステートメントの呼び出し方法

この章で説明する SQL ステートメントは、*実行可能*、または*実行不能* のいずれかに類別されます。各ステートメントの説明の*呼び出し* の項は、そのステートメントが実行可能か否かを示しています。

*実行可能*ステートメント には、以下の 4 つの呼び出し方法があります。

- 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。

注: 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。

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。各ステートメントの説明の*呼び出し* の項では、どの方式を使用できるかを示しています。

*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。

この章で説明しているステートメントに加えて、さらにもう 1 つの SQL ステートメント構成として、*選択 (SELECT)* ステートメント があります。 (467ページの『選択ステートメント』 を参照。) これは、他のステートメントと使用法が異なるので、この章では説明していません。

選択ステートメント には、以下の 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。
- 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。
- 対話式に発行する。

最初の 2 つの方法は、それぞれ*選択ステートメント* の*静的* 呼び出しおよび*動的* 呼び出しと呼ばれます。

SQL ステートメントの各呼び出し方式について、以下で詳しく説明します。各方式ごとに、実行のメカニズム、ホスト変数との対話、および実行が正常に行われたかどうかのテストについて説明しています。

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実行依頼されるソース・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* と呼ばれます。組み込むステートメントは、そのプログラムの中でホスト言語のステートメントが可能な場所であれば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 と 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、それと同じ場所にホスト言語のステートメントを指定した場合にそれが実行される時点と同じ時点に実行されます。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、次の 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。その他の参照の取り扱いについては、各ステートメントごとに個々に説明しています。

実行可能なすべての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ができます。

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 DB2 ユニバーサル・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 プロシージャーで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 プロシージャーに組み込まれるステートメントは、1158ページの『SQL プロシージャー・ステートメント』で指定されています。アプリケーションに組み込まれているステートメントとは異なり、 SQL ステートメントの前になんらかのキーワードを付ける必要はありません。 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る場合に *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)。このようにして構成されたステートメント (SELECT ステートメントを除く) は、組み込みステートメントの PREPARE によって準備され、組み込みステートメントの EXECUTE によって実行することができます。あるいは、組み込みステートメントの EXECUTE IMMEDIATE を使用して、 1 つのステップでステートメントを準備し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・マーカーの規則については、1042ページの『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。 (この置き換えの規則については、975ページの『EXECUTE』 を参照してください。) 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 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。詳細については、490ページの『SQL 戻りコード』 を参照してください。

選択ステートメントの静的呼び出し

*選択*ステートメント は、実行不能ステートメントである DECLARE CURSOR 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、組み込みステートメント OPEN 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、*選択*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 実行時点での変数の値によって置き換えられます。

選択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、組み込みステートメント PREPARE によって実行の準備が行われ、実行不能ステートメント DECLARE CURSOR によって参照することができます。このようなステートメントは、組み込みステートメント OPEN 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、*選択*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・マーカーの規則については、1042ページの『PREPARE』 を参照し

てください。) 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。 (この置き換えの規則については、1036ページの『OPEN』 を参照してください。)

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。

対話式に発行されるステートメントは、アプリケーション・プログラムの文脈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。

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できるようにするには、 2 つの方法があります。

- SQLCA と呼ばれる構造を組み込む。 REXX では、自動的に SQLCA が用意されます。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。

  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。
- 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 宣言セクションに変数 SQLCODE または SQLSTATE 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、プログラムの別の場所で変数 SQLCODE 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。

場合によっては、戻りコードに関連して、エラー条件だけでなく警告条件について説明されることがあります。警告 SQLCODE は正の値で、警告 SQLSTATE の最初の 2 文字は '01' になります。

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。詳細については、1211ページの『付録 B. SQL 連絡 (SQLCA)』を参照してください。
- 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。製品固有の意味については、*メッセージ解説書* を参照してください。

SQLSTATE

各 SQL ステートメントの実行後には、 SQLSTATE もデータベース・マネージャーによって設定されます。したがって、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、アプリケーション・プログラムに共通エラー条件に関する共通コードを示します。また、SQLSTATE は、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るように設計されています。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。 SQLSTATE の可能な値すべてのリストについては、*メッセージ解説書* を参照してください。

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
         WHEREDEPTNO = 'E21'     -- systems support dept code
         AND PRSTAFF > 1;
```

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。

- すべての区分 (またはノード)、あるいは 1 つの区分のバッファー・プールのサイズを変更する
- 拡張記憶域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ノード・グループに追加する

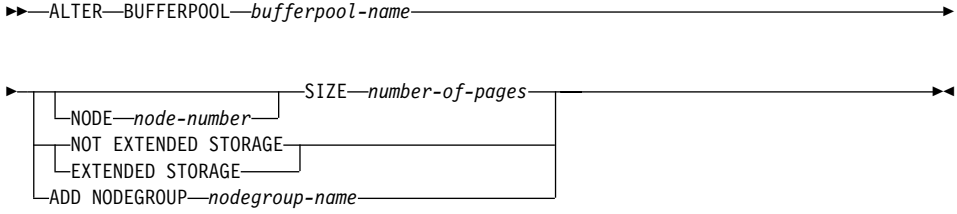
呼び出し

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。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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 識別子です (通常識別子または区切り識別子)。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。

NODE *node-number*

そのバッファー・プールのサイズを変更する区分を指定します。区分は、そのバッファー・プールのノード・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 (CREATE バッファー・プール・ステートメントの *except-on-nodes-clause* でサイズが指定されていない) すべての区分で、バッファー・プールのサイズが変更されます。

SIZE *number-of-pages*
ページ数で指定するバッファー・プールのサイズ。[56]

EXTENDED STORAGE
拡張記憶域構成がオンになっている場合、[57] このバッファー・プールの外に移送されたページは、拡張記憶域にキャッシュされます。

NOT EXTENDED STORAGE
拡張記憶域構成がオンになっていても、このバッファー・プールの外に移送されるページは、拡張記憶域にキャッシュされません。

ADD NODEGROUP *nodegroup-name*
バッファー・プール定義が適用されるノード・グループのリストに、このノード・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ノードグループにある区分については、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、この区分にバッファー・プールが作成されます。 *nodegroup-name* 内の表スペースによって、このバッファー・プールを指定できます。ノード・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)。

注

- 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は存在し、その間バッファー・プールには何の影響もありません。 新しいノード・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。
- 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。

---

56. サイズに (-1) の値を指定することができます。この値は、バッファー・プール・サイズを BUFFPAGE データベース構成パラメーターから取ることを指定します。

57. 拡張記憶域構成は、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。詳細については、*管理の手引き* を参照してください。

ALTER NICKNAME

ALTER NICKNAME ステートメントは、次のようにして、連合データベースにおけるデータ・ソース表または視点の表示を変更します。

- 表または視点の列のローカル名を変更する
- それらの列のローカル・データ・タイプを変更する
- それらの列のオプションを追加、変更、または削除する

呼び出し

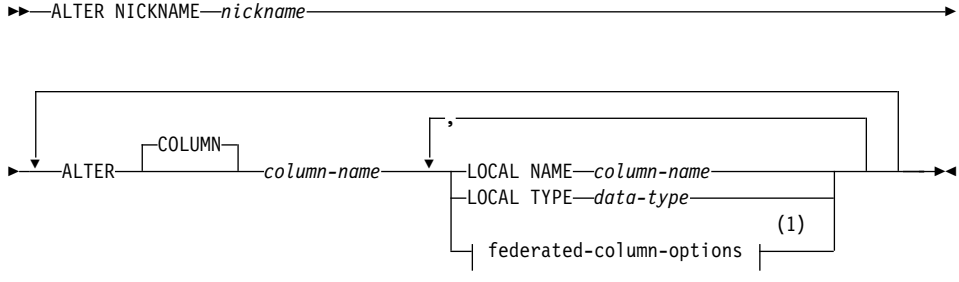
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

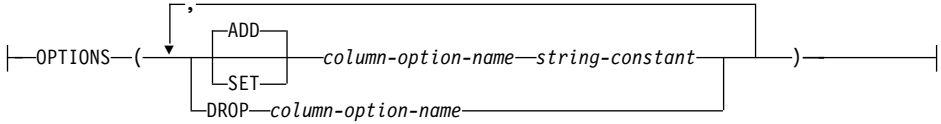
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- 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)。
- そのニックネームのカタログ視点の DEFINER 列に記録されているそのニックネームの定義者

構文



連合列オプション (federated-column-options):



注:

1 LOCAL NAME パラメーターか LOCAL TYPE パラメーターへ、 あるいはその両方へ連合列オプション (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。

説明

nickname
COLUMN キーワードの後に指定した列を含む、データ・ソース表または視点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、データ・ソースにある表または視点の列のための、連合サーバーの現在の名前です。 *column-name* は、*nickname* に示されているデータ・ソース表または視点の、既存の列を指定するものでなければなりません。

LOCAL NAME *column-name*
連合サーバーが、 ALTER COLUMN *column-name* パラメーターで指定した列を参照するときに使用する、新しい名前です。 この新しい名前は有効な DB2 識別子である必要があります。

LOCAL TYPE *data-type*
指定した列のデータ・タイプを、現在マップされているもの以外のローカル・データ・タイプにマップします。新しいタイプは *data-type* に示されます。

data-type は、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、ラージ・オブジェクト (LOB) データ・タイプ、またはユーザー定義タイプにはできません。

OPTIONS
COLUMN キーワードの後に指定した列のために、どの列オプションを使用、リセット、または削除するか指定します。 *column-option-name* とその設定の詳細は、 1353ページの『列オプション』を参照してください。

ADD
列オプションを使用可能にします。

SET
列オプションの設定を変更します。

column-option-name
使用可能にする、あるいはリセット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*
列オプションを除去します。

規則

- 特定のニックネームに対して視点を作成した場合、そのニックネームで参照する表または視点に含まれる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 42601)。しかし、それぞれの列の列オプションを使用可能にする、リセットする、あるいは除去するときには、このステートメントを使えます。

注

- ALTER NICKNAME を使用して、ニックネームが参照する表または視点に含まれる列のローカル名を変更する場合、その列の照会では、その名前を新しい名前として参照する必要があります。
- 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、あるいは除去する場合、使用中の他の列オプションには影響はありません。
- 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、その列のために集められたすべての統計 (HIGH2KEY、\ LOW2KEY、など) を無効にします。
- ALTER NICKNAME ステートメントで参照しているニックネームが、ある作業単位 (UOW) の SELECT ステートメントによって参照されている場合、そのステートメントを同じ UOW (作業単位) で処理することはできません (SQLSTATE 55007)。

例

例 1: ニックネーム NICK1 は、T1 という DB2 ユニバーサル・データベース (AS/400 版) 表を参照します。同様に、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 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 ユニバーサル・データベース (OS/390 版) 表を参照します。同様に、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' )
```

ALTER NODEGROUP

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

- ノード・グループに 1 つまたは複数の区分またはノードを追加する
- ノード・グループから 1 つまたは複数の区分を除去する

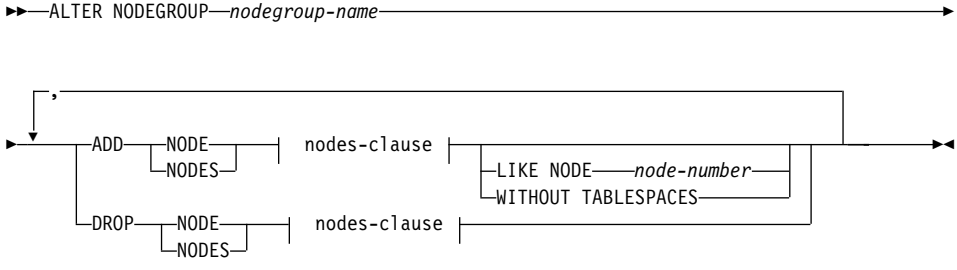
呼び出し

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。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

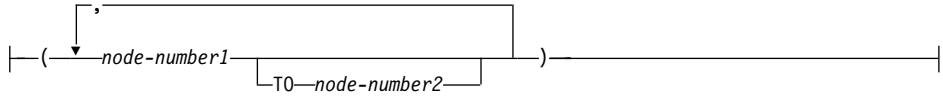
許可

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

構文



nodes-clause:



説明

nodegroup-name

ノード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 識別子です (通常識別子または区切り識別子)。ノード・

グループは、カタログで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 NODE
ノード・グループに追加する特定の 1 つまたは複数の区分を指定します。 NODES は NODE の同義語です。指定する区分は、ノード・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 NODE
ノード・グループから除去する特定の 1 つまたは複数の区分を指定します。 NODES は NODE の同義語です。指定する区分は、ノード・グループにすでに定義されている必要があります (SQLSTATE 42729)。

nodes-clause
追加または除去する 1 つまたは複数の区分を指定します。

node-number1
特定の区分番号を指定します。

TO *node-number2*
区分番号の範囲を指定します。 *node-number2* の値は、 *node-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 NODE *node-number*
ノード・グループの既存の表スペースのコンテナーが、指定した *node-number* のコンテナーと同じであることを指定します。指定する区分は、このステートメントの前にノード・グループに存在しており、同じステートメントの DROP NODE 文節に含まれていない区分である必要があります。

WITHOUT TABLESPACES
新たに追加される区分に、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 NODE 文節を用いた ALTER TABLESPACE を使用して、このノード・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、そのノード・グループに対して表スペースが定義されるたびに、新たに追加される区分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区分またはノードは、 `db2nodes.cfg` ファイルに定義されていなければなりません (SQLSTATE 42729)。このファイルについての情報は、 67ページの『複数の区分にわたるデータの区分化』を参照してください。

- ON NODES 文節にリストされる *node-number* は、それぞれ固有な区分に対する番号でなければなりません (SQLSTATE 42728)。
- 有効な区分番号は、0 ～ 999 (0 と 999 を含む) です (SQLSTATE 42729)。
- 1 つの区分を ADD と DROP の両方の文節に指定することはできません (SQLSTATE 42728)。
- ノード・グループには少なくとも 1 つの区分が残っている必要があります。最後の区分をノード・グループから除去することはできません (SQLSTATE 428C0)。
- 区分を追加する際に、 LIKE NODE 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、ノード・グループの既存の区分の最も小さい区分番号 (ここでは *2* とします) が使用され、 LIKE NODE *2* が指定された場合と同様の処理が行われます。既存の区分をデフォルト値として使用する場合、区分ではノード・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.NODEGROUPDEF の列 IN_USE が 'T' でない)。

注

- 区分またはノードがノード・グループに追加されると、その区分に対するカタログ項目が作成されます (SYSCAT.NODEGROUPDEF を参照)。以下のいずれかの場合には、区分化マップが直ちに変更され、新しい区分が、その区分が区分化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。
  - ノード・グループに表スペースが定義されていない、または
  - ノード・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、区分化マップは変更されず、標識 (IN_USE) はその区分が区分化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。
  - ノード・グループの表スペースに表が存在する、または
  - ノード・グループに表スペースが存在し、WITHOUT TABLESPACES 文節が指定された

  区分化マップを変更するには、REDISTRIBUTE NODEGROUP コマンドを使用する必要があります。このコマンドは、任意のデータを再配布し、区分化マップを変更し、標識を変更します。 WITHOUT TABLESPACES 文節が指定された場合は、データを再配布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。

- 区分がノード・グループから除去されると、その区分のカタログ項目 (SYSCAT.NODEGROUPDEF を参照) が更新されます。ノード・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、区分化マップが直ちに変更され、除去された区分を除外し、ノード・グループのその区分に関する項目が除去されます。表が存在する場合は、区分化マップは変更されず、標識 (IN_USE) はその区分が除去を待機していることを示すように設定されます。 REDISTRIBUTE NODEGROUP コマンドは、データを再配布し、ノード・グループからその区分に関する項目を除去する場合に、使用しなければなりません。

例

0、1、2、5、7、および 8 という区分を持つ、6 つの区分のデータベースがあると想定します。区分番号 3 と 6 の 2 つの区分がシステムに追加されます。

- MAXGROUP という名前のノード・グループに、両方の区分、すなわちノード 3 と 6 を追加し、区分 2 と同種の表スペース・コンテナーを持ちたいと想定します。必要なステートメントは以下のようになります。

```
ALTER  NODEGROUP MAXGROUP
ADD NODES (3,6) LIKE NODE 2
```

- 区分 1 を除去し、区分 6 をノード・グループ MEDGROUP に追加するとします。 ALTER TABLESPACE を使用して、区分 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 NODEGROUP MEDGROUP
 ADD NODE(6) WITHOUT TABLESPACES
 DROP NODE(1)
```

ALTER SERVER

ALTER SERVER ステートメント[58] は、以下の目的で使用します。

- 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、連合データベースへ何回か接続する間、継続します)。

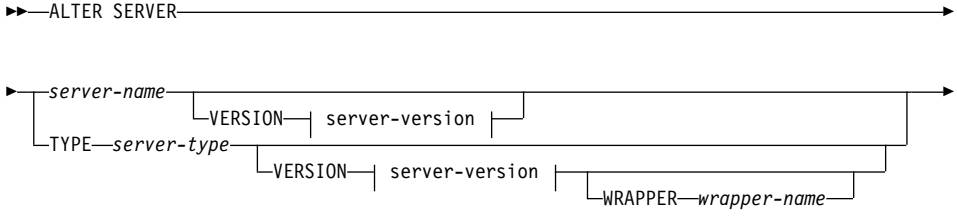
呼び出し

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、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

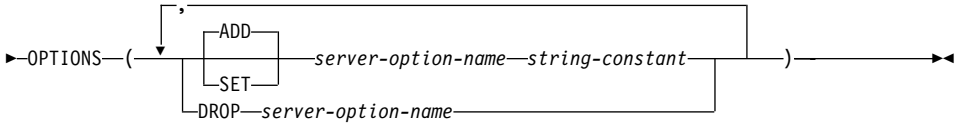
許可

このステートメントの許可 ID は、連合データベースに対する 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。

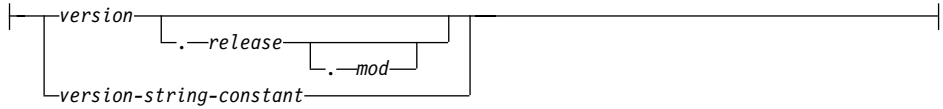
構文



58. このステートメントでは、 SERVER という語と、*server-* で開始するパラメーター名は、連合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連合サーバー、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。連合システムについての詳細は、46ページの『DB2 連合システム』を参照してください。 DRDA アプリケーション・サーバーについての詳細は、 33ページの『分散リレーショナル・データベース』を参照してください。



server-version:



説明

server-name
要求された変更を適用する対象のデータ・ソースについて、連合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。 *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')。

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。このサーバー・タイプは、カタログにリストされているものでなければなりません。

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、あるいは除去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、連合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・ソースのために、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のために、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、または除去するかを指定します。 *server-option-name* とその設定の詳細は、 1355ページの『サーバー・オプション』を参照してください。

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・オプションを除去します。

注

- このステートメントでは、 DBNAME および NODE サーバー・オプションをサポートしていません (SQLSTATE 428EE)。
- 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、あるいは除去する場合、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。

  - ステートメントが 1 つのデータ・ソースを参照していて、UOW にはすでに、このデータ・ソース内の表または視点のニックネームを参照する SELECT ステートメントが含まれる場合 (SQLSTATE 55007)。
  - 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視点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 55007)。
- 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、そ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 'N' に設定したとします。このように構成するならば、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。

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、これらのデータ・ソースは、アップグレードした CPU (ローカル CPU の半分の速度) 上での実行をすでに開始しているとします。最適化プログラム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: SUNDIAL という DB2 ユニバーサル・データベース (AS/400 版) バージョン 3.0 データ・ソースが、バージョン 3.1 にアップグレードされていることを指示します。

```
ALTER SERVER SUNDIAL
   VERSION 3.1
```

ALTER TABLE

ALTER TABLE ステートメントは、次のことを行うことにより、既存の表を変更します。

- 1 つまたは複数の列を表に追加する
- 基本キーの追加、あるいは除去を行う
- 1 つまたは複数の固有制約、または参照制約の追加、あるいは除去を行う
- 1 つまたは複数の検査制約定義の追加、あるいは除去を行う
- VARCHAR 列の長さを変更する
- 参照タイプ列を変更して、効力範囲を追加する
- 生成された列の生成式を変更する
- 区分化キーの追加、あるいは除去を行う
- 表属性 (データ・キャプチャー・オプション、pctfree、ロック・サイズ、追加モードなど) を変更する
- 表を記録されていない初期状態 (NOT LOGGED INITIALLY) に設定する

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、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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。

- 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 / 消去の場合、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
- その表に対する CONTROL 特権

- SYSADM または DBADM 権限

表 T の基本キーまたは固有制約を除去するには、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
- その表に対する CONTROL 特権
- 表のスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

(fullselect を使用して) 表を要約表に変更するには、このステートメントの許可 ID 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

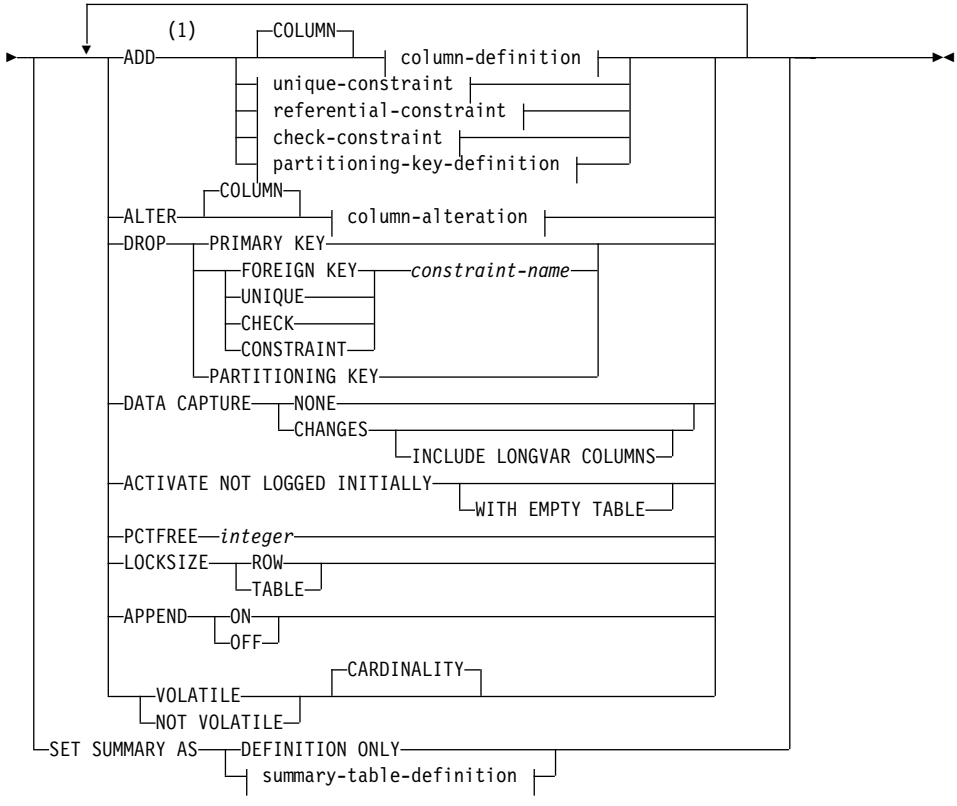
なおかつ、fullselect で識別された個々の表または視点に対する以下の特権が少なくとも 1 つ含まれている必要があります。

- その表または視点に対する SELECT および ALTER 特権
- その表または視点に対する CONTROL 特権
- その表または視点に対する SELECT 特権と、その表または視点のスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

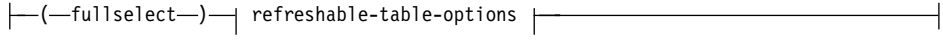
表を変更して要約表でなくなるようにするには、ステートメントの許可 ID が持っている特権に、この要約表を定義するのに使用する全選択で識別される各表または視点で、少なくとも以下の 1 つが含まれている必要があります。

- その表または視点に対する ALTER 特権
- その表または視点に対する CONTROL 特権
- その表または視点のスキーマに対する ALTERIN 特権
- SYSADM または DBADM 権限

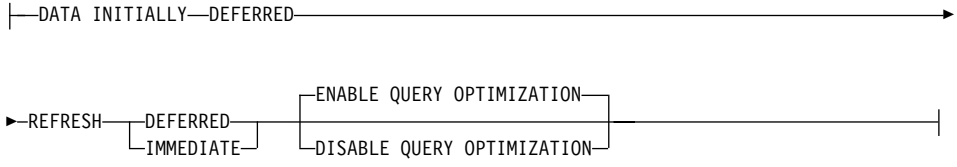
構文



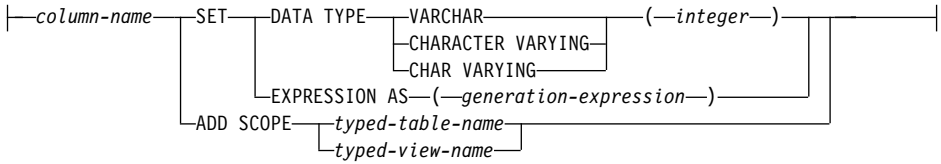
summary-table-definition:



refreshable-table-options:



column-alteration:

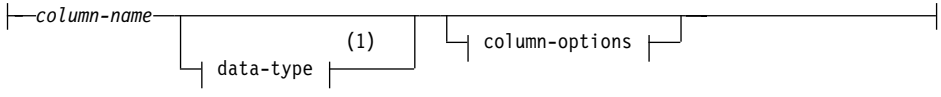


注:

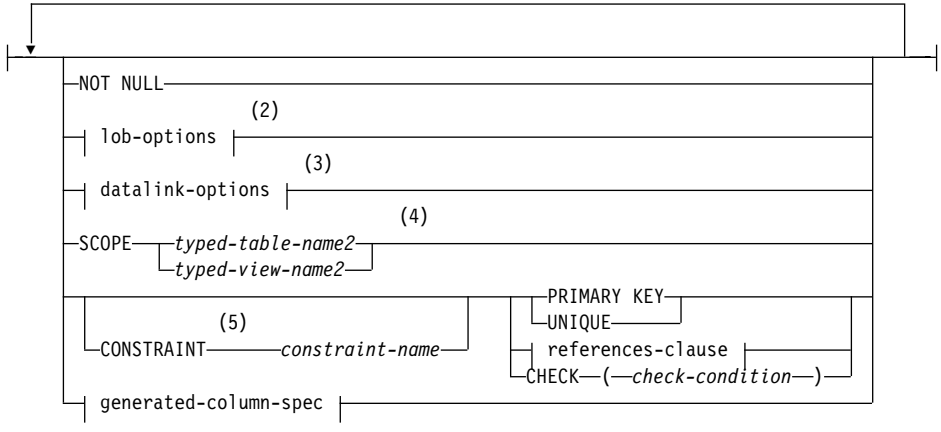
1 バージョン 1 との互換性を保つため、以下についての ADD キーワードは任意選択です。

- 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

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。

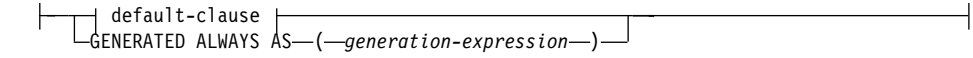
2 lob-options は、 ラージ・オブジェクト・タイプ (BLOB、CLOB、および DBCLOB) と、 ラージ・オブジェクト・タイプに基づく特殊タイプに対してのみ適用されます。

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。

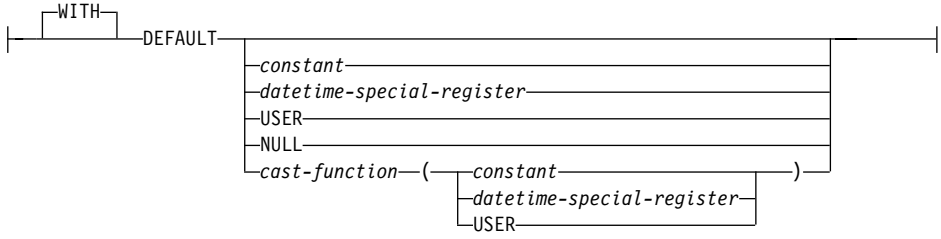
4 SCOPE 文節は REF タイプに対してのみ適用されます。

5 バージョン 1 との互換性を保つため、 references-clause (REFERENCES 文節) を定義する *column-definition* (列定義) では、 CONSTRAINT キーワードを省略することができます。

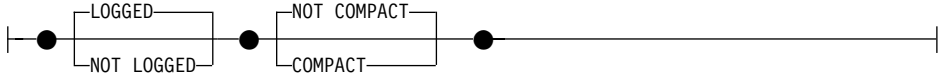
generated-column-spec:



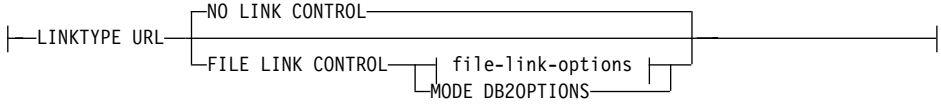
default-clause:



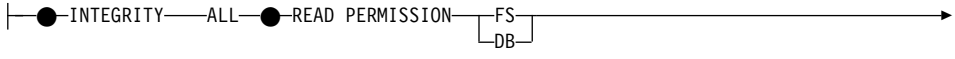
lob-options:

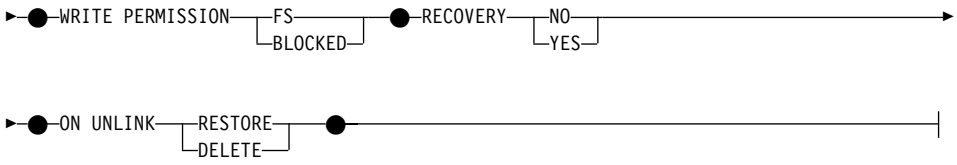


datalink-options:

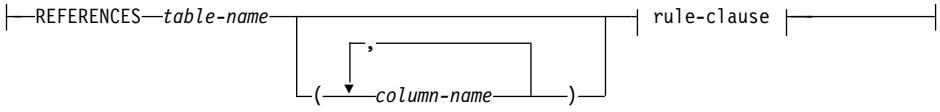


file-link-options:

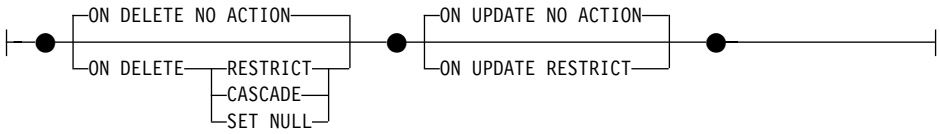




references-clause:

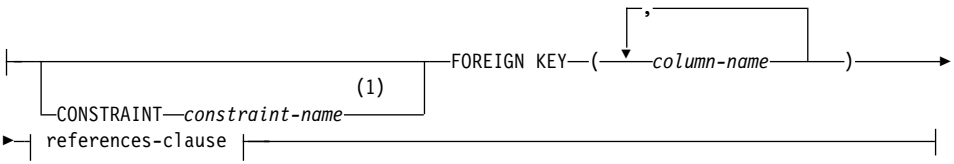


rule-clause:

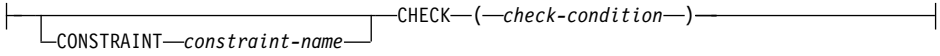


unique-constraint:

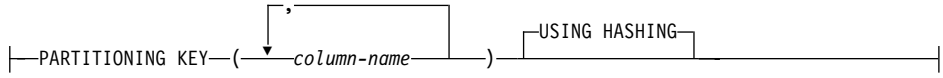
referential-constraint:



check-constraint:



partitioning-key-definition:



注:

1 バージョン 1 との互換性を保つため、 *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。

説明

table-name
変更する表を指定します。これは、カタログに記述されている表でなければならず、視点、またはカタログ表を対象にすることはできません。
table-name で要約表を指定している場合、変更は、定義のために要約表を設定することのみに限られ、まだ記録していない部分を活動化し、pctfree、locksize、append、または volatile を変更することだけができます。
table-name をニックネーム (SQLSTATE 42809) または宣言された一時表 (SQLSTATE 42995) にすることはできません。

SET SUMMARY AS
要約表のプロパティーの変更を許可します。

DEFINITION ONLY
表が要約表とみなされなくなるように、要約表を変更します。
table-name で指定される表は、複製ではない要約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。

summary-table-definition
照会の最適化で使用するために、正規表を要約表に変更します。
table-name により指定される表には、以下の条件が求められます。

- 要約表として以前に定義されていてはなりません。
- タイプ付き表であってはなりません。
- 何らかの定数、固有索引、またはトリガーが定義されていてはなりません。
- 他の要約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* の結果列と比べて、

- 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。

(SQLSTATE 428EW)。要約表に *fullselect* を指定することについての詳細は、 771ページの『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
要約表を変更するためのリフレッシュ可能オプションをリストします。

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しなければなりません。

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最新表示できます。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要約表には、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、要約表にカスケードされます。その場合、表の内容は、どの時刻指定でも、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要約表には、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
要約表を照会の最適化に使用できるようにします。

DISABLE QUERY OPTIMIZATION
要約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。

ADD *column-definition*
列を表に追加します。タイプ付き表を使用することはできません (SQLSTATE 428DH)。表に既存の行がある場合には、新たに追加した列のすべての値がその表のデフォルト値になります。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。 *n* の値は 499 を超えることはできません。

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、 1209 ページの表34 で指定した最大レコード・サイズを超えてはなりません。詳細については、816ページの『注』 を参照してください。

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
771ページの『CREATE TABLE』の項に示されるデータ・タイプのいずれかです。

NOT NULL
列にヌル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。 771ページの『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。 771ページの『CREATE TABLE』の *datalink-options* を参照してください。

SCOPE
参照タイプ列の効力範囲を指定します。

typed-table-name2
タイプ付き表の名前。 *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
タイプ付き視点の名前。 *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識別子の中で固有な 18 文字の識別子[59] がシステムによって生成されます。

PRIMARY KEY 制約または UNIQUE 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。固有制約に関連した索引名の詳細については、528ページの『注』 を参照してください。

PRIMARY KEY
これは、1 つの列からなる基本キーを定義する簡単な方法として提供されています。したがって、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。列にヌル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 つの列からなる固有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提供さ

59. 識別子は、″SQL″ と、タイム・スタンプに基づいて生成された 15 個の数字からなります。

れています。したがって、REFERENCES 文節が列 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 文節が指定された場合と同じになります。

771ページの『CREATE TABLE』で *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検査制約を定義する簡単な方法として提供されています。 771ページの『CREATE TABLE』 で *check-condition* を参照してください。

generate-column-spec
column-generation の詳細については、771ページの『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
任意選択キーワード。

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19 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 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 19. 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
固定長漢字ストリング	2 バイトのブランク
可変長漢字ストリング	長さ 0 のストリング

表 19. デフォルト値 (値が指定されない場合) (続き)

データ・タイプ	デフォルト値
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。
2 進ストリング (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ヌル値が使用されます。

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 などの先行文字も含まれます。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 または UPDATE 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

INSERT または UPDATE の実行時における USER 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。 USER を指定した場合、その列のデータ・タイプは、 USER 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、値は ALTER TABLE ステートメントの許可 ID になります。

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、必ず関数を使用しなければなりません。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
USER 特殊レジスターを指定します。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。

指定した値が無効な場合、エラー (SQLSTATE 42894) が発生します。

ADD *unique-constraint*
固有制約または基本キー制約を定義します。基本キー制約または固有制約を、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*
基本キー制約、または固有制約の名前を指定します。詳細については、771ページの『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固有キーを定義します。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、保管長の合計は 1024 を超えてはなりません (列の保管長については、 820ページの『Byte Counts』 を参照)。それぞれの列の長さは、どれも 255 バイトを超えてはなりません。固有キーの一部として、LOB、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、これらのタイプの特殊タイプ、あるいは構造タイプ列を使用することはできません (列の長さ属性が 255 バイトの限界以内に収まる場合でも) (SQLSTATE 42962)。固有キーにある一連の列は、基本キーまたは他の固有キーの一

連の列と同じにすることはできません (SQLSTATE 01543)。 [60] 指定する列の集合に存在する値は、固有である必要があります (SQLSTATE 23515)。

既存の索引が固有キー定義に一致するかどうかを判別する検査が行われます (索引の INCLUDE 列は無視されます)。列の順序や方向 (ASC/DESC) の指定に関係なく、列の同じ集合を指定していると、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、索引が固有でない場合は固有索引に変更されます (固有性を確実にした後)。その表に一致する索引が複数ある場合、既存の固有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 で説明したように、その列に対して固有索引が自動的に作成されます。固有制約に関連した索引名の詳細については、528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基本キーを定義します。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 (保管される長さの詳細は、 820ページの『Byte Counts』を参照)。それぞれの列の長さは、どれも 255 バイトを超えてはなりません。表には基本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。基本キーの一部として、LOB、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、これらのタイプの特殊タイプ、あるいは構造タイプ列を使用することはできません (列の長さ属性が 255 バイトの限界以内に収まる場合でも) (SQLSTATE 42962)。基本キーの一連の列は、固有キーの一連の列と同じであってはなりません (SQLSTATE 01543)。 [60]一連の指定された列にある既存の値は、固有でなければなりません (SQLSTATE 23515)。

既存の索引が基本キー定義に一致するかどうかを判別する検査が行われます (索引の INCLUDE 列は無視されます)。列の順序や方向 (ASC/DESC) の指定に関係なく、列の同じ集合を指定していると、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、索引が固有でない場合は固有索引に変更されます

60. LANGLEVEL が SQL92E または MIA の場合には、エラーが戻されます (SQLSTATE 42891)。

(固有性を確実にした後)。その表に一致する索引が複数ある場合、既存の固有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 で説明したように、その列に対して固有索引が自動的に作成されます。固有制約に関連した索引名の詳細については、528ページの『注』 を参照してください。

1 つの表には、基本キーを 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。 771ページの『CREATE TABLE』 で *referential-constraint* を参照してください。

ADD *check-constraint*
検査制約を定義します。 771ページの『CREATE TABLE』 の *check-constraint* (検査制約) の部分を参照してください。

ADD *partitioning-key-definition*
区分化キーを定義します。表については、単一区分のノード・グループにある表スペースで定義する必要があり、すでに区分化キーを持っていてはなりません。区分化キーが表にすでに存在している場合には、新しい区分化キーを追加する前に既存のキーを除去しなければなりません。

区分化キーを、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、区分化キーを定義します。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、区分化キーの一部として列を使用することはできません。

USING HASHING
データ配分の区分化方式として、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区分化方式です。

ALTER *column-alteration*
列の特性を変更します。

column-name
表で変更する列の名前です。 *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。

SET DATA TYPE 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、32672 です。タイプ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、 1209ページの表34 で指定した最大レコード・サイズを超えてはなりません (SQLSTATE 54010)。詳細については、816ページの『注』 を参照してください。固有制約または索引で列を使用する場合、新しい長さは 255 バイトを超えてはならず、固有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) (保管長については、820ページの『Byte Counts』 を参照)。

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検査保留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)。 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しなければなりません。 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)。

ADD SCOPE
効力範囲が未定義である既存の参照タイプ列に、効力範囲を追加します (SQLSTATE 428DK)。変更する表がタイプ付き表である場合、列をスーパー表から継承することはできません (SQLSTATE 428DJ)。例については、542ページの『ALTER TYPE (構造化)』を参照してください。

typed-table-name
タイプ付き表の名前。 *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
タイプ付き視点の名前。 *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。

DROP PRIMARY KEY
基本キーの定義、およびその基本キーに従属するすべての参照制約を除去します。表には基本キーがなければなりません。

DROP FOREIGN KEY *constraint-name*
制約名が *constraint-name* の参照制約を除去します。 *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。参照制約の除去により起こることについては、528ページの『注』を参照してください。

DROP UNIQUE *constraint-name*
固有制約 *constraint-name* の定義、およびこの固有制約に従属するすべての参照制約を除去します。 *constraint-name* は、既存の UNIQUE 制約を指定していなければなりません。固有制約の除去により起こることについては、528ページの『注』を参照してください。

DROP CONSTRAINT *constraint-name*
制約名が *constraint-name* の制約を除去します。 *constraint-name* は、表に定義されている既存の検査制約、参照制約、基本キー、または固有制約のいずれかを指定していなければなりません。制約の除去により起こることについては、528ページの『注』を参照してください。

DROP CHECK *constraint-name*
制約名が *constraint-name* の検査制約を除去します。 *constraint-name* は、表に定義されている既存の検査制約を指定していなければなりません。

DROP PARTITIONING KEY
区分化キーを除去します。表には区分化キーがある必要があり、表は単一区分のノード・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。

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。

表がタイプ付き表である場合、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、収集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取り込む場合に必須です。

カタログ区分以外の区分にデータが入れられるように表が定義されている場合 (複数区分のノード・グループ、またはカタログ区分以外の区分を持つノード・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)。

複製の使用法の詳細については、 *管理の手引き* および *レプリケーションの手引きおよび解説書* を参照してください。

INCLUDE LONGVAR COLUMNS

データ複製ユーティリティーが、 LONG VARCHAR または LONG VARGRAPHIC 列に対する変更を取り込むようにします。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。NOT LOGGED INITIALLY 属性で作成された表は使用できません (SQLSTATE 429AA)。

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、カタログがロックしてしまいます。 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。

表がタイプ付き表である場合、このオプションがサポートされるのは、タイプ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、 771ページの『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: 作業単位内で NOT LOGGED INITIALLY 属性を活動化したことにより表が変更された場合、保管点要求へのロールバックは、作業単位要求へのロールバックに変換されます (SQLSTATE 40506)。 NOT LOGGED INITIALLY 属性が活動状態にある作業単位における操作でエラーが発生すると、その作業単位全体がロールバックされます (SQLSTATE 40506)。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**、除去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、最小限に抑えられます。

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、修復したい表に定義された 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*
ロードまたは再編成時に、各ページで空きスペースとして残しておくスペースの割合を指定します。 *integer* の値は 0 ～ 99 です。各ページの最初の行は、制約なしに追加されます。行をさらに追加する場合、各ページに少なくとも *integer* パーセントを空きスペースとして残します。 PCTFREE 値は、LOAD または REORGANIZE TABLE ユーティリティーでのみ有効です。表がタイプ付き表である場合、このオプションがサポートされるのは、タイプ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。表がタイプ付き表である場合、このオプションがサポートされるのは、タイプ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、完全な表に対してはロックがすべて保留となるため、並行性も限定されます。

ロックの詳細については、*管理の手引き* を参照してください。

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・ページの空きスペースが使用可能な場所に追加するかを指定します。表がタイプ付き表である場合、このオプションがサポートされるのは、タイプ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、各ページの空き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、使用可能な空きスペース情報が不正確となるため、データ挿入時のローパフォーマンスにつながるからです。

VOLATILE
これを指定することにより、最適化プログラムに対し、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、実行時に変化し得ることを知らせます。 *table-name* にアクセスするため、最適化プログラムは、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使えることが条件になります。表がタイプ付き表である場合、このオプションがサポートされるのは、タイプ付き表階層のルート表でだけです (SQLSTATE 428DR)。

NOT VOLATILE
これを指定することにより、最適化プログラムに対して、 *table-name* のカーディナリティーが揮発ではないことを知らせます。この表に対するアクセス・プランは、既存の統計と、所定の最適化レベルに基づいて続けられます。

CARDINALITY
揮発するのが表内の行数であり、表そのものではないことを示す任意選択キーワード。

規則

- 表の区分化キー列は更新できません (SQLSTATE 42997)。
- 表に対して定義された固有キー制約または基本キー制約は、区分化キー (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)。
- 区分化キーのヌル可能な列は、関係が ON DELETE SET NULL を指定して定義されている場合は、外部キーの列として組み込むことはできません (SQLSTATE 42997)。
- 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)。
- 表に要約表があり、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)。
- 生成された列を追加する前に、SET INTEGRITY ステートメントを使用して、表を検査保留状態に設定しなければなりません (SQLSTATE 55019)。

注

- 表を要約表に変更すると、この表は検査保留状態になります。表が REFRESH IMMEDIATE として定義されている場合、この表は検査保留状態から出されなければ、全選択で参照されている表で INSERT、DELETE、または UPDATE コマンドを発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 を使用すると、表の検査保留状態を解除し、全選択に基づいて表内のデータを完全に最新表示できます。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、表の検査保留状態を解除できます。
- 表を変更して REFRESH IMMEDIATE 要約表にすると、全選択により参照される表で INSERT、DELETE、または UPDATE を使用したパッケージはどれも無効になります。

- 表を要約表から正規表 (DEFINITION ONLY) に変更すると、表に従属するパッケージはどれも無効になります。
- 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、表の既存の視点に自動的に追加されるわけではありません。
- 固有キー制約または基本キー制約に関して索引が自動的に作成された場合、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。したがって、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。
- 基本キー、固有キー、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。
  - 基本キーまたは固有キーが追加された場合、
    - パッケージ、索引、外部キー、または既存の固有キーには影響を与えません。[61]
  - 基本キーまたは固有キーが除去された場合、
    - 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、その索引は除去されます。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。

61. 基本キーまたは固有キーが、前のバージョンで作成された既存の固有索引を使用しており、固有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、関連した表の更新使用のパッケージは無効になります。

- 索引が制約に関して固有であるように変換されており、現在システムが索引を必要としていない場合、索引は非固有に戻されます。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固有索引だった場合、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が除去されます。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。

– 外部キーを追加または除去する場合、

- オブジェクト表に対して挿入の使用のあるパッケージは、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の使用のあるパッケージは、すべて無効になります。
- 親表に削除使用のある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。

- 表に列を追加すると、変更された表に対して挿入の使用のある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、変更された表でパッケージに DELETE を使用しようとしても無効になります。
- 検査保留状態 (1113ページの『SET INTEGRITY』 を参照) ではない既存の表に対して検査制約または参照制約を追加すると、その表の既存の行は、制約に対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。表が検査保留状態の場合は、検査制約または参照制約を追加しても、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、検査保留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する必要があります。
- 検査制約の追加または除去すると、対象の表に対する挿入使用、または制約に関係している少なくとも 1 つの列に対する更新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。
- 区分化キーを追加すると、区分化キーの少なくとも 1 つの列に対して更新使用を伴うパッケージは、すべて無効になります。
- 区分化キーの最初の列としてデフォルト値によって定義された基本キーは、基本キーの除去や異なる基本キーの追加によって影響を受けません。

- 列 1 を変更して長さを増やすと、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。
- 列を変更して長さを増やすと、表に従属する視点 (タイプ付き視点を除く) が再生成されます。視点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)。表に従属するタイプ付き視点は、作動不能としてマークされます。
- 長さを増やすために列を変更すると、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、変換変数および変換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除去されると、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)。
- 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。ロックの詳細については、*管理の手引き* を参照してください。
- 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えません (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が無効になります。
- 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・レコードを収集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、収集プログラムは失敗するかもしれません。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。

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。

Column Name	Data Type
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 表の基本キーです。 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 (つまりヌル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。ヌル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 を更新します。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 という名前の検査制約を追加します。

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

例 5: 表 EMPLOYEE を更新します。前に定義した制約 REVENUE を除去します。

```
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更新します。

```
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更新します。

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

例 8: EMPLOYEE 表を更新して、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、関数を使用しなければなりません。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: 以下の列が定義された CUSTOMERS という表があると想定します。

Column Name	Data Type
BRANCH_NO	SMALLINT
CUSTOMER_NO	DECIMAL(7)
CUSTOMER_NAME	VARCHAR(50)

この表では、基本キーは BRANCH_NO 列と CUSTOMER_NO 列からなります。表を区分化したいので、表に対して区分化キーを作成する必要があります。表は単一ノードのノード・グループにある表スペースに定義する必要があります。基本キーは、区分化列のスーパーセットである必要があり、基本キーの少なくとも 1 つの列が区分化キーとして使用されている必要があります。BRANCH_NO を区分化キーにしたいと想定します。以下のステートメントを用いてこれを行います。

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

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。

- DMS 表スペース (MANAGED BY DATABASE オプションによって作成) にコンテナーを追加する。
- DMS 表スペース (MANAGED BY DATABASE オプションによって作成) にあるコンテナーのサイズを大きくする。
- 現在コンテナーがない区分 (あるいはノード) にある SMS 表スペースに、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。

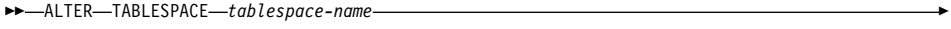
呼び出し

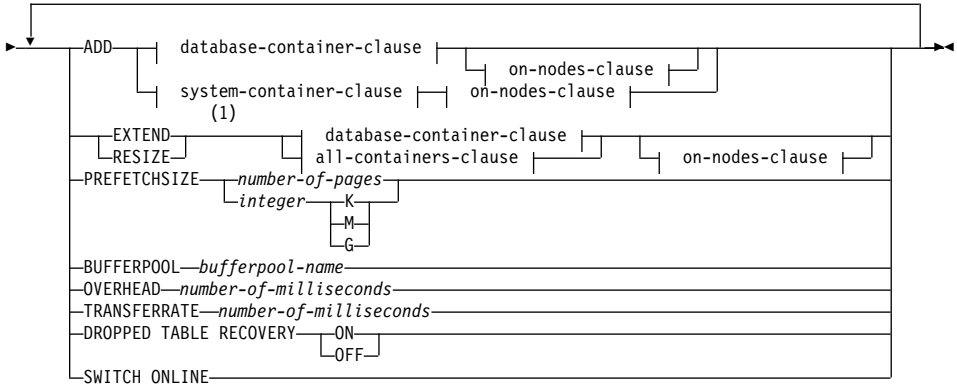
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

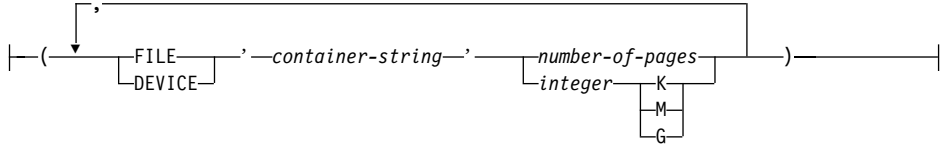
このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文

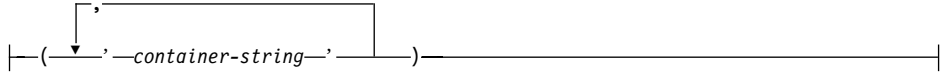




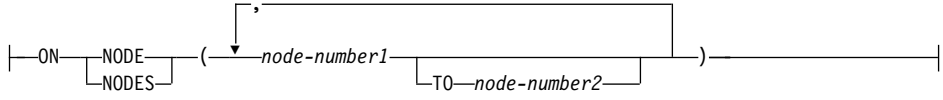
database-container-clause:



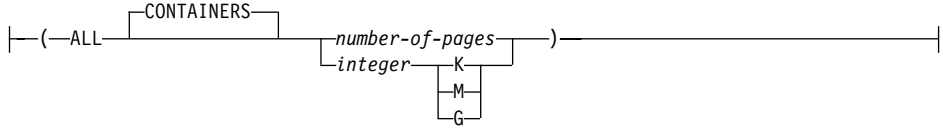
system-container-clause:



on-nodes-clause:



all-containers-clause:



注:

1 同じステートメントに ADD、EXTEND、および RESIZE 文節を指定することはできません。

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 識別子です (通常識別子または区切り識別子のいずれか)。

ADD
ADD は、新しいコンテナーを表スペースに追加することを指定します。

EXTEND
EXTEND は、既存のコンテナーのサイズを増やすように指示します。指定されるサイズは、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。

RESIZE
RESIZE は、既存のコンテナーのサイズが変更されることを指定します (コンテナー・サイズを小さくすることはできません)。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。

database-container-clause
DMS 表スペースに 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 DMS 表スペースを指定するものでなければなりません。 832 ページの *container-clause* の説明を参照してください。

system-container-clause
指定の区分あるいはノードにある SMS 表スペースに、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 SMS 表スペース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区分にコンテナーがあってはなりません (SQLSTATE 42921)。 831 ページの *system-containers* の説明を参照してください。

on-nodes-clause
追加されるコンテナーの区分を指定します。 833 ページの *on-nodes-clause* の説明を参照してください。

all-containers-clause
DMS 表スペースにあるコンテナーすべてを拡張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 DMS 表スペースを指定するものでなければなりません。

PREFETCHSIZE *number-of-pages*
データの事前取り出しの実行中に、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。 この事前取り出しサイズ値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・サイズで分割されたバイト数のフロアは、事前取り出しサイズのページ値の数を判別するために使用します。 事前取り出しでは、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。

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、この表スペースのノードグループを定義する必要があります (SQLSTATE 42735)。

OVERHEAD *number-of-milliseconds*
number-of-milliseconds (ミリ秒数) は、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*
number-of-milliseconds は、 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。

DROPPED TABLE RECOVERY
指定された表スペースから除去された表は、 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

- PREFETCHSIZE、OVERHEAD、および TRANSFERRATE の各パラメーターに最適な値を選択するための手引きと、バランスの再調整に関する情報が、*管理の手引き* に記載されています。
- 新しいコンテナーが追加されて、トランザクションがコミットされると、表スペースの内容はコンテナー間で自動的にバランスの再調整がなされます。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)。
- 同じ ALTER TABLESPACE ステートメントを使用して、表スペースの複数のコンテナー・サイズを変更しながら新しいコンテナーを追加することはできません (SQLSTATE 429BC)。複数のコンテナーのサイズを変更する際、 1 つのステートメントに EXTEND 節と RESIZE 節を同時に使用することはできません (SQLSTATE 429BC)。
- RESIZE を使用して、コンテナー・サイズを小さくすることはできません。コンテナーのサイズを小さくしようとすると、エラーが発生します (SQLSTATE 560B0)。
- 存在しないコンテナーについて拡張またはサイズ変更しようとすると、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)。

- コンテナーが拡張またはサイズ変更されてから、トランザクションがコミットされると、表スペースの内容はコンテナー間で自動的にバランスの再調整がなされます。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。
- 表スペースで複数のコンテナーを拡張する場合、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。これは、複数のコンテナーをサイズ変更する場合にも同じです。同じ表スペースで別個の ALTER TABLESPACE ステートメントを使用するものの、 1 つのトランザクションで複数のコンテナー・サイズを変更しようとすると、エラーが発生します (SQLSTATE 55041)。
- 区分データベースで、複数の区分が同じ物理ノードに存在する場合、このような区分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、それぞれの区分ごとに固有の *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、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、その間は続いて使用されます。

例

例 1: PAYROLL 表スペースに装置を追加します。

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

例 2: ACCOUNTING 表スペースの事前取り出し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、それから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 (3 つの異なる ALTER TABLESPACES を使用して、サイズを変更)。

```
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)
```

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

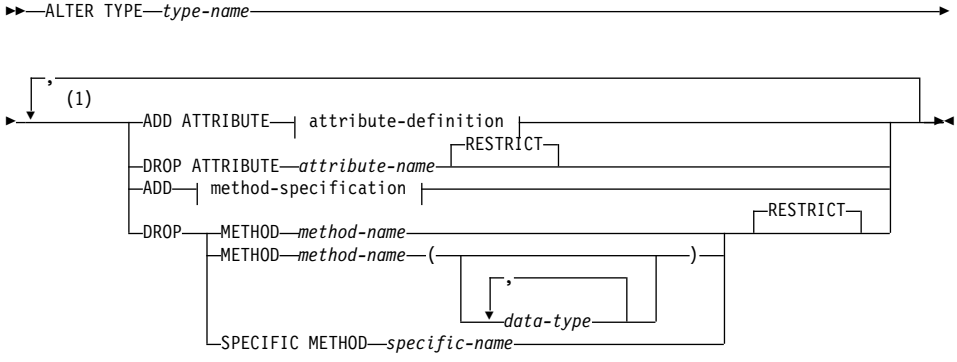
このステートメントは、アプリケーション・プログラムに組み込むか、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

このステートメントの許可 ID 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
- 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

構文



注:

1 属性とメソッドの両方が追加または削除される場合、 すべてのメソッド指定の前に、すべての属性指定が必要です。

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 特殊レジスターは、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 / バインド・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。

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

attribute-definition の詳細については、 859ページの『CREATE TYPE (構造化)』を参照してください。

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、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多くの名前は、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)。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。 771ページの『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 の属性で定義されたタイプは、タイプ付き表またはタイプ付き視点のデータ・タイプとしてのみ効果的に使用できます (SQLSTATE 01641)。

実行時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)。詳細は、99ページの『構造タイプ』を参照してください。

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、771ページの『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、771ページの『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。

DROP ATTRIBUTE

既存の構造タイプの属性を除去します。

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、視点、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、どの属性も除去できないという規則を課します。

ADD 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、859ページの『CREATE TYPE (構造化)』 を参照してください。

DROP METHOD

除去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除去してから、 ALTER TYPE DROP METHOD を使用してください。

指定するメソッドは、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 ステートメントで暗黙的に生成されたメソッド (mutators および observers など) は、除去できません (SQLSTATE 42917)。

メソッドを除去する方法としては、次のようにいくつかの方法があります。

METHOD *method-name*

特定のメソッドを指定します。名前 *method-name* およびサブジェクト・タイプ *type-name* 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。このように識別されるメソッドには、パラメーターがいくつあっても構いません。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・タイプの名前 *method-name* に複数のメソッドがある場合には、エラーが戻されます (SQLSTATE 42854)。

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

除去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。メソッド選択のアルゴリズムは使用されません。

method-name

特定のタイプを除去するメソッドの名前。この名前は、修飾子のない識別子でなければなりません。

(data-type,...)

メソッドが定義された際のメソッド指定の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・タイプの数とデータ・タイプを論理的に連結した値が、除去する特定のメソッド・インスタンスを識別するのに使用されます。

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べきことを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) のデータ・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。タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

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)。

SPECIFIC METHOD *specific-name*

メソッドの定義時に指定されたか、またはデフォルト値として与えられた特定の名前を使用して、除去するメソッドを指定します。 *specific-name* が修飾なしの名前である場合には、メソッドは、*type-name* で指定されるデータ・タイプのスキーマで暗黙的に修飾されます。 specific-name は、タイプ *type-name* のメソッドを識別しなければなりません。そうでない場合には、エラーになります (SQLSTATE 42704)。

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除去してから、 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除去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、 99ページの『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、859ページの『CREATE TYPE (構造化)』に説明されているとおり、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。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た属性は、除去できません (SQLSTATE 428DJ)。
  - DROP ATTRIBUTE は、除去された属性の mutator および observer メソッドを除去し、これらの除去されたメソッドの従属性を検査します。

注

- 属性を追加または除去することにより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が無効になります。
- 構造タイプから属性を追加または除去する場合:
  - 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。
  - 属性の追加または除去により影響を受けたタイプの 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、既存の変形機能を修正しなければなりません。

例

例 1: ALTER TYPE ステートメントを使用して、手動で参照してい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。

次の順序列で、タイプおよび表の作成ができます。

```
CREATE TYPE DEPT ...
CREATE TYPE EMP ... (REF(DEPT) というタイプの 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
```

次の順序列で、タイプおよび表の除去ができます。

```
DROP TABLE EMPLOYEE   (DEPARTMENT の MANAGER 列が効力範囲解除となる)
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)
```

このステートメントについては、731ページの『CREATE METHOD』 を参照してください。

ALTER USER MAPPING

ALTER USER MAPPING ステートメントは、指定した連合サーバーの許可 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。

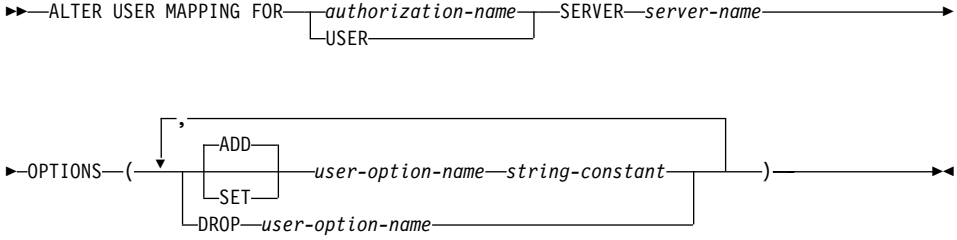
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

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、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連合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のために、使用可能にする、リセットする、または削除するユーザー・オプションを指定します。 *user-option-name* とその設定の詳細は、 1361ページの『ユーザー・オプション』を参照してください。

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・オプションを除去します。

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、あるいは除去する場合、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。
- データ・ソースの表または視点のニックネームを参照する SELECT ステートメントで、マッピングに組み込む予定のものが、すでに所定の作業単位 (UOW) に組み込まれている場合、その UOW ではユーザー・マッピングを変更することはできません。

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 は連合データベースを使用して、 DORADO という DB2 ユニバーサル・データベース (OS/390 版) 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、使用している連合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'  )
```

ALTER VIEW

ALTER VIEW ステートメントは、参照タイプ列を変更して効力範囲を追加することによって、既存の視点を変更します。

呼び出し

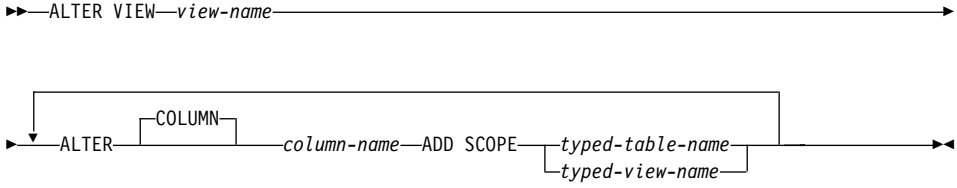
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

このステートメントの許可 ID 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
- 視点のスキーマに対する ALTERIN 特権
- 変更する視点の定義者
- 変更する視点に対する CONTROL 特権

構文



説明

view-name
変更する視点を指定します。 視点はカタログに記述されている必要があります。

ALTER COLUMN *column-name*
視点で変更する列の名前です。 *column-name* は、視点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

効力範囲が未定義である既存の参照タイプ列に、効力範囲を追加します (SQLSTATE 428DK)。列をスーパー視点から継承することはできません (SQLSTATE 428DJ)。

typed-table-name

タイプ付き表の名前。 *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

タイプ付き視点の名前。 *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。

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。

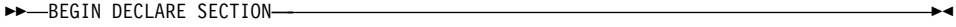
呼び出し

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のみ可能です。 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

権限は不要です。

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 (973ページの『END DECLARE SECTION』 を参照)。

規則

- BEGIN DECLARE SECTION と END DECLARE SECTION ステートメントは、対になっていなければならず、またネストされていてはなりません。
- 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。

CALL

データベースに保管されたプロシージャーを呼び出します。たとえば、ストアード・プロシージャーは、データベースの位置で実行されて、クライアント・アプリケーションにデータを戻します。

SQL CALL ステートメントを使用するプログラムは、クライアントとサーバーの 2 つの部分で実行されるように設計されます。データベースのサーバー・プロシージャーは、クライアント・アプリケーションと同じトランザクション内で実行されます。クライアント・アプリケーションとストアード・プロシージャーが同じ区分にある場合、ストアード・プロシージャーはローカルに実行されます。

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方法のみ可能です。 これは、動的に準備できない実行可能ステートメントです。ただし、プロシージャー名はホスト名によって指定することができます。これは、USING DESCRIPTOR 文節の使用とを組み合わせることによって、プロシージャー名とパラメーター・リストの両方を実行時に容易することができます。これにより、動的準備が可能なステートメントと同じ効果が得られます。

許可

許可の規則は、プロシージャーが保管されているサーバーによって異なります。

DB2 ユニバーサル・データベース:

CALL ステートメントの許可 ID の特権には、**実行時に**以下の特権のうち少なくとも 1 つが含まれていなければなりません。

- ストアード・プロシージャーに関連するパッケージの EXECUTE 特権
- ストアード・プロシージャーに関連するパッケージの CONTROL 特権
- SYSADM または DBADM 権限

DB2 ユニバーサル・データベース (OS/390 版):

CALL ステートメントの許可 ID の特権には、**バインド時に**以下の特権のうち少なくとも 1 つが含まれていなければなりません。

- ストアード・プロシージャーに関連するパッケージの EXECUTE 特権
- ストアード・プロシージャーに関連するパッケージの所有権

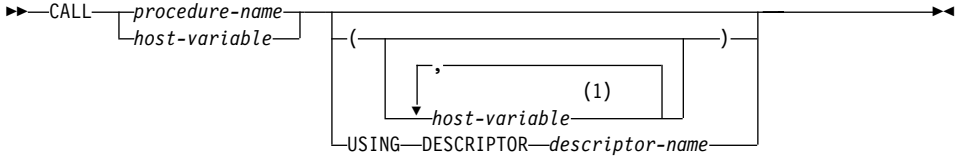
- パッケージの集合に対する PACKADM 権限
- SYSADM 権限

DB2 (AS/400 版):

CALL ステートメントの許可 ID の特権には、**バインド時に**以下の特権のうち少なくとも 1 つが含まれていなければなりません。

- ストアード・プロシージャーが REXX で作成されている場合、
  - プロシージャーに関連するソース・ファイルに対するシステム権限 *OBJOPR および *READ
  - ソース・ファイルを含むライブラリーに対するシステム権限 *EXECUTE と、 CL コマンドに対するシステム権限 *USE
- ストアード・プロシージャーが REXX で作成されていない場合、
  - プロシージャーに関連するプログラムとそのプログラムを含むライブラリーの両方に対するシステム権限 *EXECUTE
- 管理権限

構文



注:

1 DB2 ユニバーサル・データベース (OS/390 版) サーバーおよび DB2 ユニバーサル・データベース (AS/400 版) サーバーにあるストアード・プロシージャーで、 DB2 ユニバーサル・データベース (OS/390 版) クライアントまたは DB2 ユニバーサル・データベース (AS/400 版) クライアントによって呼び出されるプロシージャーでは、 プロシージャー引き数として他のソース (たとえば定数値) がサポートされます。 ただし、ストアード・プロシージャーが DB2 ユニバーサル・データベースにある場合、 またはプロシージャーが DB2 ユニバーサル・データベース・クライアントから呼び出される場合は、 引き数はすべてホスト変数を介して与える必要があります。

説明

procedure-name または *host-variable*

呼び出すプロシージャーを指定します。プロシージャー名は、直接指定す

るか (*procedure-name*)、またはホスト変数 (*host-variable*) の中に指定できます。指定するプロシージャーは、現行サーバーに存在していなければなりません (SQLSTATE 42724)。

procedure-name を指定する場合、それは 254 バイト以下の通常識別子でなければなりません。通常識別子だけを指定できるので、ブランクや特殊文字を含めることができず、値は大文字に変換されます。したがって、小文字名、ブランク、または特殊文字を使用する必要がある場合は、名前を *host-variable* (ホスト変数) によって指定する必要があります。

host-variable が指定されている場合、それは、長さ属性が 254 バイト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。値は大文字に変換**されない**ことに注意してください。*procedure-name* は左そろえでなければなりません。

プロシージャー名は、いくつかの形式のいずれかを使用して指定できます。サポートされる形式は、プロシージャーが保管されているサーバーによって異なります。

DB2 ユニバーサル・データベース:

procedure-name 実行するプロシージャーの名前 (拡張子なし)。呼び出されるプロシージャーは、以下のように決定されます。

1. *procedure-name* は、ストアード・プロシージャー・ライブラリーの名前と、そのライブラリー中の関数名の両方として使用されます。たとえば、*procedure-name* が `proclib` の場合、 DB2 サーバーは、 `proclib` という名前のストアード・プロシージャー・ライブラリーをロードし、そのライブラリー中の関数ルーチン `proclib()` を実行します。

   UNIX 系システムの場合、DB2 サーバーは、デフォルト・ディレクトリーの `sqllib/function` からストアード・プロシージャー・ライブラリーを検索します。 非分離ストアード・プロシージャーは、 `sqllib/function/unfenced` ディレクトリーにあります。

   OS/2 では、ストアード・プロシージャーの位置は、 CONFIG.SYS ファイルの LIBPATH 変数によって指定されます。非分離ストアード・

プロシージャーは、 sqllib¥dll¥unfenced ディレクトリーにあります。

2. ライブラリーまたは関数が見つからない場合、*procedure-name* を使用して、一致するプロシージャーがあるかどうか定義済みプロシージャー (SYSCAT.PROCEDURES の) が探索されます。一致するプロシージャーは、以下の手順で決定されます。

   a. カタログ (SYSCAT.PROCEDURES) から、PROCNAME が指定の *procedure-name* と一致し、 PROCSCHEMA が SQL パス (CURRENT PATH 特殊レジスター) 内のスキーマ名であるプロシージャーを見つけます。スキーマ名が明示的に指定されている場合、SQL パスは無視され、指定されたスキーマ名のプロシージャーのみが考慮されます。

   b. 次に、CALL ステートメントで指定された引き数の数と同数のパラメーターを持たないプロシージャーをすべて除去します。

   c. 残りのプロシージャーから、SQL パスの最初のプロシージャーを選択します。

   d. ステップ 2 の後で関数が残っていない場合は、エラー (SQLSTATE 42884) が戻されます。

   プロシージャーが選択されると、 DB2 は外部名によって定義されたそのプロシージャーを呼び出します。

procedure-library!function-name

感嘆符 (!) は、ストアード・プロシージャーのライブラリー名と関数名との間の区切り文字です。たとえば、proclib!func と指定した場合は、 proclib がメモリーにロードされ、そのライブラリー中の関数 func が実行されます。これによって、1 つのストアード・プロシージャー・ライブラリーの中に複数の関数を入れることができるようになります。

procedure-name の部分で説明したように、ストアード・プロシージャー・ライブラリーは、ディレクトリーに入れるか、あるいは LIBPATH 変数で指定されます。

absolute-path!function-name

absolute-path (絶対パス) には、ストアード・プロシージャー・ライブラリーまでのフルパス名を指定します。

たとえば、UNIX 系システムの場合に、`/u/terry/proclib!func` が指定されると、ストアード・プロシージャー・ライブラリーの `proclib` がディレクトリー `/u/terry` から取り出され、そのライブラリー内の関数 `func` が実行されます。

OS/2 の場合、`d:¥terry¥proclib!func` を指定すると、データベース・マネージャーは `d:¥terry¥proclib` ディレクトリーから func.dll ファイルをロードします。

いずれの場合も、暗黙のフルパスまたは明示指定のフルパスを含むプロシージャー名の全体の長さは、 254 バイトを超えてはなりません。

DB2 ユニバーサル・データベース (OS/390 版) (V4.1 以降) サーバーの場合
暗黙のまたは明示指定の 3 つの部分からなる名前。次の各部分からなります。

上位: プロシージャーが保管されているサーバーのロケーション名。

中央: SYSPROC

下位: SYSIBM.SYSPROCEDURES カタログ表の PROCEDURE 列の値。

DB2 AS/400 用 (V3.1 以降) サーバーの場合
外部プログラム名は、 *procedure-name* と同じであることが前提になります。

移植性を保つため、 *procedure-name* は、8 バイト以下の単一トークンとして指定する必要があります。

(host-variable,...)
それぞれの *host-variable* (ホスト変数) の指定は、CALL のパラメーターで

す。 CALL の n 番目のパラメーターは、サーバーのストアード・プロシージャーの n 番目のパラメーターに対応します。

各 *host-variable* は、クライアントとサーバー間の双方向のデータ交換に使用されるものとみなされます。クライアントとサーバー間での不要なデータ送信を回避するため、クライアント・アプリケーションでは、各パラメーターに標識変数を指定して、そのパラメーターがストアード・プロシージャーへのデータの送信に使用されない場合にその標識を -1 に設定する必要があります。ストアード・プロシージャーは、クライアント・アプリケーションにデータを返すために使用されないパラメーターすべてについて、標識変数を -128 に設定する必要があります。

サーバーが DB2 ユニバーサル・データベースの場合、パラメーターのデータ・タイプは、クライアントとサーバーの両方のプログラムで一致している必要があります。[62]

USING DESCRIPTOR *descriptor-name*

ホスト変数の有効な記述を含む SQLDA を指定します。 n 番目の SQLVAR 要素は、サーバーのストアード・プロシージャーの n 番目のパラメーターに対応します。

CALL ステートメントが処理される前に、アプリケーションでは、SQLDA 中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 の要素数を示す SQLN
- SQLDA に割り振る記憶域のバイト数を示す SQLDABC
- ステートメントの処理中に SQLDA で使用される変数の数を示す SQLD
- 変数の属性を示す SQLVAR オカレンス。渡される各基本 SQLVAR 要素の次のフィールドは、初期化しておく必要があります。
  - SQLTYPE
  - SQLLEN
  - SQLDATA
  - SQLIND

  渡される各 2 次 SQLVAR 要素の次のフィールドは、初期化しておく必要があります。

62. DB2 ユニバーサル・データベース (OS/390 版) サーバーと DB2 ユニバーサル・データベース (AS/400 版) サーバーでは、ストアード・プロシージャーの呼び出し時に、互換性のあるデータ・タイプの間での変換がサポートされています。たとえば、クライアント・プログラムが INTEGER データ・タイプを使用し、ストアード・プロシージャーが FLOAT を予期している場合、サーバーはそのプロシージャーの呼び出しに先立って、INTEGER の値を FLOAT に変換します。

- LEN.SQLLONGLEN
- SQLDATALEN
- SQLDATATYPE_NAME

各 SQLDA は、クライアントとサーバーの間の双方向のデータ交換に使用されるものとみなされます。クライアントとサーバー間での不要なデータの送信を避けるため、クライアント・アプリケーションでは、ストアード・プロシージャーへのデータの送信にパラメーターが使用されない場合に、 SQLIND フィールドを -1 に設定する必要があります。ストアード・プロシージャーは、クライアント・アプリケーションにデータを返すために使用されないパラメーターすべてについて、 SQLIND フィールドを -128 に設定する必要があります。

注

- ***ラージ・オブジェクト (LOB) データ・タイプの使用***

  クライアントとサーバー・アプリケーションで、 SQLDA から LOB データを指定して、SQLVAR 項目数の 2 倍を割り振る必要があります。

  LOB データ・タイプは、DB2 バージョン 2 からストアード・プロシージャーでサポートされています。 LOB データ・タイプは、それより下位レベルのクライアントまたはサーバーでは、まったくサポートされていません。

- ***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。

- ***ストアード・プロシージャーから戻される結果セット***:

  クライアント・アプリケーション・プログラムが CLI を使用して作成されている場合、結果セットをクライアント・アプリケーションに直接戻すことができます。ストアード・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。

  CLI によって呼び出された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のすべてについて、結果セットがアプリケーションに戻されます。
  - 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、それらのカーソルがオープンされた順序で戻されます。

– 読んでいない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 行がストアード・プロシージャーの終了時にストアード・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 行までがストアード・プロシージャーに戻されます。

詳細については、*アプリケーション開発の手引き* および *コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

- ***CALL ステートメントと DARI API の間の内部動作可能性***

  一般に、CALL ステートメントは、既存の DARI プロシージャーでは機能しません。詳細については、*アプリケーション開発の手引き* を参照してください。

- ***特殊レジスターの取り扱い***:

  呼び出し側の特殊レジスターの設定値は、起動時にストアード・プロシージャーに継承され、呼び出し側に戻されるとただちに復元されます。ストアード・プロシージャー内で特殊レジスターを変更してもかまいませんが、その変更で呼び出し側に影響を与えることはありません。ただし、既存のストアード・プロシージャー (パラメーター・スタイル DB2DARI で定義されているか、またはデフォルト・ライブラリーにあるもの) の場合はそうではなく、プロシージャー内で特殊レジスターに対して加えた変更は、呼び出し側の設定値になります。

例

例 1:

C において、TEAMWINS というプロシージャーを ACHIEVE ライブラリーから呼び出し、ホスト変数 HV_ARGUMENT に保管されているパラメーターをそれに渡します。

```
strcpy(HV_PROCNAME, "ACHIEVE!TEAMWINS");
CALL :HV_PROCNAME (:HV_ARGUMENT);
```

例 2:

C において、:SALARY_PROC というプロシージャーを、 INOUT_SQLDA という名前の SQLDA を使用して呼び出します。

```
struct sqlda *INOUT_SQLDA;

/* Setup code for SQLDA variables goes here */

CALL :SALARY_PROC
USING DESCRIPTOR :*INOUT_SQLDA;
```

例 3:

Java ストアード・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ストアード・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, variable1 ) ;
stpCall.setBigDecimal( 2, variable2 ) ;
stpCall.setInt( 3, variable3 ) ;

stpCall.registerOutParameter( 2, Types.DECIMAL, 2 ) ;
stpCall.registerOutParameter( 3, Types.INTEGER ) ;

stpCall.execute() ;

variable2 = stpCall.getBigDecimal(2) ;
variable3 = 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。

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。

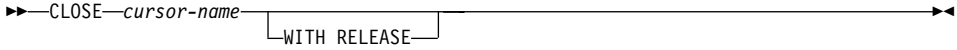
呼び出し

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、動的に準備できない実行可能ステートメントです。

許可

権限は不要です。カーソルの使用に必要な許可については、914ページの『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、すべての読み取りロックを解放しようとします。すべての読み取り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。
- CLOSE では、コミット操作やロールバック操作は行われません。
- 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 ’単独読み取り’ 状態のどちらかになる可能性があります。詳細については、1395ページの『付録I. 分離レベルの比較』を参照してください。

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、新しい読み取り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。詳細については、564ページの『注』 を参照してください。

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;
```

COMMENT ON

COMMENT ON 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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

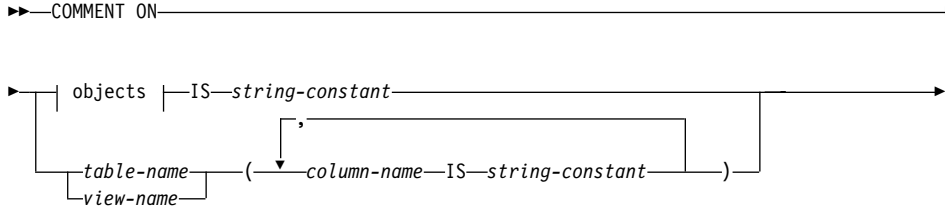
許可

COMMENT ON 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。

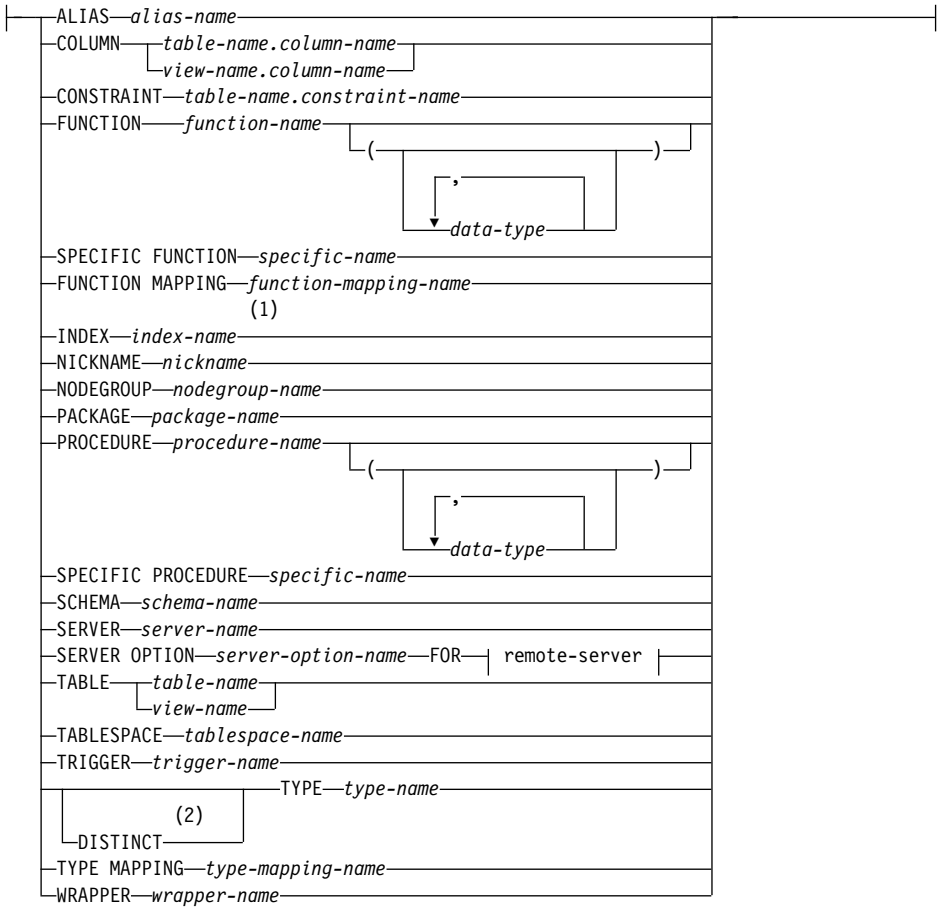
- SYSADM、または DBADM
- オブジェクトのカタログ視点の DEFINER 列 (スキーマの場合は OWNER 列) に記録されているオブジェクトの定義者 (列または制約の場合は基礎表)
- スキーマに対する ALTERIN 特権 (複数部分の名前を使用可能なオブジェクトにのみ適用される)
- オブジェクトに対する CONTROL 特権 (索引、パッケージ、表、および視点の各オブジェクトにのみ適用される)
- オブジェクトに対する ALTER 特権 (表オブジェクトにのみ適用される)

表スペースまたはノードグループの場合、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。

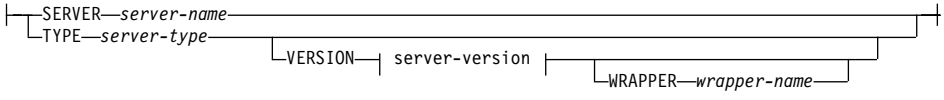
構文



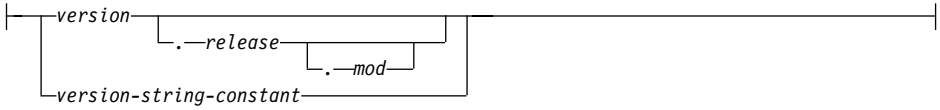
objects:



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 カタログ視点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* (視点名.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 カタログ視点のその列を記述する行の REMARKS 列の値を置き換えます。

作動不能視点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 カタログ視点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 特殊レジスターは、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 / バインド・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、エラー (SQLSTATE 42854) 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 特殊レジスターは、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 / バインド・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 ステートメントの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 が修飾されない場合、 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。 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。 タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

(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みなされません。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 特殊レジスターは、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 / バインド・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 スキーマまたは SYSFUN スキーマのいずれかにある関数にコメントを付けることはできません (SQLSTATE 42832)。

コメントは、SYSCAT.FUNCTIONS カタログ視点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 カタログ視点のうち、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 カタログ視点のうち、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 カタログ視点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。

NODEGROUP *nodegroup-name*
ノードグループ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。 *nodegroup-name* (ノードグループ名) は、カタログに記述されている特定のノード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.NODEGROUPS カタログ視点のうちそのノードグループを記述する行の REMARKS 列の値を置き換えます。

PACKAGE *package-name*
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。 *package-name* (パッケージ名) は、カタログに記述されている特定の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 カタログ視点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。

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。このように指定するプロシージャーには、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 特殊レジスターは、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 / バインド・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の名前のプロシージャー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、このプロシージャーの特定インスタンスが複数個ある場合、エラー (SQLSTATE 42854) になります。

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 特殊レジスターは、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 / バインド・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 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 が修飾されない場合、 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。 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。タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

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 特殊レジスターは、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 / バインド・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 は、指定したスキーマまたは暗黙のスキーマの特定プロシージャー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。

コメントは、SYSCAT.PROCEDURES カタログ視点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 (スキーマ名) は、カタログに記述されているスキーマを指定していなければなりません (SQLSTATE 42704)。コメントは、SYSCAT.SCHEMATA カタログ視点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 (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 (SQLSTATE 42704)。コメントは、SYSCAT.SERVERS カタログ視点のうちそのデータ・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 カタログ視点のうち、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 ユニバーサル・データベース (OS/390 版) 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*
表または視点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* (視点名) は、カタログ (SQLSTATE 42704) に記述されている表または視点 (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、宣言された一時表 (SQLSTATE 42995) を指定してはなりません。コメントは、 SYSCAT.TABLES カタログ視点のうちその表または視点を記述する行の 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 カタログ視点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 カタログ視点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 カタログ視点の 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 特殊レジスターは、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 / バインド・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 カタログ視点のうち、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 カタログ視点のうち、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 (LR) 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 ON ステートメントを使用すると、表または視点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視点の列を指定するものでなければなりません。また、その表または視点はカタログに記述されていなければなりません。 *table-name* を宣言された一時表 (SQLSTATE 42995) にすることはできません。

作動不能視点の列にコメントを作成することはできません (SQLSTATE 51024)。

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 視点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、その作業単位によって行われたデータベース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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。

許可

権限は不要です。

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。その作業単位がコミットされる過程で、実行された ALTER、 COMMENT ON、 CREATE、 DELETE、 DROP、 GRANT、 INSERT、 LOCK TABLE、 REVOKE、 SET INTEGRITY、 SET transition-variable (変位変数)、および UPDATE の各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。

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 ステートメントの発行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 PACKAGESET,
- SET CURRENT QUERY OPTIMIZATION,
- SET CURRENT REFRESH AGE,
- SET EVENT MONITOR STATE,
- SET PASSTHRU,
- 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で、 WITH HOLD を宣言されたオープン・カーソルに必要なロックを除く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オープン・カーソルはすべてクローズされます。 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。[63]すべての LOB ロケーターは解放されます。 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保管点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。各種のアプリケーション環境でのトランザクションの暗黙的な終了については、 *アプリケーション開発の手引き* を参照してください。
- 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、 975ページの『EXECUTE』を参照してください。
- 宣言された一時表に対する COMMIT の影響の可能性については、 920ページの『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
```

63. 定位置 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。

複合 SQL (組み込み)

1 つまたは複数の他の 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。 SQL プロシージャー内の複合 SQL ステートメントについては、 1157ページの『第7章 SQL プロシージャー』を参照してください。

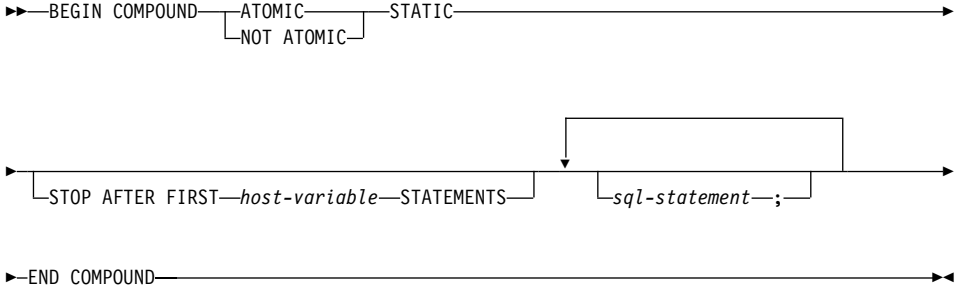
呼び出し

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方法のみ可能です。複合 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

複合 SQL ステートメント自体には必要ありません。複合 SQL ステートメントの許可 ID には、複合 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
複合 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複合 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、複合 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、複合 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精度整数。

STATEMENTS

STOP AFTER FIRST *host-variable* 文節を完結します。

sql-statement

組み込み静的複合 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
CLOSE
CONNECT
複合 SQL
DESCRIBE
DISCONNECT
EXECUTE IMMEDIATE
FETCH
OPEN
PREPARE
RELEASE (接続)
RELEASE SAVEPOINT
ROLLBACK
SAVEPOINT
SET CONNECTION

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 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複合 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 が組み込まれると、エラーが戻されます (SQLSTATE 25000)。

規則

- DB2 コネクトでは、複合 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。
- 複合 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、複合 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 複合 SQL ステートメントのみが、 DB2 コネクトにより受け入れられます。
- 複合 SQL ステートメントはネストできません。
- ATOMIC 複合 SQL ステートメントは、保管点内では発行できません (SQLSTATE 3B002)。

注

複合 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対するものです (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。[64]
- 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 複合 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、複合 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、各ストリング要素は、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。[65] 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。[65] 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、複合 SQL ステートメント内の 3 番目のサブステー

64. 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' 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。複合 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のフィールドです。

65. 数が 999 を超えると、カウントは 0 から再び始まります。

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複合 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;
```

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、暗黙にデフォルトのそ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。

接続状態の概念と詳細については、35ページの『リモート作業単位の接続管理』 を参照してください。 CONNECT の機能の枠組みを決めるプリコンパイラー・オプションについては、 44ページの『分散作業単位の意味を制御するオプション』 を参照してください。

呼び出し

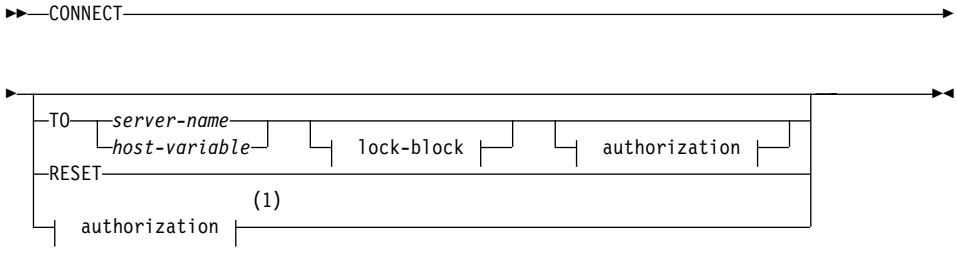
対話式 SQL 機能には外見上対話式の実行に見えるインターフェースが用意されている場合がありますが、このステートメントはアプリケーション・プログラムに組み込むことだけが可能です。 これは、動的に準備できない実行可能ステートメントです。

許可

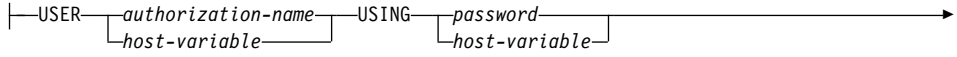
このステートメントの許可 ID には、指定されたアプリケーション・サーバーに接続するための許可が必要です。データベースの認証の設定値によっては、クライアントまたはサーバーのいずれかによって許可検査が行われる場合があります。区分データベースの場合、ユーザーとグループの定義は、区分またはノードのすべてにわたって同一である必要があります。認証設定値については、 *管理の手引き* の AUTHENTICATION データベース・マネージャー構成パラメーターの項を参照してください。

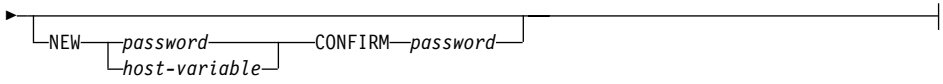
構文

CONNECT (タイプ 1)

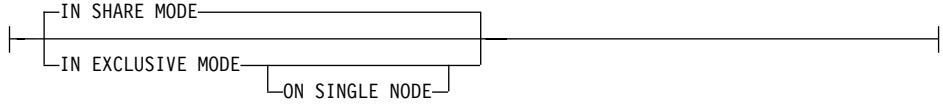


authorization:





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・サーバーの国別コードとコード・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。
- 接続された場合、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を含む *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* に入っている *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 (MVS 版) では 16 バイトのロケーション名がサポートされており、 SQL/DS と DB2/400 ではいずれも 18 バイトの宛先データベース名がサポートされます。 DB2 バージョン 7 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 (タイプ 1 CONNECT の接続状態については、 35ページの『リモート作業単位の接続管理』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、

- 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 (MVS 版) の場合は DSN
    - SQL/DS の場合は ARI
    - DB2/400 の場合は QSQ
    - DB2 ユニバーサル・データベース の場合は SQL
  - *vv* は、2 桁のバージョン識別子です ('02'など)。
  - *rr* は、2 桁のリリース識別子です ('01' など)。
  - *m* は、1 桁の修正レベル識別子です ('0'など)。

  たとえば、アプリケーション・サーバーが DB2 (OS/2 版) のバージョン 1 リリース 1 の場合、 SQLERRP の値は 'SQL01010' になります。[66]
- 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・サーバーの国別コード (DDCS を使用している場合はブランク)。

66. この DB2 ユニバーサル・データベース バージョン 7 のリリースは 'SQL07010' です。

2. アプリケーション・サーバーのコード・ページ (DDCS 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 ユニバーサル・データベース (AS/400 版)
QDB2	DB2 ユニバーサル・データベース (OS/390 版)
QDB2/2	DB2 ユニバーサル・データベース (OS/2 版)
QDB2/6000	DB2 ユニバーサル・データベース (AIX 版)
QDB2/HPUX	DB2 ユニバーサル・データベース (HP-UX 版)
QDB2/LINUX	DB2 ユニバーサル・データベース (Linux 版)
QDB2/NT	DB2 ユニバーサル・データベース (Windows NT 版)
QDB2/PTX	DB2 ユニバーサル・データベース (NUMA-Q 版)
QDB2/SCO	DB2 ユニバーサル・データベース (SCO UnixWare 版)
QDB2/SNI	DB2 ユニバーサル・データベース (Siemens Nixdorf 版)
QDB2/SUN	DB2 ユニバーサル・データベース (Solaris 実行環境版)
QDB2/Windows 95	DB2 ユニバーサル・データベース (Windows 95 版または Windows 98 版)
QSQLDS/VM	DB2 サーバー (VM 版)

QSQLDS/VSE DB2 サーバー (VSE 版)

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 要素と同じです。
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. 区分番号。非区分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. 区分データベースの区分の数。データベースを区分化できない場合、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 よりも大きい値は、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。[67]
- 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 よりも大きい値は、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。[67]
- 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によって明らかになると、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。

67. 詳細については、 *アプリケーション開発の手引き* の『複合環境におけるプログラミング』の章の『文字変換の拡張係数』の項を参照してください。

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 モニター環境の 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 接続を使用して認証された。
  3 - 分散コンピューティング環境の機密保護サービスを使用して認証された。
  255 - 認証は指定されていない。

  認証タイプの詳細については、 *管理の手引き* の『データベース・アクセスの制御』を参照してください。
- SQLCA の SQLERRD(6) フィールドは、データベースが区分化されている場合に、接続された区分の区分番号を戻します。区分化されていない場合、値 0 が戻されます。
- 正常に実行された許可 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、製品を識別する点に注意してください。たとえば、アプリケーション・リクエスターが OS/2 データベース・マネージャーの場合、最初の 3 文字は 'SQL' になります。
- 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[68]
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

ON SINGLE NODE
調整プログラムの区分を排他モードで接続し、その他のすべての区分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、区分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です。

USER *authorization-name/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 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。 *host-variable* に含まれるユーザー 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。

68. このオプションは DDCS ではサポートされません。

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

- 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。
- DB2 ユニバーサル・データベース エンタープライズ拡張エディションを使用するには、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区分のいずれかに接続する必要があります (このファイルについての詳細は、 67ページの『複数の区分にわたるデータの区分化』を参照してください)。調整プログラムの区分と同じ区分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。

例

例 1: C プログラムで、ユーザー ID FERMAT とパスワード THEOREM を使用して、アプリケーション・サーバー TOROLAB3 に接続します。ここで、TOROLAB3 は同じ名前のデータベース別名です。

```
EXEC SQL  CONNECT TO TOROLAB3 USER FERMAT USING THEOREM;
```

例 2: C プログラムで、データベース名がホスト変数 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 文字の製品識別子を、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);
```

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。

概念と詳細については、40ページの『アプリケーション制御の分散作業単位』を参照してください。

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 の要素のうちタイプ 1 とは異なる部分だけを説明します。

呼び出し

呼び出しは 589ページの『呼び出し』 の説明と同じです。

許可

許可は 589ページの『許可』 の説明と同じです。

構文

構文は 589ページの『構文』 と同じです。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、44ページの『分散作業単位の意味を制御するオプション』 を参照してください。詳細については、*コマンド解説書* および *管理 API 解説書* を参照してください。

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、サーバー名 *(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 の指定については、44ページの『分散作業単位の意味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、タイプ 1 の CONNECT と同じ方法で更新されます。 592 ページを参照してください。

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。インストール時の暗黙接続と環境変数については、 *概説およびインストール* を、認証設定値については、*管理の手引き* を参照してください。

規則

- 44ページの『分散作業単位の意味を制御するオプション』 で概略を説明したように、一連の接続オプションによって、接続管理の意味が制御されます。すべての事前処理済みソース・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かぎり、実行時に実行される最初の SQL ステートメントを含むソース・ファイルの事前処理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 ステートメントは、異なるさまざまな接続オプションで次々に事前処理され、その合間に SET CLIENT コマンドまたは API が実行されることなく実行されると、エラー (SQLSTATE 08001) になります。 SET CLIENT コマンドまたは API が実行されると、アプリケーション内のすべてのソース・ファイルを事前処理するために使用された接続オプションは無視されます。

  これらの規則は、605 ページの例 1 に示されています。

- 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か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、それは拒否されます (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。

タイプ 1 とタイプ 2 の CONNECT ステートメントの比較

CONNECT ステートメントの意味は、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 (44ページの『分散作業単位の意味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。それらの意味について、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、接続を解放保留にします。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。
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るなら、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。

タイプ 1	タイプ 2
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、および解放保留のもの) 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) を指定して前処理され、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	解放保留
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 SELECT * FROM TBLB	S1 S1	S0 S0	なし なし
3	CONNECT TO S2 UPDATE TBLC SET ...	S2 S2	S0、S1 S0、S1	なし なし
4	CONNECT TO S3 SELECT * FROM TBLD	S3 S3	S0、S1、S2 S0、S1、S2	なし 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 SELECT * FROM TBLF	S2 S2	S0 S0	なし なし

CREATE ALIAS

CREATE ALIAS ステートメントは、表、視点、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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

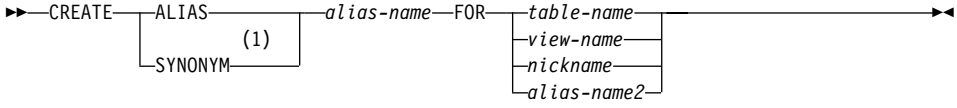
許可

このステートメントの許可 ID 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 (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



注:

1 他の SQL 処理系に存在する既存の CREATE SYNONYM ステートメントの構文を許容するために、 CREATE ALIAS の代わりに CREATE SYNONYM が受け入れられます。

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、視点、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 *(table-name)*、視点名 *(view-name)*、ニックネーム *(nickname)*、または別名 *(alias-name)* 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* を宣言された一時表 (SQLSTATE 42995) にすることはできません。

注

- 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連鎖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連鎖にも適用されます。別名連鎖は、単一の別名と同じ方法で解決されます。視点定義、パッケージ内のステートメント、または別名連鎖を指すトリガーで別名が使用された場合、その視点、パッケージ、または連鎖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。別名連鎖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 特権は PUBLIC に与えられます。

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
```

HEDGES.T1 に対して別名 MCKNIGHT.A1 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、別名連鎖が作成されます。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 を連合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。連合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。バッファー・プールの定義はトランザクションとして、コミット時にその項目がカタログ表に反映されますが、そのバッファー・プールは次回にデータベースが始動されるまではアクティブになりません。

区分データベースでは、特定の区分またはノードを上書きする機能を伴うデフォルトのバッファー・プール定義が、それぞれの区分またはノードに対して指定されます。また区分データベースでは、ノード・グループを指定しない限り、すべての区分にバッファー・プールが定義されます。ノード・グループを指定すると、そのノード・グループの区分にだけバッファー・プールが作成されます。

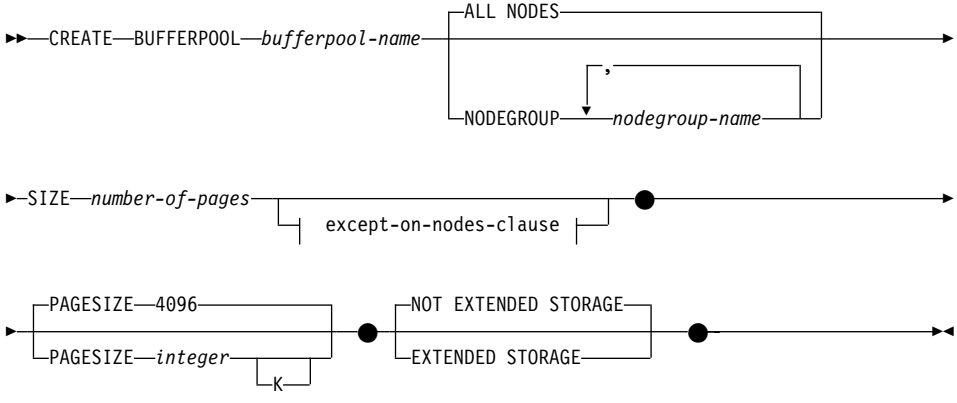
呼び出し

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。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

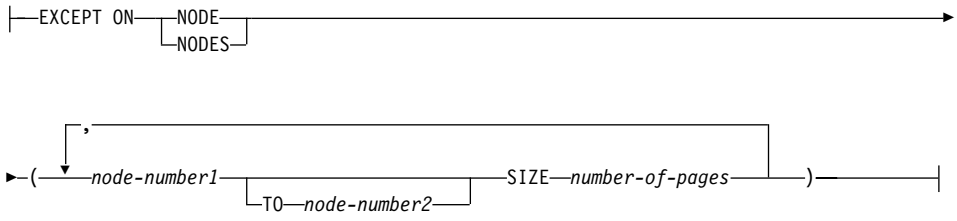
許可

このステートメントの許可 ID 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-nodes-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 識別子です (通常識別子または区切り識別子)。*bufferpool-name* 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″ または ″IBM″ で始めることはできません (SQLSTATE 42939)。

ALL NODES
このバッファー・プールは、データベースのすべての区分に作成されます。

NODEGROUP *nodegroup-name, ...*
バッファー・プール定義が適用されるノード・グループを指定します。これを指定すると、バッファー・プールはこれらのノード・グループの区分にだけ作成されます。それぞれのノード・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)。 NODEGROUP キーワードを指定しないと、このバッファー・プールはすべての区分 (およびその後データベースに追加される区分) に作成されます。

SIZE *number-of-pages*
ページ数で指定するバッファー・プールのサイズ。[69] 区分データベースでは、このサイズはバッファー・プールが存在するすべての区分のデフォルトのサイズになります。

except-on-nodes-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

69. サイズに (-1) の値を指定することができます。この値は、バッファー・プール・サイズを BUFFPAGE データベース構成パラメーターから取ることを指定します。

にしたい区分を指定します。 この文節の指定がない場合、すべての区分が、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 NODES
特定の区分を指定することを示すキーワードです。 NODE は NODES の同義語です。

node-number1
バッファー・プールが作成される区分に含まれる特定の区分の番号を指定します。

TO *node-number2*
区分番号の範囲を指定します。 *node-number2* の値は、*node-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)。指定する区分番号の範囲 (指定する区分番号を含む) のすべての区分は、バッファー・プールを作成する区分に含まれていなければなりません (SQLSTATE 42729)。

SIZE *number-of-pages*
ページ数で指定するバッファー・プールのサイズ。

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
拡張記憶域構成がオンになっている場合、[70] このバッファー・プールの外に移送されたページは、拡張記憶域にキャッシュされます。

NOT EXTENDED STORAGE
データベースの拡張記憶域構成がオンになっていても、このバッファー・プールの外に移送されるページは、拡張記憶域にキャッシュされません。

---

70. 拡張記憶域構成は、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。詳細については、*管理の手引き* を参照してください。

注

- データベースが次回始動するまで、作成される表スペースは、すでに活動状態になっている同じページ・サイズのバッファー・プールを使用します。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。
- 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 がすべてのバッファー・プールに必要な合計のメモリーを入手できない場合には、 DB2 はデフォルトのバッファー・プールのみの始動を試みます。これが失敗すると、最小のデフォルト・バッファー・プールを始動します。いずれの場合もユーザーに警告が出され (SQLSTATE 01626)、すべての表スペースのページはデフォルトのバッファー・プールを使用します。

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

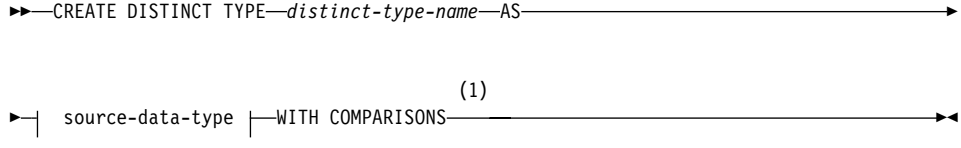
このステートメントは、アプリケーション・プログラムに組み込むか、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

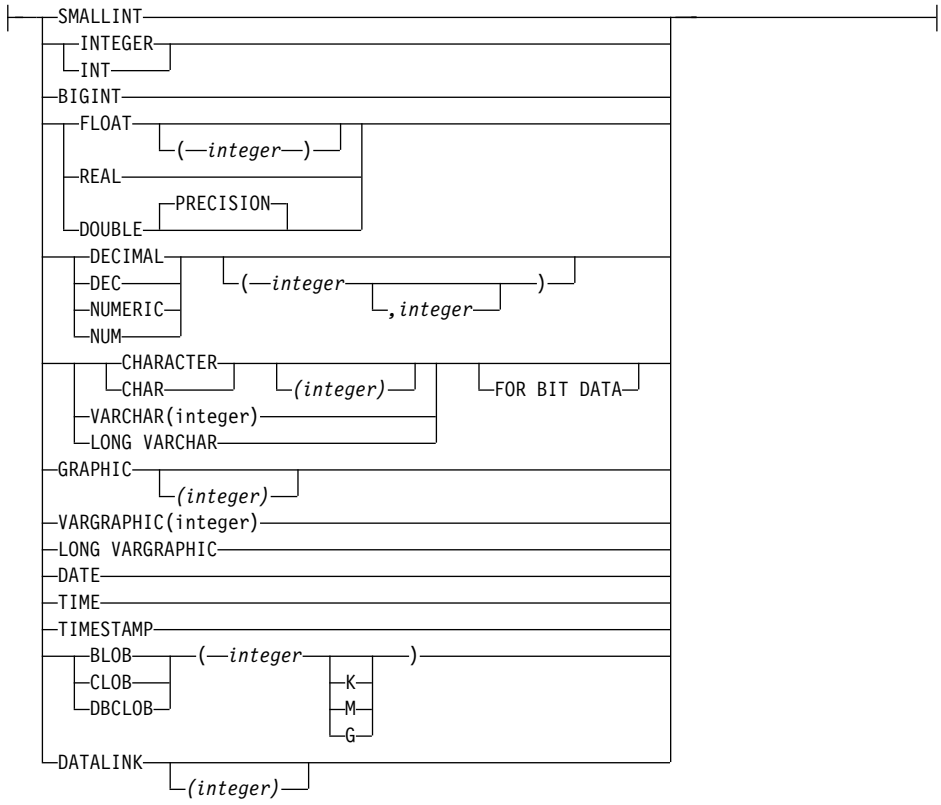
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- 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 特殊レジスターは、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 / バインド・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 識別子が続きます。

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。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。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基礎として使用されるデータ・タイプを指定します。特殊タイプと他のデータ・タイプとの関連づけについては、 97ページの『特殊タイプ』 を参照してください。データ・タイプについては、771ページの『CREATE TABLE』 を参照してください。

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

- 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 特権は PUBLIC に与えられます。
- 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、620ページの表20 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 20. 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	<特殊>	CHAR (n)	<特殊>
	CHAR	<特殊>	CHAR (n)
	<特殊>	VARCHAR (n)	<特殊>
VARCHAR	<特殊>	VARCHAR (n)	<特殊>
	VARCHAR	<特殊>	VARCHAR (n)
LONG VARCHAR	<特殊>	LONG VARCHAR	<特殊>
	LONG_VARCHAR	<特殊>	LONG VARCHAR
CLOB	<特殊>	CLOB (n)	<特殊>
	CLOB	<特殊>	CLOB (n)
BLOB	<特殊>	BLOB (n)	<特殊>
	BLOB	<特殊>	BLOB (n)
GRAPHIC	<特殊>	GRAPHIC (n)	<特殊>
	GRAPHIC	<特殊>	GRAPHIC (n)
	<特殊>	VARGRAPHIC (n)	<特殊>
VARGRAPHIC	<特殊>	VARGRAPHIC (n)	<特殊>
	VARGRAPHIC	<特殊>	VARGRAPHIC (n)
LONG VARGRAPHIC	<特殊>	LONG VARGRAPHIC	<特殊>
	LONG_VARGRAPHIC	<特殊>	LONG VARGRAPHIC
DBCLOB	<特殊>	DBCLOB (n)	<特殊>
	DBCLOB	<特殊>	DBCLOB (n)
SMALLINT	<特殊>	SMALLINT	<特殊>
	<特殊>	INTEGER	<特殊>
	SMALLINT	<特殊>	SMALLINT
INTEGER	<特殊>	INTEGER	<特殊>
	INTEGER	<特殊>	INTEGER
BIGINT	<特殊>	BIGINT	<特殊>
	BIGINT	<特殊>	BIGINT
DECIMAL	<特殊>	DECIMAL (p,s)	<特殊>
	DECIMAL	<特殊>	DECIMAL (p,s)
NUMERIC	<特殊>	DECIMAL (p,s)	<特殊>
	DECIMAL	<特殊>	DECIMAL (p,s)

表 20. 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
REAL	<特殊>	REAL	<特殊>
	<特殊>	DOUBLE	<特殊>
	REAL	<特殊>	REAL
FLOAT(*n*) ただし *n*<=24	<特殊>	REAL	<特殊>
	<特殊>	DOUBLE	<特殊>
	REAL	<特殊>	REAL
FLOAT(*n*) ただし *n*>24	<特殊>	DOUBLE	<特殊>
	DOUBLE	<特殊>	DOUBLE
FLOAT	<特殊>	DOUBLE	<特殊>
	DOUBLE	<特殊>	DOUBLE
DOUBLE	<特殊>	DOUBLE	<特殊>
	DOUBLE	<特殊>	DOUBLE
DOUBLE PRECISION	<特殊>	DOUBLE	<特殊>
	DOUBLE	<特殊>	DOUBLE
DATE	<特殊>	DATE	<特殊>
	DATE	<特殊>	DATE
TIME	<特殊>	TIME	<特殊>
	TIME	<特殊>	TIME
TIMESTAMP	<特殊>	TIMESTAMP	<特殊>
	TIMESTAMP	<特殊>	TIMESTAMP
DATALINK	<特殊>	DATALINK	<特殊>
	DATALINK	<特殊>	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、組み込み関数 (AVG、MAX、LENGTH、など) は、 CREATE FUNCTION ステートメント (634ページの『CREATE FUNCTION』を参照) を使用して、特殊タイプに対応するユーザー定義関数 (適切な組み込み関数に基づく) を登録してからでないと、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。それらの比較演算子の作成により、SYSCAT.FUNCTIONS カタログ視点に新しい関数としての項目が生成されます。

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 (1126ページの『SET PATH』、またはアプリケーション*開発の手引き* で説明されている 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。

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場所も指定されます。

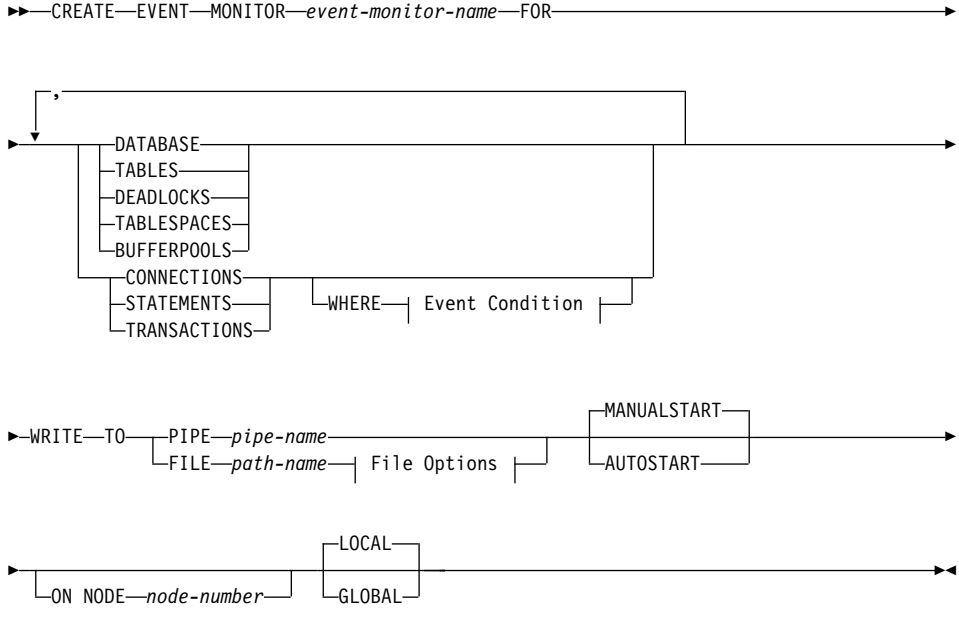
呼び出し

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。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

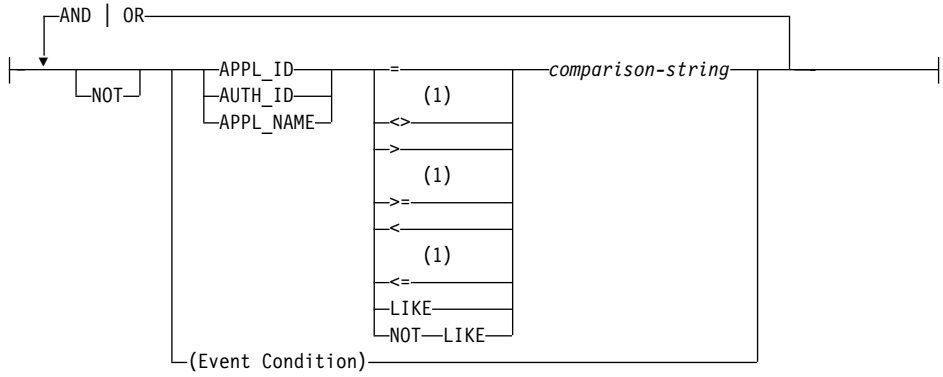
許可

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)。

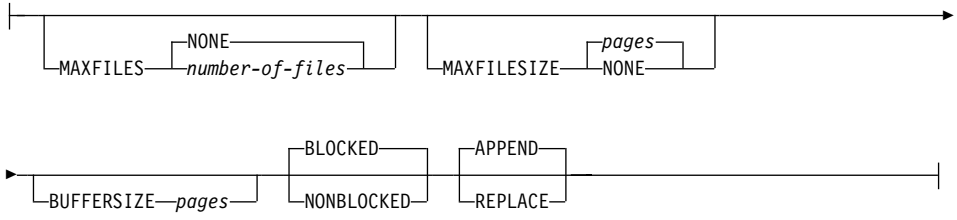
構文



イベント条件:



ファイル・オプション:



注:

1 これらの演算子の他の形式もサポートされます。 詳細については、209ページの『基本述部』を参照してください。

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 識別子です (通常識別子または区切り識別子)。*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・モニターが活動状態の各表の表イベントを記録することを指定します。活動状態の表とは、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。

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)、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あり、標準探索条件と混同してはなりません。

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の活動状態の各接続が評価されます。
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。

APPL_ID
各接続のアプリケーション ID を、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) を生成するかを判別するために *comparison-string* (比較ストリング) と比較しなければならないことを指定します。

AUTH_ID
各接続の許可 ID を、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) を生成するかを判別するために *comparison-string* と比較しなければならないことを指定します。

APPL_NAME
各接続のアプリケーション・プログラム名を、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) を生成するかを判別するために *comparison-string* と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。

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、監視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、OS/2、Windows 95 および Windows NT では、パイプ名に関して特殊な構文があります。その結果、OS/2、Windows 95 および Windows NT 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。監視元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能な場合には、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” を伴う一連の 8 文字の番号のファイルとして、データのストリームを書き出します (たとえば、00000000.evt、00000001.evt、および 00000002.evt)。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み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・モニターが、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、パス自体は別の区分またはノード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・モニターの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 OS/2 、 Windows 95 および Windows NT の場合にディスク識別子で始まるパス) 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 カタログ視点に保管されます。

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 つが初めて活動化されると、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。これはデフォルト値です。

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。

アプリケーションがイベント・ファイルを書き込んだ後、ディレクトリーからそれを削除した場合は、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。

- OS/2、Windows 95 および Windows NT - 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、このオプションの指定がない場合のデフォルト・サイズは、 4 ページです (つまり、それぞれサイズが 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ます。

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活動イベント・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、活動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場合、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・モニターがそのイベント・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 オプションを伴うイベント・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 NODE
特定の区分を指定することを示すキーワードです。

node-number
イベント・モニターが実行され、そのイベントを書き込む区分の番号を指定します。モニターの効力範囲が GLOBAL として定義されている場合、すべての区分が指定の区分番号に報告を行います。入出力構成要素は指定の区分で物理的に稼働し、レコードをその区分の /tmp/dlocks ディレクトリーに書き込みます。

GLOBAL
イベント・モニターはすべての区分について報告します。 DB2 ユニバーサル・データベース バージョン 7 の区分データベースの場合、GLOBAL として定義できるのはデッドロックイベント・モニターだけです。 グローバルイベント・モニターは、システムのすべてのノードのデッドロックを報告します。

LOCAL
イベント・モニターは稼働している区分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。これはデフォルト値です。

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

- イベント・モニターの定義は、SYSCAT.EVENTMONITORS カタログ視点に記録されます。イベント自体は、SYSCAT.EVENTS カタログ視点に記録されます。
- データベース・モニターの使用、およびパイプやファイルからのデータの解釈についての詳細は、 システム・モニター *手引きおよび解説書* を参照してください。

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'
```

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。 635ページの『CREATE FUNCTION (外部スカラー)』を参照してください。

- 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。 663ページの『CREATE FUNCTION (外部表)』を参照してください。

- 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。 681ページの『CREATE FUNCTION (OLE DB 外部表)』を参照してください。

- 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実装されます。 690ページの『CREATE FUNCTION (ソースまたはテンプレート)』を参照してください。

  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。ユーザーはこれを連合システム内のデータ・ソース関数にマップし、連合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、連合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。 701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』 を参照してください。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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 (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)

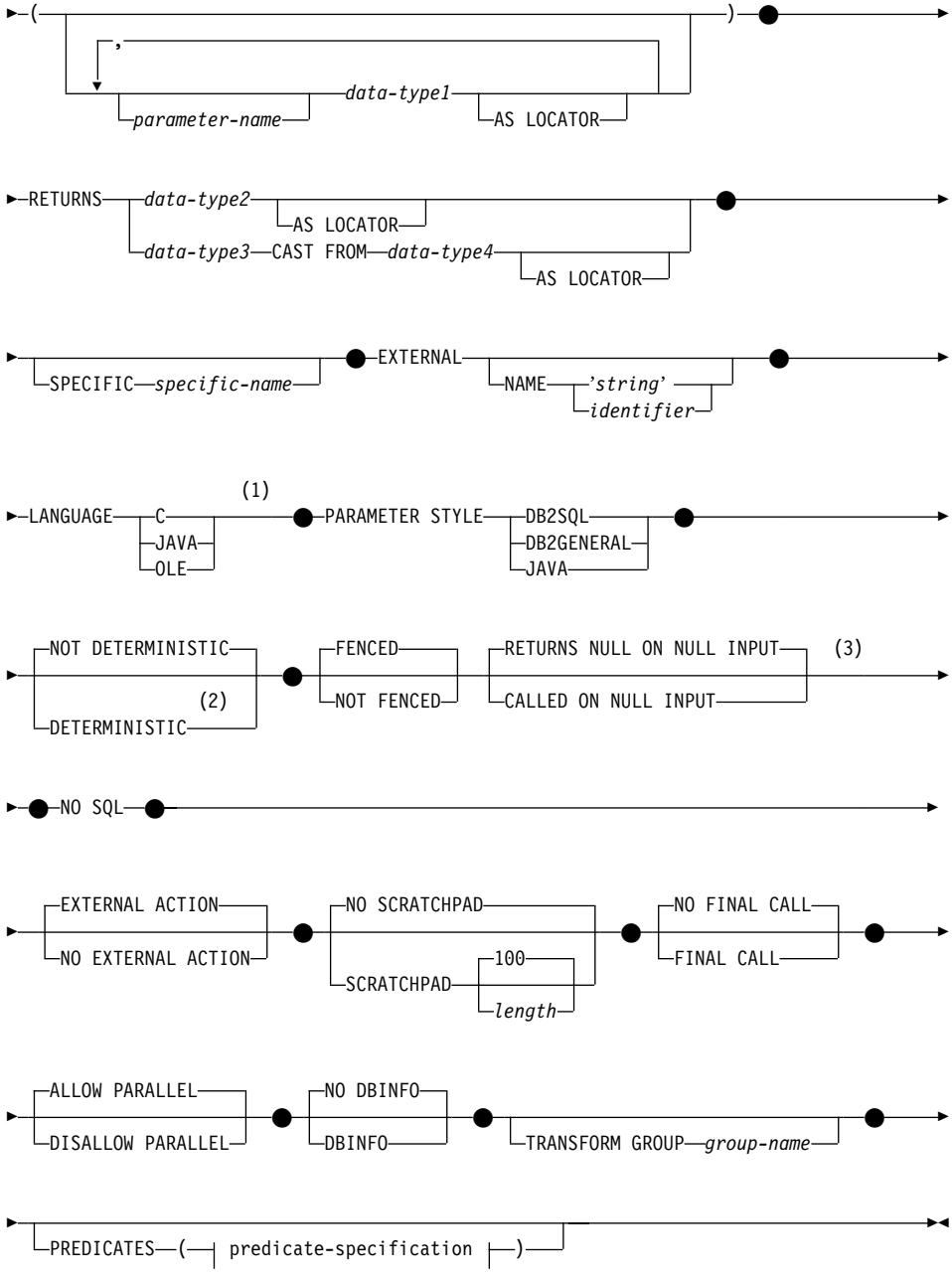
非分離の関数を作成するには、ステートメントの許可 ID の特権に、以下の特権の少なくとも 1 つが含まれている必要があります。

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

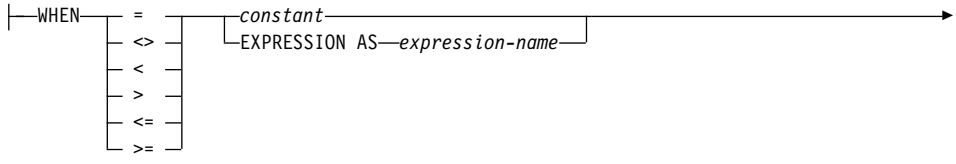
分離関数を作成する場合には、さらに別の権限や特権は必要ありません。

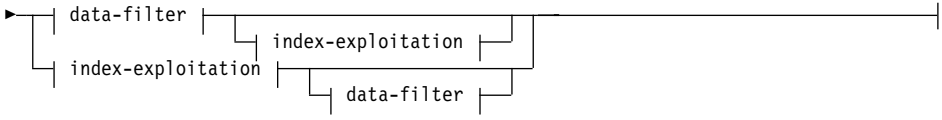
許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

構文

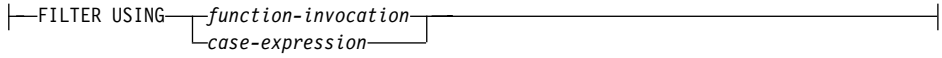


predicate-specification:

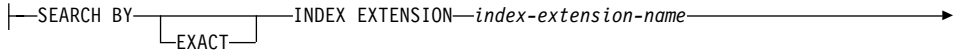




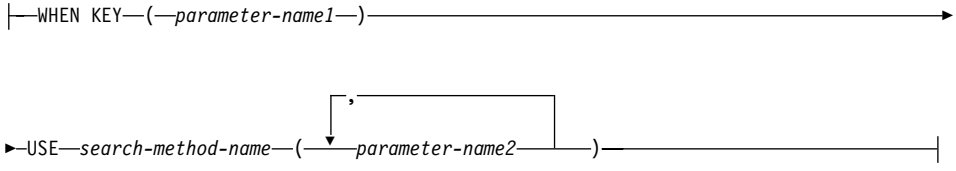
data-filter:



index-exploitation:



exploitation-rule:



注:

1 LANGUAGE SQL もサポートされています。 701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』 を参照してください。

2 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。

3 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。

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 識別子です (最大長 18)。 動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バインド・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 識別子が続きます。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固有ですが、複数のスキーマ間で固有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。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。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、意図的に指定変更を行う場合を除き、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、

```
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みなされます。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、次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、重複する関数とみなされ、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することができます。ユーザー定義関数に関する 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、 *アプリケーション開発の手引き* の言語別の項を参照してください。

- DECIMAL (および NUMERIC) は、LANGUAGE C と OLE では無効です (SQLSTATE 42815)。 DECIMAL の使用に代わる手法については、*アプリケーション開発の手引き* を参照してください。
- 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、パラメーターに効力範囲を指定してはなりません。
- 適切な変形関数が、関連する変形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)。 UDF での LOB ロケーターの使用については、*アプリケーション開発の手引き* で説明されています。

次の例は、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 以外のタイプ、または LOB に基づく特殊タイプに対して AS LOCATOR を指定すると、エラー (SQLSTATE 42601) が発生します。

関数が FENCED の場合、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、次の例で、

```
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 (キャスト可能の定義については、 102ページの『データ・タイプ間のキャスト』を参照してください)。

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実行時検査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。 UDF での LOB ロケーターの使用については、*アプリケーション開発の手引き* 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固有名を指定します。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、この関数を除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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*関数名* と同じであっても構いません。

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、固有の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。生成される固有の名前は、 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。

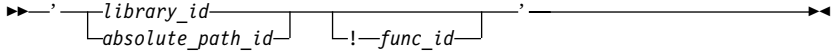
NAME *'string'*

この文節は、定義している関数を実装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、関数を 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) 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。

  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、.../sqllib/function ディレクトリー (UNIX 系システム)、または ...¥*instance_name*¥ 関数ディレクトリー (DB2INSTPROF レジストリー変数で指定した OS/2、Windows 32 ビット オペレーティング・システム) のライブラリーを探します。ここで、データベース・マネージャーの実行に使用される制御 sqllib ディレクトリーは、データベース・マネージャーが位置指定します。たとえば、UNIX 系システムの制御 sqllib ディレクトリーは、 /u/$DB2INSTANCE/sqllib です。

  UNIX 系システムの *library_id* が 'myfunc' で、データベース・マネージャーが /u/production から実行されている場合、ライブラリー /u/production/sqllib/function/myfunc から関数を探します。

  OS/2 および Windows 32 ビット オペレーティング・システムの場合、*library_id* が関数ディレクトリーにないと、データベース・マネージャーは LIBPATH または PATH を探します。
  OS/2 では、library_id に 9 文字以上を指定しないようにします。

absolute_path_id
関数を含んでいるファイルの全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。

OS/2 および Windows 32 ビット 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 ファイルをロードします。 OS/2 では、この指定の最後の部分 (すなわち dll の名前) は、9 文字以上にしないでください。

! 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。

OS/2、および 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。

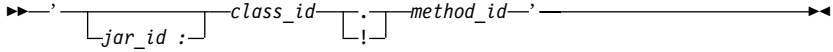
ストリングの形式が正しくない場合には、エラー (SQLSTATE 42878) になります。

すべての外部関数の本体は、データベースのすべての区分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、クラス識別子、およびメソッド識別子が含まれています。クラス識別子とメソッド識別子は、 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、識別子は、 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、メソッド識別子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。

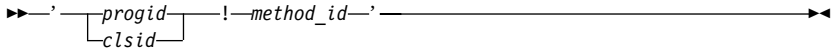
jar_id
jar の集合をデータベースへインストールしたときに、その jar の集合に付けられた jar 識別子を指定します。これは、単純識別子またはスキーマ修飾識別子のいずれかにすることができます。たとえば、'myJar' や 'mySchema.myJar' のようになります。

class_id
Java オブジェクトのクラス識別子を指定します。クラスがパッケージの一部である場合、クラス識別子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。 OS/2 および 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 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 のプログラム識別子 (progid) またはクラス識別子 (clsid)、およびメソッド識別子です。プログラム識別子またはクラス識別子、およびメソッド識別子は、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、メソッド識別子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識別子を指定します。

progid は、データベース・マネージャーには解釈されず、実行時に 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識別子を指定します。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、実行時に 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 識別子です。 SQL 識別子は、ストリングの *library-id* として使用されます。区切られた識別子でない場合、識別子は大文字に変換されます。識別子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。

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。

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。

DB2SQL

C 言語の呼び出しとリンクの規則、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、この外部関数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。

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。

パラメーターの受け渡しの詳細については、*アプリケーション開発の手引き* を参照してください。

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連続して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。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空間で実行しても “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、その内部資源 (データ・バッファーなど) を隔離して、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。

警告: 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。

FENCED を使用すれば NOT FENCED よりもデータベースの保全性を大幅に確保できるものの、適切にコード化、検討、および検査して FENCED UDF を使用しなければ、 DB2 に障害が発生することもあります。

ほとんどのユーザー定義関数は、FENCED または NOT FENCED のどちらでも実行できるはずです。 LANGUAGE OLE 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 の場合、AS LOCATOR 文節は指定できません (SQLSTATE 42613)。

FENCED から NOT FENCED に変更するには、関数を削除して再作成し、関数を再登録する必要があります。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、または特殊権限 (CREATE_NOT_FENCED) が必要です。

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ヌル値の場合に、外部関数を呼び出さないようにすることができます。パラメーターがない

ものとしてユーザー定義関数を定義すると、このヌル引き数条件が引き起こされることはなく、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ヌル値の場合、このユーザー定義関数は呼び出されず、結果はヌル値になります。

CALLED ON NULL INPUT が指定されると、引き数がヌル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、ヌル値を戻す場合も、通常の (ヌル値以外の) 値を戻す場合もあります。ただし、ヌル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、上位互換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 の同義語として使えます。

NO SQL
この文節は必須で、関数が SQL ステートメントを発行してはならないことを指定します。関数が SQL ステートメントを発行すると、実行時にエラー (SQLSTATE 38502) になります。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
この文節は任意指定であり、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処置を行うか否かを指定します。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 or **SCRATCHPAD** *length*
この文節は任意選択であり、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。

- その効力範囲は、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、次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、その効力範囲は上記とは異なります。関数が複数の区分で実行される場合、関数が処理されるそれぞれの区分において、 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、区画内並行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資源 (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資源が獲得される場合、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資源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ス指定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。

NO FINAL CALL または **FINAL CALL**

この文節は任意選択であり、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、獲得したシステム資源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資源を獲得し、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。外部関数が資源を解放できるようにする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、次の時点で行われます。
    - ステートメント終了時。これは、カーソルの関係する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” の引き数は外部関数に渡されず、最終呼び出しは行われません。

  エラー発生時の、これらの呼び出しのスカラー UDF 処理については、*アプリケーション開発の手引き* に説明があります。

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**

この文節は任意選択で、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。一般には、ほとんどのスカラー関数は

並列化可能ですが、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、デフォルトの指定を使用します。
- 1 つの区分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、デフォルトの指定を使用します。
- 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区分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**

この文節は任意選択で、 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、固有のアプリケーション 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 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、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のもとで、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。

構造の詳細、および構造がユーザー定義関数にどのようにして渡されるかについては、 *アプリケーション開発の手引き* を参照してください。

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変形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、変形が必要になります。この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 変形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)。

変形関数は、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変形を適切に実行する 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、フィルター処理や索引拡張の活用を定義します。述部仕様では、検索条件の任意選択の SELECTIVITY 文節を指定できます。 PREDICATES 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)。

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最適化プログラム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の数を減らし、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 で定義されていなければなりません (SQLSTATE 429B4)。
- NOT DETERMINISTIC または EXTERNAL ACTION で定義していてはなりません (SQLSTATE 42845)。

- 任意のパラメーターのデータ・タイプとして構造化データ・タイプがあってはなりません (SQLSTATE 428E3)。
- 副照会が含まれていてはなりません (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。

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 つの規則に適合することも必要です。

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)。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。

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しなければなりません。

EXACT
述部評価の見地から索引検索が厳密であ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で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でなければなりません。

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、ソース・タイプが実際には 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 (101ページの『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 を使用します。
  - NUMERIC- 代わりに DECIMAL を使用します。
  - LONG VARCHAR- 代わりに CLOB (または BLOB) を使用します。
- 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、859ページの『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)。
- 外部ユーザー定義関数の作成、コンパイル、およびリンクについては、 *アプリケーション開発の手引き* を参照してください。
- 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 特権は PUBLIC に与えられます。

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 DB2SQL
    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蓄積します。 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 DB2SQL
    DETERMINISTIC   FENCED   NOT NULL CALL   NO SQL   NO EXTERNAL ACTION
    SCRATCHPAD   NO FINAL CALL
    DISALLOW PARALLEL
```

例 3: 次の例は、以下の式を計算する C 言語のユーザー定義関数です。

```
output = 2 * input - 4
```

入力がヌル値の場合には (そしてその場合のみ)、ヌル値を戻します。これは、CREATE FUNCTION ステートメントで NOT NULL CALL を指定して、より簡単に (つまりヌル値検査を行わずに) 作成することができます。ユーザー定義関数プログラムの他の例が、*アプリケーション開発の手引き* に示されています。 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 DB2SQL
   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、分離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 DB2SQL
     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、以下の追加により、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します。

- **PREDICATES WHEN = 1** は、DML ステートメントの WHERE 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、これらが重なるかどうかをすばやく判別します。 2 つの入力の形の最小の境界長方形が重なら

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 DB2SQL
     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、次のようにします。

  ```
  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** 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。式の識別子は、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。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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 (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)

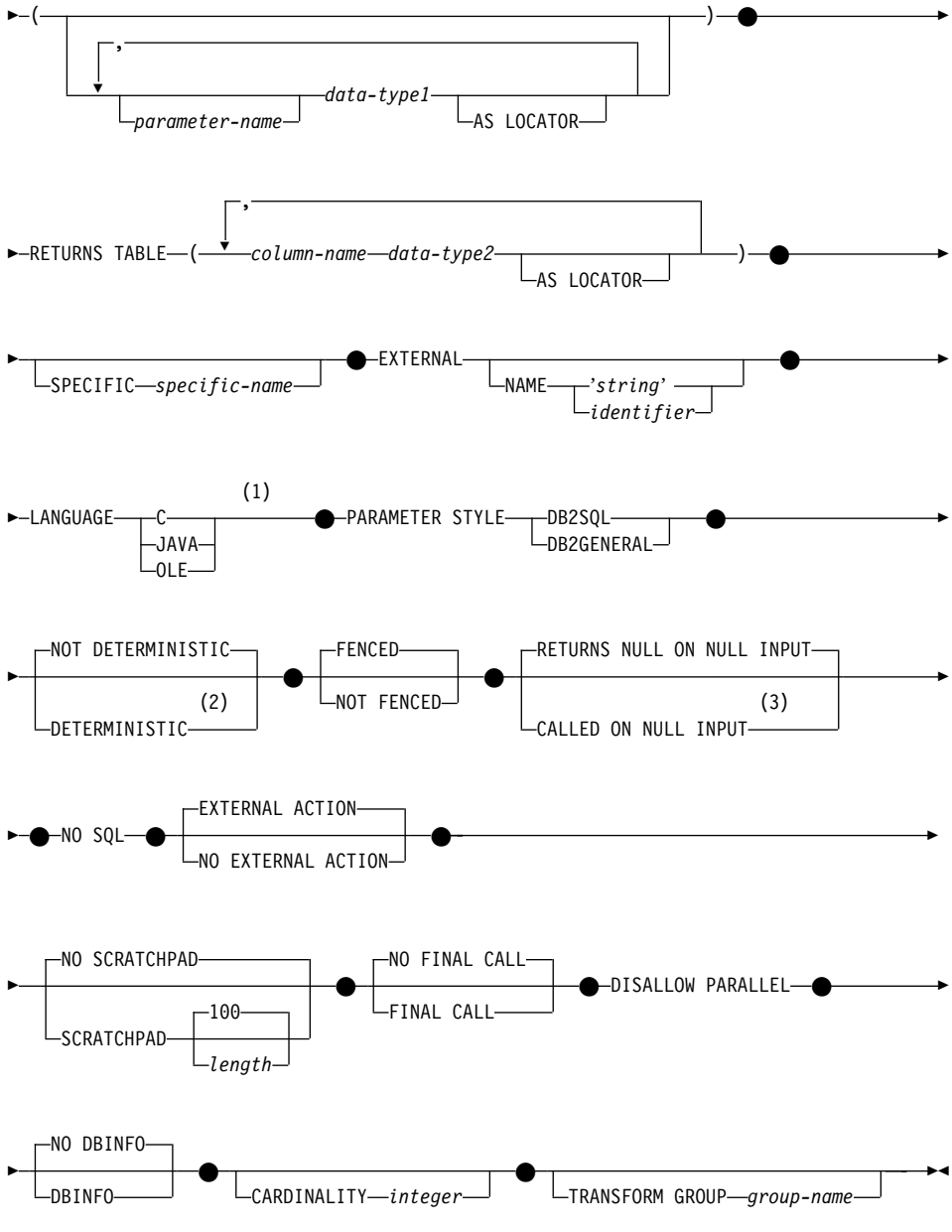
非分離の関数を作成するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

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

分離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、681ページの『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、701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』 を参照してください。

2 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。

3 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。

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 識別子です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バインド・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 識別子が続きます。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固有ですが、複数のスキーマ間で固有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 (SQLSTATE 42939)。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

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

この関数の他のパラメーターすべての名前と異なる、パラメーターの任意選択名を指定します。

(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、

```
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みなされます。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、次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、重複する関数とみなされ、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* 定義に指定が可能で、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・タイプ指定と省略形を指定できます。ユーザー定義関数に関する 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、 *アプリケーション開発の手引き* の言語別の項を参照してください。
- DECIMAL (および NUMERIC) は、LANGUAGE C と OLE では無効です (SQLSTATE 42815)。 DECIMAL の使用に代わる手法については、*アプリケーション開発の手引き* を参照してください。

- 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、パラメーターに効力範囲を指定してはなりません (SQLSTATE 42997)。
- 適切な変形関数が、関連する変形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)。 UDF での LOB ロケーターの使用については、*アプリケーション開発の手引き* で説明されています。

次の例は、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 以外のタイプ、または LOB に基づく特殊タイプに対して AS LOCATOR を指定すると、エラー (SQLSTATE 42601) が発生します。

関数が FENCED の場合、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。特定の言語において、構造タイプ以外であれば、 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、 638 ページ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固有名を指定します。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、この関数を除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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*関数名* と同じであっても構いません。

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、固有の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。生成される固有の名前は、 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。

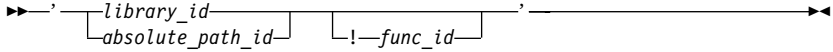
NAME *'string'*

この文節は、定義する関数を実現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、.../sqllib/function ディレクトリー (UNIX 系システム)、または ...¥*instance_name*¥ 関数ディレクトリー (DB2INSTPROF レジストリー変数で指定した OS/2、Windows 32 ビット オペレーティング・システム) のライブラリーを探します。ここで、データベース・マネージャーの実行に使用される制御 sqllib ディレクトリーは、データベース・マネージャーが位置指定します。たとえば、UNIX 系システムの制御 sqllib ディレクトリーは、 /u/$DB2INSTANCE/sqllib です。

  UNIX 系システムの *library_id* が 'myfunc' で、データベース・マネージャーが /u/production から実行されている場合、ライブラリー /u/production/sqllib/function/myfunc から関数を探索します。

OS/2 および Windows 32 ビット オペレーティング・システムの場合、*library_id* が関数ディレクトリーにないと、データベース・マネージャーは LIBPATH または PATH を探します。

OS/2 では、library_id に 9 文字以上を指定しないようにします。

absolute_path_id
関数の全パス名を指定します。

たとえば、UNIX 系システムで '/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 の中から myfunc 関数を探します。

OS/2 および Windows 32 ビット 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 myfunc.dll ファイルをロードします。

OS/2 では、この指定の最後の部分 (すなわち dll の名前) は、9 文字以上にしないでください。

! 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。

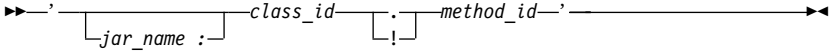
OS/2、および Windows 32 ビット オペレーティング・システムの場合 'mymod!func8' を指定すると、データベース・マネージャーは mymod.dll ファイルをロードして、そのダイナミック・リンク・ライブラリー (DLL) の func8() 関数を呼び出します。

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区分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、クラス識別子、およびメソッド識別子が含まれています。クラス識別子とメソッド識別子は、 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、識別子は、 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、メソッド識別子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_name
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 識別子を指定します。これは、単純識別子またはスキーマ修飾識別子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。

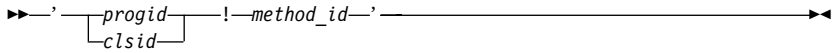
class_id
Java オブジェクトのクラス識別子を指定します。クラスがパッケージの一部である場合、クラス識別子の部分に完全なパッケージ接頭部 (例: 'myPacks.UserFuncs') が含まれている必要があります。 Java 仮想マシンは、ディレクトリー '.../myPacks/UserFuncs/' の中のクラスを探します。 OS/2 および Windows 32 ビット オペレーティング・システム では、 Java 仮想マシンはディレクトリー '...¥myPacks¥UserFuncs¥' を探索します。

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 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 のプログラム識別子 (progid) またはクラス識別子 (clsid)、およびメソッド識別子です。プログラム識別子またはクラス識別子、およびメソッド識別子は、 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、メソッド識別子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識別子を指定します。

progid は、データベース・マネージャーには解釈されず、実行時に 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識別子を指定します。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、実行時に 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実装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 識別子です。 SQL 識別子は、ストリングの *library-id* として使用されます。区切られた識別子でない場合、識別子は大文字に変換されます。識別子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。

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、681ページの『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。

DB2SQL

C 言語の呼び出しとリンクの規則に準拠する外部関数との間で、パラメーターを渡し、値を戻すために用いる規則を指定します。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。

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。

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、同じ入力を指定して連続して呼び出した場合に常に同じ表を戻します。 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 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空間で実行しても “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、その内部資源 (データ・バッファーなど) を隔離して、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。

警告: 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。

FENCED を使用すればデータベースの保全性を大幅に確保できるものの、適切にコード化、検討、および検査して FENCED UDF を使用しなければ、 DB2 に障害が発生することがあります。

ほとんどのユーザー定義関数は、FENCED または NOT FENCED のどちらでも実行できるはずです。 LANGUAGE OLE を指定した関数には、FENCED のみを指定できます (SQLSTATE 42613)。

FENCED から NOT FENCED に変更するには、関数を削除して再作成し、関数を再登録する必要があります。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、または特殊権限 (CREATE_NOT_FENCED) が必要です。

関数が FENCED の場合、AS LOCATOR 文節は指定できません (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ヌル値の場合に、外部関数を呼び出さないようにすることができます。パラメーターがない

ものとしてユーザー定義関数を定義すると、このヌル引き数条件が引き起こされることはなく、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ヌル値の場合、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、空の表になります (行のない表)。

CALLED ON NULL INPUT が指定されると、引き数がヌル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、ヌル値を戻す場合も、通常の (ヌル値以外の) 値を戻す場合もあります。ただし、ヌル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、上位互換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 の同義語として使えます。

NO SQL
この文節は必須で、関数が SQL ステートメントを発行してはならないことを指定します。関数が SQL ステートメントを発行すると、実行時にエラー (SQLSTATE 38502) になります。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
この文節は任意指定であり、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処置を行うか否かを指定します。 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*
この文節は任意選択であり、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。
- その効力範囲は、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、次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資源が解放されます。

  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資源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資源 (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資源を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ス指定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。

NO FINAL CALL または FINAL CALL

この文節は任意選択であり、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。この引き数とその値の詳細については、*アプリケーション開発の手引き* を参照してください。

エラー発生時の、これらの呼び出しの表 UDF 処理については、 *アプリケーション開発の手引き* に説明があります。

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 つの区分で実行されます。

NO DBINFO または **DBINFO**

この文節は任意選択で、 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、固有のアプリケーション 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 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。

構造の詳細、および構造が UDF にどのようにして渡されるかについては、 *アプリケーション開発の手引き* を参照してください。

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の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場合 (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)、正しく機能するために ″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、GROUP BY および ORDER BY を使用する照会があります。このような UDF は書かないことをお勧めします。

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変形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、変形が必要になります。この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)。

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 (101ページの『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 を使用します。
  - NUMERIC- 代わりに DECIMAL を使用します。
  - LONG VARCHAR- 代わりに CLOB (または BLOB) を使用します。
- 外部ユーザー定義関数の作成、コンパイル、およびリンクについては、 *アプリケーション開発の手引き* を参照してください。
- 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 特権は PUBLIC に与えられます。

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 つの文書識別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文字列が入ります。

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 最適化プログラムにとって有用な 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 DB2SQL
          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。この表関数を実装するコードの例については、*アプリケーション開発の手引き* を参照してください。

```
CREATE FUNCTION MAIL()
    RETURNS TABLE (TIMERECIEVED DATE,
                   SUBJECT VARCHAR(15),
                   SIZE INTEGER,
                   TEXT VARCHAR(30))
    EXTERNAL NAME 'tfmail.header!list'
    LANGUAGE OLE
    PARAMETER STYLE DB2SQL
    NOT DETERMINISTIC
    FENCED
    CALLED ON NULL INPUT
    SCRATCHPAD
    FINAL CALL
    NO SQL
    EXTERNAL ACTION
    DISALLOW PARALLEL
```

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

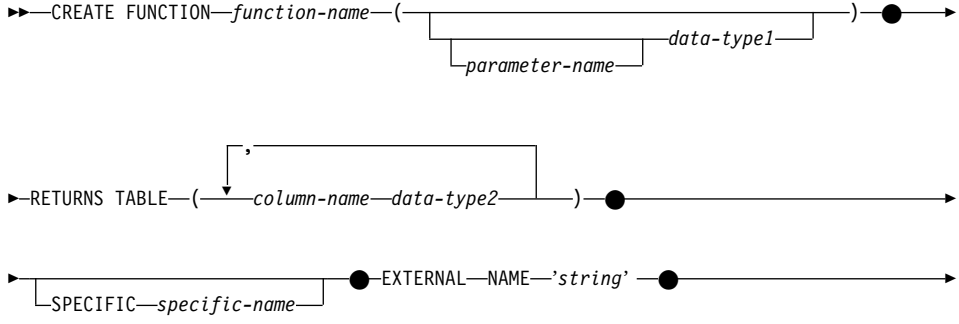
許可

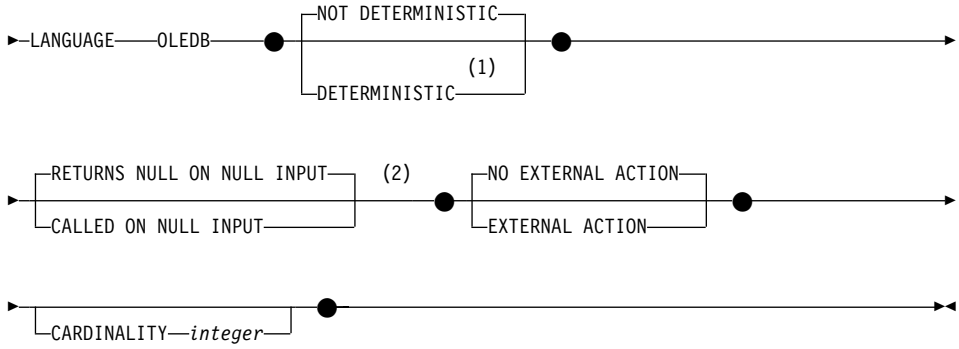
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

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





注:

1 DETERMINISTIC の代わりに NOT VARIANT を、 また NOT ?XPP:break DETERMINISTIC の代わりに VARIANT を指定することができます。

2 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。

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 識別子です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バインド・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 識別子が続きます。

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固有ですが、複数のスキーマ間で固有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 (SQLSTATE 42939)。それに含まれる名前は、SOME、ANY、ALL、NOT、AND、OR、BETWEEN、NULL、LIKE、EXISTS、IN、UNIQUE、OVERLAPS、SIMILAR、MATCH、および 209ページの『基本述部』 に記載されている比較演算子です。

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
パラメーターに任意選択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、任意の文字または漢字ストリング・データ・タイプで、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、

```
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みなされます。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 (SQL データ・タイプと OLE DB データ・タイプとの間のマッピングについては、 *アプリケーション開発の手引き* のそれぞれの言語の節を参照してください)。

SPECIFIC *specific-name*
定義する関数のインスタンスに対する固有名を指定します。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、この関数を除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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*関数名* と同じであっても構いません。

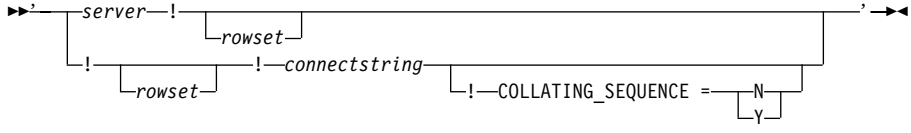
修飾子を指定しない場合、*function-name* に使用された修飾子が使用されます。修飾子を指定する場合は、 *function-name* の明示修飾子または暗黙修飾子と同じでなければなりません。そうでない場合、エラー (SQLSTATE 42882) になります。

specific-name の指定がない場合、固有の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。生成される固有の名前は、 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。詳細は、*アプリケーション開発の手引き* の『OLE DB 表関数』を参照してください。



server
766ページの『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。詳細は、*アプリケーション開発の手引き* のそれぞれの言語の節を参照してください。

COLLATING_SEQUENCE
データ・ソースが、DB2 ユニバーサル・データベース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、766ページの『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 ユニバーサル・データベース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 の消費者情報を展開し、OLE DB Provider からデータを取り出します。開発者側で表関数を実装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、同じ入力を指定して連続して呼び出した場合に常に同じ表を戻します。 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。

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ヌル値の場合に、外部関数を呼び出さないようにすることができます。パラメーターがないものとしてユーザー定義関数を定義すると、このヌル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ヌル値の場合、ユーザー定義関数は呼び出されず、結果は空表、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、引き数がヌル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、ヌル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、上位互換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 の同義語として使えます。

NO EXTERNAL ACTION または **EXTERNAL ACTION**
この文節は任意指定であり、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処置を行うか否かを指定します。 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の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場合 (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)、正しく機能するために ″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、GROUP BY および ORDER BY を使用する照会があります。このような UDF は書かないことをお勧めします。

注

- FENCED、FINAL CALL、SCRATCHPAD、PARAMETER STYLE DB2SQL、DISALLOW PARALLEL、 NO DBINFO、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。それぞれの説明については、663ページの『CREATE FUNCTION (外部表)』を参照してください。
- 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 (101ページの『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。
  - 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 を使用します。
  - NUMERIC- 代わりに DECIMAL を使用します。
  - LONG VARCHAR- 代わりに CLOB (または BLOB) を使用します。
- ユーザー定義の OLE DB 外部表関数の作成については、 *アプリケーション開発の手引き* を参照してください。
- 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 特権は PUBLIC 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。ステートメントの詳細については、766ページの『CREATE SERVER』、912ページの『CREATE WRAPPER』、および 893ページの『CREATE USER MAPPING』を参照してください。

```
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 コマンド・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;
```

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。
- 連合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、連合サーバーへ実行依頼する照会に、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、連合サーバーは、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。詳細については、54ページの『関数マッピング、関数テンプレート、および関数マッピング・オプション』 を参照してください。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

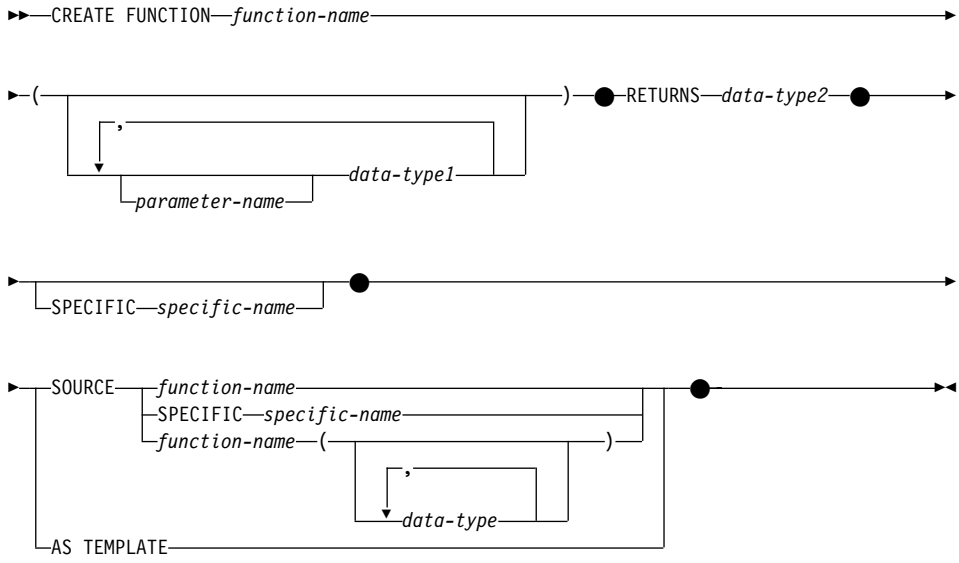
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (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。

SOURCE 文節で参照される関数に対する権限は必要ありません。

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 識別子です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バインド・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 識別子が続きます。

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固有ですが、複数のスキーマ間で固有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 (SQLSTATE 42939)。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

既存の関数と同一の機能をユーザー定義特殊タイプに対してサポートする目的で、その既存の関数に基づくユーザー定義関数を指定する場合は、該当の既存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みなされます。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、次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、重複する関数とみなされ、エラーになります。

parameter-name
この関数の他のパラメーターすべての名前と異なる、パラメーターの任意選択名を指定します。

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 (キャスト可能の定義については、 102 ページの『データ・タイプ間のキャスト』を参照してください)。 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。

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)。代わりに、CHAR() などのように、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。

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。

ソース関数の結果のタイプからキャスト可能である場合、任意の有効な SQL データ・タイプ (特殊タイプも同様) が有効です (キャスト可能の定義については、 102ページの『データ・タイプ間のキャスト』を参照してください)。

上記のソース関数からの派生関数のパラメーターの場合のように、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、 696 ページも参照してください。

SPECIFIC *specific-name*

定義する関数のインスタンスに対する固有名を指定します。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、この関数を除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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*関数名* と同じであっても構いません。

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、固有の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。生成される固有の名前は、 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) によって実装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 [71]、または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。スキーマにこの

71. COALESCE、NODENUMBER、NULLIF、PARTITION、TYPE_ID、TYPE_NAME、 TYPE_SCHEMA および VALUE は例外です。

function-name を名前とする特定の関数が、ちょうど 1 つだけ存在している場合にのみ有効です。この構文変数は、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、許可 ID の現行 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。この名前の関数の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) になります。指定したスキーマまたは見つかったスキーマに、この関数の特定インスタンスが複数個ある場合には、エラー (SQLSTATE 42725) になります。

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。この構文変数は、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、許可 ID の現行 SQL パスが関数を見つけるのに使用されます。この特定名の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) になります。

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数です。

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 (162ページの『関数解決』 を参照してください)。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み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。タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

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。

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。

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が、それ自体、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

エラーになると、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
  - SCRATCHPAD / NO SCRATCHPAD
  - FINAL CALL / NO FINAL CALL
  - RETURNS TABLE (...)
  - CARDINALITY ...
  - ALLOW PARALLEL / DISALLOW PARALLEL
  - DBINFO / NO DBINFO

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) になります。
- 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ことを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。キャスト可能の定義については、102ページの『データ・タイプ間のキャスト』 を参照してください。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) になります。

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) になります。

  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、ソース・タイプが実際には 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 (101ページの『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 特権は PUBLIC に与えられます。
- データ・ソース関数を認識する連合サーバーの場合、この関数は連合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。

  対となる部分は、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)。連合システムでの関数呼び出し時のオーバーヘッドの制御については、 *アプリケーション開発の手引き* を参照してください。

例

例 1: Pellow がオリジナルの CENTRE 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、 AVG 関数を使用してそれぞれの部門の平均の帽子サイズを計算したいとします。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: 連合システムで、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。連合サーバーは、この関数と連合データベース側の対の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、そのような対の関数は存在していません。それで、対の関数を関数テンプレートの形で指定すること、そ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、テンプレートを連合サーバーに登録します。マッピング用のコードについては、713ページの『例』 を参照してください。

```
CREATE FUNCTION NOVA.STATS (DOUBLE, DOUBLE)
     RETURNS DOUBLE
     AS TEMPLATE
```

例 4: 連合システムで、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。連合サーバーは、この関数と連合データベース側の対の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の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連合サーバーに登録します。マッピング用のコードについては、713ページの『例』 を参照してください。

```
CREATE FUNCTION BONUS ()
     RETURNS DECIMAL (8,2)
     AS TEMPLATE
```

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、変形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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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。

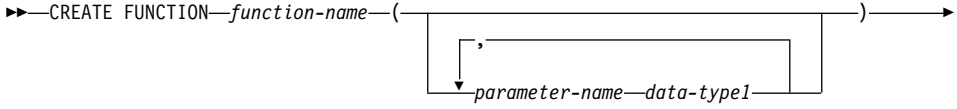
- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (関数のスキーマ名が既存のスキーマを指していない場合)
- スキーマに対する CREATEIN 特権 (関数のスキーマ名が既存のスキーマを指している場合)

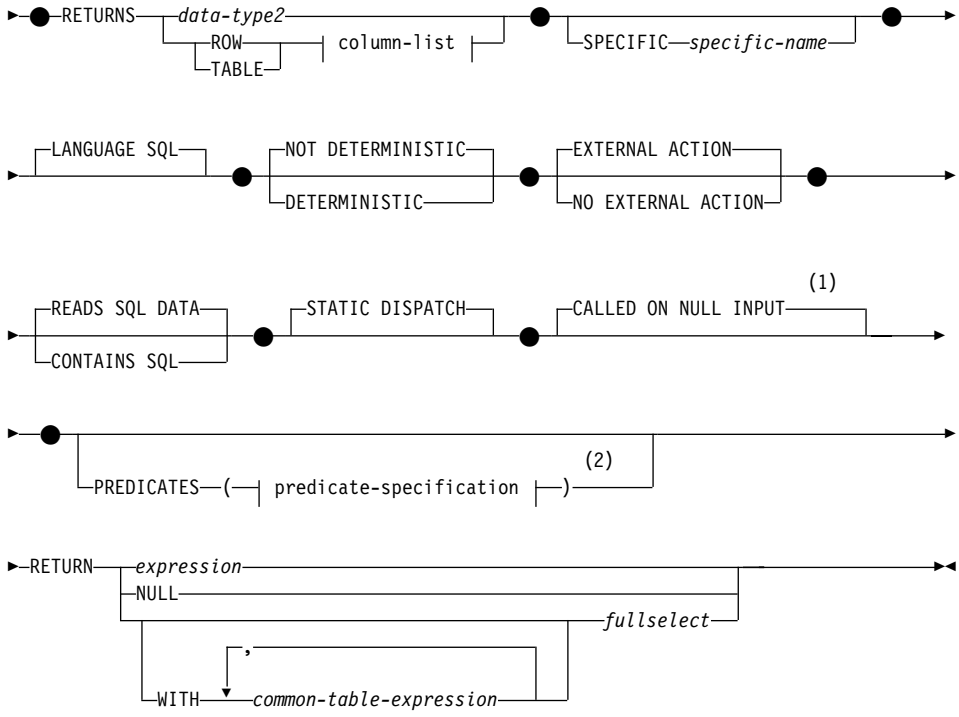
このステートメントの許可 ID に SYSADM 権限または DBADM 権限がなく、関数が表または視点を指定する場合、ステートメントの権限 ID が持つ特権 (GROUP 特権は考慮に入れない) には、表および視点それぞれに SELECT WITH GRANT OPTION が組み込まれていなければなりません。

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。

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:

注:

1 CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。

2 RETURNS がスカラー結果 (data-type2) を指定す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 識別子です (最大長 18)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バイ

ンド・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 識別子が続きます。

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固有ですが、複数のスキーマ間で固有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 (SQLSTATE 42939)。それに含まれる名前は、
SOME、ANY、ALL、NOT、AND、OR、BETWEEN、
NULL、LIKE、EXISTS、IN、UNIQUE、OVERLAPS、SIMILAR、MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

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の名前と異なる名前。

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 の効力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効力範囲を妨げることはしません。関数本体では、最初に参照タイプをキャストして効力範囲を指定してからでなければ、参照解除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、最初にこれに効力範囲を指定してからでなければ参照解除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、上記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固有名を指定します。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、この関数を除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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*関数名* と同じであっても構いません。

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、エラーになります (SQLSTATE 42882)。

specific-name の指定がない場合、固有の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。生成される固有の名前は、 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。サポートされている SQL は、現在のところ RETURN ステートメントに限られています。

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、同じ入力を指定して連続して呼び出した場合に常に同じ表を戻します。 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。

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**
この文節は任意選択であり、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処置を行うか否かを指定します。 NO EXTERNAL ACTION を指定すると、システムは関数に外部の影響が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)。

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、関数により実行で

きるように指示します (SQLSTATE 42985)。ニックネームまたは OLEDB 表関数を SQL ステートメントで参照することはできません (SQLSTATE 42997)。

CONTAINS SQL
SQL データを読み取ることも変更することもしない 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)。

STATIC DISPATCH
この任意選択の文節は、関数解決時に、 DB2 が関数のパラメーターの静的タイプ (宣言されたタイプ) に基づいて関数を選択することを指定します。

CALLED ON NULL INPUT
なんらかの引き数がヌル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、ヌル値を戻す場合も、ヌル値以外の値を戻す場合もあります。ヌル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。

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使用できることを示し、任意選択の SELECTIVITY 文節を使用して述部の検索条件を指定できます。 PREDICATES 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)。

predicate-specification
述部指定の詳細については、635ページの『CREATE FUNCTION (外部スカラー)』 を参照してください。

RETURN
関数の戻り値を指定します。パラメーター名は、RETURN ステートメントで参照できます。参照をあいまいにしないために、パラメーター名を関数名で修飾することができます。

expression
関数に戻される式を指定します。式の結果データ・タイプは、RETURNS 文節で定義されるデータ・タイプに割り当て可能 (保管割り当て規則を使用) でなければなりません (SQLSTATE 42866)。スカラー式 (スカラー全選択以外) は、表関数に指定することはできません (SQLSTATE 428F1)。

NULL
関数が RETURNS 文節で定義されるデータ・タイプのヌル値を戻すように指定します。

WITH *common-table-expression*
後に続く全選択で使用する共通表式 (*common-table-expression*) を定義します。 468ページの『共通表式』 を参照してください。

fullselect
関数に戻される行 (単数または複数) を指定します。全選択での列の数は、関数結果での列の数と一致しなければならず (SQLSTATE 42811)、全選択の静的列タイプは、列への割り当ての規則を使用して、関数結果の宣言された列タイプに割り当て可能でなければなりません (SQLSTATE 42866)。

この関数がスカラー関数である場合、全選択は 1 つの列 (SQLSTATE 42823) と、最大で 1 つの行を戻さなければなりません (SQLSTATE 21000)。

この関数が行関数の場合、多くて 1 つの行を戻さなければなりません (SQLSTATE 21505)。

この関数が表関数の場合、1 つまたは複数の列で 0 以上の行を戻せます。

注

- 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。
- 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。

– 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みなされます。

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 の変形関数を定義します。

```
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
```

この関数の定義者は EMPLOYEE 表で SELECT WITH GRANT OPTION 特権を持っていなければならないことと、すべてのユーザーが表関数 DEPTEMPLOYEES を呼び出して、各部門番号の結果列でデータへのアクセスを効果的に行えることに注意してください。

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。

- 連合データベース関数 (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。マッピングを行うことにより、連合データベースの関数またはテンプレートを、 (1) 指定したデータ・ソースあるいは (2) 特定のデータ・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データ・ソースすべてなど) のいずれかで、特定の関数と関連付けることができます。
- 連合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。

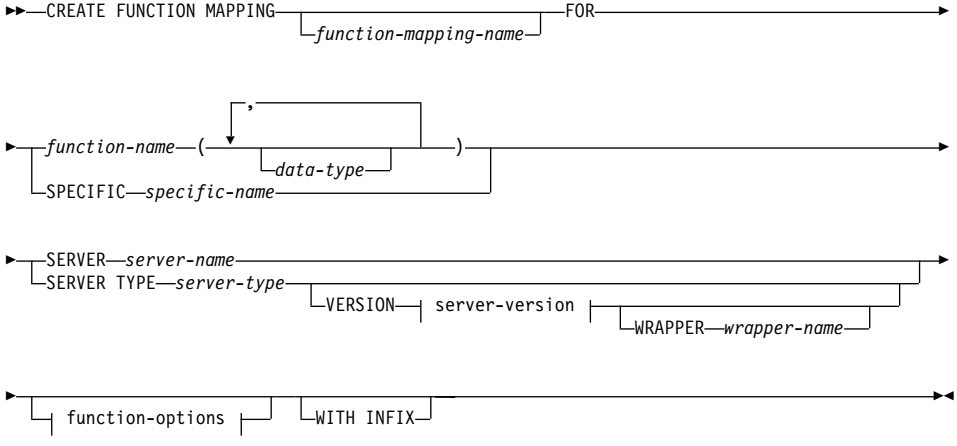
呼び出し

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、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

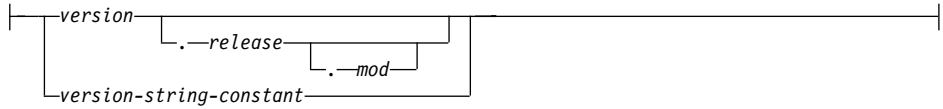
許可

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

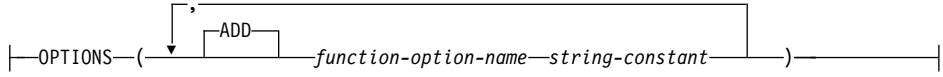
構文



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固有の名前が割り当てられます。

function-name
マップ元となる関数または関数テンプレートの修飾名または非修飾名。

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* はそのようなパラメーターのデータ・タイプを指定します。 *data-type* は、LONG VARCHAR、LONG VARGRAPHIC、DATALINK、ラージ・オブジェクト (LOB) タイプ、またはユーザー定義タイプにはできません。

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。連合データベースにおいて、関数または関数テンプレートに、固有の *function-name* がない場合、*specific-name* を指定してください。

SERVER *server-name*
マップ先の関数を含むデータ・ソースを指定します。

TYPE *server-type*
マップ先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。 *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-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、連合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。
function-option-name とその設定の詳細は、 1354ページの『関数マッピング・オプション』を参照してください。

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。

注

- 連合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。

  - 連合データベース関数またはテンプレートに、データ・ソース関数と同じ番号の入力パラメーターが付けられている場合。
  - 連合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、最適化プログラムは、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、この 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は、使用不可にすれば、操作できないよう設定できます (除去することはできない)。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。
  - ステートメントが 1 つのデータ・ソースを参照していて、UOW にはすでに、このデータ・ソース内の表または視点のニックネームを参照する SELECT ステートメントが含まれる場合。
  - ステートメントがデータ・ソースの 1 つのカテゴリーを参照していて (たとえば、特定のタイプおよびバージョンの全データ・ソース)、UOW にはすでに、これらのデータ・ソースの 1 つに含まれる表または視点のニックネームを参照する SELECT ステートメントがある場合。

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: 連合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、オーバーヘッドを少なくするために最適化プログラムで設定したところに応じて、 WEEK 関数またはその Oracle 側の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ます。 WEEK が呼び出されることを毎回確認するには、 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' )
```

CREATE INDEX

CREATE INDEX ステートメントは、以下のものを作成するときに使います。

- DB2 表での索引
- 索引指定: データ・ソース表に索引があることを最適化プログラムに通知するメタデータ

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、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

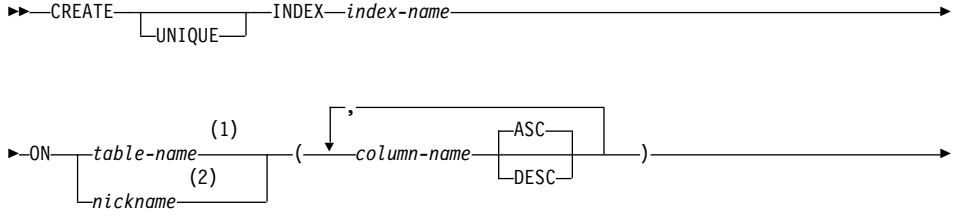
許可

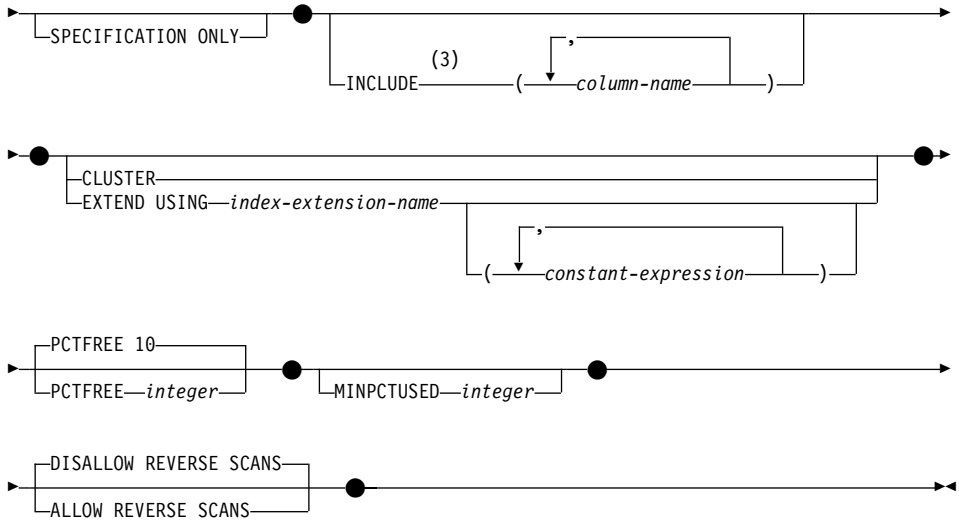
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- 以下のいずれか
  - その表に対する CONTROL 特権
  - その表に対する 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)。

構文





注:

1. 連合システムでは、*table-name* には、連合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* を指定する場合、 CREATE INDEX ステートメントにより、索引指定が作成されます。
   INCLUDE、 CLUSTER、 PCTFREE、 MINPCTUSED、 DISALLOW REVERSE SCANS、 および ALLOW REVERSE SCANS 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。詳細については、1397ページの『付録J. トリガーと制約の相互作用』を参照してください。

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、ヌル値は他の値と同様に扱われます。たとえば、キーが、ヌル値可能の単一列である場合、その列では 1 つのヌル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、しかも表に区分化キーがある場合、索引キーの列は区分化キーのスーパーセットである必要があります。つまり、固有索引キーに対して指定される列は、区分化キーのすべての列を含んでいる必要があります (SQLSTATE 42997)。

ON *nickname* が指定されている場合、索引キーのデータ・ソース表の各行に固有の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。

table-name を宣言された一時表 (SQLSTATE 42995) にすることはできません。

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を識別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)。

ON *table-name*または *nickname*

table-name には、索引を作成する表を指定します。表は、視点であってはならず、カタログに記述された基礎表または要約表でなければなりません。これをカタログ表 (SQLSTATE 42832) または宣言された一時表 (SQLSTATE 42995) にすることはできません。 UNIQUE が指定されており、 *table-name* がタイプ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)。 UNIQUE が指定されている場合は、*table-name* を要約表とすることはできません (SQLSTATE 42809)。

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・ソース視点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、連合サーバーがデータ・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* がタイプ付き表である場合は、列を 15 個まで指定できます。 *table-name* が副表であるならば、副表内に

は少なくとも 1 つの *column-name* を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* がタイプ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・タイプやヌル値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、 820ページの『Byte Counts』を参照してください。

それぞれの列の長さは、どれも 255 バイトを超えてはなりません。 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 255 バイトの限界内に収まる場合でも同じです (SQLSTATE 42962)。構造タイプ列は、EXTEND USING 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)。

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で索引のみ

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、固有キーと索引にあるすべての列にも適用されます。

column-name
索引には組み込まれているものの、固有索引キーの一部ではない列を指定します。固有索引キーの列に定義された規則と同様な次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化索引として指定します。クラスター化索引のクラスター係数は、データが関連する表に挿入される時に、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、表用のクラスター化索引が 1 つだけしかないために、それが表の既存の索引の定義に使用されていて、 CLUSTER は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、クラスター化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 (42601)。

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)。

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)。

PCTFREE *integer*
索引を構築する際に、各索引ページに残す空き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* パーセントを空き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、非葉ページには 10 パーセントの空き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)。

MINPCTUSED *integer*
索引をオンラインで再編成するかどうか、そして索引の葉ページで使用されるスペースの最小パーセンテージについて、その限界値を指定します。索引の葉ページからキーを削除した後、そのページで使うスペースのパーセンテージが *integer* のパーセンテージを下回る場合、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。

MINPCTUSED は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。

DISALLOW REVERSE SCANS
索引において、前方向走査、すなわち INDEX CREATE の実行時に定義した順序での走査だけをサポートすることを指定します。これはデフォルト値です。

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
索引が前方向走査と反対方向走査の両方、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)。

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固有でないか、または以前に存在していた索引が固有であり、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固有である場合、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

- 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 コマンドを発行することをお勧めします。 (RUNSTATS については、*コマンド解説書* を参照してください。) 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 コマンドを発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。
- 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 特権は PUBLIC に与えられます。
- 最適化プログラムは、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。詳細については、1098ページの『SET CURRENT EXPLAIN MODE』を参照してください。
- 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。
- 最適化プログラム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。
- 索引指定の詳細は、56ページの『索引指定』を参照してください。

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、最適化プログラムは、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、最適化プログラムは、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、索引グリッド・サイズを作成します。索引拡張の定義については、723ページの『CREATE INDEX EXTENSION』 を参照してください。

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
    EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

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

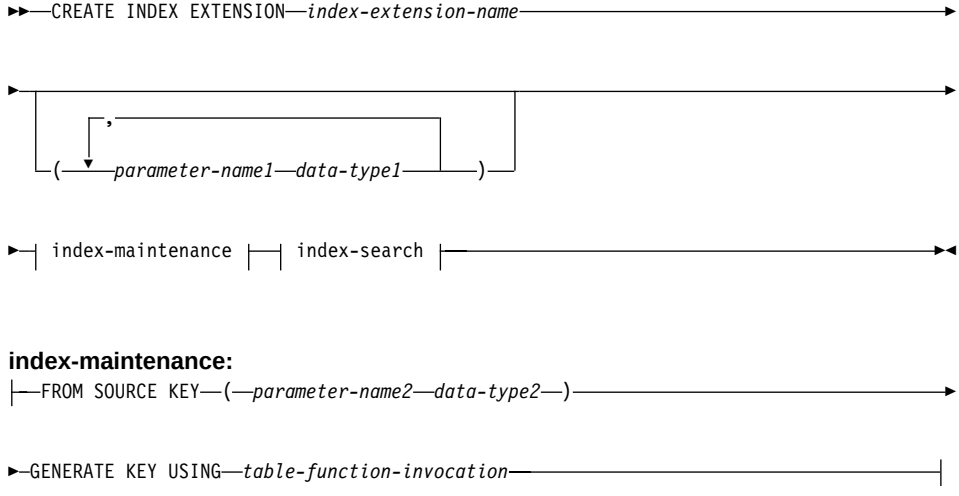
このステートメントは、アプリケーション・プログラムに組み込むか、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

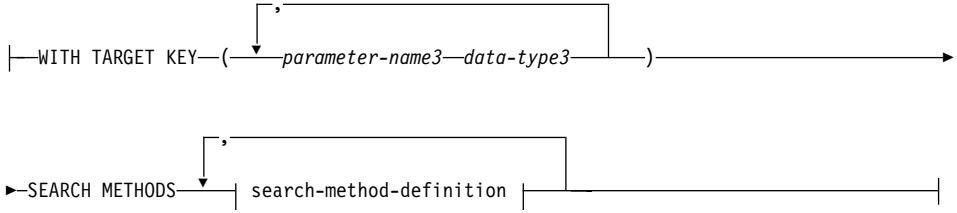
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- データベースに対する IMPLICIT_SCHEMA 権限 (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)

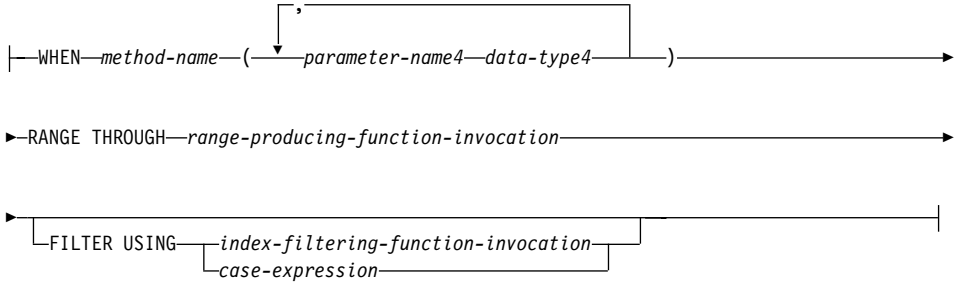
構文



index-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 つの部分からなる *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 (SQLSTATE 42939) になります。

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固有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)。定数についての詳細は、130ページの『定数』を参照してください。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・キー列に、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* は構造タイプか、あるいは LOB、DATALINK、LONG VARCHAR、LONG VARGRAPHIC を基にしたのではない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。

1. 表関数に解決されること (SQLSTATE 428E4)
2. LANGUAGE SQL で定義されていないこと (SQLSTATE 428E4)
3. NOT DETERMINISTIC (SQLSTATE 428E4) または EXTERNAL ACTION (SQLSTATE 428E4) で定義されていないこと
4. 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。
5. 副照会が含まれていないこと (SQLSTATE 428E3)
6. 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)。

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、および任意選択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 識別子です。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。

1. 表関数に解決されること (SQLSTATE 428E4)
2. 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
3. LANGUAGE SQL で定義されていないこと (SQLSTATE 428E4)
4. NOT DETERMINISTIC または EXTERNAL ACTION で定義されていないこと (SQLSTATE 428E4)
5. 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)。
   - キー変形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。
- 対応するキー変形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変形関数のデータ・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ヌル値を戻す場合、意味体系は未定義です。

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。

1. LANGUAGE SQL で定義されていないこと (SQLSTATE 429B4)
2. NOT DETERMINISTIC または EXTERNAL ACTION で定義されていないこと (SQLSTATE 42845)
3. 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)
4. 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。

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 特権は PUBLIC に与えられます。

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
```

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

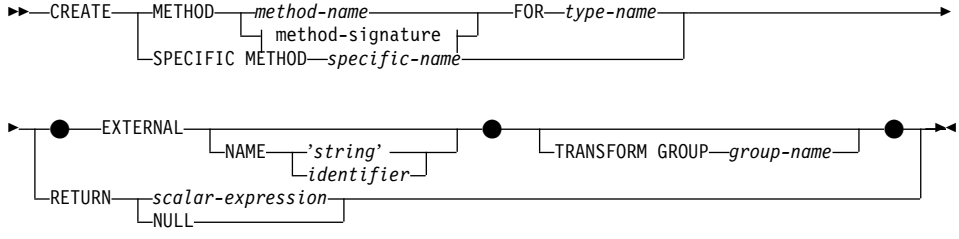
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- CREATE METHOD 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

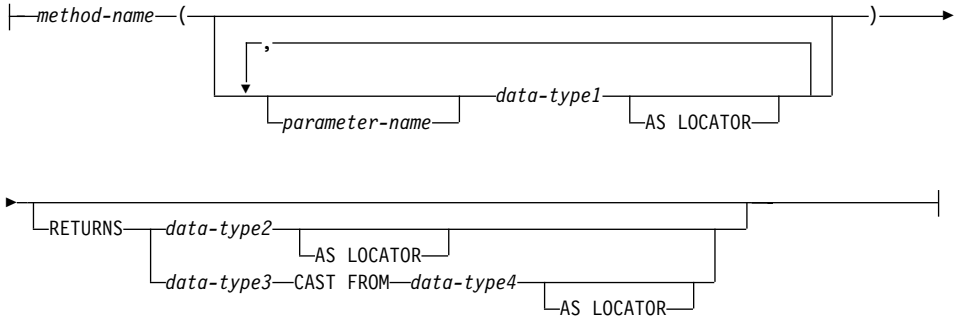
このステートメントの許可 ID に SYSADM 権限または DBADM 権限がなく、メソッドが RETURN ステートメントで表または視点を指定する場合、ステートメントの権限 ID が持つ特権 (グループ特権は考慮に入れない) には、識別される表および視点それぞれに対する SELECT WITH GRANT OPTION が組み込まれていなければなりません。

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

構文



method-signature:



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

メソッド本体が定義されてい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

メソッド本体が定義されてい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
この分節は、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 で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 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 特殊レジスターは、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 / バインド・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。

'ストリング *(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。特定の言語の規則についての詳細は、635ページの『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 識別子です。 SQL 識別子は、ストリングのライブラリー ID として使用されます。区切られた識別子でない場合、識別子は大文字に変換されます。識別子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変形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、変形が必要です。

ここで、変形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 変形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)。

RETURN *scalar-expression* **または NULL**

RETURN ステートメントは、メソッドにより戻される値を指定する SQL 制御ステートメントです。

scalar-expression

CREATE TYPE のメソッド指定で LANGUAGE SQL を指定する際の、メソッド本体を指定する式です。パラメーター名は、式で参照できます。メソッドのサブジェクトは、SELF という名前の暗黙の最初のパラメーターの形式で、メソッド・インプリメンテーションに渡されます。式の結果データ・タイプは、 CREATE TYPE におけるメソッド指定の RETURNS 文節で定義されるデータ・タイプに割り当て可能 (保管割り当て規則を使用) でなければなりません (SQLSTATE 42866)。

式は、メソッド指定の以下の部分に適合し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)

NULL

関数がヌル値を戻すように指定します。このヌル値は、 CREATE TYPE ステートメントで作成されるメソッド指定の RETURNS 文節で定義されるデータ・タイプです。

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)。

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
```

CREATE NICKNAME

CREATE NICKNAME ステートメントは、データ・ソース表または視点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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

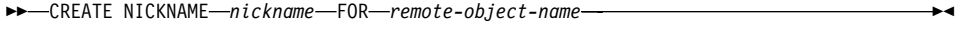
許可

このステートメントの許可 ID 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
- 連合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)

さらに、データ・ソースでのユーザーの許可 ID には、ニックネームを作成する対象となる表または視点についてのメタデータを、データ・ソース・カタログから選ぶための特権がなければなりません。

構文



説明

nickname

remote-object-name に示される表または視点のための、連合サーバーの識別子を指定します。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、視点、別名、またはニックネームを指定するものであってはなりません。スキーマ名を、SYS で始めることはできません (SQLSTATE 42939)。

remote-object-name

3 つの部分で構成される識別子を指定します。次のような形式になります。

data-source-name.remote-schema-name.remote-table-name

それぞれの意味は以下のとおりです。

data-source-name
ニックネームを作成する対象となる表または視点を含むデータ・ソースを指定します。 *data-source-name* は、 CREATE SERVER ステートメント内のデータ・ソースに割り当てられた名前と同じです。

remote-schema-name
表または視点が属するスキーマを指定します。

remote-table-name
以下のいずれかの識別子を指定します。

- DB2 ファミリーの表または視点の名前 (または別名)
- Oracle 表または視点の名前
- *table-name* を宣言済みの一時表にすることはできません (SQLSTATE 42995)。

注

- ニックネームで示されている表または視点は、 *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。
- 連合サーバーでは、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK、ラージ・オブジェクト (LOB) タイプ、およびユーザー定義のタイプなどの DB2 データ・タイプに対応する、データ・ソースのデータ・タイプをサポートしていません。データ・ソースの表または視点のためにニックネームを定義する場合、連合データベースに対して定義し、そこから照会できる列は、表または視点内にある、サポート済みのデータ・タイプの列だけです。サポートされないデータ・タイプの列を持つ表または視点に対して、 CREATE NICKNAME ステートメントを実行すると、エラーが発生します。
- データ・タイプはデータ・ソース間で非互換なので、連合サーバー側は、リモート・カタログ・データをローカルに保管するために、必要に応じて多少の調整を行います。詳細については、*アプリケーション開発の手引き* を参照してください。
- DB2 索引名に許可されている最大長は 18 文字です。この長さより長い名前の索引を持つ表にニックネームを作成する場合、その名前の全体がカタログ化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固有でない場合、 DB2 は最後の文字を 0 に置き換えて固有のストリングにしようと

します。その結果も固有でない場合は、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、固有の名前が生成されるまで繰り返してゆきます。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ため、DB2 はこの名前を新しい名前として受け入れます。

- 表または視点にニックネームを作成すると、 DB2 は表または視点の列の名前をカタログに保管します。この名前の長さが DB2 列名に許可されている最大長 (30 文字) を超える場合、 DB2 はこの名前を 30 文字に切り捨てます。切り捨てられた名前が、表または視点にある他の列の名前と同じで固有でない場合、 DB2 は前の段落で説明されている手順に従い、名前を固有のものに変更します。

例

例 1: HEDGES というスキーマにある、 DEPARTMENT という視点のニックネームを作成します。この視点は、OS390A という DB2 ユニバーサル・データベース (OS/390 版) 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視点から、すべてのレコードを選択します。この視点は、ニックネームで参照する必要があります。 (パススルー・セッションに限り、本来の名前で参照できます。)

```
SELECT * FROM OS390A.HEDGES.DEPARTMENT   無効
SELECT * FROM DEPT                       ニックネーム DEPT の作成後に有効
```

CREATE NODEGROUP

CREATE NODEGROUP ステートメントは、データベースに新たなノードグループを作成し、ノードグループに区分またはノードを割り当て、カタログにそのノードグループ定義を記録します。

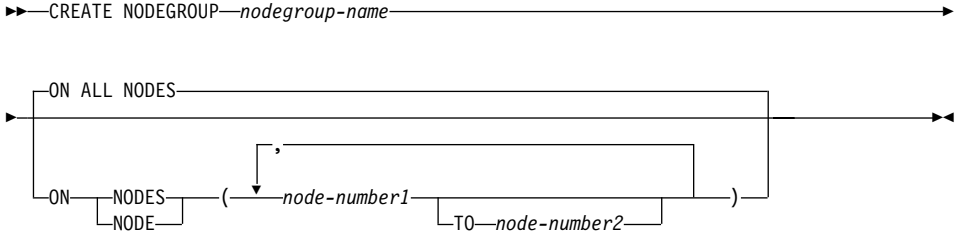
呼び出し

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、あるいは対話式に使用することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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

nodegroup-name

ノード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 識別子です (通常識別子または区切り識別子)。*nodegroup-name* (ノードグループ名) は、すでにカタログに存在するノードグループを指定するものであってはなりません (SQLSTATE 42710)。*nodegroup-name* は、文字 ″SYS″ または ″IBM″ で始めることはできません (SQLSTATE 42939)。

ON ALL NODES

ノード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区分にわたってノードグループを定義することを指定します。

データベース・システムに区分が追加された場合、 ALTER NODEGROUP ステートメントを使用して、この新しい区分をノードグループ (IBMDEFAULTGROUP を含む) に組み込む必要があります。さらに、 REDISTRIBUTE NODEGROUP コマンドを使用して、その区分にデータを移す必要があります。詳細については、*管理 API 解説書* またはコマンド *解説書* を参照してください。

ON NODES
ノードグループに入れる特定の区分を指定します。 NODE は NODES の同義語です。

node-number1
特定の区分番号を指定します。[72]

TO *node-number2*
区分番号の範囲を指定します。 *node-number2* の値は、 *node-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)。指定した区分番号の範囲 (指定した番号を含む) のすべての区分が、ノード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区分またはノードは、 `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 NODES 文節にリストするそれぞれの *node-number* は、同じであってはなりません。
- 有効な *node-number* は、 0 ～ 999 (両端を含む) です (SQLSTATE 42729)。

注

- このステートメントは、ノードグループに対する区分化マップを作成します (詳細については、 67ページの『複数の区分にわたるデータの区分化』を参照してください)。それぞれの区分化マップごとに、区分化マップ識別子 (PMAP_ID) が生成されます。この情報はカタログに記録され、 SYSCAT.NODEGROUPS と SYSCAT.PARTITIONMAPS から検索することができます。区分化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区分を指定します。単一区分ノードグループの場合、対応する区分化マップの項目は 1 つだけです。複数区分ノードグループの場合、対応す

72. 前のバージョンとの互換性を保つために、形式 'NODEnnnnn' のノード名を指定できます。

る区分化マップには 4 096 の項目があり、区分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 つの区分を持つ区分データベースがあると想定します。

- 6 つの区分すべてに対して、 MAXGROUP という名前のノード・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 NODEGROUP MAXGROUP
ON ALL NODES
```

- 区分 0、1、2、5、8 に対して、ノード・グループ MEDGROUP 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 NODEGROUP MEDGROUP
 ON NODES (0 TO 2, 5, 8)
```

- 区分 (またはノード) 7 に対して、単一区分ノード・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 NODEGROUP MINGROUP
 ON NODE (7)
```

注: キーワード NODES は単数形 (NODE) も受け入れられます。

CREATE PROCEDURE

このステートメントは、ストアード・プロシージャー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。

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、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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 (プロシージャ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 (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)

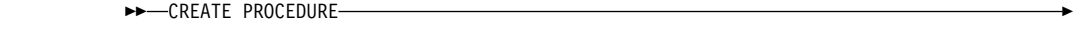
非分離のストアード・プロシージャーを作成するには、ステートメントの許可 ID の特権に以下の特権の少なくとも 1 つが含まれている必要があります。

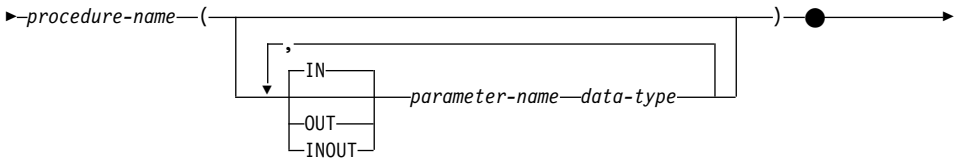
- データベースに対する CREATE_NOT_FENCED 権限
- SYSADM または DBADM 権限

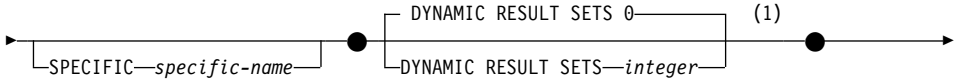
分離ストアード・プロシージャーを作成する場合、追加の権限や特権は必要ありません。

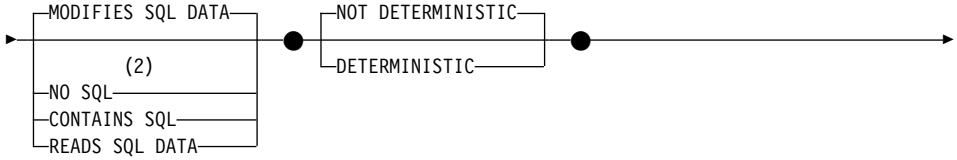
許可 ID の権限が不十分で、操作を実行できない場合には、エラー (SQLSTATE 42502) になります。

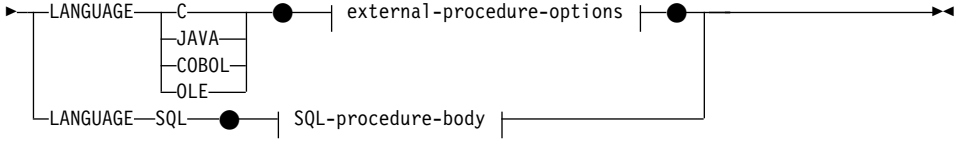
構文



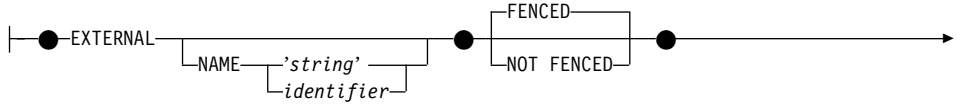


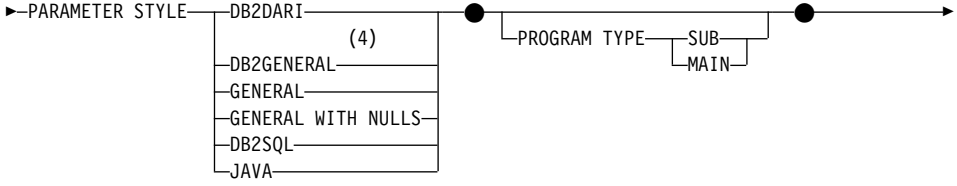


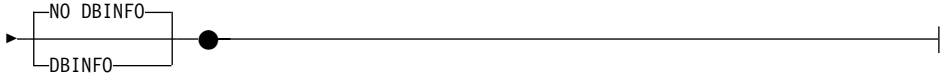




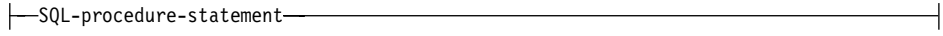
external-procedure-options:







SQL-procedure-body:



注:

1 DYNAMIC RESULT SETS の代わりに RESULT SETS を指定できます。

2 LANGUAGE SQL の場合、NO SQL を選択できません。

3 CALLED ON NULL INPUT の代わりに NULL CALL を指定できます。

4 DB2GENERAL の代わりに DB2GENRL を、GENERAL の代わりに SIMPLE CALL を、 GENERAL WITH NULLS の代わりに SIMPLE CALL WITH NULLS を指定できます。

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 識別子です (最大長 128)。動的 SQL ステートメントでは、CURRENT SCHEMA 特殊レジスターは、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 / バインド・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 識別子が続きます。

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当然固有ですが、複数のスキーマ間で固有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。

(**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、

```
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。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みなされます。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、次のステートメントの場合、

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

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であるからです。

IN | OUT | INOUT
パラメーターのモードを指定します。

- IN - パラメーターは入力のみ
- OUT - パラメーターは出力のみ
- INOUT - パラメーターは入力と出力の両方

parameter-name
パラメーターの名前を指定します。

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。ストアード・プロシージャーに関する 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、 *アプリケーション開発の手引き* の言語別の項を参照してください。

- 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)。

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固有名を指定します。この特定名は、このプロシージャーを除去する場合、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 識別子です (最大長 18)。修飾形式は、*schema-name* の後にピリオドと SQL 識別子が続きます。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* と同じであっても構いません。

修飾子の指定がない場合、*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* の明示または暗黙の修飾子と同じにする必要があります。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、固有の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。生成される固有の名前は、 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。詳細は、*SQL 解説書*の中の『ストアード・プロシージャーから戻される結果セット』の項を参照してください。

上位互換またはファミリーの互換性のために、値 RESULT SETS を DYNAMIC RESULT SETS の同義語として使うことができます。

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 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)。

この後の表は、SQL ステートメント (第 1 列に指定されているもの) を、指定された SQL データ・アクセス指示を使ってストアード・プロシージャーで実行できるかどうかを示します。 NO SQL と定義されたストアード・プロシージャー内に実行可能な SQL ステートメントが出現すると、SQLSTATE 38001 が戻されます。その他の実行コンテキストの場合、どのコンテキストでもサポートされていない SQL ステートメントは SQLSTATE 38003 を戻します。 CONTAINS SQL コンテキスト内で使えないその他の SQL ステートメントの場合は SQLSTATE 38004 が戻され、 READS SQL DATA コンテキストの場合は SQLSTATE 38002 が戻されます。 SQL プロシージャーの作成時に SQL データ・アクセス指示に一致しないステートメントがあると、 SQLSTATE 42895 が戻されることになります。

表 21. SQL ステートメントと SQL データ・アクセス指示

SQL ステートメント	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
ALTER...	N	N	N	Y
BEGIN DECLARE SECTION	Y(1)	Y	Y	Y
CALL	N	Y(4)	Y(4)	Y(4)
CLOSE CURSOR	N	N	Y	Y
COMMENT ON	N	N	N	Y
COMMIT	N	N	N	N
COMPOUND SQL	N	Y	Y	Y
CONNECT(2)	N	N	N	N

表 21. SQL ステートメントと SQL データ・アクセス指示 (続き)

SQL ステートメント	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
CREATE	N	N	N	Y
DECLARE CURSOR	Y(1)	Y	Y	Y
DECLARE GLOBAL TEMPORARY TABLE	N	Y	Y	Y
DELETE	N	N	N	Y
DESCRIBE	N	N	Y	Y
DISCONNECT(2)	N	N	N	N
DROP ...	N	N	N	Y
END DECLARE SECTION	Y(1)	Y	Y	Y
EXECUTE	N	Y(3)	Y(3)	Y
EXECUTE IMMEDIATE	N	Y(3)	Y(3)	Y
EXPLAIN	N	N	N	Y
FETCH	N	N	Y	Y
FREE LOCATOR	N	Y	Y	Y
FLUSH EVENT MONITOR	N	N	N	Y
GRANT ...	N	N	N	Y
INCLUDE	Y(1)	Y	Y	Y
INSERT	N	N	N	Y
LOCK TABLE	N	Y	Y	Y
OPEN CURSOR	N	N	Y	Y
PREPARE	N	Y	Y	Y
REFRESH TABLE	N	N	N	Y
RELEASE CONNECTION(2)	N	N	N	N
RELEASE SAVEPOINT	N	N	N	Y
RENAME TABLE	N	N	N	Y
REVOKE ...	N	N	N	Y
ROLLBACK	N	Y	Y	Y
ROLLBACK TO SAVEPOINT	N	N	N	Y
SAVEPOINT	N	N	N	Y
SELECT INTO	N	N	Y	Y
SET CONNECTION(2)	N	N	N	N

表 21. SQL ステートメントと SQL データ・アクセス指示 (続き)

SQL ステートメント	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
SET INTEGRITY	N	N	N	Y
SET 特殊レジスター	N	Y	Y	Y
UPDATE	N	N	N	Y
VALUES INTO	N	N	Y	Y
WHENEVER	Y(1)	Y	Y	Y

注:

1. NO SQL オプションは SQL ステートメントを指定できないことを暗黙指定しますが、実行不能ステートメントに対する制限はありません。
2. どのストアード・プロシージャー実行コンテキストでも、接続管理ステートメントは使えません。
3. これは、実行しようとするステートメントによって異なります。EXECUTE ステートメントで指定するステートメントは、有効な個々の SQL アクセス・レベルのコンテキストで使えるものでなければなりません。たとえば、有効な SQL アクセス・レベルが READS SQL DATA である場合、ステートメントは INSERT、UPDATE、または DELETE であってはなりません。
4. ストアード・プロシージャー内の CALL ステートメントでは、呼び出し側のストアード・プロシージャーと同じプログラム言語で作成されているストアード・プロシージャーしか参照することができません。

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。

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) として実装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 に概略されています。

LANGUAGE OLE は、 DB2 (Windows 32 ビット 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。

SQL 指定した *SQL-procedure-body* には、ストアード・プロシージャーの処理を定義しているステートメントが組み込まれています。

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。

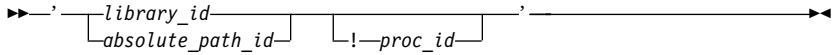
NAME *'string'*

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。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
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、.../sqllib/function/unfenced ディレクトリーと .../sqllib/function ディレクトリー (UNIX 系システムの場合)、または ...¥*instance_name*¥function¥unfenced ディレクトリーと ...¥*instance_name*¥function ディレクトリー (DB2INSTPROF レジストリー変数で指定した OS/2 の Windows 32 ビット オペレーティング・システムの場合) を調べてそのライブラリーを探します。その中で、データベース・マネージャーは、データベース・マネージャーの実行に使用されている制御 sqllib ディレクトリーを見つけ出します。たとえば、UNIX 系システムの制御 sqllib ディレクトリーは、 /u/$DB2INSTANCE/sqllib です。

UNIX 系システムの *library_id* が 'myproc' の場合に、データベース・マネージャーが /u/production から実行されていれば、データベース・マネージャーはライブラリー /u/production/sqllib/function/unfenced/myfunc と /u/production/sqllib/function/myfunc からプロシージャーを見つけます。

OS/2 の Windows 32 ビット オペレーティング・システムの場合、関数ディレクトリーで *library_id* が見つからないと、データベース・マネージャーは LIBPATH または PATH を調べ、分離 (fenced) として実行されます。

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
プロシージャーの全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 プロシージャーを探索します。

OS/2 の場合、Windows 32 ビット オペレーティング・システムの 'd:¥mylib¥myproc' 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 myproc.dll ファイルをロードします。

絶対パスを指定すると、プロシージャーは分離 (fenced) プロシージャーとして実行され、 FENCED または NOT FENCED の属性は無視されます。

! 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。 ! 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、UNIX 系システムで 'mymod!proc8' を指定すると、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 proc8 を使用します。

OS/2 の場合、Windows 32 ビット オペレーティング・システムの 'mymod!proc8' 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 proc8() 関数を呼び出します。

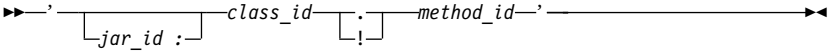
ストリングの形式が正しくない場合には、エラー (SQLSTATE 42878) になります。

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区分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、クラス識別子、およびメソッド識別子が含まれています。クラス識別子とメソッド識別子は、 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、識別子は、 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、該当のクラス識別子とメソッド識別子が

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 (SQLSTATE 42884) になります。



名前は、単一引用符で囲む必要があります。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。

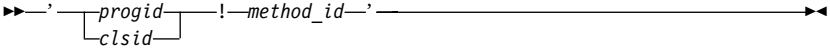
jar_id
jar の集合をデータベースへインストールしたときに、その jar の集合に付けられた jar 識別子を指定します。これは、単純識別子またはスキーマ修飾識別子のいずれかにすることができます。たとえば、'myJar' や 'mySchema.myJar' のようになります。

class_id
Java オブジェクトのクラス識別子を指定します。クラスがパッケージの一部である場合、クラス識別子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。 OS/2 および 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¥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 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 のプログラム識別子 (*progid*) またはクラス識別子 (*clsid*)、およびメソッド識別子 (*method_id*) です。プログラム識別子またはクラス識別子、およびメソッド識別子は、 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、関数を CALL ステートメントで使用する時点で、メソッド識別子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。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識別子を指定します。

progid は、データベース・マネージャーには解釈されず、実行時に 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識別子を指定します。
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、実行時に 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 識別子です。 SQL 識別子は、ストリングの *library-id* として使用されます。区切られた識別子でない場合、識別子は大文字に変換されます。識別子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・マネージャーの操作環境のプロセスまたはアドレス空間で実行しても 『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、その内部資源 (データ・バッファーなど) を隔離して、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。

ストアード・プロシージャーが .../sqllib/function/unfenced ディレクトリーおよび .../sqllib/function ディレクトリー (UNIX 系システム)、または ...¥*instance_name*¥function¥unfenced ディレクトリーおよび ...¥*instance_name*¥function ディレクトリー (OS/2、 Windows 32 ビット オペレーティング・システムの場合) に入っている場合、 FENCED または NOT FENCED の登録属性 (および他のすべての登録属性) は無視されます。

注: 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。

FENCED から NOT FENCED に変更するには、プロシージャーを削除して再作成して、再登録する必要があります。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、または特殊な権限 (CREATE_NOT_FENCED) が必要です。 LANGUAGE OLE を指定した関数には、FENCED のみを指定できます。

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。

DB2DARI

ストアード・プロシージャーは、 C 言語の呼び出しおよびリンケージの規則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 C を使用する場合にだけ指定することができます。

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ことができます。

DB2GENERAL の同義語として値 DB2GENRL が使用可能です。

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 を使用する場合にだけ、指定することができます。

ヌル標識がプログラムに直接渡されることはありません。

GENERAL の同義語として値 SIMPLE CALL が使用可能です。

GENERAL WITH NULLS
GENERAL で指定した CALL ステートメントのパラメーターに加え、別の引き数がストアード・プロシージャーに渡されます。この別の引き数には、CALL ステートメントのパラメーターごとに、ヌル標識のベクトルが含まれています。これは、C では short int の配列になります。これは、LANGUAGE C を使用する場合にだけ、指定することができます。

GENERAL WITH NULLS の同義語として値 SIMPLE CALL WITH NULLS が使用可能です。

DB2SQL
CALL ステートメントのパラメーターに加え、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、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。

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。

パラメーターの受け渡しの詳細については、*アプリケーション開発の手引き* を参照してください。

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。

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とは対照的に、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 のデフォルトは SUB です。 PROGRAM TYPE MAIN は、LANGUAGE C または COBOL で、なおかつ PARAMETER STYLE GENERAL、GENERAL WITH NULLS、または DB2SQL の場合だけ有効です。

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 を伴うプロシージャーは、同じ入力を指定して連続して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。したがって、引き数がヌル値であるか否かに関係なく、ストアード・プロシージャーは呼び出されます。これは、ヌル値を戻す場

合も、通常の (ヌル値以外の) 値を戻す場合もあります。ヌルの引き数値の有無のテストはストアード・プロシージャーが行う必要があります。

値 NULL CALL は、上位互換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。

NO DBINFO または 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、DB2GENERAL、または DB2DARI 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、固有のアプリケーション 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。
- 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 - ストアード・プロシージャーには適用されません。
- 表名 - ストアード・プロシージャーには適用されません。
- 列名 - ストアード・プロシージャー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 - 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。
- 表関数の結果の列番号 - ストアード・プロシージャーには当てはまりません。

構造の詳細、および構造がストアード・プロシージャーにどのようにして渡されるかについては、 *アプリケーション開発の手引き* を参照してください。

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。複

合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。詳しくは、1157ページの『第7章 SQL プロシージャー』を参照してください。

注

- ストアード・プロシージャー用のプログラムの作成方法については、 *アプリケーション開発の手引き* を参照してください。
- 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 特権は PUBLIC に与えられます。
- 呼び出し側の特殊レジスターの設定値は、起動時にストアード・プロシージャーに継承され、呼び出し側に戻されるとただちに復元されます。ストアード・プロシージャー内で特殊レジスターを変更してもかまいませんが、その変更で呼び出し側に影響を与えることはありません。ただし、既存のストアード・プロシージャー (パラメーター・スタイル DB2DARI で定義されているか、またはデフォルト・ライブラリーに保管されているもの) の場合はそうではなく、プロシージャー内で特殊レジスターに対して加えた変更は、呼び出し側の設定値になります。

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、アセンブリー番号を渡されて、アセンブリー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
```

例 3: 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
```

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

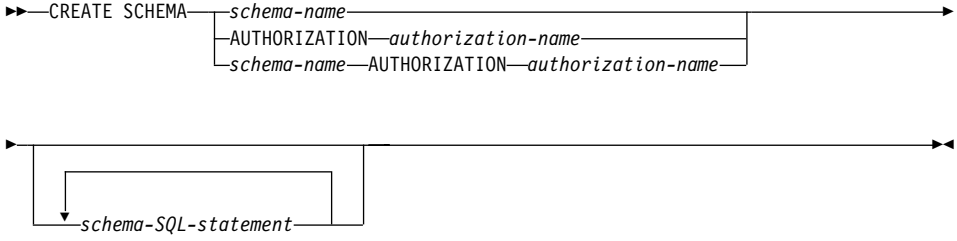
SYSADM 権限または DBADM 権限のある許可 ID は、任意の有効な *schema-name* または *authorization-name* を指定してスキーマを作成できます。

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。値 *authorization-name* 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*
authorization-name という名前のユーザーをスキーマの所有者として、*schema-name* という名前のスキーマを指定します。 *schema-name* は、カタログですでに記述されているスキーマのスキーマ名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 ステートメントの一部として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。

- タイプ付き表および要約表を除外した CREATE TABLE ステートメント (771ページの『CREATE TABLE』を参照)
- タイプ付き視点を除外した CREATE VIEW ステートメント (895ページの『CREATE VIEW』を参照)
- CREATE INDEX ステートメント (715ページの『CREATE INDEX』を参照)
- COMMENT ON ステートメント (569ページの『COMMENT ON』を参照)
- GRANT ステートメント (1011ページの『GRANT (表、視点、またはニックネーム特権)』を参照)

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です。

  - 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です。
- 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、スキーマの所有者です。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授与者でもあります。
- 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。
- AUTHORIZATION 文節が指定され、DCE 認証が使用される場合、この文節の後に続くステートメントの実行に必要な認証の評価の際に、指定した *authorization-name* のグループ・メンバーシップは考慮されません。指定された *authorization-name* がスキーマを作成した許可 ID と異なる場合、CREATE SCHEMA ステートメントの実行時に許可が失敗する場合があります。
- スキーマ名として SESSION を使用しないようお勧めします。宣言された一時表は 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された一時表に解決されます (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、SQL ステートメントのコンパイルは別々の時点で行われるので、結果は、宣言された一時表がいつ定義されたかによって異なります。持続表、視点、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) )
```

CREATE SERVER

CREATE SERVER ステートメント[73] は、データ・ソースを連合データベースへ定義します。

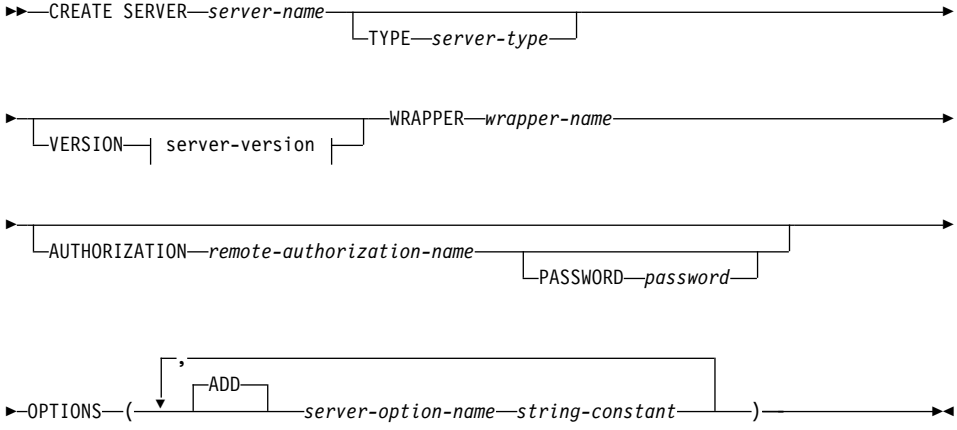
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

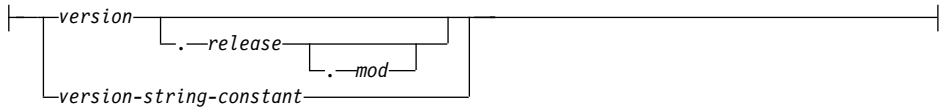
このステートメントの許可 ID には、連合データベースに対する SYSADM 権限または DBADM 権限がなければなりません。

構文



server-version:

73. このステートメントでは、 SERVER という語と、*server-* で開始するパラメーター名は、連合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連合サーバー、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。連合システムについての詳細は、46ページの『DB2 連合システム』を参照してください。 DRDA アプリケーション・サーバーについての詳細は、 33ページの『分散リレーショナル・データベース』を参照してください。



説明

server-name
連合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、連合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。

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。このオプションは、DRDA、SQLNET、および NET8 ラッパーでは必須です。サポートされているデータ・ソース・タイプのリストは、 1351ページの『付録F. 連合システム』を参照してください。

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。

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-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、連合サーバーが使用するラッパーを指定します。

AUTHORIZATION *remote-authorization-name*
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。

PASSWORD *password*
remote-authorization-name で示された許可 ID に関連付けられているパスワードを指定します。 *password* を指定しない場合のデフォルトは、連合データベースに接続するのに使用する ID のパスワードになります。

OPTIONS
使用可能にするサーバー・オプションを指定します。 *server-option-name* とその設定の詳細は、 1355ページの『サーバー・オプション』を参照してください。

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

- *remote-authorization-name* を指定しない場合、連合データベースの許可 ID が使用されます。
- データ・ソースで *password* を必要とする場合には、それを指定しなければなりません。*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。 *identifier* を指定しても *password* を指定していない場合、*server-name* で示されたデータ・ソースの認証タイプは CLIENT と見なされます。
- 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。

- あるタイプのデータ・ソースではサーバー・オプションが所定の値に設定され、このタイプのインスタンスでは同じオプションが別の値に設定される場合、インスタンスの値については、最初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、連合システムの DB2 ユニバーサル・データベース (OS/390 版) データ・ソースにおいて、 PASSWORD が 'Y' (データ・ソースでパスワードを妥当性検査する) にセットされたとします。後で、SIBYL という特定の DB2 ユニバーサル・データベース (OS/390 版) データ・ソースにおいて、このオプションのデフォルト ('N') が使われるとします。結果は、SIBYL 以外のすべての DB2 ユニバーサル・データベース (OS/390 版) データ・ソースで、パスワードの妥当性検査が実行されます。

例

例 1: DB2WRAP というラッパーを使ってアクセスできる、 DB2 (MVS/ESA 版) 4.1 データ・ソースを定義します。データ・ソース CRANDALL を呼び出します。 さらに、以下のものを指定します。

- MURROW 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。
- CRANDALL は、DB2 RDBMS に対し、MYNODE というインスタンスとして定義されます。
- 連合サーバーが CRANDALL へアクセスすると、MYDB というデータベースにつながります。
- 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。
- MYDB と連合データベースは、同じ照合順序を使用します。

```
CREATE SERVER CRANDALL
TYPE DB2/MVS
VERSION 4.1
WRAPPER DB2WRAP
AUTHORIZATION MURROW
PASSWORD DROWSSAP
OPTIONS
    ( NODE 'MYNODE',
    DBNAME 'MYDB',
    FOLD_ID 'U',
    FOLD_PW 'U',
    COLLATING_SEQUENCE 'Y' )
```

例 2: KLONDIKE というラッパーを使ってアクセスできる、 Oracle 7.2 データ・ソースを定義します。データ・ソース CUSTOMERS を呼び出します。以下のように指定します。

- CUSTOMERS は、Oracle RDBMS に対し、ABC というインスタンスとして定義されます。

最適化プログラムについて、以下の統計を指定します。

- 連合サーバーの CPU の動作速度が、 CUSTOMERS をサポートする CPU の 2 倍であること。
- 連合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 7.2
WRAPPER KLONDIKE
OPTIONS
    ( NODE 'ABC',
    CPU_RATIO '2.0',
    IO_RATIO '1.5' )
```

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、基本キーや検査制約などの表の他の属性を含めることができます。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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 CREATETAB 権限および表スペースに対する USE 特権と、以下の特権のうちの 1 つ。
  - データベースに対する IMPLICIT_SCHEMA 権限 (表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 (表のスキーマ名が既存のスキーマを指している場合)

  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。

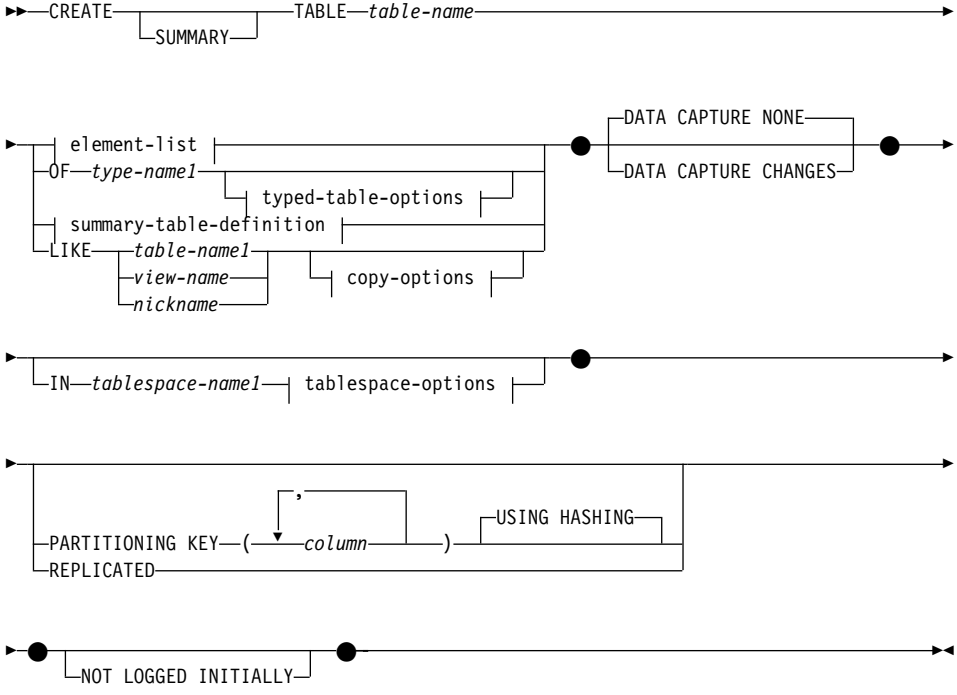
- その表に対する REFERENCES 特権
- 指定の親キーのそれぞれの列に対する REFERENCES 特権
- その表に対する CONTROL 特権
- SYSADM または DBADM 権限

(全選択を使用して) 要約表を定義するには、このステートメントの許可 ID に、全選択で識別された個々の表または視点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。

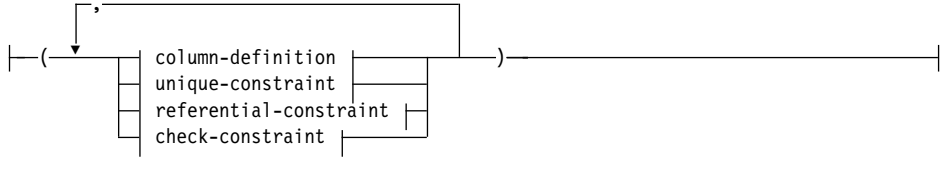
- 表または視点に対する SELECT 特権と、 REFRESH DEFERRED または REFRESH IMMEDIATE が指定されている場合には ALTER 特権。

- その表または視点に対する CONTROL 特権
- SYSADM または DBADM 権限

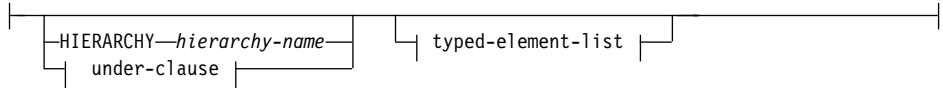
構文



element-list:

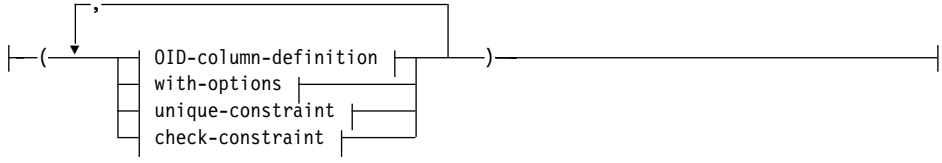


typed-table-options:

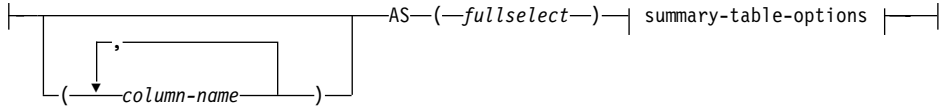


under-clause:

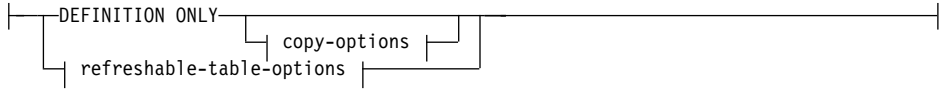
typed-element-list:



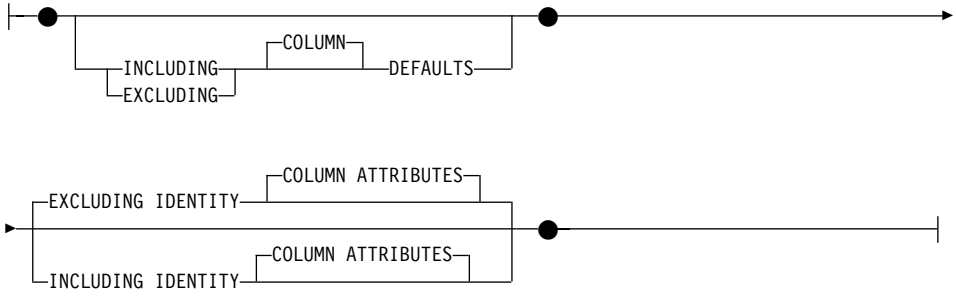
summary-table-definition:



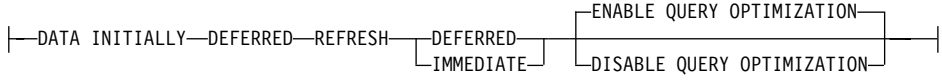
summary-table-options:



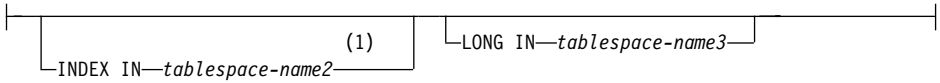
copy-options:



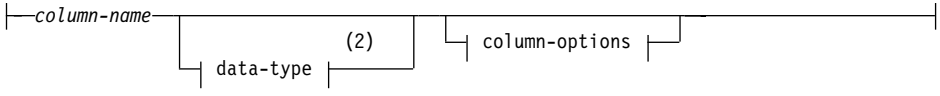
refreshable-table-options:



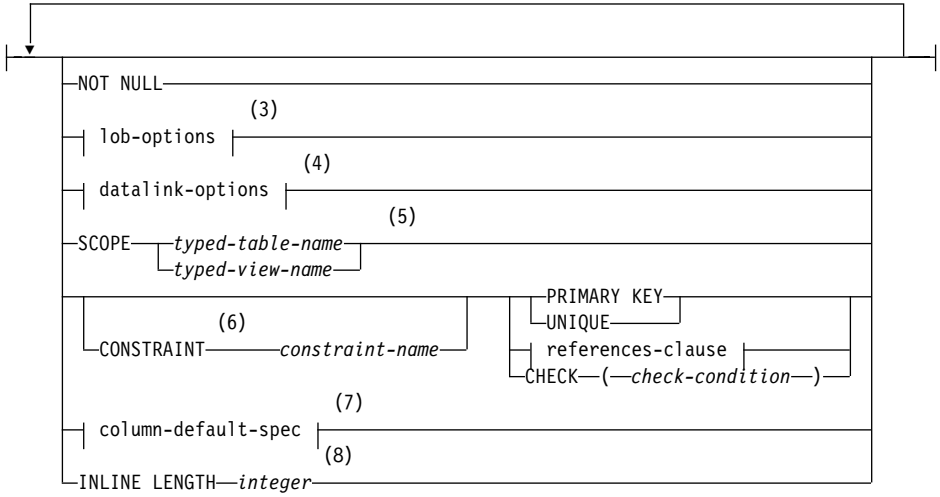
tablespace-options:



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 column-default-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-option (データ・リンク・オプション) 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。 これらのタイプには LINKTYPE URL 文節が必須です。

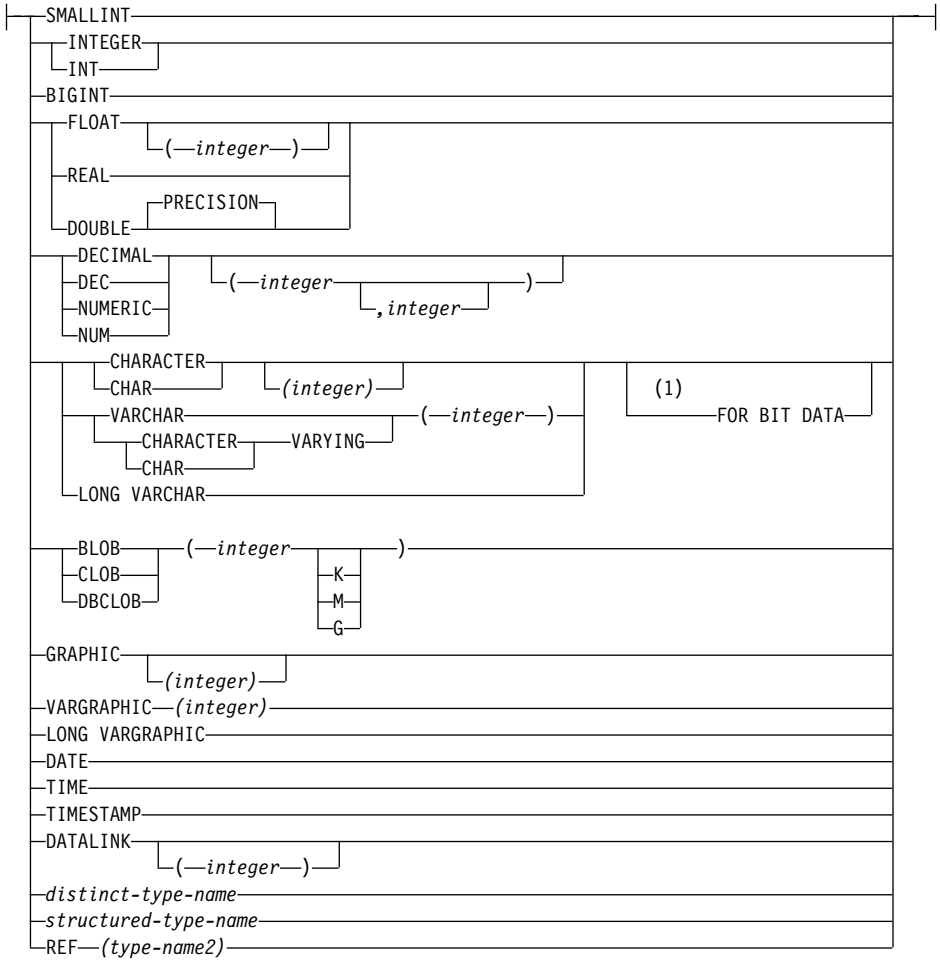
5 SCOPE 文節は REF タイプに対してのみ適用されます。

6 バージョン 1 との互換性を保つため、 references-clause (REFERENCES 文節) を定義する *column-definition* (列定義) では、 CONSTRAINT キーワードを省略することができます。

7 IDENTITY 列属性は、 複数の区画をもつ拡張エンタープライズ・エディション (EEE) データベースではサポートされません。

8 INLINE LENGTH は、構造タイプと定義された列に対してのみ用います。

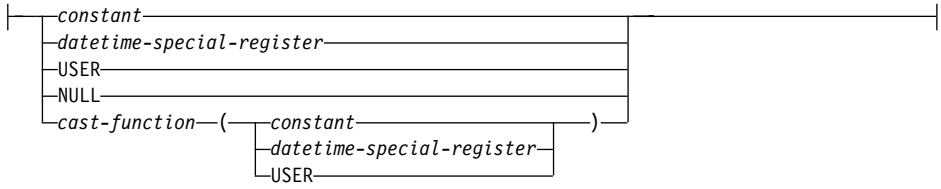
data-type:



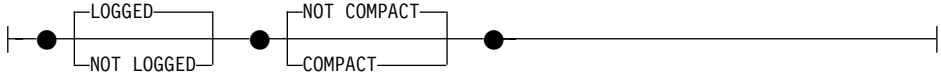
注:

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。

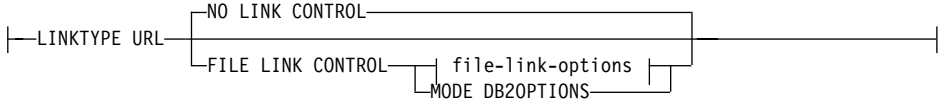
default-values:



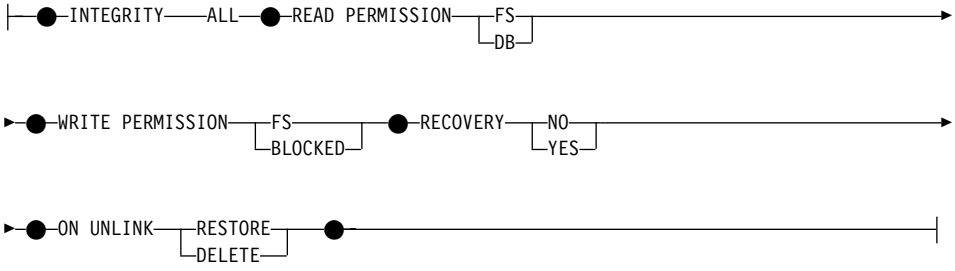
lob-options:



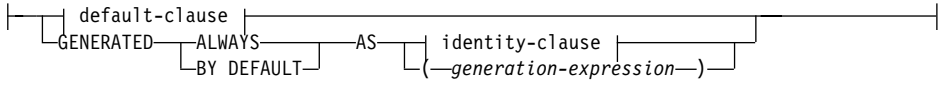
datalink-options:



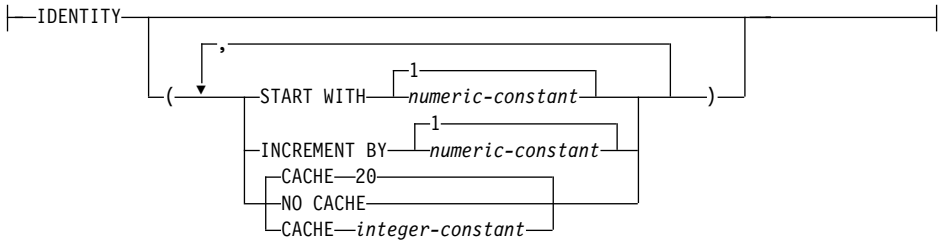
file-link-options:



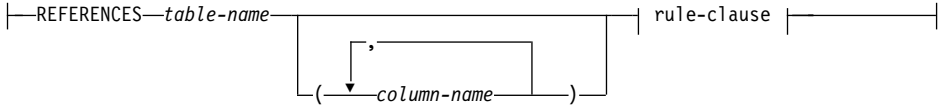
column-default-spec:



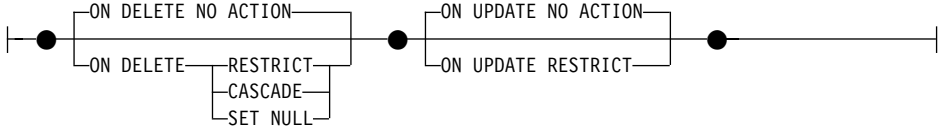
identity-clause:



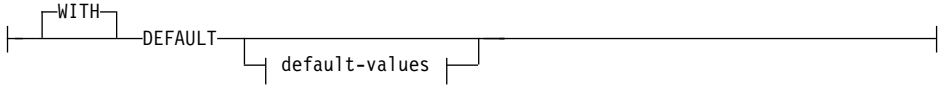
references-clause:



rule-clause:

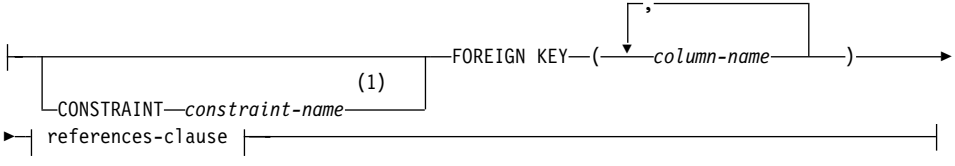


default-clause:

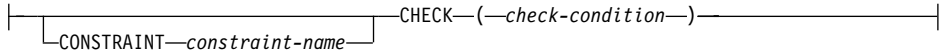


unique-constraint:

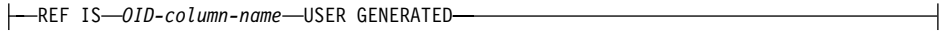
referential-constraint:



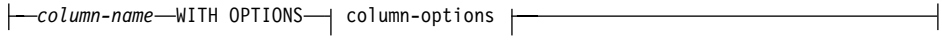
check-constraint:



OID-column-definition:



with-options:



注:

1 バージョン 1 との互換性を保つため、 constraint-name (制約名) は FOREIGN KEY に続けて (CONSTRAINT キーワードなしで) 指定することができます。

説明

SUMMARY

要約表が定義されていることを示します。このキーワードは任意選択ですが、指定する場合には、ステートメントに *summary-table-definition* を組み込む必要があります (SQLSTATE 42601)。

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、視点、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 前処理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、オブジェクト識別列を指定する必要があります (*OID-column-definition* を参照)。このオブジェクト識別列は、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。タイプ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、視点、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固有な接尾部で構成される識別子であり、既存の表、視点、別名、およびニックネームの識別子の中で固有のものです。

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識別子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、データ取り込み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、区分化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みなされます。

element-list
表の要素を定義します。これには、表の列と制約の定義が含まれます。

typed-element-list
タイプ付き表の追加要素を定義します。これには、列の追加オプション、オブジェクト識別子列 (ルート表のみ) の追加、表の制約事項などが含まれます。

summary-table-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要約表となります。

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、固有で、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
表の定義および表に含まれるデータの判別に使用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。作成した列定義は、同じ照会で定義した視点の定義と同じになります。

各選択リスト要素には名前が必要です (式には AS 文節を使用します。詳細は、423ページの『SELECT 文節』を参照してください)。指定された *summary-table-options* (要約表オプション) は、要約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。

DEFINITION ONLY が指定されている場合、タイプ付き表またはタイプ付き視点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)。

- 任意の FROM 文節でのニックネーム、要約表、宣言された一時表、またはタイプ付き表への参照

- 視点の全選択が、要約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、その視点への参照
- 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
- 外部アクションを指定する関数
- SQL で書かれた関数
- 物理的特性に依存する関数 (たとえば、NODENUMBER、PARTITION)
- システム・オブジェクトに対する表または視点参照 (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

REFRESH IMMEDIATE は、以下の場合に指定されます。

- 全選択が必ず副選択であること
- 副選択に以下のものが含まれていないこと
  - 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
- 要約表が REPLICATED でない限り、副選択に GROUP BY 文節が必ず入っていること
- 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。要約表の選択リストには、SUM(X) が含まれています。この X には、ヌル可能な引き数が入ります。さらに、要約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。
- FROM 文節で複数の表または視点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。
- すべての GROUP BY 項目が選択リストに含まれていること
- GROUPING SETS、CUBE、および ROLLUP がサポートされていること。選択リスト内の GROUP BY 項目とそれに関連した

GROUPING 列関数は、結果セットの固有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。たとえば、`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 内に出現するヌル可能な 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
- 定数でのグループ化は行えない

- HAVING 文節は許可されていない
- 複数の区分ノードグループ内の場合、区分化キーは、項目別のグループのサブセットでなければならない。そうでなければ、要約表を複製しなければならない。

summary-table-options
要約表の属性を定義します。

DEFINITION ONLY
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、表は要約表とみ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、そしてヌル可能特性を使って定義されます。

refreshable-table-options
要約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最新表示できます。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要約表には、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、要約表にカスケードされます。その場合、表の内容は、どの時刻指定でも、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要約表には、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、要約表を照会最適化に使用することができます。

DISABLE QUERY OPTIMIZATION
要約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。

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*)、視点 (*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、視点またはニックネーム、あるいは宣言された一時表を識別するものでなければなりません。タイプ付き表またはタイプ付き視点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います。*n* は、指定した表、視点またはニックネームにおける列の数です。

- 表を指定すると、暗黙的な定義には *table-name1* のおのおのの列の列名、データ・タイプ、およびヌル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。
- 視点を指定すると、暗黙的な定義には *view-name* に指定した全選択のおのおのの結果列の列名、データ・タイプ、およびヌル可能特性が入ります。
- ニックネームを指定すると、暗黙的な定義には *nickname* のおのおのの列の列名、データ・タイプ、およびヌル可能特性が入ります。

copy-attributes 文節に基づいて、列のデフォルトと識別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、視点、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、新しい表には固有制約、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で作成されます。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、視点、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内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* が基礎表または一時表を指定する場合、 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、しかも *table-name* が基礎表または一時表を指定する場合を除きます。

INCLUDING IDENTITY COLUMN ATTRIBUTES
可能であれば、ソース結果表の定義から識別列属性 (START WITH、INCREMENT BY、および CACHE 値) がコピーされます。識別列属性をコピーできるのは、表、視点、または全選択内の対応する列の要素が、識別特性をもつ基礎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視点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。以下に例を示します。

- 全選択の選択リストには、識別列名の複数のインスタンスが入ります (つまり、同一列の複数回の選択)
- 全選択の選択リストには、複数の識別列が入ります (つまり、結合が関与します)。
- 識別列は、選択リスト内の式に入ります。
- 全選択には、SET 演算 (UNION、EXCEPT、または INTERSECT) が入ります。

EXCLUDING IDENTITY COLUMN ATTRIBUTES
ソース結果表の定義から識別列属性はコピーされません。

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。

表には、以下のものを指定できます。

- 最大 500 列の 4K ページ・サイズ。列のバイト・カウントは 4K ページ・サイズで 4005 を超えてはなりません。詳細については、820ページの『行サイズ』を参照してください。
- 最大 1 012 列の 8K ページ・サイズ。列のバイト・カウントは 8101 を超えてはなりません。詳細については、820ページの『行サイズ』を参照してください。
- 最大 1 012 列の 16K ページ・サイズ。列のバイト・カウントは 16 293 を超えてはなりません。
- 最大 1 012 列の 32K ページ・サイズ。列のバイト・カウントは 32 677 を超えてはなりません。

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、次のとおりです。

SMALLINT
短精度整数。

INTEGER または **INT**
長精度整数。

BIGINT
大整数。

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* (整数) 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 (*precision-integer*) 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 (*scale-integer*) 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 32700 の可変長文字ストリング。

FOR BIT DATA

列の内容をビット (2 進) 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、データベース照合順序に関係なく 2 進で行われます。

BLOB*(integer [K | M | G])*

2 進ラージ・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。

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。

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 オプションを指定しなければなりません。

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。スペースなしでも構いません。たとえば、次の例はすべて有効です。

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

CLOB*(integer [K | M | G])*[74]
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。

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 オプションを指定しなければなりません。

DBCLOB*(integer [K | M | G])*
2 バイト文字ラージ・オブジェクト (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。

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 オプションを指定しなければなりません。

GRAPHIC*(integer)*
長さ *integer*（整数）の固定長漢字ストリング。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* の可変長漢字ストリング。長さは、1 ～ 16,336 の範囲で指定できます。

---

74. 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。

LONG VARGRAPHIC
最大長 16,350 の可変長漢字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)*
データベース外で保管されたデータ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、一そろい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す。 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
- DLURLPATH
- DLURLPATHONLY
- 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、この列を基本キーまたは固有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。列をヌル値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 前処理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、基本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 前処理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、基本キー、固有制約、外部キー、索引キー、または区分化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効力範囲は解除されます。そのような属性を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、その列はヌルにしかできません。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***)**
タイプ付き表への参照。 *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 前処理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
列にヌル値が入るのを防止します。

NOT NULL を指定しない場合、列にヌル値を含めることができます。また、そのデフォルト値はヌル値、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）によって回復することができます。 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。ログ記録しないなら、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 値をゼロによって置換したものになります。破損回復の過程で、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。 LOB 列をログ記録しない場合に起こることについては、*管理の手引き* を参照してください。

COMPACT
後続の付加操作で使用できるスペースを LOB 記憶域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) の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。これはデフォルト値です。

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。列でファイル名を検索しなくても、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、表から検出された 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。列でファイル名を検索しなくても、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しなければなりません。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。

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回復を 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回復をサポートしています。この値は、INTEGRITY ALL と WRITE PERMISSION BLOCKED 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
特定時刻での回復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は、製品によって異なった対応がなされます。[75] この値は、INTEGRITY ALL および WRITE PERMISSION BLOCKED も指定されている場合だけ、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 も指定されている場合だけです。

75. DB2 ユニバーサル・データベースの場合、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。

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 が使用されているため、ON UNLINK は適用できません。

SCOPE

参照タイプ列の効力範囲を指定します。

参照解除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、すべて効力範囲を指定しなければなりません。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

タイプ付き表の名前。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

タイプ付き視点の名前。この視点は既に存在しているものか、作成する視点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ると、表に定義されている既存の制約の識別子内で固有な 18 文字の識別子が[76] システムによって生成されます。

PRIMARY KEY 制約または UNIQUE 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 つの列からなる基本キーを定義する簡単な方法として用意されています。したがって、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。

基本キーはスーパー表から継承されたものなので、表が副表である場合 (SQLSTATE 429B3)、基本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 つの列からなる固有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。

固有制約はスーパー表から継承されたものなので、表が副表である場合 (SQLSTATE 429B3)、固有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。したがって、REFERENCES 文節が列 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 文節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。

---

76. 識別子は、 ″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。

CHECK (*check-condition***)**
これは、1 つの列に適用される検査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、基礎表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含まれませんが、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)。

column-default-spec

default-clause
列のデフォルト値を指定します。

WITH
任意選択キーワード。

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。 517ページの表19 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。

列がタイプ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しなければなりません。タイプ付き表のオブジェクト識別子の列には、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ヌル値が使用されます。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、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 などの先行文字も含まれます。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 または UPDATE 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
INSERT または UPDATE の実行時における USER 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。 USER を指定した場合、その列のデータ・タイプは、 USER 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、必ず関数を使用しなければなりません。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。 *cast-function* の使用法の例は、533 を参照してください。

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
USER 特殊レジスターを指定します。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。

指定した値が無効な場合、エラー (SQLSTATE 42894) が発生します。

GENERATED
DB2 が列の値を生成することを指示します。その列が、生成される列または IDENTITY 列とみなされることになる場合は、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 は常に列の値を生成することを指示します。この式の結果は、表に保管されます。デー

タ伝搬を使用したり、アンロードおよび再ロードの操作を行ったりするのでなければ、 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
値が指定されていなくても、行が表に挿入されると DB2 は列の値を生成することを指示します。データ伝搬を使用したり、アンロード / 再ロードを行ったりするときは、BY DEFAULT の値をお勧めします。

明示的には指示されませんが、値の固有性を確保するには、固有の 1 列の索引を生成列で定義しなければなりません。

AS IDENTITY
列をこの表の識別列にすることを指定します。[77] 1 つの表には 1 つの IDENTITY 列しかあってはなりません (SQLSTATE 428C1)。 IDENTITY キーワードを指定できるのは、列に関連付けられた *data-type* が、ゼロの位取りの完全な数値タイプである場合か、[78] または、ソース・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ユーザー定義特殊タイプである場合だけです (SQLSTATE 42815)。

識別列は暗黙で NOT NULL になります。

START WITH *numeric-constant*
識別列の最初の値を指定します。この値は、この列に割り当てることのできる任意の正または負の値でかまいません (SQLSTATE 42820)。ただし、小数点の右側に非ゼロの数字がないことを前提とします (SQLSTATE 42894)。デフォルトは 1 です。

INCREMENT BY *numeric-constant*
連続した識別列値を 1 つずつ区切る間隔を指定します。この値は、この列に割り当てることのできる任意の正または

77. 識別列は、複数区画をもつデータベースではサポートされません (SQLSTATE 42997)。データベースで複数の区画が存在する場合は、識別列を作成できません。識別列を組み込まれたデータベースを、複数の区画を指定して開始することはできません。

78. 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みなされます。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・タイプとみなされます。参照タイプは、完全な数値タイプで表されていても、識別列と定義することはできません。

負の値でかまいません (SQLSTATE 42820)。この値は、ゼロであってはならず、長精度整数定数の値を超えてはなりません (SQLSTATE 428125)。ただし、小数点の右側に非ゼロの数字がないことを前提とします (SQLSTATE 42894)。

この値が負の場合、この識別列の値の順序は降順になります。この値が正の場合、この識別列の値の順序は昇順になります。デフォルトは 1 です。

CACHE または NO 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。識別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えないときは、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、識別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、ログ記録しなくて済むので迅速化されます。これは、パフォーマンスおよびチューニングのオプションです。

CACHE *integer-constant*

DB2 で事前割り振りしてメモリーに保存する識別シーケンスの数を指定します。値を事前割り振りしてキャッシュに保管すれば、識別列用の値の生成時のログ記録が軽減されます。

識別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えないときの値の割り振り作業には、ログ記録される間の待機も含まれます。ただし、識別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、ログ記録を実行しなければ、その識別値の割り振りを迅速に行うことができます。

データベースの非活動化時には、それが通常操作[79] またはシステム障害のどちらに起因するものであっても、コミット済みのステートメントでまだ使われていないキャッシュされたシーケンス値はすべて失われま

79. データベースを明示的に活動化 (ACTIVATE コマンドまたは API を使って) していない場合に、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。

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識別列値の最大数は、 CACHE オプションに指定された値になります。

最小値は 2、最大値は 32767 です (SQLSTATE 42815)。デフォルトは CACHE 20 です。

NO CACHE
識別列の値を事前割り振りしないことを指定します。

このオプションを指定すると、識別列の値はキャッシュに保管されません。その場合、新たに識別値を要求するたびにログ記録がとられることになります。

AS (*generation-expression***)**
列定義が式に基づくことを指定します。[80] *generation-expression* には、以下のどれも入れることができません (SQLSTATE 42621)。

- 副照会
- 列関数
- 参照解除操作または 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
- SCRATCHPAD オプションを使用するユーザー定義関数
- READS SQL DATA オプションを使用するユーザー定義関数
- 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、効力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、 71ページの

80. 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。

の『第3章 言語要素』で説明されている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ヌル可能とみなされます。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。ただしこの場合、LONG VARCHAR、LONG VARGRAPHIC、LOB データ・タイプ、DATALINK、構造タイプ、および、これら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
タイプ付き表のオブジェクト識別子列を定義します。

REF IS *OID-column-name* **USER GENERATED**
オブジェクト識別子列 (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タイプ付き表 (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、システム必須の固有索引 (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*識別子列* 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
タイプ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 のオプションを指定変更します。

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、基本キーまたは固有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複製に関する特殊な情報を、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)。

表がタイプ付き表である場合、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、収集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取り込む場合に必須です。

カタログ区分以外の区分にデータが置かれるように表が定義されている場合 (複数区分のノードグループ、またはカタログ区分以外の区分を持つノードグループ)、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)。

複製の使用法の詳細については、 *管理の手引き* および *レプリケーションの手引きおよび解説書* を参照してください。

IN *tablespace-name1*
表を作成する表スペースの名前 (*tablespace-name*) 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 特権の対象の正規 (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。表階層のルート表から表スペースを継承しているため、この文節は副表を作成する際には指定できません (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 table space IBMDEFAULTGROUP over which the user has USE privilege
  exists with sufficient page size
    THEN choose it
 ELSE IF a table space over which the user has USE privilege
  exists with sufficient page size
  (複数表スペース修飾の場合は下記を参照)
    THEN choose it
 ELSE issue an error (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。

- 表スペースを除去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。 820ページの『行サイズ』を参照。

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、828ページの『CREATE TABLESPACE』 を参照してください。

INDEX IN *tablespace-name2*

表に対する索引を作成する表スペースの名前 (*tablespace-name*) を指定します。このオプションは、 IN 文節に指定された基礎表スペースが DMS 表スペースである場合にのみ使用することができます。指定する表スペースは、存在しており、ステートメントの許可 ID がもつ USE 特権が対象とする REGULAR DMS 表スペースであり、 *tablespace-name1* と同じノード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 特権をもつことがデータベース・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 文節に指定された基礎表スペースが DMS 表スペースである場合にのみ使用することができます。その表スペースは、存在していて、ステートメントの許可 ID がもつ USE 特権の対象である LONG DMS 表スペースである必要があり、 *tablspace-name1* の同じノードグループ内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***,...)**

表のデータが区分化されている場合に、区分化キーを指定します。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、区分化キーの一部として使用することはできません (SQLSTATE 42962)。表階層のルート表から区分化キーが継承されるため、副表である表には区分化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、この表が複数区分のノードグループに存在する場合、その区分化キーは次のように定義されます。

- 表がタイプ付き表である場合、オブジェクト識別子列が区分化キーであり、
- 基本キーが指定されている場合は、その基本キーの最初の列が区分化キーになります。

- これら以外の場合、 LOB 以外のデータ・タイプの最初の列、 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が区分化キーになります。

デフォルトの区分化キーの要件を満たす列が存在しない場合、表は区分化キーなしで作成されます。このような表は、単一区分のノード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。

単一区分のノード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区分化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、区分化キーは作成されません。

区分化キーに関する制約事項については、815ページの『規則』を参照してください。

USING HASHING
データ配分の区分化方式として、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区分化方式です。

REPLICATED
表が定義される表スペースのノード・グループの各データベース区画に対して、表に保管されたデータを物理的に複写することを指定します。つまり、これらのデータベース区画にはそれぞれ、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、要約表にのみ指定できます (SQLSTATE 42997)。

NOT LOGGED INITIALLY
表を作成する作業単位と同一の作業単位の Insert、 Delete、 Update、 Create Index、 Drop Index、または Alter Table 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、 816ページの『注』を参照してください。

カタログの変更と、記憶域に関連する情報は、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。

NOT LOGGED INITIALLY 属性を指定した親を参照する表に、外部キー制約を定義することはできません。この文節は、副表を作成する際には指定できません (SQLSTATE 42613)。

注: 保管点へのロールバック要求を、 NOT LOGGED INITIALLY 表の作成と同じ作業単位内で発行することはできません。発行するとエラーが生じ (SQLSTATE 40506)、その作業単位全体がロールバックされます。

unique-constraint

固有制約または基本キー制約を定義します。表に区分化キーがある場合、固有キーまたは基本キーは区分化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、固有制約または基本キーを指定することはできません (SQLSTATE 429B3)。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*

基本キー制約、または固有制約の名前を指定します。 796 ページを参照してください。

UNIQUE (*column-name***,...)**

指定した列で構成される固有キーを定義します。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、 820ページの『Byte Counts』を参照)。それぞれの列の長さは、どれも 255 バイトを超えてはなりません。その長さはデータにのみ関するものであり、出現してもヌル・バイトによって影響を受けることはありません。列がヌル可能かどうかに関係なく、列の最大データ長は 255 バイトです。列の長さ属性が 255 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、固有キーの一部として使用できません (SQLSTATE 42962)。

固有キーの一連の列は、基本キーまたは別の固有キーの一連の列と同じであってはなりません (SQLSTATE 01543)。[81]

固有制約はスーパー表から継承されたものなので、表が副表である場合 (SQLSTATE 429B3)、固有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、固有キーとその固有索引が含まれます。固有索引は、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

81. LANGLEVEL が SQL92E または MIA であると、エラーが戻されます (SQLSTATE 42891)。

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基本キーを定義します。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、 820ページの『Byte Counts』を参照)。それぞれの列の長さは、どれも 255 バイトを超えてはなりません。その長さはデータにのみ関するものであり、出現してもヌル・バイトによって影響を受けることはありません。列がヌル可能かどうかに関係なく、列の最大データ長は 255 バイトです。列の長さ属性が 255 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、基本キーの一部として使用できません (SQLSTATE 42962)。

基本キーの一連の列は、固有キーの一連の列と同じであってはなりません (SQLSTATE 01543)。[81]

1 つの表には、基本キーを 1 つだけ定義することができます。

基本キーはスーパー表から継承されたものなので、表が副表である場合 (SQLSTATE 429B3)、基本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、基本キーとその 1 次索引が含まれます。固有索引は、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。

表に区分化キーがある場合、 *unique-constraint* (固有制約) の列は区分化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。 796 ページを参照してください。

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、 820ページの『Byte Counts』を参照)。 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、グラフィック、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
参照制約の親表と親キーを指定します。

REFERENCES *table-name*
REFERENCE 文節に指定される表は、カタログに記述された基礎表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、および親表と同じである場合、参照制約は重複します。重複した参照制約は無視され、警告が出されます (SQLSTATE 01543)。

以下の説明で、T2 は指定された親表を、T1 は作成している表[82] を指しています (T1 と T2 は同じ表にできます)。

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

82. または、この文節が ALTER TABLE ステートメントの記述から参照される場合には、変更される表。

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、日付 / 時刻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、親キーまたは T2 に存在する固有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 に親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基本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ヌル可能な列がヌル値に設定されます。

SET NULL は、外部キーの列にヌル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。

複数の表の関係するサイクルでは、そのサイクル内の削除規則がすべて CASCADE でない限り、表がそれ自体に連結削除されていてはなりません。したがって、新しい関係がサイクルを形成し、T2 がすでに T1 に連結削除されている場合には、削除規則が CASCADE で、しかもそのサイクルの他の削除規則がすべて CASCADE であるのでない限り、制約を定義することはできません。

T1 が複数のパスによって T2 に連結削除されている場合、それらのパスの全体または一部を構成する関係のうち T1 が従属であるものの削除規則は同じでなければならず、 SET NULL にすることはできません。 NO ACTION アクションと RESTRICT アクションの処理は同じです。したがって、削除規則が *r* の関係で T1 が T3 の従属表であるとすると、 *r* が SET NULL のときに、次の条件のいずれかが存在するなら、参照制約を定義することはできません。

- T2 と T3 は同じ表です。
- T2 は T3 の子孫であり、T3 からの行の削除は T2 に連鎖されます。
- T3 は T2 の子孫であり、T2 からの行の削除は T3 に連鎖されます。
- T2 と T3 はともに同一の表の従属表であり、その表からの行の削除は T2 および T3 へ連鎖されます。

r が SET NULL 以外の場合、参照制約は定義できますが、FOREIGN KEY 文節で暗黙的または明示的に指定される削除規則は *r* と同一でなければなりません。

親表または従属表がタイプ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 はそれに代え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、 816ページの『注』 の CREATE TABLE を参照してください。

check-constraint
検査制約を定義します。*検査制約* は、偽以外に評価されなければならない *探索条件* です。

CONSTRAINT *constraint-name*
検査制約の名前を指定します。 796 ページを参照してください。

CHECK (*check-condition***)**
検査制約を定義します。検査条件 (*check-condition*) は、次の場合を除いて、*探索条件* です。

- 列参照が、作成する表の列でなければならない場合
- *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
  - 参照解除操作または、効力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
  - SCRATCHPAD オプションを使用するユーザー定義関数
  - READS SQL DATA オプションを使用するユーザー定義関数
  - ホスト変数
  - パラメーター・マーカー
  - 特殊レジスター
  - 別名
  - 識別列以外の生成列の参照

列定義 の一部として検査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検査制約には、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。検査制約の矛盾、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、矛盾した検査制約や冗長な検査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。

検査条件 ″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ヌル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、制約違反とみなされます。

検査制約は、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。表で定義される検査制約は、その表の副表すべてに自動的に適用されます。

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)。詳細については、820ページの『Byte Counts』、および 1205ページの表33 を参照してください。タイプ付き表の場合、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、ヌル値不可として定義されたとしても、バイト・カウントの際にはヌル値可と見なされます)。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)。タイプ付き表の場合は、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。
- タイプ付き表のオブジェクト識別子列は更新できません (SQLSTATE 42808)。
- 表の区分化キー列は更新できません (SQLSTATE 42997)。
- 表に対して定義された固有キー制約または基本キー制約は、区分化キーのスーパーセットでなければなりません (SQLSTATE 42997)。
- 区分化キーのヌル可能な列は、関係が ON DELETE SET NULL を指定して定義されている場合は、外部キーの列として組み込むことはできません (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)。

表 22. 有効な DATALINK ファイル制御オプションの組み合わせ

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

以下の規則は、区分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、単一区分ノード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。
- 複数区分のノードグループに対して定義された表スペースの表の区分化キー定義は変更できません。
- タイプ付き表の区分化キー列は OID 列にしなければなりません。

注

- 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 特権は PUBLIC に与えられます。
- 外部キーが指定されると、
  - 親表に削除使用のある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。このことが、削除または更新中に違いとして現れる場合はほとんどありません。こ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 として定義された視点の行に対する DELETE 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。
- 複数区分のノード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、区分化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、各表は同じノード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、同じノード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、各表の区分化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区分互換である必要があります。詳細については、128ページの『区分の互換性』を参照してください。
  - 区分化キーの選択も、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区分化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えません (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、表の回復が不要な場合は、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、データのログ記録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除去されます。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除去です。
  - 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、表の回復が可能になります。
- CURRENT REFRESH AGE を ANY 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 要約表を使用できます。 ENABLE QUERY OPTIMIZATION を指定して定義された REFRESH IMMEDIATE 要約表は、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 要約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。

  要約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)。
- 要約表が REFRESH IMMEDIATE で定義されている場合、基礎となる表の挿入、更新、または削除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、基礎となる表の挿入、更新、または削除は失敗します。

- 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、非ヌルの外部キーをもつ 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。アクションの始動の仕方に関係なく、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。
     - 除去済みの表が親表または従属表である参照制約では、制約は除去されます。
     - 除去済みの表のスーパー表が親表である参照制約では、その除去済みの表の行は、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。
     - 除去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。
- ***作動不能要約表:*** 作動不能要約表とは、SQL ステートメントで使用できなくなった表のことです。要約表は、次の場合に作動不能になります。
  - 要約表定義が従属する特権が取り消される場合。
  - 要約表定義が従属している表、別名、または関数などのオブジェクトが除去された場合。

  実際には、作動不能要約表とは、要約表定義が間違って除去された要約表です。たとえば、別名が除去されると、その別名を使用して定義されている要約表すべてが作動不能になります。要約表に従属するパッケージはすべて無効になります。

  作動不能要約表を明示的に作成しなおすか、あるいは除去する時点まで、その作動不能要約表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 TABLE、DROP TABLE、および COMMENT ON TABLE の各ステートメントは例外です。作動不能要約表が明示的に削除されるまで、その修飾名を使って別の視点、基礎表、または別名を作成することはできなくなります (SQLSTATE 42710)。

作動不能要約表は、作動不能要約表の定義テキストを使用して CREATE TABLE 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。この要約表照会テキストは、SYSCAT.VIEWS カタログの TEXT 列に保管されます。作動不能要約表を再作成する場合は、他のユーザーでその要約表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、要約表に作動不能のマークが付いていると、要約表のすべての許可レコードが削除されるためです。作動不能要約表を再作成するために、それを明示的に削除する必要はありません。要約表を定義する CREATE TABLE ステートメントに、動作不能要約表と同じ *table-name* を指定して発行すると、動作不能要約表は置き換えられ、 CREATE TABLE ステートメントは警告を戻します (SQLSTATE 01595)。

作動不能要約表であることは、SYSCAT.VIEWS カタログ視点の VALID 列の X、また SYSCAT.TABLES カタログ視点の STATUS 列が X であることによって示されます。

- ***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 23. 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。

– 列の合計 * 8 + LOB 列の数 * 12 + Datalink 列の数 * 28 <= 各ページ・サイズでの行サイズの限界。

- ***Byte Counts:*** 次のリストは、ヌル値を使えない列のバイト・カウントを、列のデータ・タイプ別に示したものです。ヌル値可能な列の場合、バイト・カウントは、リストに示されたカウントよりも 1 つ多くなります。

構造タイプに基づいて表が作成されると、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、ヌル値不可と定義されたとしても、バイト・カウント用にヌル値可能なものと見なされる必要があります。

データ・タイプ	バイト・カウント
INTEGER	4
SMALLINT	2
BIGINT	8
REAL	4
DOUBLE	8
DECIMAL	(*p*/2)+1 の整数部分 (*p* は精度)
CHAR(*n*)	*n*
VARCHAR(*n*)	*n* +4
LONG VARCHAR	24
GRAPHIC(*n*)	*n**2
VARGRAPHIC(*n*)	(*n**2)+4
LONG VARGRAPHIC	24
DATE	4
TIME	3
TIMESTAMP	10
DATALINK(*n*)	*n*+54
LOB types	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。次の表は、典型的なサイズを示しています。

最大 LOB 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 + 4。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。

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、列にヌル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。基本キーは、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、列にヌル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 とみなされる 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* 文節に関して説明し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: 検査制約に含まれる列がヌル値可能である場合、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、固有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
```

固有制約を課すために、 EMP_ACT_UNIQ という名前の固有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。新聞記事は削除したり、名前を変更したりできないようにリンクで接続されますが、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、スチール写真は回復されて元の所有者に戻されます。一方、ムービー写真は回復されず、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 列はヌル値可能です。

```
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
   ) DEFINITION ONLY
```

例 14: 示された属性を持つ次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 が次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、次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 が次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

バイト以内におさまらない 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礎表とは別個に) 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 つの列をもつ表 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 IDENTITY 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))
```

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。

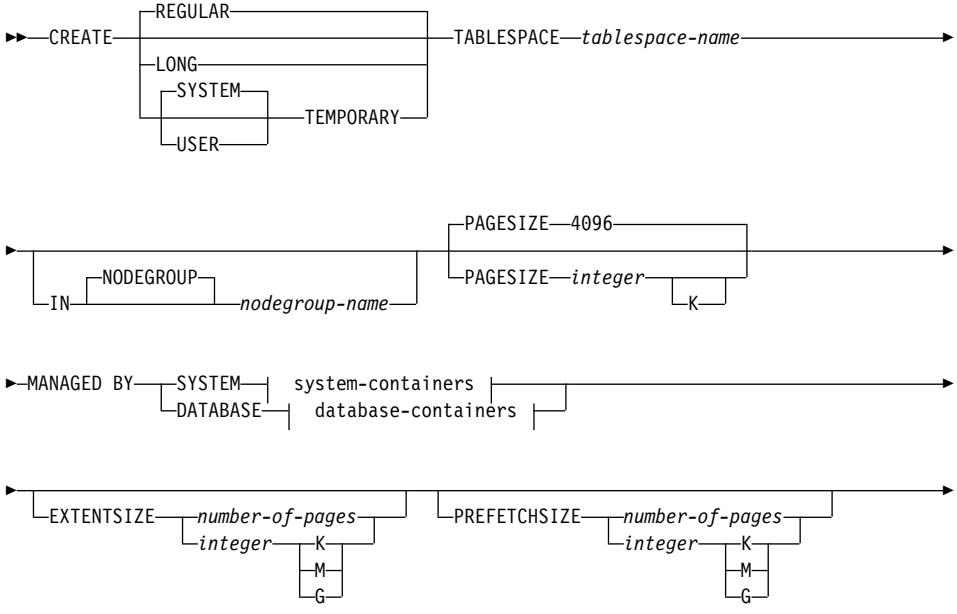
呼び出し

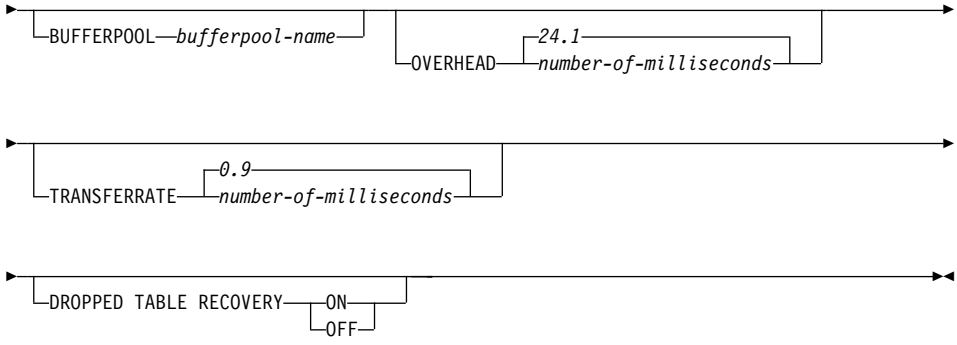
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは対話式に使用することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

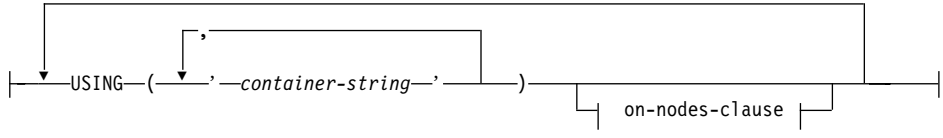
このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

構文

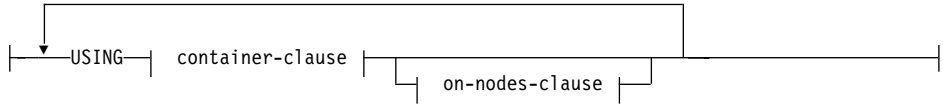




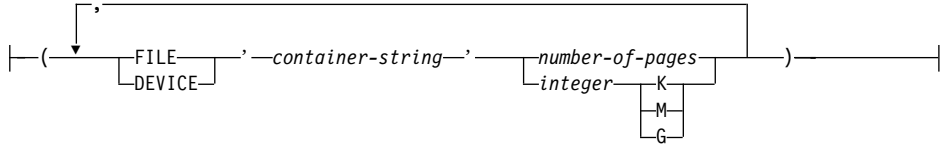
system-containers:



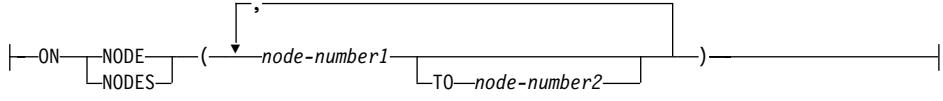
database-containers:



container-clause:



on-nodes-clause:



説明

REGULAR
一時表を除くすべてのデータを保管します。

LONG
長形式または LOB の表の列を保管します。また、構造タイプ列を保管することもできます。表スペースは DMS 表スペース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。一時表スペースは、データベースの作成時に自動的に作成されます。

詳細については、*コマンド解説書* の CREATE DATABASE を参照してください。

USER TEMPORARY
宣言された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一時表スペースは存在しないことに注意してください。宣言された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 つのユーザー一時表スペースを作成しなければなりません。

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 識別子です (通常識別子または区切り識別子)。*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 NODEGROUP *nodegroup-name*
表スペースのノードグループを指定します。該当のノード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ノードグループは、IBMTEMPGROUP だけです。NODEGROUP キーワードは任意選択です。

ノードグループを指定しないと、デフォルトのノードグループ (IBMDEFAULTGROUP) が、 REGULAR、LONG、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・ノードグループ 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
SMS 表スペースに対するコンテナーを指定します。

USING *('container-string',...)*
SMS 表スペースに対して、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構成要素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。 表スペースを除去すると、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構成要素が削除されます。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、OS/2、Windows 95 および Windows NT 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。

リモート資源 (OS/2、Windows 95 および Windows NT の LAN リダイレクト・ドライブや、 AIX の NFS マウント・ファイル・システム) はサポートされません。

on-nodes-clause
区分データベースにおいて、コンテナーを作成する区分を指定します。この文節を指定しない場合、他のどの *on-nodes-clause* にも明示的に指定されていないノードグループ内の区分でコンテナーが作成されます。ノードグループ IBMTEMPGROUP で定義されている SYSTEM TEMPORARY 表スペースについては、 *on-nodes-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区分またはノードでもコンテナーが作成されます。この文節の指定の詳細については、833 ページを参照してください。

MANAGED BY DATABASE
表スペースを、データベース管理スペース (DMS) 表スペースとして指定します。

database-containers
DMS 表スペース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
DMS 表スペースに対してコンテナーを指定します。

(FILE|DEVICE *'container-string' number-of-pages,...)*
DMS 表スペースに対して、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・サイズで分割されたバイト数のフロアは、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構成要素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。表スペースを除去すると、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構成要素が削除されます。

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、すべてのデータベースを通して固有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。 コンテナーの宣言の詳細については、管理の手引き を参照してください。

リモート資源 (OS/2、Windows 95 および Windows NT の LAN リダイレクト・ドライブや、 AIX の NFS マウント・ファイル・システム) はサポートされません。

on-nodes-clause

区分データベースにおいて、コンテナーを作成する区分を指定します。 この文節を指定しない場合、他のどの *on-nodes-clause* にも明示的に指定されていないノードグループ内の区分でコンテナーが作成されます。ノードグループ IBMTEMPGROUP で定義されている SYSTEM TEMPORARY 表スペースについては、 *on-nodes-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区分でもコンテナーが作成されます。この文節の指定の詳細については、833 ページを参照してください。

on-nodes-clause

区分データベースにおいて、コンテナーを作成する区分を指定します。

ON NODES

特定の区分を指定することを示すキーワードです。 NODE は NODES の同義語です。

node-number1
特定の区分 (またはノード) 番号を指定します。

TO *node-number2*
区分 (またはノード) 番号の範囲を指定します。 *node-number2* の値は、 *node-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)。 この表スペースのノードグループにノードが組み込まれると、コンテナーが作成される区分に、指定した区分番号の範囲 (指定した区分番号を含む) のすべての区分が組み込まれます。

番号によって指定する区分と、区分の範囲内のすべての区分 (またはノード) は、表スペースを定義するノードグループに存在している必要があります (SQLSTATE 42729)。 ある区分番号を明示的に、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-nodes-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・サイズで分割されたバイト数のフロアは、エクステント・サイズのページ値の数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 構成パラメーターによって指定されます。

PREFETCHSIZE *number-of-pages*
データの事前取り出しの実行中に、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。 この事前取り出しサイズ値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・サイズで分割されたバイト数のフロアは、事前取り出しサイズのページ値の数を判別するために使用します。 事前取り出しでは、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。 (この構成パラメーターは、他の構成パラメーターと同様に、 *管理の手引き* で詳しく説明されています。)

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、この表スペースのノードグループを定義する必要があります (SQLSTATE 42735)。

OVERHEAD *number-of-milliseconds*
number-of-milliseconds (ミリ秒数) は、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*
number-of-milliseconds は、 1 個の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 オプションを使えば、指定した表スペースで除去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)。除去後の表の回復について詳しくは、*管理の手引き*を参照してください。

注

- EXTENTSIZE、PREFETCHSIZE、OVERHEAD、および TRANSFERRATE の正しい値を決定する方法については、*管理の手引き* を参照してください。
- 表スペースをデータベース管理スペースにするか、それともシステム管理スペースにするかの選択は、トレードオフの関係を含む基本的な選択です。このトレードオフについては、*管理の手引き* を参照してください。
- 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。複数の表スペースを使用し、 EXTENTSIZE、PREFETCHSIZE、 OVERHEAD、および TRANSFERRATE のバランスを再調整する方法、およびこれらの推奨値については、 *管理の手引き* を参照してください。

- 区分データベースで、複数の区分が同じ物理ノードに存在する場合、このような区分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、それぞれの区分ごとに固有の *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・ストリング構文にノード式を指定することができます。 ノード式は一般に、区分データベース・システムで複数の論理ノード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 (データベース区画サーバー) 間で固有のものとなります。 この式を指定する場合、ノード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。

  ノード式を示すには、引き数 ″ $N″ ([ブランク]$N) を使用します。引き数は必ずコンテナー・ストリングの末尾に指定し、以下のいずれかの形式でのみ使用できます。 以下の表では、ノード番号を 5 とします。

表 24. コンテナーを作成するための引き数

構文	例	値
[blank]$N	" $N"	5
[blank]$N+[number]	" $N+1011"	1016
[blank]$N%[number]	" $N%3"	2
[blank]$N+[number]%[number]	" $N+12%13"	4
[blank]$N%[number]+[number]	" $N%3+20"	22
注: – % はモジュラス – どの場合でも、演算子は左から右へと評価されます。		

以下に、いくつかの例を示します。

例 1:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
   (device '/dev/rcont $N' 20000)
```

2 ノード・システムでは、以下のようなコンテナーが使用されます。

```
/dev/rcont0  - on NODE 0
/dev/rcont1  - on NODE 1
```

例 2:

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
   (file '/DB2/containers/TS2/container $N+100' 10000)
```

4 ノード・システムでは、次のようなコンテナーが作成されます。

```
/DB2/containers/TS2/container100  - on NODE 0
/DB2/containers/TS2/container101  - on NODE 1
/DB2/containers/TS2/container102  - on NODE 2
/DB2/containers/TS2/container103  - on NODE 3
```

例 3:

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
('/TS3/cont $N%2','/TS3/cont $N%2+2')
```

2 ノード・システムでは、以下のようなコンテナーが作成されます。

```
/TS3/cont0  - On NODE 0
/TS3/cont2  - On NODE 0
/TS3/cont1  - On NODE 1
/TS3/cont3  - On NODE 1
```

例

例 1: UNIX 系システムで、それぞれ 10 000 の 4K ページの 3 つの装置を使用する通常の DMS 表スペースを作成します。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 24.1
   TRANSFERRATE 0.9
```

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、事前取り出しサイズを 32 ページに指定して、 OS/2 または Windows NT で通常の SMS 表スペースを作成します。

```
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 の区分データベースで、ノード・グループ ODDNODEGROUP (ノード 1、3、5) に対して DMS 表スペースを作成します。すべての区分 (またはノード) で、装置 /dev/rhdisk0 の 10 000 個の 4K ページを使用します。また、それぞれの区分に、40 000 個の 4K ページがある区分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 NODE (1)
   USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
   ON NODE (3)
   USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
   ON NODE (5)
```

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のに使います。

呼び出し

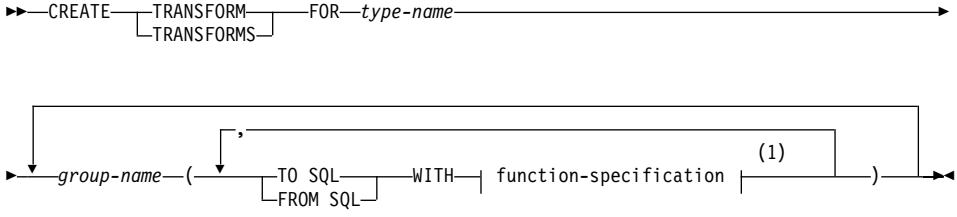
このステートメントは、アプリケーション・プログラムに組み込むか、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

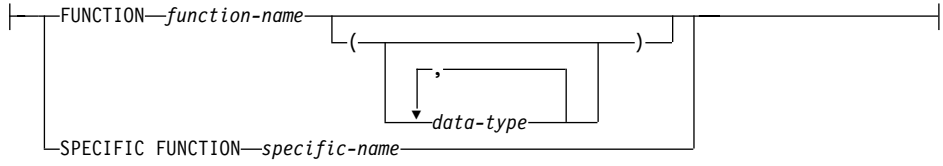
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- SYSADM または DBADM 権限
- *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定義者

構文



function-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 つ以上の変形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*
変形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 特殊レジスターは、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 / バインド・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変形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 関数をもつ変形グループの名前を指定します。その名前は固有名である必要はありません。しかし、その名前の付いた変形グループ (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 識別子である必要があり、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変形するのに使用する個々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・タイプ値に値を変形するのに使用する個々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 変形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 変形関数の使用を指定している

メソッド (FUNCTION ACCESS を使って指定したものでも) を、*function-specification* を使って変形として指定することはできません。つまり、CREATE FUNCTION ステートメントで定義されている関数だけが、変形として作動することができます (SQLSTATE 42704 または 42883)。

さらに、必須ではありませんが、FROM SQL 関数から戻された 1 つ以上の組み込みタイプが、 TO SQL 関数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。

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。このように指定するプロシージャーには、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 特殊レジスターは、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 / バインド・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、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-tape,...***)**
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 特殊レジスターは、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 / バインド・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,...***)**
ここで指定するデータ・タイプは、 CREATE FUNCTION 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。 タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。 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みなされません。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 特殊レジスターは、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 / バインド・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。

注

- 静的 SQL の場合は TRANSFORM GROUP プリコンパイル・オプションまたはバインド・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・プログラム内で変形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、変形グループ 'DB2_PROGRAM' 内の変形関数が使われます (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 変形が実行され、構造タイプは、変形関数から戻された組み込みデータ・タイプに変形されます。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 変形が実行され、組み込みデータ・タイプ値は構造タイプ値に変形されます。ユーザー定義の変形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。
- 構造タイプ・ホスト変数を組み込みデータ・タイプで表現したものは、次のように割り当て可能でなければなりません。
  - プリコンパイル・コマンドの指定 (検索割り当て規則を使用) の TRANSFORM GROUP オプションで定義されているとおりの構造タイプの FROM SQL 変形関数の結果から
  - プリコンパイル・コマンドの指定 (記憶域割り当て規則を使用) の TRANSFORM GROUP オプションで定義されているとおりの構造タイプの TO SQL 変形関数のパラメーターへ

ホスト変数が、該当する変形関数での規定のタイプと互換性のある割り当てでないと、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、108ページの『ストリングの割り当て』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を呼び出すと、そのたびに 'DB2_FUNCTION' という名前のデフォルト変形グループ内で指定されている変形関数が使われます。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を呼び出すと、 FROM SQL 変形が実行され、構造タイプは、変形関数から戻された組み込みデータ・タイプに変形されます。同様に、関数の戻りデータ・タイプがデータ・タイプ *type-name* であると、 TO SQL 変形が実行され、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、構造タイプ値に変形されます。
- 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、それに関連した変形関数は、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、変形関数の結果またはパラメーターは、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、変形関数が「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 1 つまたは複数の関数は、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変形関数は、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。
- 特定のタイプの属性またはサブタイプを作成または除去したときは、ユーザー定義構造タイプの変形関数も変更する必要があります。
- ある特定の変形グループについて、FROM SQL 関数と TO SQL 関数を指定できるのは、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除去しておかないと、 FROM SQL 関数または TO SQL 関数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 変形関数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 変形関数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変形関数と Java 用に特殊化した変形関数に関連付ける 2 つの変形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 )
```

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

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。

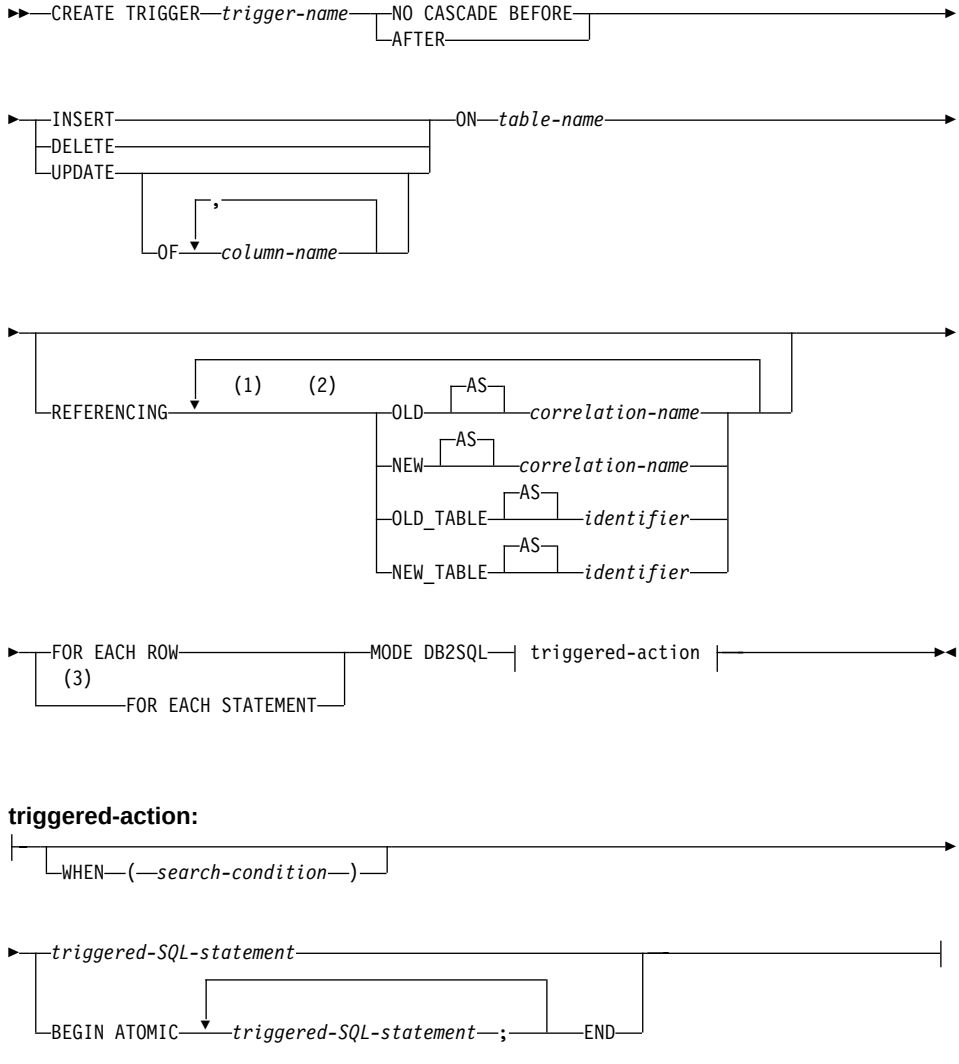
- SYSADM または DBADM 権限
- トリガーを定義する表に対する ALTER 特権、またはトリガーを定義する表のスキーマに対する ALTERIN 特権、および以下のいずれか。
  - データベースに対する IMPLICIT_SCHEMA 権限 (トリガーの暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (トリガーのスキーマ名が既存のスキーマを指している場合)。

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。

- 変換変数または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視点に対する 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:

注:

1 OLD と NEW は、それぞれ一度だけ指定できます。

2 OLD_TABLE と NEW_TABLE は、それぞれ一度だけ、 AFTER トリガーに対してのみ指定できます。

3 FOR EACH STATEMENT は、BEFORE トリガーには指定できません。

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。

INSERT
指定した基礎表に 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
指定した基礎表に 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、指定した基礎表に 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、基礎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、識別列以外の生成列ではありません (SQLSTATE 42989)。 *column-name* リストの 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。

ON *table-name*

トリガー定義の対象となる表を指定します。 *table-name* に指定する名前は、基礎表、または解決結果が基礎表になる別名でなければなりません (SQLSTATE 42809)。この名前に、カタログ表 (SQLSTATE 42832)、要約表 (SQLSTATE 42997)、宣言された一時表 (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。

REFERENCING

変換変数 の相関名と*変換表* 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 操作によって変更された行が、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さらに変更された時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 操作の前のものを指定する一時表名を *identifier* に指定します。

NEW_TABLE AS *identifier*

トリガーとなる SQL 操作による影響を受けた行が、すでに実行された BEFORE トリガーによってさらに変更された状態を指定する一時表名を *identifier* に指定します。

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、元の操作による行の新しい状態が、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ものを捕らえます。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)。
- 変換表は変更できません (SQLSTATE 42807)。
- トリガー・アクションの中での変換表の列と変換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* の効力範囲は、トリガー定義全体です。

FOR EACH ROW
対象となる表の行で、トリガーとなる SQL 操作によって影響を受ける各行ごとに、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細分性は、BEFORE 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。

MODE DB2SQL

この文節は、トリガーのモードを指定するために使用します。これは、現在サポートされている唯一有効なモードです。

triggered-action

トリガーが活動化された時に実行されるアクションを指定します。 1 つのトリガー・アクションは、 1 つまたはいくつかの *triggered-SQL-statement* と、その *triggered-SQL-statement* が実行されるためのいくつかのオプション条件によって構成されます。指定するトリガー・アクションに複数の triggered-SQL-statement がある場合、これらを BEGIN ATOMIC キーワードと END キーワードで囲んでセミコロンで区切る必要があります。[83] それらは、指定された順序で実行されます。

WHEN (*search-condition***)**

真、偽、または未知となる条件を指定します。 *search-condition* (探索条件) を使うことによって、特定のトリガー・アクションを実行するかどうかを決めることができます。

関連するアクションは、指定した探索条件が真と評価される場合にのみ実行されます。 WHEN 文節を省略すると、関連する *triggered-SQL-statement* は常に実行されることになります。

triggered-SQL-statement

トリガーが BEFORE トリガーの場合、トリガーにより実行される SQL ステートメントは、次のいずれかでなければなりません (SQLSTATE 42987)。

- 全選択[84]
- SET *変換変数* SQL ステートメント
- SIGNAL SQLSTATE ステートメント

トリガーが AFTER トリガーの場合、トリガーにより実行される SQL ステートメントは、次のいずれかでなければなりません (SQLSTATE 42987)。

- INSERT SQL ステートメント
- 探索 UPDATE SQL ステートメント
- 探索 DELETE SQL ステートメント
- SIGNAL SQLSTATE ステートメント

---

83. コマンド行プロセッサーでこの形式を使用する場合、ステートメントの終了文字をセミコロンにすることはできません。代替の終了文字の指定については、*コマンド解説書* を参照してください。

84. 全選択の前に共通表式を指定できます。

- 全選択[84]

triggered-SQL-statement で、未定義の変換変数 (SQLSTATE 42703) や、宣言された一時表 (SQLSTATE 42995) を参照することはできません。

BEFORE トリガーの *triggered-SQL-statement* は、 REFRESH IMMEDIATE を定義した要約表を参照できません (SQLSTATE 42997)。

BEFORE トリガーの *triggered-SQL-statement* は、新しい変換変数で識別列以外の生成列を参照することはできません (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、変換表に対して ″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。
- *triggered-SQL-statement* として指定された全選択の結果は、トリガーの内部でも外部でも使用できません。
- 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削除前トリガーは、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もっと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削除前トリガーや CASCADE の削除規則には、*triggered-action* に関係のある表への参照は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更新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無効になります。
  - 列リストを指定した更新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。
  - 挿入トリガーを作成した場合、ターゲット表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。トリガーは、次の場合に作動不能になります。
  - トリガーの作成者から、そのトリガーを実行するために必要な特権が取り消された場合。
  - トリガー・アクションが従属している表、視点、または別名などのオブジェクトが除去された場合。
  - トリガー・アクションが従属している視点が作動不能になった場合。
  - トリガーの対象の表である別名が除去された場合。

  実際には、作動不能トリガーとは、 DROP ステートメントまたは REVOKE ステートメントのカスケード規則の結果として、トリガー定義が除去されたトリガーです。たとえば、視点が除去されると、その視点を使用して定義された *triggered-SQL-statement* は作動不能になります。

  トリガーが作動不能になると、それまで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の含まれるパッケージすべてが無効になります。パッケージが (明示的にまたは暗黙のうちに) 再バインドされるとき、**作動**

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。

作動不能トリガーは、その作動不能トリガーの定義テキストを使用して、CREATE TRIGGER ステートメントを発行することにより再び作成できます。 このトリガー定義テキストは、SYSCAT.TRIGGERS の TEXT 列に保管されています。 作動不能トリガーを再作成するために、それを明示的に削除する必要はありません。 作動不能トリガーと同じトリガー名 (*trigger-name*) を使用して CREATE TRIGGER ステートメントを発行すると、その作動不能トリガーは置換され、警告 (SQLSTATE 01595) が出されます。

作動不能トリガーであることは、 SYSCAT.TRIGGERS カタログ視点の VALID 列が X であることによって示されます。

- ***Errors executing triggers***: 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。エラーが重大な場合は、重大エラーの SQLSTATE が戻されます。 重大でないエラーの場合の SQLCA の SQLERRMC フィールドには、トリガー名、SQLCODE、SQLSTATE、およびエラーのトークンの中からフィールドに入るだけの数のトークンが入れられます。

  triggered-SQL-statement は、SIGNAL SQLSTATE ステートメントにすることも、また RAISE_ERROR 関数を含めることもできます。いずれの場合でも、戻される SQLSTATE は、 SIGNAL SQLSTATE ステートメントまたは RAISE_ERROR 条件で指定されたものになります。
- 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 特権は PUBLIC に与えられます。
- データベース・マネージャーが識別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。
- ***トリガーおよびタイプ付き表***: 表階層のどのレベルのタイプ付き表にも、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、それぞれタイプ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、その変換変数 (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副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 コマンドを発行すると、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、除去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、除去される表が副表である場合、その除去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みなされます。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。

視点でのアクション: 視点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、視点定義を使ってそのアクションを、基礎表でのアクションに変換します。たとえば、次のようにします。

1. SQL ステートメントで UPDATE V1 を実行します。 V1 は、副視点 V2 をもつタイプ付き視点です。 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、副視点 V2 をもつタイプ付き視点です。 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 と定義されたタイプ付き視点です。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) と定義されたタイプ付き視点です。この場合、V1 が副視点をもち 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。

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 MODE DB2SQL
   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 MODE DB2SQL
   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 MODE DB2SQL
     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 MODE DB2SQL
  WHEN (N.SALARY > 1.1 * O.SALARY)
         SIGNAL SQLSTATE '75000' ('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。

```
表:     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 MODE DB2SQL
  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 MODE DB2SQL
  BEGIN ATOMIC
     INSERT INTO QUOTEHISTORY
        VALUES (NEWQUOTE.SYMBOL, NEWQUOTE.QUOTE, CURRENT TIMESTAMP);
  END
```

CREATE TYPE (構造化)

CREATE TYPE ステートメントは、ユーザー定義の構造タイプを定義します。ユーザー定義構造タイプには、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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

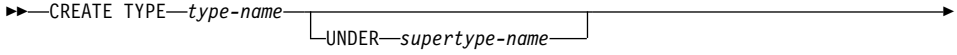
許可

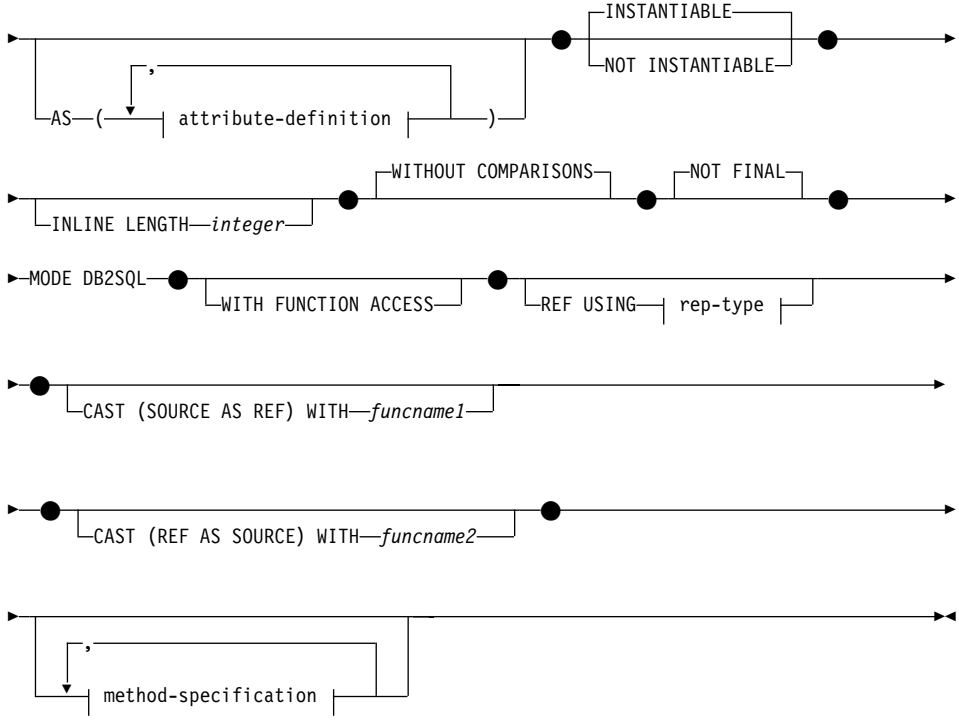
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- 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)

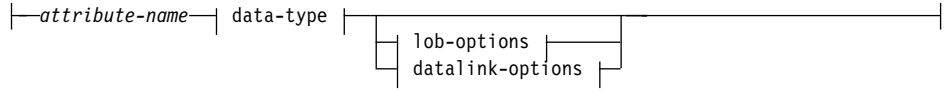
UNDER が指定されていて、このステートメントの許可 ID がタイプ階層のルート・タイプの定義者と同じではない場合には、 SYSADM または DBADM 権限が必要です。

構文

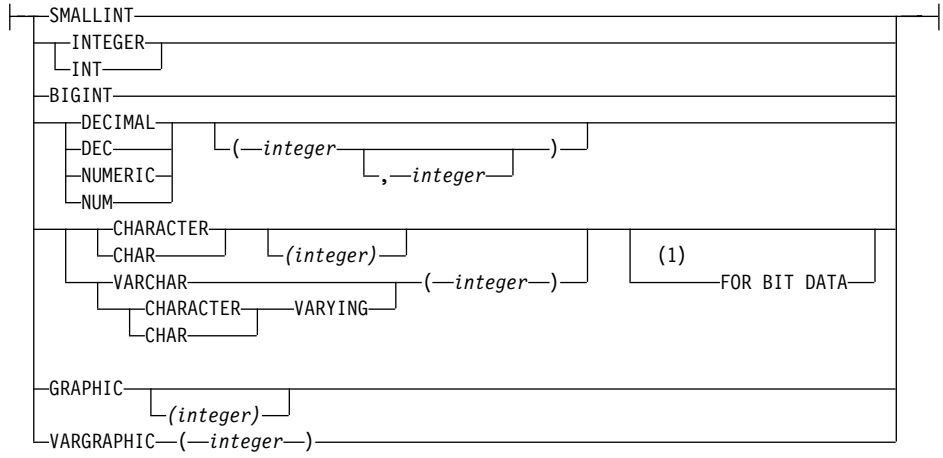




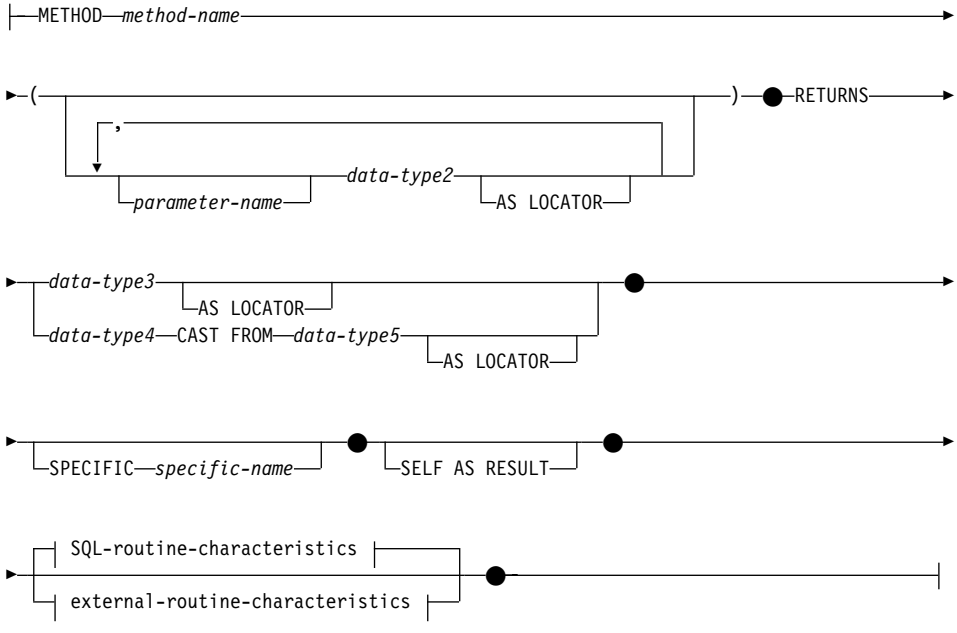
attribute-definition:



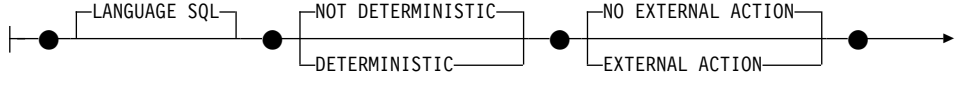
rep-type:

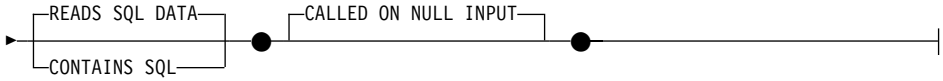


method-specification:

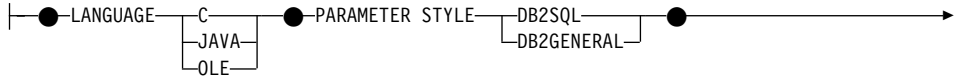


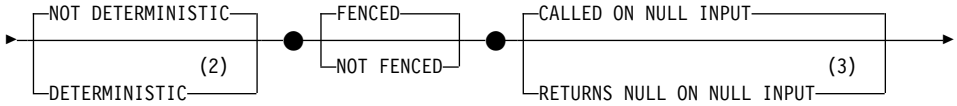
SQL-routine-characteristics:

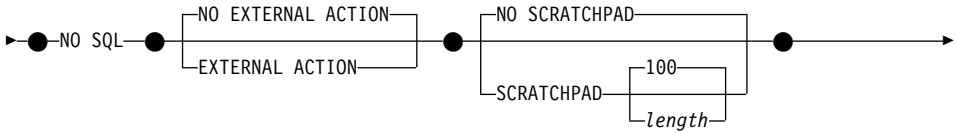


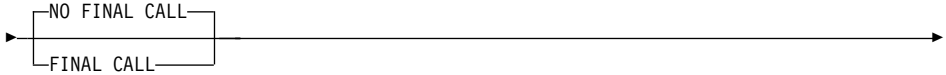


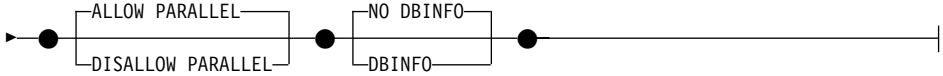
external-routine-characteristics:











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。

2 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。

3 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。

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 特殊レジスターは、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 / バインド・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多くの名前は、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)。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

2 つの部分からなる *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 (SQLSTATE 42939) になります。

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、上位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上位タイプと同じ *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多くの名前は、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)。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、 EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

data-type
属性のデータ・タイプです。 771ページの『CREATE TABLE』 (LONG VARCHAR、 LONG VARGRAPHIC を除く) でリストされているデータ・タイプの 1 つで、 LONG VARCHAR または LONG VARGRAPHIC に基づいた特殊タイプです (SQLSTATE 42601)。このデータ・タイプは既存のデータ・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。771ページの『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、タイプ付き表またはタイプ付き視点のデータ・タイプとしてのみ有効に使用することができます (SQLSTATE 01641)。

実行時に、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)。詳細は、99ページの『構造タイプ』を参照してください。

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、771ページの『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、771ページの『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、表または視点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、ヌル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、基礎表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。

タイプ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. 次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_attribute*s = 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25 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. 次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 25. 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
DATALINK(n)	n + 54

表 25. 属性データ・タイプのバイト・カウント (続き)

<table>
<tr><td rowspan="12">LOB タイプ</td><td colspan="2">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。</td></tr>
<tr><td>LOB の最大長</td><td>LOB 記述子のサイズ</td></tr>
<tr><td>1 024</td><td>72</td></tr>
<tr><td>8 192</td><td>96</td></tr>
<tr><td>65 536</td><td>120</td></tr>
<tr><td>524 000</td><td>144</td></tr>
<tr><td>4 190 000</td><td>168</td></tr>
<tr><td>134 000 000</td><td>200</td></tr>
<tr><td>536 000 000</td><td>224</td></tr>
<tr><td>1 070 000 000</td><td>256</td></tr>
<tr><td>1 470 000 000</td><td>280</td></tr>
<tr><td>2 147 483 647</td><td>316</td></tr>
<tr><td>特殊タイプ</td><td colspan="2">特殊タイプのソース・タイプの長さ。</td></tr>
<tr><td>参照タイプ</td><td colspan="2">参照タイプの基礎となる組み込みデータ・タイプの長さ。</td></tr>
<tr><td>構造タイプ</td><td colspan="2">inline_length(attribute_type)</td></tr>
</table>

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
この構造タイプを上位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のに使います。

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)。この文節が備えられているのは、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、関数表記を使えるようにするためです。

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) として使われる

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、 255 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* が付いた値を、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。文節を指定しないと、 *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。文節を指定しないと、 *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はじめて、実際に使用できるようになります (SQLSTATE 42884)。

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 識別子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多くの名前は、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)。それに含まれる名前は、 SOME、 ANY、 ALL、 NOT、 AND、 OR、 BETWEEN、 NULL、 LIKE、

EXISTS、 IN、 UNIQUE、 OVERLAPS、 SIMILAR、 MATCH、および 209ページの『基本述部』に記載されている比較演算子です。

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)。

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の手引き」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、ユーザー定義の変形関数を使用する必要があります。

DECIMAL (および NUMERIC) は、LANGUAGE C と OLE では無効です (SQLSTATE 42815)。 DECIMAL の使用に代わる手法については、*アプリケーション開発の手引き* を参照してください。

REF を指定することができますが、これには定義された効力範囲はありません。メソッドの本体で、まず参照タイプをキャストして効力範囲をもたせてはじめて、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、まずキャストして効力範囲をもたせてはじめて、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減らすことができ、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 ロケーターの使用法については *アプリケーション開発の手引き*に説明されています。

LOB 以外のタイプ、または LOB に基づく特殊タイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

関数が 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)。

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 以外のタイプ、または LOB に基づく特殊タイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

関数が 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)。

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* パラメーターに対して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 以外のタイプ、または LOB に基づく特殊タイプに対して AS LOCATOR を指定すると、エラーが発生します (SQLSTATE 42601)。

関数が 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)。

SPECIFIC *specific-name*

定義するメソッドのインスタンスに対する固有名を指定します。この名前は、メソッドの本体の作成やメソッドの除去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 識別子です (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 識別子です。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。さもないと、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。さもないと、エラーになります (SQLSTATE 42882)。

specific-name の指定がない場合、固有の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。生成される固有名は、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。
- メソッドを実装する場合、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、そしてその結果もヌルにならない (SQLSTATE 22004) ようにする必要があります。

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。

NOT DETERMINISTIC または DETERMINISTIC

この文節は任意選択ですが、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、同じ入力を行って連続して呼び出した場合に常に同じ結果を戻すことになります。 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)。

NO EXTERNAL ACTION または EXTERNAL ACTION
この文節は任意選択であり、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
この任意選択文節は、引き数がヌル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、ヌル値を戻す場合も、通常の (ヌル値以外の) 値を戻す場合もあります。ただし、ヌル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、ファミリーの互換性用の CALLED ON NULL INPUT の同義語として使うことができます。

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。

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。

DB2SQL
C 言語の呼び出しとリンクの規則、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。

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。

パラメーターの受け渡しの詳細については、*アプリケーション開発の手引き* を参照してください。

DETERMINISTIC または NOT DETERMINISTIC
この文節は任意選択ですが、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、同じ入力を行って連続して呼び出した場合に常に同じ結果を戻すことになります。 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空間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、その内部資源 (データ・バッファーなど) を隔離して、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: 十分に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 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。

FENCED を使用すると、NOT FENCED よりもデータベースの保全性の保護は強化されますが、十分なコード化、見直し、およびテストが施されないと、 FENCED メソッドでも DB2 で不用意に障害が生じる原因になることがあります。

ほとんどのメソッドは、FENCED または NOT FENCED のどちらでも実行できるはずです。 LANGUAGE OLE 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 の場合、AS LOCATOR 文節を指定できません (SQLSTATE 42613)。

FENCED から NOT FENCED に変更するには、メソッドを削除してから再作成し、メソッドを再登録する必要があります。

メソッドを NOT FENCED として登録するには、SYSADM 権限、DBADM 権限、または特殊な権限 (CREATE_NOT_FENCED) が必要です。

RETURNS NULL ON NULL INPUT または CALLED ON NULL INPUT
この任意選択文節を使用すると、非サブジェクト引き数のいずれかがヌル値の場合に、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ヌル値の場合、このメソッドは呼び出されず、結果はヌル値になります。

CALLED ON NULL INPUT を指定すると、ヌル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、ヌル値を戻す場合も、通常の (ヌル値以外の) 値を戻す場合もあります。ただし、ヌル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、上位互換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 の同義語として使えます。

次の 2 通りの場合、この指定は無視されます。

- 対象となる引き数がヌルの場合。この場合、メソッドは実行されずに結果はヌルになります。
- 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、このヌル引き数条件が成立することはありません。

NO SQL
この文節は必須で、メソッドが SQL ステートメントを発行してはならないことを指定します。発行すると、実行時にエラーになります (SQLSTATE 38502)。

EXTERNAL ACTION または NO EXTERNAL ACTION
この文節は任意選択であり、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*
この文節は任意選択であり、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* を指定すると、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。
- その効力範囲は、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、その効力範囲は上記とは異なります。メソッドが複数の区分で実行される場合、メソッドが処理されるそれぞれの区分において、 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、区画内並行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資源 (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資源が獲得される場合、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資源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ス指定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

この文節は任意選択であり、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、獲得したシステム資源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資源を獲得し、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。外部メソッドが資源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、次の時点で行われます。
  - ステートメント終了時。これは、カーソルの関係する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

この文節は任意選択で、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 つの区分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区分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

この文節は任意選択で、 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)。

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、固有のアプリケーション 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、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のもとで、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。

構造の詳細、および構造がメソッドにどのように渡されるかについては、 *アプリケーション開発の手引き*を参照してください。

注

- 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 特権は PUBLIC に与えられます。

- 属性なしで定義された構造化サブタイプは、属性をすべて上位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。
- 以下がすべてあてはまる場合、サブジェクト・タイプ T を備えたメソッド MT は、サブジェクト・タイプ S を備えた別のメソッド MS をオーバーライドすると定義されます。
  - MT と MS は、同じ非修飾名および同数のパラメーターをもっている
  - T は、S の適正なサブタイプである
  - MT の非サブジェクト・パラメーター・タイプは、それに対応する MS の非サブジェクト・パラメーター・タイプと同じである。ただし「同じ」とは、長さや精度に関係なく、VARCHAR などの基本タイプについて言います。

  どのメソッドも、別のメソッドをオーバーライドしたり、別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った場合、 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 のスーパー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。
- 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)。構造タイプを除去しないで、生成された関数またはメソッドを除去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。[85]

  参照タイプの効力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)
    ```

85. 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 が共通の上位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)。

- *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 (1126ページの『SET PATH』、またはアプリケーション開発の手引き で説明されている FUNCPATH BIND オプションを参照してください)。

- 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ヌルになります。

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 の属性をもっていると、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

- オブザーバー・メソッド

構造タイプの各属性ごとにオブザーバー・メソッドが定義されます。オブザーバー・メソッドは、各属性ごとに属性タイプを戻します。対象がヌルの場合、オブザーバー・メソッドは、属性タイプのヌル値を戻します。

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。

生成されるオブザーバー・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。

- ミューテーター・メソッド

  構造タイプの各属性ごとに、タイプ保存のミューテーター・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、ミューテーター・メソッドを使います。ミューテーター・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 のインスタンスをミュート (消音) することができます。サブジェクトがヌルの場合、ミューテーター・メソッドはエラーを生じます (SQLSTATE 2202D)。

  生成されるミュテーター・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、定数を使用するミューテーション (消音) をサポートするため、次のような追加のミューテーター・メソッドが生成されます。

  - *attribute-type* が SMALLINT の場合、追加のミューテーターはタイプ INTEGER の引き数をサポートします。
  - *attribute-type* が REAL の場合、追加のミューテーターはタイプ DOUBLE の引き数をサポートします。
  - *attribute-type* が CHAR の場合、追加のミューテーターはタイプ VARCHAR の引き数をサポートします。
  - *attribute-type* が GRAPHIC の場合、追加のミューテーターはタイプ 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変形関数を再検査し、必要があれば更新しなければなりません。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変形関数を変更して、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) に関連した一連の変形関数であるため、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、変形関数を作成しなければなりません。

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副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 DB2SQL
        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
```

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。

- 統合データベースに定義される予定の、データ・ソース表または視点の列のデータ・タイプ。
- 統合データベースに定義済みの、対応するデータ・タイプ。

マッピングを行うと、統合データベースのデータ・タイプを、 (1) 指定したデータ・ソースか (2) 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) のいずれかにあるデータ・タイプに関連付けることができます。

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。

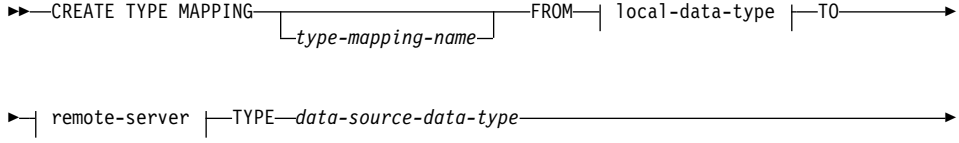
呼び出し

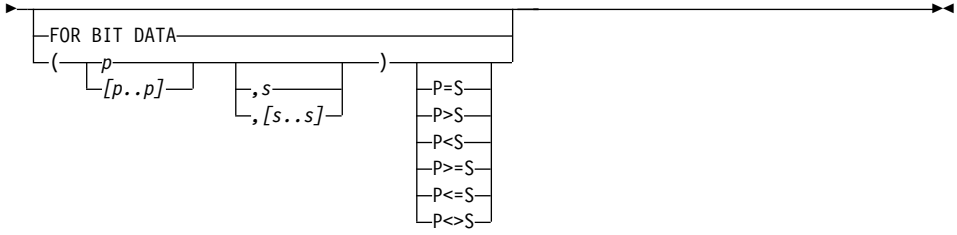
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

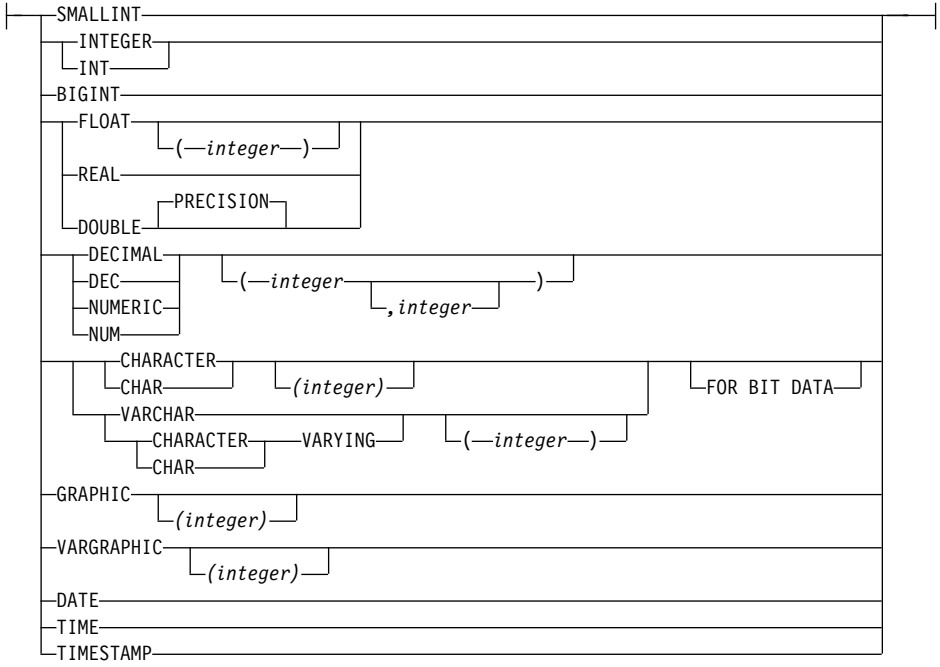
このステートメントの許可 ID が持つ特権には、 SYSADM または DBADM 権限が含まれている必要があります。

構文

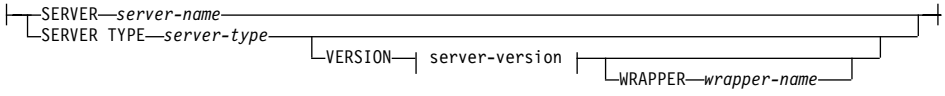




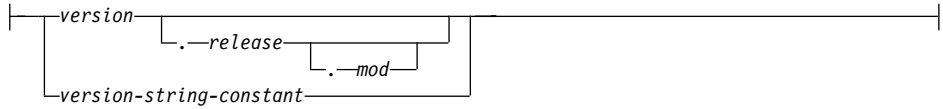
local-data-type:



remote-server:



server-version:



説明

type-mapping-name
データ・タイプの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・タイプの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、固有の名前が生成されます。

local-data-type
統合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* 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 前処理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)。 *local-data-type* に長さや精度 (および位取り) が指定されない場合は、 *source-data-type* の値が使用されます。

local-data-type を、 LONG VARCHAR、 LONG VARGRAPHIC、 DATALINK、ラージ・オブジェクト (LOB) タイプ、またはユーザー定義タイプにすることはできません (SQLSTATE 42806)。

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。 *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-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、統合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*
local-data-type にマッピングされているデータ・ソースのデータ・タイプを指定します。 *data-source-data-type* が、データ・ソースでスキーマ名によって修飾されている場合、この修飾子は、使用することができますが、必須ではありません。

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。タイプに短形式と長形式 (たとえば、CHAR と CHARACTER) がある場合には、短形式を指定してください。

p 10 進データ・タイプの場合、 *p* は値を入れられる桁数の最大数を指定します。他のすべての文字データ・タイプの場合、 *p* は値を入れられる最大文字数を指定します。 *p* は、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 (SQLSTATE 42611)。データ・タイプで *p* が必要なのにこれを指定しない場合、最もよく一致するものがシステムによって指定されます。

[p..p]
10 進データ・タイプの場合、 *[p..p]* は値を入れられる桁数の最小および最大数を指定します。他のすべての文字データ・タイプの場合、 *[p..p]* は値を入れられ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 (SQLSTATE 42611)。

s 10 進データ・タイプの場合、*s* は小数点以下の桁数の最大数を指定します。この数値は、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 (SQLSTATE 42611)。データ・タイプで数値が必要なのにこれを指定しない場合、最もよく一致するものがシステムによって指定されます。

[s..s]

10 進データ・タイプの場合、 *[s..s]* 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 (SQLSTATE 42611)。

P [operand] **S**

10 進データ・タイプの場合、 P [operand] S は最大許容精度と小数点以下の最大桁数との比較を示します。 たとえば、オペランド = は、最大許容精度と小数部分に許容できる最大桁数が同じであることを示します。 P [operand] S は、検査が必要な水準にある場合にのみ指定します。

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 2 進値が含まれる場合、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。

- ステートメントが 1 つのデータ・ソースを参照していて、UOW にはすでに、このデータ・ソース内の表または視点のニックネームを参照する SELECT ステートメントが含まれる場合。
- ステートメントがデータ・ソースの 1 つのカテゴリーを参照していて (たとえば、特定のタイプおよびバージョンの全データ・ソース)、UOW にはすでに、これらのデータ・ソースの 1 つに含まれる表または視点のニックネームを参照する SELECT ステートメントがある場合。

例

例 1: すべての Oracle データ・ソースで、 SYSIBM.DATE と Oracle データ・タイプ DATE の間のマッピングを作成します。

```
CREATE TYPE MAPPING MY_ORACLE_DATE
   FROM SYSIBM.DATE
   TO SERVER TYPE ORACLE
   TYPE DATE
```

例 2: データ・ソース ORACLE1 で、 SYSIBM.DECIMAL(10,2) と Oracle データ・タイプ NUMBER([10..38],2) との間のマッピングを作成します。

```
CREATE TYPE MAPPING MY_ORACLE_DEC
   FROM SYSIBM.DECIMAL(10,2)
   TO SERVER ORACLE1
   TYPE NUMBER([10..38],2)
```

CREATE USER MAPPING

CREATE USER MAPPING ステートメントは、統合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。

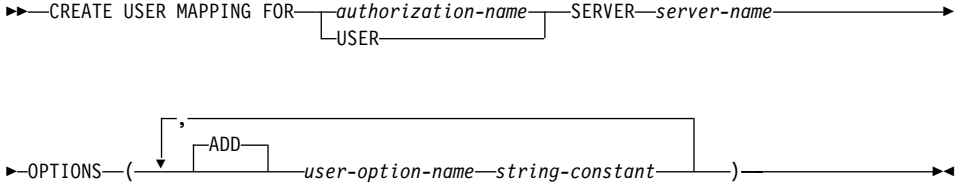
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

ステートメントの許可 ID がデータ・ソースにマッピングされている許可名と異なる場合、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、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統合データベースへ接続するときの、許可名を指定します。この名前は、*server-name* によって表されるデータ・ソースがアクセスできる識別子にマップする必要があります。

USER

特殊レジスター USER の値。 USER を指定すると、CREATE 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*

マッピング許可 ID にアクセスできるデータ・ソースを指定します。

OPTIONS

使用可能にするユーザー・オプションを指定します。 *user-option-name* とその設定についての詳細は、1361ページの『ユーザー・オプション』を参照してください。

ADD

1 つかそれ以上のユーザー・オプションを使用可能にします。

user-option-name

作成中のユーザー・マッピングを完了するのに使用するユーザー・オプションを指名します。

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。

注

- マッピングに含めるデータ・ソースで、表または視点のニックネームを参照する SELECT ステートメントが作業単位 (UOW) に含まれている場合には、指定した UOW にユーザー・マッピングを作成することはできません。

例

例 1: S1 というデータ・ソースにアクセスするには、ローカル・データベースの許可名とパスワードを S1 のユーザー ID とパスワードにマップする必要があります。許可名は RSPALTEN、S1 で使用するユーザー ID とパスワードは、それぞれ SYSTEM および MANAGER です。

```
CREATE USER MAPPING FOR RSPALTEN
   SERVER S1
   OPTIONS
   ( REMOTE_AUTHID 'SYSTEM',
      REMOTE_PASSWORD 'MANAGER' )
```

例 2: Marc は DB2 データ・ソースにすでにアクセスしています。 ここで、特定の DB2 表と Oracle 表を結合するには、 Oracle のデータ・ソースにアクセスする必要があります。 それには、Oracle データ・ソースのユーザー名とパスワードが必要です。ユーザー名は統合データベースの許可 ID と同じですが、 Oracle と統合データベースのパスワードが異なっています。統合データベースから Oracle にアクセスできるようにするには、 2 つのパスワードを一緒にマップする必要があります。

```
CREATE USER MAPPING FOR MARCR
SERVER ORACLE1
OPTIONS
( REMOTE_PASSWORD 'NZXCZY' )
```

CREATE VIEW

CREATE VIEW ステートメントは、1 つまたは複数の表、視点、またはニックネームに基づく視点を作成します。

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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、または
- 全選択に指定された表、視点、またはニックネームのそれぞれに対して、
  - その表または視点に対する CONTROL 特権、または
  - その表または視点に対する 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 (視点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 (視点のスキーマ名が既存のスキーマを指している場合)。

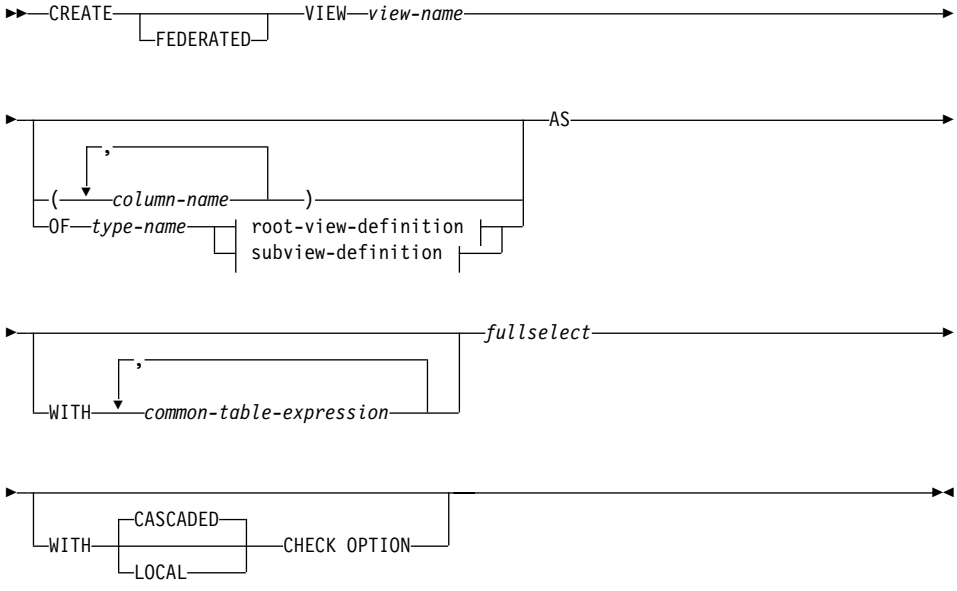
  副視点を作成するには、このステートメントの許可 ID が次の条件に適合している必要があります。
  - 表階層のルート表の定義者と同じ名前である。
  - 副視点の基礎表に対して SELECT WITH GRANT 権限を持っているか、またはスーパー視点がこの視点の定義者以外にはだれにも SELECT 権限を付与していない。

グループ特権は、CREATE VIEW ステートメントで指定された表や視点に対しては考慮されません。

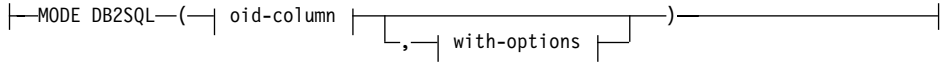
特権は、連合データベースのニックネームに視点を定義するときには考慮されません。 このニックネームで示されている表または視点のデータ・ソースの許可要件は、照会の処理時に適用されます。ステートメントの許可 ID は、別のリモート許可 ID へマップできます。

視点の定義者が SYSADM 権限を持つために、視点の作成しかできない場合、視点作成のため、その定義者には明示的な DBADM 権限が付与されます。

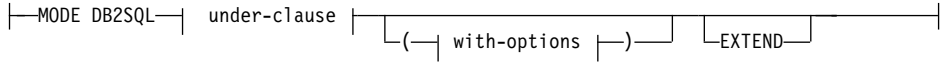
構文



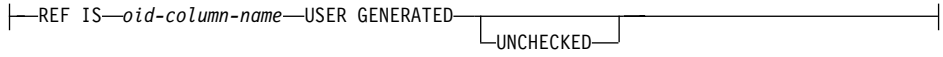
root-view-definition:



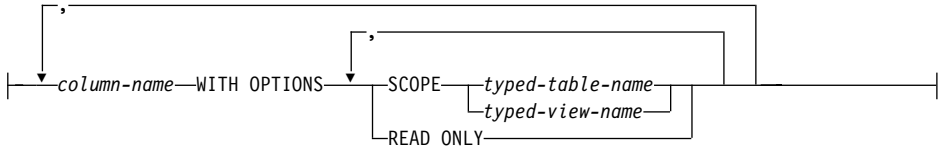
subview-definition:



oid-column:



with-options:



under-clause:

注: *common-table-expression* (共通表式) と *fullselect* (全選択) の構文については、 421ページの『第5章 照会』 を参照してください。

説明

FEDERATED

作成されている視点がニックネームまたは OLEDB 表関数を参照することを示します。この FEDERATED キーワードが指定されていない状態で、OLEDB 表関数やニックネームを全選択して直接または間接的に参照すると、 CREATE VIEW ステートメントを送信する際に警告 (SQLSTATE 01639) が出されます。ただし、警告が出されても視点は作成されます。

逆に、この FEDERATED キーワードが指定されているのに、 OLEDB 表関数やニックネームを全選択して直接または間接的に参照しないと、CREATE VIEW ステートメントを送信する際にエラー (SQLSTATE 429BA) が発生します。この場合、視点は作成されません。

view-name

視点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、視点、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、作動不能な視点の名前と同じであっても構いません (906ページの『作動不能視点』を参照)。このような場合、作動不能な視点は、CREATE VIEW ステートメントに指定した新しい視点によって置き換えられます。作動不能な視点が置き換えられると、ユーザーに警告 (SQLSTATE 01595) が出されます。バインド・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
視点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、固有で、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、視点の列は、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*
視点の列が *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 前処理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、タイプ付き視点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*
この視点が *superview-name* の副視点であることを指定します。 スーパー視点は既存の視点でなければならず (SQLSTATE 42704)、この視点は *type-name* のすぐ上位にある上位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* および *superview-name* のスキーマ名は、同一のものでなければなりません (SQLSTATE 428DQ)。 *superview-name* で指定される視点には、 *type-name* で既に定義された既存の副視点を含めることはできません (SQLSTATE 42742)。

表の列には、スーパー視点のオブジェクト ID 列が含まれています。オブジェクト識別列のタイプは 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、上位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
スーパー視点に対して SELECT 特権を持つユーザーやグループはすべて、

新しく作成した副視点に対しても同様の特権を付与されます。この特権は、副視点定義者によって付与されたものとみなされます。

OID-column
タイプ付き視点のオブジェクト ID 列を定義します。

REF IS *OID-column-name* **USER GENERATED**
オブジェクト ID 列 (OID) を視点の最初の列として定義することを指定します。視点階層のルート視点には、OID が必須です (SQLSTATE 428DX)。この視点は副視点以外のタイプ付き視点 (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* のいかなる属性の名前と同一であることはできません (SQLSTATE 42711)。全選択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 (基本キー、固有制約、固有索引、または OID 列) を使って固有なものにできる列 (ヌル可能ではない) 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、タイプ付き視点定義のオブジェクト ID の列を固有であるとみなすように定義します。この属性は、次のようなタイプ付き視点階層に定義されている表または視点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視点にまでわたっている視点の階層に必須のオプションです。 UNCHECKED を指定する場合、ユーザーの責任で視点の各行に固有の 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、視点に重複した OID 値が入ってしまうと、固有でない 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
タイプ付き視点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、視点の *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、すでにスーパー視点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

参照タイプ列の効力範囲を指定します。 参照解除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、すべて効力範囲を指定しなければなりません。

ターゲット表またはターゲット視点が定義されるように、後続する ALTER VIEW ステートメント (効力範囲が継承されていない場合) 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、相互参照表および相互参照視点の場合に適用する)。 視点の参照タイプ列で効力範囲が指定されていないのに、基礎表または視点列の効力範囲が指定された場合、基礎列の効力範囲は参照タイプ列によって継承されます。 基礎表または視点の列に効力範囲がない場合には、この列に効力範囲は指定されません。 読み取り専用カーソルと更新可能カーソルの詳細については、 904ページの『注』を参照してください。

typed-table-name

タイプ付き表の名前。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

タイプ付き視点の名前。この視点は既に存在しているものか、作成する視点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、副視点定義により、暗黙的に読み取り専用になっている同じ列の式を指定するために使用します。

AS

視点定義を指定します。

WITH *common-table-expression*

後に続く全選択で使用する共通表式 (*common-table-expression*) を定義します。タイプ付き視点を定義するときに、共通表式は指定できません。 468 ページの『共通表式』 を参照してください。

fullselect

視点を定義します。どのような場合でも、視点は、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。 fullselect でホスト変数、パラメーター・マーカー、または宣言された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視点を SQL 表関数として作成することは可能です。 701ページの『CREATE FUNCTION (SQL スカラー、表、または行)』を参照してください。

タイプ付き視点および副視点の場合 : *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、NODENUMBER または PARTITION 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。
- 視点の本体は、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。視点の本体に直接加わっている各副選択を、視点の分岐 と呼びます。視点には、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、単一の表または視点 (その分岐の*基礎* 表または視点といい、必ずしもタイプ付きではない) で構成される必要があります。
- 各分岐の基礎表または視点は、別々の階層にする必要があります (つまり、視点は、同じ階層内の基礎表または視点が付いた複数の分岐を持つことはできません)。
- タイプ付き視点定義の分岐はいずれも GROUP BY または HAVING を指定できません。
- 視点の本体に UNION ALL が含まれる場合、階層内にあるルート視点の OID 列に UNCHECKED オプションを指定する必要があります。

視点および副視点の階層の場合 : BR1 および BR2 が、階層内の視点定義に現れる分岐になるようにします。 T1 を BR1 の基礎表または視点に、T2 を BR2 の基礎表または視点にします。 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、視点階層にあるルート視点の 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タイプ付きの視点の場合 : 副視点の本体内の各分岐について :

- 各分岐の基礎表は、即時スーパー視点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、副視点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タイプ付き副視点の場合:

- 副視点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、副視点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)。
- 副視点で指定した階層上のそれぞれの分岐の OID 式は、ルート視点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)。
- スーパー視点内の 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、その副視点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

視点によって挿入または更新される行すべてが、視点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。 視点の定義に従わない行とは、視点の探索条件を満たしていない行です。

WITH CHECK OPTION は、視点が読み取り専用の場合には指定できません (SQLSTATE 42813)。 挿入が許されていない更新可能な視点に対して WITH CHECK OPTION を指定すると、制約は更新にのみ適用されます。

視点が、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、 WITH CHECK OPTION を指定することはできません (SQLSTATE 42997)。

WITH CHECK OPTION は、視点がタイプ付き視点の場合には指定できません (SQLSTATE 42997)。

WITH CHECK OPTION は、ニックネームが視点の更新目標である場合には指定できません。

WITH CHECK OPTION を省略すると、視点を使用するどのような挿入操作または更新操作の検査においても、視点の定義は使用されません。 ただし、視点が WITH CHECK OPTION を含む他の視点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、何らかの検査が行われる場合があります。 この場合、視点の定義が使用されるわけではないため、視点の定義に従っていない視点を介して、行が挿入または更新される可能性があります。

CASCADED

視点 *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視点から、制約としての探索条件を *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視点も、このような制約の対象になります。 したがって、*V* の探索条件と、 *V* が従属している各視点の探索条件との AND を取ったものが、 *V* あるいは *V* に従属するいずれかの視点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

視点 *V* に対する WITH LOCAL CHECK OPTION 制約は、 *V* の探索条件が、 *V* または *V* に従属するいずれかの視点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視点を想定します (Y は、一覧表の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 : 表 T に基づいて定義される視点
V2 : V1 に基づいて WITH Y CHECK OPTION として定義される視点
V3 : V2 に基づいて定義される視点
V4 : V3 に基づいて WITH Y CHECK OPTION として定義される視点
V5 : V4 に基づいて定義される視点
```

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探索条件を示しています。

	Y が LOCAL の場合	**Y が CASCADED の場合**
V1 での検査条件:	対象となる視点なし	対象となる視点なし
V2 での検査条件:	V2	V2、V1
V3 での検査条件:	V2	V2、V1
V4 での検査条件:	V2、V4	V4、V3、V2、V1
V5 での検査条件:	V2、V4	V4、V3、V2、V1

また、次のような更新可能視点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視点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)
```

注

- まだ存在していないスキーマ名を用いて視点を作成すると、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 特権は PUBLIC に与えられます。
- 視点列は、列が式から取り出されるときを除いて、基礎表または視点の NOT NULL WITH DEFAULT 属性を継承します。 基礎表に制約が定義されているなら、更新可能視点に行が挿入または更新されるとき、それらの制約 (基本キー、参照保全性、および検査制約) に対する検査が行われます。

- 定義の中で作動不能視点を使用して新しい視点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、宣言された一時表をサポートしていません (SQLSTATE 42995)。
- ***削除可能視点:*** 次のいずれかが当てはまる場合、視点は*削除可能* です。
  - 外部全選択の各 FROM 文節には、基礎表 (OUTER 文節なし)、削除可能視点 (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 を除くセット演算 (UNION、EXCEPT、または INTERSECT) が含まれない
  - UNION ALL のオペランドの基礎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
- ***更新可能視点:*** 次の条件がすべて当てはまる場合には、視点の列は*更新可能* です。
  - 視点が削除可能である
  - 列の解決結果が基礎表の列 (非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
  - 視点の全選択に 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)

  視点のいずれかの列が更新可能なら、視点は*更新可能* です。
- ***挿入可能視点:***

  視点のすべての列が更新可能であり、視点の全選択に UNION ALL が含まれない場合、その視点は*挿入可能* です。
- ***読み取り専用視点:*** 視点が*読み取り専用* であるのは、削除可能でない場合です。

  視点が読み取り専用かどうかは、SYSCAT.VIEWS カタログ視点の READONLY 列に示されます。

- 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。
- ***作動不能視点:*** *作動不能視点* とは、 SQL ステートメントで使用できなくなった視点のことです。視点は、次の場合に作動不能になります。
  - 視点定義が従属している特権が取り消された場合
  - 視点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除去された場合
  - 視点定義が従属している視点が作動不能になった場合
  - 視点定義のスーパー視点である視点 (副視点) が作動不能になった場合

  実際には、作動不能視点とは、視点定義が間違って除去された視点です。たとえば、別名が除去されると、その別名を使用して定義されている視点すべてが作動不能になります。それに従属するすべての視点も作動不能になり、その視点に従属するパッケージは無効になります。

  作動不能視点を明示的に作成し直すか、あるいは除去する時点まで、その作動不能視点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。作動不能視点が明示的に削除されるまで、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)。

  作動不能視点は、作動不能視点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。この視点定義テキストは、SYSCAT.VIEWS カタログの TEXT 列に保管されます。作動不能視点を再作成する場合は、他のユーザーでその視点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、視点が作動不能とみなされると、視点のすべての許可レコードが削除されるためです。作動不能視点を再作成するために、それを明示的に削除する必要はありません。作動不能視点と同じ *view-name* を指定して CREATE VIEW ステートメントを発行すると、作動不能視点は置き換えられ、CREATE VIEW ステートメントは警告を戻します (SQLSTATE 01595)。

  作動不能視点であることは、SYSCAT.VIEWS カタログ視点の VALID 列の X、また SYSCAT.TABLES カタログ視点の STATUS 列が X であることによって示されます。

- ***特権***

  視点の定義者は、視点に対する SELECT 特権と、視点を除去する権利を常に与えられます。視点の定義者は、その定義者が全選択で指定されたすべての基礎表、視点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、その視点に対する CONTROL 特権が付与されます。

視点の定義者は、その視点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、その視点に対する INSERT、UPDATE、列レベルの UPDATE、または DELETE の特権を与えられます。

視点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視点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、連合データベースのニックネームに視点を定義するときには考慮されません。ただし、ニックネームに視点を使用する場合、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視点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。視点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。

副視点が作成されると、すぐ上のスーパー視点に対して持っている SELECT 特権が自動的に副視点に対しても付与されます。

- ***効力範囲および REF 列***

  視点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・タイプと効力範囲について考慮してください。

  – 必要なターゲット・タイプおよびスコープが基礎表または基礎視点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。
  – 効力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、必要な効力範囲の表と視点を定義します。
  – 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しなければなりません。この場合、効力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)
    ```

- ***識別列*** 視点の列は、視点定義の全選択内にある対応する列のエレメントが表の識別列の名前であるか、基礎表の識別列の名前に直接または間接的にマップされた視点の列の名前である場合に識別列とみなされます。

これ以外の場合はすべて、視点の列は ID のプロパティーを取得しません。たとえば、次のような場合があります。

- 視点定義の選択リストに識別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
- 視点定義に結合が関与している場合
- 視点定義の列に識別列を参照する式が含まれている場合
- 視点定義に UNION が含まれている場合

挿入先の視点において、視点定義の選択リストに直接または間接的に基礎表の識別列の名前が含まれている場合は、 INSERT ステートメントが基礎表の識別列を直接参照する場合と同じ規則が適用されます。

- ***連合視点*** 連合視点とは、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視点のことです。この種のニックネームが存在する場合、その視点で使用する許可モデルは、視点の作成時や、視点が照会で順番に参照されるときに変更されます。ニックネームの参照や FEDERATED キーワードを含めずに視点を作成すると、ニックネームに対する参照が原因で、この視点の許可要件が異なることを示すエラー・メッセージが発行されます。

  ニックネームには DML 特権との関連が全くないため、視点を作成しても、視点定義者がニックネームや、基礎データ・ソース表または視点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。連合データベースでの表または視点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 特権を視点定義者に要求することにより、通常どおり行われます。

  連合視点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・ソース表または視点にアクセスするのに必要な特権がなければなりません。連合視点を参照する照会を発行する許可 ID には、連合サーバーに存在する (連合でない) 表または視点に対する追加の特権は必要ありません。

例

例 1: PROJECT 表に基づく視点 MA_PROJ を作成します。この視点には、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 と同様に視点を作成します。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 と同様の視点を作成します。ただし、視点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、視点の 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 という名前の視点を作成します。この視点には、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 と同様の視点を作成します。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
```

全選択に、TOTAL_PAY として式 SALARY+BONUS+COMM を指定することによって、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: 次の図に示す表と視点があると想定します。

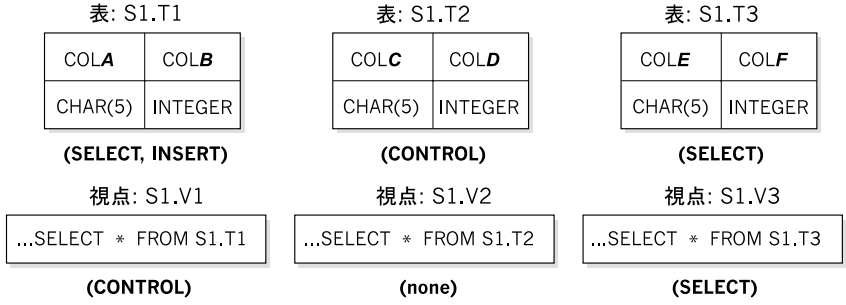


図 12. 例 6 の表と視点

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視点に対して、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。[86] 基礎となる基礎表に対して、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、次のような視点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。基礎となる基礎表 (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視点に対して、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、視点 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。視

86. 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。

点 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視点に対して、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 つの視点 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. 以下の視点定義では、ZORPIE は視点 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、視点 VE の作成には S1.V3 に対する SELECT 特権だけが必要です。視点の定義者は、視点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、その視点に対して 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。

CREATE WRAPPER

CREATE WRAPPER ステートメントは、ラッパー (連合サーバーがデータ・ソースの特定のカテゴリーと対話するためのメカニズム) を連合データベースに登録します。

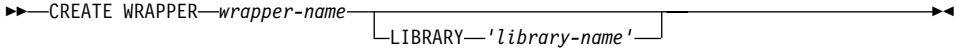
呼び出し

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、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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 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

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、連合サーバーは *'library-name'* に自動的にデフォルトを割り当てます。

  以下の事前定義名を使用できます。

DRDA	すべての DB2 ファミリーのデータ・ソース用。
NET8	Oracle 社の Net8 クライアント・ソフトウェアがサポートするすべての Oracle データ・ソース用。
OLEDB	Microsoft OLE DB がサポートするすべての OLE DB Provider 用。
SQLNET	Oracle 社の SQL*Net クライアント・ソフトウェアがサポートするすべての Oracle データ・ソース用。

- ユーザーが指定する名前。こうした名前を提供するには、*'library-name'* も指定する必要があります。

LIBRARY *'library-name'*

ラッパー・モジュールを含むファイルの名前を指定します。 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。事前定義された *wrapper-name* を指定する場合は、デフォルトで次のようなファイル名が使用されます。

表 26. LIBRARY オプションのデフォルトのファイル名

プラットフォーム	DRDA	SQLNET	NET8	OLEDB
AIX	**libdrda.a**	**libsqlnet.a**	**libnet8.a**	-
HP-UX	**libdrda.sl**	**libsqlnet.sl**	**libnet8.sl**	-
SOLARIS	**libdrda.so**	**libsqlnet.so**	**libnet8.so**	-
UNIX	**libdrda.a**	**libsqlnet.a**	**libnet8.a**	-
WINNT	**drda.dll**	**sqlnet.dll**	**net8.dll**	**db2oledb.dll**

注

ラッパーの選択および定義を行う方法については、*インストールおよび構成 補足* を参照してください。

例

例 1: 連合サーバーが、 Oracle 社の SQL*Net クライアント・ソフトウェアがサポートする Oracle データ・ソースと対話するために使用できるラッパーを登録します。事前定義名を使用します。

```
CREATE WRAPPER SQLNET
```

例 2: AIX システムで稼働する連合サーバーが、 DB2 (VM および VSE 版) データ・ソースと対話するために使用できるラッパーを登録します。これらのデータ・ソースをテストで使用していることを示す名前を指定します。

```
CREATE WRAPPER TEST
   LIBRARY 'libsqlds.a'
```

ライブラリー名の拡張子 (a) は、ラッパー TEST が AIX システムに常駐するデータ・ソース用であることを示します。

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 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

“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 ステートメントに指定する表または視点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視点のそれぞれに対する以下のいずれかの特権。
  - 表または視点に対する SELECT 特権、または
  - 表または視点に対する CONTROL 特権。

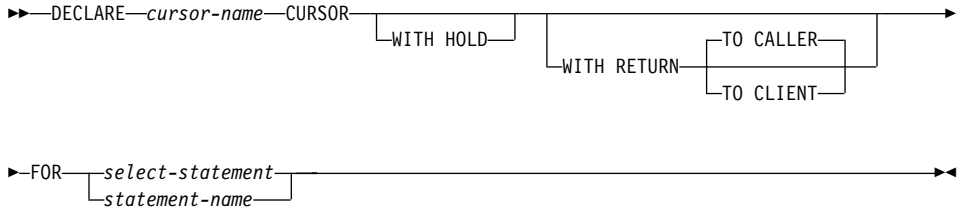
statement-name を指定した場合:

- ステートメントの許可 ID は、実行時許可 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（1036ページの『OPEN』 を参照してください)。

WITH HOLD

複数の作業単位を通して資源を維持します。 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。そうしないと、(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を除いて、すべてのロックがリリースされます。 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。

• 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 を使用して宣言されたカーソルが 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 WITH RETURN 文節は効力を持たず、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みなされます。

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。

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。 *select-statement* については、467ページの『選択ステートメント』 を参照してください。

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、1042ページの『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 で同一の値が与えられます。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。 カーソルを定位置 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。
- 次のすべてが真の場合、カーソルは*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、基礎表または削除可能視点 (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 を除くセット演算 (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
  - 選択ステートメントに ORDER BY 文節が含まれない
  - 選択ステートメントに FOR READ ONLY 文節が含まれない [87]
  - 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 [88]
    - カーソルが静的に定義されている
    - LANGLEVEL バインド・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、次のすべてが真の場合に、*更新可能* です。
  - カーソルが削除可能である

87. FOR READ ONLY 文節については、 475ページの『READ ONLY 文節』で定義されています。

88. FOR UPDATE 文節については、 474ページの『UPDATE 文節』を参照してください。

- 列の解決結果が基礎表の列となる
- LANGLEVEL バインド・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。

次のすべてが真である場合、カーソルは*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 バインド・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 バインド・オプションが ALL の場合には読み取り専用とみなされます。そうでない場合は、削除可能とみなされます。

- 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。 564ページの『注』を参照してください。

例

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';
```

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。宣言された一時表の記述は、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。同じ名前の宣言された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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。

許可

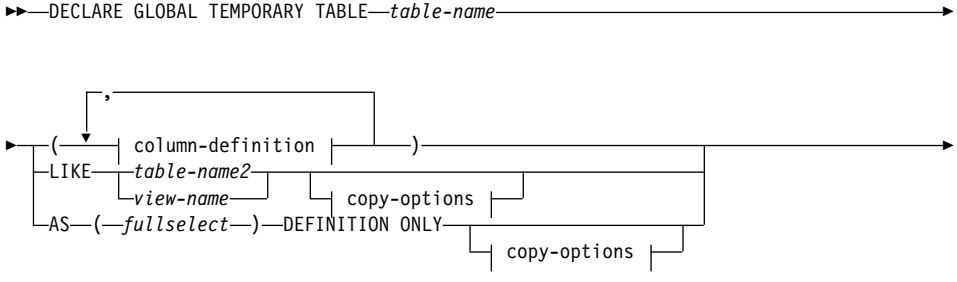
このステートメントの許可 ID には、以下の権限が少なくとも 1 つ含まれている必要があります。

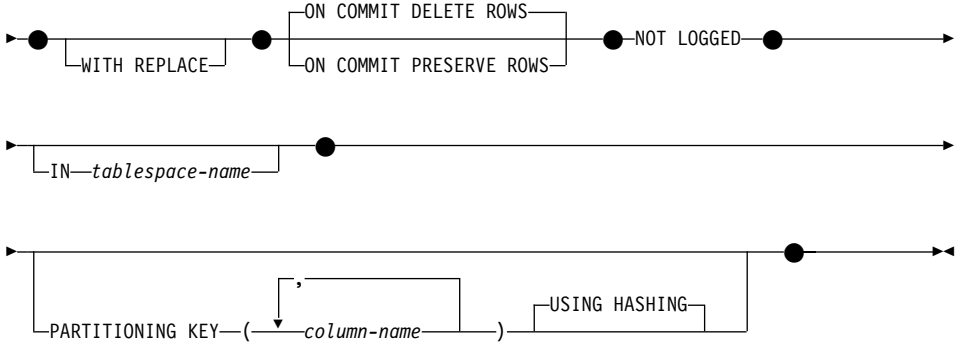
- SYSADM または DBADM 権限
- USER TEMPORARY 表スペースでの USE 権限

LIKE または全選択を使用して表を定義する場合、非分離のストアード・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、識別されているそれぞれの表または視点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。

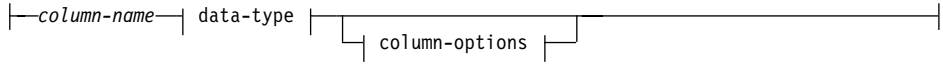
- その表または視点に対する SELECT 特権
- その表または視点に対する CONTROL 特権
- SYSADM または DBADM 権限

構文

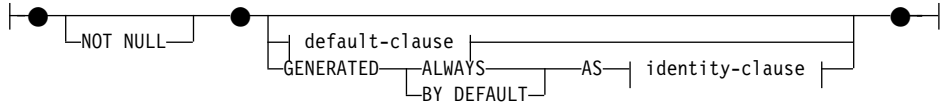




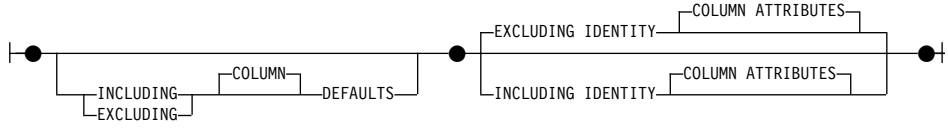
column-definition:



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* の宣言されたグローバル一時表を定義するセッションであっても、その宣言された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* を使用する宣言された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、視点、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、宣言された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。

- 最大 500 列の 4K ページ・サイズ。列のバイト・カウントは 4K ページ・サイズで 4005 を超えてはなりません。詳細については、820ページの『行サイズ』を参照してください。
- 最大 1 012 列の 8K ページ・サイズ。列のバイト・カウントは 8 101 を超えてはなりません。詳細については、820ページの『行サイズ』を参照してください。
- 最大 1 012 列の 16K ページ・サイズ。列のバイト・カウントは 16 293 を超えてはなりません。詳細については、820ページの『行サイズ』を参照してください。
- 最大 1 012 列の 32K ページ・サイズ。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、820ページの『行サイズ』を参照してください。

data-type

使用可能なタイプについては、 771ページの『CREATE TABLE』で *data-type* を参照してください。宣言された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
列にヌル値が入るのを防止します。ヌル値の指定については、771ページの『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、 771ページの『CREATE TABLE』の *default-clause* を参照してください。

identity-clause
識別列の指定については、 771ページの『CREATE TABLE』の *identity-clause* を参照してください。

LIKE *table-name2* **または** *view-name*
表の列の名前と記述が、指定された表 (*table-name2*)、視点 (*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、カタログか宣言された一時表に存在している表、視点、またはニックネームを識別するものでなければなりません。タイプ付き表またはタイプ付き視点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、指定した表または視点に含まれる列の数を表します。

- 表を指定した場合の暗黙的な定義では、*table-name2* の各列について、その列名、データ・タイプ、およびヌル可能特性を定義します。EXCLUDING COLUMN DEFAULTS を指定しないと、列のデフォルト値も入ります。
- 視点を指定すると、暗黙的な定義には *view-name* に指定した全選択のおのおのの結果列の列名、データ・タイプ、およびヌル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。

指定した表や視点のこの他の属性は、暗黙的な定義には含めません。したがって、新しい表には固有制約、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-name2 や *view-name* には、作成されてい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、宣言された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、それぞれ固有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固有の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、およびヌル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、視点、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内にデフォルトが定義されないことになります。

LIKE *table-name2* が指定されており、かつ *table-name2* が基礎表か宣言された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、しかも *table-name* が基礎表または一時表を指定する場合を除きます。

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識別列の属性 (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、視点、または全選択内の対応する列のエレメントが、識別のプロパティーが含まれている基礎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、または視点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識別のプロパティーは定義されません。たとえば、次のような場合があります。

- 全選択の選択リストに識別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識別列が含まれている (つまり、結合が関与している) 場合
- 識別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。これはデフォルト値です。

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。たとえば、作業単位 (または保管点) で表に変更を加えていた場合は、 ROLLBACK (または ROLLBACK TO SAVEPOINT) 操作を実行すると表の行がすべて削除されます。また、作業単位 (または保管点) で表を作成していた場合には、その表は除去されます。そして作業単位 (または保管点) で表を除去していた場合には、表が復元されますが、行は復元されません。さらに、表で INSERT、UPDATE、または DELETE 操作を実行するステートメントにエラーが発生した場合も、表の行はすべて削除されてしまいます。

WITH REPLACE
ユーザーが指定した名前を持つ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された一時表と置き換える (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。条件にかなう USER TEMPORARY 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。

- 表スペースを除去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、820ページの『行サイズ』を参照してください。

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

表のデータが区分化されている場合に、区分化キーを指定します。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、この表が複数区分のノードグループに存在する場合は、その区分化キーが宣言された一時表の最初の列として定義されます。

宣言された一時表では、単一区分のノード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区分化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、区分化キーは作成されません。

区分化キーの列は更新できませんのでご注意ください (SQLSTATE 42997)。

USING HASHING

データ配分の区分化方式として、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区分化方式です。

注

- **宣言されたグローバル一時表の参照:** 宣言された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。それで、同じ *table-name* という宣言されたグローバル一時表を定義するセッションであっても、その宣言された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) を使用して宣言された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、宣言された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、宣言された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:** 宣言された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、表を除去する権限も含めて、その表に対するすべての表特権が授与されます。加えて、PUBLIC に対しても同様の特権が授与されます。[89] 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された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 の中の宣言された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。

89. GRANT オプションによって授与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。

  - 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 プロシージャーの複合ステートメント (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、宣言されたグローバル一時表の効力範囲が複合ステートメントだけでなく接続にまで広がり、表は複合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。表は複合ステートメントの END で暗黙的に除去されません。宣言されたグローバル一時表は、その表が明示的に除去されない限り、同じ名前を使用してセッション内の他の複合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、作業単位または保管点でロールバック操作が終了し、その作業単位または保管点に SESSION.T への変更が含まれている場合、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。

    P において、作業単位または保管点でロールバック操作が終了し、その作業単位または保管点に 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、作業単位または保管点でロールバック操作が終了し、その作業単位または保管点に宣言された一時表 SESSION.T の除去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除去は取り消されますが、表の行は復元されません。
  - 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 は除去され、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 は除去され、そのインスタンス化された行は破棄されます。

- **宣言されたグローバル一時表の使用に関する制限:** 宣言されたグローバル一時表には、次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 INDEX、 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)。

DELETE

DELETE ステートメントは、表または視点から行を削除します。 視点から行を削除すると、その視点の基礎となる表から行を削除することになります。

このステートメントには、以下の 2 つの形式があります。

- *探索 (Searched)* DELETE 形式。この形式は、 1 つまたは複数の行を削除するのに使用します (削除する行は探索条件によって、自由に限定できます)。
- *位置設定 (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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な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または視点に対する DELETE 特権
- 削除する行を含む表または視点に対する CONTROL 特権
- SYSADM または DBADM 権限

探索 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または視点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

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[90] 、探索 DELETE 形式の *search-condition* に表または視点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

90. ステートメントの処理に使用されるパッケージは、値 SQL92E または MIA を指定したオプション LANGLEVEL を使用してプリコンパイルされます。

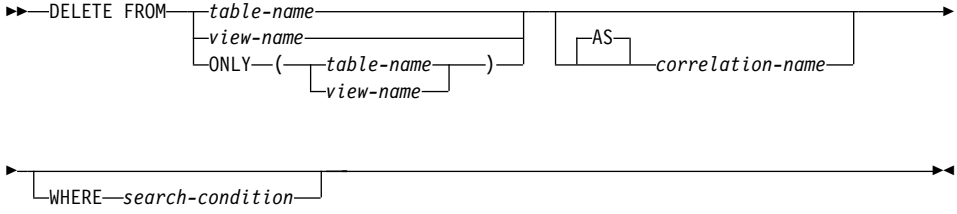
- CONTROL 特権
- SYSADM または DBADM 権限

指定した表または視点が 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、指定した表または視点の副表または副視点ごとに SELECT 特権が含まれている必要があります。

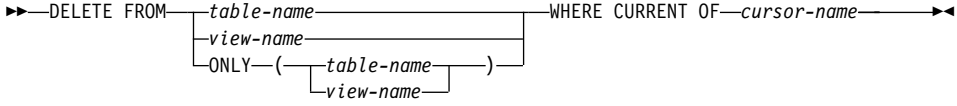
静的 DELETE ステートメントの場合、グループ特権は検査されません。

構文

Searched DELETE:



Positioned DELETE:



説明

FROM *table-name* または *view-name*

行を削除する表の名前 (*table-name*) または視点の名前 (*view-name*) を指定します。 この名前は、カタログに存在する表または視点を指定するものでなければならず、カタログ表、カタログ視点、要約表、または読み取り専用の視点は指定できません。 (読み取り専用の視点については、895ページの『CREATE VIEW』 を参照してください)。

table-name がタイプ付きの表である場合は、このステートメントを使用すれば、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 がタイプ付き視点である場合は、このステートメントを使用すれば、その基礎表の行またはその視点に関係する副視点の基礎表を削除で

きます。 *view-name* が基礎表 (タイプ付き表) を伴う通常の視点である場合、このステートメントを使用すれば、そのタイプ付き表またはそれに関係する副表の行を削除できます。

WHERE 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視点を、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***)**
タイプ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をステートメントが適用し、その表に関係する副表の行は削除でき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 を指定して指定しなければなりません。 *table-name* がタイプ付き表でない場合は、このステートメントに ONLY 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タイプ付き視点のみに適用されます。 ONLY キーワードは、指定された視点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。副視点の行は、このステートメントでは削除できません。 位置指定 DELETE の場合は、FROM 文節に指定されているのと同じ視点を、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* がタイプ付き視点でない場合は、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-name
探索条件 (*search-condition*) の中で、表または視点を指定するために使用することができます。 (相関名 (correlation-name) については、 71ページの『第3章 言語要素』 を参照してください。)

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、探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、表または視点のすべての行が削除されます。

search-condition
227ページの『探索条件』 に説明されているように、探索条件を指定します。 副照会以外の探索条件の各*列名* は、該当の表または視点の列を指定するものでなければなりません。

search-condition は、該当の表または視点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。

探索条件に副照会が含まれる場合、その副照会は、*探索条件* が行に適用されるたびに実行され、その結果が*探索条件* 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 ステートメントの前になければなりません。

指定する表または視点は、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、914ページの『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。

規則

- 指定する表または指定する視点の基礎表が親である場合、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子孫であってはならず、DELETE は RESTRICT の削除規則との関係において子孫である下層行に連鎖してはなりません。

  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。選択された行の子孫である行もすべて影響を受けます。

  – 削除規則が SET NULL の関係において子孫であるすべての行の外部キーのヌル値可列は、ヌル値に設定されます。
  – 削除規則が CASCADE の関係において子孫であるすべての行も削除され、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、非ヌル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の位置に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、または探索 DELETE の結果として) 行 R の前の位置にあるとします。 R の派生元の基礎表に影響する 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)。
- ステートメントの実行後に、対象となる表から削除された行数が SQLCA の SQLERRD(3) 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は含まれません。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。 (SQLCA については、1211ページの『付録B. SQL 連絡 (SQLCA)』 を参照してください。)
- あるエラーが起きたために探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。

- 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復元されたり削除されたりします。

  データベース・サーバーにファイル・サーバーの値が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
```

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。準備済みステートメントについては、1042ページの『PREPARE』を参照してください。

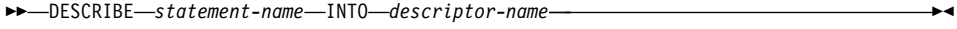
呼び出し

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方法のみ可能です。これは、動的に準備できない実行可能ステートメントです。

許可

権限は不要です。

構文



説明

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するものでなければなりません。

INTO *descriptor-name*
SQL 記述子域 (SQLDA) を指定します (1217ページの『付録C. SQL 記述子域 (SQLDA)』 を参照)。 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 もし作成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、その列にヌル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
派生された列が単一の列参照でない場合、 sqlname には派生された列の選択リスト内での元の位置を表す ASCII 数値リテラル値が入ります。それ以外の場合、sqlname には列の名前が入ります。

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 カタログ視点の 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 の出現数に設定し、 SQLN 個のオカレンスが入るだけの十分な記憶域を割り振っておく必要があります。 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 のオカレンス数が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記憶域を割り振るまでの間に、 SQLDA を修正して、SQL_TYP_xLOB の SQLTYPE を SQL_TYP_xLOB_LOCATOR に、または対応する変更を含む SQL_TYP_xLOB_FILE を SQLLEN などの他のフィールドに変更してください。その後、SQLTYPE に基づいて記憶域を割り振ってから、処理を継続します。

  SQLDA とともにロケーターとファイル参照変数を使用する方法の詳細については、 *アプリケーション開発の手引き* を参照してください。
- 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。このような状態の処理については、*アプリケーション開発の手引き* を参照してください。
- 構造タイプが選択されているのに、 FROM SQL 変換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、大量の記憶域を使用し、その記憶域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。[91]
  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 よりも記憶域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。

---

91. 上記の正の SQLCODE の戻り値は、 SQLWARN バインド・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。

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;
.
.
.
```

DISCONNECT

DISCONNECT ステートメントは、活動状態の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。[92]

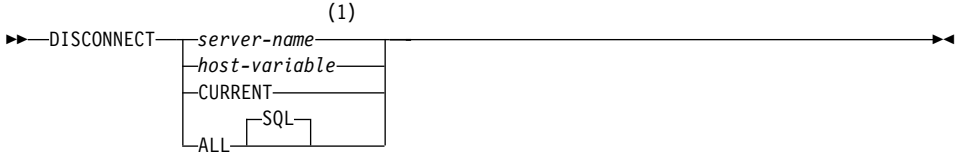
呼び出し

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、動的に準備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* に入っている *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。

92. 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、そのデータベースが既存のどの作業単位にも関係しておらず、しかも活動状態の作業単位が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することは可能です。

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活動状態の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が破棄されます。

  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 の 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、さまざまな資源が必要になります。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、実行時に 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、44ページの『分散作業単位の意味を制御するオプション』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;
```

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。 (詳細については、853ページの『作動不能トリガー』および 906ページの『作動不能視点』を参照してください。) 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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

2 つの部分からなる名前が使用可能なオブジェクトを除去する場合、 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視点の 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、および視点にのみあてはまる)
- カタログ視点 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除去している場合にのみあてはまる)

表または視点の階層を除去する場合、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視点について、上記の特権のいずれかを持っている必要があります。

スキーマを除去する場合、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。

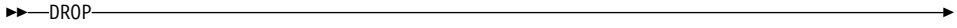
バッファー・プール、ノード・グループ、または表スペースを除去する場合、DROP ステートメントの許可 ID は SYSADM 権限または SYSCTRL 権限を持っている必要があります。

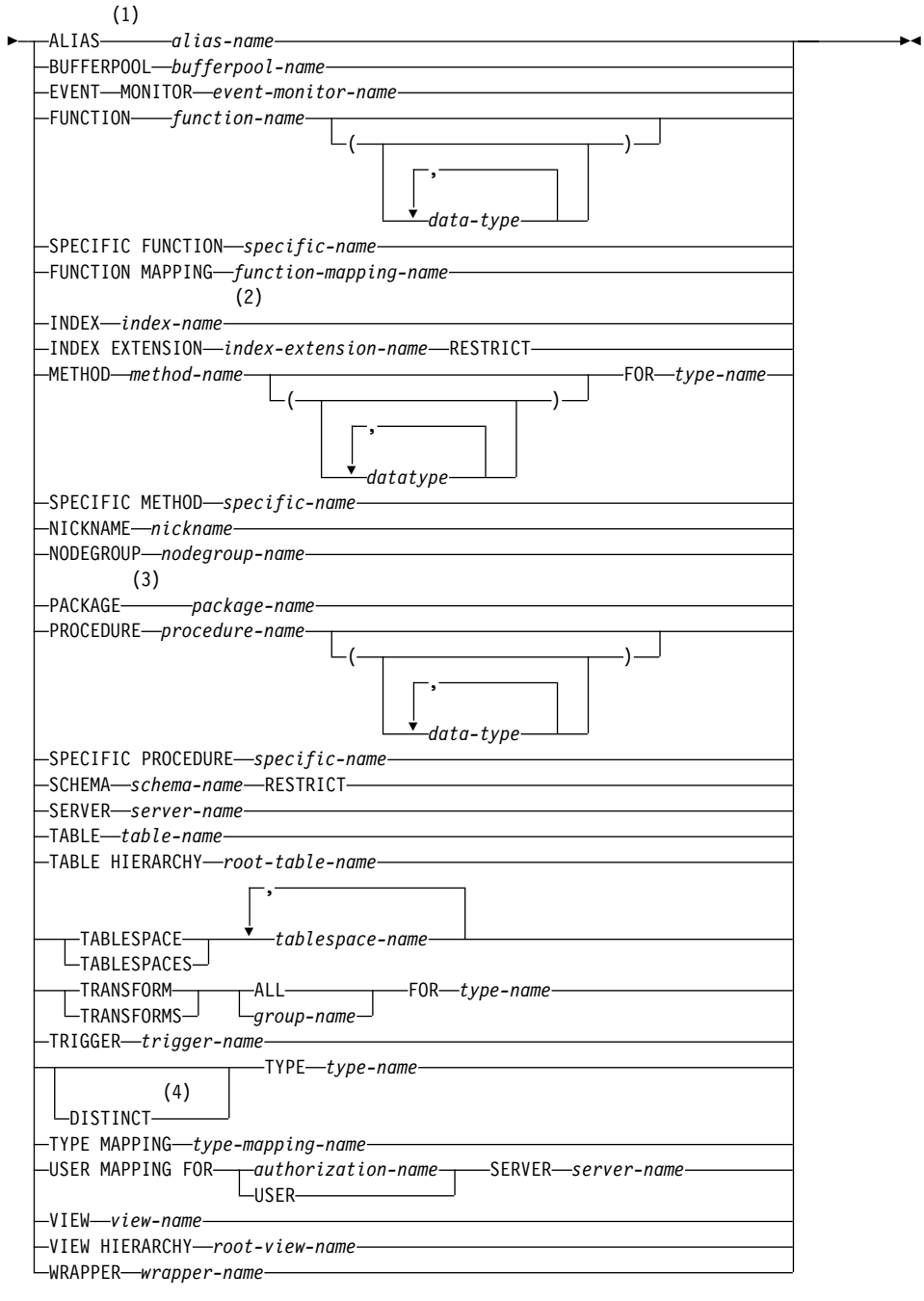
イベント・モニター、サーバー定義、データ・タイプ・マッピング、関数マッピング、またはラッパーを除去する場合、 DROP ステートメントの許可 ID は SYSADM 権限または DBADM 権限を持っている必要があります。

ユーザー・マッピングを除去する際、この許可 ID がマッピング内にある連合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。

変形を除去する際、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




(1)
ALIAS alias-name
BUFFERPOOL bufferpool-name
EVENT MONITOR event-monitor-name
FUNCTION function-name
(, data-type)
SPECIFIC FUNCTION specific-name
FUNCTION MAPPING function-mapping-name
(2)
INDEX index-name
INDEX EXTENSION index-extension-name RESTRICT
METHOD method-name
(, datatype)
FOR type-name
SPECIFIC METHOD specific-name
NICKNAME nickname
NODEGROUP nodegroup-name
(3)
PACKAGE package-name
PROCEDURE procedure-name
(, data-type)
SPECIFIC PROCEDURE specific-name
SCHEMA schema-name RESTRICT
SERVER server-name
TABLE table-name
TABLE HIERARCHY root-table-name
TABLESPACE
TABLESPACES
, tablespace-name
TRANSFORM
TRANSFORMS
ALL
group-name
FOR type-name
TRIGGER trigger-name
TYPE type-name
DISTINCT
(4)
TYPE MAPPING type-mapping-name
USER MAPPING FOR
authorization-name
USER
SERVER server-name
VIEW view-name
VIEW HIERARCHY root-view-name
WRAPPER wrapper-name

注:

1. ALIAS の同義語として SYNONYM を使用できます。
2. *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。
3. PACKAGE の同義語として PROGRAM を使用できます。
4. 任意のユーザー定義タイプを除去するのに DATA を使用することもできます。

説明

ALIAS *alias-name*

除去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)。指定した別名が削除されます。

その別名を参照する表、視点、およびトリガーは、作動不能になります。[93]

BUFFERPOOL *bufferpool-name*

除去するバッファー・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てはなりません (SQLSTATE 42893)。IBMDEFAULTBP バッファー・プールは除去できません (SQLSTATE 42832)。バッファー・プールとしての記憶域は、データベースが停止するまでは解放されません。

EVENT MONITOR *event-monitor-name*

除去するイベント・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) になります。 ON でない場合、そのイベント・モニターが削除されます。

イベント・モニターを除去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。

---

93. 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。

FUNCTION

除去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除去できません。

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 特殊レジスターは、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 / バインド・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 42854) になります。

FUNCTION *function-name* **(***data-type***,...)**

除去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

除去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 特殊レジスターは、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 / バインド・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 ステートメントの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、除去する特定の関数を識別するのに使用されます。

data-type が修飾されない場合は、タイプ名は SQL パスでスキーマを検索することによって決定されます。 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べきことを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。タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

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、除去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 特殊レジスターは、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 / バインド・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 スキーマまたは SYSFUN スキーマ (SQLSTATE 42832) の関数は除去できません。

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。そのような関数を除去する場合には、その前にそのような従属オブジェクトをすべて除去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除去しようとすると、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、963 ページを参照してください。

関数が除去可能な場合、その関数が除去されます。

除去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。これらのコマンドについては、*コマンド解説書* を参照してください。

FUNCTION MAPPING *function-mapping-name*

除去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、除去することができません。しかし、それらを使用不可にすることができます。例については、710ページの『CREATE FUNCTION MAPPING』の例 3 を参照してください。

除去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*

除去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、システムが必要としている基本キーまたは固有制約の索引であってはなりません (SQLSTATE 42917)。指定した索引または索引指定は削除されます。

除去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**

除去する索引の拡張を指定します。 *index-extension-name* (索引の拡張名) は、カタログに記述されている索引の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、この索引の拡張の定義に従って索引を定義できないように規則を課します (SQLSTATE 42893)。

METHOD

除去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を除去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。メソッドの本体が除去された後、メソッドの指定は ALTER TYPE DROP METHOD によってサブジェクト・タイプの定義から削除することができます。

除去するメソッド本体は、次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*
除去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 42854)。

METHOD *method-name* **(***data-type***,...)**
除去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、除去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、メソッドの指定の相当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、除去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べきことを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。タイプが REAL か DOUBLE かによって、生じる一致は異なってきます。

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除去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 / バインド・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。

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、除去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 / バインド・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)。

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、メソッドを除去する前にそれらの従属関係をすべて取り除く必要があります (ただし、そのメソッドが除去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除去しようとすると、エラーが戻されます (SQLSTATE 42893)。

除去できる状態にあれば、メソッドは除去されます。

除去する特定のメソッドに従属しているパッケージは、そのメソッドが除去されると、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。これらのコマンドについては、*コマンド解説書* を参照してください。

NICKNAME *nickname*

除去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索引指定は除去されます。ニックネームに従属したすべての視点に作動不能のマークが付けられます。除去された索引指定または作動不能視点に従属したパッケージはすべて無効です。ニックネームが参照するデータ・ソースは影響を受けません。

NODEGROUP *nodegroup-name*

除去するノード・グループを指定します。 *nodegroup-name* (ノード・グループ名) は、カタログに記述されているノード・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。

ノード・グループを除去すると、そのノード・グループに定義されている表スペースがすべて除去されます。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、除去されるか、または無効になり (該当する場合)、従属する視点とトリガーは作動不能になります。

システム定義のノード・グループは除去できません (SQLSTATE 42832)。

現在データ再配布が行われているノード・グループに対して DROP NODEGROUP を発行すると、 DROP NODEGROUP 操作は失敗し、エラーが戻されます (SQLSTATE 55038)。ただし、部分的に再配布されたノード・グループは除去できます。ノード・グループは、REDISTRIBUTE NODEGROUP コマンドが完了するまで実行されなかった場合に、部分的に再配布の状態になります。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。[94]

94. 部分的に再配布されたノード・グループの場合は、 SYSCAT.NODEGROUPS カタログ内の REBALANCE_PMAP_ID が -1 になりません。

PACKAGE *package-name*
除去するパッケージを指定します。 *package-name* (パッケージ名) は、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。そのパッケージに対する特権もすべて削除されます。

PROCEDURE
除去するストアード・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 / バインド・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 42854)。

PROCEDURE *procedure-name* **(***data-type***,...)**
除去する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。プロシージャー選択のアルゴリズムは使用されません。

procedure-name
除去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 / バインド・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 ステートメントの相当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、除去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。タイプが REAL か DOUBLE かによって、一致する値は異なってきます。

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、除去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 / バインド・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)。

SCHEMA *schema-name* **RESTRICT**
除去する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)。

SERVER *server-name*
カタログから定義を除去するデータ・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視点のニックネームはすべて除去されます。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除去されます。除去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除去されます。除去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*
除去する基礎表、宣言された一時表、または要約表を指定します。 *table-name* に指定される表名は、カタログに記述されている基礎表や要約表を識別する名前でなければなりません。また、宣言された一時表を指定する場合は、スキーマ名 SESSION によって修飾された、アプリケーションに存在する一時表の名前を *table-name* に指定しなければなりません (SQLSTATE 42704)。タイプ付き表の副表は、それぞれスーパー表に従属しています。ですから、スーパー表を除去する前には、副表をすべて除去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、基本キー、外部キー、検査制約、および要約表は除去されます。その表を参照するすべての視点およびトリガー[95]は、作動不能になります。除去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

95. 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。

ージが含まれます。参照列の中で、除去された表を参照の効力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。

宣言された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、宣言された一時表が除去されてもパッケージが無効になることはありません。

DATALINK 列で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。

表階層から副表を除去すると、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)。副表を除去すると、スーパー表から副表の列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。

宣言された一時表が、現在の作業単位または保管点が活動状態になる前に作成されたものである場合は、その一時表を除去すると機能上で表が除去されてしまうため、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、あるいは保管点が終了するまで、依然としてこの表が予約された状態にあるため、USER TEMPORARY 表スペースを除去したり、 USER TEMPORARY 表スペースのノードグループを再配布することはできません。宣言された一時表が除去されると、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。

TABLE HIERARCHY *root-table-name*

除去するタイプ付き表階層を指定します。 *root-table-name* で指定するタイプ付き表は、タイプ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* で指定するタイプ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。

除去された表を参照するすべての索引、要約表、基本キー、外部キー、および検査制約は除去されます。除去された表を参照するすべての視点およびトリガーは、作動不能になります。除去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、除去された表を参照の効力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 列で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かもしれません。

単一の副表を除去する場合とは違い、表階層を除去しても、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*
除去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。

表の一部が除去される表スペースに保管され、 1 つかそれ以上の部分が除去されない別の表スペースに保管されている場合、この表スペースは除去されません (SQLSTATE 55024)。 (そのような表を前もって除去する必要があります。) システム表スペースは除去できません (SQLSTATE 42832)。データベースに一時表スペースが 1 つしか存在しない場合は、SYSTEM TEMPORARY 表スペースを除去することはできません。宣言された一時表が作成されている USER TEMPORARY 表スペースは除去できません (SQLSTATE 55039)。 USER TEMPORARY 表スペースでは、宣言された一時表が削除されていても、 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みなされます。

表スペースを除去すると、その表スペースに定義されているオブジェクトがすべて除去されます。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・オブジェクトは除去されるか、または無効になり、従属している視点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・ディレクトリーから下にあるすべてのコンテナーは削除されます。 SMS 表スペースでは、すべての接続が切断されるか DEACTIVATE DATABASE コマンドが出されるまで削除は行われません。

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変形グループが除去されることを示します。これらのグループで参照される変形関数は除去されません。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 / バインド・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-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 に定義されている変形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変形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変形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。

変形グループの中に 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変形関数のいずれかに従属している場合、その変形グループは除去されません (SQLSTATE 42893)。このようなユーザー定義関数が従属している変形関数は、 *type-name* で定義された参照先の変形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変形関数が、指定された変形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、指定した変形グループが除去されることを示します。このグループで参照される変形関数は除去されません。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 / バインド・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-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* に存在している変形グループを指定しなければなりません。

TRIGGER *trigger-name*
除去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)。指定したトリガーが削除されます。

トリガーを除去すると、特定のパッケージが無効としてマークされます。トリガーの作成 (同じ規則に従う) についての、 846ページの『CREATE TRIGGER』の “注” を参照してください。

TYPE *type-name*
除去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 特殊レジスターは、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 / バインド・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除去されます。 *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。以下の場合、このタイプは除去されません (SQLSTATE 42893)。

- 表または視点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。
- タイプ付き表またはタイプ付き視点のデータ・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場所で *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 を除去することはできません。
- タイプが、表または視点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、除去できない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。
- このタイプが検査制約、トリガー、視点定義、または索引の拡張で使用されている。

除去されるタイプを使用する関数について: ユーザー定義タイプが除去可能な場合、除去するそのタイプ (または除去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、従属する関数も除去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除去リストに含まれている場合には、ユーザー定義タイプの除去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)。

除去されるタイプを使用するメソッドについて: ユーザー定義タイプが除去可能な場合、除去するそのタイプ (または除去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 が失敗する場合があります。

TYPE MAPPING *type-mapping-name*
除去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。

その他に除去されるオブジェクトはありません。

USER MAPPING FOR *authorization-name* **| USER SERVER** *server-name*
除去するユーザー・マッピングを指定します。このマッピングは、連合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* (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 (SQLSTATE 42704)。 ユーザー・マッピングが削除されます。

その他に除去されるオブジェクトはありません。

VIEW *view-name*
削除する視点を指定します。*view-name* (視点名) は、カタログに記述されている視点を指定していなければなりません (SQLSTATE 42704)。タイプ

付き表の副視点は、それぞれスーパー表に従属しています。ですから、スーパー視点を除去する前に、副視点をすべて除去する必要があります (SQLSTATE 42893)。

指定した視点が削除されます。直接的または間接的にその視点に従属する視点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要約表はすべて除去されます。除去された視点または作動不能としてマークされた視点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、副視点よりも上位の階層であるスーパー視点に従属するパッケージが含まれます。参照列の中で、除去された視点を参照の効力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*

除去するタイプ付き視点階層を指定します。 *root-view-name* で指定するタイプ付き視点は、タイプ付き視点階層のルート視点でなければなりません (SQLSTATE 428DR)。 *root-view-name* で指定するタイプ付き視点とその視点のすべての副表が、データベースから削除されます。

直接的または間接的に除去された視点に従属する視点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。除去された視点やトリガー、または作動不能としてマークされた視点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、除去された視点や作動不能とマークされた視点を参照の効力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*

除去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・マッピングは除去されます。除去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除去されます。 除去されたニックネームに従属する索引指定はすべて除去され、こうしたニックネームに従属する視点はすべて、作動不能としてマークが付けられます。 除去されたオブジェクトと作動不能視点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: 964ページの表27 は、オブジェクトの相互間の従属関係を示します。[96] 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除去できません。

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除去すると、その従属オブジェクトも同時に除去されます。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削除できない場合には、基礎となるオブジェクトの除去は失敗します。

X 作動不能 (Inoperative) を意味します。基礎となるオブジェクトを除去すると、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除去すると、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。

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、964ページの表27 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、区分化キー、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ができます。
  - 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視点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)。

96. 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、パッケージがどの制約に対して従属しているかは記録されません。

– 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視点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 27. 従属関係

オブジェクト・タイプ → / ステートメント ↓	制約	関数	関数マッピング	索引	索引の拡張	メソッド	ニックネーム	ノードグループ	パッケージ	サーバー	表	表スペース	トリガー	タイプ	タイプ・マッピング	ユーザー・マッピング	視点
ALTER NICKNAME	-	-	-	-	-	-	-	-	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 D R O P ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	-	-	-	-	-	A[3]	-	R[3]	-	X[3]	-	-	-	X[3]
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-
DROP FUNCTION	R	R[7]	R	-	R	R[7]	-	-	X	-	R	-	R	-	-	-	R

表 27. 従属関係 (続き)

オブジェクト・タイプ → / ステートメント ↓	制約	関数	関数マッピング	索引	索引の拡張	メソッド	ニックネーム	ノード・グループ	パッケージ	サーバー	表	表スペース	トリガー	タイプ	タイプ・マッピング	ユーザー・マッピング	視点
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	R[17]
DROP INDEX EXTENSION	-	R	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R[7]	R	-	R	R	-	-	X	-	R	-	R	-	-	-	R
DROP NICKNAME	-	R	-	C	-	-	-	-	A	-	-	-	-	-	-	-	X[16]
DROP NODEGROUP	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-
DROP SERVER	-	C[21]	C[19]	-	-	-	C	-	A	-	-	-	-	-	C[19]	C	-
DROP TABLE	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]	-	-	-	-	-	-	-	-
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]	-	-	-	-	-	-	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、または区分化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、視点に影響を与える INSERT、UPDATE、または DELETE ステートメントが含まれている場合、そのパッケージは視点

の基礎表に対して挿入、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礎表の各列ごとに更新操作を実行します。

タイプ付き視点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、同じ視点階層内で視点を作成したり除去したりすると、パッケージが無効になります。

3 パッケージ、要約表、視点、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。別名が連鎖している場合、その連鎖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・タイプを除去すると、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除去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除去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除去すると、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除去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除去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除去できません。

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 つまたは複数の変形関数に従属することができます。

8 要約表が除去されたり、視点が作動不能になるのは、SELECT 特権が

ない場合だけです。作動不能にされた視点がタイプ付き視点階層に含まれていれば、その副視点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実行します。

タイプ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除去したりすると、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、または視点の 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副視点で 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、または視点で DEREF(*Y*) が使用されていて、 *Y* が *Z* という表または視点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副視点で SELECT 特権に対する従属関係が存在します。

11 要約表は、基礎表、あるいは表定義の全選択で指定された表に従属しています。

カスケードのセマンティクスが、従属する要約表に適用されます。

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除去されるまで、スーパー表は除去できません。

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構造タイプすべてと従属関係にあります。構造タイプを除去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。

13 あるタイプが検査制約またはトリガーで使用されている場合、検査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。検査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 あるタイプが視点定義で使用されている場合、視点はこのタイプに従属

する関係にあります (タイプ付き視点のタイプも含まれます)。視点定義内の 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 副視点はスーパー視点に従属しており、この従属関係はルート視点にまで及びます。従属するすべての副視点が除去されるまで、スーパー視点は除去できません。視点の従属関係の詳細については、 16 を参照してください。

16 トリガーまたは視点は参照解除操作または DEREF 関数のターゲット表やターゲット視点にも従属しています。 FROM 文節のトリガーまたは視点で OUTER(*Z*) を含むものは、トリガーまたは視点が作成された時点に存在した *Z* の副表または副視点すべてに対して従属関係にあります。

17 タイプ付き視点は固有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識別子列が固有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 サーバーを除去すると、連鎖的に、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・マッピングが除去されます。

20 複数区分のノードグループにある表に対して区分化キーが定義されている場合、この区分化キーは必須です。

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、サーバーを除去できません。

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 が除去されると、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タイプ付き表である。
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 定義された関数に SELECT WITH GRANT OPTION 特権がなくなると、 SQL 関数の本体で使用される表または視点に対する SELECT 特権の REVOKE により、特権を失った関数の除去が試行されます。これらの関数が視点やトリガーで使用されている場合は、これを除去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除去されます。

注

- ユーザー定義関数を使用中に、そのユーザー定義関数を除去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除去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 からその関数を除去することはできません。その時点で、関数は除去され、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は、関数本体が除去される場合があります (これは関数の除去とは違うことです)。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記憶域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。
- 除去された表の中に 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復元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 データ・リンク・マネージャー を使用できないときに、 DROP TABLE または DROP TABLESPACE を使って DATALINK 列を含んだ表が除去されると、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、変形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変形が必要な場合は、その関数またはメソッドに、 TO SQL か FROM SQL の必要な変形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 ステートメントで変形関数が必要になる場合は、そのパッケージに、TO SQL か FROM SQL の指定された変形関数に対する従属関係が記録されます。

  上記の部分では、変形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変形関数に従属することはありません。一方、変形関数を明示的に呼び出した場合 (たとえば、視点やトリガーなどで) は、これらの他のタイプのオブジェクトが通常どおり変形関数に従属します。したがって、変形に対するこれらの「明示的な」タイプの従属が除去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、暗黙的な変形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。それで、変形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、関数上の変形プロパティーが除去されなかったり、パッケージが作動不能としてマークされてしまいます。

例

例 1: 表 TDEPT を除去します。

```
DROP TABLE TDEPT
```

例 2: 視点 VDEPT を除去します。

```
DROP VIEW VDEPT
```

例 3: 許可 ID HEDGES が別名を除去します。

```
DROP ALIAS A1
```

別名 HEDGES.A1 がカタログから除去されます。

例 4: Hedges は別名の除去を試みますが、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 ノードグループを除去します。このノードグループを除去するには、まずノードグループ内の表スペース (ACCOUNTING と PLANS) を除去する必要があります。

```
DROP TABLESPACE ACCOUNTING
   DROP TABLESPACE PLANS
   DROP NODEGROUP BUSINESS_OPS
```

例 6: Pellow は CENTRE 関数を除去します。この関数は、除去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 関数を除去します。この関数は、除去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 を除去します。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 を除去します。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・タイプを除去します。

```
   DROP DISTINCT TYPE SHOESIZE
```

例 11: SMITHPAY イベント・モニターを除去します。

```
   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除去します。 PART という名前の表をまず削除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 プロシージャーを除去します。このプロシージャーは、除去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 を除去します。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、連合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。マッピングを除去します。

```
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。最適化プログラムにこの索引を認識させるために作成した索引指定を除去します。

```
DROP INDEX INDEXSPEC
```

例 17: 変形グループ MYSTRUCT1 を除去します。

```
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 を除去します。

```
DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

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。

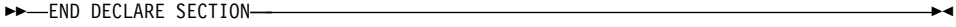
呼び出し

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のみ可能です。 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

権限は不要です。

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 (555ページの『BEGIN DECLARE SECTION』 を参照)。

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。[97] 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。

97. LOB ロケーターとファイル参照変数の場合に、ホスト変数を REXX で宣言する方法については、 555ページの『規則』を参照してください。

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。

例

END DECLARE SECTION ステートメントの使用例については、 555ページの『BEGIN DECLARE SECTION』 を参照してください。

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。

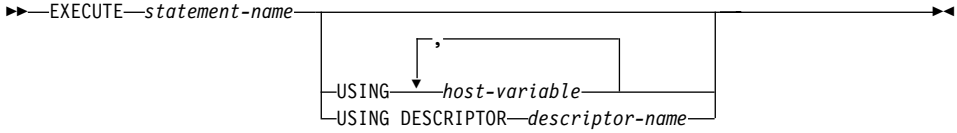
呼び出し

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のみ可能です。これは、動的に準備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。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。

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 (?) に置き換わる値を含むホスト変数のリストを指定します。 (パラメーター・マーカーについては、1042ページの『PREPARE』 を参照してください。) 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る場合、USING は必須です。

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。 該当する場合には、パラメーター・マーカーに対する値のソースとしてロケーター変数とファイル参照変数も指定できます。

DESCRIPTOR *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 に割り振る記憶域のバイト数を示す 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記憶域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 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。詳しくは、1217ページの『付録C. SQL 記述子域 (SQLDA)』を参照してください。

注

- 準備済みステートメントを実行する前に、各パラメーター・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。タイプ付きパラメーター・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・マーカーの文脈に従って決定されます。パラメーター・マーカーに関連する規則については、1043ページの『規則』 を参照してください。

  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 SQL ステートメントのキャッシュ***

  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 ステートメント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用できます。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用されます。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用されます。 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ュを使用すると、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。

以下の事象は、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 NICKNAME
- ALTER SERVER
- ALTER TABLE
- ALTER TABLESPACE
- ALTER TYPE
- CREATE FUNCTION
- CREATE FUNCTION MAPPING
- CREATE INDEX
- 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
- 視点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 (およびその後) に活動状態の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。 h1 - h4 は、TDEPT の形式に対応するホスト変数です。

```
strcpy (s,"INSERT INTO TDEPT VALUES(?,?,?,?)");
EXEC SQL PREPARE DEPT_INSERT FROM :s;
   .
   .
(正常実行の検査を行い、:h1, :h2, :h3, :h4 に値を入れる)
   .
   .
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
```

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。

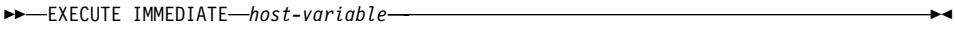
呼び出し

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のみ可能です。これは、動的に準備できない実行可能ステートメントです。

許可

権限の規則は、EXECUTE IMMEDIATE ステートメントに指定する SQL ステートメントにより定義されます。

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ご注意ください。

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
- COMMENT ON

- COMMIT
- CREATE
- DELETE
- DECLARE GLOBAL TEMPORARY TABLE
- DROP
- GRANT
- INSERT
- LOCK TABLE
- REFRESH TABLE
- RELEASE SAVEPOINT
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- ROLLBACK
- SAVEPOINT
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET CURRENT TRANSFORM GROUP
- 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 ステートメントと CREATE PROCEDURE ステートメントでは、ト

リガーによって実行される SQL ステートメントを区切ったり、 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 (;) を使用することができます。

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。詳しくは、977ページの『動的 SQL ステートメントのキャッシュ』を参照してください。

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;
```

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取り込むとともに、この情報を Explain 表に入れます。 (Explain 表と表定義については、1401ページの『付録K. Explain 表と定義』 を参照してください。)

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、あるいは対話式に使用することができます。このステートメントは、動的に準備可能な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。

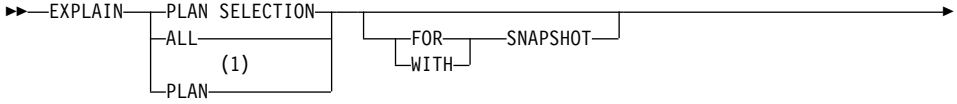
許可

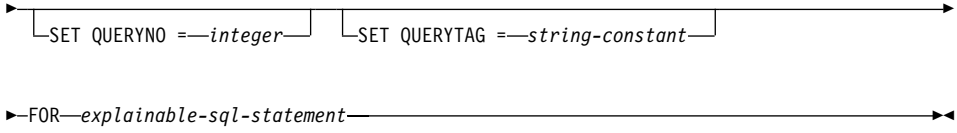
EXPLAIN ステートメントに指定された SQL ステートメントに定義されているのと同じ許可規則が適用されます。 たとえば、*explainable-sql-statement* (次ページのステートメント構文を参照) として DELETE ステートメントが使用された場合、 DELETE ステートメントに定義されている許可規則が、DELETE ステートメントの Explain 情報を取り出す場合にも適用されます。

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 (MVS 版) 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 表に存在するものを除き、他の Explain 情報は取り込まれません。

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。

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 情報を取り込むか?	Visual Explain 用にスナップショットを取るか?
なし	取り込む	取らない
FOR SNAPSHOT	取り込まない	取る
WITH SNAPSHOT	取り込む	取る

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 を呼び出す前に、Explain 表を作成しておく必要があります。 (Explain 表と表定義については、1401ページの『付録K. Explain 表と定義』 を参照してください。) 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 表には情報が取り込まれません。

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、情報はバインド実行時および明示的な再バインド時に Explain 表に挿入されます (*コマンド解説書* の REBIND の項を参照)。 プリコンパイル中、静的 EXPLAIN ステートメントについてのコメントは、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た場合、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かを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実行依頼されるときです。パッケージが実行された場合、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実行依頼されるときです。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。

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;
```

このステートメントは成功します。

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;
```

このステートメントは成功します。

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;
```

このステートメントは成功します。

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)。

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。

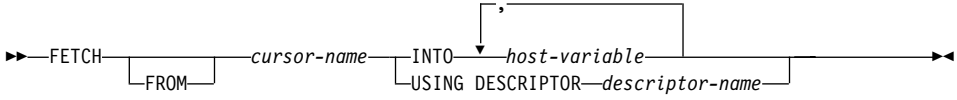
呼び出し

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、動的に準備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、914ページの『DECLARE CURSOR』 を参照してください。

構文



説明

cursor-name

取り出し操作で使用するカーソルを指定します。 914ページの『DECLARE CURSOR』 で説明されているように、*cursor-name* (カーソル名) は、宣言されたカーソルを指定してい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。
- ホスト変数には値が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位置づけされ、 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位置づけされ、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 (*host-variable*) 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。

- SQLN (その SQLDA に用意される SQLVAR のオカレンスの数を示す)
- SQLDABC (その SQLDA に割り振られる記憶域のバイト数を示す)
- SQLD (ステートメントの処理中にその SQLDA で使用される変数の数を示す)
- 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記憶域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。
SQLDOUBLED、LOB、および構造タイプの列について説明した、1225ページの『SQLDA に対する DESCRIBE の効果』を参照してください。

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。詳しくは、1217ページの『付録C. SQL 記述子域 (SQLDA)』を参照してください。

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、『第3章 言語要素』で説明されている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、ロケーター資源の限度を考慮して、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、先回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 (*管理の手引き* を参照してください)。
- 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。詳細については、976ページの『注』 を参照してください。
- 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。

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
```

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、いつでも部分事象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* 識別子が付けられます。

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。

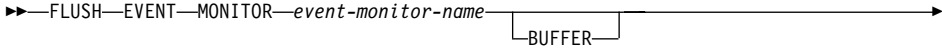
呼び出し

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、あるいは対話式に使用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 識別子です。

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。

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。

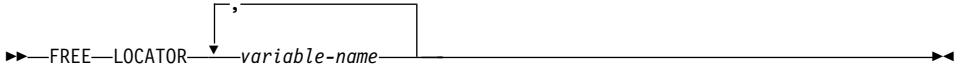
呼び出し

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方法のみ可能です。 これは、動的に準備できない実行可能ステートメントです。

許可

権限は不要です。

構文



説明

LOCATOR *variable-name,* **...**

1 つまたは複数のロケーター変数 (*variable-name*) 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 (FETCH ステートメントまたは SELECT 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、エラーになります (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が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.
```

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。

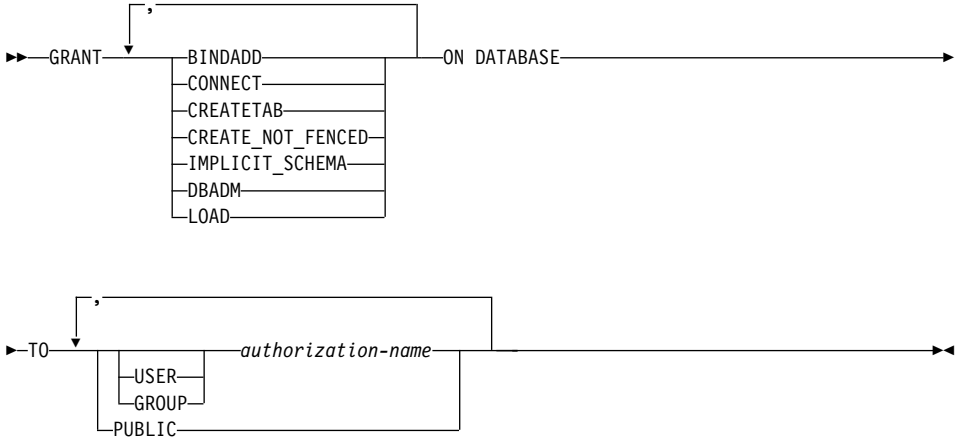
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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

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
基礎表を作成する権限を付与します。基礎表の作成者には、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。

視点作成に必要な明示的な権限は特にありません。視点の作成に使用するステートメントの許可 ID に各視点の基礎表に対する 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、いつでも視点を作成できます。

CREATE_NOT_FENCED
データベース・マネージャーの処理の中で実行する関数を登録する権限を与えます。そのようにして登録された関数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 (詳細については、 648 ページの FENCED 文節または NOT FENCED 文節を参照してください)。

関数が非分離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。

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。

BINDADD、CONNECT、CREATETAB、CREATE_NOT_FENCED および IMPLICIT_SCHEMA は、 DBADM 権限を与えられている *authorization-name* に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。

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 であるとみなされます。
  - 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 であるとみなされます。
  - 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、または DCE 認証が使用されている場合、エラー (SQLSTATE 56092) が発生します。

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)。

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

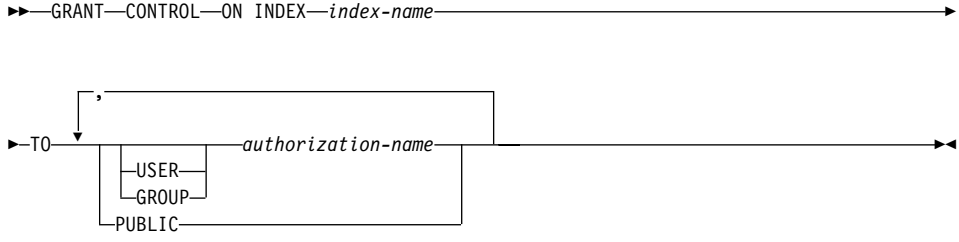
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

このステートメントの許可 ID 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。

- DBADM 権限
- SYSADM 権限

構文



説明

CONTROL
索引を除去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 であるとみなされます。
  - 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 であるとみなされます。
  - 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、または DCE 認証が使用されている場合、エラー (SQLSTATE 56092) が発生します。

例

```
GRANT CONTROL ON INDEX DEPTIDX TO USER USER4
```

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・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)。

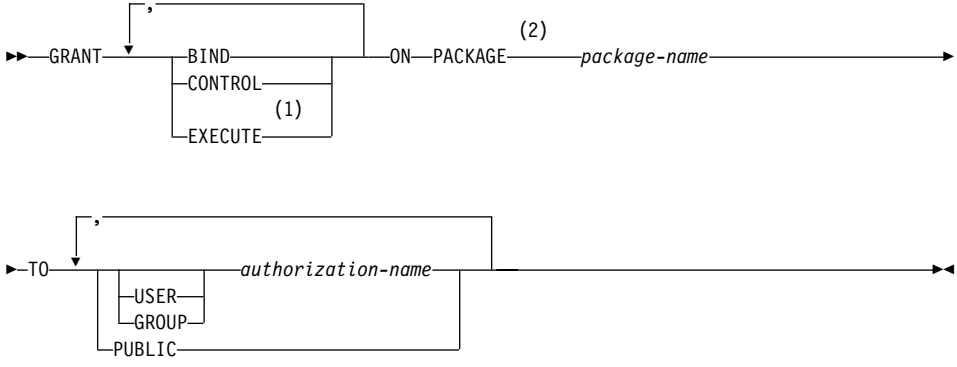
許可

このステートメントの許可 ID 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。

- 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。パッケージは、存在するために BINDADD 権限が与えられたユーザーによってすでにバインドされていなければならないので、 BIND 特権は実際には再バインド特権です。

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、除去、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・バインダーか、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。

EXECUTE
パッケージを実行する特権を与えます。

ON PACKAGE *package-name*
特権の対象となるパッケージ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 であるとみなされます。
  - *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 であるとみなされます。
  - 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、または DCE 認証が使用されている場合は、エラー (SQLSTATE 56092) が発生します。

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
```

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・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)。

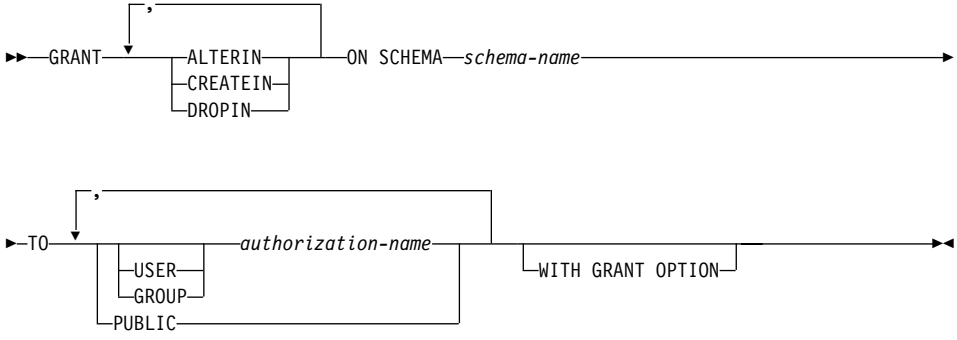
許可

このステートメントの許可 ID 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。

- *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。

構文



説明

ALTERIN

スキーマ内のすべてのオブジェクトの更新、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除去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 であるとみなされます。
  - 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 であるとみなされます。
  - 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、または DCE 認証が使用されている場合、エラー (SQLSTATE 56092) が発生します。
- 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付与のみを処理し、 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)。[98]

例

例 1: USER2 にスキーマ CORPDATA にオブジェクトを作成する特権を与えます。

```
GRANT CREATEIN ON SCHEMA CORPDATA TO USER2
```

例 2: BIGGUY にスキーマ CORPDATA のオブジェクトを作成および除去する特権を与えます。

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO BIGGUY
```

98. 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、授与者が授与の対象に対して NO 特権を持っている場合以外は警告が戻されます (SQLSTATE 01007)。

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。

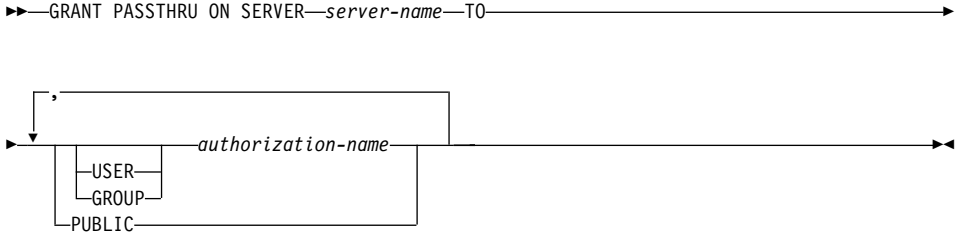
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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。それで、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。

GRANT (表、視点、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、視点、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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成することはできません (SQLSTATE 42509)。

許可

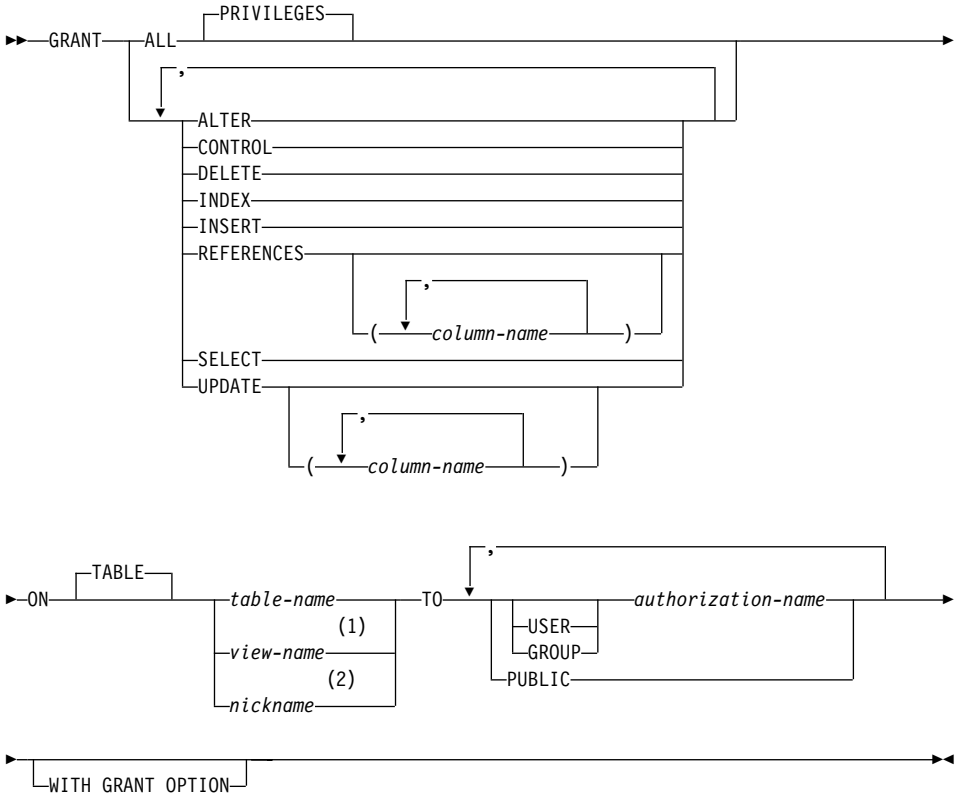
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

- 参照されている表、視点、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、視点、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視点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、視点には適用されません。

2 DELETE、INSERT、SELECT、および UPDATE 特権は、ニックネームには適用できません。

説明

ALL または **ALL PRIVILEGES**

ON 文節で指定する基礎表、視点、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、視点、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、視点、またはニックネームに対して持っているすべての授与可能な特権です。

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。

- 基礎表の定義に列を追加する。
- 基礎表の基本キー制約または固有制約を作成または除去する。基本キー制約または固有制約の作成または除去に必要な権限の詳細については、507ページの『ALTER TABLE』を参照してください。
- 基礎表の外部キーを作成または除去する。

  親表のそれぞれの列に対する REFERENCES 特権も必要です。
- 基礎表の検査制約を作成または除去する。
- 基礎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、または除去する。
- ニックネームの列名またはデータ・タイプを変更する。
- 基礎表、視点、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、
  - 基礎表に対する ALTER、CONTROL、DELETE、INSERT、INDEX、REFERENCES、SELECT、および UPDATE
  - 視点に対する 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) を授与する特権。
- 基礎表、視点、またはニックネームを除去する特権。

  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。RUNSTATS については、*コマンド解説書* を参照してください。
- 基礎表または要約表に対して SET INTEGRITY を発行する特権。

基礎表または要約表の定義者には、自動的に CONTROL 特権が付与されます。

視点の定義者に全選択で指定されているすべての表、視点、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視点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。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除去できます)。 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視点に行を挿入し、 IMPORT ユーティリティーを実行する特権を与えます。

REFERENCES
表を親として参照する外部キーの作成や除去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。外部キーの作成または除去に必要な権限については、507ページの『ALTER TABLE』 を参照してください。

ニックネームを参照するために外部キーを定義できなくても、この特権はニックネームに付与することができます。

REFERENCES (*column-name*,...)
列のリスト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除去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。タイプ付き表、タイプ付き視点、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。

- 表または視点から列を検索する特権。
- 表に視点を作成する特権。
- 表または視点に対して EXPORT ユーティリティーを実行する特権。EXPORT については、*コマンド解説書* を参照してください。

UPDATE
ON 文節で指定される表または更新可能な視点に対して 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
- その表または視点に対する CONTROL 特権
- その表または視点に対する UPDATE WITH GRANT OPTION

特権を与えられたユーザーは、付与者が付与特権を持っている表または視点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 が指定の表または視点に対して持っているすべての列の授与可能な 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視点の列を指定する非修飾名でなければなりません。 タイプ付き表、タイプ付き視点、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、視点、またはニックネームを指定します。

作動不能な視点または作動不能な要約表に対する特権を付与することはできません (SQLSTATE 51024)。宣言された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。[99]

グループに与えた特権は、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では使用されません。 それは、CREATE VIEW ステートメントの処理過程での基礎表に対する許可検査においても使用されません。

DB2 ユニバーサル・データベースの場合、グループに付与される表特権は、動的に作成された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;
```

ただし、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。[100]

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、

99. ステートメントを発行しているユーザーの許可 ID への授与に関する以前のバージョンでの制約はなくなりました。

100. 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バージョンでの制約は除かれました。

  - *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 であるとみなされます。
  - *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 であるとみなされます。
  - 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、または DCE 認証が使用されている場合は、エラー (SQLSTATE 56092) が発生します。
- 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付与のみを処理し、 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)。[101] 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

- 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 (表または視点) 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視点で必要とされます。
- ニックネームを参照するステートメントを処理する際、ニックネームに対する操作はデータ・ソースで使用する許可 ID の特権によって決まるため、DELETE、 INSERT、 SELECT、および UPDATE 特権がニックネームに定義されることはありません。

例

例 1: 表 WESTERN_CR に対するすべての特権を PUBLIC に与えます。

---

101. 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、授与者が授与の対象に対して何の特権も持っていない場合以外は警告が戻されます (SQLSTATE 01007)。

```
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 カタログ視点には 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最適化プログラム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
```

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

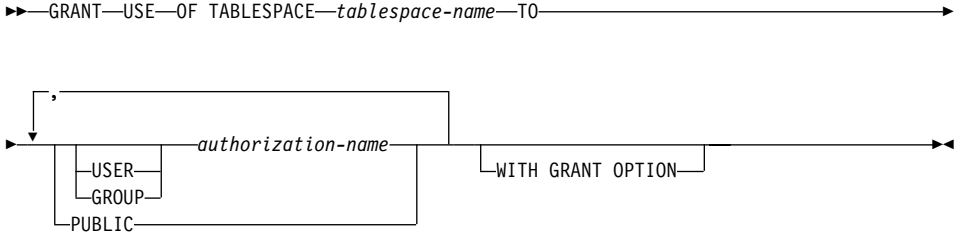
このステートメントは、アプリケーション・プログラムに組み込むか、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合、ステートメントを動的に準備することはできません (SQLSTATE 42509)。

許可

このステートメントの許可 ID 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。

- 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 特権と授与オプションが自動的に付与されます。

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) やシステム一時表スペース (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授与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 特権を授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 であるとみなされます。
- *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 であるとみなされます。
- 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、または DCE 認証が使用されている場合は、エラーが戻されます (SQLSTATE 56092)。

例

例 1: ユーザー BOBBY に、表スペース PLANS で表を作成する許可と、この特権を他のユーザーに授与する許可を与えます。

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

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のみ可能です。 これは、実行可能ステートメントではありません。

許可

権限は不要です。

構文

説明

SQLCA

SQL 連絡域 (SQLCA) の記述を組み込む (挿入) ことを指定します。 SQLCA については、1211ページの『付録B. SQL 連絡 (SQLCA)』 を参照してください。

SQLDA

SQL 記述子域 (SQLDA) の記述を組み込むことを指定します。 SQLDA については、1217ページの『付録C. SQL 記述子域 (SQLDA)』 を参照してください。

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 識別子、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 識別子は、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。

ホスト言語固有の情報については、*アプリケーション開発の手引き* を参照してください。

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・ステートメントによって置き換えられます。 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 識別子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;

(Print results)

}

EXEC SQL CLOSE C1;
```

INSERT

INSERT ステートメントは、表または視点に行を挿入します。 行を視点に挿入することは、その行をその視点の基礎表に挿入することでもあります。

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、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な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。

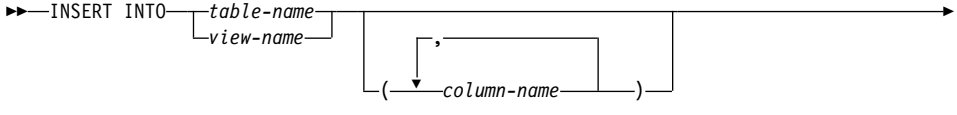
- 行を挿入する表または視点に対する INSERT 特権
- 行を挿入する表または視点に対する CONTROL 特権
- SYSADM または DBADM 権限

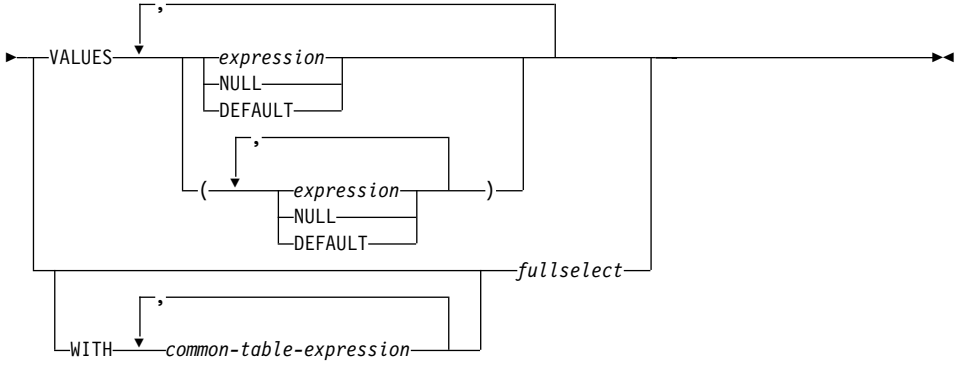
さらに、ステートメントの許可 ID には、 INSERT ステートメントで使用する全選択で参照される表または視点のそれぞれに対して、以下の特権の少なくとも 1 つが含まれている必要があります。

- 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 特権は検査されません。

構文





注: *common-table-expression* (共通表式) と *fullselect* (全選択) の構文については、 421ページの『第5章 照会』 を参照してください。

説明

INTO *table-name* または *view-name*

挿入操作の対象のオブジェクトを指定します。 *table-name* (表名) または *view-name* (視点名) は、それぞれアプリケーション・サーバーに存在する表または視点を指定していなければならず、カタログ表、要約表、カタログ表の視点、または読み取り専用の視点は指定できません。

視点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た列。
- その視点の他の列と同じ基礎表の列から得られた列。
- ニックネームから派生した列。

挿入操作の対象となる視点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、そのリストにそれらの列を指定してはなりません。

(*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、表または視点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視点の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、その表または視点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、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。

暗黙の列リストは、準備時に確立されます。 したがって、アプリケーション・プログラムに組み込んだ INSERT ステートメントでは、準備後に表または視点に追加された列は使用されません。

VALUES

挿入したい 1 つまたは複数の値を、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。

各行ごとの値の数は、列リストの名前の数と同じでなければなりません。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

176ページの『式』 で定義されている *expression* (式) を使用できます。

NULL

ヌル値を指定します。これはヌル値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 (771ページの『CREATE TABLE』 の *default-clause* を参照してください)。
- WITH DEFAULT 文節、GENERATED 文節、および NOT NULL 文節が使用されていない場合は、 NULL の値が挿入されます。
- 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が DEFAULT NULL は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)。

WITH *common-table-expression*
後に続く全選択で使用する共通表式 (*common-table-expression*) を定義します。 *common-table-expression* (共通表式) については、468ページの『共通表式』を参照してください。

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。

規則

- ***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ヌル値のいずれかになります。ヌル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、視点への挿入の場合、基礎表の列で、視点にはない列に挿入される値は、その列のデフォルト値か、またはヌル値のいずれかになります。したがって、基礎表に存在し、視点にはない行はすべて、デフォルト値があるか、またはヌル値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、 『第3章 言語要素』で説明されている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視点の基礎表に 1 つまたは複数の固有索引がある場合、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 を伴う視点が指定された場合、その視点に挿入する各行は、その視点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、895ページの『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ヌル値以外の値は、それぞれ親表の基本キーの値に等しくなければなりません。
- ***検査制約:*** 挿入値は、表に定義されている検査制約の検査条件を満たしていなければなりません。検査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。
- ***トリガー:*** 挿入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。
- ***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

- プログラムに組み込まれた 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、挿入された行の数を示します。 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
- ロックについての詳細は、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。
- 区分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。 詳細については、*アプリケーション開発の手引き* のバッファー挿入の項を参照してください。

- 識別列が含まれている表に行が挿入されると、DB2 は識別列の値を生成します。
  - GENERATED ALWAYS 識別列に対しては、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、その識別列に対して 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識別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。[102]
- GENERATED ALWAYS 識別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 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 識別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、 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。それで、指定された値が必ず固有なものとなるよう、識別列で固有の索引を作成する必要があります。

---

102. 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。

列のリストを指定せずに、GENERATED BY DEFAULT 識別列のある表に挿入するときは、識別列の値を表す DEFAULT キーワードを VALUES 文節で指定することができます。 DB2 は、指定された値を識別列に生成します。

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
  VALUES (DEFAULT, :hv_name, :hv_addr)
```

この例では、EMPID が識別列として定義され、この列に挿入される値は DB2 によって生成されます。

- 副選択を使用して識別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。識別列に値を指定できるのは、識別列が 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 の識別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、識別列の値を指定することができます。しかしこれは、T2 の識別列が 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、識別列に値を指定するとエラーが戻されます (SQLSTATE 428C9)。

  表に GENERATED ALWAYS 識別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 識別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。識別列の値は 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識別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、識別列に関連した 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、識別列に固有の索引が作成されていると想定します。識別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)、その識別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、識別列が 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 ステートメントで識別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 は識別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。

- 識別列に対して生成される値が識別列の最大値 (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、より広い範囲を持つ識別列 (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) で、新しい表の DROP および CREATE を実行する必要があります。

  たとえば、データ・タイプ SMALLINT で定義されている識別列があり、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。識別の列を INTEGER として再定義するには、データをアンロードし、表を除去し、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、START WITH の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、識別列の 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) を使用して照会を実行します。

例

例 1: DEPARTMENT 表に、以下の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 (ヌル) 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);
```

LOCK TABLE

LOCK TABLE ステートメントは、複数の並行するアプリケーション・プロセスによる表の変更や、表の使用を防止します。

呼び出し

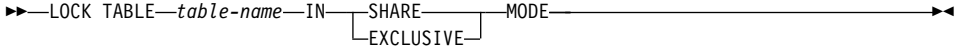
このステートメントは、アプリケーション・プログラムに組み込むか、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。

許可

このステートメントの許可 ID には、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 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
該当の表を指定します。 *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。ニックネーム (SQLSTATE 42809) や宣言された一時表 (SQLSTATE 42995) を指定することはできません。 *table-name* がタイプ付き表である場合、その表は表階層のルート表でなければなりません (SQLSTATE 428DR)。

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

- ロックは、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。
- 区分データベースでは、表ロックはノード・グループ内の最初の区分 (もっとも番号の小さい区分) で最初に獲得され、その後他の区分で獲得されます。 LOCK TABLE ステートメントが割り込まれると、表は一部の区分ではロックされ、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区分に対してロックを完了するか、 COMMIT または ROLLBACK ステートメントを出して現在のロックを解放します。
- このステートメントは、ノード・グループのすべての区分に影響を与えます。

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
```

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。

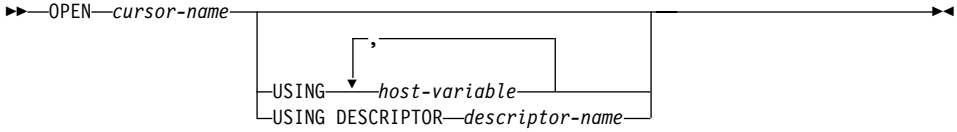
呼び出し

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、 914ページの『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。
- 準備される 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 ステートメントに指定されているホスト変数の現行値、またはこの 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を使用して、その SELECT ステートメントを評価することによって求められます。 結果表の行は、OPEN ステート

メントの実行の過程で求められ、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求められます。 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、準備される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。 (パラメーター・マーカーについては、1042ページの『PREPARE』 を参照してください。) 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されるステートメントを指定した場合、 USING の使用は必須です。 準備される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、準備される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 に割り振る記憶域のバイト数を示す SQLDABC
- ステートメントの処理中に SQLDA で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記憶域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。 SQLDOUBLED と LOB の列については、1225ページの『SQLDA に対する DESCRIBE の効果』 を参照してください。

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。詳しくは、1217ページの『付録C. SQL 記述子域 (SQLDA)』を参照してください。

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。 タイプ付きパラメーター・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・マーカーの文脈に従って決定されます。パラメーター・マーカーに関連する規則については、 1043ページの『規則』 を参照してください。
- 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。

  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される 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指定変更されることになります。

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、カーソルは、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、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、一時表方式が使用される場合もあります。この方式では、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 2 つ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。

  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ます。 *アプリケーション開発の手引き* は、並行する作業単位によって実行される INSERT、UPDATE、および DELETE の操作の影響を制御するロックの使用方法について説明しています。 結果表は、自分自身の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。詳細については、976ページの『注』 を参照してください。

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のに使用します。
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) の値を渡すのに使用します。 (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 SQ  BEGIN DECLARE SECTION;
  static short    hv_int;
  char            hv_vchar64[64];
  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;
```

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。

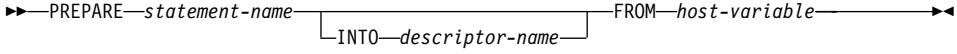
呼び出し

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のみ可能です。これは、動的に準備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。

INTO

INTO を使用すると、PREPARE ステートメントが正常に実行された場合に、準備済みステートメントについての情報が、 *descriptor-name* で指定する SQLDA に入れられます。

descriptor-name
SQLDA の名前です。[103]

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。これは、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
  - COMMENT ON
  - COMMIT
  - CREATE
  - DECLARE GLOBAL TEMPORARY TABLE
  - DELETE
  - DROP
  - EXPLAIN
  - FLUSH EVENT MONITOR
  - GRANT
  - INSERT
  - LOCK TABLE
  - REFRESH TABLE
  - RELEASE SAVEPOINT
  - RENAME TABLE
  - RENAME TABLESPACE
  - REVOKE
  - ROLLBACK

103. この文節の代わりに、DESCRIBE ステートメントを使用できます。 936ページの『DESCRIBE』 を参照してください。

- SAVEPOINT
- 選択ステートメント
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- 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。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った場合、パラメーター・マーカーは、ホスト変数を記述できる場所に宣言します。 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、 1036ページの『OPEN』 と 975ページの『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。

  タイプ付きパラメーター・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、実行時のパラメーター・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、実行時に与えられます。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、その文脈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。

タイプ付きパラメーター・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28 に示しています。この表で、位置は、式、述部、および関数に類別されており、タイプなしパラメーター・マーカーが使用可能か否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 28. 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。

表 28. 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
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 以外の文字データ・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)。
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。これは変形関数の戻りタイプも示している。
単純な CASE 式の CASE 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 121ページの『結果のデータ・タイプに関する規則』を適用した結果。

表 28. 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
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、 結果のデータ・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121ページの『結果のデータ・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。これは変形関数の戻りタイプも示している。
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。これは変形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

表 28. 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
BETWEEN 述部の 第 1 と第 2、または 第 1 と第 3 オペランド	もう一方のオペランドと同じ。
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121ページの『結果のデータ・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 オペランドと第 2 オペランドの両方	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のデータ・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。
IN リストの第 1 オペランドと、それ以外のオペランドのゼロ個以上	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のデータ・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 (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 28. 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 (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) とみなされます。
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 (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) とみなされます。
NULL 述部のオペランド	エラー
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 1 つ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 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 121ページの『結果のデータ・タイプに関する規則』を適用した結果。
もう一方のオペランドがタイプなしパラメーター・マーカー以外の NULLIF のオペランド。	もう一方のオペランドのデータ・タイプ
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グラフィック・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

表 28. 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
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 オペランドがグラフィック・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 オペランドがグラフィック・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
ユーザー定義関数を含む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

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。

参照が可能なステートメント	準備済みステートメント...
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。詳細については、976ページの『注』 を参照してください。
- サポートされるホスト言語での動的 SQL ステートメントの例については、*アプリケーション開発の手引き* を参照してください。

例

例 1: 選択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 と同様に選択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。

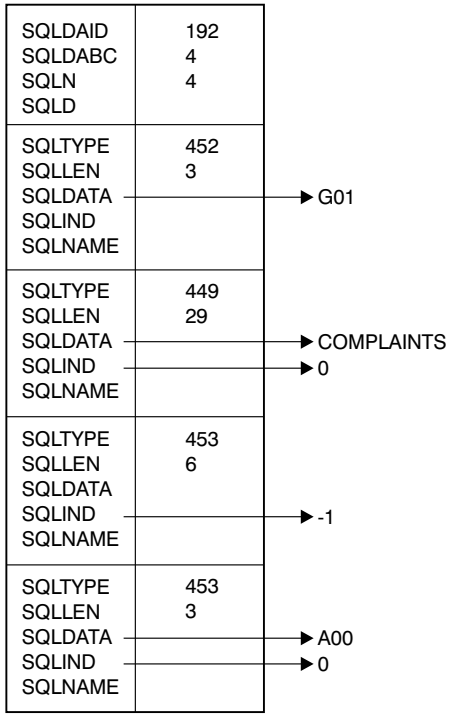
```
EXEC SQL  PREPARE STMT_NAME FROM :holder;
EXEC SQL  EXECUTE STMT_NAME USING DESCRIPTOR :insert_da;
```

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 は上記の値を持っていなければなりません。

REFRESH TABLE

REFRESH TABLE ステートメントは、要約表内のデータを最新表示します。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な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、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
- 表に対する CONTROL 特権

構文

説明

table-name
表名を指定します。

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要約表が含まれます。

- REFRESH IMMEDIATE
- REFRESH DEFERRED

RELEASE (接続)

このステートメントは、1 つまたは複数の接続を解放保留状態にします。

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、動的に準備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* に入っている *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保留状態になります。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。 オプションのキーワードである SQL は、 DB2/MVS の SQL 構文との互換性を保つ目的で含まれています。

注

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、指定された保管点の保持を、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、その保管点までロールバックすることはできなくなります。

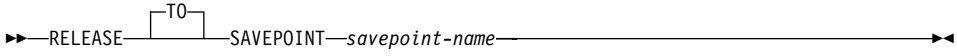
呼び出し

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、動的に準備可能な実行可能ステートメントです。

許可

権限は不要です。

構文



説明

savepoint-name
解放する保管点を指定します。解放された保管点までロールバックすることはできなくなります。指定された保管点が存在しない場合は、エラーが戻されます (SQLSTATE 3B001)。

注

- 同じ保管点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 度解放された保管点の名前を他の 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 という保管点を解放します。

```
RELEASE SAVEPOINT SAVEPOINT1
```

RENAME TABLE

RENAME TABLE ステートメントは、既存の表の名前を変更します。

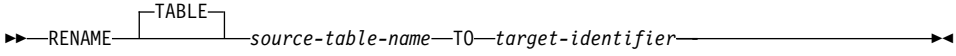
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、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、DBADM、または CONTROL 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

source-table-name

その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)。その表を指定する別名であっても構いません。この名前に、カタログ表 (SQLSTATE 42832)、要約表、タイプ付き表 (SQLSTATE 42997)、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。

target-identifier

該当の表のスキーマ名を伴わない新しい名前を指定します。*source-table-name* のスキーマ名が、表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、視点、または別名を指定するものであってはなりません (SQLSTATE 42710)。

規則

ソース表は、以下に該当していてはなりません。

- 既存の視点定義または要約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
- 検査制約がある
- 識別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
- 既存の参照列の効力範囲内である

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。

注

- カタログ項目が更新され、新しい表名が反映されます。
- ソース表名に関連するすべての 許可は、新しい表名に転送 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)。
- 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。
- *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。
- 基本キー制約または固有制約のある表の名前は、基本キーまたは固有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 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。

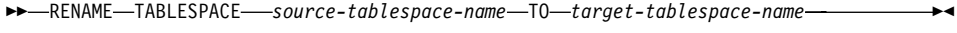
呼び出し

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、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 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 識別子です (通常識別子または区切り識別子)。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 識別子です (通常識別子または区切り識別子)。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)。
- 「ロールフォワード保留中」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復元するときに、新しい表スペース名を使用する必要があります。バックアップの復元に関する詳細は、*管理 API 解説書* または *コマンド解説書* を参照してください。

例

表スペースの名前 USERSPACE1 を DATA2000 に変更します。

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成することはできません (SQLSTATE 42509)。

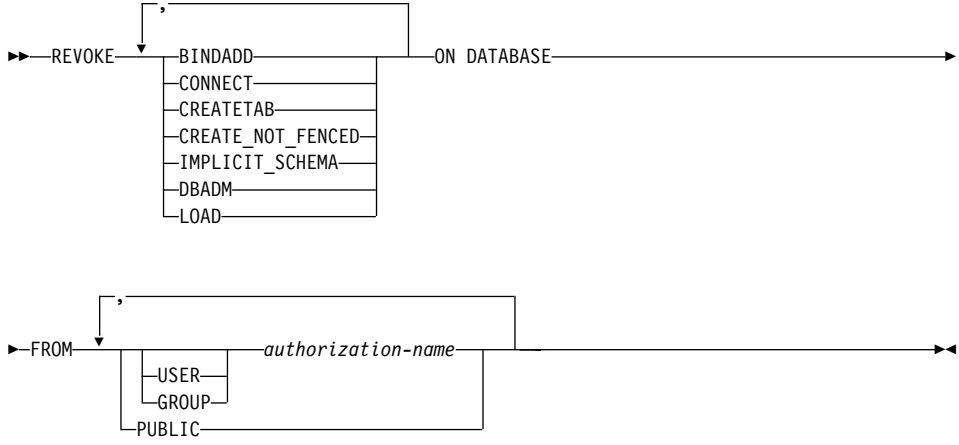
許可

このステートメントの許可 ID 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。

- 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 *許可 ID* 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 *許可 ID* 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 *許可 ID* から CREATETAB 権限を取り消すことはできません (SQLSTATE 42504)。

CREATE_NOT_FENCED
データベース・マネージャーの処理の中で実行する関数を登録する権限を取り消します。 ただし、関数が非分離としていったん登録されると、それ以降にその関数を登録した許可 ID から CREATE_NOT_FENCED 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 *許可 ID* から CREATE_NOT_FENCED 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)。

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 BINDADD、CONNECT、CREATETAB、IMPLICIT_SCHEMA、または CREATE_NOT_FENCED のいずれの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。

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。

規則

- USER も GROUP も指定しない場合には、
  - SYSCAT.DBAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合には、 USER であるとみなされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
  - 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)。
  - DCE 認証が使用されている場合、エラーが発生します (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: 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 カタログ視点には、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)。

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。

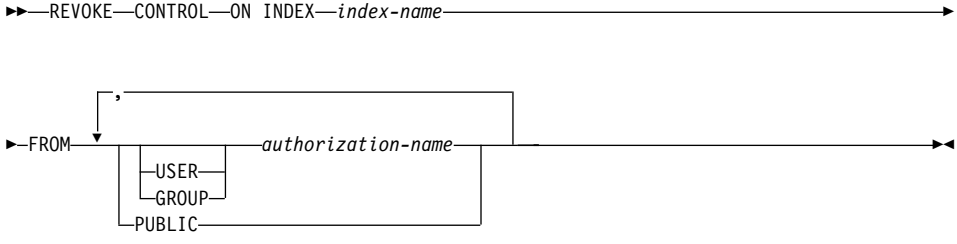
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に作成できるステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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
索引を除去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。

規則

- USER も GROUP も指定しない場合には、
  SYSCAT.INDEXAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合には、 USER であるとみなされます。
  すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
  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)。
  DCE 認証が使用されている場合、エラーが発生します (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 を除去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 を除去する特権を取り消します。

```
REVOKE CONTROL ON INDEX LUNCHITEMS
   FROM USER CHEF, GROUP WAITERS
```

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に作成できるステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成することはできません (SQLSTATE 42509)。

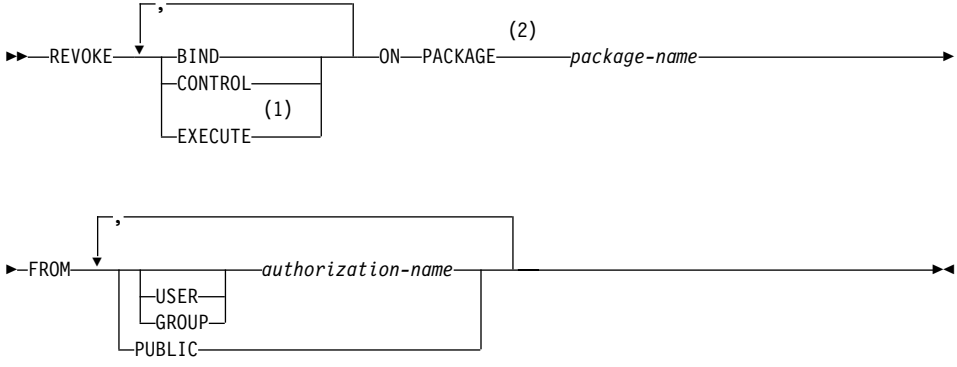
許可

このステートメントの許可 ID 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。

- 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 *許可 ID* から BIND 特権を取り消すことはできません。

CONTROL
パッケージを除去する特権、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 *許可 ID* から EXECUTE 特権を取り消すことはできません。

ON PACKAGE *package-name*
特権を取り消す対象のパッケージを指定します。

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。

規則

- USER も GROUP も指定しない場合には、

- SYSCAT.PACKAGEAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合、USER であるとみなされます。
- すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
- 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)。
- DCE 認証が使用されている場合、エラーが発生します (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
```

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。

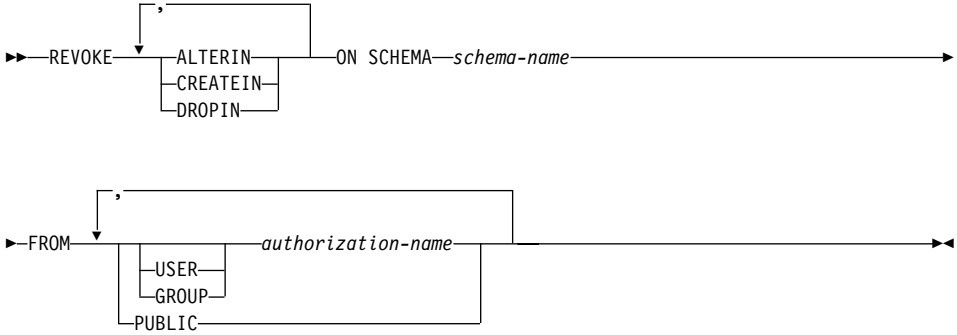
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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更新、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除去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。

規則

- USER も GROUP も指定しない場合には、
  - SYSCAT.SCHEMAAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合、 USER であるとみなされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
  - 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)。
  - DCE 認証が使用されている場合、エラーが発生します (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 のオブジェクトを除去する特権を取り消します。

```
REVOKE DROPIN ON SCHEMA LUNCH
   FROM USER CHEF, GROUP WAITERS
```

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。

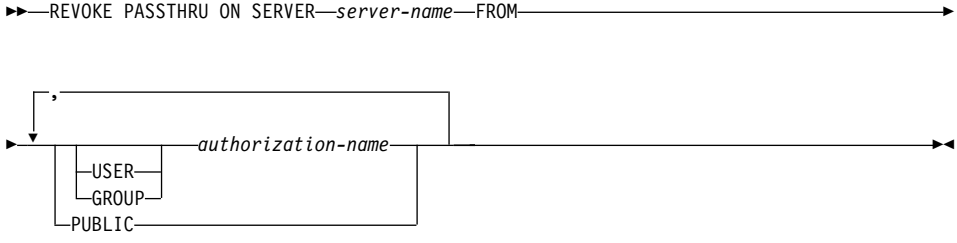
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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。

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。

REVOKE (表、視点、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、視点、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か、あるいは動的 SQL ステートメントの使用によって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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成することはできません (SQLSTATE 42509)。

許可

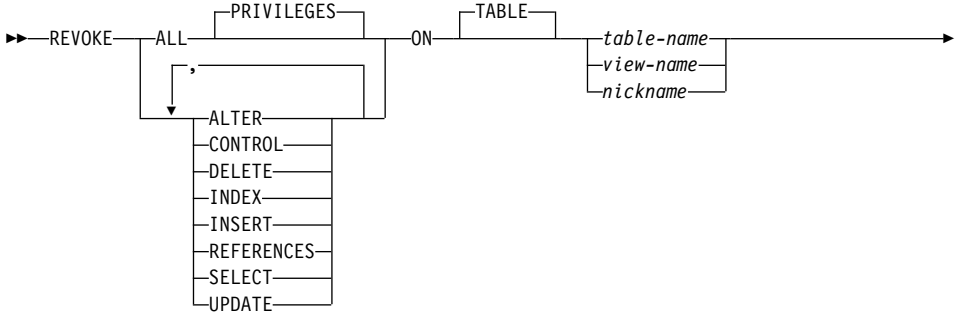
このステートメントの許可 ID には、以下の特権が少なくとも 1 つ含まれている必要があります。

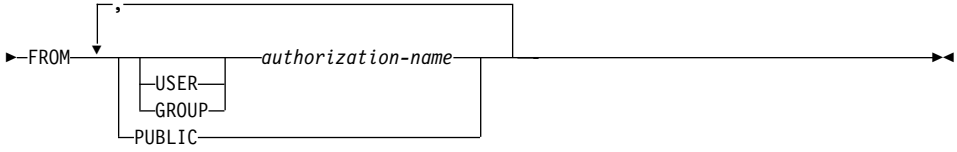
- SYSADM または DBADM 権限
- 参照されている表、視点、またはニックネームに対する CONTROL 特権

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。

カタログの表や視点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、視点、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取り消します。

ALL を使用しない場合、以下に示す 1 つまたは複数のキーワードを使用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、視点、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
基礎表の定義への列の追加、表の基本キーまたは固有制約の作成または除去、表の外部キーの作成または除去、表、視点、またはニックネームに対するコメントの追加や変更、検査制約の作成または除去、トリガーの作成、リセットの追加、ニックネームの列オプションの除去、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、視点、またはニックネームを除去する権限、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
表または更新可能な視点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除去できます)。 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視点に行を挿入したり、 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、または除去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
表または視点からの行の検索、表に対する視点の作成、および表または視点に対して 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、視点によっては作動不能になるものがあります。 作動不能な視点については、904ページの『注』 を参照してください。

UPDATE
表または更新可能な視点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、視点、またはニックネームを指定します。 *table-name* には、宣言された一次表を指定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。

規則

- USER も GROUP も指定しない場合には、

- SYSCAT.TABAUTH および SYSCAT.COLAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合には、USER であるとみなされます。
- すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
- 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)。
- DCE 認証が使用されている場合、エラーが発生します (SQLSTATE 56092)。

注

- 視点の作成に使用された *authorization-name* (これは、 SYSCAT.VIEWS における視点の DEFINER と呼ばれる) から特権が取り消されると、従属する視点の特権も取り消されます。
- 視点の DEFINER が、その視点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、 (またはその視点定義が従属するオブジェクトが除去されるか、または別の視点のために作動不能になった場合)、その視点は作動不能になります (作動不能の視点については、 895ページの『CREATE VIEW』 の『注意事項』の項を参照してください)。

  ただし、DBADM または SYSADM が明示的に DEFINER から視点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、視点には何も影響がなく作動可能のままになります。
- 作動不能な視点に対する特権は取り消すことはできません。
- 特権を取り消す対象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、付加的な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、または視点の 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副視点で 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、または視点で DEREF(*Y*) が使用されていて、 *Y* が *Z* という表または視点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副視点で 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視点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、視点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視点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。
- 要約表の DEFINER が、要約表定義が従属している表に対する SELECT 特権を失った場合、 (または要約表定義が従属する表が除去される場合)、要約表は作動不能になります (作動不能の要約表については、 816ページの『注』を参照してください)。

  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、要約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 (表または視点) 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除去できない SQL 関数がある場合は、そのSQL 関数で*直接*または*間接的*に参照される表や視点に対する SELECT 特権も取り消せない場合があります (SQLSTATE 42893)。

注: 表や視点などのオブジェクト相互の可能な従属関係についてのリストは、963ページの『規則』 に示されています。

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 カタログ視点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 カタログ視点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 カタログ視点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
```

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。

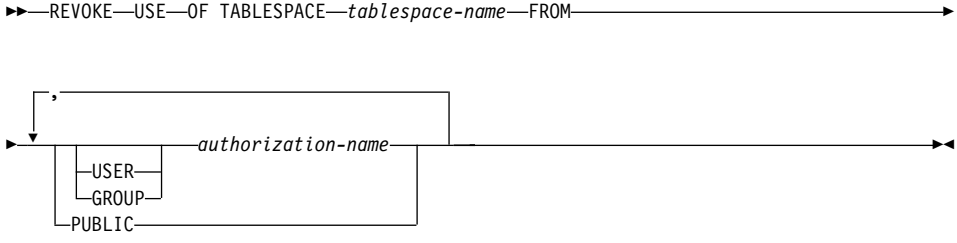
呼び出し

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か、または動的 SQL ステートメントを使用して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。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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。

規則

- USER も GROUP も指定しない場合には、
  - SYSCAT.TBSPACEAUTH カタログ視点の権限保持者のすべての行の GRANTEETYPE が U の場合、USER であるとみなされます。
  - すべての行の GRANTEETYPE が G の場合、GROUP であるとみなされます。
  - 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)。
  - DCE 認証が使用されている場合は、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
```

ROLLBACK

ROLLBACK ステートメントは、作業単位または保管点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。

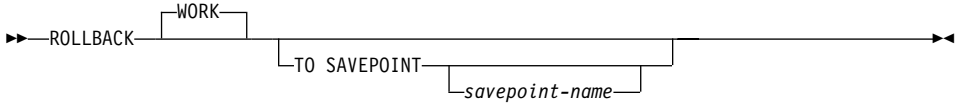
呼び出し

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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。

許可

権限は不要です。

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 ステートメントの発行とは独立しています。

- SET CONNECTION,
- SET CURRENT DEGREE,
- SET CURRENT DEFAULT TRANSFORM GROUP,
- SET CURRENT EXPLAIN MODE,
- SET CURRENT EXPLAIN SNAPSHOT,
- SET CURRENT PACKAGESET,
- SET CURRENT QUERY OPTIMIZATION,
- SET CURRENT REFRESH AGE,
- SET EVENT MONITOR STATE,
- SET PASSTHRU,
- SET PATH,
- SET SCHEMA,
- SET SERVER OPTION.

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。活動中の保管点がない場合は、SQL エラーが戻されます (SQLSTATE 3B502)。保管点は、ROLLBACK が正常に完了した後もそのまま存続します。 *savepoint-name* が指定されない場合は、最後に設定された保管点までロールバックが実行されます。

この文節を省略して ROLLBACK WORK 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保管点は解放されます。

savepoint-name

どの保管点 (*savepoint-name*) までロールバックするかを指定します。*savepoint-name* によって定義された保管点は、 ROLLBACK が正常に完了した後もそのまま存続します。指定された名前の保管点が存在しない場合は、エラーが戻されます (SQLSTATE 3B001)。保管点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。詳細については、976ページの『注』 を参照してください。
- アトミック複合ステートメントやトリガーのようなアトミック実行のコンテキストでは、保管点を使用できません。
- ROLLBACK TO SAVEPOINT がカーソルに与える影響は、保管点に含まれているステートメントによって異なります。
  - 保管点に 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。

  - 保管点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。[104]
  - 保管点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、保管点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、保管点の中で指定されていた宣言された一時表はすべて除去されます。宣言された一時表を保管点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
```

104. 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。

SAVEPOINT

SAVEPOINT ステートメントを使用して、トランザクション内に保管点を設定します。

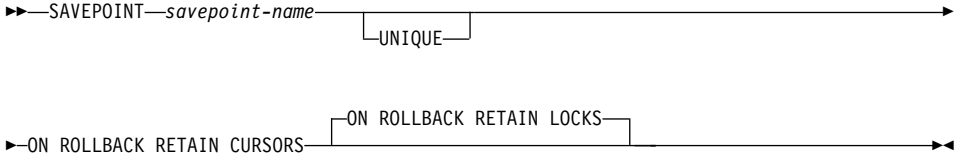
呼び出し

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、動的に準備可能な実行可能ステートメントです。

許可

権限は不要です。

構文



説明

savepoint-name
savepoint (保管点) の名前を指定します。

UNIQUE
UNIQUE 保管点を指定すると、保管点が活動中の間、この保管点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示します。

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、この保管点へのロールバックでのシステムの動作を指定します。ここで、RETAIN CURSORS 文節は、保管点へのロールバックにおいて、可能な限りカーソルを変更しないことを示します。どのような場合にカーソルが保管点へのロールバックから影響を受けるかについては、 1083ページの『ROLLBACK』を参照してください。

ON ROLLBACK RETAIN LOCKS
保管点の設定後にかけられるロックに関して、この保管点へのロールバッ

クでのシステムの動作を指定します。この保管点以降に掛けられたロックは、この保管点へのロールバックでは追跡およびロールバック (解放) されません。

規則

- 保管点をネストすることはできません。すでに確立された保管点が存在している状態で SAVEPOINT ステートメントを実行すると、エラーが戻されます (SQLSTATE 3B002)。

注

- UNIQUE キーワードは、DB2 ユニバーサル・データベース (OS/390 版) と互換性を持つシステムで使用します。 DB2 ユニバーサル・データベース (OS/390 版) における動作について以下に説明します。

  保管点の名前 *savepoint-name* がトランザクション内にすでに存在していると、エラーが戻されます (SQLSTATE 3B501)。 UNIQUE 文節を省略してステートメントが実行された場合は、アプリケーションがトランザクション内でこの保管点の名前を再使用する可能性があります。指定された *savepoint-name* がトランザクション内に存在しているときは、既存の保管点が破棄され、*savepoint-name* の名前で新しい保管点が作成されます。

  他の保管点で名前を再使用するために保管点が破棄されるのと、 RELEASE SAVEPOINT ステートメントによって古い保管点が解放されるのとは異なります。名前の再使用によって保管点が破棄される場合、破棄されるのはその保管点だけです。一方 RELEASE SAVEPOINT ステートメントを使用して保管点を解放した場合は、指定された保管点だけでなく、その保管点より後に確立されたすべての保管点が解放されます。
- ある保管点において、処理の途中でユーティリティー、SQL、または DB2 コマンドが断続的に COMMIT ステートメントを実行した場合、その保管点は暗黙的に解放されます。
- SQL ステートメント SET INTEGRITY は、保管点における DDL ステートメントと同じ働きをします。
- アプリケーションでは、挿入がバッファーに入れられる場合があります (アプリケーションが INSERT BUF オプションで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。

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。 詳細については、467ページの『選択ステートメント』 および 422ページの『副選択』 を参照してください。

SELECT INTO

SELECT INTO ステートメントは、多くても 1 つの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、および 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のみ可能です。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。

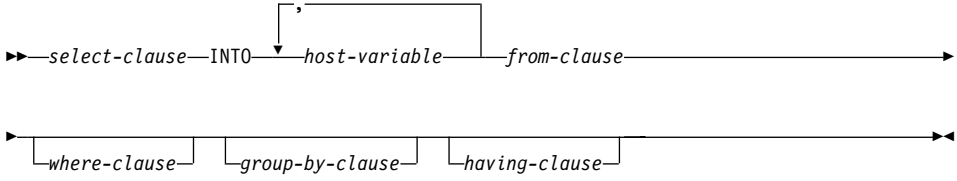
許可

この SELECT INTO ステートメントの許可 ID の特権には、このステートメントで参照される各表または視点に対する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 SELECT INTO ステートメントの場合、GROUP 特権は検査されません。

構文



説明

select-clause (SELECT 文節)、 *from-clause* (FROM 文節)、 *where-clause* (WHERE 文節)、 *group-by-clause* (GROUP BY 文節)、および *having-clause* (HAVING 文節) についての説明は、 421ページの『第5章 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 (1211ページの『付録B. SQL 連絡 (SQLCA)』 を参照)。

各変数には、106ページの『割り当てと比較』で説明されている規則に基づいて値が割り当てられます。 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 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 の従業員 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';
```

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、動的に準備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* に入っている *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、タイプ 1 CONNECT で説明した方法と同じ方法で更新されます。詳細については 592 ページを参照してください。

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 (44ページの『分散作業単位の意味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、次に現行状態に復元すると、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。

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な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

変形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 識別子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、指定された変形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)。
- 変形グループ *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。
- 特殊レジスターの使用に関するその他の規則については、133ページの『CURRENT DEFAULT TRANSFORM GROUP』を参照してください。

例

例 1: デフォルトの変形グループを MYSTRUCT1 に設定します。変形グループ 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
```

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。

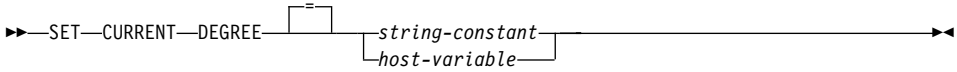
呼び出し

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な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区画内並行性は使用されません。

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、指定した度合いの区画内並行性を使用できます。

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区画内並行性を使用できます。

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する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 ステートメントの区画内並行性の度合いは、 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。 これらのコマンドの詳細については、*コマンド解説書* を参照してください。

区画内並行性の実際の実行時の度合いは、以下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

区画内並行性を使用するには、 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区画内並行性を使用しません (SQLSTATE 01623)。

SQL ステートメントによっては、区画内並行性を使用できません。 区画内並行性の度合いの詳細、および制約事項のリストについては、 *管理の手引き* を参照してください。

例

例 1: 以下のステートメントは、区画内並行性を禁止する 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、区画内並行性を許可する 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。

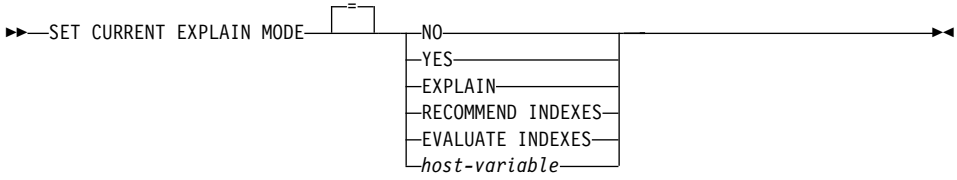
呼び出し

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、動的に準備可能な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 情報は取り込まれません。 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 情報を取り込みます。 ただし、動的ステートメントは実行されません。

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 情報が取り込まれますが、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。 最適化プログラムは、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。

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する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取り込むことができます。 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、実行時に動的 SQL ステートメントの Explain 情報が取り込まれます。 CURRENT EXPLAIN MODE レジスターの値が NO 以外の場合、 EXPLAIN バインド・オプションの値は無視されます。 EXPLAIN オプションと CURRENT EXPLAIN MODE 特殊レジスターとの相互間の作用については、
1438ページの表143 を参照してください。

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 コマンドだけです。 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 SNAPSHOT コマンド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。

詳細については、*管理の手引き* を参照してください。

例

例 1: 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
```

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。

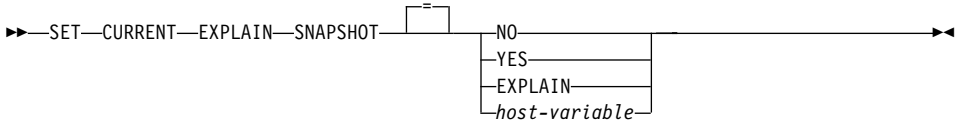
呼び出し

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、動的に準備可能な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 (1401ページの『付録K. Explain 表と定義』 を参照) 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。

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する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、実行時に動的 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。 EXPLSNAP オプションと CURRENT EXPLAIN SNAPSHOT 特殊レジスターとの相互作用については、 1439ページの表144 を参照してください。

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。

詳細については、*管理の手引き* を参照してください。

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;
```

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 (コレクション識別子) 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。

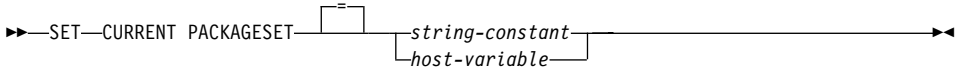
呼び出し

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のみ可能です。これは、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

権限は不要です。

構文



説明

string-constant

30 文字を超えない文字ストリング定数。最大長を超える場合、その部分は実行時に切り捨てられます。

host-variable

30 文字を超えないタイプ CHAR または VARCHAR の変数。 NULL に設定することはできません。最大長を超える場合、その部分は実行時に切り捨てられます。

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 バインド・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。詳細については、*コマンド解説書* を参照してください。

- DB2 (MVS/ESA 版) 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、異なるバインド・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。

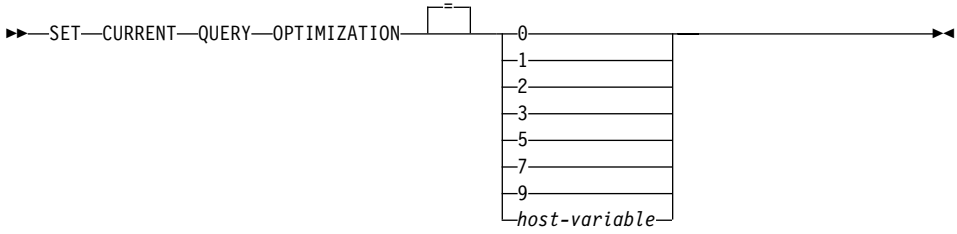
呼び出し

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な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、または実行時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 (詳細については、*管理の手引き* を参照してください)。

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、単純な動的 SQL により、適切な索引を伴う表にアクセスする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礎表ではなく要約表が使用されます。

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・プランを生成できるかどうか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、および必要な資源に影響を与えます。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* が消費する資源が、照会の実行に必要な資源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、消費する資源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。それぞれの照会最適化クラスの詳細については、*管理の手引き* を参照してください。

- 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、最近時に実行された 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、前処理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。詳細については、*コマンド解説書* の BIND コマンドの項を参照してください。
- 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 カタログ視点を使用して、 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。

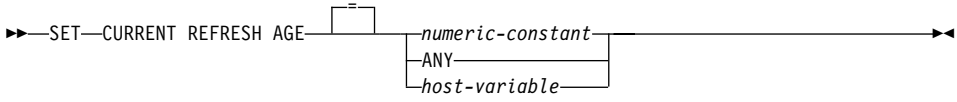
呼び出し

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、動的に準備可能な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)。

0
: REFRESH IMMEDIATE で定義された要約表だけを使って照会処理を最適化できることを示しています。

99999999999999
: REFRESH DEFERRED または REFRESH IMMEDIATE で定義された任意の要約表を使って照会処理を最適化できることを示しています。この値は 9 999 年、99 か月、99 日、99 時間、99 分、99 秒を表します。

ANY

これは、99999999999999 を簡略化したものです。

host-variable

タイプ DECIMAL(20,6) の値、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 99 999 999 999 999.000000 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 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。 基礎となる基礎表の値を表さない要約表を使って照会処理を最適化できるようにすると、照会結果が基礎表のデータを正確に反映しない場合があります。 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 99 999 999 999 999.000000 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、 CURMAXAGE という名前のホスト変数に CURRENT REFRESH AGE 特殊レジスターの現行値を入れます。

```
EXEC SQL VALUES (CURRENT REFRESH AGE) INTO  :CURMAXAGE;
```

値は、前の例で設定された 99999999999999.000000 になります。

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 (活動状態または非活動状態) 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。

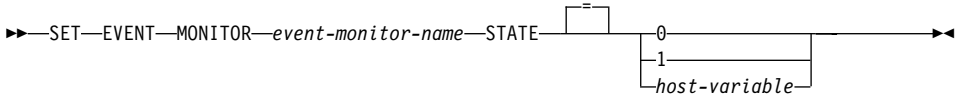
呼び出し

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か、または動的 SQL ステートメントを使用して発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成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 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、または実行時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・モニターはすでに活動状態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、同時に活動化できるイベント・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。

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[105] ステートメントは、以下を行う場合に使用します。

- 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、検査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 を指定した要約表である場合、データの即時最新表示はオフになります。これによって、表は、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ない*検査保留状態* になる点に注意してください。 基本キー制約と固有制約は引き続き検査されます。
- 1 つまたは複数の表についての両方の保全性検査をオンに戻し、据え置かれている検査をすべて実行する。表が要約表で、データが必要に応じて最新表示される場合、 REFRESH IMMEDIATE 属性を定義すると、データの即時最新表示はオンになります。
- 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 属性で定義した要約表であれば、データの即時最新表示はオンになります。
- 表がすでにデータ・リンク調整保留 (DRP) またはデータ・リンク調整不能 (DRNP) 状態である場合に、表を検査保留状態にする。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、検査保留状態になります。

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、システムはデフォルトで、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。ただし、データの保全性を保証するために、システムは (制約に違反する表全体を検査するという) 完全検査を行うことにする場合もあります。 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することが必要な場合もあります。詳細については、1119ページの『注』 を参照してください。

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か、または動的 SQL ステートメントを使用して発行することができます。このステート

105. DB2 では、保全性の検査を行う方法として、 SET CONSTRAINTS ステートメントよりも SET INTEGRITY ステートメントが頻繁に使用されます。

メントは、動的に準備可能な実行可能ステートメントです。しかし、バインド・オプション DYNAMICRULES BIND を適用する場合は、ステートメントを動的に作成することはできません (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。

1. 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。

   - 参照保全制約の表とそのすべての従属表および子孫に対する CONTROL 特権
   - SYSADM または DBADM 権限
   - LOAD 権限

2. 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。

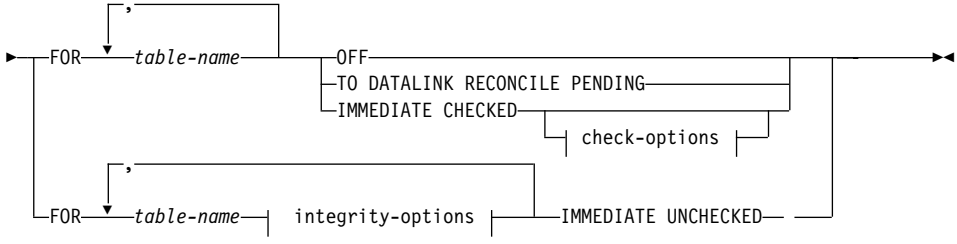
   - SYSADM または DBADM 権限
   - 検査する表に対する CONTROL 特権、**および** 1 つまたは複数の表に対して例外が追加された場合は例外表に対する INSERT 特権
   - LOAD 権限。 1 つまたは複数の表に対して例外が通知される場合は、以下の特権も必要。
     - 検査されるそれぞれの表に対する SELECT および DELETE 特権、および
     - 例外表に対する INSERT 特権

3. 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。

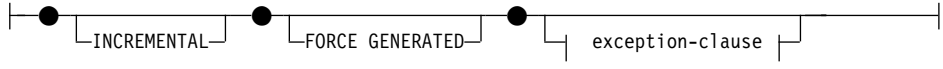
   このステートメントの許可 ID には、少なくとも次のいずれかが必要です。

   - SYSADM または DBADM 権限
   - 検査する表に対する CONTROL 特権
   - LOAD 権限

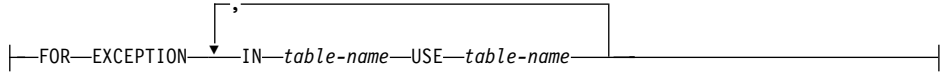
構文



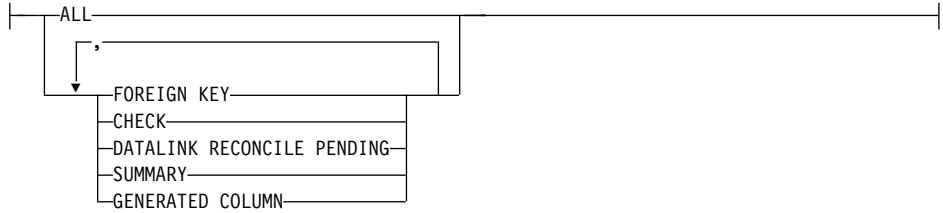
check-options:



exception-clause:



integrity-options:



注:

1 以前のバージョンとの互換性を保つために、 キーワード CONSTRAINTS は引き続きサポートされています。

説明

table-name

保全性処理を行う表を指定します。これは、カタログに記述されている表でなければならず、視点、カタログ表、またはタイプ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、検査制約、および列の生成をオフにして、表を検査保

留状態にすることを指定します。表が要約表であれば、即時最新表示はオフになり (該当する場合)、要約表は検査保留状態に置かれます。

表が、保全性検査の 1 つのタイプだけがオフの、検査保留状態にすで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。

リストに指定する表のいずれかが親表である場合、外部キー制約に関する検査保留状態は、その従属表と子孫表のすべてに拡張されます。

リストに指定する表のいずれかが要約表の基礎表である場合、検査保留状態はそのような要約表にも拡張されます。

検査保留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。 それらの制約は、1119ページの『注』にリストされています。

TO DATALINK RECONCILE PENDING

表の DATALINK 保全性制約検査をオフにし、検査保留状態にするよう指定します。 表がすでにデータ・リンク調整不能 (DRNP) 状態である場合は、そのまま検査保留状態になります。 そうでない場合には、表はデータ・リンク調整保留 (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、

- STATUS の値は C (表が検査保留状態) でなければなりません。それ以外の場合には、エラー (SQLSTATE 51027) になります。
- 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。

表が要約表であれば、データは必要に応じて照会で検査され、最新表示されます。

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。 表は検査保留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。 これにより、DATALINK 値の調整をある時間まで据え置いておいて、その間に表を使用することが可能になります。

check-options

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。

INCREMENTAL
表の追加部分 (もしあれば) に対して保全性検査が据え置かれているアプリケーションを指定します。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。属性を指定しない場合、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。増分処理よりも完全処理 (表全体の保全性検査) を好むシステムについては、注を参照してください。また、INCREMENTAL オプションを必要とする状態と、INCREMENTAL オプションを指定できない状態については、注を参照してください。

表が検査保留状態にない場合、エラー (SQLSTATE 55019) が戻されます。

exception-clause

FOR EXCEPTION
外部キー制約または検査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。 それらのユーザー定義表については、1447ページの『付録N. 例外表』 を参照してください。 エラーが検出されても、制約は再びオンに戻り、表は検査保留状態ではなくなります。 1 つまたは複数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検査保留状態のままになります。 *table-name* が要約表であれば、この文節は指定できません (SQLSTATE 42997)。

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。

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。

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
すべての保全性オプションをオンにすることを指定します。

FOREIGN KEY
外部キー制約をオンにすることを指定します。

CHECK
検査制約をオンにすることを指定します。

DATALINK RECONCILE PENDING
DATALINK 保全性制約をオンにすることを指定します。

SUMMARY
REFRESH IMMEDIATE 属性が指定された要約表で、即時最新表示をオンにすることを指定します。

GENERATED COLUMN
生成された列をオンにすることを指定します。

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、検査保留状態ではなくなります)。あるいは、要約表の即時最新表示はオンにし、検査保留状態を解除します。

  これは、次のようにして特定の表に指定します。ALL を指定する。その表の検査制約だけがオフの場合には CHECK を指定する。その表の外部キー制約だけがオフの場合には FOREIGN KEY を指定する。その表の DATALINK 保全性制約だけがオフの場合には DATALINK RECONCILE PENDING を指定する。その要約表の要約表照会検査だけがオフの場合には SUMMARY を指定する。そしてその表に対する列の生成だけがオフの場合には GENERATED COLUMN を指定する。

- 表の 1 つのタイプの保全性検査だけをオンにしますが、表は検査保留状態のままにします。

  これは、特定の表に対して、どのタイプの制約もオフの場合に、CHECK、FOREIGN KEY、SUMMARY、GENERATED COLUMN、または DATALINK RECONCILE PENDING だけを指定することによって指定されます。

状態の変更は、リストに明示的に含まれていない表には拡張されません。

従属表の親が検査保留状態である場合、従属表の外部キー制約の検査を回避することはできません (検査制約の検査は回避できます)。

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。 『注』 を参照してください。

注

- 検査保留状態の表への影響 :
  - SELECT、INSERT、UPDATE、または DELETE は、以下の状態の表に対しては実行できません。
    - 検査保留状態のその表自体
    - 検査保留状態の他の表へのアクセスが必要な表

    たとえば、検査保留状態にある従属表にカスケードする親表の行の DELETE は実行できません。
  - 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、表が検査保留状態の場合は、表が検査保留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。
  - CREATE INDEX ステートメントでは、検査保留状態にある表を参照できません。 同様に、基本キー制約または固有制約を追加する ALTER TABLE では、検査保留状態にある表を参照できません。
  - ユーティリティー EXPORT、IMPORT、REORG、および REORGCHK は、検査保留状態の表に対しては実行できません。 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。
  - ユーティリティー LOAD、BACKUP、RESTORE、ROLLFORWARD、UPDATE STATISTICS、RUNSTATS、LIST HISTORY、および ROLLFORWARD は、検査保留状態の表に対して実行可能です。
  - ステートメント ALTER TABLE、 COMMENT ON、 DROP TABLE、CREATE ALIAS、 CREATE TRIGGER、 CREATE VIEW、 GRANT、REVOKE、および SET INTEGRITY では、検査保留状態の表を参照することができます。
  - 検査保留状態の表に従属しているパッケージ、視点、およびその他のオブジェクトは、実行時にその表がアクセスされると、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。

- 増分処理がデフォルトの動作であるため、INCREMENTAL オプションは多くの場合必要ありません。しかし、このオプションが必要になるケースが 2 つあります。
  - IMMEDIATE UNCHECKED オプションを使って、以前に検査保留状態から解かれた表に対して増分処理を行うため。デフォルトでは、すべてのデータの保全性を検査するため、システムは完全処理を選択します。このデフォルトの動作は、新しく追加した部分だけを検査するために、INCREMENTAL オプションを指定して上書きすることができます。 (詳しくは、黒丸の ″IMMEDIATE UNCHECKED 文節の使用に関する警告″ を参照してください。)
  - 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。INCREMENTAL オプションを指定すると、データ保全性を保証するには完全処理が必要だとシステムが判断したときに、システムはエラー (SQLSTATE 55019) を戻します。
- 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。

    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 視点の 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、以下のいずれかの時点までそのままになります。

    - 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、表が検査保留状態に戻る。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みなされていて、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除去される。
    - 要約表用に REFRESH TABLE ステートメントが発行される。

    '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事実を記録します。また、選択が与えられると、システムは表全体のデータ保全性を再検査し、この状態を 'Y' 状態に変更します。 選択が与えられない場合 (たとえば、 IMMEDIATE UNCHECKED または INCREMENTAL を指定すると)、システムによって

検査されていないデータがいくつかあることを記録するため、この状態を 'U' 状態に戻します。 後者 (INCREMENTAL) の場合、警告 (SQLSTATE 01636) が戻されます。

- Load Insert を使っ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て、その表の検査保留状態を解きます。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。増分処理ができない場合、システムは表全体の保全性違反を検査します (システムが完全処理を好む状況については、以下をご覧ください)。
- SET INTEGRITY for T IMMEDIATE CHECKED ステートメントに INCREMENTAL オプションを指定しなかった場合、システムが表全体の保全性検査を行う状況は、以下のとおりです。
  1. 表 T の SYSCAT.TABLES カタログの CONST_CHECKED 列に、 1 つまたは複数の 'W' 値がある場合。
- SET INTEGRITY for T IMMEDIATE CHECKED ステートメントに対して表全体の保全性 (INCREMENTAL オプションは指定できない) をシステムが検査しなければならない状況は、以下のとおりです。
  1. T そのものか、検査保留状態にある T の親のどれかに新しい制約が追加された場合。
  2. T に対する最後の保全性検査の後、Load Replace が T に行われた場合、あるいは NOT LOGGED INITIALLY WITH EMPTY TABLE オプションが活動化された場合。
  3. (完全処理の影響により) T の親のどれかに対して Load Replaced が行われた場合、または非増分の保全性検査が行われた場合。
  4. 移行の前に表が検査保留状態にあったため、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。
  5. 表スペースに含まれている表やその親が特定の時点までロールフォワードされている場合。
- 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。 表をデータ・リンク調整不能状態から解放することの詳細については、 *管理の手引き* を参照してください。
- 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 呼び出しにおいてリスト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。
- ログ・スペースが不足しており、これを十分に増やせないために FORCE GENERATED 文節を含む SET INTEGRITY ステートメントが失敗する場合は、 **db2gncol** コマンドで断続的なコミットを使用して値を生成することができます。その後、FORCE GENERATED 文節を省略して SET INTEGRITY を再実行します。

例

例 1: 以下は、表の検査保留状態に関する情報を入手する照会の例です。 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 番目バイトは検査制約を表します。

```
SELECT TABNAME,
  SUBSTR( CONST_CHECKED, 1, 1 ) AS FK_CHECKED,
  SUBSTR( CONST_CHECKED, 2, 1 ) AS CC_CHECKED
  FROM SYSCAT.TABLES
  WHERE STATUS = 'C'
```

例 2: 表 T1 と T2 を検査保留状態に設定します。

```
SET INTEGRITY FOR T1, T2 OFF
```

例 3: 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 4: 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 5: T1 の FOREIGN KEY 制約検査を使用可能にし、 T2 の CHECK 制約検査を IMMEDIATE CHECKED 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 6: 2 つの ALTER TABLE ステートメントを使用して、検査制約と外部キーを EMP_ACT 表に追加します。 表を単一パスで制約の検査を実行するには、保全性検査をオフにして ALTER ステートメントを実行し、その実行後に制約検査を行います。

```
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 7: 生成された列に保全性を設定します。

```
SET INTEGRITY FOR T1 IMMEDIATE CHECKED
   FORCE GENERATED
```

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

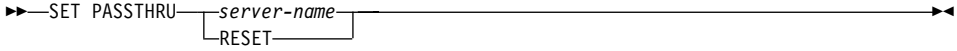
このステートメントは、対話式に発行することができます。このステートメントは、動的に準備可能な実行可能ステートメントです。

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・ソースでの機密保護の制限を満たす。

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* は、カタログに記述され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

パススルーを使用する際の指針および制約事項については、 1364ページの『パススルー機能の処理』を参照してください。

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・セッションを終了します。

```
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。

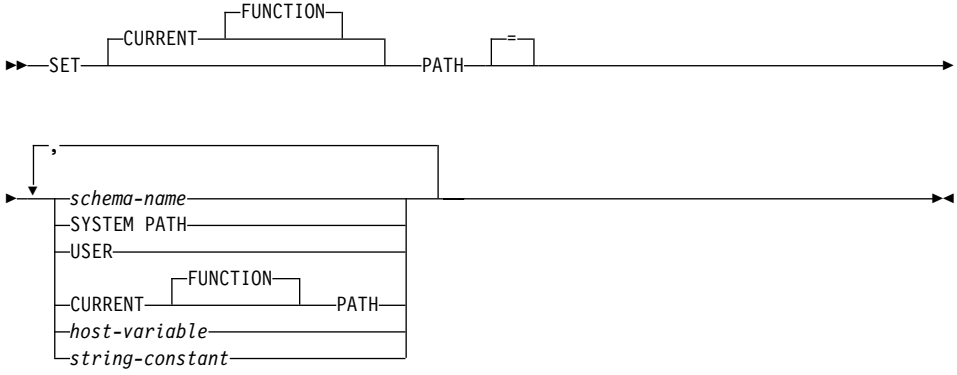
呼び出し

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、動的に準備可能な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。 *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。

SYSTEM PATH
この値の指定は、スキーマ名として *"SYSIBM"*、 *"SYSFUN"* 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 の値。 CURRENT FUNCTION PATH も指定できます。

host-variable
CHAR または VARCHAR のタイプの変数です。 *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
8 文字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

- CURRENT PATH 特殊レジスターの初期値は、 ″SYSIBM″、″SYSFUN″、″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みなされます (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 バ

インド・オプションによって指定されます。 BIND コマンドの FUNCPATH オプションの使用に関する詳細は、*コマンド解説書* を参照してください。

例

例 1: 以下のステートメントは、CURRENT FUNCTION 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何も行いません。

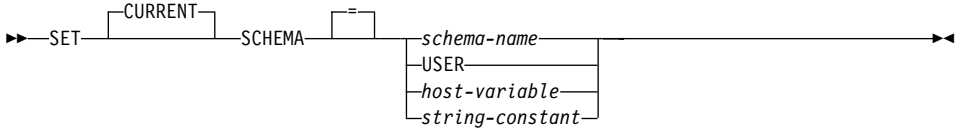
呼び出し

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、動的に準備可能な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。

host-variable

CHAR または VARCHAR のタイプの変数です。 *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 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 バインド・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 (QUALIFIER オプションの使用に関する詳細は、 *コマンド解説書* を参照してください)。

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連合データベースに接続中にも有効な、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。

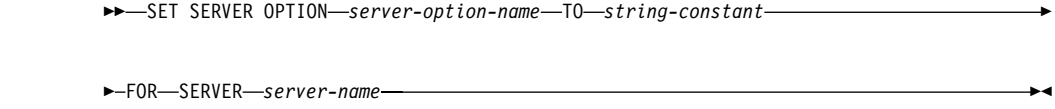
呼び出し

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、動的に準備可能な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、連合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。サーバー・オプションに関する詳細については、 1355ページの『サーバー・オプション』を参照してください。

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。 可能な設定値に関する詳細については、1355ページの『サーバー・オプション』を参照してください。

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。

- 現在のところ、SET SERVER OPTION では、 password、fold_id、および fold_pw サーバー・オプションしかサポートされていません。
- ユーザーまたはアプリケーションが連合データベースに接続する際、 1 つまたは複数の SET SERVER OPTION ステートメントを発信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。

例

例 1: DJDB という連合データベースに、 RATCHIT という Oracle データ・ソースを定義します。 RATCHIT は、計画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 ratchit");
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・ソースでの認証パスワード) に設定しました。しかし、特定の Oracle 8 データ・ソース (連合データベース 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連合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;
```

SET transition-variable

SET transition-variable ステートメントは、新しい変換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、細分性が FOR EACH ROW の BEFORE トリガーのトリガー・アクションで、トリガーにより実行される SQL ステートメントとしてのみ使用することができます (846ページの『CREATE TRIGGER』 を参照してください)。

許可

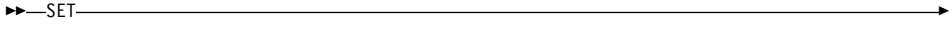
トリガーの作成者の許可 ID の特権には、以下の特権の少なくとも 1 つが含まれていなければなりません。

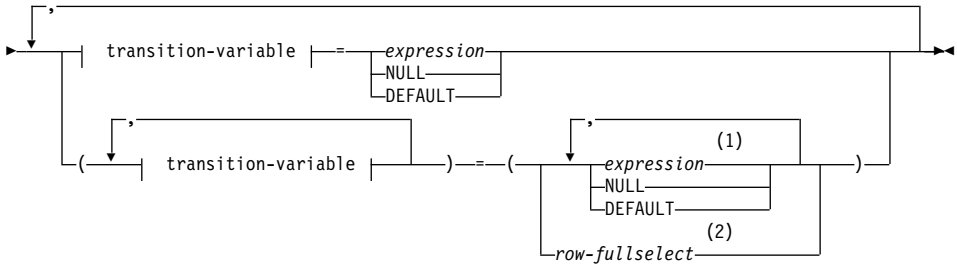
- 割り当て式の左辺で参照される列に対する UPDATE、および右辺で参照される列に対する SELECT
- 表 (トリガーの対象表) に対する CONTROL 特権
- SYSADM または DBADM 権限

割り当ての右辺に *row-fullselect* (行全選択) の指定があるこのステートメントを実行する場合には、トリガー作成者の許可 ID に与えられている特権には、参照される表または視点のそれぞれに対する特権として、以下の少なくとも 1 つが含まれていなければなりません。

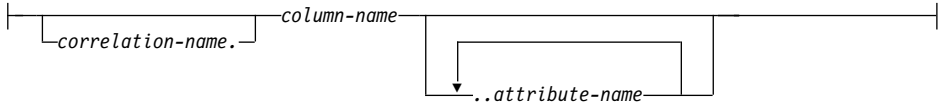
- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM

構文





transition-variable:



注:

1 式、NULL、および DEFAULT の数は、 *transition-variable* (変換変数) の数と同じでなければなりません。

2 選択リストの列の数は、 *transition-variable* の数と同じでなければなりません。

説明

transition-variable

トリガーの影響を受ける一連の行の列を指定します。

correlation-name

新しい (NEW) 変換変数を参照するための*相関名*。この*相関名* は、CREATE TRIGGER の REFERENCING 文節の NEW の後に指定した相関名と同じでなければなりません。

その REFERENCING 文節に OLD が指定されていない場合、*相関名* のデフォルト値は NEW の後に指定した*相関名* になります。REFERENCING 文節に NEW と OLD の両方が指定されている場合は、各*列名* ごとに*相関名* の指定が必要になります (SQLSTATE 42702)。

column-name

更新する列を指定します。 *column-name* (列名) は、トリガーの対象表の列を指定していなければなりません (SQLSTATE 42703)。 1 つの列を 2 回以上指定することはできません (SQLSTATE 42701)。

..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、通常の割り当てとみなされます。

expression
列の新しい値を指定します。この expression (式) として、 176ページの『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。スカラー全選択に現れる式を除き、この式に列関数を含めることはできません (SQLSTATE 42903)。 *expression* には、 OLD および NEW の変換変数への参照を含めることができますが、どの変換変数かを指定するために、 *correlation-name* (相関名) で修飾する必要があります (SQLSTATE 42702)。

NULL
ヌル値を指定します。これはヌル値可能な列に対して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。

- 列が WITH DEFAULT 文節を指定して定義されている場合、値はその列に対して定義されたデフォルト値に設定されます (507ページの『ALTER TABLE』 の default-clause を参照してください)。
- IDENTITY 文節を使用して列が定義された場合は、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、IDENTITY 文節、および NOT NULL 文節のいずれも指定せずに列が定義された場合、値は NULL になります。
- 列の定義に NOT NULL 文節が使用されたが IDENTITY 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)。

row-fullselect
割り当て式に指定した列名 の数に対応する数の列を含む 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。 この行

全選択 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、 OLD および NEW の変換変数に対する参照を含めることができます。そのような場合、使用する変換変数を*相関名*で修飾して指定する必要があります (SQLSTATE 42702)。 結果に複数の行が含まれる場合には、エラーになります (SQLSTATE 21000)。

規則

- 式、NULL、DEFAULT、または *row-fullselect* から割り当てられる値の数は、割り当て式に指定した列の数と同じでなければなりません (SQLSTATE 42802)。
- BEFORE UPDATE トリガーでこのステートメントを使用する場合、 *transition-variable* (変換変数) として指定する *column-name* (列名) は 区分化キーの列であってはなりません (SQLSTATE 42997)。

注

- 複数の割り当て式が含まれている場合には、割り当ての実行に先立って、すべての *expression* と *row-fullselect* が評価されます。 したがって、式または行全選択の中の列に対する参照は、その単一の SET transition-variable ステートメントでの割り当て以前の変換変数の値になります。
- 特殊タイプとして定義された識別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。そして計算された値は、値が列に実際に割り当てられるときに、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。[106]
- 識別列に対する 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 が識別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。
- 識別列に生成されるシーケンス値の使用に関する詳細は、 1025ページの『INSERT』を参照してください。
- 識別列で値が最大値を超えた場合の詳細は、 1025ページの『INSERT』を参照してください。

106. 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。

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、更新される行の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、および元の給与値 (つまり SET ステートメントの実行前の) 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);
```

SIGNAL SQLSTATE

SIGNAL SQLSTATE ステートメントは、エラーをシグナルするのに使用されます。指定した SQLSTATE と、指定した *diagnostic-string* (診断ストリング) で戻されるエラーを生じさせます。

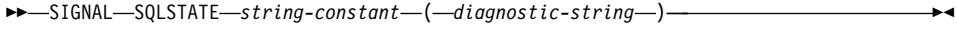
呼び出し

SIGNAL SQLSTATE ステートメントは、トリガー内の起動された SQL ステートメントとしてのみ使用できます。

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

string-constant

指定する *string-constant* (ストリング定数) は、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、以下のアプリケーション定義の SQLSTATE の規則に基づい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'、'01'、または '02' であってはなりません (これらの値はエラー・クラスではないので)。
- SQLSTATE クラス (最初の 2 文字) が文字 '0'～'6' または '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 (最初の 2 文字) が文字 '7'、'8'、'9'、または 'I'～'Z' で始まっている場合、サブクラス (最後の 3 文字) として '0'～'9' または 'A'～'Z' のいずれでも使用することができます。

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)。

diagnostic-string
エラー条件について説明する最高 70 バイトの文字ストリングを戻すタイプ CHAR または VARCHAR の式です。ストリングが 70 バイトを超える場合には切り捨てられます。

例

PARTS 表に十分な在庫がある場合にのみ、ORDERS 表 (ORDERNO、CUSTNO、 PARTNO、 QUANTITY) に受注を記録する受注システムを想定します。

```
CREATE TRIGGER check_avail
      NO CASCADE BEFORE INSERT ON orders
      REFERENCING NEW AS new_order
      FOR EACH ROW MODE DB2SQL
      WHEN (new_order.quantity > (SELECT on_hand FROM parts
                                  WHERE new_order.partno=parts.partno))
        BEGIN ATOMIC
          SIGNAL SQLSTATE '75001' ('Insufficient stock for order');
        END
```

UPDATE

UPDATE ステートメントは、表または視点の行の指定された列の値を更新します。 視点の行の更新により、その基礎表の行が更新されます。

このステートメントの形式は以下のとおりです。

- *探索条件付き* UPDATE 形式は、 1 つまたは複数の行 (任意指定の探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か、あるいは動的 SQL ステートメントの使用によって発行することができます。このステートメントは、動的に準備可能な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、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。

- 更新する行を含む表または視点に対する UPDATE 特権
- 更新するそれぞれの行に対する UPDATE 特権
- その行を更新する表または視点に対する 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、参照される表または視点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または視点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

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[107] 、UPDATE の探索条件付き形式で *assignment-clause* の右側または *search-condition* のいずれかの個所に表または視点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。

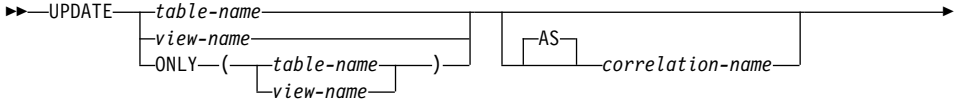
- SELECT 特権
- CONTROL 特権
- SYSADM または DBADM 権限

指定した表または視点が 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、指定した表または視点の副表または副視点ごとに SELECT 特権が含まれている必要があります。

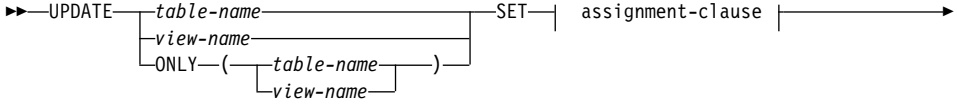
静的 UPDATE ステートメントの場合、GROUP 特権は検査されません。

構文

Searched (探索条件付き) UPDATE:

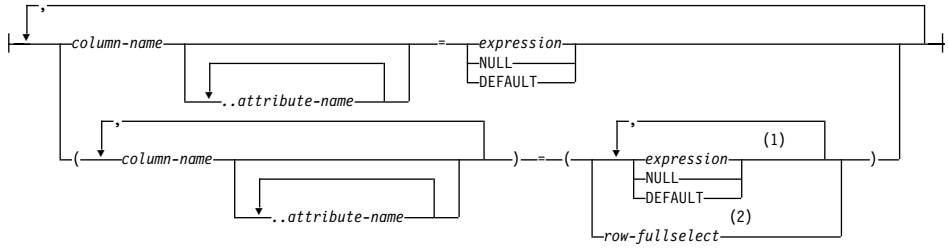


Positioned (位置指定) UPDATE:



assignment-clause:

107. ステートメントの処理に使用されるパッケージは、値 SQL92E または MIA を指定したオプション LANGLEVEL を使用してプリコンパイルされます。



注:

1. *expression*、NULL、および DEFAULT の数は、 *column-name* の数と同じでなければなりません。
2. 選択リスト中の列の数は、 *column-name* の数と同じでなければなりません。

説明

table-name または *view-name*
更新する表または視点の名前です。この名前は、カタログに記述されている表または視点を指定する名前でなければならず、カタログ表、カタログ表の視点 (更新可能な SYSSTAT 視点を除く)、要約表、読み取り専用の視点、またはニックネームを指定することはできません。 (読み取り専用の視点については、895ページの『CREATE VIEW』 を参照してください。更新可能なカタログ視点については、1231ページの『付録D. カタログ視点』を参照してください。)

table-name がタイプ付き表である場合は、このステートメントを使用すれば、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 節に設定または参照できるのは、指定された表の列だけです。 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または視点を、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。

ONLY (*table-name***)**
タイプ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をステートメントが適用し、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 を使用して指定しなければなりません。 *table-name* がタイプ付き表でない場合は、このステートメントに ONLY を使用しても効果はありません。

ONLY (*view-name*)

タイプ付き視点の場合に適用できます。 ONLY キーワードは、指定された視点のデータだけをステートメントが適用し、その表に関係する副視点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視点を、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* がタイプ付き視点でない場合は、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。

AS

correlation-name (相関名) の前に任意に指定できるキーワードです。

correlation-name

探索条件 (*search-condition*) の中で、表または視点を指定するのに使用することができます。 (相関名については、144ページの『相関名』 を参照してください。)

SET

この後に、列名への値の割り当て (assignment-clause) を指定します。

assignment-clause

column-name

更新したい列を指定します。 *column-name* (列名) は、指定した表または視点の更新可能な列を指定していなければなりません。[108] タイプ付き表のオブジェクト識別子列は更新できません (SQLSTATE 428DZ)。属性名 (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)。

位置指定 UPDATE の場合 :

- カーソルの選択ステートメントに UPDATE 文節を指定した場合、この *assignment-clause* の各列名は、その UPDATE 文節にも指定されていなければなりません。
- カーソルの選択ステートメントに UPDATE 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 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。

108. 区分化キーの列は更新できません (SQLSTATE 42997)。区分化キーの列を変更するには、そのデータの行を削除して挿入し直す必要があります。

- カーソルの選択ステートメントに UPDATE 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* とみなされます。

expression

列の新しい値を指定します。この expression (式) として、 176ページの『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。 スカラー全選択に現れる式を除き、この式に列関数を含めることはできません (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。

NULL

ヌル値を指定します。これはヌル値可能な列に対して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。
- IDENTITY 文節を使用して列が定義された場合は、データベース・マネージャーによって値が生成されます。
- 列が WITH DEFAULT 文節を指定して定義されている場合、値はその列に対して定義されたデフォルト値に設定されます (507ページの『ALTER TABLE』 の default-clause を参照してください)。

- WITH DEFAULT 文節、GENERATED 文節、および NOT NULL 文節のいずれも指定せずに列が定義された場合は、値 NULL が使用されます。
- 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)。

row-fullselect

割り当て式に指定した*列名* の数に対応する数の列を含む 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*行全選択* 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。 結果に複数の行が含まれる場合には、エラーになります (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、探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、表または視点のすべての行が更新されます。

search-condition

71ページの『第3章 言語要素』 の説明に基づいて、探索条件を指定します。 副照会以外の探索条件の各*列名* は、表または視点の列を指定していなければなりません。 探索条件に、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。

探索条件は、表または視点の各行に適用され、探索条件の結果が「真」の行が更新されます。

探索条件に副照会が含まれる場合、その副照会は、探索条件が 1 つの行に適用されるたびに実行され、その結果は探索条件の適用に使用されるものとみなされます。 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。 914ページの『DECLARE CURSOR』 で説明されているように、 *cursor-name* (カーソル名) は、宣言されたカーソルを識別してい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。

指定する表または視点は、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、914ページの『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。

視点を定義する全選択の選択リストの OLAP 関数を含む視点を更新する場合は、この形式の UPDATE を使用することはできません (SQLSTATE 42828)。

規則

- ***割り当て:*** 更新値は、『第3章 言語要素』で説明されている割り当て規則に従って列に割り当てられます。
- ***妥当性:*** 更新される行は、更新される列の固有索引がある場合には、それによってその表 (または視点の基礎表) に適用される制約に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視点が使用される場合、行が変更され、その結果、それらの行がその視点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、視点の基礎表で更新され、その視点には現われなくなります。

  WITH CHECK OPTION を用いて定義された視点を使用する場合、更新された行は、その視点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、895ページの『CREATE VIEW』 を参照してください。
- ***検査制約:*** 更新値は、表に定義されている検査制約の検査条件を満たしていなければなりません。

検査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検査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、親の固有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、親の固有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、関連する親表の基本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。
- UPDATE ステートメントの実行が完了すると、 SQLCA の SQLERRD(3) の値は更新された行の数を示します。 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。 SQLCA については、1211ページの『付録B. SQL 連絡 (SQLCA)』 を参照してください。
- 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 ステートメントを実行によって、 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)。 ロックについては、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。

  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 値を更新してヌルにすることは、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)。

- タイプ付き表の列分散統計を更新する場合は、列を最初に参照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識別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。そして計算された値は、値が列に実際に割り当てられるときに、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。[109]
- 識別列に対する 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 が識別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。
- 識別列に生成されるシーケンス値の使用に関する詳細は、 1025ページの『INSERT』を参照してください。
- 識別列で値が最大値を超えた場合の詳細は、 1025ページの『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'
  ```

109. 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。

- *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 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
```

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。詳細については、461ページの『全選択』 を参照してください。

VALUES INTO

VALUES INTO ステートメントは、多くても 1 つの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。

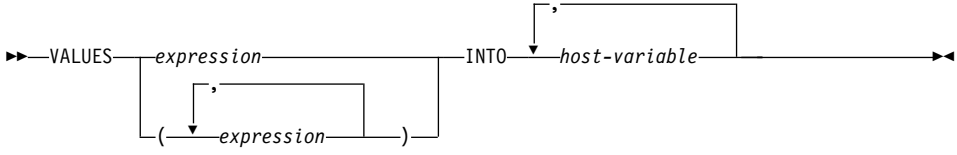
呼び出し

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のみ可能です。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

権限は不要です。

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 (1211ページの『付録B. SQL 連絡 (SQLCA)』 を参照)。

各変数には、106ページの『割り当てと比較』で説明されている規則に基づいて値が割り当てられます。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;
```

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。

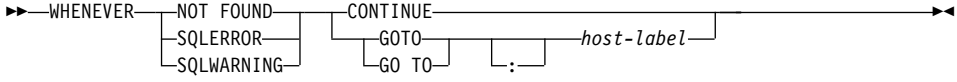
呼び出し

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のみ可能です。 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

権限は不要です。

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 ステートメントの効力範囲内にあります。WHENEVER ステートメントの効力範囲は、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 ステートメントの効力範囲内にあります。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 ステートメントの効力範囲内にあります。

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;
```

第7章 SQL プロシージャー

SQL プロシージャーは、 CREATE PROCEDURE ステートメントとプロシージャー本体から成り立っています。

この章では、 SQL プロシージャー・ステートメント内のプロシージャー本体の構文およびパラメーターについて説明しています。

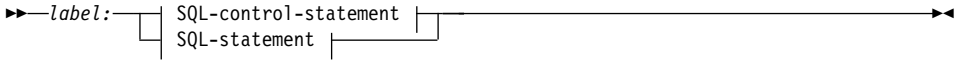
SQL プロシージャー・ステートメント

SQL ストアード・プロシージャー定義内のプロシージャー本体には、そのストアード・プロシージャーのソース・ステートメントが含まれます。

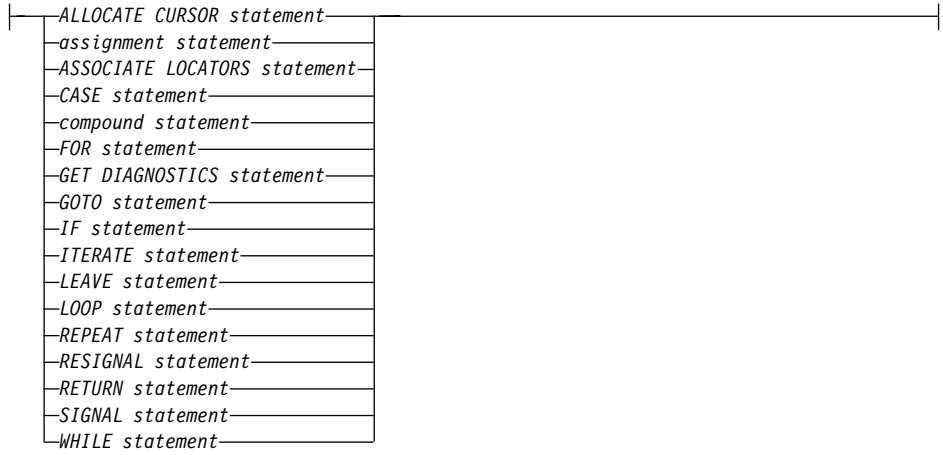
この章では、プロシージャー本体を構成するステートメントの構文図、意味の説明、規則、および使用例を示しています。

SQL 制御ステートメントを SQL プロシージャー本体として指定すれば、制御ステートメント内に複数のステートメントを指定することができます。これらのステートメントは、SQL プロシージャー・ステートメントとして定義されます。

構文



SQL-control-statement:



説明

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。このラベルは、 SQL プロシージャー・ステートメント (リスト内でネストされているあらゆる複合ステートメントも含まれる) のリスト内で固有でなければなりません。ネストされていない複合ステートメントでは、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、複数の SQL 制御ステートメントで使用できます。

SQL-statement

SQL プロシージャー本体には、実行可能なすべての SQL ステートメントを含めることができます。ただし、以下のものは含めることができません。

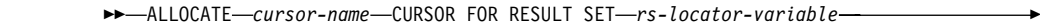
- CONNECT
- CREATE (索引、表、または視点以外のあらゆるオブジェクトの)
- DESCRIBE
- DISCONNECT
- DROP (索引、表、または視点以外のあらゆるオブジェクトの)
- FLUSH EVENT MONITOR
- REFRESH TABLE
- RELEASE (接続のみ)
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- SET CONNECTION
- SET INTEGRITY

注: CALL ステートメントは SQL プロシージャー本体に含めることができますが、それらの CALL ステートメントで呼び出せるのは、他の SQL プロシージャーだけです。 SQL プロシージャー内の CALL ステートメントでは、他のタイプのストアード・プロシージャーを呼び出すことはできません。

ALLOCATE CURSOR ステートメント

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、1164ページの『ASSOCIATE LOCATORS ステートメント』を参照してください。

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・ロケーター変数を指定します。 SQL 変数の宣言について詳しくは、1171ページの『SQL 変数宣言』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)。

注

- **動的に準備された ALLOCATE CURSOR ステートメント**: 動的に準備された ALLOCATE CURSOR ステートメントを実行するには、 USING 文節を使った EXECUTE ステートメントを使用しなければなりません。動的に準備されたステートメントでは、ホスト変数への参照はパラメーター・マーカー (疑問符) で示されます。

  ALLOCATE CURSOR ステートメントでは、*rs-locator-variable* は常にホスト変数です。そのため、動的に準備された ALLOCATE CURSOR ステートメントの場合、 EXECUTE ステートメントの USING 文節では、*rs-locator-variable* を表すパラメーター・マーカーの代わりとして使用される値のホスト変数が識別されなければなりません。
- ALLOCATE CURSOR ステートメントを準備する際に、 DECLARE CURSOR ステートメントですでに使用されているステートメント識別子を指定することはできません。たとえば、以下の SQL ステートメントは無効に

なります。これは、 PREPARE ステートメントで ALLOCATE CURSOR ステートメントの識別子として STMT1 が使用されているものの、 STMT1 は DECLARE CURSOR ステートメントですでに使用されているためです。

```
DECLARE CURSOR C1 FOR STMT1;

PREPARE STMT1 FROM
   'ALLOCATE C2 CURSOR FOR RESULT SET ?';
```

規則

- 割り当てられたカーソルを使用する際には、以下の規則が適用されます。
  - 割り当てられたカーソルは、 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)。
  - 割り当てられたカーソルは、CLOSE ステートメントによってクローズできます。割り当てられたカーソルをクローズすると、ストアード・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。
  - 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。
- 割り当てられたカーソルは、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、割り当てられたカーソルで、ストアード・プロシージャーによって WITH HOLD が定義されていないものが破棄されます。
- 割り当てられた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

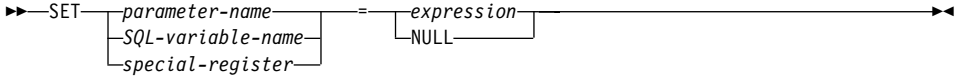
以下の SQL プロシージャーの例では、カーソル C1 を定義し、結果セット・ロケーター変数 LOC1、および SQL プロシージャーによって戻される関連する結果セットに関連付けます。

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

割り当てステートメント

割り当てステートメントは、出力パラメーター、ローカル変数、または特殊レジスターに値を割り当てます。

構文



説明

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-variable-name
割り当てのターゲットとなる SQL 変数を識別します。 SQL 変数は、使用する前に定義しなければなりません。 SQL 変数は複合ステートメントで定義できます。

special-register
割り当てのターゲットとなる特殊レジスターを識別します。特殊レジスターの値としてスキーマ名を使用する場合 (CURRENT FUNCTION PATH または CURRENT SCHEMA 特殊レジスターなど)、割り当てパラメーターが SQL 変数かどうか、DB2 が判別します。割り当てパラメーターが SQL 変数であれば、DB2 はその特殊レジスターに SQL 変数の値を割り当てます。割り当てパラメーターが SQL 変数でなければ、DB2 はその割り当てパラメーターをスキーマ名とみなし、その特殊レジスターにスキーマ名を割り当てます。

SQL プロシージャーの特殊レジスター値の初期設定は、そのプロシージャーの呼び出し側から継承します。新しい設定を割り当てると、SQL プロシージャー全体のその値が設定されている部分で有効になり、以降の呼び出しの際にもプロシージャーに継承されます。プロシージャーが呼び出し側に戻されると、特殊レジスターは呼び出し側の元の設定に復元されます。

expression **または NULL**
割り当てのソースとなる式または値を指定します。

規則

- 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。
- ターゲットとソースのデータ・タイプは互換性がなければなりません。
- ストリングが固定長変数に割り当てられる際に、ストリングの長さがターゲットの長さ属性よりも短い場合、ストリングの右端に必要な数の 1 バイト、2 バイト、または UCS-2 のブランクが埋め込まれます。
- ストリングが変数に割り当てられる際に、ストリングが変数の長さ属性より長い場合、エラーが出されます。
- 変数に割り当てられるストリングは、必要なら、まずターゲットのコード・ページに変換されます。
- 数値変数の割り当てで、数値の部分すべてが切り捨てられると、エラーが発生します。

例

SQL 変数 p_salary を 10 % ずつ増加させます。

```
SET p_salary = p_salary + (p_salary * .10)
```

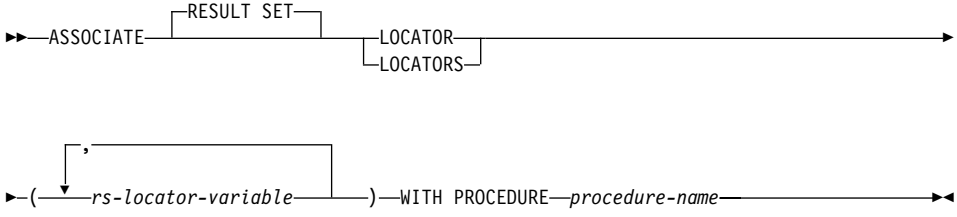
SQL 変数 p_salary をヌル値に設定します。

```
SET p_salary = NULL
```

ASSOCIATE LOCATORS ステートメント

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。

構文



説明

rs-locator-variable
複合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識別子です。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識別子です。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修飾子にできます。

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、先回実行した 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;
```

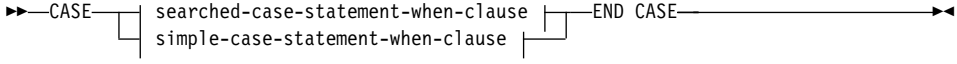
例 2: 例 1 のシナリオを繰り返しますが、スキーマ MYSCHEMA で確実にストアード・プロシージャー P1 が使用されるように、 2 つの部分からなる名前を使用し、ストアード・プロシージャーの明示的なスキーマ名を指定します。

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

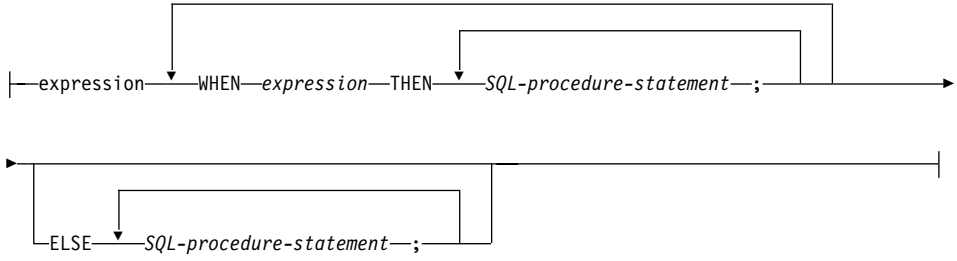
CASE ステートメント

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。

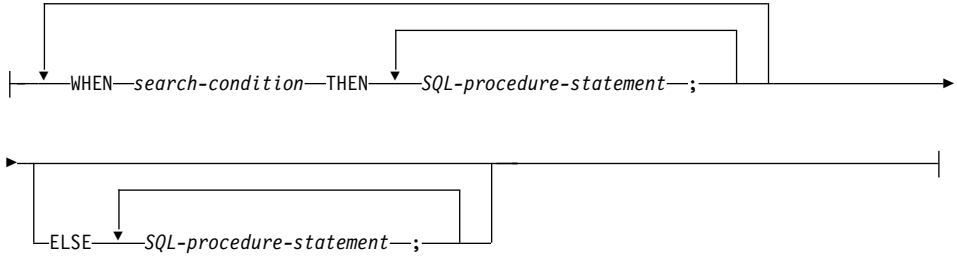
構文



simple-case-statement-when-clause:



searched-case-statement-when-clause:



説明

CASE

case-expression 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。結果が不定または偽の場合、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。偽または不定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。

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)。
- ご使用の CASE ステートメントでは、考えられるあらゆる実行条件を必ず網羅してください。

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。

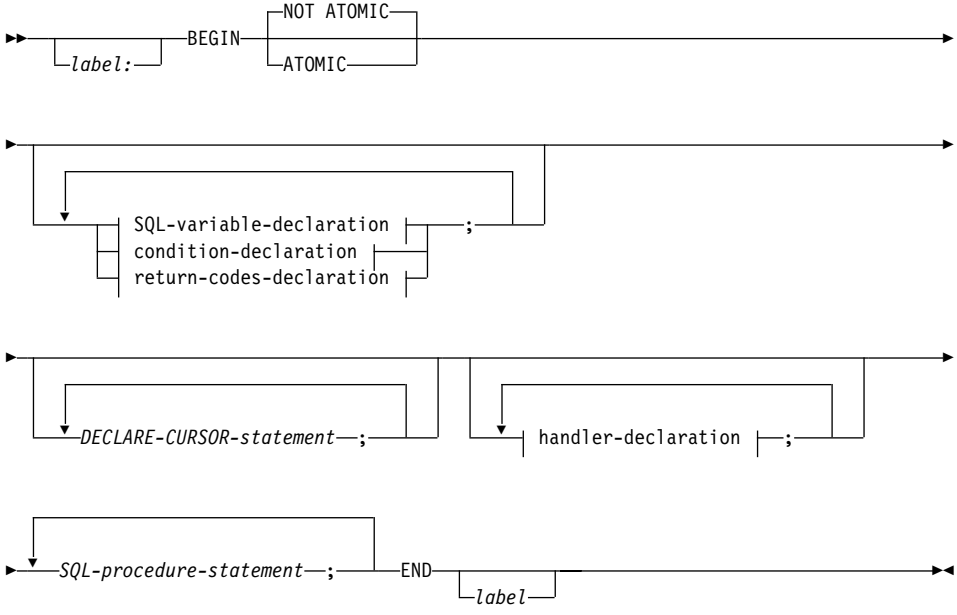
```
CASE
   WHEN v_workdept = 'A00'
     THEN UPDATE department
     SET deptname = 'DATA ACCESS 1';
   WHEN v_workdept = 'B01'
```

```
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 2';
  ELSE UPDATE department
    SET deptname = 'DATA ACCESS 3';
END CASE
```

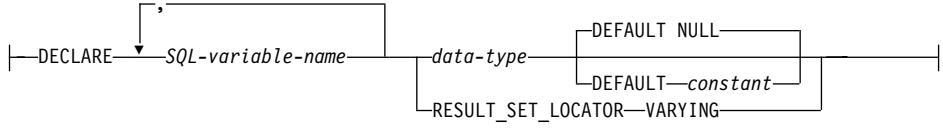
複合ステートメント

複合ステートメントは、SQL プロシージャー内の他のステートメントを 1 つにグループ化します。複合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。

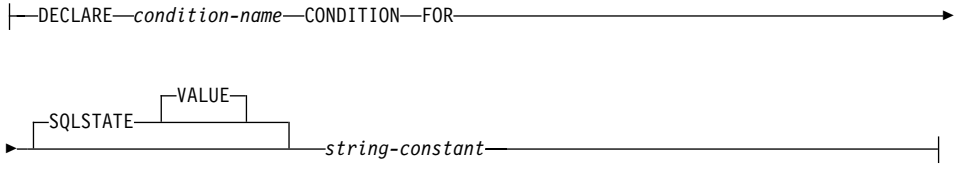
構文



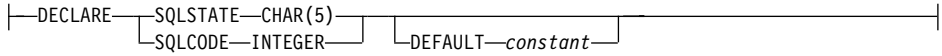
SQL-variable-declaration:



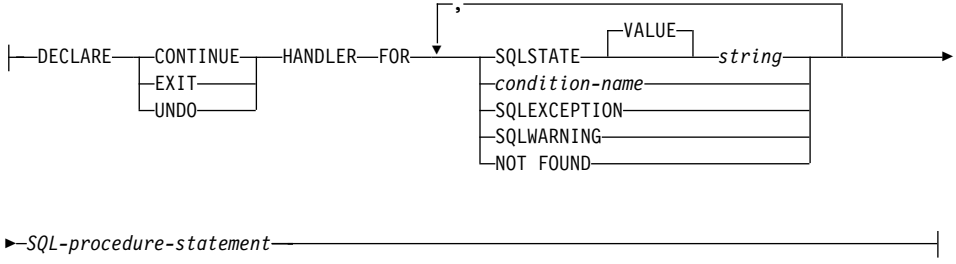
condition-declaration:



return-codes-declaration:



handler-declaration:



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、複合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、開始ラベルと同じにしなければなりません。

ATOMIC または NOT ATOMIC

ATOMIC は、複合ステートメントでエラーが発生したときに、その複合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、複合ステートメントでエラーが発生しても、その複合ステートメントをロールバックしません。

SQL-variable-declaration

複合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。 SQL 変数は DB2 によって大文字に変換されます。この名前は、同じ複合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 ステートメントに SQL 変数および列参照と同じ名前の識別子が含まれている場合、 DB2 はその識別子を列と解釈します。

data-type

変数のデータ・タイプを指定します。 SQL データ・タイプの詳細については、84ページの『データ・タイプ』を参照してください。ユーザー定義データ・タイプ、漢字タイプ、および FOR BIT DATA はサポートされていません。

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、この変数は 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の中で固有でなければならず、参照できるのは、その条件名が宣言されている複合ステートメントの中だけです。

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。

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複合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 1 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。カーソルごとに固有の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、複合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)、オープン・データベース・コネクティビティー (ODBC)、Java データベース・コネクティビティー (JDBC)、 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、914ページの『DECLARE CURSOR』を参照してください。

handler-declaration

ハンドラー (つまり、複合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ステートメントの集合) 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。

ハンドラーが有効なのは、それが宣言されている複合ステートメントの中だけです。

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複合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、複合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複合ステートメントの最後に制御が戻されます。 UNDO を指定した場合、ATOMIC を指定しなければなりません。

ハンドラーが有効になる条件を以下に示します。

SQLSTATE *string*
ハンドラーが呼び出される SQLSTATE を指定します。 SQLSTATE は '00000' にすることはできません。

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。

SQLEXCEPTION
SQLEXCEPTION 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。 SQLEXCEPTION は、最初の 2 文字が "00"、"01"、または "02" ではない SQLSTATE です。

SQLWARNING
SQLWARNING 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。 SQLWARNING は、最初の 2 文字が "01" である SQLSTATE です。

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 は、最初の 2 文字が "02" である SQLSTATE に対応しています。

規則

- ATOMIC 複合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。
  - SQLEXCEPTION、SQLWARNING、または NOT FOUND を含んでいるハンドラーの宣言には、追加の SQLSTATE または条件名を含めることはできません。
  - 同じ複合ステートメントの中のハンドラーの宣言には、重複条件を含めることはできません。
  - ハンドラーの宣言では、同一の条件コードまたは 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 値を表す条件名を含めることもできません。 SQLSTATE 値のリストおよび詳細については、*メッセージ解説書*を参照してください。

– 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。最も適したハンドラーとは、複合ステートメントで定義されているハンドラー (例外または完了条件で使用される) で、例外または完了条件が発生したステートメントの効力範囲に最も近いもののことです。発生した例外に使用するハンドラーがない場合、複合ステートメントの実行が終了します。

例

以下の処置を実行する複合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。

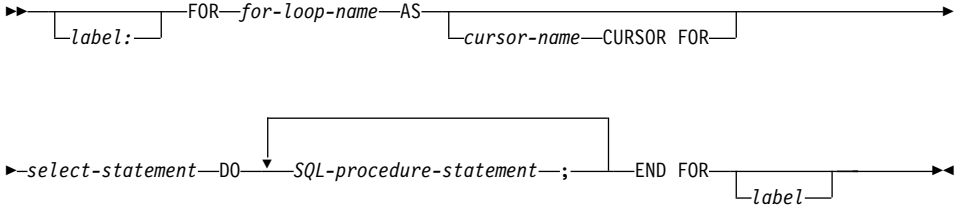
1. SQL 変数を宣言します。
2. IN パラメーターによって判別される部門の従業員の給与を戻すカーソルを宣言します。 SELECT ステートメントで、 *salary* 列のデータ・タイプを DECIMAL から DOUBLE にキャストします。
3. 条件 NOT FOUND (ファイル終わり) に EXIT ハンドラーを宣言します。これにより、値 '6666' が OUT パラメーター medianSalary に割り当てられます。
4. 指定された部門の従業員の数を SQL 変数 numRecords に選択します。
5. 50% + 1 の従業員が検索されるまで、WHILE ルー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
```

FOR ステートメント

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。

構文



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 ステートメントで使用することができます。終了ラベルを指定する場合、開始ラベルと同じにしなければなりません。

for-loop-name

生成された暗黙的な複合ステートメントのラベルを指定し、FOR ステートメントを実装します。これは、複合ステートメントのラベルに関する規則に従いますが、 FOR ステートメント内では ITERATE または LEAVE ステートメントを併用できないという点は除きます。 *for-loop-name* は、指定した *select-statement* から戻される列名を修飾するのに使用します。

cursor-name

SELECT ステートメントで生成される結果表から行を選択するのに使用するカーソルに、名前を指定します。指定されていない場合、DB2 が固有のカーソル名を生成します。

select-statement

カーソルの SELECT ステートメントを指定します。選択リスト内のすべての列に名前を付けなければなりません。また、複数の列に同じ名前を付けることはできません。

SQL-procedure-statement

表の行ごとに呼び出すステートメント (複数も可) を指定します。

規則

- 選択リストは固有の列名で構成されていなければなりません。また、選択リストで指定する表は、プロシージャーが作成される時点で実際に存在している表か、または以前の SQL プロシージャー・ステートメントで作成されている表でなければなりません。
- FOR ステートメントで指定されているカーソルは、その FOR ステートメントの外側で参照することはできません。また、OPEN、FETCH、または CLOSE ステートメントで指定することもできません。

例

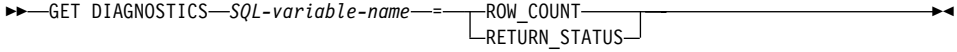
以下の例では、FOR ステートメントを使用して、employee 表全体を繰り返します。表の中の行ごとに、SQL 変数 fullname は、従業員のラストネーム (姓)、コンマ、ファーストネーム (名)、ブランク・スペース、そしてミドルネームのイニシャルという順序で設定されます。 fullname の各値は、表 tnames に挿入されます。

```
BEGIN
   DECLARE fullname CHAR(40);
   FOR v1 AS
      SELECT firstnme, midinit, lastname FROM employee
         DO
         SET fullname = lastname || ',' || firstnme ||' ' || midinit;
         INSERT INTO tnames VALUE (fullname);
   END FOR
END
```

GET DIAGNOSTICS ステートメント

GET DIAGNOSTICS ステートメントは、直前に呼び出された SQL ステートメントに関する情報を入手します。

構文



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。この変数は整変数でなければなりません。 SQL 変数は複合ステートメントで定義できます。

ROW_COUNT
直前に呼びされた 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 ステートメントの場合、 ROW_COUNT はそのステートメントによって、それぞれ削除、挿入、または更新された行数を戻します。その際に、トリガーまたは参照保全制約によって影響を受ける行は除外されます。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。

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。

規則

- 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;
-- At this point, rcount contains the number of rows that were updated.
    ...
       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 = RETURN_STATUS;
       IF RETVAL <> 0 THEN
          ...
          LEAVE A1;
       ELSE
          ...
       END IF;
    END A1
```

GOTO ステートメント

GOTO ステートメントは、SQL ルーチン内のユーザー定義ラベルに分岐させます。

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 ステートメントの効力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複合ステートメントは除きます。
- GOTO ステートメントが複合ステートメントで定義されている場合、*label* は同じ複合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複合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* は他の効力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効力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)。

規則

- 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

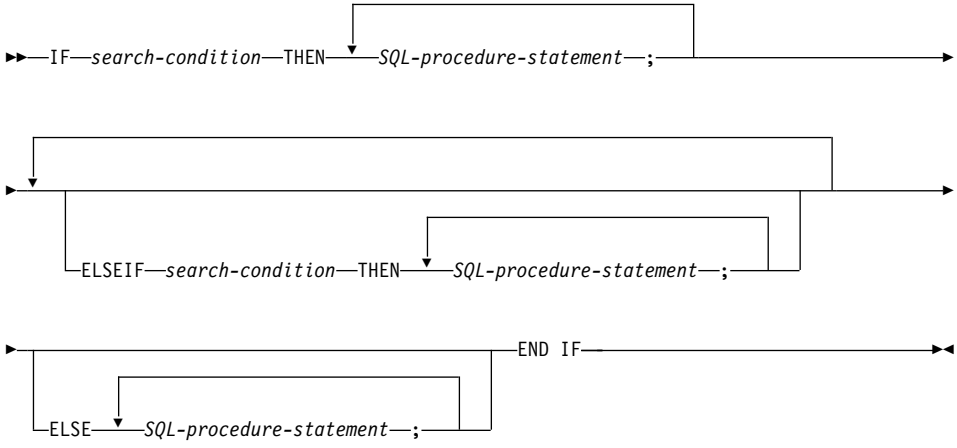
以下の複合ステートメントでは、パラメーター *rating* および *v_empno* がプロシージャーに渡されます。そして、日付期間として出力パラメーター *return_parm* が戻されます。従業員のその会社での就労期間が 6 か月未満の場合、 GOTO ステートメントは制御をプロシージャーの最後に移動させ、*new_salary* は未変更のままになります。

```
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
```

IF ステートメント

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。

構文



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。条件が不定または偽の場合、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。真と評価される search-condition がない場合、 ELSE キーワードの後に続く *SQL-procedure-statement* が呼び出されます。

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
```

ITERATE ステートメント

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。

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 ステートメント

LEAVE ステートメントは、プログラム制御をループまたは複合ステートメントの外側に移動させます。

構文

説明

label
終了する複合、FOR、LOOP、REPEAT、または WHILE ステートメントのラベルを指定します。

規則

LEAVE ステートメントが複合ステートメントの外側に制御を移動すると、その複合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。

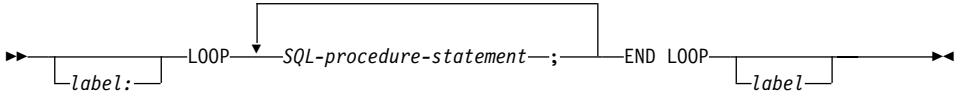
```
CREATE PROCEDURE LEAVE_LOOP(OUT counter INTEGER)
   LANGUAGE SQL
   BEGIN
      DECLARE v_counter INTEGER;
     DECLARE v_firstnme VARCHAR(12);
     DECLARE v_midinit CHAR(1);
     DECLARE v_lastname VARCHAR(15);
     DECLARE at_end SMALLINT DEFAULT 0;
     DECLARE not_found CONDITION FOR SQLSTATE '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
```

LOOP ステートメント

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。

構文



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ステートメントを指定します。

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。

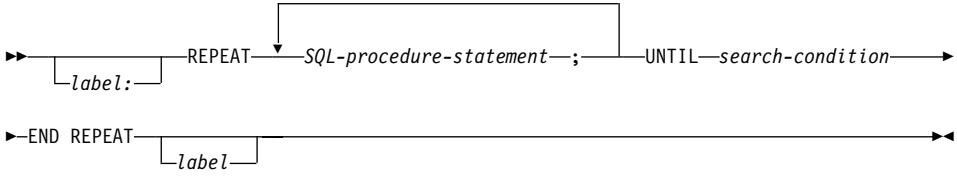
```
CREATE PROCEDURE LOOP_UNTIL_SPACE(OUT counter INTEGER)
   LANGUAGE SQL
   BEGIN
     DECLARE v_counter INTEGER DEFAULT 0;
     DECLARE v_firstnme VARCHAR(12);
     DECLARE v_midinit CHAR(1);
     DECLARE v_lastname VARCHAR(15);
     DECLARE c1 CURSOR FOR
         SELECT firstnme, midinit, lastname
           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
```

REPEAT ステートメント

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。

構文



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

ループで実行する SQL ステートメントを指定します。

search-condition

SQL プロシージャー・ステートメントが実行される前に評価される条件を指定します。条件が偽の場合、DB2 はループ内で SQL プロシージャー・ステートメントを実行します。

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。

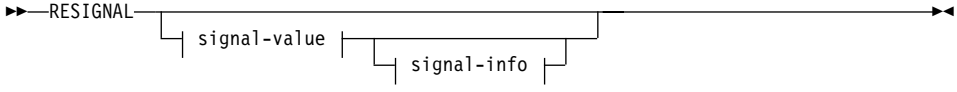
```
CREATE PROCEDURE REPEAT_STMT(OUT counter INTEGER)
   LANGUAGE SQL
   BEGIN
     DECLARE v_counter INTEGER DEFAULT 0;
     DECLARE v_firstnme VARCHAR(12);
     DECLARE v_midinit CHAR(1);
     DECLARE v_lastname VARCHAR(15);
     DECLARE at_end SMALLINT DEFAULT 0;
     DECLARE not_found CONDITION FOR 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
```

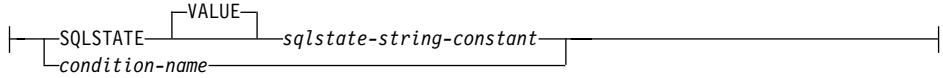
RESIGNAL ステートメント

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 と任意選択のメッセージ・テキストが、エラーまたは警告とともに戻されます。

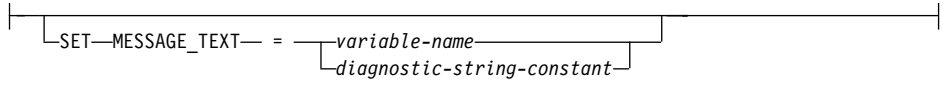
構文



signal-value:



signal-info:



説明

SQLSTATE VALUE *sqlstate-string-constant*
ここで指定したストリング定数は 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' であってはなりません (これは正常終了を表すため)。

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)。

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。このストリングは SQLCA の SQLERRMC フィールドに戻されます。実際のストリングが 70

バイトを超える場合、警告が出されずに切り捨てられます。この文節を指定する場合、 SQLSTATE または *condition-name* も指定しなければなりません (SQLSTATE 42601)。

variable-name
複合ステートメント内で宣言しなければならない SQL 変数を識別します。この SQL 変数は、CHAR または VARCHAR データ・タイプとして定義しなければなりません。

diagnostic-string-constant
メッセージ・テキストを含んでいる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発生した場合に、SQL ルーチンは呼び出し側に戻ります。
- RESIGNAL ステートメントが発行され、 SQLSTATE または *condition-name* を指定している場合、割り当てられる SQLCODE は以下のようになります。

  +438 (SQLSTATE が '01' または '02' で始まる場合)

  -438 (それ以外の場合)

  オプションを指定せずに RESIGNAL ステートメントを発行すると、SQLCODE にはハンドラーを呼び出した例外からの変更が行われません。
- 例外 ('01' または '02' 以外の SQLSTATE クラス) が通知されていることを、 SQLSTATE または条件が示している場合:
  - ハンドラーが、 RESIGNAL ステートメントがあるハンドラーを含んでいる複合ステートメントから見て、 1 つ外側の複合ステートメント (つまり前者の複合ステートメントのさらに外側の複合ステートメント) にあり、なおかつ、指定された SQLSTATE、condition-name、または SQLEXCEPTION のハンドラーが、その複合ステートメントに含まれていれば、例外が処理され、制御がそのハンドラーに移動します。
  - そうでなければ、例外は処理されず、制御が即時にその複合ステートメントの最後に戻されます。
- 警告 (SQLSTATE クラス '01') または NOT FOUND 条件 (SQLSTATE クラス '02') が通知されていることを、 SQLSTATE または条件が示している場合:
  - ハンドラーが、 RESIGNAL ステートメントがあるハンドラーを含んでいる複合ステートメントから見て、 1 つ外側の複合ステートメント (つま

り前者の複合ステートメントのさらに外側の複合ステートメント) にあり、なおかつ、指定されている SQLSTATE、condition-name、SQLWARNING (SQLSTATE クラスが '01' の場合)、または NOT FOUND (SQLSTATE クラスが '02' の場合) のハンドラーが、その複合ステートメントに含まれている場合、その警告または NOT FOUND 条件が処理され、制御がそのハンドラーに移動します。

– それ以外の場合、警告は処理されず、処理は次のステートメントに続行します。

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
         CONTAINS 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
```

RETURN ステートメント

RETURN ステートメントは、ルーチンから戻すのに使用します。 SQL 関数またはメソッドの場合、その関数またはメソッドの結果が戻されます。 SQL プロシージャーの場合、任意選択で整数状況値が戻されます。

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、 *expression* を指定しなければならず (SQLSTATE 42630)、また *expression* 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)。
- ルーチンがプロシージャーの場合、 *expression* のデータ・タイプは INTEGER でなければなりません (SQLSTATE 428E2)。

注

- 値がプロシージャーから戻されるとき、呼び出し側は以下の方法でその値にアクセスすることができます。
  - GET DIAGNOSTICS ステートメントを使用して、その SQL プロシージャーが別の SQL プロシージャーから呼び出された時点の RETURN_STATUS を検索する。
  - CLI アプリケーションで、ESCAPE 文節の CALL 構文 (?=CALL...) に、戻り値パラメーター・マーカー用にバインドされているパラメーターを使用する。
  - SQLCODE がゼロ未満でない場合に、値 SQLERRD[0] (SQLCODE がゼロ未満の場合、値は -1 とみなされる) を検索して、 SQL プロシージャーの CALL の処理から戻される SQLCA から直接アクセスする。

例

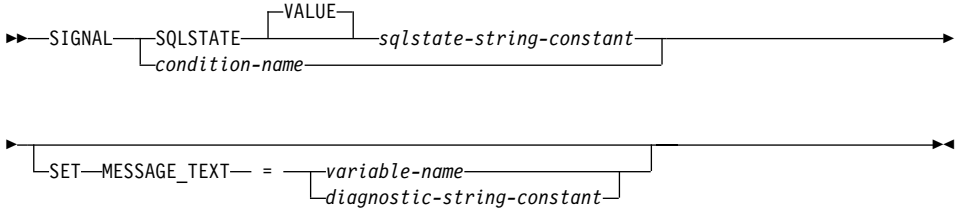
RETURN ステートメントを使用して、SQL ストアード・プロシージャーから状況値を戻します。成功した場合は値ゼロが、失敗した場合は -200 が戻されます。

```
BEGIN
...
   GOTO FAIL
...
   SUCCESS: RETURN 0
   FAIL: RETURN -200
       END
```

SIGNAL ステートメント

SIGNAL ステートメントは、エラーまたは警告条件を通知するのに使用します。これを使用すると、指定した SQLSTATE と任意選択のメッセージ・テキストが、エラーまたは警告とともに戻されます。

構文



説明

SQLSTATE VALUE sqlstate-string-constant

ここで指定したストリング定数は 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' であってはなりません (これは正常終了を表すため)。

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)。

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固有でなければならず、参照できるのは、その条件名が宣言されている複合ステートメントの中だけです。

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。このストリングは SQLCA の SQLERRMC フィールドに戻されます。実際のストリングが 70 バイトを超える場合、警告が出されずに切り捨てられます。この文節を指定する場合、 SQLSTATE または condition-name も指定しなければなりません (SQLSTATE 42601)。

variable-name
複合ステートメント内で宣言しなければならない SQL 変数を識別します。この SQL 変数は、CHAR または VARCHAR データ・タイプとして定義しなければなりません。

diagnostic-string-constant
メッセージ・テキストを含んでいる文字ストリング定数を指定します。

注

- SIGNAL ステートメントが発行される場合、割り当てられる SQLCODE は以下のようになります。

  +438 (SQLSTATE が '01' または '02' で始まる場合)

  -438 (それ以外の場合)

- 例外 ('01' または '02' 以外の SQLSTATE クラス) が通知されていることを、 SQLSTATE または条件が示している場合:
  - ハンドラーが、 SIGNAL ステートメントと同じ複合ステートメント (つまり外側の複合ステートメント) にあり、なおかつ、指定された SQLSTATE、condition-name、または SQLEXCEPTION のハンドラーが、その複合ステートメントに含まれていれば、例外が処理され、制御がそのハンドラーに移動します。
  - そうでなければ、例外は処理されず、制御が即時にその複合ステートメントの最後に戻されます。
- 警告 (SQLSTATE クラス '01') または NOT FOUND 条件 (SQLSTATE クラス '02') が通知されていることを、 SQLSTATE または条件が示している場合:
  - ハンドラーが、 SIGNAL ステートメントと同じ複合ステートメント (つまり外側の複合ステートメント) にあり、なおかつ、指定されている SQLSTATE、condition-name、 SQLWARNING (SQLSTATE クラスが '01' の場合)、または NOT FOUND (SQLSTATE クラスが '02' の場合) のハンドラーが、その複合ステートメントに含まれている場合、その警告または NOT FOUND 条件が処理され、制御がそのハンドラーに移動します。
  - それ以外の場合、警告は処理されず、処理は次のステートメントに続行します。
- SQLSTATE 値は、最初の 2 文字がクラス・コード値で、その後の 3 文字がサブクラス・コード値です。クラス・コード値は、成功および失敗した実行条件のクラスを表します。

  SIGNAL ステートメントでは、有効な SQLSTATE 値を使用することができます。ただし、プログラマーはアプリケーション用に予約されている範囲に

基づいて、新しい SQLSTATE を定義するようにお勧めします。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かもしれない SQLSTATE 値を、意図せずに使用してしまうことを避けられます。

– 定義できるのは、 '7'〜'9' または 'I'〜'Z' という文字で始まる SQLSTATE クラスです。これらのクラス内では、任意のサブクラスを定義できます。
– '0'〜'6' または 'A'〜'H' という文字で始まる 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、'0'〜'H' という文字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。 'I'〜'Z' という文字で始まるサブクラスは定義できます。

例

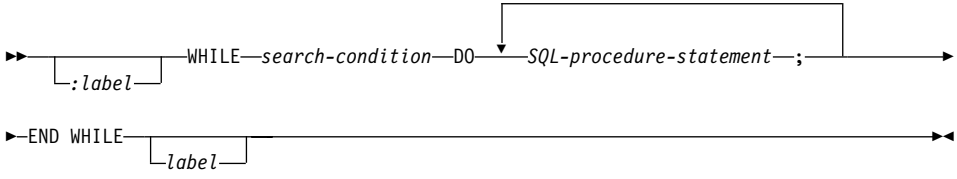
顧客番号をアプリケーションが認識しない場合に、そのアプリケーション・エラーを通知する受注システムの SQL プロシージャー。 ORDERS 表には CUSTOMER 表に対する外部キーが含まれています。ただし、受注を挿入する前に CUSTNO が存在していなければなりません。

```
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
```

WHILE ステートメント

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。

構文



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、開始ラベルと同じにし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 (複数も可) を指定します。

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
```

付録A. SQL の制限値

以下の表は、特定の SQL 制限値を示しています。最も制限が厳しい場合に準拠することによって、プログラマーは容易に移植できるアプリケーション・プログラムを設計することができます。

表 29. 識別子の長さの制限値

	説明	制限値 (バイト単位)
1	許可名の最大長 (1 バイト文字のみ可)	30
2	制約名の最大長	18
3	相関名の最大長	128
4	条件名の最大長	64
5	カーソル名の最大長	18
6	データ・ソースの列名の最大長	128
7	データ・ソースの索引名の最大長	128
8	データ・ソース名の最大長	128
9	データ・ソースの表名 (*リモート表名*) の最大長	128
10	外部プログラム名の最大長	8
11	ホスト識別名の最大長 [a]	255
12	データ・ソースのユーザー (*リモート許可名*) の識別子の最大長	30
13	ラベル名の最大長	64
14	メソッド名の最大長	18
15	パラメーター名の最大長 [b]	128
16	データ・ソースにアクセスするパスワードの最大長	32
17	保管点名の最大長	128
18	スキーマ名の最大長 [c]	30
19	サーバー名 (データベース別名) の最大長	8
20	SQL 変数名の最大長	64
21	ステートメント名の最大長	18
22	変形グループ名の最大長	18
23	非修飾の列名の最大長	30

表 29. 識別子の長さの制限値 (続き)

	説明	制限値 (バイト単位)
24	非修飾のパッケージ名の最大長	8
25	非修飾のユーザー定義タイプ、ユーザー定義関数、バッファー・プール、表スペース、ノード・グループ、トリガー、索引、または索引指定の名前の最大長	18
26	非修飾表名、視点名、ストアード・プロシージャー、ニックネーム、または別名の最大長	128
27	ラッパー名の最大長	128

注:

a ホスト言語コンパイラーによっては、変数名に関してより厳しい制限がある場合があります。

b SQL プロシージャー内のパラメーター名の場合、最大長は 64 バイトです。

c ユーザー定義構造タイプのスキーマ名の場合、最大長は 8 バイトです。

表 30. 数値の制限値

	説明	制限値
1	INTEGER (整数) の最小値	-2 147 483 648
2	INTEGER (整数) の最大値	+2 147 483 647
3	BIGINT (大整数) の最小値	-9 223 372 036 854 775 808
4	BIGINT (大整数) の最大値	+9 223 372 036 854 775 807
5	SMALLINT (小整数) の最小値	-32 768
6	SMALLINT (小整数) の最大値	+32 767
7	10 進数の精度の最大値	31
8	DOUBLE (倍精度) の最小値	-1.79769E+308
9	DOUBLE (倍精度) の最大値	+1.79769E+308
10	DOUBLE (倍精度) の正の最小値	+2.225E-307
11	DOUBLE (倍精度) の負の最大値	-2.225E-307
12	REAL (実数) の最小値	-3.402E+38
13	REAL (実数) の最大値	+3.402E+38
14	REAL (実数) の正の最大値	+1.175E-37
15	REAL (実数) の負の最大値	-1.175E-37

表 31. ストリングの制限値

	説明	制限値
1	CHAR の最大長 (バイト単位)	254
2	VARCHAR の最大長 (バイト単位)	32 672
3	LONG VARCHAR の最大長 (バイト単位)	32 700
4	CLOB の最大長 (バイト単位)	2 147 483 647
5	GRAPHIC の最大長 (文字単位)	127
6	VARGRAPHIC の最大長 (文字単位)	16 336
7	LONG VARGRAPHIC の最大長 (文字単位)	16 350
8	DBCLOB の最大長 (文字単位)	1 073 741 823
9	BLOB の最大長 (バイト単位)	2 147 483 647
10	文字定数の最大長	32 672
11	漢字定数の最大長	16 336
12	連結後の文字ストリングの最大長	2 147 483 647
13	連結後の漢字ストリングの最大長	1 073 741 823
14	連結後の 2 進ストリングの最大長	2 147 483 647
15	16 進定数の最大桁数	16 336
16	カタログ・コメントの最大サイズ (バイト単位)	254
17	実行時の構造タイプ列オブジェクトの最大インスタンス	1 GB

表 32. 日付 / 時刻の制限値

	説明	制限値
1	DATE (日付) の最小値	0001-01-01
2	DATE (日付) の最大値	9999-12-31
3	TIME (時刻) の最小値	00:00:00
4	TIME (時刻) の最大値	24:00:00
5	TIMESTAMP (タイム・スタンプ) の最小値	0001-01-01-00.00.00.000000
6	TIMESTAMP (タイム・スタンプ) の最大値	9999-12-31-24.00.00.000000

表 33. データベース・マネージャーの制限値

	説明	制限値
1	表の列の最大数 [g]	1 012
2	視点の列の最大数 [a]	5 000
3	すべてのオーバーヘッドを含む行の最大長 [b] [g]	32 677

表 33. データベース・マネージャーの制限値 (続き)

	説明	制限値
4	区分当たりの表の最大サイズ (ギガバイト単位) [c] [g]	512
5	区分当たりの索引の最大サイズ (ギガバイト単位)	512
6	区分当たりの表の行の最大数	4×10^9
7	索引キーの最大長 (すべてのオーバーヘッドを含む) (バイト単位)	1 024
8	索引キーの列の最大数	16
9	表の索引の最大数	32 767 または記憶域のサイズによる
10	SQL ステートメントまたは視点で参照される表の最大数	記憶域
11	プリコンパイル済みプログラム中のホスト変数宣言の最大数 [c]	記憶域
12	SQL ステートメント中のホスト変数参照の最大数	32 767
13	挿入または更新に使用するホスト変数の最大長 (バイト単位)	2 147 483 647
14	SQL ステートメントの最大長 (バイト単位)	65 535
15	選択リストでのエレメントの最大数 [g]	1 012
16	WHERE または HAVING 文節の述部の最大数	記憶域
17	GROUP BY 文節中の列の最大数 [g]	1 012
18	GROUP BY 文節中の列の合計長の最大値 (バイト単位) [g]	32 677
19	ORDER BY 文節中の列の最大数 [g]	1 012
20	ORDER BY 文節中の列の合計長の最大値 (バイト単位) [g]	32 677
21	SQLDA の最大サイズ (バイト単位)	記憶域
22	準備されるステートメントの最大数	記憶域
23	プログラムで宣言できるカーソルの最大数	記憶域
24	1 度にオープンできるカーソルの最大数	記憶域
25	SMS 表スペース中の表の最大数	65 534
26	表に対する制約の最大数	記憶域
27	副照会ネストの最大レベル	記憶域

表 33. データベース・マネージャーの制限値 (続き)

	説明	制限値
28	単一のステートメント中の副照会の最大数	記憶域
29	INSERT ステートメント中の値の最大数 [g]	1 012
30	単一の UPDATE ステートメント中の SET 文節の最大数 [g]	1 012
31	固有 (UNIQUE) 制約の列の最大数 (固有索引によってサポートされる)	16
32	固有 (UNIQUE) 制約の列の結合後の最大長 (固有索引によってサポートされる) (バイト単位)	1 024
33	外部キーで参照される列の最大数	16
34	外部キーで参照される列の結合後の最大長 (バイト単位)	1 024
35	検査制約の指定の最大長 (バイト単位)	65 535
36	区分化キーの列の最大数 [e]	500
37	作業単位で変更される行の最大数	記憶域
38	パッケージの最大数	記憶域
39	ステートメント中の定数の最大数	記憶域
40	サーバーの並行ユーザーの最大数 [d]	64 000
41	ストアード・プロシージャー中のパラメーターの最大数	32 767
42	ユーザー定義関数のパラメーターの最大数	90
43	トリガーのカスケード実行時の最大の深さ	16
44	同時に起動できるイベント・モニターの最大数	32
45	通常 DMS 表スペース中の最大サイズ (ギガバイト単位) [c] [g]	512
46	長形式 DMS 表スペースの最大サイズ (テラバイト単位) [c]	2
47	一時 DMS 表スペースの最大サイズ (テラバイト単位) [c]	2
48	インスタンス当たりの並行使用可能なデータベースの最大数	256
49	インスタンス当たりの並行ユーザーの最大数	64 000
50	インスタンス当たりの並行アプリケーションの最大数	1 000

表 33. データベース・マネージャーの制限値 (続き)

	説明	制限値
51	トリガーのカスケードの最大の深さ	16
52	最大区分番号	999
53	DMS 表スペース中の表オブジェクトの最大数 [f]	51 000
54	変数の索引キー部分の最大長 (バイト単位)	255
55	ニックネームによって参照されるデータ・ソース表または視点にある列の最大数	5 000
56	32 ビット・リリースでのバッファー・プールの最大 NPAGES	524 288
57	64 ビット・リリースでのバッファー・プールの最大 NPAGES	2 147 483 647
58	ストアード・プロシージャーの最大ネスト・レベル	16
59	データベース内の表スペースの最大数	4096
60	構造タイプ内の属性の最大数	4082

表 33. データベース・マネージャーの制限値 (続き)

	説明	制限値

注:

a この最大値は、 CREATE VIEW ステートメントで結合を使うことによって達成できます。そのような視点からの選択は、選択リスト内のエレメントの最大数の制限値によって制限されます。

b BLOB、CLOB、LONG VARCHAR、DBCLOB、および LONG VARGRAPHIC の列の実際のデータは、このカウントには含まれません。しかし、そのデータの格納場所についての情報のために、行内に一定のスペースが確保されます。

c 示されている数値は、アーキテクチャー上の制限値であり、近似値です。実際の制限値はもっと小さい場合があります。

d 実際の値は、MAXAGENTS 構成パラメーターの値になります。 MAXAGENTS については、*管理の手引き* を参照してください。

e これは、アーキテクチャー上の制限値です。実際上の制限値としては、索引キーの列の最大数に関する制限値を使用する必要があります。

f 表オブジェクトには、データ、索引、LONG VARCHAR/VARGRAPHIC 列、および LOB 列が含まれます。表データと同じ表スペース中にある表オブジェクトは、制限値に対して余分のオブジェクトとしてカウントされません。しかし、表データとは異なる表スペースにある各表オブジェクトは、表オブジェクトがある表スペース中の表当たりの表オブジェクト・タイプの制限値に対して、実際に 1 個のオブジェクトとしてカウントされます。

g ページ・サイズ固有の値に関しては、 表34 を参照してください。

表 34. データベース・マネージャーのページ・サイズ固有の制限値

	説明	4K ページ・サイズの制限値	8K ページ・サイズの制限値	16K ページ・サイズの制限値	32K ページ・サイズの制限値
1	表の列の最大数	500	1 012	1 012	1 012
3	すべてのオーバーヘッドを含む行の最大長	4 005	8 101	16 293	32 677
4	区分当たりの表の最大サイズ (ギガバイト単位)	64	128	256	512
5	区分当たりの索引の最大サイズ (ギガバイト単位)	64	128	256	512
15	選択リストのエレメントの最大数	500	1 012	1 012	1 012

表 34. データベース・マネージャーのページ・サイズ固有の制限値 (続き)

	説明	4K ページ・サイズの制限値	8K ページ・サイズの制限値	16K ページ・サイズの制限値	32K ページ・サイズの制限値
17	GROUP BY 文節中の列の最大数	500	1 012	1 012	1 012
18	GROUP BY 文節中の列の合計長の最大値 (バイト単位)	4 005	8 101	16 293	32 677
19	ORDER BY 文節中の列の最大数	500	1 012	1 012	1 012
20	ORDER BY 文節中の列の合計長の最大値 (バイト単位)	4 005	8 101	16 293	32 677
29	INSERT ステートメント中の値の最大数	500	1 012	1 012	1 012
30	単一の UPDATE ステートメント中の SET 文節の最大数	500	1 012	1 012	1 012
45	通常 DMS 表スペース中の最大サイズ (ギガバイト単位)	64	128	256	512

付録B. SQL 連絡 (SQLCA)

SQLCA は、すべての SQL ステートメントのそれぞれ実行の終了時に更新される変数の集まりです。実行可能な SQL ステートメント (DECLARE、INCLUDE、および WHENEVER を除く) を含み、オプション LANGLEVEL SAA1 (デフォルト) または MIA を指定してプリコンパイルされたプログラムは、 1 つだけの SQLCA を用意する必要があります。ただし、複数スレッドのアプリケーションでは、スレッドごとに 1 つの SQLCA を用意し、その結果 SQLCA が複数になることがあります。

オプション LANGLEVEL SQL92E を指定してプログラムをプリコンパイルした場合、 SQLCODE 変数または SQLSTATE 変数を SQL 宣言セクションで宣言でき、また SQLCODE 変数をプログラムで宣言できます。

LANGLEVEL SQL92E を使用すると、SQLCA は用意されません。 REXX 以外のすべての言語では、 SQL INCLUDE ステートメントを使用して SQLCA を宣言することができます。 REXX では、自動的に SQLCA が用意されます。

SQLCA の対話式表示

コマンド行プロセッサーでそれぞれのコマンドを使用した後に SQLCA を表示するには、コマンド **db2 -a** を使用します。これにより、SQLCA は後続のコマンドの出力の一部として表示されます。また、SQLCA は db2diag.log ファイルにダンプされます。

SQLCA のフィールドの説明

表 35. SQLCA のフィールド

名前[110]	データ・タイプ	フィールドの値
sqlcaid	CHAR(8)	'SQLCA' を含む記憶域ダンプの「目印」。 SQL プロシージャー本体の解析から行番号情報が戻される場合、6 番目のバイトは 'L' になります。
sqlcabc	INTEGER	SQLCA の長さ (136)。

110. この表によって示されているフィールド名は、 INCLUDE ステートメントによって得られる SQLCA のものです。

表 35. SQLCA のフィールド (続き)

名前[110]	データ・タイプ	フィールドの値
sqlcode	INTEGER	SQL 戻りコード。 個々の SQL 戻りコードの意味については、 *メッセージ解説書* のメッセージの節を参照してください。 **コード 意味** **0** 正常に実行された (1 つ以上の SQLWARN 標識が設定される場合があります)。 **正の値** 正常に実行されたが、警告状態である。 **負の値** エラー状態。
sqlerrml	SMALLINT	*sqlerrmc* の長さ標識 (0 ～ 70 の範囲)。 0 は、*sqlerrmc* の値が関係ないことを示します。
sqlerrmc	VARCHAR (70)	X'FF' で区切られた 1 つまたは複数のトークンが入り、エラー条件の説明の中の変数を置き換えます。 このフィールドは、正常に接続が完了した場合にも使用されます。 NOT ATOMIC 複合 SQL ステートメントが発行されると、最大 7 つのエラーに関する情報を含むことができます。 個々の SQL 戻りコードの意味については、 *メッセージ解説書* のメッセージの節を参照してください。
sqlerrp	CHAR(8)	製品を示す 3 文字の識別子の後に、製品のバージョン、リリース、およびモディフィケーション・レベルを示す 5 桁の数字が続きます。たとえば SQL07010 は、DB2 ユニバーサル・データベース バージョン 7 リリース 1 修正レベル 0 を指します。 SQLCODE がエラー条件を示している場合、そのエラーを戻したモジュールがこのフィールドに示されます。 このフィールドは、正常に接続が完了した場合にも使用されます。
sqlerrd	ARRAY	診断情報の提供に使用される 6 個の INTEGER 変数。 これらの値は、エラーがない場合は通常は空ですが、区分データベースの sqlerrd(6) は例外です。

表 35. SQLCA のフィールド (続き)

名前[110]	データ・タイプ	フィールドの値
sqlerrd(1)	INTEGER	接続が正常に起動された場合、アプリケーションのコード・ページからデータベースのコード・ページに変換する際に予想される混合文字データ (CHAR データ・タイプ) の長さの差の最大値が入ります。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。[a] SQL プロシージャーから正常に戻された場合、その SQL プロシージャーからの戻り状況値が入ります。
sqlerrd(2)	INTEGER	接続を呼び出して正常に実行した場合、データベースのコード・ページからアプリケーションのコード・ページに変換する際に予想される混合文字データ (CHAR データ・タイプ) の長さの差の最大値が入ります。 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。 [a] SQLCA がエラーの発生した NOT ATOMIC 複合 SQL ステートメントの結果である場合、この値はエラーの発生したステートメントの番号に設定されます。
sqlerrd(3)	INTEGER	PREPARE を呼び出して正常に実行した場合は、戻される行の数の見積もりが入ります。 INSERT、UPDATE、および DELETE を実行した後は、実際に影響を受けた行の数になります。複合 SQL を呼び出した場合は、すべてのサブステートメント行の累計が入ります。 CONNECT が呼び出され、データベースが更新可能な場合は 1、データベースが読み取り専用の場合は 2 が入ります。 SQL プロシージャーで CREATE PROCEDURE を呼び出したものの、 SQL プロシージャー本体の解析でエラーが発生した場合、エラーが発生した行番号が入ります。この場合、有効な行番号を表すために、sqlcaid の 6 バイト目は必ず 'L' になります。

表 35. SQLCA のフィールド (続き)

名前[110]	データ・タイプ	フィールドの値
sqlerrd(4)	INTEGER	PREPARE を呼び出して正常に実行した場合は、ステートメントの処理に必要なリソースの相対的なコストの見積もりが入ります。 複合 SQL を呼び出した場合は、正常に実行されたサブステートメントの数のカウントが入ります。 CONNECT を呼び出した場合は、下位レベルのクライアントからの 1 フェーズ・コミットなら 0、 1 フェーズ・コミットなら 1、1 フェーズの読み取り専用コミットなら 2、 2 フェーズ・コミットなら 3 が入ります。
sqlerrd(5)	INTEGER	以下の両方の結果として削除、挿入、または更新された行の合計数が入ります。 • 削除操作が正常に実行された後の制約の実施 • 起動したトリガーから引き起こされた SQL ステートメントの処理 複合 SQL を呼び出した場合は、すべてのサブステートメントの上記のような行の累計が入ります。場合によっては、エラーが発生した場合に、内部エラー・ポインターである負の値が入ります。 CONNECT を呼び出した場合は、サーバーの認証の場合は認証タイプ値 0、クライアントの認証の場合は 1、DB2 コネクトを使用した認証の場合は 2、 DCE 機密保護サービスの認証の場合は 3、認証が指定されていない場合は 255 が入ります。
sqlerrd(6)	INTEGER	区分データベースの場合、エラーまたは警告が出された区分の区分番号が入ります。 エラーまたは警告が出されなかった場合は、このフィールドには調整プログラム・ノードの区分番号が入ります。 このフィールドに入る番号は、db2nodes.cfg ファイルで区分に対して指定されたものと同じです。
sqlwarn	配列	一連の警告標識で、それぞれの標識はブランクか W です。複合 SQL を呼び出した場合は、すべてのサブステートメントについての警告標識の累積が入ります。
sqlwarn0	CHAR(1)	他の標識がすべてのブランクの場合はブランク、 1 つでもブランクでない標識があれば W。
sqlwarn1	CHAR(1)	ホスト変数への割り当て時にストリング列の値が切り捨てられた場合に W になります。 ヌル終止符が切り捨てられた場合は N になります。 CONNECT または ATTACH が正常に実行され、その接続の authID が 8 バイトを超える場合は A になります。

表 35. SQLCA のフィールド (続き)

名前[110]	データ・タイプ	フィールドの値
sqlwarn2	CHAR(1)	関数の引き数からヌル値が削除された場合、W が入ります。[b]
sqlwarn3	CHAR(1)	列の数とホスト変数の数が等しくない場合、W が入ります。
sqlwarn4	CHAR(1)	準備された UPDATE または DELETE ステートメントに WHERE 文節が指定されていない場合に、 W が入ります。
sqlwarn5	CHAR(1)	将来の使用のために予約済み。
sqlwarn6	CHAR(1)	存在しない日付を避けるために日付演算の結果が調整された場合に W が入ります。
sqlwarn7	CHAR(1)	将来の使用のために予約済み。 CONNECT が呼び出されて正常に実行された場合に、DYN_QUERY_MGMT データベース構成パラメーターが使用可能であれば、'E' が入ります。
sqlwarn8	CHAR(1)	変換できなかった文字が置換文字に置き換えられた場合に W になります。
sqlwarn9	CHAR(1)	エラーのある算術式が列関数の処理時に無視された場合に W になります。
sqlwarn10	CHAR(1)	SQLCA 内のフィールドのいずれかで、文字データ値の変換時に変換エラーが起きた場合、W が入ります。
sqlstate	CHAR(5)	直前に実行された SQL ステートメントの結果を示す戻りコード。

注:

a 詳細については、 *アプリケーション開発の手引き* の『複合環境におけるプログラミング』の章の『文字変換の拡張係数』の節を参照してください。

b 一部の関数では、ヌル値が除去されても SQLWARN2 が W にならないことがあります。これは、結果がヌル値の除去によるものでない場合に生じます。

エラー報告の順序

エラー報告の順序は、以下のとおりです。

1. 重大エラー条件は常に報告されます。重大エラーが報告される場合、SQLCA に追加されるものはありません。
2. 重大エラーが発生しなかった場合、デッドロック・エラーはその他のエラーに優先します。

3. その他のエラーの場合、最初の負の SQL コードの SQLCA が戻されます。
4. 負の SQL コードが検出されない場合、最初の警告の SQLCA (つまり正の SQL コード) が戻されます。

   DB2 エンタープライズ拡張エディションの場合、ある区分では空で、他のノードにはデータがある表に対してデータ操作コマンドが発行されると、この規則に例外が発生します。すべての区分で表が空であるため、または UPDATE ステートメントの WHERE 文節の条件を満たす行がもうないために、すべての区分のエージェントが SQL0100W を返した場合のみ、アプリケーションに SQLCODE +100 が返されます。

DB2 エンタープライズ拡張エディションでの SQLCA の使用

DB2 ユニバーサル・データベース エンタープライズ拡張エディションでは、異なる区分にある多くのエージェントが 1 つの SQL ステートメントを実行する場合があり、それぞれのエージェントが異なるエラーまたは警告についての異なる SQLCA を戻す場合があります。 また、調整プログラムのエージェントには独自の SQLCA があります。

アプリケーションについての表示に一貫性を持たせるために、 SQLCA の値はすべて 1 つの構造の中に組み合わされ、 SQLCA のフィールドはグローバルなカウントを表示します。たとえば、次のとおりです。

- エラーと警告のすべてについて、*sqlwarn* フィールドにはすべてのエージェントから受け取った警告標識が入ります。
- 行カウントを表示する *sqlerrd* フィールドの値は、すべてのエージェントからの累計です。

トリガーにより実行された SQL ステートメントの処理中に発生したエラーのケースによっては、 SQLSTATE 09000 が戻されないことがあるので注意してください。

付録C. SQL 記述子域 (SQLDA)

SQLDA は、SQL DESCRIBE ステートメントの実行に必要な変数の集まりです。 SQLDA 変数は、PREPARE、OPEN、FETCH、EXECUTE、および CALL ステートメントでは、オプションとして使用できます。 SQLDA は、動的 SQL との間で情報を伝えます。これは、DESCRIBE ステートメントで使用され、ホスト変数のアドレスで変更して、 FETCH ステートメントで再び使用することができます。

SQLDA は、すべての言語でサポートされていますが、事前定義の宣言を使用できるのは、 C、REXX、FORTRAN、および COBOL に対してだけです。 REXX では、SQLDA は他の言語とは多少異なります。 REXX での SQLDA の使用については、*アプリケーション開発の手引き* を参照してください。

SQLDA の情報の意味は、その使用法によって異なります。 PREPARE と DESCRIBE の場合、 SQLDA は準備されるステートメントに関する情報をアプリケーション・プログラムに提供します。 OPEN、EXECUTE、FETCH、および CALL の場合、SQLDA はホスト変数を記述します。

DESCRIBE および PREPARE において、記述する列のいずれかが LOB タイプ[111]、参照タイプ、またはユーザー定義タイプの場合、 SQLDA 全体の SQLVAR 項目の数を 2 倍にする必要があります。たとえば、次のとおりです。

- 3 つの VARCHAR の列と 1 つの INTEGER の列の含まれている表を記述する場合、 SQLVAR 項目は 4 つになります。
- 2 つの VARCHAR の列と 1 つの CLOB の列、および 1 つの INTEGER の列の含まれている表を記述する場合には、 SQLVAR 項目は 8 つになります。

EXECUTE、FETCH、OPEN、および CALL において、記述する変数のいずれかが LOB タイプ[111] または構造タイプの場合、 SQLDA 全体の SQLVAR 項目の数を 2 倍にする必要があります。[112]

111. LOB ロケーターとファイル参照変数は、 2 倍の SQLDA を必要としません。

112. 特殊タイプと参照タイプは、これらの場合には関係ありません。これは、それらのタイプの場合、データベースが 2 倍の数の項目の追加情報を必要とはしないないためです。

フィールドの説明

SQLDA は、4 つの変数と、その後に SQLVAR と総称して呼ばれる変数の任意の数のオカレンスによって構成されています。 OPEN、FETCH、EXECUTE、および CALL では、 SQLVAR の各オカレンスによってホスト変数が記述されます。 DESCRIBE と PREPARE では、 SQLVAR の各オカレンスによって結果表の列が記述されます。 SQLVAR の項目には、次の 2 つのタイプがあります。

1. **基本 SQLVAR:** これらの項目は常に存在します。これらの項目には、データ・タイプのコード、長さ属性、列名、ホスト変数のアドレス、および標識変数アドレスなどの列またはホスト変数に関する基本的な情報が入れられます。
2. **副次 SQLVAR:** これらの項目は、上記で概説した規則に従って SQLVAR 項目の数が 2 倍になった場合にのみ存在します。ユーザー定義タイプ (特殊または構造) の場合は、ユーザー定義タイプ名が入ります。参照タイプの場合は、参照のターゲット・タイプが入れられます。 LOB の場合は、ホスト変数の長さ属性と、実際の長さの入っているバッファーを指すポインターが入れられます。[113] ロケーターまたはファイル参照変数を使用して LOB を示す場合、これらの項目は必要ありません。

SQLDA に上記 2 つのタイプの項目が両方とも含まれる場合、基本 SQLVAR は副次 SQLVAR のブロックの前のブロックに入れられます。それぞれの場合において、項目の数は SQLD の値で示されます (副次 SQLVAR 項目の多くは使用されない場合があります)。

DESCRIBE によって SQLVAR 項目が設定される環境については、 1225ページの『SQLDA に対する DESCRIBE の効果』に示されています。

113. 特殊タイプと LOB の情報が重なり合うことはないので、 DESCRIBE においては、SQLVAR 項目を 3 倍にしなくても、LOB に基づく特殊タイプを使用できます。

SQLDA ヘッダーのフィールド

表 36. SQLDA ヘッダーのフィールド

C での名前	SQL データ・タイプ	DESCRIBE および PREPARE を使用する場合 (SQLN を除き、データベース・マネージャーが設定)	FETCH、OPEN、EXECUTE、および CALL で使用する場合 (ステートメントの実行前にアプリケーションにより設定)
sqldaid	CHAR(8)	このフィールドの 7 番目のバイトは、SQLDOUBLED という名前のフラグ・バイトです。 データベース・マネージャーは、それぞれの列に対して 2 つの SQLVAR 項目が作成された場合は SQLDOUBLED を文字 '2' に設定し、その他の場合はブランク (ASCII では X'20'、 EBCDIC では X'40') に設定します。 SQLDOUBLED がいつ設定されるかについては、 1225ページの『SQLDA に対する DESCRIBE の効果』を参照してください。	このフィールドの 7 番目のバイトは、SQLVAR の数が 2 倍になった場合に使用されます。 これは SQLDOUBLED という名前のフィールドです。記述されるホスト変数が構造タイプ、BLOB、CLOB、または DBCLOB の場合、この 7 番目のバイトは文字 '2' に設定され、それ以外の場合は任意の文字に設定できます (ブランクの使用をお勧めします)。 CALL ステートメントを使用し、 1 つまたは複数の SQLVAR でデータ・フィールドを FOR BIT DATA として定義する場合には、 6 番目のバイトを '+' の文字に、それ以外の場合は任意の文字に設定できます (ブランクの使用を勧めします)。
sqldabc	INTEGER	32 ビットの場合、SQLDA の長さ = SQLN*44+16。 64 ビットの場合、SQLDA の長さ = SQLN*56+16。	32 ビットの場合、SQLDA の長さ >= SQLN*44+16。 64 ビットの場合、SQLDA の長さ >= SQLN*56+16。
sqln	SMALLINT	データベース・マネージャーはこれを変更しません。 DESCRIBE ステートメントを実行する前に、ゼロまたはゼロより大きい値を設定する必要があります。 これは、SQLVAR のオカレンスの合計数を示します。	SQLDA の SQLVAR のオカレンスの合計。 SQLN には、ゼロまたはゼロより大きい値を設定する必要があります。
sqld	SMALLINT	データベース・マネージャーによって、結果表の列の数 (記述するステートメントが選択ステートメントでない場合はゼロ) に設定されます。	SQLVAR のオカレンスにより記述されるホスト変数の数

基本 SQLVAR のオカレンスのフィールド

表 37. 基本 SQLVAR のフィールド

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
sqltype	SMALLINT	列のデータ・タイプと、その列がヌル値可能かどうかを示します。 1227ページの表39 は、許される値とその意味をリストしています。 特殊タイプまたは参照タイプの場合は、その基本タイプのデータ・タイプがこのフィールドに入れられます。構造タイプの場合は、そのタイプの変換グループ (CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターに基づく) の FROM SQL 変換関数が入れられます。基本 SQLVAR には、それがユーザー定義タイプまたは参照タイプの記述の一部であるかどうかを示す標識はありません。	ホスト変数の場合と同じ。日付 / 時刻の値のホスト変数は、文字ストリング変数でなければなりません。 FETCH の場合、日付 / 時刻のタイプ・コードは、固定長文字ストリングを意味します。 sqltype が偶数値の場合、sqlind フィールドは無視されます。
sqllen	SMALLINT	列の長さ属性。日付 / 時刻の列の場合は、値のストリング表記の長さ。 1227ページの表39 を参照してください。 ラージ・オブジェクト・ストリングの場合、この値は 0 に設定されます。その長さ属性が 2 バイト整数に入る小さいものであっても、設定値はやはり 0 になります。	ホスト変数の長さ属性。 1227ページの表39 を参照してください。 CLOB、DBCLOB、および BLOB の列の場合、データベース・マネージャーはこの値を無視します。代わりに、副次 SQLVAR の len.sqllonglen フィールドが使用されます。

表 37. 基本 SQLVAR のフィールド (続き)

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
sqldata	ポインター	文字ストリング SQLVAR の場合、FOR BIT DATA 属性で列を定義すると、sqldata は 0 になります。 列に FOR BIT DATA 属性が指定されていない場合、値はデータのエンコードによって異なります。 1 バイト SBCS エンコード・データの場合、sqldata はその SBCS コード・ページになります。 混合 DBCS エンコード・データの場合は、 sqldata は、複合 DBCS コード・ページに関連付けられた SBCS コード・ページになります。 日本語または中国語 (繁体字) EUC エンコード・データの場合には、 sqldata は複合 EUC コード・ページになります。 他のすべてのタイプの列の場合、sqldata は未定義です。	ホスト変数のアドレスを含みます (取り出したデータを保管する場所)。
sqlind	ポインター	文字ストリング SQLVAR の場合、sqlind は 0 になります。ただし、sqlind が複合 DBCS コード・ページに関連付けられた DBCS コード・ページの場合の、混合 DBCS エンコード・データは例外です。 他のすべてのタイプの列の場合、sqlind は未定義です。	関連する標識変数があれば、そのアドレスが入ります。それ以外の場合は、使用されません。 sqltype が偶数値の場合、sqlind フィールドは無視されます。

表 37. 基本 SQLVAR のフィールド (続き)

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
sqlname	VARCHAR(30)	非修飾の列名。 システムが生成した名前を持つ列 (結果列は単一の列から直接得られたものでなく、 AS 文節を使用して名前を指定していない) の場合、 13 バイト目は X'FF' に設定されます。 AS 文節によって列名が指定された場合は、このバイトは X'00' になります。	DRDA アプリケーション・サーバーへのアクセスに CALL ステートメントで使用する場合、 FOR BIT DATA ストリングを指定するには、sqlname を次のように設定します。 • sqlname の長さは 8 • sqlname の最初の 4 バイトは X'00000000' • sqlname の残りのバイトは予約済み (現在は無視される) さらに、sqltype は CHAR、VARCHAR、または LONG VARCHAR を示すものでなければならず、 sqldaid フィールドの 6 バイト目は文字 '+' に設定されます。 この手法は、OPEN および EXECUTE で、 DB2 コネクトを使用してサーバーにアクセスする際に使用できます。

副次 SQLVAR のオカレンスのフィールド

表 38. 副次 SQLVAR のフィールド

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
len.sqllonglen	INTEGER	BLOB、CLOB、または DBCLOB の列の長さ属性。	BLOB、CLOB、または DBCLOB ホスト変数の長さ属性。データベース・マネージャーは、それらのデータ・タイプに対しては基本 SQLVAR の SQLLEN フィールドを無視します。 長さ属性は、BLOB または CLOB ではバイト数、DBCLOB では文字数になります。

表 38. 副次 SQLVAR のフィールド (続き)

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
reserve2	32 ビットの場合は CHAR(3)、64 ビットの場合は CHAR(11)。	使用されません。	使用されません。
sqlflag4	CHAR(1)	SQLVAR の表している参照タイプが sqldatatype_name に指定されたターゲット・タイプに関連付けられたものである場合、この値は X'01' になります。 SQLVAR の表している構造タイプで、 sqldatatype_name にユーザー定義タイプ名が指定されている場合、値は X'12' になります。それ以外の場合は、値は X'00' です。	SQLVAR の表している参照タイプが sqldatatype_name に指定されたターゲット・タイプに関連付けられたものである場合、 X'01' に設定されます。 SQLVAR の表している構造タイプで、 sqldatatype_name にユーザー定義タイプ名が指定されている場合、 X'12' に設定されます。それ以外の場合は、値は X'00' です。

表 38. 副次 SQLVAR のフィールド (続き)

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合
sqldatalen	ポインター	使用されません。	BLOB、CLOB、および DBCLOB ホスト変数でのみ使用されます。 このフィールドが NULL (ヌル値) の場合は、データの直前に実際の長さ (文字単位) を 4 バイトで保管し、 SQLDATA はフィールド長の最初のバイトを指すようにする必要があります。 このフィールドが NULL (ヌル値) でない場合は、対応する基本 SQLVAR 内の SQLDATA フィールドの指すバッファー内のデータの実際の長さ (バイト単位、 DBCLOB の場合も含む) の入っている 4 バイト長のバッファーを指すポインターが入れられます。 このフィールドを使用するか否かに関係なく、 len.sqllonglen フィールドは設定する必要があります。
sqldatatype_name	VARCHAR(27)	ユーザー定義タイプの列の場合、データベース・マネージャーはこれを完全修飾ユーザー定義タイプ名に設定します。 [1] 参照タイプの場合は、データベース・マネージャーはこれを参照のターゲット・タイプの完全修飾タイプ名に設定します。	構造タイプの場合、表の注[1] で示されている形式の完全修飾ユーザー定義タイプ名に設定されます。
予約済み	CHAR(3)	使用されません。	使用されません。

表 38. 副次 SQLVAR のフィールド (続き)

資料名	データ・タイプ	DESCRIBE および PREPARE で使用する場合	FETCH、OPEN、EXECUTE、および CALL で使用する場合

注:

1. 最初の 8 バイトには、タイプのスキーマ名が入れられます (必要に応じて右側にスペースが入れられます)。 バイト 9 はドット文字 (.) です。バイト 10 ～ 27 には、タイプ名のうちの下位部分が入れられます。それは、右側にスペースを入れて拡張することはできません。

このフィールドの主な目的は、タイプの名前を入れることですが、 IBM の定義済みデータ・タイプ用に設定することもできます。この場合、スキーマ名は SYSIBM、名前の下位部分は DATATYPES カタログ視点の TYPENAME 列に保管されている名前になります。たとえば、次のとおりです。

```
type name         length   sqldatatype_name
---------         ------   ----------------
A.B               10       A       .B
INTEGER           16       SYSIBM  .INTEGER
"Frank's".SMINT   13       Frank's .SMINT
MY."type  "       15       MY      .type
```

SQLDA に対する DESCRIBE の効果

DESCRIBE または PREPARE INTO ステートメントの場合、データベース・マネージャーは、常に SQLD を結果セットの列の数に設定します。

SQLDA の SQLVAR は、以下の場合に設定されます。

- SQLN >= SQLD で、しかも LOB、ユーザー定義タイプ、または参照タイプの列がない

  最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。

- SQLN >= 2*SQLD で、しかも少なくとも 1 つの列が LOB、ユーザー定義タイプ、または参照タイプである

  2 倍の数の SQLD SQLVAR 項目が設定され、SQLDOUBLED は '2' に設定されます。

- SQLD <= SQLN < 2*SQLD で、しかも少なくとも 1 つの列が特殊タイプまたは参照タイプで、 LOB の列または構造タイプの列はない

  最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。 SQLWARN バインド・オプションが YES の場合は、警告 SQLCODE +237 (SQLSTATE 01594) が出されます。

SQLDA の SQLVAR は、以下の場合には設定されません (さらに多くのスペースの割り振りと別の DESCRIBE が必要)。

- SQLN < SQLD で、しかも LOB、ユーザー定義タイプ、または参照タイプの列がない

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。SQLWARN バインド・オプションが YES の場合は、警告 SQLCODE +236 (SQLSTATE 01005) が出されます。

  DESCRIBE が正常に実行される場合には、SQLD 個の SQLVAR が割り振られます。

- SQLN < SQLD で、しかも少なくとも 1 つの列が特殊タイプまたは参照タイプで、 LOB の列または構造タイプの列はない

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。SQLWARN バインド・オプションが YES の場合は、警告 SQLCODE +239 (SQLSTATE 01005) が出されます。

  特殊タイプ名および参照タイプのターゲット・タイプを含む DESCRIBE が正常に実行されると、 2*SQLD 個の SQLVAR が割り振られます。

- SQLN < 2*SQLD で、しかも少なくとも 1 つの列が LOB または構造タイプである

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。(SQLWARN バインド・オプションの設定に関係なく) 警告 SQLCODE +238 (SQLSTATE 01005) が出されます。

  DESCRIBE が正常に実行される場合には、2*SQLD 個の SQLVAR が割り振られます。

上記リストでの「LOB 列」には、ソース・タイプが LOB タイプである特殊タイプも含まれます。

DESCRIBE (または PREPARE INTO) から警告 SQLCODE +236、+237、+239 を戻すかどうかを制御するには、 BIND または PREP コマンドの SQLWARN オプションを使用します。 使用するアプリケーション・コードでは、これらの SQLCODE がいつ戻されてもよいようにしておいてください。選択リストに LOB または構造タイプの列が含まれていて、 SQLDA の中の SQLVAR が不足している場合には、常に警告 SQLCODE +238 が戻されます。これは、結果セット内に LOB または構造タイプの列があるために SQLVAR 数を 2 倍にする必要があることをアプリケーションに認識させる唯一の方法です。

構造タイプの列を記述しようとしているものの、 FROM SQL 変換が定義されていない場合 (CURRENT DEFAULT TRANSFORM GROUP 特殊レジスターを

使用した TRANSFORM GROUP の指定が行われていない (SQLSTATE 42741) か、またはその名前グループが FROM SQL 変換関数を定義していない (SQLSTATE 42744) ため)、 DESCRIBE はエラーを戻します。このエラーは、構造タイプの列がある表の DESCRIBE で戻されるエラーと同じです。

SQLTYPE と SQLLEN

表39 に、 SQLDA の SQLTYPE フィールドと SQLLEN フィールドに現れる値を示します。 DESCRIBE と PREPARE INTO においては、 SQLTYPE の値が偶数ならその列ではヌル値 (NULL) が使えないこと、また奇数ならその列でヌル値が可能であることを意味しています。 FETCH、OPEN、EXECUTE、および CALL において、 SQLTYPE の値が偶数の場合には標識変数がないこと、奇数の場合には SQLIND に標識変数のアドレスが入れられていることを意味しています。

表 39. SQLTYPE と SQLLEN の値 (DESCRIBE、FETCH、OPEN、EXECUTE、および CALL の場合)

	DESCRIBE および PREPARE INTO の場合		**FETCH、OPEN、EXECUTE、および CALL の場合**	
SQLTYPE	**列のデータ・タイプ**	**SQLLEN**	**ホスト変数のデータ・タイプ**	**SQLLEN**
384/385	日付	10	日付の固定長文字ストリング表示	ホスト変数の長さ属性
388/389	時刻	8	時刻の固定長文字ストリング表示	ホスト変数の長さ属性
392/393	タイム・スタンプ	26	タイム・スタンプの固定長文字ストリング表示	ホスト変数の長さ属性
396/397	DATALINK	列の長さ属性	DATALINK	ホスト変数の長さ属性
400/401	適用外	適用外	NUL 文字で終了する漢字ストリング	ホスト変数の長さ属性
404/405	BLOB	0 *	BLOB	使用されません。*
408/409	CLOB	0 *	CLOB	使用されません。*
412/413	DBCLOB	0 *	DBCLOB	使用されません。*
448/449	可変長文字ストリング	列の長さ属性	可変長文字ストリング	ホスト変数の長さ属性
452/453	固定長文字ストリング	列の長さ属性	固定長文字ストリング	ホスト変数の長さ属性

表 39. SQLTYPE と SQLLEN の値 (DESCRIBE、FETCH、OPEN、EXECUTE、および CALL の場合) (続き)

	DESCRIBE および PREPARE INTO の場合		FETCH、OPEN、EXECUTE、および CALL の場合	
SQLTYPE	**列のデータ・タイプ**	**SQLLEN**	**ホスト変数のデータ・タイプ**	**SQLLEN**
456/457	長形式可変長文字ストリング	列の長さ属性	長形式可変長文字ストリング	ホスト変数の長さ属性
460/461	適用外	適用外	NUL 文字で終了する文字ストリング	ホスト変数の長さ属性
464/465	可変長漢字ストリング	列の長さ属性	可変長漢字ストリング	ホスト変数の長さ属性
468/469	固定長漢字ストリング	列の長さ属性	固定長漢字ストリング	ホスト変数の長さ属性
472/473	長形式可変長漢字ストリング	列の長さ属性	長形式漢字ストリング	ホスト変数の長さ属性
480/481	浮動小数点数	倍精度の場合は 8、単精度の場合は 4	浮動小数点数	倍精度の場合は 8、単精度の場合は 4
484/485	パック 10 進数	バイト 1 は精度、バイト 2 は位取り	パック 10 進数	バイト 1 は精度、バイト 2 は位取り
492/493	大整数	8	大整数	8
496/497	大整数	4	大整数	4
500/501	小整数	2	小整数	2
916/917	該当なし	該当なし	BLOB ファイル参照変数	267
920/921	該当なし	該当なし	CLOB ファイル参照変数	267
924/925	該当なし	該当なし	DBCLOB ファイル参照変数	267
960/961	該当なし	該当なし	BLOB ロケーター	4
964/965	該当なし	該当なし	CLOB ロケーター	4
968/969	該当なし	該当なし	DBCLOB ロケーター	4

注:

- 副次 SQLVAR の len.sqllonglen フィールドに、列の長さ属性が入れられます。
- SQLTYPE は、DB2 での移行性のために旧バージョンから変更されました。 旧バージョンの値 (旧バージョンの SQL 解説書を参照) は、引き続きサポートされています。

認識されない非サポート SQLTYPE

SQLDA の SQLTYPE フィールドに表示される値は、データの送信側および受信側で使用可能なデータ・タイプ・サポートのレベルによって異なります。 これは、新しいデータ・タイプが製品に追加される場合に特に重要です。

新しいデータ・タイプは、データの送信側または受信側にサポートされることもあれば、サポートされないこともあり、データの送信側や受信側に認識されないことさえあります。 状況に応じて、新しいデータ・タイプが戻されたり、送信側と受信側の両方が認めた互換データ・タイプが戻されたり、あるいは結果としてエラーが発生したりします。

送信側と受信局が互換データ・タイプの使用に同意する場合、以下に示すマッピングが実行されます。このマッピングは、送信側または受信側の少なくともどちらかが指定データ・タイプをサポートしない場合に実行されます。 非サポート・データ・タイプは、アプリケーションまたはデータベース・マネージャーのどちらかによって提供されます。

データ・タイプ	互換データ・タイプ
BIGINT	DECIMAL(19, 0)
ROWID	VARCHAR(40) FOR BIT DATA[114]

SQLDA では、データ・タイプが置換されたことは示されないので注意してください。

パック 10 進数

パック 10 進数は、一種の 2 進コードによる 10 進数 (BCD) 表記で保管されます。 BCD においては、1 ニブル (4 ビット) で 10 進数の 1 桁が表されます。たとえば、0001 0111 1001 は 179 を表します。したがって、パック 10 進数の値はニブルごとに読む必要があります。 値の保管はバイト単位で行い、16 進数表記としてそれらのバイトを読み、それを 10 進数に戻します。たとえば、0001 0111 1001 は、2 進表記では 00000001 01111001 となります。 この数値を 16 進数として読むと、0179 になります。

小数点は、位取りによって決まります。 たとえば、DEC(12,5) の列の場合、小数点より右側に 5 桁あることになります。

符号は、桁数を表すニブルの右側のニブルで示します。 正記号または負記号は、以下のように示します。

114. ROWID は DB2 ユニバーサル・データベース (OS/390 版) バージョン 6 でサポートされています。

表 40. パック 10 進数の符号標識の値

	表記		
符号	**2 進**	**10 進**	**16 進**
正符号 (+)	1100	12	C
負符号 (-)	1101	13	D

まとめ:
1. 値を保管するときは、*p* /2+1 個のバイトを割り振ります (*p* は精度)。
2. 値を表すために、ニブルを左から右へ割り当てます。 数値の精度が偶数の場合は、最初にニブルを追加します。この割り当てには、先行 (無効な) ゼロと後続 (有効な) ゼロの桁が含まれます。
3. 符号ニブルは、最後のバイトの第 2 ニブルになります。

パック 10 進数を変換するもう 1 つの方法があります。 283ページの『CHAR』 を参照してください。

たとえば、次のとおりです。

列	値 バイトごとにグループにした 16 進のニブル
DEC(8,3)	6574.23 00 65 74 23 0C
DEC(6,2)	-334.02 00 33 40 2D
DEC(7,5)	5.2323 05 23 23 0C
DEC(5,2)	-23.5 02 35 0D

10 進数の SQLLEN フィールド

SQLLEN フィールドには、 10 進数の列の精度 (第 1 バイト) と位取り (第 2 バイト) が入れられます。 アプリケーションを移植可能にするには、精度のバイトと位取りのバイトを短整数として一度に設定するのではなく、個々に設定するようにしてください。これによって、整数のバイト逆転の問題が回避されます。

たとえば、C の場合には以下のようにします。

```
((char *)&(sqlda->sqlvar[i].sqllen))[0] = precision;
((char *)&(sqlda->sqlvar[i].sqllen))[1] = scale;
```

付録D. カタログ視点

データベース・マネージャーは、 2 組みのシステム・カタログの視点を作成し、保守しています。この付録は、列名とデータ・タイプを含めて、個々のシステム・カタログ視点について説明しています。すべてのシステム・カタログ視点は、 CREATE DATABASE コマンドによってデータベースが作成される時点で作成されます。カタログ視点は、明示的に作成または除去することはできません。システム・カタログ視点は、SQL データ定義ステートメント、環境ルーチン、および特定のユーティリティーに対応する通常の操作の過程で更新されます。システム・カタログ視点のデータは、通常の SQL 照会機能によって使用可能です。特定の更新可能なカタログ視点を除いて、システム・カタログ視点は、通常の SQL データ操作コマンドを使用して変更することはできません。

カタログ視点は、バージョン 1 でのカタログの基礎表に加えてサポートされます。視点は、SYSCAT スキーマに入っており、すべての視点に対する SELECT 特権が、デフォルト解釈により PUBLIC に付与されています。アプリケーション・プログラムは、カタログの基礎表に対してではなく、これらの視点を対象に作成する必要があります。 SYSCAT スキーマ中の視点のサブセットから成る視点の 2 番目のセットには、最適化プログラムによって使用される統計情報が入ります。 SYSSTAT スキーマの視点には、更新可能な列がいくつか含まれています。

警告: ここでは、アプリケーションで SYSSTAT 視点を使って特定の列を更新しても、 SYSCAT 視点は読み取り専用にすることを意図しています。現時点では、SYSCAT 視点は読み取り専用ではありません。アプリケーション開発者は、アプリケーションを作成しても、その目的を SYSSTAT 視点を使ったカタログ情報の更新に限定するよう警告されています。 SYSCAT 視点は次期バージョンの移行後に読み取り専用になる予定です。

カタログ視点は、カタログの基礎表よりもさらに一貫した規則に基づいて設計されています。そういうわけで、列の順序はリリースによって変更されることがあります。このプログラミング論理の影響を受けないようにするには、SELECT * を使ってデフォルトに任せるのではなく、選択リストでそれらの列を必ず明示的に指定します。列の名前は、それが記述するオブジェクトのタイプに基づいて次のような一貫した名前が用いられています。

記述されるオブジェクト	列名
表	TABSCHEMA, TABNAME
索引	INDSCHEMA, INDNAME
視点	VIEWSCHEMA, VIEWNAME
制約	CONSTSCHEMA, CONSTNAME
トリガー	TRIGSCHEMA, TRIGNAME
パッケージ	PKGSCHEMA, PKGNAME
タイプ	TYPESCHEMA, TYPENAME, TYPEID
関数	FUNCSCHEMA, FUNCNAME, FUNCID
列	COLNAME
スキーマ	SCHEMANAME
表スペース	TBSPACE
ノードグループ	NGNAME
バッファー・プール	BPNAME
イベント・モニター	EVMONNAME
タイム・スタンプ	CREATE_TIME

- 『更新可能なカタログ視点』
- 1233ページの『カタログ視点の「ロードマップ」』
- 1235ページの『更新可能なカタログ視点の「ロードマップ」』

更新可能なカタログ視点

更新可能な視点には、最適化プログラムで使用される統計情報が入れられます。仮のデータベースのパフォーマンスを調べるために、これらの視点の列の一部を変更することができます。特定のユーザーの更新可能カタログ視点にオブジェクト (表、列、関数、または索引) が現れるのは、そのユーザーがそれを作成した場合か、そのユーザーにそのオブジェクトに対する CONTROL 特権、または明示的な DBADM 特権が与えられている場合だけです。これらの視点は、SYSSTAT スキーマに入っています。それらはシステム・カタログ基礎表の先頭で定義されます。

統計値を初めて変更する場合には、その前に RUNSTATS コマンドを発行して、すべての統計量に現在の状態が反映されるようにすることをお勧めします。

カタログ視点の「ロードマップ」

説明	カタログ視点	ページ
構造化データ・タイプの属性	SYSCAT.ATTRIBUTES	1237
データベースに対する権限	SYSCAT.DBAUTH	1255
ノードグループのバッファー・プールの構成	SYSCAT.BUFFERPOOLS	1240
ノードのバッファー・プールのサイズ	SYSCAT.BUFFERPOOLNODES	1239
cast 関数	SYSCAT.CASTFUNCTIONS	1241
検査制約	SYSCAT.CHECKS	1242
列の特権	SYSCAT.COLAUTH	1243
列	SYSCAT.COLUMNS	1247
検査制約によって参照される列	SYSCAT.COLCHECKS	1244
キーで使用される列	SYSCAT.KEYCOLUSE	1280
列オプションの詳細	SYSCAT.COLOPTIONS	1246
列統計値の詳細	SYSCAT.COLDIST	1245
制約の従属関係	SYSCAT.CONSTDEP	1252
データ・タイプ	SYSCAT.DATATYPES	1253
イベント・モニターの定義	SYSCAT.EVENTMONITORS	1257
現在モニター中のイベント	SYSCAT.EVENTS	1259
階層 (タイプ、表、視点)	SYSCAT.FULLHIERARCHIES	1260
関数の従属関係	SYSCAT.FUNCDEP	1261
関数マッピング	SYSCAT.FUNCMAPPINGS	1264
関数マッピングのオプション	SYSCAT.FUNCMAPOPTIONS	1262
関数マッピングのパラメーター・オプション	SYSCAT.FUNCMAPPARMOPTIONS	1263
関数のパラメーター	SYSCAT.FUNCPARMS	1265
階層 (タイプ、表、視点)	SYSCAT.HIERARCHIES	1272
索引特権	SYSCAT.INDEXAUTH	1273
索引列	SYSCAT.INDEXCOLUSE	1274
索引の従属関係	SYSCAT.INDEXDEP	1275
索引	SYSCAT.INDEXES	1276
索引のオプション	SYSCAT.INDEXOPTIONS	1279
ノードグループの定義	SYSCAT.NODEGROUPS	1283
ノードグループのノード	SYSCAT.NODEGROUPDEF	1282

説明	カタログ視点	ページ
オブジェクト・マッピング	SYSCAT.NAMEMAPPINGS	1281
パッケージの従属関係	SYSCAT.PACKAGEDEP	1285
パッケージ特権	SYSCAT.PACKAGEAUTH	1284
パッケージ	SYSCAT.PACKAGES	1286
区分化マップ	SYSCAT.PARTITIONMAPS	1290
パススルー特権	SYSCAT.PASSTHRUAUTH	1291
プロシージャーのオプション	SYSCAT.PROCOPTIONS	1295
プロシージャーのパラメーター・オプション	SYSCAT.PROCPARMOPTIONS	1296
プロシージャーのパラメーター	SYSCAT.PROCPARMS	1297
DB2 ユニバーサル・データベース (OS/390 版) との互換性を提供	SYSIBM.SYSDUMMY1	1236
参照制約	SYSCAT.REFERENCES	1299
リモート表オプション	SYSCAT.TABOPTIONS	1316
反対方向データ・タイプ・マッピング	SYSCAT.REVTYPEMAPPINGS	1300
スキーマ特権	SYSCAT.SCHEMAAUTH	1302
スキーマ	SYSCAT.SCHEMATA	1303
サーバー・オプション	SYSCAT.SERVEROPTIONS	1304
サーバー・オプションの値	SYSCAT.USEROPTIONS	1322
パッケージ中のステートメント	SYSCAT.STATEMENTS	1306
ストアード・プロシージャー	SYSCAT.PROCEDURES	1292
システム・サーバー	SYSCAT.SERVERS	1305
表の制約	SYSCAT.TABCONST	1309
表の特権	SYSCAT.TABAUTH	1307
表	SYSCAT.TABLES	1310
表スペース	SYSCAT.TABLESPACES	1314
表スペースの使用特権	SYSCAT.TBSPACEAUTH	1317
トリガーの従属関係	SYSCAT.TRIGDEP	1318
トリガー	SYSCAT.TRIGGERS	1319
タイプ・マッピング	SYSCAT.TYPEMAPPINGS	1320
ユーザー定義関数	SYSCAT.FUNCTIONS	1267
視点の従属関係	SYSCAT.VIEWDEP	1323
視点	SYSCAT.TABLES	1310
	SYSCAT.VIEWS	1324

説明	カタログ視点	ページ
ラッパー・オプション	SYSCAT.WRAPOPTIONS	1325
ラッパー	SYSCAT.WRAPPERS	1326

更新可能なカタログ視点の「ロードマップ」

説明	カタログ視点	ページ
列	SYSSTAT.COLUMNS	1328
索引	SYSSTAT.INDEXES	1332
列統計値の詳細	SYSSTAT.COLDIST	1327
表	SYSSTAT.TABLES	1335
ユーザー定義関数	SYSSTAT.FUNCTIONS	1330

SYSIBM.SYSDUMMY1

1 つの行が入っています。この視点は、DB2 ユニバーサル・データベース (OS/390 版) との互換性が求められるアプリケーションで使用できます。

表 41. SYSCAT.DUMMY1 カタログ視点

列名	データ・タイプ	ヌル値可	説明
IBMREQD	CHAR(1)		Y

SYSCAT.ATTRIBUTES

ユーザー定義の構造化データ・タイプに定義されている各属性 (継承された属性の中で適用できるものを含む) ごとに 1 つの行が含まれます。

表 42. SYSCAT.ATTRIBUTES カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPESCHEMA	VARCHAR(128)		属性を含む構造化データ・タイプの修飾名。
TYPENAME	VARCHAR(18)		
ATTR_NAME	VARCHAR(18)		属性名。
ATTR_TYPESCHEMA	VARCHAR(128)		属性のタイプの修飾名。
ATTR_TYPENAME	VARCHAR(18)		
TARGET_TYPESCHEMA	VARCHAR(128)		属性のタイプが REFERENCE の場合、ターゲット・タイプの修飾名。属性のタイプが REFERENCE でない場合は、ヌル値。
TARGET_TYPENAME	VARCHAR(18)		
SOURCE_TYPESCHEMA	VARCHAR(128)		属性が使われたデータ・タイプ階層における、データ・タイプの修飾名。非継承属性の場合、これらの列は TYPESCHEMA および TYPENAME と同じです。
SOURCE_TYPENAME	VARCHAR(18)		
ORDINAL	SMALLINT		構造化データ・タイプの定義における属性の位置 (ゼロから開始する)。
LENGTH	INTEGER		データの最大長。特殊タイプの場合は 0。LENGTH 列は、DECIMAL フィールドに対する精度を示します。
SCALE	SMALLINT		DECIMAL フィールドの位取り。 DECIMAL 以外の場合は 0 になります。
CODEPAGE	SMALLINT		属性のコード・ページ。 FOR BIT DATA 属性を指定して定義されていない文字ストリング属性の場合、値はデータベース・コード・ページです。漢字ストリング属性の場合、値は (複合) データベース・コード・ページによる暗黙の DBCS コード・ページです。それ以外の場合の値は 0 です。
LOGGED	CHAR(1)		LOB タイプまたは LOB に基づく特殊タイプの属性だけに適用されます (それ以外はブランク)。 Y = 属性のログ記録を取る。 N = 属性のログ記録を取らない。

表 42. SYSCAT.ATTRIBUTES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
COMPACT	CHAR(1)		LOB タイプまたは LOB に基づく特殊タイプの属性だけに適用されます (それ以外はブランク)。 Y = 記憶域内で属性を圧縮する。 N = 属性を圧縮しない。
DL_FEATURES	CHAR(10)		DATALINK タイプ属性だけに適用されます。REFERENCE タイプ属性の場合はブランクです。それ以外の場合、ヌル値です。リンク・タイプ、制御モード、回復、およびリンク解除特性など、さまざまな DATALINK 機能をエンコードします。

SYSCAT.BUFFERPOOLNODES

ノードに対するバッファー・プールのサイズが、 SYSCAT.BUFFERPOOLS の列 NPAGES のデフォルトのサイズと異なるバッファー・プール内のノードごとに、 1 つの行が入っています。

表 43. SYSCAT.BUFFERPOOLNODES カタログ視点

列名	データ・タイプ	ヌル値可	説明
BUFFERPOOLID	INTEGER		内部バッファー・プール識別子
NODENUM	SMALLINT		ノード番号
NPAGES	INTEGER		このノードに対するバッファー・プールのページ数

SYSCAT.BUFFERPOOLS

ノードグループ内のバッファー・プールごとに 1 つの行が入っています。

表 44. SYSCAT.BUFFERPOOLS カタログ視点

列名	データ・タイプ	ヌル値可	説明
BPNAME	VARCHAR(18)		バッファー・プールの名前
BUFFERPOOLID	INTEGER		内部バッファー・プール識別子
NGNAME	VARCHAR(18)	Yes	ノードグループ名 (そのバッファー・プールがデータベース内のすべてのノードに対して存在する場合は NULL)
NPAGES	INTEGER		バッファー・プールのページ数
PAGESIZE	INTEGER		このバッファー・プールのページ・サイズ
ESTORE	CHAR(1)		N = このバッファー・プールは拡張記憶域を使用しない。 Y = このバッファー・プールは拡張記憶域を使用する。

SYSCAT.CASTFUNCTIONS

cast 関数ごとに 1 つの行が含まれています。組み込み cast 関数は含まれません。

表 45. SYSCAT.CASTFUNCTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FROM_TYPESCHEMA	VARCHAR(128)		パラメーターのデータ・タイプの修飾名。
FROM_TYPENAME	VARCHAR(18)		
TO_TYPESCHEMA	VARCHAR(128)		キャスト後の結果のデータ・タイプを示す修飾名。
TO_TYPENAME	VARCHAR(18)		
FUNCSCHEMA	VARCHAR(128)		関数の修飾名。
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		関数インスタンスの名前。
ASSIGN_FUNCTION	CHAR(1)		Y = 暗黙的な割り当て関数 N = 割り当て関数ではない

SYSCAT.CHECKS

検査制約ごとに 1 つの行が含まれています。

表 46. SYSCAT.CHECKS カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		検査制約の名前 (表内で固有)。
DEFINER	VARCHAR(128)		その検査制約を定義した許可 ID。
TABSCHEMA	VARCHAR(128)		この制約が適用される表の修飾名。
TABNAME	VARCHAR(128)		
CREATE_TIME	TIMESTAMP		制約が定義された時刻。この制約で使用される関数の解決に使用されます。制約の定義後に作成された関数は選択されません。
QUALIFIER	VARCHAR(128)		オブジェクト定義時のデフォルト・スキーマの値。 非修飾参照を完了するために使用します。
TYPE	CHAR(1)		検査制約のタイプ: A = GENERATED ALWAYS 列のシステム生成検査制約 C = 検査制約
FUNC_PATH	VARCHAR(254)		制約作成時に使用された現行 SQL パス。
TEXT	CLOB(64K)		CHECK 文節のテキスト。

SYSCAT.COLAUTH

列レベルの特権が与えられているユーザーまたはグループごとに 1 つまたは複数の行が入っており、特権のタイプとその特権が付与可能か否かを示します。

表 47. SYSCAT.COLAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を付与したユーザーの許可 ID、または SYSIBM。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
TABSCHEMA	VARCHAR(128)		表または視点の修飾名。
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		この特権が適用される列の名前。
COLNO	SMALLINT		表または視点の中のこの列の番号。
PRIVTYPE	CHAR(1)		表または視点に対する特権のタイプを次のように示します。 U = 更新特権 R = 参照特権
GRANTABLE	CHAR(1)		特権が付与可能か否かを示します。 G = 付与可能 N = 付与不能

SYSCAT.COLCHECKS

各行は、検査制約によって参照される列を表します。

表 48. SYSCAT.COLCHECKS カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		検査制約の名前。 (表内で固有。システム生成の名前も含む。)
TABSCHEMA	VARCHAR(128)		参照される列を含む表の修飾名。
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		列の名前。
USAGE	CHAR(1)		R = 列は検査制約内で参照されます。 S = 列は生成される列をサポートするシステム生成検査制約のソース列です。 T = 列は生成される列をサポートするシステム生成検査制約のターゲット列です。

SYSCAT.COLDIST

最適化プログラムで使用される列の詳細な統計値が入れられます。各行は、列の N 番目の最大頻出値を記述しています。

表 49. SYSCAT.COLDIST カタログ視点

列名	データ・タイプ	ヌル値可	説明
TABSCHEMA	VARCHAR(128)		この項目が適用される表の修飾名。
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		この項目が適用される列の名前。
TYPE	CHAR(1)		F = 頻度 (最大頻出値) Q = 変位値
SEQNO	SMALLINT		• TYPE=F の場合、この列の N は第 N 頻出値。 • TYPE=Q の場合、この列の N は第 N 変位値。
COLVALUE	VARCHAR(254)	Yes	データ値 (文字リテラルまたはヌル値)。
VALCOUNT	BIGINT		• TYPE=F の場合、VALCOUNT は、その列の中の COLVALUE の出現回数。 • TYPE=Q の場合、VALCOUNT は、値が COLVALUE 以下の行の数。
DISTCOUNT	BIGINT	Yes	TYPE=Q の場合、この列は COLVALUE 以下の値の種類数 (入手不能の場合はヌル) を記録します。

SYSCAT.COLOPTIONS

各行には、列固有のオプション値が含まれます。

表 50. SYSCAT.COLOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TABSCHEMA	VARCHAR(128)		ニックネームの修飾子。
TABNAME	VARCHAR(128)		列のニックネーム。
COLNAME	VARCHAR(128)		ローカル列名。
OPTION	VARCHAR(128)		列オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.COLUMNS

表または視点に定義されている列 (継承された列の中で適用できるものを含む) ごとに 1 行が含まれています。すべてのカタログ視点は、SYSCAT.COLUMNS 表に項目を持っています。

表 51. SYSCAT.COLUMNS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TABSCHEMA	VARCHAR(128)		この列を含む表または視点の修飾名。
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		列名。
COLNO	SMALLINT		表または視点におけるその列の位置を示す番号。番号は、0 から始まります。
TYPESCHEMA	VARCHAR(128)		列のデータ・タイプが特殊タイプの場合、その修飾名。それ以外の場合、TYPESCHEMA の値は SYSIBM、 TYPENAME はその列のデータ・タイプ (CHARACTER などの長い形式で) になります。 FLOAT または *n* が 24 より大きい FLOAT(*n*) が指定された場合、TYPENAME は DOUBLE に名前変更されます。 *n* が 25 より小さい FLOAT(*n*) が指定された場合は、 TYPENAME は REAL に名前変更されます。また、NUMERIC は DECIMAL に名前変更されます。
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		データの最大長。特殊タイプの場合は 0。LENGTH 列は、DECIMAL フィールドに対する精度を示します。
SCALE	SMALLINT		DECIMAL フィールドの位取り。 DECIMAL 以外の場合は 0 になります。

表 51. SYSCAT.COLUMNS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
DEFAULT	VARCHAR(254)	Yes	列のデータ・タイプに対応する定数、特殊レジスター、またはキャスト関数によって表された表の列に対するデフォルト値。キーワード NULL の場合もあります。 値は、デフォルト値として指定された値から変換されることがあります。たとえば、日時の定数は ISO 形式で表示され、キャスト機能名はスキーマ名で修飾され、識別子は区切られています (注 3 を参照)。 DEFAULT 文節が指定されていない場合、または列が視点の列の場合は、ヌル値になります。
NULLS	CHAR(1)		Y = 列はヌル値可能。 N = 列はヌル値不可。 式または関数から入手した視点の列の場合、値は N である可能性があります。この場合でも、この視点を使用するステートメントが処理され、算術計算エラーの警告を出された場合は、この列にヌル値が入ります。 注 1 を参照。
CODEPAGE	SMALLINT		列のコード・ページ。 FOR BIT DATA 属性を指定して定義されていない文字ストリング列の場合、値はデータベース・コード・ページです。漢字ストリング列の場合、値は (複合) データベース・コード・ページによる暗黙の DBCS コード・ページです。それ以外の場合の値は 0 です。
LOGGED	CHAR(1)		LOB タイプまたは LOB に基づく特殊タイプの列だけに適用されます (それ以外はブランク)。 Y = 列のログ記録を取る。 N = 列のログ記録を取らない。

表 51. SYSCAT.COLUMNS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
COMPACT	CHAR(1)		LOB タイプまたは LOB に基づく特殊タイプの列だけに適用されます (それ以外はブランク)。 Y = 記憶域内で列を圧縮する。 N = 列を圧縮しない。
COLCARD	BIGINT		列の値の種類数。統計が収集されていない場合は -1。継承列および階層表の列の場合は -2。
HIGH2KEY	VARCHAR(254)	Yes	列の 2 番目の最高値。統計が収集されておらず、継承列および階層表の列の場合、このフィールドは空です。注 2 を参照。
LOW2KEY	VARCHAR(254)	Yes	列の 2 番目の最低値。統計が収集されておらず、継承列および階層表の列の場合、このフィールドは空です。注 2 を参照。
AVGCOLLEN	INTEGER		列の平均の長さ。 長形式フィールドまたは LOB の場合、または統計が収集されていない場合は -1。継承列および階層表の列の場合は -2。
KEYSEQ	SMALLINT	Yes	表の基本キー内の列の位置番号。副表および階層表の場合、このフィールドはヌル値です。
PARTKEYSEQ	SMALLINT	Yes	表の区分化キー内の列の位置番号。列が区分化キーの一部でない場合、このフィールドはヌル値または 0。副表および階層表の場合も、このフィールドはヌル値です。
NQUANTILES	SMALLINT		この列の SYSCAT.COLDIST に記録された変位値の数。統計がない場合は -1。継承列および階層表の列の場合は -2。
NMOSTFREQ	SMALLINT		この列の SYSCAT.COLDIST に記録された最大頻出値の数。統計がない場合は -1。継承列および階層表の列の場合は -2。
NUMNULLS	BIGINT		列のヌル値の数が含まれています。統計が収集されていない場合は -1。
TARGET_TYPESCHEMA	VARCHAR(128)	Yes	列のタイプが REFERENCE の場合、ターゲット・タイプの修飾名。列のタイプが REFERENCE でない場合は、ヌル値。
TARGET_TYPENAME	VARCHAR(18)	Yes	

表 51. SYSCAT.COLUMNS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
SCOPE_TABSCHEMA	VARCHAR(128)	Yes	列のタイプが REFERENCE の場合、効力範囲 (ターゲット表) の修飾名。列のタイプが REFERENCE でない場合や、効力範囲が定義されていない場合は、ヌル値。
SCOPE_TABNAME	VARCHAR(128)	Yes	
SOURCE_TABSCHEMA	VARCHAR(128)		列が使われたそれぞれの表階層における、表または視点の修飾名。非継承列の場合、値は TBCREATOR および TBNAME と同じです。非タイプ付き表および視点の場合はヌル値。
SOURCE_TABNAME	VARCHAR(128)		
DL_FEATURES	CHAR(10)	Yes	DATALINK タイプ列だけに適用されます。それ以外の場合、ヌル値です。各文字の位置は、次のようにして定義されます。 1. リンク・タイプ (URL の場合は U) 2. リンク制御 (ファイルの場合は F、ファイルでない場合は N) 3. 保全性 (すべての場合は A、なしの場合は N) 4. 読み取り許可 (ファイル・システムの場合は F、データベースの場合は D) 5. 書き込み許可 (ファイル・システムの場合は F、ブロック済みの場合は B) 6. 回復 (する場合は Y、しない場合は N) 7. リンク解除時の処理 (復元の場合は R、削除の場合は D、適用しない場合は N) 文字 8 ～ 10 は、将来の使用のために予約されています。
SPECIAL_PROPS	CHAR(8)	Yes	REFERENCE タイプ列だけに適用されます。それ以外の場合、ヌル値です。各文字の位置は、次のようにして定義されます。 オブジェクト識別子 (OID) 列 (yes の場合は Y、no の場合は N) ユーザー生成またはシステム生成 (ユーザーの場合は U、システムの場合は S)
HIDDEN	CHAR(1)		隠し列のタイプ S = システムに管理される隠し列 列が隠されていない場合はブランク

表 51. SYSCAT.COLUMNS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
INLINE_LENGTH	INTEGER		基礎表の行で保持できる構造タイプの列の長さ。 ALTER/CREATE TABLE ステートメントで明示的に設定された値がない場合は 0 です。
IDENTITY	CHAR(1)		'Y' はその列が識別列であることを示し、'N' はその列が識別列ではないことを示しています。
GENERATED	CHAR(1)		生成される列のタイプ A = 列の値が常に生成される D = 列の値がデフォルトで生成される 列が生成されない場合はブランク
TEXT	CLOB(64K)		キーワード AS で始まる、生成される列のテキストが入ります。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメント。

注:

1. バージョン 2 以降では、値 D (デフォルト値がヌルでないことを示す) は使用されなくなりました。代わりに、WITH DEFAULT の使用は、 DEFAULT 欄の値がヌル以外であることによって示されます。
2. バージョン 2 以降では、数値データの表現が文字リテラルに変更されました。サイズが 16 バイトから 33 バイトに拡大されました。
3. バージョン 2.1.0 では cast-function 名は区切られていなかったため、 DEFAULT 列にこのように表示される場合があります。また、視点列にはデフォルト値が含まれていたため、これも DEFAULT 列に表示されます。

SYSCAT.CONSTDEP

他の特定のオブジェクトへの制約の従属関係ごとに 1 つの行が含まれています。

表 52. SYSCAT.CONSTDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		制約の名前。
TABSCHEMA	VARCHAR(128)		制約が適用される表の修飾名。
TABNAME	VARCHAR(128)		
BTYPE	CHAR(1)		この制約が従属しているオブジェクトのタイプ。可能な値: F = 関数インスタンス I = 索引インスタンス R = 構造タイプ
BSCHEMA	VARCHAR(128)		この制約が依存するオブジェクトの修飾名。
BNAME	VARCHAR(18)		

SYSCAT.DATATYPES

組み込みタイプおよびユーザー定義タイプを含むすべてのデータ・タイプごとに、 1 つの行が含まれています。

表 53. SYSCAT.DATATYPES カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPESCHEMA	VARCHAR(128)		データ・タイプの修飾名 (組み込みタイプの場合、TYPESCHEMA は SYSIBM です)。
TYPENAME	VARCHAR(18)		
DEFINER	VARCHAR(128)		タイプ作成時の許可 ID。
SOURCESCHEMA	VARCHAR(128)	Yes	特殊タイプのソース・タイプの修飾名。参照タイプとして使用される組み込みタイプのうち、その参照タイプが構造タイプへ参照するための表示として使用されているものの修飾名。その他の場合はヌル値。
SOURCENAME	VARCHAR(18)	Yes	
METATYPE	CHAR(1)		S = システムで定義済みのタイプ T = 特殊タイプ R = 構造タイプ
TYPEID	SMALLINT		システムで生成された、データ・タイプの内部識別子。
SOURCETYPEID	SMALLINT	Yes	ソース・タイプの内部タイプ ID (組み込みタイプの場合はヌル値)。ユーザー定義構造タイプの場合、これは参照表示タイプの内部タイプ ID になります。
LENGTH	INTEGER		タイプの最大長。システムで定義済みのパラメーター化タイプ (DECIMAL や VARCHAR など) の場合は 0。ユーザー定義構造タイプの場合、これは参照表示タイプの長さを示します。
SCALE	SMALLINT		システムで定義済みの DECIMAL タイプに基づく特殊タイプまたは参照表示タイプの位取り。他のすべてのタイプの場合は 0 (DECIMAL 自体を含む)。ユーザー定義構造タイプの場合、これは参照表示タイプの長さを示します。
CODEPAGE	SMALLINT		文字特殊タイプおよび漢字特殊タイプ、または参照表示タイプの場合はコード・ページ。それ以外の場合は 0。
CREATE_TIME	TIMESTAMP		データ・タイプの作成時刻。
ATTRCOUNT	SMALLINT		データ・タイプ内の属性の数。

表 53. SYSCAT.DATATYPES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
INSTANTIABLE	CHAR(1)		Y = タイプをインスタンス化できる。 N = タイプをインスタンス化できない。
WITH_FUNC_ACCESS	CHAR(1)		Y = 関数表記を使ってこのタイプ用のメソッドすべてを呼び出せる。 N = 関数表記を使ってこのタイプ用のメソッドすべてを呼び出せない。
FINAL	CHAR(1)		Y = ユーザー定義タイプにサブタイプを指定できない。 N = ユーザー定義タイプにサブタイプを指定できる。
INLINE_LENGTH	INTEGER		基礎表の行で保持できる構造タイプの長さ。CREATE TYPE ステートメントで明示的に設定された値がない場合は 0 です。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.DBAUTH

ユーザーが持つデータベース権限を記録します。

表 54. SYSCAT.DBAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		SYSIBM または特権を授与したユーザーの許可 ID。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
DBADMAUTH	CHAR(1)		GRANTEE がデータベースに対する DBADM 権限を持っているか否か。 Y = 権限を持っている。 N = 権限を持っていない。
CREATETABAUTH	CHAR(1)		GRANTEE がデータベースに表を作成できるか否か (CREATETAB)。 Y = 特権がある。 N = 特権がない。
BINDADDAUTH	CHAR(1)		GRANTEE がデータベース中に新しいパッケージを作成できるか否か (BINDADD)。 Y = 特権がある。 N = 特権がない。
CONNECTAUTH	CHAR(1)		GRANTEE がデータベースに接続できるか否か (CONNECT)。 Y = 特権がある。 N = 特権がない。
NOFENCEAUTH	CHAR(1)		GRANTEE に非分離関数を作成する特権があるか否か。 Y = 特権がある。 N = 特権がない。
IMPLSCHEMAAUTH	CHAR(1)		GRANTEE がデータベース内に暗黙的にスキーマを作成できるか否か (IMPLICIT_SCHEMA)。 Y = 特権がある。 N = 特権がない。

表 54. SYSCAT.DBAUTH カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
LOADAUTH	CHAR(1)		GRANTEE がデータベースに対する LOAD 権限を持っているか否か。 Y = 権限を持っている。 N = 権限を持っていない。

SYSCAT.EVENTMONITORS

定義されているイベント・モニターごとに 1 つの行が含まれます。

列名	データ・タイプ	ヌル値可	説明
EVMONNAME	VARCHAR(18)		イベント・モニターの名前。
DEFINER	VARCHAR(128)		イベント・モニターの定義者の許可 ID。
TARGET_TYPE	CHAR(1)		イベント・データの書き込み先 (ターゲット) のタイプ。値: F = ファイル P = パイプ
TARGET	VARCHAR(246)		イベント・データの書き込み先 (ターゲット) の名前。ファイルの絶対パス名、またはパイプの絶対名。
MAXFILES	INTEGER	Yes	このイベント・モニターの 1 つのイベント・パスで許されるイベント・ファイルの最大数。最大値がない場合、または書き込み先が FILE でない場合はヌル値。
MAXFILESIZE	INTEGER	Yes	各イベント・ファイルの最大サイズ (4K ページ単位)。このサイズに達すると、イベント・モニターは新しいファイルを作成します。最大値がない場合、または書き込み先が FILE でない場合はヌル値。
BUFFERSIZE	INTEGER	Yes	ファイルに出力する場合、イベント・モニターの使用するバッファー・サイズ (4K ページ単位)。それ以外の場合はヌル値。
IO_MODE	CHAR(1)	Yes	ファイル入出力のモード。 B = ブロック化 N = 非ブロック化 宛先タイプが FILE でない場合はヌル値。
WRITE_MODE	CHAR(1)	Yes	このイベント・モニターの起動時に、モニターが既存のイベント・データを処理する方法。値: A = 追加 R = 置換 宛先タイプが FILE でない場合はヌル値。

SYSCAT.EVENTMONITORS

列名	データ・タイプ	ヌル値可	説明
AUTOSTART	CHAR(1)		データベース開始時にイベント・モニターが自動的に活動化されるか否か。 Y = はい N = いいえ
NODENUM	SMALLINT		イベント・モニターが稼働してイベントをログ記録する区分 (またはノード) の番号。
MONSCOPE	CHAR(1)		モニターの効力範囲: L = ローカル G = グローバル
REMARKS	VARCHAR(254)	Yes	将来の使用のために予約済み。

SYSCAT.EVENTS

モニターするイベントごとに 1 つの行が含まれます。一般に、1 つのイベント・モニターは複数のイベントをモニターします。

表 55. SYSCAT.EVENTS カタログ視点

列名	データ・タイプ	ヌル値可	説明
EVMONNAME	VARCHAR(18)		このイベントをモニターするイベント・モニターの名前。
TYPE	VARCHAR(18)		モニターされるイベントのタイプ。可能な値: DATABASE CONNECTIONS TABLES STATEMENTS TRANSACTIONS DEADLOCKS TABLESPACES
FILTER	CLOB(32K)	Yes	このイベントに適用される WHERE 文節のテキスト全体。

SYSCAT.FULLHIERARCHIES

各行は、副表とスーパー表、サブタイプとスーパータイプ、または副視点とスーパー視点の関係を表しています。この視点には、直接の関係をはじめとする、すべての階層関係が含まれています。

表 56. SYSCAT.FULLHIERARCHIES カタログ視点

列名	データ・タイプ	ヌル値可	説明
METATYPE	CHAR(1)		次のように関係のタイプをエンコードします。 R = 構造タイプの相互関係 U = タイプ付き表の相互関係 W = タイプ付き視点の相互関係
SUB_SCHEMA	VARCHAR(128)		サブタイプ、副表、または副視点の修飾名。
SUB_NAME	VARCHAR(128)		
SUPER_SCHEMA	VARCHAR(128)		スーパータイプ、スーパー表、またはスーパー視点の修飾名。
SUPER_NAME	VARCHAR(128)		
ROOT_SCHEMA	VARCHAR(128)		階層のルートにある表、視点、またはタイプの修飾名。
ROOT_NAME	VARCHAR(128)		

SYSCAT.FUNCDEP

各行は、何らかの他のオブジェクトに対する関数またはメソッドの従属関係を表します。

表 57. SYSCAT.FUNCDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCSCHEMA	VARCHAR(128)		別のオブジェクトに従属する関数の修飾名またはメソッドの名前。
FUNCNAME	VARCHAR(18)		
BTYPE	CHAR(1)		関数またはメソッドが従属しているオブジェクトのタイプ。 A = 別名 F = 関数インスタンスまたはメソッド・インスタンス O = 表または視点階層内のすべての副表または副視点に対する特権の従属関係 R = 構造タイプ S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点 X = 索引の拡張
BSCHEMA	VARCHAR(128)		関数またはメソッドが従属しているオブジェクトの修飾名 (BTYEPE='F' の場合、これは関数の固有の名前になります)。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE = O、S、T、U、V、または W の場合、従属関数または従属メソッドで必要な表または視点に対する特権を示すコード。それ以外の場合はヌル値。

SYSCAT.FUNCMAPOPTIONS

各行には、関数マッピングのオプション値が含まれています。

表 58. SYSCAT.FUNCMAPOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCTION_MAPPING	VARCHAR(18)		関数マッピングの名前。
OPTION	VARCHAR(128)		関数マッピングのオプション名。
SETTING	VARCHAR(255)		値。

SYSCAT.FUNCMAPPARMOPTIONS

各行には、関数マッピングのパラメーター・オプションの値が含まれています。

表 59. SYSCAT.FUNCMAPPARMOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCTION_MAPPING	VARCHAR(18)		関数マッピングの名前。
ORDINAL	SMALLINT		パラメーターの位置。
LOCATION	CHAR(1)		L = ローカル R = リモート
OPTION	VARCHAR(128)		関数マッピングのパラメーター・オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.FUNCMAPPINGS

各行には関数マッピングが含まれています。

表 60. SYSCAT.FUNCMAPPINGS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCTION_MAPPING	VARCHAR(18)		関数マッピングの名前 (システム生成の場合もある)。
FUNCSCHEMA	VARCHAR(128)	Yes	関数のスキーマ。システム組み込み関数の場合はヌル値。
FUNCNAME	VARCHAR(1024)	Yes	ローカル関数 (組み込みまたはユーザー定義) の名前。
FUNCID	INTEGER	Yes	内部的に割り当てられた識別子。
SPECIFICNAME	VARCHAR(18)	Yes	ローカル関数インスタンスの名前。
DEFINER	VARCHAR(128)		このマッピングの作成に使用された許可 ID。
WRAPNAME	VARCHAR(128)	Yes	マッピングが適用されるラッパー名。
SERVERNAME	VARCHAR(128)	Yes	データ・ソースの名前。
SERVERTYPE	VARCHAR(30)	Yes	マッピングが適用されるデータ・ソースのタイプ。
SERVERVERSION	VARCHAR(18)	Yes	マッピングが適用されるサーバー・タイプのバージョン。
CREATE_TIME	TIMESTAMP	Yes	マッピングが作成された時刻。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.FUNCPARMS

SYSCAT.FUNCTIONS に定義されている関数またはメソッドの、パラメーターまたは結果ごとに、1 つの行が入れられます。

表 61. SYSCAT.FUNCPARMS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCSCHEMA	VARCHAR(128)		関数の修飾名。
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		関数インスタンスの名前 (システム生成の名前の場合もある)。
ROWTYPE	CHAR(1)		P = パラメーター R = キャスト前の結果 C = キャスト後の結果
ORDINAL	SMALLINT		ROWTYPE=P の場合、関数シグニチャー中のパラメーターの位置番号。 ROWTYPE= R であり、なおかつ関数が表を戻す場合、その結果表内の列の位置番号。それ以外の場合は 0。
PARMNAME	VARCHAR(128)		パラメーターまたは結果の列の名前、あるいは名前が存在しない場合はヌル値。
TYPESCHEMA	VARCHAR(128)		パラメーターまたは結果のデータ・タイプの修飾名。
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		パラメーターまたは結果の長さ。パラメーターまたは結果が特殊タイプの場合は 0。注 1 を参照。
SCALE	SMALLINT		パラメーターまたは結果の位取り。パラメーターまたは結果が特殊タイプの場合は 0。注 1 を参照。
CODEPAGE	SMALLINT		パラメーターのコード・ページ。該当しない場合、または FOR BIT DATA 属性を指定して宣言された文字データの列の場合は 0。
CAST_FUNCID	INTEGER	Yes	内部関数の ID。
AS_LOCATOR	CHAR(1)		Y = パラメーターまたは結果は、ロケーターの形式で渡される N = ロケーターの形式では渡されない

表 61. SYSCAT.FUNCPARMS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
TARGET_TYPESCHEMA	VARCHAR(128)		パラメーターまたは結果のタイプが REFERENCE の場合、ターゲット・タイプの修飾名。パラメーターまたは結果のタイプが REFERENCE でない場合は、ヌル値。
TARGET_TYPENAME	VARCHAR(18)		
SCOPE_TABSCHEMA	VARCHAR(128)		パラメーターまたは結果のタイプが REFERENCE の場合、効力範囲 (ターゲット表) の修飾名。パラメーターまたは結果のタイプが REFERENCE でない場合や、効力範囲が定義されていない場合は、ヌル値。
SCOPE_TABNAME	VARCHAR(128)		
TRANSFORM_GRPNAME	VARCHAR(18)	Yes	構造タイプ関数のパラメーターの場合は、変換グループの名前。

注:

1. ソース関数からの派生関数 (他の関数を参照して定義された関数) はソースのパラメーターの長さと位取りを継承するので、そのような関数の LENGTH と SCALE は 0 に設定されます。

SYSCAT.FUNCTIONS

ユーザー定義関数 (スカラー、表、またはソース)、システム生成メソッド、またはユーザー定義メソッドごとに、1 つの行が含まれます。組み込み関数は含まれません。

注: 特に説明がない限り、「関数」に関する説明はメソッドにも適用されます。

表 62. SYSCAT.FUNCTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCSCHEMA	VARCHAR(128)		関数の修飾名。
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		関数インスタンスの名前 (システム生成の名前の場合もある)。
DEFINER	VARCHAR(128)		関数定義者の許可 ID。
FUNCID	INTEGER		内部割り当てによる関数 ID。
RETURN_TYPE	SMALLINT		関数の戻りタイプの内部タイプ・コード。
ORIGIN	CHAR(1)		B = 組み込み E = ユーザー定義、外部 Q = ユーザー定義、SQL U = ユーザー定義、ソース関数に基づく S = システム生成
TYPE	CHAR(1)		C = 列関数 R = 行関数 S = スカラー関数 T = 表関数
METHOD	CHAR(1)		Y = メソッド N = メソッドではない
EFFECT	CHAR(2)		MU = ミューテーター・メソッド OB = オブザーバー・メソッド CN = コンストラクター・メソッド ブランク = システム生成メソッドではない
PARM_COUNT	SMALLINT		関数のパラメーターの数。

表 62. SYSCAT.FUNCTIONS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
PARM_SIGNATURE	VARCHAR(180) FOR BIT DATA		内部形式での 90 までのパラメーター・タイプを連結した値。関数にパラメーターがない場合は、長さ 0。
CREATE_TIME	TIMESTAMP		関数作成時のタイム・スタンプ。バージョン 1 の関数の場合は 0 に設定されます。
QUALIFIER	VARCHAR(128)		オブジェクト定義時のデフォルト・スキーマの値。
WITH_FUNC_ACCESS	CHAR(1)		Y = このメソッドは関数表記を使用して呼び出すことができます。 N = このメソッドは関数表記を使用して呼び出すことはできません。
TYPE_PRESERVING	CHAR(1)		Y = 戻りタイプは ″type-preserving″ パラメーターで管理される。システム生成のミューテーター・メソッドはすべてタイプ保持です。 N = 戻りタイプはメソッドの宣言された戻りタイプ。
VARIANT	CHAR(1)		Y = 可変 (結果は毎回同じとは限らない) N = 不変 (結果は一貫している) ORIGIN が E でない場合はブランク
SIDE_EFFECTS	CHAR(1)		E = 関数に外部副次作用がある (呼び出しの数が重要) N = 副次作用がない ORIGIN が E でない場合はブランク
FENCED	CHAR(1)		Y = 分離 N = 非分離 ORIGIN が E でない場合はブランク
NULLCALL	CHAR(1)		Y = CALLED ON NULL INPUT N = RETURNS NULL ON NULL INPUT (オペランドにヌル値が含まれる場合、暗黙のうちに関数結果はヌル値) ORIGIN が E でない場合はブランク

表 62. SYSCAT.FUNCTIONS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
CAST_FUNCTION	CHAR(1)		Y = キャスト関数 N = キャスト関数ではない
ASSIGN_FUNCTION	CHAR(1)		Y = 暗黙的な割り当て関数 N = 割り当て関数ではない
SCRATCHPAD	CHAR(1)		Y = スクラッチ・パッドあり N = スクラッチ・パッドなし ORIGIN が E でない場合はブランク
FINAL_CALL	CHAR(1)		Y = ステートメント終了の実行時に、この関数に対して最終呼び出しが行われる N = 最終呼び出しを行わない ORIGIN が E でない場合はブランク
PARALLELIZABLE	CHAR(1)		Y = 関数を並列して実行できる N = 関数を並列して実行できない ORIGIN が E でない場合はブランク
CONTAINS_SQL	CHAR(1)		関数またはメソッドが SQL を含むかどうかを示します。 C = CONTAINS SQL: SQL データの読み取りまたは変更を行わない SQL に限り許可されている。 N = NO SQL: SQL は許可されていない。 R = READS SQL DATA: SQL データを読み取る SQL に限り許可されている。
DBINFO	CHAR(1)		DBINFO パラメーターが外部関数に渡されるか否かを示します。 Y = DBINFO は渡される N = DBINFO は渡されない ORIGIN が E でない場合はブランク
RESULT_COLS	SMALLINT		表関数 (TYPE=T) の場合は結果表の列数。その他の場合は 1。
LANGUAGE	CHAR(8)		関数本体の実装言語。可能な値は C、JAVA、OLE、または OLEDB。 ORIGIN が E または Q でない場合はブランク。

表 62. SYSCAT.FUNCTIONS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
IMPLEMENTATION	VARCHAR(254)	Yes	ORIGIN=E の場合、この関数を実現するためのパス / モジュール / 関数。 ORIGIN=U、かつソース関数が組み込み関数の場合、ソース関数の名前とシグニチャー。それ以外の場合、ヌル値です。
CLASS	VARCHAR(128)	Yes	LANGUAGE=JAVA の場合、この関数を実装するクラス。それ以外の場合、ヌル値です。
JAR_ID	VARCHAR(128)	Yes	LANGUAGE=JAVA の場合、この関数を実装する jar ファイル。それ以外の場合、ヌル値です。
PARM_STYLE	CHAR(8)		CREATE FUNCTION ステートメントで宣言されたパラメーターのスタイル。値: DB2SQL DB2GENRL JAVA ORIGIN が E でない場合はブランク
SOURCE_SCHEMA	VARCHAR(128)	Yes	ORIGIN=U かつソース関数がユーザー定義関数の場合、ソース関数の修飾名。 ORIGIN=U で、しかもソース関数が組み込み関数の場合、SOURCE_SCHEMA は 'SYSIBM'、SOURCE_SPECIFIC は '組み込み関数の場合は N/A'。 ORIGIN が U でない場合は、ヌル値。
SOURCE_SPECIFIC	VARCHAR(18)	Yes	
IOS_PER_INVOC	DOUBLE		呼び出しごとの入出力回数の見積もり。不明の場合は -1 (デフォルト値は 0)。
INSTS_PER_INVOC	DOUBLE		呼び出しごとの命令の数の見積もり。不明の場合は -1 (デフォルト値は 450)。
IOS_PER_ARGBYTE	DOUBLE		入力引き数 1 バイトごとの入出力回数の見積もり。不明の場合は -1 (デフォルト値は 0)。
INSTS_PER_ARGBYTE	DOUBLE		入力引き数 1 バイトごとの命令数の見積もり。不明の場合は -1 (デフォルト値は 0)。
PERCENT_ARGBYTES	SMALLINT		関数が実際に読み取る入力引き数バイトの平均パーセント値の見積もり。不明の場合は -1 (デフォルト値は 100)。
INITIAL_IOS	DOUBLE		関数が最初 / 最後に呼び出されたときに実行される入出力の回数の見積もり。不明の場合は -1 (デフォルト値は 0)。

表 62. SYSCAT.FUNCTIONS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
INITIAL_INSTS	DOUBLE		最初 / 最後の関数呼び出し時に実行される命令の数の見積もり。不明の場合は -1 (デフォルト値は 0)。
CARDINALITY	BIGINT		予想される表関数のカーディナリティー。不明の場合、または関数が表関数でない場合は -1。
IMPLEMENTED	CHAR(1)		Y = 関数が実装されている。 M = メソッドが実装されているものの、関数アクセスはない。注 1 を参照。 H = メソッドが実装されており、関数アクセスがある。注 1 を参照。 N = 実装は行われずにメソッドが指定されている。
SELECTIVITY	DOUBLE		ユーザー定義述部に使用します。ユーザー定義述部がない場合は -1 です。注 2 を参照。
OVERRIDEN_FUNCID	INTEGER	Yes	将来の使用のために予約済み。
SUBJECT_TYPESCHEMA	VARCHAR(128)	Yes	ユーザー定義メソッドの対象となるタイプ・スキーマ。
SUBJECT_TYPENAME	VARCHAR(18)	Yes	ユーザー定義メソッドの対象となるタイプ名。
FUNC_PATH	VARCHAR(254)	Yes	関数の定義された時点の関数パス。
BODY	CLOB(1M)	Yes	言語が SQL の場合、CREATE FUNCTION または CREATE METHOD ステートメントのテキスト。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

注:

1. この値は、DB2 の将来のバージョンでは使用されない可能性があります。
2. どのようなユーザー定義関数でも、パック記述子およびシステム・カタログでの移行中は、この列は -1 に設定されます。ユーザー定義述部の場合、システム・カタログでの選択性は -1 になります。この場合、最適化プログラムが使用する選択性の値は 0.01 です。

SYSCAT.HIERARCHIES

各行は、副表とすぐ上のスーパー表、サブタイプとすぐ上のスーパータイプ、または副視点とすぐ上のスーパー視点の関係を表しています。この視点には、直接の階層関係しか含まれていません。

表 63. SYSCAT.HIERARCHIES カタログ視点

列名	データ・タイプ	ヌル値可	説明
METATYPE	CHAR(1)		次のように関係のタイプをエンコードします。 R = 構造タイプの相互関係 U = タイプ付き表の相互関係 W = タイプ付き視点の相互関係
SUB_SCHEMA	VARCHAR(128)		サブタイプ、副表、または副視点の修飾名。
SUB_NAME	VARCHAR(128)		
SUPER_SCHEMA	VARCHAR(128)		スーパータイプ、スーパー表、またはスーパー視点の修飾名。
SUPER_NAME	VARCHAR(128)		
ROOT_SCHEMA	VARCHAR(128)		階層のルートにある表、視点、またはタイプの修飾名。
ROOT_NAME	VARCHAR(128)		

SYSCAT.INDEXAUTH

索引に関する特権ごとに 1 つの行が含まれます。

表 64. SYSCAT.INDEXAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を与えたユーザーの許可 ID。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
INDSCHEMA	VARCHAR(128)		索引の名前。
INDNAME	VARCHAR(18)		
CONTROLAUTH	CHAR(1)		GRANTEE に索引に対する CONTROL 特権があるか否か。 Y = 特権がある。 N = 特権がない。

SYSCAT.INDEXCOLUSE

索引に関与するすべての列をリストします。

表 65. SYSCAT.INDEXCOLUSE カタログ視点

列名	データ・タイプ	ヌル値可	説明
INDSCHEMA	VARCHAR(128)		索引の修飾名。
INDNAME	VARCHAR(18)		
COLNAME	VARCHAR(128)		列の名前。
COLSEQ	SMALLINT		索引内の列の位置番号 (最初の位置 = 1)。
COLORDER	CHAR(1)		索引内のこの列にある値の順序。値: A = 昇順 D = 降順 I = INCLUDE 列 (順序は無視される)

SYSCAT.INDEXDEP

各行は、何らかの他のオブジェクトに対する索引の従属関係を表します。

表 66. SYSCAT.INDEXDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
INDSCHEMA	VARCHAR(128)		別のオブジェクトに従属する索引の修飾名。
INDNAME	VARCHAR(18)		
BTYPE	CHAR(1)		索引が従属しているオブジェクトのタイプ。 A = 別名 F = 関数インスタンス O = 表または視点階層内のすべての副表または副視点に対する特権の従属関係 R = 構造タイプ S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点 X = 索引の拡張
BSCHEMA	VARCHAR(128)		索引が従属しているオブジェクトの修飾名。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE = O、S、T、U、V、または W の場合、従属索引で必要な表または視点に対する特権を示すコード。それ以外の場合はヌル値。

SYSCAT.INDEXES

表に定義されている索引 (継承された列の中で適用できるものを含む) ごとに 1 行が含まれます。

表 67. SYSCAT.INDEXES カタログ視点

列名	データ・タイプ	ヌル値可	説明
INDSCHEMA	VARCHAR(128)		索引の名前。
INDNAME	VARCHAR(18)		
DEFINER	VARCHAR(128)		索引を作成したユーザー。
TABSCHEMA	VARCHAR(128)		その索引の定義されている表またはニックネームの修飾名。
TABNAME	VARCHAR(128)		
COLNAMES	VARCHAR (640)		列名の先頭に、昇順か降順かを示す + または - を付けたもののリスト。警告: この列は将来除去されます。詳しくは、1274ページの『SYSCAT.INDEXCOLUSE』を参照してください。
UNIQUERULE	CHAR(1)		固有値に関する規則: D = 重複可 P = 1 次索引 U = 固有項目のみ可
MADE_UNIQUE	CHAR(1)		Y = 索引は元は非固有だったが、固有キー制約または基本キー制約をサポートするために、固有索引に変換された。制約が除去されると、この索引は非固有に戻る。 N = 索引は作成時のまま。
COLCOUNT	SMALLINT		キー内の列数と組み込み列 (存在する場合) の数の合計。
UNIQUE_COLCOUNT	SMALLINT		固有キーに必要な列の数。常に <=COLCOUNT。組み込み列がある場合にのみ < COLCOUNT。索引に固有キーがない場合は -1 (重複可能)。
INDEXTYPE	CHAR(4)		索引のタイプ。 CLUS = クラスター化 REG = 通常

表 67. SYSCAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
ENTRYTYPE	CHAR(1)		H = 階層表の索引 L = タイプ付き表の論理索引 非タイプ付き表の索引の場合は、ブランク
PCTFREE	SMALLINT		索引を最初に作成する際に予約する索引葉ページのパーセンテージ。このスペースは、索引の作成後に行う挿入用に使用可能です。
IID	SMALLINT		索引の内部 ID。
NLEAF	INTEGER		葉ページの数。統計が収集されていない場合は -1。
NLEVELS	SMALLINT		索引レベルの数。統計が収集されていない場合は -1。
FIRSTKEYCARD	BIGINT		最初のキーの値の種類数。統計が収集されていない場合は -1。
FIRST2KEYCARD	BIGINT		索引の最初の 2 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。
FIRST3KEYCARD	BIGINT		索引の最初の 3 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。
FIRST4KEYCARD	BIGINT		索引の最初の 4 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。
FULLKEYCARD	BIGINT		キー全体の値の種類数。統計が収集されていない場合は -1。
CLUSTERRATIO	SMALLINT		索引によるデータ・クラスター化の程度。統計が収集されていない場合、または詳細な索引統計が収集されている場合は -1 (それらの場合は CLUSTERFACTOR のほうが使用されます)。
CLUSTERFACTOR	DOUBLE		クラスター化の程度の詳細測定値。詳細索引統計が収集されていない場合、またはニックネームの索引が定義されていない場合は -1。
SEQUENTIAL_PAGES	INTEGER		索引キーの順序でディスクに存在し、それらの間に大きなギャップがないか、わずかなギャップしかない葉ページの数。 (統計が入手できない場合は -1。)

表 67. SYSCAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
DENSITY	INTEGER		索引によって占有されているページの範囲内の、ページ数に対する SEQUENTIAL_PAGES の比率。パーセントで表現される。(0 ～ 100 の整数。統計が入手できない場合は -1。)
USER_DEFINED	SMALLINT		この索引がユーザー定義であって除去されていない場合は 1、それ以外の場合は 0。
SYSTEM_REQUIRED	SMALLINT		この索引が基本キー制約または固有キー制約に必要であるか、またはこの索引がタイプ付き表のオブジェクト識別子 (OID) 列上にある索引の場合は 1。 この索引が基本キー制約または固有キー制約に必要であり、しかもタイプ付き表のオブジェクト識別子 (OID) 列の索引である場合は 2。 それ以外の場合は 0。
CREATE_TIME	TIMESTAMP		索引の作成された時刻。
STATS_TIME	TIMESTAMP	Yes	この索引について記録されている統計値が最後に変更された時刻。統計が使用可能でない場合は、ヌル値。
PAGE_FETCH_PAIRS	VARCHAR(254)		文字形式で表される、整数の対のリスト。それぞれの対は、仮のバッファー内のページ数と、その仮のバッファーを使用した表の走査に必要なページ取り出しの回数を表しています。 (データが入手できない場合は、長さ 0 のストリング。)
MINPCTUSED	SMALLINT		ゼロでない場合は、オンライン索引再編成が使用可能になり、その値は、ページをマージをする前に使用される最小スペースのしきい値です。
REVERSE_SCANS	CHAR(1)		Y = 索引は逆走査をサポートする N = 索引は逆走査をサポートしない
INTERNAL_FORMAT	SMALLINT		索引の内部表記をエンコードします。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.INDEXOPTIONS

各行には、索引固有のオプション値が含まれます。

表 68. SYSCAT.INDEXOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
INDSCHEMA	VARCHAR(128)		索引のスキーマ名。
INDNAME	VARCHAR(18)		索引のローカル名。
OPTION	VARCHAR(128)		索引オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.KEYCOLUSE

固有キー、基本キー、または外部キーの制約によって定義されるキー (継承された基本キーまたは固有キーの中で適用できるものを含む) に関与する列をすべてリストします。

表 69. SYSCAT.KEYCOLUSE カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		制約の名前 (表内で固有)。
TABSCHEMA	VARCHAR(128)		列を含む表の修飾名。
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		列の名前。
COLSEQ	SMALLINT		キー内の列の位置番号 (最初の位置 = 1)。

SYSCAT.NAMEMAPPINGS

各行は、論理オブジェクトと、論理オブジェクトを実装するそれぞれの実装オブジェクトのマッピングを表します。

表 70. SYSCAT.NAMEMAPPINGS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPE	CHAR(1)		C = 列 I = 索引 U = タイプ付き表
LOGICAL_SCHEMA	VARCHAR(128)		論理オブジェクトの修飾名。
LOGICAL_NAME	VARCHAR(128)		
LOGICAL_COLNAME	VARCHAR(128)	Yes	TYPE = C の場合は、論理列の名前です。それ以外の場合はヌル値。
IMPL_SCHEMA	VARCHAR(128)		論理オブジェクトを実装する実装オブジェクトの修飾名。
IMPL_NAME	VARCHAR(128)		
IMPL_COLNAME	VARCHAR(128)	Yes	TYPE = C の場合は、実装列の名前です。それ以外の場合はヌル値。

SYSCAT.NODEGROUPDEF

ノードグループに含まれる区分ごとに 1 つの行が入っています。

表 71. SYSCAT.NODEGROUPDEF カタログ視点

列名	データ・タイプ	ヌル値可	説明
NGNAME	VARCHAR(18)		区分 (またはノード) を含むノードグループの名前。
NODENUM	SMALLINT		ノードグループに含まれる区分の区分 (またはノード) 番号。有効な区分番号は、0 ～ 999 (両端を含む) です。
IN_USE	CHAR(1)		区分 (またはノード) の状況。 A = 新しく追加された区分は区分化マップに入っていないが、ノードグループ内の表スペースのコンテナーが作成された。区分は、ノードグループ再配布 (Redistribute Nodegroup) 操作が正常に完了した場合に、区分化マップに追加されます。 D = 区分は、ノードグループ再配布操作の完了時に除去される。 T = 新しく追加された区分は、区分化マップに入っておらず、 WITHOUT TABLESPACES 文節を使用して追加された。このノードグループの表スペースに、コンテナーを明示的に追加する必要があります。 Y = 区分は区分化マップに入っている。

SYSCAT.NODEGROUPS

ノードグループごとに 1 つの行が含まれます。

表 72. SYSCAT.NODEGROUPS カタログ視点

列名	データ・タイプ	ヌル値可	説明
NGNAME	VARCHAR(18)		ノードグループの名前。
DEFINER	VARCHAR(128)		ノードグループの定義者の許可 ID。
PMAP_ID	SMALLINT		SYSCAT.PARTITIONMAPS 内の区分化マップの識別子。
REBALANCE_PMAP_ID	SMALLINT		再配布用に現在使用されている区分化マップの識別子。再配布が現在進行中でない場合は、値は -1。
CREATE_TIME	TIMESTAMP		ノードグループの作成時刻。
REMARKS	VARCHAR(254)	Yes	ユーザーが入力したコメント。

SYSCAT.PACKAGEAUTH

パッケージに関する特権ごとに 1 つの行が含まれています。

表 73. SYSCAT.PACKAGEAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を与えたユーザーの許可 ID。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
PKGSCHEMA	VARCHAR(128)		特権の対象となるパッケージの名前。
PKGNAME	CHAR(8)		
CONTROLAUTH	CHAR(1)		GRANTEE に、パッケージに対する CONTROL 特権があるか否か。 Y = 特権がある。 N = 特権がない。
BINDAUTH	CHAR(1)		GRANTEE に、パッケージに対する BIND 特権があるか否か。 Y = 特権がある。 N = 特権がない。
EXECUTEAUTH	CHAR(1)		GRANTEE に、パッケージに対する EXECUTE 特権があるか否か。 Y = 特権がある。 N = 特権がない。

SYSCAT.PACKAGEDEP

索引、表、視点、関数、別名、タイプ、および階層に対するパッケージの従属関係ごとに 1 つの行が含まれています。

表 74. SYSCAT.PACKAGEDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
PKGSCHEMA	VARCHAR(128)		パッケージの名前。
PKGNAME	CHAR(8)		
BINDER	VARCHAR(128)	Yes	パッケージをバインドしたユーザー。
BTYPE	CHAR(1)		オブジェクト BNAME のタイプ。 A = 別名 D = サーバー定義 F = 関数インスタンス I = 索引 M = 関数マッピング N = ニックネーム O = 表または視点階層内のすべての副表または副視点に対する特権の従属関係 P = ページ・サイズ R = 構造タイプ S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点
BSCHEMA	VARCHAR(128)		パッケージが従属しているオブジェクトの修飾名。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE が O、S、T、U、V または W の場合、このパッケージで必要な特権 (SELECT、INSERT、DELETE、UPDATE) を示すコード。

注: 依存する先の関数インスタンスが除去されると、パッケージは「作動不能」状態になり、明示的に再バインドする必要があります。依存先のその他のオブジェクトが除去されると、パッケージは「無効」状態になり、パッケージが最初に参照されるときにシステムによって自動的に再バインドが試みられます。

SYSCAT.PACKAGES

アプリケーション・プログラムをバインドして作成されたパッケージごとに 1 つの行が含まれています。

表 75. SYSCAT.PACKAGES カタログ視点

列名	データ・タイプ	ヌル値可	説明
PKGSCHEMA	VARCHAR(128)		パッケージの名前。
PKGNAME	CHAR(8)		
BOUNDBY	VARCHAR(128)		パッケージをバインドしたユーザーの許可 ID (OWNER)。
DEFINER	VARCHAR(128)		パッケージをバインドしたユーザーのユーザー ID。
DEFAULT_SCHEMA	VARCHAR(128)		静的 SQL ステートメントの非修飾名に使用されるデフォルト・スキーマ (QUALIFIER) の名前。
VALID	CHAR(1)		Y = 有効 N = 無効 X = パッケージが従属している関数インスタンスが除去されたので、パッケージは作動不能。明示的な再バインドが必要です。 1285ページの『SYSCAT.PACKAGEDEP』の注 1 を参照。
UNIQUE_ID	CHAR(8)		パッケージを最初に作成された時点を示す内部日付 / 時刻情報。
TOTAL_SECT	SMALLINT		パッケージのセクションの合計数。
FORMAT	CHAR(1)		パッケージに関連した日付と時刻の形式。 0 = データベースの国別コードに関連した形式 1 = USA 形式の日付、時刻 2 = EUR 形式の日付、時刻 3 = ISO 形式の日付、時刻 4 = JIS 形式の日付、時刻 5 = LOCAL 形式の日付、時刻

表 75. SYSCAT.PACKAGES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
ISOLATION	CHAR(2)	Yes	分離レベル。 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
BLOCKING	CHAR(1)	Yes	カーソル・ブロック化オプション N = ブロック化なし U = 確定カーソルをブロック化 B = すべてのカーソルをブロック化
INSERT_BUF	CHAR(1)		バインド時に使用された挿入オプション Y = 挿入内容はバッファーに入れられる N = 挿入内容はバッファーに入れられない
LANG_LEVEL	CHAR(1)	Yes	BIND 時に使用された LANGLEVEL 値 0 = SAA1 1 = SQL92E または MIA
FUNC_PATH	VARCHAR(254)		このパッケージの最後の BIND コマンドで使用された SQL パス。これは REBIND のデフォルトのパスとして使用されます。バージョン 2 より前のパッケージでは SYSIBM。
QUERYOPT	INTEGER		このパッケージをバインドした最適化クラス。再バインドに使用されます。 0、1、3、5、および 9 のクラスがあります。
EXPLAIN_LEVEL	CHAR(1)		EXPLAIN または EXPLSNAP バインド・オプションを使用して、 Explain が要求されたか否か。 P = プラン選択レベル Explain が要求されていない場合は、ブランク
EXPLAIN_MODE	CHAR(1)		EXPLAIN バインド・オプションの値 Y = Yes (静的) N = No A = All (静的と動的)

表 75. SYSCAT.PACKAGES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
EXPLAIN_SNAPSHOT	CHAR(1)		EXPLSNAP バインド・オプションの値 Y = Yes (静的) N = No A = All (静的と動的)
SQLWARN	CHAR(1)		動的 SQL ステートメントからアプリケーションに戻される結果の正の SQLCODE。 Y = Yes N = No (抑制される)
SQLMATHWARN	CHAR(1)		バインド時のデータベース構成パラメーター DFT_SQLMATHWARN の値。静的 SQL ステートメントの算術計算エラーと検索変換エラーを、警告を伴うヌル値として扱うか否か。 Y = Yes N = No (抑制される)
EXPLICIT_BIND_TIME	TIMESTAMP		このパッケージを最後に明示的にバインドまたは再バインドした時刻。パッケージを暗黙に再バインドすると、この時点以降に作成された関数インスタンスは選択されません。
LAST_BIND_TIME	TIMESTAMP		このパッケージが最後に明示的にバインドまたは再バインドされた時刻。
CODEPAGE	SMALLINT		バインド実行時のアプリケーションのコード・ページ (不明の場合は -1)。
DEGREE	CHAR(5)		パッケージのバインド時の区画内並行性に関する限界の指定 (バインド・オプションとしての)。 1 = 区画内並行性はない。 2 ～ 32 767 = 区画内並行性の度合い。 ANY = 度合いはデータベース・マネージャーによって決定される。
MULTINODE_PLANS	CHAR(1)		Y = パッケージは複数区分の環境でバインドされた。 N = パッケージは単一区分の環境でバインドされた。

表 75. SYSCAT.PACKAGES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
INTRA_PARALLEL	CHAR(1)		パッケージ内の静的 SQL ステートメントによる区画内並行性の使用を示します。 Y = パッケージ内の 1 つまたは複数の静的 SQL ステートメントが区画内並行性を使用する。 N = パッケージ内の静的 SQL ステートメントは区画内並行性を使用しない。 F = パッケージ内の 1 つまたは複数の静的 SQL ステートメントは区画内並行性を使用可能。この並列性は、区画内並行性を使用するように構成されていないシステムでは使用不可になっています。
VALIDATE	CHAR(1)		B = すべての検査は BIND 実行時に行う必要がある R = 予約済み
DYNAMICRULES	CHAR(1)		B = 動的 SQL ステートメントは実行時の静的 SQL ステートメントと同様に扱われ、バインドしたユーザーの権限 ID が使用される。 R = 動的 SQL ステートメントは実行時の動的 SQL ステートメントと同様に扱われ、実行者の権限 ID が使用される。 初期値は R。
SQLERROR	CHAR(1)		パッケージをバインドまたは再バインドした最新のサブコマンドの SQLERROR オプション。 C = 予約済み N = パッケージなし
REFRESHAGE	DECIMAL (20,6)		タイム・スタンプ期間。REFRESH TABLE ステートメントが要約表に実行されてから、要約表が基礎表の代わりに使用されるまでの最大時間を示す。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.PARTITIONMAPS

表の区分化キーのハッシュに基づいて、ノードグループ内の区分の間で表の行を再配布するために使用される区分化マップごとに、 1 つの行が含まれます。

表 76. SYSCAT.PARTITIONMAPS カタログ視点

列名	データ・タイプ	ヌル値可	説明
PMAP_ID	SMALLINT		区分化マップの識別子。
PARTITIONMAP	LONG VARCHAR FOR BIT DATA		実際の区分化マップ。複数ノードのノードグループの場合は、 4 096 個の 2 バイト整数から成るベクトル。単一ノードのノード・グループの場合は、単一ノードの区分 (またはノード) 番号を示す項目が 1 つあります。

SYSCAT.PASSTHRUAUTH

このカタログ視点には、パススルー・セッション内のデータ・ソースを照会できる権限に関する情報が含まれます。基礎表の制約により、 SERVER の値は、SYSCAT.SERVERS の SERVER 列の値に対応している必要があります。SYSCAT.PASSTHRUAUTH にはヌル可能なフィールドはありません。

表 77. SYSCAT.PASSTHRUAUTH カタログ視点内の列

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を授与したユーザーの許可 ID。
GRANTEE	VARCHAR(128)		特権を保持するユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		GRANTEE のタイプを指定する文字: U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
SERVERNAME	VARCHAR(128)		ユーザーまたはグループが許可を授与されるデータ・ソースの名前。

SYSCAT.PROCEDURES

作成されたストアード・プロシージャーごとに 1 つの行が含まれます。

表 78. SYSCAT.PROCEDURES カタログ視点

列名	データ・タイプ	ヌル値可	説明
PROCSCHEMA	VARCHAR(128)		プロシージャーの修飾名。
PROCNAME	VARCHAR(128)		
SPECIFICNAME	VARCHAR(18)		プロシージャー・インスタンスの名前 (システム生成の名前も含む)。
PROCEDURE_ID	INTEGER		ストアード・プロシージャーの内部 ID。
DEFINER	VARCHAR(128)		プロシージャー定義者の許可。
PARM_COUNT	SMALLINT		プロシージャーのパラメーター数。
PARM_SIGNATURE	VARCHAR(180) FOR BIT DATA		内部形式での 90 までのパラメーター・タイプを連結した値。プロシージャーにパラメーターがない場合は、長さ 0。
ORIGIN	CHAR(1)		常に 'E' = ユーザー定義、外部
CREATE_TIME	TIMESTAMP		プロシージャーの登録のタイム・スタンプ
DETERMINISTIC	CHAR(1)		Y = 結果は決定性。 N = 結果は決定性でない。
FENCED	CHAR(1)		Y = 分離 N = 非分離
NULLCALL	CHAR(1)		常に Y=NULLCALL
LANGUAGE	CHAR(8)		プロシージャー本体の実装言語。可能な値は次のとおりです。 C COBOL JAVA SQL
IMPLEMENTATION	VARCHAR(254)	Yes	プロシージャーを実装するパス / モジュール / 関数 (LANGUAGE=C または COBOL)、またはメソッド (LANGUAGE=JAVA) の識別。
CLASS	VARCHAR(128)	Yes	LANGUAGE=JAVA の場合、このプロシージャーを実装するクラス。それ以外の場合、ヌル値です。

表 78. SYSCAT.PROCEDURES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
JAR_ID	VARCHAR(128)	Yes	LANGUAGE=JAVA の場合、このプロシージャーを実装する jar ファイル。それ以外の場合、ヌル値です。
PARM_STYLE	CHAR(8)		DB2DARI = 言語は C DB2GENRL = 言語は Java DB2SQL = 言語は C または COBOL JAVA = 言語は Java または SQL GENERAL = 言語は C または COBOL GNLRNULL = 言語は C または COBOL
CONTAINS_SQL	CHAR(1)		プロシージャーに SQL が含まれるか否か。 C = CONTAINS SQL: SQL データの読み取りまたは変更を行わない SQL に限り許可されている。 M = MODIFY SQL DATA: プロシージャーで許可されているすべての SQL が許可されている。 N = NO SQL: SQL は許可されていない。 R = READS SQL DATA: SQL データを読み取る SQL に限り許可されている。
DBINFO	CHAR(1)		DBINFO パラメーターがプロシージャーに渡されるか否かを示します。 N = DBINFO は渡されない Y = DBINFO は渡される
PROGRAM_TYPE	CHAR(1)		プロシージャーを呼び出す方法。 M = メイン S = サブルーチン
RESULT_SETS	SMALLINT		戻される結果セットの上限の見積もり。

表 78. SYSCAT.PROCEDURES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
VALID	CHAR(1)		ブランク = SQL プロシージャーではない Y = SQL プロシージャーが有効 N = SQL プロシージャーが無効 X = SQL プロシージャーが必要とする何らかの関数インスタンスが除去されたので、その SQL プロシージャーは作動不能。その SQL プロシージャーを明示的に除去し、再作成しなければならない。
TEXT_BODY_OFFSET	INTEGER		SQL プロシージャーの場合、この列には、CREATE PROCEDURE ステートメントのテキスト全体の中の、 SQL プロシージャー本体の開始位置に対するオフセットが入れられます。外部プロシージャーの場合、値は 0 です。
TEXT	CLOB(1M)	Yes	SQL プロシージャーの場合、この列には、CREATE PROCEDURE ステートメントのテキスト全体が、入力されているとおりに入れられます。テキスト全体が 1M よりも大きい場合、または外部プロシージャーの場合、ヌルになります。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.PROCOPTIONS

各行には、プロシージャー固有のオプション値が含まれます。

表 79. SYSCAT.PROCOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
PROCSCHEMA	VARCHAR(128)		ストアード・プロシージャーの名前またはニックネームの修飾子。
PROCNAME	VARCHAR(128)		ストアード・プロシージャーの名前またはニックネーム。
OPTION	VARCHAR(128)		ストアード・プロシージャー・オプションの名前。
SETTING	VARCHAR(255)		ストアード・プロシージャー・オプションの値。

SYSCAT.PROCPARMOPTIONS

各行には、プロシージャー・パラメーター固有のオプション値が含まれます。

表 80. SYSCAT.PROCPARMOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
PROCSCHEMA	VARCHAR(128)		プロシージャーの修飾名またはニックネーム。
PROCNAME	VARCHAR(128)		
ORDINAL	SMALLINT		プロシージャーのシグニチャー内でのパラメーターの位置番号。
OPTION	VARCHAR(128)		ストアード・プロシージャー・オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.PROCPARMS

ストアード・プロシージャーのパラメーターごとに 1 つの行が含まれます。

表 81. SYSCAT.PROCPARMS カタログ視点

列名	データ・タイプ	ヌル値可	説明
PROCSCHEMA	VARCHAR(128)		プロシージャーの修飾名。
PROCNAME	VARCHAR(128)		
SPECIFICNAME	VARCHAR(18)		プロシージャー・インスタンスの名前 (システム生成の名前も含む)。
SERVERNAME	VARCHAR(128)	Yes	ストアード・プロシージャーがあるデータ・ソースの名前。
ORDINAL	SMALLINT		プロシージャーのシグニチャー内でのパラメーターの位置番号。
PARMNAME	VARCHAR(18)		パラメーター名。
TYPESCHEMA	VARCHAR(128)		パラメーターのデータ・タイプの修飾名。
TYPENAME	VARCHAR(18)		
TYPEID	SMALLINT	Yes	内部タイプ ID。
SOURCETYPEID	SMALLINT	Yes	ソース・タイプの内部タイプ ID。組み込みタイプの場合はヌル値。
NULLS	CHAR(1)		統合データベースのヌル値可能規則: Y = ヌル値可能 N = ヌル値不可
LENGTH	INTEGER		パラメーターの長さ。
SCALE	SMALLINT		パラメーターの位取り。
PARM_MODE	VARCHAR(5)		IN = 入力 OUT = 出力 INOUT = 入出力
CODEPAGE	SMALLINT		パラメーターのコード・ページ。該当しない場合、または FOR BIT DATA 属性を指定して宣言された文字データのパラメーターの場合は 0。
DBCS_CODEPAGE	SMALLINT	Yes	DBCS コード・ページ。数値フィールドの場合はヌル値。
AS_LOCATOR	CHAR(1)		常に 'N'

表 81. SYSCAT.PROCPARMS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
TARGET_TYPESCHEMA	VARCHAR(128)	Yes	パラメーターのタイプが参照であれば、ターゲット行タイプの修飾名が含まれます。それ以外の場合、ヌル値です。
TARGET_TYPENAME	VARCHAR(18)		
SCOPE_TABSCHEMA	VARCHAR(128)	Yes	パラメーターのタイプが参照であれば、効力範囲 (ターゲット表) の修飾名が含まれます。それ以外の場合、ヌル値です。
SCOPE_TABNAME	VARCHAR(128)		

SYSCAT.REFERENCES

定義された参照制約ごとに 1 つの行が含まれています。

表 82. SYSCAT.REFERENCES カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		制約の名前。
TABSCHEMA	VARCHAR(128)		制約の修飾名。
TABNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		制約を作成したユーザー。
REFKEYNAME	VARCHAR(18)		親キーの名前。
REFTABSCHEMA	VARCHAR(128)		親表の名前。
REFTABNAME	VARCHAR(128)		
COLCOUNT	SMALLINT		外部キーを構成する列の数。
DELETERULE	CHAR(1)		削除規則。 A = NO ACTION (アクションなし) C = CASCADE (削除カスケード) N = SET NULL (NULL 設定) R = RESTRICT (削除制限)
UPDATERULE	CHAR(1)		更新規則。 A = NO ACTION (アクションなし) R = RESTRICT (削除制限)
CREATE_TIME	TIMESTAMP		参照制限が定義されたタイム・スタンプ。
FK_COLNAMES	VARCHAR (640)		外部キーの列名のリスト。警告: この列は将来除去されます。 詳しくは、1280ページの『SYSCAT.KEYCOLUSE』を参照してください。
PK_COLNAMES	VARCHAR (640)		親キーの列名のリスト。警告: この列は将来除去されます。詳しくは、1280ページの『SYSCAT.KEYCOLUSE』を参照してください。

注:

1. SYSCAT.REFERENCES 視点は、バージョン 1 の SYSIBM.SYSRELS 表に基づいています。

SYSCAT.REVTYPEMAPPINGS

各行には、反対方向データ・タイプ・マッピング (ローカルに定義されたデータ・タイプから、データ・ソースのデータ・タイプにマップすること) が含まれます。このバージョンにデータはありません。将来の利用のためにデータ・タイプ・マッピングを使って定義されます。

表 83. SYSCAT.REVTYPEMAPPINGS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPE_MAPPING	VARCHAR(18)		反対方向タイプ・マッピングの名前 (システム生成の名前の場合もある)。
TYPESCHEMA	VARCHAR(128)	Yes	タイプのスキーマ名。システム組み込みタイプの場合はヌル値。
TYPENAME	VARCHAR(18)		反対方向タイプ・マッピングでのローカル・タイプの名前。
TYPEID	SMALLINT		タイプ識別子。
SOURCETYPEID	SMALLINT		ソース・タイプ識別子。
DEFINER	VARCHAR(128)		このタイプ・マッピングの作成に使用された許可 ID。
LOWER_LEN	INTEGER	Yes	ローカル・タイプの長さ / 精度の下限。
UPPER_LEN	INTEGER	Yes	ローカル・タイプの長さ / 精度の上限。ヌル値であれば、システムは最善の長さ / 精度属性を判別します。
LOWER_SCALE	SMALLINT	Yes	ローカルの 10 進数データ・タイプの位取りの下限。
UPPER_SCALE	SMALLINT	Yes	ローカルの 10 進数データ・タイプの位取りの上限。ヌル値であれば、システムは最善の位取りの属性を判別します。
S_OPR_P	CHAR(2)	Yes	ローカルの位取りとローカルの精度の関係。基本比較演算子を使用できます。ヌル値であれば、特定の関係は不要です。
BIT_DATA	CHAR(1)	Yes	Y = タイプはビット・データ用。 N = タイプはビット・データ用ではない。 NULL = 文字データ・タイプではなく、システムはビット・データ属性を判別しない。
WRAPNAME	VARCHAR(128)	Yes	このデータ・アクセス・プロトコルにマッピングが適用されます。

表 83. SYSCAT.REVTYPEMAPPINGS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
SERVERNAME	VARCHAR(128)	Yes	データ・ソースの名前。
SERVERTYPE	VARCHAR(30)	Yes	このタイプのデータ・ソースにマッピングが適用されます。
SERVERVERSION	VARCHAR(18)	Yes	このバージョンの SERVERTYPE にマッピングが適用されます。
REMOTE_TYPESCHEMA	VARCHAR(128)	Yes	リモート・タイプのスキーマ名。
REMOTE_TYPENAME	VARCHAR(128)		データ・ソースに対して定義されたデータ・タイプの名前。
REMOTE_META_TYPE	CHAR(1)	Yes	S = リモート・タイプはシステム組み込みタイプ。 T = リモート・タイプは特殊タイプ。
REMOTE_LENGTH	INTEGER	Yes	リモート 10 進タイプの最大桁数とリモート・文字タイプの最大文字数。それ以外の場合はヌル値。
REMOTE_SCALE	SMALLINT	Yes	小数点以下に使用できる最大桁数 (リモート 10 進タイプの場合)。それ以外の場合はヌル値。
REMOTE_BIT_DATA	CHAR(1)	Yes	Y = タイプはビット・データ用。 N = タイプはビット・データ用ではない。 NULL = 文字データ・タイプではなく、システムはビット・データ属性を判別しない。
USER_DEFINED	CHAR(1)		ユーザーによって定義される。
CREATE_TIME	TIMESTAMP		このマッピングが作成された時刻。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.SCHEMAAUTH

データベースの特定のスキーマに関する特権が付与されているユーザーまたはグループごとに 1 つまたは複数の行が含まれています。付与する側の特定のユーザーから、付与される側の特定のユーザーに与えられた 1 つのスキーマに関するすべてのスキーマ特権が 1 つの行に示されます。

表 84. SYSCAT.SCHEMAAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を付与したユーザーの許可 ID、または SYSIBM。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
SCHEMANAME	VARCHAR(128)		スキーマの名前。
ALTERINAUTH	CHAR(1)		GRANTEE に、スキーマに対する ALTERIN 特権があるか否か。 Y = 特権がある。 G = 特権があり、授与可能。 N = 特権がない。
CREATEINAUTH	CHAR(1)		GRANTEE に、スキーマに対する CREATEIN 特権があるか否か。 Y = 特権がある。 G = 特権があり、授与可能。 N = 特権がない。
DROPINAUTH	CHAR(1)		GRANTEE に、スキーマに対する DROPIN 特権があるか否か。 Y = 特権がある。 G = 特権があり、授与可能。 N = 特権がない。

SYSCAT.SCHEMATA

スキーマごとに 1 つの行が含まれています。

表 85. SYSCAT.SCHEMATA カタログ視点

列名	データ・タイプ	ヌル値可	説明
SCHEMANAME	VARCHAR(128)		スキーマの名前。
OWNER	VARCHAR(128)		スキーマの許可 ID。暗黙的に作成されたスキーマの値は SYSIBM。
DEFINER	VARCHAR(128)		スキーマを作成したユーザー。
CREATE_TIME	TIMESTAMP		オブジェクトが作成された時点を示すタイム・スタンプ。
REMARKS	VARCHAR(254)	Yes	ユーザーが入力したコメント。

SYSCAT.SERVEROPTIONS

各行には、サーバー・レベルの構成オプションが含まれます。

表 86. SYSCAT.SERVEROPTIONS カタログ視点の列

列名	データ・タイプ	ヌル値可	説明
WRAPNAME	VARCHAR(128)	Yes	ラッパー名。
SERVERNAME	VARCHAR(128)	Yes	サーバーの名前。
SERVERTYPE	VARCHAR(30)	Yes	サーバー・タイプ。
SERVERVERSION	VARCHAR(18)	Yes	サーバーのバージョン。
CREATE_TIME	TIMESTAMP		項目が作成された時刻。
OPTION	VARCHAR(128)		サーバー・オプションの名前。
SETTING	VARCHAR(2048)		サーバー・オプションの値。
SERVEROPTIONKEY	VARCHAR(18)		行を一意的に識別する。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.SERVERS

各行はデータ・ソースを表します。このカタログ表を含む同じインスタンスに保管されている表の場合、カタログ項目は不要です。

表 87. SYSCAT.SERVERS カタログ視点の列

資料名	データ・タイプ	ヌル値可	説明
WRAPNAME	VARCHAR(128)		ラッパー名。
SERVERNAME	VARCHAR(128)		システムに認識されているデータ・ソースの名前。
SERVERTYPE	VARCHAR(30)	Yes	データ・ソースのタイプ (常に大文字)。
SERVERVERSION	VARCHAR(18)	Yes	データ・ソースのバージョン。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.STATEMENTS

データベースの各パッケージの中の各 SQL ステートメントごとに、 1 つまたは複数の行が含まれています。

表 88. SYSCAT.STATEMENTS カタログ視点

列名	データ・タイプ	ヌル値可	説明
PKGSCHEMA	VARCHAR(128)		パッケージの名前。
PKGNAME	CHAR(8)		
STMTNO	INTEGER		アプリケーション・プログラムのソース・モジュールにおける SQL ステートメントの行番号。
SECTNO	SMALLINT		この SQL ステートメントを含むパッケージ・セクションの番号。
SEQNO	SMALLINT		常に 1。
TEXT	CLOB(64K)		SQL ステートメントのテキスト。

SYSCAT.TABAUTH

データベース内の特定の表または視点に関する特権を付与されているユーザーまたはグループごとに 1 つまたは複数の行が含まれています。特定のユーザーから特定のユーザーに付与された 1 つの表または視点に関するすべての表特権が 1 つの行に示されます。

表 89. SYSCAT.TABAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	VARCHAR(128)		特権を付与したユーザーの許可 ID、または SYSIBM。
GRANTEE	VARCHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
TABSCHEMA	VARCHAR(128)		表または視点の修飾名。
TABNAME	VARCHAR(128)		
CONTROLAUTH	CHAR(1)		GRANTEE に、表または視点に対する CONTROL 特権があるか否か。 Y = 特権がある。 N = 特権がない。
ALTERAUTH	CHAR(1)		GRANTEE に、表に対する ALTER 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。
DELETEAUTH	CHAR(1)		GRANTEE に、表または視点に対する DELETE 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。
INDEXAUTH	CHAR(1)		GRANTEE に、表に対する INDEX 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。

表 89. SYSCAT.TABAUTH カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
INSERTAUTH	CHAR(1)		GRANTEE に、表または視点に対する INSERT 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。
SELECTAUTH	CHAR(1)		GRANTEE に、表または視点に対する SELECT 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。
REFAUTH	CHAR(1)		GRANTEE に、表または視点に対する REFERENCE 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。
UPDATEAUTH	CHAR(1)		GRANTEE に、表または視点に対する UPDATE 特権があるか否か。 Y = 特権がある。 N = 特権がない。 G = 特権があり、授与可能。

SYSCAT.TABCONST

それぞれの行は、タイプ CHECK、 UNIQUE、PRIMARY KEY、または FOREIGN KEY の表制約を示しています。

表 90. SYSCAT.TABCONST カタログ視点

列名	データ・タイプ	ヌル値可	説明
CONSTNAME	VARCHAR(18)		制約の名前 (表内で固有)。
TABSCHEMA	VARCHAR(128)		この制約が適用される表の修飾名。
TABNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		制約の定義時の許可 ID。
TYPE	CHAR(1)		制約の種類。 F = FOREIGN KEY (外部キー) K = CHECK (検査) P = PRIMARY KEY (基本キー) U = UNIQUE (固有)
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.TABLES

作成されている表、視点、ニックネーム、または別名ごとに 1 つの行が含まれます。カタログ表とカタログ視点はすべて、SYSCAT.TABLES カタログ視点に項目を持っています。

表 91. SYSCAT.TABLES カタログ視点

列名	データ・タイプ	ヌル値可	説明
TABSCHEMA	VARCHAR(128)		表、視点、ニックネーム、または視点の修飾名。
TABNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		表、視点、ニックネーム、または別名を作成したユーザー。
TYPE	CHAR(1)		オブジェクトの種類。 A = 別名 H = 階層表 N = ニックネーム S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点
STATUS	CHAR(1)		オブジェクトの種類。 N = 通常の表、視点、別名、またはニックネーム C = 表またはニックネームについての検査保留 X = 作動不能の視点またはニックネーム
BASE_TABSCHEMA	VARCHAR(128)	Yes	TYPE=A の場合、これらの列は該当の別名によって参照される表、視点、別名、またはニックネームを示し、その他の場合はヌル値です。
BASE_TABNAME	VARCHAR(128)	Yes	
ROWTYPESCHEMA	VARCHAR(128)	Yes	該当する場合、この表の行タイプの修飾名が含まれます。それ以外の場合、ヌル値です。
ROWTYPENAME	VARCHAR(18)		
CREATE_TIME	TIMESTAMP		オブジェクトが作成された時点を示すタイム・スタンプ。
STATS_TIME	TIMESTAMP	Yes	この表に対する何らかの変更が最後に統計に記録された時刻。統計が使用可能でない場合は、ヌル値。

表 91. SYSCAT.TABLES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
COLCOUNT	SMALLINT		表の列の数。
TABLEID	SMALLINT		表の内部識別子。
TBSPACEID	SMALLINT		この表の 1 次表スペースの内部識別子。
CARD	BIGINT		表の中の行の総数。表階層内の表の場合、統計が収集されていないか、この行が視点または別名を記述している場合は、階層 -1 の指定レベルでの行数。階層表の場合は -2。
NPAGES	INTEGER		表の行が存在しているページの総数。統計が収集されていないか、この行が視点または別名を記述している場合は -1。副表または表階層の場合は -2。
FPAGES	INTEGER		ページの総数。統計が収集されていないか、この行が視点または別名を記述している場合は -1。副表または表階層の場合は -2。
OVERFLOW	INTEGER		表のオーバーフロー・レコードの総数。統計が収集されていないか、この行が視点または別名を記述している場合は -1。副表または表階層の場合は -2。
TBSPACE	VARCHAR(18)	Yes	表の 1 次表スペースの名前。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。別名および視点の場合は、ヌル値です。
INDEX_TBSPACE	VARCHAR(18)	Yes	この表に作成されたすべての索引が保管されている表スペースの名前。別名および視点の場合、または INDEX IN 文節の指定がないか、または INDEX IN 文節に CREATE TABLE ステートメントの IN 文節と同じ値が指定されている場合は、ヌル値。
LONG_TBSPACE	VARCHAR(18)	Yes	この表のすべての長形式データ (LONG または LOB の列タイプ) が入っている表スペースの名前。別名および視点の場合、または LONG IN 文節が指定されていないか、または LONG IN 文節に CREATE TABLE ステートメントの IN 文節と同じ値が指定されている場合は、ヌル値。
PARENTS	SMALLINT	Yes	この表の親表の数 (この表が従属表になっている参照制約の数)。

表 91. SYSCAT.TABLES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
CHILDREN	SMALLINT	Yes	この表に従属している表の数 (この表が親表になっている参照制約の数)。
SELFREFS	SMALLINT	Yes	この表に対する自己参照になっている参照制約の数 (この表が親表であり、また従属表でもある参照制約の数)。
KEYCOLUMNS	SMALLINT	Yes	表の基本キーを構成する列の数。
KEYINDEXID	SMALLINT	Yes	1 次索引の索引 ID。基本キーがない場合はヌル値 (NULL) または 0。
KEYUNIQUE	SMALLINT		この表に定義された固有制約 (基本キー以外) の数。
CHECKCOUNT	SMALLINT		この表に定義された検査制約の数。
DATACAPTURE	CHAR(1)		Y = 表はデータ複製に関与している N = 関与していない L = 表はデータ複製 (LONG VARCHAR および LONG VARGRAPHIC 列の複製を含む) に関与している
CONST_CHECKED	CHAR(32)		バイト 1 は、外部キー制約を表します。バイト 2 は、検査制約を表します。バイト 5 は、要約表を表します。バイト 6 は生成される列を表します。他のバイトは予約済み。検査に関する制約情報を示すコード。値: Y = システムによる検査 U = ユーザーによる検査 N = 検査なし (保留状態) W = 表が検査保留 (保留状態) になったときに 'U' の状態だった
PMAP_ID	SMALLINT	Yes	この表が使用する区分化マップの識別子。別名および視点の場合は、ヌル値です。
PARTITION_MODE	CHAR(1)		区分データベースの表に使用されるモード。 H = 区分化キーのハッシュ R = データベース区分間で複製された表 別名、視点、および区分化キーが定義されていない単一区分のノード・グループの表の場合は、ブランク。ニックネームの場合もブランク。

表 91. SYSCAT.TABLES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
LOG_ATTRIBUTE	CHAR(1)		0 = デフォルトのログ記録 1 = 初期にログ記録を取らずに作成された表
PCTFREE	SMALLINT		将来の挿入用に予約されたページのパーセンテージ。 ALTER TABLE によって変更可能。
APPEND_MODE	CHAR(1)		ページ上での行の挿入方法、 N = 新規行は使用可能な既存スペースに挿入される Y = 新規行はデータの終わりに追加される 初期値は N。
REFRESH	CHAR(1)		最新表示モード D = 据え置き I = 即時 O = 1 回 要約表でなければブランク
REFRESH_TIME	TIMESTAMP	Yes	REFRESH=D または O の場合、データを最後に最新表示した REFRESH TABLE ステートメントのタイム・スタンプ。それ以外の場合はヌル値。
LOCKSIZE	CHAR(1)		DML ステートメントがアクセスしたときに優先する表ロックの細分性を示します。表にのみ適用されます。可能な値は次のとおりです。 R = 行 T = 表 該当しない場合はブランク 初期値は R。
VOLATILE	CHAR(1)		C = 表のカーディナリティーは揮発性 該当しない場合はブランク
REMARKS	VARCHAR(254)	Yes	ユーザーが入力したコメント。

SYSCAT.TABLESPACES

表スペースごとに 1 つの行が含まれています。

表 92. SYSCAT.TABLESPACES カタログ視点

列名	データ・タイプ	ヌル値可	説明
TBSPACE	VARCHAR(18)		表スペースの名前。
DEFINER	VARCHAR(128)		表スペースの定義者の許可 ID。
CREATE_TIME	TIMESTAMP		表スペースの作成時刻。
TBSPACEID	INTEGER		表スペースの内部識別子。
TBSPACETYPE	CHAR(1)		表スペースのタイプ。 S = システム管理スペース D = データベース管理スペース
DATATYPE	CHAR(1)		保管できるデータのタイプ。 A = すべてのデータ・タイプの永続データ L = 長形式データのみ T = システム一時表のみ U = 宣言された一時表のみ
EXTENTSIZE	INTEGER		サイズ PAGESIZE をページ単位とするエクステント・サイズ。表スペースのコンテナーにこの数のページが書き込まれると、次のコンテナーに切り替わります。
PREFETCHSIZE	INTEGER		事前取り出しの実行時に読み取るサイズ PAGESIZE のページ数。
OVERHEAD	DOUBLE		コントローラー・オーバーヘッドおよびディスク・シークおよび待ち時間 (ミリ秒単位)。
TRANSFERRATE	DOUBLE		サイズ PAGESIZE から成る 1 ページをバッファーに読み込む時間。
PAGESIZE	INTEGER		表スペースのページのサイズ (バイト単位)。
NGNAME	VARCHAR(18)		表スペースのノードグループの名前。
BUFFERPOOLID	INTEGER		この表スペースにより使用されるバッファー・プールの ID (1 はデフォルトのバッファー・プールを示す)。
DROP_RECOVERY	CHAR(1)		N = 表は DROP TABLE ステートメントの後に回復不可 Y = 表は DROP TABLE ステートメントの後に回復可能

表 92. SYSCAT.TABLESPACES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
REMARKS	VARCHAR(254)	Yes	ユーザーが入力したコメント。

SYSCAT.TABOPTIONS

各行には、リモート表に関連するオプションが含まれます。

表 93. SYSCAT.TABOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TABSCHEMA	VARCHAR(128)		表、視点、別名、またはニックネームの修飾名。
TABNAME	VARCHAR(128)		
OPTION	VARCHAR(128)		表、視点、別名、またはニックネームのオプション名。
SETTING	VARCHAR(255)		値。

SYSCAT.TBSPACEAUTH

データベース内の特定の表スペースに対する USE 特権を付与されているユーザーまたはグループごとに 1 つの行が含まれます。

表 94. SYSCAT.TBSPACEAUTH カタログ視点

列名	データ・タイプ	ヌル値可	説明
GRANTOR	CHAR(128)		特権を付与したユーザーの許可 ID、または SYSIBM。
GRANTEE	CHAR(128)		特権を付与されたユーザーまたはグループの許可 ID。
GRANTEETYPE	CHAR(1)		U = GRANTEE は個々のユーザー。 G = GRANTEE はグループ。
TBSPACE	VARCHAR(18)		表スペースの名前
USEAUTH	CHAR(1)		GRANTEE に、表スペースに対する USE 特権があるか否か。 G = 特権があり、授与可能。 N = 特権がない。 Y = 特権がある。

SYSCAT.TRIGDEP

他の特定のオブジェクトへのトリガーの従属関係ごとに 1 つの行が含まれています。

表 95. SYSCAT.TRIGDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
TRIGSCHEMA	VARCHAR(128)		トリガーの修飾名。
TRIGNAME	VARCHAR(18)		
BTYPE	CHAR(1)		オブジェクト BNAME のタイプ。 A = 別名 F = 関数インスタンス N = ニックネーム O = 表または視点階層内のすべての副表または副視点に対する特権の従属関係 R = 構造タイプ S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点 X = 索引の拡張
BSCHEMA	VARCHAR(128)		トリガーが従属しているオブジェクトの修飾名。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE= O、S、T、U、V、または W の場合、従属索引で必要な表または視点に対する特権を示すコード。

SYSCAT.TRIGGERS

トリガーごとに 1 つの行が含まれています。表階層の場合、トリガーはそれぞれ作成された階層レベルでのみ記録されます。

表 96. SYSCAT.TRIGGERS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TRIGSCHEMA	VARCHAR(128)		トリガーの修飾名。
TRIGNAME	VARCHAR(18)		
DEFINER	VARCHAR(128)		トリガーの定義時の許可 ID。
TABSCHEMA	VARCHAR(128)		このトリガーが適用される表の修飾名。
TABNAME	VARCHAR(128)		
TRIGTIME	CHAR(1)		トリガーを起動したイベントに関連して、いつトリガー・アクションが該当の基礎表に適用するか。 A = トリガーはイベントの後に適用される B = トリガーはイベントの前に適用される
TRIGEVENT	CHAR(1)		トリガーを起動するイベント。 I = 挿入 D = 削除 U = 更新
GRANULARITY	CHAR(1)		トリガーが実行される単位。 S = ステートメント R = 行
VALID	CHAR(1)		Y = トリガーは有効 X = トリガーは作動不能で、再作成が必要
CREATE_TIME	TIMESTAMP		トリガーが定義された時刻。関数とタイプの解決に使用されます。
QUALIFIER	VARCHAR(128)		オブジェクト定義時のデフォルト・スキーマの値が含まれます。
FUNC_PATH	VARCHAR(254)		トリガーの定義された時点の関数パス。関数とタイプの解決に使用されます。
TEXT	CLOB(64K)		入力されたとおりの CREATE TRIGGER ステートメントのテキスト全体。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.TYPEMAPPINGS

各行には、リモート組み込みデータ・タイプからローカル組み込みデータ・タイプへのユーザー定義マッピングが含まれます。

表 97. SYSCAT.TYPEMAPPINGS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPE_MAPPING	VARCHAR(18)		タイプ・マッピングの名前 (システム生成の場合もある)。
TYPESCHEMA	VARCHAR(128)	Yes	タイプのスキーマ名。システム組み込みタイプの場合はヌル値。
TYPENAME	VARCHAR(18)		データ・タイプ・マッピングでのローカル・タイプの名前。
TYPEID	SMALLINT		タイプ識別子。
SOURCETYPEID	SMALLINT		ソース・タイプ識別子。
DEFINER	VARCHAR(128)		このタイプ・マッピングの作成に使用された許可 ID。
LENGTH	INTEGER	Yes	データ・タイプの最大長または精度。ヌル値の場合、システムは最善の長さ / 精度を判別します。
SCALE	SMALLINT	Yes	DECIMAL フィールドの位取り。ヌル値の場合、システムは最善の位取りの属性を判別します。
BIT_DATA	CHAR(1)	Yes	Y = タイプはビット・データ用。 N = タイプはビット・データ用ではない。 NULL = 文字データ・タイプではなく、システムはビット・データ属性を判別しない。
WRAPNAME	VARCHAR(128)	Yes	このデータ・アクセス・プロトコルにマッピングが適用されます。
SERVERNAME	VARCHAR(128)	Yes	データ・ソースの名前。
SERVERTYPE	VARCHAR(30)	Yes	このタイプのデータ・ソースにマッピングが適用されます。
SERVERVERSION	VARCHAR(18)	Yes	このバージョンの SERVERTYPE にマッピングが適用されます。
REMOTE_TYPESCHEMA	VARCHAR(128)	Yes	リモート・タイプのスキーマ名。

表 97. SYSCAT.TYPEMAPPINGS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
REMOTE_TYPENAME	VARCHAR(128)		データ・ソースに対して定義されたデータ・タイプの名前。
REMOTE_META_TYPE	CHAR(1)	Yes	S = リモート・タイプはシステム組み込みタイプ。 T = リモート・タイプは特殊タイプ。
REMOTE_LOWER_LEN	INTEGER	Yes	リモート 10 進数タイプの長さ / 精度の下限。文字データ・タイプの場合、このフィールドは文字数を示します。
REMOTE_UPPER_LEN	INTEGER	Yes	リモート 10 進数タイプの長さ / 精度の上限。文字データ・タイプの場合、このフィールドは文字数を示します。
REMOTE_LOWER_SCALE	SMALLINT	Yes	リモート・タイプの位取りの下限。
REMOTE_UPPER_SCALE	SMALLINT	Yes	リモート・タイプの位取りの上限。
REMOTE_S_OPR_P	CHAR(2)	Yes	リモートの位取りとリモートの精度の関係。基本比較演算子を使用できます。ヌル値であれば、特定の関係は不要です。
REMOTE_BIT_DATA	CHAR(1)	Yes	Y = タイプはビット・データ用。 N = タイプはビット・データ用ではない。 NULL = 文字データ・タイプではなく、システムはビット・データ属性を判別しない。
USER_DEFINED	CHAR(1)		ユーザーが提供する定義。
CREATE_TIME	TIMESTAMP		このマッピングが作成された時刻。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSCAT.USEROPTIONS

各行には、サーバー固有のオプション値が含まれます。

表 98. SYSCAT.USEROPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
AUTHID	VARCHAR(128)		ローカル許可 ID (常に大文字)
SERVERNAME	VARCHAR(128)		ユーザーが定義されたサーバーの名前。
OPTION	VARCHAR(128)		ユーザー・オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.VIEWDEP

他の特定のオブジェクトへの視点または要約表の従属関係ごとに 1 つの行が含まれています。また、この視点に対する特権が、基礎表または視点に対する特権にどのように従属しているかを示すコードも含まれています。

表 99. SYSCAT.VIEWDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
VIEWSCHEMA	VARCHAR(128)		基礎表への従属関係を持つ要約表の名前または視点の名前。
VIEWNAME	VARCHAR(128)		
DTYPE	CHAR(1)		S = 要約表 V = 視点 (非タイプ付き) W = タイプ付き視点
DEFINER	VARCHAR(128)	Yes	視点の作成者の許可 ID。
BTYPE	CHAR(1)		オブジェクト BNAME のタイプ。 A = 別名 F = 関数インスタンス N = ニックネーム O = 表または視点階層内のすべての副表または副視点に対する特権の従属関係 I = 索引 (基礎表への従属関係を記録する場合) R = 構造タイプ S = 要約表 T = 表 U = タイプ付き表 V = 視点 W = タイプ付き視点
BSCHEMA	VARCHAR(128)		視点が従属しているオブジェクトの修飾名。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE= O、S、T、U、V、W の場合、この視点が従属している基礎表または基礎視点に対する特権を示すコード。それ以外の場合はヌル値。

SYSCAT.VIEWS

作成された視点ごとに 1 つまたは複数の行が含まれています。

表 100. SYSCAT.VIEWS カタログ視点

列名	データ・タイプ	ヌル値可	説明
VIEWSCHEMA	VARCHAR(128)		要約表の定義に使用する表の名前または視点の名前。
VIEWNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		視点の作成者の許可 ID。
SEQNO	SMALLINT		常に 1。
VIEWCHECK	CHAR(1)		視点の検査のタイプ。 N = 検査オプションはない L = ローカル検査オプション C = カスケード検査オプション
READONLY	CHAR(1)		Y = 視点はその定義により読み取り専用 N = 視点は読み取り専用ではない
VALID	CHAR(1)		Y = 視点または要約表の定義が有効である。 X = 視点または要約表の定義が作動不能である。再作成が必要。
QUALIFIER	VARCHAR(128)		オブジェクト定義時のデフォルト・スキーマの値が含まれます。
FUNC_PATH	VARCHAR(254)		視点作成時の視点作成者の SQL パス。視点をデータ操作ステートメントで使用する場合、視点における関数呼び出しを解決するのにこのパスを使用する必要があります。バージョン 2 より前に作成された視点の場合は SYSIBM です。
TEXT	CLOB(64k)		CREATE VIEW ステートメントのテキスト。

SYSCAT.WRAPOPTIONS

各行には、ラッパー固有のオプションが含まれます。

表 101. SYSCAT.WRAPOPTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
WRAPNAME	VARCHAR(128)		ラッパー名。
OPTION	VARCHAR(128)		ラッパー・オプションの名前。
SETTING	VARCHAR(255)		値。

SYSCAT.WRAPPERS

各行には、登録済みラッパーに関する情報が含まれます。

表 102. SYSCAT.WRAPPERS カタログ視点

列名	データ・タイプ	ヌル値可	説明
WRAPNAME	VARCHAR(128)		ラッパー名。
WRAPTYPE	CHAR(1)		N = 非リレーショナル R = リレーショナル
WRAPVERSION	INTEGER		ラッパーのバージョン。
LIBRARY	VARCHAR(255)		このラッパーに関連したデータ・ソースとの通信に使用するコードが入っているファイルの名前。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

SYSSTAT.COLDIST

各行は、列の N 番目の頻出値または N 番目の変位値を記述しています。タイプ付き表の継承列の場合、統計は記録されません。

表 103. SYSSTAT.COLDIST カタログ視点

列名	データ・タイプ	ヌル値可	説明	更新可能
TABSCHEMA	VARCHAR(128)		この項目が適用される表の修飾名。	
TABNAME	VARCHAR(128)			
COLNAME	VARCHAR(128)		この項目が適用される列の名前。	
TYPE	CHAR(1)		収集する統計のタイプ。 F = 頻度 (最大頻出値) Q = 変位値	
SEQNO	SMALLINT		TYPE=F の場合、この列の N は第 N 頻出値。 TYPE=Q の場合、この列の N は第 N 変位値。	
COLVALUE	VARCHAR(254)	Yes	データ値 (文字リテラルまたはヌル値)。 この列は、統計に関連している列に対応する値の有効な表記を用いて更新可能です。必要な頻出度がヌル値の場合は、この列を NULL に設定する必要があります。	Yes
VALCOUNT	BIGINT		TYPE=F の場合、VALCOUNT は、その列の中の COLVALUE の出現回数。 TYPE=Q の場合、VALCOUNT は、値が COLVALUE 以下の行の数。 この列は、次の値でのみ更新可能です。 • >= 0 (ゼロ)	Yes
DISTCOUNT	BIGINT		TYPE=q の場合、この列は COLVALUE に等しいかより小さい特殊値の数 (入手不能の場合はヌル) を記録します。値が COLVALUE より小さいか等しい行の数。	Yes

SYSSTAT.COLUMNS

統計を更新できる列ごとに 1 つの行が含まれています。タイプ付き表の継承列の場合、統計は記録されません。

表 104. SYSSTAT.COLUMNS カタログ視点

列名	データ・タイプ	ヌル値可	説明	更新可能
TABSCHEMA	VARCHAR(128)		列を含む表の修飾名。	
TABNAME	VARCHAR(128)			
COLNAME	VARCHAR(128)		列名。	
COLCARD	BIGINT		列の値の種類数。統計が収集されていない場合は -1。継承列または階層表の列の場合は -2。 どのような列についても、COLCARD は、その列を含む表のカーディナリティーを超える値になることはありません。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
HIGH2KEY	VARCHAR(33)	Yes	列の 2 番目の最高値。統計が収集されておらず、継承列および階層表の列の場合、このフィールドは空です。 この列は、統計に関連している列に対応する値の有効な表記を用いて更新可能です。 LOWKEY2 は HIGH2KEY を超えてはなりません。	Yes
LOW2KEY	VARCHAR(33)	Yes	列の 2 番目の最低値。統計が収集されておらず、継承列および階層表の列の場合は空です。 この列は、統計に関連している列に対応する値の有効な表記を用いて更新可能です。	Yes

表 104. SYSSTAT.COLUMNS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明	更新可能
AVGCOLLEN	INTEGER		列の平均の長さ。長形式フィールドまたは LOB の場合、または統計が収集されていない場合は -1。継承列および階層表の列の場合は -2。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
NUMNULLS	BIGINT		列のヌル値の数が含まれています。統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes

SYSSTAT.FUNCTIONS

ユーザー定義関数 (スカラーまたは集合) ごとに 1 つの行が含まれています。組み込み関数は含まれません。タイプ付き表の継承列の場合、統計は記録されません。

表 105. SYSSTAT.FUNCTIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明	更新可能
FUNCSCHEMA	VARCHAR(128)		関数の修飾名。	
FUNCNAME	VARCHAR(18)			
SPECIFICNAME	VARCHAR(18)		関数の特定名 (インスタンス名)。	
IOS_PER_INVOC	DOUBLE		呼び出しごとの入出力回数の見積もり。不明の場合は -1 (デフォルト値は 0)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
INSTS_PER_INVOC	DOUBLE		呼び出しごとの命令の数の見積もり。不明の場合は -1 (デフォルト値は 450)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
IOS_PER_ARGBYTE	DOUBLE		入力引き数 1 バイトごとの入出力回数の見積もり。不明の場合は -1 (デフォルト値は 0)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
INSTS_PER_ARGBYTE	DOUBLE		入力引き数 1 バイトごとの命令数の見積もり。不明の場合は -1 (デフォルト値は 0)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
PERCENT_ARGBYTES	SMALLINT		関数が実際に読み取る入力引き数バイトの平均パーセント値の見積もり。不明の場合は -1 (デフォルト値は 100)。 この列は、次の値でのみ更新可能です。 • -1 または 0 ～ 100	Yes

表 105. SYSSTAT.FUNCTIONS カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明	更新可能
INITIAL_IOS	DOUBLE		関数が最初 / 最後に呼び出されたときに実行される入出力の回数の見積もり。不明の場合は -1 (デフォルト値は 0)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
INITIAL_INSTS	DOUBLE		最初 / 最後の関数呼び出し時に実行される命令の数の見積もり。不明の場合は -1 (デフォルト値は 0)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
CARDINALITY	BIGINT		予想される表関数のカーディナリティー。不明の場合、または関数が表関数でない場合は -1。	Yes
SELECTIVITY	DOUBLE		ユーザーが定義した述部用に使用される。デフォルト = -1 (ユーザー定義述部がない場合)。注 1 を参照。	

注:

1. どのようなユーザー定義関数でも、システム・カタログでの DB2 バージョン 5.2 から 6.1 への移行中は、この列は -1 に設定されます。ユーザー定義述部の場合、システム・カタログでの選択性は -1 になります。この場合、最適化プログラムが使用する選択性の値は 0.01 です。

SYSSTAT.INDEXES

表に定義されている各索引ごとに 1 つの行が含まれます。

表 106. SYSSTAT.INDEXES カタログ視点

列名	データ・タイプ	ヌル値可	説明	更新可能
INDSCHEMA	VARCHAR(128)		索引の修飾名。	
INDNAME	VARCHAR(18)			
TABSCHEMA	VARCHAR(128)		表名の修飾子。	
TABNAME	VARCHAR(128)		その索引の定義されている表またはニックネームの名前。	
COLNAMES	CLOB(1M)		+ または - の接頭部が付いた列名のリスト。	
NLEAF	INTEGER		葉ページの数。統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または > 0 (ゼロ)	Yes
NLEVELS	SMALLINT		索引レベルの数。統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または > 0 (ゼロ)	Yes
FIRSTKEYCARD	BIGINT		最初のキーの値の種類数。統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
FIRST2KEYCARD	BIGINT		索引の最初の 2 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes

表 106. SYSSTAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明	更新可能
FIRST3KEYCARD	BIGINT		索引の最初の 3 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
FIRST4KEYCARD	BIGINT		索引の最初の 4 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
FULLKEYCARD	BIGINT		キー全体の値の種類数。統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
CLUSTERRATIO	SMALLINT		これは最適化プログラムにより使用されます。索引のデータ・クラスター化の程度を示し、統計が収集されていない場合、または詳細な索引統計が収集されている場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または 0 ～ 100	Yes
CLUSTERFACTOR	DOUBLE		これは最適化プログラムにより使用されます。クラスター化の程度の詳細測定値。詳細索引統計が収集されていない場合は -1。 この列は、次の値でのみ更新可能です。 • -1 または 0 ～ 1	Yes

表 106. SYSSTAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明	更新可能
SEQUENTIAL_PAGES	INTEGER		索引キーの順序でディスクに存在し、それらの間に大きなギャップがないか、わずかなギャップしかない葉ページの数。(統計が入手できない場合は -1。) この列は、次の値でのみ更新可能です。 • -1 または >= 0 (ゼロ)	Yes
DENSITY	INTEGER		索引によって占有されているページの範囲内の、ページ数に対する SEQUENTIAL_PAGES の比率。パーセントで表現される (0 ～ 100 の整数。統計が入手できない場合は -1。) この列は、次の値でのみ更新可能です。 • -1 または 0 ～ 100	Yes
PAGE_FETCH_PAIRS	VARCHAR(254)		文字形式で表される、整数の対のリスト。それぞれの対は、仮のバッファーのページ数と、その仮のバッファーを使用した索引の走査に必要なページ取り出しの回数を表しています。 (データが入手できない場合は、長さ 0 のストリング。) この列は、次の入力値でのみ更新できます。 • 対の区切り文字および対の区切り記号は、数字以外の文字だけが受け入れられます。 • ブランクは、対の区切り文字および対の区切り記号として認識される唯一の文字です。 • 各数値項目には、対応するパートナーの数値項目がなければならず、それら 2 つは対区切り記号で区切る必要があります。 • 対と対の間は、対区切り文字で区切る必要があります。 • 予期される数値項目は、0 ～ 9 の正の数でなければなりません。	Yes

SYSSTAT.TABLES

基礎 表ごとに 1 つの行が含まれています。視点または別名は含まれません。タイプ付き表の場合、この視点に含まれるのは表階層のルート表だけです。タイプ付き表の継承列の場合、統計は記録されません。 CARD 値は、他の統計が表階層全体に適用される場合にのみ、ルート表に適用されます。

表 107. SYSSTAT.TABLES カタログ視点

列名	データ・タイプ	ヌル値可	説明	更新可能
TABSCHEMA	VARCHAR(128)		表の修飾名。	
TABNAME	VARCHAR(128)			
CARD	BIGINT		表の中の行の総数。統計が収集されていない場合は -1。 表の CARD を更新する場合、その表の列のいずれかの COLCARD の値よりも小さい値を割り当ててはなりません。この列は、次の値でのみ更新できます。[115] • -1 または >= 0 (ゼロ)	Yes
NPAGES	INTEGER		表の行が存在しているページの総数。統計が収集されていない場合は -1。副表および階層表の場合は -2。 この列は、次の値でのみ更新可能です。[115] • -1 または >= 0 (ゼロ)	Yes
FPAGES	INTEGER		ファイルのページの総数。統計が収集されていない場合は -1。副表および階層表の場合は -2。 この列は、次の値でのみ更新可能です。[115] • -1 または >= 0 (ゼロ)	Yes

115. 値 -2 は変更できません。また、列の値は直接 -2 に設定できません。

表 107. SYSSTAT.TABLES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明	更新可能
OVERFLOW	INTEGER		表のオーバーフロー・レコードの総数。統計が収集されていない場合は -1。副表および階層表の場合は -2。 この列は、次の値でのみ更新可能です。[115] • -1 または >= 0 (ゼロ)	Yes

付録E. 構造タイプで使用するカタログ視点

拡張索引を使用する場合、追加のカタログ視点を使用すれば、 SYSCAT カタログ視点の実装に関する有用な情報を得ることができます。これらの視点は自動的には作成されません。このような視点は OBJCAT スキーマで作成され、すべての視点に対する SELECT 特権が、デフォルト解釈により PUBLIC に付与されます。

警告: 視点のこのセットは、次のバージョンでカタログ移行がサポートされるまで一時的に使用するためのものです。これらの視点があらゆるデータベースにあるものと、アプリケーションがみなすことがないようにしてください。また、将来のバージョンでは、これらのカタログはおそらく提供されなくなるということを考慮してください。これらの視点からの情報は、将来のバージョンでは SYSCAT 視点を使用してサポートされる予定です。

視点を作成するには、以下のステップを実行してください。

- コマンド行プロセッサーを使用し、 SYSADM または DBADM 権限がある許可 ID でデータベースに接続します。
- DB2 インスタンスのホーム・ディレクトリーにいることを確認します。
- UNIX ベースのシステムの場合、以下のコマンドを出します。

  ```
  db2 -tvf sqllib/bin/objcat.db2
  ```

- OS/2 または Windows ベースのシステムの場合、以下のコマンドを出します。

  ```
  db2 -tvf sqllib¥bin¥objcat.db2
  ```

objcat.db2 で作成した視点を除去するには、以下のステップを実行してください。

- コマンド行プロセッサーを使用し、 SYSADM または DBADM 権限がある許可 ID でデータベースに接続します。
- DB2 インスタンスのホーム・ディレクトリーにいることを確認します。
- UNIX ベースのシステムの場合、以下のコマンドを出します。

  ```
  db2 -tvf sqllib/bin/objcatdp.db2
  ```

- OS/2 または Windows ベースのシステムの場合、以下のコマンドを出します。

```
db2 -tvf sqllib¥bin¥objcatdp.db2
```

注: データベースにすでに OBJCAT というスキーマがある場合、ファイル **objcat.db2** の独自のコピーを作成し、 2 番目および 3 番目の CREATE SCHEMA ステートメントのスキーマ名を、適当な名前に変更する必要があります。

OBJCAT.DB2 ファイル内のステートメントでは、追加の OBJCAT カタログ視点がすべて作成されます。

この付録は、個々の OBJCAT カタログ視点について説明しています。関連する SYSCAT 視点については、1231ページの『付録D. カタログ視点』を参照してください。

カタログ視点は、SQL データ定義ステートメント、環境ルーチン、および特定のユーティリティーに対応する通常の操作の過程で更新されます。カタログ視点のデータは、通常の SQL 照会機能によって使用可能です。列の名前は、それが記述するオブジェクトのタイプに基づいて次のような一貫した名前が用いられています。

記述されるオブジェクト	列名
表	TABSCHEMA, TABNAME
索引	INDSCHEMA, INDNAME
視点	VIEWSCHEMA, VIEWNAME
制約	CONSTSCHEMA, CONSTNAME
トリガー	TRIGSCHEMA, TRIGNAME
パッケージ	PKGSCHEMA, PKGNAME
タイプ	TYPESCHEMA, TYPENAME, TYPEID
関数	FUNCSCHEMA, FUNCNAME, FUNCID
列	COLNAME
属性	ATTR_NAME
スキーマ	SCHEMANAME
表スペース	TBSPACE
ノードグループ	NGNAME
バッファー・プール	BPNAME

イベント・モニター	EVMONNAME
タイム・スタンプ	CREATE_TIME

カタログ視点の「ロードマップ」

説明	カタログ視点	ページ
索引	OBJCAT.INDEXES	1340
索引活用規則	OBJCAT.INDEXEXPLOITRULES	1343
索引拡張従属関係	OBJCAT.INDEXEXTENSIONDEP	1344
索引拡張メソッド	OBJCAT.INDEXEXTENSIONMETHODS	1345
索引拡張パラメーター	OBJCAT.INDEXEXTENSIONPARMS	1346
索引拡張	OBJCAT.INDEXEXTENSIONS	1347
述部指定	OBJCAT.PREDICATESPECS	1348
変形	OBJCAT.TRANSFORMS	1349

OBJCAT.INDEXES

表 108. OBJCAT.INDEXES カタログ視点

列名	データ・タイプ	ヌル値可	説明
INDSCHEMA	VARCHAR(128)		索引の名前。
INDNAME	VARCHAR(18)		
DEFINER	VARCHAR(128)		索引を作成したユーザー。
TABSCHEMA	VARCHAR(128)		その索引の定義されている表またはニックネームの修飾名。
TABNAME	VARCHAR(128)		
COLNAMES	VARCHAR (640)		列名の先頭に、昇順か降順かを示す + または - を付けたもののリスト。警告: この列は将来除去されます。詳しくは、1274ページの『SYSCAT.INDEXCOLUSE』を参照してください。
UNIQUERULE	CHAR(1)		固有値に関する規則: D = 重複可 P = 1 次索引 U = 固有項目のみ可
MADE_UNIQUE	CHAR(1)		Y = 索引は元は非固有だったが、固有キー制約または基本キー制約をサポートするために、固有索引に変換された。制約が除去されると、この索引は非固有に戻る。 N = 索引は作成時のまま。
COLCOUNT	SMALLINT		キー内の列数と組み込み列 (存在する場合) の数の合計。
UNIQUE_COLCOUNT	SMALLINT		固有キーに必要な列の数。常に COLCOUNT 以下。組み込み列がある場合にのみ COLCOUNT より少ない。索引に固有キーがない場合は -1 (重複可能)。
INDEXTYPE	CHAR(4)		索引のタイプ。 CLUS = クラスター化 REG = 通常
ENTRYTYPE	CHAR(1)		H = 階層表の索引 L = タイプ付き表の論理索引 非タイプ付き表の索引の場合は、ブランク

表 108. OBJCAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
PCTFREE	SMALLINT		索引を最初に作成する際に予約する索引葉ページのパーセンテージ。このスペースは、索引の作成後に行う挿入用に使用可能です。
IID	SMALLINT		内部索引 ID
NLEAF	INTEGER		葉ページの数。統計が収集されていない場合は -1。
NLEVELS	SMALLINT		索引レベルの数。統計が収集されていない場合は -1。
FIRSTKEYCARD	BIGINT		最初のキーの値の種類数 (統計が収集されていない場合は -1)。
FIRST2KEYCARD	BIGINT		索引の最初の 2 つの列を使用するキーの種類数 (統計が収集されない場合、または適用されない場合は -1)。
FIRST3KEYCARD	BIGINT		索引の最初の 3 つの列を使用するキーの種類数 (統計が収集されない場合、または適用されない場合は -1)。
FIRST4KEYCARD	BIGINT		索引の最初の 4 つの列を使用するキーの種類数 (統計が収集されない場合、または適用されない場合は -1)。
FULLKEYCARD	BIGINT		キー全体の値の種類数。統計が収集されていない場合は -1。
CLUSTERRATIO	SMALLINT		索引によるデータ・クラスター化の程度。統計が収集されていない場合、または詳細な索引統計が収集されている場合は -1 (それらの場合は CLUSTERFACTOR のほうが使用されます)。
CLUSTERFACTOR	DOUBLE		クラスター化の程度の詳細測定値。詳細索引統計が収集されていない場合、またはニックネームの索引が定義されていない場合は -1。
SEQUENTIAL_PAGES	INTEGER		索引キーの順序でディスクに存在し、それらの間に大きなギャップがなく、わずかなギャップしかない葉ページの数 (統計が使用できない場合は -1)。
DENSITY	INTEGER		索引によって占有されているページの範囲内の、ページ数に対する SEQUENTIAL_PAGES の比率。パーセントで表現される (0 ～ 100 の整数。統計が入手できない場合は -1)。

表 108. OBJCAT.INDEXES カタログ視点 (続き)

列名	データ・タイプ	ヌル値可	説明
USER_DEFINED	SMALLINT		この索引がユーザー定義であって除去されていない場合は 1、それ以外の場合は 0。
SYSTEM_REQUIRED	SMALLINT		この索引が基本または固有キー制約に必要であるか、またはこの索引がタイプ付き表のオブジェクト識別子 (OID) 列上にある索引の場合は 1。 この索引が基本キー制約または固有キー制約に必要であり、しかもタイプ付き表のオブジェクト識別子 (OID) 列の索引である場合は 2。 それ以外の場合は 0。
CREATE_TIME	TIMESTAMP		索引の作成された時刻。
STATS_TIME	TIMESTAMP	Yes	この索引について記録されている統計値が最後に変更された時刻。統計が使用可能でない場合は、ヌル値。
PAGE_FETCH_PAIRS	VARCHAR(254)		文字形式で表される、整数の対のリスト。それぞれの対は、仮のバッファー内のページ数と、その仮のバッファーを使用した表の走査に必要なページ取り出しの回数を表しています。 (データが入手できない場合は、長さ 0 のストリング)。
MINPCTUSED	SMALLINT		ゼロでない場合は、オンライン索引再編成が使用可能になり、その値は、ページをマージをする前に使用される最小スペースのしきい値です。
REVERSE_SCANS	CHAR(1)		Y = 索引は逆走査をサポートする N = 索引は逆走査をサポートしない
INTERNAL_FORMAT	SMALLINT		索引の内部表記をエンコードします。
IESCHEMA	VARCHAR(128)	Yes	索引拡張の修飾名。通常の索引の場合はヌル。
IENAME	VARCHAR(18)	Yes	
IEARGUMENTS	CLOB(32K)	Yes	索引の作成時に指定されるパラメーターの外部情報。通常の索引の場合はヌル。
REMARKS	VARCHAR(254)	Yes	ユーザー提供のコメントまたはヌル値。

OBJCAT.INDEXEXPLOITRULES

各行は、索引活用を表します。

表 109. OBJCAT.INDEXEXPLOITRULES カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCID	INTEGER		関数 ID。
SPECID	SMALLINT		CREATE FUNCTION ステートメント内の述部指定の数。
IESCHEMA	VARCHAR(128)		索引拡張の修飾名。
IENAME	VARCHAR(18)		
RULEID	SMALLINT		固有の活用規則 ID。
SEARCHMETHODID	SMALLINT		特定の索引拡張での検索メソッド ID。
SEARCHKEY	VARCHAR(320)		索引を活用するのに使用するキー。
SEARCHARGUMENT	VARCHAR(1800)		索引活用で使用する検索引き数。

OBJCAT.INDEXEXTENSIONDEP

さまざまなデータベース・オブジェクトに対する索引拡張の従属関係ごとに、1つの行が含まれます。

表 110. OBJCAT.INDEXEXTENSIONDEP カタログ視点

列名	データ・タイプ	ヌル値可	説明
IESCHEMA	VARCHAR(128)		別のオブジェクトに従属する索引拡張の修飾名。
IENAME	VARCHAR(18)		
BTYPE	CHAR(1)		索引拡張が従属しているオブジェクトのタイプ。 A = 別名 F = 関数インスタンスまたはメソッド・インスタンス J = サーバー定義 O = 階層 SELECT 特権への "外部" 従属関係 R = 構造タイプ S = 要約表 T = 表 (タイプ付きではない) U = タイプ付き表 V = 視点 (タイプ付きではない) W = タイプ付き視点 X = 索引の拡張
BSCHEMA	VARCHAR(128)		索引拡張が従属しているオブジェクトの修飾名 (BTYPE='F' の場合、これは関数の固有名になります)。
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	BTYPE='O'、'T'、'U'、'V'、または 'W' の場合、従属トリガーで必要な表 (または視点) に対する特権を示すコード。それ以外の場合はヌル。

OBJCAT.INDEXEXTENSIONMETHODS

各行は、検索メソッドを表します。 1 つの索引拡張に、複数の検索メソッドを含めることができます。

表 111. OBJCAT.INDEXEXTENSIONMETHODS カタログ視点

列名	データ・タイプ	ヌル値可	説明
METHODNAME	VARCHAR(18)		検索メソッドの名前。
METHODID	SMALLINT		索引拡張内のメソッドの数。
IESCHEMA	VARCHAR(128)		索引拡張の修飾名。
IENAME	VARCHAR(18)		
RANGEFUNCSCHEMA	VARCHAR(128)		範囲指定関数の修飾名。
RANGEFUNCNAME	VARCHAR(18)		
RANGESPECIFICNAME	VARCHAR(18)		範囲指定関数の固有名。
FILTERFUNCSCHEMA	VARCHAR(128)		フィルター関数の修飾名。
FILTERFUNCNAME	VARCHAR(18)		
FILTERSPECIFICNAME	VARCHAR(18)		フィルター関数の関数固有名。
REMARKS	VARCHAR(254)	Yes	ユーザー提供またはヌル。

OBJCAT.INDEXEXTENSIONPARMS

各行は、索引拡張のインスタンス・パラメーターまたはソース・キー定義を表します。

表 112. OBJCAT.INDEXEXTENSIONPARMS カタログ視点

列名	データ・タイプ	ヌル値可	説明
IESCHEMA	VARCHAR(128)		索引拡張の修飾名。
IENAME	VARCHAR(18)		
ORDINAL	SMALLINT		パラメーターまたはソース・キーのシーケンス番号。
PARMNAME	VARCHAR(18)		パラメーターまたはソース・キーの名前。
TYPESCHEMA	VARCHAR(128)		インスタンス・パラメーターまたはソース・キーのデータ・タイプの修飾名。
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		インスタンス・パラメーターまたはソース・キーのデータ・タイプの長さ。
SCALE	SMALLINT		インスタンス・パラメーターまたはソース・キーのデータ・タイプの位取り。適用できない場合はゼロ (0)。
PARMTYPE	CHAR(1)		以下のように、行によって表されるタイプ。 P = 索引拡張パラメーター K = キー列
CODEPAGE	SMALLINT		索引拡張パラメーターのコード・ページ。ストリング・タイプでない場合はゼロ。

OBJCAT.INDEXEXTENSIONS

索引拡張ごとに 1 つの行が含まれます。

表 113. OBJCAT.INDEXEXTENSIONS カタログ視点

列名	データ・タイプ	ヌル値可	説明
IESCHEMA	VARCHAR(128)		索引拡張の修飾名。
IENAME	VARCHAR(18)		
DEFINER	VARCHAR(128)		索引拡張の定義時の許可 ID。
CREATE_TIME	TIMESTAMP		索引拡張が定義された時刻。
KEYGENFUNCSCHEMA	VARCHAR(128)		キー生成関数の修飾名。
KEYGENFUNCNAME	VARCHAR(18)		
KEYGENSPECIFICNAME	VARCHAR(18)		キー生成関数の固有名。
TEXT	CLOB(64K)		CREATE INDEX EXTENSION ステートメントのテキスト全体。
REMARKS	VARCHAR(254)		ユーザー提供のコメントまたはヌル値。

OBJCAT.PREDICATESPECS

表 114. OBJCAT.PREDICATESPECS カタログ視点

列名	データ・タイプ	ヌル値可	説明
FUNCSCHEMA	VARCHAR(128)		関数の修飾名。
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		関数インスタンスの名前。
FUNCID	INTEGER		関数 ID。
SPECID	SMALLINT		この述部指定の ID。
CONTEXTOP	CHAR(8)		比較演算子は組み込み関係演算子 (=、<、>= など) のいずれか。
CONTEXTEXP	CLOB(32K)		定数、または SQL 式。
FILTERTEXT	CLOB(32K)	Yes	データ・フィルター式のテキスト。

OBJCAT.TRANSFORMS

指名された変形グループに含まれているユーザー定義タイプ内の変形関数タイプごとに、 1 つの行が含まれます。

表 115. OBJCAT.TRANSFORMS カタログ視点

列名	データ・タイプ	ヌル値可	説明
TYPEID	SMALLINT		SYSCAT.DATATYPES で定義されている内部タイプ ID。
TYPESCHEMA	VARCHAR(128)		特定のユーザー定義構造タイプの修飾名。
TYPENAME	VARCHAR(18)		
GROUPNAME	VARCHAR(18)		変形グループ名。
FUNCID	INTEGER	Yes	SYSCAT.FUNCTIONS で定義されている、関連する変形関数の内部関数 ID。ヌルになるのは、内部システム関数の場合だけです。
FUNCSCHEMA	VARCHAR(128)		関連する変形関数の修飾名。
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		関数の特定名 (インスタンス名)。
TRANSFORMTYPE	VARCHAR(8)		'FROM SQL' = 変形関数は SQL から構造タイプを変形する 'TO SQL' = 変形関数は SQL に構造タイプを変形する
FORMAT	CHAR(1)		'U' = ユーザー定義
MAXLENGTH	INTEGER	Yes	FROM SQL 変形からの出力の最大長 (バイト単位)。 TO SQL 変形の場合はヌル。
ORIGIN	CHAR(1)		'I' = 継承された下位のタイプ階層。 'U' = ユーザー定義。
REMARKS	VARCHAR(254)	Yes	ユーザー提供コメントまたはヌル。

付録F. 連合システム

この付録には、以下の点が記されています。

- DB2 連合システムを確立して使用するために SQL ステートメントに定義できるサーバー・タイプ
- DB2 連合システムを確立して使用するために SQL ステートメントに定義できるオプション
- 連合サーバーがサポートするデータ・タイプとデータ・ソースがサポートするデータ・タイプの間のデフォルト・マッピング
- パススルーの使用時に考慮すべき要素と守るべき制約事項

サーバー・タイプ

サーバー・タイプは、サーバーが表すデータ・ソースの種類を示します。サーバー・タイプは、ベンダー、目的、およびプラットフォームによって異なります。サポートされる値は、使用するラッパーによって異なります。

- **DRDA ラッパー**

  DB2 ファミリー

表 116. IBM DB2 ユニバーサル・データベース

サーバー・タイプ	データ・ソース
DB2/UDB	IBM DB2 ユニバーサル・データベース
DataJoiner	IBM DB2 DataJoiner V2.1 および V2.1.1
DB2/6000	DB2 (AIX 版)
DB2/HPUX	IBM DB2 (HP-UX 版) V1.2
DB2/NT	IBM DB2 (Windows NT 版)
DB2/EEE	IBM DB2 エンタープライズ拡張エディション
DB2/SUN	IBM DB2 (Solaris 実行環境版) V1 および V1.2
DB2/2	IBM DB2 (OS/2 版)
DB2/LINUX	IBM DB2 (Linux 版)
DB2/DYNIX/ptx	IBM DB2 (NUMA-Q 版)
DB2/SCO	IBM DB2 (SCO UnixWare 版)

表 117. IBM DB2 ユニバーサル・データベース (AS/400 版)

サーバー・タイプ	データ・ソース
DB2/400	IBM DB2 (AS/400 版)

表 118. IBM DB2 ユニバーサル・データベース (OS/390 版)

サーバー・タイプ	データ・ソース
DB2/390	IBM DB2 (OS/390 版)
DB2/MVS	IBM DB2 (MVS 版)

表 119. IBM DB2 サーバー (VM および VSE 版)

サーバー・タイプ	データ・ソース
DB2/VM	IBM DB2 (VM 版)
DB2/VSE	IBM DB2 (VSE 版)
SQL/DS	IBM SQL/DS

- **SQLNET ラッパー**

  Oracle データ・ソースは、 Oracle SQL*Net V1 または V2 クライアント・ソフトウェアによってサポートされています。

サーバー・タイプ	データ・ソース
ORACLE	Oracle V7.0.13 以降

- **NET8 ラッパー**

  Oracle データ・ソースは、Oracle Net8 クライアント・ソフトウェアによってサポートされています。

サーバー・タイプ	データ・ソース
ORACLE	Oracle V7.0.13 以降

- **OLE DB ラッパー**

  Microsoft OLE DB 2.0 以降に対応している OLE DB Provider。

サーバー・タイプ	データ・ソース
-	すべての OLE DB Provider。

- **他のラッパー**

  ラッパーに付属している資料を参照してください。

連合システム用の SQL オプション

この節には、以下の点が記されています。

- ALTER NICKNAME ステートメントで指定できる列オプション
- CREATE FUNCTION MAPPING ステートメントで指定できる関数マッピング・オプション
- CREATE SERVER、ALTER SERVER、および SET SERVER OPTION ステートメントで指定できるサーバー・オプション
- CREATE USER MAPPING および ALTER USER MAPPING ステートメントで指定できるユーザー・オプション

列オプション

列オプションの主な目的は、SQL コンパイラーにニックネーム列の情報を提供することです。 1 つまたは複数の列の列オプションを 'Y' に設定すると、コンパイラーは、評価操作を実行する述部でのプッシュダウンの可能性をさらに考慮できます。プッシュダウン処理についての詳細は、*管理の手引き: パフォーマンス* を参照してください。

表 120. 列オプションとその設定値

オプション	有効な設定値		デフォルト設定値
numeric_string	'Y'	はい - この列には数値データのストリングだけが含まれます。重要: この列に、後書きブランクの付いた数値ストリングだけが含まれる場合、 'Y' を指定することはお勧めできません。	'N'
	'N'	いいえ - この列は数値データのストリングに限定されていません。	
	列の numeric_string を 'Y' に設定すると、列データの分類に干渉するブランクがこの列には含まれないことを、最適化プログラムに知らせることになります。このオプションは、データ・ソースの照合順序が DB2 の照合順序とは異なる場合に役立ちます。このオプションでマークされた列は、照合順序が異なるためにローカルな (データ・ソースの) 評価から除かれるということはありません。		

表 120. 列オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
varchar_no_trailing_blanks	特定の VARCHAR 列に後書きブランクがないかどうかを示します。 'Y' はい - この VARCHAR 列に後書きブランクはありません。 'N' いいえ - この VARCHAR 列には後書きブランクがあります。 データ・ソース VARCHAR 列に埋め込みブランクがない場合、最適化プログラムがその列にアクセスするときの戦略は、列に後書きブランクが含まれているかどうかによって異なる部分があります。デフォルトでは、最適化プログラムは後書きブランクを実際に含んでいるものと「想定」しています。この想定に基づき、これらの列から返された値が希望する値になるように照会を変更するアクセス戦略が開発されます。ただし、VARCHAR 列に後書きブランクがなく、そのことを最適化プログラムに知らせてある場合、さらに効果的なアクセス戦略を開発することができます。特定の列に後書きブランクがないことを最適化プログラムに知らせるには、 ALTER NICKNAME ステートメントでその列を指定してください (構文については、 *SQL 解説書* を参照してください)。	'N'

関数マッピング・オプション

関数マッピング・オプションの主な目的は、データ・ソースでデータ・ソース関数を実行した場合のコストに関する情報を提供することです。プッシュダウン分析により、マッピング内で 2 つの関数のいずれかが呼び出せると判断された場合、最適化プログラムはマッピング定義で提供される統計情報を使って、データ・ソース関数の実行コストの見積もりを DB2 関数の実行コストの見積もりと比較することができます。

表 121. 関数マッピング・オプションとその設定値

オプション	有効な設定値	デフォルト設定値
disable	デフォルト関数マッピングを使用不可にする。有効値は 'Y' および 'N'。	'N'
initial_insts	データ・ソース関数が呼び出される最初と最後のときに処理される命令数の見積もり。	'0'

表 121. 関数マッピング・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
initial_ios	データ・ソース関数が呼び出される最初と最後のときに実行される入出力回数の見積もり。	'0'
ios_per_argbyte	データ・ソース関数に渡される引き数セットのバイトごとに使用される入出力回数の見積もり。	'0'
ios_per_invoc	データ・ソース関数の呼び出しごとの入出力回数の見積もり。	'0'
insts_per_argbyte	データ・ソース関数に渡される引き数セットのバイトごとに処理される命令数の見積もり。	'0'
insts_per_invoc	データ・ソース関数の呼び出しごとに処理される命令数の見積もり。	'450'
percent_argbytes	データ・ソース関数が実際に読み取る入力引き数バイトの平均パーセント値の見積もり。	'100'
remote_name	データ・ソース関数の名前。	ローカル名

サーバー・オプション

サーバー・オプションは、サーバーを記述するために使用します。位置情報 (データ・ソース・マシンの名前など) に加えて、オプションでは、データ・ソースの機密保護およびパフォーマンス属性を指定できます。機密保護オプションは、パスワード通信の制御 (データ・ソースに送信される場合もされない場合もある) と、認証情報 (大文字小文字を問わない ID およびパスワード) を提供します。パフォーマンス・オプションは、最適化プログラムが、データ・ソースで評価操作を実行できるかどうかを判別し、データ・ソースからデータを検索する照会を完了するための最適コスト・モデルを判別するのに役立ちます。

表 122. サーバー・オプションとその設定値

オプション	有効な設定値	デフォルト設定値
collating_sequence	データ・ソースが連合データベースと同じデフォルト照合順序を使用しているかどうかを、コード・セットと国別情報に基づいて指定する。データ・ソースの照合順序が DB2 の照合順序と異なっている場合、 DB2 の照合順序に依存しているほとんどの操作は、データ・ソースにおいてリモートに評価できません。一例として、データ・ソースのあるニックネーム文字列に対して異なる照合順序で MAX 列関数を実行する場合があります。 MAX 関数がリモート・データ・ソースにおいて評価される場合は結果が異なるので、 DB2 は集約操作と MAX 関数をローカルに実行します。 照会に等号が含まれている場合は、照合順序が異なっていても ('N' に設定されている)、照会のその部分をプッシュダウンすることができます。たとえば、述部 C1 = 'A' をデータ・ソースにプッシュダウンすることができます。もちろん、データ・ソースでの照合順序で大文字小文字が区別されない場合、そのような照会をプッシュダウンすることはできません。データ・ソースが大文字小文字を区別しない場合、 C1= 'A' と C1 = 'a' の結果は同じです。これは大文字小文字の区別を行う環境 (DB2) では受け入れられません。 管理者は、データ・ソースの照合順序と一致する特定の照合順序の連合データベースを作成できます。この方法を用いると、すべてのデータ・ソースが同じ照合順序を使用する場合、または、ほとんどあるいはすべての列関数が同じ照合順序を使用するデータ・ソースに向けられている場合は、パフォーマンスが高速になります。 'Y' データ・ソースの照合順序は、連合データベースの照合順序と同じ。 'N' データ・ソースの照合順序は、連合データベースの照合順序と異なる。 'I' データ・ソースの照合順序は、連合データベースの照合順序と異なっており、大文字小文字を区別しない (たとえば、'TOLLESON' と 'TolLESon' は同じものとみなされる)	'N'

表 122. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
comm_rate	連合サーバーとその関連データ・ソース間の通信速度を指定する。秒当たりの MB 単位で表されます。 有効な値は 1 ～ 2147483647 です。値は整数だけ (たとえば 12) で表さなければなりません。	'2'
connectstring	OLE DB Provider への接続に必要な初期化特性を指定します。接続ストリングの詳細な構文と意味については、 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, Microsoft Press, 1998* の『Data Link API of the OLE DB Core Components』を参照してください。	なし
cpu_ratio	データ・ソースの CPU が連合サーバーの CPU より、どれほど速いまたは遅いかを表す。 有効な値は、0 より大きく $1x10^{23}$ より小さい値です。値は任意の有効な double 表記 (たとえば、123E10、123、または 1.21E4) で表すことができます。	'1.0'
dbname	連合サーバーがアクセスするデータ・ソース・データベースの名前。 DB2 ファミリー・データ・ソースでは必須ですが、Oracle** データ・ソースには適用されません。これは、Oracle インスタンスにはデータベースが 1 つしか含まれていないためです。 DB2 の場合、この値はインスタンス内の特定のデータベース (DB2 (OS/390 版) では、データベース LOCATION 値) に対応しています。	なし。

表 122. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
fold_id (この表の最後にある注 1 および 4 を参照。)	連合サーバーが認証のためにデータ・ソースに送信するユーザー ID に適用される。有効な値は以下のとおりです。 'U' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID を大文字に変換します。これは、DB2 ファミリーおよび Oracle** データ・ソースについて論理的に選択できます (この表の最後にある注 2 を参照)。 'N' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID に対して何の処理も行いません。 (この表の最後の注 2 を参照。) 'L' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID を小文字に変換します。 これらの設定値のいずれも使用しない場合は、連合サーバーはユーザー ID を大文字にしてデータ・ソースに送信しようとします。そのユーザー ID を正常に送信できない場合は、サーバーはユーザー ID を小文字で送信しようとします。	なし。
fold_pw (この表の最後にある注 1、3 および 4 を参照。)	連合サーバーが認証のためにデータ・ソースに送信するパスワードに適用される。有効な値は以下のとおりです。 'U' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードを大文字に変換します。これは、DB2 ファミリーおよび Oracle** データ・ソースについて論理的に選択できます。 'N' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードに対して何の処理も行いません。 'L' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードを小文字に変換します。 これらの設定値のいずれも使用しない場合は、連合サーバーはパスワードを大文字にしてデータ・ソースに送信しようとします。そのパスワードを正常に送信できない場合は、サーバーはパスワードを小文字で送信しようとします。	なし。

表 122. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
io_ratio	データ・ソースの入出力システムが連合サーバーの入出力システムより、どれほど速いまたは遅いかを表す。 有効な値は、0 より大きく $1x10^{23}$ より小さい値です。値は任意の有効な double 表記 (たとえば、123E10、123、または 1.21E4) で表すことができます。	'1.0'
node	データ・ソースを RDBMS のインスタンスとして定義するための名前。すべてのデータ・ソースについて必須です。 DB2 ファミリー・データ・ソースの場合、これは、連合データベースの DB2 ノード・ディレクトリーで指定されているノードです。このディレクトリーを表示するには、 **db2 list node directory** コマンドを発行します。 Oracle** データ・ソースの場合、この名前は Oracle** tnsnames.ora ファイルで指定されているサーバー名です。Windows NT プラットフォームでこの名前にアクセスするには、 Oracle** SQL Net Easy Configuration ツールの「**View Configuration Information**」オプションを指定します。	なし。
password	パスワードがデータ・ソースに送信されるかどうかを指定する。 'Y' パスワードは常にデータ・ソースに送信されて妥当性検査されます。 これはデフォルト値です。 'N' パスワードはデータ・ソースに送信されず (ユーザー・マッピングに関係なく)、妥当性検査されません。 'ENCRYPTION' パスワードは常に暗号化された形式でデータ・ソースに送信されて妥当性検査されます。暗号化されたパスワードをサポートする DB2 ファミリー・データ・ソースについてのみ有効です。	'Y'

表 122. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
plan_hints	**プラン・ヒント**を使用可能にするかどうかを指定する。プラン・ヒントはステートメントの一部分であり、データ・ソース最適化プログラムについての追加情報を提供します。特定の照会タイプについてこの情報を利用すれば、照会パフォーマンスを改善することができます。プラン・ヒントは、データ・ソース最適化プログラムが索引を使用するかどうか、どの索引を使用するか、またはどの表結合順序を使うかを判別するのに役立ちます。 'Y' データ・ソースがプラン・ヒントをサポートしている場合は、プラン・ヒントが使用可能になります。 'N' プラン・ヒントはデータ・ソースで使用可能になりません。	'N'
pushdown	'Y' DB2 はデータ・ソースに操作を評価させることを考慮します。 'N' DB2 はリモート・データ・ソースからの列しか検索せず、データ・ソースに結合などのその他の操作を評価させません。	'Y'
varchar_no_trailing_blanks	このデータ・ソースが、ブランクが埋め込まれていない varchar 比較セマンティクスを使用するかどうかを指定する。後書きブランクを含んでいない可変長文字ストリングについて、一部の DBMS のブランク埋め込みなしの比較セマンティクスでは、 DB2 の比較セマンティクスと同じ結果が戻されます。データ・ソースにあるすべての VARCHAR 表 / 視点の列に後書きブランクが含まれていないことが確かである場合は、データ・ソースについてこのサーバー・オプションを 'Y' に設定することを考慮してください。このオプションは、Oracle** データ・ソースでしばしば使用されます。ニックネーム (視点を含む) を持つ可能性のあるすべてのオブジェクトを考慮に入れてください。 'Y' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、DB2 と同じです。 'N' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、 DB2 と同じではありません。	'N'

1356ページの表122 の注:

1. このフィールドは、認証に指定される値に関係なく適用されます。
2. DB2 はユーザー ID を大文字で保管するので、値 'N' と 'U' は論理的に互いに同等です。
3. パスワードの設定が 'N' である場合は、fold_pw を設定しても効果はありません。パスワードが送信されないので、大文字小文字の区別は意味をなしません。
4. いずれかのオプションについてヌル値設定を使用することは避けてください。 DB2 がユーザー ID とパスワードの解決を複数回試行することになるので、ヌル値設定は有用に思えますが、パフォーマンスが低下する場合があります (DB2 が、データ・ソース認証に成功するまでにユーザー ID とパスワードを 4 回送信するということもあり得ます)。

ユーザー・オプション

ユーザー・オプションは、ユーザー・マッピングの許可および会計ストリング情報を提供します。これらのオプションを使用すると、データ・ソースでの認証時に DB2 認証 ID を表すための ID およびパスワードを指定できます。

表 123. ユーザー・オプションとその設定値

オプション	有効な設定値	デフォルト設定値
remote_authid	データ・ソースでの許可 ID を示す。有効な設定値には、長さ 255 以下の任意のストリングが含まれます。このオプションを指定しない場合、データベースへの接続に使用された ID が使用されます。	なし。
remote_domain	このデータ・ソースへの接続に関する認証をユーザーに与えるのに使用される、 Windows NT ドメインを示します。有効な設定には、何らかの有効な Windows NT ドメイン名が含まれます。このオプションを指定しない場合、データ・ソースへの認証は、そのデータベースのデフォルト認証ドメインを使用して行われます。	なし。
remote_password	データ・ソースでの許可パスワードを示す。有効な設定値には、長さ 32 以下の任意のストリングが含まれます。このオプションを指定しない場合、データベースへの接続に使用されたパスワードが使用されます。	なし。

表 123. ユーザー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定値
accounting_string	DRDA 会計ストリングを指定するために使用される。有効な設定値には、長さ 255 以下の任意のストリングが含まれます。このオプションは、会計情報を渡す必要がある場合にのみ必須です。 *DB2 コネクト 使用者の手引き* を参照。	なし。

デフォルト・データ・タイプ・マッピング

この節では、連合サーバーがサポートする DB2 データ・タイプと、以下のデータ・ソースがサポートするデータ・タイプのデフォルト・マッピングを示します。

- DB2 ユニバーサル・データベース (OS/390 版) および DB2 (MVS/ESA 版)
- DB2 ユニバーサル・データベース (AS/400 版) および DB2 (AS/400 版)
- Oracle
- DB2 (VM および VSE 版)、SQL/DS

ここに示したのは、異なるデータ・タイプ間のマッピングです。同一データ・タイプ間のマッピングは示されていません。

DB2 と DB2 ユニバーサル・データベース (OS/390 版) (および DB2 (MVS/ESA 版)) のデータ・ソース間のデフォルト・タイプ・マッピング

表 124. DB2 と DB2 ユニバーサル・データベース (OS/390 版) (および DB2 (MVS/ESA 版)) のデータ・ソース間のデフォルト・タイプ・マッピング

DB2 (OS/390 版)	DB2
varchar(n)、n <= 32672	varchar(n)
vargraphic(n)、n <= 16336	vargraphic(n)
char(255)	varchar(255)
ビット・データの場合は char(255)	ビット・データの場合は varchar(255)

DB2 と DB2 ユニバーサル・データベース (AS/400 版) (および DB2 (AS/400 版)) のデータ・ソース間のデフォルト・タイプ・マッピング

表 125. DB2 と DB2 ユニバーサル・データベース AS/400 用 (および DB2 (AS/400 版)) のデータ・ソース間のデフォルト・タイプ・マッピング

DB2 (AS/400 版)	DB2
char(n)、n <= 254	char(n)
char(n)、255 ～ 32672 の n	varchar(n)
varchar(n)、n <= 32672	varchar(n)
graphic(n)、n <= 127	graphic(n)
graphic(n)、127 ～ 16336 の n	vargraphic(n)
vargraphic(n)、n <= 16336	vargraphic(n)

DB2 と Oracle のデータ・ソース間のデフォルト・タイプ・マッピング

表 126. DB2 と Oracle のデータ・ソース間のデフォルト・タイプ・マッピング

Oracle	DB2
rowid	char(18)
char(n)、n <= 254	char(n)
nchar(n)、n <= 254	char(n)
char(255)	varchar(255)
varchar2(n)、n <= 32672	varchar(n)
nvarchar2(n)、n <= 32672	varchar(n)
number(p,s)、p <= 4 および s = 0	smallint
number(p,s)、4 <= p <= 9 および s = 0	integer
number(p,s)、10 <= p <= 18 および s = 0	bigint
number(p,s)、p <= 31 および 0 <= s <= p、そして前の 2 つの文字種は一致しない	decimal
number(p,s)、以前の 4 以外の文字種すべて	double
raw(n)、n <= 254	ビット・データの場合は char(n)
raw(255)	ビット・データの場合は varchar(255)
date (char(9))	timestamp

DB2 と DB2 (VM および VSE 版) (および SQL/DS) のデータ・ソース間のデフォルト・タイプ・マッピング

表 127. DB2 と DB2 (VM および VSE 版) (および SQL/DS) のデータ・ソース間のデフォルト・タイプ・マッピング

DB2 (OS/390 版)、SQL/DS	DB2
varchar(n)、n <= 32672	varchar(n)
vargraphic(n)、n <= 16336	vargraphic(n)

パススルー機能の処理

パススルー と呼ばれる機能は、あるデータ・ソースに固有の SQL のデータ・ソースを照会する場合に使用できます。この節の内容は以下のとおりです。

- 連合サーバーと関連データ・ソースがパススルー・セッションで処理する SQL ステートメントの種類。
- パススルーを使用するときに留意すべき考慮事項と制約事項。

パススルー・セッションにおける SQL 処理

以下の規則により、 SQL ステートメントは DB2 とデータ・ソースのいずれかによって処理されます。

- パススルー・セッションで処理するように、 SQL ステートメントをデータ・ソースに実行依頼する場合、その SQL ステートメントは、そのセッションで動的に準備され、セッションがまだオープンしている間に実行されなければなりません。これを行うには、以下のいくつかの方法があります。
  - SELECT ステートメントを実行依頼する場合、PREPARE ステートメントを使用して準備してから、 OPEN、FETCH、および CLOSE ステートメントを使用して、照会の結果にアクセスします。
  - SELECT 以外のサポートされているステートメントを使用する場合、以下のいずれかを行います。
    - PREPARE ステートメントを使用して、サポートされているステートメントを準備してから、 EXECUTE ステートメントを使用して実行します。
    - EXECUTE IMMEDIATE ステートメントを使用して、準備と実行を行います。
- パススルー・セッションで静的ステートメントが実行依頼された場合、そのステートメントは連合サーバーに送られて処理されます。

- パススルー・セッション中に COMMIT または ROLLBACK コマンドが発行されると、そのコマンドは現行の作業単位 (UOW) を完了させます。

考慮事項と制約事項

パススルーに適用される考慮事項と制約事項は多岐にわたります。一般的な性質のものもあれば、Oracle データ・ソースだけに関するものもあります。

すべてのデータ・ソースでのパススルーの使用

以下の情報は、すべてのデータ・ソースに適用されます。

- パススルー・セッション内で準備したステートメントは、同じパススルー・セッション内で実行しなければなりません。パススルー内で準備したステートメントを、そのパススルー・セッション以外で実行すると、失敗します (SQLSTATE 56098)。
- ユーザーおよびアプリケーションはパススルーを使ってデータ・ソースへの書き込みを行えます。たとえば、表の行を挿入、更新、削除できます。注意点として、パススルー・セッションでデータ・ソース・オブジェクトに対してカーソルを直接オープンすることはできません (SQLSTATE 25000)。
- アプリケーションでは、実際にいくつかの SET PASSTHRU ステートメントを異なるデータ・ソースに対して同時に使用できます。アプリケーションが複数の SET PASSTHRU ステートメントを発行しても、パススルー・セッションが実際にネストされるわけではありません。連合サーバーはデータ・ソース間のパススルーを行いません。むしろ、サーバーは、個々のデータ・ソースに直接アクセスします。
- 複数のパススルー・セッションが同時にオープンされる場合、各セッション内の個々の作業単位は COMMIT または ROLLBACK ステートメントで終わる必要があります。こうすればセッションを、 SET PASSTHRU ステートメントと RESET オプションを指定した 1 つの操作で終了することができます。
- 1 回に複数のデータ・ソースをパススルーすることはできません。
- パススルーは、ストアード・プロシージャーの呼び出しをサポートしません。
- パススルーでは、SELECT INTO ステートメントはサポートされていません。

Oracle データ・ソースでのパススルーの使用

以下の情報は、Oracle データ・ソースに適用されます。

- リモート・クライアントがコマンド行プロセッサー (CLP) から SELECT ステートメントを発行すると、次の制約事項が適用されます。すなわち、クラ

イアント・コードが DB2 ユニバーサル・データベース バージョン 5 以前の DB2 SDK であれば、 SELECT は SQLSTATE 25000 を誘発します。このエラーを回避したい場合、リモート・クライアントはバージョン 5 以上の DB2 SDK を使用しなければなりません。

- Oracle サーバーに対して発行された DDL ステートメントは解析時に実行され、トランザクションのセマンティクスには左右されません。操作は完了した時点で Oracle 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る場合、DDL はロールバックされません。
- 生データ・タイプから SELECT ステートメントを発行する場合は、RAWTOHEX 関数を呼び出して 16 進値を受け取るようにします。生データ・タイプに INSERT を実行する場合は、16 進表記で行います。

付録G. サンプル・データベース表

この付録では、サンプル・データベース SAMPLE のサンプル表に含まれている情報と、それを作成する方法、および除去する方法について示します。

ビジネス・インテリジェンス機能について説明するための、追加のサンプル・データベースが DB2 ユニバーサル・データベースに付属しています。これらはビジネス・インテリジェンスの学習で使用されます。しかし、サンプル・データベース SAMPLE の内容について説明しているのは、この付録だけです。ビジネス・インテリジェンスのサンプル・データベースについては、 データウェアハウスセンター *管理の手引き* を参照してください。

このサンプル表は、このマニュアルやこのライブラリーに属する他のマニュアルに含まれている例の中で使われているものです。また、サンプル・ファイルのうち、BLOB および CLOB データ・タイプのデータについても示します。

この付録の内容は、次のとおりです。

1383ページの『Nicholls の履歴書』
1385ページの『Adamson の写真』
1385ページの『Adamson の履歴書』
1386ページの『Walker の写真』
1386ページの『Walker の履歴書』

サンプル表のうち、ハイフン (-) は NULL 値を表しています。

サンプル・データベース

このマニュアルの例では、サンプル (SAMPLE) データベースを使っています。それらの例を使うには、 SAMPLE データベースを作成する必要があります。それを使うには、データベース・マネージャーがインストールされていなければなりません。

サンプル・データベースの作成

サンプル・データベースは、実行可能ファイルを使って作成します。[116] データベースを作成するためには、SYSADM 権限が必要です。

- **UNIX 系のプラットフォームを使う場合**

  オペレーティング・システムのコマンド・プロンプトを使っている場合は、次のように入力します。

  ```
  sqllib/bin/db2sampl <path>
  ```

  path は、サンプル・データベースの作成先のパスを指定するオプション・パラメーターです。 Enter キーを押します。[117] DB2SAMPL のスキーマは、CURRENT SCHEMA 特殊レジスター値です。

- **OS/2 または Windows プラットフォームを使う場合**

  オペレーティング・システムのコマンド・プロンプトを使っている場合は、次のように入力します。

  ```
  db2sampl e
  ```

  ここで、*e* はデータベースを作成するドライブを指定するオプション・パラメーターです。 Enter キーを押します。[118]

116. このコマンドについては、 *コマンド解説書* の中の DB2SAMPL コマンドの説明を参照してください。

117. path パラメーターを指定しないなら、サンプル・データベースはデータベース・マネージャー構成ファイルの中の DFTDBPATH パラメーターによって指定されるデフォルト・パスに作成されます。

118. ドライブ・パラメーターの指定がない場合は、サンプル・データベースは、DB2 と同じドライブに作成されます。

ユーザー・プロファイル管理によってワークステーションにログオンしていない場合には、ログオンを求めるプロンプトが出されます。

サンプル・データベースの消去

サンプル・データベースにアクセスする必要がなくなった場合には、 DROP DATABASE コマンドを使用して、サンプル・データベースを消去することができます。

```
db2 drop database sample
```

CL_SCHED 表

名前:	CLASS_CODE	DAY	STARTING	ENDING
タイプ:	char(7)	smallint	time	time
説明:	クラス・コード (教室: 教師)	4 日単位の予定の日番号	クラス開始時刻	クラス終了時刻

DEPARTMENT 表

名前:	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
タイプ:	char(3) 非ヌル値	varchar(29) 非ヌル値	char(6)	char(3) 非ヌル値	char(16)
説明:	部門番号	部門の一般的な活動を記述する名前	部門管理者の従業員番号 (EMPNO)	この部門の報告提出先の部門 (DEPTNO)	遠隔地の名前
値:	A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	-
	B01	PLANNING	000020	A00	-
	C01	INFORMATION CENTER	000030	A00	-
	D01	DEVELOPMENT CENTER	-	A00	-
	D11	MANUFACTURING SYSTEMS	000060	D01	-
	D21	ADMINISTRATION SYSTEMS	000070	D01	-
	E01	SUPPORT SERVICES	000050	A00	-
	E11	OPERATIONS	000090	E01	-
	E21	SOFTWARE SUPPORT	000100	E01	-

EMPLOYEE 表

名前:	EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
タイプ:	char(6) 非ヌル値	varchar(12) 非ヌル値	char(1) 非ヌル値	varchar(15) 非ヌル値	char(3)	char(4)	date
説明:	従業員番号	ファーストネーム (名)	ミドルネームのイニシャル	ラストネーム (姓)	従業員の所属部門 (DEPTNO)	電話番号	入社日

JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(8)	smallint 非ヌル値	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
担当業務	学校教育の年数	性別 (男性は M、女性は F)	誕生日	年収	年間ボーナス	年間歩合給

EMPLOYEE 表に含まれている値については、次のページ以降を参照してください。

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(6) 非ヌル値	varchar(12) 非ヌル値	char(1) 非ヌル値	varchar(15) 非ヌル値	char(3)	char(4)	date	char(8)	smallint 非ヌル値	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

EMP_ACT 表

名前:	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
タイプ:	char(6) 非ヌル値	char(6) 非ヌル値	smallint 非ヌル値	dec(5,2)	date	date
説明:	従業員番号	プロジェクト番号	活動番号	従業員がプロジェクトに費やす時間の割合	活動開始日	活動終了日
値:	000010	AD3100	10	.50	1982-01-01	1982-07-01
	000070	AD3110	10	1.00	1982-01-01	1983-02-01
	000230	AD3111	60	1.00	1982-01-01	1982-03-15
	000230	AD3111	60	.50	1982-03-15	1982-04-15
	000230	AD3111	70	.50	1982-03-15	1982-10-15
	000230	AD3111	80	.50	1982-04-15	1982-10-15
	000230	AD3111	180	1.00	1982-10-15	1983-01-01
	000240	AD3111	70	1.00	1982-02-15	1982-09-15
	000240	AD3111	80	1.00	1982-09-15	1983-01-01
	000250	AD3112	60	1.00	1982-01-01	1982-02-01
	000250	AD3112	60	.50	1982-02-01	1982-03-15
	000250	AD3112	60	.50	1982-12-01	1983-01-01
	000250	AD3112	60	1.00	1983-01-01	1983-02-01
	000250	AD3112	70	.50	1982-02-01	1982-03-15
	000250	AD3112	70	1.00	1982-03-15	1982-08-15
	000250	AD3112	70	.25	1982-08-15	1982-10-15
	000250	AD3112	80	.25	1982-08-15	1982-10-15
	000250	AD3112	80	.50	1982-10-15	1982-12-01
	000250	AD3112	180	.50	1982-08-15	1983-01-01
	000260	AD3113	70	.50	1982-06-15	1982-07-01
	000260	AD3113	70	1.00	1982-07-01	1983-02-01
	000260	AD3113	80	1.00	1982-01-01	1982-03-01
	000260	AD3113	80	.50	1982-03-01	1982-04-15
	000260	AD3113	180	.50	1982-03-01	1982-04-15
	000260	AD3113	180	1.00	1982-04-15	1982-06-01
	000260	AD3113	180	.50	1982-06-01	1982-07-01
	000270	AD3113	60	.50	1982-03-01	1982-04-01
	000270	AD3113	60	1.00	1982-04-01	1982-09-01
	000270	AD3113	60	.25	1982-09-01	1982-10-15
	000270	AD3113	70	.75	1982-09-01	1982-10-15
	000270	AD3113	70	1.00	1982-10-15	1983-02-01

名前:	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
	000270	AD3113	80	1.00	1982-01-01	1982-03-01
	000270	AD3113	80	.50	1982-03-01	1982-04-01
	000030	IF1000	10	.50	1982-06-01	1983-01-01
	000130	IF1000	90	1.00	1982-01-01	1982-10-01
	000130	IF1000	100	.50	1982-10-01	1983-01-01
	000140	IF1000	90	.50	1982-10-01	1983-01-01
	000030	IF2000	10	.50	1982-01-01	1983-01-01
	000140	IF2000	100	1.00	1982-01-01	1982-03-01
	000140	IF2000	100	.50	1982-03-01	1982-07-01
	000140	IF2000	110	.50	1982-03-01	1982-07-01
	000140	IF2000	110	.50	1982-10-01	1983-01-01
	000010	MA2100	10	.50	1982-01-01	1982-11-01
	000110	MA2100	20	1.00	1982-01-01	1982-03-01
	000010	MA2110	10	1.00	1982-01-01	1983-02-01
	000200	MA2111	50	1.00	1982-01-01	1982-06-15
	000200	MA2111	60	1.00	1982-06-15	1983-02-01
	000220	MA2111	40	1.00	1982-01-01	1983-02-01
	000150	MA2112	60	1.00	1982-01-01	1982-07-15
	000150	MA2112	180	1.00	1982-07-15	1983-02-01
	000170	MA2112	60	1.00	1982-01-01	1983-06-01
	000170	MA2112	70	1.00	1982-06-01	1983-02-01
	000190	MA2112	70	1.00	1982-02-01	1982-10-01
	000190	MA2112	80	1.00	1982-10-01	1983-10-01
	000160	MA2113	60	1.00	1982-07-15	1983-02-01
	000170	MA2113	80	1.00	1982-01-01	1983-02-01
	000180	MA2113	70	1.00	1982-04-01	1982-06-15
	000210	MA2113	80	.50	1982-10-01	1983-02-01
	000210	MA2113	180	.50	1982-10-01	1983-02-01
	000050	OP1000	10	.25	1982-01-01	1983-02-01
	000090	OP1010	10	1.00	1982-01-01	1983-02-01
	000280	OP1010	130	1.00	1982-01-01	1983-02-01
	000290	OP1010	130	1.00	1982-01-01	1983-02-01
	000300	OP1010	130	1.00	1982-01-01	1983-02-01
	000310	OP1010	130	1.00	1982-01-01	1983-02-01
	000050	OP2010	10	.75	1982-01-01	1983-02-01
	000100	OP2010	10	1.00	1982-01-01	1983-02-01
	000320	OP2011	140	.75	1982-01-01	1983-02-01

名前:	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
	000320	OP2011	150	.25	1982-01-01	1983-02-01
	000330	OP2012	140	.25	1982-01-01	1983-02-01
	000330	OP2012	160	.75	1982-01-01	1983-02-01
	000340	OP2013	140	.50	1982-01-01	1983-02-01
	000340	OP2013	170	.50	1982-01-01	1983-02-01
	000020	PL2100	30	1.00	1982-01-01	1982-09-15

EMP_PHOTO 表

名前:	EMPNO	PHOTO_FORMAT	PICTURE
タイプ:	char(6) 非ヌル値	varchar(10) 非ヌル値	blob(100k)
説明:	従業員番号	写真の形式	従業員の写真
値:	000130	bitmap	db200130.bmp
	000130	gif	db200130.gif
	000130	xwd	db200130.xwd
	000140	bitmap	db200140.bmp
	000140	gif	db200140.gif
	000140	xwd	db200140.xwd
	000150	bitmap	db200150.bmp
	000150	gif	db200150.gif
	000150	xwd	db200150.xwd
	000190	bitmap	db200190.bmp
	000190	gif	db200190.gif
	000190	xwd	db200190.xwd

- 従業員 Delores Quintana の写真は、1381ページの『Quintana の写真』に示されています。
- 従業員 Heather Nicholls の写真は、1383ページの『Nicholls の写真』に示されています。
- 従業員 Bruce Adamson の写真は、1385ページの『Adamson の写真』に示されています。
- 従業員 James Walker の写真は、1386ページの『Walker の写真』に示されています。

EMP_RESUME 表

名前:	EMPNO	RESUME_FORMAT	RESUME
タイプ:	char(6) 非ヌル値	varchar(10) 非ヌル値	clob(5k)
説明:	従業員番号	履歴書の形式	従業員の履歴書
値:	000130	ascii	db200130.asc
	000130	script	db200130.scr
	000140	ascii	db200140.asc
	000140	script	db200140.scr
	000150	ascii	db200150.asc
	000150	script	db200150.scr
	000190	ascii	db200190.asc
	000190	script	db200190.scr

- 従業員 Delores Quintana の履歴書は、1382ページの『Quintana の履歴書』に示されています。
- 従業員 Heather Nicholls の履歴書は、1383ページの『Nicholls の履歴書』に示されています。
- 従業員 Bruce Adamson の履歴書は、1385ページの『Adamson の履歴書』に示されています。
- 従業員 James Walker の履歴書は、1386ページの『Walker の履歴書』に示されています。

IN_TRAY 表

名前:	RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
タイプ:	timestamp	char(8)	char(64)	varchar(3000)
説明:	受取日時	メモの送信先のユーザー ID	要旨	メモ

ORG 表

名前:	DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
タイプ:	smallint 非ヌル値	varchar(14)	smallint	varchar(10)	varchar(13)
説明:	部門番号	部門名	管理者番号	企業の区分	市
値:	10	Head Office	160	Corporate	New York
	15	New England	50	Eastern	Boston
	20	Mid Atlantic	10	Eastern	Washington
	38	South Atlantic	30	Eastern	Atlanta

名前:	DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
	42	Great Lakes	100	Midwest	Chicago
	51	Plains	140	Midwest	Dallas
	66	Pacific	270	Western	San Francisco
	84	Mountain	290	Western	Denver

PROJECT 表

名前:	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
タイプ:	char(6) 非ヌル値	varchar(24) 非ヌル値	char(3) 非ヌル値	char(6) 非ヌル値	dec(5,2)	date	date	char(6)
説明:	プロジェクト番号	プロジェクト名	担当部門	責任者	スタッフ平均人数の見積もり	開始日付の見積もり	終了日付の見積もり	主プロジェクト(副プロジェクトの場合)
値:	AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	-
	AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
	AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
	AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
	AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
	IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	-
	IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	-
	MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	-
	MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
	MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
	MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
	MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
	OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	-
	OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
	OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	-
	OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
	OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010

名前:	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
	OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
	OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
	PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

SALES 表

名前:	SALES_DATE	SALES_PERSON	REGION	SALES
タイプ:	date	varchar(15)	varchar(15)	int
説明:	売上日	従業員の姓	売上地域	売上数量
値:	12/31/1995	LUCCHESSI	Ontario-South	1
	12/31/1995	LEE	Ontario-South	3
	12/31/1995	LEE	Quebec	1
	12/31/1995	LEE	Manitoba	2
	12/31/1995	GOUNOT	Quebec	1
	03/29/1996	LUCCHESSI	Ontario-South	3
	03/29/1996	LUCCHESSI	Quebec	1
	03/29/1996	LEE	Ontario-South	2
	03/29/1996	LEE	Ontario-North	2
	03/29/1996	LEE	Quebec	3
	03/29/1996	LEE	Manitoba	5
	03/29/1996	GOUNOT	Ontario-South	3
	03/29/1996	GOUNOT	Quebec	1
	03/29/1996	GOUNOT	Manitoba	7
	03/30/1996	LUCCHESSI	Ontario-South	1
	03/30/1996	LUCCHESSI	Quebec	2
	03/30/1996	LUCCHESSI	Manitoba	1
	03/30/1996	LEE	Ontario-South	7
	03/30/1996	LEE	Ontario-North	3
	03/30/1996	LEE	Quebec	7
	03/30/1996	LEE	Manitoba	4
	03/30/1996	GOUNOT	Ontario-South	2
	03/30/1996	GOUNOT	Quebec	18
	03/30/1996	GOUNOT	Manitoba	1
	03/31/1996	LUCCHESSI	Manitoba	1
	03/31/1996	LEE	Ontario-South	14
	03/31/1996	LEE	Ontario-North	3
	03/31/1996	LEE	Quebec	7
	03/31/1996	LEE	Manitoba	3
	03/31/1996	GOUNOT	Ontario-South	2
	03/31/1996	GOUNOT	Quebec	1

名前:	SALES_DATE	SALES_PERSON	REGION	SALES
	04/01/1996	LUCCHESSI	Ontario-South	3
	04/01/1996	LUCCHESSI	Manitoba	1
	04/01/1996	LEE	Ontario-South	8
	04/01/1996	LEE	Ontario-North	-
	04/01/1996	LEE	Quebec	8
	04/01/1996	LEE	Manitoba	9
	04/01/1996	GOUNOT	Ontario-South	3
	04/01/1996	GOUNOT	Ontario-North	1
	04/01/1996	GOUNOT	Quebec	3
	04/01/1996	GOUNOT	Manitoba	7

STAFF 表

名前:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
タイプ:	smallint 非ヌル値	varchar(9)	smallint	char(5)	smallint	dec(7,2)	dec(7,2)
説明:	従業員番号	従業員名	部門番号	職種	勤続年数	現在の給与	歩合給
値:	10	Sanders	20	Mgr	7	18357.50	-
	20	Pernal	20	Sales	8	18171.25	612.45
	30	Marenghi	38	Mgr	5	17506.75	-
	40	O’Brien	38	Sales	6	18006.00	846.55
	50	Hanes	15	Mgr	10	20659.80	-
	60	Quigley	38	Sales	-	16808.30	650.25
	70	Rothman	15	Sales	7	16502.83	1152.00
	80	James	20	Clerk	-	13504.60	128.20
	90	Koonitz	42	Sales	6	18001.75	1386.70
	100	Plotz	42	Mgr	7	18352.80	-
	110	Ngan	15	Clerk	5	12508.20	206.60
	120	Naughton	38	Clerk	-	12954.75	180.00
	130	Yamaguchi	42	Clerk	6	10505.90	75.60
	140	Fraye	51	Mgr	6	21150.00	-
	150	Williams	51	Sales	6	19456.50	637.65
	160	Molinare	10	Mgr	7	22959.20	-
	170	Kermisch	15	Clerk	4	12258.50	110.10
	180	Abrahams	38	Clerk	3	12009.75	236.50
	190	Sneider	20	Clerk	8	14252.75	126.50
	200	Scoutten	42	Clerk	-	11508.60	84.20
	210	Lu	10	Mgr	10	20010.00	-

名前:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
	220	Smith	51	Sales	7	17654.50	992.80
	230	Lundquist	51	Clerk	3	13369.80	189.65
	240	Daniels	10	Mgr	5	19260.25	-
	250	Wheeler	51	Clerk	6	14460.00	513.30
	260	Jones	10	Mgr	12	21234.00	-
	270	Lea	66	Mgr	9	18555.50	-
	280	Wilson	66	Sales	9	18674.50	811.50
	290	Quill	84	Mgr	10	19818.00	-
	300	Davis	84	Sales	5	15454.50	806.10
	310	Graham	66	Sales	13	21000.00	200.30
	320	Gonzales	66	Sales	4	16858.20	844.00
	330	Burke	66	Clerk	1	10988.00	55.50
	340	Edwards	84	Sales	7	17844.00	1285.00
	350	Gafney	84	Clerk	5	13030.50	188.00

STAFFG 表

注: STAFFG は 2 バイト・コード・ページ用としてのみ作成されます。

名前:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
タイプ:	smallint 非ヌル値	vargraphic(9)	smallint	graphic(5)	smallint	dec(9,0)	dec(9,0)
説明:	従業員番号	従業員名	部門番号	職種	勤続年数	現在の給与	歩合給
値:	10	Sanders	20	Mgr	7	18357.50	-
	20	Pernal	20	Sales	8	18171.25	612.45
	30	Marenghi	38	Mgr	5	17506.75	-
	40	O'Brien	38	Sales	6	18006.00	846.55
	50	Hanes	15	Mgr	10	20659.80	-
	60	Quigley	38	Sales	-	16808.30	650.25
	70	Rothman	15	Sales	7	16502.83	1152.00
	80	James	20	Clerk	-	13504.60	128.20
	90	Koonitz	42	Sales	6	18001.75	1386.70
	100	Plotz	42	Mgr	7	18352.80	-
	110	Ngan	15	Clerk	5	12508.20	206.60
	120	Naughton	38	Clerk	-	12954.75	180.00
	130	Yamaguchi	42	Clerk	6	10505.90	75.60
	140	Fraye	51	Mgr	6	21150.00	-
	150	Williams	51	Sales	6	19456.50	637.65

名前:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
	160	Molinare	10	Mgr	7	22959.20	-
	170	Kermisch	15	Clerk	4	12258.50	110.10
	180	Abrahams	38	Clerk	3	12009.75	236.50
	190	Sneider	20	Clerk	8	14252.75	126.50
	200	Scoutten	42	Clerk	-	11508.60	84.20
	210	Lu	10	Mgr	10	20010.00	-
	220	Smith	51	Sales	7	17654.50	992.80
	230	Lundquist	51	Clerk	3	13369.80	189.65
	240	Daniels	10	Mgr	5	19260.25	-
	250	Wheeler	51	Clerk	6	14460.00	513.30
	260	Jones	10	Mgr	12	21234.00	-
	270	Lea	66	Mgr	9	18555.50	-
	280	Wilson	66	Sales	9	18674.50	811.50
	290	Quill	84	Mgr	10	19818.00	-
	300	Davis	84	Sales	5	15454.50	806.10
	310	Graham	66	Sales	13	21000.00	200.30
	320	Gonzales	66	Sales	4	16858.20	844.00
	330	Burke	66	Clerk	1	10988.00	55.50
	340	Edwards	84	Sales	7	17844.00	1285.00
	350	Gafney	84	Clerk	5	13030.50	188.00

BLOB および CLOB データ・タイプを含むサンプル・ファイル

ここに示すのは、 EMP_PHOTO ファイル (従業員の写真) と EMP_RESUME ファイル (従業員の履歴書) に含まれているデータです。

Quintana の写真

図 13. Delores M. Quintana

Quintana の履歴書

以下のテキストは、 db200130.asc ファイルと db200130.scr ファイルに入っています。

Resume: Delores M. Quintana

Personal Information	
Address:	1150 Eglinton Ave Mellonville, Idaho 83725
Phone:	(208) 555-9933
Birthdate:	September 15, 1925
Sex:	Female
Marital Status:	Married
Height:	5’2″
Weight:	120 lbs.
Department Information	
Employee Number:	000130
Dept Number:	C01
Manager:	Sally Kwan
Position:	Analyst
Phone:	(208) 555-4578
Hire Date:	1971-07-28
Education	
1965	Math and English, B.A. Adelphi University

1960	Dental Technician Florida Institute of Technology

Work History

10/91 - present	Advisory Systems Analyst Producing documentation tools for engineering department.
12/85 - 9/91	Technical Writer Writer, text programmer, and planner.
1/79 - 11/85	COBOL Payroll Programmer Writing payroll programs for a diesel fuel company.

Interests

- Cooking
- Reading
- Sewing
- Remodeling

Nicholls の写真

図 14. Heather A. Nicholls

Nicholls の履歴書

以下のテキストは、 db200140.asc ファイルと db200140.scr ファイルに入っています。

Resume: Heather A. Nicholls

Personal Information

Address:	844 Don Mills Ave Mellonville, Idaho 83734
Phone:	(208) 555-2310
Birthdate:	January 19, 1946
Sex:	Female
Marital Status:	Single
Height:	5’8″
Weight:	130 lbs.

Department Information

Employee Number:	000140
Dept Number:	C01
Manager:	Sally Kwan
Position:	Analyst
Phone:	(208) 555-1793
Hire Date:	1976-12-15

Education

1972	Computer Engineering, Ph.D. University of Washington
1969	Music and Physics, M.A. Vassar College

Work History

2/83 - present	Architect, OCR Development Designing the architecture of OCR products.
12/76 - 1/83	Text Programmer Optical character recognition (OCR) programming in PL/I.
9/72 - 11/76	Punch Card Quality Analyst Checking punch cards met quality specifications.

Interests

- Model railroading
- Interior decorating
- Embroidery
- Knitting

Adamson の写真

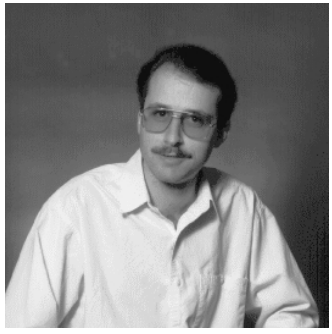


図 15. Bruce Adamson

Adamson の履歴書

以下のテキストは、 db200150.asc ファイルと db200150.scr ファイルに入っています。

Resume: Bruce Adamson

Personal Information	
Address:	3600 Steeles Ave Mellonville, Idaho 83757
Phone:	(208) 555-4489
Birthdate:	May 17, 1947
Sex:	Male
Marital Status:	Married
Height:	6’0″
Weight:	175 lbs.
Department Information	
Employee Number:	000150
Dept Number:	D11
Manager:	Irving Stern
Position:	Designer
Phone:	(208) 555-4510
Hire Date:	1972-02-12

Education

1971	Environmental Engineering, M.Sc. Johns Hopkins University
1968	American History, B.A. Northwestern University

Work History

8/79 - present	Neural Network Design Developing neural networks for machine intelligence products.
2/72 - 7/79	Robot Vision Development Developing rule-based systems to emulate sight.
9/71 - 1/72	Numerical Integration Specialist Helping bank systems communicate with each other.

Interests

- Racing motorcycles
- Building loudspeakers
- Assembling personal computers
- Sketching

Walker の写真

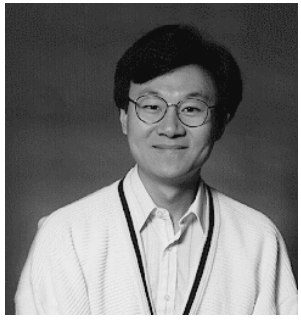


図 16. James H. Walker

Walker の履歴書

以下のテキストは、 db200190.asc ファイルと db200190.scr ファイルに入っています。

Resume: James H. Walker

Personal Information

Address:	3500 Steeles Ave Mellonville, Idaho 83757
Phone:	(208) 555-7325
Birthdate:	June 25, 1952
Sex:	Male
Marital Status:	Single
Height:	5’11″
Weight:	166 lbs.

Department Information

Employee Number:	000190
Dept Number:	D11
Manager:	Irving Stern
Position:	Designer
Phone:	(208) 555-2986
Hire Date:	1974-07-26

Education

1974	Computer Studies, B.Sc. University of Massachusetts
1972	Linguistic Anthropology, B.A. University of Toronto

Work History

6/87 - present	Microcode Design Optimizing algorithms for mathematical functions.
4/77 - 5/87	Printer Technical Support Installing and supporting laser printers.
9/74 - 3/77	Maintenance Programming Patching assembly language compiler for mainframes.

Interests

- Wine tasting
- Skiing
- Swimming

- Dancing

付録H. 予約スキーマ名と予約語

この付録では、データベース・マネージャーにより使用される特定の名前に関する制約事項について説明します。名前によっては、予約済みで、アプリケーション・プログラムで使用できない名前があります。また、データベース・マネージャーによって、その使用は禁止されてはいませんが、アプリケーション・プログラムによる使用をお勧めできない名前もあります。

予約スキーマ

以下のスキーマ名は、予約されています。

- SYSCAT
- SYSFUN
- SYSIBM
- SYSSTAT

さらに、SYS は規則によりシステムで予約されている領域を示すのに使用されるので、 SYS の接頭部で始まるスキーマ名は使用しないようにしてください。

ユーザー定義関数、ユーザー定義タイプ、トリガー、または別名を、 SYS で始まる名前のスキーマに入れることはできません (SQLSTATE 42939)。

また、SESSION はスキーマ名として使用しないようにお勧めします。宣言される一時表は SESSION で修飾しなければならないため、アプリケーションが持続表の名前と同じ名前を指定して一時表を宣言してしまい、アプリケーションのロジックが複雑になってしまう場合があります。このような事態を避けるため、宣言される一時表を扱う場合を除いて、スキーマ SESSION を使用することは避けてください。

予約語

DB2 バージョン 7 では、特に予約されている予約語はありません。

キーワードも、それが SQL キーワードとして解釈されることになる文脈以外であれば、通常識別子として使用することができます。キーワードとして解釈される文脈の場合は、その語を区切り識別子として指定する必要があります。たとえば、SELECT ステートメントに、区切り識別子でない COUNT を列名として使用することはできません。

IBM SQL と ISO/ANSI SQL92 には、次のリストに示す予約語が含まれています。 DB2 ユニバーサル・データベースはこれらの予約語を予約していませんが、移行性が低下するので、通常識別子としての使用を避けることをお勧めします。

IBM SQL の予約語

IBM SQL の予約語は、次のとおりです。

ACQUIRE
ADD
AFTER
ALIAS
ALL
ALLOCATE
ALLOW
ALTER
AND
ANY
AS
ASC
ASUTIME
AUDIT
AUTHORIZATION
AUX
AUXILIARY
AVG

BEFORE
BEGIN
BETWEEN
BINARY
BUFFERPOOL
BY

CALL
CALLED
CAPTURE
CASCADED
CASE
CAST
CCSID
CHAR
CHARACTER
CHECK
CLOSE
CLUSTER
COLLECTION
COLLID
COLUMN
COMMENT
COMMIT
CONCAT
CONDITION
CONNECT
CONNECTION
CONSTRAINT
CONTAINS
CONTINUE
COUNT
COUNT_BIG
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_LC_PATH
CURRENT_PATH
CURRENT_SERVER
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TIMEZONE
CURRENT_USER
CURSOR

DATA
DATABASE
DATE
DAY
DAYS
DBA
DBINFO
DBSPACE
DB2GENERAL
DB2SQL
DECLARE
DEFAULT
DELETE
DESC
DESCRIPTOR
DETERMINISTIC
DISALLOW
DISCONNECT
DISTINCT
DO
DOUBLE
DROP
DSSIZE
DYNAMIC
EDITPROC
ELSE
ELSEIF
END
END-EXEC
ERASE
ESCAPE
EXCEPT
EXCEPTION
EXCLUSIVE
EXECUTE
EXISTS
EXIT
EXPLAIN
EXTERNAL

FENCED
FETCH
FIELDPROC
FILE
FINAL
FOR
FOREIGN
FREE
FROM
FULL
FUNCTION

GENERAL
GENERATED
GO
GOTO
GRANT
GRAPHIC
GROUP

HANDLER
HAVING
HOUR
HOURS

IDENTIFIED
IF
IMMEDIATE
IN
INDEX
INDICATOR
INNER
INOUT
INSENSITIVE
INSERT
INTEGRITY
INTERSECT
INTO
IS
ISOBID
ISOLATION

JAVA
JOIN

KEY

LABEL
LANGUAGE
LC_CTYPE
LEAVE
LEFT
LIKE
LINKTYPE
LOCAL
LOCALE
LOCATOR
LOCATORS
LOCK
LOCKSIZE
LONG
LOOP

MAX
MICROSECOND
MICROSECONDS
MIN
MINUTE
MINUTES
MODE
MODIFIES
MONTH
MONTHS

予約スキーマ名と予約語

NAME
NAMED
NHEADER
NO
NODENAME
NODENUMBER
NOT
NULL
NULLS
NUMPARTS

OBID
OF
ON
ONLY
OPEN
OPTIMIZATION
OPTIMIZE
OPTION
OR
ORDER
OUT
OUTER

PACKAGE
PAGE
PAGES
PARAMETER
PART
PARTITION
PATH
PCTFREE
PCTINDEX
PIECESIZE
PLAN
POSITION
PRECISION
PREPARE
PRIMARY
PRIQTY
PRIVATE
PRIVILEGES
PROCEDURE
PROGRAM
PSID
PUBLIC

QUERYNO

READ
READS
RECOVERY
REFERENCES
RELEASE
RENAME
REPEAT
RESET
RESOURCE
RESTRICT
RESULT
RETURN
RETURNS
REVOKE
RIGHT
ROLLBACK
ROW
ROWS
RRN
RUN

SCHEDULE
SCHEMA
SCRATCHPAD
SECOND
SECONDS
SECQTY
SECURITY
SELECT
SET
SHARE
SIMPLE
SOME
SOURCE
SPECIFIC
SQL
STANDARD
STATIC
STATISTICS
STAY
STOGROUP
STORES
STORPOOL
STYLE
SUBPAGES
SUBSTRING
SUM
SYNONYM

TABLE
TABLESPACE
THEN
TO
TRANSACTION
TRIGGER
TRIM
TYPE

UNDO
UNION
UNIQUE
UNTIL
UPDATE
USAGE
USER
USING

VALIDPROC
VALUES
VARIABLE
VARIANT
VCAT
VIEW
VOLUMES

WHEN
WHERE
WHILE
WITH
WLM
WORK
WRITE

YEAR
YEARS

ISO/ANS SQL92 の予約語

ISO/ANS SQL92 の予約語で、IBM SQL のリストにない予約語は、次のとおりです。

ABSOLUTE
ACTION
ARE
ASSERTION
AT

BIT_LENGTH
BOTH

CATALOG
CHAR_LENGTH
CHARACTER_LENGTH
COALESCE
COLLATE
COLLATION
CONSTRAINTS
CONVERT
CORRESPONDING

DEALLOCATE
DEC
DECIMAL
DEFERRABLE
DEFERRED
DESCRIBE
DIAGNOSTICS
DOMAIN

EXEC
EXTRACT

FALSE
FIRST
FLOAT
FOUND
FULL

GET
GLOBAL

IDENTITY
INITIALLY
INPUT
INTERVAL

LAST
LEADING
LEVEL
LOWER

MATCH
MODULE

NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NULLIF
NUMERIC

OCTET_LENGTH
OUTPUT
OVERLAPS

PAD
PARTIAL
PRESERVE
PRIOR

REAL
RELATIVE

SCROLL
SECTION
SESSION
SESSION_USER
SIZE
SMALLINT
SPACE
SQLCODE
SQLERROR
SQLSTATE
SYSTEM_USER

TEMPORARY
TIMEZONE_HOUR
TIMEZONE_MINUTE
TRAILING
TRANSLATION
TRUE

UNKNOWN
UPPER

VALUE
VARCHAR
VARYING

WHENEVER

ZONE

付録I. 分離レベルの比較

次の表は、29ページの『分離レベル』で説明している分離レベルについて要約しています。

	UR	CS	RS	RR
アプリケーションは、他のアプリケーション・プロセスによって行われた未コミットの変更を見ることができるか ?	Yes	No	No	No
アプリケーションは、他のアプリケーション・プロセスによって行われた未コミットの変更を更新できるか ?	No	No	No	No
ステートメントをもう一度実行した場合、他のアプリケーション・プロセスによる影響があるか ? *下記の現象 P3 (幻像) を参照。*	Yes	Yes	Yes	No
「更新された」行を他のアプリケーション・プロセスで更新することができるか ?	No	No	No	No
「更新された」行を、UR 以外の分離レベルの他のアプリケーション・プロセスで読み取れるか ?	No	No	No	No
「更新された」行を UR の分離レベルの他のアプリケーション・プロセスで読み取れるか ?	Yes	Yes	Yes	Yes
「アクセスされた」行を、他のアプリケーション・プロセスによって更新することが可能か ? *下記の現象 P2 (反復不能読み取り) を参照。*	Yes	Yes	No	No
「アクセスされた」行を他のアプリケーション・プロセスが読み取ることができるか ?	Yes	Yes	Yes	Yes
「現在」行を、他のアプリケーション・プロセスによって更新または削除することが可能か ? *下記の現象 P1 (ダーティー読み取り) を参照。*	下記の注を参照	下記の注を参照	No	No

注:

1. カーソルが更新可能でない場合、CS では、現在行を他のアプリケーション・プロセスによって更新または削除できる場合もあります。

	UR	CS	RS	RR

現象の例:

P1 *ダーティー読み取り*。作業単位 UW1 が行を変更するとします。 UW1 が COMMIT を実行する前に、作業単位 UW2 がその行を読み取るとします。次に UW1 が ROLLBACK を実行したとすると、 UW2 は実行しない行を読み取ったことになります。

P2 *反復不能読み取り*。作業単位 UW1 が行を読み取るとします。作業単位 UW2 がその行を変更してから、COMMIT を実行するとします。 UW1 がもう一度その行を読み取ると、値が修正されていることがあります。

P3 *幻像*。作業単位 UW1 が、ある探索条件を満たしている *n* 個の行を読み取るとします。次に作業単位 UW2 が、その探索条件を満たしている 1 つまたは複数の行を挿入するとします。 UW1 が同じ探索条件でもう一度最初の読み取りを実行すると、元の行のほかに挿入された行が追加されていることになります。

付録J. トリガーと制約の相互作用

この付録では、更新操作の結果として生じる可能性のある参照制約および検査制約とトリガーとの相互作用について説明します。 図17 とその後の説明は、データベースのデータを更新する SQL ステートメントに対して行われる典型的な処理を示しています。

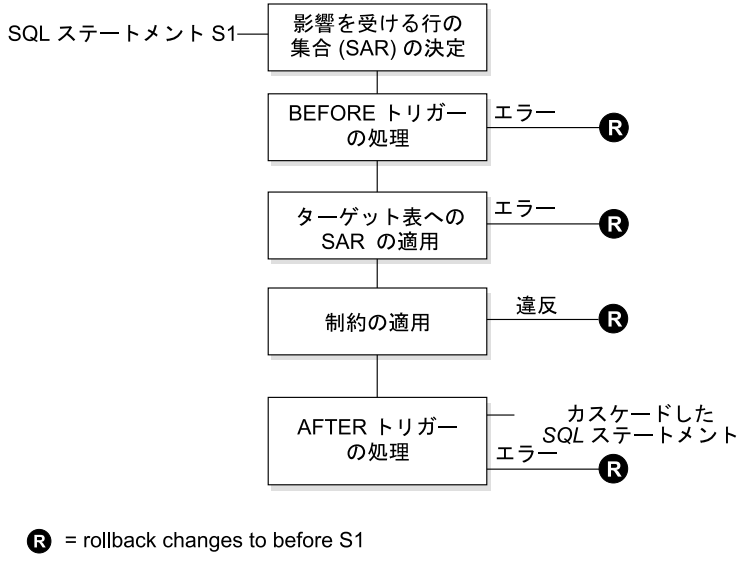


図 17. 関連するトリガーと制約を伴う SQL ステートメントの処理

図17 は、表を更新する SQL ステートメントの一般的な処理の順序を示しています。ここでは、表に、「BEFORE」のトリガー、参照制約、検査制約、およびカスケードする「AFTER」トリガーとが含まれることを想定しています。図17 に示されているボックスやその他の項目について、以下に説明します。

- SQL ステートメント S_1

  これは、その処理を開始する DELETE、INSERT、または UPDATE ステートメントです。 SQL ステートメント S_1 は、この説明においてターゲット表 と呼ぶ表 (または表に対する更新可能な視点) を指定しています。
- 影響を受ける行の集合 *(SAR)* の決定

このステップは、CASCADE および SET NULL の参照制約の削除規則と、AFTER トリガーからカスケードした SQL ステートメントについて繰り返される処理の開始点です。

このステップの目的は、その SQL ステートメントで影響を受ける行の集合を決定することです。 *SAR* に含まれる行の集合には、以下の基準で行が含められます。

– DELETE の場合、ステートメントの探索条件を満たしているすべての行 (定位置 DELETE の場合は現在行)
– INSERT の場合、VALUES 文節または全選択によって指定される行
– UPDATE の場合、探索条件を満たしているすべての行 (定位置 UPDATE の場合は現在行)

SAR が空の場合、 BEFORE トリガー、ターゲット表に適用される変更、または SQL ステートメントの処理に対する制約はありません。

- BEFORE トリガーの処理

  BEFORE トリガーの処理はすべて作成の昇順で行われます。各 BEFORE トリガーは、 *SAR* 内の各行ごとに 1 回ずつトリガー・アクションを処理します。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合には元の SQL ステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

  BEFORE トリガーがない場合、または *SAR* が空の場合、このステップはスキップされます。

- ターゲット表への *SAR* の適用

  実際の削除、挿入、または更新は、 *SAR* を使ってデータベース内のターゲット表に適用されます。

  SAR の適用時にエラーが生じることがあり (固有索引のある場所に重複するキーを持つ行を挿入しようとした場合など)、そのような場合は元の SQL ステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

- 制約の適用

  SAR が空の場合には、ターゲット表に関連した制約が適用されます。この制約には、固有制約、固有索引、参照制約、検査制約、視点に対する WITH CHECK OPTION に関連した検査などがあります。カスケードまたは NULL 設定の削除規則の参照制約があると、さらに別のトリガーが起動される場合があります。

索引または WITH CHECK OPTION に違反があると、エラーが発生し、 S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

- AFTER トリガーの処理

  S_1 によって活動化された AFTER トリガーは、作成の昇順に処理されます。

  FOR EACH STATEMENT トリガーでは、*SAR* が空の場合にも、 1 回だけトリガー・アクションが処理されます。 FOR EACH ROW トリガーでは、*SAR* 内の各行ごとに 1 回ずつトリガー・アクションが処理されます。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合は元の S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

  トリガーのトリガー・アクションには、トリガーによって実行される DELETE、INSERT、または UPDATE などの SQL ステートメントが含まれている場合があります。この説明では、そのようなステートメントは、*カスケードした SQL ステートメント* とみなされます。

  カスケードした SQL ステートメントとは、 AFTER トリガーのトリガー・アクションの一部として処理される DELETE、 INSERT、または UPDATE ステートメントのことです。そのステートメントによって、カスケード・レベルのトリガー処理が開始されます。これは、新しい S_1 としてトリガー SQL ステートメントを割り当てて、ここで説明した手順をすべて再帰的に実行することと見なすことができます。

  各 S_1 ごとに起動されるすべての AFTER トリガーによって実行されるすべての SQL ステートメントの処理が完了すると、元の S_1 の処理が完了します。

- **R** = 変更を S_1 の前までロール・バックする操作

  制約違反も含めて、処理中にエラーが発生すると、元の SQL ステートメント S_1 の結果として直接または間接になされたすべての変更がロール・バックされます。その場合、データベースは、元の SQL ステートメント S_1 の実行直前と同じ状態に戻ります。

付録K. Explain 表と定義

Explain 表は、Explain 機能が活動化された時点のアクセス・プランをキャプチャーします。この付録では、以下の Explain 表と定義を示します。

Explain 表は、Explain 機能を呼び出す前に作成しておく必要があります。'sqllib' ディレクトリーの 'misc' サブディレクトリーの中に入っているファイル EXPLAIN.DDL の中に用意されているコマンド行プロセッサー用入力スクリプトの例を使用してください。 Explain 表が要求されているデータベースに接続します。それからコマンド `db2 -tf EXPLAIN.DDL` を発行すると表が作成されます。詳細については、1425ページの『Explain 表の定義』を参照してください。

Explain 機能によって Explain 表にデータを入れても、トリガーが起動したり、参照制約または検査制約が起動したりはされません。たとえば、EXPLAIN_INSTANCE 表に対する挿入トリガーを定義して、該当するステートメントが Explain されても、そのトリガーは起動しません。

Explain 機能の詳細については、 *管理の手引き* を参照してください。

Explain 表の凡例:

見出し	説明
列名	列の名前
データ・タイプ	列のデータ・タイプ

ヌル値	Yes: ヌル値は許される
可能 ?	No: ヌル値は許されない
キー ?	PK: この列は基本キーの一部である
	FK: この列は外部キーの一部である
説明	列についての説明

EXPLAIN_ARGUMENT 表

EXPLAIN_ARGUMENT 表は、個々の演算子に固有の特性がある場合、それを示します。

この表の定義については、1426ページの『EXPLAIN_ARGUMENT 表の定義』を参照してください。

表 128. EXPLAIN_ARGUMENT 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	No	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	No	No	この照会内の演算子の固有の ID。
ARGUMENT_TYPE	CHAR(8)	No	No	この演算子の引き数のタイプ。
ARGUMENT_VALUE	VARCHAR(1024)	Yes	No	この演算子の引き数の値。値が LONG_ARGUMENT_VALUE にある場合は NULL。
LONG_ARGUMENT_VALUE	CLOB(1M)	Yes	No	この演算子の引き数の値 (テキストが ARGUMENT_VALUE に適合しない場合)。値が ARGUMENT_VALUE にある場合は NULL。

表 129. ARGUMENT_TYPE および ARGUMENT_VALUE 列の値

ARGUMENT_TYPE の値	可能な ARGUMENT_VALUE 値	説明
AGGMODE	COMPLETE PARTIAL INTERMEDIATE FINAL	部分集約標識。
BITFLTR	TRUE FALSE	ハッシュ結合でビット・フィルターを使ってパフォーマンスを向上させる。
CSETEMP	TRUE FALSE	共通副次式フラグに対する一時表。
DIRECT	TRUE	取り出し標識を指示する。
DUPLWARN	TRUE FALSE	警告標識を複写する。
EARLYOUT	TRUE FALSE	Early out を指示する。
ENVVAR	このタイプの各行には以下のものが含まれます。 • 環境変数名 • 環境変数値	最適化プログラムに影響する環境変数
FETCHMAX	IGNORE INTEGER	FETCH 演算子の MAXPAGES 引き数の値をオーバーライドする。
GROUPBYC	TRUE FALSE	Group By 列が与えられているかどうか。
GROUPBYN	Integer	比較列の数。
GROUPBYR	このタイプの各行には以下のものが含まれます。 • group by 文節内の列の順序値 (後に、コロンとスペースが続く) • 列の名前	Group By 要件。
INNERCOL	このタイプの各行には以下のものが含まれます。 • 配列内の列の順序値 (後に、コロンとスペースが続く) • 列の名前 • 配列の値 **(A)** 昇順 **(D)** 降順	内部配列の列。
ISCANMAX	IGNORE INTEGER	ISCAN 演算子の MAXPAGES 引き数の値をオーバーライドする。

表 129. ARGUMENT_TYPE および ARGUMENT_VALUE 列の値 (続き)

ARGUMENT_TYPE の値	可能な ARGUMENT_VALUE 値	説明
JN_INPUT	INNER OUTER	演算子が内部または外部のどちらの結合を送る演算子であるかを示す。
LISTENER	TRUE FALSE	Listener 表キューの標識。
MAXPAGES	ALL NONE INTEGER	事前取り出しのための最大ページ数。
MAXRIDS	NONE INTEGER	個々のリスト事前取り出し要求に組み込まれる最大行識別子。
NUMROWS	INTEGER	分類されるべき行の数。
ONEFETCH	TRUE FALSE	1 つの取り出し標識。
OUTERCOL	このタイプの各行には以下のものが含まれます。 • 配列内の列の順序値 (後に、コロンとスペースが続く) • 列の名前 • 配列の値 **(A)** 昇順 **(D)** 降順	外部配列の列。
OUTERJN	LEFT RIGHT	外部結合の標識。
PARTCOLS	列の名前	演算子の区分化列。
PREFETCH	LIST NONE SEQUENTIAL	事前取り出し有資格属性のタイプ。
RMTQTEXT	照会テキスト	リモート照会テキスト
ROWLOCK	EXCLUSIVE NONE REUSE SHARE SHORT (INSTANT) SHARE UPDATE	行ロック意図。
ROWWIDTH	INTEGER	分類される行の幅。
SCANDIR	FORWARD REVERSE	走査の方向。
SCANGRAN	INTEGER	区画内並行性、区画内並行性の走査の細分性。 SCANUNIT の単位で表される。

表 129. ARGUMENT_TYPE および ARGUMENT_VALUE 列の値 (続き)

ARGUMENT_TYPE の値	可能な ARGUMENT_VALUE 値	説明
SCANTYPE	LOCAL PARALLEL	区画内並行性、索引または表の走査。
SCANUNIT	ROW PAGE	区画内並行性、走査の細分性の単位。
SERVER	リモート・サーバー	リモート・サーバー
SHARED	TRUE	区画内並行性、共用 TEMP 標識。
SLOWMAT	TRUE FALSE	低速実体化フラグ。
SNGLPROD	TRUE FALSE	区画内並行性、単一エージェントによる分類または一時作成。
SORTKEY	このタイプの各行には以下のものが含まれます。 • キー内の列の順序値 (後に、コロンとスペースが続く) • 列の名前 • 配列の値 **(A)** 昇順 **(D)** 降順	分類キーの列。
SORTTYPE	PARTITIONED SHARED ROUND ROBIN REPLICATED	区画内並行性、分類タイプ。
TABLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE REUSE SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	表ロック意図。
TQDEGREE	INTEGER	区画内並行性、表キューにアクセスするサブエージェントの数。
TQMERGE	TRUE FALSE	マージする (ソート済み) 表キューの標識。
TQREAD	READ AHEAD STEPPING SUBQUERY STEPPING	表キューの読み取り特性。

表 129. ARGUMENT_TYPE および ARGUMENT_VALUE 列の値 (続き)

ARGUMENT_TYPE の値	可能な ARGUMENT_VALUE 値	説明
TQSEND	BROADCAST DIRECTED SCATTER SUBQUERY DIRECTED	表キューの送信特性。
TQTYPE	LOCAL	区画内並行性、表キュー。
TRUNCSRT	TRUE	切り捨て分類 (作成される行の数を制限する)。
UNIQUE	TRUE FALSE	固有性の標識。
UNIQKEY	このタイプの各行には以下のものが含まれます。 • キー内の列の順序値 (後に、コロンとスペースが続く) • 列の名前	固有キーの列。
VOLATILE	TRUE	揮発性表

EXPLAIN_INSTANCE 表

EXPLAIN_INSTANCE 表は、すべての Explain 情報の主要な管理表です。
Explain 表の各データ行は、この表の固有の 1 行に明示的にリンクされています。
EXPLAIN_INSTANCE 表には、Explain の対象の SQL ステートメントのソースについての基本情報、および Explain 機能の環境についての情報が示されます。

この表の定義については、1427ページの『EXPLAIN_INSTANCE 表の定義』を参照してください。

表 130. EXPLAIN_INSTANCE 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	PK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	PK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	PK	Explain 要求のソースのスキーマ、または修飾子。

表 130. EXPLAIN_INSTANCE 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_OPTION	CHAR(1)	No	No	この要求でどんな Explain 情報が要求されたかを示します。 可能な値は次のとおりです。 **P** PLAN SELECTION
SNAPSHOT_TAKEN	CHAR(1)	No	No	この要求で Explain スナップショットが撮られたかどうかを示します。 可能な値は次のとおりです。 **Y** Explain スナップショット (1 枚以上) が撮られ、 EXPLAIN_STATEMENT 表に保管されました。通常の Explain 情報も取り込まれました。 **N** Explain スナップショットは撮られませんでした。通常の Explain 情報は取り込まれました。 **O** Explain スナップショットのみが撮られました。通常の Explain 情報は取り込まれませんでした。
DB2_VERSION	CHAR(7)	No	No	この Explain 要求を処理した DB2 ユニバーサル・データベースの製品リリース番号。これは、vv.rr.m の形式です。ただし、 **vv** バージョン番号 **rr** リリース番号 **m** メインテナンス・リリース番号
SQL_TYPE	CHAR(1)	No	No	Explain インスタンスが静的 SQL 用か、または動的 SQL 用かを示します。 可能な値は次のとおりです。 **S** 静的 SQL **D** 動的 SQL
QUERYOPT	INTEGER	No	No	Explain 呼び出し時に SQL コンパイラーで使用する照会最適化クラスを示します。この値は、Explain 対象の SQL ステートメントについて、 SQL コンパイラーがどのレベルの照会最適化を実行するかを示します。

表 130. EXPLAIN_INSTANCE 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
BLOCK	CHAR(1)	No	No	SQL ステートメントのコンパイル時に使用されたカーソルのブロック化タイプを示します。詳細については、SYSCAT.PACKAGES の BLOCK 列を参照してください。 可能な値は次のとおりです。 **N** ブロック化なし **U** 確定カーソルのブロック **B** すべてのカーソルのブロック
ISOLATION	CHAR(2)	No	No	SQL ステートメントのコンパイル時に使用された分離の種類を示します。詳細については、SYSCAT.PACKAGES の ISOLATION 列を参照してください。 可能な値は次のとおりです。 **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
BUFFPAGE	INTEGER	No	No	Explain 呼び出し時に BUFFPAGE データベース構成設定値が入れられます。
AVG_APPLS	INTEGER	No	No	Explain 呼び出し時に、AVG_APPLS データベース構成パラメーターの値が入れられます。
SORTHEAP	INTEGER	No	No	Explain 呼び出し時に SORTHEAP データベース構成設定値が入れられます。
LOCKLIST	INTEGER	No	No	Explain 呼び出し時に LOCKLIST データベース構成設定値が入れられます。
MAXLOCKS	SMALLINT	No	No	Explain 呼び出し時に MAXLOCKS データベース構成設定値が入れられます。
LOCKS_AVAIL	INTEGER	No	No	各ユーザーごとに最適化プログラムで使用できることになっているロック数が入れられます。(LOCKLIST および MAXLOCKS から導出される)
CPU_SPEED	DOUBLE	No	No	Explain 呼び出し時に、CPUSPEED データベース・マネージャー構成設定値が入れられます。
REMARKS	VARCHAR(254)	Yes	No	ユーザーが入力したコメント。
DBHEAP	INTEGER	No	No	Explain 呼び出し時に DBHEAP データベース構成設定値が入れられます。
COMM_SPEED	DOUBLE	No	No	Explain 呼び出し時に COMM_BANDWIDTH データベース構成設定値が入れられます。

表 130. EXPLAIN_INSTANCE 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
PARALLELISM	CHAR(2)	No	No	可能な値は次のとおりです。 • N = 並列性なし • P = 区画内並行性 • IP = 区分間並列性 • BP = 区画内並行性と区分間並列性
DATAJOINER	CHAR(1)	No	No	可能な値は次のとおりです。 • N = 非連合システム・プラン • Y = 連合システム・プラン

EXPLAIN_OBJECT 表

EXPLAIN_OBJECT 表は、 SQL ステートメントを満たすために生成されるアクセス・プランが必要とするデータ・オブジェクトを指定します。

この表の定義については、1428ページの『EXPLAIN_OBJECT 表の定義』を参照してください。

表 131. EXPLAIN_OBJECT 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	No	FK	この Explain 情報に関連するパッケージのセクション番号。
OBJECT_SCHEMA	VARCHAR(128)	No	No	このオブジェクトが属しているスキーマ。
OBJECT_NAME	VARCHAR(128)	No	No	オブジェクト名。
OBJECT_TYPE	CHAR(2)	No	No	オブジェクトのタイプの記述ラベル。
CREATE_TIME	TIMESTAMP	Yes	No	オブジェクトの作成時刻。表関数の場合はヌル値。

表 131. EXPLAIN_OBJECT 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
STATISTICS_TIME	TIMESTAMP	Yes	No	このオブジェクトを最後に更新した時刻。このオブジェクトに統計がない場合はヌル値。
COLUMN_COUNT	SMALLINT	No	No	このオブジェクトの列数。
ROW_COUNT	INTEGER	No	No	このオブジェクトの行数の見積もり。
WIDTH	INTEGER	No	No	オブジェクトの平均幅をバイトで示したもの。索引の場合は -1 に設定します。
PAGES	INTEGER	No	No	オブジェクトがバッファー・プール内で必要とするページ数の見積もり。表関数には、-1 に設定します。
DISTINCT	CHAR(1)	No	No	オブジェクトの行が特殊である (複写ではない) かを示す。 可能な値は次のとおりです。 **Y** Yes **N** No
TABLESPACE_NAME	VARCHAR(128)	Yes	No	このオブジェクトが保管されている表スペースの名前。表スペースが関係していない場合は、ヌル値に設定する。
OVERHEAD	DOUBLE	No	No	指定された表スペースへのランダム入出力 1 度に対するオーバーヘッドの合計の見積もり (ミリ秒)。コントローラー・オーバーヘッド、ディスク・シーク、および待ち時間を含む。表スペースが関係していない時は -1 に設定します。
TRANSFER_RATE	DOUBLE	No	No	指定の表スペースからデータ・ページを読み取るための時間の見積もり (ミリ秒単位)。表スペースが関係していない時は -1 に設定します。
PREFETCHSIZE	INTEGER	No	No	事前取り出しの実行時に読み取るデータ・ページの数。表関数には、-1 に設定します。
EXTENTSIZE	INTEGER	No	No	データ・ページを単位とするエクステント・サイズ。表スペースの中のコンテナーにこの数のページが書き込まれたら、次のコンテナーに切り替わるようになります。表関数には、-1 に設定します。
CLUSTER	DOUBLE	No	No	索引とクラスター化しているデータの度数 1 以上の場合、これは CLUSTERRATIO。 0 以上かつ 1 より小さい場合、これは CLUSTERFACTOR。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。

表 131. EXPLAIN_OBJECT 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
NLEAF	INTEGER	No	No	この索引オブジェクトの値が必要とする葉ページの数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
NLEVELS	INTEGER	No	No	この索引オブジェクトのツリー内の索引レベルの数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
FULLKEYCARD	BIGINT	No	No	この索引オブジェクトに含まれる、特殊なフル・キー値の数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
OVERFLOW	INTEGER	No	No	表内のオーバーフロー・レコードの合計数。索引、表関数について、またはこの統計が利用不能な場合は、-1 に設定します。
FIRSTKEYCARD	BIGINT	No	No	最初のキーの値の種類数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
FIRST2KEYCARD	BIGINT	No	No	索引の最初の {2,3,4} 列を使用する最初の特殊キー値の数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
FIRST3KEYCARD	BIGINT	No	No	
FIRST4KEYCARD	BIGINT	No	No	
SEQUENTIAL_PAGES	INTEGER	No	No	索引キーの順序でディスクに存在し、それらの間に大きなギャップがないか、わずかなギャップしかない葉ページの数。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。
DENSITY	INTEGER	No	No	索引によって占有されているページの範囲内の、ページ数に対する SEQUENTIAL_PAGES の比率。パーセントで表現される (0〜100 の整数)。表、表関数について、またはこの統計が利用不能の場合は、-1 に設定します。

表 132. 可能な OBJECT_TYPE 値

値	説明
IX	索引
TA	表
TF	表関数

EXPLAIN_OPERATOR 表

EXPLAIN_OPERATOR 表には、 SQL コンパイラーが SQL ステートメントを満たすために必要とするすべての演算子が含まれます。

この表の定義については、1429ページの『EXPLAIN_OPERATOR 表の定義』を参照してください。

表 133. EXPLAIN_OPERATOR 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	No	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	No	No	この照会内の演算子の固有の ID。
OPERATOR_TYPE	CHAR(6)	No	No	演算子のタイプの記述ラベル。
TOTAL_COST	DOUBLE	No	No	選択したアクセス・プランを実行するとき、またこの演算子を含めるときにかかる合計コスト (タイマーオン単位) の累積の見積もり。
IO_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの実行にかかる入出力コスト (データ・ページの入出力単位) の累積の見積もり。
CPU_COST	DOUBLE	No	No	選択したアクセス・プランを実行するとき、またこの演算子を含めるときにかかる CPU コスト (命令数) の累積の見積もり。
FIRST_ROW_COST	DOUBLE	No	No	アクセス・プランへの 1 行目を取り出すとき、またこの演算子を含めるときにかかる累積合計 (timerons 単位) の見積もり。この値には、要求される初期オーバーヘッドが含まれます。
RE_TOTAL_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかるコスト (timerons 単位) の累積の見積もり。
RE_IO_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかる入出力コスト (データ・ページの入出力単位) の累積の見積もり。

表 133. EXPLAIN_OPERATOR 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
RE_CPU_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかる CPU コスト (timerons 単位) の累積の見積もり。
COMM_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの実行にかかる通信コスト (TCP/IP フレーム単位) の累積の見積もり。
FIRST_COMM_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの最初の行の取り出しにかかる通信コスト (TCP/IP フレーム単位) の累積の見積もり。この値には、要求される初期オーバーヘッドが含まれます。
BUFFERS	DOUBLE	No	No	この演算子とその入力についてのバッファー所要量の見積もり。
REMOTE_TOTAL_COST	DOUBLE	No	No	リモート・データベースの操作を実行する場合の合計コスト (timerons 単位) の累積の見積もり。
REMOTE_COMM_COST	DOUBLE	No	No	この演算子に至るまで (この演算子を含む) の、選択したリモート・アクセス・プランの実行にかかる通信コストの累積の見積もり。

表 134. OPERATOR_TYPE の値

値	説明
DELETE	削除
FETCH	取り出し
FILTER	行フィルター
GENROW	生成行
GRPBY	Group By
HSJOIN	ハッシュ結合
INSERT	挿入
IXAND	AND された動的ビットマップ索引
IXSCAN	索引走査
MSJOIN	走査結合のマージ
NLJOIN	ネスト・ループの結合
RETURN	結果
RIDSCN	行識別子 (RID) 走査
RQUERY	リモート照会
SORT	分類
TBSCAN	表走査
TEMP	一時表構造

表 134. OPERATOR_TYPE の値 (続き)

値	説明
TQ	表キュー
UNION	共用体
UNIQUE	複写除去
UPDATE	更新

EXPLAIN_PREDICATE 表

EXPLAIN_PREDICATE は、特定の演算子によって適用される述部を識別します。

この表の定義については、1430ページの『EXPLAIN_PREDICATE 表の定義』を参照してください。

表 135. EXPLAIN_PREDICATE 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	No	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	No	No	この照会内の演算子の固有の ID。
PREDICATE_ID	INTEGER	No	No	特定の演算子のための述部の固有の ID。
HOW_APPLIED	CHAR(5)	No	No	特定の演算子によって述部がどのように使用されているか。

表 135. EXPLAIN_PREDICATE 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
WHEN_EVALUATED	CHAR(3)	No	No	この述部で使用される副照会をいつ評価するかの指示。 可能な値は次のとおりです。 **ブランク** この述部には副照会が含まれません。 **EAA** この述部で使用される副照会は、適用時に評価されます (EAA)。つまり、指定の演算子によって行が処理されるたびに、述部が適用されるので副照会が再評価されます。 **EAO** この述部で使用される副照会は、オープン時に評価されます (EAO)。つまり、指定の演算子に対して副照会は 1 回だけ再評価され、結果はそれぞれの行へ述部を適用する際に再利用されます。 **MUL** この述部の副照会のタイプは複数あります。
RELOP_TYPE	CHAR(2)	No	No	この述部で使用される関係演算子のタイプ。
SUBQUERY	CHAR(1)	No	No	この述部に対して、副照会からのデータ・ストリームが要求されているか。複数の副照会ストリームが要求されることがあります。 可能な値は次のとおりです。 **N** 副照会ストリームは要求されていない **Y** 1 つ以上の副照会ストリームが要求されている
FILTER_FACTOR	DOUBLE	No	No	この述部が修飾する小数部の行数の見積もり。
PREDICATE_TEXT	CLOB(1M)	Yes	No	SQL ステートメントの内部表示から再作成された述部のテキスト。 利用不能の場合はヌル値。

表 136. 可能な HOW_APPLIED 値

値	説明
JOIN	以前は表を結合した。
RESID	残余述部として評価された。

表 136. 可能な HOW_APPLIED 値 (続き)

値	説明
SARG	索引またはデータ・ページの検索引き数述部として評価された。
START	開始条件として使用された。
STOP	停止条件として使用された。

表 137. 可能な RELOP_TYPE の値

値	説明
ブランク	適用されない
EQ	等しい
GE	より大きいかまたは等しい
GT	より大きい
IN	リストされている
LE	より小さいかまたは等しい
LK	類似
LT	より小さい
NE	等しくない
NL	ヌル値
NN	ヌル値以外

EXPLAIN_STATEMENT 表

EXPLAIN_STATEMENT 表には、 Explain 情報の異なるレベルごとに存在している SQL ステートメントのテキストが入れられます。この表には、ユーザーが入力した元の SQL ステートメントと、その SQL ステートメントを満たすアクセス・プランを選択するのに (最適化プログラムで) 使用されるバージョンが保管されます。バージョンが後のものほど、SQL コンパイラーの判断にしたがって書き直されたり、述部が追加されたりして、オリジナルとの類似性は少なくなります。

この表の定義については、1431ページの『EXPLAIN_STATEMENT 表の定義』を参照してください。

表 138. EXPLAIN_STATEMENT 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK, FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	PK, FK	Explain 要求の開始時刻。

表 138. EXPLAIN_STATEMENT 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
SOURCE_NAME	VARCHAR(128)	No	PK, FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	PK, FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	PK	この行に関連する Explain 情報のレベル。 有効な値は以下のとおりです。 **O** 元のテキスト (ユーザーが入力したもの) **P** PLAN SELECTION
STMTNO	INTEGER	No	PK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。動的 Explain SQL ステートメントの場合は 1。静的 SQL ステートメントの場合、この値は SYSCAT.STATEMENTS カタログ視点で使用されているものと同じ。
SECTNO	INTEGER	No	PK	パッケージ内のセクションのうち、この SQL ステートメントを含むもののセクション番号。動的 Explain SQL ステートメントの場合、これは、実行時にこのステートメントのセクションを保持するのに使用されるセクション番号。静的 SQL ステートメントの場合、この値は SYSCAT.STATEMENTS カタログ視点で使用されているものと同じ。
QUERYNO	INTEGER	No	No	Explain 対象の SQL ステートメントの識別子の数値識別子。動的 SQL ステートメント (Explain SQL ステートメント以外) に対して、 CLP または CLI を通して発行されるデフォルト値。この値は、順次増加します。そうでない場合、デフォルト値は静的 SQL ステートメントに対する STMTNO の値、また動的 SQL ステートメントに対しては 1 となります。
QUERYTAG	CHAR(20)	No	No	Explain 対象の各 SQL ステートメントの識別子タグ。 CLP を通して発行された動的 SQL (Explain SQL ステートメント以外) に対するデフォルト値は 'CLP'。 CLI を通して発行された動的 SQL (Explain SQL ステートメント以外) に対するデフォルト値は 'CLI'。それ以外の場合、使用されるデフォルト値はブランクです。

表 138. EXPLAIN_STATEMENT 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
STATEMENT_TYPE	CHAR(2)	No	No	Explain 対象の照会のタイプの記述ラベル。 可能な値は次のとおりです。 **S** 選択 **D** 削除 **DC** カーソルの現在位置の削除 **I** 挿入 **U** 更新 **UC** カーソルの現在位置の更新
UPDATABLE	CHAR(1)	No	No	このステートメントが更新可能とみなされるかどうか。これは、潜在的に更新可能であるとみなされる SELECT ステートメントに特に関係があります。 可能な値は次のとおりです。 ' ' 該当しない (ブランク) **N** No **Y** Yes
DELETABLE	CHAR(1)	No	No	このステートメントが削除可能とみなされるかどうか。これは、潜在的に削除可能であるとみなされる SELECT ステートメントに特に関係があります。 可能な値は次のとおりです。 ' ' 該当しない (ブランク) **N** No **Y** Yes
TOTAL_COST	DOUBLE	No	No	このステートメントについて選択されたアクセス・プランの実行のための合計コストの見積もり (timerons 単位)。 EXPLAIN_LEVEL が *O* (オリジナル・テキスト) の場合は、この時点で選択されているアクセス・プランがないため、ゼロに設定されます。
STATEMENT_TEXT	CLOB(1M)	No	No	Explain 対象の SQL ステートメントのテキスト、またはテキストの一部。 Explain 機能のプラン選択レベルで表示されるテキストは、内部表記から再構成されたものであり、本質的に SQL ステートメントに似たものです。再構成されたステートメントが正しい SQL 構文に準拠しているという保証はありません。

表 138. EXPLAIN_STATEMENT 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
SNAPSHOT	BLOB(10M)	Yes	No	示されている Explain_Level でのこの SQL ステートメントの内部表記のスナップショット。 この列は、DB2 Visual Explain で使用することを意図したものです。 EXPLAIN_LEVEL が *0* (オリジナル・テキスト) の場合は、ステートメントのこの特定バージョンが取り込まれた時点でアクセス・プランが選択されていないため、この列はヌル値に設定されます。
QUERY_DEGREE	INTEGER	No	No	Explain の呼び出し時の区画内並行性の度数。元のステートメントの場合、ここには区画内並行性の指定された度数が入ります。 PLAN SELECTION の場合、ここには使用のプランに応じて生成された区画内並行性の度数が入ります。

EXPLAIN_STREAM 表

EXPLAIN_STREAM 表は、個々の演算子とデータ・オブジェクトの間の入出力データ・ストリームを示します。データ・オブジェクト自体は、EXPLAIN_OBJECT 表で示されています。データ・ストリームに関連する演算子は、EXPLAIN_OPERATOR 表で探せます。

この表の定義については、1432ページの『EXPLAIN_STREAM 表の定義』を参照してください。

表 139. EXPLAIN_STREAM 表

列名	データ・タイプ	ヌル値可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	No	FK	この Explain 情報に関連するパッケージのセクション番号。

表 139. EXPLAIN_STREAM 表 (続き)

列名	データ・タイプ	ヌル値可能?	キー?	説明
STREAM_ID	INTEGER	No	No	このデータ・ストリームに対する特定の演算子の固有 ID。
SOURCE_TYPE	CHAR(1)	No	No	このデータ・ストリームのソースを示す。 **O** 演算子 **D** データ・オブジェクト
SOURCE_ID	SMALLINT	No	No	このデータ・ストリームのソースである照会内の、演算子に対する固有 ID。 SOURCE_TYPE が 'D' の場合、-1 に設定します。
TARGET_TYPE	CHAR(1)	No	No	このデータ・ストリームのターゲットを示します。 **O** 演算子 **D** データ・オブジェクト
TARGET_ID	SMALLINT	No	No	このデータ・ストリームのターゲットである照会内の、演算子に対する固有 ID。 TARGET_TYPE が 'D' の場合、-1 に設定します。
OBJECT_SCHEMA	VARCHAR(128)	Yes	No	影響を受けたデータ・オブジェクトが属するスキーマ。 SOURCE_TYPE および TARGET_TYPE が共に 'O' である場合、ヌル値に設定します。
OBJECT_NAME	VARCHAR(128)	Yes	No	データ・ストリームの対象となるオブジェクトの名前。 SOURCE_TYPE および TARGET_TYPE が共に 'O' である場合、ヌル値に設定します。
STREAM_COUNT	DOUBLE	No	No	データ・ストリームのカーディナリティーの見積もり。
COLUMN_COUNT	SMALLINT	No	No	データ・ストリーム内の列数。
PREDICATE_ID	INTEGER	No	No	ストリームが述部に対する副照会の一部である場合、 ID が反映される。それ以外は列は -1 に設定される。
COLUMN_NAMES	CLOB(1M)	Yes	No	この列には、このストリームに関連した列の名前や配列情報が含まれています。 名前は以下の形式に従います。 `NAME1(A)+NAME2(D)+NAME3+NAME4` ここで、*(A)* は昇順の列、 *(D)* は降順の列を示し、配列情報がないものは、列が配列されていないか、配列が関係ないかのいずれかを示します。
PMID	SMALLINT	No	No	区分化マップの ID。

表 139. EXPLAIN_STREAM 表 (続き)

列名	データ・タイプ	ヌル値可能?	キー?	説明
SINGLE_NODE	CHAR(5)	Yes	No	このデータ・ストリームが単一または複数の区分にあるかどうかを示します。 **MULT** 複数の区分にある **COOR** 調整プログラム・ノードにある **HASH** ハッシュを使用して指定される **RID** 行 ID を使用して指定される **FUNC** 関数を使用して指定される (PARTITION() または NODENUMBER()) **CORR** 相関値を使用して指定される **Numberic** 事前に決められた単一ノードに指定される
PARTITION_COLUMNS	CLOB(64K)	Yes	No	このデータ・ストリームが区分化される列のリスト。

ADVISE_INDEX 表

ADVISE_INDEX 表は、推奨された索引を示します。

この表の定義については、1433ページの『ADVISE_INDEX 表の定義』を参照してください。

表 140. ADVISE_INDEX 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	No	No	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	No	No	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	No	No	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	No	No	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	No	No	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	No	No	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。

表 140. ADVISE_INDEX 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
SECTNO	INTEGER	No	No	この Explain 情報に関連するパッケージのセクション番号。
QUERYNO	INTEGER	No	No	Explain 対象の SQL ステートメントの識別子の数値識別子。動的 SQL ステートメント (Explain SQL ステートメント以外) に対して、 CLP または CLI を通して発行されるデフォルト値。この値は、順次増加します。そうでない場合、デフォルト値は静的 SQL ステートメントに対する STMTNO の値、また動的 SQL ステートメントに対しては 1 となります。
QUERYTAG	CHAR(20)	No	No	Explain 対象の各 SQL ステートメントの識別子タグ。 CLP を通して発行された動的 SQL (Explain SQL ステートメント以外) に対するデフォルト値は 'CLP'。 CLI を通して発行された動的 SQL (Explain SQL ステートメント以外) に対するデフォルト値は 'CLI'。それ以外の場合、使用されるデフォルト値はブランクです。
NAME	VARCHAR(128)	No	No	索引の名前。
CREATOR	VARCHAR(128)	No	No	索引名の修飾子。
TBNAME	VARCHAR(128)	No	No	その索引の定義されている表またはニックネームの名前。
TBCREATOR	VARCHAR(128)	No	No	表名の修飾子。
COLNAMES	CLOB(64K)	No	No	列名のリスト。
UNIQUERULE	CHAR(1)	No	No	固有値に関する規則: D = 重複可 P = 1 次索引 U = 固有項目のみ可
COLCOUNT	SMALLINT	No	No	キー内の列数と組み込み列 (存在する場合) の数の合計。
IID	SMALLINT	No	No	索引の内部 ID。
NLEAF	INTEGER	No	No	葉ページの数。統計が収集されていない場合は -1。
NLEVELS	SMALLINT	No	No	索引レベルの数。統計が収集されていない場合は -1。
FULLKEYCARD	BIGINT	No	No	キー全体の値の種類数。統計が収集されていない場合は -1。
FIRSTKEYCARD	BIGINT	No	No	最初のキーの値の種類数。統計が収集されていない場合は -1。

表 140. ADVISE_INDEX 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
CLUSTERRATIO	SMALLINT	No	No	索引によるデータ・クラスター化の程度。統計が収集されていない場合、または詳細な索引統計が収集されている場合は -1 (それらの場合は CLUSTERFACTOR のほうが使用されます)。
CLUSTERFACTOR	DOUBLE	No	No	クラスター化の程度の詳細測定値。詳細索引統計が収集されていない場合、またはニックネームの索引が定義されていない場合は -1。
USERDEFINED	SMALLINT	No	No	ユーザーによる定義。
SYSTEM_REQUIRED	SMALLINT	No	No	この索引が基本キー制約または固有キー制約に必要であるか、またはこの索引がタイプ付き表のオブジェクト識別子 (OID) 列上にある索引の場合は 1。 この索引が基本キー制約または固有キー制約に必要であり、しかもタイプ付き表のオブジェクト識別子 (OID) 列の索引である場合は 2。 それ以外の場合は 0。
CREATE_TIME	TIMESTAMP	No	No	索引の作成された時刻。
STATS_TIME	TIMESTAMP	Yes	No	この索引について記録されている統計値が最後に変更された時刻。統計が使用可能でない場合は、ヌル値。
PAGE_FETCH_PAIRS	VARCHAR(254)	No	No	文字形式で表される、整数の対のリスト。それぞれの対は、仮のバッファー内のページ数と、その仮のバッファーを使用した表の走査に必要なページ取り出しの回数を表しています。 (データが入手できない場合は、長さ 0 のストリング。)
REMARKS	VARCHAR(254)	Yes	No	ユーザー提供のコメントまたはヌル値。
DEFINER	VARCHAR(128)	No	No	索引を作成したユーザー。
CONVERTED	CHAR(1)	No	No	将来の使用のために予約済み。
SEQUENTIAL_PAGES	INTEGER	No	No	索引キーの順序でディスクに存在し、それらの間に大きなギャップがないか、わずかなギャップしかない葉ページの数。 (統計が入手できない場合は -1。)
DENSITY	INTEGER	No	No	索引によって占有されているページの範囲内の、ページ数に対する SEQUENTIAL_PAGES の比率。パーセントで表現される (0 ～ 100 の整数。統計が入手できない場合は -1。)
FIRST2KEYCARD	BIGINT	No	No	索引の最初の 2 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。
FIRST3KEYCARD	BIGINT	No	No	索引の最初の 3 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。

表 140. ADVISE_INDEX 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
FIRST4KEYCARD	BIGINT	No	No	索引の最初の 4 つの列を使用するキーの種類数 (統計がない場合、または適用されない場合は -1)。
PCTFREE	SMALLINT	No	No	索引を最初に作成する際に予約する索引葉ページのパーセンテージ。このスペースは、索引の作成後に行う挿入用に使用可能です。
UNIQUE_COLCOUNT	SMALLINT	No	No	固有キーに必要な列の数。常に <=COLCOUNT。組み込み列がある場合にのみ < COLCOUNT。索引に固有キーがない場合は -1 (重複可能)。
MINPCTUSED	SMALLINT	No	No	ゼロでない場合は、オンライン索引再編成が使用可能になり、その値は、ページをマージをする前に使用される最小スペースのしきい値です。
REVERSE_SCANS	CHAR(1)	No	No	Y = 索引は逆走査をサポートする N = 索引は逆走査をサポートしない
USE_INDEX	CHAR(1)	Yes	No	Y = 推奨または評価された索引 N = 推奨されない索引
CREATION_TEXT	CLOB(1M)	No	No	索引の作成に使用された SQL ステートメント。
PACKED_DESC	BLOB(20M)	Yes	No	表の内部記述。

ADVISE_WORKLOAD 表

ADVISE_WORKLOAD 表は、作業負荷を形成するステートメントを示します。作業負荷についての詳細は、*管理の手引き: パフォーマンス* を参照してください。

この表の定義については、1435ページの『ADVISE_WORKLOAD 表の定義』を参照してください。

表 141. ADVISE_WORKLOAD 表

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
WORKLOAD_NAME	CHAR(128)	No	No	このステートメントが属する SQL ステートメント (作業負荷) の集合を示す名前。
STATEMENT_NO	INTEGER	No	No	この Explain 情報に関連する作業負荷のステートメント番号。
STATEMENT_TEXT	CLOB(1M)	No	No	SQL ステートメントの内容。
STATEMENT_TAG	VARCHAR(256)	No	No	Explain 対象の各 SQL ステートメントの識別子タグ。

表 141. ADVISE_WORKLOAD 表 (続き)

列名	データ・タイプ	ヌル値可能 ?	キー ?	説明
FREQUENCY	INTEGER	No	No	このステートメントが作業負荷内で出現する回数。
IMPORTANCE	DOUBLE	No	No	ステートメントの重要性。
COST_BEFORE	DOUBLE	Yes	No	推奨された索引が作成されない場合の照会にかかるコスト (timerons 単位)。
COST_AFTER	DOUBLE	Yes	No	推奨された索引が作成される場合の照会にかかるコスト (timerons 単位)。

Explain 表の定義

Explain 表は、Explain 機能を呼び出す前に作成しておく必要があります。次の定義によって、必要な Explain 表の作成方法を示します。

- 1426ページの『EXPLAIN_ARGUMENT 表の定義』
- 1427ページの『EXPLAIN_INSTANCE 表の定義』
- 1428ページの『EXPLAIN_OBJECT 表の定義』
- 1429ページの『EXPLAIN_OPERATOR 表の定義』
- 1430ページの『EXPLAIN_PREDICATE 表の定義』
- 1431ページの『EXPLAIN_STATEMENT 表の定義』
- 1432ページの『EXPLAIN_STREAM 表の定義』
- 1433ページの『ADVISE_INDEX 表の定義』
- 1435ページの『ADVISE_WORKLOAD 表の定義』

また、'sqllib' ディレクトリーの 'misc' サブディレクトリーの中の EXPLAIN.DDL ファイルに用意されているコマンド行プロセッサー用入力スクリプトの例を使用することもできます。 Explain 表が要求されているデータベースに接続します。それからコマンド `db2 -tf EXPLAIN.DDL` を発行すると表が作成されます。

EXPLAIN_ARGUMENT 表の定義

```
CREATE TABLE EXPLAIN_ARGUMENT ( EXPLAIN_REQUESTER   VARCHAR(128)  NOT NULL,
                                EXPLAIN_TIME        TIMESTAMP     NOT NULL,
                                SOURCE_NAME         VARCHAR(128)  NOT NULL,
                                SOURCE_SCHEMA       VARCHAR(128)  NOT NULL,
                                EXPLAIN_LEVEL       CHAR(1)       NOT NULL,
                                STMTNO              INTEGER       NOT NULL,
                                SECTNO              INTEGER       NOT NULL,
                                OPERATOR_ID         INTEGER       NOT NULL,
                                ARGUMENT_TYPE       CHAR(8)       NOT NULL,
                                ARGUMENT_VALUE      VARCHAR(1024) NOT NULL,
                                LONG_ARGUMENT_VALUE CLOB(1M)      NOT LOGGED,
                                   FOREIGN KEY (EXPLAIN_REQUESTER,
                                                EXPLAIN_TIME,
                                                SOURCE_NAME,
                                                SOURCE_SCHEMA,
                                                EXPLAIN_LEVEL,
                                                STMTNO,
                                                SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_INSTANCE 表の定義

```
CREATE TABLE EXPLAIN_INSTANCE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_OPTION    CHAR(1)      NOT NULL,
                                SNAPSHOT_TAKEN    CHAR(1)      NOT NULL,
                                DB2_VERSION       CHAR(7)      NOT NULL,
                                SQL_TYPE          CHAR(1)      NOT NULL,
                                QUERYOPT          INTEGER      NOT NULL,
                                BLOCK             CHAR(1)      NOT NULL,
                                ISOLATION         CHAR(2)      NOT NULL,
                                BUFFPAGE          INTEGER      NOT NULL,
                                AVG_APPLS         INTEGER      NOT NULL,
                                SORTHEAP          INTEGER      NOT NULL,
                                LOCKLIST          INTEGER      NOT NULL,
                                MAXLOCKS          SMALLINT     NOT NULL,
                                LOCKS_AVAIL       INTEGER      NOT NULL,
                                CPU_SPEED         DOUBLE       NOT NULL,
                                REMARKS           VARCHAR(254),
                                DBHEAP            INTEGER      NOT NULL,
                                COMM_SPEED        DOUBLE       NOT NULL,
                                PARALLELISM       CHAR(2)      NOT NULL,
                                DATAJOINER        CHAR(1)      NOT NULL,
                                    PRIMARY KEY (EXPLAIN_REQUESTER,
                                                 EXPLAIN_TIME,
                                                 SOURCE_NAME,
                                                 SOURCE_SCHEMA))
```

EXPLAIN_OBJECT 表の定義

```
CREATE TABLE EXPLAIN_OBJECT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128) NOT NULL,
                              OBJECT_NAME       VARCHAR(128) NOT NULL,
                              OBJECT_TYPE       CHAR(2)      NOT NULL,
                              CREATE_TIME       TIMESTAMP,
                              STATISTICS_TIME   TIMESTAMP,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              ROW_COUNT         INTEGER      NOT NULL,
                              WIDTH             INTEGER      NOT NULL,
                              PAGES             INTEGER      NOT NULL,
                              DISTINCT          CHAR(1)      NOT NULL,
                              TABLESPACE_NAME   VARCHAR(128),
                              OVERHEAD          DOUBLE       NOT NULL,
                              TRANSFER_RATE     DOUBLE       NOT NULL,
                              PREFETCHSIZE      INTEGER      NOT NULL,
                              EXTENTSIZE        INTEGER      NOT NULL,
                              CLUSTER           DOUBLE       NOT NULL,
                              NLEAF             INTEGER      NOT NULL,
                              NLEVELS           INTEGER      NOT NULL,
                              FULLKEYCARD       BIGINT       NOT NULL,
                              OVERFLOW          INTEGER      NOT NULL,
                              FIRSTKEYCARD      BIGINT       NOT NULL,
                              FIRST2KEYCARD     BIGINT       NOT NULL,
                              FIRST3KEYCARD     BIGINT       NOT NULL,
                              FIRST4KEYCARD     BIGINT       NOT NULL,
                              SEQUENTIAL_PAGES  INTEGER      NOT NULL,
                              DENSITY           INTEGER      NOT NULL,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_OPERATOR 表の定義

```
CREATE TABLE EXPLAIN_OPERATOR ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                STMTNO            INTEGER      NOT NULL,
                                SECTNO            INTEGER      NOT NULL,
                                OPERATOR_ID       INTEGER      NOT NULL,
                                OPERATOR_TYPE     CHAR(6)      NOT NULL,
                                TOTAL_COST        DOUBLE       NOT NULL,
                                IO_COST           DOUBLE       NOT NULL,
                                CPU_COST          DOUBLE       NOT NULL,
                                FIRST_ROW_COST    DOUBLE       NOT NULL,
                                RE_TOTAL_COST     DOUBLE       NOT NULL,
                                RE_IO_COST        DOUBLE       NOT NULL,
                                RE_CPU_COST       DOUBLE       NOT NULL,
                                COMM_COST         DOUBLE       NOT NULL,
                                FIRST_COMM_COST   DOUBLE       NOT NULL,
                                REMOTE_TOTAL_COST DOUBLE       NOT NULL,
                                REMOTE_COMM_COST  DOUBLE       NOT NULL,
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO)
                                     REFERENCES EXPLAIN_STATEMENT
                                     ON DELETE CASCADE )
```

EXPLAIN_PREDICATE 表の定義

```
CREATE TABLE EXPLAIN_PREDICATE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 OPERATOR_ID       INTEGER      NOT NULL,
                                 PREDICATE_ID      INTEGER      NOT NULL,
                                 HOW_APPLIED       CHAR(5)      NOT NULL,
                                 WHEN_EVALUATED    CHAR(3)      NOT NULL,
                                 RELOP_TYPE        CHAR(2)      NOT NULL,
                                 SUBQUERY          CHAR(1)      NOT NULL,
                                 FILTER_FACTOR     DOUBLE       NOT NULL,
                                 PREDICATE_TEXT    CLOB(1M)     NOT LOGGED,
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                      REFERENCES EXPLAIN_STATEMENT
                                      ON DELETE CASCADE )
```

EXPLAIN_STATEMENT 表の定義

```
CREATE TABLE EXPLAIN_STATEMENT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 QUERYNO           INTEGER      NOT NULL,
                                 QUERYTAG          CHAR(20)     NOT NULL,
                                 STATEMENT_TYPE    CHAR(2)      NOT NULL,
                                 UPDATABLE         CHAR(1)      NOT NULL,
                                 DELETABLE         CHAR(1)      NOT NULL
                                 TOTAL_COST        DOUBLE       NOT NULL,
                                 STATEMENT_TEXT    CLOB(1M)     NOT NULL
                                                                NOT LOGGED,
                                 SNAPSHOT          BLOB(10M)    NOT LOGGED,
                                 QUERY_DEGREE      INTEGER      NOT NULL,
                                     PRIMARY KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO),
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA)
                                     REFERENCES EXPLAIN_INSTANCE
                                     ON DELETE CASCADE )
```

EXPLAIN_STREAM 表の定義

```
CREATE TABLE EXPLAIN_STREAM ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              STREAM_ID         INTEGER      NOT NULL,
                              SOURCE_TYPE       CHAR(1)      NOT NULL,
                              SOURCE_ID         SMALLINT     NOT NULL,
                              TARGET_TYPE       CHAR(1)      NOT NULL,
                              TARGET_ID         SMALLINT     NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128),
                              OBJECT_NAME       VARCHAR(128),
                              STREAM_COUNT      DOUBLE       NOT NULL,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              PREDICATE_ID      INTEGER      NOT NULL,
                              COLUMN_NAMES      CLOB(1M)     NOT LOGGED,
                              PMID              SMALLINT     NOT NULL,
                              SINGLE_NODE       CHAR(5),
                              PARTITION_COLUMNS CLOB(64K)    NOT LOGGED,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

ADVISE_INDEX 表の定義

```
CREATE TABLE ADVISE_INDEX (EXPLAIN_REQUESTER VARCHAR(128) NOT NULL
                              WITH DEFAULT '',
                           EXPLAIN_TIME      TIMESTAMP    NOT NULL
                              WITH DEFAULT CURRENT TIMESTAMP,
                           SOURCE_NAME       VARCHAR(128) NOT NULL
                              WITH DEFAULT '',
                           SOURCE_SCHEMA     VARCHAR(128) NOT NULL
                              WITH DEFAULT '',
                           EXPLAIN_LEVEL     CHAR(1)      NOT NULL
                              WITH DEFAULT '',
                           STMTNO            INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           SECTNO            INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           QUERYNO           INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           QUERYTAG          CHAR(20)     NOT NULL
                              WITH DEFAULT '',
                           NAME              VARCHAR(128) NOT NULL,
                           CREATOR           VARCHAR(128) NOT NULL
                              WITH DEFAULT '',
                           TBNAME            VARCHAR(128) NOT NULL,
                           TBCREATOR         VARCHAR(128) NOT NULL
                              WITH DEFAULT '',
                           COLNAMES          CLOB(64K)    NOT NULL,
                           UNIQUERULE        CHAR(1)      NOT NULL
                              WITH DEFAULT '',
                           COLCOUNT          SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           IID               SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           NLEAF             INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           NLEVELS           SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           FIRSTKEYCARD      BIGINT       NOT NULL
                              WITH DEFAULT 0,
                           FULLKEYCARD       BIGINT       NOT NULL
                              WITH DEFAULT 0,
                           CLUSTERRATIO      SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           CLUSTERFACTOR     DOUBLE       NOT NULL
                              WITH DEFAULT 0,
                           USERDEFINED       SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           SYSTEM_REQUIRED   SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           CREATE_TIME       TIMESTAMP    NOT NULL
                              WITH DEFAULT CURRENT TIMESTAMP,
                           STATS_TIME        TIMESTAMP
                              WITH DEFAULT CURRENT TIMESTAMP,
                           PAGE_FETCH_PAIRS  VARCHAR(254) NOT NULL
                              WITH DEFAULT '',
```

```
REMARKS           VARCHAR(254)
                     WITH DEFAULT '',
DEFINER           VARCHAR(128) NOT NULL
                     WITH DEFAULT '',
CONVERTED          CHAR(1) NOT NULL
                     WITH DEFAULT '',
SEQUENTIAL_PAGES  INTEGER      NOT NULL
                     WITH DEFAULT 0,
DENSITY           INTEGER      NOT NULL
                     WITH DEFAULT 0,
FIRST2KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
FIRST3KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
FIRST4KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
PCTFREE           SMALLINT     NOT NULL
                     WITH DEFAULT -1,
UNIQUE_COLCOUNT   SMALLINT     NOT NULL
                     WITH DEFAULT -1,
MINPCTUSED        SMALLINT     NOT NULL
                     WITH DEFAULT 0,
REVERSE_SCANS     CHAR(1)      NOT NULL
                     WITH DEFAULT 'N',
USE_INDEX         CHAR(1),
CREATION_TEXT     CLOB(1M)     NOT NULL
                     NOT LOGGED WITH DEFAULT '',
PACKED_DESC       BLOB(1M)     NOT LOGGED)
```

ADVISE_WORKLOAD 表の定義

```
CREATE TABLE ADVISE_WORKLOAD (WORKLOAD_NAME  CHAR(128)    NOT NULL
                                WITH DEFAULT 'WK0',
                              STATEMENT_NO   INTEGER      NOT NULL
                                WITH DEFAULT 1,
                              STATEMENT_TEXT CLOB(1M)     NOT NULL NOT LOGGED,
                              STATEMENT_TAG  VARCHAR(256) NOT NULL
                                WITH DEFAULT '',
                              FREQUENCY      INTEGER      NOT NULL
                                WITH DEFAULT 1,
                              IMPORTANCE     DOUBLE       NOT NULL
                                WITH DEFAULT 1,
                              COST_BEFORE    DOUBLE,
                              COST_AFTER     DOUBLE)
```

付録L. Explain レジスターの値

この付録は、 CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスターの値の相互作用、また PREP および BIND コマンドとの相互作用を説明します。

CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスター値は、以下のような方法で動的 SQL に作用します。

表 142. 動的 SQL に対する Explain 特殊レジスターの値の作用

EXPLAIN SNAPSHOT の値	EXPLAIN MODE の値				
	NO	YES	EXPLAIN	RECOMMEND INDEXES	EVALUATE INDEXES
NO	• 戻された照会の結果	• 移植された Explain 表 • 戻された照会の結果	• 移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引	• 移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引
YES	• 取られた Explain スナップショット • 戻された照会の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻された照会の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引

表 142. 動的 SQL に対する Explain 特殊レジスターの値の作用 (続き)

EXPLAIN SNAPSHOT の値	EXPLAIN MODE の値				
	NO	YES	EXPLAIN	RECOMMEND INDEXES	EVALUATE INDEXES
EXPLAIN	• 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引	• 移植された Explain 表 • 取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引

CURRENT EXPLAIN MODE 特殊レジスターは、動的 SQL に対して以下のような方法で EXPLAIN バインド・オプションに作用します。

表 143. EXPLAIN バインド・オプションと CURRENT EXPLAIN MODE の作用

EXPLAIN MODE の値	EXPLAIN バインド・オプションの値		
	NO	YES	ALL
NO	• 戻された照会の結果	• 静的 SQL に対して移植された Explain 表 • 戻された照会の結果	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻された照会の結果
YES	• 動的 SQL に対して移植された Explain 表 • 戻された照会の結果	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻された照会の結果	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻された照会の結果

表 143. *EXPLAIN バインド・オプションと CURRENT EXPLAIN MODE の作用 (続き)*

EXPLAIN MODE の値	EXPLAIN バインド・オプションの値		
	NO	YES	ALL
EXPLAIN	• 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果
RECOMMEND INDEXES	• 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 推奨された索引
EVALUATE INDEXES	• 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引	• 静的 SQL に対して移植された Explain 表 • 動的 SQL に対して移植された Explain 表 • 戻されなかった照会 (実行されなかった動的ステートメント) の結果 • 評価された索引

CURRENT EXPLAIN SNAPSHOT 特殊レジスターは、動的 SQL に対して以下のような方法で EXPLSNAP バインド・オプションに作用します。

表 144. *EXPLSNAP バインド・オプションと CURRENT EXPLAIN SNAPSHOT の作用*

EXPLAIN SNAPSHOT の値	EXPLSNAP バインド・オプションの値		
	NO	YES	ALL
NO	• 戻された照会の結果	• 静的 SQL に対して取られた Explain スナップショット • 戻された照会の結果	• 静的 SQL に対して取られた Explain スナップショット • 動的 SQL に対して取られた Explain スナップショット • 戻された照会の結果

表 144. EXPLSNAP バインド・オプションと CURRENT EXPLAIN SNAPSHOT の作用 (続き)

EXPLAIN SNAPSHOT の値	EXPLSNAP バインド・オプションの値		
	NO	YES	ALL
YES	• 動的 SQL に対して取られた Explain スナップショット • 戻された照会の結果	• 静的 SQL に対して取られた Explain スナップショット • 動的 SQL に対して取られた Explain スナップショット • 戻された照会の結果	• 静的 SQL に対して取られた Explain スナップショット • 動的 SQL に対して取られた Explain スナップショット • 戻された照会の結果
EXPLAIN	• 動的 SQL に対して取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 静的 SQL に対して取られた Explain スナップショット • 動的 SQL に対して取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果	• 静的 SQL に対して取られた Explain スナップショット • 動的 SQL に対して取られた Explain スナップショット • 戻されなかった照会 (実行されなかった動的ステートメント) の結果

付録M. 再帰の例: 部品構成表

部品構成表 (BOM) のアプリケーションは、多くの業務環境において一般的に必要になります。 BOM アプリケーションの再帰的共通表式の機能を示すため、部品とそれに関連する副部品、そして各部品に必要な副部品の数量を示す表について考えてみます。この例では、以下のようにして表を作成します。

```
CREATE TABLE PARTLIST
               (PART VARCHAR(8),
                SUBPART VARCHAR(8),
                QUANTITY INTEGER);
```

この例の照会結果を示すために、PARTLIST 表には次のようなデータが入っているものとします。

```
PART     SUBPART  QUANTITY
-------- -------- -----------
00       01                5
00       05                3
01       02                2
01       03                3
01       04                4
01       06                3
02       05                7
02       06                6
03       07                6
04       08               10
04       09               11
05       10               10
05       11               10
06       12               10
06       13               10
07       14                8
07       12                8
```

例 1: 単一レベルの展開

最初の例は、単一レベルの展開と呼ばれるものです。これは「'01' で示されている部品を作成するにはどの部品が必要か」という質問に答えるものです。このリストには、直接の副部品、副部品の副部品、・・・が含まれます。しかし、ある部品が何回も使用される場合でも、その副部品は 1 回しかリストに示されません。

```
WITH RPL (PART, SUBPART, QUANTITY) AS
     (  SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
        FROM PARTLIST ROOT
        WHERE ROOT.PART = '01'
```

```
      UNION ALL
        SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
        FROM RPL PARENT, PARTLIST CHILD
        WHERE  PARENT.SUBPART = CHILD.PART
     )
SELECT DISTINCT PART, SUBPART, QUANTITY
 FROM RPL
  ORDER BY PART, SUBPART, QUANTITY;
```

上記の照会では、*RPL* という名前で指定されている共通表式が含まれており、それによってこの照会の再帰的な部分が表されています。この照会には、再帰的共通表式の基本的な要素が示されています。

UNION の第 1 オペランド (全選択) は*初期化全選択* と呼ばれるもので、それによって部品 '01' の直接の子が求まります。この全選択の FROM 文節ではソース表が参照されていますが、それ自身 (この場合は *RPL*) を参照することはありません。この最初の全選択の結果が、共通表式 *RPL* (再帰的 PARTLIST) の中に入れられることになります。この例の場合、UNION は常に UNION ALL でなければなりません。

UNION の第 2 オペランド (全選択) は、 *RPL* を使って、副部品の副部品を計算しています。これは、FROM 文節で共通表式 *RPL* とソース表 (CHILD: 子) の部品を、 *RPL* (PARENT: 親) に含まれている現行の結果の副部品に結び付けることによります。この結果は、再度 *RPL* に入れられます。このようにして、UNION の第 2 オペランドは、子が存在しなくなるまで繰り返し使用されます。

この照会の主要な全選択の SELECT DISTINCT では、同じ部品 / 副部品が 2 回以上リストに現れることがないようにしています。

照会結果は、次のようになります。

```
PART     SUBPART  QUANTITY
-------- -------- -----------
01       02                 2
01       03                 3
01       04                 4
01       06                 3
02       05                 7
02       06                 6
03       07                 6
04       08                10
04       09                11
05       10                10
05       11                10
```

```
06       12                10
06       13                10
07       12                 8
07       14                 8
```

この結果では、部品 '01' から '02' へ、そこからさらに '06' へと進むようになっています。さらに、部品 '06' へは、'01' から直接に 1 回、 '02' から 1 回の計 2 回達することに注意してください。しかしこの出力では、その副構成要素が 1 回しかリストに現れないようになっています (これは SELECT DISTINCT を使用した結果です)。

再帰的共通表式では、*無限ループ* になる可能性を必ず考慮に入れてください。この例で、親表と子表を結合する第 2 オペランドの探索条件を以下のようにコーディングしたとすると、無限ループが作成されることになります。

```
PARENT.SUBPART = CHILD.SUBPART
```

無限ループが発生するこの例は、意図したとおりにコーディングされていない場合であることは明らかです。しかし、再帰サイクルが必ず終了するようにコーディングすることにも注意を払ってください。

この例の照会によって得られる結果は、再帰的共通表式を使用しなくても、アプリケーション・プログラム内で作成することができます。しかし、そのような方法では、すべての再帰レベルごとに新しい照会を開始する必要があります。さらに、すべての結果をデータベースに入れ、その結果を並べ替えることを、アプリケーションで行う必要があります。そのような方法では、アプリケーションのロジックが複雑になり、パフォーマンスはよくありません。合計型展開やインデント型展開の照会など、その他の部品構成表の照会では、アプリケーションのロジックがさらに複雑で効率の悪いものとなってしまいます。

例 2: 合計型展開

2 番目の例は、合計型展開の例です。ここでの質問は、「部品 '01' の作成には各部品が合計どれくらい必要か」というものです。単一レベル展開と異なる主な点は、数量を集計する必要があるということです。例 1 は、部品が必要になったときにそれに必要な副部品の数量を示すものです。部品 '01' を作成するのに、副部品が結局どれだけ必要かは示されていません。

```
WITH RPL (PART, SUBPART, QUANTITY) AS
   (
      SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
       FROM PARTLIST ROOT
       WHERE ROOT.PART = '01'
       UNION ALL
      SELECT PARENT.PART, CHILD.SUBPART, PARENT.QUANTITY*CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
```

```
      WHERE PARENT.SUBPART = CHILD.PART
  )
SELECT PART, SUBPART, SUM(QUANTITY) AS "Total QTY Used"
 FROM RPL
  GROUP BY PART, SUBPART
  ORDER BY PART, SUBPART;
```

上記の照会では、 *RPL* という名前で指定されている再帰的共通表式の中の UNION の第 2 オペランドの選択リストによって、数量の合計が示されています。副部品の使用量を求めるには、親の数量に、親 1 個当たりの子の数量を乗算します。 1 つの部品が異なる複数の場所で何回も使用される場合は、もう 1 つ最終的な集計が必要になります。これは、共通表式 *RPL* をグループ化し、主要全選択の選択リストの中で SUM 列関数を使用することによって行います。

照会結果は、次のようになります。

```
PART     SUBPART  Total Qty Used
-------- -------- --------------
01       02                    2
01       03                    3
01       04                    4
01       05                   14
01       06                   15
01       07                   18
01       08                   40
01       09                   44
01       10                  140
01       11                  140
01       12                  294
01       13                  150
01       14                  144
```

この出力のうち、副部品が '06' の行に注目してください。合計使用量の値 15 は、部品 '01' のための直接の数 3 と、部品 '02' のための数 (6) に部品 '01' の数 (2) を掛けたものとを加えた数です。

例 3: 深さの制御

この表の中に存在する部品のレベルが、とりあえず照会で必要なレベルより深い場合はどうなるのでしょうか。つまり、「'01' で指定される部品を作成するために必要な部品の最初の 2 つのレベルはどんなものか」という質問に答えるためには、どんな照会を作成したらよいでしょうか。例をわかりやすいものにするため、レベル番号を結果に含めることにします。

```
WITH RPL (LEVEL, PART, SUBPART, QUANTITY) AS
     (
        SELECT 1,              ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
         FROM PARTLIST ROOT
```

```
        WHERE ROOT.PART = '01'
      UNION ALL
        SELECT PARENT.LEVEL+1, CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
         FROM RPL PARENT, PARTLIST CHILD
         WHERE PARENT.SUBPART = CHILD.PART
           AND PARENT.LEVEL < 2
    )
SELECT PART, LEVEL, SUBPART, QUANTITY
  FROM RPL;
```

この照会は例 1 に似ています。元の部品からのレベルを示すために、列 *LEVEL* を使っています。初期化全選択では、*LEVEL* 列の値を 1 に初期化しています。それ以降の全選択では、親のレベルに 1 ずつ加算します。次に、結果のレベル数を制御するため、2 番目の全選択に、親のレベルが 2 未満でなければならないという条件を含めています。これによって、2 番目の全選択では、子の処理が第 2 レベルまでしか行われないことになります。

照会結果は、次のようになります。

```
PART     LEVEL       SUBPART  QUANTITY
-------- ----------- -------- -----------
01                 1 02                 2
01                 1 03                 3
01                 1 04                 4
01                 1 06                 3
02                 2 05                 7
02                 2 06                 6
03                 2 07                 6
04                 2 08                10
04                 2 09                11
06                 2 12                10
06                 2 13                10
```

付録N. 例外表

例外表は、 IMMEDIATE CHECKED オプションを指定した SET INTEGRITY を使用して検査する対象として指定された表の定義を模倣して作成されたユーザー作成の表です。例外表は、検査対象の表の行のうち制約に違反しているもののコピーを保管するのに使用されます。

LOAD で使用する例外表は、ここで使用するものと同じです。 したがってそれらは、SET INTEGRITY ステートメントでの検査中に再使用できます。

例外表の作成規則

例外表を作成する際の規則は、次のとおりです。

1. 例外表の最初の『n』列は、検査対象の表の列と同じです。名前、タイプ、および長さなど、すべての列属性が同じでなければなりません。
2. 例外表のすべての列は、制約やトリガーに束縛されないようにする必要があります。制約には、参照保全、検査制約、さらには挿入時にエラーの原因となる固有索引の制約が含まれます。
3. 例外表の第『(n+1)』列は、任意選択の TIMESTAMP 列です。これは、データを検査するための SET INTEGRITY を発行する前に例外表内の行が削除されなかった場合に、同じ表の SET INTEGRITY ステートメントによる検査の連続呼び出しを検出するのに使用します。
4. 第『(n+2)』列は、CLOB(32K) タイプまたはそれより大きいタイプでなければなりません。この列は、行内のデータが違反している制約の名前を示すために使用されるもので、任意選択ではありますが、なるべく使用するようにしてください。この列を用意しなかった場合 (たとえば、元の表の列数がすでに可能な最大値になっていた場合にはそれが可能)、制約違反が検出された行だけがコピーされます。
5. 『(n+1)』列と『(n+2)』列の両方を備えた例外表を作成する必要があります。
6. 上記の追加列の特定の名前には制約がありません。しかし、タイプの指定は、正確でなければなりません。
7. それ以外の列は使用できません。
8. 元の表に DATALINK 列がある場合は、例外表の対応する列に NO LINK CONTROL を指定する必要があります。これを指定しておけば、行

(DATALINK 列を持つ) が挿入されてもファイルにリンクされることはなく、例外表から行が選択されてもアクセス・トークンは生成されません。

9. 生成される列 (IDENTITY プロパティーも含む) が元の表にある場合は、例外表の対応する列に、生成されるプロパティーを指定しないでください。
10. また、SET INTEGRITY を呼び出してデータを検査するユーザーには、例外表に対して INSERT 特権が必要です。

「メッセージ」列の情報は、以下の構造に従うものになります。

表 145. 例外表のメッセージ列の構造

フィールド番号	内容	サイズ	コメント
1	制約違反の数	5 バイト	先頭に '0' を付加して右そろえ
2	最初の制約違反の種類	1 バイト	'K' - 検査制約違反 'F' - 外部キー違反 'G' - 生成される列違反 'I' - 固有索引違反[a] 'L' - DATALINK ロード違反
3	制約 / 列[b]/ 索引 ID[c] /DLVDESC[d] の長さ	5 バイト	先頭に '0' を付加して右そろえ
4	制約名 / 列名[b]/ 索引 ID[c] /DLVDESC[d]	直前のフィールドで指定される長さ	
5	区切り記号	3 バイト	<スペース><コロン><スペース>
6	次の制約違反の種類	1 バイト	'K' - 検査制約違反 'F' - 外部キー違反 'G' - 生成される列違反 'I' - 固有索引違反 'L' - DATALINK ロード違反
7	制約 / 列 / 索引 ID / DLVDESC の長さ	5 バイト	先頭に '0' を付加して右そろえ
8	制約名 / 列名 / 制約 ID/ DLVDESC	直前のフィールドで指定される長さ	
.....	.....	.....	違反ごとにフィールド 5 ～ 8 を繰り返す。

表 145. 例外表のメッセージ列の構造 (続き)

フィールド番号	内容	サイズ	コメント

- [a]SET INTEGRITY を使用した検査では固有索引違反は起こりません。しかし、FOR EXCEPTION オプションを選択した場合に LOAD を実行すると、それが報告されます。一方、LOAD は検査制約、生成される列、および外部キーに関する違反を例外表に報告しません。
- [b] 生成される列の式をカタログ視点から取り出すには、選択ステートメントを使用します。たとえば、フィールド 4 が MYSCHEMA.MYTABLE.GEN_1 の場合、 SELECT SUBSTR(TEXT, 1, 50) FROM SYSCAT.COLUMNS WHERE TABSCHEMA='MYSCHEMA' AND TABNAME='MYNAME' AND COLNAME='GEN_1'; は、式の最初の 50 文字を ″AS (<expression>)″ の形式で戻します。
- [c] カタログ視点から索引 ID を取り出すには、選択ステートメントを使用します。たとえば、フィールド 4 が 1234 であれば、 SELECT INDSCHEMA, INDNAME FROM SYSCAT.INDEXES WHERE IID=1234 となります。
- [d]DLVDESC は、次の表で説明する DATALINK ロード違反記述子 (DATALINK Load Violation DESCriptor) です。

表 146. DATALINK ロード違反記述子 (DLVDESC)

フィールド番号	内容	サイズ	コメント
1	違反している DATALINK 列の数	4 バイト	先頭に '0' を付加して右そろえ
2	最初に違反している列の DATALINK 列番号	4 バイト	先頭に '0' を付加して右そろえ
2	2 番目に違反している列の DATALINK 列番号	4 バイト	先頭に '0' を付加して右そろえ
.....	.....	.....	違反している列番号ごとに繰り返し

注:

- DATALINK 列番号は、該当する表の SYSCAT.COLUMNS に示される COLNO です。

例外表の行の処理

例外表の情報は、必要な任意の方法で処理できます。その行を使用することにより、データを修正したり、元の表にその行を再び挿入したりすることができます。

元の表に INSERT トリガーがないなら、例外表に対する副照会を含む INSERT ステートメントを発行することによって、修正した行を転送します。

INSERT トリガーがあり、トリガーを起動することなく例外表からの修正済みの行によるロードを完了したい場合は、次のような方法があります。

- 目的に合わせて明示的に定義された列において、値に応じて起動されるように INSERT トリガーを設計します。
- 例外表からのデータをアンロードして、LOAD を使用してそれらを付加します。 このケースでデータを再検査する場合、 DB2 バージョン 7 では制約違反検査は付加された行のみに限定されません。
- 関連するカタログ表のトリガー・テキストを保存します。次に、INSERT トリガーを除去し、INSERT を使用して修正された行を例外表から転送します。最後に、保存した情報を使ってもう一度トリガーを作成します。

DB2 バージョン 7 では、例外表から行を挿入する際にトリガーが起動しないように防止する明示的な備えは行われません。

固有索引の違反に対しては、行ごとに 1 つの違反しか報告されません。

長形式ストリングまたは LOB データ・タイプの値が表の中に含まれている場合、固有索引違反があっても、その値は例外表に挿入されません。

例外表の照会

例外表のメッセージ列の構造は、前述の制約の名前、長さ、および区切り文字を連結したリストです。 この情報についての照会を作成したい場合があるかもしれません。

たとえば、すべての違反のリストを作成し、各行ごとにその制約名だけを示すための照会を作成してみましょう。元の表 T1 に 2 つの列 C1 および C2 があるとします。また、対応する例外表 E1 には、T1 内のそれらの列に対応する列 C1 および C2 と、メッセージ列としての MSGCOL があるとします。以下の照会では再帰を使っており、行ごとに 1 つの制約名を示します (複数の違反については行を反復します)。

```
WITH IV  (C1, C2, MSGCOL, CONSTNAME, I, J) AS
  (SELECT C1, C2, MSGCOL,
       CHAR(SUBSTR(MSGCOL, 12,
                   INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0)))),
       1,
       15+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0))
     FROM E1
   UNION ALL
   SELECT C1, C2, MSGCOL,
       CHAR(SUBSTR(MSGCOL, J+6,
                 INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0)))),
       I+1,
```

```
      J+9+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0))
    FROM IV
    WHERE I < INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,1,5)),5,0))
) SELECT C1, C2, CONSTNAME FROM IV;
```

特定の制約に違反したすべての行のリストを作成したい場合は、次のようにしてこの照会を拡張します。

```
WITH IV  (C1, C2, MSGCOL, CONSTNAME, I, J) AS
  (SELECT C1, C2, MSGCOL,
      CHAR(SUBSTR(MSGCOL, 12,
                  INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0)))),
      1,
      15+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0))
    FROM E1
   UNION ALL
   SELECT C1, C2, MSGCOL,
      CHAR(SUBSTR(MSGCOL, J+6,
                INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0)))),
      I+1,
      J+9+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0))
    FROM IV
    WHERE I < INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,1,5)),5,0))
) SELECT C1, C2, CONSTNAME FROM IV WHERE CONSTNAME = 'constraintname';
```

検査制約のすべての違反のリストを作成するには、以下のものを実行します。

```
WITH IV  (C1, C2, MSGCOL, CONSTNAME, CONSTTYPE, I, J) AS
  (SELECT C1, C2, MSGCOL,
      CHAR(SUBSTR(MSGCOL, 12,
                  INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0)))),
      CHAR(SUBSTR(MSGCOL, 6, 1)),
      1,
      15+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,7,5)),5,0))
    FROM E1
   UNION ALL
   SELECT C1, C2, MSGCOL,
      CHAR(SUBSTR(MSGCOL, J+6,
                INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0)))),
      CHAR(SUBSTR(MSGCOL, J, 1)),
      I+1,
      J+9+INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,J+1,5)),5,0))
    FROM IV
    WHERE I < INTEGER(DECIMAL(VARCHAR(SUBSTR(MSGCOL,1,5)),5,0))
) SELECT C1, C2, CONSTNAME FROM IV WHERE CONSTTYPE = 'K';
```

付録O. 日本語および繁体字中国語 EUC についての考慮事項

日本語および中国語 (繁体字) の拡張 UNIX コード (EUC) では、 1 ～ 4 文字セットをサポートできるエンコード規則の集合を定義しています。日本語 EUC (eucJP) や中国語 (繁体字) EUC (eucTW) など、場合によっては 2 バイトより多いバイト数を使用して文字がエンコードされている場合もあります。このようなエンコード・スキームを使用するのは、データベース・サーバーまたはデータベース・クライアントのコード・ページとして使用する場合です。主な考慮事項には、以下のものがあります。

- EUC コード・ページと 2 バイトのコード・ページとの間で変換されるときに拡張または短縮されるストリング
- eucJP (日本語) または eucTW (中国語 (繁体字)) コード・ページで定義されたデータベース・サーバー中に格納された図形データのコード・ページとして使用される汎用文字セット 2 (UCS-2)

これらの考慮事項を除き、EUC を使用しても、 2 バイト文字セット (DBCS) サポートと矛盾しません。本書 (および他の資料) を通じて、 *2 バイト文字* に言及している箇所は、*マルチバイト文字* に変更されています。このため、2 バイトより多いバイト数を必要とする文字表示を可能にするエンコード規則のサポートが可能になります。日本語および繁体字中国語 EUC のサポートについての詳細な考慮事項は、この付録に載せられており、本書の本文の内容と同じ構成になっています。この情報は、EUC データベース・サーバーまたは EUC データベース・クライアントで SQL を使用する場合に考慮する必要があり、 *アプリケーション開発の手引き* のアプリケーション開発情報とともに使用します。

言語エレメント

文字

各マルチバイト文字は*文字* と見なされますが、例外として 2 バイトのブランク文字は*特殊文字* と見なされます。

トークン

マルチバイトの英小文字は大文字に変換されません。ただし、通常大文字に変換されるトークン中の 1 バイトの英小文字は例外です。

識別子

SQL 識別子

2 バイトのコード・ページと EUC コード・ページとの間の変換により、 2 バイト文字を 2 バイトより大きいバイト数でエンコードされるマルチバイト文字に変換できます。その結果、2 バイトのコード・ページの最大長に合う識別子が、 EUC コード・ページの長さを超えることがあります。このタイプの環境の識別子は慎重に選択し、識別子の最大長を超えないようにする必要があります。

データ・タイプ

文字ストリング

MBCS データベースでは、文字ストリングには 1 バイト文字セット (SBCS) とマルチバイト文字セット (MBCS) の文字が混合して含まれる場合があります。そのようなストリングを使用する場合、それらが文字ベース (データを文字単位で処理) であるか、またはバイト・ベース (データをバイト単位で処理) であるかによって同じ操作でも結果が異なることがあります。関数または操作の説明を調べて、混合ストリングを処理する方法を決めてください。

漢字ストリング

漢字ストリングは、一連の 2 バイト文字データとして定義されます。日本語または中国語 (繁体字) の EUC データを漢字列に記憶できるようにするために、 EUC は UCS-2 でエンコードされています。サポートされているすべてのエンコード・スキーム (たとえば、 PC または EBCDIC DBCS) の中で 2 バイト文字ではない文字は、漢字列では使用できません。 2 バイト文字以外の文字を使用すると、その結果変換中に代用文字によって置換されることがあります。そのようなデータを検索しても、入力された値と同じ値は戻されません。ホスト変数で漢字データを扱う方法の詳細については、 *アプリケーション開発の手引き* のプログラム言語の節を参照してください。

割り当てと比較

ストリングの割り当て

ストリングの変換は、割り当ての前に行われます。 eucJP/eucTW コード・ページおよび DBCS コード・ページが含まれている場合、文字ストリングは (DBCS から eucJP/eucTW へ) 長くなるか、または (eucJP/eucTW から DBCS へ) 短くなります。これにより、記憶域の割り当て時にエラーが生じ、検索割り当て時に切り捨てが生じることがあります。変換時の拡張のために記憶域割り当てのエラーが生じる場合、 SQLSTATE 22001 の代わりに SQLSTATE 22524 が返されます。

同様に、漢字ストリングを割り当てると、その結果 UCS-2 エンコード 2 バイト文字が、対応する 2 バイト文字を持たない文字用に、 PC または EBCDIC DBCS コード・ページ中の代用文字に変換されることがあります。 SQLCA の SQLWARN10 フィールドを 'W' に設定すると、文字を代用文字に置換する割り当てはこのようになります。

マルチバイト文字ストリングを含む検索割り当ての過程における切り捨ての場合、切り捨てのポイントはマルチバイト文字の一部になることがあります。この場合、文字の一部である各バイトは、1 バイトのブランクに置換されます。このため、複数の 1 バイトのブランクが、切り捨てられた文字ストリングの終わりに現れることがあります。

ストリングの比較

ストリングの比較は、バイト・ベースで行われます。文字ストリングは、データベース用に定義された照合順序も使用します。漢字ストリングは照合順序を使用せず、eucJP または eucTW データベースでは、 UCS-2 を使用してエンコードされます。このように、たとえ同じ文字が含まれていても、2 つの混合文字ストリングの比較と、 2 つの漢字ストリングの比較とでは異なる結果になることがあります。同様に、その結果生じる混合文字列および漢字列の分類順序は異なることがあります。

結果データ・タイプに関する規則

文字ストリングの結果データ・タイプは、ストリングが拡張しても影響を受けません。たとえば、2 つの CHAR オペランドを結合しても CHAR のままです。しかし、文字ストリングのオペランドの 1 つを変換して最大拡張により長さ属性が 2 つのオペランドのうち最大になるようにする場合、結果文字ストリングの長さ属性は影響を受けます。たとえば、データ・タイプが VARCHAR(100) と VARCHAR(120) である CASE 式の結果式を考えます。VARCHAR(100) 式は (変換する必要があるかもしれない) 混合ストリングのホスト変数であり、 VARCHAR(120) 式は eucJP データベース中の列であると仮定します。可能な変換に備えて VARCHAR(100) が 2 バイトになっているため、結果データ・タイプは VARCHAR(200) です。同じシナリオで eucJP データベースまたは eucTW データベースを使用しない場合、結果タイプは VARCHAR(120) になります。

ホスト変数長が 2 倍になっていることは、データベース・サーバーが日本語 EUC または繁体字中国語 EUC であるということに基づいていることに注意してください。クライアントが eucJP または eucTW であったとしても、ホスト変数長は 2 倍にされます。これにより、同じアプリケーション・パッケージを 2 バイトまたはマルチバイトのクライアントが使用できます。

ストリング変換に関する規則

SQL 解説書の該当するセクションにリストされているタイプの操作は、オペランドをアプリケーションまたはデータベースのコード・ページに変換することがあります。

日本語または繁体字中国語 EUC を含む混合コード・ページ環境でそのような操作を行うと、混合文字ストリング・オペランドが長くなったり短くなったりすることがあります。そのため、結果のデータ・タイプは可能なら、最大の拡張を収容する長さ属性を持ちます。データ・タイプの長さ属性に制約がある場合、そのデータ・タイプに許された最大長が使用されます。たとえば、最大拡張が 2 倍の環境では、VARCHAR(200) ホスト変数は VARCHAR(400) として、 CHAR(200) ホスト変数は CHAR(254) として処理されます。変換されたストリングがデータ・タイプの最大長を超える場合、変換の実行時に実行時エラーが生じることがあります。たとえば、CHAR(200) と CHAR(10) を結合すると、結果データ・タイプは CHAR(254) になります。 254 文字より多い文字が必要な場合、結合した左側の値が変換されると、エラーが生じます。

場合によっては、変換のための最大増加分の余裕が取られるために、長さ属性が限界を超えることがあります。たとえば、結合に許可される列は 254 バイトまでです。したがって、可変長文字ストリング 128 バイト長として定義された DBCS 混合文字ストリングであるホスト変数が列リスト (:hv1 と呼ぶ) に含まれている結合を使用して照会を行うと、アプリケーション内での照会が 254 バイトより大きい列を持っていないように思えたとしても、データ・タイプが VARCHAR(256) に設定され、照会の準備をしているときにエラーが発生してしまいます。実際のストリングが 254 バイトを超えて拡張する可能性が低い場合は、ステートメントの作成に以下のものを使用できます。

```
SELECT CAST(:hv1 CONCAT ' AS VARCHAR(254)), C2 FROM T1
UNION
SELECT C1, C2 FROM T2
```

ヌル値ストリングをホスト変数に連結すると、キャストが実行される前に変換が行われます。この照会は、実行時に切り捨てエラーが生じることがありますが、 DBCS で eucJP/eucTW 環境に対して行うことができます。

この技法 (ヌル値ストリングとキャストとの連結) を使用して、 SELECT DISTINCT の同じ 254 バイト限界を処理するか、または ORDER BY または GROUP BY 文節の列を使用することができます。

定数

漢字ストリング定数

日本語または中国語 (繁体字) EUC クライアントの場合、 1 バイト文字またはマルチバイト文字を入れることができます (混合文字ストリングと同様)。ストリングに 2 000 文字より多くの文字を含めることはできません。漢字定数では、すべての関連 PC および EBCDIC 2 バイト・コード・ページの 2 バイト文字に変換される文字だけを使用することをお勧めします。 SQL ステートメント中の漢字ストリング定数は、クライアント・コード・ページからデータベース・サーバーの 2 バイト・エンコードに変換されます。日本語または繁体字中国語 EUC サーバーでは、定数は UCS-2、すなわち漢字ストリングに使用される 2 バイト・エンコードに変換されます。 2 バイト・サーバーの場合、定数はクライアントのコード・ページからサーバーの DBCS コード・ページに変換されます。

関数

ユーザー定義関数の設計においては、パラメーターのデータ・タイプに日本語 EUC または中国語 (繁体字) EUC をサポートすることの影響を考慮する必要があります。関数解決の一部では、関数呼び出しに対する引き数のデータ・タイプを考慮します。日本語または繁体字中国語 EUC クライアントを含む混合文字ストリング引き数では、引き数を指定するために追加バイトが必要になることがあります。これには、長さを大きくするためにデータ・タイプの変更が必要になることがあります。たとえば、サーバーの VARCHAR(4000) ストリングに合うアプリケーション (LONG VARCHAR) の文字ストリングを表示するのに、 4001 バイトが必要になることがあります。引き数が LONG VARCHAR であることを示す関数シグニチャーが含まれていない場合、関数解決は関数を見つけることができません。

種々の理由で長ストリングが認められていない関数があります。そのような関数に LONG VARCHAR または CLOB 引き数を使用しても成功しません。たとえば、組み込み POSSTR 関数の 2 番目の引き数として LONG VARCHAR を使用すると、関数解決が失敗します (SQLSTATE 42884)。

式

連結演算子を使用する式

連結オペランドのいずれかが拡張すると、日本語または繁体字中国語 EUC データベース・サーバーが含まれている環境では、連結オペランドのデータ・タイプと長さが変更されます。たとえば、ホスト変数の値の長さを 2 倍にする EUC サーバーを使用する場合、以下の例を考慮してください。

```
CHAR200 CONCAT :char50
```

列 *CHAR200* は CHAR(200) タイプです。ホスト変数 *char50* は CHAR(50) として定義されています。この連結演算の結果タイプは通常 CHAR(250) になります。しかし、eucJP または eucTW データベース・サーバーでは、ストリングが拡張して長さが 2 倍になると仮定しています。このため、*char50* は CHAR(100) として処理され、結果データ・タイプは VARCHAR(300) です。結果データ・タイプが VARCHAR だとしても、それには後続ブランクを含む 300 バイトのデータが常に含まれていることに注意してください。余分の後続ブランクが必要ない場合、ホスト変数を CHAR(50) ではなく VARCHAR(50) と定義してください。

述部

LIKE 述部

EUC データベースに混合文字ストリングを含む LIKE 述部の場合、

- 1 バイトの下線は、1 バイトの文字を表します。
- 1 バイトのパーセント記号は、ゼロまたはそれ以上の文字 (1 バイトまたはマルチバイト文字) のストリングを表します。
- 2 バイトの下線は、マルチバイト文字を表します。
- 2 バイトのパーセント記号は、ゼロまたはそれ以上の文字 (1 バイトまたはマルチバイト文字) のストリングを表します。

エスケープ文字は、1 バイト文字または 2 バイト文字でなければなりません。

下線文字を使用すると、 LIKE 操作のコード・ページによって異なる結果が生じる場合があることに注意してください。たとえば、カタカナ文字は、日本語 EUC ではマルチバイト文字 (CS2) ですが、日本語 DBCS コード・ページでは 1 バイト文字です。 *pattern-expression* の 1 バイトの下線で照会すると、カタカナ文字のオカレンスが日本語 DBCS サーバーから下線の位置に戻されます。しかし、日本語 EUC サーバーの同じ表の同じ行は戻されません。カタカナ文字は 2 バイトの下線にしか一致しないからです。

EUC データベースに漢字ストリングを含む LIKE 述部の場合、

- 下線とパーセントに使用される文字は、それぞれ下線文字とパーセント文字に対応していなければなりません。
- 下線は UCS-2 文字を表します。
- パーセントはゼロまたはそれ以上の UCS-2 文字のストリングを表します。

関数

LENGTH

この関数の処理は、EUC 環境の混合文字ストリングでは変わりません。戻される値は、引き数のコード・ページでのストリングの長さです。この関数を使用して値の長さを決める場合、その長さを使用する方法を十分考慮する必要があります。これは混合ストリング定数を使用する場合は特に当てはまります。長さが文字単位ではなくてバイト単位で与えられているからです。たとえば、LENGTH 関数によって戻された DBCS データベースの混合ストリング列の長さは、 eucJP または eucTW クライアントのその列の検索値の長さよりも短いことがあります。それは、いくつかの DBCS 文字がマルチバイト eucJP または eucTW 文字に変換されるためです。

SUBSTR

SUBSTR 関数は、混合文字ストリング上でバイト単位で演算を行います。そのため、結果ストリングの最初または終わりにマルチバイト文字のフラグメントが含まれていることがあります。文字のフラグメントの検出または処理は行われません。

TRANSLATE

TRANSLATE 関数は、マルチバイト文字を含む混合文字ストリングをサポートします。 *to-string-exp* と *from-string-exp* の対応する文字は、同じ数のバイトを持っていなければならず、マルチバイト文字の一部で終了することはできません。

char-string-exp が文字ストリングである場合、 *pad-char-exp* の結果は 1 バイト文字でなければなりません。 TRANSLATE は *char-string-exp* のコード・ページで実行されるため、 *pad-char-exp* はマルチバイト文字から 1 バイト文字に変換されることがあります。

マルチバイト文字の一部で終了する *char-string-exp* は、変換されたバイトを持ちません。

VARGRAPHIC

日本語 EUC または中国語 (繁体字) EUC のコード・ページの文字ストリング・オペランドに対する VARGRAPHIC 関数は、 UCS-2 コード・ページの漢字ストリングを戻します。

- 1 バイト文字は、まず最初に、帰属する (eucJP または eucTW) コード・セットの該当する 2 バイト文字に変換されます。その後、該当する UCS-2 表

示に変換されます。 2 バイト表示がない場合、文字は、UCS-2 表示に変換される前に、そのコード・セットに定義された 2 バイトの代用文字に変換されます。

- カタカナ (eucJP CS2) である eucJP の文字は実際には、あるエンコード・スキームでは 1 バイト文字です。このように、それらの文字は UCS-2 に変換される前に、 eucJP の該当する 2 バイト文字または 2 バイトの代用文字に変換されます。
- マルチバイト文字は、UCS-2 表示に変換されます。

ステートメント

CONNECT

クライアントまたはサーバーに日本語または繁体字中国語 EUC コード・ページが組み込まれている環境でアプリケーションがデータを処理する可能性がある場合に、 CONNECT ステートメントを正常に処理すると、重要な SQLCA の情報が戻されます。 *SQLERRD(1)* フィールドは、アプリケーションのコード・ページからデータベース・コード・ページに変換するときに混合文字ストリングを最大拡張します。 *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・ページに変換するときに混合文字ストリングを最大拡張します。拡張が生じる場合の値は正で、短縮が生じる場合の値は負になります。値が負の場合、短縮できず、変換後にストリングの全長が必要になる最悪の状況に備え、値は常に -1 にしてあります。正の値は 2 と同じ大きさになることがあります。これは、変換後に文字ストリングのストリング長を 2 倍にする必要が生じるという最悪の状況に備えたものです。

アプリケーション・サーバーおよびアプリケーション・クライアントのコード・ページは、 SQLCA の SQLERRMC フィールドでも使用できます。

PREPARE

タイプなしパラメーター・マーカー (1045ページの表28 で説明されている) 用に決められたデータ・タイプは、日本語または繁体字中国語 EUC を含む環境では変更されません。その結果、ある場合、タイプ付きパラメーター・マーカーを使用して、 eucJP または eucTW の混合文字ストリングの十分な長さを提供することが必要になることがあります。たとえば、CHAR(10) 列への挿入を考慮してください。以下のステートメントを準備すると、

```
INSERT INTO T1 (CH10) VALUES (?)
```

その結果パラメーター・マーカーのデータ・タイプは CHAR(10) になります。クライアントが eucJP または eucTW であった場合、挿入するストリングを表

示するのに 10 バイトより大きいバイト数が必要になることがありますが、データベースの DBCS コード・ページでは 10 バイトを超えることはありません。この場合、準備するステートメントには、 10 より大きい長さのタイプ付きパラメーター・マーカーが含まれている必要があります。このため、以下のステートメントを準備すると、

```
INSERT INTO T1 (CH10) VALUES (CAST(? AS VARCHAR(20))
```

パラメーター・マーカーのデータ・タイプは VARCHAR(20) になります。

付録P. DATALINK での BNF 指定

DATALINK 値はカプセル化された値で、データベース以外の場所に保管されているファイルへのデータベースからの論理参照を含んでいます。

このカプセル化された値のデータ位置属性は、URL の形式でのファイルへの論理参照です。この属性の値は、以下の BNF[119] によって指定される、URL の構文に準拠しています。これは、RFC 1738 : Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M. McCahill, December 1994 に基づいています。

BNF 指定では、以下の規則が使用されます。

- 代替を指定するのに "|" を使用する
- 任意選択または繰り返されるエレメントの周りに大括弧 [ ] を使用する
- リテラルは "" で囲む
- エレメントの前に [*n*]* を付ければ、その直後のエレメントを *n* 回以上反復させることができる。 *n* が指定されていない場合、デフォルトは 0 です。

DATALINK での BNF 指定を以下に示します。

URL

```
url          =    httpurl | fileurl | uncurl | dfsurl | emptyurl
```

HTTP

```
httpurl      =    "http://" hostport [ "/" hpath ]
hpath        =    hsegment *[ "/" hsegment ]
hsegment     =    *[ uchar | ";" | ":" | "@" | "&" | "=" ]
```

RFC1738 の元の BNF での検索エレメントは除去されている点に注意してください。これは、そのエレメントがファイル参照の本質的な部分ではなく、 DATALINK のコンテキストでは意味をなさないためです。

FILE

```
fileurl      =    "file://" host "/" fpath
fpath        =    fsegment *[ "/" fsegment ]
fsegment     =    *[ uchar | "?" | ":" | "@" | "&" | "=" ]
```

119. BNF は ″バッカス正規形式″ (特定の言語の構文を記述するための正式な表記) です。

RFC1738 とは異なり、host が任意選択ではなく、 "localhost" ストリングが特別な意味を持たない点に注意してください。これにより、"localhost" の解釈で、クライアント / サーバーと EEE 構成で混乱が生じるのを避けることができます。

UNC

```
uncurl       =    "unc:¥¥" hostname "¥" sharename "¥" uncpath
sharename    =    *uchar
uncpath      =    fsegment *[ "¥" fsegment ]
```

NT では、一般的に使用されている UNC 命名規則をサポートしています。これは RFC1738 での標準方式ではありません。

DFS

```
dfsurl        =   "dfs://.../" cellname "/" fpath
cellname      =   hostname
```

DFS 命名方式をサポートしています。これは RFC1738 での標準方式ではありません。

EMPTYURL

```
emptyurl     =    ""
hostport     =    host [ ":" port ]
host         =    hostname | hostnumber
hostname     =    *[ domainlabel "." ] toplabel
domainlabel  =    alphadigit | alphadigit *[ alphadigit | "-" ] alphadigit
toplabel     =    alpha | alpha *[ alphadigit | "-" ] alphadigit
alphadigit   =    alpha | digit
hostnumber   =    digits "." digits "." digits "." digits
port         =    digits
```

DATALINK 値では、空の (長さゼロ) URL もサポートされています。これらは、調整例外が報告されてヌル不可の DATALINK 列が呼び出される場合に、 DATALINK 列を更新するのに便利です。長さゼロの URL を使用すれば、その列が更新され、リンクが解除されます。

各種の定義

```
lowalpha      =   "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                  "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                  "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                  "y" | "z"
hialpha       =   "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                  "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
                  "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
                  "Y" | "Z"
alpha         =   lowalpha | hialpha
digit         =   "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                  "8" | "9"
safe          =   "$" | "-" | "_" | "." | "+"
```

```
extra         =   "!" | "*" | "'" | "(" | ")" | ","
hex           =   digit | "A" | "B" | "C" | "D" | "E" | "F" |
                  "a" | "b" | "c" | "d" | "e" | "f"
escape        =   "%" hex hex
unreserved    =   alpha | digit | safe | extra
uchar         =   unreserved | escape
digits        =   1*digit
```

前後のブランク文字は、構文解析中に DB2 によって切り詰められます。また、スキーム名 ('HTTP'、'FILE'、'UNC'、'DFS') および host では大文字小文字が区別されず、いつでも大文字でデータベースに保管されます。

付録Q. 用語集

A

異常終了 (abend). 「タスクの異常終了 (abnormal end of task)」を参照。

異常終了理由コード (abend reason code). DB2 UDB (OS/390 版) に起こった問題を一意的に示す 4 バイトの 16 進コード。

タスクの異常終了 (異常終了) (abnormal end of task, abend). DB2 UDB (OS/390 版) では、実行中にエラー状態が発生したが、回復機能によって解決できないためにタスク、ジョブ、またはサブシステムを終了させること。

異常終了 (abnormal termination). (1) システム障害または操作員の処置によって、ジョブが完了せずに終了すること。 (2) DB2 では、 trap または segv などの、プログラム制御を受けない終了のこと。

絶対パス (absolute path). オブジェクトの全パス名。絶対パス名は (スラッシュ文字 (/) または円記号 (¥) で識別される) 最上位のレベル、つまり「ルート」ディレクトリーから始まる。

アクセス関数 (access function). 列に保管されるテキストのデータ・タイプを、テキスト・エクステンダーで処理できるタイプに変換する、ユーザー提供の関数。

アクセス方式サービス (access method services). VSAM キー順データ・セットを定義し、複製する機能。

アクセス・パス (access path). (1) 特定の表からデータを取り出すために最適化プログラムで選択される方式。たとえば、アクセス・パスでは索引、順次走査、またはこの 2 つの組み合わせが使用されることがある。 (2) SQL ステートメントに指定されたデータを検索するために使用されるパス。アクセス・パスは索引付きまたは順次になる場合もある。

アクセス・プラン (access plan). 特定の SQL ステートメントを評価するために最適化プログラムによって選択されたアクセス・パスの集まり。アクセス・プランは、実行プランを解決するための操作の順序、実施メソッド (JOIN など)、およびステートメント内で参照される各表のアクセス・パスを指定する。

会計ストリング (accounting string). DB2 コネクトから DRDA® サーバーに送られるユーザー定義の会計情報。この情報は、次のロケーションのいずれかで指定することができる。

- SQLESACT API または DB2ACCOUNT 環境変数を使用するクライアント・ワークステーション
- DFT_ACCOUNT_STR データベース・マネージャー構成パラメーターを使用する DB2 コネクト ワークステーション

アクティブ・ログ (active log). (1) DB2 UDB では、破損回復およびロールバックのために必要とされる 1 次および 2 次ログ・ファイル。「アーカイブ・ログ (archive log)」と対比。 (2) ログ・レコードを生成する際に書き込む DB2 UDB (OS/390 版) ログの部分。アクティブ・ログには常に最新のログ・レコードが入っているのに対し、アーカイブ・ログには古くなって、アクティブ・ログには適さなくなったレコードが保留されている。

隣接ノード (adjacent node). 他のノードを接続していない最低限 1 つのパスによって接続された 2 つのノード。

管理権限 (administrative authority). 一連のオブジェクトに対するユーザー特権を与えるための権限レベル。たとえば、DBADM 権限はデータベース内のすべてのオブジェクトに対する特権を、 SYSADM 権限はシステム内のすべてのオブジェクトに対する特権を与える。

管理サポート表 (administrative support table). DB2 エクステンダーがイメージ、オーディオ、ビデオ・オブジェクトでユーザー要求を処理する際に使用する表。管理サポート表には、エクステンダーで使用可能になっているユーザー表と列を識別するものがある。他の管理サポート表には、使用可能な列のオブジェクトに関する属性情報が含まれている。「メタデータ表 (metadata table)」とも呼ばれる。

ADSM. 「*Tivoli Storage Manager*」を参照。

拡張分散ネットワーク機能 (APPN) (Advanced Peer-to-Peer Networking, APPN). 分散ネットワーク制御、ネットワーク資源の動的定義、および資源の自動登録およびディレクトリー検索を特長とする SNA 拡張機能。

拡張分散ネットワーク機能 (APPN) ネットワーク (Advanced Peer-to-Peer Networking (APPN) network). 相互接続ネットワーク・ノードとそのクライアント・エンド・ノードの集合。

拡張プログラム間通信機能 (APPC) (Advanced program-to-program communication, APPC). LU 6.2 体系と製品におけるその各種実装を特徴とする一般機能。

事後イメージ (after-image). DB2 複製では、変更データ表、またはデータベース・ログかジャーナルに記録されたソース表要素の更新後の内容。「*事前イメージ* (before-image)」と対比。

エージェント (agent). (1) 特定のクライアント・アプリケーションから出されたすべての DB2 要求を実行する別のプロセスまたはスレッド。 (2) DB2 UDB (OS/390 版) では、DB2 UDB (OS/390 版) 作業単位にかかわるすべてのプロセスに関連する構造体。「*接続エージェント* (allied agent)」は、一般に「*接続スレッド* (allied thread)」と同義である。「システム・エージェント (System agents)」は、事前取り出し処理、据え置き書き出し、およびサービス・タスクなどの接続エージェントとは無関係に処理する作業単位。

エージェント・サイト (agent site). データウェアハウスセンターでは、単一のネットワーク・ホスト名で定義されるロケーションで、エージェント・アプリケーションがインストールされる。

総計関数 (aggregate function). 「*列関数* (column function)」の同義語。

アラート (alert). パフォーマンス変数が警告またはアラームのしきい値より大きくなったりそれより小さくなったりしたときに生成されるビープ音または警告などのアクション。

別名 (alias). 表、視点、データベース、またはニックネームを識別するのに使われる代替名。同一またはリモート DB2 サブシステムに置かれている表または視点を参照するために、 SQL ステートメントで使用できる。

別名連鎖 (alias chain). 反復しない方法で順次に相互に参照する一連の表別名。

接続アドレス空間 (allied address space). DB2 UDB (OS/390 版) では、DB2 UDB (OS/390 版) の外部にあるが、DB2 UDB (OS/390 版) に接続されている記憶域。 DB2 UDB (OS/390 版) サービスを要求できる。

接続スレッド (allied thread). ローカル DB2 UDB (OS/390 版) サブシステム側にあって、リモート DB2 UDB (OS/390 版) サブシステム側にあるデータをアクセスできるスレッド。

割り当てカーソル (allocated cursor). DB2 UDB (OS/390 版) では、SQL ステートメント ALLOCATE CURSOR を使用してストアード・プロシージャーの結果セットに定義されるカーソル。

検査済み (already verified). 会話を割り当てる際、ユーザーの検査済み許可 ID を DB2 UDB (OS/390 版) によって提供されるようにする LU 6.2 セキュリティー・オプション。パートナー・サブシステムはユーザーの妥当性検査をしない。

未確定カーソル (ambiguous cursor). (1) 定義または文脈からは、更新可能とも読み取り専用とも判別できないカーソル。 (2) DB2 UDB (OS/390 版) では、FOR FETCH ONLY 文節または FOR UPDATE OF 文節によって定義されておらず、読み取り専用結果表で定義されておらず、 SQL UPDATE または DELETE ステートメントの WHERE CURRENT 文節のターゲットでなく、また PREPARE または EXECUTE IMMEDIATE SQL ステートメントのいずれかを含むプランまたはパッケージ内にあるデータベース・カーソル。

APF. 「*許可プログラム機能* (authorized program facility)」を参照。

API. 「*アプリケーション・プログラミング・インターフェース* (application programming interface)」を参照。

APPC. 「*拡張プログラム間通信* (Advanced program-to-program communication)」を参照。

APPL. SNA LU 6.2 プロトコルを使用するアプリケーション・プログラムとして DB2 UDB (OS/390 版) を VTAM に定義するために使用する、 VTAM® ネットワーク定義ステートメント。

アプリケーション (application). ある仕事を実行する 1 つのプログラムまたはプログラムの集まり。たとえば、給与計算アプリケーション。

アプリケーション ID (application ID). ネットワーク全体でアプリケーションを固有識別するストリング。 ID は、アプリケーションがデータベースに接続すると生成される。この ID はクライアントとサーバーで認識され、アプリケーションの 2 つの部分を相関させるのに使うことができる。

アプリケーション・プラン (application plan). バインド処理中に作成される制御構造。 DB2 UDB (OS/390 版) がステートメント実行中に検出した SQL ステートメントを処理するために使用する。

アプリケーション・プロセス (application process). 資源およびロックを割り当てる単位。アプリケーション・プロセスには、1 つ以上のプログラムの実行が関与する。

アプリケーション・プログラミング・インターフェース (API) (application programming interface, API). (1) オペレーティング・システムに、または個別に発注可能なライセンス・プログラムに用意されている機能インターフェース。高水準言語で作成されたアプリケーション・プログラムは、オペレーティング・システムまたはライセンス・プログラムの特定のデータまたは機能を使うことができる。 (2) DB2 では、インターフェース内の機能 (たとえば、エラー・メッセージ獲得 API)。

アプリケーション・リクエスター (application requester). アプリケーション・プロセスからデータベース要求を受け入れてから、それをアプリケーション・サーバーに渡す機能。

アプリケーション・サーバー (application server). アプリケーション・プロセスの接続先のローカルまたはリモート・データベース・マネージャー。

変更適用プログラム (Apply program). DB2 複製では、ソース対ターゲットの適用規則に応じてターゲット表を最新表示したり更新したりするのに使われるプログラム。「*収集プログラム* (Capture program)」および「*収集トリガー* (Capture trigger)」と対比。

変更適用修飾子 (Apply qualifier). DB2 複製では、変更適用プログラムの各インスタンスごとに固有のサブスクリプション定義を識別する文字ストリング。

APPN. 「*拡張対等通信ネットワーク機能* (Advanced Peer-to-Peer Networking)」を参照。

アーカイブ・ログ (archive log). (1) クローズされ、通常の処理には必要なくなったログ・ファイルの集まり。それらのファイルは、ロールフォワード回復で使うためとっておかれる。「*アクティブ・ログ* (active log)」と対比。 (2) アクティブ・ログからコピーされたログ・レコードが入っている DB2 UDB (OS/390 版) ログの部分。

引き数 (argument). 実行時に関数またはプロシージャーから渡されるかまたは戻される値。

非同期 (asynchronous). 規定の時刻との関連性がないこと。プログラム命令の処理に関して予期したり予想したりされないこと。「*同期* (synchronous)」と対比。

非同期バッチ更新 (asynchronous batched update). ソースに対するすべての変更内容を記録し、それを指定の時間間隔で既存のターゲット・データに対して適用するプロセス。「*非同期連続更新* (asynchronous continuous update)」と対比。

非同期連続更新 (asynchronous continuous update). ソースに対するすべての変更内容が基礎表においてコミットされた後、それを記録し、指定の時間間隔で既存の宛先に対して適用するプロセス。「*非同期バッチ更新* (asynchronous batched update)」と対比。

接続する (attach). DB2 では、インスタンス・レベルでオブジェクトにリモート・アクセスすること。

接続機能 (attachment facility). DB2 UDB (OS/390 版) と TSO、IMS™、 CICS、またはバッチ・アドレス空間の間のインターフェース。 接続機能により、アプリケーション・プログラムは DB2 UDB (OS/390 版) にアクセスすることができる。

属性 (attribute). SQL データベース設計では、エンティティー (実体) がもつ特性。たとえば、従業員の電話番号はその従業員の属性の 1 つ。

権限 (authority). 「*管理権限* (administrative authority)」を参照。

許可 ID (authorization ID). (1) 一連の特権を指定するための、ステートメント内の文字ストリング。データベース・マネージャーによって権限検査に使われ、表、視点、および索引などのオブジェクト名の暗黙修飾子としても使われる。 (2) DB2 UDB (OS/390 版) と接続するときに検証されるストリングで、一連の特権が許可されているもの。個人、組織グループ、または機能を表すことができるが、 DB2 UDB (OS/390 版) ではこの表現内容は判別されない。

許可プログラム機能 (APF) (authorized program facility, APF). DB2 UDB (OS/390 版) では、制限された機能を使用するために許可されるプログラムの識別を認める機能。

自動確定 (autocommit). 各 SQL ステートメント実行後、現在の作業単位を自動的にコミットすること。

自動再バインド (automatic rebind). (1) 手動で **BIND** コマンドが入力されなくても、またはバインド・ファイルがなくても、無効になったパッケージを自動的に再バインドする機能。 (2) DB2 UDB (OS/390 版) では、アプリケーション・プロセスが実行を開始し、そのプロセスが必要とするバインド済みアプリケーション・プランまたはパッケージが有効でないとき、 SQL ステートメントが自動的にバインドされる (ユーザーが BIND コマンドを出さずに実行される) プロセス。「バインド (bind)」も参照。

補助索引 (auxiliary index). DB2 UDB (OS/390 版) では、各索引項目が LOB を参照する補助表の索引。

補助表 (auxiliary table). DB2 UDB (OS/390 版) では、列が定義されている表の外部にその列を保管する表。「*基礎表* (base table)」と対比。

B

バックアップ保留 (backup pending). データベースまたは表スペースの状態。データベースまたは表スペースがバックアップされるまで操作は行われない。

逆方向ログ回復 (backward log recovery). 再始動処理の第 4 および最終フェーズ。このフェーズで、DB2 UDB (OS/390 版) は、ログを逆方向に走査して、打ち切られたすべての変更の UNDO ログ・レコードを適用する。

基礎集約表 (base aggregate table). DB2 複製では、ソース表またはある時間間隔における時刻表から集計されたデータの入った一種のターゲット表。

基礎表 (base table). (1) CREATE TABLE ステートメントで作成された表。この表には、データベースに物理的に記憶された記述とデータが入っている。「*視点* (view)」と対比。 (2) DB2 UDB (OS/390 版) では、(a) SQL CREATE TABLE ステートメントで作成され、持続データを保持する表。「*結果表* (result table)」および「*一時表* (temporary table)」と対比。 (b) LOB 列定義を含む表。実際の LOB 列データは、基礎表と一緒に保管されない。基礎表は、各行ごとに行識別子、および LOB 列ごとに標識表を含んでいる。「*補助表* (auxiliary table)」と対比。

基礎表スペース (base table space). DB2 UDB (OS/390 版) では、基礎表を含む表スペース。

基本的会話 (basic conversation). APPC 基本的会話 API を使用する 2 つのトランザクション・プログラムの間の LU 6.2 会話。「マップ式会話 (mapped conversation)」と対比。

基本述部 (basic predicate). 2 つの値を比較する述部。

基本順次アクセス方式 (BSAM) (basic sequential access method, BSAM). DB2 UDB (OS/390 版) が順次アクセス装置または直接アクセス装置のいずれかを用いて、データ・ブロックを連続して格納または検索するアクセス方式。

事前イメージ (before-image). DB2 複製では、最新表示される前の、変更データ表またはデータベース・ログかジャーナルに記録されているソース表列の内容。「*事後イメージ* (after-image)」と対比。

BEFORE トリガー (before trigger). DB2 UDB (OS/390 版) では、トリガー活動化時間を BEFORE と指定して定義したトリガー。

2 進整数 (binary integer). 基本データ・タイプの 1 つ。これは、さらに短精度整数と長精度整数に分類することができる。

2 進ラージ・オブジェクト (BLOB) (binary large object, BLOB). バイト・シーケンス。そのシーケンスのサイズは、0 バイトから 2 ギガバイトまでの範囲である。 このストリングには、関連したコード・ページおよび文字セットがない。 イメージ、音声、およびビデオ・オブジェクトは BLOB に記憶される。「*文字ラージ・オブジェクト (CLOB)* (character large object, CLOB)」と対比。

バイナリー・ストリング (binary string). DB2 UDB (OS/390 版) では、CCSID と関連付けられていない一連のバイト。たとえば、BLOB データ・タイプはバイナリー・ストリングである。

バインド (bind). (1) SQL では、SQL プリコンパイラーからの出力を、「アクセス・プラン」という名前の使用可能な構造体に変換するプロセス。このプロセスの間、データへのアクセス・パスが選択され、許可検査が実行される。 (2) DB2 UDB (OS/390 版) では、DBMS プリコンパイラーからの出力を、「パッケージ (package)」または「アプリケーション・プラン (application plan)」と呼ばれる使用可能な制御構造に変換するためのプロセス。処理の間、データへのアクセス・パスが選択され、許可検査が実行される。「*自動再バインド* (automatic rebind)」、「*動的バインド* (dynamic bind)」、「*追加バインド* (incremental bind)」、「*静的バインド* (static bind)」も参照。

バインダリー・オブジェクト名 (bindery object name). NetWare ファイル・サーバーのバインダリー・オブジェクトの名前の入った 48 バイトの文字ストリング。データベース・マネージャー構成フィールドの objectname は、固有の DB2 サーバー・インスタンスを表し、 NetWare ファイル・サーバーのバインダリーにオブジェクトとして記憶される。

バインド・ファイル (bind file). BINDFILE オプションと一緒に **BIND** コマンドまたは API が使われたときにプリコンパイラーが作成するファイル。このファイルには、アプリケーション・プログラム内のすべての SQL ステートメントについての情報が含まれる。

ビット・データ (bit data). 文字型 CHAR または VARCHAR を持つデータで、コード化文字セットと関連性がないので、変換されることはない。

BLOB. 「*2 進ラージ・オブジェクト* (binary large object)」を参照。

ブロック (block). 1 単位として記録または伝送されるデータ・エレメントのストリング。

ブロック化 (blocking). アプリケーションをバインドするときに指定するオプション。このオプションを指定すると、通信サブシステムが複数行の情報をキャッシュできるようになるため、 FETCH ステートメントで各要求を 1 行ずつネットワークに伝送する必要がなくなる。「*データ・ブロック化* (data blocking)」と対比。

ブートストラップ・データ・セット (BSDS) (bootstrap data set, BSDS). すべてのアクティブ・ログ・データ・セットおよびアーカイブ・ログ・データ・セットに関する RBA 範囲指定とともに、 DB2 UDB (OS/390 版) の名前と状況情報が入っている VSAM データ・セット。さらに、DB2 UDB (OS/390 版) ディレクトリーとカタログについてのパスワード、ならびに条件付き再始動およびチェックポイント・レコードのリストが含まれる。

同報通信結合 (broadcast join). 表のすべての区画をすべてのノードに送るための結合。

ブラウザー (browser). コンピューターのモニターにテキストを表示できるようにするテキスト・エクステンダー機能。

BSAM. 「*基本順次アクセス方式* (basic sequential access method)」を参照。

BSDS. 「*ブートストラップ・データ・セット* (bootstrap data set)」を参照。

バッファー・プール (buffer pool). DB2 UDB (OS/390 版) では、1 つまたは複数の表スペースあるいは索引用のバッファー要件を満たすために予約された主記憶域。

組み込み関数 (built-in function). DB2 から提供される SQL 関数。 SYSIBM スキーマ内に現れる。「*ユーザー定義関数* (user-defined function)」と対比。

ビジネス・メタデータ (business metadata). 情報資産をビジネス用語によって記述したデータ。ビジネス・メタデータは情報カタログ内に保管され、ユーザーは必要な情報を見つけて理解するためにこれにアクセスする。たとえば、プログラムのビジネス・メタデータには、そのプログラムの機能およびプログラムが使用する表についての記述が含まれる。「テクニカル・メタデータ (technical metadata)」と対比。

ビジネス名 (business name). データウェアハウスセンターで、ステップを参照する名前。各ステップには、ビジネス名およびそのステップに関連した DB2 表名がある。通常、ビジネス名はウェアハウスのユーザーによって使用される。 DB2 表名は SQL ステートメントで使用される。

バイト逆転 (byte reversal). 最下位のバイトから先に数値データを記憶する技法。

C

キャッシュ (cache). 頻繁にアクセスされる命令とデータの入ったバッファー。アクセス時間を短縮するのに使われる。

キャッシュ・マネージャー (Cache Manager). Net.Data® で、 1 つのワークステーションのキャッシュを管理するプログラム。キャッシュ・マネージャーは複数のキャッシュを管理できる。

キャッシュ構造 (cache structure). 並列シスプレックス®のすべてのメンバーで使用できるデータを保管する結合機能構造。 DB2 UDB (OS/390 版) データ共用グループは、キャッシュ構造をグループ・バッファー・プールとして使用する。

キャッシング (caching). Web サーバーへの要求から得られる結果のうち、頻繁に使用されるものを即時に検索するためにローカルに保管するプロセス。結果は情報を更新する時点まで保管される。

CAF. 「*呼び出し接続機能* (call attachment facility)」を参照。

呼び出し接続機能 (CAF) (call attachment facility, CAF). TSO または MVS™ バッチで実行されるアプリケーション・プログラムのための DB2 UDB (OS/390 版) 接続機能。 CAF は DSN コマンド処理プログラムに代わるもので、実行環境の制御を強化することができる。

コール・レベル・インターフェース (CLI) (call level interface, CLI). データベース・アクセスのために呼び出せる API。組み込み SQL API の代わりに使われる。組み込み SQL と違って、CLI はユーザーによる再コンパイルやバインドを必要としない。その代わりに、実行時に SQL ステートメントと関連サービスを処理するための一連の標準機能を提供する。

収集プログラム (Capture program). DB2 複製では、 DB2 ソース表に加えられた変更に関するデータを取り込むために、データベース・ログまたはジャーナルの記録を読み取るプログラム。「*変更適用プログラム* (Apply program)」および「*収集トリガー* (Capture trigger)」と対比。

収集トリガー (Capture trigger). DB2 複製では、IBM 以外のソース表に対して実行される削除、更新、および挿入操作を収集するためのメカニズム。「*収集プログラム* (Capture program)」および「*変更適用プログラム* (Apply program)」と対比。

カーディナリティー (cardinality). データベース表内の行数。

カスケード (cascade). データウェアハウスセンターで、一連のイベントを実行すること。 1 つのステップが別のステップにカスケードすると、 2 つのステップは連続して、または同時に実行される。あるステップをプログラムにカスケードすることもできる。この場合、ステップの実行完了後にプログラムが実行される。

連鎖削除 (cascade delete). DB2 UDB (OS/390 版) が削除された親行の下層行をすべて削除するときに行う参照制約の実行。

カスケード拒否 (cascade rejection). DB2 複製では、対立が検出され、しかもそれ自身が拒否されたトランザクションと関連しているために複製トランザクションを拒否するプロセス。

CASE 式 (CASE expression). DB2 UDB (OS/390 版) では、1 つまたは複数の条件の評価に基づいて式を選択できるようにする式。

cast 関数 (cast function). あるデータ・タイプのインスタンス (起点) を別のデータ・タイプのインスタンス (宛先) に変換するのに使われる関数。一般に、cast 関数には宛先データ・タイプの名前が付く。この関数は、起点のデータ・タイプと同じタイプの単一の引き数をもつ。その戻り型は宛先データ・タイプになる。

カタログ (catalog). データベース・マネージャーが保守する表および視点の集合。それらの表と視点には、表、視点、および索引の記述などのデータベースの情報が入っている。

カタログ・ノード (catalog node). カタログ表が常駐するノード。カタログ・ノードは、データベースごとに異なったノードでもかまわない。

カタログ表 (catalog table). DB2 UDB (OS/390 版) カタログ内の表。

カタログ視点 (catalog view). 管理目的のためにテキスト・エクステンダーが作成するシステム表の視点。カタログ視点には、テキスト・エクステンダーが使用できるようにする表と列に関する情報が含まれる。

CCD 表 (CCD table). 「*整合した変更データ表* (consistent-change-data table)」を参照。

CCSID. 「*コード化文字セット識別子* (coded character set identifier)」を参照。

CDB. 「*通信データベース* (communications database)」を参照。

CDRA. 「*文字データ表示体系* (Character Data Representation Architecture)」を参照。

CD 表 (CD table). 「*変更データ表* (Change data table)」を参照。

CEC. 中央電子複合システム (Central electronic complex)。「*中央処理装置複合システム* (central processor complex)」を参照。

中央処理装置複合システム (CPC) (central processor complex, CPC). (ES/3090 のような) ハードウェアの物理的集合で、主記憶域、1 つまたはそれ以上の中央処理装置、タイマー、およびチャネルで構成される。

CFRM ポリシー (CFRM policy). DB2 UDB (OS/390 版) では、結合機能構造の割り振り規則に関する、MVS 管理者による宣言。

変更集約表 (change aggregate table). DB2 複製では、ソース表用に記録された変更内容に基づいたデータ集約が入っている一種のターゲット表。

変更データ (CD) 表 (change data (CD) table). 複製ソース表の変更データの入った、ソース・サーバーの複製制御表。

文字データ表示体系 (CDRA) (Character Data Representation Architecture, CDRA). ストリング・データの表現、処理、および交換を統一化するために使用される体系。

文字ラージ・オブジェクト (CLOB) (character large object, CLOB). 長さが 2 ギガバイト以下の文字シーケンス（1 バイトまたはマルチバイト、あるいはその両方）。大きなテキスト・オブジェクトの保管に使われる。文字ラージ・オブジェクト・ストリングともいう。「*2 進ラージ・オブジェクト (BLOB)* (binary large object の BLOB)」と比較。

文字ストリング (character string). バイトまたは文字のシーケンス。

文字ストリング区切り文字 (character string delimiter). インポートまたはエクスポートされる区切り付き ASCII ファイル内で文字ストリングを囲むのに使われる文字。「*区切り文字* (delimiter)」を参照。

CHECK 文節 (CHECK clause). SQL では、表検査制約を指定する SQL CREATE TABLE ステートメントと SQL ALTER TABLE ステートメントの拡張部分。

検査条件 (check condition). 制限された形式の検索条件。検査制約で使われる。

検査制約 (check constraint). 制約を定義された表の各行に偽でないという検査条件を指定する制約。

検査保全性 (check integrity). DB2 UDB (OS/390 版) では、表の中の各行がその表で定義された表検査制約に適合するときに現れる条件。検査保全性を保守するには、データを追加または変更する操作に表検査制約を実行させることが、DB2 UDB (OS/390 版) に必要になる。

検査保留中 (check pending). 表がとることのある状態の 1 つ。この状態のとき、表に対して実行できる活動は限定され、表が更新されても制約は調べられない。

チェックポイント (checkpoint). DB2 UDB (OS/390 版) がログに内部状況情報を記録する点。サブシステムが異常終了する場合に、回復処理がこの情報を使用する。

CI. 「*制御インターバル* (control interval)」を参照。

CICS. 重要なビジネス・アプリケーションのトランザクションや処理のサービスおよび管理機能をオンラインで行うことを可能にする IBM® ライセンス・プログラム。 DB2 UDB (OS/390 版) 関連情報では、この用語は以下の製品を指します。

CICS Transaction Server for OS/390®: Customer Information Control Center Transaction Server for OS/390

CICS/ESA: 顧客情報管理システム / エンタープライズ・システム体系 (Customer Information Control System/Enterprise Systems Architecture)

CICS/MVS: 顧客情報管理システム / 多重仮想記憶 (Customer Information Control System/Multiple Virtual Storage)

CICS 接続機能 (CICS attachment facility). MVS サブシステム・インターフェース (SSI) および記憶域連係を使用して、 CICS から DB2 UDB (OS/390 版) への要求を処理し、資源のコミットを調整する DB2 UDB (OS/390 版) の副構成要素。

CIDF. 「*制御インターバル定義フィールド* (control interval definition field)」を参照。

循環ログ (circular log). 活動中のデータベースが必要としなくなったログ記録が上書きされるデータベース・ログ。したがって、障害が生じた場合、失われたデータは順方向回復中には復元できない。「*回復可能ログ* (recoverable log)」と対比。

クレーム (claim). DB2 UDB (OS/390 版) では、あるオブジェクトがアクセスされていることを DBMS に通知すること。クレームが解放されるまでドレーンが発生しないようにする。それは通常コミット・ポイントである。「*ドレーン* (drain)」も参照。

クレーム・クラス (claim class). DB2 UDB (OS/390 版) では、特定タイプのオブジェクト・アクセスで、以下のいずれかの可能性がある。カーソル固定 (CS)、反復可能読み取り (RR)、書き込み。

クレーム・カウント (claim count). DB2 UDB (OS/390 版) で、1 つのオブジェクトをアクセスしているエージェント数のカウント。

サービス・クラス (class of service). DB2 UDB (OS/390 版) では、ネットワークを介する経路のリストのための VTAM 用語。この経路を使用する優先順序で配置される。

文節 (clause). DB2 UDB (OS/390 版) SQL では、SELECT 文節や WHERE 文節などのように、ステートメントの明確な一部分。

クレンジング (する) (cleanse). 運用システムから抽出されたデータを操作して、データウェアハウスで使用できるようにするプロセス。

CLI. 「*コール・レベル・インターフェース* (call level interface)」を参照。

クライアント (client). (1) データベース・サーバーと通信したりアクセスしたりするプログラム（またはそのプログラムが実行されるワークステーション）。 (2) 「*リクエスター* (requester)」を参照。

クライエット (cliette). Net.Data Live Connection で、 Web サーバーからの要求を処理する、長時間実行されるプロセス。接続マネージャーは、これらの要求を処理するためにクライエットをスケジュール化する。

CLIST. コマンド・リスト。 DB2 UDB (OS/390 版) が TSO タスクを実行するために使用する言語。

CLOB. 「*文字ラージ・オブジェクト* (character large object)」を参照。

CLP. 「*コマンド行プロセッサー (CLP)* (Command Line Processor, CLP)」を参照。

CLPA. 「*リンク・パック域の作成* (create link pack area)」を参照。

クラスター索引 (clustered index). 表に記憶されている行順序に厳密に対応するキー値順序をもった索引。その対応の程度は、最適化プログラムが使用する統計で測定される。

コード化文字セット (coded character set). 文字セットを確立し、かつその文字セットの文字とコード化表現との間で 1 対 1 対応の関係を確立する明確な規則の集まり。

コード化文字セット識別子 (CCSID) (coded character set identifier, CCSID). コード化スキーマ識別子、文字セット識別子、コード・ページ識別子、およびコード化された図形文字表示を固有に識別するその他の情報を含む数字。

コード・ページ (code page). コード・ポイントに割り当てた文字の集合。

コード・ポイント (code point). CDRA では、コード・ページで 1 つの文字を表す固有のビット・パターン。

コード・セット (code set). システムとその入出力装置との間にインターフェースを提供する文字セット用のコード化値。 ISO では、「コード・セット」を IBM 定義の用語である「コード・ページ」と同じ意味で使用する。

コールド・スタート (cold start). (1) 初期プログラム・ロード・プロシージャーを使用して、システムまたはプログラムを始動するプロセス。「ウォーム・スタート (warm start)」と対比。 (2) DB2 UDB (OS/390 版) がどのログ・レコードも処理しないで再始動するときのプロセス。

照合順序 (collating sequence). 索引付きデータを順序どおりにソート、連結、比較、 および処理できるように文字を並べる順序。

コレクション (collection). DB2 UDB (OS/390 版) では、同じ修飾子をもつパッケージのグループ。

照合結合 (collocated join). 以下の条件が合った、 2 つの表を結合した結果。

- 2 つの表が同じデータベース区画の中の単一区画ノード・グループにある。または 2 つの表が同じ区画化ノード・グループにあって同じ数の区画列がある (このとき、それらの区画列は区画互換であり、両方の表が同じ区分化関数を使用する)。
- 対応する区分化キー列のすべての対が equijoin 述部の一部となっている。

列分布値 (column distribution value). 特定の列の中の最も頻繁に現れる値または変位値を記述する統計。その値は、最善のアクセス・プランを判別するために最適化プログラムで使われる。

列関数 (column function). (1) 照会に使われる操作。いくつかの行の値に対して適用される。列関数には、SUM、AVG、MIN、MAX、COUNT、STDDEV、および VARIANCE が含まれる。「*総計関数* (aggregate function)」の同義語。 (2) DB2 UDB (OS/390 版) では、1 行または複数行の値の集まりから、その結果を導く SQL 操作。「*スカラー関数* (scalar function)」と対比。

"発信元" チェック ("come from" checking). パートナー LU から DB2 UDB (OS/390 版) への接続を認める許可 ID のリストを定義する、 LU 6.2 セキュリティー・オプション。

コマンド (command). DB2 UDB (OS/390 版) 操作員コマンドまたは DSN サブコマンド。 SQL ステートメントとは異なる。

コマンド行プロセッサー (CLP) (Command Line Processor, CLP). SQL ステートメントを入力するための文字ベースのインターフェース、あるいはデータベース・マネージャー・コマンド。

コマンド接頭部 (command prefix). DB2 UDB (OS/390 版) では、1 ～ 8 文字のコマンド ID。コマンド接頭部は、コマンドが OS/390 ではなくアプリケーションまたはサブシステムに属していることを見分けるのを助ける。

コマンド認識文字 (CRC) (command recognition character, CRC). MVS コンソール操作員または IMS サブシステム・ユーザーが特定 DB2 UDB (OS/390 版) サブシステムに DB2 コマンドを経路指定するときに使用できる文字。

コマンド・スコープ (command scope). DB2 UDB (OS/390 版) では、データ共用グループのコマンド操作の効力範囲。コマンドにメンバー*効力範囲* がある場合、コマンドは 1 人だけのメンバーからの情報を表示するか、またはそのメンバーがローカルに所有する非共用資源だけに影響を与える。コマンドにグループ*効力範囲* がある場合、コマンドはすべてのメンバーからの情報を表示し、すべてのメンバーがローカルに所有する非共用資源に影響を与え、共用可能資源に関する情報を表示し、または共用可能資源に影響を与える。

コミット (commit). ロックを解除することにより作業単位を終了し、その作業単位によって行われたデータベースの変更を他のプロセスが認識できるようにする操作。この操作により、データ変更が永続化される。

コミットメント制御 (commitment control). 資源に対する操作が作業単位に含まれる、 Net.Data が実行されているプロセス内での境界の設定。

コミット・ポイント (commit point). データに整合性があるとみなされる一時点。「*一貫性ポイント (point of consistency)*」の同義語。

コミットされたフェーズ (committed phase). DB2 UDB (OS/390 版) では、複数サイト更新プロセスの 2 番目フェーズであり、すべての参加プログラムが論理作業単位の効果をコミットするように要求するフェーズ。

共通索引表 (common-index table). テキスト列が共通のテキスト索引を共有する DB2 表。「*複数索引表 (multi-index table)*」を参照。

共通プログラミング・インターフェース・コミュニケーション (CPI-C) (Common Programming Interface Communications, CPI-C). プログラム間通信を必要とするアプリケーションのための API。SNA の LU 6.2 を使って一連のプログラム間サービスを作成する。

共通サービス域 (CSA) (common service area, CSA). OS/390 では、すべてのアドレス空間によってアドレス可能なデータ域が入っている共通域の一部分。

共通表式 (common table expression). WITH 文節の後に続く全選択内の任意の FROM 文節内の表名に指定できる名前（修飾 SQL 識別子）の付いた結果表を定義する式。

通信データベース (CDB) (communications database, CDB). リモート・データベース管理システムとの会話を確立する際に使用される DB2 UDB (OS/390 版) カタログにある一連の表。

比較演算子 (comparison operator). 比較式で使われる挿入演算子。比較演算子は、¬< (より小さくない)、<= (より小さいかまたは等しい)、 ¬= (等しくない)、= (等しい)、>= (より大きいかまたは等しい)、> (より大きい)、および ¬> (より大きくない)。

完全な (complete). 該当する各基本キー値につき 1 行ずつが表に入っていることを示す表属性。そのため、完全なソース表を使うと、ターゲット表を最新表示できる。

完全 CCD 表 (complete CCD table). ソース表または視点からのソース視点および述部を満たすすべての行を含む CCD 表。「**不完全 CCD 表** (noncomplete CCD table)」と対比。

複合キー (composite key). 同じ表のキー列が一定順序に並んだものの集合。

複合 SQL ステートメント (compound SQL statement). アプリケーション・サーバーに対する 1 回の呼び出しで実行される SQL ステートメント・ブロック。

圧縮辞書 (compression dictionary). DB2 UDB (OS/390 版) では、圧縮および解凍の処理を制御する辞書。この辞書は、表スペースまたは表スペース区分から作成される。

並行性 (concurrency). 複数の対話式ユーザーまたはアプリケーション・プロセスが同時に資源を共用すること。

圧縮された (condensed). データに加えられた変更の活動記録ではなく、現行データが表に入っていることを示す表属性。圧縮表には、表内の各基本キー値につき 1 つ以上の行が含まれることはない。そのため、圧縮表は、最新表示のための現行情報を提供するのに使うことができる。

圧縮 CCD 表 (condensed CCD table). DB2 複製では、行の最新の値のみを含む CCD 表。このような表は、リモート・ロケーションへのステージング変更、およびホットスポット更新の要約に役立つ。「**非圧縮 CCD 表** (noncondensed CCD table)」と対比。

条件付き再始動 (conditional restart). ユーザー定義の条件付き再始動制御レコード (CRCR) により指示される DB2 UDB (OS/390 版) 再始動。

対立検出 (conflict detection). 即時更新複製構成において、以下のものを表す。

- 制約エラーの検出プロセス。
- ソース表とターゲット表の中の同じ行が、同じ複製サイクルの中で更新されたことを検出するためのプロセス。対立が検出されると、その対立の原因になったトランザクションは拒否される。「*拡張対立検出* (enhanced conflict detection)」、「*標準対立検出* (standard conflict detection)」、および「*行レプリカ対立検出* (row-replica conflict detection)」も参照。

接続する (connect). DB2 では、データベース・レベルでオブジェクトにアクセスすること。

接続 (connection). (1) アプリケーション・プロセスとアプリケーション・サーバーとの相関関係。 (2) データ通信では、情報を伝えるために機能単位相互間に確立される相関関係。 (3) SNA では、2 つのパートナーの LU 間に通信パスが存在し、情報交換を可能にすること (たとえば、 2 つの DB2 UDB (OS/390 版) サブシステムを接続し、対話による通信を行えるようにすること)。

接続ハンドル (connection handle). CLI では、接続に関連した情報の入っているデータ・オブジェクト。この情報には、一般的な状況情報、トランザクション状況、および診断情報などが含まれる。

接続 ID (connection ID). DB2 UDB (OS/390 版) では、接続機能によって提供され、特定のアドレス空間接続に関連する識別子。

接続マネージャー (Connection Manager). Net.Data の、Live Connection サポートに必要な実行可能ファイル dtwcm。

整合性トークン (consistency token). DB2 UDB (OS/390 版) では、アプリケーションのバージョン ID を生成するために使用されるタイム・スタンプ。

整合した変更データ (CCD) 表 (consistent-change-data (CCD) table). DB2 複製では、データの監査またはステージングに使われる一種のターゲット表。「**完全 CCD 表** (complete CCD table)」、「**圧縮された CCD 表** (condensed CCD table)」、「*外部 CCD 表* (external CCD table)」、「*内部 CCD 表* (internal CCD table)」、「**不完全 CCD 表** (noncomplete CCD table)」、および「**圧縮されていない CCD 表** (noncondensed CCD table)」も参照。

定数 (constant). 不変の値を指定する言語要素。定数は、ストリング定数と数値定数に分けられる。「*変数* (variable)」と対比。

制約 (constraint). 表の中で挿入、削除、または更新できる値を限定するための規則。「*検査制約* (check constraint)」、「*参照制約* (referential constraint)」、および「*固有制約* (unique constraint)」を参照。

コンテナー (container). 「*表スペース・コンテナー* (table space container)」を参照。

競合 (contention). データベース・マネージャーでは、すでにロックされている行または表をトランザクションがロックしようとしている状態。

コントロール・センター (Control Center). データベース・オブジェクト (データベースや表など) とその相互関係を示すグラフィカル・インターフェース。コントロール・センターから、DBA ユーティリティー、Visual Explain、およびパフォーマンス・モニター・ツールによって提供されるタスクを実行できる。「**DataJoiner 複写管理 (DJRA) ツール** (DataJoiner Replication Administration (DJRA) tool)」と対比。

制御インターバル (CI) (control interval, CI). VSAM では、VSAM がレコードを保管し、分散空きスペースを作成する、直接アクセス記憶の固定長域。キー順データ・セットまたはファイルにおいて、順序セット索引レコードの項目によって指される一連のレコード。制御インターバルは、 VSAM が直接アクセス記憶装置との間で伝送する情報の単位。制御インターバルには、常に整数の物理レコード数が含まれる。

制御インターバル定義フィールド (CIDF) (control interval definition field, CIDF). VSAM では、各制御インターバルの終わりの 4 バイトにあるフィールド。このフィールドは、制御インターバル内の空きスペースを記述する (ある場合)。

コントロール・メタデータ (control metadata). データウェアハウスセンターで、ウェアハウスの内容の変更に関する情報 (たとえば、ステップ処理によって表が更新された日時)。

制御点 (control point). (1) APPN では、ノードの資源を管理するための構成要素であり、任意選択でネットワーク内の他のノードにサービスを提供する。その例に、タイプ 5 ノード内のシステム・サービス制御点 (SSCP)、タイプ 4 ノード内の物理装置制御点 (PUCP)、タイプ 2.1 (T2.1) 内のネットワーク・ノード制御点 (NNCP)、および T2.1 エンド・ノード内のエンド・ノード制御点 (ENCP) がある。 SSCP と NNCP は、他のノードにサービスを提供できる。 (2) T2.1 ノードの資源を管理するための構成要素。 T2.1 ノードが APPN ノードの場合、制御点は、他の APPN ノードとの制御点間セッションにかかわることができる。 T2.1 ノードがネットワーク・ノードの場合、制御点は、その T2.1 ネットワーク内の隣接エンド・ノードにサービスを提供することもできる。「*物理装置* (physical unit)」も参照。

制御特権 (control privilege). オブジェクトを完全に制御する権限。それには、オブジェクトをアクセス、消去、または代替する権限と、オブジェクトに対する他のユーザーの特権を拡張したり、取り消したりする権限が含まれる。

制御サーバー (control server). DB2 複製では、適用するサブスクリプション定義、および変更適用プログラムの制御表のデータベース位置。

制御表 (control table). DB2 複製では、複製ソースやサブスクリプションの定義、または他の複製制御情報が保管される表。

会話 (conversation). APPC で、論理装置間 (LU 対 LU) セッションを通した 2 つのトランザクション・プログラムの接続。その接続によって、トランザクションの処理時に互いに通信することができる。

会話型トランザクション (conversational transaction). APPC で、論理装置 (LU) のサービスを使って通信する複数のプログラム。

会話機密保護 (conversation security). APPC では、接続の確立の前にユーザー ID またはグループ ID とパスワードを検証するプロセス。

会話機密保護プロファイル (conversation security profile). 会話機密保護のために APPC で使われるユーザー ID またはグループ ID とパスワードのセット。

協定世界時 (UTC) (Coordinated Universal Time, UTC). 「グリニッジ標準時 (Greenwich Mean Time)」の同義語。

調整エージェント (coordinating agent). アプリケーションからの要求をデータベース・マネージャーが受け取ったときに始動されるエージェント。 そのアプリケーションが存続する限り、そのアプリケーションと関連性をもち続ける。このエージェントは、そのアプリケーションのために働くサブエージェントを調整する。「サブエージェント (subagent)」も参照。

調整プログラム (coordinator). DB2 UDB (OS/390 版) では、他の 1 つまたは複数のシステム上で行われる処理を組み込んだ作業単位のコミットまたはロールバックを調整する、システム構成要素。

調整プログラム・ノード (coordinator node). アプリケーションの元の接続先であり、調整エージェントが常駐するノード。

調整プログラム・サブセクション (coordinator subsection). 他のサブセクション (ある場合) を始動するアプリケーションのサブセクションであり、そのアプリケーションに結果を戻す。

相関関係のある列 (correlated columns). SQL では、ある 1 つの列の値ともう 1 つの列の値との関係。

相関参照 (correlated reference). 副照会の外側にある表の列に対する参照。

相関副照会 (correlated subquery). 副照会の外にある表の列への相関参照を含む副照会。

相関 ID (correlation ID). DB2 UDB (OS/390 版) では、特定のスレッドに関連する識別子。 TSO では、許可 ID またはジョブ名。

相関名 (correlation name). 単一 SQL ステートメント内で表または視点を指定する識別子。これは、FROM 文節で定義することも、 UPDATE または DELETE ステートメントの最初の文節で定義することもできる。

コスト区分 (cost category). SQL ステートメントのバインド時に DB2 UDB (OS/390 版) がそのステートメントに関するコスト見積もりを入れる区分のこと。コスト見積もりは、以下のコスト区分のいずれかに入れることができる。

- A: DB2 UDB (OS/390 版) がデフォルト値を使用しなくてもコスト見積もりを作成できるほど十分な情報を持っていることを示す。
- B: DB2 UDB (OS/390 版) が見積もりのためにデフォルト値の使用を強制された原因となった条件がいくつか存在することを示す。

コスト区分は、ステートメントに対して EXPLAIN が実行されたときに DSN_STATEMNT_TABLE の COST_CATEGORY 列において外部化される。

国別コード (country code). データベースへのアクセス時、日時の表示 (表示および印刷) 形式を指定するのに、アプリケーションの国別コードが使われる。また、データベースの省略時照合順序を指定するのに、コード・ページと一緒に使われる。

結合機能 (coupling facility). OS/390 環境では、特殊な PR/SM™ LPAR 論理区画で、結合機構制御プログラムを実行し、高速キャッシュ、リスト処理、およびロックなどの機能を並列シスプレックスに提供する。

CP. 「*制御点* (control point)」を参照。

CPC. 「*中央処理装置複合システム* (central processor complex)」を参照。

CPI-C. 「*共通プログラミング・インターフェース通信機能* (Common Programming Interface Communications)」を参照。

CPI-C サイド情報プロファイル (CPI-C side information profile). SNA では、リモート・トランザクション・プログラムで会話を割り当てるときに使う会話特性を指定するプロファイル。このプロファイルは、CPI 通信を介して通信するローカル・トランザクション・プログラムで使われる。これは、パートナー LU 名 (リモート LU 名の入った接続プロファイルの名前)、モード名、およびリモート・トランザクション・プログラム名を指定する。

CP 名 (CP name). 制御点名。制御点のネットワーク修飾名。制御点ノードが帰属するネットワークを識別するネットワーク ID 修飾子で構成される。

破損回復 (crash recovery). 即時障害から回復するプロセス。

CRC. 「*コマンド認識文字* (command recognition character)」を参照。

CRCR. DB2 UDB (OS/390 版) では、条件付き再始動制御レコード (conditional restart control record)。「*条件付き再始動* (conditional restart)」を参照。

リンク・パック域作成 (CLPA) (create link pack area, CLPA). IPL 中に、リンク・パック・ページ可能域の初期化に使われるオプション。

仮想記憶間連係 (cross-memory linkage). OS/390 環境では、異なるアドレス空間でプログラムを呼び出すための方式。この呼び出しは、呼び出し側に関して同期。

システム間結合機能 (XCF) (cross-system coupling facility, XCF). 並列シスプレックス内で実行されている許可プログラム間で協調し合えるようにする機能を提供する OS/390 の構成要素。

システム間拡張サービス (XES) (cross-system extended services, XES). 並列シスプレックス内の異なるシステムで稼働しているアプリケーションやサブシステムの複数インスタンスが、結合機能を使用して高度のパフォーマンスと高い使用可能性が備わったデータ共用を実行できるようにする、一組の OS/390 サービス。

CS. 「*カーソル固定* (cursor stability)」を参照。

CSA. 「*共通サービス域* (common service area)」を参照。

CT. 「*カーソル表* (cursor table)」を参照。

現行データ (current data). DB2 UDB (OS/390 版) では、ホスト構造内のデータであり、基礎表内のデータと同じもの。

現行関数パス (current function path). 関数およびデータ・タイプに対する非修飾の参照の解決で使われるスキーマ名の番号付きリスト。動的 SQL では、現行関数パスは、CURRENT FUNCTION PATH 特殊レジスター内にある。静的 SQL では、PREP コマンドと BIND コマンドの FUNCPATH オプションに定義される。

現行状況再作成 (current status rebuild). DB2 UDB (OS/390 版) では、ログ上の情報からサブシステムの状況の再構成が行われる、再始動処理の第 2 フェーズ。

現行作業ディレクトリー (current working directory). すべての相対パス名を決定する基準となる、デフォルト・ディレクトリー。

カーソル (cursor). いくつかの行の番号付きセット内の個々の行を指し示すのにアプリケーション・プログラムが使う名前付き制御構造。カーソルは、そのセットから行を取り出すのに使われる。

カーソル固定 (CS) (cursor stability, CS). カーソルが任意の行上に置かれているときに、アプリケーションのトランザクションがアクセスするその行をロックする分離レベル。次の行が取り出されるか、またはトランザクションが終了するまで、そのロックは有効のままになる。行内のいずれかのデータが変更された場合、データベースに対してその変更内容がコミットされるまでロックは保持される。

カーソル表 (CT) (cursor table, CT). DB2 UDB (OS/390 版) では、実行するアプリケーション・プロセスが使用するスケルトン・カーソル表のコピー。

サイクル (cycle). DB2 UDB (OS/390 版) では、各表がその前の表の下層となり、最初の表が最後の表の下層となるように順序付けできる一連の表。自己参照表は、単一メンバーの場合のサイクル。

D

DARI. データベース・アプリケーション・リモート・インターフェース (Database Application Remote Interface). 「*ストアード・プロシージャー* (stored procedure)」の以前の用語。

データ域 (data area). プログラムが情報を保持するのに使うメモリー領域。

データベース・アクセス・スレッド (database access thread). DB2 UDB (OS/390 版) では、リモート・サブシステムに代わってローカル・サブシステムでデータにアクセスできるスレッド。

データベース管理者 (DBA) (database administrator, DBA). データベースの設計、開発、操作、セキュリティー、保守、および使用についての責任者。

データベース・アプリケーション・リモート・インターフェース (Database Application Remote Interface, DARI). 「*ストアード・プロシージャー* (stored procedure)」の以前の用語。

データベース・カタログ (database catalog). データウェアハウスセンターでは、表、視点、および索引などのデータベース・オブジェクトの記述を収めている表の集合。

データベース・クライアント (database client). データベース・サーバーにあるデータベースにアクセスするのに使われるワークステーション。

データベース接続サービス (DCS) ディレクトリー (database connection services (DCS) directory). リモート・データベースとそれにアクセスするのに使われるアプリケーション・リクエスター用の項目の入ったディレクトリー。

データベース記述子 (DBD) (database descriptor, DBD). DB2 UDB (OS/390 版) カタログにあるデータ定義を反映する DB2 UDB (OS/390 版) データベース定義の内部表示。 データベース記述子で定義されるオブジェクトは、表スペース、表、索引、索引スペース、および関係である。

データベース・ディレクトリー (database directory). クライアントから接続できるすべてのデータベースについてのデータベース・アクセス情報の入ったディレクトリー。

データベース・エンジン (database engine). データベースを使用するために必要な基本となる関数や構成ファイルを提供するデータベース・マネージャーの部分。

データベース・ログ (database log). データベースに対するすべての変更内容を記録するログ・レコードで構成される一連の 1 次および 2 次ログ・ファイル。データベース・ログを使って、コミットされない作業単位に加えられた変更をロールバックしたり、データベースを整合性のある状態に回復したりする。

データベース管理スペース (DMS) 表スペース (database-managed space (DMS) table space). データベースによってスペースを管理される表スペース。「*システム管理スペース (SMS) 表スペース* (system system-managed space (SMS) table space)」と対比。

データベース管理システム (DBMS) (database management system, DBMS). 「データベース・マネージャー (database manager)」の同義語。

データベース・マネージャー (database manager). 効率のよいアクセス、整合性、回復、並行性制御、プライバシー、および機密保護のための中央制御、データ独立性、および複合物理構造の各種サービスを提供することによってデータを管理するコンピューター・プログラム。

データベース・マネージャー・インスタンス (database manager instance). 実際のデータベース・マネージャー環境のイメージに似た論理データベース・マネージャー環境。 1 つのワークステーション上にデータベース・マネージャー製品の複数のインスタンスがあることがある。そのようなインスタンスを使うと、実働環境から開発環境を分離したり、特定の環境に合わせてデータベース・マネージャーを調整したり、特定の人々のグループに対して重要な情報を保護したりできる。

データベース・ノード (database node). 「*データベース区画* (database partition)」を参照。

データベース・オブジェクト (database object). SQL を使って作成または操作できるすべてのもの。たとえば、表、視点、索引、パッケージ、トリガー、または表スペース。

データベース区画 (database partition). 固有のユーザー・データ、索引、構成ファイル、およびトランザクション・ログから成るデータベースの部分。*ノード* または*データベース・ノード* とも呼ぶ。

データベース要求モジュール (DBRM) (database request module, DBRM). DB2 UDB (OS/390 版) プリコンパイラーにより作成され、 SQL ステートメントについての情報が入っているデータ・セット・メンバー。 DBRM は、バインド・プロセスで使用される。

データベース・サーバー (database server). データベースにデータベース・サービスを提供する機能単位。

データベース・システム・モニター (database system monitor). データベース・マネージャーおよび DB2 コネクトを使って、データベース・マネージャー、データベース、およびアプリケーションについてのパフォーマンスと状況に関する情報をモニターするプログラミング API の集合。

データ・ブロック化 (data blocking). サブスクリプション・サイクルで、変更すべきデータの 1 つの部分の複写に何分費やすかを指定するプロセス。「*ブロック化* (blocking)」と対比。

データの現行性 (data currency). DB2 UDB (OS/390 版) では、検索されてプログラムのホスト変数に入れられたデータが、基礎表中のデータのコピーであるという状態。

データ定義言語 (DDL) (data definition language, DDL). データベース内のデータとその相関関係を記述するための言語。「*データ記述言語* (data description language)」の同義語。

データ定義名 (DD 名) (data definition name, ddname). DB2 UDB (OS/390 版) では、同じ名前が入っているデータ制御ブロックに対応する、データ定義 (DD) ステートメントの名前。

データ記述言語 (data description language). 「*データ定義言語* (data definition language)」の同義語。

DataJoiner. 分散データに対するクライアント・アプリケーション統合アクセスを提供し、異種環境の単一データベース・イメージを提供する別途注文の製品。 DataJoiner を使うと、クライアント・アプリケーションは、 (単一の SQL ステートメントを使って) 複数のデータベース管理システムにまたがって配布されているデータを結合したり、データがローカルのものであるかのように単一のリモート・データ・ソースを更新したりできる。

DataJoiner 複写管理 (DJRA) ツール (DataJoiner Replication Administration (DJRA) tool). さまざまな複製管理タスクの実行に使用できる、データベース管理ツール。コントロール・センターとは異なり、DJRA は IBM 以外のデータベースの複製管理に使用できる。「**コントロール・センター** (Control Center)」と対比。

DATALINK. データベースから、データベース外に保管されているファイルへの論理参照を可能にする、DB2 データ・タイプの 1 つ。

データ・リンク制御 (DLC) (data link control, DLC). SNA では、 2 つのノード間でのリンクを経由したデータ転送をスケジュールしたり、そのリンクのエラー制御を行ったりするリンク・ステーションで構成されるプロトコル層。

データ操作言語 (DML) (data manipulation language, DML). データを操作するのに使われる SQL ステートメントのサブセット。

データマート (datamart). ある部署またはチーム専用のデータを収容するデータウェアハウスのサブセット。データマートを、OLAP ツールに含まれるデータのように、組織全体のデータウェアハウスのサブセットにすることができる。

データ区画 (data partition). OS/390 環境では、区分表スペースに入っている VSAM データ・セット。

データ共用 (data sharing). 2 つまたはそれ以上の DB2 UDB (OS/390 版) サブシステムが、単一セットのデータに直接アクセスし、それを変更できる機能。

データ共用グループ (data sharing group). 1 つまたは複数の DB2 UDB (OS/390 版) サブシステムの集合。データ保全性を保持しつつ、同じデータに直接アクセスし、それを変更する。

データ共用メンバー (data sharing member). XCF サービスによって、データ共用グループに割り当てられる DB2 UDB (OS/390 版) サブシステム。

データ空間 (data space). DB2 UDB (OS/390 版) では、プログラムが直接操作できる 2 GB までの連続する仮想記憶域アドレス。アドレス空間とは異なって、データ空間はデータのみを保持することができ、共通域、システム・データ、あるいはプログラムは含まれない。

データ・タイプ (data type). SQL では、列、リテラル、ホスト変数、特殊レジスター、および関数と式の結果の属性。

データウェアハウスセンター (Data Warehouse Center). ウェアハウス構成要素に関する操作を可能にする、グラフィカル・インターフェースとその背後のソフトウェア。データウェアハウスセンターを使用すると、ウェアハウス・データの定義や管理、およびデータをウェアハウス内に作成するプロセスの定義や管理が可能になります。

データウェアハウスセンター コントロール・インターフェース (Data Warehouse Center administrative interface). データウェアハウスセンターの管理機能のユーザー・インターフェース。このインターフェースは、データウェアハウスセンター・サーバー上に配置でき、複数の管理者の異なるマシン上にも配置することができます。

データウェアハウスセンター プログラム (Data Warehouse Center program). データウェアハウスセンター付属のプログラム。データウェアハウスセンターから開始することができ、 DB2 ロード・プログラムやトランスフォーマーとして自動的に定義できる。

データウェアハウスセンター プロパティー (Data Warehouse Center property). データウェアハウスセンターの複数のセッションにまたがって適用される属性 (たとえばテクニカル・メタデータを含むウェアハウス・コントロール・データベース)。「*プロパティー* (property)」を参照。

日付 (date). 日、月、および年を示す 3 つの部分から構成される値。

日付期間 (date duration). 年、月、および日の数を表す DECIMAL(8,0) 値。

日時値 (datetime value). データ・タイプ DATE、TIME、または TIMESTAMP の値。

DBA. 「*データベース管理者* (database administrator)」を参照。

DBA ユーティリティー (DBA Utility). DB2 ユーザーが、グラフィカル・インターフェースを使って、データベースおよびデータベース・マネージャー・インスタンスを構成したり、ローカルおよびリモートのデータベースにアクセスするのに必要なディレクトリーを管理したり、データベースまたは表スペースをバックアップおよび回復したり、システム上の媒体を管理したりするためのツール。このツールで提供されたタスクは、コントロール・センターからアクセスできる。

DBCLOB. 「*2 バイト文字ラージ・オブジェクト* (double-byte character large object)」を参照。

DBCS. 「*2 バイト文字セット* (double-byte character set)」を参照。

DBD. 「*データベース記述子* (database descriptor)」を参照。

DBID. データベース識別子 (Database identifier)。

DBMS. データベース管理システム (Database management system)。「データベース・マネージャー (database manager)」を参照。

DBMS インスタンス接続 (DBMS instance connection). アプリケーションと、 DB2 インスタンスが所有するエージェント・プロセスまたはスレッドとの間の論理結合。

DBRM. 「*データベース要求モジュール* (database request module)」を参照。

DB2 CLI. DB2 コール・レベル・インターフェース (DB2 Call Level Interface)。DB2 機能の利点を活かした DB2 製品ファミリー用の代替 SQL インターフェース。

DB2 コマンド (DB2 command). ユーザーによる DB2 UDB (OS/390 版) の開始または停止、現行ユーザーについての情報の表示、データベースの開始または停止、データベースの状況に関する情報の表示、などが行えるようにする、 DB2 UDB (OS/390 版) サブシステムへの命令。

DB2 コネクト (DB2 Connect). クライアント・アプリケーションに必要な機能 (DRDA アプリケーション・リクエスター・サポート) を提供する製品。クライアント・アプリケーションはその製品を使って、 DRDA アプリケーション・サーバーに記憶されているデータを読み取ったり、更新したりする。

DB2 エクステンダー (DB2 extender). 従来の数値および文字データ以外のデータ・タイプ (たとえば画像、音声、ビデオ・データ)、およびこれらの複合した文書を保管および検索できるプログラム。

DB2I. DB2 UDB (OS/390 版) では、DB2 対話機能 (DATABASE 2 Interactive)。

DB2I 漢字フィーチャー (DB2I Kanji Feature). DB2 UDB (OS/390 版) では、DB2I パネルを漢字で表示できるようにするパネルとジョブを収めているテープ。

DB2 PM. DB2 UDB (OS/390 版) では、DATABASE 2 パフォーマンス・モニター (DATABASE 2 Performance Monitor)。

DB2 SDK. 「*DB2 ソフトウェア開発者キット (DB2 SDK)* (DB2 Software Developer's Kit, DB2 SDK)」を参照。

DB2 ソフトウェア開発者キット (DB2 SDK) (DB2 Software Developer's Kit, DB2 SDK). 開発者がデータベース・アプリケーションを作成するときに役に立つツールの集合。

DB2 スレッド (DB2 thread). アプリケーションの接続を記述し、その過程をトレースし、資源機能を処理し、そしてその DB2 UDB (OS/390 版) 資源とサービスへのアクセス可能性の範囲を区切る DB2 UDB (OS/390 版) 構造。

DB2UEXIT. アーカイブ・ログ・ファイルを移動または検索するのにデータベース・マネージャーが呼び出すユーザー作成の任意指定の実行可能プログラム。

DCE. 「*分散コンピューティング環境* (Distributed Computing Environment)」を参照。

DCE チケット (DCE ticket). OS/390 環境では、開始プリンシパルの識別をそのターゲットに伝送する透過アプリケーションのメカニズム。単一のチケットには、ターゲットの機密キーによって封印された、プリンシパルの ID、セッション・キー、タイム・スタンプ、およびその他の情報が含まれる。

DCLGEN. 「*宣言生成機能* (declarations generator)」を参照。

DDF. 「*分散データ機能* (distributed data facility)」を参照。

DDL. 「*データ定義言語* (data definition language)」を参照。

DD 名 (ddname). 「*データ定義名* (data definition name)」を参照。

デッドロック (deadlock). あるトランザクションが、他のいずれかのトランザクションによってロックされている排他的資源に依存している (そのために、ロックしているそのトランザクションは、元のトランザクションが使用する排他的資源に依存することになる) ため、先に進めない状態。

デッドロック検出機能 (deadlock detector). デッドロック条件が存在するかどうかを判別するためにロックの状態をモニターするデータベース・マネージャー内のプロセス。 デッドロック条件が検出されると、検出機能は、そのデッドロックに関与しているトランザクションの 1 つを停止する。このトランザクションはロールバックされ、他のトランザクションは処理を継続する。

宣言生成機能 (DCLGEN) (declarations generator, DCLGEN). SQL 表宣言、ならびにその表に従った COBOL、C、または PL/I データ構造宣言を生成する、DB2 UDB (OS/390 版) の副構成要素。 宣言は、DB2 UDB (OS/390 版) システム・カタログ情報から生成される。 DCLGEN も DSN サブコマンドである。

据え置き組み込み SQL (deferred embedded SQL). DB2 UDB (OS/390 版) では、完全に静的でも、完全に動的でもない SQL ステートメント。静的ステートメントのようにアプリケーション内に組み込まれるが、動的ステートメントのようにアプリケーションの実行中に作成されるステートメント。

定義メタデータ (definition metadata). データウェアハウスセンターで、データウェアハウスの形式 (スキーマ)、データのソース、およびデータのロード中に適用されるトランスフォーメーションについての情報。

並列性の程度 (degree of parallelism). DB2 UDB (OS/390 版) では、照会の処理のために開始する、同時に実行された操作の数。

削除連鎖 (delete-connected). SQL では、表 P の従属、または表 P からの削除操作が連鎖する表の従属のものである表。

削除規則 (delete rule). 親行の削除を制限したり、従属行でのそのような削除による効果を指定したりするための参照制約に関連した規則。

削除トリガー (delete trigger). DB2 UDB (OS/390 版) では、トリガー用の SQL 操作 DELETE を指定して定義したトリガーのこと。

区切り識別子 (delimited identifier). 二重引用符で囲まれた文字列。この文字列は、1 文字とその後につづくゼロ個以上の文字 (そのおのおのが文字、数字、または下線文字) で構成されていなければならない。

区切り文字 (delimiter). データ項目をグループ化したり分離したりするための文字またはフラグ。

区切りトークン (delimiter token). 構文図に示されているストリング定数、区切り識別子、演算子、または特殊文字。

従属 (dependent). SQL では、最低 1 つの親を持つオブジェクト (行、表、または表スペース)。「*親行* (parent row)」、「*親表* (parent table)」、「*親表スペース* (parent table space)」を参照。

従属論理装置 (DLU) (dependent logical unit, DLU). LU-LU (論理装置間) セッションをインスタンス化するためにシステム・サービス制御点 (SSCP) の補助を必要とする論理装置。

従属行 (dependent row). 親行内の親キーの値に一致する外部キーの入った行。 外部キーの値は、従属行から親行への参照を表す。

従属表 (dependent table). 最低限 1 つの参照制約に従属する表。

下層 (descendent). あるオブジェクトの従属であるオブジェクト、またはあるオブジェクトの下層の従属であるオブジェクト。

下層行 (descendent row). 別の行に従属するか、または従属行の下層にある行に従属する行。

下層表 (descendent table). 別の表に従属するか、または従属表の下層の表に従属する表。

deterministic 関数 (deterministic function). 「*非可変関数* (not-variant function)」を参照。

装置名 (device name). システムで予約されているか、または特定の装置を参照する装置ドライバーで予約されている名前。

DFHSM. OS/390 環境では、データ機能階層記憶管理プログラム (Data Facility Hierarchical Storage Manager)。

DFP. OS/390 環境では、データ機能プロダクト (Data Facility Product)。

ディクショナリー (dictionary). テキスト・エクステンダーがテキストの分析、索引付け、検索を行ったり、特定言語の文書を強調表示したりするのに使用する、言語関連の情報のコレクション。

差分最新表示 (differential refresh). DB2 複製では、変更されたデータだけをターゲット表にコピーして、既存のデータと置き換えるプロセス。「**全最新表示** (full refresh)」と対比。

ディメンション (dimension). OLAP Starter Kit におけるデータ区分 (たとえば、時刻、会計、製品、または市場)。ディメンションは、多次元データベースの枠組みの中の最上位の統合レベルを表す。

指示結合 (directed join). 結合した表の 1 つあるいは両方にあるすべての行が、結合述部に基づいてハッシュし直され新しいデータベース区画に宛先指定される関係操作。ある表内のすべての区分化キー列が equijoin 述部に関係する場合、もう片方の表が作り直される。そうでない場合 (少なくとも 1 つの equijoin 述部がある場合)、両方の表が作り直される。

ディレクトリー (directory). データベース記述子およびスケルトン・カーソル表のような内部オブジェクトが入っている、 DB2 UDB (OS/390 版) システム・データベース。

ディレクトリー・サービス (directory services). APPN ネットワーク内の資源位置についての情報を保持する APPN プロトコルの部分。

使用不能にする (disable). 使用可能プロセス中に作成された項目を除去することにより、データベース、テキスト・テーブル、またはテキスト列をテキスト・エクステンダーで使用可能になる前の状態に復元すること。

特殊タイプ (distinct type). 内部的には既存のタイプ (そのソース・タイプ) として表示されるが、セマンティクスとしての用途においては別個で非互換のタイプとみなされるユーザー定義のデータ・タイプ。

分散コンピューティング環境 (DCE) (Distributed Computing Environment, DCE). 異種のコンピューティング環境で分散アプリケーションの作成、使用、および保守をサポートする一連のサービスとツール。 DCE は、オペレーティング・システムおよびネットワークから独立したものであり、異種のプラットフォーム相互間での相互操作性と移行性を提供する。

分散データ機能 (DDF) (distributed data facility, DDF). DB2 UDB (OS/390 版) が他の RDBMS と通信を行うための、一組の DB2 UDB (OS/390 版) 構成要素。

分散ディレクトリー・データベース (distributed directory database). APPN ネットワーク全体に散在する個々のディレクトリー内に維持されている、ネットワークのすべての資源の完全リスト。各ノードには、完全ディレクトリーが 1 つずつあるが、どのノードもリスト全体を持つ必要はない。システム定義、操作員処置、自動登録、および現在のネットワーク検索手順を通して、項目が作成、変更、および削除される。「*分散ネットワーク・ディレクトリー* (distributed network directory)」の同義語。

分散ネットワーク・ディレクトリー (distributed network directory). 「*分散ディレクトリー・データベース* (distributed directory database)」を参照。

分散リレーショナル・データベース (distributed relational database). 相互接続された別のコンピューター・システムに表を記憶されているデータベース。

分散リレーショナル・データベース体系 (DRDA) (Distributed Relational Database Architecture, DRDA). リモート・データに透過アクセスするための形式とプロトコルを定義する体系。 DRDA は、アプリケーション・リクエスター機能と、アプリケーション・サーバー機能の 2 つのタイプの機能を定義する。

分散要求 (distributed request). 連合データベース・システムにおいて、複数のデータ・ソースに送信される SQL 照会。

分散作業単位 (DUOW) (distributed unit of work, DUOW). 複数のリレーショナル・データベース管理システムに SQL ステートメントの実行を要求するための作業単位。ただし、各 SQL ステートメントにつき複数のシステムに実行要求することはできない。

DJRA ツール (DJRA tool). さまざまな複製管理タスクの実行に使用できる、データベース管理ツール。コントロール・センターとは異なり、 DJRA は IBM 以外のデータベースの複製管理にも使用できる。「**コントロール・センター** (Control Center)」と対比。

DLC. 「*データ・リンク制御* (data link control)」を参照。

DLU. 「*従属論理装置* (dependent logical unit)」を参照。

DML. 「*データ操作言語* (data manipulation language)」を参照。

DMS 表スペース (DMS table space). 「*データベース管理スペース表スペース* (Database managed space table space)」を参照。

DNS. 「*ドメイン・ネーム・システム* (domain name system)」を参照。

文書アクセス定義 (DAD) (Document Access Definition, DAD). XML コレクション (XML 形式) の XML エクステンダー列を使用可能にするために使う定義。

文書モデル (document model). 文書が含むセクションに関する、文書の構造の定義。テキスト・エクステンダーが索引付け時に文書モデルを使用する。

ドメイン名 (domain name). TCP/IP アプリケーションが TCP/IP ネットワーク内で TCP/IP ホストを参照する名前。ドメイン名は、一つながりの名前とそれらを区切るドットで構成される。

ドメイン・ネーム・サーバー (DNS) (domain name server, DNS). TCP/IP ネットワーク・サーバーで、TCP/IP ホスト名を IP アドレスにマップする場合に使用される分散ディレクトリーを管理する。

ドメイン・ネーム・システム (domain name system). 人の読めるマシン名を IP アドレスにマップするため、 TCP/IP で使われる分散データベース・システム。

Domino™ *Go* Web server. Lotus® Corp. および IBM が提供する、通常の接続と保護接続の両方を可能にする Web サーバー。 ICAPI と GWAPI は、このサーバーで提供されるインターフェース。

2 バイト文字ラージ・オブジェクト (DBCLOB) (double-byte character large object, DBCLOB). 一つながりの 2 バイト文字。最大長は、2 ギガバイト。大きな 2 バイト・テキスト・オブジェクトの記憶に使われるデータ・タイプ。 2 バイト文字ラージ・オブジェクト・ストリングとも呼ばれる。このようなストリングには必ず関連したコード・ページがある。

2 バイト文字セット (DBCS) (double-byte character set, DBCS). おのおのの文字が 2 バイトで表現される文字セット。

倍精度浮動小数点 (double-precision floating point number). SQL では、実数の 64 ビットによる近似表現。

ドレーン (drain). DB2 UDB (OS/390 版) では、ロックされた資源を、そのオブジェクトへのアクセスを静止させて、獲得すること。

ドレーン・ロック (drain lock). DB2 UDB (OS/390 版) では、クレームの発生を防ぐ、クレーム・クラスへのロック。

DRDA. 「*分散リレーショナル・データベース体系 (DRDA)* (Distributed Relational Database Architecture, DRDA)」を参照。

DRDA アクセス (DRDA access). DB2 UDB (OS/390 版) では、分散データにアクセスする方式。これにより、SQL ステートメントを用いて他のロケーションに接続し、そのロケーションで前にバインドされたパッケージを実行できる。アプリケーション・サーバーの識別には SQL の CONNECT ステートメントまたは 3 部分名ステートメントが使用され、それらのサーバーにおいて事前にバインド済みのパッケージを使用して SQL ステートメントが実行される。「*私用プロトコル・アクセス* (private protocol access)」と対比。

DSN. (1) 省略時の DB2 UDB (OS/390 版) のサブシステム名。 (2) DB2 UDB (OS/390 版) の TSO コマンド・プロセッサーの名前。 (3) DB2 UDB (OS/390 版) モジュールおよびマクロ名の先頭 3 文字。

DUOW. 「*分散作業単位* (distributed unit of work)」を参照。

期間 (duration). SQL では、時間間隔を表す数値。「*日付期間* (date duration)」、「*ラベル付き期間* (labeled duration)」、および「*時刻期間* (time duration)」を参照。

動的バインド (dynamic bind). SQL ステートメントがその入力と同時にバインドされるプロセス。「*バインド* (bind)」も参照。

動的 SQL (dynamic SQL). 実行中のプログラムで準備され実行される SQL ステートメント。動的 SQL では、SQL ソースは、プログラムにコード化されるのではなく、ホスト言語に含められる。プログラムの実行時に SQL ステートメントは複数回変更されることがある。

E

EA 使用可能表スペース (EA-enabled table space). DB2 UDB (OS/390 版) では、4 ギガバイトより大きい個別の区分 (LOB 表スペースの場合は部分) を含んでおり、拡張アドレッシング機能 (EA) (extended addressability, EA) が使用可能となっている表スペースまたは索引スペース。

EBCDIC. 拡張 2 進化 10 進コード (Extended binary-coded decimal interchange code)。 256 個の 8 ビット文字のコード化文字セット。

EDM プール (EDM pool). DB2 UDB (OS/390 版) では、データベース記述子、アプリケーション・プラン、許可キャッシュ、アプリケーション・パッケージ、および動的ステートメント・キャッシュのために使用される主記憶域内のプール。

EID. イベント識別子 (Event identifier)。

組み込み SQL (embedded SQL). アプリケーション・プログラム内にコーディングされる SQL ステートメント。「*静的 SQL* (static SQL)」を参照。

EN. 「*エンド・ノード* (end node)」を参照。

使用可能にする (enable). データベース、テキスト・テーブル、またはテキスト列をテキスト・エクステンダーで使用できるようにすること。

エンクレーブ (enclave). 言語環境プログラム (DB2 UDB (OS/390 版) によって使用される) では、ルーチンの独立コレクションを指す。この 1 つがメインルーチンとして指定される。エンクレーブは、プログラムまたは実行単位と似ている。

コード化スキーム (encoding scheme). 文字データを表す一連の規則。

エンド・ノード (EN) (end node, EN). APPN では、ローカル制御点と隣接ネットワーク・ノードの制御点との間のセッションをサポートするノード。

拡張対立検出 (enhanced conflict detection). すべてのレプリカとソース表の間でのデータ保全性を確保するための対立検出。変更適用プログラムは、将来のトランザクションに対して設定されたサブスクリプション・セット内のすべてのレプリカまたはユーザー表をロックする。その後、ロックが捕そくされる前に加えられたすべての変更の後の検出を開始する。「*対立検出* (conflict detection)」、「*標準対立検出* (standard conflict detection)」、および「*行レプリカ対立検出* (row-replica conflict detection)」も参照。

環境ハンドル (environment handle). データベース・アクセス用のグローバル文脈を識別するハンドル。環境内のすべてのオブジェクトに適したすべてのデータは、このハンドルに関連付けられている。

環境プロファイル (environment profile). テキスト・エクステンダーが提供するスクリプト。環境変数の設定を含む。

EOM. メモリーの終了 (End of memory)。

EOT. タスクの終了 (End of task)。

等価結合 (equijoin). 述部に等号演算子 (=) を含む結合 (たとえば、T1.C1 = T2.C2)。

エラー・ページ範囲 (error page range). 物理的に損傷していると考えられるページ範囲。 DB2 UDB (OS/390 版) では、ユーザーがこの範囲内にあるページにアクセスすることを認めない。

エスケープ文字 (escape character). SQL 区切り識別子を囲むために使用される記号。エスケープ文字は二重引用符である。ただし、COBOL アプリケーションでは、二重引用符またはアポストロフィである記号をユーザーが割り当てることができる。

ESDS. OS/390 環境では、入力順データ・セット (entry sequenced data set)。

ESMT. OS/390 環境で、IMS の外部サブシステム・モジュール表 (external subsystem module table)。

EUC. 「*拡張 UNIX® コード* (Extended UNIX Code)」を参照。

イベント・モニター (event monitor). ある期間のデータベース活動についてのデータをモニターしたり、収集したりするためのデータベース・オブジェクト。

イベント・タイミング (event timing). DB2 複製では、サブスクリプション・サイクルを開始するタイミングを制御する最も精密な方法。イベント、およびそのイベントを処理したい時間を指定する必要がある。「**インターバル・タイミング** (interval timing)」および「**オンデマンド・タイミング** (on-demand timing)」と対比。

例外表 (exception table). DB2 UDB (OS/390 版) では、参照制約または表検査制約に違反し、CHECK DATA ユーティリティーによって検出された行を保持している表。

排他ロック (exclusive lock). 並行して実行されているアプリケーション・プロセスがデータベース・データにアクセスしないようにするロック。

実行可能ステートメント (executable statement). アプリケーション・プログラムに組み込んだり、動的に準備して実行したり、対話式で発行したりできる SQL ステートメント。

出口ルーチン (exit routine). 特定の関数を実行するために、他のプログラム (たとえば DB2 UDB (OS/390 版)) から制御を受け取るプログラム。

Explain (explain). SQL ステートメントを解決するため SQL コンパイラーが選んだアクセス・プランについての詳細な情報を獲得すること。この情報は、そのアクセス・プランを選ぶのに使われた決定基準を記述している。

Explain 可能ステートメント (explainable statement). Explain 操作が実行可能な SQL ステートメント。 Explain 可能ステートメントは SELECT、UPDATE、INSERT、DELETE、および VALUES。

Explained ステートメント (explained statement). Explain 操作が実行された SQL ステートメント。

Explained 統計 (explained statistics). SQL ステートメントに対して Explain が実行されたときにそのステートメント内で参照されたデータベース・オブジェクトの統計。

Explain スナップショット (explain snapshot). SQL 照会とその関連情報の現行の内部表記を獲得したもの。この情報は、Visual Explain ツールに必要である。

明示的階層ロッキング (explicit hierarchical locking). DB2 UDB (OS/390 版) では、IRLM に認識される資源の間に親子関係を設定するために使用するロッキング。 この種のロッキングは、資源に DB2 間インタレストがないときに、グローバル・ロック・オーバーヘッドを避ける。

明示特権 (explicit privilege). 名前を持ち、SQL GRANT および REVOKE ステートメントの結果として保持される特権 (たとえば SELECT 特権)。「*暗黙特権* (implicit privilege)」と対比。

エクスポート (export). PC/IXF、DEL、WSF、または ASC などの形式を使って、データベース・マネージャー表からファイルにデータを複写すること。「インポート (import)」と対比。

直接的な名前 (exposed name). 相関名を指定されていない FROM 文節に指定された相関名、表、または視点名。

式 (expression). 1 つの値を生み出す 1 つの SQL オペランド、または演算子とオペランドの集合。

拡張回復機能 (XRF) (extended recovery facility, XRF). OS/390 環境で、高可用性アプリケーションと指定した端末との間のセッション中に、MVS、VTAM、ホスト・プロセッサー、または高可用性アプリケーションにおける障害の影響を最小限に抑える機能。この機能は、障害のあったセッションからセッションを引き継ぐ代替サブシステムを提供する。

拡張 UNIX コード (EUC) (Extended UNIX Code, EUC). 長さが 1〜4 バイトの文字セットをサポートできるプロトコル。 EUC は、コード・ページの集合を指定する手段である。コード・ページの実際のコード化スキーマそのものではない。これは、PC 2 バイト (DBCS) コード・ページのコード化スキーマの UNIX 用の代替コードである。

エクステント (extent). 表スペースのコンテナー内での、スペースの単一データベース・オブジェクトへの割り振り。この割り振りは複数のページで構成される。

エクステント・マップ (extent map). 表スペース内に記憶されるメタデータ構造体。表スペース内の各オブジェクトに対するエクステントの割り振りを記録する。

外部 CCD 表 (external CCD table). DB2 複製で、登録された複製ソースであるために、直接サブスクリプションできる CCD 表。その独自の行がレジスター表の中に存在し、レジスター表では SOURCE_OWNER および SOURCE_TABLE として参照されている。「**内部 CCD 表** (internal CCD table)」と対比。

外部関数 (external function). DB2 UDB (OS/390 版) では、その本体が、それぞれの呼び出しごとにスカラー引き数値を使用してスカラー結果を出すプログラム言語で書かれた関数。「*ソース関数* (sourced function)」および「*組み込み関数* (built-in function)」と対比。

外部ルーチン (external routine). DB2 UDB (OS/390 版) では、外部プログラム言語で書かれたコードに基づく、ユーザー定義関数またはストアード・プロシージャー。

F

ファクト表 (fact table). OLAP Starter Kit で、リレーショナル・キューブのすべての値を含んでいる DB2 の表、または多くの場合 4 つの表からなるセット。

障害メンバー状態 (failed member state). DB2 UDB (OS/390 版) では、データ共用グループのメンバーの状態。メンバーに障害が起こると、XCF が障害メンバー状態を永続的に記録する。この状態はメンバーのタスク、アドレス空間または MVS システムが活動状態から静止状態になる前に終了することを意味している。

フォールバック (fallback). 現行リリースへの移行を試みた後、または移行を完了した後に、前の DB2 UDB (OS/390 版) のリリースに戻るプロセス。

偽グローバル・ロック競合 (false global lock contention). DB2 UDB (OS/390 版) では、複数のロック名が同じ標識にハッシュされる場合、および実競合がない場合の結合機能からの競合指示。

高速コミュニケーション・マネージャー (FCM) (fast communication manager, FCM). ノード間のコミュニケーション・サポートを提供する機能グループ。

連合データベース・システム (federated database system). (1) 複数のデータベースから構成されるが、単一のデータベース・イメージを提供するデータベース・システム。 1 つの DB2 サーバーと、そのサーバーが照会を送る複数のデータ・ソースで構成される。連合データベース・システムでは、クライアント・アプリケーションは、 (単一の SQL ステートメントを使って) 複数のデータベース管理システムにまたがって分散しているデータを結合し、データがローカルのものであるかのように表示できる。 (2) 以下のもので構成される分散コンピューティング・システム。

- 1 つの DB2 サーバー (**連合サーバー** (federated server) という)。
- 連合サーバーが照会を送る複数のデータ・ソース。

  各データ・ソースは、リレーショナル・データベース管理システムのインスタンスが 1 つ、およびそのインスタンスがサポートする 1 つまたは複数のデータベースで構成される。

  これらのデータ・ソースは半自動走行式である。たとえば、Oracle アプリケーションが Oracle データ・ソースにアクセスしている時、同時に連合サーバーがこれらのデータ・ソースに照会を送ることができる。

分離 (fenced). 関数による DBMS の修正に対して保護するよう定義されているユーザー定義関数またはストアード・プロシージャーのタイプに関連すること。 DBMS は、障壁によって関数またはストアード・プロシージャーから分離される。「*非分離の* (not-fenced)」と対比。

フィールド・プロシージャー (field procedure). DB2 UDB (OS/390 版) では、単一の値を受け取り、それをユーザーが指定できる任意の方法でそれを変換 (コード化またはデコード) するように設計された、ユーザー作成の出口ルーチン。

ファイル参照変数 (file reference variable). クライアント・メモリー・バッファーにではなく、クライアントのファイル内にデータを常駐させるよう指示するのに使われるホスト変数。

ファイル・サーバー (file server). NetWare オペレーティング・システム・ソフトウェアを実行し、ネットワーク・サーバーとして働くワークステーション。 DB2 は、ファイル・サーバーを使用して DB2 サーバーのアドレス情報を記憶し、それを DB2 クライアント・サーバーが取り出して IPX/SPX クライアント / サーバー接続を確立する。

フィルター係数 (filter factor). DB2 UDB (OS/390 版) では、述部が真である表の行の比率を見積もる 0 ～ 1 の数値。

基本障害保守ログ (first failure service log). 診断メッセージ、診断データ、アラート情報、およびそれに関連したダンプ情報の入ったファイル (db2diag.log)。このファイルは、データベース管理者によって使われる。

固定長ストリング (fixed-length string). 長さが特定されていて、変更できない文字または漢字のストリング。「*可変長ストリング* (varying-length string)」と対比。

標識機能 (flagger). 選択された検証基準 (たとえば、 ISO/ANSI SQL92 エントリー・レベル標準) に合致しない、アプリケーション内の SQL ステートメントを識別するプリコンパイラー・オプション。

フラット・ファイル・インターフェース (flat file interface). 平文テキスト・ファイルからの読み取りおよび書き込みを可能にする、 Net.Data 標準装備の関数のセット。

外部更新 (foreign update). ターゲット表に対して適用され、ローカル表に複写される更新。

順方向ログ回復 (forward log recovery). DB2 UDB (OS/390 版) がログを順方向に処理し、すべての REDO ログ・レコードを適用する、再始動処理の第 3 フェーズ。

順方向回復 (forward recovery). データベースまたは表スペースをロールフォーワードするのに使われるプロセス。このプロセスで、データベース・ログに記録されている変更内容を使って、復元データベースや表スペースを期限内に指定地点に再作成することができる。

空きスペース (free space). DB2 UDB (OS/390 版) では、ページ内の合計未使用スペース。レコードや制御情報の保管に使用されていないスペース。

全外部結合 (full outer join). 結合中の両方の表の一致した行が入っていて、両方の表の不一致行を保存する、SQL 結合操作の結果。「*結合* (join)」を参照。

全最新表示 (full refresh). DB2 複製では、ユーザー表から必要なデータをターゲット表にコピーして、既存のデータと置き換えるプロセス。 「*差分最新表示* (differential refresh)」と対比。

全選択 (fullselect). セット演算子によって結合された副選択、値文節、またはこの両方。

完全修飾 LU 名 (fully qualified LU name). 「*ネットワーク修飾名* (network-qualified name)」を参照。

関数 (function). (1) プログラム（関数本体）として実現されたマッピング。ゼロ個以上の入力値（引き数）を単一値（結果）に入力するという方法で起動できる。 (2) DB2 UDB (OS/390 版) では、エンティティーの特定の目的、あるいは列関数またはスカラー関数といった、エンティティーの特性を示すアクション。さらに、関数には、ユーザー定義関数、組み込み関数、あるいは DB2 UDB (OS/390 版) 生成関数がある。

関数本体 (function body). 関数を実装するコード。

関数の定義者 (function definer). DB2 UDB (OS/390 版) では、 CREATE FUNCTION ステートメントで指定されている関数のスキームの所有者の許可 ID。

関数ファミリー (function family). 同一の関数名の付いた一連の関数。文脈によって、使用法は特定のスキーマ内の一連の関数に関するものか、または現行関数パス内の同一名のすべての関連関数に関するものかが判別される。

関数の実現者 (function implementer). DB2 UDB (OS/390 版) で、関数プログラムおよび関数パッケージの所有者の許可 ID。

関数呼び出し (function invocation). 関数本体に渡されるすべての引き数値と一緒に関数を使うこと。関数は、その名前で呼び出される。

関数パッケージ (function package). DB2 UDB (OS/390 版) では、関数プログラム用の DBRM をバインドすると結果的に作成されるパッケージ。

関数パッケージ所有者 (function package owner). DB2 UDB (OS/390 版) では、関数プログラムの DBRM を関数パッケージにバインドしたユーザーの許可 ID。

関数パス (function path). 非修飾関数の呼び出しの検索範囲を制限し、関数選択プロセスの最終決定者を提供するスキーマ名を並べたリスト。

関数パス・ファミリー (function path family). ユーザーの関数パス内で識別される (または省略時値として使われる) すべてのスキーマ内のある特定の名前のすべての関数。

関数解決 (function resolution). DBMS の内部のプロセス。そのために、呼び出す特定の関数インスタンスが選択される。選択には、関数名、引き数のデータ・タイプ、および関数パスが使われる。「*関数選択* (function selection)」の同義語。

関数選択 (function selection). 「*関数解決* (function resolution)」を参照。

関数シッピング (function shipping). 適用可能データの入った特定のノードに対する、要求のサブセクションのシッピング (出荷)。

関数シグニチャー (function signature). 完全修飾関数名と、そのすべてのパラメーターのデータ・タイプとの論理連結。スキーマ内のすべての関数に固有シグニチャーがなければならない。

関数テンプレート (function template). 連合データベースにおいて、実行可能コードを持たない部分関数。ユーザーはこれをデータ・ソース関数にマップして、このデータ・ソース関数を連合サーバーから起動できるようにする。

G

ギャップ (gap). DB2 複製では、収集プログラムが一定範囲のログまたはジャーナル・レコードを読み取ることができずに、変更データを失う可能性のある状態。

GBP. グループ・バッファー・プール (Group buffer pool)。

GBP 従属 (GBP-dependent). DB2 UDB (OS/390 版) では、グループ・バッファー・プールに従属するページ・セットまたはページ・セット区画状況。このページ・セットの読み取り / 書き込みインタレストが DB2 サブシステムで活動状態であるか、またはまだ DASD に入れられていないグループ・バッファー・プール内に、変更されたページを持つページ・セットがある。

汎用トレース機能 (GTF) (generalized trace facility, GTF). OS/390 環境では、入出力割り込み、SVC 割り込み、プログラム割り込み、または外部割り込みなどの重要なシステム・イベントを記録するサービス・プログラム。

汎用リソース名 (generic resource name). OS/390 環境で、VTAM で使用する名前で、並列シスプレックス内でセッションの分散と平衡化を処理するために同じ機能を提供する数種類のアプリケーション・プログラムを表す。

ページ取得 (getpage). DB2 UDB (OS/390 版) によるデータ・ページのアクセスが行われる操作。

GIMSMP. OS/390 環境では、システム修正変換プログラム、導入、変更、およびプログラミング・システムへの変更の制御のための基本ツール用ロード・モジュール名。

グローバル・ロック (global lock). DB2 UDB (OS/390 版) で、複数の DB2 サブシステム内での同時制御を提供するロック。ロックの効力範囲は、データ共用グループのすべての DB2 サブシステムに渡る。

グローバル・ロック競合 (global lock contention). 共用資源を逐次化しようとする試みに関して、データ共用グループの異なる DB2 UDB (OS/390 版) メンバー間のロッキング要求の競合。

グローバル表ロック (global table lock). 表のノード・グループ内のすべてのノードに対してかけられるロック。

グローバル・トランザクション (global transaction). 複数の資源管理プログラムを必要とする分散トランザクション処理環境内の作業単位。

管理プログラム (governor). 「*資源限定機能* (resource limit facility)」を参照。

授与 (grant). 許可 ID に特権または権限を与えること。

図形文字 (graphic character). DBCS 文字。

漢字ストリング (graphic string). DBCS 文字列。

グロス・ロック (gross lock). DB2 UDB (OS/390 版) では、表、区分または表スペースに対する、*共用* (shared)、*更新* (update)、または*排他* (exclusive) モードのロック。

グループ (group). (1) アクティビティーまたは資源アクセス権限に応じて ID をもつユーザーの論理編成。 (2) サテライト・エディションで、データベース構成、およびサテライト上で実行されるアプリケーションなどの特性を共有するサテライトの集合。

グループ・バッファー・プール二重化 (group buffer pool duplexing). OS/390 環境では、グループ・バッファー・プール構造の 2 つのインスタンス、すなわち *1 次グループ・バッファー・プール* と *2 次グループ・バッファー・プール* にデータを書き込む機能のこと。 OS/390 資料はこれらのインスタンスを'元の' (1 次) および '新しい' (2 次) 構造体として参照する。

グループ名 (group name). OS/390 環境では、データ共用グループの XCF 識別子。

グループ再始動 (group restart). OS/390 環境では、ロックまたは共用通信域が失われた後の、データ共用グループの少なくとも 1 つのメンバーの再始動。

グループ効力範囲 (group scope). 「*コマンド効力範囲* (command scope)」を参照。

GTF. 「*汎用トレース機能* (generalized trace facility)」を参照。

GWAPI. Domino *Go* Web サーバー API。

H

ハンドル (handle). (1) ソフトウェア・システム内の内部構造を表す変数。 (2) 表内のイメージ、オーディオ、またはビデオ・オブジェクトを表すために使う、エクステンダーによって作成される文字ストリング。オブジェクトのハンドルは、ユーザー表と管理サポート表に保管される。この方法でエクステンダーは、ユーザー表に保管されているハンドルと、管理サポート表に保管されているオブジェクトに関する情報とリンクする。 (3) テキスト文書を識別する 2 進値。ハンドルは、テキスト列がテキスト・エクステンダーで*使用可能* な場合に、そのテキスト列内のテキスト文書ごとに作成される。

ハッシュ区分化 (hash partitioning). ハッシュ関数を区分化キー値に適用して、行を割り当てるデータベース区画を判別する区分化の手法。

ハイパースペース (hiperspace). OS/390 環境では、プログラムがバッファーとして使用できる最大 2 G バイトまでの連続する仮想記憶アドレス範囲のこと。アドレス空間と同様に、ハイパースペースはユーザー・データを保持することができるが、共通域やシステム・データは含まない。また、アドレス空間またはデータ空間とは異なり、ハイパースペースのデータに直接アクセスすることはできない。ハイパースペースのデータを操作するには、データを 4-KB ブロックにしてアドレス空間に持ち込むこと。

ホーム・アドレス空間 (home address space). OS/390 環境では、現在 OS/390 がディスパッチされると認識する記憶域。

ホップ (hop). APPN では、中間ノードのないルートの部分。ホップは、隣接ノードを接続する単一の伝送グループで構成される。

ホスト (host). TCP/IP で、それに関連した IP アドレスが少なくとも 1 つ存在するシステム。

ホスト・コンピューター (host computer). (1) コンピューター・ネットワークでは、計算、データベース・アクセス、およびネットワーク制御機能などのサービスを提供するコンピューター。 (2) 複数のコンピューター導入システムにおける 1 次または制御コンピューター。

ホスト識別名 (host identifier). ホスト・プログラムで宣言される名前。

ホスト言語 (host language). SQL ステートメントを組み込める任意のプログラミング言語。

ホスト・ノード (host node). SNA では、システム・サービス制御点 (SSCP) の入ったサブエリア・ノード。たとえば、MVS および VTAM を備えた IBM System/390® コンピューター。

ホスト・プログラム (host program). 組み込み SQL ステートメントを含んだホスト言語で作成されたプログラム。

ホスト構造 (host structure). アプリケーション・プログラムにおいて、組み込み SQL ステートメントによって参照される構造。

ホスト変数 (host variable). アプリケーション・ホスト・プログラムでは、組み込み SQL ステートメントで参照される変数。ホスト変数は、アプリケーション・プログラムのプログラミング変数であり、データベース内の表とアプリケーション・プログラムの作業域との間でデータを転送するための 1 次機構である。

HSM. OS/390 環境では、階層記憶管理機能 (Hierachical Storage Manager)。

I

ICAPI. インターネット接続 API (Internet Connection API)。

ICF. OS/390 環境における、統合カタログ機能 (integrated catalog facility)。

IDCAMS. OS/390 環境で、アクセス方式サービス・コマンドの処理に使用される IBM プログラム。このプログラムは、TSO 端末またはユーザー・アプリケーション・プログラム内から、ジョブもしくはジョブ・ステップとして呼び出すことができる。

IDCAMS LISTCAT. OS/390 環境では、アクセス方式サービス・カタログに含まれる情報を入手するための機能。

識別する (identify). DB2 UDB (OS/390 版) とは別のアドレス空間の接続サービス・プログラムが、MVS サブシステム・インターフェース経由で出し、DB2 UDB (OS/390 版) にその存在を知らせ、 DB2 UDB (OS/390 版) に接続するプロセスを開始する要求。

IFCID. DB2 UDB (OS/390 版) では、計測機能構成要素識別子 (Instrumentation facility component identifier)。

IFI. DB2 UDB (OS/390 版) では、計測機能インターフェース (Instrumentation facility interface)。

IFI 呼び出し (IFI call). DB2 UDB (OS/390 版) では、定義した関数の 1 つによる、計測機能インターフェース (IFI) の呼び出し。

IFP. OS/390 環境では、IMS 高速パス (IMS Fast Path)。

ILU. 「*独立論理装置* (independent logical unit)」を参照。

イメージ・コピー (image copy). 表スペースの全部または一部の精密な複製。 DB2 UDB (OS/390 版) は、全体イメージ・コピー (表スペース全体のコピー) または増分イメージ・コピー (最後のイメージ・コピー以降に変更されたページ分のみのコピー) を作成するユーティリティー・プログラムを提供する。

暗黙特権 (implicit privilege). オブジェクトの所有権 (たとえば所有している同義語を除去する特権)、あるいは権限の保持 (たとえばユーティリティー・ジョブを終了する SYSADM 権限) を伴う特権。

インポート (import). PC/IXF、DEL、WSF、または ASC などの形式を使って、外部ファイルからデータベース・マネージャー表に外部ファイルを複写すること。「エクスポート (export)」と対比。

インポート・メタデータ (import metadata). (ユーザー・インターフェースから) 動的に、またはバッチによって、メタデータをデータウェアハウスセンターに取り入れるプロセス。

インポート・ユーティリティー (import utility). ユーザー提供のレコード・データを表に挿入するトランザクション・ユーティリティー。「*ロード・ユーティリティー* (load utility)」と対比。

IMS. 情報管理システム (Information Management System)。

IMS 接続機能 (IMS attachment facility). OS/390 サブシステム・インターフェース (SSI) プロトコルおよび仮想記憶間連係を用いて、IMS から DB2 UDB (OS/390 版) への要求を処理し、資源コミットを調整する、DB2 UDB (OS/390 版) 副構成要素。

IMS DB. 情報管理システム・データベース (Information Management System Database)。

IMS TM. 情報管理システム・トランザクション・マネージャー (Information Management System Transaction Manager)。

打ち切り中 (in-abort). 回復単位の状況の 1 つ。回復単位がロールバックを開始した後、プロセスが完了する前に DB2 UDB (OS/390 版) が失敗した場合、再始動中に、DB2 UDB (OS/390 版) はその変更のバックアウトを継続する。

コミット中 (in-commit). 回復単位の状況の 1 つ。 DB2 UDB (OS/390 版) は、その 2 フェーズ・コミット処理の開始後に失敗すると、再始動したとき、データに加えられた変更が整合していることを『認識する』。

追加バインド (incremental bind). SQL ステートメントがアプリケーション・プロセスの実行時にバインドされるプロセス。これは、SQL ステートメントがバインド・プロセス時にバインドされることができず、 VALIDATE(RUN) が指定されていたためである。「*バインド* (bind)」も参照。

独立 (independent). DB2 UDB (OS/390 版) では、オブジェクト (行、表、もしくは表スペース) は、別のオブジェクトの親や従属のどちらでもなければ独立である。

独立論理装置 (ILU) (independent logical unit, ILU). システム・サービス制御点 (SSCP) の補助がなくても、 LU 対 LU セッションを活動化できる論理装置。 ILU は、SSCP-to-LU セッションをもたない。「*従属論理装置* (dependent logical unit)」と対比。

索引 (index). キーの値に基づいて論理的に順序付けされている、一組のポインター。索引は、データに迅速にアクセスするのに使われ、表内の行を固有化することができる。

索引ファイル (index file). ビデオ・エクステンダーがショット やビデオ・クリップの個々のフレームを検出するときに使用する索引情報が入っているファイル。

索引キー (index key). 索引項目の順序を決定するために使用される、表内の列のセット。

索引区分 (index partition). 特定のノードにある表区分に関連した索引の部分。表に定義された索引は、表区分につき 1 つずつ、複数の索引区分によって実施される。

索引検索引き数述部 (index sargable predicates). SQL 要求を修飾するための索引項目数を減らすため、索引葉ページにおいて索引項目に対して適用される SQL 述部。これによって、アクセスされるデータ行の数が減少する。

索引スペース (index space). DB2 UDB (OS/390 版) では、1 つの索引の項目を保管するときに使用するページ・セット。

索引の指定 (index specification). 連合データベース・システムにおいて、あるデータ・ソース表に関連のあるメタデータのセット。このメタデータは、通常の索引定義に含まれる情報 (たとえば、情報をすばやく取り出すにはどの列を検索すべきか) で構成される。表に索引がない場合、または連合サーバーが認識できない索引の場合は、ユーザーがこのメタデータを連合サーバーに提供することができる。このメタデータの目的は、表のデータの検索機能を提供することである。

標識列 (indicator column). DB2 UDB (OS/390 版) では、基礎表内の LOB 列の 1 つに保管される 4 バイトの値のこと。

標識変数 (indicator variable). アプリケーション・プログラムで NULL 値を表すために使用される変数。選択された列の値が NULL であると、負の値が標識変数に入れられる。

未確定 (indoubt). 回復単位の状況の 1 つ。 DB2 UDB (OS/390 版) がそのフェーズ 1 のコミット処理の終了後、フェーズ 2 を開始する前に失敗した場合、この回復単位をコミットするか、ロールバックするかが分かるのは、コミット調整プログラムのみである。緊急再始動時に、DB2 UDB (OS/390 版) がその決定に必要な情報を持っていない場合、 DB2 UDB (OS/390 版) が調整プログラムからその情報を得るまで、その回復単位は*未確定* である。再始動時に、未確定の回復単位が複数存在することがある。

未確定解決 (indoubt resolution). DB2 UDB (OS/390 版) では、未確定の論理作業単位の状況を、コミット状態またはロールバック状態に変える処理。

未確定トランザクション (indoubt transaction). 2 フェーズ・コミットの片方のフェーズは正常に完了したにもかかわらず、次のフェーズが完了する前にシステムに障害が起きたトランザクション。

未完了 (inflight). 回復単位の状況の 1 つ。 DB2 UDB (OS/390 版) は、その回復単位がコミット処理のフェーズ 1 を完了する前に失敗した場合、再始動時にその回復単位の更新を単にバックアウトする。これらの回復単位は、*未完了* と呼ばれる。

情報カタログ (information catalog). 情報カタログ・マネージャーによって管理されるデータベース。これは、組織内で入手可能なデータや情報をユーザーが識別および検索するのに役立つ記述データ (ビジネス・メタデータ) を含んでいる。情報カタログには、何らかのテクニカル・メタデータ も含まれる。

情報カタログ・マネージャー (Information Catalog Manager). ビジネス情報の編成、保守、検索、および使用のためのアプリケーション。

継承 (inheritance). 親クラスから、クラス階層を下って子クラスにクラス資源や属性を渡すこと。

初期化全選択 (initialization fullselect). ソース表から初期値の直接子を入手する再帰的共通表式内の最初の全選択。

内部結合 (inner join). 結合しようとするすべての表に共通しているわけではない列を、処理結果の表から消去するような結合方式。「*外部結合* (outer join)」と対比。

作動不能パッケージ (inoperative package). 従属先の関数が消去されているために、使用できないパッケージ。このようなパッケージは明示的に再バインドする必要がある。「*無効パッケージ* (invalid package)」と対比。

作動不能トリガー (inoperative trigger). 除去または作動不能にされたオブジェクトや、失効した特権に依存するトリガー。

作動不能視点 (inoperative view). 以下のいずれかの状態になったため、もはや使えなくなった視点。

- 視点が従属する表または視点に対する SELECT 特権が、その視点の定義元から取り消された。
- 視点定義が従属するオブジェクトが消去された (または、別の視点の場合は作動不能にされたと考えられる)。

挿入規則 (insert rule). 行を表に挿入するためには従う必要のある、データベース・マネージャーで実施されている条件。

挿入トリガー (insert trigger). DB2 UDB (OS/390 版) では、トリガー用の SQL 操作 INSERT を指定して定義したトリガーのこと。

インストール (install). OS/390 サブシステムとして操作するため DB2 UDB (OS/390 版) サブシステムを準備するプロセス。

インストール検査シナリオ (installation verification scenario). メイン DB2 UDB (OS/390 版) 機能を実行し、 DB2 UDB (OS/390 版) が正しくインストールされたかどうかをテストする一連の操作。

インスタンス (instance). (1) 「データベース・マネージャー・インスタンス (database manager instance)」を参照。 (2) 論理的な DB2 エクステンダーのサーバー環境。同一のワークステーション上に DB2 エクステンダー・サーバーのインスタンスを複数持つことができるが、個々の DB2 インスタンスには 1 つのインスタンスしか持てない。

計測機能構成要素識別子 (IFCID) (instrumentation facility component identifier, IFCID). DB2 UDB (OS/390 版) で、追跡可能なイベントの追跡レコードに名前を付けて識別する値。 START TRACE および MODIFY TRACE コマンドのパラメーターの場合、対応するイベントのトレースを指定する。

計測機能インターフェース (IFI) (instrumentation facility interface, IFI). DB2 UDB (OS/390 版) に関するオンライン・トレース・データの入手、DB2 UDB (OS/390 版) コマンドの実行依頼、および DB2 UDB (OS/390 版) へのデータ送信を可能にするプログラミング・インターフェース。

対話式システム生産性向上機能 (ISPF) (Interactive System Productivity Facility, ISPF). OS/390 環境では、対話式のダイアログ・サービスを提供する IBM ライセンス・プログラム。

DB2 間 R/W インタレスト (inter-DB2 R/W interest). DB2 UDB (OS/390 版) では、データ共用グループの複数のメンバーによってオープンされ、そのうち少なくとも 1 メンバーが書き込みのためにオープンしているような、表スペース、索引、または区分の中のデータの特性。

中間ネットワーク・ノード (intermediate network node). APPN で、ルートの起点論理装置 (OLU) と宛先論理装置 (DLU) の間にあるノード。ただし、OLU も DLU も含まず、また OLU または DLU のどちらにもネットワーク・サーバーとしてサービスを提供しない。

内部 CCD 表 (internal CCD table). 直接にはサブスクリプションできない CCD 表。レジスター表にはその独自の行が存在しない。関連した複製ソースの行では、CCD_OWNER および CCD_TABLE として参照される。「**外部 CCD 表** (external CCD table)」と対比。

内部資源ロック管理プログラム (IRLM) (internal resource lock manager, IRLM). OS/390 環境では、DB2 UDB (OS/390 版) が使用する、通信およびデータベース・ロッキングを制御するサブシステム。

インターネット・プロトコル (IP) (Internet Protocol, IP). インターネット環境でソースから宛先へデータを経路指定するのに使われるプロトコル。

Internetwork Packet Exchange (IPX). 無接続データグラム・プロトコル。リモート・ノードにデータを転送するために NetWare LAN 環境で使われる。 IPX によってデータ・パケットの送信が試みられるが、そのデータが確実に送達するという保証はない。

区画間並行処理 (inter-partition parallelism). 区分データベースの複数の区画間で同時に複数のデータベース操作 (索引作成、データベース・ロード、および SQL 照会など) を実行する能力のこと。「*区画内並行* (intra-partition parallelism)」と対比。

プロセス間通信 (IPC) (Inter-Process Communication, IPC). プロセスが相互に通信するための、オペレーティング・システムの機構。

インターバル・タイミング (interval timing). DB2 複製では、サブスクリプション・サイクルを開始するタイミングを制御する最も簡単な方法。ユーザーは、サブスクリプション・サイクルを開始する日付と時刻を指定し、サブスクリプション・サイクルを実行する頻度を示す時間間隔を設定する必要がある。「**イベント・タイミング** (event timing)」および「**オンデマンド・タイミング** (on-demand timing)」と対比。

区画内並行処理 (intra-partition parallelism). 単一のデータベース区画内で同時に複数のデータベース操作 (索引作成、データベース・ロード、および SQL 照会など) を実行する能力のこと。「*区画間並行* (inter-partition parallelism)」と対比。

内部照会並行 (intra-query parallelism). 区画内並行または区画間並行のどちらか、あるいはその両方を使用して、同時に単一照会の複数部分を処理すること。

無効パッケージ (invalid package). 従属先のオブジェクトが消去されると無効になるパッケージ。 (オブジェクトは、索引などの、関数以外のタイプのもの。) このようなパッケージは、呼び出し時に暗黙的に再バインドされる。「*作動不能パッケージ* (inoperative package)」と対比。

非可変文字セット (invariant character set). DB2 UDB (OS/390 版) で、(1) 構文文字セットのように、別のコード・ページに移ってもそのコード・ポイント割り当てが変わらない文字セット。 (2) すべての文字セットの一部として使用可能な最小限の文字のセット。

入出力並行処理 (I/O parallelism). 「*並列入出力* (parallel I/O)」を参照。

IP. 「*インターネット・プロトコル* (Internet Protocol)」を参照。

IP アドレス (IP address). 一意に TCP/IP ホストを識別する 4 バイトの値。

IPX. Internetwork Packet Exchange。

IRLM. DB2 UDB (OS/390 版) では、内部資源ロック管理機能 (Internal Resource Lock Manager)。

ISAPI. Microsoft® Internet Server API.

分離レベル (isolation level). アプリケーション・プログラムを、現在実行されている他のアプリケーション・プロセスから分離する程度を定義する属性。

ISPF. OS/390 環境では、対話式システム生産性向上機能 (Interactive System Productivity Facility)。

ISPF/PDF. OS/390 環境では、対話式システム生産性向上機能 / プログラム開発機能 (Interactive System Productivity Facility/Program Development Facility)。

J

JCL. 「*ジョブ制御言語* (job control language)」を参照。

JES. 「ジョブ入力サブシステム (Job Entry Subsystem)」を参照。

ジョブ制御言語 (JCL) (Job control language, JCL). ジョブをオペレーティング・システムに識別させ、ジョブの要件を示すために使われる制御言語。

ジョブ入力サブシステム (JES) (Job Entry Subsystem, JES). ジョブをシステム内に取り入れて、ジョブの生成するすべての出力データを処理する IBM ライセンス・プログラム。

ジョブ・スケジューラー (job scheduler). データベース・ジョブを実行および管理する特定のタスクを自動化するのに使われるプログラム。

結合 (join). 列値を突き合わせて複数の表からデータを取り出すための SQL 関係操作。

K

キー (key). 表、索引、または参照制約の記述内で識別される列または順番に並べた列の集合。

キー順データ・セット (KSDS) (key-sequenced data set, KSDS). OS/390 環境で、レコードがキー順にロードされ索引によって制御される VSAM ファイルまたはデータ・セット。

キー値ベースの区分化法 (key-value based partitioning strategy). 表内の行をデータベース区画に割り当てる方法。区分化キー列の値に基づいて行が割り当てられる。

キーワード (keyword). (1) コンピューター、コマンド言語、またはアプリケーションの事前定義語の 1 つ。 (2) SQL ステートメントで使用されたオプションを識別する名前。

KSDS. 「*キー順データ・セット* (key-sequenced data set)」を参照。

L

ラベル付き期間 (labeled duration). 年、月、日、時、分、秒、またはマイクロ秒の期間を表す数値。

Language Environment®. Net.Data マクロから外部データ・ソース (たとえば DB2)、またはプログラミング言語 (たとえば Perl) へのアクセスを提供するモジュール。

ラージ・オブジェクト (LOB) (large object, LOB). 2 ギガバイトまでの長さのバイト・シーケンス。BLOB (2 進数)、CLOB (1 バイト文字または混合文字)、または DBCLOB (2 バイト文字) の 3 つのタイプのどれでもよい。

ラッチ (latch). 並行イベントまたはシステム資源の使用を制御するための DB2 UDB (OS/390 版) 内部メカニズム。

LCID. OS/390 環境では、ログ制御間隔定義 (Log control interval definition)。

LDS. 「*線形データ・セット* (linear data set)」を参照。

葉ページ (leaf page). DB2 UDB (OS/390 版) では、キーと RID のペアを含み、実際のデータを指すページ。「*非葉ページ* (nonleaf page)」と対比。

左方外部結合 (left outer join). DB2 UDB (OS/390 版) では、結合中の両方の表の一致した行が入っていて、最初の表の不一致行を保存する、結合操作の結果。「*結合* (join)」および「*右方外部結合* (right outer join)」を参照。

長さ属性 (length attribute). ストリングの宣言された固定長または最大長を表す、ストリングに関連した値。

LEN ノード (LEN node). 「*ローエントリー・ネットワーク・ノード* (low-entry networking node)」を参照。

線形データ・セット (LDS) (linear data set, LDS). OS/390 環境では、データを含むが、制御情報は含まない、VSAM データ・セット。仮想記憶域において線形データ・セットは、バイト・アドレス可能ストリングとしてアクセスできる。

リンケージ・エディター (linkage editor). モジュール間の相互参照を決定し、必要に応じてアドレスを調整することによって、 1 つまたは複数のオブジェクト・モジュールやロード・モジュールからロード・モジュールを作成するコンピューター・プログラム。

連係編集 (link-edit). DB2 UDB (OS/390 版) では、リンケージ・エディターを用いてロード可能なコンピューター・プログラムを作成すること。

リスト事前取り出し (list prefetch). データに逐次アクセスしない照会においても事前取り出しの利点が活かされるアクセス方式。そのため、データ・ページにアクセスする前に索引が走査されて RID が収集される。次にそれらの RID は記憶され、このリストを使ってデータが事前取り出しされる。

リスト構造 (list structure). OS/390 環境では、結合機能構造。これにより、データを共用し、待ち行列の要素として操作することができる。

Live Connection. 接続マネージャーおよび複数のクライエットからなる Net.Data 構成要素。 Live Connection は、データベースの再利用、および Java® 仮想マシンの接続を管理する。

L ロック (L-lock). 「*論理ロック* (logical lock)」を参照。

ロード・コピー (load copy). 事前にロードされているデータのバックアップ・イメージであり、ロールフォワード回復時に復元することができる。

ロード・モジュール (load module). 実行のために主記憶域にロードするのに適したプログラム単位。リンケージ・エディターの出力。

ロード・ユーティリティー (load utility). 表データのブロック更新を行う非トランザクション・ユーティリティー。「*インポート・ユーティリティー* (import utility)」と対比。

LOB. 「*ラージ・オブジェクト* (large object)」を参照。

LOB ロケーター (LOB locator). アプリケーション・プログラムがデータベース・システム内のラージ・オブジェクト (LOB) 値を操作できるようにするメカニズム。 LOB ロケーターは、単一の LOB 値を表す単純なトークン値である。 アプリケーション・プログラムは、LOB ロケーターをホスト変数に取り込んでから、それに関連した LOB 値に対してロケーターを介して SQL 関数を適用する。

LOB ロック (LOB lock). DB2 UDB (OS/390 版) で、LOB 値のロック。

LOB 表スペース (LOB table space). DB2 UDB (OS/390 版) では、関連する基礎表内の特定の LOB 列に関するすべてのデータが入っている表スペース。

ローカル (local). ローカル・サブシステムが保守する任意のオブジェクトを参照する方法。 DB2 UDB (OS/390 版) では、たとえば、ローカル表は、ローカル DB2 サブシステムによって維持されている表である。「リモート (remote)」と対比。

ローカル・データベース (local database). 使用中のワークステーションに物理的に存在するデータベース。「リモート・データベース (remote database)」と対比。

ローカル・データベース・ディレクトリー (local database directory). データベースが物理的に常駐するディレクトリー。ローカル・データベース・ディレクトリーに表示されるデータベースは、システム・データベース・ディレクトリーと同じノードに位置する。

ロケール (locale). DB2 UDB (OS/390 版) では、ユーザー環境のサブセットの定義のこと。特定の言語用および国別に定義されている文字と CCSID とを結合したものである。

ローカル・ロック (local lock). DB2 内の並行性制御を提供するが、DB2 間の並行性制御は提供しないロック。すなわち、その効力範囲は単一の DB2 UDB (OS/390 版) である。

ローカル・サブシステム (local subsystem). ユーザーもしくはアプリケーション・プログラムが直接 (DB2 UDB (OS/390 版) の場合は、 DB2 UDB (OS/390 版) 接続機能の 1 つにより) 接続する固有の RDBMS。

ローカル表ロック (local table lock). 単一データベース区画においてのみ獲得される表ロック。

ローカル更新 (local update). レプリカにではなく、基礎表に対する更新。

ロケーション名 (location name). DB2 UDB (OS/390 版) が、サブシステムのネットワーク内で、特定の DB2 サブシステムを参照するときに使用する名前。「*LU 名* (LU name)」と対比。

ロケーション・パス (location path). XPath によって定義される、ロケーション・パスの省略された構文から成るサブセット。 XML 要素または属性を識別する、一連の XML タグ。これは、ユーザー定義の関数の抽出の際に抽出する件名の識別のために使用され、またテキスト・エクステンダーのユーザー定義の関数検索機能においては、検索基準の識別に使用される。

ロケーター (locator). 「*LOB ロケーター* (LOB locator)」を参照。

ロック (lock). (1) イベントまたはデータへのアクセスの逐次化のための手段。 (2) あるアプリケーション・プロセスが加えた未コミットの変更が、別のアプリケーション・プロセスで認識されないようにしたり、アプリケーション・プロセスが、別のプロセスからアクセスされているデータを変更しないようにしたりするための手段。 (3) 並行イベントまたはデータへのアクセスを制御する手段。 DB2 UDB (OS/390 版) のロッキングは、IRLM により行われる。

ロック期間 (lock duration). DB2 UDB (OS/390 版) ロックを保持する時間間隔。

ロック調整 (lock escalation). データベース・マネージャーでは、あるエージェント用に発行されたロック数が、データベース構成内に指定された上限を超えたときにとられる措置。この上限値は、MAXLOCKS 構成パラメーターで定義される。ロック・エスカレーション時には、表の行に対するロックを、表に対する 1 つのロックに変換することによってロックが解放される。これは、上限値をもう超えなくなるまで繰り返される。

ロッキング (locking). データの整合性を確保するためにデータベース・マネージャーで使われる機構。ロッキングにより、複数の並行ユーザーが非整合データにアクセスできないようにする。

ロック・モード (lock mode). DB2 UDB (OS/390 版) ロックが保留する資源に、並行して実行中のプログラムが持てるアクセスのタイプに関する表示。

ロック対象 (lock object). DB2 UDB (OS/390 版) ロックによって制御される資源。

ロック親 (lock parent). DB2 UDB (OS/390 版) の明示の階層ロッキングの場合に、資源上に保留されていて、その階層の下の方に位置する子ロックをもつロックを指す。通常、表スペースや区分の意図ロックは親ロックである。

ロックの格上げ (lock promotion). DB2 UDB (OS/390 版) ロックのサイズまたはモードを上のレベルに変更するプロセス。

ロック・サイズ (lock size). 表データに対する DB2 UDB (OS/390 版) ロックによって制御されるデータの量。その値として、行、ページ、LOB、区分、表、または表スペースを取ることができる。

ロック構造 (lock structure). DB2 UDB (OS/390 版) では、結合機能データ構造の 1 つ。一連のロック項目から成り、論理資源の共用ロッキングおよび排他ロッキングをサポートする。

ログ (log). (1) システムで行われた変更を記録するのに使われるファイル。 (2) DB2 UDB (OS/390 版) 実行中に発生するイベントおよびその順序を記述する、レコードの集合。このように記録された情報は、DB2 UDB (OS/390 版) 実行中に障害が起きた場合の回復に使用される。 (3) 「データベース・ログ (database log)」を参照。

ログ先頭 (log head). アクティブ・ログ内で一番先に書き込まれたログ・レコード。

論理クレーム (logical claim). DB2 UDB (OS/390 版) では、非区分索引の論理区分のクレーム。

論理ドレーン (logical drain). DB2 UDB (OS/390 版) では、非区分索引の論理区分のドレーン。

論理索引区分 (logical index partition). DB2 UDB (OS/390 版) では、同じデータ区分を参照するすべてのキーの集合。

論理ロック (L-Lock) (logical lock, L-lock). DB2 UDB (OS/390 版) では、トランザクション間で DB2 内および DB2 間のデータ並行性を制御するためにトランザクションが使用するロック・タイプ。「*物理ロック* (physical lock)」と対比。

論理ノード (logical node). プロセッサーに複数のノードが割り当てられている場合の、1 つのノード。「ノード (node)」も参照。

論理演算子 (logical operator). 複数の検索条件の評価の仕方 (AND、OR) や、検索条件の論理的な意味を逆転する (NOT) かどうかを指定するキーワード。

論理ページ・リスト (LPL) (logical page list, LPL). DB2 UDB (OS/390 版) では、エラーが生じたページのリスト。ページが回復するまでアプリケーションによって参照できない。実際のメディア (結合機能 または DASD) にはエラーが発生していないことがあるため、ページは論理エラーの状態にある。通常、メディアへの接続は失われている。

論理区画 (logical partition). DB2 UDB (OS/390 版) では、特定の区分と関連付けられている非区分索引内のキーまたは RID のペアの集合。

論理回復保留 (LRECP) (logical recovery pending, LRECP). DB2 UDB (OS/390 版) では、矛盾するデータを参照するデータと索引キーの状態。

論理装置 (LU) (logical unit, LU). (1) SNA では、エンド・ユーザーが、別のエンド・ユーザーと通信するために SNA ネットワークにアクセスするときに経由するポート。 LU は、他の LU との多数のセ

ッションをサポートすることができる。 (2) OS/390 環境では、アプリケーション・プログラムが別のアプリケーション・プログラムと通信するために SNA ネットワークにアクセスする、アクセス点。「*LU 名 (LU name)*」も参照。

論理装置 タイプ 6.2 (LU 6.2) (logical unit 6.2, LU 6.2). APPC を使ったアプリケーション相互間のセッションをサポートする LU タイプ。

論理作業単位 (LUW) (logical unit of work, LUW). プログラムが同期点間で行う処理。

論理作業単位識別子 (LUWID) (logical unit of work identifier, LUWID). OS/390 環境では、ネットワーク内のスレッドを固有に識別する名前。この名前は、完全修飾 LU ネットワーク名、LUW インスタンス番号、および LUW 順序番号から構成される。

ログ初期設定 (log initialization). DB2 UDB (OS/390 版) がログの現行の終わりの検索を試みる、再始動処理の第 1 フェーズ。

ログ区分 (log partition). 各データベース区画ごとにデータベース活動を記録するデータベース区画専用のログ・ファイル。

ログ・レコード (log record). ある作業単位時に実行された、データベースに対する更新のレコード。このレコードは、アクティブ・ログのログ末尾の後に書き込まれる。

ログ・レコード順序番号 (LRSN) (log record sequence number, LRSN). DB2 UDB (OS/390 版) が生成し、各ログ・レコードに関連付ける番号。ページのバージョン管理にも、LRSN が使用される。特定の DB2 UDB (OS/390 版) データ共用グループが生成する LRSN は、各 DB2 ログの厳密な増加順序、およびデータ共用グループ内の各ページの厳密な増加順序を形成する。

ログ・テーブル (log table). テキスト文書の索引付けに関する情報を含む、テキスト・エクステンダーが作成するテーブル。

ログ末尾 (log tail). アクティブ・ログに一番後から書き込まれたログ・レコード。

ログ切り捨て (log truncation). DB2 UDB (OS/390 版) では、RBA の明示的な開始を確立するプロセス。この RBA は、ログ・データの次のバイトが書かれるポイントである。

長ストリング (long string). (1) 最大長が 254 バイトより長い可変長ストリング。 (2) DB2 UDB (OS/390 版) では、255 バイトまたは 127 個の 2 バイト文字よりも長い、実際の長さを持つストリング、または最大長の可変長ストリング。 LOB を評価する LOB 列、LOB ホスト変数、または式はすべて長ストリングと考えられる。

長形式表スペース (long table space). 長ストリングまたはラージ・オブジェクト (LOB) データだけを記憶できる表スペース。

ローエントリー・ネットワーキング・ノード (LEN ノード) (low-entry networking node, LEN node). 非従属 LU プロトコルをサポートし、 CP to CP セッションをサポートしないタイプ 2.1 ノード。これは、サブエリア・ネットワーク内の境界ノードに接続された周辺ノード、 APPN ネットワーク内の

APPN ネットワーク・ノードに接続されたエンド・ノード、または別の LEN ノードまたは APPN エンド・ノードに直接接続された対等接続ノードのどれでもかまわない。

LPL. 「*論理ページ・リスト* (logical page list)」を参照。

LRECP. 「*論理回復保留中* (logical recovery pending)」を参照。

LRH. DB2 UDB (OS/390 版) では、ログ・レコード・ヘッダー (log record header)。

LRSN. 「*ログ・レコード順序番号* (log record sequence number)」を参照。

LU. 「*論理装置* (logical unit)」を参照。

LU 名 (LU name). OS/390 環境で、VTAM がネットワーク内のノードを参照するときに使う名前。「*ロケーション名* (location name)」と対比。

LU 6.2. 「*論理装置 6.2* (Logical unit 6.2)」を参照。

LU タイプ (LU type). SNA プロトコルの特定のサブセットと、特定のセッションについてそれがサポートするオプションに重点を置いた論理装置のクラス分類。特に以下の点が重視される。

- 値はセッション活動化要求において許可される。
- データ・ストリーム制御機能、機能管理ヘッダー、要求単位パラメーター、およびセンス・データ値の使用法。
- 表示サービス・プロトコル（機能管理ヘッダーに関連したものなど）。

LUW. 「*論理作業単位* (logical unit of work)」を参照。

LUWID. 「*論理作業単位識別子* (logical unit of work identifier)」を参照。

M

マップ式会話 (mapped conversation). APPC では、APPC マップ式会話 API を使用してトランザクション・プログラム (TP) 間で行う会話。通常、エンド・ユーザー TP はマップ式会話を使用し、サービス TP は基本会話を使用する。どちらのプログラムも両方の会話を使用できる。「*基本会話* (basic conversation)」と対比。

マスク文字 (masking character). 検索する用語の最初、最後、またはその中の任意の文字を表すために使用する文字。 索引でその用語の関連部分を検索するときに使用する。

実体化 (materialize). DB2 UDB (OS/390 版) では、(1) 表示またはネストした表式からの行を、照会によってさらに処理するために作業ファイルに入れるプロセス。

(2) LOB 値を連続する記憶域に配置すること。 LOB 値は非常に大きくなる可能性があるため、 DB2 UDB (OS/390 版) では絶対に必要となるまで LOB データの実体化を避けている。

MBCS. 「*マルチバイト文字セット* (multi-byte character set)」を参照。

メンバー (member). (1) DB2 では、「サブスクリプション・セット・メンバー (subscription-set member)」。 (2) OLAP Starter Kit では、3 つ以上のディメンションでデータを参照する方法。ファクト表内の個々のデータ値は、各ディメンションによる 1 つのメンバーの座標点である。

メンバー名 (member name). データ共用グループ中の特定の DB2 UDB (OS/390 版) サブシステムの XCF 識別子。

メンバー効力範囲 (member scope). 「*コマンド効力範囲* (command scope)」を参照。

メニュー (menu). DB2 UDB (OS/390 版) では、オペレーターが選択できる機能の表示リスト。メニューはメニュー・パネル とも呼ばれる。

メタデータ (metadata). 保管データの特性を記述するデータ、つまり記述データ。たとえば、データベース表のメタデータには、表名、その表を含むデータベースの名前、表内の列の名前、および列に関する記述が、技術的な用語またはビジネス用語で含まれる。

メタデータ・パブリケーション・プロセス (metadata publication process). データウェアハウスセンターによって作成されるプロセスの 1 つ。パブリケーション後に、公表されたメタデータと元のメタデータとの同期を保つために作成されるすべてのステップを含む。

移行 (migration). (1) データの変換をせずに、1 つのコンピューター・システムから別のコンピューター・システムにデータを移動する処理。 (2) あるプログラムの新しいバージョンまたはリリースをインストールし、以前のバージョンまたはリリースと置き換えること。 (3) 既存の DB2 UDB (OS/390 版) サブシステムを、更新リリースまたは現行リリースに変換する処理。 この処理では、前リリースで作成したデータを失うことなく、更新または現行リリースの機能を得ることができる。

混合文字ストリング (mixed-character string). 単一バイト文字およびマルチバイト文字が混在しているストリング。 「*混合データ・ストリング* (mixed data string)」とも呼ばれる。

混合データ・ストリング (mixed-data string). 「*混合文字ストリング* (mixed-character string)」を参照。

モバイル・クライアント (mobile client). 通常はラップトップ・コンピューターであるノード。モバイル環境で使用されるモバイル・イネーブラー、複製ソース表や複製ターゲット表が置かれる。 モバイル複製モードはモバイル・クライアントから起動される。

モバイル複製イネーブラー（mobile replication enabler). モバイル・クライアントでモバイル複製モードを開始する複製プログラム。

モバイル複製モード (mobile replication mode). 自律的また継続的にではなく、必要に応じて収集プログラムと変更適用プログラムが作動する、複製のモード。このモードはモバイル・クライアントから起動され、モバイル・クライアントがソース・サーバーまたはターゲット・サーバーへの接続に使用できるようになったときにデータが複製できるようになる。

モード (mode). データウェアハウスセンターで、ステップの開発段階 (たとえば開発、テスト、または実動)。

MODEENT. OS/390 環境では、ログオン・モード名を、セッション・プロトコルを示すパラメーターの集合と関連付ける VTAM マクロ命令。 MODEENT マクロ命令の集合はログオン・モード表を定義する。

モデル統計 (modeled statistics). SQL ステートメントで参照されるかどうかにかかわらず、現在 Explain モデルに存在するデータベース・オブジェクトの統計。 オブジェクトが現在データベースに存在するかどうかは関係しない。

モード名 (mode name). (1) APPC では、セッションの開始プログラムがそのセッションに適する特性を指定するときに使用する名前。たとえば、トランスポート・ネットワークのメッセージ長の制限、同期点、サービス・クラスや、セッションの経路指定、遅延特性など。 (2) OS/390 環境では、物理特性および論理特性とセッションの属性の集まりの VTAM 名。

変更ロック (modify locks). DB2 UDB (OS/390 版) で、MODIFY 属性を持つ L ロックまたは P ロック。これらの活動中のロックのリストは、常に結合機能のロック構造に保持されている。要求していたサブシステムに障害がおこった場合、サブシステムの変更ロックは保持ロックに変換される。

モニター・セッション (monitoring session). データベース・マネージャーを監視すること、またはすでに監視済みのデータベース・マネージャーからの情報を再生すること。 DB2 パフォーマンス・モニターを使用して、モニター・セッションの作成と、監視するデータベース・オブジェクトの選択を行う。

モニター・スイッチ (monitor switch). パフォーマンス・スナップショットに戻す情報のタイプと量を制御するために、ユーザーが操作するデータベース・マネージャーのパラメーター。

MPP. (1) 大量並列処理 (Massively parallel processing)。 (2) IMS の OS/390 環境では、メッセージ処理プログラム (message processing program)。

MSS. OS/390 環境では、大容量記憶サブシステム (Mass Storage Subsystem)。

MTO. OS/390 環境では、マスター端末操作員 (master terminal operator)。

マルチバイト文字セット (MBCS) (multi-byte character set, MBCS). 各文字が複数バイトである文字のセット。 2 バイトの文字のみを使用するものは、*2 バイト文字セット* と呼ばれる。

多次元 (multidimensional). OLAP Starter Kit では、3 つ以上のディメンションでデータを参照する方法。ファクト表内の個々のデータ値は、各ディメンションによる 1 つのメンバーの座標点である。

多次元データベース (multidimensional database). OLAP Starter Kit で、OLAP 分析のためにリレーショナル・データをコピーする複写先の非リレーショナル・データベース。

複数サイト更新 (multi-site update). DB2 UDB (OS/390 版) では、単一作業単位内の複数のロケーションでデータの更新が行われる、分散リレーショナル・データベースの処理。

マルチタスキング (multitasking). 複数のタスクの並行実行またはインターリーブ実行を行う操作モード。

完了必須 (must-complete). データ保全性を維持するため操作全体を完了しなければならない、 DB2 UDB (OS/390 版) 処理の状態。

MVS. 多重仮想記憶 (Multiple Virtual Storage)、OS/390 の一部。

MVS/ESA™. 多重仮想記憶 / エンタープライズ・システム体系 (Multiple Virtual Storage/Enterprise Systems Architecture)、OS/390 の一部。

N

NAU. 「*ネットワーク・アドレス可能単位* (network addressable unit)」を参照。

NDS. 「*ネットワーク・ディレクトリー・サービス* (Network Directory Services)」を参照。

交渉可能ロック (negotiable lock). DB2 UDB (OS/390 版) では、そのモードが、競合するユーザー間の同意によって、すべてのユーザーに適用できるモードに下げることが可能なロック。物理ロックは、交渉可能ロックの 1 つ。

ネストされた表の式 (nested table expression). (1) FROM 節で指定された全選択の評価によって直接または間接的にほかの 1 つ以上の表から得られた結果表。 (2) DB2 UDB (OS/390 版) では、FROM 文節の副選択項目 (括弧で囲まれている)。

NETID. ネットワーク識別子 (Network identifier)。「ネットワーク名 (network name)」を参照。

ネットワーク・アドレス (network address). ネットワーク内のノードの識別子。

ネットワーク・アドレス可能単位 (NAU) (network addressable unit, NAU). パス制御ネットワークによって伝送される情報の発信元または宛先。 NAU は、論理装置 (LU)、物理装置 (PU)、制御点 (CP)、またはシステム・サービス制御点 (SSCP) のいずれか。「ネットワーク名 (network name)」も参照。

ネットワーク・ディレクトリー・サービス (NDS) (Network Directory Services, NDS). NetWare がネットワーク上の資源に関する情報を保守し、資源へのアクセスを提供する、グローバルかつ分散、複製されているデータベース。 NetWare ディレクトリー・データベースは、物理的位置にかかわりなく、ディレクトリー・ツリーと呼ばれる階層木構造にオブジェクトを編成する。

ネットワーク ID (NID) (network identifier, NID). OS/390 環境では、IMS または CICS によって割り当てられたネットワーク ID、または接続タイプが RRSAF である場合は、OS/390 RRS 回復単位 ID (URID)。

ネットワーク名 (network name). SNA では、エンド・ユーザーがネットワーク・アドレス可能単位 (NAU)、リンク・ステーションまたはリンクを参照するときに使用する記号名。「*NETID*」の同義語。

ネットワーク・ノード (NN) (network node, NN). APPN では、分散ディレクトリー・サービス、他の APPN ネットワーク・ノードとのトポロジー・データベースの交換およびセッション経路指定サービスを提供するネットワーク上のノード。「*APPN* ネットワーク・ノード (APPN network node)」の同義語。

ネットワーク・ノード・サーバー (network node server). ローカル論理装置および隣接するエンド・ノードに対してネットワーク・サービスを提供する APPN ネットワーク・ノード。

ネットワーク修飾名 (network-qualified name). 相互接続 SNA ネットワーク内で知られている LU の名前。ネットワーク修飾名は、各サブネットワークを識別するネットワーク名およびネットワーク LU 名で構成されている。また、ネットワーク修飾名は相互接続ネットワーク内で固有な名前である。「*ネットワーク修飾 LU 名* (network-qualified LU name)」または「*完全修飾 LU 名* (fully qualified LU name)」としても知られている。

ネットワーク・サービス (network services). SSCP-SSCP、SSCP-PU、SSCP-LU および CP-CP セッション中のネットワーク操作を制御するネットワーク・アドレス可能単位のサービス。

ニックネーム (nickname). (1) データ・ソース表または視点を参照するために連合サーバーが使用する識別子。 (2) IBM 以外のデータベースの物理データベース・オブジェクト (たとえば表やストアード・プロシージャー) を表すために、 DB2 DataJoiner データベースで定義された名前。

NID. 「*ネットワーク識別子* (network identifier)」を参照。

NN. 「*ネットワーク・ノード* (network node)」を参照。

ノード (node). (1) データベース区画化で、データベース区画の同義語。 (2) ハードウェアでは、クラスター・システムまたは大量並列処理 (MPP) システムの一部である、単一プロセッサー・コンピューターまたは対称マルチプロセッサー (SMP) コンピューター。たとえば、RS/6000® SP™ は高速ネットワークによって接続されたいくつかのノードから成る MPP システムである。 (3) 通信では、通信リンクの端点またはネットワーク内の複数のリンクに共通の接合点。ノードになるのは、処理装置、通信制御装置、クラスター制御装置、端末またはワークステーション。経路指定および他の機能性は、ノードごとに異なる。

ノード・ディレクトリー (node directory). クライアント・ワークステーションからすべての適用可能なデータベース・サーバーへの接続を確立するために必要な情報があるディレクトリー。

ノードグループ (nodegroup). 1 つまたは複数のデータベース区画をまとめて名前を付けたグループ。

不完全な CCD 表 (noncomplete CCD table). DB2 複製で、作成された時に空で、ソースの内容が変更されるにつれて行が付加される CCD 表。「**完全な CCD 表** (complete CCD table)」と対比。

非圧縮属性 (noncondensed attribute). 現在のデータではなく、データの変更の活動記録を含む表であることを示す表属性。この属性を持つ表では、キー値に複数の行が組み込まれている。

非圧縮 CCD 表 (noncondensed CCD table). DB2 複製で、行の値の変更履歴を含む CCD 表。このような表は、監査に役立つ。「**圧縮された CCD 表** (condensed CCD table)」と対比。

区切りなし ASCII (ASC) 形式 (nondelimited ASCII (ASC) format). データのインポートのときに使用するファイル形式。区切りなし ASCII とは、ASCII 製品間でデータ交換を行うときに使用する行区切り文字がある順次 ASCII ファイルのこと。

非葉ページ (nonleaf page). DB2 UDB (OS/390 版) では、索引中に他のページ (葉ページ、または非葉ページ) のキーおよび番号を含んでいるページ。非葉ページは実データを指すことはない。 「葉ページ (leaf page)」と対比。

非区分化索引 (nonpartitioning index). DB2 UDB (OS/390 版) では、区分化索引でない索引。

正規化 (normalization). データベースでは、データ・モデルの関係を最も単純な形式に再構造化するプロセス。

非 deterministic 関数 (not-deterministic function). DB2 UDB (OS/390 版) では、ユーザー定義関数で、その結果が、単に入力引き数の値によるわけではないもの。同じ引き数値を使用して連続的に呼び出した場合に異なる応答を生成する可能性がある。このような関数を、*可変* 関数 (variant function) ともいう。同じ入力に対して常に同じ結果を返す「*deterministic 関数* (deterministic function)」(*非可変関数* (not-variant function) ともいう) と対比。

非分離の (not-fenced). ユーザー定義関数またはストアード・プロシージャーのタイプの 1 つで、DBMS 処理で実行するよう定義されたもの。 「*分離* (fenced)」と対比。

通知プロセス (notification process). データウェアハウスセンターによって作成されるプロセスの 1 つ。ステップ完了時の通知のために作成されるすべてのステップを含む。

非可変関数 (not-variant function). 結果が入力引き数の値に従属しているユーザー定義関数。同じ引き数値を続けて呼び出すとすべて同じ結果になる。「*可変関数* (variant function)」と対比。

NRE. OS/390 環境では、ネットワーク回復要素 (network recovery element)。

NSAPI. Netscape API。

NUL. C 言語では、ストリングの終了を表す単一の文字。

NULLIF. DB2 UDB (OS/390 版) では、引き数が等しい場合は NULL を戻し、等しくない場合は最初の引き数の値を戻して、2 つの渡された式を評価するスカラー関数。

ヌル (null). DB2 UDB (OS/390 版) では、情報が存在しないことを示す特殊な値。

ヌル可能 (nullable). 列、関数パラメーター、または結果に値がなくてもよい状態。たとえば、ミドル・ネームのイニシャルを入力するフィールドは値がなくてもよく、ヌル可能とされる。

ヌル値 (null value). 値が指定されていないパラメーターの位置。

NUL で終了するホスト変数 (NUL-terminated host variable). DB2 UDB (OS/390 版) では、データの終わりが NUL 終止符の存在によって示される可変長のホスト変数。

NUL 終止符 (NUL terminator). C 言語では、ストリングの終了を示す値。文字ストリングの場合の NUL 終止符は、X'00'。

O

OASN (起点アプリケーション・スケジュール番号) (origin application schedule number). IMS のある OS/390 環境では、IMS の最後のコールド・スタート以降の各 IMS スケジュールに順次割り当てられる、4 バイトの番号。 OASN は作業単位の識別子として使用される。 8 バイト形式では、先頭の 4 バイトにはスケジュール番号が入り、後の 4 バイトには現行スケジュール時の IMS 同期点 (コミット・ポイント (commit points)) の数が入る。 OASN は、IMS 接続のための NID の一部である。

OBID. DB2 UDB (OS/390 版) では、データ・オブジェクト識別子 (Data object identifier)。

オブジェクト (object). (1) SQL で作成または操作できるもの。たとえば、表、視点、索引、パッケージなど。 (2) オブジェクト指向設計またはオブジェクト指向プログラミングでは、データとそのデータに関連付けられた操作からなる抽象的なもの。 (3) NetWare では、ネットワーク上で定義され、ファイル・サーバーへのアクセスが与えられているエンティティー。

オブジェクト特性 (object property). オブジェクトに関連する情報のカテゴリーを識別する特性。 NetWare バインダリー・オブジェクトを、1 つ以上の特性に割り当てることができる。 DB2 サーバー・インスタンス・オブジェクトにはオブジェクト特性 NET_ADDR があり、オブジェクト内のレコードのロケーションを表示する。

オブジェクト・タイプ (object type). (1) NetWare ファイル・サーバーのバインダリー内のオブジェクトを分類する 2 バイトの数値。 062B は DB2 データベース・サーバーのオブジェクト・タイプを表す。 (2) 似たような性質および特性を共用するオブジェクト・インスタンスのカテゴリー化またはグループ化。

ODBC. 「オープン・データベース・コネクティビティー (Open Database Connectivity)」を参照。

ODBC ドライバー (ODBC driver). ODBC 関数呼び出しを実行し、データ・ソースと対話するドライバー。

オフライン・バックアップ (offline backup). データベースまたは表スペースがアプリケーションによってアクセスされていないときに作成されたデータベースまたは表スペースのバックアップ。バックアップ・データベース・ユーティリティーは、バックアップが完了するまでデータベースを排他的に使用する権限を持つ。「オンライン・バックアップ (online backup)」と対比。

オフライン復元 (offline restore). データベースまたは表スペースの、バックアップからのコピーによる復元。バックアップ・データベース・ユーティリティーは、復元が完了するまでデータベースを排他的に使用する権限を持つ。「オンライン復元 (online restore)」と対比。

OLAP. 「*オンライン分析処理* (online analytical processing)」を参照。

オンデマンド・タイミング (on-demand timing). 断続的に接続されるシステムの複製のタイミングを制御する方式。収集プログラムおよび変更適用プログラムを操作するために、ASNSAT プログラムの使用が必須である。「**イベント・タイミング** (event timing)」および「**インターバル・タイミング** (interval timing)」と対比。

オンライン分析処理 (OLAP) (online analytical processing, OLAP). OLAP Starter Kit で、多次元、複数ユーザーのクライアント・サーバー・コンピューティング環境。企業の統合データをリアルタイムで分析する必要のあるユーザー向け。

オンライン・バックアップ (online backup). データベースまたは表スペースが他のアプリケーションによってアクセスされているときに作成されるデータベースまたは表スペースのバックアップ。「オフライン・バックアップ (offline backup)」と対比。

オンライン・モニター (online monitor). 「パフォーマンス・モニター *(Performance Monitor)*」を参照。

オンライン復元 (online restore). データベースまたは表スペースが他のアプリケーションによってアクセスされているときに行われる、バックアップからのコピーによる復元。「オフライン復元 (offline restore)」と対比。

オープン・データベース・コネクティビティー (ODBC) (Open Database Connectivity, ODBC). 呼び出し可能 SQL (SQL プリプロセッサーを必要としない) を使用して、データベース管理システムにアクセスできるようにする API。 ODBC アーキテクチャーを使用すると、データベース・ドライバー (database driver) と呼ばれるモジュールを追加することができる。これらのモジュールはアプリケーションを、選択したデータベース管理システムに実行時にリンクする。アプリケーションがサポートされるすべてのデータベース管理システムのモジュールに直接的にリンクされる必要はない。

オペランド (operand). 操作が実行されるエンティティー。

最適化 SQL テキスト (optimized SQL text). Explain 機能で作成された SQL テキストで、アクセス・プランを選択するときに実際に最適化プログラムが行う照会に基づいているもの。この照会は、ステートメント・コンパイル時に、 SQL コンパイラーのさまざまな構成要素によって補足され、再記述される。テキストは初期表示から再構造されるため、元の SQL テキストとは異なる。最適化ステートメントは元のステートメントと同じ結果を生成する。

最適化プログラム (optimizer). 代替アクセス・プランの実行コストをモデル化し、見積コストが最低のアクセス・プランを選択することによって、データ操作言語ステートメント (DML) のアクセス・プランを選択する、 SQL コンパイラーの構成要素。

通常識別子 (ordinary identifier). (1) SQL では、名前を作成するために使用される文字。英字 (a〜z および A〜Z) の後に、ゼロ個以上の英字、記号、数字、または下線文字が続く。 (2) DB2 UDB (OS/390 版) では、英大文字 の後にゼロ個以上の文字が続く。それぞれの文字は英大文字、数字、または下線文字である。通常識別子は予約語であってはならない。

通常トークン (ordinary token). 数値定数、通常識別子、ホスト識別子、またはキーワード。

親タスク (originating task). DB2 UDB (OS/390 版) では、並列グループにおいて、照会の一部を並列で実行している他の実行単位 (*並列タスク* と呼ばれる) からデータを受信する 1 次エージェント。

外部結合 (outer join). (1) 結合された表すべてに共通ではない列が結果表の一部になる場合の結合方式。「*内部結合* (inner join)」と対比。 (2) DB2 UDB (OS/390 版) では、結合中の両方の表の一致した行が入っていて、結合中の表の不一致の行の一部または全部を保存する、結合操作の結果。「*結合* (join)」も参照。

アウトライン (outline). OLAP Starter Kit で、データベースのすべての要素を OLAP Starter Kit 内で定義する構造。たとえば、アウトラインはディメンション、メンバー、および公式の定義を含んでいる。

出力ファイル (output file). レコードの書き込みを認めるオプション付きでオープンできるデータベースまたは装置ファイル。

オーバーフロー・レコード (overflow record). (1) 間接的にアドレス指定されたファイルでは、全トラックのアドレスまたはホーム・レコードのアドレスにランダム化されたキーを持つレコード。 (2) DB2 では、現在保管されているページには大きすぎて入りきらない更新済みレコード。レコードは別のページにコピーされ、元の位置は新しい位置のポインターと置き換えられる。 (3) データベース・モニターでは、事象モニター・データ・ストリームに挿入されたレコード。このレコードは、名前付きパイプがいっぱいで時間内にレコードが処理されなかったためレコードが廃棄されたことを示す。オーバーフロー・レコードはまた、廃棄されたレコード数も示す。

オーバーロード関数名 (overloaded function name). 関数パスまたはスキーマ内にいくつもの関数がある関数名。同じスキーマ内の関数は、異なるシグニチャーを持っていなければならない。

P

パッケージ (package). SQL ステートメントを実行するために使用されるプログラム準備中に作成される制御構造。

パッケージ・リスト (package list). DB2 UDB (OS/390 版) では、アプリケーション・プランを拡張する際に使用できる、パッケージ名の順序付きリスト。

パッケージ名 (package name). DB2 UDB (OS/390 版) では、BIND PACKAGE または REBIND PACKAGE コマンドによって作成されたオブジェクトの名前。オブジェクトは、データベース要求モジュール (DBRM) のバインド済みバージョンである。この名前は、ロケーション名、コレクション ID、パッケージ ID、およびバージョン ID から構成される。

パケット (packet). データ通信では、2 進数字の列。複合体全体として送信され切り替えられるデータおよび制御信号を含んでいる。

ページ (page). (1) 表または索引内の記憶域のブロックで、4096 バイト (4 KB)。 (2) DB2 UDB (OS/390 版) では、表スペース内または索引スペース内の記憶域の単位。そのサイズは、表スペースの場合は 4 K バイト、8 K バイト、16 K バイト、または 32 K バイトであり、索引スペースの場合は 4 K バイトである。表スペース内においては、1 つのページには表の行が 1 つまたは複数入っている。 LOB 表スペースでは、1 つの LOB 値が複数のページにまたがる可能性があるが、 1 ページには 1 つの LOB 値しか保管されない。

ページ・セット (page set). OS/390 環境では、表スペースあるいは索引スペースを参照する別の方法。各ページ・セットは、VSAM データ・セットの集合から構成される。

ページ・セット回復保留 (PSRCP) (page set recovery pending, PSRCP). DB2 UDB (OS/390 版) では、索引スペースの限定状態の 1 つ。そこではページ・セットすべてが回復される。論理部分の回復は禁じられている。

パネル (panel). DB2 UDB (OS/390 版) では、表示面で表示フィールドの位置および特性を定義する、事前定義された表示イメージ (たとえば、メニュー・パネル)。

並列グループ (parallel group). OS/390 環境では、同じ数の並列タスクが含まれている連続操作の集合で、並列に実行される。

並列入出力 (parallel I/O). 応答時間を短縮するために、同時に複数の入出力装置に対して読み取りまたは書き込みを実行するプロセス。

並列入出力処理 (parallel I/O processing). DB2 UDB (OS/390 版) が単一ユーザーの照会に対して複数の並行要求を開始し、複数のデータ区分で同時に (並列して) 入出力処理を行う、入出力処理形式の 1 つ。

並列処理 (parallelism). 同時に (並列して) 複数のデータベース操作を実行する能力。「*区画間並行* (inter-partition parallelism)」、「*区画内並行* (intra-partition parallelism)」、または「*並列入出力* (parallel I/O)」を参照。

並列セッション (parallel session). SNA で、同じ論理装置 2 台の間で同時に活動中の 2 つ以上のセッション。各セッションは、異なるセッション・パラメーターを持つことができる。「セッション (session)」を参照。

並列シスプレックス (Parallel Sysplex). 特定の多重システム・ハードウェア構成要素およびソフトウェア・サービスを介して相互に通信および調整を行う、一連の複数の OS/390 システム。

並列タスク (parallel task). OS/390 環境では、照会を並列して行うために動的に作成される実行単位。MVS サービス要求ブロックによって実行される。

パラメーター化データ・タイプ (parameterized data type). 任意の長さ、位取りまたは精度に定義できるデータ・タイプ。ストリングまたは 10 進データ・タイプがパラメーター化される。

パラメーター・マーカー (parameter marker). 動的 SQL ステートメントのストリングにある疑問符 (?)。疑問符は、ステートメント・ストリングが静的 SQL ステートメントの場合にホスト変数が表示されるところに現れる。

親キー (parent key). 参照制約で使用される基本キーまたは固有キー。親キーの値は、制約内の外部キーの有効値を判別する。

親行 (parent row). 少なくとも 1 つの従属行を持つ行。

親表 (parent table). 少なくとも 1 つの参照制約で親である表。

親表スペース (parent table space). DB2 UDB (OS/390 版) では、親表が入っている表スペース。その表の従属が入っている表スペースは、従属表スペースである。

参加プログラム (participant). OS/390 環境では、コミット・プロセスに参加する、コミット調整プログラム以外のエンティティー。 SNA における「エージェント (agent)」の同義語。

区分、区画 (partition). OS/390 環境では、ページ・セットの一部分。各区分は、単一の、独立して拡張できるデータ・セットに対応する。区分のサイズは、区分ページ・セット内の区分の数に応じて、最大 1、2、または 4 ギガバイトに拡張できる。 1 つのページ・セットのすべての区分は同じ最大サイズを持つ。

区分互換性結合 (partition compatible join). 結合されたすべての行が同じデータベース区画に置かれる結合。

区分データベース (partitioned database). 複数のデータベース区画をもつデータベース。ユーザー表のデータは、1 つまたは複数のデータベース区画に置くことができる。 1 つの表が複数の区画に分散している場合、行の一部がある区画に保管され、それ以外の行が他の区画に保管される。「*データベース区画 (database partition)*」を参照。

区分データ・セット (PDS) (partitioned data set, PDS). OS/390 環境では、メンバーと呼ばれる、区画に分けられた直接アクセス記憶装置のデータ・セット。このおのおのにプログラム、プログラムの一部、またはデータが入っている。「*プログラム・ライブラリー (program library)*」の同義語。

区分ページ・セット (partitioned page set). OS/390 環境では、区分表スペースまたは索引スペース。ヘッダー・ページ、スペース・マップ・ページ、データ・ページ、および索引ページは、区分の有効範囲内のデータだけを参照する。

区分表スペース (partitioned table space). OS/390 環境では、(索引キー範囲に基づいて) 部分に分割された表スペース。そのそれぞれはユーティリティーによって個別に処理できる。

区分化関数 (partitioned function). 行の区分化キー値を入力として取り、区分番号を出力として生成する関数。

区分化キー (partitioning key). (1) 与えられた表の 1 つ以上の列の順序セット。表の各行ごとに、行が属するデータベース区画がどれかを判別するために区分化キー列の値が使用される。 (2) 複製では、表内での 1 つまたは複数の列の順序セット。ソース表内の各行ごとに、行が属するターゲット表がどれかを判別するために区分化キー列の値が使用される。

区分化マップ (partitioning map). 区分化マップ索引をノード・グループ内のデータベース区画にマップする、区分番号のベクトル。

区分化マップ索引 (partitioning map index). ハッシュ区分または範囲区分に割り当てられた番号。

パートナー論理装置 (LU) (partner logical unit, LU). (1) SNA では、セッション内のリモート参加者。 (2) VTAM 会話によってローカル DB2 UDB (OS/390 版) に接続された、SNA ネットワーク内のアクセス点。

パススルー (pass-through). 連合データベース・システムにおける機能の 1 つ。ユーザーはこれを使用して、データ・ソースの SQL ダイアレクトでデータ・ソースと通信できる。

パス (path). 「*SQL パス* (SQL path)」を参照。

PCT. CICS では、プログラム制御テーブル (program control table)。

PDS. 「*区分データ・セット* (partitioned data set)」を参照。

対等通信 (peer-to-peer communication). 2 つの SNA 論理装置 (LU) 間のホストによって管理されない通信。 LU 6.2 ノードを参照するときに使用される。

パフォーマンス・メトリックス (performance metrics). 同じデータベース・オブジェクトに属するパフォーマンス変数の集合。

パフォーマンス・モニター (Performance Monitor). データベース管理者がチューニングの目的で、グラフィカル・インターフェースを使用して DB2 システムのパフォーマンスをモニターするのを可能にするツール。このツールは、コントロール・センターからアクセスできる。

パフォーマンス・スナップショット (performance snapshot). ある時点でデータベース・マネージャーから検索された一連のデータベース・オブジェクトのためのパフォーマンス・データ。

パフォーマンス変数 (performance variable). データベース・マネージャーから得たパフォーマンス・データから派生した統計。この変数の式はユーザー定義することができる。

パフォーマンス変数プロファイル (performance variable profile). パフォーマンス変数の定義を含むフラット・ファイル。このファイルの編集、コピーおよび共有が可能。同じパフォーマンス・モニターが別プロファイルを使用して、さまざまな計算を実行することができる。

パーシスタンス (persistence). Net.Data で、割り当てられた値を、 (複数の Net.Data 呼び出しにまたがる) トランザクション全体にわたって保持している状態。パーシスタンス (持続) 状態になる可能性があるのは、変数のみ。さらに、コミットメント制御によって影響を受ける資源に対する操作は、コミットやロールバックが明示的に行われるまで、またはトランザクションが完了するまで、アクティブに保たれる。

単独読み取り行 (phantom row). 反復可能読み取り以外の分離レベルで実行しているアプリケーション・プロセスで読み取ることができる表の行。アプリケーション・プロセスが同じ作業単位内で同じ照会を複数回発行すると、同時に実行している他のアプリケーション・プロセスによってデータが挿入されコミットされるために、照会と次の照会の間に他の行が追加されて表示されるようになる。

物理クレーム (physical claim). DB2 UDB (OS/390 版) で、非区分索引全体のクレーム。

物理的な整合性 (physical consistency). DB2 UDB (OS/390 版) では、部分変更状態でないページの状態。

物理ドレーン (physical drain). DB2 UDB (OS/390 版) で、非区分索引全体のドレーン処理。

物理ロック (P ロック) (physical lock, P-lock). 複数の異なる DB2 UDB (OS/390 版) サブシステムでキャッシュに入れられるデータの整合性を提供するために DB2 UDB (OS/390 版) が獲得するロックのタイプ。物理ロックは、データ共用環境でのみ使用される。「*論理ロック (L ロック)* (logical lock, L-lock)」と対比。

物理ロック競合 (physical lock contention). DB2 UDB (OS/390 版) では、物理ロックのリクエスターが対立している状態。「*交渉可能ロック* (negotiable lock)」も参照。

物理的な完了 (physically complete). DB2 UDB (OS/390 版) では、並行コピー・プロセスが完了し、出力データ・セットが作成されている状態。

物理装置 (PU) (physical unit, PU). SSCP-PU セッション中に SSCP によって要求された場合、ノードに関連する資源 (付加リンク、隣接リンク・ステーションなど) を管理しモニターする構成要素。 SSCP は PU とのセッションを活動化し、 PU を通して付加リンクなどのノードの資源を間接的に管理する。この用語は、タイプ 2.0、4、および 5 のノードのみに使われる。「*制御点* (control point)」も参照。

ピース (piece). OS/390 環境では、区分化されていないページ・セット。

プラン (plan). 「*アプリケーション・プラン* (application plan)」を参照。

プラン割り振り (plan allocation). プランを実行する準備のために、プランに DB2 UDB (OS/390 版) 資源を割り振るプロセス。

プラン名 (plan name). DB2 UDB (OS/390 版) では、アプリケーション・プランの名前。

プラン・セグメント化 (plan segmentation). DB2 UDB (OS/390 版) で、各プランをセクションに分割すること。セクションが必要なとき、独立して EDM プールにもたらされる。

P ロック (P-lock). 「*物理ロック* (physical lock)」を参照。

PLT. CICS では、プログラム・リスト・テーブル (program list table)。

時刻表 (point-in-time table). DB2 複製では、すべてまたは一部のソース表と内容が一致するターゲット表のタイプ。ソース・システムで特定の行が挿入または更新された大体の時刻を識別する追加システム列を含む。

一貫性ポイント (point of consistency). プログラムがアクセスする、回復可能データに一貫性がある時刻。一貫性ポイントは、更新、挿入、または削除が物理データベースをコミットしたりロールバックするいずれかの場合に発生する。「コミット・ポイント (commit point)」および「*同期点* (sync point)」の同義語。

ポリシー (policy). 「*CFRM ポリシー* (CFRM policy)」を参照。

打ち切り延期 UR (postponed abort UR). DB2 UDB (OS/390 版) では、未了または打ち切り中であったときにシステム障害または取り消しによって割り込まれ、その後再始動時にバックアウトを完了しなかった回復単位のこと。

PPT. (1) CICS では、処理プログラム・テーブル (processing program table)。 (2) OS/390 では、プログラム特性テーブル (program properties table)。

精度 (precision). 数値データ・タイプにおける、 2 進数または 10 進数の桁数の合計 (符号を除く)。

プリコンパイル (precompile). SQL ステートメントを含むプログラムをコンパイルする前に処理すること。 SQL ステートメントは、ホスト言語コンパイラーによって認識されるステートメントに置き換えられる。プリコンパイル・プロセスの出力には、コンパイラーに発信されバインド処理で使用されるソース・コードが含まれる。

述部 (predicate). 比較演算を明示または暗黙指定する検索条件の要素。

事前取り出し (prefetch). データの使用に先行してデータを読み取ること。

準備 (prepare). (1) SQL ステートメントを SQL コンパイラーに送信し、テキスト形式から実行可能な形式に変換すること。 (2) DB2 UDB (OS/390 版) では、すべての参加プログラムがコミットの準備を要求される、 2 フェーズ・コミット・プロセスの第 1 フェーズ。

準備済み SQL ステートメント (prepared SQL statement). SQL では、PREPARE ステートメントによって処理される SQL ステートメントの実行可能形式である名前付きのオブジェクト。

1 次許可 ID (primary authorization ID). DB2 UDB (OS/390 版) に対しアプリケーション・プロセスを識別するときに使用する許可 ID。

1 次グループ・バッファー・プール (primary group buffer pool). 二重化グループ・バッファー・プールの場合に、キャッシュ・データの一貫性を維持するために使用される DB2 UDB (OS/390 版) 構造のこと。この構造は、ページ登録および相互無効化に使用される。 OS/390 でこれに該当するものは、*古い* 構造 (old structure) である。「*2 次グループ・バッファー・プール* (secondary group buffer pool)」と比較。

1 次索引 (primary index). DB2 UDB (OS/390 版) では、基本キーの一意性を強める索引。

基本キー (primary key). 表の定義の一部である固有キー。基本キーは、参照制約定義の親キーの省略時値。

1 次ログ (primary log). 変更をデータベースに記録するために使用される 1 つ以上のログ・ファイル。これらのファイルの記憶域は前もって割り当てられる。「*2 次ログ* (secondary log)」と対比。

プリンシパル (principal). OS/390 環境では、別のエンティティーと確実に通信できるエンティティー。DCE の場合、プリンシパルは、DCE レジストリー・データベースの項目として表される。これには、ユーザー、サーバー、コンピューターなどが入る。

プリンシパル名 (principal name). OS/390 環境では、DCE セキュリティー・サービスが認識しているプリンシパルの名前。

私用接続 (private connection). DB2 UDB (OS/390 版) に固有の通信接続。

私用プロトコル・アクセス (private protocol access). 分散データにアクセスする方式。これにより、照会を別の DB2 システムに送信することができる。「*DRDA アクセス* (DRDA access)」と対比。

私用プロトコル接続 (private protocol connection). アプリケーション・プロセスの DB2 私用接続。「*私用接続* (private connection)」も参照。

特権 (privilege). (1) 特定の方法で特定のデータベース・オブジェクトにアクセスする権利。これらの権利は、SYSADM (システム管理者) 権限または DBADM (データベース管理者) 権限を用いてユーザーにより制御される。また、オブジェクト作成者によっても制御される。特権には、表にデータを作成する権利、表からデータを削除および選択する権利が含まれる。 (2) DB2 UDB (OS/390 版) では、特定の機能を、ときには特定のオブジェクトに対し、実行する能力。「*明示特権* (explicit privilege)」および「*暗黙特権* (implicit privilege)」も参照。

特権セット (privilege set). インストール・システム SYSADM ID の場合、可能なすべての特権のセット。その他の許可 ID の場合、 DB2 UDB (OS/390 版) カタログ内のその ID について記録されているすべての特権のセット。

プロシージャー (procedure). 「*ストアード・プロシージャー* (stored procedure)」を参照。

プロセス (process). (1) データウェアハウスセンターで、ソース・データに対して操作を行う一連のステップ。これらのステップは、データを元の形式から意思決定に役立つ形式に変更する。データウェアハウスセンターのプロセスはすべて、1 つまたは複数のソース、 1 つまたは複数のステップ、および 1 つまたは複数のターゲットで構成される。 (2) DB2 UDB (OS/390 版) では、DB2 UDB (OS/390 版) が資源およびロックを割り当てる単位。プロセスには、1 つまたは複数のプログラムの実行が含まれる。 SQL ステートメントの実行は、常に何かのプロセスに関係する。プロセスを開始および終了する方法は、環境によって異なる。「*アプリケーション・プロセス* (application process)」の同義語。

プロパティー (property). データウェアハウスセンターで、情報単位についての特性または属性の記述。各オブジェクト・タイプには、関連したプロパティーのセットがある。各オブジェクトには、複数の値からなるセットがプロパティーに割り当てられる。

保護会話 (protected conversation). OS/390 環境では、2 フェーズ・コミットの流れをサポートする VTAM 会話。

protocol.ini. すべてのプロトコルおよび媒体アクセス制御 (MAC) システム・モジュールの LAN 構成情報およびバインド情報を含むファイル。

PSRCP. DB2 UDB (OS/390 版) では、ページ・セット回復保留 (page set recovery pending)。

PU. 「*物理装置* (physical unit)」を参照。

共通権限 (public authority). すべてのユーザーに与えられているオブジェクト権限。

PU タイプ (PU type). SNA では、常駐しているノードのタイプによる物理装置の種別。

Q

QSAM. 待機順次アクセス方式 (Queued sequential access method)。

比較述部 (quantified predicate). ある値を値の集合と比較する述部。

照会 (query). (1) 特定の条件に基づいてデータベースから情報を要求すること。たとえば、顧客表から残高が ¥100,000 以上の顧客すべてのリストを要求するなど。 (2) DB2 UDB (OS/390 版) では、結果表を指定する、特定の SQL ステートメントの構成要素。

照会ブロック (query block). DB2 UDB (OS/390 版) では、FROM 文節の 1 つによって表される照会の一部。照会の DB2 UDB (OS/390 版) 内部プロセスによって、FROM 文節ごとに複数の照会ブロックがありうる。

イメージ内容による照会 (QBIC) (Query by Image Content, QBIC). イメージ・エクステンダーが提供する機能。この機能を使用して、ユーザーは平均色、テクスチャーなどの可視特性によって画像を検索できる。

照会 CP 並列処理 (query CP parallelism). DB2 UDB (OS/390 版) では、複数のタスクを使用して単一の照会を並列して実行すること。 「*シスプレックス照会並列処理* (Sysplex query parallelism)」と比較。

照会入出力並列処理 (query I/O parallelism). DB2 UDB (OS/390 版) では、単一の照会で複数の入出力要求を生成して行われるデータの並列アクセス。

待機順次アクセス方式 (QSAM) (queued sequential access method, QSAM). 基本順次アクセス方式 (BSAM) (basic sequential access method, BSAM) の拡張版。この方式が使われる時、待ち行列は処理待ちの入力データ・ブロックで構成されるか、または補助記憶装置や出力装置への転送を待っている出力データ・ブロックで構成される。

静止 (quiesce). 新たな作業要求を拒否し、正常に操作を完了してプロセスを終了すること。

静止メンバー状態 (quiesced member state). DB2 UDB (OS/390 版) では、データ共用グループのメンバーの状態。 STOP DB2 コマンドが障害なく有効となるときに、活動状態のメンバーは静止状態になる。コマンドが有効になる前に、メンバーのタスク、アドレス空間、または OS/390 システムに障害が発生すると、メンバー状態は障害となる。

引用符付き名前 (quoted name). 「*区切り識別子* (delimited identifier)」を参照。

R

RACF®. OS/390 環境では、資源アクセス管理機能 (Resource Access Control Facility)。

RAMAC®. OS/390 環境では、IBM の企業向けディスク装置システム・プロダクトのファミリー。

RBA. 「*相対バイト・アドレス* (relative byte address)」を参照。

RCT. CICS 接続機能のある DB2 UDB (OS/390 版) における、資源管理テーブル (resource control table)。

RDB. 「*リレーショナル・データベース* (relational database)」を参照。

RDBMS. 「*リレーショナル・データベース管理システム* (relational database management system)」を参照。

RDBNAM. 「*リレーショナル・データベース名* (relational database name)」を参照。

RDF. DB2 UDB (OS/390 版) では、レコード定義フィールド (record definition field)。

読み取り固定 (RS) (read stability, RS). アプリケーションがトランザクション中に検索する行のみをロックする分離レベル。行の読み取りの修飾がトランザクションが完了するまで他のアプリケーション・プロセスによって変更されないようにする。また、他のアプリケーション・プロセスによって変更された行の読み取りは、変更がプロセスによってコミットされるまで行われない。読み取り固定は、反復可能読み取りより並行性が高いが、カーソル固定より低い。

再バインド (rebind). すでにバインドされているアプリケーション・プログラムのためにパッケージを作成すること。たとえば、プログラムによってアクセスされた表に索引が追加された場合、新しい索引を利用するためにはパッケージの再バインドをする必要がある。

レコード (record). 表またはその他のデータの単一行の記憶域を表す用語。

レコード識別子 (RID) (record identifier, RID). レコード ID。DB2 が表内のレコードを固有に識別するために内部で使用する番号。レコードが保管されているページをアドレス指定するのに必要な情報を含む。「*行識別子* (row ID)」と比較。

レコード識別子 (RID) プール (record identifier (RID) pool). DB2 UDB (OS/390 版) では、16 メガバイトの境界より上の主記憶域の区域。リスト事前取り出し処理中のレコード識別子の分類用に予約される。

記録 (recording). 後に表示可能なパフォーマンス・スナップショットからの情報。

回復可能ログ (recoverable log). すべてのログ・レコードが保存されているデータベース・ログ。障害時に、順方向回復によりデータ脱落を回復できる。「*循環ログ* (circular log)」と対比。

回復 (recovery). (1) システムまたはシステムに保管されているデータが損傷を受けたあと、操作可能な状態にリセットすること。 (2) バックアップを復元し、関連するログをロールフォワードしてデータベースを再構築するプロセス。

回復ログ (recovery log). 「*データベース・ログ* (database log)」を参照。

回復保留中 (recovery pending). データベースまたは表スペースの状態。データベースまたは表スペースがバックアップから復元されると回復保留状態になる。データベースまたは表スペースが回復保留状態のとき、データのアクセスはできない。

回復トークン (recovery token). DB2 UDB (OS/390 版) では、回復時に使用されるエレメントの識別子 (たとえば、*NID* または *URID*)。

RECP. DB2 UDB (OS/390 版) では、回復保留中 (recovery pending)。

再帰サイクル (recursion cycle). 共通表式の全選択に FROM 文節の共通表式名が組み込まれているときに起きるサイクル。

再帰的共通表式 (recursive common table expression). 全選択から FROM 文節内の自分自身を参照する共通表式。再帰的共通表式は再帰的照会を書き込むときに使用される。

再帰的照会 (recursive query). 再帰的共通表式を使用する全選択。

再実行 (redo). DB2 UDB (OS/390 版) では、行われた変更を DASD 媒体に再度適用し、データの完全性を確保すべきことを示す回復単位の状態。

参照制約 (referential constraint). 外部キーの非ヌル値が有効なのは親キーの値として表示されるときだけである、という参照保全規則。

参照保全 (referential integrity). (1) 外部キーの値がすべて有効であるデータベースの状態。 (2) ある表の 1 つの列のデータから同じまたは別の表の別の列のデータへの意図的な参照がすべて有効であるときに存在する状態。参照保全を維持するには、すべての LOAD、RECOVER、INSERT、UPDATE、および DELETE 操作に DB2 UDB (OS/390 版) が参照制約を強制する必要がある。

参照構造 (referential structure). DB2 UDB (OS/390 版) では、少なくとも 1 つの表を含む表および関係の集合。その集合の各表について、表が参加するすべての関係および関連するすべての表。

最新表示 (refresh). ユーザー表から必要なデータをターゲット表にコピーして、既存のデータと置き換えるプロセス。「**全最新表示** (full refresh)」および「**差分最新表示** (differential refresh)」を参照。

登録 (registration). 「*複製ソース* (replication source)」を参照。

登録プロセス (registration process). DB2 複製では、複製ソースを定義するプロセス。「**サブスクリプション・プロセス** (subscription process)」と対比。

レジストリー・データベース (registry database). OS/390 環境では、プリンシパル、グループ、編成、アカウント、および機密保護ポリシーに関する機密保護情報のデータベース。 DCE 機密保護構成要素によって保守される。

正規表スペース (regular table space). 非一時データを保管できる表スペース。

拒否されたトランザクション (rejected transaction). DB2 複製では、1 つ以上のレプリカ表からの更新データを含むトランザクション。これらの更新データはソース表と比較して古い。

リレーショナル・キューブ (relational cube). 全体として多次元データベースを定義するデータおよびメタデータからなる集合。リレーショナル・キューブは、多次元データベースの一部で、リレーショナル・データベースに格納される。「多次元データベース (multidimensional database)」も参照。

リレーショナル・データベース (relational database). 表の集合とみなすことができるデータベースで、データのリレーショナル・モデルに従って操作できるもの。

リレーショナル・データベース管理システム (RDBMS) (relational database management system, RDBMS). DB2 UDB (OS/390 版) で、リレーショナル・データベースへのアクセスを編成および提供するハードウェアとソフトウェアの集まり。

リレーショナル・データベース名 (RDBNAM) (relational database name, RDBNAM). ネットワークにおける RDBMS の固有の識別子。 DB2 UDB (OS/390 版) では、CDB 内の表 SYSIBM.LOCATIONS の LOCATION 列にある値にする必要がある。 DB2 UDB (OS/390 版) の資料では、LOCATION 値またはロケーション名として他の RDBMS の名前を参照する。

関係 (relationship). DB2 UDB (OS/390 版) では、1 つの表の行の間または 2 つの表の行の間の定義済み接続。関係は、参照制約の内部表記である。

相対バイト・アドレス (RBA) (relative byte address, RBA). OS/390 環境では、データ・セットまたはそれが属するファイルに割り振られた記憶スペースの先頭からの、データ・レコードまたは制御インターバルのオフセット。

再移行 (remigration). DB2 UDB (OS/390 版) の直前のリリースにフォールバックしたあと、次にくる現行リリースに戻るプロセス。この手順は、別の移行プロセスを構成する。

リモート (remote). DB2 UDB (OS/390 版) では、リモート DB2 サブシステムによって保守されるオブジェクト。たとえば、リモート視点は、リモート DB2 サブシステムによって維持されている視点である。「*ローカル* (local)」と対比。

リモート接続要求 (remote attach request). DB2 UDB (OS/390 版) では、リモート・ロケーションによってローカル DB2 サブシステムに接続する要求。具体的には、この要求送信は SNA 機能管理ヘッダー 5 である。

リモート・データベース (remote database). 現在使用しているワークステーション以外のワークステーションに物理的に置かれているデータベース。「*ローカル・データベース* (local database)」と対比。

リモート・サブシステム (remote subsystem). DB2 UDB (OS/390 版) では、*ローカル・サブシステム* (local subsystem) 以外の RDBMS。ユーザーまたはアプリケーションはこれと通信することができる。サブシステムは、物理的な意味でリモートにある必要はなく、同一 OS/390 システム下の同じプロセッサーで稼働するものであってもよい。

リモート作業単位 (RUOW) (remote unit of work, RUOW). SQL ステートメントのリモート準備および実行が可能である作業単位。

再最適化 (reoptimization). 実行時に SQL ステートメントのアクセス・パスを再考慮する DB2 UDB (OS/390 版) プロセス。再最適化の実行中、DB2 UDB (OS/390 版) はホスト変数、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する。

REORG 保留中 (REORP) (REORG pending, REORP). DB2 UDB (OS/390 版) では、再編成が必要なオブジェクトに対する SQL アクセスおよびほとんどのユーティリティー・アクセスを制限する条件のこと。

REORP. 「*REORG 保留中 (REORP)* (REORG pending, REORP)」を参照。

反復可能読み取り (RR) (repeatable read, RR). トランザクション中に参照される、アプリケーション内の行すべてをロックする分離レベル。プログラムが反復可能読み取り保護を使用する場合、プログラムによって参照される行は、そのプログラムが現行のトランザクションを終了しない限り他のプログラムによって変更できない。

レプリカ (replica). ローカルに更新でき、サブスクリプション定義を通してユーザー表から更新情報を受け取るタイプのターゲット表。ユーザー表または読み取り専用ターゲット表を更新するときのソースとなる。

レプリカ・ターゲット表 (replica target table). ターゲット・サーバーにある複製表の 1 つで、随時更新ターゲット表の一種。

複製 (replication). 複数の場所にある定義済みデータ集合を保守するためのプロセス。これには、特定の変更内容をある場所 (ソース) から別の場所 (ターゲット) にコピーしたり、 2 つの場所にあるデータを同期化することが関係する。

複製管理者 (replication administrator). 複製ソースおよびサブスクリプションの定義に関して責任のあるユーザー。このユーザーは収集プログラムおよび変更適用プログラムも実行できる。

複製ソース (replication source). コピー要求を受け入れ、サブスクリプション・セットのソース表となるデータベース表または視点。「サブスクリプション・セット (subscription set)」も参照。

複製サブスクリプション (replication subscription). 変更データを複製ソースからターゲット表へ、指定時刻に指定頻度でコピーする仕様。拡張データ・オプションがある。データをコピーするために変更適用プログラムによって要求された情報をすべて定義する。

要求コミット (request commit). DB2 UDB (OS/390 版) では、参加プログラムがデータを修正し、コミットまたはロールバックの準備が整った場合に準備フェーズに提出される決定。

リクエスター (requester). DB2 UDB (OS/390 版) では、リモート RDBMS に要求するソース。つまりデータを要求するシステム。「アプリケーション・リクエスター (application requester)」の同義語。

予約語 (reserved word). (1) プログラムまたはコンパイラーが行う処理を記述するためにソース・プログラムによって使用される語。 ユーザー定義名またはシステム名として使ってはいけない。 (2) SQL 標準で特別に使用するために取ってある語。

資源 (resource). DB2 UDB (OS/390 版) では、ロックまたはクレームのオブジェクト。これには表スペース、索引スペース、データ区分、索引区分、または論理区分がある。

資源割り振り (resource allocation). DB2 UDB (OS/390 版) で、特にデータベース資源を扱うプラン割り振りの一部。

資源管理テーブル (RCT) (resource control table, RCT). DB2 UDB (OS/390 版) では、サイト提供のマクロ・パラメーターによって作成される CICS 接続機能の構成。トランザクションまたはトランザクション・グループの許可およびアクセス属性を定義する。

資源定義オンライン (resource definition online). CICS の OS/390 環境で、表をアセンブルせずにオンラインで CICS 資源を定義できるようにする機能。

資源限定機能 (RLF) (resource limit facility, RLF). 動的操作可能 SQL ステートメントが指定の時間制限を超えないようにする、 DB2 UDB (OS/390 版) コードの一部。「*管理プログラム* (governor)」の同義語。

資源限定表 (resource limit specification table). DB2 UDB (OS/390 版) では、資源限定機能によって実行される制限を指定する、サイト定義の表。

再始動保留中 (RESTP) (restart pending, RESTP). DB2 UDB (OS/390 版) では、ページ・セットまたは区画に対する限定状態であり、そのオブジェクトに対して再始動 (バックアウト) 作業を実行する必要があることを示す。ページ・セットあるいは区分へのアクセスは、RECOVER POSTPONED コマンドによるアクセス、または (システム・パラメーターが LBACKOUT=AUTO の場合、再始動後に DB2 UDB (OS/390 版) によって起動される) 自動オンライン・バックアウトによるアクセスを除いて、すべて拒否される。

RESTP. 「*RESTART 保留中 (RESTP)* (restart pending, RESTP)」を参照。

復元 (restore). バックアップ・コピーを活動記憶装置のロケーションに戻すこと。

復元セット (restore set). データベースまたは表スペース (ログ・ファイルも含む場合がある) のバックアップ・コピー。復元およびロールフォワードが行われたとき、データベースまたは表スペースを一貫性のある状態に戻す。

結果セット (result set). ストアード・プロシージャーが戻す行のセット。

結果セット・ロケーター (result set locator). ストアード・プロシージャーが戻す照会結果セットを固有に識別するために DB2 UDB (OS/390 版) が使用する 4 バイトの値。

結果表 (result table). SELECT ステートメントの評価によって生成された行のセット。

保持ロック (retained lock). DB2 UDB (OS/390 版) サブシステム障害時にそのサブシステムが保持していた MODIFY ロック。ロックは DB2 UDB (OS/390 版) の障害の間、結合機能のロック構造に保持される。

取り消し (revoke). 特権または権限を許可 ID から除くこと。

RID. 「*レコード識別子 (RID)* (record identifier, RID)」を参照。

RID プール (RID pool). 「*レコード識別子プール* (record identifier pool)」を参照。

右方外部結合 (right outer join). DB2 UDB (OS/390 版) では、結合中の両方の表の一致した行が入っていて、 2 つめの結合オペランドの不一致行を保存する、結合操作の結果。「*結合* (join)」を参照。

RLF. 「*資源限定機能* (resource limit facility)」を参照。

RO. DB2 UDB (OS/390 版) では、読み取り専用アクセス (read-only access)。

ロールバック (rollback). SQL ステートメントによって変更されたデータを、その最後のコミット・ポイントにおける状態に復元するプロセス。「*一貫性ポイント* (point of consistency)」を参照。

ロールフォワード (roll-forward). データベース・ログに記録されている変更を適用することによって、復元されたデータベースのデータを更新するプロセス。「*順方向回復* (forward recovery)」を参照。

ルート・ページ (root page). DB2 UDB (OS/390 版) では、最初の索引スペースのマップ・ページが後に続く索引ページ・セットのページ。ルート・ページは、索引の最上位レベル (または開始点)。

ルーチン (routine). DB2 UDB (OS/390 版) では、ユーザー定義関数またはストアード・プロシージャー。

行 (row). 表の各列の値の順番からなる表の水平構成要素。

ROWID. 「*行識別子* (row identifier)」を参照。

行識別子 (ROWID) (row identifier, ROWID). DB2 UDB (OS/390 版) では、行を固有に識別する値。この値は当該の行と一緒に保管され、変更されない。

行ロック (row lock). DB2 UDB (OS/390 版) では、データの単一の行のロック。

行レプリカ (row-replica). DB2 複製では、トランザクションのセマンティクスなしで DataPropagator for Microsoft Jet が保守する一種の随時更新レプリカ。

行レプリカ対立検出 (row-replica conflict detection). DB2 複製では、トランザクションごとではなく、行ごとになされる対立検出。これは、DB2 レプリカに対してなされるのと同じ。

行トリガー (row trigger). DB2 UDB (OS/390 版) では、各行に対して、トリガー細分性を指定して定義したトリガー。

RR. 「*反復可能読み取り* (repeatable read)」を参照。

RRE. IMS の OS/390 環境における、残余回復項目 (residual recovery entry)。

RS. 「*読み取り固定* (read stability)」を参照。

RRSAF. 回復可能資源管理プログラム・サービス接続機能 (RRSAF)。 DB2 UDB (OS/390 版) の副構成要素であり、OS/390 Transaction Management and Recoverable Resource Manager Services を使用して、DB2 UDB (OS/390 版) およびその他の資源管理プログラム (OS/390 システムで、 OS/390 RRS をやはり使用するもの) の間で資源コミットメントを調整する。

RUOW. 「*リモート作業単位* (remote unit of work)」を参照。

S

検索引き数述部 (sargable). 検索引き数として評価される述部。

サテライト (satellite). サテライト制御データベースでグループと同期化される DB2 サーバーを持つ、断続的に接続されるクライアント。

サテライト・アドミニストレーション・センター (Satellite Administration Center). サテライトのために集中管理サポートを提供するユーザー・インターフェース。

サテライト制御サーバー (satellite control server). サテライト制御データベース SATCTLDB を含む DB2 ユニバーサル・データベース・システム。

SBCS. 「*1 バイト文字セット* (single-byte character set)」を参照。

SCA. DB2 UDB (OS/390 版) では、共有連絡域 (shared communications area)。

スカラー全選択 (scalar fullselect). 単一値（1 行 1 列のみで構成されているデータ）を戻す全選択。

スカラー関数 (scalar function). 別の値から単一の値を作成する SQL 操作。これは括弧に囲まれた引き数のリストが後に続いた関数名によって表現される。「*列関数* (column function)」と対比。

位取り (scale). 小数部の桁数。

スキーマ (schema). (1) 表、視点、索引またはトリガーなどデータベース・オブジェクトのコレクション。データベース・オブジェクトの論理種別を提供する。 (2) DB2 UDB (OS/390 版) では、ユーザー定義関数、特殊タイプ、トリガー、およびストアード・プロシージャーを論理的にグループ化したもの。これらのタイプのいずれかのオブジェクトが 1 つ作成されると、そのオブジェクトは、そのオブジェクト名によって判別される 1 つのスキーマに割り当てられる。 (3) データウェアハウスセンターで、ウェアハウス・ターゲット表、およびウェアハウス・ターゲット表間の関連から成る集合。これを構成するターゲット表は、1 つまたは複数のウェアハウス・ターゲットに由来する。

SDK. 「*ソフトウェア開発者キット* (Software Developer's Kit)」を参照。

SDWA. OS/390 環境では、システム診断作業域 (system diagnostic work area)。

検索条件 (search condition). 表から行を選択するための基準。 1 つの検索条件は、1 つまたは複数の述部から構成される。

2 次許可 ID (secondary authorization ID). DB2 UDB (OS/390 版) では、許可出口ルーチンによって 1 次許可 ID に結び付けられた許可 ID。

2 次グループ・バッファー・プール (secondary group buffer pool). DB2 UDB (OS/390 版) 環境の二重化されたグループ・バッファー・プールのための、 1 次グループ・バッファー・プールに書き込まれる変更されたページをバックアップするために使用される構造。 2 次グループ・バッファー・プールを使用

していると、ページの登録および相互無効化は一切行われない。 OS/390 での「*新規の* (new)」構造と等価。「*1 次グループ・バッファー・プール* (primary group buffer pool)」と比較。

2 次ログ (secondary log). 変更をデータベースに記録するために使用される 1 つ以上のログ・ファイル。これらのファイルの記憶域は、1 次ログがいっぱいになったときに必要に応じて割り当てられる。「*1 次ログ* (primary log)」と対比。

セクション (section). DB2 UDB (OS/390 版) では、単一 SQL ステートメントの実行可能な構造体を含むプランまたはパッケージのセグメント。ほとんどの SQL ステートメントでは、ソース・プログラムの各 SQL ステートメントごとに、プランの中にセクションが 1 つある。しかし、カーソルに関連付けられているステートメント DECLARE、OPEN、FETCH、および CLOSE ステートメントの場合は、それぞれ DECLARE CURSOR ステートメントで指定されている SELECT ステートメントを参照するため、どれもが同じセクションを参照する。また、COMMIT や ROLLBACK などの SQL ステートメントと一部の SET ステートメントはセクションを使用しない。

セグメント化表スペース (segmented table space). DB2 UDB (OS/390 版) では、セグメントと呼ばれる同一サイズのページのグループに分けられた表スペース。セグメントの表への割り当ては、異なる表の行が、同じセグメントに保管されないように行われる。

自己参照制約 (self-referencing constraint). DB2 UDB (OS/390 版) では、表がそれ自体従属である関連を定義する参照制約。

自己参照行 (self-referencing row). 自分の親である行。

自己参照副照会 (self-referencing subquery). SQL ステートメントのオブジェクトである表を参照する DELETE、 INSERT、または UPDATE ステートメントの副選択または全選択。

自己参照表 (self-referencing table). 同じ参照制約にある親および従属表。

順次データ・セット (sequential data set). 磁気テープ上のような、連続した物理位置に基づいてレコードの編成が行われる、非 DB2 UDB (OS/390 版) データ・セット。 DB2 UDB (OS/390 版) データベース・ユーティリティーには、順次データ・セットを必要とするものがいくつかある。

順次事前取り出し (sequential prefetch). DB2 UDB (OS/390 版) では、連続する非同期入出力操作を起動するメカニズム。ページは必要とされる前に取り出され、いくつかのページが単一の入出力操作で読み取られる。

サーバー (server). (1) ネットワークにおいて、他のステーションに機能を提供するノード。たとえば、ファイル・サーバー、プリンター・サーバー、メール・サーバーなど。 (2) 連合データベース・システムにおいて、データ・ソースの連合サーバーを識別する情報単位。この情報には、サーバーの名前、タイプ、バージョン、および (データ・ソースとの通信やデータ検索のために) 連合サーバーが使用するラッパーの名前が含まれる。 (3) ネットワークを通して 1 つまたは複数のクライアントにサービスを提供する機能単位。 DB2 UDB (OS/390 版) 環境では、サーバーはリモート RDBMS からの要求のターゲットであり、データを提供する RDBMS でもある。「アプリケーション・サーバー (application server)」も参照。

サービス・クラス (service class). DB2 UDB (OS/390 版) では、MVS ワークロード管理プログラムが顧客のパフォーマンスのゴールを特定の DDF スレッドやストアード・プロシージャーに関連付けるために使用する 8 文字の識別子。並列補助機能の作業を分類するためにも使用される。

サービス名 (service name). リモート・ノードで使用されるポート番号を指定する記号方式を提供する名前。 TCP/IP 接続では、リモート・ノードのアドレスと、アプリケーションを識別するためにリモート・ノードで使用されるポート番号が必要である。

セッション (session). 2 つのステーションまたは SNA ネットワーク・アドレス可能単位 (NAU) の間の論理結合。これにより、2 つのステーションまたは NAU 間の通信ができる。

セッション限度 (session limit). SNA で、特定の論理単位 (LU) がサポートできる、並行して活動状態にある論理装置－論理装置 (LU 対 LU) セッションの最大数。

セッション・パートナー (session partner). SNA では、活動セッションに参加している 2 つのネットワーク・アドレス可能単位 (NAU) のうちの一方。

セッション・プロトコル (session protocols). DB2 UDB (OS/390 版) では、SNA 通信要求および応答で使用できるセット。

セッション機密保護 (session security). LU 6.2 では、パートナー LU 検査およびセッション・データの暗号化。データを暗号化された形式で送信できるシステム・ネットワーク体系 (SNA) 機能。

セット演算子 (set operator). 合併、差、および論理積といった関係演算子に対応する SQL 演算子 UNION、EXCEPT および INTERSECT。集合演算子は、2 つの別々の結果表を結合させて結果表を派生する。

シャドー (shadowing). 現行ページが重ね書きされないようにする回復方法。そのかわりに、値が置換されているページがシャドー・コピーとして保存されている間に、新しいページが割り当てられ、書き込まれる。シャドー・コピーは、トランザクション・ロールバックによるシステム状態の復元をサポートする必要がある間保存されている。

共用連絡域 (SCA) (shared communications area, SCA). DB2 UDB (OS/390 版) データ共用グループが DB2 間の通信に使用する、結合機能のリスト構造。

共用ロック (shared lock). 並行して実行されているアプリケーション・プロセスをデータベースのデータの読み取り専用操作に限定するロック。「*排他ロック* (exclusive lock)」と対比。

シフトイン文字 (shift-in character). EBCDIC システムで使用される特殊制御文字 (X'0F')。これ以降のバイトが SBCS 文字であることを表す。「シフトアウト文字 (shift-out character)」と対比。

シフトアウト文字 (shift-out character). EBCDIC システムで使用される特殊制御文字 (X'0E')。これ以降のバイトから次のシフトイン文字までが DBCS 文字であることを表す。「シフトイン文字 (shift-in character)」と対比。

短ストリング (short string). (1) 最大長が 254 以下である固定長ストリングまたは可変長ストリング。(2) DB2 UDB (OS/390 版) では、実際の長さを持つストリング、または最大長の可変長ストリングで、255 バイト (127 個の 2 バイト文字) またはそれより短いもの。長さにかかわらず、LOB ストリングは短ストリングではない。

サインオン (sign-on). 個々の CICS もしくは IMS アプリケーション・プロセスに代わって接続機能が行う要求。これによって、DB2 UDB (OS/390 版) はアプリケーション・プロセスに DB2 UDB (OS/390 版) 資源の使用が許可されているかを検査することができる。

単純ページ・セット (simple page set). DB2 UDB (OS/390 版) では、区分化されていないページ・セット。単純ページ・セットは最初は単一のデータ・セット (ページ・セット部分) から成る。データ・セットが 2 ギガバイトにまで拡張される場合は、別のデータ・セットが作成されて、合計 32 までのデータ・セットとなる。 DB2 UDB (OS/390 版) はこのデータ・セットを、最大 64 ギガバイトを含む単一の隣接した線形アドレス空間であるとみなす。データは、区分化している構造とはみなされず、このアドレス空間内で次に使用可能なロケーションに保管される。

単純表スペース (simple table space). DB2 UDB (OS/390 版) では、区分化されたりセグメント化されない表スペース。

1 バイト文字セット (SBCS) (single-byte character set, SBCS). 各文字が単一バイト・コードで表されている文字セット。

単精度浮動小数点数 (single-precision floating point number). 実数を 32 ビットで近似表現したもの。

SMF. OS/390 環境では、システム管理機能 (system management facility)。

SMS. OS/390 環境では、記憶管理サブシステム (Storage Management Subsystem)。

SMS 表スペース (SMS table space). 「*システム管理スペース表スペース* (system-managed space table space)」を参照。

SNA. 「*システム・ネットワーク体系* (Systems Network Architecture)」を参照。

SNA ネットワーク (SNA network). システム・ネットワーク体系 (SNA) の形式およびプロトコルに適合するユーザー・アプリケーション・ネットワークの一部。ユーザー間の確実な転送を可能にし、いろいろなネットワーク構成の資源を制御するプロトコルを提供する。 SNA ネットワークは、ネットワーク・アドレス可能単位 (NAU)、ゲートウェイ機能、中間セッション経路指定関数構成要素およびトランスポート・ネットワークから成る。

スナップショット (snapshot). 「*パフォーマンス・スナップショット* (performance snapshot)」および「*Explain スナップショット* (explain snapshot)」を参照。

ソケット (socket). 呼び出し可能な TCP/IP プログラミング・インターフェース。 TCP/IP ネットワーク・アプリケーションでリモートの TCP/IP パートナーと通信するために使用される。

ソフト・チェックポイント (soft checkpoint). 情報をログ・ファイルのヘッダーに書き込むプロセス。この情報は、データベースの再始動要求が出されたときに、ログでの開始点を判別するために使用される。

ソフトウェア開発者キット (SDK) (Software Developer's Kit, SDK). クライアントのワークステーションでのアプリケーション開発を可能にするアプリケーション開発製品。この開発は、ホスト・リレーショナル・データベースを含むリモート・データベース・サーバーに DB2 コネクト製品を使ってアクセスするためのもの。

ソース (source). データウェアハウスセンターで、ステップへの入力となる表、視点、またはファイル。

ソース関数 (source function). 1 つ以上の UDF を実行するために使用されるユーザー定義関数 (UDF)。

ソース関数 (sourced function). DB2 UDB (OS/390 版) では、データベース・マネージャーがすでに認識している別の組み込み関数またはユーザー定義関数によってその機能が実現されている関数。この関数となり得るのは、スカラー関数または列 (集合) 関数のいずれかである。この関数は、値の集合 (たとえば MAX や AVG など) の中から単一の値を戻す。「*外部関数* (external function)」および「*組み込み関数* (built-in function)」と対比。

ソース・プログラム (source program). SQL プリコンパイラーによって処理されるホスト言語ステートメントおよび SQL ステートメントのセット。

ソース・サーバー (source server). DB2 複製では、複製ソースおよび収集プログラムのデータベースのロケーション。

ソース表 (source table). DB2 複製では、ターゲット表にコピーされるデータを含む表。 ソース表は複製ソース表、変更データ表、整合した変更データ表がある。 「*ターゲット表* (target table)」と対比。

ソース・タイプ (source type). 特殊タイプを内部的に表示するために使用される既存のタイプ。

特殊レジスター (special register). データベース・マネージャーによってアプリケーション・プロセス用に定義された記憶域。 SQL ステートメントで参照される情報を保管するときに使用する。たとえば、USER および CURRENT DATE など。

特定関数名 (specific function name). (1) システムに固有の関数を識別する名前。 (2) DB2 UDB (OS/390 版) では、その特定の名前によってデータベース・マネージャーに認識されている特定のユーザー定義関数のこと。多くの特定のユーザー定義関数が同じ関数名を持つことができる。 データベースに対してユーザー定義関数を定義すると、各関数にはそれが属すスキーマの中で固有の特定名が割り当てられる。ユーザーはこの名前を使用できる。またはデフォルト名が使用される。

予備ファイル (spill file). DB2 複製では、変更適用プログラムが作成する一時ファイル。複数のターゲット表に対してデータを更新するためのソースとして使用される。

表計算アドイン (Spreadsheet Add-in). OLAP Starter Kit で、Microsoft Excel とロータス 1-2-3 をマージしてデータの多次元分析を可能にするソフトウェア。このソフトウェア・ライブラリーはスプレッドシートのメニュー・アドインとして表示され、結合、ズームイン、計算などの多次元分析機能を提供する。

SPUFI. DB2 UDB (OS/390 版) では、ファイル入力式 SQL 処理プログラム (SQL Processor Using File Input)。

SQL. 「*構造化照会言語* (Structured Query Language)」を参照。

SQL 許可 ID (SQL ID) (SQL authorization ID, SQL ID). DB2 UDB (OS/390 版) では、特定の状態での動的 SQL ステートメントをチェックするために用いられる許可 ID。

SQLCA. 「*SQL 通信域* (SQL communication area)」を参照。

SQL 連絡域 (SQLCA) (SQL communication area, SQLCA). アプリケーション・プログラムに SQL ステートメントの実行またはデータベース・マネージャーからの要求に関する情報を提供する変数のセット。

SQL 接続 (SQL connection). DB2 UDB (OS/390 版) では、アプリケーション・プロセスとローカルまたはリモートのアプリケーション・サーバーの間の関連。

SQLDA. 「*SQL 記述子域* (SQL descriptor area)」を参照。

SQL 記述子域 (SQLDA) (SQL descriptor area, SQLDA). (1) 特定の SQL ステートメントの処理で使用される変数のセット。 SQLDA は動的 SQL プログラムを対象としたもの。 (2) 入力変数、出力変数、または結果表の列を記述する構造。

SQL エスケープ文字 (SQL escape character). DB2 UDB (OS/390 版) では、SQL 区切り識別子を囲むために使用される記号。この記号は二重引用符 (") となる。「エスケープ文字 (escape character)」と比較。

SQL ID. 「*SQL 許可 ID* (SQL authorization ID)」を参照。

SQL パス (SQL path). DB2 UDB (OS/390 版) では、ユーザー定義関数、特殊タイプ、およびストアード・プロシージャーへの修飾されない参照を解決するために使用される、スキーマ名の順序付きリスト。動的 SQL では、現行パスは CURRENT PATH 特殊レジスターの中に入っている。静的 SQL では、現行パスは PATH バインド・オプションの中で定義される。

SQL 処理会話 (SQL processing conversation). アプリケーションまたは動的照会リクエスターのいずれかによって、DB2 UDB (OS/390 版) データのアクセスを必要とする会話。

ファイル入力式 SQL 処理プログラム (SPUFI) (SQL Processor Using File Input, SPUFI). DB2 UDB (OS/390 版) では、ファイル入力式 SQL 処理プログラム (SQL Processor Using File Input)。 DB2I ユーザーが SQL ステートメントをアプリケーション・プログラムに組み込まずに実行できるようにする TSO 接続副構成要素の機能。

SQL 戻りコード (SQL return code). SQLCODE または SQLSTATE のいずれか。

SQL ルーチン (SQL routine). DB2 UDB (OS/390 版) では、SQL で書かれたコードを基にしたユーザー定義関数またはストアード・プロシージャー。

SQL ストリング区切り文字 (SQL string delimiter). DB2 UDB (OS/390 版) では、SQL ストリング定数を囲むために使用される記号。 SQL ストリング区切り文字は、アポストロフィ (') であるが、COBOL アプリケーションは例外で、ユーザーが記号 (アポストロフィまたは二重引用符(")) を割り当てる。

SSCP. 「システム・サービス*制御点* (system services control point)」を参照。

SSI. OS/390 環境では、サブシステム・インターフェース (subsystem interface)。

SSM. DB2 UDB (OS/390 版) では、サブシステム・メンバー (subsystem member)。

スタック (stack). 一時レジスター情報、パラメーターおよびサブルーチンの戻りアドレスを保管するメモリー内の領域。

ステージング表 (staging table). DB2 複製では、複製のターゲット表に対してデータを更新するためのソースとして使用できる CCD 表。

スタンドアロン (stand-alone). DB2 UDB (OS/390 版) サービスを使用せずに、 DB2 UDB (OS/390 版) から独立して実行できることを意味する、プログラムの属性。

標準対立検出 (standard conflict detection). 変更適用プログラム (Apply program) が行内の対立を検索する対立検出。この対立は、レプリカの変更データ表またはユーザー表ですでに収集されている。「*対立検出* (conflict detection)」、「*拡張対立検出* (enhanced conflict detection)」、および「*行レプリカ対立検出* (row-replica conflict detection)」も参照。

スタースキーマ (star schema). OLAP Starter Kit が使用するリレーショナル・データベース・スキーマのタイプ。多くの場合、データウェアハウスセンター内で作成される。

ステートメント (statement). プログラムまたはプロシージャーでの命令。

ステートメント・ハンドル (statement handle). CLI では、SQL ステートメントに関する情報を含むデータ・オブジェクトを参照するハンドル。動的引き数、動的引き数および列のバインド、カーソル情報、結果値および状況情報などを含む。各ステートメント・ハンドルは、接続ハンドルと関連している。

ステートメント・ストリング (statement string). DB2 UDB (OS/390 版) 環境の動的 SQL ステートメントでは、ステートメントの文字ストリング書式。

静的バインド (static bind). SQL ステートメントが事前コンパイルされた後でバインドされるというプロセス。すべての静的 SQL ステートメントの実行準備は、同時に行われる。「*バインド* (bind)」も参照。

ステートメント・トリガー (statement trigger). DB2 UDB (OS/390 版) では、各ステートメントに対して、トリガー細分性を指定して定義したトリガー。

静的 SQL (static SQL). プログラムに組み込まれている SQL ステートメント。このステートメントは、プログラムを実行する前にプログラム準備処理中に準備されるもの。準備された後、ステートメントで指定されたホスト変数が変更されても、静的 SQL ステートメントは変更されない。

状況 (status). データウェアハウスセンターで、進行中のステップ処理の状態。たとえば、スケジュール済み、移植中、成功。

ステップ (step). データウェアハウスセンターで、ウェアハウス・プロセスでのデータに対する個々の操作。ほとんどの場合、ステップは、ウェアハウス・ソース、データのトランスフォーメーションまたは移動に関する記述、およびターゲットで構成される。スケジュールに従ってステップを実行することができる。または、別のステップからカスケードすることもできる。

ステップ・エディション (step edition). データウェアハウスセンターで、ウェアハウス・ソース内の、特定時刻におけるデータのスナップショット。

記憶域グループ (storage group). DB2 UDB (OS/390 版) データを保管できる DASD ボリュームの名前付きセット。

ストアード・プロシージャー (stored procedure). (1) 手続き構成および組み込み SQL のブロック。データベースに保管され、名前で呼び出される。ストアード・プロシージャーを使用して、 1 つのアプリケーション・プログラムを 2 つの部分で実行できる。クライアントとサーバーの両方で実行できる。これにより、1 回の呼び出しでデータベースへの複数のアクセスが可能となる。「*プロシージャー (procedure)*」の同義語。 (2) DB2 UDB (OS/390 版) では、ユーザー作成のアプリケーション・プログラムで、 SQL CALL ステートメントを使用することにより呼び出される。

ストアード・プロシージャー・ビルダー (Stored Procedure Builder). ストアード・プロシージャーの作成、ローカルまたはリモート DB2 サーバー上でのストアード・プロシージャーの構築、既存のストアード・プロシージャーの変更と再構築、およびインストールしたストアード・プロシージャーのテストとデバッグを、グラフィカル・インターフェースを使用して行うツール。このツールはスタンドアロンで実行できる。または、さまざまな統合開発環境からアクセスすることもできる。

ストアード・プロシージャー・ビルダー・プロジェクト (Stored Procedure Builder project). ストアード・プロシージャー・ビルダーの作成するファイルで、接続情報、およびデータベース内に正常に構築されなかったストアード・プロシージャー・オブジェクトを含んでいる。

ストーリーボード (storyboard). ビデオの可視的な要約。ビデオ・エクステンダーには、ビデオの代表ショットであるビデオ・フレームを識別し、保管する機能がある。これらの代表フレームを使用して、ストーリーボードを作成することができる。

ストリング (string). プログラミング言語では、テキストの保管および操作に使用されるデータの形式。

強力型指定 (strong typing). DB2 UDB (OS/390 版) では、その指定されたタイプには、ユーザー定義関数および特殊タイプに対して定義された操作しか適用できないことを保証するプロセス。たとえば、カ

ナダ・ドルと米ドルのように、2 種類の通貨タイプを直接比較することはできない。しかし、片方の通貨からもう一方の通貨へと変換するユーザー定義関数を提供し、それによって比較を行うことができるようにしている。

構造化照会言語 (SQL) (Structured Query Language, SQL). リレーショナル・データベースのデータを定義および操作するための標準化言語。

サブエージェント (subagent). サブリクエストで動くエージェントのタイプ。 1 つのアプリケーションが多くの要求を出す場合、各要求はいくつかのサブリクエストに分割できる。したがって、同じアプリケーションに対して複数のサブエージェントが使用されることがある。 1 つのアプリケーションで使用されるサブエージェントはすべて、そのアプリケーションのコーディネート・エージェントで調整される。

副構成要素 (subcomponent). 一般機能を提供するため一緒になって作業する最も関連した DB2 UDB (OS/390 版) モジュールのグループ。

サブジェクト・エリア (subject area). データウェアハウスセンターで、特定の論理ビジネス領域用のウェアハウス・データを作成するプロセスから成るセット。サブジェクト・エリア内のプロセスは、特定のサブジェクト用のデータに対して操作を行い、そのサブジェクトで必要な詳細データ、データ要約、およびキューブを作成する。

従属エージェント (subordinate agent). 「サブエージェント (subagent)」を参照。

サブページ (subpage). DB2 UDB (OS/390 版) では、物理的索引ページを分割できる単位。

副照会 (subquery). 別の SQL ステートメントの WHERE 文節または HAVING 文節の内側に置かれた SELECT ステートメント。ネストされた SQL ステートメント。

サブスクリプション (subscription). 「サブスクリプション・セット (subscription set)」を参照。

サブスクリプション・サイクル (subscription cycle). DB2 複製における 1 つのプロセスで、変更適用プログラムが所定のサブスクリプション・セットの変更済みデータを取り出し、変更内容をターゲット表に複写し、進行過程を反映するように該当する複製制御表を更新する。

サブスクリプション・プロセス (subscription process). DB2 複製で、ユーザーがサブスクリプション・セットとサブスクリプション・セット・メンバーを定義するプロセス。「**登録プロセス** (registration process)」と対比。

サブスクリプション・セット (subscription set). DB2 複製では、ソース表やターゲット表のグループおよび変更データの複製を管理する制御情報の指定。「**サブスクリプション・セット・メンバー** (subscription-set member)」を参照。

サブスクリプション・セット・メンバー (subscription-set member). DB2 複製では、サブスクリプション・セットのメンバー。ソースとターゲットの組み合わせごとにメンバーが 1 つずつある。各メンバーは、ターゲット表の構造と、ソース表から複製される行と列を定義する。

副選択 (subselect). ORDER BY 文節、UPDATE 文節または UNION 演算子を含まない照会書式。

置換文字 (substitution character). SQL では、ターゲットのコード化表現に一致する文字のない、ソース・プログラム内の任意の文字を文字変換するときに、置換される固有の文字。

サブシステム (subsystem). DB2 UDB (OS/390 版) で、リレーショナル・データベース管理システム (RDBMS) の個別インスタンス。

記号宛先名 (symbolic destination name). リモート・パートナー名を指定する。 CPI 通信サイド情報表の項目に対応する名前。サーバーへの APPC 接続をセットアップするためにクライアントが必要とする情報 (パートナー LU 名、モード名、パートナー TP 名など) を含む。

同期レベル (synchronization level). APPC では、対応するトランザクション・プログラムが確認要求および応答を交換することを示す指定。

同期 (synchronous). 共通時刻信号など特定の事象のオカレンスに依存する、関連のある複数のプロセス。「*非同期* (asynchronous)」と対比。

同期点 (sync point). 「*一貫性ポイント* (point of consistency)」を参照。

同義語 (synonym). DB2 UDB (OS/390 版) では、SQL における、表または視点の代替名。同義語を使用できるのは、同義語が定義されるサブシステムのオブジェクトを参照する場合のみである。

構文文字セット (syntactic character set). 文字セット 00640 として IBM レジストリーに登録されている、 81 個の図形文字から成る文字セットのこと。このセットは、もともとはプログラム言語コミュニティーを対象として、システム間および国境を越えた移行性と互換性とを最大化することを目的とする構文への使用が推奨されていたものであった。少数の例外を除き、この文字セットはほとんどの登録済み 1 次文字セットの中に含まれている。「*非可変文字セット* (invariant character set)」と対比。

シスプレックス (Sysplex). 「*並列シスプレックス* (Parallel Sysplex)」を参照。

シスプレックス照会並列処理 (Sysplex query parallelism). 複数の DB2 UDB (OS/390 版) サブシステムで複数のタスクを使用して単一の照会を並列実行すること。「*照会 CP 並列処理* (query CP parallelism)」も参照。

システム管理者 (system administrator). コンピューターの設置において、コンピューター・システムの設計、制御、および使用管理を行う担当者。

システム・エージェント (system agent). 事前取り出し処理、据え置き書き出し、およびサービス・タスクなどの、 DB2 UDB (OS/390 版) が内部的に作成する作業要求。

システム・カタログ (system catalog). 「*カタログ* (catalog)」を参照。

システム会話 (system conversation). 分散処理が始まる前にシステム・メッセージを処理するために、 2 つの DB2 UDB (OS/390 版) が確立しておかなければならない会話。

システム・データベース・ディレクトリー (system database directory). データベース・マネージャーを使用してアクセスできるデータベースの項目を含むディレクトリー。最初のデータベースがシステムで作成またはカタログされたときに作成される。

システム診断作業域 (SDWA) (system diagnostic work area, SDWA). OS/390 環境では、プログラムまたはハードウェア・エラーを記述する SYS1.LOGREC 項目に記録されるデータ。

システム管理スペース (SMS) 表スペース (system-managed space (SMS) table space). オペレーティング・システムで管理される表スペース。この記憶域モデルは、サブディレクトリーで作成されたファイルに基づいており、ファイル・システムで管理される。「*データベース管理スペース (DMS) 表スペース* (database managed space (DMS) table space)」と対比。

システム・サービス制御点 (SSCP) (system services control point, SSCP). 従属ノードにネットワーク・サービスを提供する SNA ネットワークの制御点。

システム・ネットワーク体系 (SNA) (Systems Network Architecture, SNA). 論理構造、書式、プロトコルや、ネットワーク上に情報単位を送信する操作手順およびネットワークの構成や操作を制御する操作手順の記述。

SYS1.DUMPxx データ・セット (SYS1.DUMPxx data set). OS/390 環境では、システム・ダンプを含むデータ・セット。

SYS1.LOGREC. OS/390 環境では、プログラムおよびハードウェア・エラーについての重要な情報が入っている保守援助プログラム。

T

表 (table). 名前が付けられているデータ・オブジェクト。特定の数の列およびいくつかの順不同の行で構成されている。「*基礎表* (base table)」も参照。

表検査制約 (table check constraint). DB2 UDB (OS/390 版) では、基礎表の特定の列に入れられる値を指定する、ユーザー定義の制約。

表指定機能 (table designator). 特定のオブジェクト表を指定する列名修飾子。

表関数 (table function). DB2 UDB (OS/390 版) では、一連の引き数を受け取り、この関数を参照する SQL ステートメントに表を戻す関数。表関数は、副選択の FROM 文節でのみ参照できる。

表ロケーター (table locator). DB2 UDB (OS/390 版) では、SELECT ステートメントの FROM 文節および INSERT ステートメントの副選択の中で、または、ユーザー定義関数の内部から、トリガー変換表にアクセスすることを可能にするメカニズム。表ロケーターは、変換表を表すフルワードの整数値である。

表待ち行列 (table queue). データベース・ノード間で行を転送する機構。表待ち行列は、行の挿入および除去に関する単純化された規則のある分散表ストリームである。表待ち行列は、順次データベースの異なるプロセス間で行の送達を行うためにも使用される。

表スペース (table space). (1) データベース・オブジェクトが保管されているコンテナーの一まとまりを指す抽象概念。表スペースは、データベースとデータベースに保管されている表との間の間接参照レベルを提供する。表スペースは、

- 割り当てられた媒体記憶装置にスペースを持つ。

- その中に作成された表を持つ。これらの表は、表スペースに属するコンテナーのスペースを使用する。表のデータ、索引、長フィールドおよび LOB 部分は同じ表スペースに保管できる。また、それぞれ別個の表スペースに保管することもできる。

(2) DB2 UDB (OS/390 版) では、レコードを 1 つまたは複数の表に保管するときに使用するページ・セット。

表スペース・コンテナー (table space container). 表スペースへのスペースの割り振りを記述する総称用語。表スペース・タイプによって、コンテナーはディレクトリー、装置またはファイルとなる。

表スペース・セット (table space set). DB2 UDB (OS/390 版) では、以下の理由の 1 つで、同時に回復させなければならない表スペースならびに区画のセット。
- それぞれが他のいずれかに入っている表の親表または下層表を含んでいる。
- セットが基礎表および関連補助表を含んでいる。

表スペース・セットは、上記の 2 つのタイプの関係を両方とも包含することができる。

ターゲット (target). データウェアハウスセンターで、ステップによって生成または移植される表、視点、またはファイル。つまり、ステップの出力。

ターゲット・サーバー (target server). DB2 複製では、ターゲット表のデータベース・ロケーション。通常、変更適用プログラムのロケーションでもある。

ターゲット表 (target table). DB2 複製では、データがコピーされるターゲット・サーバー上の表。 ターゲット表には、ユーザー・コピー表、時刻表、基礎集約表、変更集約表、整合した変更データ表またはレプリカ表がある。

タスク制御ブロック (TCB) (task control block, TCB). DB2 UDB (OS/390 版) に接続されたアドレス空間内のタスクに関する情報を伝達するときに用いられる、制御ブロック。アドレス空間は多数のタスク接続をサポートできるが (タスク当たり 1 つまで)、サポートできるアドレス空間接続は 1 つだけである。

TCB. 「*タスク制御ブロック* (task control block)」を参照。

TCP/IP. 「*伝送制御プロトコル / インターネット・プロトコル* (Transmission Control Protocol/Internet Protocol)」を参照。

TCP/IP ポート (TCP/IP port). TCP/IP ホスト内でエンド・ユーザーや TCP/IP ネットワーク・アプリケーションを識別する 2 バイトの値。

テクニカル・メタデータ (technical metadata). データウェアハウスセンターで、データの技術的な面 (たとえば、データベース・タイプや長さ) を記述するデータ。テクニカル・メタデータには、データ・ソースに関する情報、およびデータの抽出、消去、変換に使われる規則に関する情報が含まれる。テクニカル・メタデータは、データウェアハウスセンターのメタデータのかなりの部分を占める。「ビジネス・メタデータ (business metadata)」と対比。

一時表 (temporary table). SQL ステートメントを処理して中間結果を入れるために作成される表。「*結果表* (result table)」と対比。

一時表スペース (temporary table space). 一時表のみ保管する表スペース。

テリトリー (territory). データベース・マネージャーによる内部処理のために国別コードにマップされた POSIX ロケールの部分。

スレッド (thread). (1) いくつかのオペレーティング・システムでは、プロセス内の最小実行単位。 (2) アプリケーションの接続を記述し、その過程をトレースし、資源機能を処理し、そしてその DB2 UDB (OS/390 版) 資源とサービスへのアクセス可能性の範囲を区切る DB2 UDB (OS/390 版) 構造。 DB2 UDB (OS/390 版) 機能は、大部分がスレッド構造下で実行する。「*接続スレッド* (allied thread)」および「*データベース・アクセス・スレッド* (database access thread)」も参照。

3 部分名 (three-part name). 表、視点、もしくは別名のフルネーム。ピリオドで区切られた、ロケーション名、許可 ID、およびオブジェクト名から構成される。

しきい値トリガー (threshold trigger). パフォーマンス変数の値がユーザー定義のしきい値を超えるか、それ未満になる場合に発生するイベント。しきい値トリガーの結果として起こりうるアクションには、以下のものがあります。

- アラート・ログ・ファイルの情報のログ。
- アラート・ログ・ウィンドウの情報の表示。
- オーディオ・アラームの生成。
- メッセージ・ウィンドウの発行。
- 事前定義されたコマンドまたはプログラムの呼び出し。

時刻 (time). 時刻を時、分、秒で示す 3 つの部分から成る値。

時刻期間 (time duration). 時間、分、秒を表す DECIMAL(6,0) の値。

タイマーオン (timeron). 資源のおおまかな相対見積もり、またはコストの測定単位。同じ照会に対して 2 つの計画を実行するためにデータベース・サーバーが要求する。見積もりで計算された資源には、重み付けしたプロセッサーおよび入出力のコストが含まれる。

タイムアウト (timeout). DB2 UDB (OS/390 版) サブシステムかアプリケーションのどちらかが、資源を利用できないために異常終了すること。インストールの指定では、DB2 UDB (OS/390 版) が始動後に IRLM サービスを待機する時間と、アプリケーション要求が利用できない資源を IRLM が待機する時間の両方を判別するために設定される。これらの時刻指定のどちらかを超過すると、タイムアウトが宣言される。

タイム・スタンプ (timestamp). 年、月、日、時、分、秒、およびマイクロ秒で表される日時で構成される、7 つの部分から成る値。

タイム・スタンプ期間 (timestamp duration). 年、月、日、時間、分、秒およびマイクロ秒を表す DECIMAL(20,6) の値。

Tivoli Storage Manager (TSM). 異種環境でストレージ管理およびデータ・アクセス・サービスを提供するクライアント / サーバー製品。 TSM は様々な通信の方式をサポートしており、ファイルのバックアップや保管を管理するための管理機能が備わっている。また、バックアップ操作をスケジュールする機能もある。

TM データベース (TM Database). 「トランザクション・マネージャー・データベース (Transaction Manager Database)」を参照。

TMP. OS/390 環境では、端末モニター・プログラム (Terminal Monitor Program)。

処理待ち (to-do). 回復単位の 1 つの状態。回復可能 DB2 UDB (OS/390 版) 資源に行った回復単位の変更が未確定であり、コミット調整プログラムの判断にしたがって、その変更を DASD 媒体に適用するかロールバックする必要があることを示している状態。

トークン (token). 計算言語の基本構文単位。トークンは、ブランク文字やストリング定数または区切り識別子に使われている文字以外の 1 つ以上の文字で構成される。

トポロジー経路指定サービス (TRS) (topology and routing services, TRS). トポロジー・データベースを管理し、経路を計算する APPN 制御点の構成要素。

TP. 「トランザクション・プログラム (transaction program)」を参照。

トレース (trace). DB2 UDB (OS/390 版) のモニター、監査、パフォーマンス、会計、統計、および保守容易性 (グローバル) 関連のデータをモニターし、収集する能力を提供する DB2 UDB (OS/390 版) 機能。

トランザクション (transaction). (1) ワークステーションとプログラム、 2 つのワークステーション、または 2 つのプログラム間での交換で、特定のアクションまたは結果が伴うもの。たとえば、顧客の預金の入力、顧客の貸借の更新など。「*作業単位* (unit of work)」の同義語。 (2) Net.Data の 1 回の呼び出し。持続している Net.Data を使用する場合は、 1 つのトランザクションが複数の Net.Data 呼び出しにまたがることがある。

トランザクション補償 (transaction compensation). 拒否されたコミット・トランザクションの影響を受けた行の復元を行うプロセス。コミット・トランザクションが拒否されると、トランザクションがコミットされる前の状態に行が復元される。

トランザクション・ロック (transaction lock). DB2 UDB (OS/390 版) では、SQL ステートメントの並行実行を制御するために使用されるロック。

トランザクション・マネージャー (transaction manager). トランザクションに識別子を割り当て、進行状況をモニターし、トランザクションの完了および障害回復の責任を持つ機能。

トランザクション・マネージャー・データベース (TM Database) (Transaction Manager Database, TM Database). 2 フェーズ・コミット (SYNCPOINT TWOPHASE) が DB2 データベースで使用される場合に、トランザクションをログに記録するために使用するデータベース。トランザクション障害の発生時には、TM データベース情報がアクセスされ、失敗したトランザクションに組み込まれていたデータベースが再同期化される。

トランザクション・プログラム (TP) (transaction program, TP). APPC を使用してパートナー・アプリケーション・プログラムと通信するアプリケーション・プログラム。

トランザクション・プログラム名 (transaction program name). SNA LU 6.2 の会話では、会話の相手方となるリモート論理装置のプログラムの名前。

トランスフォーメーション (transformation). データウェアハウスセンターで、データに対して行われる 1 回の操作。トランスフォーメーションのタイプには、ピボットおよびクレンジングがある。

トランスフォーマー (transformer). ウェアハウス・データに対して操作を行うプログラム。データウェアハウスセンターの提供するトランスフォーマーには 2 つのタイプがある。 1 つはデータに関する統計を 1 つまたは複数の表で提供する統計トランスフォーマー、もう 1 つは、分析用にデータを加工するウェアハウス・トランスフォーマーである。各ステップにはトランスフォーマーに対応するタイプがあり、1 つのプロセス内でさまざまなタイプのデータ操作を行う際に使用される。たとえば、消去ステップは Clean トランスフォーマーを使用する。

変換表 (transition table). トリガー修正の影響を受けた行の変換値を含む、名前が付けられた一時表。古い変換表には、修正が適用される前の行の値が含まれており、新しい変換表には、修正後の行の値が含まれている。

変換変数 (transition variable). FOR EACH ROW トリガーでのみ有効な変数。変換変数を用いて現在行の変換値にアクセスできる。古い変換変数は、修正が適用される前の行の値であり、新しい変換変数は、修正後の行の値である。

伝送制御プロトコル / インターネット・プロトコル (TCP/IP) (Transmission Control Protocol/Internet Protocol, TCP/IP). ローカル・エリア・ネットワークおよび広域ネットワークに対等通信接続性を提供する通信プロトコルのセット。

トリガー (trigger). (1) DB2 では、特定の SQL が実行されているときに、データベース・マネージャーが間接的に呼び出すデータベースのオブジェクト。 (2) DB2 UDB (OS/390 版) データベースに保管され、 DB2 UDB (OS/390 版) 表にある事象が発生したときに実行される一連の SQL ステートメント。

トリガー起動 (trigger activation). DB2 UDB (OS/390 版) では、トリガー定義の中で定義されているトリガー事象の実行時に発生するプロセスのこと。トリガー起動は、トリガー・アクション条件の評価と、トリガー SQL ステートメントの条件付き実行から構成される。

トリガー起動時間 (trigger activation time). DB2 UDB (OS/390 版) では、トリガー定義において、トリガーを起動すべき時が当該トリガー事象の前か後かという指示。

トリガー本体 (trigger body). DB2 UDB (OS/390 版) では、トリガーが起動され、そのトリガー・アクション条件が真であると評価されたときに実行される一連の SQL ステートメント。

トリガー連鎖 (trigger cascading). DB2 UDB (OS/390 版) では、1 つのトリガーのトリガー・アクションによって別のトリガーが起動されるときに発生するプロセス。

トリガー・アクション (triggered action). (1) トリガー事象が発生したときに実行されるプロセス。(2) DB2 UDB (OS/390 版) では、トリガーが起動されるときに実行される SQL 論理。トリガー・アクションは、任意選択の 1 つのトリガー・アクション条件と、その条件が真であると評価された場合にのみ実行される一連のトリガー SQL ステートメントとから構成される。

トリガー・アクション条件 (triggered-action condition). (1) トリガー・アクションで SQL ステートメントの実行を制御する検索条件。 (2) DB2 UDB (OS/390 版) で、トリガー・アクションの任意指定部分。このブール条件は WHEN 文節として現れ、当該トリガー SQL ステートメントを実行すべきかどうかを判別する際に DB2 が評価する条件を指定する。

トリガー SQL ステートメント (triggered SQL statements). DB2 UDB (OS/390 版) では、トリガーが起動され、そのトリガー・アクション条件が真であると評価されたときに実行される一連の SQL ステートメント。トリガー SQL ステートメントを、*トリガー本体* (trigger body) ともいう。

トリガー・イベント (trigger event). トリガー定義では、トリガーを実行させる更新操作 (INSERT、UPDATE または DELETE ステートメント)。

トリガー細分性 (trigger granularity). DB2 UDB (OS/390 版) におけるトリガーの特性で、トリガーが起動される時点を次のいずれかに決定する。

- それぞれのトリガー元 SQL ステートメントごとに 1 回のみ。
- SQL ステートメントが変更する各行ごとに 1 回。

トリガー・パッケージ (trigger package). DB2 UDB (OS/390 版) では、CREATE TRIGGER ステートメントの実行時に作成されるパッケージ。このパッケージは当該トリガーが起動されたときに実行される。

トリガー・イベント (triggering event). DB2 UDB (OS/390 版) では、トリガー定義の中で指定され、当該のトリガーの活動化のきっかけとなる操作のこと。トリガー・イベントは、トリガー操作 (INSERT、UPDATE、または DELETE) と、この操作の実行対象となるトリガー表とから構成される。

トリガー SQL 操作 (triggering SQL operation). DB2 UDB (OS/390 版) では、トリガー表に対して実行されたときにトリガーの起動のきっかけとなる SQL 操作。

トリガー表 (triggering table). DB2 UDB (OS/390 版) では、トリガーが作成される対象となる表。定義したトリガー・イベントがこの表に対して発生すると、トリガーが起動される。

切り捨て (truncation). メモリー容量または記憶容量を超える処理が行われた結果の一部を破棄するプロセス。

TSO. OS/390 環境では、タイム・シェアリング・オプション (Time-Sharing Option)。

TSO 接続機能 (TSO attachment facility). DSN コマンド処理プログラムおよび DB2I から構成される DB2 UDB (OS/390 版) 機能。 CICS または IMS 環境用に書かれていないアプリケーションは、TSO 接続機能下で実行することができる。

チューニング・パラメーター表 (tuning parameters table). 収集プログラムが使用するタイミング情報を含むソース・サーバーの表。以下の情報が含まれる。

- 変更データ表に行を保持する時間。
- 変更がデータベース・ログまたはジャーナルに保管される前の経過時間。
- 変更データを作業単位表にコミットする回数。

2 フェーズ・コミット (two-phase commit). 回復可能な資源および外部サブシステムがコミットされるときの 2 ステップ・プロセス。最初のステップでは、データベース・マネージャー・サブシステムがコミットできる状態にあることを確認するために、データベース・マネージャー・サブシステムがポーリングされる。すべてのサブシステムが肯定応答の場合、データベース・マネージャーはコミット命令を出す。

タイプ付きパラメーター・マーカー (typed parameter marker). 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。一般書式は次のとおり。

```
CAST(? AS data-type)
```

タイプ 1 索引 (type 1 indexes). DB2 (MVS/ESA 版) バージョン 4 以前のリリースで作成された、または第 4 版でタイプ 1 索引と指定されている索引。「*タイプ 2 索引* (type 2 indexes)」と対比。 DB2 UDB (OS/390 版) バージョン 7 では、タイプ 1 索引はもはやサポートされない。

タイプ 2 索引 (type 2 indexes). DB2 (OS/390 版) バージョン 6 より後のリリースで作成した索引、あるいはバージョン 4 またはバージョン 6 でタイプ 2 索引として指定した索引。「*タイプ 1 索引* (type 1 indexes)」と対比。

U

UDF. 「*ユーザー定義関数* (user-defined function)」を参照。

UDT. 「*ユーザー定義タイプ* (user-defined type)」を参照。

確定カーソル (unambiguous cursor). リレーショナル・データベースが応答セットでブロック化できるかどうかを判別できるようにするカーソル。 FOR FETCH ONLY または FOR READ ONLY で定義されているカーソルはブロック化で使用できるが、 FOR UPDATE で定義されているカーソルは使用できない。

アンバインド・セッション (UNBIND) (unbind session, UNBIND). 2 つの論理装置 (LU) 間のセッションを非活動化する要求。

非コミット読み取り (UR) (uncommitted read, UR). アプリケーションが他のトランザクションの非コミット変更にアクセスするのを可能にする分離レベル。他のアプリケーションが表を消去または更新しようとしない限り、アプリケーションは自分が読み取っている行以外のアプリケーションをロックしない。

非調整トランザクション (uncoordinated transaction). 1 つ以上の資源にアクセスするトランザクションで、そのコミットまたはロールバックはトランザクション・マネージャーで調整されていないもの。

基本視点 (underlying view). DB2 UDB (OS/390 版) では、別の視点が直接または間接に定義されている視点。

やり直し (undo). 回復可能な DB2 UDB (OS/390 版) リソースに対して回復単位が行った変更はバックアウトしなければならないことを示す、回復単位の状態。

ユニコード (Unicode). ISO 10646 標準のサブセットである国際文字コード化スキーム。各文字は固有の 2 バイト・コードで定義される。

固有制約 (unique constraint). 基本キーまたは固有索引のキー内の 2 つの値が同一になり得ないという規則。「*固有性制約* (uniqueness constraint)」ともいう。

固有索引 (unique index). 表に同一のキー値がないことを保証する索引。

固有キー (unique key). 値が同じものがないように制約されているキー。

回復単位 (unit of recovery). DB2 UDB (OS/390 版) のインスタンスなどの単一の資源管理プログラム内での一連の回復可能操作。「*作業単位* (unit of work)」と対比。

作業単位 (unit of work). アプリケーション・プロセスにおける回復可能な一連の操作。アプリケーション・プロセスは、常に単一の作業単位であるが、コミットまたはロールバック操作のため、アプリケーション・プロセス全体には複数の作業単位が関与する。 DB2 UDB (OS/390 版) *複数サイト更新* (multi-site update) 操作では、単一の作業単位に複数の*回復単位* (unit of recovery) を組み込むことができる。「*トランザクション* (transaction)」の同義語。

作業単位表 (unit-of-work table). データベース・ログまたはジャーナルから読み取ったコミット・レコードを含むソース・サーバーの複製制御表。レコードには、回復単位 ID が組み込まれている。この回復単位 ID は、作業単位表と変更データ表を結合させて、トランザクション間で整合性のある変更データを生成するために使用される。 DB2 では、作業単位表に、オプションとして相関 ID が組み込まれている。この相関 ID は、監査するときに役立つ。

ロック解除 (unlock). 以前にロックされたオブジェクトまたはシステム資源を解放して、それを DB2 UDB (OS/390 版) 内で一般に使用できるように戻すこと。

タイプ無しパラメーター・マーカー (untyped parameter marker). 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。その書式は単一の疑問符 (?) である。

更新規則 (update rule). データベース・マネージャーが強制する条件。列の更新が行われる前に満たされなければならない。

更新トリガー (update trigger). DB2 UDB (OS/390 版) で、トリガー用の SQL 操作 UPDATE を指定して定義したトリガーのこと。

アップストリーム (upstream). DB2 UDB (OS/390 版) では、同期点ツリーにあるノードで、他の回復または資源管理プログラムに加えて、2 フェーズ・コミットの実行の調整を行う。

UR. 「*非コミット読み取り* (uncommitted read)」を参照。

URE. DB2 UDB (OS/390 版) では、回復単位のエレメント (unit of recovery element)。

URID (回復単位) (URID, unit of recovery ID). DB2 UDB (OS/390 版) では、回復単位の先頭ログ・レコードの LOGRBA。 URID は、その回復単位の後続のすべてのログ・レコードにも現れる。

ユーザー・コピー表 (user copy table). DB2 複製では、ソース表の全部または一部と一致する内容を持つターゲット表。ユーザー・データ列のみを含む。

ユーザー定義データ・タイプ (UDT) (user-defined data type, UDT). 「*特殊タイプ* (distinct type)」を参照。

ユーザー定義特殊タイプ (user-defined distinct type). 「*特殊タイプ* (distinct type)」を参照。

ユーザー定義関数 (UDF) (user-defined function, UDF). データベース管理システムに定義されており、SQL 照会で参照できる関数。以下のいずれかの関数である。

- 外部関数。関数本体がプログラミング言語で書かれているもの。引き数はスカラー値で、スカラー結果は呼び出すたびに生成される。
- ソース関数。すでに DBMS に認識されている別の組み込み関数またはユーザー定義関数によって実装される。この関数は、スカラー関数または列 (集合) 関数のいずれかで、 MAX、AVG などの値のセットから単一の値を戻す。

ユーザー定義パフォーマンス変数 (user-defined performance variable). ユーザーにより作成され、パフォーマンス変数プロファイルに追加されたパフォーマンス変数。

ユーザー定義プログラム (user-defined program). ユーザーが提供してデータウェアハウスセンターに定義するプログラム。これとは対照的に、付属のプログラムは自動的にデータウェアハウスセンターに含まれて定義される。

ユーザー定義タイプ (UDT) (user-defined type, UDT). データベース・マネージャーにもともとあったものではなく、ユーザーにより作成されたデータ・タイプ。 DB2 UDB (OS/390 版) では、ユーザー定義タイプの代わりに、「*特殊タイプ* (distinct type)」という用語を使用する。

ユーザー・マッピング (user mapping). ユーザーが連合サーバーに接続するための許可と、ユーザーがデータ・ソースに接続するための許可との間の関係。

ユーザー表 (user table). DB2 複製では、複製ソースとして定義される前に、アプリケーションのために作成され使用される表。読み取り専用ターゲット表、整合した変更データ表、レプリカ、および行レプリカ表の更新を行うときのソースとして使用される。

UT. DB2 UDB (OS/390 版) で、ユーティリティー専用のアクセス。

UTC. 「*協定世界時 (UTC)* (Coordinated Universal Time, UTC)」を参照。

V

値 (value). (1) SQL で操作される最小データ単位。 (2) 列と行の交点での特定のデータ項目。

変数 (variable). 変更可能な値を指定するデータ要素。

可変関数 (variant function). 結果が入力パラメーター値だけでなく他の要素に依存するユーザー定義関数。同じパラメーター値で続けて呼び出しを行うと、異なる結果が生成される場合がある。「*非可変関数* (not-variant function)」と対比。

可変長ストリング (varying-length string). 文字ストリング、グラフィック・ストリング、またはバイナリー・ストリングで、長さが固定されていないが、長さの変更範囲はセット限界内である。「*可変長ストリング* (variable-length string)」ともいう。

バージョン (version). DB2 UDB (OS/390 版) では、同種のプログラム、DBRM、パッケージの集合のメンバー。
- プログラムのバージョンは、プログラムのプリコンパイルによって作成されたソース・コード。プログラムのバージョンは、プログラム名とタイム・スタンプ (整合性トークン) によって識別される。
- DBRM のバージョンは、プログラムのプリコンパイルによって作成される DBRM。 DBRM のバージョンは、対応するプログラムのバージョンと同じプログラム名およびタイム・スタンプによって識別される。
- パッケージのバージョンは、特定のデータベース・システム内の DBRM をバインドした結果である。パッケージのバージョンは、DBRM と同じプログラム名および整合性トークンによって識別される。
- LOB のバージョンは、ある時点における LOB 値のコピーである。 LOB のバージョン番号は、その LOB に関する補助索引項目の中に保管されている。

視点 (view). 照会によって生成されたデータからなる論理表。「*基礎表* (base table)」と対比。

視点チェック・オプション (view check option). DB2 UDB (OS/390 版) では、視点を介して挿入または更新される行のすべてがその視点の定義に従う必要があるかどうかを指定するオプション。視点チェック・オプションは、CREATE VIEW ステートメントの CASCADED CHECK OPTION、 WITH CHECK OPTION、または WITH LOCAL CHECK OPTION 文節を使用して指定することができる。

仮想記憶アクセス方式 (VSAM) (Virtual Storage Access Method, VSAM). 直接アクセス装置上の固定長または可変長レコードに対する、直接処理または順次処理のためのアクセス方式。 VSAM データ・セットまたはファイル内のレコードは、キー・フィールドによる論理順序 (キー順)、データ・セットまたはファイル内に書き込まれる物理的順序 (入力順)、または相対レコード番号を使用して編成することができる。

仮想記憶通信アクセス方式 (VTAM) (Virtual Telecommunications Access Method, VTAM). OS/390 環境では、IBM ライセンス・プログラムの 1 つで、通信を制御し、SNA ネットワーク内のデータの流れを制御するもの。

Visual Explain. データベース管理者およびアプリケーション・プログラマーがグラフィカル・インターフェースを使って、与えられた SQL ステートメントのアクセス・プランの詳細情報を表示および分析できるようにするツール。このツールで提供されたタスクは、コントロール・センター からアクセスできる。

VSAM. 「*仮想記憶アクセス方式* (Virtual Storage Access Method)」を参照。

VTAM. 「*仮想通信アクセス方式* (Virtual Telecommunication Access Method)」を参照。

W

ウェアハウス (warehouse). 戦略的な意思決定を支援する、サブジェクト指向の不揮発性データ集合。ウェアハウスは、ビジネス情報管理のためのデータ統合で中心的な役割を果たす。これは企業内のデータマートのためのデータ・ソースであり、企業データの共通の視点を提供する。

ウェアハウス・エージェント (warehouse agent). データウェアハウスセンターで、データの移動や変換を管理する実行時プロセス。

ウェアハウス・コントロール・データベース (warehouse control database). データウェアハウスセンター メタデータの保管に必要な制御表が入っている、データウェアハウスセンター データベース。

ウェアハウス・プログラム・グループ (warehouse program group). データウェアハウスセンターで、プログラム・オブジェクトを保持するコンテナー (フォルダー)。

ウェアハウス・ソース (warehouse source). 単一のデータベースからの表と視点から成るサブセット、または複数のファイルから成るセットで、データウェアハウスセンターにすでに定義されているもの。

ウェアハウス・ターゲット (warehouse target). 単一のデータベースからの表、索引、および別名から成るサブセットで、データウェアハウスセンターによって管理されるもの。

ウォーム・スタート (warm start). (1) 以前に初期化された入出力作業待ち行列の再使用を可能にする再始動。「*コールド・スタート* (cold start)」と対比。 (2) DB2 複製では、以前に初期設定された入出力作業待ち行列の再使用を可能にする収集プログラムの始動。

予約済みアドレス (well known address). ノード間の接続を確立するためにネットワーク上の特定ノードを固有に識別するときに使用されるアドレス。予約済みアドレスは、ネットワーク・アドレスと論理ノードで使用されるポートの組み合わせである。

WLM アプリケーション環境 (WLM application environment). 1 つ以上のストアード・プロシージャーと関連付けられる MVS ワークロード管理プログラムの属性。 WLM アプリケーション環境によって、指定した DB2 UDB (OS/390 版) ストアード・プロシージャーを実行するアドレス空間を判別する。

作業ファイル (work file). DB2 複製では、サブスクリプション・セットの処理中に変更適用プログラムが使用する一時ファイル。

ラッパー (wrapper). 連合データベース・システムで、データ・ソースと通信してデータを検索するために連合サーバーがルーチンを呼び出すメカニズム。ルーチンが入っているライブラリーを、「**ラッパー・モジュール** (wrapper module)」という。

操作員向け書き込み (WTO) (write to operator, WTO). 任意選択のユーザーがコードを提供するサービスで、エラーおよび (修正の必要のある) システム異常を伝えるメッセージをシステム・コンソール操作員に向けて発信することができる。

WTO. 「*操作員向け書き込み* (write to operator)」を参照。

WTOR. 応答付き操作員向け書き込み (write to operator with reply)。

X

XCF. 「*システム共通結合機能* (cross-system coupling facility)」を参照。

XID. 端末識別交換 (exchange station ID)。

XRF. 「*拡張回復機能* (extended recovery facility)」を参照。

付録R. DB2 ライブラリーの使用法

DB2 ユニバーサル・データベース ライブラリーは、オンライン・ヘルプ、ブック (PDF および HTML)、および HTML 形式のサンプル・プログラムから成っています。このセクションでは、ユーザーに提供される情報について紹介し、その入手方法を示します。

オンライン製品情報をご利用になるには、インフォメーション・センターを使用することができます。詳細については、1575ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。ここではタスク情報、DB2 ブック、トラブルシューティング情報、サンプル・プログラム、および Web の DB2 情報を見ることができます。

DB2 PDF ファイルおよびハードコピー版資料

DB2 情報

以下に示す表では、DB2 ブックを 4 つのカテゴリーに分類しています。

DB2 の手引きおよび解説書
これらの資料は、すべてのプラットフォームに共通の DB2 情報を含んでいます。

DB2 のインストールおよび構成の情報
これらの資料は、特定のプラットフォーム上の DB2 ごとに用意されています。たとえば、 OS/2、Windows、および UNIX ベースのプラットフォームで稼働するそれぞれの DB2 用に、別個の*概説およびインストール* 資料が用意されています。

プラットフォーム共通のサンプル・プログラム (HTML 形式)
これらのサンプルは、アプリケーション開発クライアントとともにインストールされるサンプル・プログラムの HTML 版です。これらのサンプルは参考用であり、実際のプログラムに代わるものではありません。

リリース情報
これらのファイルには、DB2 ブックには含められなかった最新の情報が記載されています。

インストール情報、リリース情報、およびチュートリアルは、製品 CD-ROM から HTML 形式で参照することができます。ほとんどの資料は、製品

CD-ROM から HTML 形式で表示できますし、 DB2 の資料 CD-ROM から Adobe Acrobat (PDF) 形式で表示し印刷することができます。 IBM にハードコピー版の資料を注文したい場合は、 1571ページの『印刷資料の注文方法』を参照してください。注文可能な資料については、以下の表をご覧ください。

OS/2 および Windows プラットフォームの場合、 HTML ファイルは sqllib¥doc¥html ディレクトリーにインストールできます。 DB2 情報はいくつかの言語で提供されています。しかし、すべての言語に翻訳されているわけではありません。ある言語で情報が提供されていない場合は、英語版の情報が提供されます。

UNIX プラットフォームの場合、言語ごとに異なる複数の HTML ファイルを doc/%*L*/html ディレクトリーにインストールできます。ここで、*%L* は地域を表しています。詳細については、適切な*概説*および*インストールの手引き* を参照してください。

DB2 ブックを入手して情報を利用するには、次のようなさまざまな方法があります。

- 1574ページの『オンライン情報の表示』
- 1579ページの『オンライン情報の検索』
- 1571ページの『印刷資料の注文方法』
- 1571ページの『PDF 資料の印刷』

表 147. DB2 情報

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
	DB2 の手引きおよび解説書情報		
管理の手引き	*管理の手引き: 計画* は、データベース概念について概説し、設計 (たとえば、論理および物理データベース設計) に関する情報を提供し、高い可用性について解説しています。	第 1 巻 SC88-8513 db2d1x70	db2d0
	管理の手引き: インプリメンテーション は、設計、データベースへのアクセス、監査、バックアップ、および回復などのインプリメンテーションについて説明しています。	第 2 巻 SC88-8511 db2d2x70	
	管理の手引き: パフォーマンス は、データベース環境について解説し、さらにアプリケーションのパフォーマンスの評価と調整の方法について説明しています。	第 3 巻 SC88-8512 db2d3x70	
管理 API 解説書	データベースの管理に使用できる DB2 アプリケーション・プログラミング・インターフェース (API) およびデータ構造について説明します。また、この資料は、アプリケーションから API を呼び出す方法も示します。	SC88-8514 db2b0x70	db2b0
アプリケーション構築の手引き	環境設定に関する情報を提供し、Windows、OS/2、および UNIX ベースのプラットフォームでの DB2 アプリケーションのコンパイル、リンク、実行の各ステップについて説明します。	SC88-8515 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	DB2 ユニバーサル・データベース製品をご使用中に発生する可能性のあるセンス・コード APPC、CPI-C、および SNA についての一般情報を提供します。 HTML 形式でのみご利用いただけます。	資料番号なし db2apx70	db2ap

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
アプリケーション開発の手引き	DB2 データベースにアクセスするアプリケーションを、組み込み SQL または Java (JDBC および SQLJ) を使用して開発する方法について説明します。さらに、ストアード・プロシージャーの作成方法、ユーザー定義関数の作成方法、ユーザー定義タイプの作成方法、トリガーの使用方法、区画化されている環境または統合されているシステムでのアプリケーションの開発方法などについて解説されています。	SC88-8516 db2a0x70	db2a0
コール・レベル・インターフェースの手引きおよび解説書	DB2 データベースにアクセスするアプリケーションを、 DB2 コール・レベル・インターフェース (Microsoft ODBC 仕様互換の呼び出し可能 SQL) を使用して開発する方法について説明します。	SC88-8517 db2l0x70	db2l0
コマンド解説書	コマンド行プロセッサーの使用法について説明し、データベースの管理に使用できる DB2 コマンドについて解説しています。	SC88-8518 db2n0x70	db2n0
コネクティビティー 補足	DB2 (AS/400 版)、DB2 (OS/390 版)、DB2 (MVS 版)、または DB2 (VM 版) を DRDA アプリケーション・リクエスターとして DB2 ユニバーサル・データベースとともに使用するためのセットアップ情報および参照情報を提供します。また、この資料は DRDA アプリケーション・サーバーを DB2 コネクト アプリケーション・リクエスターとともに使用する方法の詳細を示します。 HTML と PDF でのみ利用可能	資料番号なし db2h1x70	db2h1
データ移動ユーティリティー 手引きおよび解説書	データの移動を行う DB2 ユーティリティー (インポート、エクスポート、ロード、AutoLoader、および DPROP など) の使用法について説明しています。	SC88-8522 db2dmx70	db2dm

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウスセンター 管理の手引き	データウェアハウスセンターを使用してデータウェアハウスを構築および保守する方法を説明します。	SC88-8545 db2ddx70	db2dd
データウェアハウスセンター アプリケーション統合の手引き	プログラマーがアプリケーションをデータウェアハウスセンターおよび情報カタログ・マネージャーと統合するのに役立つ情報を提供します。	SC88-8546 db2adx70	db2ad
DB2 コネクト 使用者の手引き	DB2 コネクト製品の概念、プログラミング、および一般的な使用方法に関する情報を提供します。	SC88-8521 db2c0x70	db2c0
DB2 クエリー・パトローラー 管理の手引き	DB2 クエリー・パトローラー・システムの運用の概説を行い、運用および管理に関する詳細情報、および管理用グラフィカル・ユーザー・インターフェース・ユーティリティーについてのタスク情報を提供します。	SC88-8525 db2dwx70	db2dw
DB2 クエリー・パトローラー 使用者の手引き	DB2 クエリー・パトローラーのツールや関数の使用方法を説明します。	SC88-8527 db2wwx70	db2ww
用語集	DB2 およびその構成要素で使用される用語の定義を示します。 HTML 形式と SQL 解説書 で利用可能	資料番号なし db2t0x70	db2t0
イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	DB2 エクステンダーの一般情報について提供し、画像、音声、およびビデオ (IAV) エクステンダーの管理と構成について、および IAV エクステンダーを使用したプログラミングについて説明しています。さらに、参照情報、診断情報 (メッセージ解説)、およびサンプルも収録されています。	SC88-8609 dmbu7x70	dmbu7
情報カタログ・マネージャー 管理の手引き	情報カタログを管理するためのガイドです。	SC88-8547 db2dix70	db2di
情報カタログ・マネージャー プログラミングの手引きおよび解説書	情報カタログ・マネージャー用の体系化されたインターフェースの定義を示します。	SC88-8549 db2bix70	db2bi

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
情報カタログ・マネージャー 使用者の手引き	情報カタログ・マネージャー・ユーザー・インターフェースの使用に関する情報を提供します。	SC88-8548 db2aix70	db2ai
インストールおよび構成補足	プラットフォーム固有の DB2 クライアントの計画、インストール、およびセットアップのガイドです。この補足資料には、バインド、クライアント / サーバー通信の設定、 DB2 GUI ツール、DRDA AS、分散インストール、分散要求の構成、および異種データ・ソースへのアクセスについても説明されています。	GC88-8524 db2iyx70	db2iy
メッセージ解説書	DB2、情報カタログ・マネージャー、およびデータウェアハウスセンターから出されるメッセージとコードをリストし、取るべき処置を解説しています。	第 1 巻 GC88-8543 db2m1x70 第 2 巻 GC88-8544 db2m2x70	db2m0
OLAP Integration Server Administration Guide	OLAP Integration Server の Administration Manager 構成要素の使用方法を説明します。	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	標準の OLAP Metaoutline インターフェースを使用して (Metaoutline Assistant を使用するのではなく) OLAP metaoutline を作成しデータを取り込む方法を説明しています。	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	(Model Assistant ではなく) 標準的な OLAP Model Interface を使用して OLAP モデルを作成する方法を説明します。	SC27-0783 db2lpx70	n/a
OLAP のセットアップおよび使用者の手引き	OLAP スターター・キットの構成およびセットアップに関する情報を提供します。	SC88-8652 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Excel 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0786 db2epx70	db2ep

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	ロータス 1-2-3 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0785 db2tpx70	db2tp
レプリケーションの手引きおよび解説書	DB2 に付属の IBM レプリケーション・ツールの計画、構成、管理、および使用方法に関する情報を提供します。	SC88-8550 db2e0x70	db2e0
地理情報エクステンダー使用者の手引きおよび解説書	地理情報エクステンダーのインストール、構成、管理、プログラミング、およびトラブルシューティングに関する情報を提供します。また、地理情報データの概念についての重要事項を示し、地理情報エクステンダー固有の参照情報 (メッセージおよび SQL) を提供します。	SC88-8624 db2sbx70	db2sb
SQL 概説	SQL の概念を紹介し、構造体とタスクの例を多数提供しています。	SC88-8539 db2y0x70	db2y0
SQL 解説書	SQL の構文、セマンティクス、および言語規則について説明します。また、この資料には、各リリース間の互換性、製品の制限事項、およびカタログ・ビューも含まれます。	第 1 巻 SC88-8540 db2s1x70 第 2 巻 SC88-8540 db2s2x70	db2s0
システム・モニター 手引きおよび解説書	データベースおよびデータベース・マネージャーに関連したさまざまな情報を収集する方法を示します。この資料は、この情報を利用して、データベース活動の把握、パフォーマンス向上、および問題原因の判別を行う方法を説明しています。	SC88-8523 db2f0x70	db2f0

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
テキスト・エクステンダー *管理およびプログラミング*	DB2 エクステンダーの一般情報、テキスト・エクステンダーの管理および構成情報、およびテキスト・エクステンダーを使用したプログラミングの方法について解説します。この資料には、参照情報、診断情報 (メッセージ解説)、およびサンプルが含まれています。	SC88-8610 desu9x70	desu9
問題判別の手引き	エラーの原因の判別、問題からの回復、および DB2 カスタマー・サービスの支援の下での診断ツールの使用法を記載しています。	GD88-7271 db2p0x70	db2p0
新機能	DB2 ユニバーサル・データベース バージョン 7 の新しい機能および拡張機能について説明します。	SC88-8541 db2q0x70	db2q0
DB2 のインストールおよび構成の情報			
DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 コネクト エンタープライズ・エディションで、計画、移行、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8520 db2c6x70	db2c6
DB2 コネクト エンタープライズ・エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 コネクト エンタープライズ・エディションの計画、移行、インストール、構成、およびタスクに関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8519 db2cyx70	db2cy

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 コネクト パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システムの DB2 コネクト パーソナル・エディションで、計画、移行、インストール、および構成を行う場合のタスク情報を提供します。また、この資料はサポートされているすべてのクライアントのインストールおよびセットアップについても説明します。	GC88-8533 db2c1x70	db2c1
DB2 コネクト パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 コネクト パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8528 db2c4x70	db2c4
DB2 データ・リンク・マネージャー 概説およびインストール	AIX および Windows 32 ビット・オペレーティング・システムの DB2 データ・リンク・マネージャーで、計画、インストール、構成を行う場合の情報を提供します。	GC88-8532 db2z6x70	db2z6
DB2 エンタープライズ拡張エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 エンタープライズ拡張エディションの計画、インストール、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8530 db2v3x70	db2v3
DB2 エンタープライズ拡張エディション (Windows 版) 概説およびインストール	Windows 32 ビット・オペレーティング・システムの DB2 エンタープライズ拡張エディションで、計画、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8529 db2v6x70	db2v6

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール	OS/2 オペレーティング・システムでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8534 db2i2x70	db2i2
DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。 また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8536 db2ixx70	db2ix
DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール	Windows 32 ビット オペレーティング・システムの DB2 ユニバーサル・データベースで、計画、インストール、移行、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8537 db2i6x70	db2i6
DB2 パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8535 db2i1x70	db2i1
DB2 パーソナル エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 ユニバーサル・データベース・パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8538 db2i4x70	db2i4
DB2 クエリー・パトローラー インストールの手引き	DB2 クエリー・パトローラーのインストール情報を提供します。	GC88-8526 db2iwx70	db2iw

表 147. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
ウェアハウス・マネージャー *インストールの手引き*	ウェアハウス・エージェント、ウェアハウス・トランスフォーマー、および情報カタログ・マネージャーのインストール情報を提供します。	GC88-8572 db2idx70	db2id
プラットフォーム共通のサンプル・プログラム (HTML 形式)			
サンプル・プログラム (HTML)	DB2 のサポートするすべてのプラットフォームでのプログラム言語用に、サンプル・プログラム (HTML 形式) を提供します。これらのサンプル・プログラムは、参照用としてのみ提供されています。サンプルは、すべてのプログラミング言語で利用できるわけではありません。HTML サンプルが利用できるのは、 DB2 アプリケーション開発クライアントがインストールされている場合だけです。 プログラムの詳細については、*アプリケーション構築の手引き* を参照してください。	資料番号なし	db2hs
リリース情報			
DB2 コネクト リリース情報	DB2 コネクトの資料には含められなかった最新の情報が収録されています。	注 #2 を参照してください。	db2cr
DB2 インストール情報	DB2 ブックには含められなかったインストールに関する最新の情報が収録されています。	製品 CD-ROM からのみ利用できます。	
DB2 リリース情報	DB2 ブックには含められなかった DB2 製品とその機能に関する最新の情報が収録されています。	注 #2 を参照してください。	db2ir

注:

1. ファイル名の 6 桁目の文字 *x* は、その資料の言語を表します。たとえば、ファイル名 `db2d0e70` は、*管理の手引き* の英語版であることを示し、ファイル名 `db2d0f70` は同じ資料のフランス語版を示します。資料の言語を表すためにファイル名の 6 桁目で使用されている文字は以下のとおりです。

言語	識別子
ブラジル・ポルトガル語	b
ブルガリア語	u
チェコ語	x
デンマーク語	d
オランダ語	q
英語	e
フィンランド語	y
フランス語	f
ドイツ語	g
ギリシャ語	a
ハンガリー語	h
イタリア語	i
日本語	j
韓国語	k
ノルウェー語	n
ポーランド語	p
ポルトガル語	v
ロシア語	r
簡体字中国語	c
スロベニア語	l
スペイン語	z
スウェーデン語	s
繁体字中国語	t
トルコ語	m

2. DB2 ブックには含められなかった最新の情報が、「リリース情報」で HTML 形式および ASCII ファイルとして利用できます。 HTML 版は、インフォメーション・センターおよび製品 CD-ROM からご利用になれます。 ASCII ファイルの参照方法:
   - UNIX ベースのプラットフォームでは、ファイル Release.Notes を参照してください。このファイルは DB2DIR/Readme/*%L* ディレクトリーにあります。ここで *%L* は地域名を、 DB2DIR は以下のものを表します。
     - /usr/lpp/db2_07_01 (AIX の場合)
     - /opt/IBMdb2/V7.1 (HP-UX、DYNIX/ptx、Solaris、および Silicon Graphics IRIX の場合)
     - /usr/IBMdb2/V7.1 (Linux の場合)
   - これ以外のプラットフォームでは、ファイル RELEASE.TXT を参照してください。このファイルは、製品がインストールされているディレクトリーにあります。 OS/2 プラットフォームでは、 **IBM DB2** フォルダーをダブルクリックし、 **Release Notes** アイコンをダブルクリックすることもできます。

PDF 資料の印刷

資料のハードコピー版が必要な場合、 DB2 の資料 CD-ROM にある PDF ファイル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することができます。ライブラリー内の各資料のファイルについては、1561ページの表147 を参照してください。

Adobe Acrobat Reader の最新版は、Adobe の Web サイト http://www.adobe.co.jp/ から入手できます。

PDF ファイルは、DB2 の資料 CD-ROM に収録されており、ファイル拡張子 PDF が付いています。 PDF ファイルにアクセスするには以下のようにします。

1. DB2 の資料 CD-ROM を挿入します。 UNIX ベースのプラットフォームの場合は、 DB2 資料 CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
2. Acrobat Reader を起動します。
3. 以下に示すいずれかの位置から必要な PDF ファイルを開きます。
   - OS/2 および Windows プラットフォームでは:

     x:¥doc¥*language* ディレクトリー。ここで、*x* は CD-ROM ドライブを、*language* は 2 桁の言語を表す国コード (たとえば、EN は英語) を示します。
   - UNIX ベースのプラットフォームでは:

     CD-ROM の */cdrom*/doc/*%L* ディレクトリー。ここで、*/cdrom* は CD-ROM のマウント・ポイントを、 *%L* は地域名を表します。

さらに、PDF ファイルを CD-ROM からローカル・ドライブまたはネットワーク・ドライブにコピーし、そこから参照することもできます。

印刷資料の注文方法

ハードコピー版の DB2 ブックは、個別に注文することができます。資料を注文するには、IBM 承認の販売業者または営業担当員に連絡してください。

オンライン・ヘルプへのアクセス

すべての DB2 構成要素で、オンライン・ヘルプを利用できます。以下の表に、さまざまな種類のヘルプを示します。

ヘルプの種類	内容	利用方法
コマンド・ヘルプ	コマンド行プロセッサーのコマンド構文について説明します。	コマンド行プロセッサーの対話モードから、次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 コマンドのヘルプが表示されます。
クライアント構成アシスタントのヘルプ **コマンド・センターのヘルプ** **コントロール・センターのヘルプ** **データウェアハウスセンターのヘルプ** **イベント・アナライザーのヘルプ** **情報カタログ・マネージャーのヘルプ** **サテライト管理センターのヘルプ** **スクリプト・センターのヘルプ**	そのウィンドウまたはノートブックで実行できるタスクについて説明します。このヘルプは、知っておく必要のある概説および前提条件に関する情報を含みます。また、ウィンドウやノートブックの制御の使用方法を示します。	ウィンドウまたはノートブックから、**「ヘルプ (Help)」**押しボタンをクリックするか、または **F1** キーを押します。

ヘルプの種類	内容	利用方法

メッセージ・ヘルプ	メッセージの原因、および取るべき処置を説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。
		`? XXXnnnnn`
		ここで、*XXXnnnnn* は有効なメッセージ識別子を表します。
		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。
		一度に 1 画面分のメッセージ・ヘルプを表示させるには、次のように入力します。
		`? XXXnnnnn | more`
		メッセージ・ヘルプをファイルに保管するには、次のように入力します。
		`? XXXnnnnn > filename.ext`
		ここで、*filename.ext* はメッセージ・ヘルプを保管するファイルを表します。

SQL ヘルプ	SQL ステートメントの構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `help statement` ここで、*statement* は SQL ステートメントを表します。 たとえば、help SELECT と入力すると、 SELECT ステートメントのヘルプが表示されます。 **注:** UNIX ベースのプラットフォームでは、SQL ヘルプを利用できません。
SQLSTATE ヘルプ	SQL 状態およびクラス・コードについて説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? sqlstate` or `? 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` によって SQL 状態 08003 のヘルプが表示され、 `? 08` によってクラス・コード 08 のヘルプが表示されます。

オンライン情報の表示

この製品に付属のブックは、ハイパーテキスト・マークアップ言語 (HTML) ソフトコピー形式です。ソフトコピー形式では情報を検索または表示したり、ハイパーテキスト・リンクを利用して関連情報に移動したりすることができます。また、1 つの端末を超えてライブラリーを容易に共用することができます。

オンライン・ブックやサンプル・プログラムは、 HTML バージョン 3.2 仕様に準拠するすべてのブラウザーを使って表示できます。

オンライン・ブックまたはサンプル・プログラムは、次のようにして表示します。

- DB2 管理ツールを実行している場合、インフォメーション・センターを使用します。
- ブラウザーで、**ファイル (File) —>ページを開く (Open Page)** をクリックします。次のようなページを開いて、DB2 情報に関する説明とリンクを表示してください。
  - UNIX ベースのプラットフォームでは、以下のページを開きます。

    INSTHOME/sqllib/doc/*%L*/html/index.htm

    ここで *%L* はロケール名です。
  - その他のプラットフォームでは、以下のページを開きます。

    sqllib¥doc¥html¥index.htm

    パスは DB2 がインストールされているドライブです。

    インフォメーション・センターをインストールしていない場合、 **DB2 Information** アイコンをダブルクリックしてページを開くことができます。このアイコンは、ご使用のシステムに応じて、製品のメイン・フォルダー内または Windows 「スタート」メニューにあります。

Netscape ブラウザーのインストール

システムに Web ブラウザーがインストールされていない場合、製品の箱の中にある Netscape CD-ROM から Netscape をインストールすることができます。インストールに関する詳細な説明については、以下を参照してください。

1. Netscape CD-ROM を挿入します。
2. UNIX ベースのプラットフォームでは、CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。

3. インストールの手順については、 CDNAV*nn*.txt ファイルを参照します。ここで、*nn* は 2 桁の言語識別子を表します。ファイルは CD-ROM のルート・ディレクトリーにあります。

インフォメーション・センターを使用した情報へのアクセス

インフォメーション・センターを使用すると、DB2 製品情報にすばやくアクセスすることができます。インフォメーション・センターは、DB2 管理ツールを使用できるすべてのプラットフォームで利用できます。

インフォメーション・センターは「インフォメーション・センター (Information Center)」アイコンをダブルクリックすることによってオープンできます。このアイコンのある場所はシステムによって異なります。メイン・プロダクト・フォルダーか Windows の**「スタート」**メニューのどちらかです。

Windows プラットフォームの DB2 では、ツールバーおよび**ヘルプ**・メニューを使用して、インフォメーション・センターにアクセスすることもできます。

インフォメーション・センターは 6 種類の情報を提供します。適切なタブをクリックすると、種類ごとに提供されているトピックが表示されます。

タスク (Tasks)
DB2 を使用して実行できる主要なタスク。

参照 (Reference)
DB2 参照情報 (キーワード、コマンド、API など)。

ブック (Books)
DB2 ブック。

トラブルシューティング (Troubleshooting)
エラー・メッセージのカテゴリーと、メッセージに対する回復処置。

サンプル・プログラム (Sample Programs)
DB2 アプリケーション開発クライアントに付属のサンプル・プログラム。 DB2 アプリケーション開発クライアントをインストールしていない場合、このタブは表示されません。

Web
WWW 上にある DB2 情報。この情報にアクセスするには、ご使用のシステムから Web への接続が必要です。

リストから項目を 1 つ選択すると、インフォメーション・センターはビューアーを立ち上げて情報を表示します。選択した情報の種類に応じて、ビューアーはシステム・ヘルプ・ビューアー、エディター、または Web ブラウザーです。

インフォメーション・センターには検索機能が備わっており、リストを参照せずに特定のトピックを探すことができます。

テキストの全検索を行うには、インフォメーション・センター内のハイパーテキスト・リンク「**DB2 オンライン情報の検索 (Search DB2 Online Information)**」検索フォームに従います。

通常、HTML 検索サーバーは自動的に始動します。 HTML 情報の検索がうまくいかない場合は、以下の方法の 1 つを使用して、検索サーバーを始動しなければならない場合もあります。

Windows では

「スタート」をクリックし、**「プログラム」—>「IBM DB2」—>「Information」—>「Start HTML Search Server」**を選択します。

OS/2 では

「DB2 (OS/2 版)」フォルダーをダブルクリックして、**「Start HTML Search Server」**アイコンをダブルクリックします。

HTML 情報の検索でこの他の問題が発生した場合は、リリース情報を参照してください。

注: 検索機能は、Linux、DYNIX/ptx、および Silicon Graphics IRIX 環境では利用できません。

DB2 ウィザードの使用

ウィザードを使用すると、各タスクをステップごとに進めることによって、さまざまな管理タスクを遂行することができます。ウィザードは、コントロール・センターおよびクライアント構成アシスタントを通して使用できます。以下の表では、ウィザードとその目的をリストしています。

注: データベース作成、索引作成、複数サイト更新の構成、およびパフォーマンス構成ウィザードは、区分化データベース環境で使用できます。

ウィザード	内容	利用方法
データベース追加 (Add Database)	クライアント・ワークステーション上にデータベースのカタログを作成します。	クライアント構成アシスタントから、「**追加 (Add)**」をクリックします。

ウィザード	内容	利用方法
データベース・バックアップ (Back up Database)	バックアップ計画を決定、作成、およびスケジュールします。	「コントロール・センター (Control Center)」からバックアップするデータベースを右クリックし、**「バックアップ (Backup)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
複数サイト更新の構成 (Configure Multisite Update)	複数サイト更新、分散トランザクション、または 2 フェーズ・コミットを構成します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「複数サイト更新 (Multisite Update)」**を選択します。
データベース作成 (Create Database)	データベースを作成し、いくつかの基本的な構成タスクを実行します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「作成 (Create)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
表作成 (Create Table)	基本的なデータ・タイプを選択して、表の基本キーを作成します。	「コントロール・センター (Control Center)」から、**「表 (Tables)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表 (Table Using Wizard)」**を選択します。
表スペース作成 (Create Table Space)	新しい表スペースを作成します。	「コントロール・センター (Control Center)」から、**「表スペース (Table Spaces)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表スペース (Table Space Using Wizard)」**を選択します。
索引作成 (Create Index)	すべての照会について、作成すべき索引および除去すべき索引を提案します。	「コントロール・センター (Control Center)」から、**「索引 (Index)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する索引 (Index Using Wizard)」**を選択します。

ウィザード	内容	利用方法
パフォーマンス構成 (Performance Configuration)	ビジネス要件に適合するように構成パラメーターを更新して、データベースのパフォーマンスを調整します。	「コントロール・センター (Control Center)」から、調整したいデータベースを右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。 区分データベース環境では、「Database Partitions」視点から、調整したい最初のデータベース区画を右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。
データベース復元 (Restore Database)	障害の後、データベースを回復します。どのバックアップを使用し、どのログを再生するかの判別を支援します。	「コントロール・センター (Control Center)」から復元するデータベースを右クリックし、**「復元 (Restore)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。

文書サーバーのセットアップ

デフォルトでは、DB2 情報はローカル・システムにインストールされます。つまり、DB2 情報にアクセスする必要のある各担当者が同じファイルをインストールする必要があります。 DB2 情報を 1 か所に格納するには、次のようにします。

1. ¥sqllib¥doc¥html のすべてのファイルとサブディレクトリーを、ローカル・システムから Web サーバーにコピーします。各ブックには独自のサブディレクトリーがあり、そのブックを構成する必要な HTML および GIF ファイルが入っています。ディレクトリー構造は常に同じ状態に保つ必要があります。
2. Web サーバーを構成して、ファイルを新しい場所で検索するようにします。さらに詳しい情報については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。
3. インフォメーション・センターの Java バージョンをご使用の場合は、すべての HTML ファイルのベース URL を指定できます。この URL はブックのリストに使用してください。

4. 資料ファイルが表示されるようになったなら、よく使うトピックにはブックマークを付けておいてください。ブックマークを付けるページは、たとえば以下のものがあります。
   - ブックのリスト
   - 頻繁に使用されるブックの目次
   - 頻繁に参照する情報 (たとえば、ALTER TABLE トピックなど)
   - 検索フォーム

中央のマシンから DB2 ユニバーサル・データベース オンライン文書ファイルを提供する方法については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。

オンライン情報の検索

HTML ファイルの情報を検索するには、以下の方法のどれか 1 つを使用してください。

- 最上部にある**「検索 (Search)」**をクリックします。検索フォームを使用して特定のトピックを見つけます。この機能は、 Linux、DYNIX/ptx、または Silicon Graphics IRIX 環境ではご利用になれません。
- 最上部にある**「索引 (Index)」**をクリックします。索引を使用して、ブック内の特定のトピックを見つけます。
- HTML 資料またはヘルプの目次あるいは索引を表示してから、 Web ブラウザーの検索機能を利用して資料内の特定のトピックを見つけます。
- Web ブラウザーのブックマーク機能を使用して、特定のトピックにすばやく戻ります。
- インフォメーション・センターの検索機能を使用して、特定のトピックを検索します。詳しくは、1575ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。

付録S. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品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。これらのプログラムまたは製品に代えて、IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働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 および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & 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。

本書に含まれる情報には、技術的に不正確なもの、または誤植が含まれる場合があります。これらに対する変更は、定期的に行われます。これらの変更は、資料の改訂版に含まれます。 IBM は、本書で説明している製品、プログラムに対して、予告なく改良、変更を加える場合があります。

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、お客様になんら義務も負わせない適切な方法で、使用もしくは配布することがあります。

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
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な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。

本書に含まれるパフォーマンス・データは、制御された環境下で決定されています。したがいまして、その他の稼働環境で得られる結果とは、かなり異なる可能性もあります。一部の測定値は、開発中のシステムを使用している場合があり、これらの測定値が一般的に提供可能なシステムで同様の数値になることを保証するものではありません。さらに、一部の測定値が推定されたものもあります。実測値と異なる場合があります。本書のユーザーは、使用される特定の環境での該当データを確認してください。

IBM 以外の製品については、当該製品の提供者から直接、出版されている資料または一般公開されている情報から入手しました。 IBM は、これらの製品についてはテストを行っておらず、これらの IBM 以外の製品に関する性能、互換性またはその他の主張について確認することはできません。 IBM 以外の製品の機能に対する質問は、それぞれの製品提供者にお問い合わせください。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみ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が、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、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。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

著作権：

本書に含まれる情報には、サンプル・アプリケーション・プログラムがソース言語の形式で含まれており、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技法を示しています。お客様は、これらのサンプル・プログラムが書かれているオペレーティング・プラットフォームでアプリケーション・プログラミング・インターフェースが実行可能となるためのアプリケーション・プログラムを開発、使用、販売または配布もしくは転送する目的のためにのみ、サンプル・プログラムを、IBM に対する別途料金を支払うことなく、複製、変更、配布または転送することができます。これらのサンプルは、すべての条件下で十分にテストを行っていません。したがいまして、IBM は、これらのプログラムの信頼性、実用性または機能について、いかなる保証も負いません。

サンプル・プログラムまたはその改変版の複製物には、全部複製か部分複製かを問わず、次の著作権表示を必ず行うものとします。

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

次のものは、他社の商標または登録商標です。

Tivoli および NetView は、米国およびその他の国における Tivoli Systems Inc. の商標です。

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。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国ならびに他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている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または登録商標です。

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]

［ワ行］

［数字］

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

S

T

U

V

W

Y

［特殊文字］

IBM と連絡をとる

技術上の問題がある場合は、時間をとって*問題判別の手引き* に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 **IBM Software Support Handbook** の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

Printed in Japan

SC88-8540-00

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