Oracle Lite

SQL リファレンス

リリース 4.0.1

2000 年 10 月

部品番号 : J02405-01

ORACLE®

Oracle Lite SQL リファレンス , リリース 4.0.1

部品番号 : J02405-01

原本名：Oracle Lite SQL Reference, Release 4.0.1

原本部品番号：A86146-01

Printed in Japan.

目次

2 SQL 演算子

3 SQL 関数

4 SQL コマンド

A Oracle Lite キーワードと予約語

B Oracle Lite データ型

C Oracle Lite リテラル

D SQL*Plus

E 索引作成オプション

F 構文図の規則

用語集

索引

はじめに

ここでは、『Oracle Lite SQL リファレンス』の構成について説明します。このマニュアルでは、Oracle Lite データベースにある情報を管理するために使用される構造化問合せ言語（SQL）について説明します。

Oracle SQL は、ANSI（American National Standards Institute、米国規格協会）および ISO（International Standards Organization、国際標準化機構）によって定義された SQL-92 標準のスーパーセットです。

このマニュアルは次の章から構成されます。

第 1 章「SQL の使用方法」	Oracle Lite SQL と Oracle SQL との間の重要な違いを挙げて解説し、SQL の使用方法を説明します。
第 2 章「SQL 演算子」	次のタイプの Oracle Lite SQL 演算子を説明します。算術、文字、比較、論理、集合、その他。
第 3 章「SQL 関数」	次のタイプの Oracle Lite SQL 関数を説明します。数値、文字、数値を返す文字、日付、変換、グループ、その他。
第 4 章「SQL コマンド」	次のタイプの Oracle Lite SQL コマンドを説明します。データ定義言語（DDL、Data Definition Language）、データ操作言語（DML、Data Manipulation Language）、トランザクション制御、句および疑似列。
付録 A「Oracle Lite キーワードと予約語」	Oracle Lite のキーワードおよび予約語のリストと説明があります。
付録 B「Oracle Lite データ型」	Oracle Lite のデータ型のリストと説明があります。
付録 C「Oracle Lite リテラル」	Oracle Lite のリテラルのリストと説明があります。
付録 D「SQL*Plus」	SQL*Plus の簡単な概要があります。

付録 E「索引作成オプション」	CREATE INDEX コマンドに追加されたオプションを説明します。
付録 F「構文図の規則」	『Oracle Lite SQL リファレンス』で使用される構文図とその規則を説明します。

1

SQL の使用方法

この章では、Oracle Lite での SQL の使用方法を説明します。説明する内容は次のとおりです。

- SQL の概要
- Oracle Lite SQL と Oracle SQL の比較
- Oracle Lite SQL 規則
- ODBC SQL 構文規則
- Oracle Lite データベース・オブジェクトのネーミング規則
- 書式
- SQL 条件の指定
- 式の指定
- Oracle Lite SQL のデータ型およびリテラル
- SQL 文内のコメント
- SQL*Plus

SQL の概要

Oracle Lite は、SQL（Structured Query Language、構造化問合せ言語）データベース言語を使用して、データを格納および取得します。これには、次のカテゴリの SQL 文が含まれます。

- DDL（Data Definition Language、データ定義言語）

  スキーマ、表、列、ビュー、順序などのデータベース・オブジェクトの作成、変更、および削除に使用されます。たとえば、ALTER、CREATE、DROP、GRANT および REVOKE コマンドを使用する文です。

- DML（Data Manipulation Language、データ操作言語）

  既存のスキーマ・オブジェクト内のデータの問合せや操作に使用されます。たとえば、コマンドの SELECT、INSERT、UPDATE、および DELETE コマンドを使用する文です。

- TCL（Transaction Control Language、トランザクション制御言語）

  DML 文内の変更を管理します。たとえば、COMMIT、ROLLBACK および SAVEPOINT コマンドを使用する文です。

- 句

  コマンドを変更するコマンドのサブセットです。Oracle Lite は、CONSTRAINT および DROP 句をサポートします。

- 疑似列

  コマンドから生成された値で、表の列のように動作しますが、実際は表に格納されていない値です。Oracle Lite は、LEVEL および ROWNUM 疑似列をサポートします。

- 関数

  データに対して変換または集計を操作します。たとえば、TO_DATE は日付列を特定の書式に変換し、SUM は列内のすべての値を合計します。

例

このリファレンスには、SQL 文の例が提供されています。例はすべて、デフォルトの Oracle Lite データベース・オブジェクトをベースにしています。

Oracle SQL と SQL-92

Oracle Lite は、デフォルトの SQL 言語として Oracle SQL を使用します。Oracle SQL は、計算結果と日付データを SQL-92 とは違った方法で操作します。次に、Oracle SQL と SQL-92 の相違点をリストします。

Oracle SQL	SQL-92
除算は、3.333 のように、倍精度の結果になります。たとえば、8/3 の結果は、2.666 です。	除算は、3 のように、オペランドのデータ型になります。たとえば、8/3 の結果は、2 です。
DATE データ型は、完全なタイム・スタンプ情報を格納しますが、日付部分のみを表示します。	DATE データ型は、日付を格納し表示しますが、タイム・スタンプ情報は格納も表示もしません。

Oracle Lite は Oracle SQL を使用しますが、デフォルトで次に示す SQL-92 機能のいくつかをサポートします。

- 列データ型 : TIME、TIMESTAMP、TINYINIT および BIT
- CASE 式
- CAST 式

Oracle Lite での SQL-92 の実行

前の項で述べたように、Oracle Lite は、デフォルトで Oracle SQL を使用します。しかし、Oracle SQL ではなく SQL-92 をデフォルトでサポートする場合、**POLITE.INI** ファイル内の SQL 互換パラメータを SQL-92 に変更できます。パラメータを変更するには、次の行を **POLITE.INI** ファイルに追加します。

```
SQLCOMPATIBILITY=SQL92
```

POLITE.INI ファイルの詳細は、『Oracle Lite ユーザーズ・ガイド』を参照してください。

Oracle Lite SQL と Oracle SQL の比較

Oracle Lite データベースによってサポートされる SQL 言語は、Oracle によってサポートされる SQL 言語のサブセットです。Oracle Lite は、追加の SQL-92 データベース・オブジェクト、関数およびコマンドをいくつかサポートします。

オブジェクト

次に、Oracle Lite でサポートされるデータベース・オブジェクトと、Oracle でサポートされるデータベース・オブジェクトの相違点をリストします。詳細は、「Oracle Lite データベース・オブジェクトのネーミング規則」を参照してください。

Oracle Lite でサポート	Oracle でサポート
表、ビュー、索引、順序、スキーマおよびスナップショット。	すべてのデータベース・オブジェクト。
列、索引、表およびスキーマに対する名前識別子は、最大 128 文字まで。ユーザー名識別子は、最大 30 文字まで。	名前識別子は、最大 31 文字まで。

演算子

第 2 章「SQL 演算子」に、Oracle Lite データベースによってサポートされる演算子をリストしています。一般に Oracle Lite データベースは、Oracle がサポートする演算子をすべてサポートします。

データ型関連の相違点を除いて、対応する演算子は常に同じように機能します。

関数

第 3 章「SQL 関数」に、Oracle Lite データベースによってサポートされる関数をリストしています。次の関数は、Oracle と Oracle Lite では異なる結果になります。

関数	Oracle Lite でサポート	Oracle でサポート
ROWID	16 文字長	18 文字長
TO_CHAR	「nlsparams」を受け入れない	「nlsparams」を受け入れる
TO_DATE	「nlsparams」を受け入れない	「nlsparams」を受け入れる
TO_NUMBER	「nlsparams」を受け入れない	「nlsparams」を受け入れる

コマンド

第 4 章「SQL コマンド」に、Oracle Lite データベースによってサポートされるコマンドをリストしています。Oracle Lite は、次のタイプのコマンドを追加サポートします。

コマンド・タイプ	Oracle Lite でサポート
埋込み SQL コマンド	WHENEVER
埋込み SQL の中で使用されるコマンド	DELETE、INSERT、SELECT、UPDATE

Oracle コマンドの中には、Oracle Lite でさらに機能が制限されるものがあります。次に、Oracle Lite でサポートされていない Oracle コマンド・パラメータを表示します。

コマンド	Oracle Lite でサポートされていない要素
CREATE TABLE	表および列制約の索引句
	表および列制約の例外句
	物理編成句
	列および表の遅延オプション
CREATE TRIGGER	ビュー上
	OR REPLACE
	INSTEAD OF
	REFERENCING OLD
	REFERENCING NEW
	WHEN
	OR
ALTER TABLE	RENAME
ALTER INDEX	索引の名前変更オプション
	索引の再作成オプション
SET TRANSACTION	READ ONLY
	READ WRITE
UPDATE	複数の列を選択する副問合せを含む設定句
	更新された行の行 ID が返される位置を返す句
TO_CHAR	日付値からタイム・スタンプを抽出するときに使用

注意： Oracle および Oracle Lite の副問合せには、相違点がある場合があります。

Oracle Lite は、次のコマンドと句をサポートしません。

- 次のデータベース・オブジェクトに関連するコマンド
  - クラスタ
  - データベース・リンク
  - ストアド・ファンクションと Java ストアド・プロシージャ以外のストアド・プロシージャ
  - パッケージ
  - プロファイル
  - ロールバック・セグメント
  - スナップショット・ログ
  - 表領域
- PCTFREE などの物理データ記憶域句

DDL（データ定義言語）

Oracle Lite は、領域管理、表領域および INITRANS をサポートしません。

Oracle Lite DDL は、Oracle と同じように実行されたときはコミットしませんが、現在のトランザクションの一部としてコミットします。

データ型

Oracle Lite は、Oracle より多くのデータ型をサポートします。Oracle Lite で Oracle と同様の結果を得るには、NUMBER を使用して精度とスケールを指定してください。

Oracle では、返すデータ型とその表示を想定しています。Oracle Lite では文中に特定の CAST （*one_datatype* AS *another_datatype*）を必要とする場合でも、結果を自動的に生成することがあります。異種マシンへの移植性が必要な場合は、INT、FLOAT および DOUBLE の使用を避けてください。Oracle がこれらのデータ型を特定の NUMBER データ型にマップするのに対し、Oracle Lite はシステム固有の実装タイプを使用します。

標識変数

Oracle Lite は、32 ビットの LONG 標識変数整数を使用します。Oracle は、16 ビットの SHORT 標識変数整数を使用します。

算術操作の間のデータ精度

Oracle データベースは、列に格納する結果の小数点以下の桁数を決めるときに、割当ての左辺のデータ型を見ます。Oracle Lite は、SQL-92 規則に従い、割当ての右辺からの精度の最大桁数だけを提供します。

データ・ディクショナリ

Oracle Lite データ・ディクショナリは、Oracle データ・ディクショナリと異なります。Oracle Lite は、`ALL_TABLES` や `ALL_INDEXES` をはじめとする共通に使用される多くのシステム・ビューを提供します。

Oracle Lite にはインストールされない表

表 `system.product_privs` は、製品ユーザー・プロファイルとして Oracle データベース内に存在しますが、Oracle Lite 内には存在しません。このため、SQL*Plus は Oracle Lite に最初に接続するときに、エラー・メッセージを発行します。このエラー・メッセージは無視できます。

メッセージ

Oracle Lite は、Oracle データベースが SQL*Plus コマンドに対して生成するものと同じメッセージを生成するとは限りません。エラー・コードも異なる可能性があります。アプリケーションは、エラー発生を認識するために、特定のエラー・コードやメッセージ・テキストに依存しないようにする必要があります。

順序

Oracle Lite は、順序文中の CYCLE および CACHE 句はサポートしません。一部の環境では、順序番号もまた ROLLBACK の対象となります。

PL/SQL

Oracle Lite は PL/SQL をサポートしません。しかし、Oracle Lite は Java で書かれたストアド・プロシージャおよびトリガーをサポートします。

SQL 関数

Oracle Lite は三角関数、SOUNDEX およびビット操作をサポートしません。

ロックおよびトランザクション

Oracle Lite は、SELECT を最初に使用するときにトランザクションを開始します。分離レベルによっては、ある接続で SELECT を使用すると、同じ表に対する UPDATE をロックする可能性があります。ロックを解除して UPDATE が実行できるようにするには、SELECT の後で COMMIT を行う必要があります。

Oracle Lite SQL 規則

SQL 文を発行するとき、コマンドの定義内で空白を置くところに、タブ、改行、空白またはコメントを 1 つ以上含めることができます。Oracle Lite SQL は、次の 2 つの文を同じ方法で評価します。

```
SELECT ENAME,SAL*12,MONTHS_BETWEEN(HIREDATE,SYSDATE) FROM EMP;

SELECT ENAME,
   SAL * 12,
      MONTHS_BETWEEN( HIREDATE, SYSDATE )
   FROM EMP;
```

予約語、キーワード、識別子およびパラメータでは、大 / 小文字は区別されません。しかし、テキスト・リテラルおよび引用符に入れられた名前では、大 / 小文字が区別されます。第 3 章「SQL 関数」および第 4 章「SQL コマンド」にある構文説明を参照してください。

SQL 文の構文

SQL 構文定義は、次の規則を使用します。SQL 構文定義は常に等幅のテキストで表されます。

大文字

SELECT

表示されているとおりに入力する必要のあるリテラル・テキストを示します。

イタリック体小文字

table_name

該当する値または式で置き換えられるプレースホルダを示します。引用符など、置換値または式が必要とするその他のデリミタが表示されています。

大カッコでの囲み

[PUBLIC] OR [MAXVALUE | NOMAXVALUE]

オプションの項目または句を示します。複数の項目または句は、垂直バーで区切ります。大カッコまたは垂直バーは入力しないでください。

中カッコ

{ENABLE | DISABLE | COMPILE}

複数の選択必須項目は、垂直バーで区切って中カッコで囲みます。中カッコまたは垂直バーは入力しないでください。

垂直バー

{IDENTITY | NULL} OR [MAXVALUE integer | NOMAXVALUE]

垂直バーは中カッコ { } で囲まれた引数または大カッコ [ ] で囲まれたオプションの引数の、2 つ以上の選択肢を区切ります。垂直バー、中カッコまたは大カッコは入力しないでください。

省略記号

[, *column*] ...

同じ書式で表された引数を繰り返して指定できることを示します。省略符号は入力しないでください。

下線

[ASC | DESC]

垂直バーで区切られているオプションをどれも指定しなかった場合に使用されるデフォルト値を示します。

太字

PCTFREE

示されているとおりに入力する必要のあるキーワードを表します。

文頭のコロン

:*integer_value*

埋込み SQL 構文で、該当するホスト変数参照により置き換えられるプレースホルダを示します。文頭のコロンはホスト変数参照に含めます。

SQL 表

データベースは、1 つ以上のデータベース・ファイル、または ODBC および SQL-92 の「カタログ」から作成できます。SQL 内の基本的な記憶単位は、列に構成されたデータ行から成る表です。表、ビューおよび索引をはじめとするデータベース・オブジェクトは、すべてユーザー名またはスキーマによって所有されます。Oracle Lite のデフォルトでは、表は、ユーザー・スキーマ、つまりログイン ID と同じ名前を持つスキーマの一部として作成されます。

SQL オブジェクト名

SQL でのオブジェクト名は、先頭を文字にする必要がありますが、数字や特殊文字の「_」と「$」を含めることができます。名前は、一般に大 / 小文字は区別されません。大 / 小文字が混在する名前は、二重引用符（" "）で囲んで使用できます。

オブジェクト名は、修飾子をピリオド「.」で区切って、オブジェクトが所属するカタログとスキーマで修飾できます。たとえば、次のとおりです。

```
production.payroll.emp.salary
```

この例は、`production` カタログ内の `payroll` スキーマが所有する `emp` 表の `salary` 列を参照しています。

SQL 演算子の優先順位

SQL 演算子の相対優先順位を次のリストに示します。リストの一番上の演算子の優先順位が一番高く（最初に評価され）、リストの一番下の演算子の優先順位が一番低くなります（最後に評価されます）。優先順位の等しい演算子は、左から右に評価されます。

1. +（単項演算子）、-（単項演算子）、PRIOR
2. *、/
3. +、-、||
4. すべての比較演算子
5. NOT
6. AND
7. OR

式の中にカッコを使用すると、演算子の優先順位を変更できます。カッコの中の式はカッコの外の式より先に評価されます。

SQL セッション

SQL 文の実行には、SQL セッションが存在することが必要です。アプリケーションは次の方法で SQL セッションを確立できます。

- SQL セッションが必要な SQL 文を発行する（デフォルト・セッションは暗黙的に確立されます）。
- `SQLConnect` または `SQLDriverConnect` ODBC コールを発行する。

SQL セッションは、次のいずれかが発生するとクローズされます。

- ODBC 内で `SQLDisconnect` API がコールされる。
- ODBC プログラムが終了する。

SQL トランザクション

SQL データベースは、トランザクションと呼ばれる論理作業単位でリクエストを処理します。トランザクションとは、データベースに対するすべての変更が最終的な形にされる前に、実行が成功して終了する必要のある一連の関連した操作です。

SQL トランザクションは、セッション内で DDL や DML 文が実行されると開始されます。トランザクションの間にエラーが発生しなければ、COMMIT コマンドを使用してトランザクションを終了できます。そして、データベースはこの操作を反映して変更されます。エラーが発生した場合、ROLLBACK コマンドを使用して変更を破棄できます。

Oracle Lite は、COMMIT コマンドが発行されるまで、DDL 文をコミットしません。Oracle は、即座にすべての DDL 文をコミットします。

プログラムからの SQL 文の発行

Oracle Lite データ型およびオブジェクト・クラスは、他のプログラミング言語と相互操作可能です。適切な ODBC または JDBC ドライバを使用してアプリケーション内からデータベースに接続している場合、ホスト言語で書かれた SQL 文を Oracle Lite に対して発行できます。

SQL と ODBC

Microsoft の ODBC（Open Database Connectivity）インタフェースは、異なるデータベース間の相互操作可能性を提供するために、コール・レベル・インタフェースを定義しています。ODBC は、次のことを可能にするインタフェース機能を指定しています。

- 異なるベンダーのデータベースに接続する
- 共通言語で書かれた SQL 文を準備し、実行する
- 問合せ結果をプログラムのローカル変数に取り入れる

Oracle Lite は、ODBC 2.0 のコール・レベル・インタフェース（CLI）をサポートします。Oracle Lite SQL は、必要なときに、文字列型から別のデータ型への暗黙的な型変換をサポートします。たとえば列 AGE のデータ型が INTEGER で、次の文を実行したとします。

```
UPDATE EMPLOYEE SET AGE = '30' WHERE NAME = 'John'
```

「30」は自動的に INTEGER 型に変換されます。

ODBC SQL 構文規則

データベース固有の SQL 構文ではなく ODBC SQL 構文を使用する理由は、主に 2 つあります。

第 1 に、ODBC 構文で作成された SQL 文は、ODBC 準拠のデータベース間で簡単に交換できます。ODBC SQL 構文には、特定のデータベースにとって意味のある機能を起動するキーワードや引数の多くが含まれていませんが、ODBC 構文で作成した SQL 文は ODBC 準拠のすべてのデータベース間で完全に移植できます。

第 2 に、馴染みのないデータベースに対して、ODBC SQL 構文を使用して SQL 文を実行できます。ODBC SQL 構文は、データベース自体の SQL 構文のようにデータベースのすべての機能を引き出すことはできませんが、一般的で重要なデータベース機能の多くを実行できます。

データベース固有の SQL 構文は、ODBC 経由でデータベースに接続しているときでも常に使用できます。これは、ODBC では接続済データベースに SQL 文がそのまま渡されるためです。

Oracle Lite データベース・オブジェクトのネーミング規則

この項には、Oracle Lite データベース・オブジェクトとその一部に対するネーミング規則がリストされています。

1. ユーザー名は、1 ～ 30 文字の長さにしてください。列、索引、表およびスキーマの名前は 128 文字までの長さにできます。Oracle Lite では名前の長さに制限はありませんが、30 文字に制限することをお薦めします。
2. 名前に引用符を含めることはできません。
3. 名前では大文字と小文字は区別されません。
4. 名前はアルファベット文字で開始する必要があります。
5. 名前には、英数字、_、$、# のみを含めることができます。ただし、$ と # は使用しないようにしてください。
6. 名前に、Oracle Lite の予約語は使用できません。
7. 「DUAL」という語は、オブジェクトの名前またはその一部に使用しないでください。

8. Oracle Lite SQL 言語には、特別の意味を持つ他のキーワードもあります。これらのキーワードは予約語ではないため、オブジェクトやオブジェクトの部分の名前として使用できます。ただし、キーワードを名前として使用すると、SQL 文が読みにくくなります。Oracle Lite キーワードのリストは、付録 A「Oracle Lite キーワードと予約語」を参照してください。
9. 名前はネームスペース内で一意にする必要があります。
10. 名前は二重引用符で囲むことができます。このような名前には、前述の 3 から 7 までの制約を無視して、どのような文字の組合せでも入れられます。
11. 名前にピリオド「.」を含めることはできません。

書式

「数値書式要素」および「日付書式要素」の項に、有効な数値書式または日付書式の作成に使用できる要素をリストします。書式は、SQL 関数、TO_DATE、TO_NUMBER、TO_CHAR および TRUNC に対する引数として使用できます。

数値書式要素

次に、Oracle Lite の数値書式をリストします。

要素	例	説明
9	9999	9 の個数は、返される有効桁数を示します。先行ゼロおよび値 0 には空白が返されます。
0	0000.00	空白ではなく、0 として先行ゼロまたは 0 の値を返します。
$	$9999	値の前にドル記号を入れます。
B	B9999	書式モデル内のゼロには関係なく、空白として値 0 を返します。
MI	9999MI	負の値の後に「-」を返します。正の値の場合は、後続のスペースが返されます。
S	S9999	正の値には「+」を返し、負の値には「-」を返します。
PR	9999PR	山カッコ < > の中に負の値を返します。正の値の場合は、前後にスペースをつけて返します。
D	99D99	小数点を返し、数値の整数部と小数部を区切ります。
G	9G999	グループ・セパレータを返します。
C	C999	ISO 通貨記号を返します。
L	L999	ローカルの通貨記号を返します。

要素	例	説明
,（カンマ）	9,999	カンマを返します。
.（ピリオド）	99.99	ピリオドを返し、数値の整数部と小数部を区切ります。
EEEE	9.999EEEE	値を科学表記法で返します。

日付書式要素

次に、Oracle Lite の日付書式をリストします。

要素	説明
SCC または CC	世紀。「S」によって紀元前の日付の先頭に「-」が付けられます。
YYYY または SYYYY	4 桁年。「S」によって紀元前の日付の先頭に「-」が付けられます。
IYYY	ISO 標準に基づく 4 桁の年表記。
YYY または YY または Y	年表記の最後の 3 桁、2 桁または 1 桁。
IYY、IY、または I	ISO 年表記の最後の 3 桁、2 桁または 1 桁。
Y,YYY	カンマ付きの年。
Q	年の四半期（1、2、3、4; 1 月～ 3 月 = 1）。
MM	月（01 ～ 12; 1 月 = 01）。
MONTH	月の名前。9 文字になるまで空白が埋め込まれます。
MON	月の名前の省略形。
WW	年間通算週（1 ～ 53）。第 1 週は、その年の 1 月 1 日で始まり、1 月 7 日で終了します。
IW	ISO 標準に基づく年間通算週（1 ～ 52 または 1 ～ 53）。
W	月間通算週（1 ～ 5）。第 1 週はその月の 1 日で始まり、7 日で終了します。
DDD	年間通算日（1 ～ 366）。
DD	月間通算日（1 ～ 31）。
D	曜日（1 ～ 7）。
DAY	曜日。9 文字になるまで空白が埋め込まれます。
DY	曜日の省略形。

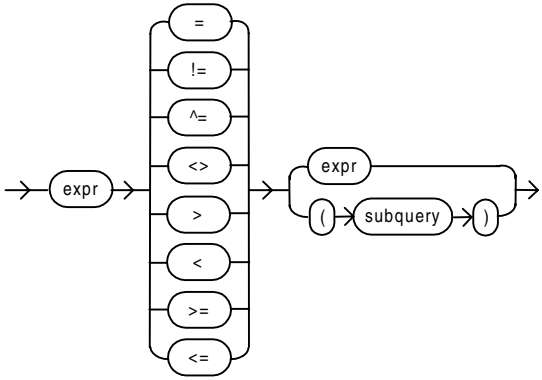
要素	説明
AM または PM	正午標識。
A.M. または P.M.	ピリオド付きの正午標識。
HH または HH12	時刻（1 ～ 12）。
HH24	時刻（0 ～ 23）。
MI	分（0 ～ 59）。
RR	年号の最後の 2 桁 ; 外国での年号。
SS	秒（0 ～ 59）。
SSSSS	午前 0 時以降の秒数（0 ～ 86399）。
- / . ; : 「テキスト」	句読点と引用符付きテキストが結果にコピーされます。

SQL 条件の指定

SQL 条件を指定するには、次の構文書式のいずれかを使用します。

単純比較条件

単純比較条件は、式または副問合せの結果との比較を指定します。



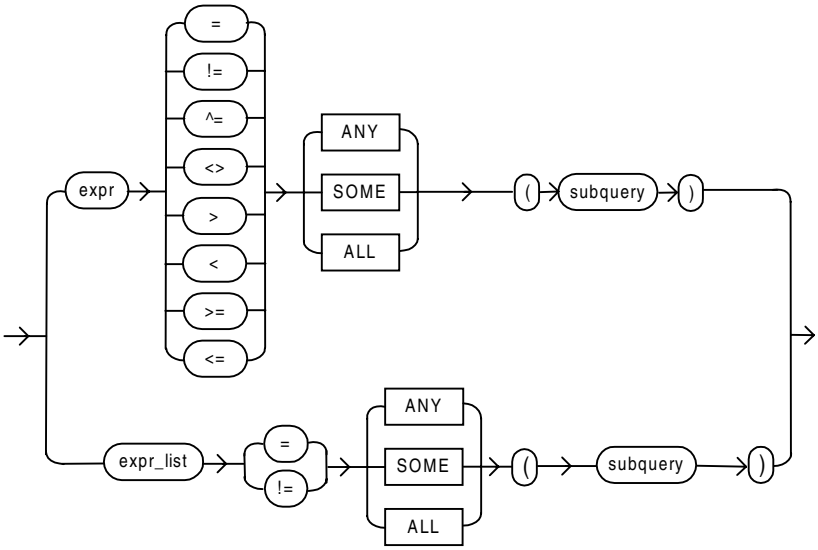
たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE SAL > 2000;
```

比較演算子の詳細は、「比較演算子」を参照してください。

グループ比較条件

グループ比較条件は、リストまたは副問合せ内の任意またはすべてのメンバーとの比較を指定します。



たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE ENAME = any ('SMITH', 'WARD', 'KING');
```

副問合せ比較内の Row_Value_Constructor

これにより、複数列の結果を返す副問合せを使用して、列または式の比較が可能です。この機能により、カンマで区切りカッコで囲んだ式のリストのような行値コンストラクタをユーザーが提供できるようになります。

列位置の副問合せ

算術式または列を指定できるところであれば、副問合せを挿入できます。副問合せは、カッコで囲み 1 つの列につき最大 1 行を返すものに限定されます。

たとえば、次のとおりです。

1. 選択リスト内の副問合せ。次の問合せがサポートされます（c1 と c2 は表 t1 内の列で、c1 は主キーとします）。

```
SELECT (select c1 from t1 b where a.c1 = b.c1),
    c2 from t1 a where <condition>
```

選択リスト内の副問合せの選択リストには、副問合せを含むことができます。ネストされる副問合せの数に制限はありません。

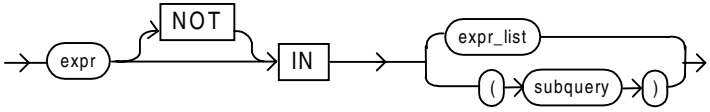
2. 式内の副問合せ。次の問合せがサポートされます（例 1 と同じ条件とします）。

```
SELECT *  from t1 a  where
   (select c1 from t1 where c1 = 10) =
   (select c1 from t1 b where a.c1 = b.c1) - 20;
```

3. 副問合せには GROUP BY、UNION、MINUS および INTERSECT を含めますが、ORDER BY 句を含むことはできません。

メンバーシップ条件

メンバーシップ条件は、リストまたは副問合せ内のメンバーシップをテストします。



たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE ENAME not in ('SMITH', 'WARD', 'KING');
```

範囲条件

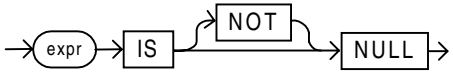
範囲条件は、範囲に含まれているかどうかをテストします。

たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE SAL between 2000 and 50000;
```

NULL 条件

NULL 条件は、NULL であるかどうかをテストします。

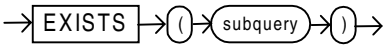


たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE MGR IS NOT NULL;
```

EXISTS 条件

EXISTS 条件は、副問合せ内の列の存在をテストします。

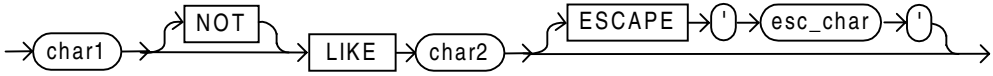


たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);
```

LIKE 条件

LIKE 条件は、パターン一致を含むテストを指定します。

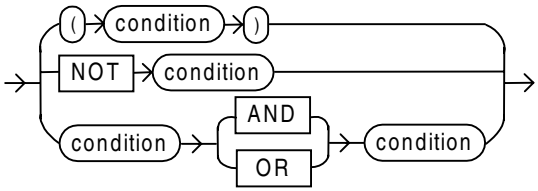


たとえば、次のとおりです。

```
SELECT * FROM EMP WHERE NAME like 'SM%"
```

複合条件

複合条件は、他の条件の組合せを指定します。



たとえば、次のとおりです。

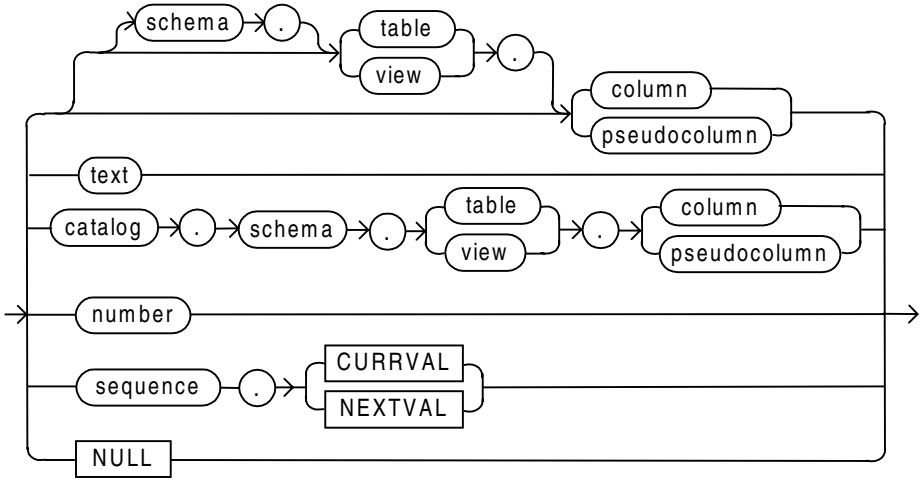
```
SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;
```

式の指定

SQL 式を指定するには、次の構文書式のいずれかを使用します。

書式 I、単純式

単純式は、列、疑似列、定数、順序番号または NULL を指定します。



ユーザーのスキーマの他に、"PUBLIC" も使用できます（二重引用符が必要です）。この場合、表、ビューまたはマテリアライズド・ビューのパブリック・シノニムを修飾する必要があります。"PUBLIC" によるパブリック・シノニムの修飾は、データ操作言語（DML）文でのみサポートされます。データ定義言語（DDL）ではサポートされません。

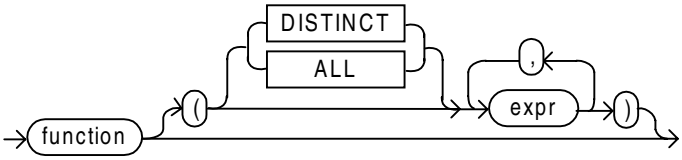
疑似列は、LEVEL、ROWID または ROWNUM のいずれかを使用できます。疑似列を使用できるのは表のみで、ビューまたはマテリアライズド・ビューでは使用できません。

例

```
emp-ename
'this is a text string'
10
```

書式 II、関数式

組込み関数式は、単一行関数へのコールを指定します。



有効な組込み関数式の例は、次のとおりです。

```
LENGTH('BLAKE')
ROUND(1234.567*43)
SYSDATE
```

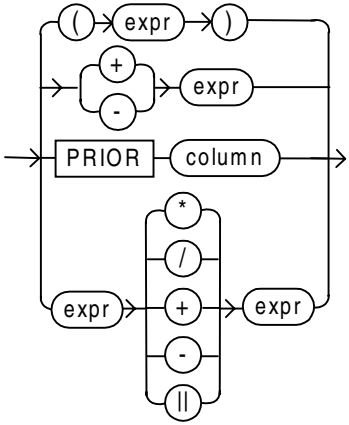
書式 III、Java 関数式

java_function_name (*expr* , *expr*...)
schema.table.java_function_name (*expr* , *expr*...)

Java 関数の使用方法の詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

書式 IV、複合式

複合式は、他の式の組合せを指定します。



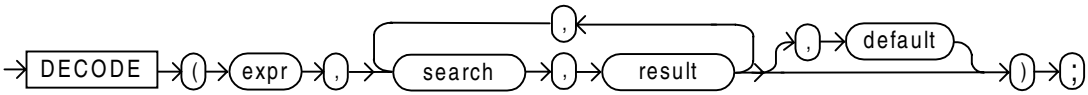
関数の組合せの中には、不適当で拒否されるものがあります。たとえば、LENGTH 関数は集計関数内では不適当です。

例

```
('CLARK' || 'SMITH')
LENGTH('MOOSE') * 57
SQRT(144) + 72
my_fun(TO_CHAR(sysdate,'DD-MM-YY'))
```

書式 V、DECODE 式

DECODE 式には、特別な DECODE 構文を使用します。



この式を評価するために、Oracle Lite は、*expr* を各 *search* と 1 つずつ比較します。*expr* と等しい *search* がある場合、Oracle Lite は、対応する *result* を返します。一致するものがないと、Oracle Lite は *default* を返すか、*default* が省略されている場合、NULL を返します。*expr* および *search* に文字データが含まれている場合、Oracle Lite は、非空白埋め比較を使用して、それらを比較します。

search、*result* および *default* 値は、式から導出できます。Oracle Lite は、すべての *search* 値を評価してから *expr* と比較するのではなく、*expr* と比較する直前にのみそれぞれの *search* 値を評価します。したがって、前の *search* と *expr* が等しい場合、Oracle Lite は *search* を評価しません。

Oracle Lite は、*expr* と各 *search* 値を最初の *search* 値のデータ型に自動的に変換してから比較します。Oracle Lite は、戻り値を最初の *result* 値と同じデータ型に自動的に変換します。最初の *result* のデータ型が CHAR、あるいは最初の *result* が NULL の場合、Oracle Lite は、戻り値をデータ型 VARCHAR2 に変換します。

DECODE 式の中では、Oracle Lite は、2 つの NULL 値を等価であるとみなします。*expr* が NULL の場合、Oracle Lite は、同様に NULL である最初の *search* の *result* を返します。*expr*、*search*、*result* および *default* を含めた、DECODE 式内のコンポーネントの最大数は、255 です。

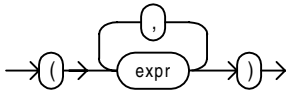
例

この式は、値 DEPTNO をデコードします。この例では、DEPTNO が 10 の場合、式の評価は 'ACCOUNTING' となります。DEPTNO が 10、20、30 または 40 でないと、式は「NONE」を返します。

```
DECODE (deptno,10, 'ACCOUNTING',
               20, 'RESEARCH',
               30, 'SALES',
               40, 'OPERATION',
                   'NONE')
```

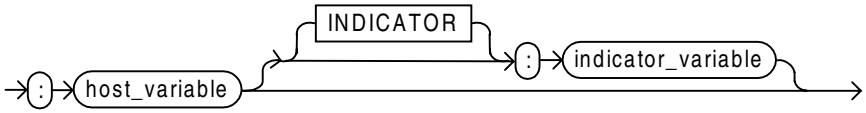
書式 VI、式リスト

式リストは、カンマで区切った一連の式です。全体をカッコで囲みます。



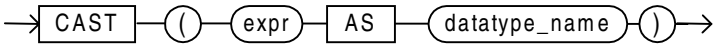
書式 VII、変数式

変数式は、オプションの標識変数を持ったホスト変数を指定します。この書式の式は、プログラム・インタフェースに使用されることがあります。



書式 VIII、CAST 式

CAST 式は、組込みデータ型またはコレクション型の値を、別の組込み型またはコレクション型の値に変換します。



オペランドの場合、*expr* は組込みデータ型です。次の表は、どの組込みデータ型が別データ型への CAST 変換を受け入れるかを示します。（CAST は LONG、LONG RAW または LOB データ型をサポートしません。）

変換元 / 変換先	CHAR、VARCHAR2	NUMERIC	DATE	TIME	TIMESTAMP	RAW
CHAR VARCHAR2	X	X	X	X	X	X
NUMERIC	X	X				
DATE	X		X		X	
TIME	X			X	X	
TIMESTAMP	X		X	X	X	
RAW	X					X

DATE データ型は、**POLITE.INI** ファイルに定義した `SQLCompatibility` の設定の影響を受けます。

- 次のように設定されていると、DATE および TIMESTAMP は等価です。
  `SQLCompatibility=Oracle`
- 次のように設定されていると、DATE および TIMESTAMP は等価でありません。
  `SQLCompatibility=SQL92`

POLITE.INI ファイルの詳細は、『Oracle Lite ユーザーズ・ガイド』を参照してください。

NUMERIC には、データ型 BIGINT、BINARY、BIT、DECIMAL、DOUBLE PRECISION、FLOAT、INTEGER、NUMBER、NUMERIC、REAL、SMALLINT および TINYINT が含まれます。

組込みデータ型の例

```
SELECT CAST ('1997-10-22' AS DATE) FROM DUAL;
SELECT * FROM t1 WHERE CAST (ROWID AS CHAR(5)) = '01234';
```

Oracle Lite SQL のデータ型およびリテラル

Oracle Lite SQL データ型のリストは、付録 B「Oracle Lite データ型」を参照してください。リテラルの詳細は、付録 C「Oracle Lite リテラル」を参照してください。

文字列比較の規則

Oracle Lite は、次の比較規則の 1 つを使用して文字列値を比較します。

- 空白埋め比較方法
- 非空白埋め比較方法

次の項では、これら 2 つの比較方法を説明します。異なる比較方法を使用すると、2 つの文字値の比較結果が変わります。次の表に、各比較方法を使用した文字値の比較結果を 5 つ示します。一般に、空白埋めおよび非空白埋めの比較の結果は同じになります。表の最後の比較は、空白埋めと非空白埋めの比較方法の相違点を示しています。

空白埋め	非空白埋め
`'ab' > 'aa'`	`'ab' > 'aa'`
`'ab' > 'a '`	`'ab' > 'a  '`
`'ab' > 'a'`	`'ab' > 'a'`
`'ab' = 'ab'`	`'ab' = 'ab'`
`'a  ' = 'a'`	`'a  ' > 'a'`

空白埋め比較方法

2 つの値の長さが異なる場合、Oracle Lite は最初に短い文字列の最後に空白を追加して長さが等しくなるようにします。Oracle Lite は、その後、1 文字ずつ相違が見つかるまで比較します。最初に相違が見つかった位置の文字が「より大きい」（>）の値の方が、大きいと解釈されます。2 つの値に異なる文字がない場合は、文字列が等しいと解釈されます。この規則は、後続の空白のみが異なる場合、2 つの値は等しいことを意味します。比較する両方の値がデータ型 CHAR、テキスト・リテラル、または USER および DATABASE 関数により返された値のいずれかである場合にのみ、Oracle Lite は空白埋め比較方法を使用します。

非空白埋め比較方法

Oracle Lite は、2 つの値を最初の異なる文字まで比較します。異なる位置の文字が「より大きい」（>）の値の方が、大きいと解釈されます。長さの異なる 2 つの値が短い文字列の終わりまで等しい場合、長い方の値が大きいと解釈されます。同じ長さの 2 つの値に異なる文字がない場合は、同じであると解釈されます。比較の一方または両方の値がデータ型 VARCHAR2 である場合に、Oracle Lite は非空白埋め比較を使用します。結果として、

CHAR 値と VARCHAR2 値を比較するとき、Oracle Lite は文字値 'a　' と 'a' を等しくないとみなします。

SQL 文内のコメント

SQL 文およびスキーマ・オブジェクトにコメントを関連付けることができます。SQL 文内のコメントは、文の実行に影響を与えませんが、アプリケーションを読みやすくしてメンテナンスを容易にできます。

コメントは、文の中のキーワード、パラメータまたは句読記号の間に入れることができます。次のいずれかの方法で文の中にコメントを含めることができます。

- コメントをスラッシュとアスタリスク（/*）で開始します。コメントのテキストを続けます。このテキストは、複数の行に渡ることができます。コメントをアスタリスクとスラッシュ（*/）で終了します。開始と終了の文字列とテキストを、スペースや改行で分ける必要はありません。
- コメントを --（2 つのハイフン）で開始します。コメントのテキストを続けます。これは新たな行に拡張できません。コメントを改行で終了します。

SQL 文は、両方のスタイルで複数のコメントを含むことができます。コメントのテキストは、データベースのキャラクタ・セット内の印字可能文字を含むことができます。

注意： このようなスタイルのコメントは、SQL スクリプト内の SQL 文の間では使用できません。このためには、SQL*Plus REMARK コマンドを使用します。これらの文の詳細は、Oracle SQL*Plus のユーザー・マニュアルを参照してください。

例 1

```
SELECT * FROM EMP WHERE EMP.DEPTNO = /* The subquery matches values in EMP.DEPTNO
with values in DEPT.DEPTNO */ (SELECT DEPTNO FROM DEPT WHERE LOC='DALLAS');
```

この文は次のものを返します。

```
    EMPNO ENAME      JOB              MGR HIREDATE        SAL      COMM    DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
     7566 JONES      MANAGER        7839 1981-04-0      2975                  20
     7902 FORD       ANALYST        7566 1981-12-0      3000                  20
     7369 SMITH      CLERK          7902 1980-12-1       800                  20
     7788 SCOTT      ANALYST        7566 1982-12-0      3000                  20
     7876 ADAMS      CLERK          7788 1983-01-1      1100                  20
```

例 2

```
SELECT ENAME, -- select the employee name
 SAL          -- and the salary
 FROM EMP     -- from the EMP table
 WHERE SAL    -- where the salary
 >=           -- is greater than or equal to
 3000         -- 3000
 ;
```

この文は次のものを返します。

```
ENAME            SAL
---------- ---------
KING            5000
FORD            3000
SCOTT           3000
```

SQL*Plus

Oracle Lite は、Oracle Lite データベースに対して SQL を実行するために使用できる SQL*Plus と呼ばれるツールを提供します。詳細は、付録 D「SQL*Plus」を参照してください。

2

SQL 演算子

この章では、Oracle Lite の SQL 演算子を説明します。説明する内容は次のとおりです。

- SQL 演算子の概要
- 算術演算子
- 文字演算子
- 比較演算子
- 論理演算子
- 集合演算子
- その他の演算子

SQL 演算子の概要

演算子は、個々のデータ項目を操作して、結果を返します。データ項目は、「オペランド」または「引数」と呼ばれます。演算子は、特殊文字またはキーワードで表されます。たとえば、乗算演算子は、アスタリスク（*）で表され、NULL をテストする演算子は、キーワード IS NULL で表されます。演算子には、単項演算子とバイナリ演算子の 2 つの一般クラスがあります。Oracle Lite SQL は、集合演算子もサポートします。

単項演算子

単項演算子は、オペランドを 1 つのみ使用します。単項演算子は通常、オペランドとともに次の書式で使用されます。

```
operator operand
```

バイナリ演算子

バイナリ演算子は、2 つのオペランドを使用します。バイナリ演算子は、オペランドとともに次の書式で使用されます。

```
operand1 operator operand2
```

集合演算子

集合演算子は、個々のデータ項目ではなく、問合せによって返される列を組み合せます。集合演算子の優先順位はすべて同等です。Oracle Lite は、次の集合演算子をサポートします。

- UNION
- UNION ALL
- INTERSECT
- AND MINUS

次に、Oracle Lite の SQL 演算子を優先順位の高い方から順にリストします。同じ行にリストされた演算子は優先順位が同じです。

優先順位のレベル	SQL 演算子
1	単項 + - 算術演算子、PRIOR 演算子
2	* / 算術演算子
3	バイナリ + - 算術演算子、|| 文字演算子
4	すべての比較演算子
5	NOT 論理演算子

優先順位のレベル	SQL 演算子
6	AND 論理演算子
7	OR 論理演算子

その他の演算子

特別の書式を持つその他の演算子は、3 つ以上のオペランドを受け入れます。演算子が NULL 演算子を受け取ると、結果は常に NULL です。このルールに従わない演算子は、CONCAT のみです。

算術演算子

算術演算子は数値オペランドを操作します。- 演算子は、日付計算でも使用されます。

演算子	説明	例
+（単項演算子）	オペランドを正にします。	`SELECT +3 FROM DUAL;`
-（単項演算子）	オペランドを負にします。	`SELECT -4 FROM DUAL;`
/	除算（数値および日付）	`SELECT SAL / 10 FROM EMP;`
*	乗算	`SELECT SAL * 5 FROM EMP;`
+	加算（数値および日付）	`SELECT SAL + 200 FROM EMP;`
-	減算（数値および日付）	`SELECT SAL -100 FROM EMP;`

文字演算子

文字演算子は式内で文字列を操作するために使用します。

演算子	説明	例
`||`	文字列を連結します	`SELECT 'The Name of the employee is: ' || ENAME FROM EMP;`

文字列の連結

Oracle Lite で、文字列を連結した結果は次のようになります。

- 2 つの文字列を連結した結果は、別の文字列となります。
- Oracle Lite では、連結における文字列内の後続する空白は、文字のデータ型に関わらず保持されます。
- Oracle Lite は、CONCAT 文字関数を垂直バー演算子の代替として提供します。たとえば、次のとおりです。

  ```
  SELECT CONCAT (CONCAT (ENAME, ' is a '),job) FROM EMP WHERE SAL > 2000;
  ```

  これは、次の結果を返します。

  ```
  CONCAT(CONCAT(ENAME
  -------------------------
  KING       is a PRESIDENT
  BLAKE      is a MANAGER
  CLARK      is a MANAGER
  JONES      is a MANAGER
  FORD       is a ANALYST
  SCOTT      is a ANALYST

  6 rows selected.
  ```

- Oracle Lite は、長さゼロの文字列は NULL として扱います。長さゼロの文字列を別のオペランドと連結すると、結果は常に、その別のオペランドになります。結果が NULL 値となるのは、2 つの NULL 文字列を連結した場合のみです。

比較演算子

比較演算子は 1 つの式を別の式と比較する条件内で使用します。比較の結果は、TRUE、FALSE または UNKNOWN になります。

演算子	説明	例
=	等しいかどうかを検査します。	`SELECT ENAME "Employee" FROM EMP WHERE SAL = 1500;`
!=, ^=, <>	等しくないかどうかを検査します。	`SELECT ENAME FROM EMP WHERE SAL ^= 5000;`
>	より大きいかどうかを検査します。	`SELECT ENAME "Employee", JOB "Title" FROM EMP WHERE SAL > 3000;`

演算子	説明	例
<	より小さいかどうかを検査します。	`SELECT * FROM PRICE WHERE MINPRICE < 30;`
>=	より大きいかまたは等しいかを検査します。	`SELECT * FROM PRICE WHERE MINPRICE >= 20;`
<=	より小さいかまたは等しいかを検査します。	`SELECT ENAME FROM EMP WHERE SAL <= 1500;`
IN	「要素のいずれかに等しい」かを検査します。「= ANY」に相当します。	`SELECT * FROM EMP WHERE ENAME IN ('SMITH', 'WARD');`
ANY/SOME	リスト内の各値または問合せで返された値と指定された値を比較します。=、!=、>、<、<= または >= の後に続けてください。問合せにより行が返されなかった場合には、FALSE に評価します。	`SELECT * FROM DEPT WHERE LOC = SOME ('NEW YORK','DALLAS');`
NOT IN	「!= ANY」に相当します。集合の要素のいずれかが NULL の場合、FALSE に評価されます。	`SELECT * FROM DEPT WHERE LOC NOT IN ('NEW YORK', 'DALLAS');`
ALL	リスト内のすべての値または問合せで返された値と指定された値を比較します。=、!=、>、<、<= または >= の後に続けてください。問合せにより行が返されなかった場合には、TRUE に評価します。	`SELECT * FROM emp  WHERE sal >= ALL (1400, 3000);`
[NOT] BETWEEN *x* and *y*	*x* 以上 *y* 以下の範囲にあります（ありません）。	`SELECT ENAME, JOB FROM EMP WHERE SAL BETWEEN 3000 AND 5000;`
EXISTS	副問合せが少なくとも 1 行を返す場合は TRUE です。	`SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);`

演算子	説明	例
x [NOT] LIKE *y* [ESCAPE *z*]	*x* が *y* のパターンに一致する（しない）場合、TRUE です。*y* の中の「%」文字は、NULL を除く 0 文字以上の任意の文字列に一致します。文字「_」は、どの単一文字にも一致します。ESCAPE に続く文字はいずれも文字どおり解釈されます。*y* にパーセント (%) および下線 (_) を含むときに便利です。	`SELECT * FROM EMP WHERE ENAME LIKE '%E%';`
IS [NOT] NULL	NULL かどうかを検査します。NULL の検査に使用できる唯一の演算子です。	`SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;`

論理演算子

論理演算子は、条件の結果を操作します。

演算子	説明	例
NOT	後続する条件が FALSE の場合、TRUE を返します。TRUE の場合、FALSE を返します。UNKNOWN の場合、UNKNOWN のままです。	`SELECT * FROM EMP WHERE NOT (job IS NULL)` `SELECT * FROM EMP WHERE NOT (sal BETWEEN 1000 AND 2000)`
AND	両方のコンポーネント条件が TRUE の場合、TRUE を返します。どちらかが FALSE の場合、FALSE を返します。それ以外は、UNKNOWN を返します。	`SELECT * FROM EMP WHERE job='CLERK' AND deptno=10`
OR	どちらかのコンポーネント条件が TRUE の場合、TRUE を返します。両方が FALSE の場合、FALSE を返します。それ以外は、UNKNOWN を返します。	`SELECT * FROM emp WHERE job='CLERK' OR deptno=10`

集合演算子

集合演算子は、2 つの問合せの結果を 1 つの結果にまとめます。

演算子	説明	例
UNION	どちらかの問合せで選択された重複していない行をすべて返します。	`SELECT * FROM (SELECT ENAME FROM EMP WHERE JOB = 'CLERK' UNION SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');`
UNION ALL	重複行も含めて、どちらかの問合せで選択された行をすべて返します。	`SELECT * FROM (SELECT SAL FROM EMP WHERE JOB = 'CLERK' UNION SELECT SAL FROM EMP WHERE JOB = 'ANALYST');`
INTERSECT および INTERSECT ALL	両方の問合せで選択された重複していない行をすべて返します。	`SELECT * FROM orders_list1 INTERSECT SELECT * FROM orders_list2`
MINUS	最初の問合せで選択された、他と重複しない行をすべて返します。2 番目の問合せで選択された行は返されません。	`SELECT * FROM (SELECT SAL FROM EMP WHERE JOB = 'PRESIDENT' MINUS SELECT SAL FROM EMP WHERE JOB = 'MANAGER');`

注意： INTERSECT ALL の構文はサポートされていますが、INTERSECT と同じ結果が返されます。

その他の演算子

次にその他の演算子をリストします。

演算子	説明	例
(+)	先行する列が結合内の外部結合列であることを指定します。	`SELECT ENAME, DNAME FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO (+);`
PRIOR	以下の階層型、またはツリー構造の問合せ内のカレント行の親行の式を評価します。このような問合せでは、この演算子を CONNECT BY 句内で使用して親と子行の関連を定義する必要があります。	`SELECT EMPNO, ENAME, MGR FROM EMP CONNECT BY PRIOR EMPNO = MGR;`

3

SQL 関数

この章では、Oracle Lite の SQL 関数を説明します。説明する内容は次のとおりです。

- SQL 関数のタイプ
- SQL 関数の概要
- SQL 関数のアルファベット順のリスト

SQL 関数のタイプ

この項では、いろいろなタイプの SQL 関数をリストします。「SQL 関数の概要」に、各関数の説明があります。

数値関数

CEIL

FLOOR

ROUND - 数値関数

TRUNC

文字関数

CHR	LTRIM	TRANSLATE
CONCAT	REPLACE	TRIM
INITCAP	RPAD	UCASE（UPPER を参照）
LCASE（LOWER を参照）	ROUND - 日付関数	UPPER
LOWER	SUBSTR	USER
LPAD	SUBSTRB	

数値を返す文字関数

ASCII	INSTR	LENGTHB
BIT_LENGTH（LENGTH を参照）	INSTRB	OCTET_LENGTH（LENGTH を参照）
CHAR_LENGTH（LENGTH を参照）	LENGTH	POSITION

日付関数

ADD_MONTHS	HOUR	ROUND - 日付関数
CURDATE	LAST_DAY	SECOND
CURRENT_DATE	MINUTE	SYSDATE
CURRENT_TIME	MOD	TIMESTAMPADD
CURRENT_TIMESTAMP	MONTH	TIMESTAMPDIFF
CURTIME	MONTHS_BETWEEN	TRUNC
DAYOFMONTH	NEXT_DAY	WEEK
DAYOFWEEK	NOW	YEAR

DAYOFYEAR	QUARTER	
変換関数		
CAST	TO_DATE	
CONVERT	TO_NUMBER	
TO_CHAR		
その他の関数		
CASE	GREATEST	LOCATE
DATABASE	FNULL（CASE および NVL を参照）	NVL
DECODE	INTERVAL	SUBSTR
EXTRACT	LEAST	USER
グループ関数		
AVG	STDDEV	
COUNT	SUM	
MAX	VARIANCE	
MIN		

SQL 関数の概要

どちらもデータ項目を操作して結果を返すという点で、SQL 関数と SQL 演算子はよく似ています。SQL 関数と SQL 演算子の相違点は、SQL 関数は引数を取るということです。SQL 関数の書式では、関数は 0 個以上の引数を操作できます。

```
function(argument1, argument2, ...) alias
```

予期していたデータ型とは違うデータ型の引数が渡されると、ほとんどの関数が実行前に引数のデータ型を暗黙的に変換します。NULL 値が渡されると、ほとんどの関数が NULL 値を返します。

SQL 関数は SQL 文内で、SQL コマンドとともにのみ使用されます。SQL 文には、単一行（またはスカラー）関数と集計関数の 2 つの一般型があります。この 2 つの相違点は、対象となるデータベース行の数です。単一行関数は問合せ内の単一行に基づく値を返し、集計関数は問合せ内のすべての行に基づく値を返します。

単一行の SQL 関数は、選択リスト（GROUP BY 句を含む SELECT 文以外）および WHERE 句で使用できます。

集計関数は、AVG、MIN、MAX、SUM および COUNT の集合関数です。これらの関数には、GROUP BY 関数が使用できる別名を提供する必要があります。

ほとんどの関数には SQL 書式と ODBC 書式の両方がありますが、この 2 つは機能的に若干異なる場合があります。埋込み SQL では両方の書式を使用できます。

数値関数

数値関数は、数値入力を受け入れて、数値を返します。

文字関数

単一行文字関数は、文字入力を受け入れて、文字値および数値の両方を返すことができます。

数値を返す文字関数

文字関数によっては数値のみを返すものもあります。

日付関数

日付関数は、DATE データ型の値に対して演算を行います。日付関数は、数を返す `MONTHS_BETWEEN` 関数以外、すべて DATE データ型の値を返します。

変換関数

変換関数は、値をあるデータ型から別のデータ型へ変換します。一般に、関数名の形式は、*datatype* TO *datattype* になっています。最初の datatype が入力のデータ型で、最後の datatype が出力のデータ型です。

SQL 関数のアルファベット順のリスト

この項では、Oracle Lite SQL 関数をアルファベット順にリストし、それぞれの関数を定義しています。説明には次の項目が含まれます。

- 構文
- 用途
- 引数と説明
- 例
- 使用上の注意
- ODBC 関数（該当する場合）

ADD_MONTHS

構文

ADD_MONTHS(*d*, *n*)

d は、日付データ型の値。

n は、月の数値を表す整数。

用途

指定された日付 *d* を指定された月数 *n* に加えて、結果の日付を返します。*d* が月の最終日の場合、または、結果の月の最終日が *d* の日付要素よりも小さい場合、ADD_MONTHS は結果の月の最終日を返します。それ以外の場合、ADD_MONTHS は *d* と同じ日付要素を持つ値を返します。

例

```
SELECT TO_CHAR(ADD_MONTHS(hiredate,1)),'DD-MM-YYYY' "Next month"FROM emp WHERE ename
= 'SMITH'
```

次の結果を返します。

```
TO_CHAR(ADD_MONTHS(HIREDATE              Next month
---------------------------------------- ----------
1981-01-17                               DD-MM-YYYY
```

ASCII

構文

ASCII(*char*)

用途

char の最初のバイトについて、データベース・キャラクタ・セットにおける 10 進表現を返します。データベース・キャラクタ・セットが 7 ビット ASCII の場合、この関数により ASCII 値が返されます。

例

```
SELECT ASCII('Q') FROM DUAL;
```

次の結果を返します。

```
ASCII('Q')
----------
        81
```

AVG

構文

```
AVG([DISTINCT | ALL] n)
```

用途

列 *n* の平均値を返します。

例 1

```
SELECT AVG(SAL) FROM EMP;
```

次の結果を返します。

```
 AVG(SAL)
---------
  2073.21
```

例 2

```
SELECT {FN AVG (SAL)} FROM EMP;
```

次の結果を返します。

```
{FNAVG(SAL)}
------------
     2073.21
```

例 3

```
SELECT AVG (DISTINCT DEPTNO) FROM EMP;
```

次の結果を返します。

```
AVG(DISTINCTDEPTNO)
-------------------
                 20
```

例 4

```
SELECT AVG (ALL DEPTNO) FROM EMP;
```

次の結果を返します。

```
AVG(ALLDEPTNO)
--------------
        22.142
```

ODBC 関数

```
{FN AVG ([DISTINCT | ALL] n)}
```

n は数値列の名前です。

CASE

構文

```
CASE
        WHEN condition 1 THEN result 1
        WHEN condition 2 THEN result 2
        ...
        WHEN condition n THEN result n
        ELSE result x
END,
```

用途

条件値を指定します。

引数	説明
WHEN	条件句を開始します。
condition	条件を指定します。
THEN	結果句を開始します。
result	関連付けられた条件の結果を指定します。
ELSE	条件句に記述されていない値の結果を指定するオプションの句です。
END	ケース文を終了します。

使用上の注意

CASE 関数は、選択文または更新文の条件と結果を指定します。CASE 関数は、特定の条件に基づいてデータを検索する場合や、条件に基づいて値を更新する場合に使用できます。

例

```
SELECT CASE JOB
            WHEN 'PRESIDENT' THEN 'The Honorable'
            WHEN 'MANAGER' THEN 'The Esteemed'
            ELSE 'The good'
        END,
ENAME
FROM EMP;
```

次の結果を返します。

```
CASEJOBWHEN'PRESI ENAME
----------------- ----------
The Honorable     KING
The Esteemed      BLAKE
The Esteemed      CLARK
The Esteemed      JONES
The good          MARTIN
The good          ALLEN
The good          TURNER
The good          JAMES
The good          WARD
The good          FORD
The good          SMITH
The good          SCOTT
The good          ADAMS
The good          MILLER

14 rows selected.
```

CAST

構文

SELECT CAST (<*source_operand* > AS <*data_type* >) FROM DUAL;

用途

あるデータ型のデータを別のデータ型に変換します。

引数	説明
<*source_operand*>	値式または NULL。
<*data_type*>	対象データのデータ型。

使用上の注意

次の表は、ソースのオペランドからデータ型への変換の結果を表示します。

変換の結果

		＜*data_type*＞								
		EN	AN	VC	FC	D	T	TS	YM	DT
＜*source_operand*＞	EN	V	V	V	V	X	X	X	R	R
	AN	V	V	V	X	X	X	X	X	X
	C	V	R	R	V	V	V	V	V	X
	D	X	X	V	V	V	X	V	X	X
	T	X	X	V	V	X	V	V	X	X
	TS	X	X	V	V	V	V	V	X	X
	YM	R	X	V	V	X	X	X	V	X
	DT	R	X	V	V	X	X	X	X	V

次の表は、ソースのオペランドからデータ型への変換の結果を定義します。

結果の定義	
EN = 真数	D = 日付
C = 固定長または可変長の文字	TS = タイム・スタンプ
VC = 可変長文字	DT = 日付時刻
T = 時刻	V = 有効
YM = 年月間隔	R = 制限付きの有効
AN = 概数	X = 無効
FC = 固定長文字	

<source_operand> が真数で *<data_type>* が間隔である場合、間隔には単一の日付時刻フィールドを含みます。

<source_operand> が間隔で *<data_type>* が真数である場合、間隔には単一の日付時刻フィールドを含みます。

<source_operand> が文字列で *<data_type>* が文字列を指定する場合、それらの文字レパートリは同じです。

<data_type> が数値で、先頭の桁の表示が失われる場合、データ例外、数値が範囲外という例外状況が発生します。

例 1

```
SELECT CAST('0' AS INTEGER) FROM DUAL;
```

次の結果を返します。

```
CAST('0'ASINTEGER)
------------------
                 0
```

例 2

```
SELECT CAST(0 AS REAL) FROM DUAL;
```

次の結果を返します。

```
CAST(0ASREAL)
-------------
            0
```

例 3

```
SELECT CAST(1E0 AS NUMERIC(12, 2)) FROM DUAL;
```

次の結果を返します。

```
CAST(1E0ASNUMERIC(12
--------------------
                   1
```

例 4

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(30)) FROM DUAL;
```

次の結果を返します。

```
CAST(CURRENT_TIMESTAMPASVARCH
-----------------------------
1999-04-12 14:53:53
```

CEIL

構文

CEIL (*n*)

用途

n 以上で最小の整数が返されます。

例

```
SELECT CEIL(15.7) "Ceiling" FROM DUAL;
```

次の結果を返します。

```
  Ceiling
---------
       16
```

CHR

構文

CHR (*n*)

用途

データベース・キャラクタ・セット内の *n* のバイナリ等価を持つ文字を返します。

例

```
SELECT CHR(68)||CHR(79)||CHR(71) "Dog" FROM DUAL;
```

次の結果を返します。

```
Dog
---
CDT
```

CONCAT

構文

CONCAT(*char1*, *char2*)

または

CHAR1 || CHAR2

用途

char2 と連結された *char1* を返します。*char1* および *char2* は文字列引数です。この関数は連結演算子（||）と同じです。

例

この例ではネストを使用して 3 つの文字列を連結します。

```
SELECT CONCAT( CONCAT(ename, ' is a '), job) "Job"
FROM emp
WHERE empno = 7900;
```

次の結果を返します。

```
Job
-------------------------
JAMES      is a CLERK
```

ODBC 関数

{FN CONCAT (char1, char2)}

CONVERT

構文

{ fn CONVERT(value_exp, data_type) }

用途

文字列をあるキャラクタ・セットから別のキャラクタ・セットに変換します。

value_exp 引数は、変換される値です。

data_type 引数は、*char* が変換される先のキャラクタ・セットの名前です。

使用上の注意

次は、一般的なキャラクタ・セットをリストしたものです。

一般的なキャラクタ・セット	
US7ASCII	WE8ISO8859P1
WE8DEC	HP West European Laserjet 8 ビット・キャラクタ・セット
WE8HP	DEC French 7 ビット・キャラクタ・セット
F7DEC	IBM West European EBCDIC Code Page 500
WE8EBCDIC500	IBM PC Code Page 850 ISO 8859-1 West European 8 ビット・キャラクタ・セット
WE8PC850	ISO 8859-1 West European 8 ビット・キャラクタ・セット

例

```
SELECT {fn CONVERT('Groß', 'US7ASCII') }
"Conversion" FROM DUAL;
```

次の結果を返します。

```
conversi
--------
Groß
```

COUNT

構文

COUNT([* | [DISTINCT | ALL] *expr*])

用途

問合せ内の行数を返します。

例 1

```
SELECT COUNT(*) "Total" FROM emp;
```

次の結果を返します。

```
Total
----------
14
```

例 2

```
SELECT COUNT(job) "Count" FROM emp;
```

次の結果を返します。

```
Count
----------
14
```

例 3

```
SELECT COUNT(DISTINCT job) "Jobs" FROM emp;
```

次の結果を返します。

```
Jobs
----------
5
```

例 4

```
SELECT COUNT (ALL JOB) FROM EMP;
```

次の結果を返します。

```
COUNT(ALLJOB)
-------------
```

CURDATE

構文

{ fn CURDATE (<*value_expression* >) }

用途

現在の日付を返します。

使用上の注意

expr（式）を指定すると、この関数は *expr* が NULL でない行を返します。全行数または *expr* で指定した値の行数のみをカウントします。

アスタリスク（*）を指定した場合、この関数は重複行および NULL 行を含むすべての行数を返します。

例 1

```
SELECT {fn CURDATE()} FROM DUAL;
```

次の結果を返します。

```
{FNCURDATE
-----------
1999-04-12
```

例 2

```
SELECT {fn WEEK({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNWEEK({FNCURDATE()})}
-----------------------
                     16
```

CURRENT_DATE

構文

```
CURRENT_DATE
```

用途

現在の日付を返します。

例

```
SELECT CURRENT_DATE FROM DUAL;
```

次の結果を返します。

```
CURRENT_DATE
------------
1999-04-12
```

ODBC 関数

```
{fn CURDATE()}
```

CURRENT_TIME

構文

```
CURRENT_TIME
```

用途

現在の時刻を返します。

例

```
SELECT CURRENT_TIME FROM DUAL;
```

次の結果を返します。

```
CURRENT_T
---------
15:54:18
```

ODBC 関数

```
{fn CURTIME()}
```

CURRENT_TIMESTAMP

構文

```
CURRENT_TIMESTAMP
```

用途

現在のローカルの日付と時刻をタイム・スタンプ値として返しますが、デフォルトでは現在のローカルの日付のみを表示します。現在のローカルの時刻情報は、CURRENT_TIMESTAMP を TO_CHAR 関数の値として使用し、時間書式を含めることで表示できます。例 2 を参照してください。

例 1

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

次の結果を返します。

```
CURRENT_TI
---------
1999-04-12
```

例 2

```
SELECT TO_CHAR (CURRENT_TIMESTAMP, 'HH24:MM:SS, Day, Month, DD, YYYY')FROM DUAL;
```

次の結果を返します。

```
TO_CHAR(CURRENT_TIMESTAMP
----------------------------------------
18:04:05, Tuesday  , April    , 06, 1999
```

ODBC 関数

```
{fn CURTIME()}
```

CURTIME

構文

{ fn CURTIME (<*value_expression* >) }

用途

現在の時刻を返します。

例 1

```
SELECT {fn CURTIME()} FROM DUAL;
```

次の結果を返します。

```
{FNCURTIM
---------
11:09:59
```

例 2

```
SELECT {fn HOUR({fn CURTIME()})} FROM DUAL;
```

次の結果を返します。

```
{FNHOUR({FNCURTIME()})}
-----------------------
                     11
```

DATABASE

構文

```
{ fn DATABASE () }
```

用途

データベースの名前を指定します。ODBC 使用時は、DATABASE 関数は現行のデフォルト・データベース・ファイルを **.ODB** 拡張子なしで返します。

使用上の注意

データベース名関数は、オプション `SQL_CURRENT_QUALIFIER` を指定した `SQLGetConnectOption()` と同じ値を返します。

例

次の例では、デフォルト・データベースに接続されたユーザーの結果を返します。

```
SELECT {fn DATABASE () } FROM DUAL;
```

次の結果を返します。

```
{FNDATABASE()}
--------------
POLITE
```

DAYOFMONTH

構文

```
{ fn DAYOFMONTH ( <value_expression > ) }
```

用途

日にちを整数として返します。

引数	説明
<value_expression>	日にちを計算する日付。結果は 1 ～ 31 の間で、1 が 1 日を表します。

例 1

```
SELECT {fn DAYOFMONTH ({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNDAYOFMONTH({FNCURDATE()})
----------------------------
                          12
```

例 2

```
SELECT {fn DAYOFMONTH('1997-07-16')} "DayOfMonth" FROM DUAL;
```

次の結果を返します。

```
DayOfMonth
----------
        16
```

DAYOFWEEK

構文

{ fn DAYOFWEEK (<*value_expression* >) }

用途

曜日を整数として返します。

引数	説明
<*value_expression*>	曜日を計算する日付。結果は 1 ～ 7 で、1 が日曜を表します。

例 1

```
SELECT {fn DAYOFWEEK ({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNDAYOFWEEK({FNCURDATE()})}
----------------------------
                           2
```

例 2

```
SELECT {fn DAYOFWEEK('1997-07-16')} "DayOfWeek" FROM DUAL;
```

次の結果を返します。

```
DayOfWeek
------------------
4
```

DAYOFYEAR

構文

{ fn DAYOFYEAR (<*value_expression* >) }

用途

年間通算日を整数として返します。

引数	説明
<*value_expression*>	年間通算日を計算する日付。結果は 1 ～ 366 です。

例 1

```
SELECT {fn DAYOFYEAR ({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNDAYOFYEAR({FNCURDATE()})}
----------------------------
                         102
```

例 2

```
SELECT {fn DAYOFYEAR('1997-07-16')} "DAYOFYEAR" FROM DUAL;
```

次の結果を返します。

```
DayOfYear
---------
197
```

DECODE

構文

```
DECODE (expr, search, result [, search, result...] [,default])
```

用途

式の値を検索し、それを指定された結果（result）として評価します。

使用上の注意

式を評価するために、Oracle Lite は、式を各検索値（search）と 1 つずつ比較します。式と等しい検索値がある場合、Oracle Lite は、その検査値に対応する結果を返します。一致するものがないと、Oracle Lite はデフォルトを返すか、デフォルトが省略されている場合は、NULL を返します。式および検索値に文字データが含まれている場合、Oracle Lite は非空白埋め比較を使用して、それらを比較します。

検索（search）、結果（result）およびデフォルト（default）の各値を式から導出できます。Oracle Lite は、すべての検索値を評価してから式と比較するのではなく、式と比較する直前にのみそれぞれの検索値を評価します。したがって、Oracle は、前の検索が式と等しい場合、検索を評価しません。

Oracle Lite は、比較の前に、式と各検索値を最初の検索値のデータ型に自動的に変換します。Oracle Lite は、戻り値を最初の結果と同じデータ型に自動的に変換します。最初の結果のデータ型が CHAR、あるいは最初の結果が NULL の場合、Oracle Lite は、戻り値をデータ型 VARCHAR2 に変換します。

DECODE 式の中では、Oracle Lite は、2 つの NULL 値を等価であるとみなします。式が NULL の場合、Oracle Lite は、同様に NULL である最初の検索の結果を返します。

式、検索、結果、およびデフォルトも含めた、DECODE 式内のコンポーネントの最大数は、255 です。

例 1

次の式は、DEPT 表内の DEPTNO 列をデコードします。DEPTNO が 10 の場合、式は「ACCOUNTING」と評価され、DEPTNO が 20 の場合、「RESEARCH」に評価されていきます。DEPTNO が 10、20、30、または 40 でないと、式は「NONE」を返します。

```
DECODE (deptno, 10, 'ACCOUNTING',
20, 'RESEARCH',
30, 'SALES',
40, 'OPERATIONS',
'NONE')
```

例 2

次の例は、SELECT 文の中で DECODE 句を使用します。

```
SELECT DECODE (deptno, 10, 'ACCOUNTING',
20, 'RESEARCH',
30, 'SALES',
40, 'OPERATIONS',
'NONE')
FROM DEPT;
```

次の結果を返します。

```
DECODE(DEP
----------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

EXTRACT

構文

EXTRACT (*extract-field FROM extract source*)

用途

extract-source の *i* 部分から、情報を返します。*extract-source* 引数は、日付時刻または間隔の式を含みます。*extract-field* 引数は、YEAR、MONTH、DAY、HOUR、MINUTE または SECOND のキーワードの内の 1 つを含みます。

戻り値の精度は、実装内で定義されます。SECOND が指定されていないと、スケールは 0 です。SECOND が指定されていると、スケールは、少なくとも *extract-source* フィールドの秒の小数部の精度かそれ以上になります。

例 1

```
SELECT EXTRACT (DAY FROM '06-15-1966') FROM DUAL;
```

次の結果を返します。

```
EXTRACT(DAY
-----------
         15
```

例 2

```
SELECT  EXTRACT (YEAR FROM {FN CURDATE()}) FROM DUAL;
```

次の結果を返します。

```
EXTRACT(YEAR
------------
        1999
```

FLOOR

構文

FLOOR (*n*)

用途

n 以下で最大の整数が返されます。

例

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;
```

次の結果を返します。

```
    Floor
---------
       15
```

GREATEST

構文

GREATEST(*expr* [,*expr*] ...)

用途

exprs（式）のリストから、最大値を返します。2 番目以降の *exprs* はすべて、比較の前に、最初の *exprs* のデータ型に暗黙的に変換されます。Oracle Lite は、非空白埋め比較を使用して、*exprs* を比較します。文字比較は、データベース・キャラクタ・セットの文字の値に基づきます。文字の値が高い文字の方がもう 1 つの文字より大きいといえます。この関数によって返された値が文字データの場合、データ型は常に VARCHAR2 になります。

例

```
SELECT GREATEST('HARRY','HARRIOT','HAROLD') "GREATEST" FROM DUAL;
```

次の結果を返します。

```
GREATEST
--------
HARRY
```

HOUR

構文

HOUR (*time_exp*)

用途

時間を 0 ～ 23 までの整数値として返します。

例 1

```
SELECT {FN HOUR ('14:03:01')} FROM DUAL;
```

次の結果を返します。

```
{FNHOUR('14:03:01')}
--------------------
                  14
```

例 2

```
SELECT {fn HOUR({fn CURTIME()})} FROM DUAL;
```

次の結果を返します。

```
{FNHOUR({FNCURTIME()})}
-----------------------
                     11
```

INITCAP

構文

INITCAP(*char*)

用途

各単語の最初の文字を大文字、残りの文字を小文字にして、*char* を返します。単語は空白か、英数字でない文字で区切られます。

例

```
SELECT INITCAP('the soap') "Capitals" FROM DUAL;
```

次の結果を返します。

```
Capitals
--------
The Soap
```

INSTR

構文

INSTR(*char1*, *char2*, [, *n* [, *m*]])

用途

char1 の *n* 番目の文字から *char2* の探索を開始し、*m* 番目に出現した位置を返します。*m* および *n* は数値引数です。最初に出現した文字の *char1* の中の位置を返します。

使用上の注意

n が負の場合、INSTR は *char1* の終わりから逆にカウントおよび検索します。*m* の値は正になります。*n* と *m* 両方のデフォルト値は 1 で、この設定によって、*char2* の最初の出現に対して、*char1* の最初の文字で探索を開始します。戻り値は *n* の値に関わらず、*char1* の先頭に相対し、文字で表されます。検索が失敗すると（*char2* が *char1* の *n* 番目の文字の後に *m* 回現われない場合）、戻り値は *0* になります。詳細は、「POSITION」関数の構文を参照してください。

例

```
SELECT INSTR('CORPORATE FLOOR','OR',3,2) "Instring" FROM DUAL;
```

次の結果を返します。

```
 Instring
---------
       14
```

INSTRB

構文

INSTRB(*char1*, *char2*, [, *n* [, *m*]])

用途

char1 の *n* バイト目の文字から *char2* の探索を開始し、*m* 番目に出現した位置を返します。*m* および *n* は数値引数です。最初に出現した 1 バイト目の *char1* の中の位置を返します。*n* および関数の戻り値が文字ではなくバイトで表示されていることを除き、INSTR と同じです。シングルバイトのデータベース・キャラクタ・セットの場合、INSTRB は、INSTR と同じです。

例

```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2) "Instring in bytes" FROM DUAL;
```

次の結果を返します。

```
Instring in bytes
-----------------
               14
```

INTERVAL

構文

INTERVAL (*datetime values*)

用途

1 つの datetime から別の datetime を差し引いて、結果を生成します。1 つの間隔に別の間隔を追加したり差し引いたりした結果は常に間隔になります。間隔は、数値定数で乗除できます。

例 1

```
SELECT CURRENT_DATE - INTERVAL '8' MONTH FROM DUAL;
```

次の結果を返します。

```
CURRENT_DATE-INTERVAL
---------------------
1998-08-09
```

例 2

```
SELECT TO_CHAR (INTERVAL '6' DAY * 3) FROM DUAL;
```

次の結果を返します。

```
TO_CHAR(INTERVAL'6'DAY*3)
-------------------------
18
```

LAST_DAY

構文

LAST_DAY(*d*)

用途

日付 *d* が発生する月の最後の日付を返します。

使用上の注意

この関数は、現在の月の残りの日数を判断する場合に使用できます。

例 1

```
SELECT LAST_DAY (SYSDATE) FROM DUAL;
```

次の結果を返します。

```
LAST_DAY
----------
1999-04-30
```

例 2

```
SELECT SYSDATE,
LAST_DAY(SYSDATE) "Last",
LAST_DAY(SYSDATE) - SYSDATE "Days Left"
FROM DUAL;
```

次の結果を返します。

```
{FNNOW()}   Last        Days Left
----------  ----------  ---------
1999-04-12  1999-04-30         18
```

LEAST

構文

LEAST(*expr* [,*expr*] ...)

用途

exprs（式）のリストから、最小値を返します。2 番目以降の *exprs* はすべて、比較の前に、最初の *exprs* のデータ型に暗黙的に変換されます。Oracle Lite は、非空白埋め比較を使用して、*exprs* を比較します。文字比較は、データベース・キャラクタ・セットの文字の値に基づきま

す。文字の値が低い場合、その文字はもう 1 つの文字より小さいといえます。この関数によって返された値が文字データの場合、データ型は常に VARCHAR2 になります。

例

```
SELECT LEAST('HARRY','HARRIOT','HAROLD') "LEAST" FROM DUAL;
```

次の結果を返します。

```
LEAST
-------
HAROLD
```

LENGTH

構文

LENGTH (*char*)
{fn LENGTH(*char*)}
BIT_LENGTH (*char*)
CHAR_LENGTH (*char*)
OCTET_LENGTH (*char*)

LENGTH は、*char* の文字数を返します。BIT_LENGTH、CHAR_LENGTH および OCTET_LENGTH は、*char* の長さをそれぞれビット数、文字数およびオクテット数で返します。

用途

文字列引数 *char* の長さを返します。*char* のデータ型が CHAR の場合、長さには後続する空白がすべて含まれます。*char* が NULL の場合は、NULL が返されます。

使用上の注意

BIT_LENGTH、CHAR_LENGTH および OCTET_LENGTH は SQL-92 関数です。CHAR_LENGTH は LENGTH と同じで、OCTET_LENGTH は LENGTHB と同じです。

例

```
SELECT LENGTH('CANDIDE') "Length in characters" FROM DUAL;
```

次の結果を返します。

```
Length in characters
--------------------
                   7
```

LENGTHB

構文

LENGTHB(*char*)
{fn LENGTHB(*char*)}

用途

文字列引数 *char* のバイト数を返します。*char* が NULL の場合は、NULL が返されます。シングルバイトのデータベース・キャラクタ・セットの場合、LENGTHB は、「LENGTH」と同じです。

例

```
SELECT LENGTHB('CANDIDE') "Length in bytes" FROM DUAL;
```

次の結果を返します。

```
Length in bytes
---------------
              7
```

LOCATE

構文

LOCATE (*string_exp1, string_exp2[,start]*)

用途

string_exp2 の最初の文字位置以降で、*string_exp1* が最初に出現する開始位置を返します。検索の開始位置を *string_exp2* の最初の文字位置ではなく、*start* 値で指定できます。

例 1

次の例は、EMP 表のすべての行に対して、文字列式「TURNER」内の文字「R」の開始位置を選択します。

```
SELECT {FN LOCATE ('R', 'TURNER')} FROM EMP ENAME;
```

次の結果を返します。

```
{FNLOCATE('R'
-------------
            3
            3
            3
            3
```

```
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3

14 rows selected.
```

例 2

次の例は、文字列式「TURNER」内の文字「R」の開始位置を選択し、検索を「TURNER」内の 4 番目の文字から開始します。例は、EMP 表のすべての列で検出されたすべての「TURNER」に対する結果を表示しています。

```
SELECT {FN LOCATE ('R', 'TURNER',4)} FROM EMP ENAME;
```

次の結果を返します。

```
{FNLOCATE('R'
-------------
            6
            6
            6
            6
            6
            6
            6
            6
            6
            6
            6
            6
            6
            6

14 rows selected.
```

LOWER

構文

LOWER(*char*)

用途

文字列引数 *char* の文字をすべて小文字にして返します。戻り値のデータ型は *char* のデータ型と同じで、CHAR または VARCHAR2 のいずれかです。

例

```
SELECT LOWER('LOWER') FROM DUAL;
```

次の結果を返します。

```
LOWER
-----
lower
```

ODBC 関数

{fn LCASE (char)}

LPAD

構文

LPAD(*char1*,*n* [,*char2*])

用途

char1 の左に *char2* で指定した文字を連続的に埋め込んで *n* 桁にして返します。*char2* は単一の空白にデフォルト設定されています。*char1* が *n* よりも長い場合、関数は *char1* の *n* に収まる部分を返します。

引数 *n* は、端末画面に表示される際の戻り値の合計長です。ほとんどのキャラクタ・セットで、これは戻り値の文字の数でもあります。ただし、マルチバイト・キャラクタ・セットの中には、文字列の表示長が文字列内の文字数によって異なるものがあります。

例

```
SELECT LPAD('Page1',15,'*.') "LPAD example" FROM DUAL;
```

次の結果を返します。

```
LPAD example
-----------------
*.*.*.*.*.Page1
```

LTRIM

構文

LTRIM(*char* [, *set*])

用途

set に指定されたすべての文字を *char* の左側から順に取り除きます。*set* のデフォルト値は単一の空白です。

例

```
SELECT LTRIM ('xyxXxyLAST WORD','xy') "LTRIM example" FROM DUAL;
```

次の結果を返します。

```
LTRIM example
---------------
XxyLAST WORD
```

ODBC 関数

```
{fn LTRIM (char) }      (trims leading blanks)
```

MAX

構文

MAX([DISTINCT | ALL] *expr*)

用途

引数 *expr* で指定されている式の最大値を返します。

例

```
SELECT MAX(SAL) FROM EMP;
```

次の結果を返します。

```
MAX(SAL)
---------
5000
```

MIN

構文

MIN([DISTINCT | ALL] *expr*)

用途

引数 *expr* で指定されている式の最小値を返します。

例

```
SELECT MIN(SAL), MAX(SAL) FROM EMP;
```

次の結果を返します。

```
 MIN(SAL)
---------
      800
```

MINUTE

構文

MINUTE (*time_exp*)

用途

分を 0 ～ 59 までの整数値として返します。

例 1

```
SELECT {FN MINUTE ('14:03:01')} FROM DUAL;
```

次の結果を返します。

```
{FNMINUTE('14:03:01')}
----------------------
                     3
```

例 2

```
SELECT {fn MINUTE({fn CURTIME()})} FROM DUAL;
```

次の結果を返します。

```
{FNMINUTE({FNCURTIME()})}
-------------------------
                       23
```

MOD

構文

```
MOD (m,n)
```

用途

m を *n* で除算した余りを返します。*n* が 0 の場合は、*m* を返します。

例

```
SELECT MOD (26,11) "ABLOMOV" FROM DUAL;
```

次の結果を返します。

```
  ABLOMOV
---------
        4
```

MONTH

構文

```
MONTH (date_exp)
```

用途

月を 1 ～ 12 までの整数値として返します。

例 1

```
SELECT {FN MONTH ('06-15-1966')} FROM DUAL;
```

次の結果を返します。

```
{FNMONTH('06-15-1966')}
-----------------------
                      6
```

例 2

```
SELECT {fn MONTH({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNMONTH({FNCURDATE()})}
------------------------
                       4
```

MONTHS_BETWEEN

構文

MONTHS_BETWEEN(*d1*, *d2*)

用途

日付 *d1* と *d2* との間の月数を返します。*d1* が *d2* より遅い場合結果は正になり、早い場合は負になります。*d1* と *d2* が、同じ日あるいは月の最後の日の場合、結果は常に整数です。そうでない場合、Oracle Lite は、1 か月が 31 日として結果の端数部分を計算し、*d1* と *d2* の時間コンポーネント内の相違とみなします。

例

```
SELECT MONTHS_BETWEEN(
TO_DATE('02-02-1995','MM-DD-YYYY'),
TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
FROM DUAL;
```

次の結果を返します。

```
   Months
---------
1.0322581
```

NEXT_DAY

構文

NEXT_DAY(*d*, *char*)

用途

日付 *d* よりも後で、*char* で指定された曜日の最初の日付を返します。引数 *char* は、ユーザーのセッションの日付言語で記述された曜日にする必要があります。戻り値は引数 *d* と同じ時間、分、秒のコンポーネントを持っています。

例

```
SELECT NEXT_DAY('15-MAR-92','TUESDAY') "NEXT DAY" FROM DUAL;
```

次の結果を返します。

```
NEXT DAY
----------
1992-03-17
```

NOW

構文

NOW

用途

現在のローカルの日付と時刻をタイム・スタンプ値として返しますが、デフォルトでは現在のローカルの日付のみを表示します。現在のローカルの時刻情報は、NOW を TO_CHAR 関数の値として使用し、時間書式を含めることで表示できます。例 2 を参照してください。

例 1

```
SELECT {FN NOW()} FROM DUAL;
```

次の結果を返します。

```
{FNNOW()}
----------
1999-04-07
```

例 2

```
SELECT TO_CHAR ({fn NOW ('YYYY, Month, DD, HH24:MM:SS')}) FROM DUAL;
```

次の結果を返します。

```
TO_CHAR({FNNOW('YYYY
----------------------------------------
1999-04-07 12:55:31
```

NVL

構文

NVL(*expr1*, *expr2*)

用途

expr1 が NULL の場合、*expr2* が返されます。*expr1* が NULL でない場合、*expr1* が返されます。引数 *expr1* および *expr2* は同じデータ型にしてください。

例 1

```
SELECT ename, NVL(TO_CHAR(COMM),'NOT APPLICABLE') "COMMISSION"
FROM emp
WHERE deptno = 30;
```

次の結果を返します。

```
ENAME      COMMISSION
---------- ----------------
BLAKE      NOT APPLICABLE
MARTIN     1400.00
ALLEN      300.00
TURNER     .00
JAMES      NOT APPLICABLE
WARD       500.00

6 rows selected.
```

例 2

```
SELECT {fn IFNULL(Emp.Ename, 'Unknown')},
NVL (Emp.comm, 0) FROM EMP;
```

次の結果を返します。

```
{FNIFNULL( 'UNKNOWN')}
---------- -----------
KING                0
BLAKE               0
CLARK               0
JONES               0
MARTIN           1400
ALLEN             300
TURNER              0
JAMES               0
WARD              500
FORD                0
```

```
SMITH                0
SCOTT                0
ADAMS                0
MILLER               0

14 rows selected.
```

例 3

```
SELECT sal+NVL(comm, 0) FROM EMP;
```

次の結果を返します。

```
SAL+NVL(COMM
------------
        5000
        2850
        2450
        2975
        2650
        1900
        1500
         950
        1750
        3000
         800
        3000
        1100
        1300

14 rows selected.
```

ODBC 関数

{fn IFNULL (*expr1*, *expr2*)}

POSITION

構文

```
POSITION ( <substring_value_expression>
                IN <value_expression> )
```

引数	説明
<value_expression>	検索対象のソース文字列。
<substring_value_expression>	検索される副文字列。
<start_len_cnt>	検索の開始位置。

用途

文字列内で副文字列の最初に出現する開始位置を返します。

使用上の注意

<substring_value_expression> の長さが 0 の場合、結果は NULL となります。
<substring_value_expression> が *<value_expression>* 内で発生した場合、結果は *<substring_value_expression>* の最初の文字位置です。そうでない場合は、結果は 0 です。
<start_len_cnt> が省略された場合、関数は検索を 1 の位置から開始します。詳細は、INSTR および INSTRB 関数を参照してください。

例

```
SELECT POSITION ('CAT' IN 'CATCH') FROM DUAL;
```

次の結果を返します。

```
POSITION('CAT'IN'CATCH')
------------------------
                       1
```

ODBC 関数

```
{fn LOCATE ( <substring_value_expression> ,
  <value_expression>[, <start_len_cnt> ] ) }
```

QUARTER

構文

```
{ fn QUARTER ( <value_expression> ) }
```

引数	説明
<value_expression>	四半期を計算する日付。結果は 1 ～ 4 の間で、1 が 1 月 1 日～ 3 月 31 日までを表します。

用途

指定した日が含まれる四半期を整数として返します。

例

```
SELECT {fn QUARTER ({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNQUARTER({FNCURDATE()})}
--------------------------
                         2
```

REPLACE

構文

REPLACE(*char*, *search_string* [, *replacement_string*])

用途

replacement_string で *search_string* をすべて置換して *char* を返します 。*char*、*search_string* および *replacement_string* は文字列引数です。

使用上の注意

replacement_string を指定しない場合、または NULL の場合、*search_string* がすべて取り除かれます。*search_string* が NULL の場合は、char が返されます。この関数は、TRANSLATE 関数を拡張したものです。TRANSLATE は 1 文字単位で置換します。REPLACE では、文字列を別の文字列で置き換えたり、文字列を削除できます。

例

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes" FROM DUAL;
```

次の結果を返します。

```
Changes
---------------
BLACK and BLUE
```

ROUND - 日付関数

構文

ROUND(*d* [,*fmt*])

次の表には、ROUND（および TRUNC）日付関数と使用する書式モデル、および日付が四捨五入される単位がリストされています。デフォルト・モデル「DD」は午前 0 時で切り上げられた日付を返します。

書式モデル	丸めの単位
CC または SCC	世紀
YYYY, SYYYY, YEAR, SYEAR, YYY, YY, Y	年（7 月 1 日で切り上げ）
IYYY, IYY, IY, I	ISO 年度
Q	四半期（四半期内の 2 か月目の 16 日で切り上げ）
MONTH, MON, MM, RM	月（16 日以降は切り上げ）
WW	年の最初の日と同じ曜日
IW	ISO 年度の最初の日と同じ曜日
W	月の最初の日と同じ曜日
DDD, DD, J	日
DAY, DY, D	開始曜日
HH, HH12, HH24	時間
MI	分

用途

d を書式モデル *fmt* で指定した単位に丸めて返します。*fmt* を指定しないと、*d* は最も近い日に丸められます。

例

```
SELECT ROUND(TO_DATE('27-OCT-92'),'YEAR')
"FIRST OF THE YEAR" FROM DUAL;
```

次の結果を返します。

```
FIRST OF
---------
1993-01-0
```

ROUND - 数値関数

構文

ROUND(*n* [,*m*])

用途

n を、小数点以下 *m* 桁で丸めた結果を返します。*m* が省略されると、0 が置かれます。*m* を負にして、小数点以上の桁で丸めることができます。*m* は整数にする必要があります。

例 1

```
SELECT ROUND (54.339, 2) FROM DUAL;
```

次の結果を返します。

```
ROUND(54.339
------------
54.34
```

RPAD

構文

RPAD(*char1*,*n* [,*char2*])

用途

char1 の右に *char2* で指定した文字を連続的に埋め込んで *n* 桁にして返します。*char2* は単一の空白にデフォルト設定されます。*char1* が *n* よりも長い場合、関数は *char1* の *n* に収まる部分を返します。

引数 *n* は、端末画面に表示される際の戻り値の合計長です。ほとんどのキャラクタ・セットで、これは戻り値の文字の数でもあります。ただし、マルチバイト・キャラクタ・セットの中には、文字列の表示長が文字列内の文字数によって異なるものがあります。

例

```
SELECT RPAD('ename',12,'ab') "RPAD example"
FROM emp
WHERE ename = 'TURNER';
```

次の結果を返します。

```
RPAD example
-------------
enameababababa
```

RTRIM

構文

RTRIM(*char* [,*set*])

用途

set に指定されたすべての文字を char の右側から順に取り除き、*char* を返します。このデフォルト値は単一の空白です。

例 1

```
SELECT RTRIM ('TURNERyxXxy', 'xy') "RTRIM example" FROM DUAL;
```

次の結果を返します。

```
RTRIM examp
-----------
TURNERyxX
```

例 2

```
SELECT {fn RTRIM ('TURNERyxXxy', 'xy')} FROM DUAL;
```

次の結果を返します。

```
{{FNRTRIM('T
-----------
TURNERyxX
```

ODBC 関数

```
{fn RTRIM (char)}    (trims leading blanks)
```

SECOND

構文

SECOND (*time_exp*)

用途

秒を 0 ～ 59 までの整数値として返します。

例 1

```
SELECT {FN SECOND ('14:03:01')} FROM DUAL;
```

次の結果を返します。

```
{FNSECOND('14:03:01')}
----------------------
                     1
```

例 2

```
SELECT {fn SECOND({fn CURTIME()})} FROM DUAL;
```

次の結果を返します。

```
{FNSECOND({FNCURTIME()})}
-------------------------
                       59
```

STDDEV

構文

STDDEV([DISTINCT|ALL] *x*)

用途

数値として、*x* の標準偏差を返します。Oracle Lite は、標準偏差を VARIANCE グループ関数のために定義された分散の平方根として計算します。

例

```
SELECT STDDEV(sal) "Deviation" FROM emp;
```

次の結果を返します。

```
Deviation
---------
1182.5032
```

SUBSTR

構文

SUBSTR(*char*, *m* [, *n*])

用途

char の *m* 番目の文字で始まる、長さ *n* 文字の文字列を抜き出して返します。

使用上の注意

m が正の場合、SUBSTR は *char* の先頭から数えて最初の文字を検索します。*m* が負の場合、SUBSTR は *char* の終わりから逆方向に数えます。値 *m* は 0 にはできません。*n* が省略されると、SUBSTR は *char* の終わりまでのすべての文字を返します。*n* に 1 未満の値は指定できません。

例

```
SELECT SUBSTR('ABCDEFG',3,4) "Subs" FROM DUAL;
```

次の結果を返します。

```
Subs
----
CDEF
```

SUBSTRB

構文

SUBSTRB(*char*, *m* [,*n*])

用途

文字列引数 *char* の *m* バイト目の文字で始まる、長さ *n* バイトの文字列を抜き出して返します。引数 *m* と *n* を文字数ではなくバイト数で指定することを除くと、SUBSTR と同じです。シングルバイトのデータベース・キャラクタ・セットの場合、SUBSTRB は、SUBSTR と同じです。

例

```
SELECT SUBSTRB('ABCDEFG',5,4) "Substring with bytes" FROM DUAL;
```

次の結果を返します。

```
Substring with bytes
--------------------
EFG
```

SUM

構文

SUM([DISTINCT | ALL] *n*)

用途

n の値の合計値を返します。

例

```
SELECT deptno, SUM(sal) TotalSalary FROM emp GROUP BY deptno;
```

次の結果を返します。

```
   DEPTNO TOTALSALARY
--------- -----------
       10        8750
       20       10875
       30        9400
```

SYSDATE

構文

```
SYSDATE
```

用途

現在の日付と時刻を返します。引数は必要ありません。

使用上の注意

この関数は Oracle Lite DATE 型列の条件としては使用できません。時刻は TIME 列の中でのみ、また日付と時刻は TIMESTAMP 列でのみ使用できます。

例

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') NOW FROM DUAL;
```

次の結果を返します。

```
NOW
-------------------
04-12-1999 19:13:48
```

TIMESTAMPADD

構文

{fn TIMESTAMPADD (<*interval*>, <*value_exp1* >, <*value_exp2* >)}
<*value_exp1* > + <*value_exp2* >

引数	説明
<interval>	2 番目のオペランド *<value_exp1>* の単位を指定する。次のキーワードが間隔の有効値です。 SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR
<value_exp1>	整数。
<value_exp2>	タイム・スタンプ。
<value_expression>	オペランド。

用途

カレント・タイム・スタンプに日付時刻値を加算します。

例

次の例は、1999-04-13 のカレント・タイム・スタンプに 1 日を加算します。

```
SELECT {fn TIMESTAMPADD (SQL_TSI_DAY, 1, {fn NOW()})} FROM DUAL;
```

次の結果を返します。

```
{FNTIMESTA
----------
1999-04-14
```

TIMESTAMPDIFF

構文

```
{fn TIMESTAMPDIFF (<interval>, <value_exp1 >, <value_exp2 >)}
<value_expression > - <value_expression >
```

引数	説明
<interval>	2 番目のオペランド *<value_exp1>* の単位を指定する。次のキーワードが間隔の有効値です。 SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR
<value_exp1>	整数。
<value_exp2>	タイム・スタンプ。
<value_expression>	オペランド。

用途

指定した間隔を使用して、2 つのタイム・スタンプ値の差を計算します。

例 1

```
SELECT {fn TIMESTAMPDIFF (SQL_TSI_DAY, {fn CURDATE()}, '1998-12-09')} FROM DUAL;
```

次の結果を返します。

```
{FNTIMESTAMPDIFF(SQL_TSI_DAY
----------------------------
                        -125
```

例 2

```
SELECT ENAME, {fn TIMESTAMPDIFF (SQL_TSI_YEAR, {fn CURDATE()},HIREDATE)} FROM EMP;
```

次の結果を返します。

```
ENAME      {FNTIMESTAMPDIFF(SQL_TSI_YEA
---------- ----------------------------
KING                                 -17
BLAKE                                -17
CLARK                                -17
```

```
JONES                                -18
MARTIN                               -17
ALLEN                                -18
TURNER                               -17
JAMES                                -17
WARD                                 -18
FORD                                 -17
SMITH                                -18
SCOTT                                -16
ADAMS                                -16
MILLER                               -17

14 rows selected.
```

TO_CHAR

日付の場合の構文

TO_CHAR(*d* [, *fmt*])

数値の場合の構文

TO_CHAR(*n* [, *fmt*])

用途

オプションの書式 *fmt* を使用して、日付または数値を VARCHAR2 データ型に変換します。

引数	説明
d	日付列または SYSDATE
fmt	書式文字列
n	数値列またはリテラル

使用上の注意

- *fmt* を指定しない場合、引数 *d* または *n* は、VARCHAR2 値に変換されます。日付の場合、引数 *d* はデフォルトの日付書式で返されます。数値の場合、引数 *n* は数値の有効桁に適切な長さの値に変換されます。
- TO_CHAR への引数として使用する場合、日付リテラルは DATE キーワードの後に続けてください。

POLITE.INI ファイル内に NLS_DATE_FORMAT パラメータを設定して、コンピュータ上のすべてのデータベースのデフォルトの日付書式を指定できます。**POLITE.INI** ファイル内

の NLS_DATE_FORMA パラメータの設定の詳細は、『Oracle Lite ユーザーズ・ガイド』を参照してください。

例

```
SELECT TO_CHAR (SYSDATE, 'Day, Month, DD, YYYY')"TO_CHAR example" FROM DUAL;
```

次の結果を返します。

```
TO_CHAR example
--------------------------------
Saturday , May      , 22, 1999
```

TO_DATE

構文

TO_DATE(*char* [, *fmt*])

用途

文字列引数 *char* を、DATE データ型の値に変換します。*fmt* 引数は、*char* の書式を指定する日付書式です。

例

```
SELECT TO_DATE('January 26, 1996, 12:38 A.M.', 'Month dd YYYY HH:MI A.M.') FROM
DUAL;
```

次の結果を返します。

```
TO_CHAR(TO_DATE('JANUARY26
----------------------------------------
1996-01-26 12:38:00
```

TO_NUMBER

構文

TO_NUMBER(*char* [, *fmt*])

用途

オプションの書式モデル *fmt* で指定された書式の数値を含む文字列引数 *char* を、NUMBER データ型の戻り値に変換します。

使用上の注意

- 日付書式および数値書式の詳細は、「書式」を参照してください。
- TO_DATE 関数の *char* 引数に DATE 値を指定しないでください。
- *fmt* またはデフォルトの日付書式によっては、返される DATE 値の世紀値が元の *char* とは異なる場合があります。
- Oracle 書式（'06-JUN-85' および '6-JUN-1985' など）、SQL-92 書式（'1989-02-28' など）、または NLS_DATE_FORMAT パラメータに指定された書式の日付は、日付列に挿入されるとき自動的に変換されます。
- **POLITE.INI** ファイル内に NLS_DATE_FORMAT パラメータを設定して、コンピュータ上のすべてのデータベースのデフォルトの日付書式を指定できます。**POLITE.INI** ファイル内の NLS_DATE_FORMA パラメータの設定の詳細は、『Oracle Lite ユーザーズ・ガイド』を参照してください。

例

次の例は、TO_NUMBER 関数で指定された値に従って、Blake という名前の従業員の給与を更新します。この例では、最初に Blake の給与を表示します。次に、Blake の給与を更新して、再表示します。

```
SELECT * FROM EMP WHERE ENAME = 'BLAKE';
```

次の結果を返します。

```
    EMPNO ENAME      JOB              MGR HIREDATE        SAL      COMM    DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
     7698 BLAKE      MANAGER        7839 1981-05-0      2850                  30

UPDATE EMP SET SAL = SAL + TO_NUMBER('100.52','9,999.99') WHERE ENAME = 'BLAKE';
```

次の結果を返します。

```
1 row updated.

SELECT * FROM EMP WHERE ENAME = 'BLAKE';
```

次の結果を返します。

```
    EMPNO ENAME      JOB              MGR HIREDATE        SAL      COMM    DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
     7698 BLAKE      MANAGER        7839 1981-05-0   2950.52                  30
```

TRANSLATE

構文

```
TRANSLATE(char, from, to)
```

用途

from 内のすべての文字を *to* 内の対応する文字で置換して *char* を返します。*char*、*from* および *to* は文字列引数です。

使用上の注意

- *char* 内の文字で *from* にない文字は置き換えられません。
- 引数 *from* には、*to* より多くの文字が含まれていてもかまいません。この場合、*from* の終わりのほうの余分な文字は *to* 内に対応する文字がありません。これらの余分な文字が *char* 内にあると、それらの文字は戻り値から削除されます。

空の文字列を *to* に使用して、*from* のすべての文字を戻り値から削除できません。TRANSLATE は空の文字列を NULL と解釈するため、NULL の引数があると NULL を返します。

例

```
SELECT TRANSLATE('2KRW229', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') "Licence" FROM DUAL;
```

次の結果を返します。

```
Licence
-------
9XXX999
```

TRIM

構文

```
TRIM( [[<trim_spec >] char ]
    FROM ] string )
```

<trim_spec> が省略されると、BOTH が含まれます。*char* が省略されると、空白が含まれます。

引数	説明
<trim_spec>	指定 : LEADING、TRAILING または BOTH
char	単一の文字。
string	切捨て対象のターゲット文字列。

用途

文字列から先行または後続の空白（またはその他の文字）、あるいは先行と後続の両方を削除します。

例

```
SELECT TRIM ('OLD' FROM 'OLDMAN') FROM DUAL;
```

次の結果を返します。

```
TRIM('
------
MAN
```

TRUNC

構文 - 数値引数の場合

TRUNC(*n* [, *m*])

構文 - 日付引数の場合

TRUNC(*d* [, *fmt*])

用途 - 数値引数の場合

小数第 *m* 位で切り捨てられた *n* を返します。*m* および *n* は数値引数です。*m* を指定しないと、小数第 1 位で切り捨てられます。*m* が負の場合、小数点の左の *m* 桁を切り捨てます（0 にします）。

用途 - 日付引数の場合

時刻部分が書式モデル *fmt* で指定された時刻単位まで切り捨てられた日付 *d* を返します。*fmt* を指定しない場合、*d* は近似日付に切り捨てられます。

使用上の注意

次の表には TRUNC（および ROUND）日付関数と使用する書式モデル、および日付が四捨五入される単位がリストされています。デフォルト・モデル「DD」は午前 0 時で切り上げられた日付を返します。

書式モデル	丸めの単位
CC または SCC	世紀
YYYY, SYYYY, YEAR, SYEAR, YYY, YY, Y	年（7 月 1 日で切り上げ）
IYYY, IYY, IY, I	ISO 年度
Q	四半期（四半期内の 2 か月目の 16 日で切り上げ）
MONTH, MON, MM, RM	月（16 日以降は切り上げ）
WW	年の最初の日と同じ曜日
IW	ISO 年度の最初の日と同じ曜日
W	月の最初の日と同じ曜日
DDD, DD, J	日
DAY, DY, D	開始曜日
HH, HH12, HH24	時間
MI	分

例 1

```
SELECT TRUNC(TO_DATE('27-OCT-92', 'DD-MON-YY'), 'YEAR') "First Of The Year"
FROM DUAL;
```

次の結果を返します。

```
First Of T
----------
1992-01-01
```

例 2

```
SELECT TRUNC(15.79,1) "Truncate" FROM DUAL;
```

次の結果を返します。

```
 Truncate
---------
     15.7
```

例 3

```
SELECT TRUNC(15.79,-1) "Truncate" FROM DUAL;
```

次の結果を返します。

```
 Truncate
---------
       10
```

UPPER

構文

UPPER(*char*)

用途

文字がすべて大文字に変換された文字列引数 *char* を返します。戻り値のデータ型は *char* と同じです。

例

```
SELECT UPPER('Carol') FROM DUAL;
```

次の結果を返します。

```
UPPER
-----
CAROL
```

ODBC 関数

{fn UCASE (char)}

USER

構文

```
USER
```

用途

現行のスキーマ名を文字列として返します。

例 1

```
SELECT USER "User" FROM DUAL;
```

次の結果を返します。

```
User
--------
SYSTEM
```

例 2

```
SELECT {fn USER()} FROM DUAL;
```

次の結果を返します。

```
{FNUSER()}
------------------------------
SYSTEM
```

ODBC 関数

```
{ fn USER()}
```

VARIANCE

構文

```
VARIANCE([DISTINCT|ALL] x)
```

用途

数値として、*x* の分散を返します。Oracle Lite は、次の式を使用して *x* の分散を計算します。

xi は、*x* の要素の 1 つです。

n は、*x* 集合の要素の数値です。*n* が 1 の場合、分散は 0 と定義されます。

例

```
SELECT VARIANCE(sal) "Variance" FROM emp;
```

次の結果を返します。

```
 Variance
---------
1398313.9
```

WEEK

構文

{ fn WEEK (<*value_expression*>) }

用途

年間通算週を整数で返します。

引数	説明
<*value_expression*>	週を計算する日付。結果は 1 ～ 53 です。

例 1

```
SELECT {fn WEEK({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNWEEK({FNCURDATE()})}
-----------------------
                     16
```

例 2

```
SELECT {fn week('1999-06-15')} FROM DUAL;
```

次の結果を返します。

```
EK('1999-06-15')}
-----------------
               25
```

YEAR

構文

YEAR (*date_exp*)

用途

YEAR を整数として返します。

例 1

```
SELECT {FN YEAR ('06-15-1966')} FROM DUAL;
```

次の結果を返します。

```
{FNYEAR('06-15-1966')}
----------------------
                  1966
```

例 2

```
SELECT {fn YEAR({fn CURDATE()})} FROM DUAL;
```

次の結果を返します。

```
{FNYEAR({FNCURDATE()})}
-----------------------
                   1999
```

4

SQL コマンド

この章では、Oracle Lite の SQL コマンドを説明します。説明する内容は次のとおりです。

- SQL コマンド・タイプ
- SQL コマンドの概要
- SQL コマンドのアルファベット順のリスト

SQL コマンド・タイプ

句および疑似列を含めた様々なタイプの SQL コマンドを以下にリストします。各 SQL コマンド、句および疑似列の説明は、「SQL コマンドの概要」を参照してください。

DDL（Data Definition Language、データ定義言語）コマンド

ALTER SEQUENCE	CREATE JAVA	DROP INDEX
ALTER SESSION	CREATE PROCEDURE	DROP JAVA
ALTER TABLE	CREATE SCHEMA	DROP PROCEDURE
ALTER TRIGGER	CREATE SEQUENCE	DROP SCHEMA
ALTER USER	CREATE SYNONYM	DROP SEQUENCE
ALTER VIEW	CREATE TABLE	DROP SYNONYM
CREATE DATABASE	CREATE TRIGGER	DROP TABLE
CREATE FUNCTION	CREATE USER	DROP TRIGGER
CREATE GLOBAL TEMPORARY 表	CREATE VIEW	DROP USER
CREATE INDEX	DROP FUNCTION	DROP VIEW
GRANT	REVOKE	TRUNCATE TABLE

DML（Data Manipulation Language、データ操作言語）

DELETE

EXPLAIN PLAN

INSERT

SELECT

subquery::=

UPDATE

トランザクション制御コマンド

COMMIT

ROLLBACK

SAVEPOINT

SET TRANSACTION

句

CONSTRAINT 句

DROP 句

疑似列

LEVEL 疑似列

ROWNUM 疑似列

SQL コマンドの概要

Oracle Lite では、いくつか異なるタイプの SQL コマンドを使用しています。この項では、様々なタイプの SQL コマンドについて説明します。

DDL（Data Definition Language、データ定義言語）コマンド

データ定義言語（DDL、Data definition language）コマンドで、次のタスクを実行できます。

- スキーマ・オブジェクトの作成、変更および削除
- 権限およびロールの付与と取消し
- データ・ディクショナリへのコメントの追加

CREATE、ALTER および DROP コマンドは、操作対象のオブジェクトに対する排他的アクセスが必要です。たとえば、別のユーザーが指定された表にトランザクションをオープンしていると、ALTER TABLE コマンドは失敗します。

DML（Data Manipulation Language、データ操作言語）コマンド

データ操作言語（DML、Data manipulation language）コマンドは、既存のスキーマ・オブジェクト内のデータの問合せと操作を行います。これらのコマンドは、カレント・トランザクションを暗黙的にはコミットしません。

トランザクション制御コマンド

トランザクション制御コマンドは、DML コマンドによる変更を管理します。

句

句は、コマンドを変更する、コマンドのサブセットです。

疑似列

疑似列は、コマンドから生成された値で、表内の列のように動作しますが、実際に表内に格納されません。疑似列は Oracle によってサポートされていますが、SQL-92 の一部ではありません。

SQL コマンドのアルファベット順のリスト

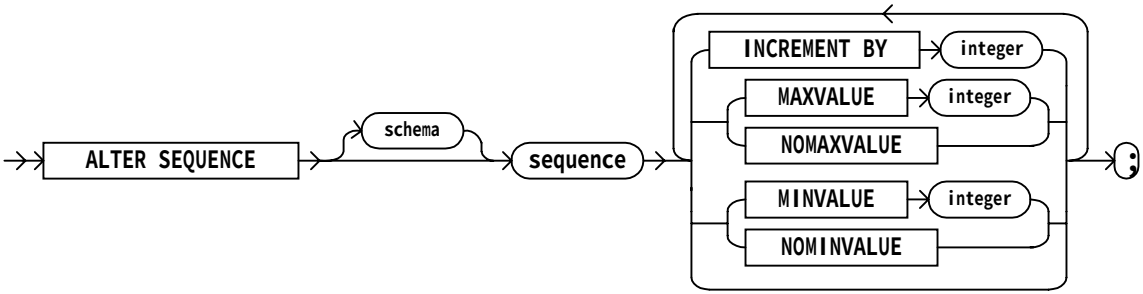
この項では、Oracle Lite SQL コマンド、句および疑似列をアルファベット順にリストし、それぞれの説明をしています。説明には次の項目が含まれます。

- 構文
- 用途
- 前提条件
- 引数と説明
- 使用上の注意
- 例
- 関連項目
- ODBC 関数（該当する場合）

注意： 例はすべて Oracle Lite と共に提供されるサンプル・データベース・オブジェクトを参照します。サンプル・データベース・オブジェクトの構造やデータを変更する DDL の例もあります。サンプル・データベース・オブジェクトが変更されないように、DDL の例をデータベース内で試した後は、そのつど ROLLBACK コマンドを使用してください。

ALTER SEQUENCE

構文



用途

次の方法のいずれかを使用して、順序を変更します。

- 将来の順序番号間の増分値を変更する
- 最小値または最大値を設定または排除する

前提条件

順序がユーザーのスキーマ内に必要です。

引数	説明
schema	順序を含むスキーマの名前。*schema* を省略すると、Oracle Lite はユーザー自身のスキーマ内で順序を変更します。
sequence	変更される順序の名前。
INCREMENT BY	順序番号間の間隔を指定します。0 以外の任意の正の整数または負の整数にできます。負の整数の場合、順序は降順になります。正の整数を指定すると、順序は昇順になります。この値は 10 桁以下になります。この値の絶対値は、MAXVALUE と MINVALUE の差より小さくする必要があります。INCREMENT BY 句を指定しない場合、デフォルトは 1 です。
MAXVALUE	順序で生成できる最大値を指定します。この整数値は 10 桁以下になります。MAXVALUE は、MINVALUE より大きくする必要があります。

引数	説明
NOMAXVALUE	昇順に対しては 2147483647、降順に対しては -1 の最大値を指定します。
MINVALUE	順序で生成できる最小値を指定します。この整数値は 10 桁以下になります。MINVALUE は、MAXVALUE よりも小さくする必要があります。
NOMINVALUE	昇順に対しては 1、降順に対しては -2147483647 の最小値を指定します。

使用上の注意

- 別の数値で順序を再開するには、順序を削除し再作成してください。ALTER SEQUENCE コマンドの影響を受けるのは、将来の順序数値だけです。
- Oracle Lite は、いくつか妥当性チェックを実行します。たとえば、現在の順序番号より小さい新規 MAXVALUE、または現在の順序番号より大きい新規 MINVALUE は指定できません。

例

この文は、ESEQ 順序に新規 MAXVALUE を設定します。

```
ALTER SEQUENCE eseq MAXVALUE 1500
```

ODBC 2.0

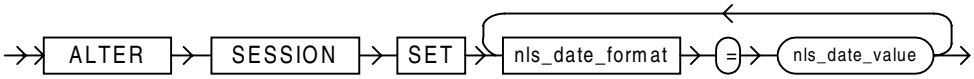
ALTER SEQUENCE コマンドは ODBC SQL の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

関連項目

CREATE SEQUENCE、DROP SEQUENCE

ALTER SESSION

構文



用途

データベースへの接続に影響を与える条件やパラメータを指定または変更します。Oracle Lite では、NLS 日付書式の指定や変更に対してのみ、このコマンドの SET 句を使用できます。文は、ユーザーがデータベースから切断されるまで有効です。

前提条件

なし。

引数	説明
parameter_name	Oracle Lite では、ALTER SESSION コマンドは NLS_DATE_FORMAT という名前のパラメータを 1 つのみ持ちます。
parameter_value	NLS 日付書式です。たとえば、次のとおりです。'YYYY MM DD HH24:MI:SS'

例

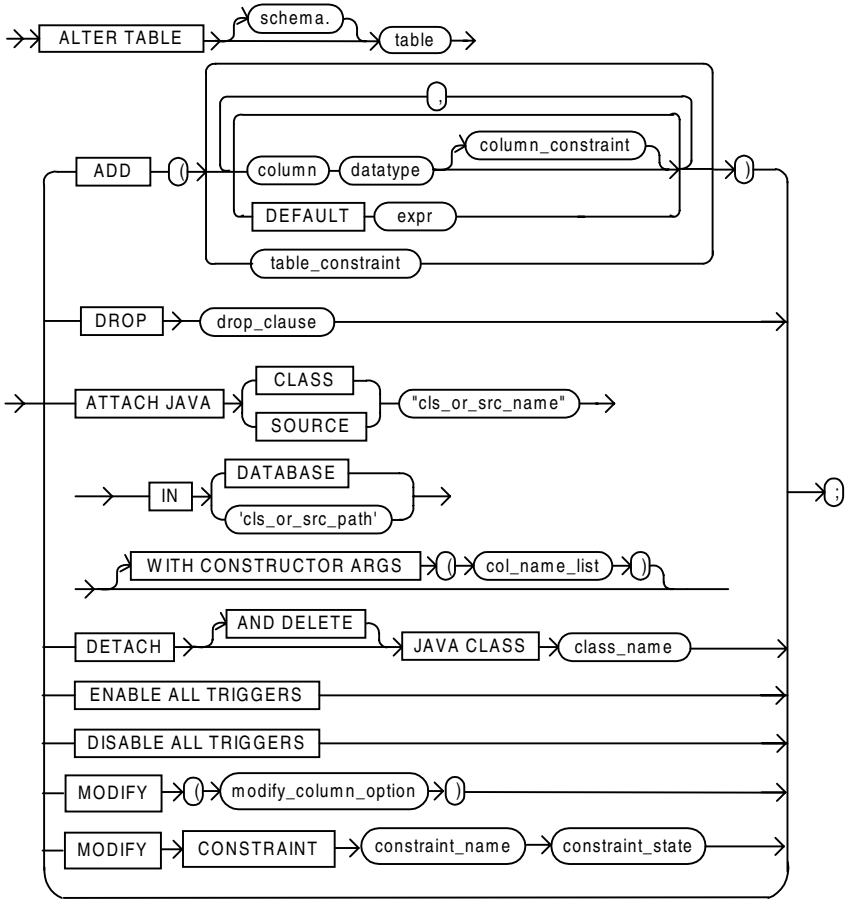
```
ALTER SESSION
SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
```

Oracle Lite は、新しいデフォルトの日付書式を使用します。

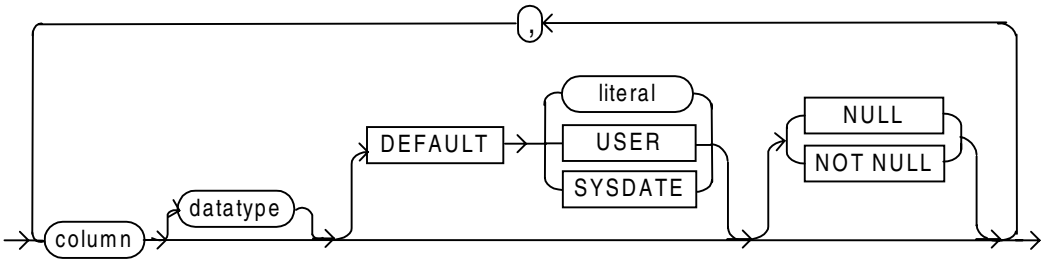
```
SELECT TO_CHAR(SYSDATE) Today FROM DUAL;

TODAY
-------------------
1997 08 12 14:25:56
```

ALTER TABLE

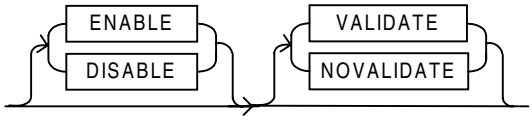
構文



modify_column_option::=



constraint_state::=



用途

次の方法のいずれかを使用して、表の定義を変更します。

- 列または整合性制約を追加する
- 列または整合性制約を削除する
- Java クラスを連結する
- Java クラスを連結解除する
- 列のデフォルト値を追加または変更する
- 列のデータ型またはサイズを変更する
- 制約を無効化または有効化する
- 列が NULL かどうかのプロパティを変更する

前提条件

表が、ユーザーのスキーマ内に必要です。データベースには SYSTEM、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	スキーマの名前で、最大 128 文字の文字列。各ユーザー名は、同じ名前のデフォルトのスキーマが付いているため、スキーマ名は、どのユーザー名とも異なる必要があります。ユーザー名と同じ名前のスキーマを作成すると、Oracle Lite は、エラーを返します。詳細は、「CREATE USER」を参照してください。
table	データベースの表の名前。
ADD	列または整合性制約がデータベースの表に追加されるよう指定します。
DROP	列または整合性制約がデータベースの表から削除されるよう指定します。
column	データベースの列の名前。
datatype	データベースの列のデータ型。
DEFAULT	新しい列に対してデフォルト値 *expr*（式）を指定するか、既存の列に対して新規デフォルト *expr* を指定します。
expr	有効な式。式が評価されるのは ALTER TABLE が実行されるときで、行にデフォルト値が挿入されるときではありません。詳細は、「式の指定」を参照してください。
column_constraint	列整合性制約。詳細は、「CONSTRAINT 句」を参照してください。NOT NULL 制約を持つ列を、すでにデータを含んでいる表に追加できません。
table_constraint	表整合性制約。詳細は、「CONSTRAINT 句」を参照してください。
drop_clause	削除される整合性制約。詳細は、「DROP 句」を参照してください。
ATTACH JAVA	Java クラスまたはソース・ファイルをデータベースの表に連結します。
IN	Java クラスまたはソース・ファイルを、データベース内、Java クラスまたはソース・パスに連結する必要があることを示します。
DATABASE	Java クラスまたはソース・パスを連結するデータベース。
DETACH	Java クラスをデータベースの表から削除します。
CLASS	Java クラスを指定します。
SOURCE	Java ソース・ファイルを指定します。
cls_or_src_name	完全修飾 Java クラス名またはソース・ファイル名。
cls_or_src_path	指定された Java クラスまたはソース・ファイルを含むディレクトリ。

引数	説明
WITH CONSTRUCTOR ARGS	Java コンストラクタの引数として使用されるクラスの属性を指定します。
col_name_list	データベースの表内の列（属性）のリスト。
AND DELETE	Java クラスをデータベースから削除します。
class_name	完全修飾 Java クラス名。
ENABLE ALL TRIGGERS	表に対応付けられたトリガーをすべて有効にします。トリガーは、トリガー条件が満たされたときに起動されます。単一のトリガーを有効にするには、ALTER TRIGGER の ENABLE 句を使用します。「ALTER TRIGGER」を参照してください。
DISABLE ALL TRIGGERS	表に対応付けられたトリガーをすべて無効にします。無効にされたトリガーは、トリガー条件が満たされても起動されません。単一のトリガーを無効にするには、ALTER TRIGGER の DISABLE 句を使用します。「ALTER TRIGGER」を参照してください。
MODIFY	既存の列に新しいデフォルトを指定します。後続の INSERT 文が列の値を省略した場合、Oracle Lite がその列にこの値を割り当てます。デフォルト値のデータ型は、列に指定されたデータ型に一致する必要があります。また、列にはデフォルト値を保持するための十分な長さが必要です。
modify_column_option	既存の列の定義を変更します。データ型、デフォルト値（リテラル、USER または SYSDATE）または列制約状態（NULL、NOT NULL）など、省略された列定義のオプション部分は変更されません。既存のデータが、データ変換によりエラーを発生しないようなものであれば、既存のデータを新しいデータ型に変更できます。既存のサイズが 15 文字以上の VARCHAR 列のサイズを増加した場合、データ変換は必要ありません。それ以外の変更はすべてデータ変換が必要です。各列は個別に変換されます。データ型を変更すると、そのつどすべてのオブジェクトが再書込みされ、依存するすべての索引が作成されます。 KEY COLUMNS 句を使用して作成された索引の列のデータ型を変更すると、索引の再作成で KEY COLUMNS オプションの再設定が不可能なために ALTER TABLE MODIFY コマンドが失敗する場合があります。KEY COLUMNS 句を使用して作成された索引は、列を変更する前に削除する必要があります。
CONSTRAINT	既存の制約の状態を変更します。ENABLE は、表内のすべての新規データに制約が適用されることを指定します。参照整合性制約を有効にする前に、参照される制約を有効にする必要があります。

引数	説明
ENABLE VALIDATE	この設定では、既存のデータがすべて制約上準拠することを指定します。制約検証を有効にすると、それ以降すべてのデータの有効性が保証されます。主キー制約を ENABLE VALIDATE モードで設定すると、検証処理により主キー列に NULL が含まれないように保証されます。 VALIDATE や NOVALIDATE が省略された場合、デフォルトは VALIDATE です。
ENABLE NOVALIDATE	この設定により、制約付きデータに対するすべての新規 DML 操作が制約に準拠することが保証されますが、表内の既存のデータが制約に準拠することは保証されません。 主キー制約を有効にすると自動的に 1 次索引が作成され、制約が施行されます。主キー制約が無効にされた場合、この索引は通常の索引に変換されます。制約が再度有効になると、主キー制約に対して索引がチェックされ、違反が検出されない場合は主キー状態にリストアされます。
DISABLE VALIDATE	この設定は制約を無効にして、主キー制約の索引を通常の索引に変換しますが、制約は有効なまま保持します。SQLRT エンジンを介して表に DML 文を実行できませんが、Oracle Lite Java アクセス・クラス（JAC）を介すると DML 文を実行できます。 VALIDATE や NOVALIDATE が省略された場合、デフォルトは NOVALIDATE です。
DISABLE NOVALIDATE	この設定では、Oracle Lite は制約をメンテナンスせず（無効にされているため）、また制約が真であることを保証できません（検証されていないため）。主キー制約索引は通常の索引にダウングレードされます。 外部キー制約が DISABLE NOVALIDATE 状態でも、主キーが外部キーにより参照されている表は削除できません。

使用上の注意

表に新規列を追加するのに ADD 句を使用すると、新規列の各行の初期値は NULL となります。表が空であるかどうかに関わらず、デフォルト値が指定されている場合に限り、NOT NULL 制約を持つ列を追加できます。

VALIDATE や NOVALIDATE が ENABLE 引数から省略された場合、デフォルトは NOVALIDATE です。

VALIDATE や NOVALIDATE が DISABLE 引数から省略された場合、デフォルトは NOVALIDATE です。

NULL かどうかの制約が、MODIFY 句を整合性制約構文付きで使用して既存の列に追加できる唯一の整合性制約です。NOT NULL は、列に NULL が含まれていない場合のみ追加できます。NULL 制約は、主キー制約の構成要素でない列に対して追加できます。

例

次の文は、列 THRIFTPLAN と LOANCODE を EMP 表に追加します。THRIFTPLAN は、最大 7 桁、小数点以下 2 桁の、データ型 NUMBER を持ちます。LOANCODE は、サイズ 1、NOT NULL 整合性制約の、データ型 CHAR を持ちます。

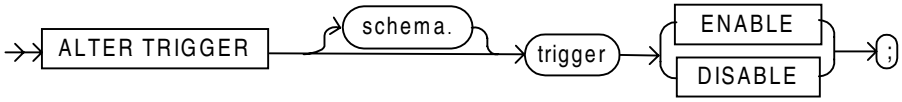
```
ALTER TABLE emp
ADD (thriftplan NUMBER(7,2),
loancode CHAR(1));
```

関連項目

CONSTRAINT 句、CREATE TABLE、CREATE VIEW

ALTER TRIGGER

構文



用途

データベース・トリガーを有効化または無効化します。トリガー作成の詳細は、「CREATE TRIGGER」を参照してください。トリガー削除の詳細は、「DROP TRIGGER」を参照してください。

注意： この文では、既存のトリガーの宣言または定義は変更されません。トリガーの再宣言または再定義には、CREATE TRIGGER 文に OR REPLACE を付けて使用してください。

前提条件

トリガーを変更するには、DBA/DDL 権限が必要です。

schema	トリガーを含むスキーマ。schema を省略すると、Oracle Lite は、トリガーがユーザー自身のスキーマ内にあると解釈します。
trigger	変更されるトリガーの名前。
ENABLE	トリガーを有効にします。表に対応付けられたトリガーをすべて有効にするには、ALTER TABLE の ENABLE ALL TRIGGERS 句も使用できます。「ALTER TABLE」を参照してください。
DISABLE	トリガーを無効にします。表に対応付けられたトリガーをすべて無効にするには、ALTER TABLE の DISABLE ALL TRIGGERS 句も使用できます。「ALTER TABLE」を参照してください。

例

INVENTORY 表に作成された REORDER という名前のトリガーを考えます。UPDATE 文で特定の部品の数が減らされ、在庫が部品の再注文基準を下回ると、常にトリガーが起動されます。このトリガーは、部品番号、再注文量および今日の日付を含む行を保留注文の表に挿入します。

このトリガーが作成されると、Oracle Lite は自動的にこれを有効にします。次の文を使用してトリガーを無効にできます。

```
ALTER TRIGGER reorder DISABLE;
```

トリガーを無効にしたときは、UPDATE 文により部品の在庫が再注文基準を下回っても Oracle Lite はトリガーを起動しません。

トリガーを無効にした後は、次の文を使用してトリガーを有効にできます。

```
ALTER TRIGGER reorder ENABLE;
```

トリガーを再度有効にした後は、UPDATE 文の結果として部品の在庫が再注文基準を下回ったときに Oracle Lite がトリガーを起動します。トリガーが無効なときに、部品の在庫が再注文基準を下回る可能性があります。この場合、トリガーを再度有効にしても、別のトランザクションが在庫をさらに減らすまで、Oracle Lite はこの部品に対するトリガーを自動的には起動しません。

関連項目

CREATE TRIGGER

ALTER USER

用途

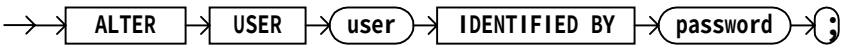
データベースのユーザー・パスワードを変更します。

前提条件

次の条件のいずれかを満たすと、データベース内のユーザー・パスワードを変更できます。

- そのユーザーとしてデータベースに接続している。
- SYSTEM として、または DBA/DDL か ADMIN 権限を持つユーザーとしてデータベースに接続している。
- ユーザーに ADMIN または DBA/DDL ロールの権限を付与されている。

構文



引数	説明
user	変更されるユーザー。user は、30 文字以内の、文字で始まる一意のユーザー名です。user の最初の文字に、空白文字は使用できません。
IDENTIFIED BY	Oracle Lite がユーザー・アクセスを許可する方法を示します。
password	最大 128 文字の名前の、ユーザーの新しいパスワードを指定します。パスワードは、引用符には入れられず、大 / 小文字区別はありません。

例

次の例は、「todd」という名前で、パスワード「tiger」で識別されるユーザーを作成します。次に、ユーザーのパスワードを「lion」に変更します。

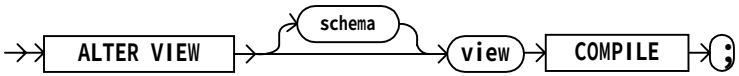
```
CREATE USER todd IDENTIFIED BY tiger;

ALTER USER todd IDENTIFIED BY lion;
```

関連項目

CREATE USER、DROP USER

ALTER VIEW

構文



用途

ビューを再コンパイルします。

前提条件

ビューがユーザーのスキーマ内に必要です。データベースには SYSTEM、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	ビューを含むスキーマ。schema を指定しない場合、Oracle Lite はユーザーのスキーマにビューを作成します。
view	再コンパイルされるビューの名前。
COMPILE	Oracle Lite によってビューが再コンパイルされます。COMPILE キーワードは必須です。

使用上の注意

ALTER VIEW を使用して、無効なビューを明示的に再コンパイルできます。明示的な再コンパイルにより、ランタイム前に再コンパイル・エラーを検出できます。ベース表の 1 つを変更した後に、ビューを明示的に再コンパイルすることが必要な場合があります。これは、変更により依存するビューまたはその他のオブジェクトに影響を与えないようにするためです。ALTER VIEW 文を発行すると、Oracle Lite は、ビューが有効か無効かにかかわらず、ビューを再コンパイルします。また、Oracle Lite は、ビューに依存するローカル・オブジェクトをどれも無効にします。

このコマンドは、既存のビューの定義を変更しません。ビューを再定義するには、CREATE VIEW コマンドを OR REPLACE オプションと共に使用してください。

例

次のコードは、ALTER VIEW SQL コマンドを表します。COMPILE キーワードは必須です。

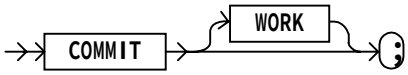
```
ALTER VIEW customer_view COMPILE;
```

関連項目

CREATE VIEW、DROP VIEW

COMMIT

構文



用途

カレント・トランザクションを終了し、データベースの変更をすべて永久的なものにします。

前提条件

なし。

引数	説明
WORK	影響を与えることのないオプションの引数。WORK は、標準 SQL 準拠という目的でのみサポートされています。文 COMMIT と COMMIT WORK は同じです。

使用上の注意

Oracle Lite は、CREATE DATABASE 以外は、どの DDL 文も自動コミットしません。カレント・トランザクションをコミットして、データベースの変更をすべて永久的なものにする必要があります。

例

次のコードは、COMMIT コマンドを表します。この例では行が DEPT 表に挿入され、変更がコミットされます。WORK 引数はオプションです。

```
INSERT INTO dept VALUES (50, 'Marketing', 'TAMPA');

COMMIT;
```

ODBC 2.0

COMMIT コマンドは ODBC SQL 構文の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

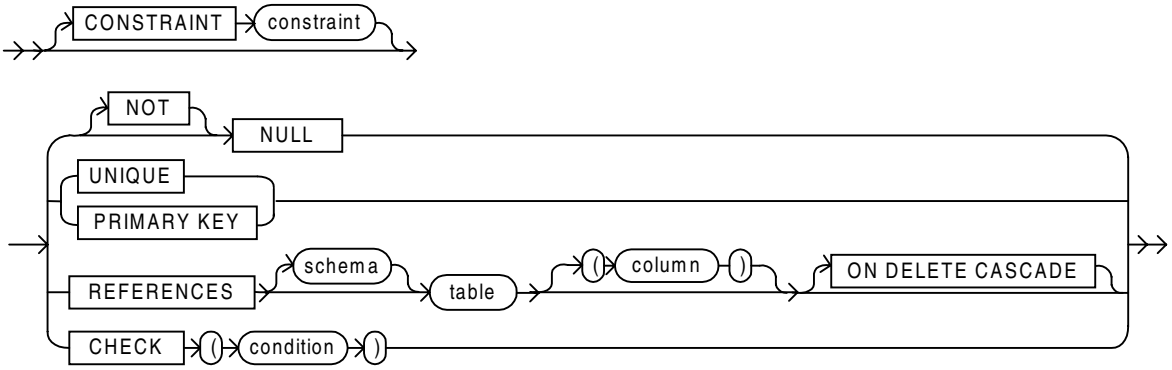
ODBC プログラムは通常、SQL_COMMIT フラグを指定した API コール `SQLTransact()` を使用します。

関連項目

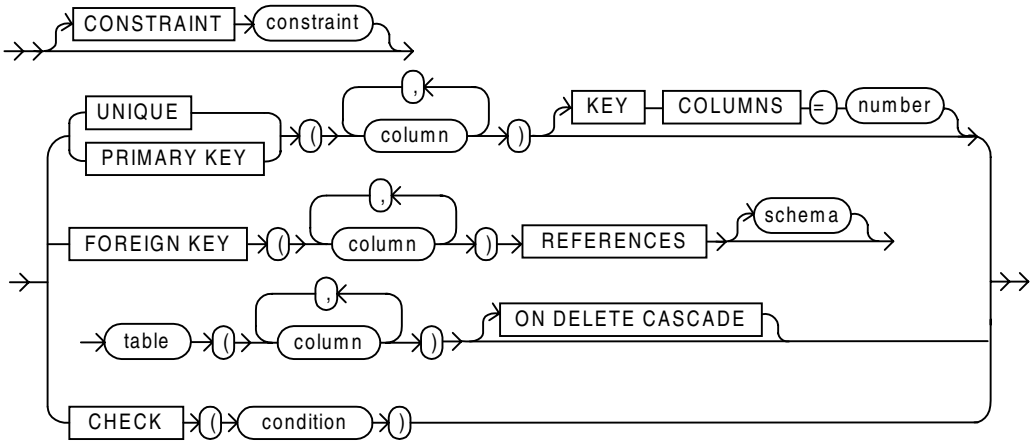
ROLLBACK

CONSTRAINT 句

列制約の構文



表制約の構文



用途

整合性制約を定義します。

前提条件

CONSTRAINT 句は、CREATE TABLE および ALTER TABLE コマンドの中に入れることができます。整合性制約を定義するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。Oracle Lite のみが整合性制約を持ちます。

引数	説明
CONSTRAINT	constraint 引数で指定された整合性制約を識別します。Oracle Lite は、制約名と定義をデータ・ディクショナリに格納します。CONSTRAINT キーワードを省略すると、Oracle Lite は形式 POL_SYS_CONS*n* で名前を生成します。*n* は、データベース内でその名前を一意にする整数値です。
constraint	追加される制約の名前。
NULL	列に NULL 値を入れられることを指定します。
NOT NULL	列に NULL 値は入れられないことを指定します。デフォルトでは、列に NULL 値を入れられます。

引数	説明
UNIQUE	列または列の組合せを一意キーとして指定します。
PRIMARY KEY	列または列の組合せを表の主キーとして指定します。
KEY COLUMNS =	索引の作成に使用する列数を指定します。この句を使用すると索引のサイズが減るため、大量の列に索引が必要なときに便利です。複数の行が、KEY COLUMNS の値で指定された索引キーの列を接頭辞として修飾している場合、問合せの性能に影響する可能性があります。これは、一致する行を検索するためにデータベースがすべての修飾行を参照することが原因です。
number	KEY COLUMNS の個数を指定する整数。
FOREIGN KEY	子表内の列または列の組合せを、参照整合性制約内の外部キーとして指定します。
schema	スキーマの名前で、最大 128 文字の文字列。各ユーザー名は、同じ名前のデフォルトのスキーマが付いているため、スキーマ名は、どのユーザー名とも異なる必要があります。ユーザー名と同じ名前のスキーマを作成すると、Oracle Lite は、エラーを返します。詳細は、CREATE USER を参照してください。
REFERENCES	参照整合性制約内の外部キーによって参照される親表の主キーまたは一意キーを識別します。
table	制約が置かれる表を指定します。*table* のみを指定して、*column* 引数を省略すると、外部キーは自動的に表の主キーを参照します。
column	制約が置かれる表の列を指定します。
ON DELETE CASCADE	主キーまたは一意キー値を削除するときに、Oracle Lite が、依存外部キー値を自動的に削除して、参照整合性をメンテナンスすることを指定します。
CHECK	表内の各列でチェックされる条件を指定します。Oracle Lite は、CHECK 条件内で、次の演算子と関数のみをサポートします。 + - / * = ! = < > < = > = IS NULL、LIKE、BETWEEN TO_NUMBER、TO_DATE、TRANSLATE
condition	表の各行が満たすべき条件を指定します。有効な条件の作成の詳細は、「SQL 条件の指定」を参照してください。

例

次の例は、列 A、B を持つ表 T を作成します。例では、PRIMARY KEY 制約句を使用して、列 A を表の主キーにします。

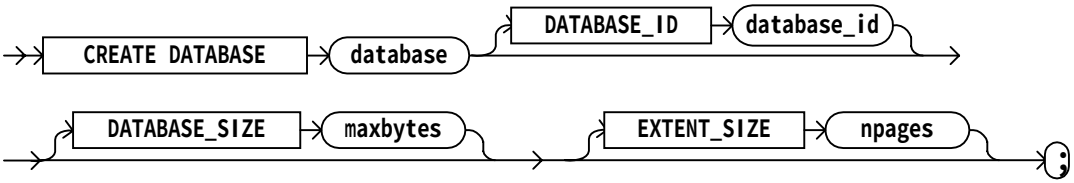
```
CREATE TABLE T (A CHAR(20) PRIMARY KEY, B CHAR(20));
```

関連項目

ALTER TABLE、CREATE TABLE

CREATE DATABASE

構文



用途

データベースを作成します。

前提条件

なし。

引数	説明
database	データ・ファイル名またはフルパス名。フルパス名は、二重引用符で囲む必要があります。パス名が指定されていないと、ODBC 接続の場合、データ・ソース名（DSN）に指定されたディレクトリ内にデータ・ファイルが作成されます。フルパス名も DSN も有効でない場合、カレント作業ディレクトリ内にデータベースが作成されます。*database* の長さは、オペレーティング・システムまたはファイル・システムにより制限されます。重複するデータベース名を使用すると、エラーが発生します。
DATABASE_ID	データベースのオプションの数値識別子。

引数	説明
database_id	データベースの一意の識別子。16 ～ 32765 の一意の数値にする必要があります。指定がない場合、デフォルトの初期値は 64 です。**POLITE.INI** ファイル内の *database_id* パラメータは、次に利用可能なデータベース ID を指定します。同じデータベース ID で 2 つのデータベースを作成することは可能ですが、2 つのデータベースに、同時に接続できません。
DATABASE_SIZE	データベースのサイズ。
maxbytes	データベースが成長できる最大ファイル・サイズ。省略された場合、デフォルト値は 256M です。K、M、G などの省略形をそれぞれキロバイト、メガバイトおよびギガバイトのかわりに使用できます。省略形が指定されていないと、デフォルトは K です。省略形を指定する場合は、250 キロバイトから 4 ギガバイトまでの間の整数値を使用する必要があります。たとえば、256M、1000K、2G などです。
EXTENT_SIZE	データベース・ファイル内の増分ページ量。データベースがカレント・ファイル内でページが足りなくなると、ファイルの大きさをこのページ数ずつ拡張します。
npages	エクステント（表の割当ての最小単位）を構成する 4K（キロバイト）のページの数値。*npages* には 2 の倍数の数値が必須です。デフォルト値は 4 です。0 に設定すると、Oracle Lite は、*npages* にデフォルト値を設定します。

使用上の注意

ページ数は、64 以下にしてください。

キーワードはどのような順序でもリストできます。

新規に作成されたデータベースを実行する前に、ODBC Administrator を使用して、ODBC データ・ソース名（DSN）を構成する必要があります。DSN の作成および ODBC Administrator の使用方法については、『Oracle Lite ユーザーズ・ガイド』を参照してください。

他の DDL 文と異なり、Oracle Lite は、CREATE DATABASE コマンドを自動コミットします。CREATE DATABASE コマンドは、ROLLBACK 文を使用して取消しを行うことはできません。

例

データ・ファイル **LIN.ODB** を C:¥TMP ディレクトリ内に **.ODB** ファイル拡張子を付けて作成するには、以下を使用します。

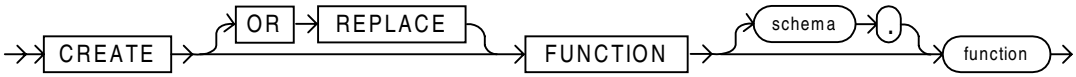
```
CREATE DATABASE "C:¥TMP¥LIN"
```

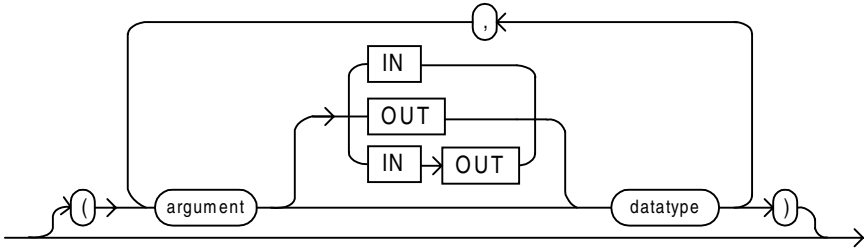
関連項目

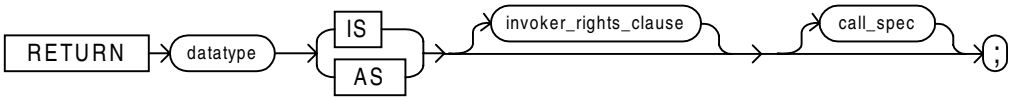
ROLLBACK

CREATE FUNCTION

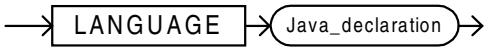
構文



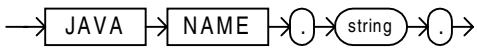




call_spec::=



Java_declaration::=



用途

ストアド・ファンクションのコール仕様を作成します。

「ストアド・ファンクション」（「ユーザー・ファンクション」とも呼ばれる）は、値を返す Java ストアド・プロシージャです。ストアド・ファンクションはプロシージャとよく似ていますが、プロシージャの場合、コールされた環境内に値を返しません。プロシージャおよびファンクションの全般的な説明は、「CREATE PROCEDURE」を参照してください。関数の作成例は、CREATE FUNCTION の例を参照してください。

「コール仕様」は、SQL からコールできるように Java メソッドを宣言します。コール仕様は、コール時にどの Java メソッドをコールするかを Oracle Lite に知らせます。また、Oracle Lite に、引数および戻り値に対してどの型変換を行うかも知らせます。

CREATE FUNCTION 文は、関数をスタンドアロンのスキーマ・オブジェクトとして作成します。スタンドアロン関数の削除については、「DROP FUNCTION」を参照してください。

前提条件

ユーザー自身のスキーマ内で関数を作成するには、データベースに SYSTEM として接続するか、DBA/DDL 権限が必要です。

コール仕様を呼び出すには、DBA/DDL 権限が必要です。

引数	説明
OR REPLACE	関数がすでに存在する場合、再作成します。この句は、すでに付与されているオブジェクト権限を変更（削除、再作成または再付与）せずに、既存の関数の定義を変更するために使用します。 再定義される関数に対する権限を前もって付与されているユーザーは、権限を再付与されなくてもその関数にアクセスできます。この関数に依存するファンクション索引があれば、Oracle Lite は、索引に DISABLED のマークを付けます。
schema	関数を含むスキーマ。schema を指定しないと、Oracle Lite はユーザーのカレント・スキーマに関数を作成します。
function	作成する関数の名前。「使用上の注意」を参照してください。
argument	関数の引数の名前。関数が引数を受け入れない場合、関数名の後のカッコを省略できます。
IN	関数をコールするときに、ユーザーが引数の値を提供する必要があることを指定します。これがデフォルトです。
OUT	関数が引数の値を設定することを指定します。

引数	説明
IN OUT	引数の値をユーザーが指定でき、さらに関数で設定される場合もあることを指定します。 ■ このパラメータまたは別のパラメータに行われた変更は、同じ変数が両方に渡された場合、両方の名前を通して即座に参照できます。 ■ 関数が未処理例外によって終了した場合、このパラメータに対して行われた割当ては、コール元の変数内で参照できます。 このような効果は、特定のコールで発生することもあれば、発生しないこともあります。これらの影響が問題ではない場合にのみ、NOCOPY を使用します。
datatype	引数のデータ型。引数は、SQL がサポートするどのデータ型でも取ることができます。データ型で長さ、精度またはスケールは指定できません。Oracle Lite は、引数の長さ、精度またはスケールを、関数のコール元の環境から導出します。
RETURN *datatype*	関数の戻り値のデータ型を指定します。関数は値を返す必要があるため、この句は必須です。戻り値は、SQL がサポートするどのデータ型でも取ることができます。 データ型で長さ、精度またはスケールは指定できません。Oracle Lite は、戻り値の長さ、精度またはスケールを、関数のコール元の環境から導出します。
IS	SQL 識別子を Java メソッドに対応付けます。
AS	SQL 識別子を Java メソッドに対応付けます。
invoker_rights_clause	Oracle との互換性のために、Oracle Lite は、*invoker_rights_clause* を認識しますが、施行はしません。
call_spec	Java メソッド名、パラメータ型および戻り型を、対応する SQL 要素にマップします。 LANGUAGE — *call_spec* の言語を指定します。Oracle8 では、C または Java が使用できます。Oracle Lite では、Java のみが使用できます。 *java_declaration* — Java クラス内のメソッド名を識別します。
JAVA NAME	Java メソッド名。
string	メソッドの Java 実装を識別します。詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

使用上の注意

ユーザー定義関数は、定義が変更できない状況では使用できません。次の場所では、ユーザー定義関数は使用できません。

- CREATE TABLE 文または ALTER TABLE 文における CHECK 制約句。
- CREATE TABLE 文または ALTER TABLE 文における DEFAULT 句。

さらに、問合せまたは DML 文の中からコールされる関数では次のことはできません。

- OUT または IN OUT パラメータを持つこと。
- カレント・トランザクションのコミットやロールバック、セーブポイントの作成やロールバック、またはセッションやシステムの変更。DDL 文は、カレント・トランザクションを暗黙的にコミットします。そのため、ユーザー定義関数は DDL 文を実行できません。
- 関数が SELECT 文からコールされているときに、データベースに書き込むこと。ただし、関数が DML 文内の副問合せからコールされている場合は、データベースに書き込めます。
- 関数が DML 文からコールされているときに、関数をコールしている文によって変更が行われている表と同じ表に書き込むこと。

OUT および IN OUT パラメータに対する制約以外は、Oracle Lite は、SQL 文から直接コールされる関数だけでなく、関数をコールする関数に対して、これらの制約を施行します。また、Oracle Lite は、その関数またはその関数からコールされる関数によって実行される SQL 文からコールされる関数に対しても、これらの制約を施行します。

例

次の例は、関数の作成とテストについての完全な手順を示しています。

1. 次の Java プログラムを作成してコンパイルし、その名前を `Employee.java` にします。

```
public class Employee {
  public static String paySalary (float sal, float fica, float sttax,
     float ss_pct, float espp_pct) {
   float deduct_pct;
   float net_sal;

   /* compute take-home salary */
   deduct_pct = fica + sttax + ss_pct + espp_pct;
   net_sal = sal * deduct_pct;

   String returnstmt = "Net salary is " + net_sal;
   return returnstmt;
 } /*paySalary */
}
```

2. Employee クラスを Oracle Lite データベースにロードします。一度ロードすると、Employee クラス・メソッドは、Oracle Lite データベース内のストアド・プロシージャになります。

```
CREATE JAVA CLASS USING BFILE ('C:¥', 'Employee.class');
```

3. employeeSalary メソッドは値を返すため、それを CREATE FUNCTION 文を使用して公開します。

```
CREATE FUNCTION
PAY_SALARY(
    sal float, fica float, sttax float, ss_pct float, espp_pct float)
    return varchar2
as language java name
'Employee.paySalary (float, float, float, float, float)
 return java.lang.String';
.
/
```

4. dual から、PAY_SALARY ストアド・プロシージャを選択します。

```
SELECT PAY_SALARY(6000.00, 0.2, 0.0565, 0.0606, 0.1) from dual;
```

次の結果を返します。

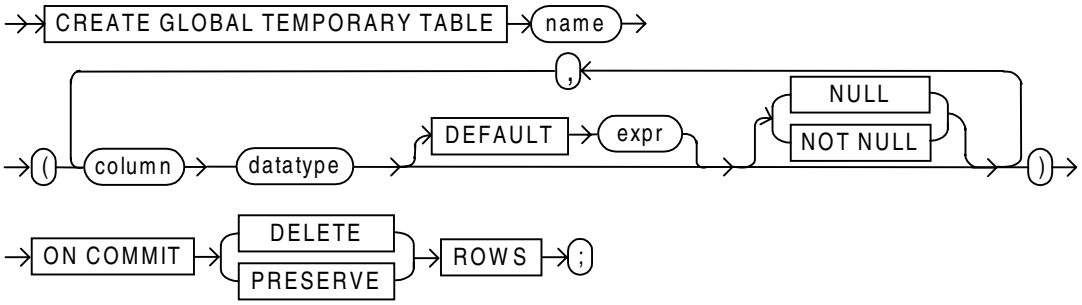
```
PAY_SALARY
----------------------------------------
Net Salary is 2502.6
```

関連項目

DROP FUNCTION

CREATE GLOBAL TEMPORARY 表

構文



用途

CREATE GLOBAL TEMPORARY TABLE コマンドは、トランザクション固有またはセッション固有となる一時表を作成します。トランザクション固有の一時表では、データが存在するのはトランザクション期間中です。セッション固有の一時表では、データが存在するのはセッション期間です。一時表内のデータはセッション用のプライベート・データです。各セッションは、自身のデータのみを表示および変更できます。トランザクションのロールバックでは、グローバル一時表に対するすべての変更が失われます。

引数と説明

引数	説明
name	オプションの修飾表名。
schema	所有者であるユーザーと同じ名前のスキーマ。省略すると、デフォルトのスキーマ名が使用されます。
column	表の列の名前。
datatype	column のデータ型。副問合せでは使用できません。
DEFAULT	新規列に、新しいデフォルト値 *expr*（式）を指定します。以下のいずれかになります。 ■ DEFAULT NULL ■ DEFAULT USER（表が作成された際のユーザー名） ■ DEFAULT リテラル 式の詳細は、「式の指定」を参照してください。

使用上の注意

一時表をパーティション化したり、索引に編成したり、クラスタ化することはできません。

一時表に、参照整合性（外部キー）制約は指定できません。

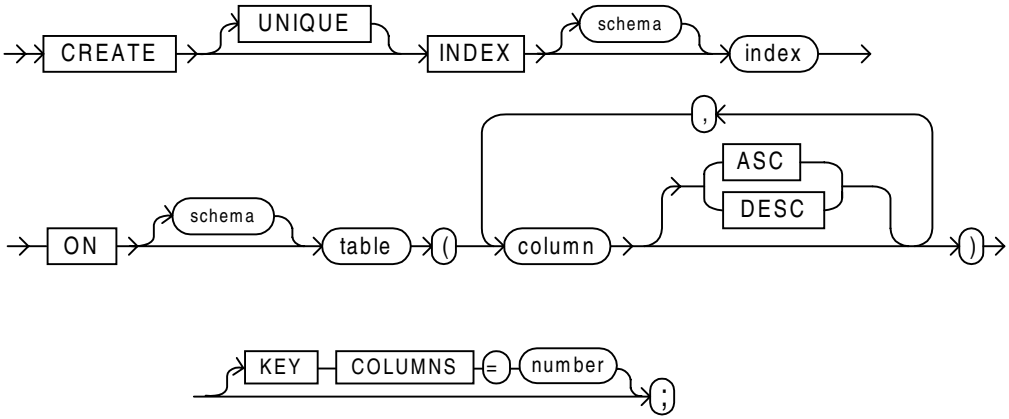
例

次の文は、自動航空機予約スケジューリング・システムで使用する一時表 FLIGHT_SCHEDULE を作成します。各クライアントは独自のセッションを持ち、一時スケジュールを格納できます。一時スケジュールは、セッション終了時に削除されます。

```
CREATE GLOBAL TEMPORARY TABLE flight_schedule (
    startdate DATE,
    enddate DATE,
    cost NUMBER)
    ON COMMIT PRESERVE ROWS;
```

CREATE INDEX

構文



用途

索引を表の 1 つ以上の列上に作成します。

前提条件

索引を付ける表がユーザーのスキーマ内に必要です。データベースには SYSTEM、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
UNIQUE	列または列の組合せを一意キーとして指定します。
schema	CREATE INDEX の後に来る場合は、索引を含むスキーマ。*schema* を省略すると、Oracle Lite はユーザー自身のスキーマ内で索引を作成します。 ON 句の中で使用した場合は、索引を作成する表を含むスキーマ。
index	作成する索引の名前。複数の索引に同じ列と列順を使用しない場合は、表にいくつでも索引を作成できます。
table	索引が作成される表の名前。スキーマで表を修飾しない場合、Oracle Lite は表がユーザー自身のスキーマに含まれていると解釈します。
column	表内の列の名前。索引列のデータ型には LONG または LONG RAW は指定できません。
ASC | DESC	DB2 との互換性のためにのみ提供されています。索引は常に昇順に作成されます。
KEY COLUMNS =	索引の作成に使用する列数を指定します。この句を使用すると索引のサイズが減るため、大量の列に索引が必要なときに便利です。複数の行が、KEY COLUMNS の値で指定された索引キーの列を接頭辞として修飾している場合、問合せの性能に影響する可能性があります。データベースは、一致する行を検索するためにすべての修飾行を参照します。
number	KEY COLUMNS の個数を指定する整数。

使用上の注意

チューニングのために、他の索引作成オプションも使用できます。しかし、これらのオプションは、データベースのパフォーマンスを下げる恐れがあるので、必要なときにのみ使用します。詳細は、表 E「索引作成オプション」を参照してください。

CREATE ANY INDEX を使用して別のスキーマに索引を作成できますが、DBA/DDL ロールが必要です。

例

次の例は、EMP 表の SAL 列に索引を作成します。

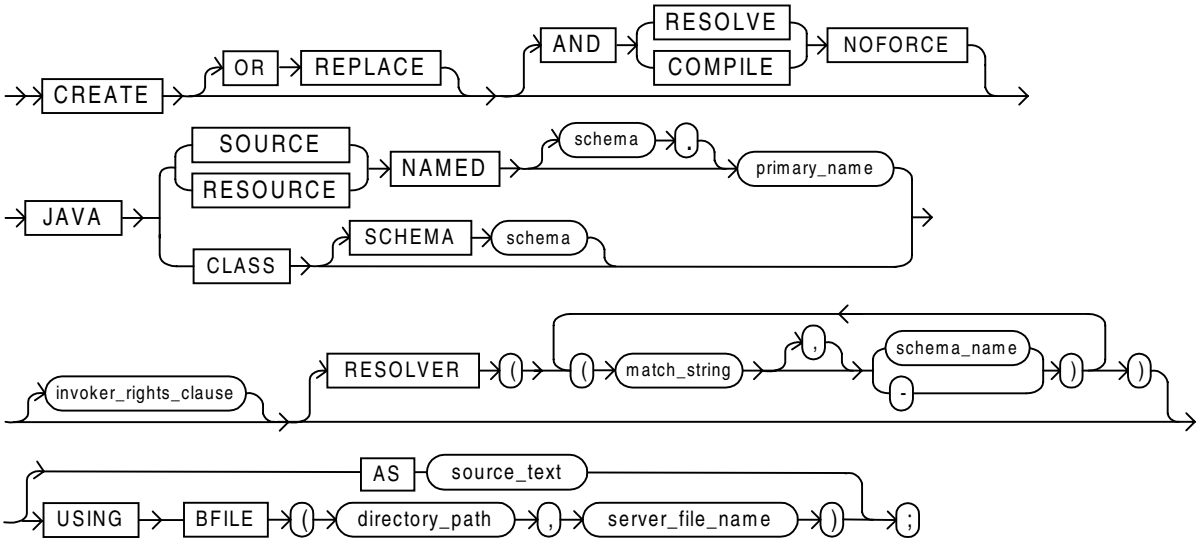
```
CREATE INDEX SAL_INDEX ON EMP(SAL);
```

関連項目

CONSTRAINT 句、CREATE TABLE、DROP INDEX

CREATE JAVA

構文



用途

Java ソース、クラスまたはリソースを含むスキーマ・オブジェクトを作成します。

注意： Java ストアド・プロシージャおよび JDBC をはじめとする Java 概念の詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

前提条件

ユーザー自身のスキーマ内で、Java ソース、クラスまたはリソースを含んだスキーマ・オブジェクトを作成または置換するには、データベースに SYSTEM として接続するか、DBA/DDL 権限が必要です。

引数	説明
OR REPLACE	Java クラス、ソースまたはリソースを含んだスキーマ・オブジェクトがすでに存在する場合、再作成します。この句は、すでに付与されているオブジェクト権限を変更（削除、再作成または再付与）せずに、既存のオブジェクトの定義を変更するために使用します。 Java スキーマ・オブジェクトを再定義して、RESOLVE または COMPILE を指定すると、Oracle Lite は、これらのパラメータを認識はしますが、無視します。 再定義される関数に対する権限を前もって付与されているユーザーは、引き続きその関数にアクセスできます。ユーザーに再度権限を付与する必要はありません。
RESOLVE | COMPILE	Oracle Lite は、このパラメータを認識しますが、無視します。Oracle では、この文が成功したときに作成される Java スキーマ・オブジェクトをデータベースで解決するように指定します。 ■ クラスに適用されると、別のクラス・スキーマ・オブジェクトに対する参照名の解決が行われます。 ■ ソースに適用されると、ソースのコンパイルが行われます。 制限事項 **:** Java リソースにこの句は指定できません。
NOFORCE	Oracle Lite は、このパラメータを認識しますが、無視します。Oracle では、RESOLVE OR COMPILE を指定して解決またはコンパイルが失敗した場合、NO FORCE はこの CREATE コマンドの結果をロールバックします。このオプションを指定していないと、解決またはコンパイルが失敗した場合、Oracle は、何もアクションを実行しません（すなわち、作成されたスキーマ・オブジェクトが残ります）。
CLASS	Java クラス・ファイルをロードします。
RESOURCE	Java リソース・ファイルをロードします。
SOURCE	Java ソース・ファイルをロードします。AS *source_text* 句を使用する必要があります。

引数	説明
NAMED	Oracle Lite は、このパラメータを認識しますが、無視します。Oracle では、これは、Java ソースまたはリソースに「必須」です。 ■ Java ソースの場合、この句は、ソース・コードが保持されるスキーマ・オブジェクトの名前を指定します。CREATE JAVA SOURCE 文が成功した場合、ソースで定義された各 Java クラスを保持するためのスキーマ・オブジェクトも作成されます。 ■ Java リソースの場合、この句は、Java リソースを保持するスキーマ・オブジェクトの名前を指定します。 schema を指定しないと、Oracle はユーザー自身のスキーマ内にオブジェクトを作成します。 制限事項 : ■ Java クラスに NAMED は指定できません。 ■ *primary_name* は、データベース・リンクを含むことはできません。
SCHEMA *schema*	Oracle Lite は、このパラメータを認識しますが、無視します。Oracle では、Java クラスにのみ適用されます。オプション指定のこの句は、Java ファイルを含むオブジェクトが常駐するスキーマを指定します。SCHEMA を指定せず、また NAMED（上記）を指定しないと、Oracle は、ユーザー自身のスキーマ内にオブジェクトを作成します。
invoker_rights_clause	Oracle との互換性のために、Oracle Lite は、*invoker_rights_clause* を認識しますが、施行はしません。
RESOLVER	Oracle Lite は、このパラメータを認識しますが、無視します。Oracle では、完全修飾 Java 名の Java スキーマ・オブジェクトへのマッピングを指定します。 ■ *match_string* は、完全修飾 Java 名、そのような Java 名に一致するワイルド・カード、または任意の名前に一致するワイルド・カードのいずれかです。 ■ *schema_name* は、対応する Java スキーマ・オブジェクトを検索するためのスキーマを示します。 ■ ダッシュ（-）は、*schema_name* の代替として、*match_string* が有効な Java 名に一致したとき、Oracle がスキーマを未解決のままにできることを示します。解決は成功しますが、クラスは、名前をランタイムに使用できません。 このマッピングは、後の解決（暗黙的、または、ALTER...RESOLVE 文による明示的解決）で使用するために、作成されたスキーマ・オブジェクトの定義とともにこのコマンド内に格納されます。

引数	説明
AS *source_text*	Java ソース・プログラムのテキスト。
USING BFILE	クラス・ファイルの形式を識別します。BFILE は、CREATE JAVA CLASS または CREATE JAVA RESOURCE によって、バイナリ・ファイルと解釈されます。

使用上の注意

Oracle Lite は、Java クラスをデータベースにロードするとき、依存クラスはロードしません。一般に、Java クラスをデータベースにロードするには、`loadjava` ユーティリティを使用してください。`loadjava` ユーティリティの詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

Java クラスの例

次の文は、スキーマ・オブジェクトを作成し、指定された Java クラスを新たに作成されたスキーマ・オブジェクトにロードします。

```
CREATE JAVA CLASS USING BFILE (bfile_dir, 'Agent.class');
```

この例は、ディレクトリ・パス `bfile_dir` を想定しています。これは、既存の Java クラス `Agent.class` を含むオペレーティング・システムのディレクトリを指します。この例では、クラスの名前が Java クラス・スキーマ・オブジェクトの名前を決定します。

Java ソースの例

次の文は、Java ソース・スキーマ・オブジェクトを作成します。

```
CREATE OR REPLACE JAVA SOURCE AS
    /* This is a class Test */
    import java.math.*; /* */
    public class Test {
        public static BigDecimal myfunc(BigDecimal a, BigDecimal b)
        { return a.add(b); }
        public static Strin myfunc2(String a, String b)
        { return (a+b); }
    };
```

注意： キーワード「public class」を、最初の public class 文の前のコメントに使用しないでください。

Java リソースの例

次の文は、バイナリ・ファイルから、APPTEXT という名前の Java リソース・スキーマ・オブジェクトを作成します。

```
CREATE JAVA RESOURCE NAMED "appText"
   USING BFILE ('C:¥TEMP', 'textBundle.dat');
```

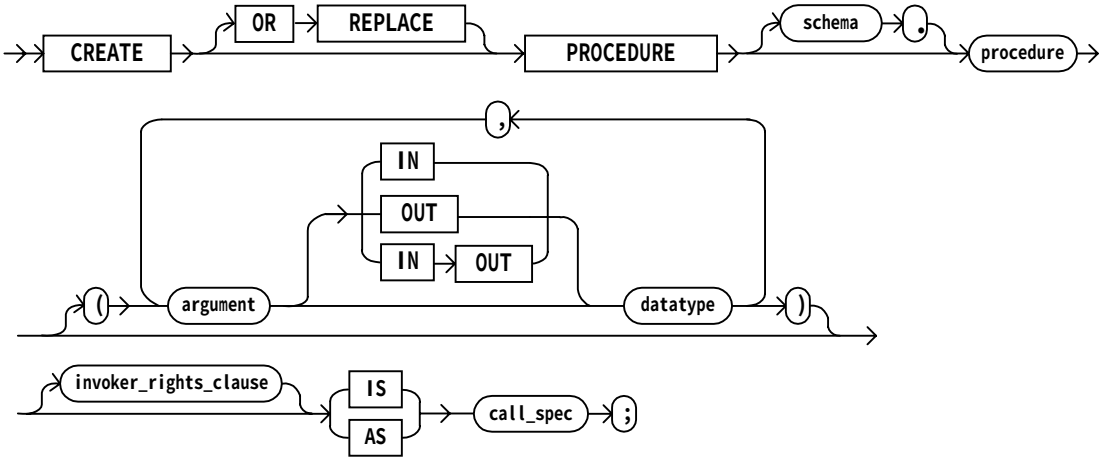
注意： Java 文を埋め込む場合、SQL*Plus 文の文末にセミコロン文字「;」を使用しないでください。セミコロンを行末に置く必要がある場合、「/**/」を使用して空白のコメント行を追加する必要があります。通常のコメント記号「//」は、この状況では機能しません。行末に /**/ を置くと、SQLPlus はセミコロンを SQL 文の終わりと解釈しません。

関連項目

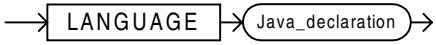
DROP JAVA

CREATE PROCEDURE

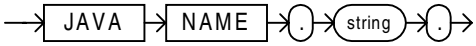
構文



call_spec::=



Java_declaration::=



用途

スタンドアロンのストアド・プロシージャのコール仕様を作成します。

コール仕様（「call spec」）は、SQL からコールできるように Java メソッドを宣言します。call spec は、コール時にどの Java メソッドをコールするかを Oracle Lite に知らせます。また、Oracle Lite に、引数および戻り値に対してどの型変換を行うかも知らせます。

ストアド・プロシージャによって、開発、整合性、セキュリティおよびメモリー割当ての領域で恩恵が得られます。ストアド・プロシージャのコール方法などストアド・プロシージャの詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

ストアド・プロシージャとストアド・ファンクションは似ています。ストアド・ファンクションは、コールされた環境内に値を返すのに対し、ストアド・プロシージャは返しません。関数に固有の情報は、「CREATE FUNCTION」を参照してください。

CREATE PROCEDURE 文は、スタンドアロンのスキーマ・オブジェクトとしてプロシージャを作成します。スタンドアロンのプロシージャの削除については、「DROP PROCEDURE」を参照してください。

前提条件

ユーザー自身のスキーマ内でプロシージャを作成するには、データベースに SYSTEM として接続するか、DBA/DDL 権限が必要です。

引数	説明
OR REPLACE	プロシージャがすでに存在する場合、再作成します。この句は、すでに付与されているオブジェクト権限を変更（削除、再作成または再付与）せずに、既存のプロシージャの定義を変更するために使用します。 このパッケージに依存するファンクション索引があれば、Oracle Lite は、索引に DISABLED のマークを付けます。
schema	プロシージャを含むスキーマ。schema を指定しないと、Oracle Lite はユーザーのカレント・スキーマにプロシージャを作成します。
procedure	作成するプロシージャの名前。
argument	プロシージャの引数の名前。プロシージャが引数を受け入れない場合、プロシージャ名の後のカッコを省略できます。
IN	プロシージャをコールするときに、引数の値を指定する必要があることを指定します。
OUT	実行後、プロシージャがこの引数の値をコール側の環境に渡すことを指定します。
IN OUT	プロシージャをコールするときに引数の値を指定する必要があること、および実行後、プロシージャがこの引数の値をコール側の環境に渡すことを指定します。 IN、OUT および IN OUT を省略した場合、引数のデフォルトは、IN です。 このパラメータまたは別のパラメータに行われた変更は、同じ変数が両方に渡された場合、両方の名前を通して即座に参照できます。 プロシージャが未処理例外によって終了した場合、このパラメータに対して行われた割当ては、コール側の変数内で参照できます。 このような効果は、特定のコールで発生することもあれば、発生しないこともあります。これらの影響が問題ではない場合にのみ、NOCOPY を使用します。
datatype	引数のデータ型。引数は、Oracle Lite SQL がサポートするどのデータ型でも取ることができます。 データ型で長さ、精度またはスケールは指定できません。たとえば、VARCHAR2(10) は無効で、VARCHAR2 は有効です。Oracle Lite は、引数の長さ、精度またはスケールを、プロシージャのコール側の環境から導出します。
invoker_rights_clause	Oracle との互換性のために、Oracle Lite は *invoker_rights_clause* を認識しますが、施行はしません。

引数	説明
IS	SQL 識別子を Java メソッドに対応付けます。
AS	SQL 識別子を Java メソッドに対応付けます。
call_spec	Java メソッド名、パラメータ型および戻り型を、対応する SQL 要素にマップします。
LANGUAGE	*call_spec* の言語を指定します。Oracle では、C または Java が使用できます。Oracle Lite では、Java のみが使用できます。
Java_declaration	Java クラス内のメソッド名を識別します。
JAVA NAME	Java メソッド名。
string	メソッドの Java 実装を識別します。詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

使用上の注意

Oracle Lite は、<*invoker_rights_clause*> を認識しますが、施行はしません。Oracle Lite は常に、`AUTHID` に `current_user` を使用します。

例

次の例は、Java プロシージャを作成してコンパイルし、それを Oracle Lite データベースに対してテストします。

1. 次の Java プログラムを作成してコンパイルし、その名前を **EMPTrigg.java** にします。

```
import java.sql.*;

public class EMPTrigg {
   public static final String goodGuy = "Oleg";

   public static void NameUpdate(String oldName, String[] newName) {
      if (oldName.equals(goodGuy))
         newName[0] = oldName;
   }

   public static void SalaryUpdate(String name, int oldSalary,
                              int newSalary[])
   {
      if (name.equals(goodGuy))
         newSalary[0] = Math.max(oldSalary, newSalary[0])*10;
   }

   public static void AfterDelete(Connection conn, String name,
               int salary) {
```

```
        if (name.equals(goodGuy))
           try {
              Statement stmt = conn.createStatement();
              stmt.executeUpdate(
                 "insert into employee values('" + name + "', " +
                                        salary + ")");
              stmt.close();
           } catch(SQLException e) {}
     }
  }
```

2. NAME と SALARY 列を持った EMPLOYEE 表を作成します。

```
CREATE TABLE EMPLOYEE (NAME VARCHAR(32), SALARY INT);
```

3. 次の文を入力して、EMPLOYEE 表に値を挿入します。

```
INSERT INTO EMPLOYEE VALUES ('Alice', 100);

INSERT INTO EMPLOYEE VALUES ('Bob', 100);

INSERT INTO EMPLOYEE VALUES ('Oleg', 100);
```

4. `EMPTrigg` クラスを Oracle Lite データベースにロードします。一度ロードすると、`EMPTrigg` クラス・メソッドは、Oracle Lite データベース内のストアド・プロシージャになります。

```
CREATE JAVA CLASS USING BFILE ('c:¥', 'EMPTrigg.class');
```

5. CREATE PROCEDURE 文を使用して SQL が `EMPTrigg` クラス内のメソッドをコールできるようにします。

```
CREATE PROCEDURE name_update(
old_name in varchar2, new_name in out varchar2)
is language java name
'EMPTrigg.NameUpdate (java.lang.String, java.lang.String[])';
/

 CREATE PROCEDURE salary_update(
 ename varchar2, old_salary int, new_salary in out int)
 as language java name
 'EMPTrigg.SalaryUpdate (java.lang.String, int, int[])';
 /
```

```
CREATE PROCEDURE after_delete(
ename varchar2, salary int)
as language java name
'EMPTrigg.AfterDelete (java.sql.Connection, java.lang.String, int)';
/
```

6. 各ストアド・プロシージャのトリガーを作成します。

```
CREATE TRIGGER NU BEFORE UPDATE OF NAME ON EMPLOYEE FOR EACH ROW
name_update (old.name, new.name);
/

CREATE TRIGGER SU BEFORE UPDATE OF SALARY ON EMPLOYEE FOR EACH ROW
salary_update (name, old.salary, new.salary);
/

CREATE TRIGGER AD AFTER DELETE ON EMPLOYEE FOR EACH ROW
after_delete (salary, salary);
/
```

7. EMPLOYEE 表からすべての列を選択します。

```
SELECT * FROM EMPLOYEE;
```

次の結果を返します。

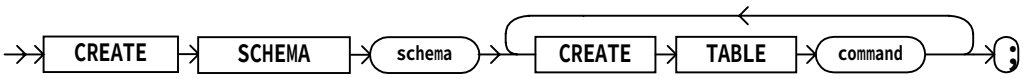
```
NAME                                SALARY
------------------------------- ---------
Alice                                  100
Bob                                    100
Oleg                                   100
```

関連項目

DROP PROCEDURE

CREATE SCHEMA

構文



用途

スキーマまたは表、索引およびビューの所有者を作成します。CREATE SCHEMA を使用して、シングル・トランザクション内で複数の表およびビューを作成することもできます。

前提条件

CREATE SCHEMA 文は、CREATE TABLE、CREATE VIEW および GRANT 文を含むことができます。CREATE SCHEMA 文を発行するには、データベースに SYSTEM として、あるいは DDL または DMA/ADMIN 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	スキーマの名前で、最大 128 文字の文字列。各ユーザー名は、同じ名前のデフォルトのスキーマを持つため、スキーマ名は、どのユーザー名とも異なる必要があります。ユーザー名と同じ名前のスキーマを作成すると、Oracle Lite は、エラーを返します。詳細は、「CREATE USER」を参照してください。
CREATE TABLE	CREATE SCHEMA 文の一部として発行される CREATE TABLE 文。
command	CREATE TABLE または CREATE VIEW コマンドの引数およびキーワードがすべて含まれています。

使用上の注意

- Oracle Lite は、スキーマをユーザーのプライベート・データベースとして扱います。スキーマは非公式に、別の名前空間および所有者有効範囲を定義します。つまり、常駐するスキーマが異なれば、同じ名前の表が 2 つあってもよいということです。同一のスキーマ内の表およびビューは、すべてそのスキーマの所有者に所有されています。現在使用中のスキーマ以外のスキーマを使用するには、現在使用中のスキーマから切断してから新規スキーマに接続してください。
- CREATE SCHEMA は、独立した文のグループを 1 つの文として扱います。グループ内の 1 つの文が失敗すると、すべての文の実行内容が元に戻されます。
- POL__SCHEMATA ビューに表示される新しいスキーマの名前。

例 1

HOTEL_OPERATION と呼ばれるサンプル・スキーマを作成するには、次を使用します。

```
CREATE SCHEMA HOTEL_OPERATION;
```

例 2

HOTEL_OPERATION スキーマを表 HOTEL_DIR およびビュー LARGE_HOTEL と作成するには、次を使用します。

```
CREATE SCHEMA HOTEL_OPERATION
CREATE TABLE HOTEL_DIR(
HOTELNAME CHAR(40) NOT NULL,
RATING INTEGER,
ROOMRATE FLOAT,
LOCATION CHAR(20) NOT NULL,
CAPACITY INTEGER);
```

ODBC 2.0

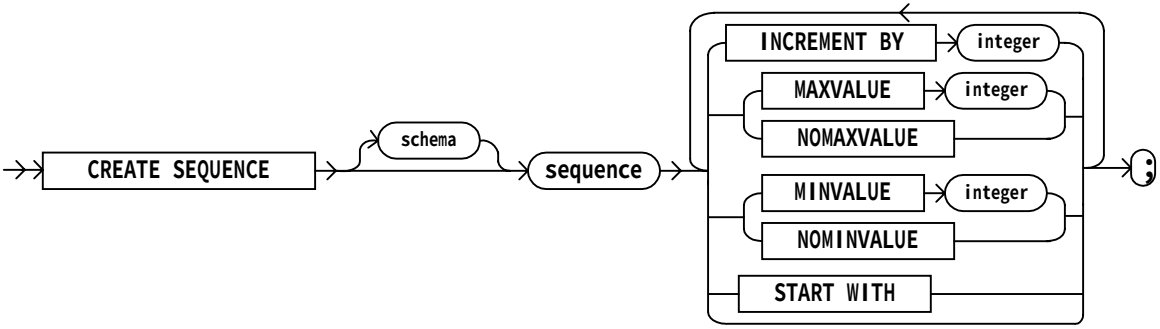
CREATE SCHEMA コマンドは ODBC SQL 構文の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

関連項目

GRANT、CREATE TABLE、CREATE VIEW

CREATE SEQUENCE

構文



用途

順序を作成します。

前提条件

なし。

引数	説明
schema	順序を含むスキーマの名前。schema を省略すると、Oracle Lite ではユーザー自身のスキーマ内で順序が作成されます。
sequence	作成される順序の名前。
INCREMENT BY	順序番号間の間隔を指定します。0 以外の任意の正の整数または負の整数にできます。負の整数の場合、順序は降順になります。正の整数を指定すると、順序は昇順になります。INCREMENT BY 句を指定しない場合、デフォルトは 1 です。
START WITH	生成される最初の順序数値を指定します。このオプションは、昇順を最小値（デフォルト）より大きい値で開始する場合、または降順を最大値（デフォルト）より小さい値で開始するときに使用します。
MAXVALUE	順序で生成できる最大値を指定します。この整数値は 9 桁以下になります。MAXVALUE は MINVALUE より大きくする必要があります。
NOMAXVALUE	昇順に対しては 2147483647、降順に対しては -1 の最大値を指定します。
MINVALUE	順序で生成できる最小値を指定します。この整数値は 9 桁以下になります。MINVALUE は、MAXVALUE よりも小さくする必要があります。
NOMINVALUE	昇順に対しては 1、降順に対しては -2147483647 の最小値を指定します。

使用上の注意

Oracle Lite は、NEXTVAL 関数がアクセスされたときに、順序番号をコミットします。ただし、Oracle と異なり、Oracle Lite は、順序を自動的にはコミットしません。したがって、Oracle Lite では、順序をロールバックできます。ROLLBACK コマンドを使用しているとき順序をメンテナンスするには、順序を作成した後、それをコミットする必要があります。

例

次の文は、順序 ESEQ を作成します。

```
CREATE SEQUENCE ESEQ INCREMENT BY 10;
```

最初の ESEQ.NEXTVAL への参照では、1 が返されます。2 回目では、11 が返されます。その後の参照では、それぞれ、前の戻り値より 10 大きい値が返されます。

ODBC 2.0

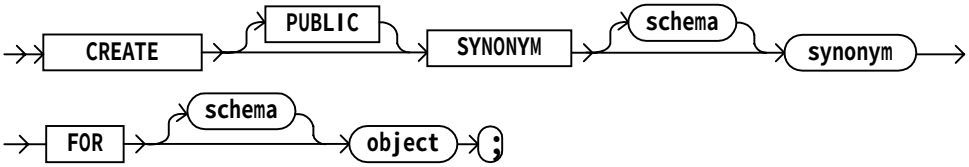
CREATE SEQUENCE コマンドは ODBC SQL 構文の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

関連項目

ALTER SEQUENCE、DROP SEQUENCE

CREATE SYNONYM

構文



用途

パブリックまたはプライベート SQL シノニムを作成します。

前提条件

なし。

引数	説明
PUBLIC	パブリック・シノニムを作成します。パブリック・シノニムは、すべてのユーザーでアクセスできます。このオプションを省略すると、シノニムはプライベートとなり、そのスキーマ内でしかアクセスできなくなります。
schema	シノニムを含むスキーマ。*schema* を省略すると、Oracle Lite はユーザー自身のスキーマ内で索引を作成します。PUBLIC を指定している場合、schema は指定できません。

引数	説明
synonym	作成されるシノニムの名前。
FOR *object*	シノニムが作成されるオブジェクトを指定します。schema のあるオブジェクトを修飾しない場合、Oracle Lite ではユーザー自身のスキーマにオブジェクトが含まれていると解釈されます。オブジェクトは表、ビュー、順序または別のシノニムです。オブジェクトは、現時点で存在している必要はなく、またオブジェクトへのアクセス権限も必要ありません。

使用上の注意

プライベート・シノニム名は、そのスキーマ内の他のすべてのオブジェクトから区別できる必要があります。

シノニムは、INSERT、SELECT、UPDATE および DELETE 文でのみ使用できます。DROP ではシノニムを使用できません。

例

スキーマ SCOTT 内の PRODUCT 表のシノニム PROD を定義するには、次の文を発行します。

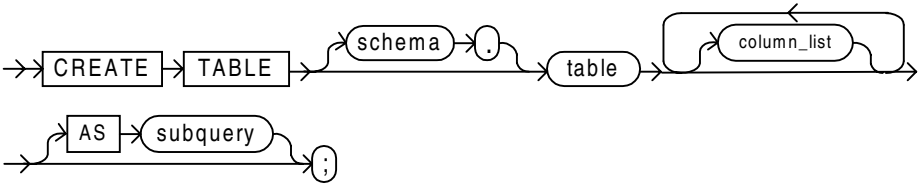
```
CREATE SYNONYM PROD FOR SCOTT.PRODUCT;
```

関連項目

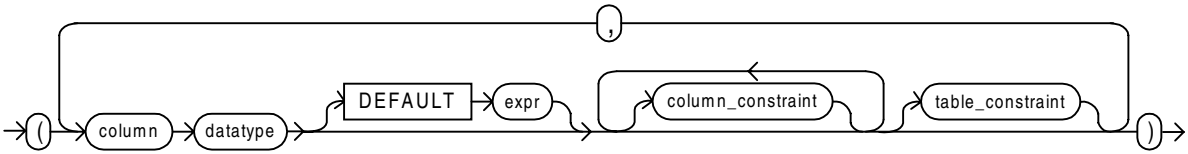
CREATE TABLE、CREATE VIEW、CREATE SEQUENCE、DROP SYNONYM

CREATE TABLE

構文



column_list::=



用途

データベース表を作成します。

また、指定された副問合せの結果に基づいてデータベースを作成し移入します。列のデータ型は、副問合せの結果集合から導出されます。詳細は、「使用上の注意」を参照してください。

前提条件

ユーザーのスキーマまたは別のスキーマ内に表を作成するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	所有者であるユーザーと同じ名前のスキーマ。省略すると、デフォルトのスキーマ名が使用されます。
table	データベースの表の名前。表名にはピリオド「.」を含めることはできません。また、アンダースコア「_」の付いた文字で始めることもできません。
column	表の列の名前。
datatype	column のデータ型。副問合せでは使用できません。
DEFAULT	新しい列に対してデフォルト値 *expr*（式）を指定するか、既存の列に対して新規デフォルト *expr* を指定します。以下のいずれかになります。 ■ DEFAULT NULL ■ DEFAULT USER（表が作成された際のユーザー名） ■ DEFAULT リテラル 式の詳細は、「式の指定」を参照してください。
column_constraint	列整合性制約を追加します。詳細は、「CONSTRAINT 句」を参照してください。

引数	説明
table_constraint	表整合性制約を追加します。詳細は、「CONSTRAINT 句」を参照してください。
AS *subquery*	SELECT 文。

使用上の注意

CREATE ANY TABLE を使用して別のスキーマに表を作成できますが、DBA/DDL ロールが必要です。

各表は、最大 1000 列を持つことができ、複数の主キー制約を持つことはできません。

次の構文は、Oracle Lite ではサポートされていません。

```
CREATE TABLE <newtablename> AS SELECT * FROM <oldtablename>;
```

適切な DDL を使用して表を作成する必要があります。その後、次の文を使用して、既存の表を新しい表にコピーできます。

```
INSERT INTO <newtablename> SELECT * FROM <oldtablename>;
commit;
```

column_list が省略された場合は、次のようになります。

- 副問合せを指定したときに表の列が定義されていない場合、列名は *subquery* により選択された式から導出されます。
- 選択リスト内の式に別名が含まれている場合は、列名として別名が使用されます。
- 式が別名を持たない列の場合、その名前が列名として使用されます。列ではなく、別名を持たない式は無効です。表の列のデータ型は、副問合せの選択リスト内の対応する式のデータ型と同じです。
- 副問合せが UNION または MINUS を含む場合、最初の選択文がこの目的に使用されます。

column_list が含まれている場合は、次のようになります。

- *column_list* 内の列の数は、副問合せの式の数と等しくする必要があります。
- 列定義は、列名、デフォルト値および整合性制約のみを指定できますが、データ型は指定できません。
- 参照整合性制約は、CREATE TABLE 文のこの構成では定義できません。かわりに、後で ALTER TABLE 文を使用して参照整合性制約を作成できます。

subquery で ORDER BY 句が使用された場合、データはその順序で表に挿入されます。これは通常、列による順序に従ったデータのクラスタとなりますが、保証はされません。

例 1

次の文は、HOTEL_NAME と CAPACITY の 2 つの列がある表 HOTEL_DIR を作成します。HOTEL_NAME は主キーで、CAPACITY は NULL にはできず、デフォルト値が 0 です。

```
CREATE TABLE HOTEL_DIR (HOTEL_NAME CHAR(40) PRIMARY KEY,
CAPACITY INTEGER DEFAULT 0 NOT NULL)
```

例 2

次の文は、表 HOTEL_RESTAURANT を作成します。

```
CREATE TABLE HOTEL_RESTAURANT(REST_NAME CHAR(50) UNIQUE,
HOTEL_NAME CHAR(40) REFERENCES HOTEL_DIR,
RATING FLOAT DEFAULT NULL)
```

列には、次のものが含まれます。

- REST_NAME はレストランの名前です。
- HOTEL_NAME はレストランのあるホテルの名前です。
- RATING はレストランの評価で、そのデフォルト値は NULL です。

表には、次の整合性制約があります。

- 同じ名前のホテル・レストランが 2 つ以上あってはならない。
- HOTEL_NAME は、HOTEL_DIR 表内のホテルを参照する必要がある。

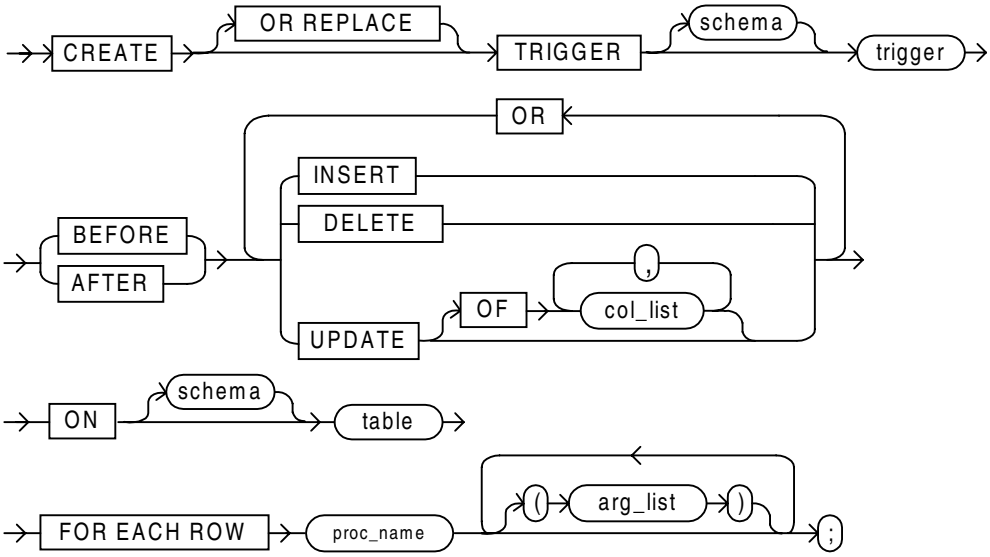
注意： この他に、例としてサンプル・スクリプトの **poldemo.sql** を参照できます。これは、*Oracle_Home*¥DBS ディレクトリにあります。このスクリプトは、**polite.odb** デモ・データベースを作成します。

関連項目

CONSTRAINT 句、DROP TABLE、ALTER TABLE、SELECT

CREATE TRIGGER

構文



用途

データベース・トリガーを作成し、使用可能にします。

前提条件

なし。

引数	説明
OR REPLACE	トリガーがすでに存在する場合、再作成します。存在しない場合、トリガーを作成します。すでに付与されているオブジェクト権限を変更（削除、再作成または再付与）せずに、既存のトリガーの定義を変更するために使用します。
schema	トリガーを含むスキーマ。スキーマを指定しない場合、Oracle Lite はユーザーのスキーマにトリガーを作成します。
table	データベース内の表の名前。
trigger	作成されるトリガーの名前。
BEFORE	トリガーを実行する文を実行する前に、トリガーが起動されるよう指定します。行トリガーでは、関連のある行が変更される前に別々に起動されます。
AFTER	トリガーを実行する文を実行した後で、トリガーが起動されるよう指定します。行トリガーでは、関連のある行が変更された後に別々に起動されます。
DELETE	DELETE 文によって表から行が削除されるたびに、トリガーが起動されるよう指定します。
INSERT	INSERT 文によって表に行が追加されるたびに、トリガーが起動されるよう指定します。
UPDATE OF	UPDATE 文によって OF 句に指定された列の 1 つにある値が変更されるたびに、トリガーが起動されるよう指定します。OF 句を省略すると、Oracle Lite は、UPDATE 文によって表の列の値が変更されるたびに、トリガーを起動します。
col_list	更新された際にトリガーが起動される原因となる列。
ON	トリガーが作成される表のスキーマおよび名前を指定します。省略した場合、Oracle Lite では表がユーザーのスキーマ内にあると解釈されます。
FOR EACH ROW	トリガーを行トリガーに指定します。Oracle Lite は、トリガー文によって影響を受ける各行に 1 回、行トリガーを起動します。この句を省略すると、トリガーは文トリガーです。Oracle Lite では、オプションのトリガー制約が一致すれば、トリガー文が発行されるときに、1 回のみ文トリガーを起動します。
proc_name	Oracle Lite がトリガーを起動するために実行する Java メソッド。
arg_list	Java メソッドに渡される引数。

例

次の例は、トリガーの作成とテストについての手順を示しています。

1. 次のプログラムを作成し、それを TriggerExample.java と命名します。

```
import java.lang.*;
    import java.sql.*;
    class TriggerExample {
        public void EMP_SAL(Connection conn, int new_sal)
        {
            System.out.println("new salary is :"+new_sal);
        }
    }
```

2. TriggerExample.java を EMP 表に連結します。

```
ALTER TABLE EMP ATTACH JAVA SOURCE "TriggerExample" in '.';
```

3. Java トリガーを作成します。

```
CREATE TRIGGER SAL_CHECK BEFORE UPDATE OF SAL ON EMP FOR EACH ROW
EMP_SAL(NEW.SAL);
 .
 /
```

4. Java トリガーを使用して、EMP 表を更新します。

```
update emp set sal=sal+5000 where sal=70000;
```

次の結果を返します。

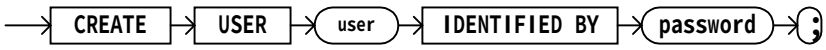
```
new salary is:75000

1 row updated
```

関連項目

ALTER TRIGGER、ALTER VIEW、CREATE VIEW、DROP TRIGGER

CREATE USER

構文



用途

権限を持たないデータベース・ユーザーを作成します。

前提条件

ユーザーのスキーマまたは別のスキーマ内にユーザーを作成するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
user	作成されるユーザー。ここで、*user* は、他と重複しないユーザー名で、1 文字から最大 128 文字の文字列です。
IDENTIFIED BY	Oracle Lite がユーザー・アクセスを許可する方法を示します。
password	最大 128 文字の名前の、ユーザーの新しいパスワードを指定します。パスワードは、引用符には入れられず、大 / 小文字区別はありません。

使用上の注意

Oracle Lite では、CREATE USER コマンドを使用してマルチ・ユーザーを作成できます。ユーザーはスキーマではありません。ユーザーを作成するとき、Oracle Lite は、同じ名前のスキーマを作成し、それを自動的にそのユーザーにデフォルトのスキーマとして割り当てます。新規ユーザーの名前は、ALL_USERS ビューに表示されます。新しいユーザーのデフォルトのスキーマは、POL__SCHEMATA ビューに表示されます。

Oracle Lite データベースにユーザーとして接続すると、ユーザー名がそのセッションのデフォルトのスキーマになります。ユーザー名と一致するスキーマがない場合、Oracle Lite は接続を拒否します。デフォルトのスキーマ内のデータベース・オブジェクトは、スキーマ名を接頭辞として付けずにアクセスできます。

適切な特権を持つユーザーは、CREATE SCHEMA コマンドを使用して、別のスキーマを作成できますが、デフォルトのスキーマのみがデータベースに接続できます。スキーマは、それを作成したユーザーが所有し、そのオブジェクトにアクセスするには、スキーマ名の接頭辞が必要です。

CREATEDB ユーティリティまたは CREATE DATABASE コマンドを使用してデータベースを作成するとき、Oracle Lite は、SYSTEM と呼ばれる特別のユーザーを作成します。このユーザーは、すべてのデータベース権限を持ちますが、パスワードは割り当てられていません。必要であれば、SYSTEM にパスワードを割り当てられます。必要になって自分のユーザー名を確立するまで、SYSTEM をデフォルトのユーザー名として使用できます。

Oracle Lite では、SYSTEM 以外のユーザーが、自分のものでないスキーマ内でデータをアクセスしたり、操作を実行したりできません。ユーザーは、次の条件の内の 1 つが満たされた

場合にのみ、別のユーザーのスキーマ内でデータをアクセスしたり、操作を実行したりできます。

- ユーザーは別のユーザーのスキーマ内で前もって定義されたロールの権限が付与されています。これによって、ユーザーが操作を実行することが許可されます。
- ユーザーは、別のユーザーのスキーマ内で特定の権限を付与されています。

注意： 新しいユーザーがデータベース・オブジェクトを作成できるようにするには、SYSTEM が新しいユーザーに DBA/DDL 権限または RESOURCE 権限を付与する必要があります。できるだけ、DDL のかわりに DBA ロールを使用することをお薦めします。

例

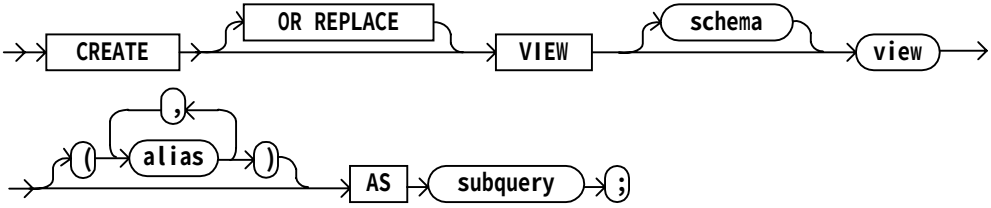
```
CREATE USER SCOTT IDENTIFIED BY TIGER;
```

関連項目

ALTER USER、GRANT

CREATE VIEW

構文



用途

ビューを作成または置換します。

前提条件

データベースには SYSTEM、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
OR REPLACE	ビューがすでに存在する場合、再作成します。すでに付与されているオブジェクト権限を変更（削除、再作成または再付与）せずに、既存のビューの定義を変更するために使用します。
schema	ビューを含むスキーマ。*schema* を省略すると、Oracle Lite はユーザー自身のスキーマ内でビューを作成します。
view	ビューの名前。
alias	ビューの問合せに選択された式の名前を指定します。別名の数は、ビューで選択された式の数と一致する必要があります。別名は、スキーマ・オブジェクトに名前を付ける際の Oracle Lite のルールに従う必要があります。それぞれの *alias* はビューの中で他と重複しないようにする必要があります。
AS *subquery*	ビューが基盤とする表の列および行を識別します。ビューの問合せは、ORDER BY または FOR UPDATE 句のない SELECT 文はどれでも可能です。選択リストには、最大 254 個の式を入れられます。

使用上の注意

以下の場合、ビューは更新可能です。

- subquery は、単一のベース表から、または別の更新可能なビューから選択します。
- 選択された式のそれぞれが、そのベース表または更新可能なビューの列への参照となります。
- 選択リスト内の 2 つ以上の列参照が同じ列を参照することはありません。

CREATE ANY VIEW を使用して別のスキーマにビューを作成できますが、DBA/DDL ロールが必要です。

例

次の例は、EMP 表内の各行の名前、仕事および給与を表示する EMP_SAL という名前のビューを作成します。

```
CREATE VIEW EMP_SAL (Name, Job, Salary) AS SELECT ENAME, JOB, SAL FROM EMP;

SELECT * FROM EMP_SAL;
```

次の結果を返します。

```
NAME       JOB          SALARY
---------- --------- ---------
KING       PRESIDENT      5000
BLAKE      MANAGER        2850
CLARK      MANAGER        2450
JONES      MANAGER        2975
MARTIN     SALESMAN       1250
ALLEN      SALESMAN       1600
TURNER     SALESMAN       1500
JAMES      CLERK           950
WARD       SALESMAN       1250
FORD       ANALYST        3000
SMITH      CLERK           800
SCOTT      ANALYST        3000
ADAMS      CLERK          1100
MILLER     CLERK          1300

14 rows selected.
```

ODBC 2.0

CREATE VIEW の ODBC SQL 構文は OR REPLACE 引数をサポートしませんが、ODBC はこのコマンドをそのままデータベースに渡します。

ビューでのデータ編集

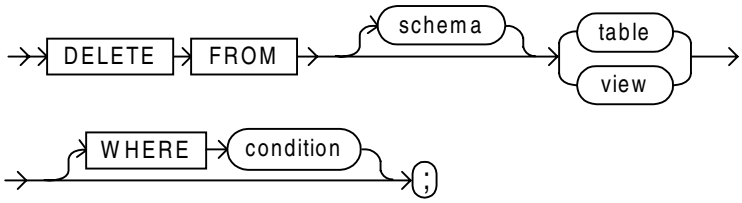
ODBC ベースのツールは、そのほとんどがビューの更新を許可する前に主キーを必要とします。Oracle Lite は、ビューの主キーを記録しないため、WHERE 句を使用して 1 つまたは複数のターゲット行を指定し、ビューの更新や削除を実行する SQL コマンドを発行する必要があります。

関連項目

DROP SEQUENCE、CREATE TABLE、DROP VIEW

DELETE

構文



用途

表またはビューのベース表から行を削除します。

前提条件

ユーザーのスキーマ内の表またはビューからのみ行を削除できます。

引数	説明
schema	表またはビューを含むスキーマ。*schema* を省略すると、Oracle Lite ではユーザー自身のスキーマ内に表またはビューがあると解釈されます。
table	行を削除する表の名前。
view	ビューの名前。*view* を指定すると、Oracle Lite ではビューのベースから行が削除されます。
WHERE *condition*	condition 引数に指定された条件を満たす行しか削除されません。有効な条件の作成の詳細は、「SQL 条件の指定」を参照してください。

使用上の注意

WHERE 句が指定されていない場合、表の行がすべて削除されます。

位置づけ DELETE では、カーソルを更新可能にしてください。

例

```
DELETE FROM PRICE WHERE MINPRICE < 2.4;
```

ODBC 2.0

DELETE の ODBC SQL 構文は、SQL と同じです。さらに、ODBC 構文は CURRENT OF *cursor_name* キーワードと引数を含みます。これらは、次に示すように、DELETE 操作が実行されるカーソル位置を指定するための WHERE 句内で使用されます。

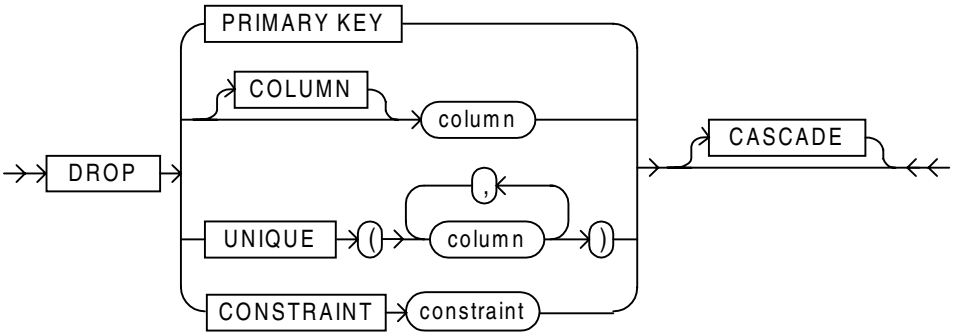
```
WHERE CURRENT OF cursor_name
```

関連項目

UPDATE

DROP 句

構文



用途

整合性制約がデータベースから削除されます。

前提条件

DROP 句は、ALTER TABLE 文内にのみ現れます。整合性制約を削除するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
PRIMARY KEY	表の PRIMARY KEY 制約を削除します。
UNIQUE	指定の列から UNIQUE 制約を削除します。
COLUMN	表から列を削除します。

引数	説明
column	列制約が削除される列、または DROP COLUMN の場合は、表から削除される列を指定します。
CONSTRAINT	CONSTRAINT という名前の整合性制約を削除します。詳細は、「CONSTRAINT 句」を参照してください。
constraint	削除される整合性制約の名前。
RESTRICT	削除される制約に依存している整合性制約がある場合、DROP コマンドは失敗します。
CASCADE	CONSTRAINT 句内で指定された制約に依存するその他の整合性制約をすべて削除します。

例

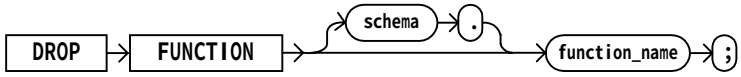
```
ALTER TABLE EMP DROP COLUMN COMM;
```

関連項目

ALTER TABLE、CONSTRAINT 句

DROP FUNCTION

構文



用途

スタンドアロンのストアド・ファンクションをデータベースから削除します。関数作成の詳細は、「CREATE FUNCTION」を参照してください。

前提条件

関数を削除するには、次の要件の 1 つを満たしている必要があります。

- 関数がユーザーのスキーマ内に必要です。
- データベースに SYSTEM として接続している必要があります。
- DBA/DDL 権限が必要です。

引数	説明
schema	関数を含むスキーマ。schema を省略すると、Oracle Lite は、関数がユーザー自身のスキーマ内にあると解釈します。
function_name	削除する関数の名前。 Oracle Lite は、削除された関数に依存している、またはその関数をコールしているローカル・オブジェクトをすべて無効にします。その後、これらのオブジェクトの 1 つを参照すると、Oracle Lite は、そのオブジェクトを再コンパイルしようとし、削除された関数が再作成されていないと、エラーを返します。

例

次の文は、PAY_SALARY 関数を削除します。この関数は、CREATE FUNCTION の例で作成したものです。PAY_SALARY 関数を削除すると、PAY_SALARY に依存するオブジェクトがすべて無効になります。

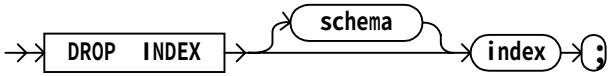
```
DROP FUNCTION PAY_SALARY;
```

関連項目

CREATE FUNCTION

DROP INDEX

構文



用途

索引がデータベースから削除されます。

前提条件

索引を削除するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	削除する索引の入ったスキーマ。schema を省略すると、Oracle Lite では索引がユーザー自身のスキーマ内にあると解釈されます。
index	削除する索引の名前。

例

次の例は、EMP 表の SAL 列の索引を削除します。

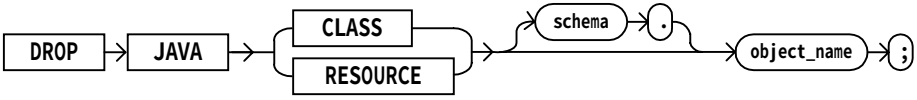
```
DROP INDEX SAL_INDEX;
```

関連項目

CREATE INDEX

DROP JAVA

構文



用途

Java クラスまたはリソース・スキーマ・オブジェクトを削除します。

Java クラスの解決およびリソースの詳細は、『Oracle Lite Java 開発者ガイド』を参照してください。

前提条件

クラスまたはリソース・スキーマ・オブジェクトを削除するには、次の要件に一致する必要があります。

- Java クラスまたはリソースがユーザーのスキーマ内に必要です。
- データベースに SYSTEM として接続するか、DBA/DDL 権限が必要です。

引数	説明
JAVA CLASS	Java クラス・スキーマ・オブジェクトを削除します。
JAVA RESOURCE	Java リソース・スキーマ・オブジェクトを削除します。
object_name	既存の Java クラス、ソースまたはリソースのスキーマ・オブジェクトの名前を指定します。

使用上の注意

Oracle Lite は、*schema_name* が指定されるとそれを認識しますが、それを施行はしません。

例

次の文は、Java クラス `MyClass` を削除します。

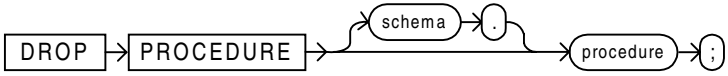
```
DROP JAVA CLASS "MyClass";
```

関連項目

CREATE JAVA

DROP PROCEDURE

構文



用途

スタンドアロンのストアド・プロシージャをデータベースから削除します。この文を、パッケージの一部に含まれるプロシージャの削除に使用しないでください。かわりに、DROP PACKAGE 文を使用してパッケージ全体を削除するか、そのプロシージャを除いて、OR REPLACE 句付きの CREATE PACKAGE 文を使用して、再定義します。

プロシージャ作成の詳細は、「CREATE PROCEDURE」を参照してください。

前提条件

プロシージャを、データベースにスキーマとして接続するか、ユーザーが DBA/DDL 権限を持っている必要があります。

引数	説明
schema	プロシージャを含むスキーマ。*schema* を省略すると、Oracle Lite ではユーザー自身のスキーマ内にプロシージャがあると解釈されます。
procedure	削除するプロシージャの名前。 プロシージャを削除するとき、Oracle Lite は、削除されるプロシージャに依存するローカル・オブジェクトをすべて無効にします。その後、これらのオブジェクトの 1 つを参照すると、Oracle Lite は、そのオブジェクトを再コンパイルしようとし、削除されたプロシージャが再作成されていないと、エラー・メッセージを返します。

例

次の文は、ユーザー KERNER が所有するプロシージャ TRANSFER を削除して、TRANSFER に依存するオブジェクトをすべて無効にします。

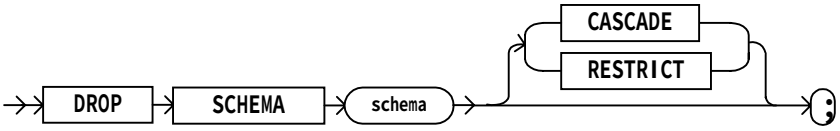
```
DROP PROCEDURE kerner.transfer
```

関連項目

CREATE PROCEDURE

DROP SCHEMA

構文



用途

スキーマがデータベースから削除されます。

前提条件

スキーマを削除するには、データベースに SYSTEM として、あるいは DBA/DDL または ADMIN 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	データベースから削除するスキーマ。
CASCADE	指定されたビューに依存する定義を持ったその他のオブジェクトがすべて、スキーマと共に自動的に削除されるよう指定します。
RESTRICT	指定されたスキーマに依存する定義を持ったその他のオブジェクトがある場合、DROP SCHEMA 操作が失敗するよう指定します。

使用上の注意

オプションを指定しない場合は、RESTRICT 引数によってデフォルト動作が決まります。

例

次の例は、CREATE SCHEMA の例で作成した HOTEL_OPERATION スキーマを削除します。

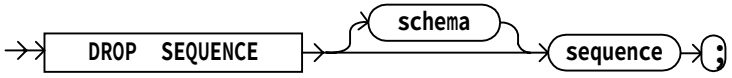
```
DROP SCHEMA HOTEL_OPERATION CASCADE;
```

関連項目

CREATE SCHEMA

DROP SEQUENCE

構文



用途

順序がデータベースから削除されます。

前提条件

データベースに SYSTEM としてログインするか、順序がユーザーのスキーマ内に必要です。

引数	説明
schema	削除する順序の入ったスキーマ。schema を省略すると、Oracle Lite は、順序がユーザー自身のスキーマ内にあると解釈します。
sequence	データベースから削除する順序の名前。

使用上の注意

順序を再起動する方法の 1 つに、削除して再作成する方法があります。たとえば、現在の設定値が 150 の順序があり、27 という値を使用して順序を再起動する場合、以下の手順を行います。

- 順序を削除します。
- 同一名および START WITH 値 27 で、作成します。

例

次の例は、CREATE SEQUENCE の例で作成した ESEQ 順序を削除します。

```
DROP SEQUENCE ESEQ;
```

ODBC 2.0

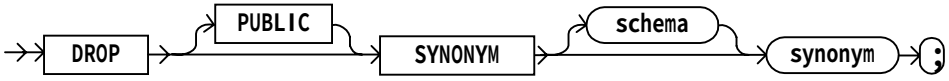
DROP SEQUENCE コマンドは ODBC SQL 構文の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

関連項目

ALTER SEQUENCE、CREATE SEQUENCE

DROP SYNONYM

構文



用途

パブリックまたはプライベート SQL シノニムを、データベースから削除します。

前提条件

データベースからシノニムを削除するには、データベースに SYSTEM としてログインするか、シノニムがユーザーのスキーマ内に必要です。

引数	説明
PUBLIC	パブリック・シノニムを指定します。パブリック・シノニムを削除するには、PUBLIC を指定してください。
schema	シノニムを含むスキーマ。schema を省略すると、Oracle Lite ではユーザー自身のスキーマ内のシノニムが削除されます。PUBLIC を指定している場合、schema は指定できません。
synonym	削除されるシノニムの名前。

例

次の例は、CREATE SYNONYM の例で作成した PROD という名前のシノニムを削除します。

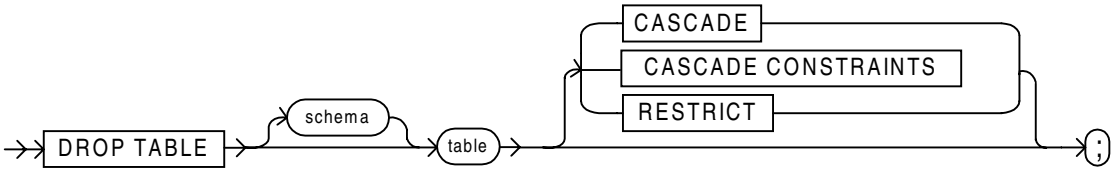
```
DROP SYNONYM PROD;
```

関連項目

CREATE SYNONYM

DROP TABLE

構文



用途

表がデータベースから削除されます。

前提条件

表をデータベースから削除するには、データベースに SYSTEM として、または DBA/DDL 権限を持つユーザーとしてログインする必要があります。

引数	説明
schema	削除する表の入ったスキーマ。schema を省略すると、Oracle Lite は、表がユーザー自身のスキーマ内にあると解釈します。
table	データベースから削除する表の名前。
CASCADE	表がビューのベース表の場合、または表内の主キーを参照する参照整合性制約がある場合、それらが表と共に自動的に削除されるよう指定します。
CASCADE CONSTRAINTS	表内の主キーを参照する参照整合性制約がすべて、表と共に自動的に削除されるよう指定します。
RESTRICT	表がビューのベース表の場合、または表が参照整合性制約で参照されている場合、DROP TABLE 操作が失敗するよう指定します。

使用上の注意

オプションの指定がなく、また、その表を参照する参照整合性制約がない場合、Oracle Lite は表を削除します。オプションの指定がなく、その表を参照する参照整合性制約がある場合、Oracle Lite はエラー・メッセージを返します。

例

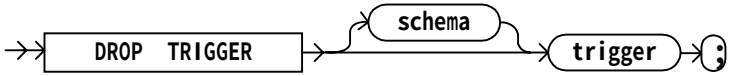
```
DROP TABLE EMP;
```

関連項目

ALTER TABLE、CREATE TABLE

DROP TRIGGER

構文



用途

データベース・トリガーをデータベースから削除します。

前提条件

データベースに SYSTEM としてログインするか、トリガーがユーザーのスキーマ内になければなりません。

引数	説明
schema	トリガーを含むスキーマ。schema を省略すると、Oracle Lite は、トリガーがユーザー自身のスキーマ内にあると解釈します。
trigger	トリガーの名前。

例

次の文は、SAL_CHECK トリガーを削除します。このトリガーは、CREATE TRIGGER の例で作成したものです。

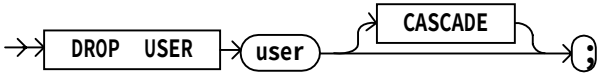
```
DROP TRIGGER ruth.reorder
```

関連項目

CREATE TRIGGER

DROP USER

構文



用途

ユーザーがデータベースから削除されます。

前提条件

ユーザーをデータベースから削除するには、データベースに SYSTEM としてログインするか、あるいは、DBA/DDL または ADMIN 権限が必要です。

引数	説明
user	削除するユーザーの名前。
CASCADE	ユーザーに対応付けられたオブジェクトをすべて削除します。

使用上の注意

データベースに SYSTEM として接続しているか、あるいは ADMIN または DBA/DDL ロールの権限を付与されている場合は、ユーザーを削除できます。

例

次の文は、ユーザー Michael を削除します。

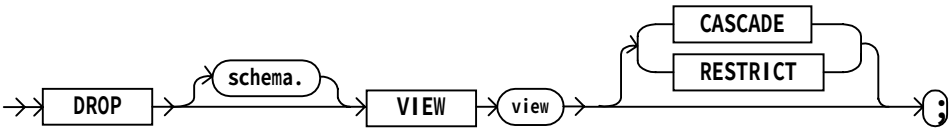
```
DROP USER MICHAEL;
```

関連項目

CREATE USER

DROP VIEW

構文



用途

ビューがデータベースから削除されます。

前提条件

ビューをデータベースから削除するには、データベースにログインしている必要があり、また、次の要件の 1 つを満たしている必要があります。

- データベースに SYSTEM としてログインしている必要があります。
- DBA/DDL 権限が必要です。
- ビューがユーザーのスキーマ内に必要です。

引数	説明
schema	削除するビューの入ったスキーマ。schema を省略すると、Oracle Lite は、ビューがユーザー自身のスキーマ内にあると解釈します。
view	データベースから削除するビューの名前。

引数	説明
CASCADE	定義が指定のビューに依存するその他のビューがすべて、ビューと共に自動的に削除されるよう指定します。
RESTRICT	定義が指定のビューに依存するその他のビューがある場合、DROP VIEW 操作が失敗するよう指定します。

使用上の注意

オプションが指定されていないと、Oracle Lite はこのビューのみを削除します。その他の依存ビューは影響を受けません。

例

次の例は、CREATE VIEW の例で作成した EMP_SAL ビューを削除します。

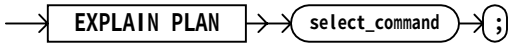
```
DROP VIEW EMP_SAL;
```

関連項目

CREATE SYNONYM、CREATE TABLE、CREATE VIEW

EXPLAIN PLAN

構文



用途

subquery::= 文の Oracle Lite データベース・オプティマイザによって選択された実行計画を表示します。

引数	説明
EXPLAIN PLAN	問合せの実行計画を決定します。
select_command	実行計画を決定する問合せ。

使用上の注意

Oracle Lite は、**execplan.txt** と呼ばれるファイルに実行計画を出力します。このファイルは、SQL*Plus と同じディレクトリにあります。Oracle Lite は、このファイルに新しい各実行計画を追加します。

EXPLAIN PLAN コマンドの実行のたびに、Oracle Lite は、EXPLAIN COMMAND という 1 行に続けて、1 行以上の実行計画の行を出力します。

実行計画は、問合せブロック 1 つにつき 1 行含まれます。問合せブロックは、subquery::= キーワードで始まります。

計画出力は、インデントによってネストの状態が表されます。UNION および MINUS によって結合された問合せもすべてインデントされます。計画出力の各行は、一般に次の形式になっています。

```
table-name [(column-name)] [{NL(rows)|IL(rows)} table-name [(column-name)] ]
```

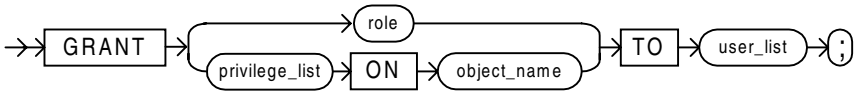
パラメータ	定義
table-name	完全修飾の別名または表の名前。
column-name	索引キーの最初の列の名前。
NL	ネスト・ループ結合。
IL	索引ループ結合は、「IL」の後に続く表を結合するために使用されます。
(rows)	オプティマイザの、結合の結果の行の推定値。

表は、左から右に実行されます。左端の表は、反復の最も外側のループを形成します。

Oracle Lite は、表の順序付けをするために行推定値を使用しますが、実際の値は重要ではありません。オプティマイザは可能な限りの最適な索引を推定します。実行時にはさらに精度が上がるため、オブジェクト・カーネルは、異なる索引を選択することもできます。

GRANT

構文



用途

ユーザーに ADMIN、DBA、DDL または RESOURCE ロールの権限を付与します。または、ユーザーにデータベース・オブジェクトに対する権限を付与します。できるだけ、DDL のかわりに DBA ロールを使用することをお薦めします。

前提条件

ロール権限を付与するには、データベースに SYSTEM として、または DBA/DDL および ADMIN 権限あるいは RESOURCE 権限を持つユーザーとしてログインする必要があります。

引数	説明
role	ADMIN、DBA/DDL または RESOURCE ロール。
user_list	ユーザー、またはカンマで区切られたユーザーのリスト。
ON	ロールを付与するデータベース・オブジェクトを示します。
privilege_list	次の権限をカンマで区切ったリストか、ALL と呼ばれる組合せ。INSERT、DELETE、UPDATE (*col_list*)、SELECT、および REFERENCES。
TO	ロールを付与するユーザーまたはユーザー・リストを示します。
object_name	オプションとして接頭辞にスキーマ名が付いた表名。

前もって定義されたロール

Oracle Lite は、便宜上いくつかの特権をまとめて、前もって定義されたロールにします。多くの場合、ユーザーに、別のスキーマの特定の特権を付与するよりも、前もって定義されたロール権限を付与する方が簡単です。Oracle Lite は、ロールの作成や削除をサポートしません。次は、Oracle Lite の前もって定義されたロールのリストです。

ロール名	ロールに付与される権限
ADMIN	ユーザーは別のユーザーを作成でき、スキーマ内のどのオブジェクトに対しても DDL および ADMIN 以外の権限を付与できます。ユーザーは、SQL 文内で、次のコマンドを実行できます。 CREATE SCHEMA、CREATE USER、ALTER USER、DROP USER、DROP SCHEMA、GRANT、および REVOKE。
DBA/DDL	ユーザーは、別の状況では SYSTEM によってしか発行できない、次の DDL 文を発行できます。 All ADMIN 権限、CREATE TABLE、CREATE ANY TABLE、CREATE VIEW、CREATE ANY VIEW、CREATE INDEX、CREATE ANY INDEX、ALTER TABLE、ALTER VIEW、DROP TABLE、DROP VIEW および DROP INDEX。
RESOURCE	RESOURCE ロールは DBA/DDL ロールと同じレベルの制御を付与しますが、ユーザー自身のドメイン上でのみ可能です。ユーザーは、SQL 文内で、次のコマンドを実行できます。 ユーザー自身のスキーマの下の任意のオブジェクトの CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE CONSTRAINT、ALTER TABLE、ALTER VIEW、ALTER INDEX、ALTER CONSTRAINT、DROP TABLE、DROP VIEW、DROP INDEX、DROP CONSTRAINT および GRANT または REVOKE 権限。

使用上の注意

privilege_list が ALL の場合、ユーザーは、表またはビューから INSERT、DELETE、UPDATE または SELECT を実行できます。*privilege_list* が、INSERT、DELETE、UPDATE または SELECT のいずれかの場合、ユーザーは表に対してその特権を持ちます。

ユーザーに表の UPDATE 権限を付与し、その後、列を追加して表を変更した場合、そのユーザーは、新しい列を更新できません。新しい列を作成した後に権限付与の文を発行した場合、ユーザーは新しい列を更新できます。たとえば、次のとおりです。

```
CREATE TABLE t1 (c1 NUMBER c2 INTEGER);
CREATE USER a IDENTIFIED BY a;
GRANT SELECT, UPDATE ON t1 TO a;
ALTER TABLE t1 ADD c3 INT;
COMMIT;
```

前の例で、GRANT 文は ALTER TABLE 文の後で発行する必要があります。そうでないと、ユーザーは新しい列 c3 を更新できません。

例 1

次の例は、MICHAEL という名前のユーザーを作成し、そのユーザーに ADMIN ロールの権限を付与します。

```
CREATE USER MICHAEL IDENTIFIED BY SWORD;

GRANT ADMIN TO MICHAEL;
```

例 2

次の例は、MICHAEL という名前のユーザーを作成し、ユーザーに、EMP 表に対する INSERT および DELETE 権限を付与します。

```
CREATE USER MICHAEL IDENTIFIED BY SWORD;

GRANT INSERT, DELETE ON EMP TO MICHAEL;
```

例 3

次の例は、新しく作成されたユーザー MICHAEL に、PRODUCT 表に対する ALL 権限を付与します。

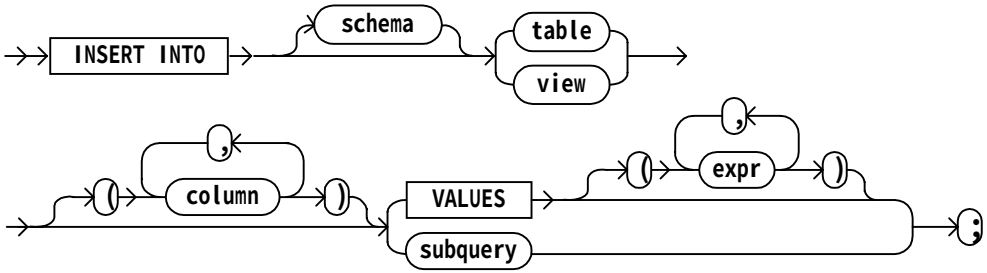
```
GRANT ALL ON PRODUCT TO MICHAEL;
```

関連項目

REVOKE

INSERT

構文



用途

表またはビューのベース表に行を追加します。

前提条件

表またはビューに列を追加するには、データベースに SYSTEM としてログインするか、表およびビューがユーザーのスキーマ内に必要です。

引数	説明
schema	表またはビューを含むスキーマ。schema を省略すると、Oracle Lite は、表またはビューがユーザー自身のスキーマ内にあると解釈します。
table	行を挿入する表の名前。
view	ベース表に行を挿入するビューの名前。
column	表またはビューの列。挿入された行の中でこの引数にリストされている各列には、VALUES 句または副問合せからの値が割り当てられます。 この引数からの表の列の 1 つを省略すると、挿入された行の列値は、表が作成された時に指定された列のデフォルト値となります。column 引数を指定しない場合、VALUES 句または問合せで表内のすべての列の値を指定する必要があります。
VALUES	表またはビューに挿入される行の値を指定します。VALUES 句には、column 引数で指定した各列の値を指定します。
expr	対応する列に割り当てられた値。これは、ホスト変数を指定できます。詳細は、「式の指定」を参照してください。
subquery	表に挿入される行を返す SELECT 文。この副問合せの SELECT リストには、INSERT 文の列リストと同数の列が必要です。

使用上の注意

- column 引数内では、同じ列名は一度しか使用できません。
- column 引数からの列を省略すると、Oracle Lite は、表が作成されるときに指定されたデフォルト値を割り当てます。
- column 引数に指定される列数は、提供される値の数と同数にしてください。column 引数を省略する場合、値の数は表の次数と同じにしてください。
- 列にユーザー定義のデフォルト値がない場合、デフォルト値は NULL となります。列に NOT NULL 制約がある場合でも、デフォルト値は NULL となります。このため、INSERT 文によりそのような列に明示的な値が提供されない場合、Oracle Lite は、整合性違反エラー・メッセージを生成します。

例

```
INSERT INTO EMP (EMPNO, ENAME, DEPTNO) VALUES ('7010', 'VINCE', '20');
```

関連項目

DELETE、UPDATE

LEVEL 疑似列

用途

LEVEL 疑似列は、階層問合せを実行する SELECT 文内で使用できます。階層問合せで返された各行について、LEVEL 疑似列はルート・ノードには 1、ルートの子には 2、というように返します。階層問合せでは、ルート・ノードは逆ツリー内で最も上位のノード、子ノードはルート以外のノード、親ノードは子のあるノード、リーフ・ノードは子のないノードです。

前提条件

なし。

使用上の注意

階層問合せにより返されるレベル数は、32 以下に制限されています。

例

次の文により階層順序内の従業員がすべて返されます。ルート行は、仕事が PRESIDENT の従業員になるように定義されます。親行の子行が、親行の従業員番号をマネージャ番号として含む行として定義されています。

```
SELECT LPAD(' ',2*(LEVEL-1)) || ename org_chart,
empno, mgr, job
FROM emp
START WITH job = 'PRESIDENT'
CONNECT BY PRIOR empno = mgr;
```

次の結果を返します。

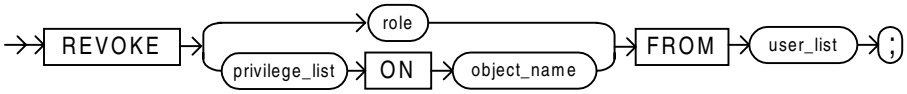
```
ORG_CHART              EMPNO       MGR JOB
------------------ --------- --------- ---------
                        7839           PRESIDENT
    JONES               7566      7839 MANAGER
    SCOTT               7788      7566 ANALYST
    ADAMS               7876      7788 CLERK
    FORD                7902      7566 ANALYST
    SMITH               7369      7902 CLERK
    CLARK               7782      7839 MANAGER
    MILLER              7934      7782 CLERK
    BLAKE               7698      7839 MANAGER
    WARD                7521      7698 SALESMAN
    JAMES               7900      7698 CLERK
    TURNER              7844      7698 SALESMAN
    ALLEN               7499      7698 SALESMAN
    MARTIN              7654      7698 SALESMAN

14 rows selected.
```

関連項目

ROWNUM 疑似列

REVOKE

構文



用途

ユーザーの ADMIN、DBA/DDL または RESOURCE ロールを取り消します。または、ユーザーのデータベース・オブジェクトに対する権限を取り消します。DDL のかわりに DBA ロールを使用することをお薦めします。

前提条件

ユーザーからロールを取り消すには、データベースに SYSTEM として、あるいは DBA または ADMIN 権限を持つユーザーとしてログインする必要があります。

引数	説明
role	ADMIN、DBA/DDL または RESOURCE ロール。
user_list	ユーザー、またはカンマで区切られたユーザーのリスト。
privilege_list	次の権限をカンマで区切ったリストか、ALL と呼ばれる組合せ。INSERT、DELETE、UPDATE（*col_list*）および SELECT。
object_name	接頭辞にスキーマ名が付いた表名。

使用上の注意

privilege_list が、INSERT、DELETE、UPDATE または SELECT のいずれかの場合、ユーザーは表またはビューに対してその権限を持ちます。*privilege_list* が ALL の場合、ユーザーは、表またはビューから INSERT、DELETE、UPDATE または SELECT を実行できます。

例 1

次の例は、STEVE という名前のユーザーを作成し、そのユーザーに ADMIN ロールの権限を付与します。次に、ADMIN ロールをユーザー STEVE から取り消します。

```
CREATE USER STEVE IDENTIFIED BY STINGRAY;
GRANT ADMIN TO STEVE;
REVOKE ADMIN FROM STEVE;
```

例 2

次の例は、ユーザー SCOTT から、EMP 表に対する INSERT および DELETE 権限を取り消します。

```
REVOKE INSERT,DELETE ON EMP FROM SCOTT;
```

例 3

次の例は、CHARLES という名前のユーザーを作成し、そのユーザーに、PRICE 表に対する INSERT および DELETE 権限と、ITEM 表に対する ALL 権限を付与します。次に、ユーザー CHARLES から、PRICE 表と ITEM 表に対する権限をすべて取り消します。

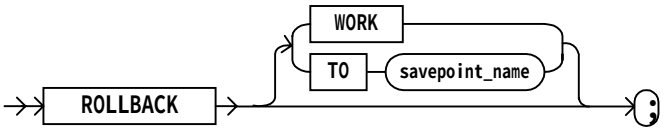
```
CREATE USER CHARLES IDENTIFIED BY VORTEX;
GRANT INSERT, DELETE, UPDATE ON PRICE TO CHARLES;
GRANT ALL ON ITEM TO CHARLES;
REVOKE ALL ON PRICE FROM CHARLES;
REVOKE ALL ON ITEM FROM CHARLES;
```

関連項目

GRANT

ROLLBACK

構文



用途

カレント・トランザクションで実行された操作をすべて取り消します。

前提条件

なし。

引数	説明
work	ANSI 互換性を提供するためにサポートされているオプションの引数。
TO	セーブポイントまでロールバックできるようにするためのオプションの引数。
savepoint_name	その地点までロールバックするセーブポイントの名前。

使用上の注意

まだトランザクションに入っていないと、Oracle Lite は、SQL 文を最初に発行した地点から開始します。COMMIT または ROLLBACK コマンドを使用するまでは、発行する文はすべてトランザクションと解釈されます。

COMMIT コマンドは、トランザクションの開始に至るまでのすべてを保存して、データベース内のデータに対する変更を永久的なものにします。変更がコミットされるまで、変更を格納できるように、あるいは、データを前の状態にリストアできるように、新旧両方のデータが存在します。

ROLLBACK コマンドは、保留されているカレント・トランザクションで行われたデータに対する変更を破棄して、トランザクションの開始前の状態にデータベースをリストアします。SAVEPOINT を指定して、トランザクションの部分をロールバックできます。

重要： Oracle Lite は、CREATE DATABASE 以外は、DDL コマンドを自動的にコミットすることはありません。Oracle Lite 内の DDL コマンドは、ロールバックの対象となります。

例

次の例は、新しい行を DEPT 表に挿入し、その後、そのトランザクションをロールバックします。この例は、ROLLBACK および ROLLBACK WORK の両方について、同じ結果を返します。

```
INSERT INTO DEPT (deptno, dname, loc) VALUES (50, 'Design', 'San Francisco');
SELECT * FROM dept;
```

次の結果を返します。

```
   DEPTNO DNAME          LOC
--------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
       50 DESIGN         SAN FRANCISCO

ROLLBACK WORK;
SELECT * FROM dept;
```

次の結果を返します。

```
   DEPTNO DNAME          LOC
--------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
```

ODBC 2.0

ROLLBACK コマンドは ODBC SQL 構文の一部ではありませんが、ODBC はこのコマンドをそのままデータベースに渡します。

ODBC プログラムは通常、`SQL_ROLLBACK` フラグを指定した API コール `SQLTransact()` を使用します。

関連項目

SAVEPOINT

ROWNUM 疑似列

用途

問合せによって返される各行について、ROWNUM 疑似列は、Oracle Lite が表または結合行のセットから行を選択した順番を示す番号を返します。選択された最初の行は、ROWNUM は 1 で、2 番目の行は 2、というようになります。

前提条件

なし。

使用上の注意

同じ副問合せ内で ROWNUM の後に ORDER BY 句が続く場合、行は ORDER BY 句によって並べ替えられます。結果は、行がアクセスされる方法によって異なることがあります。たとえば、ORDER BY 句によって Oracle Lite がデータのアクセスに索引を使用する場合、Oracle Lite は、索引なしの場合と異なる順番で行を取り出す可能性があります。

副問合せ内に ORDER BY 句を入れて、最上位の問合せに ROWNUM 条件が指定されている場合、行の並べ替えの後で、ROWNUM 条件の適用を強制できます。例 3 を参照してください。

例 1

次の例は、問合せで返される行数を制限するために、ROWNUM を使用します。

```
SELECT * FROM emp WHERE ROWNUM < 10;
```

例 2

次の例は、同じ問合せ内で、ORDER BY 句を ROWNUM の後に指定します。したがって、行は ORDER BY 句によって並べ替えられ、前の例と同じ結果にはなりません。

```
SELECT * FROM emp WHERE ROWNUM < 11 ORDER BY empno;
```

例 3

次の問合せは、最小の従業員番号を 10 個返します。これは、「トップ N 問合せ」と呼ばれることもあります。

```
SELECT * FROM
   (SELECT empno FROM emp ORDER BY empno)
   WHERE ROWNUM < 11;
```

例 4

次の問合せは行を返しません。

```
SELECT * FROM emp WHERE ROWNUM > 1;
```

最初にフェッチされた行は、ROWNUM に 1 が割り当てられ、条件が偽にされます。2 番目にフェッチされる行は、今では最初の行となり、それにも ROWNUM に 1 が割り当てられ、条件が偽にされます。後に続く行はすべて条件を満たすことに失敗し、そのため、行は何も返されません。

例 5

次の文は、表の各行に一意の値を割り当てます。

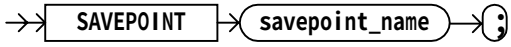
```
UPDATE tabx SET col1 = ROWNUM;
```

関連項目

LEVEL 疑似列

SAVEPOINT

構文



用途

後でロールバックを可能にするための、トランザクション内のポイントを識別します。

前提条件

なし。

使用上の注意

セーブポイントを設定すると、そこまでロールバックできます。または、後でそれを削除できます。セーブポイントまでロールバックするには、次の文を使用します。

```
ROLLBACK TO <savepoint_name>
```

セーブポイントを削除するには、次の文を使用します。

```
REMOVE SAVEPOINT <savepoint_name>
```

セーブポイントまでロールバックするか、セーブポイントを削除する場合、ネストされたセーブポイントもすべてロールバックまたは削除されます。メモリー使用量を削減するために、できるだけ早くセーブポイントを削除してください。

注意： REMOVE SAVEPOINT コマンドは、SQL*Plus を介して発行されたときは機能しません。SQL*Plus はこのコマンドの実行を許していません。

ユーザー定義セーブポイントには名前を付けることができます。また、トランザクション処理内でカレント・ポイントをマークできます。ROLLBACK と一緒に使用することで、SAVEPOINT によって、トランザクション全体ではなく、トランザクションの部分を取り消すことができます。セーブポイントまでロールバックすると、そのセーブポイントの後にマークされたセーブポイントは、すべて削除されます。COMMIT 文は、最後にコミットまたはロールバックされた後にマークされたセーブポイントは、すべて削除します。

セッションごとに定義できる「アクティブ」セーブポイントの数に制限はありません。アクティブ・セーブポイントとは、最後にコミットまたはロールバックされてからマークされたセーブポイントのことです。

例

次の例は、2 人の従業員 Blake と Clark の給与を更新します。次に、EMP 表内で給与の合計をチェックします。例は、各従業員の給与のセーブポイントまでロールバックし、Clark の給与を更新します。

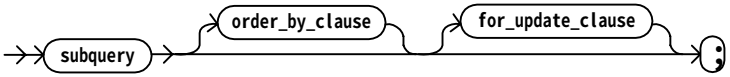
```
UPDATE emp
    SET sal = 2000
    WHERE ename = 'BLAKE';

SAVEPOINT blake_sal;

UPDATE emp
    SET sal = 1500
    WHERE ename = 'CLARK';

SAVEPOINT clark_sal;
SELECT SUM(sal) FROM emp;
ROLLBACK TO SAVEPOINT blake_sal;
UPDATE emp
    SET sal = 1300
    WHERE ename = 'CLARK';
COMMIT;
```

関連項目

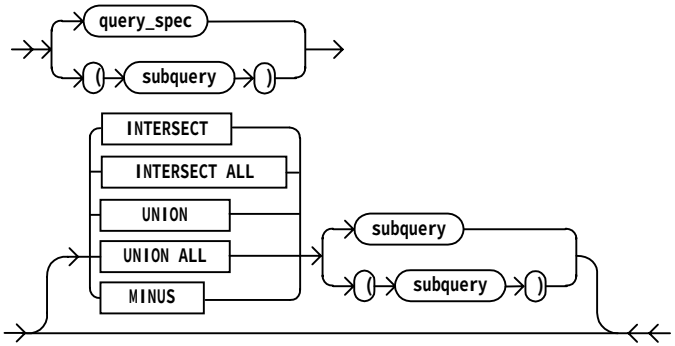
COMMIT、SAVEPOINT、ROLLBACK

SELECT

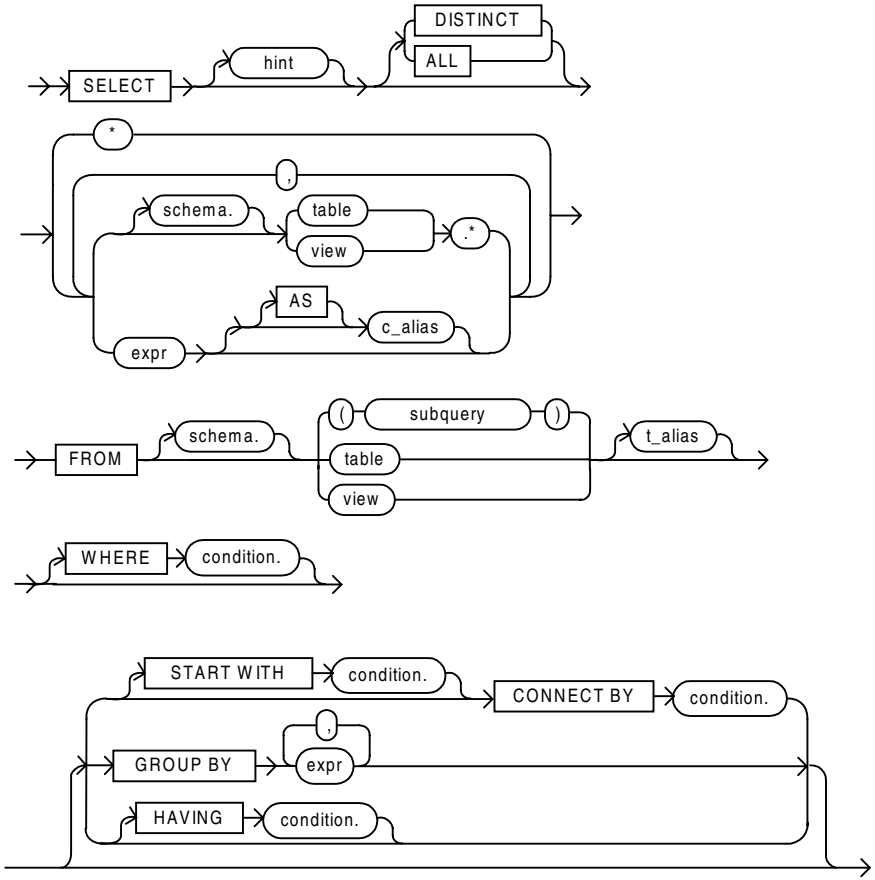
構文



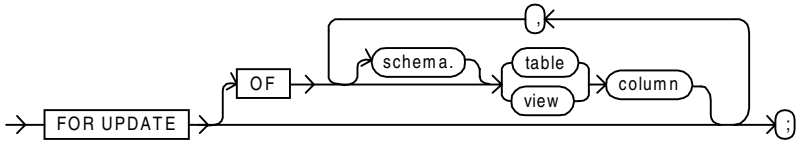
subquery::=



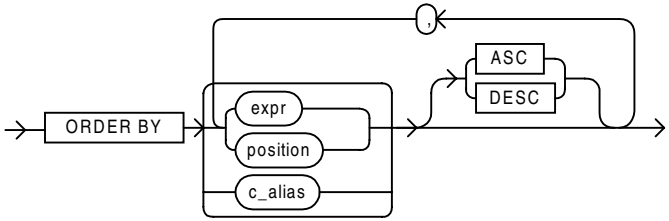
query_spec::=



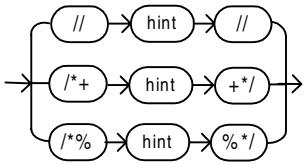
for_update_clause::=



order_by_clause::=



hint::=



用途

1 つ以上の表またはビューからデータを取り出します。SELECT 文を使用して Java ストアド・プロシージャを起動することもできます。

前提条件

表またはビューからデータを取り出すには、データベースに SYSTEM としてログインするか、表およびビューがユーザーのスキーマ内に必要です。

引数	説明
DISTINCT	選択された重複行の各セットのコピーを 1 つのみ返します。重複行とは、選択リスト内の各式に一致する値を持つ複数の行です。
ALL	重複行も含めて、選択された行をすべて返します。デフォルトは ALL です。
*	FROM 句にリストされた表、ビューまたはスナップショットからの列をすべて返します。
table.*	選択された表からすべての列を選択します。ユーザー自身のスキーマ以外のスキーマから選択する場合は、*schema* 修飾子を使用します。

引数	説明
view.*	選択されたビューからすべての列を選択します。ユーザー自身のスキーマ以外のスキーマから選択する場合は、schema 修飾子を使用します。
expr	通常は、FROM 句内の表またはビューの 1 つから得た列の値に基づいて式を選択します。このリストの列を含む表またはビュー自体が FROM 句で schema を使用して修飾されているときにのみ、列名を schema で修飾できます。詳細は、「式の指定」を参照してください。
hint	ヒントは Oracle Lite オプティマイザにより処理され、文実行の選択肢を示します。詳細は、「ヒントの使用方法」を参照してください。
/*+ ... +*/	Oracle および Oracle Lite の両方で処理されるヒント。SQL*Plus では無視されます。
/*% ...%*/	Oracle ではコメントとして扱われ、Oracle Lite では処理されるヒント。SQL*Plus では無視されます。
// ... //	Oracle および Oracle Lite の両方で処理されるヒント。SQL*Plus では処理されます。
c_alias	列の別名を提供します。これは列の式とは異なる名前で、列の別名が列ヘッダーで使用されるようになります。列の別名は、列の実際の名前には影響しません。別名は、ORDER BY 句の中でのみ使用できます。問合せ内の別の句では使用できません。
schema	選択された表、ビューまたはスナップショットの入ったスキーマ。*schema* を省略すると、Oracle Lite ではユーザー自身のスキーマ内に表、ビューまたはスナップショットがあると解釈されます。
table	データが選択される表。
view	データが選択されるビュー。
t_alias	問合せを評価する目的で、表、ビューまたはスナップショットに対して異なった名前、つまり別名を指定します。相関問合せで一番よく使用されます。その問合せの中で、表、ビューまたはスナップショットを参照する場合は、別名を参照する必要があります。

引数	説明
WHERE	指定された condition が TRUE である行に選択を制限します。WHERE 句を省略すると、Oracle Lite は、FROM 句内の表、ビューまたはスナップショットのすべての列を返します。埋込み SQL の SELECT 文では、WHERE 句の condition にホスト変数を入れられます。WHERE は、TRUE または FALSE と評価される条件式を指定します。詳細は、「式の指定」を参照してください。
condition	検索条件。有効な条件の作成の詳細は、「SQL 条件の指定」を参照してください。
START WITH	行を階層順序で返します。
CONNECT BY	階層問合せ内の親行と子行の関連を指定します。condition はこの関連を指定し、PRIOR 演算子を使用して親行を参照する必要があります。親行の子を検索するために、Oracle Lite は、表内の各行の PRIOR 式を評価します。条件が TRUE である行は、親行の子です。詳細は、PRIOR 演算子を参照してください。
GROUP BY	各行の *expr* 引数の値に基づいて選択された行をグループ化し、各グループのサマリー情報を含んだ単一の行を返します。
HAVING	指定された condition が TRUE であるグループに返される行のグループを制限します。この句を省略すると、Oracle Lite は、すべてのグループのサマリー行を返します。詳細は、「SQL 条件の指定」を参照してください。
INTERSECT	両方の問合せで選択された重複していない行をすべて返します。INTERSECT は UNION より高い優先順位を持ちます。
INTERSECT ALL	両方の問合せで選択された重複していない行をすべて返します。INTERSECT と同じ結果となります。この構文はサポートされていますが、機能はありません。
UNION	両方の問合せで選択された重複していない行をすべて返します。
UNION ALL	重複行も含めて、どちらかの問合せで選択された行をすべて返します。
MINUS	最初の問合せで選択された、他と重複しない行をすべて返します。2 番目の問合せで選択された行は返されません。
command	それ自身が別の SELECT コマンドのパラメータである SELECT コマンドのパラメータをすべて参照します。SELECT コマンド内にある SELECT コマンドのパラメータを入力する場合、WHERE 文は使用できません。

引数	説明
ORDER BY	以下の引数に従って、SELECT 文で返される行を順序指定します。 *expr*（式）は、*expr* の値に基づいて行を順序指定します。式は選択リストの列に基づくか、または FROM 句にある表、ビューまたはスナップショット内の列に基づきます。 *position* では、選択リストのこの位置にある式の値に基づいて行を順序指定します。 ASC では、昇順のソート順序を指定します。ASC がデフォルトです。 DESC では、降順のソート順序を指定します。
FOR UPDATE	選択された行をロックします。 FOR UPDATE 句内の列リストは無視されます。 FOR UPDATE 句は、ORDER BY 句の前または後のどちらでも使用できます。
column	更新される列。

使用上の注意

WHERE 句を指定していないときに FROM 句に表が 2 つ以上ある場合、Oracle Lite は、関わっている表すべての直積演算を計算します。

LEVEL 疑似列は、階層問合せを実行する SELECT 文内で使用できます。詳細は、「LEVEL 疑似列」を参照してください。階層問合せでは、結合を実行したり、ビューからデータを選択したりすることはできません。

式を使用して列を選択する場合、それらの列には別名が必要です。別名は、問合せによって選択される列式の名前を指定します。別名の数は、問合せで選択される式の数と一致する必要があります。別名は、問合せ内で一意にしてください。

例 1

```
SELECT * FROM EMP WHERE SAL = 1300;
```

次の結果を返します。

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
     7782 CLARK      MANAGER        7839 1981-06-0      1300                   10
     7934 MILLER     CLERK          7782 1982-01-2      1300                   10
```

例 2

```
SELECT 'ID=',EMPNO, 'Name=',ENAME, 'Dept=',DEPTNO
FROM EMP ORDER BY DEPTNO;
```

次の結果を返します。

```
'ID     EMPNO 'NAME ENAME      'DEPT    DEPTNO
--- --------- ----- ---------- ----- ---------
ID=      7839 Name= KING       Dept=        10
ID=      7934 Name= MILLER     Dept=        10
ID=      7782 Name= CLARK      Dept=        10
ID=      7566 Name= JONES      Dept=        20
ID=      7876 Name= ADAMS      Dept=        20
ID=      7788 Name= SCOTT      Dept=        20
ID=      7369 Name= SMITH      Dept=        20
ID=      7902 Name= FORD       Dept=        20
ID=      7521 Name= WARD       Dept=        30
ID=      7900 Name= JAMES      Dept=        30
ID=      7844 Name= TURNER     Dept=        30
ID=      7499 Name= ALLEN      Dept=        30
ID=      7654 Name= MARTIN     Dept=        30
ID=      7698 Name= BLAKE      Dept=        30

14 rows selected.
```

例 3

```
SELECT 'ID=', EMPNO,
'Name=', ENAME,
'Dept=', DEPTNO
FROM EMP WHERE SAL >= 1300;
```

次の結果を返します。

```
'ID     EMPNO 'NAME ENAME      'DEPT    DEPTNO
--- --------- ----- ---------- ----- ---------
ID=      7839 Name= KING       Dept=        10
ID=      7698 Name= BLAKE      Dept=        30
ID=      7782 Name= CLARK      Dept=        10
ID=      7566 Name= JONES      Dept=        20
ID=      7499 Name= ALLEN      Dept=        30
ID=      7844 Name= TURNER     Dept=        30
ID=      7902 Name= FORD       Dept=        20
ID=      7788 Name= SCOTT      Dept=        20
ID=      7934 Name= MILLER     Dept=        10

9 rows selected.
```

例 4

```
SELECT * FROM (SELECT ENAME FROM EMP WHERE JOB = 'CLERK'
UNION
SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');
```

次の結果を返します。

```
ENAME
----------
ADAMS
FORD
JAMES
MILLER
SCOTT
SMITH
```

ヒントの使用方法

SQL 文の中で特別なテキストを使用して、指示やヒントを Oracle Lite データベース・オプティマイザに渡すことができます。オプティマイザはこれらのヒントを、その文の実行計画を選択するためのアドバイスとして使用します。ヒント・テキストは、使用しているアプリケーションに応じて次の記号セットにより区切られます。最初の 2 つ、/*+ ... +*/ および /*% ... %*/ は、SQL*Plus に影響を与えません。図中のサンプル構文はすべて、ユーザーの書いたアプリケーションから動作します。

/*% ... %*/ 構文は、Oracle8*i* オプティマイザによりコメントとして扱われますが、/*+ ORDERED +*/ は処理されます。Oracle Lite と Oracle8*i* で同じコードを共有して、Oracle Lite のみにヒントを指定するには、構文 /*% ORDERED %*/ を使用します。Oracle Lite と Oracle の両方のオプティマイザにヒントを指定するには、構文 /*+ ORDERED +*/ を使用します。ヒント・テキストは、SELECT キーワードの後に置く必要があります。ヒント・テキストは大 / 小文字を区別しません。

例 5

この例では、「ordered」ヒントは、結合順序の中で、EMP 表を最も外側の表として選択します。オプティマイザは依然として、実行に最適の索引を取り出そうとします。たとえばビューの代替や副問合せの非ネストなど、その他の最適化を依然として実行しようとします。

```
Select //ordered//  Eno, Ename, Loc from Emp, Dept
where Dept.DeptNo = Emp.DeptNo and Emp.Sal > 50000;
```

例 6

この例では、ヒントは、表（Product、Item および Ord）を指定された順序、すなわち、Product、Item、Ord の順番で結合します。ヒントは副問合せのみに制限されます。

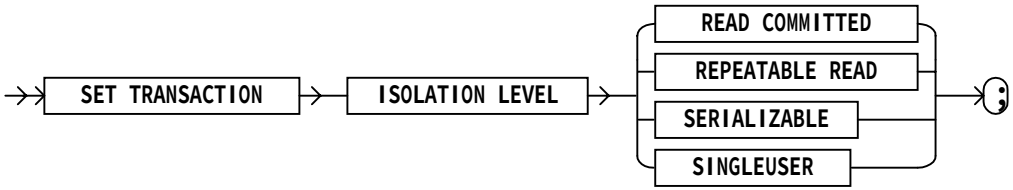
```
Select CustId, Name, Phone from Customer
Where CustId In ( Select //ordered// Ord.CustId from Product, Item, Ord
    Where Ord.OrdId = Item.OrdId And
        Item.ProdId = Product.ProdId And
        Product.Descrip like '%TENNIS%')
```

関連項目

CONSTRAINT 句、DELETE、UPDATE

SET TRANSACTION

構文



用途

カレント・トランザクションの分離レベルを確立します。

注意： Oracle Lite は、データ定義言語文の実行の前後で、カレント・トランザクションをコミットします。

前提条件

SET TRANSACTION 文を使用する場合、その文がトランザクションの最初の文になります。しかし、トランザクションは、SET TRANSACTION 文を持つ必要はありません。

引数	説明
SET TRANSACTION	カレント・トランザクションの分離レベルを確立します。SET TRANSACTION 文によって実行される操作は、ユーザーのカレント・トランザクションにのみ影響し、別のユーザーや別のトランザクションには影響しません。ユーザーのトランザクションは、COMMIT または ROLLBACK 文を発行すると、いつでも終了します。
ISOLATION LEVEL	データベースの変更を含んだトランザクションの処理方法を指定します。
READ COMMITTED	分離レベル。トランザクションは、別のトランザクションによって書込みロックされた行がアンロックされるまで、実行されません。トランザクションは、カレント行を読み取るときに、読取りロックを行い、カレント行を更新または削除するときに、書込みロックを行います。これによって、別のトランザクションがそれを更新または削除することを防止できます。トランザクションは、トランザクションがカレント行から離れたときに、読取りロックを解除し、トランザクションがコミットまたはロールバックされたときに、書込みロックを解除します。
REPEATABLE READ	分離レベル。トランザクションは、別のトランザクションによって書込みロックされた行がアンロックされるまで、実行されません。トランザクションは、トランザクションがアプリケーションに返すすべての行の読取りロックを維持し、挿入、更新または削除するすべての行の書込みロックを維持します。トランザクションは、トランザクションがコミットまたはロールバックされたときにのみそのロックを解除します。
SERIALIZABLE	分離レベル。トランザクションは、別のトランザクションによって書込みロックされた行がアンロックされるまで、実行されません。トランザクションは、行の範囲を読み取るときに、読取りロックを行い、行の範囲を更新または削除するときに、書込みロックを行います。これによって、別のトランザクションが行を更新または削除することを防止できます。
SINGLEUSER	分離レベル。トランザクションはロックを行いません。そのため、使用されるメモリー量は少なくなります。これは、データベースのバルク・ロードに使用します。

使用上の注意

なし。

例

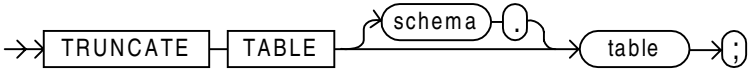
```
SET TRANSACTION ISOLATION LEVEL SINGLEUSER;
```

関連項目

COMMIT、ROLLBACK

TRUNCATE TABLE

構文



用途

表からすべての行を削除します。この文は、Oracle8*i* との互換性のために提供されています。この文は、次の文と同じアクションを実行します。

```
DELETE FROM table_name ;
```

引数	説明
schema	表の入ったスキーマ。
table	切り捨てられる表の名前。

使用上の注意

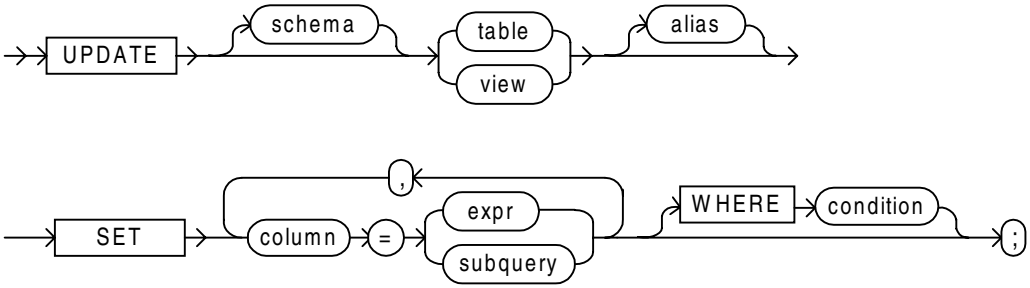
主キーを含み、依存表に行が存在する表は切り捨てられません。

例

```
TRUNCATE TABLE emp;
```

UPDATE

構文



用途

表内またはビューのベース表内にある値を変更します。

前提条件

データベース表またはビューにある値を更新するには、データベースに SYSTEM としてログインするか、表およびビューがユーザーのスキーマ内に必要です。

引数	説明
schema	表またはビューを含むスキーマ。*schema* を省略すると、Oracle Lite ではユーザー自身のスキーマ内に表またはビューがあると解釈されます。
table	更新される表の名前。
view	ベース表を更新するビューの名前。
alias	UPDATE コマンド内のもう 1 つの句内の表またはビューの名前のラベル付けを再度行います。
SET	後続する列が特定値に設定されるよう指定します。
column	更新される表またはビューの列名。SET 句内で表の列を 1 つ省略すると、その列の値は変更されません。
expr	対応する列に割り当てられた新規作成値。これは、ホスト変数を指定できます。
subquery	更新される副問合せ。

引数	説明
WHERE	指定の condition が TRUE である行に更新される行を制限します。WHERE 句を省略すると、Oracle Lite は、表またはビュー内の行をすべて更新します。
condition	検索条件。有効な条件の作成の詳細は、「SQL 条件の指定」を参照してください。

使用上の注意

- SET 句内では、同じ列名は一度しか使用できません。
- WHERE 句が指定されていない場合、表のすべての列が更新されます。
- 位置づけ UPDATE では、カーソルを更新可能にしてください。

例

```
UPDATE EMP SET SAL = SAL * .45 WHERE JOB = 'PRESIDENT';
```

ODBC 2.0

UPDATE の ODBC SQL 構文は指定されたものと同じです。さらに、次の構文がサポートされています。

```
WHERE CURRENT OF CURSOR cursor_name
```

関連項目

DELETE、INSERT

A

Oracle Lite キーワードと予約語

この付録では、Oracle Lite のキーワードと予約語をリストします。

Oracle Lite キーワード

キーワードは予約語ではありませんが、特定のコンテキストで特別の意味を持ちます。キーワードを使用して、表および列の名前を定義できます。

注意： Java クラスの新しいメソッドを集計名（AVG、MAX、MIN、COUNT、SUM）として定義し、関数名（UPPER、LOWER、など）がユーザー定義メソッド名よりも優先する場合は、十分に注意する必要があります。

次に、Oracle Lite のキーワードをリストします。

文字	キーワード	
A	ADD_MONTHS	ASCII
	AFTER	AUTOCOMMIT
	ARGS	AVG
B	BEFORE	BIT
	BIGINT	BIT_LENGTH
	BINARY	

文字	キーワード	
C	CASCADE	COMPILE
	CAST	CONCAT
	CATALOG	CONSTRUCTOR
	CEIL	CONVERT
	CHAR	COUNT
	CHAR_LENGTH	CURDATE
	CHARACTER	CURTIME
	CHR	CURTIMESTAMP
	COMMITTED	CURVAL
D	DATABASE_ID	DEC
	DATABASE_SIZE	DECIMAL
	DATE	DOUBLE
	DAY	DUAL（これはすでにダミー表の名前であるため、表名として使用しないでください。）
	DAYOFMONTH	DUMP$
	DAYOFWEEK	
	DAYOFYEAR	
	DEBUG_LITE	
E	EXTENT_SIZE	
	EXTRACT	
F	FLOOR	
G	GREATEST	
H	HIDE	
	HOUR	

文字	キーワード	
I	IFNULL	INT
	INITCAP	INTERVAL
	INSTR	ISOLATION
	INSTRB	
K	KEY	
L	LAST_DAY	LEVEL
	LCASE	LOCATE
	LEAST	LOWER
	LENGTH	LPAD
	LENGTHB	LTRIM
M	MAX	MINVALUE
	MAXVALUE	MOD
	MIN	MONTH
	MINUTE	MONTHS_BETWEEN
N	NEXT_DAY	NUMBER
	NEXTVAL	NUMERIC
	NOMAXVALUE	NVL
	NOMINVALUE	
	NOW	
O	OCTET_LENGTH	
	OJ	
P	POSITION	
	PRECISION	
Q	QUARTER	

文字	キーワード	
R	READ	ROUND
	REAL	ROWID
	REPEATABLE	RPAD
	REPLACE	RTRIM
	RESTRICT	
S	SAVEPOINT	SQL_TSI_MONTH
	SCHEMA	SQL_TSI_QUARTER
	SECOND	SQL_TSI_SECOND
	SEQUENCE	SQL_TSI_WEEK
	SERIALIZABLE	SQL_TSI_YEAR
	SMALLINT	START
	SOURCE	STDDEV
	SQL_TSI_DAY	SUBSTR
	SQL_TSI_FRAC_SECOND	SUBSTRB
	SQL_TSI_HOUR	SUBSTRING
	SQL_TSI_MINUTE	SUM
T	TIME	TO_DATE
	TIMESTAMP	TO_NUMBER
	TIMESTAMPADD	TRANSACTION
	TIMESTAMPDIFF	TRANSLATE
	TINYINT	TRIMBOTH
	TO_CHAR	TRUNC
U	UCASE	UPPER
	UNCOMMITTED	
V	VARBINARY	VARYING
	VARIANCE	

文字	キーワード
W	WEEK
	WORK
Y	YEAR
Z	ZONE

注意： キーワード NEXTVAL および CURVAL を表の列名として使用できます。ただし、Oracle Lite は、*tablename*.NEXTVAL および *tablename*.CURVAL を順序への参照として扱います。

Oracle Lite 予約語

予約語を、データベース・オブジェクトの名前またはその一部として使用できません。次に、Oracle Lite の予約語をリストします。Oracle Lite の予約語のいくつかは、Oracle の予約語でもあります。後ろにアスタリスク（*）が付いた予約語は、Oracle Lite のみの予約語です（Oracle の予約語ではありません）。

文字	予約語	
A	ADD	ANY
	ALL	AS
	ALTER	ASC
	AND	ATTACH*
B	BETWEEN	BY
	BOTH*	

文字	予約語	
C	CALL*	CONNECT
	CASE*	CONSTRAINT*
	CAST*	CONSTRAINTS*
	CHECK	CREATE
	CLASS*	CURRENT
	COLUMN	CURRENT_DATE*
	COMMENT	CURRENT_TIME*
	COMMIT*	CURRENT_TIMESTAMP*
D	DATABASE	DESC
	DECODE*	DETACH*
	DEFAULT	DISTINCT
	DELETE	DROP
E	EACH*	ESCAPE*
	ELSE	EXISTS
	END*	
F	FLOAT	FROM
	FOR	FULL*
	FOREIGN*	
G	GRANT	
	GROUP	
H	HAVING	

文字	予約語	
I	IN	INTERSECT
	INCREMENT	INTERVAL*
	INDEX	INTO
	INSERT	IS
	INTEGER	
J	JOIN*	
	JAVA	
L	LEADING*	LOCK
	LEFT*	LONG
	LIKE	
M	MINUS	
N	NOT	NULL
	NOWAIT	
O	OF	ORDER
	OFF*	OUTER*
	ON	OPTION
	OR	
P	PRIMARY*	PUBLIC
	PRIOR	
R	RAW	RIGHT*
	REFERENCES*	ROLLBACK*
	REVOKE	ROW

文字	予約語	
S	SELECT	SQL_INTEGER*
	SESSION	SQL_LONGVARCHAR*
	SET	SQL_REAL*
	SOME*	SQL_SMALLINT*
	SQL_BIGINT*	SQL_TIME*
	SQL_CHAR*	SQL_TIMESTAMP*
	SQL_DATE*	SQL_VARCHAR*
	SQL_DECIMAL*	START
	SQL_DOUBLE*	SYNONYM
	SQL_FLOAT*	SYSDATE
T	TABLE	TIMEZONE_MINUTE*
	THEN	TO
	TIMESTAMPADD*	TRAILING*
	TIMESTAMPDIFF*	TRIGGER
	TIMEZONE_HOUR*	TRIM*
U	UNION	UPDATE
	UNIQUE	USER
V	VALUES	VARCHAR2
	VARCHAR	VIEW
W	WHEN*	WITH
	WHERE	

B

Oracle Lite データ型

Oracle Lite は、次のデータ型をサポートします。

データ型	説明
BIGINT	精度が 19 桁の整数データ型。
BINARY	最大 4,096 バイトのバイナリ・データを格納できます。
BIT	アプリケーションは文字セマンティクスによる制約のないビットを格納できます。
BLOB	バイナリ・ラージ・オブジェクト。最大サイズは、2GB です。
CHAR	長さが size バイトの固定長の文字データ。最大サイズは、4,096 バイトです。デフォルトおよび最小サイズは、1 バイトです。
CLOB	シングルバイト文字を含むキャラクタ・ラージ・オブジェクト。固定幅および可変幅の両方のキャラクタ・セットがサポートされ、共に、CHAR データベース・キャラクタ・セットを使用します。最大サイズは、2GB です。
DATE	BC 4712 年 1 月 1 日から、AD 4712 年 12 月 31 日までの有効日付。
DECIMAL	精度（10 進数値）またはスケール（小数値）で測定可能な数値。精度は、DECIMAL (*p*) を使用して計算できます。スケールは、NUMERIC (*p*, *s*) を使用して計算できます。指定した精度よりも高い精度は受け入れられますが、指定したものより低い精度は受け入れられません。

データ型	説明
DOUBLE PRECISION	実装時に定義された精度を含みます。これは、REAL よりも高くしてください。
FLOAT	精度を指定可能にします。結果の精度は、少なくとも要求された精度よりも高くしてください。値によっては、FLOAT (*p*) と入力して精度を指定できるものもあります。たとえば、可搬性のあるアプリケーションで、あるプラットフォームでは単精度、別のプラットフォームでは倍精度を使用できます。
INTEGER	精度（格納される 10 進数値またはビットの数）が実装時に定義される整数値。
LONG	最大 2GB、または 231 ～ 1 バイトの可変長文字データ。
LONG RAW	最大 2GB の可変長ロー・バイナリ・データ。
LONG VARBINARY	格納されるが解析されない最大 2GB の可変バイナリ・データ。
LONG VARCHAR	最大長バイトの可変長文字列。最大サイズは 2GB で、最小サイズは 1 です。VARCHAR2 のサイズを指定する必要があります。
NUMBER	精度 *p* およびスケール *s* を持つ数値。精度 *p* の範囲は、1 ～ 38 です。スケール *s* の範囲は、-84 ～ 127 です。Oracle 互換性モードでは、精度の指定のない数値は、DOUBLE PRECISION にマップされます。
NUMERIC	精度（10 進数値）またはスケール（小数値）で測定可能な数値。精度は、DECIMAL (*p*) を使用して計算できます。スケールは、NUMERIC (*p, s*) を使用して計算できます。スケールは、負にすることはできません。また、数値自体より大きくすることはできません。
RAW	長さバイトのロー・バイナリ・データ。最大サイズは、4,096 バイトです。RAW 値のサイズを指定する必要があります。
REAL	オプションなしの単精度浮動小数点を要求できます。精度は、実装によって選択され、通常、ハードウェア・プラットフォームのデフォルトの単精度となります。
ROWID	16 バイトの 16 進文字列で、表内の行の一意のアドレスを表示します。ROWID は主に ROWID 疑似列によって返される値のためのものです。
SMALLINT	精度は実装時に定義され、値は INTEGER の実装値を超えない整数値。

データ型	説明
TIME	時間値を、時、分および秒で格納します。時は、00 ～ 23 の 2 桁の数で表されます。分も、00 ～ 59 の 2 桁の数で表されます。秒は、0 ～ 61.999 の値です。
TIMESTAMP	日付の年、月、日と、時刻の時、分、秒の値を格納します。TIMESTAMP の長さと制限は、DATE 値と TIME 値に対応します。ただし、デフォルトが、TIME では 0 であるのに対し、TIMESTAMP では 6 です。
TINYINT	精度が 1 バイトの整数（-128 ～ +127）。
VARBINARY	格納されるが解析されない可変バイナリ・データ。
VARCHAR	VARCHAR2 を参照してください。
VARCHAR2	最大長が 4,096 バイト（最小は 1）の可変長文字列。VARCHAR2 のサイズを指定する必要があります。

BIGINT

[ODBC]

大整数型。SQL_C_CHAR または SQL_C_BINARY 変数と結合します。

構文

```
BIGINT
```

使用上の注意

BIGINT は精度 19 およびスケール 0 の真数値で、通常は 8 バイトです。`-10^19 < n < 10^19` で、*n* は、BIGINT 値です。

例

```
BIGINT
```

BINARY

[ODBC]

可変長バイナリ・データ型。SQL_C_CHAR または SQL_C_BINARY 配列と結合します。

構文

```
BINARY [( <precision> )]
```

キーワードとパラメータ

<precision> バイトの最大値。

使用上の注意

BINARY は VARBINARY および RAW と同義です。

例

```
BINARY(1024)
```

BIT

BIT データ型。

構文

```
BIT
```

使用上の注意

精度は 1 です。

例

```
BIT
```

BLOB

BLOB データ型は、テキスト、イメージ、ビデオおよび空間データなど、最大 2GB の構造化されていないラージ・データを格納できます。

構文

```
BLOB
```

使用上の注意

表を作成するとき、オプションで、BLOB 列に異なる表領域および記憶特性を指定できます。

EMPTY_BLOB を挿入して、列を BLOB データ型に初期化できます。例 2 を参照してください。

BLOB 列は、アウト・ラインまたはインラインの LOB 値を参照できる LOB ロケータを含みます。表から LOB を選択すると、実際は LOB のロケータが返され、LOB 値全体が返されるわけではありません。

BLOB は、LONG 型および LONG RAW 型と似ていますが、次の点が異なります。

- BLOB は、ユーザー定義データ型（オブジェクト）の属性にできます。
- BLOB ロケータは、実際の BLOB を伴って、または伴わないで、表列に格納されます。BLOB 値は、別々の表領域に格納できます。
- BLOB 列にアクセスすると、ロケータが返されます。
- BLOB のサイズは、最大 2GB まで可能です。
- BLOB は、データに対して効率的でランダムな個別アクセス、および操作が可能です。
- 1 つの表に複数の BLOB 列を定義できます。
- 1 つのオブジェクト内に、1 つ以上の BLOB 属性を定義できます。
- BLOB バインド変数を宣言できます。
- BLOB 列および BLOB 属性を選択できます。
- 新しい行を挿入するか、1 つ以上の BLOB 列または 1 つ以上の BLOB 属性を持つオブジェクト（あるいはその両方）を含んだ既存の行を更新できます （内部 BLOB 値を、NULL や空に設定したり、BLOB 全体をデータと置換したりできます）。
- BLOB 行 / 列共通部分または BLOB 属性を、別の BLOB 行 / 列共通部分または BLOB 属性で更新できます。
- BLOB 列または BLOB 属性を含む行を削除できます。これで、BLOB 値も削除されます。

内部 BLOB 列（データベースに格納された BLOB 列）の行をアクセスまたは移入するには、最初に INSERT 文を使用して内部 BLOB 値を空に初期化します。

例 1

次の例は、BLOB 列が入った表を作成します。

```
CREATE TABLE  PERSON_TABLE (NAME CHAR(40),
                            PICTURE BLOB);
```

例 2

次の例は、EMPTY_BLOB を挿入して、列を BLOB データ型に初期化します。

```
INSERT INTO PERSON_TABLE (NAME, PICTURE) VALUES ('Steve', EMPTY_BLOB());
```

CHAR

[ODBC] [SQL-92] [Oracle]

固定長の文字列型。CHAR 列は、データベース行内に固定スペースを割り当て、最大長になるようにします。最大長より短い文字列には、後続する空白が埋め込まれます。

構文

```
CHAR
CHARACTER
CHAR ( <length> )
CHARACTER ( <length> )
```

キーワードとパラメータ

<length> は文字列の文字数。上限は 4,096 バイトです。

使用上の注意

<length> を指定しない場合、1 が指定されたものとみなされます。

例

```
CHAR
CHAR(20)
```

CLOB

CLOB データ型は、テキストおよび空間データなど、最大 2GB のサイズまでの構造化されていないラージ・データを格納できます。

構文

```
CLOB
```

使用上の注意

表を作成するとき、オプションで、CLOB 列に異なる表領域および記憶特性を指定できます。

EMPTY_CLOB を挿入して、列を CLOB データ型に初期化できます。例 2 を参照してください。

CLOB 列は、アウト・ラインまたはインラインの LOB 値を参照できる LOB ロケータを含みます。表から LOB を選択すると、実際は LOB のロケータが返され、LOB 値全体が返されるわけではありません。

CLOB は、LONG 型および LONG RAW 型と類似していますが、次の点が異なります。

- CLOB は、ユーザー定義データ型（オブジェクト）の属性にできます。
- CLOB ロケータは、実際の CLOB を伴って、または伴わないで、表列に格納されます。CLOB 値は、別々の表領域に格納できます。
- CLOB 列にアクセスすると、ロケータが返されます。
- CLOB のサイズは、最大 2GB まで可能です。

- CLOB は、データに対して効率的でランダムな個別アクセス、および操作が可能です。
- 1 つの表に複数の CLOB 列を定義できます。
- 1 つのオブジェクト内に、1 つ以上の CLOB 属性を定義できます。
- CLOB バインド変数を宣言できます。
- CLOB 列および CLOB 属性を選択できます。
- 新しい行を挿入するか、1 つ以上の CLOB 列または 1 つ以上の CLOB 属性を持つオブジェクト（あるいはその両方）を含んだ既存の行を更新できます （内部 CLOB 値を、NULL や空に設定したり、CLOB 全体をデータと置換したりできます）。
- CLOB 行 / 列共通部分または CLOB 属性を、別の CLOB 行 / 列共通部分または CLOB 属性で更新できます。
- CLOB 列または CLOB 属性を含む行を削除できます。また、これで CLOB 値も削除されます。

内部 CLOB 列（データベースに格納された CLOB 列）の行をアクセスまたは移入するには、最初に INSERT 文を使用して内部 CLOB 値を空に初期化します。

例 1

次の例は、CLOB 列が入った表を作成します。

```
CREATE TABLE WORK_HISTORY (NAME CHAR (40),
                           RESUME CLOB);
```

例 2

次の例は、EMPTY_CLOB を挿入して、列を CLOB データ型に初期化します。

```
INSERT INTO WORK_HISTORY (NAME, RESUME) VALUES ('Steve', EMPTY_CLOB());
```

DATE

[ODBC] [SQL-92]

SQL-92 および ODBC で日、月および年を格納します。Oracle では、時刻も格納します。

構文

DATE

例

DATE

DECIMAL

[ODBC] [SQL-92]

DECIMAL 数（10 進数）型。

構文

```
DECIMAL [ ( <precision>[, <scale> ] ) ] | DEC [ ( <precision>[, <scale> ] ) ]
```

キーワードとパラメータ

<precision> は小数点数の精度。

<scale> は小数点数のスケール（小数点以下の桁数）。

使用上の注意

DECIMAL は真数値です。DECIMAL データはデフォルトで文字列または SQL_C_CHAR として返されますが、SQL_C_LONG、SQL_C_FLOAT またはその他のデータ型への変換もサポートされています。*<precision>* を指定しない場合、38 が指定されたものとみなされます。*<scale>* を指定しない場合、0 が指定されたものとみなされます。0 <= *<scale>* <= *<precision>* <= 38。

DECIMAL は NUMERIC および NUMBER と同義です。

例

```
DECIMAL
DEC (5)
DECIMAL (10, 5)
```

DOUBLE PRECISION

[ODBC]

倍精度浮動小数点数型。SQL_C_DOUBLE 変数と結合します。

構文

```
DOUBLE PRECISION
```

使用上の注意

DOUBLE PRECISION は仮数小数点精度 15 の、符号付き概数値です。その絶対値は 0、または 10^-308 と 10^308 の間です。

例

```
DOUBLE PRECISION
```

FLOAT

[ODBC]

浮動小数点数型。SQL_C_DOUBLE 変数と結合します。

構文

```
FLOAT [ ( <precision> ) ]
```

キーワードとパラメータ

<precision> は浮動小数点数の精度。

使用上の注意

FLOAT は仮数小数点精度 15 の、符号付き概数値です。その絶対値は 0、または 10^-308 と 10^308 の間です。現在の実装では、FLOAT の精度は常に 15 に設定されています。

例

```
FLOAT
FLOAT (10)
```

INTEGER

[ODBC] [SQL-92]

整数型。

構文

```
INTEGER
INT
```

使用上の注意

INTEGER は精度 10 およびスケール 0 の真数値で、通常は 4 バイトです。SQL_C_LONG または SQL_C_ULONG および SQL_C_SLONG 配列と結合します。-2^31 < n < 2^31。*n* は INTEGER 値です。

例

```
INTEGER
INT
```

LONG

[Oracle]

可変長文字の文字列型。文字列の長さが 4,096 バイトを超える場合に使用されます。

構文

```
LONG
```

キーワードとパラメータ

<length> は文字列の最大文字数。

使用上の注意

LONG の最大長は、20 億バイトです。*<length>* を指定しない場合、2 MB が指定されたものとみなされます。LONG 列上で索引を作成できますが、索引内では最初の 2,000 バイトしか使用されません。

例

```
LONG
```

LONG RAW

[Oracle]

可変長バイナリ・データ型。LONG VARBINARY と似ています。VARBINARY 列が 4,096 バイトを超える場合、この型を使用します。

構文

```
LONG RAW [( <precision> )]
```

キーワードとパラメータ

<precision> バイトの最大値。未指定の場合、デフォルトは 2 MB です。

使用上の注意

LONG RAW の最大長は、20 億バイトです。

例

```
LONG RAW(1048576)
```

LONG VARBINARY

[ODBC]

可変長バイナリ・データ型。

構文

```
LONG BINARY [( <precision> )]
```

キーワードとパラメータ

<precision> バイトの最大値。未指定の場合、デフォルトは 2 MB です。

使用上の注意

1 <= *<precision>* <= 2G。

例

```
LONG VARBINARY(1048576)
```

LONG VARCHAR

[ODBC]

可変長文字の文字列型。文字列の長さが 4,096 バイトを超える場合に使用されます。

構文

```
LONG VARCHAR
LONG VARCHAR ( <length> )
```

キーワードとパラメータ

<length> は文字列の最大文字数。

使用上の注意

LONG VARCHAR の最大長は、20 億バイトです。*<length>* を指定しない場合、2 MB が指定されたものとみなされます。LONG VARCHAR 列に索引を作成できますが、索引内では最初の 2,000 バイトしか使用されません。

例

```
LONG VARCHAR
```

NUMBER

[Oracle]

DECIMAL 数（10 進数）型。

構文

```
NUMBER [ ( <precision>[, <scale> ] ) ]
```

キーワードとパラメータ

<precision> は小数点数の精度。

<scale> は小数点数のスケール（小数点以下の桁数）。

使用上の注意

NUMBER は真数値です。NUMBER データはデフォルトで文字列または SQL_C_CHAR として返されますが、SQL_C_LONG、SQL_C_FLOAT またはその他のデータ型への変換もサポートされています。*<precision>* を指定しない場合、38 が指定されたものとみなされます。*<scale>* を指定しない場合、0 が指定されたものとみなされます。0 <= *<scale>* <= *<precision>* <= 38。

NUMBER は DECIMAL および NUMERIC と同義です。

例

```
NUMBER
NUMBER (10, 5)
```

NUMERIC

[ODBC] [SQL-92]

DECIMAL 数（10 進数）型。

構文

```
NUMERIC [ ( <precision>[, <scale> ] ) ]
```

キーワードとパラメータ

<precision> は小数点数の精度。

<scale> は小数点数のスケール（小数点以下の桁数）。

使用上の注意

NUMERIC は真数値です。NUMERIC データはデフォルトで文字列または SQL_C_CHAR として返されますが、SQL_C_LONG、SQL_C_FLOAT またはその他のデータ型への変換もサ

ポートされています。*<precision>* を指定しない場合、38 が指定されたものとみなされます。*<scale>* を指定しない場合、0 が指定されたものとみなされます。0 <= *<scale>* <= *<precision>* <= 38。

NUMERIC は DECIMAL および NUMBER と同義です。

例

```
NUMERIC
NUMERIC (10, 5)
```

RAW

[Oracle]

可変長バイナリ・データ型。SQL_C_CHAR または SQL_C_BINARY 配列と結合します。

構文

```
RAW [( <precision> )]
```

キーワードとパラメータ

<precision> バイトの最大値。

使用上の注意

RAW は BINARY および VARBINARY と同義ですが、制限は 4,096 バイトです。

例

```
RAW(1024)
```

REAL

[ODBC]

浮動小数点数型。SQL_C_REAL 変数と結合します。

構文

```
REAL
```

使用上の注意

REAL は仮数小数点精度 7 の、符号付き概数値です。その絶対値は 0、または 10^-38 と 10^38 の間です。

例

```
5600E+12
```

ROWID

16 バイトの 16 進文字列で、表内の行の一意のアドレスを表示します。ROWID は主に ROWID 疑似列によって返される値のためのものです。

使用上の注意

Oracle Lite では ROWID は 16 進文字列で、一意のオブジェクト識別子を表します。Oracle の ROWID とは互換性がありませんが、行を一意に識別して更新することはできます。ROWID リテラルは引用符内に囲まれている必要があります。

例

```
A80000.00.03000000
```

SMALLINT

[ODBC] [SQL-92]

小整数型。

構文

```
SMALLINT
```

使用上の注意

SMALLINT は精度 5 およびスケール 0 の真数値で、通常は 2 バイトつまり 16 ビットです。符号付きの場合は、範囲は -32,768 から +32,767（SQL_C_SSHORT または SQL_C_SHORT）または符号なしの場合は、0 から 65,535（SQL_C_USHORT）の間です。-32,768 <= *n* <= 32,767。*n* は SMALLINT 値です。

例

```
SMALLINT
```

TIME

[ODBC] [SQL-92]

時、分、秒および秒の小数部も格納します。

構文

```
TIME
TIME ( <precision> ) [SQL-92]
```

キーワードとパラメータ

<precision> 秒表示の小数部の桁数。

例

```
TIME
TIME (3)
```

TIMESTAMP

[ODBC] [SQL-92]

SQL-92 内で日付と時間の両方を格納し、Oracle DATE データ型に相当します。

構文

```
TIMESTAMP [ ( <precision> ) ]
```

キーワードとパラメータ

<precision> 秒表示の小数部の桁数。0 <= *<precision>* <= 6

使用上の注意

Oracle 表のレプリケーション中、Oracle の DATE 列は Oracle Lite では TIMESTAMP 列として格納されます。

例

```
TIMESTAMP
TIMESTAMP (3)
```

TINYINT

[ODBC]

1 バイト整数型。

構文

```
TINYINT
```

使用上の注意

1 バイト整数で、符号なしの場合（SQL_C_UTINYINT）は 0 から 127 までの範囲内、符号付きの場合（SQL_C_STINYINT）は -128 から +127 までの範囲内です。

例

```
TINYINT
```

VARBINARY

[ODBC]

可変長バイナリ・データ型。SQL_C_CHAR または SQL_C_BINARY 配列と結合します。

構文

```
VARBINARY [( <precision> )]
```

キーワードとパラメータ

<precision> バイトの最大値。

使用上の注意

VARBINARY は BINARY および RAW と同義です。

例

```
VARBINARY(1024)
```

VARCHAR

[ODBC] [SQL-92] [Oracle]

可変長文字の文字列型。

構文

```
VARCHAR ( <length> )
CHAR VARYING ( <length> )
CHARACTER VARYING ( <length> )
```

キーワードとパラメータ

<length> は文字列内の最大文字数で 1 ～ 4,096。

使用上の注意

<length> を指定しない場合、1 が指定されたものとみなされます。

例

```
VARCHAR(20)
CHAR VARYING(20)
CHARACTER VARYING(20)
```

VARCHAR2

[Oracle]

可変長文字の文字列型。VARCHAR および VARCHAR2 は長さが最大長を超えない限り、渡されたとおりに格納されます。空白バイトの埋込みは追加されません。VARCHAR および VARCHAR2 は等価です。

構文

```
VARCHAR2 ( <length> )
CHAR VARYING ( <length> )
CHARACTER VARYING ( <length> )
```

キーワードとパラメータ

<length> は文字列内の最大文字数で 1 ～ 4,096。

使用上の注意

<length> を指定しない場合、1 が指定されたものとみなされます。

例

```
VARCHAR2(20)
CHAR VARYING(20)
CHARACTER VARYING(20)
```

C

Oracle Lite リテラル

Oracle Lite は、次のリテラルをサポートします。

- CHAR、VARCHAR
- DATE
- DECIMAL、NUMERIC、NUMBER
- REAL、FLOAT、DOUBLE PRECISION
- SMALLINT、INTEGER、BIGINT、TINYINT
- TIME
- TIMESTAMP

CHAR、VARCHAR

文字列リテラル値。

構文

```
'<letters>'
```

キーワードとパラメータ

<letters> 改行を除く、0 個以上の印字可能文字のシーケンス。

使用上の注意

引用符がリテラルの一部に含まれている場合、その前に引用符を追加する必要があります（エスケープ文字として使用）。リテラルの最大文字長は、1024 です。

例

```
'a string'
'a string containing a quote '''
```

DATE

日付リテラル値。

構文

```
[DATE] ' <year1 ><month1 ><day >' [SQL-92]
{ d ' <year1 ><month1 ><day >' [ODBC]
--(* d ' <year1 ><month1 ><day >' *)-- [ODBC]
' <day ><month2 ><year2 >' [Oracle]
' <day ><month2 ><year1 >' [Oracle]
' <month1 ><day ><year2 >' [Oracle]
' <month1 ><day ><year1 >' [Oracle]
```

キーワードとパラメータ

<year1> 4 桁の数値で年を表示、たとえば 1994。

<year2> 2 桁の数値で年の最後の 2 桁を表示。

<month1> 01 ～ 12 の 2 桁の数値。

<month2> 3 文字の頭文字で月を表示（大文字小文字を区別しない）。

<day> 01 ～ 31 の 2 桁の数値 (月による)。

例

```
'1994-11-07' [SQL-92]
{ d '1994-11-07' }
--(* d '1994-11-07' *)--
DATE '10-23-94'
'23-Nov-1994' [Oracle]
'23-Nov-94'
```

DECIMAL、NUMERIC、NUMBER

10 進数リテラル値。

構文

```
[+|- ]<digits>
[+|- ]<digits>.[<digits>]
[+|- ].<digits>
```

キーワードとパラメータ

<digits> 1 桁以上の数字のシーケンス

例

```
54321
-123.
+456
64591.645
+.12345
0.12345
```

REAL、FLOAT、DOUBLE PRECISION

浮動小数点数リテラル値。

構文

```
[+|- ]<digits ><exp >[+|- ]<digits >
[+|- ]<digits >. [<digits >]<exp >[+|- ]<digits >
[+|- ].<digits ><exp >[+|- ]<digits >
```

キーワードとパラメータ

<digits> 1 桁以上の数字のシーケンス

<exp>「E」または「e」。

例

```
+1.5e-7
12E-5
-.12345e+6789
```

SMALLINT、INTEGER、BIGINT、TINYINT

[ODBC]

整数リテラル値。

構文

[+|-]<*digits*>

キーワードとパラメータ

<*digits*> 1 桁以上の数字のシーケンス

使用上の注意

n を、リテラルで表される数とします。

TINYINT の場合、-128 <=*n* <= 127

SMALLINT の場合、-32768 <= *n* <= 32767

INTEGER の場合、-2^31 < *n* < 2^31

BIGINT の場合、-10^19 < *n* < 10^19

例

```
12345
```

TIME

時間リテラル値。

構文

[TIME]' <*hour*>:<*minute*>:<*second*>[.[<*fractional_second*>]]'

キーワードとパラメータ

<*hour*> 00 ～ 23 の 2 桁の数。

<*minute*> 00 ～ 59 の 2 桁の数。

<second> 00 ～ 59 の 2 桁の数。

<fractional_second> 6 桁までの数値。

例

```
'23:00:00'
TIME '23:00:00.'
TIME '23:01:59.134343'
```

TIMESTAMP

タイム・スタンプ・リテラル値。

構文

```
TIMESTAMP ' <DATE_literal_value > <TIME_literal_value >'
```

キーワードとパラメータ

<DATE_literal_value> 日付リテラル。

<TIME_literal_value> 時間リテラル。

使用上の注意

タイム・スタンプ・リテラルでは、日付リテラルと時間リテラルとの間に 1 文字のみ空白があります。

例

```
TIMESTAMP '1994-11-07 23:00:00'
'94-06-01 12:02:00'
Examples: CHAR (10)
```

D

SQL*Plus

SQL*Plus の概要

SQL*Plus は、Oracle Lite または、リモート Oracle7 または Oracle8 データベースに対して SQL コマンドを実行するためのコマンドライン・インタフェースを提供する汎用ツールです。これには、出力書式に関する豊富な機能が含まれており、データベースから単純なレポートを作成するために使用できます。Oracle Lite は、SQL*Plus バージョン 3.3 または 8.0 が必要です。

SQL*Plus の実行

SQL*Plus は、「Oracle for Windows 95」（または「Oracle for Windows 98」や「Oracle for Windows NT」）プログラム・グループ内のアイコンをクリックして起動できます。ダイアログ・ボックスに、ユーザー名、パスワードおよび接続文字列の入力を求めるプロンプトが表示されます。リモート Oracle サーバー・データベースへの接続文字列は、一般に、OracleNet8 Easy Configuration ユーティリティを使用して、**TNSNAMES.ORA** ファイル内に構成する別名です。

Oracle Lite 初期データベースに接続するには、接続文字列 `ODBC:POLITE` を使用します。EXDB などのデータベースに接続するには、接続文字列 `ODBC:EXDB` を使用します。EXDB データベースに、SYSTEM 以外のユーザーとして接続する場合、最初に、ユーザーを POLITE データベース内に作成し、その後、EXDB データベース内に作成する必要があります。

別のデータ・ソースに接続するには、次を使用します。

```
ODBC:data_source_name
```

data_source_name は、ODBC Administrator を使用して作成するデータ・ソースの名前です。この接続文字列は、Oracle8 Navigator から作成されたデータベースに使用できません。`ODBC:` によって、SQL*Plus は、接続の際に Oracle Net8 を使用せず、Oracle Open Client Adapter（OCA）を使用してユーザーの ODBC データベースにアクセスすることを認識します。

SQL*Plus バージョン 8.0 は、DOS コマンドラインから次の 2 つの形式で起動できます。

- **PLUS80W.exe**（ウィンドウを開きます）
- **PLUS80.exe**（プログラムを DOS ウィンドウ内で実行します）

スクリプトの実行やレポートの実行を自動化する引数をコマンドラインに入れることができます。引数はすべてオプションです。コマンドラインの完全な構文は次のとおりです。

```
C:¥> PLUS80[W] username/password @connect_string @script.sql parameter1
    parameter2 ...
```

username は、Oracle Lite データベースに対しては、一般的に、SYSTEM です。

password は、Oracle Lite データベースのユーザー SYSTEM の場合は無視されますが、空白にしておくことはできません。

SYSTEM 以外のユーザーの場合は、CREATE USER または ALTER USER コマンドで生成されたパスワードと同じにしてください。

connect_string は、リモート・ホストの Oracle Net8 別名か、`ODBC:data_source_name` のどちらかです。

script.sql は、SQL コマンドのファイルへのパスです。

parameter1 および *parameter2* は、スクリプトの中で相互に置き換えることができる単一語か数値引数です。

設定コマンド

SQL*Plus では、SET コマンドで設定されるいくつかの内部パラメータを使用できます。これは、Oracle7 または Oracle8 の SQL コマンドではありません。そのため、SET SCHEMA や SET CATALOG など Oracle Lite が使用する SET コマンドのいくつかは、SQL*Plus によってインターセプトされ、Oracle Lite データベースには渡されません。

SQL*Plus の終了

SQL*Plus は、`SQL>` コマンド・プロンプトで、「`EXIT`」または「`QUIT`」と入力して、終了することができます。終了の前に、変更をすべてコミットする必要があります。

E

索引作成オプション

前のリリースでは、Oracle Lite は、表のすべての列に一意の索引を作成して、一連の列の一意性を施行していました。この方法では、長いキーまたは多数の列を含むキーの場合に大量のディスク領域が必要でした。

Oracle Lite での一意制約

Oracle Lite では、アプリケーションは、大量のディスク領域を必要とせずに多数の列に一意制約を施行できます。これは、多くの列に一意制約が必要ではあるが、これらの列の小さいサブセットに同じ値がある行を持つ表列のアプリケーションに役に立ちます。

アドレス表の例

この項の説明と例は、すべて次の表を参照します。

```
ADDRESS (STREET VARCHAR(40), CITY VARCHAR(40), STATE VARCHAR(20), ZIP VARCHAR(12));
```

一意制約の使用

ADDRESS 内に、行の各列の値が別の行のものとまったく同じ行がないように、一意制約を施行する場合、表のすべての列に一意索引を作成できます。しかし、この方法は、大量のディスク領域を必要とします。

STREET と CITY 列に同じ値が入った行が非常に少ない場合は、STREET と CITY にのみ一意索引を作成できます。2 つの行が STREET 列と CITY 列に同じ値を持つ場合、Oracle Lite は、残りの列が一意であるかテストします。

この方法はディスク領域は少なくて済みますが、いくつか欠点があります。データベースは、一意または主キー列が同じでないことを保証するために、すべてのレコードを検索する必要があるため、次のアクションでデータベースのパフォーマンスが低下します。

- 行の挿入と更新
- 主キーをベースにした行の問合せ

Oracle Lite が、同じ値を持つ索引列をロックすると、これらの列に複数のデータベース・ユーザーが同時にアクセスできません。

索引内の列数の指定

索引内の列の数を、**POLITE.INI** ファイル、または、次の SQL 文のいずれかで指定できます。

- CREATE INDEX
- CREATE TABLE
- ALTER TABLE

POLITE.INI ファイル

POLITE.INI ファイル内の MAXINDEXCOLUMNS 値は、索引内の列の最大数を指定します。ユーザーが新しい索引を作成すると、索引には、MAXINDEXCOLUMNS 変数に指定された数の列のみが含められます。たとえば、**POLITE.INI** ファイルの次の行は、新たに作成される索引に、それが参照する表の最初の 2 列を含む必要があることを指定します。

```
MAXINDEXCOLUMNS=2
```

この例を ADDRESS 表に適用すると、次の文は、STREET 列と CITY 列を含む索引を作成します。

```
CREATE INDEX IDX1 ON ADDRESS(STREET, CITY, STATE, ZIP);
```

また、次の文は、STREET 列と CITY 列を含んだ一意索引を作成します。

```
CREATE UNIQUE INDEX IDX1 ON ADDRESS(STREET, CITY, STATE, ZIP);
```

文に UNIQUE 句が入っているため、Oracle Lite は、指定された列のすべてを一意キーと指示します。

CREATE UNIQUE INDEX 文

Oracle Lite では、CREATE UNIQUE INDEX 文には、索引列の数を指定する次のオプションの句が含まれます。

KEY COLUMNS = <*number_of_columns*>

Oracle Lite は、KEY COLUMNS 句に指定した数の列を含む索引を作成します。たとえば、次の文は、STREET と CITY の 2 つの列を含む索引を作成します。

```
CREATE UNIQUE INDEX IDX1 ON ADDRESS (STREET, CITY, STATE, ZIP) KEY COLUMNS = 2;
```

文に UNIQUE 句が入っているため、Oracle Lite は、指定された列のすべてを一意キーと指示します。

CREATE TABLE および ALTER TABLE 文

CREATE TABLE および ALTER TABLE 文内の PRIMARY KEY 句は、索引列の数を指定する次の句をサポートします。

KEY COLUMNS = <*number_of_columns*>

次の例は、表を作成し、その 4 列を主キーと指示します。しかし、主キーを施行する索引は、最初の 2 列のみを含みます。

```
CREATE TABLE ADDRESS (STREET VARCHAR(40), CITY VARCHAR(40), STATE VARCHAR(20), ZIP
VARCHAR(12), PRIMARY KEY(STREET, CITY, STATE, ZIP)  KEY COLUMNS = 2);
```

使用上の注意

POLITE.INI ファイルに `MAXINDEXCOLUMNS` 変数が含まれていなくて、SQL 文が KEY COLUMNS オプションを使用していない場合、Oracle Lite は、指定されたすべての列を使用して索引を作成します。

POLITE.INI ファイルが `MAXINDEXCOLUMNS` 値を指定している場合、SQL 文の KEY COLUMNS 句でオーバーライドされない限り、Oracle Lite は、この値を使用してすべての索引と主キーを作成します。

F

構文図の規則

この付録では、『Oracle Lite SQL リファレンス』で使用される構文図について説明します。説明する内容は次のとおりです。

- 概要
- 必須のキーワードとパラメータ
- オプションのキーワードとパラメータ
- 構文ループ
- マルチパート図
- データベース・オブジェクト

概要

構文図は、有効な SQL 構文を示した図です。図は、矢印で示された方向に、左から右に読み進みます。

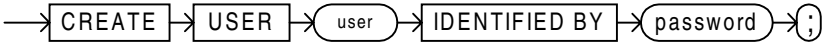
コマンドや他のキーワードは、四角形の中に大文字で表示されます。これは、四角形の中に示されたとおりに入力します。パラメータは、楕円の中に小文字で表示されます。変数は、パラメータで使用されます。句読点、演算子、デリミタおよび終了記号は、円の中に表示されます。

構文図に 2 つ以上のパスがある場合、どのパスでも選択できます。

キーワード、演算子またはパラメータで、2 つ以上の選択肢がある場合、オプションは垂直にリスト表示されます。

必須のキーワードとパラメータ

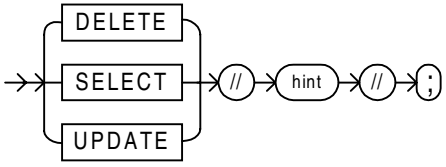
必須のキーワードおよびパラメータは、単独で、または選択肢の垂直リストとして表示されます。単一の必須のキーワードまたはパラメータは、メイン・パス内、すなわち、現在進んでいる水平の線上に表示されます。次の図で、*user* および *password* は、必須パラメータです。



この構文図によれば、次の構文は有効です。

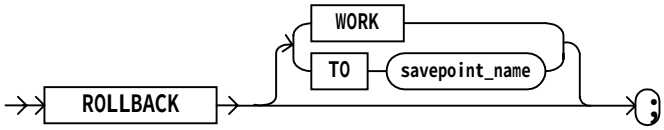
```
CREATE USER hannibal IDENTIFIED BY hanna;
```

次の図では、DELETE、SELECT、または UPDATE が必須パラメータです。



オプションのキーワードとパラメータ

キーワードおよびパラメータがメイン・パスから外れた垂直リスト内に表示された場合、オプションです。次の例で、オプションの垂直リストから選択することも、メイン・パスに沿って進むこともできます。

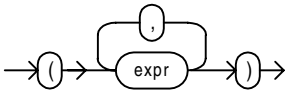


構文図によると、次の文は有効です。

```
ROLLBACK WORK;
ROLLBACK TO savepoint_1;
ROLLBACK;
```

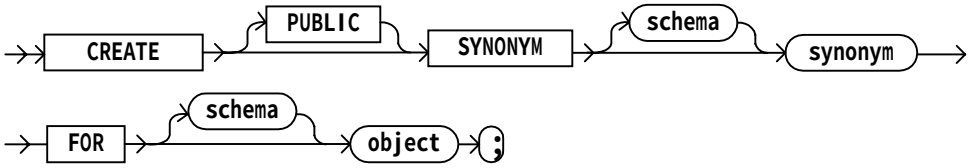
構文ループ

ループによって、構文内で何度でもその構文を使用できます。次の例で、1 つの式を選択した後、もう一度戻って、カンマで区切られた別の式を繰り返し選択できます。



マルチパート図

マルチパート図は、すべてのメイン・パスの端と端が接続されているかのように読みます。次の例は、2 つのパートからなる図です。



この構文図によると、次の文は有効です。

```
CREATE SYNONYM prod FOR product;
CREATE SYNONYM prod FOR scott.product;
CREATE SYNONYM scott.prod FOR scott.product;
```

データベース・オブジェクト

たとえば表や列のような Oracle Lite 識別子の名前は、長さが 30 文字を超えることはできません。先頭は、文字にしてください。残りは、文字、数字、ドル記号（$）、ポンド記号（#）およびアンダースコア（_）を自由に組み合せることができます。

しかし、Oracle Lite 識別子が二重引用符（"）で囲まれている場合、引用符以外は、スペースも含めて、有効文字を自由に組み合せることができます。Oracle Lite 識別子は、二重引用符に囲まれている場合以外、大小文字は区別しません。

用語集

SQL

SQL（構造化問合せ言語）は、リレーショナル・データベース・エンジンのほとんどで使用される非手続き型データベース・アクセス言語です。SQL 文はデータ・セットに対して実行される操作を記述します。SQL 文がデータベースに送られると、データベース・エンジンは指定されたタスクを実行するプロシージャを自動的に生成します。

一意キー

表の一意キーは、表の各列での一意の列または列グループです。UNIQUE KEY 制約を満たすには、一意キーの値が表の複数の行に出現することはできません。ただし、PRIMARY KEY 制約とは異なり、単一列からなる一意キーは NULL 値を含むことができます。

位置づけ DELETE

位置づけ DELETE 文により、カーソルのカレント行が削除されます。書式は以下のとおりです。

```
DELETE FROM table
   WHERE CURRENT OF cursor_name
```

位置づけ UPDATE

位置づけ UPDATE 文により、カーソルのカレント行が更新されます。書式は以下のとおりです。

```
UPDATE table SET set_list
   WHERE CURRENT OF cursor_name
```

外部キー

外部キーとは、表またはビューに存在する列または列グループのことで、その値は別の表またはビューに存在する行を参照します。外部キーには一般に、別の表の主キー値と一致する値が含まれます。

結合

2 つの異なる表またはビューに存在するキー（主キーと外部キーの両方）の間に確立された関係です。結合は、リレーショナル・データベース内の重複したデータを排除するために正規化された表のリンクに使用します。結合リンクの一般的なものとしては、1 つの表の主キーを別の表の外部キーにリンクして、マスター・ディテール・リレーションを確立するものがあります。結合は SQL 文の WHERE 句条件に対応します。

索引

索引は、表内のそれぞれの行に対する高速アクセスを提供するデータベース・オブジェクトです。索引を作成し、表のデータに対して実行される問合せおよびソート操作を高速化できます。また、索引を使用して、UNIQUE KEY 制約や PRIMARY KEY 制約などの制約を表に対して実行することもできます。

索引はいったん作成されると自動的にメンテナンスされ、データベース・エンジンにより可能な限りデータ・アクセスのために使用されます。

参照整合性

参照整合性はレコードが追加、修正または削除された時にメンテナンスされるマスター・ディテール・リレーション内の表間のリンクの精度として定義されます。

マスター・ディテール・リレーションを注意深く定義しておくことにより、参照整合性が高まります。データベース内に制約を適用すると、データベース（クライアント / サーバー環境でのサーバー）レベルの参照整合性が強化されます。

参照整合性の目的は、孤立したレコード（マスター・レコードとの有効なリンクを持たないディテール・レコード）が作成されないようにすることです。参照整合性の実施規則により、結果として孤立したレコードを作成するような、マスター・レコードの削除や更新、またはディテール・レコードの挿入や更新を予防できます。

シノニム

シノニムとは、表、ビュー、順序、スナップショットまたは別のシノニムに対する代替名（エイリアス）のことです。

主キー

表の主キーは、表内の各行を一意に識別するのに使用される 1 つの列、または列グループのことです。主キーを使うと表のレコードにすばやくアクセスでき、また主キーは 2 つの表またはビューの間の結合の基礎として頻繁に使用されます。それぞれの表に対して主キーは 1 つしか定義できません。

PRIMARY KEY 制約を満たすには、主キー値が表の 2 列以上で使われたり、主キーの部分である列に NULL 値が含まれていないようにします。

順序

順序は、他と重複しない一連の整数を生成するデータベース・オブジェクトです。順序は一般に主キー値などのように、他と重複してはいけないデータ値を生成するために使用します。

スキーマ

スキーマとは、表、ビュー、索引、順序などを含む、名前の付いたデータベース・オブジェクトの集まりです。

整合性制約

整合性制約とは、表の 1 つ以上の列に入力できる値を制限する規則です。

データベース・オブジェクト

データベース・オブジェクトは、表、ビュー、順序、索引、スナップショットまたはシノニムなどの名前のつけられたデータベース構造体です。

トランザクション

リレーショナル・データベース内の選択されたデータに対して加えられる一連の変更。トランザクションは通常、ADD、UPDATE、DELETE などの SQL 文を使用して実行します。トランザクションは、コミットされた（変更が永続的になる）とき、またはロールバックされた（変更が破棄された）ときに完了します。

トランザクションの前に問合せが実行されることがよくあります。問合せを使用して、変更対象の特定のレコードをデータベースから選択しておきます。

ビュー

ビューは、1 つ以上の表（または他のビュー）から選択されたデータをカスタマイズして表したものです。ビューは「仮想的な表」のようなもので、複数の表（ベース表と呼ばれます）およびビューからのデータを関連させ、また組み合わせることができます。ビューは表示されるデータの選択条件を指定できるため、一種の「格納された問合せ」といえます。

ビューは、表のように、行と列に編成されます。しかし、ビューには、データそのものは含まれません。ビューを使用すると、複数の表またはビューを 1 つのデータベース・オブジェクトとして扱うことができます。

表

表は、行と列に編成されたデータを格納するデータベース・オブジェクトです。上手に設計されたデータベースでは、各表に単一のトピックに関する情報（たとえば従業員や顧客の住所など）が格納されます。

ベース表

ビューの基になるデータのソースで、表またはビューのことです。ビュー内のデータにアクセスするとき、実際はベース表のデータにアクセスしています。ビューのベース表は、CREATE VIEW で指定します。

マスター・ディテール・リレーション

1 つの表またはビュー（ディテール表またはビュー）の複数の行が、別の表またはビュー（マスター表またはビュー）の単一のマスター行に関連付けられている場合に、マスター・ディテール・リレーションがデータベース内の表またはビューの間に存在すると言います。

マスター行およびディテール行は通常、ディテール表またはビュー内の外部キー列と一致するマスター表またはビュー内の主キー列により結合されます。

主キーの値を変更した場合、アプリケーションでは外部キーの値が主キーの値と一致するように、一連の新しいディテール・レコードを問い合せる必要があります。たとえば、EMP 表内のディテール・レコードが、DEPT 表内のマスター・レコードと同期される場合、DEPT 内の主キーは DEPTNO で、EMP 内の外部キーは DEPTNO にします。

索引

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

え

か

き

く

こ

し

せ

た

ち

て

と

は

ひ

ふ

へ

め

も

よ

り

れ