Oracle8 Server

SQL リファレンス : Vol.1

リリース 8.0

1998 年 2 月

部品番号 A56823-1

ORACLE®

The Database for Network Computing™

Oracle8 Server SQL リファレンス Vol.1、リリース 8.0

部品番号：A56823-1

第 1 版：1998 年 2 月

原本名：Oracle8 SQL Reference, Release 8.0

原本部品番号：A58240-01

原本著者：Diana Lorentz

協力者：Steve Bobrowski, Robert Jenkins, Susan Kotsovolos, AndreKruglikov, Vishu Krishna, Muralidhar Krishnaprasa, Michael Kung, PaulLane, Nina Lewis, Lefty Leverenz, Phil Locke, William Maimone, MohammadMonajjemi, Rita Moran, Thomas Portfolio, Valarie Moore, Denis Raphaely,Richard Sarwal, Rick Wong

Printed In Japan

目 次

第 3 章　演算子、関数、式、条件

第 4 章　コマンド

はじめに

このマニュアルでは、Oracle のデータベース内の情報を管理するために使用される構造化問合せ言語 (SQL) について詳しく説明します。

Oracle SQL は、米国規格協会 (ANSI) と国際標準化機構 (ISO) の SQL92 規格に基本レベルで準拠していますが、さらに多くの内容を盛り込んでいます。

SQL を拡張したオラクル社のプロシージャ型言語である PL/SQL については、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

Oracle の埋込み SQL については、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』および『SQL*Module for Ada Programmer's Guide』、『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照してください。

特徴と機能

『Oracle8 Server SQL リファレンス』には、Oracle8 および Oracle8 Enterprise Edition 製品の特徴と機能の説明があります。Oracle8 および Oracle8 Enterprise Edition の基本的な機能は同じです。ただし、Enterprise Edition だけで利用可能な拡張機能がいくつかあります。また、さらにオプションが必要な場合もあります。たとえば、CREATE TYPE コマンドを使用する場合には、Enterprise Edition と Object Option を持っていなければなりません。

Oracle8 と Oracle8 Enterprise Edition の違いや、ご使用の Oracle で利用できる機能とオプションについては、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

対象読者

このマニュアルは、Oracle SQL のすべてのユーザーを対象とします。

このマニュアルの構成

このマニュアルの構成は次のとおりです。

第 1 部

第 1 章「概要」

SQL について定義し、その歴史と SQL を使用したリレーショナル・データベースへのアクセスの利点について説明します。

第 2 章「Oracle8 SQL の要素」

Oracle データベースと Oracle SQL の要素の基本的な構成ブロックについて説明します。

第 3 章「演算子、関数、式、条件」

SQL の演算子と関数を、式や条件の中で組み合せて使用する方法について説明します。

第 2 部

第 4 章「コマンド」

すべての SQL コマンドを、アルファベット順に説明します。

付録 A「構文図」

このマニュアルで使用する構文図の読み方を説明します。

付録 B「Oracle と標準 SQL」

ANSI および ISO の規格に対する Oracle の準拠性について説明し、これらの標準規格を拡張した部分を示します。

付録 C「Oracle の予約語とキーワード」

Oracle の予約語とキーワードを示します。

このマニュアルで使用する表記上の規則

このマニュアルで使用する表記上の規則について説明します。項目は次のとおりです。

- 本文
- 構文図と表記法
- コード例
- 例題のデータ

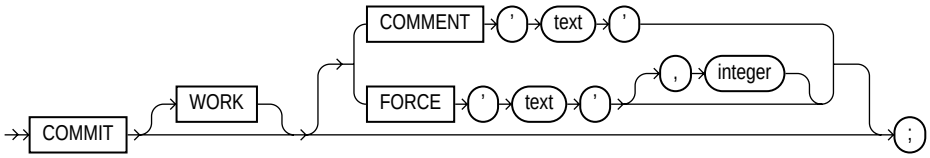
本文

このマニュアルの本文は、次の規則に従って記述されています。

大文字	アルファベットの大文字は、SQL コマンド、キーワード、ファイル名、初期化パラメータを示しています。
イタリック	イタリック体の文字は、用語の定義と SQL コマンドのパラメータを示しています。

構文図と表記法

構文図 . このマニュアルでは、第 4 章「コマンド」における SQL コマンドの説明や、第 2 章「Oracle8 SQL の要素」および第 3 章「演算子、関数、式、条件」におけるその他の SQL 言語の要素の説明に、構文図を使用します。これらの構文図では、次のように、線と矢印を使用して構文構造を示しています。



このタイプの構文図になじみが薄い場合は、付録 A「構文図」を参照して、読み方を理解してください。この項では、構文図のコンポーネントと、SQL 文の書き方の例を示します。構文図は次の項目で構成されます。

キーワード . キーワードは、SQL 言語の中では特別な意味を持っています。このマニュアルの構文図では、キーワードは大文字で記述されています。大文字小文字の違いを除いて、キーワードは構文図に表記されているとおりに SQL 文で使用しなければなりません。たとえば、CREATE TABLE 文では、CREATE TABLE 構文図に表記されているとおり、CREATE キーワードを使用して文を開始する必要があります。

パラメータ . パラメータは、構文図の中でプレースホルダの役目を果たします。パラメータは、小文字で記述されます。パラメータは、通常はデータベース・オブジェクト名または Oracle データ型名、式です。構文図にパラメータがある場合、実際の SQL 文では、そのパラメータを適切な型のオブジェクトまたは式で置き換えます。たとえば、CREATE TABLE 文を作成する場合、構文図の table パラメータのかわりに、作成する表の名前 (たとえば EMP) を使用します。パラメータ名は、本文中ではイタリック体で記述されています。

このマニュアルの構文図で使用されるパラメータと、各文でそのパラメータに代入する値の例を次に示します。

パラメータ	説明	例
table	パラメータによって指定された型のオブジェクト名で置き換えなければなりません。オブジェクト型の一覧は「スキーマ・オブジェクト」（2-40 ページ）を参照してください。	`emp`
c	使用しているデータベースのキャラクタ・セットの単一文字で置き換えなければなりません。	`T` `s`
'text'	一重引用符で囲んだテキスト文字列で置き換えなければなりません。text' の構文については、「Text(テキスト)」（2-2 ページ）を参照してください。	`'Employee records'`
char	データ型 CHAR または VARCHAR2 の式か、一重引用符で囲んだ文字リテラルで置き換えなければなりません。	`ename` `'Smith'`
condition	TRUE または FALSE に評価される条件で置き換えなければなりません。condition の構文については、「条件」（3-84 ページ）を参照してください。	`ename > 'A'`

date
d
	日付定数または DATE データ型の式で置き換えなければなりません。	`TO_DATE(`
		`'01-Jan-1994',`
		`'DD-MON-YYYY')`

expr	「式」（3-73 ページ）にある expr の構文の説明で定義されている任意のデータ型の式で置き換えることができます。	`sal + 1000`
integer	「Integer(整数)」（2-3 ページ）にある integer の構文の説明で定義されている整数で置き換えなければなりません。	`72`

パラメータ	説明	例
label	MLSLABEL データ型の式で置き換えなければなりません。このような式については、使用している Trusted Oracle のマニュアルを参照してください。	`TO_LABEL( 'SENSITIVE:ALPHA')`
number *m* *n*	NUMBER データ型の式、または「Number (数)」（2-3 ページ）にある number の構文の説明で定義されている数値定数で置き換えなければなりません。	`AVG(sal)` `15 * 7`
raw	RAW データ型の式で置き換えなければなりません。	`HEXTORAW('7D')`
rowid	ROWID データ型の式で置き換えなければなりません。	`AAAAZzAABAAABrXAAA`
subquery	他の SQL 文の中で使われる SELECT 文で置き換えなければなりません。「副問合せ」（4-525 ページ）を参照してください。	`SELECT ename FROM emp`
:host_variable	埋込み SQL プログラムで宣言されている変数の名前で置き換えなければなりません。このマニュアルでは、特定のデータ型を表すために、:host_integer と :d も使用しています。	`:employee_number`
cursor	埋込み SQL プログラム内のカーソルの名前で置き換えなければなりません。	`curs1`
db_name	埋込み SQL プログラム内のデフォルト以外のデータベースの名前で置き換えなければなりません。	`sales_db`
db_string	Net8 データベース接続のデータベース識別文字列で置き換えなければなりません。詳細は、使用している Net8 プロトコルのユーザーズ・ガイドを参照してください。	
statement_name *block_name*	SQL 文または PL/SQL ブロックに対する識別子で置き換えなければなりません。	`s1` `lab1`

コード例

このマニュアルには、多数の SQL 文の例を示しています。これらの例は、SQL 文の要素の使用方法を示しています。CREATE TABLE 文の例を次に示します。

```
CREATE TABLE accounts
    ( accno NUMBER,
      owner VARCHAR2(10),
```

```
      balance NUMBER(7,2) );
```

例は、本文と異なるフォントで表記されています。

次の規則に従って、例では大文字と小文字を区別して使用しています。

- CREATE や NUMBER などのキーワードは大文字で表記する。
- ACCOUNTS や ACCNO などのデータベースのオブジェクトやその一部の名前は小文字で表記する。ただし、本文では大文字で表記されています。

SQL では（引用符で囲まれた識別子を除いて）大文字と小文字は区別されないため、実際の SQL 文を作成するときに、これらの規則に従う必要はありませんが、このように区別しておくことで文が読みやすくなります。

Oracle Tools によっては、SQL 文を特殊文字で終了させる必要があります。たとえば、このマニュアルで示すコード例は SQL*Plus で書かれているので、セミコロン (;) で終了しています。しかし、これらの例文を使う場合には、使用している Oracle Tool ごとに規定された特殊文字を使って文を終了させなければなりません。

例題のデータ

このマニュアルの例題の多くでは、サンプルの表に対して操作を行います。これらの表の一部は、配布メディアに収録されている SQL スクリプトで定義されています。多くのオペレーティング・システムで、このスクリプトの名前は UTLSAMPL.SQL となっていますが、正確な名前と位置はオペレーティング・システムによって異なります。このスクリプトでは、サンプルのユーザーを作成し、ユーザー SCOTT のスキーマに次のサンプル表を作成します。

```
CREATE TABLE dept
    (deptno NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
     dname VARCHAR2(14),
     loc       VARCHAR2(13) );
CREATE TABLE emp
    (empno NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
     ename VARCHAR2(10),
     job VARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal NUMBER(7,2),
     comm NUMBER(7,2),
     deptno    NUMBER(2)    CONSTRAINT fk_deptno REFERENCES dept );
CREATE TABLE bonus
    (ename VARCHAR2(10),
     job VARCHAR2(9),
     sal NUMBER,
     comm      NUMBER );
CREATE TABLE salgrade
    (grade NUMBER,
```

```
     losal NUMBER,
     hisal     NUMBER );
```

このスクリプトはサンプル表に次のデータを挿入します。

```
SELECT * FROM dept

DEPTNO  DNAME      LOC
------- ---------- ---------
10      ACCOUNTING NEW YORK
20      RESEARCH   DALLAS
30      SALES      CHICAGO
40      OPERATIONS BOSTON

SELECT * FROM emp

EMPNO  ENAME    JOB           MGR HIREDATE     SAL    COMM  DEPTNO
-----  -------  --------- ------ --------- ------ ------ -------
 7369  SMITH    CLERK       7902 17-DEC-80    800              20
 7499  ALLEN    SALESMAN    7698 20-FEB-81   1600     300      30
 7521  WARD     SALESMAN    7698 22-FEB-81   1250     500      30
 7566  JONES    MANAGER     7839 02-APR-81   2975              20
 7654  MARTIN   SALESMAN    7698 28-SEP-81   1250    1400      30
 7698  BLAKE    MANAGER     7839 01-MAY-81   2850              30
 7782  CLARK    MANAGER     7839 09-JUN-81   2450              10
 7788  SCOTT    ANALYST     7566 19-APR-87   3000              20
 7839  KING     PRESIDENT        17-NOV-81   5000              10
 7844  TURNER   SALESMAN    7698 08-SEP-81   1500              30
 7876  ADAMS    CLERK       7788 23-MAY-87   1100              20
 7900  JAMES    CLERK       7698 03-DEC-81    950              30
 7902  FORD     ANALYST     7566 03-DEC-81   3000              20
 7934  MILLER   CLERK       7782 23-JAN-82   1300              10

SELECT * FROM salgrade

GRADE  LOSAL  HISAL
-----  -----  -----
1      700    1200
2      1201   1400
3      1401   2000
4      2001   3000
5      3001   9999
```

スクリプトのすべての操作を実行するには、ユーザー SYSTEM として Oracle にログインしているときにスクリプトを実行してください。

1

概要

構造化問合せ言語 (SQL) は、Oracle データベース内のデータにアクセスするために、すべてのプログラムとユーザーが使用しなければならない一連のコマンドです。アプリケーション・プログラムや Oracle のツールを使用すれば、SQL を直接使用しなくてもデータベースにアクセスできることが少なくありません。しかし、アプリケーションがユーザーの要求を実行するときには必ず SQL を使用します。この章では、多くのリレーショナル・データベース・システムに採用されている SQL の背景について説明します。説明する内容は次のとおりです。

- SQL の歴史
- SQL の規格
- 埋込み SQL
- 字句規則
- サポートするツール

SQL の歴史

1970 年 6 月に ACM(Association of Computer Machinery) が刊行した「*Communications of the ACM*」誌で、E. F. Codd 博士の論文「大型共用データ・バンク用のデータのリレーショナル・モデル」が発表されました。Codd 博士のモデルは、現在ではリレーショナル・データベース管理システム (RDBMS) の完成したモデルとして受け入れられています。構造化英語問合せ言語 (SEQUEL) は、IBM 社が Codd 博士のモデルを使用するために開発したものです。この SEQUEL が後の SQL です (発音は "SEQUEL" と同じです)。1979 年、Relational Software, Inc.(現在のオラクル社) は商業的に利用可能な最初の SQL の処理系を導入しました。今日、SQL は標準の RDBMS 言語として受け入れられています。

SQL の規格

Oracle SQL は、業界標準に準拠しています。オラクル社は、SQL 標準化委員会の主要メンバーとして積極的に標準化の活動に関わっていくことで、発展し続ける SQL 規格に今後も対応していきます。業界で認知されている委員会には、米国規格協会 (ANSI) およびは国際

電気標準会議 (IEC) が電気・電子部門を担当している国際標準化機構 (ISO) があります。ANSI と ISO/IEC はともに、SQL をリレーショナル・データベースの標準言語として認めています。新しい SQL 規格がこの両機関から同時に発表された場合、その規格の名前は、各機関の規則に従って命名されますが、技術的な詳細はまったく同じです。

ANSI および ISO が発表した最新の SQL 規格は、SQL-92(または SQL2) と呼ばれます。新規格の正式名称は、次のとおりです。

- ANSI X3.135-1992、「Database Language SQL」
- ISO/IEC 9075:1992、「Database Language SQL」

SQL-92 では、規格への準拠に基本、変換、中間、完全の 4 レベルを定義しています。SQL 規格に準拠する処理系は、少なくとも基本レベルの SQL に準拠していなければなりません。Oracle8 リリース 8.0 は、基本レベルの SQL に完全に準拠していますが、変換レベルまたは中間レベル、完全レベルに準拠する多くの機能もあります。

Oracle8 が基本レベルの SQL-92 に準拠していることは、米国連邦情報・技術局 (NIST) が連邦情報処理標準 (FIPS) の FIPS PUB 127-2 を用いて試験済みです。

> **追加情報**： Oracle および SQL の詳細は、付録 B「Oracle と標準 SQL」を参照してください。

SQL の特長

SQL はアプリケーション・プログラマ、データベース管理者、管理職、エンド・ユーザーなど、あらゆる分野のユーザーに利益をもたらします。技術的な言い方をすれば、SQL はデータ副言語です。つまり、SQL の目的は、Oracle のようなリレーショナル・データベースとのインタフェースを提供することであり、すべての SQL 文はデータベースに対する命令です。この点において、SQL は、C や BASIC のような汎用プログラミング言語と異なります。SQL には、次のような特長があります。

- 個々の単位としてではなくグループとして一連のデータを処理。
- データへの自動的なナビゲーション・アクセス（経路設定）の実行。
- SQL で使用する文は、それぞれが複雑、強力で、スタンド・アロン型の文である。これまで、SQL にはフロー制御文は含まれていませんでしたが、SQL, ISO/IEC 9075-5 : 1996 の最近受け入れられたオプション部分にはフロー制御文が含まれています。フロー制御文は、永続保存モジュール (PSM) としてよく知られており、SQL の拡張版である Oracle の PL/SQL は PSM に類似しています。

SQL では、データを論理的なレベルで処理できます。処理系について考えるのは、データの細部を操作する場合だけで済みます。たとえば、表から一連の行を検索するには、行をフィルタ処理するための条件を定義します。この条件を満たす行のすべてが 1 つの手順で検索され、ユーザーまたは別の SQL 文、アプリケーションに 1 つの単位として渡されます。行単位で処理する必要がなく、行の物理的な格納方法や検索方法を気にする必要もありません。SQL 文を実行すると、Oracle に備わっている問合せオプティマイザが働きます。この機能によって、指定したデータに速くアクセスする方法が決定されます。Oracle には、オプティマイザの性能を向上させる方法も用意されています。

SQL コマンドを使って、次の処理を行うことができます。

- データの問合せ
- 表の中の行の挿入、更新、削除
- オブジェクトの作成、置換、変更、削除
- データベースとデータベース・オブジェクトへのアクセス制御
- データベースの一貫性と整合性の保証

SQL では、上記のすべてのタスクを 1 つの一貫性のある言語に統一しました。

すべてのリレーショナル・データベースに共通の言語

主なリレーショナル・データベース管理システムはすべて SQL をサポートしているため、SQL で得た技術的な知識を他のデータベースでも生かすことができます。さらに、SQL で記述したプログラムは移植性に優れているため、わずかな変更だけで他のデータベースに移行できます。

埋込み SQL

埋込み SQL とは、プロシージャ型プログラミング言語に埋め込んで使用する標準の SQL コマンドのことです。埋込み SQL コマンドは、Oracle プリコンパイラ関連のマニュアル、『SQL*Module for Ada Programmer's Guide』、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』および 『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』に記載されています。

埋込み SQL は以下のコマンドの集まりです。

- 対話形式のツールを使って SQL で使用できる全 SQL コマンド (SELECT、INSERT など)。
- 動的 SQL 実行コマンド (PREPARE、OPEN など)。プロシージャ型プログラミング言語に標準 SQL コマンドを統合します。

埋込み SQL には標準 SQL コマンドの拡張機能も含まれています。埋込み SQL は Oracle プリコンパイラによってサポートされます。Oracle プリコンパイラによって、埋込み SQL 文が解釈され、プロシージャ型言語のコンパイラが処理できる文に変換されます。

次の各 Oracle プリコンパイラによって、埋込み SQL のプログラムは異なるプロシージャ型言語に変換されます。

- Pro*C/C++ プリコンパイラ
- Pro*COBOL プリコンパイラ
- Pro*FORTRAN プリコンパイラ
- SQL*Module、ADA

追加情報： Oracle プリコンパイラおよび埋込み SQL コマンドの定義については、『SQL*Module for Ada Programmer's Guide 』、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』および『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照してください。

字句規則

SQL 文の記述に関する以下の字句規則は、Oracle の SQL インプリメンテーションに対してだけ適用されますが、他の SQL インプリメンテーションにも一般的に適用されます。

SQL 文では、コマンドの定義の中で空白が入る可能性がある任意の位置に、1 つ以上のタブ、改行文字、空白、コメントを記述できます。したがって、Oracle は次の 2 つの文を同一と解釈します。

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

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

予約語およびキーワード、識別子、パラメータは大文字と小文字を区別せずに記述できます。テキスト・リテラルと引用符で囲んだ名前では大文字と小文字は区別されます。構文の説明は、「Text(テキスト)」（2-2 ページ）を参照してください。

サポートするツール

Oracle の提供するツールの大部分（すべてではありません）が、Oracle の SQL の機能をすべてサポートしています。このマニュアルでは、SQL の全機能について記述します。オラクル提供のツールでサポートしていない機能がある場合、『PL/SQL ユーザーズ・ガイドおよびリファレンス』などのツールについて記述したマニュアルにその制限が示されていますので、参照してください。

2

Oracle8 SQL の要素

この章には、Oracle SQL の基本要素に関する参照情報が記載されています。

> **注意：** OBJ が前についているコマンドと説明は、Oracle Object Option がデータベース・サーバーにインストールされている場合だけ有効です。

第 4 章「コマンド」で説明するコマンドを使用する前に、この章で説明する次の概念を理解しておく必要があります。

- リテラル
- Text(テキスト)
- Integer(整数)
- Number (数)
- データ型
- NULL
- 疑似列
- コメント
- データベース・オブジェクト
- スキーマ・オブジェクト名および修飾子
- スキーマ・オブジェクトと部分を参照する

リテラル

リテラルと定数値という用語は同義であり、固定データ値のことを表しています。たとえば、’JACK’ および ’BLUE ISLAND’、’101’ はすべて文字リテラルです。なお、文字リテラル

は、単一引用符で囲みます。単一引用符を付けると、Oracle は文字リテラルとスキーマ・オブジェクト名を区別します。

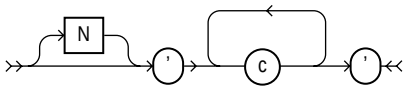
多くの SQL 文と関数では、文字リテラルと数値リテラルを指定する必要があります。式と条件の一部として、リテラルを指定できます。文字リテラルは *'text'* の表記法を、各国文字リテラルは N'*text*' の表記法を、数値リテラルはリテラルのコンテキストにより *integer* または *number* の表記法を使用して指定できます。これらの表記法の構文については、次の項で説明します。

Text(テキスト)

テキスト・リテラルまたは文字リテラルを指定します。このマニュアルの他の箇所で、式および条件、SQL 関数、SQL コマンドに示されている *'text'*(テキスト) や char(文字) に値を指定するときには、必ずこの表記法を使用してください。

text(テキスト) の構文は次の通りです。

text::=



ここで、それぞれの記号の意味は次のとおりです。

N	各国キャラクタ・セットを使ってリテラルを指定します。この表記法を使って入力したテキストは、使用時に Oracle によって各国キャラクタ・セットに変換されます。
c	単一引用符 (') を除く、データベースのキャラクタ・セットの任意の要素です。
' '	テキスト・リテラルの始まりと終わりを示す 2 つの単一引用符です。リテラル内で単一引用符を表すには、単一引用符を 2 つ使用します。

テキスト・リテラルは、単一引用符で囲んでください。このマニュアルでは、テキスト・リテラルと文字リテラルは同じ用語として使用しています。

テキスト・リテラルは、次のように CHAR データ型と VARCHAR2 データ型の両方の特性を持ちます。

- 式と条件の中のテキスト・リテラルは、Oracle によりデータ型 CHAR として扱われ、空白を埋めて長さを揃えた後で比較される。
- テキスト・リテラルの最大長は 4000 バイト。

有効なテキスト・リテラルの例は次のとおりです。

```
'Hello'
'ORACLE.dbs'
```

```
'Jackie''s raincoat'
'09-MAR-92'
N'nchar literal'
```

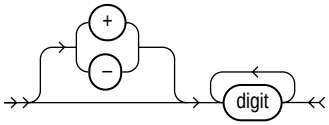
詳細は、「式」（3-73 ページ）にある *expr* の構文の説明を参照してください。

Integer(整数)

このマニュアルの他の箇所で、式および条件、SQL 関数、SQL コマンドに示されている *integer*(整数) に値を指定するときには、必ずこの表記法を使用してください。

integer(整数) の構文は次のとおりです。

integer::=



ここで、それぞれの記号の意味は次のとおりです。

digit 0、1、2、3、4、5、6、7、8、9 の 1 つです。

整数は最大 38 桁の精度を記憶できます。

有効な integers(整数) の例は次のとおりです。

```
7
+255
```

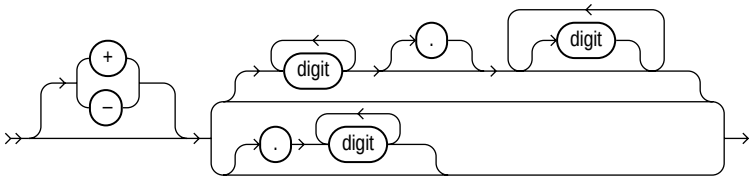
詳細は、「式」（3-73 ページ）にある expr の構文の説明を参照してください。

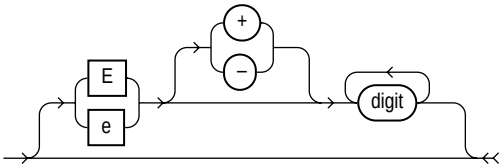
Number (数)

このマニュアルの他の箇所で、式および条件、SQL 関数、SQL コマンドに示されている *number*(数) に値を指定するときには、必ずこの表記法を使用してください。

number(数) の構文は次のとおりです。

number::=





ここで、それぞれの記号の意味は次のとおりです。

+, -	正の値または負の値を示します。符号を指定しない場合、正の値がデフォルトです。
digit	0、1、2、3、4、5、6、7、8、9 の中の 1 つです。
e, E	数が科学表記法で指定されることを示します。E の後の数字が指数を指定します。指数は -130 から 125 までの範囲で指定します。

number(数) は最大 38 桁の精度を記憶できます。

初期化パラメータ NLS_NUMERIC_CHARACTERS を使ってピリオド (.) 以外の小数点文字を設定している場合は、'*text*' の表記法で数値リテラルを指定する必要があります。この場合、Oracle は自動的にテキスト・リテラルを数値に変換します。

たとえば、NLS_NUMERIC_CHARACTERS パラメータでカンマを小数点文字に設定している場合、数値 5.123 は次のように指定します。

```
'5,123'
```

このパラメータの詳細は、『Oracle8 Server リファレンス』を参照してください。

有効な *number*(数) の例は、次のとおりです。

```
25
+6.34
0.5
25e-03
-1
```

詳細は、「式」（3-73 ページ）にある *expr* の構文の説明を参照してください。

データ型

Oracle によって操作されるリテラル値や列値は、それぞれデータ型を持っています。値のデータ型は、値に対して固定されたある特性を持つ集合を対応付けます。これらの特性に応じて、Oracle はあるデータ型の値に対して別のデータ型の値とは異なる扱いをします。たとえば、NUMBER データ型の値は加算できますが、RAW データ型の値は加算できません。

表やクラスタを作成するとき、その列のそれぞれについて内部データ型を指定しなければなりません。プロシージャやストアド・ファンクションを作成するとき、その各引数にデータ型を指定しなければなりません。データ型により各列が含むことのできる値のドメインまたは各引数が持つことのできる値のドメインが規定されます。たとえば、DATE 列は、2 月 29 日（うるう年を除く）または 2、'SHOE' という値を受け入れることはできません。列に入れられる各値は列のデータ型を受け継ぎます。たとえば、DATE 列に '01-JAN-92' を挿入すると、Oracle はそれが有効な日付に解釈されることを確認した後で、文字列 '01-JAN-92' を DATE 値として扱います。

表 2–1 に、Oracle の内部データ型をまとめます。これらのデータ型について次に詳しく説明します。

注意： Oracle プリコンパイラにより埋込み SQL プログラムで他のデータ型が区別されます。このようなデータ型は、外部データ型と呼ばれ、ホスト変数に対応付けられています。内部データ型を外部データ型と混同しないでください。Oracle による内部データ型と外部データ型の間の変換を含めて、外部データ型の説明は、『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』および『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』、『SQL*Module for Ada Programmer's Guide』を参照してください。

表 2–1　内部データ型のまとめ

コード	内部のデータ型	説明
1	`VARCHAR2(`*`size`*`)`	最大長が *size* バイトの可変長文字列。*size* の最大は 4000、最小は 1。VARCHAR2 では *size* を必ず指定しなければならない。
	`NVARCHAR2(`*`size`*`)`	最大長が *size* 文字またはバイト（選択された各国キャラクタ・セットによる）の可変長文字列。最大サイズは、各文字を保存するのに必要なバイト数によって決まるが、最大で 4000 バイト。NVARCHAR2 では *size* を必ず指定しなければならない。
2	`NUMBER(`*`p,s`*`)`	精度 *p*、位取り *s* を持つ数。精度 *p* には 1 から 38 までの値を指定できる。位取り *s* には -84 から 127 までの値を指定できる。

表 2–1　内部データ型のまとめ

コード	内部のデータ型	説明
8	LONG	最大 2 ギガバイト (2 の 31 乗から 1 を引いたバイト数) の可変長の文字データ。
12	DATE	紀元前 4712 年 1 月 1 日から紀元 4712 年 12 月 31 日までの日付を指定できる。
23	RAW(*size*)	長さが *size* バイトのバイナリ・データ。*size* の最大は 2000 バイト。RAW 値では *size* を必ず指定しなければならない。
24	LONG RAW	最大 2GB の可変長のバイナリ・データ。
69	ROWID	表の中の行のアドレスを一意に表す 16 進数ストリング。主に、ROWID 疑似列によって戻される値のためのデータ型。
96	CHAR(*size*)	長さ size バイトの固定長文字データ。size の最大は 2000 バイト。デフォルトと最小 size は 1。
	NCHAR(*size*)	長さ *size* 文字またはバイト（各国キャラクタ・セットの選択内容に応じる）の固定長文字データ。*size* の最大は、各文字を保存するのに必要なバイト数によって決まるが、最大で 2000 バイト。デフォルトと最小 *size* は 1 文字または 1 バイト（キャラクタ・セットに応じる）。
106	MLSLABEL	バイナリ形式のオペレーティング・システム・ラベル。このデータ型は、Trusted Oracle との下位互換性に使います。
112	CLOB	シングルバイト・キャラクタを含む文字ラージ・オブジェクト。可変幅のキャラクタ・セットはサポートされません。最大サイズは 4GB。
	NCLOB	固定幅のマルチバイト・キャラクタを含む文字ラージ・オブジェクト。可変幅のキャラクタ・セットはサポートされません。最大サイズは 4GB。各国キャラクタ・セットのデータを保存します。
113	BLOB	バイナリ・ラージ・オブジェクト。最大サイズは 4GB。
114	BFILE	データベース外に保存された大きなバイナリ・ファイルへロケータが格納されます。データベース・サーバー機上に存在する外部 LOB へのバイト・ストリーム I/O アクセスを可能にします。最大サイズは 4GB。

上記のデータ型のコードは Oracle によって内部的に使用されます。DUMP 関数を使用すると、列またはオブジェクト属性のデータ型コードが戻されます。

文字データ型

文字データ型を使用すると、単語や自由形式のテキストなど、データベースのキャラクタ・セットまたは各国キャラクタ・セット内の文字 (英数字) データを格納できます。他のデータ型よりも制限が少なく、その結果、特性も少なくなっています。たとえば、文字データ型の列はすべての英数字の値を格納できますが、NUMBER 型の列は数値しか格納できません。

文字データは、7 ビット ASCII や EBCDIC コード・ページ 500 など、データベース作成時に指定されたキャラクタ・セットの 1 つに対応しているバイト値で文字列に格納されます。Oracle は、シングルバイトのキャラクタ・セットとマルチバイトのキャラクタ・セットの両方をサポートしています。

以下のデータ型が文字データに対して使用されます。

- CHAR データ型
- NCHAR データ型
- NVARCHAR2 データ型
- VARCHAR2 データ型

CHAR データ型

CHAR データ型は固定長の文字列を指定します。CHAR 列で表を作成する場合、列の長さをバイト単位で指定します。Oracle では、その列の中に格納される値がすべてこの長さを持つように調整します。列の長さよりも短い値が挿入されると、その値の後に空白を埋め込んで列の長さに合わせます。列に対して長すぎる値を挿入しようとすると、Oracle はエラーを戻します。

CHAR 列のデフォルトの長さは 1 文字で、この許容最大値は 2000 文字です。長さがゼロの文字列を CHAR 列に挿入できますが、この CHAR 列が比較されるときには、空白が 1 文字埋め込まれます。比較方法の説明は、「データ型の比較規則」（2-22 ページ）を参照してください。

NCHAR データ型

NCHAR データ型は固定長の各国キャラクタ・セット文字列を指定します。NCHAR 列で表を作成する場合、列の長さを文字単位またはバイト単位のいずれかで定義します。使用する各国キャラクタ・セットは、データベースを作成するときに指定します。

指定した各国キャラクタ・セットが固定幅である場合は（たとえば、JA16EUCFIXED など）、NCHAR の列サイズを文字列の長さの文字数で宣言します。各国キャラクタ・セットが可変幅（たとえば、JA16SJIS など）である場合は、列サイズをバイト単位で宣言します。次の文では、各国キャラクタ・セットとして JA16EUCFIXED を使用すると、長さが 30 文字までの文字列を保存できる 1 つの NCHAR 列が含まれた表が作成されます。

```
CREATE TABLE tab1 (col1 NCHAR(30));
```

列の最大長は、各国キャラクタ・セットの定義によって決まります。文字データ型 NCHAR の幅指定では、各国キャラクタ・セットが固定幅である場合には文字数が参照され、各国キャラクタ・セットが可変幅である場合にはバイト数が参照されます。許容最大列サイズは 2000 バイトです。固定幅のマルチバイト・キャラクタ・セットの場合は、列の許容最大長が、2000 バイト以下の文字数になります。

列の長さよりも短い値が挿入されると、その値の後に空白を埋め込んで列の長さに合わせます。CHAR 値を NCHAR 列に挿入することや、NCHAR 値を CHAR 列に挿入することはできません。

次の例では、tab1 の col1 列と各国キャラクタ・セットの文字列 NCHAR リテラルを比較します。

```
SELECT * FROM tab1 WHERE col1 = N'NCHAR literal';
```

OBJ NCHAR 属性を持つオブジェクトは作成できませんが、メソッドでは NCHAR パラメータを指定できます。

NVARCHAR2 データ型

NVARCHAR2 データ型は可変長の各国キャラクタ・セット文字列を指定します。NVARCHAR2 列で表を作成する場合、保持できる最大の文字数またはバイト数を指定します。Oracle では、列の最大長を超えていない限り、各値を指定されたとおりに正確に列に保存します。

列の最大長は、各国キャラクタ・セットの定義によって決まります。文字データ型 NVARCHAR2 の幅指定では、各国キャラクタ・セットが固定幅である場合には文字数が参照され、各国キャラクタ・セットが可変幅である場合にはバイト数が参照されます。許容最大列サイズは 4000 バイトです。固定幅のマルチバイト・キャラクタ・セットの場合は、列の許容最大長が、4000 バイト以下の文字数になります。

次の文では、各国キャラクタ・セットとして JA16EUCFIXED を使用すると、長さが 2000 文字（各文字が 2 バイトなので 4000 バイトとして保存される）の 1 つの NVARCHAR2 列が含まれた表が作成されます。

```
CREATE TABLE tab1 (col1 NVARCHAR2(2000));
```

OBJ NVARCHAR2 属性を持つオブジェクトは作成できませんが、メソッドでは NVARCHAR2 パラメータを指定できます。

VARCHAR2 データ型

VARCHAR2 データ型は可変長の文字列を指定します。VARCHAR2 列を作成する場合、保持できる最大のデータのバイト数を指定できます。Oracle では、列の最大長を超えていない限り、各値を指定されたとおりに正確に列に保存します。保存される文字列の実際の長さは 0(ゼロ) でも構いませんが、最大長は 1 バイト以上でなければなりません。最大長を超える値を挿入しようとすると、Oracle はエラーを戻します。

VARCHAR2 列では最大長を指定しなければなりません。VARCHAR2 データの最大長は 4000 バイトです。Oracle は、非空白埋め比較を使用して VARCHAR2 値を比較します。比較方法については、「データ型の比較規則」（2-22 ページ）を参照してください。

VARCHAR データ型

現在、VARCHAR データ型は VARCHAR2 データ型と同義です。VARCHAR よりも VARCHAR2 を使用することをお薦めします。将来、VARCHAR データ型が変更され、異なる比較方法で比較される別の可変長文字列の型になる可能性もあります。

NUMBER データ型

NUMBER データ型は、38 桁の精度を持ち、ゼロおよび絶対値が、1.0×10^{-130} から $9.9...9 \times 10^{125}$(38 個の 9 の後に 0 が 88 個続く) までの範囲にある正と負の固定小数点数および浮動小数点数を格納するために使用されます。1.0×10^{126} 以上の値を持つ算術式を指定した場合、Oracle はエラーを戻します。

次の書式で固定小数点数を指定できます。

```
NUMBER(p,s)
```

ここで、それぞれの意味は次のとおりです。

p 精度 (*precision*)、すなわち全体の桁数です。Oracle は 38 桁までの精度で数の移植性を保証します。

s 位取り (*scale*)、すなわち小数点の右側にある桁数です。位取りの有効範囲は -84 から 127 までです。

次の書式で整数を指定できます。

`NUMBER (p)` 精度が p で位取りが 0 の固定小数点数です (NUMBER(p,0) と同じです)。

次の書式で浮動小数点を指定できます。

`NUMBER` 精度が 38 桁の浮動小数点数です。なお、浮動小数点数では、位取りは指定できません。（詳細は、「浮動小数点数」（2-10 ページ）を参照してください。)

位取りと精度

入力に対する特別の整合性検査として、固定小数点数列の位取りと精度を指定してください。位取りと精度を指定しても、すべての値が固定長に強制されるわけではありません。値が精度の有効範囲を超えると、Oracle はエラーを戻します。値が位取りの有効範囲を超えると、Oracle はその値を丸めます。

次に、いろいろな精度と位取りを使用して Oracle がデータを格納する例を示します。

実際のデータ	指定する精度と位取り	格納されるデータ
7456123.89	NUMBER	7456123.89
7456123.89	NUMBER(9)	7456124
7456123.89	NUMBER(9,2)	7456123.89
7456123.89	NUMBER(9,1)	7456123.9
7456123.89	NUMBER(6)	精度を超える
7456123.89	NUMBER(7,-2)	7456100
7456123.89	NUMBER(7,2)	精度を超える

負の位取り

位取りが負の場合には、実際のデータは整数部分の右から指定された桁数だけ丸められます。たとえば、(10,-2) と指定すると 100 の位まで丸められます。

精度より大きな位取り

通常はありえないことですが、精度よりも大きな位取りを指定できます。この場合、精度は小数点の右側にある最大有効桁数を示します。すべての NUMBER データ型と同じように、値が精度を超えると Oracle はエラー・メッセージを戻します。値が位取りの有効範囲を超えると、Oracle はその値を丸めます。たとえば、NUMBER(4,5) として定義された列は、小数点の後の最初の桁が 0(ゼロ) でなければならず、小数点以下 5 桁を超える値はすべて丸められます。次に、精度より大きい位取りを指定した場合の例を示します。

実際のデータ	指定する精度と位取り	格納されるデータ
.01234	NUMBER(4,5)	.01234
.00012	NUMBER(4,5)	.00012
.000127	NUMBER(4,5)	.00013
.0000012	NUMBER(2,7)	.0000012
.00000123	NUMBER(2,7)	.0000012

浮動小数点数

浮動小数点数は、最初の桁から最後の桁までの任意の位置に小数点を置くことも、小数点を省略することもできます。小数点以降の桁数に制限はないため、浮動小数点数に対して位取りは指定できません。

「NUMBER データ型」（2-9 ページ）で説明している形式で浮動小数点数を指定できます。また、Oracle は ANSI データ型 FLOAT もサポートします。次の構文書式のいずれかを使用してこのデータ型を指定します。

`FLOAT` 38 桁の 10 進精度または 126 桁の 2 進精度で浮動小数点数を指定します。

`FLOAT(`*b*`)` 2 進精度 *b* で浮動小数点数を指定します。精度 *b* は 1 から 126 までの範囲です。2 進精度から 10 進精度に変換するには、*b* に 0.30103 を乗算します。10 進精度から 2 進精度に変換するには、10 進精度に 3.32193 を乗算します。2 進数精度の 126 桁は 10 進精度の 38 桁におよそ等しくなります。

LONG データ型

LONG 列には、最大 2GB(2 の 31 乗から 1 を引いたバイト数) の可変長の文字列を格納できます。LONG 列には多くの点で VARCHAR2 列と同じ特性があります。LONG 列を使用すると、長いテキスト列を格納できます。Oracle は、ビュー定義のテキストを格納するために、データ・ディクショナリ内で LONG 列を使用しています。LONG 値の長さは、使用しているコンピュータで利用できるメモリーによって制限される可能性もあります。

SQL 文の中の次の場所で LONG 列を参照できます。

- SELECT リスト
- UPDATE 文の SET 句
- INSERT 文の VALUES 句

LONG 値を使用する場合には次の制限があります。

- 表には複数の LONG 列を含めることはできない。
- LONG 列は整合性制約に使用できない。ただし、NULL 制約と NOT NULL 制約を除く。
- LONG 列に索引を付けることはできない。
- ストアド・ファンクションは LONG 値を戻すことはできない。
- 単一の SQL 文に指定する、すべての LONG 列、更新対象の表、ロック対象の表は、同一データベース上に位置していなければならない。

また、LONG 列は SQL 文の次のような部分では使用できません。

- SELECT 文の WHERE 句、GROUP BY 句、ORDER BY 句、CONNECT BY 句または DISTINCT 演算子
- SELECT 文の UNIQUE 句
- CREATE CLUSTER 文の列リスト
- CREATE SNAPSHOT 文の CLUSTER 句

- SQL 関数 (SUBSTR や INSTR など)
- 式または条件
- GROUP BY 句を含む問合せの SELECT リスト
- 集合演算子によって結合されている副問合せや問合せの SELECT リスト
- CREATE TABLE ... AS SELECT 文の SELECT リスト
- INSERT 文の副問合せの SELECT リスト

トリガーでは、LONG データ型は次のように使用されます。

- トリガー内の SQL 文で、データを LONG 列に挿入できる。
- LONG 列のデータを CHAR や VARCHAR2 などの制約があるデータ型に変換できる場合は、トリガー内の SQL 文で LONG 列を参照できる。これらのデータ型の最大長は 32KB です。
- トリガー内の変数は、LONG データ型を使用して宣言できない。
- :NEW と :OLD は LONG 列で使用できない。

Oracle コール・インタフェースを使用して、データベースから LONG 値の一部を検索できます。『Oracle コール・インタフェース・プログラマーズ・ガイド』を参照してください。

DATE データ型

DATE データ型は日付と時間の情報を格納するために使用されます。日付と時間の情報は CHAR データ型と NUMBER データ型で表現できますが、DATE データ型には特別に対応付けられている特性があります。各 DATE 値には、世紀、年、月、日、時、分、秒の情報が格納されます。

日付値を指定するには、文字値や数値を TO_DATE 関数によって日付値に変換しなければなりません。デフォルト日付書式の文字値が日付式で使用されると、Oracle は自動的にそれらを日付値に変換します。デフォルトの日付書式は、初期化パラメータ NLS_DATE_FORMAT によって指定され、たとえば 'DD-MON-YY' のような文字列になります。'DD-MON-YY' は、日付としての 2 桁の数、月の名前の省略形、年の下 2 桁を含む日付書式です。

日付値を指定する場合に時間コンポーネントを指定しないと、デフォルト時間の 12:00:00 a.m.(真夜中) が採用されます。日付値を指定する場合に日付を指定しないと、デフォルト日付である現在の月の最初の日が採用されます。

日付関数 SYSDATE は現在の日付と時間を戻します。SYSDATE 関数と TO_DATE 関数およびデフォルト日付書式の説明は、第 3 章「演算子、関数、式、条件」を参照してください。

日付算術

日付に対しては、日付の加算や減算だけでなく、数定数の加算や減算ができます。Oracle は算術日付式の数定数を日付の数として解釈します。たとえば、SYSDATE + 1 は明日です。SYSDATE - 7 は 1 週間前です。SYSDATE + (10/1440) は 10 分後です。SYSDATE から EMP

表の HIREDATE 列を引くと、各従業員が雇用されてから経過した日数が戻ります。DATE 値の乗算や除算はできません。

Oracle は一般的な日付操作のために関数を用意しています。たとえば、ADD_MONTHS 関数は日付に月を加算したり、減算したりできます。MONTHS_BETWEEN 関数は 2 つの日付の間の月数を戻します。結果の小数部は月 (1ヶ月は 31 日) を単位として表されています。日付関数の詳細は、「日付関数」（3-34 ページ）を参照してください。

各日付には時間コンポーネントが含まれるため、日付操作のほとんどの結果には小数部が含まれます。この小数部は日を単位として表されています。たとえば、1.5 日は 36 時間です。

ユリウス暦の使用方法

ユリウス暦は紀元前 4712 年 1 月 1 日から経過した日数です。ユリウス暦によって共通の基準で日付を算定できます。日付関数 TO_DATE と TO_CHAR で日付書式モデル「J」を使用して、Oracle の DATE 値とユリウス暦の間で変換を行うことができます。

例 次の文では 1997 年 1 月 1 日と等価なユリウス暦が戻されます。

```
SELECT TO_CHAR(TO_DATE('01-01-1997', 'MM-DD-YYYY'),'J')
    FROM DUAL;

TO_CHAR
--------
2450450
```

DUAL 表については、「DUAL 表からの選択」（4-530 ページ）を参照してください。

RAW データ型と LONG RAW データ型

RAW データ型と LONG RAW データ型のデータは、Oracle によって解釈されません（異なるシステム間でデータを移動するときに変換されない）。これらのデータ型は、バイナリ・データまたはバイト列に使用されます。たとえば、LONG RAW は、図形または音声、文書、バイナリ・データの配列の格納に使用できますが、解釈方法は用途によって異なります。

RAW は、VARCHAR2 文字データ型と同様の可変長データ型です。ただし、Net8(ユーザー・セッションとインスタンスを接続する) と Import および Export ユーティリティは、RAW または LONG RAW データの転送時には文字変換を行いません。これに対し、Net8 と Import/Export は、データベースのキャラクタ・セットとユーザーのセッションのキャラクタ・セット (ALTER SESSION コマンドの NLS_LANGUAGE パラメータによって設定) が異なる場合に、CHAR および VARCHAR2、LONG データをこれら 2 つのキャラクタ・セット間で自動的に変換します。

RAW データまたは LONG RAW データと CHAR データ間の自動変換時に、バイナリ・データは 16 進数で表現されます。1 つの 16 進文字で 4 ビットの RAW データを表します。たと

えば、ビット列が 11001011 で表示される 1 バイトの RAW データは、'CB' として表示あるいは入力されます。

RAW データには索引を付けられますが、LONG RAW データには索引を付けられません。

ラージ・オブジェクト (LOB) データ型

内部 LOB データ型の BLOB、CLOB、NCLOB、および外部データ型の BFILE には、構造化されていない大きいデータ（テキスト、イメージ、ビデオ、スペーシャル・データなど）を最大 4GB まで格納できます。

表を作成するときに、内部 LOB 列または内部 LOB オブジェクト属性にオプションで表に指定したものとは異なる表領域と記憶特性を指定できます。

内部 LOB 列には、ライン外 LOB 値またはインライン LOB 値を参照できる LOB ロケータが含まれています。表から LOB を選択すると、実際には LOB のロケータが戻され、LOB 値全体は戻されません。LOB に対する DBMS_LOB パッケージと OCI の操作は、これらのロケータを介して実行されます。これらのインタフェースおよび LOB の詳細は、『Oracle8 Server アプリケーション開発者ガイド』および『Oracle8 コール・インタフェース・プログラマーズ・ガイド』を参照してください。

LOB は、LONG 型および LONG RAW 型と似ていますが、次の点で異なります。

- LOB は、ユーザー定義のデータ型（オブジェクト）の属性に指定できる。
- LOB ロケータは、表の列に格納される。実際の LOB 値は表の列に格納される場合とされない場合がある。格納されない場合、BLOB および NCLOB、CLOB の値は、別々の表領域に格納される。また、BFILE データは、サーバー上の外部ファイルに格納される。
- LOB 列にアクセスしたときに戻されるのはロケータである。
- LOB のサイズは最大 4GB である。BFILE の最大サイズはオペレーティング・システムによって異なるが、4GB を超えることはない。
- LOB では、データの効果的かつランダムな断片単位のアクセスおよび操作が可能である。
- 1 つの表内に 2 つ以上の LOB 列を定義できる。
- NCLOB の例外を除いては、1 つのオブジェクトに 1 つ以上の LOB 属性を定義できる。
- LOB バインド変数を宣言できる。
- LOB 列と LOB 属性を選択できる。
- 1 つ以上の LOB 列または 1 つ以上の LOB 属性を持つ、オブジェクトが含まれている新しい行を挿入したり、既存の行を更新したりできる。（内部 LOB 値を NULL つまり空に設定したり、LOB 全体をデータに置き換えたりできる。BFILE は、NULL に設定、または別のファイルを指すように設定できる。)

- LOB 行と列の交差部または LOB 属性を、別の LOB 行と列の交差部または LOB 属性を使って更新できる。
- LOB 列または LOB 属性が含まれている行を削除できる（これにより LOB 値も削除される）。BFILE の場合、実際のオペレーティング・システム・ファイルは削除されない。

詳細は、『Oracle8 Server アプリケーション開発者ガイド』にある LOB の制限に関する説明を参照してください。

内部 LOB 列の行にアクセスし、行を挿入するには、INSERT 文を使用して最初に内部 LOB 値を空に初期化します。行を挿入したら、空の LOB を選択し、DBMS_LOB パッケージまたは OCI を使ってそれを移入できます。

次の例では、LOB 列を持つ表が作成されます。

```
CREATE TABLE person_table (name CHAR(40),
                           resume CLOB,
                           picture BLOB)
  LOB (resume) STORE AS
               ( TABLESPACE resumes
                  STORAGE (INITIAL 5M NEXT 5M) );
```

LOB 値の大容量のデータの読み書きには LOB を使用します。

BFILE データ型

BFILE データ型を使用すると、Oracle データベース外のファイル・システムに格納されているバイナリ・ファイル LOB にアクセスできます。BFILE 列または属性には、サーバーのファイル・システム上のバイナリ・ファイルに対するポインタとして機能する、BFILE ロケータが格納されます。ロケータには、ディレクトリ別名とファイル名が保持されます。「CREATE DIRECTORY コマンド」（4-226 ページ）を参照してください。

バイナリ・ファイル LOB は、トランザクションにはかかわりません。ファイルの統合性と耐久性を提供しているのは基本にあるオペレーティング・システムです。サポートされるファイルの最大サイズは 4GB です。

データベース管理者は、ファイルが存在すること、および Oracle のプロセスがファイルに対するオペレーティング・システムの読込み許可があることを確認する必要があります。

BFILE データ型を使用すると、大きいバイナリ・ファイルの読込み専用サポートが可能になります。この場合、ファイルは修正できません。Oracle では、ファイル・データにアクセスするために API が提供されています。ファイル・データにアクセスするために使用する主インタフェースは、DBMS_LOB パッケージと OCI です。LOB の詳細は、『Oracle8 Server アプリケーション開発者ガイド』および『Oracle8 コール・インタフェース・プログラマーズ・ガイド』を参照してください。

BLOB データ型

BLOB データ型は、構造化されていないバイナリラージ・オブジェクトを格納するために使用します。BLOB は、キャラクタ・セットの意味を持たないビットストリームとして考えることができます。BLOB には、4GB までのバイナリ・データを格納できます。

BLOB では、トランザクションが完全にサポートされます。SQL または DBMS_LOB パッケージ、OCI を介して行った変更は、すべてトランザクションに反映されます。BLOB 値の操作は、コミットまたはロール・バックできます。1 つのトランザクション内の PL/SQL または OCI 変数を BLOB ロケータに保存し、そのロケータを別のトランザクションまたはセッションで使用することはできません。

CLOB データ型

CLOB データ型は、シングルバイト文字のラージ・オブジェクト・データを格納するために使用します。可変幅のキャラクタ・セットはサポートされません。CLOB には、4GB までの文字データを格納できます。

CLOB では、トランザクションが完全にサポートされます。SQL または OCI、DBMS_LOB パッケージを介して行った変更はすべて、トランザクションに反映されます。CLOB 値の操作は、コミットまたはロール・バックできます。1 つのトランザクション内の PL/SQL または OCI 変数を CLOB ロケータに保存し、そのロケータを別のトランザクションまたはセッションで使用することはできません。

NCLOB データ型

NCLOB データ型には、固定幅のマルチバイト各国キャラクタ・セット文字 (NCHAR) データを保存するために使用します。可変幅のキャラクタ・セットはサポートされません。NCLOB には、4GB までの文字テキスト・データを保存できます。

NCLOB では、トランザクションが完全にサポートされます。SQL または DBMS_LOB パッケージ、OCI を介して行った変更は、すべてトランザクションに反映されます。NCLOB 値の操作は、コミットまたはロール・バックできます。1 つのトランザクション内の PL/SQL または OCI 変数を NCLOB ロケータに保存し、そのロケータを別のトランザクションまたはセッションで使用することはできません。

OBJ NCLOB 属性を持つオブジェクトは作成できませんが、メソッドで NCLOB パラメータを指定することはできます。

ROWID データ型

データベース内の各行にはアドレスがあります。疑似列 ROWID を問い合せることによって行のアドレスを調べることができます。この疑似列の値は、各行のアドレスを表す 16 進数ストリングです。16 進数ストリングのデータ型は ROWID です。ROWID 疑似列の詳細は、「疑似列」（2-30 ページ）を参照してください。また、ROWID データ型を持つ実際の列を含む表やクラスタを作成することもできます。Oracle では、そのような列の値が有効な ROWID であることは保証されません。

制限 ROWID

Oracle8 Server では、パーティション表と索引および表領域関連のデータ・ブロック・アドレス (DBA) を明確かつ効果的にサポートするために、ROWID の拡張形式が採用されています。

Oracle7 以前のリリースでは、ROWID は次のような文字値で表現されます。

```
block.row.file
```

ここで、それぞれの意味は次のとおりです。

block 行を含むデータ・ファイルのデータ・ブロックを識別する 16 進数ストリングです。このストリングの長さはオペレーティング・システムによって異なることがあります。

row データ・ブロック内の行を識別する 4 桁の 16 進数ストリングです。ブロック内の最初の行は 0 になります。

file 行を含むデータ・ファイルを識別する 16 進数ストリングです。最初のデータ・ファイルは 1 になります。このストリングの長さはオペレーティング・システムによって異なることがあります。

Oracle8 では、この種の ROWID は制限 ROWID と呼ばれます。

拡張 ROWID

ユーザー列に格納される Oracle8 の拡張 ROWID データ型には、制限 ROWID のデータに加え、データ・オブジェクト番号が含まれます。データ・オブジェクト番号は、すべてのデータベース・セグメントに割り当てられる識別番号です。データ・オブジェクト番号は、データ・ディクショナリ・ビューの USER_OBJECTS および DBA_OBJECTS、ALL_OBJECTS から取り出すことができます。同じセグメントを共有するオブジェクト（たとえば、同じクラスタ内のクラスタ化された表など）には、同じオブジェクト番号が付けられます。

拡張 ROWID は直接利用できません。拡張 ROWID の内容を解釈するには、提供されているパッケージ DBMS_ROWID を使用します。パッケージ関数を使用すると、制限の ROWID から直接入手された情報が取り出され、提供されます。DBMS_ROWID パッケージの関数は、ビルトインの SQL 関数と同様に使用できます。表 2–2 は、DBMS_ROWID パッケージの関数とプロシージャのリストです。DBMS_ROWID の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

表 2–2　DBMS_ROWID 関数

関数名	説明
ROWID_CREATE	検査目的だけの ROWID を作成する。
ROWID_TYPE	ROWID のタイプを戻す。0 が制限、1 が拡張。

表 2–2　DBMS_ROWID 関数

関数名	説明
ROWID_OBJECT	拡張 ROWID のオブジェクト番号を戻す。
ROWID_RELATIVE_FNO	ROWID のファイル番号を戻す。
ROWID_BLOCK_NUMBER	ROWID のブロック番号を戻す。
ROWID_ROW_NUMBER	行番号を戻す。
ROWID_TO_ABSOLUTE_ FNO	特定の表内の行の ROWID に対応付けられている絶対ファイル番号を戻す。
ROWID_TO_EXTENDED	ROWID を制限形式から拡張形式に変換する。
ROWID_TO_RESTRICTED	拡張 ROWID を制限形式に変換する。
ROWID_VERIFY	ROWID が ROWID_TO_EXTENDED 関数によって正しく拡張されるかどうかをチェックする。

互換性と移行

制限形式の ROWID は、旧リリースとの互換性を持たせるために Oracle8 でもサポートされていますが、すべての表で拡張形式の ROWID が返されます。互換性と移行の問題については、『Oracle8 Server 移行ガイド』を参照してください。

MLSLABEL データ型

MLSLABEL データ型は、安全性の高いオペレーティング・システムで使用されるバイナリ形式のラベルを格納するために使用します。このデータ型は、Oracle8 では Trusted Oracle を使用した以前のバージョンの Oracle サーバーとの下位互換性のためにサポートされています。

ラベルは、情報へのアクセスを取り次ぐために Trusted Oracle によって使用されます。標準の Oracle Server を使用している場合も、これらのデータ型で列を定義できます。このデータ型とラベルを含めた Trusted Oracle の詳細は、使用している Trusted Oracle のマニュアルを参照してください。

ANSI、DB2、SQL/DS のデータ型

表とクラスタを作成する SQL コマンドは、ANSI データ型、および IBM 社の製品 SQL/DS と DB2 のデータ型も受け入れます。Oracle では ANSI または IBM のデータ型の名前を認識し、列のデータ型の名前として記録します。次に、表 2–3 および表 2–4 で規定される変換に基づいて Oracle のデータ型で列データを格納します。

表 2–3　Oracle データ型に変換される ANSI データ型

ANSI SQL データ型	Oracle データ型
CHARACTER(*n*)	CHAR(*n*)
CHAR(*n*)	
CHARACTER VARYING(*n*)	VARCHAR(*n*)
CHAR VARYING(*n*)	
NATIONAL CHARACTER(*n*)	NCHAR(*n*)
NATIONAL CHAR(*n*)	
NCHAR(*n*)	
NATIONAL CHARACTER VARYING(*n*)	NVARCHAR2(*n*)
NATIONAL CHAR VARYING(*n*)	
NCHAR VARYING(*n*)	
NUMERIC(*p*,*s*)	NUMBER(*p*,*s*)
DECIMAL(*p*,*s*)[a]	
INTEGER	NUMBER(38)
INT	
SMALLINT	
FLOAT(*b*)[b]	NUMBER
DOUBLE PRECISION[c]	
REAL[d]	

[a]NUMERIC データ型および DECIMAL データ型では、固定小数点数だけを指定できます。これらのデータ型では、s のデフォルトは 0 です。

[b]FLOAT データ型は、2 進精度 b を持つ浮動小数点数です。このデータ型のデフォルト精度は、126 桁の 2 進精度または 38 桁の 10 進精度です。

[c]DOUBLE PRECISION データ型は 126 桁の 2 進精度を持つ浮動小数点数です。

[d]REAL データ型は 63 桁の 2 進精度または 18 桁の 10 進精度を持つ浮動小数点数です。

表 2–4　Oracle データ型に変換される SQL/DS と DB2 のデータ型

SQL/DS と DB2 データ型	Oracle データ型
CHARACTER(*n*)	CHAR(*n*)
VARCHAR(*n*)	VARCHAR(*n*)
LONG VARCHAR(*n*)	LONG
DECIMAL(*p,s*)[a]	NUMBER*(p,s)*
INTEGER	NUMBER(38)
SMALLINT	
FLOAT(*b*)[b]	NUMBER

[a]DECIMAL データ型では固定小数点数だけを指定できます。このデータ型では、s のデフォルトは 0 です。

[b]FLOAT データ型は、2 進精度 b を持つ浮動小数点数です。このデータ型のデフォルト精度は、126 桁の 2 進精度または 38 桁の 10 進精度です。

次の SQL/DS と DB2 のデータ型には対応する Oracle データ型がありません。これらのデータ型を持つ列を定義しないでください。

- GRAPHIC
- LONG VARGRAPHIC
- VARGRAPHIC
- TIME
- TIMESTAMP

TIME と TIMESTAMP のデータは、Oracle の DATE データとして表現できることにも注意してください。

OBJ ユーザー定義型

ユーザー定義のデータ型には、Oracle ビルトイン・データ型と他のユーザー定義のデータ型が、アプリケーション内のデータの構造と動作をモデル化する型の構築ブロックとして使用されます。Oracle ビルトイン・データ型については、『Oracle8 概要』を参照してください。ユーザー定義型の作成方法については、「CREATE TYPE コマンド」（4-342 ページ）および「CREATE TYPE BODY コマンド」（4-350 ページ）を参照してください。ユーザー定義型の使用方法については、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

オブジェクト型

オブジェクト型とは、実社会エンティティ (たとえば、発注書など) を抽象化し、アプリケーション・プログラムで処理できるようにしたものです。1 つのオブジェクト型は、3 種類のコンポーネントをもつスキーマ・オブジェクトです。

- 名前とは、スキーマ内でオブジェクト型を一意に識別するためのものです。
- 属性とは、ビルトイン型または他のユーザー定義型です。属性は、実社会エンティティの構造をモデル化します。
- メソッドとは、PL/SQL で記述され、データベースに格納される関数またはプロシージャ、もしくは、C などの言語で記述され、外部に格納される関数またはプロシージャのことです。メソッドは、アプリケーションが実社会エンティティに対して実行できる操作をインプリメントします。

REF

オブジェクト識別子 (OID) を使用することにより、オブジェクトを一意に識別し、他のオブジェクトまたは関係表からそのオブジェクトを参照できます。REF と呼ばれるビルトイン・データ型が、そのような参照を表します。REF は、オブジェクト識別子のコンテナです。REF は、オブジェクトへのポインタとなります。

REF 値が、存在しないオブジェクトを指している場合、その REF は DANGLING(参照先がない) 状態であるといいます。DANGLING は、NULL の状態とは異なります。REF が DANGLING 状態であるかないかを確認するには、IS [NOT] DANGLING という述語を使います。たとえば、列型が EMP_T 型を指している REF である MGR 列があります。この MGR 列を含む表 DEPT は、次のように指定します。

```
SELECT t.mgr.name
FROM dept t
WHERE t.mgr IS NOT DANGLING;
```

VARRAY

配列とは、順序付けられたデータ要素の集合です。ある特定の配列のすべての要素は、同じデータ型です。各要素には索引があります。索引は、各要素の配列内での位置に対応する番号です。

配列内の要素数は、その配列のサイズを表します。Oracle の配列は可変サイズなので、VARRAY と呼ばれます。配列を宣言する場合は、最大サイズを指定する必要があります。

VARRAY 宣言時には領域は割り当てられません。VARRAY では、次のような型を定義します。

- 関係表の列のデータ型
- オブジェクト型属性
- PL/SQL の変数またはパラメータ、関数の戻り型

注意： 索引構成表の列のデータ型には VARRAY は指定できません。

1 つの配列オブジェクトはインライン形式でその行の他のデータと同じ表領域に格納されます。

ネストした表

ネストした表の型は順序付けされていない要素のセットを表現するのに使われます。その要素は、ビルトイン型またはユーザー定義型です。ネストした表は、単一列の表として表示できます。ネストした表がオブジェクト型の場合、オブジェクト型のそれぞれの属性を表す複数列の表としても表示できます。

ネストした表を定義した場合、領域は割り当てられません。次のものを宣言するための型を定義します。

- 関係表の列
- オブジェクト型属性
- PL/SQL の変数およびパラメータ、関数の戻り値

ネストした表が、関係表内の列型として現れた場合、またはオブジェクト表の基礎を形成するオブジェクト型の属性として現れた場合、Oracle は、ネストした表のすべてのデータを単一表に格納し、その単一表を、ネストした表を囲む関係表またはオブジェクト表に対応付けます。

データ型の比較規則

ここでは Oracle が各データ型の値を比較する方法について記述します。

数値

大きな値は小さな値よりも大きいとみなされます。すべての負の数は、ゼロとすべての正の数より小さいとみなされます。したがって、-1 は 100 より小さく、-100 は -1 より小さいとみなされます。

日付値

後の日付は前の日付よりも大きいとみなされます。たとえば、’29-MAR-1991’(1991 年 3 月 29 日) に等しい日付は ’05-JAN-1992’(1992 年 1 月 5 日) に等しい日付よりも小さく、’05-JAN-1992 1:35pm’(1992 年 1 月 5 日午後 1 時 35 分) に等しい日付は ’05-JAN-1992 10:09am’(1992 年 1 月 5 日午前 10 時 9 分) に等しい日付よりも大きいとみなされます。

文字列値

文字値は、次のどちらかの比較方法を使用して比較されます。

- 空白埋め比較
- 非空白埋め比較

ここでは、これらの比較方法について説明します。これらの異なる比較方法を使用して 2 つの文字値を比較した場合、その結果が異なることもあります。表 2–5 に、それぞれの比較方法を使用して 5 組の文字を比較した結果を示します。通常、空白埋め比較と非空白埋め比較の結果は同じです。表に示されている最後の比較では、空白埋め比較と非空白埋め比較の違いが明確になっています。

表 2–5　空白埋め比較と非空白埋め比較による比較結果

空白埋め比較	非空白埋め比較
'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　2 つの値の長さが異なる場合、Oracle はまず短い方の値の最後に空白を追加して、2 つの値が同じ長さになるようにします。次に Oracle は、その 2 つの値を、最初に異なる文字まで 1 文字ずつ比較します。最初に異なる文字位置で、より大きい文字をもつ値の方が大きいとみなされます。2 つの値に異なる文字がない場合、その 2 つの値は等価とみなされます。この規則は、2 つの値の後続する空白数だけが異なる場合、その 2 つの値は等価であるということを意味します。Oracle では、比較する両方の値が、CHAR データ型、NCHAR データ型、テキスト・リテラルのいずれかの式の場合、または USER 関数の戻り値の場合だけ空白埋め比較方法を使います。

非空白埋め比較　Oracle は、2 つの値を、最初に異なる文字まで 1 文字ずつ比較します。最初に異なる文字位置で、より大きい文字をもつ値の方が大きいとみなされます。長さの異なる 2 つの値を短い方の値の最後まで比較して、すべて同じ文字だった場合、長い方の値が大きいとみなされます。同じ長さの 2 つの値に異なる文字がない場合、その 2 つの値は等価とみなされます。Oracle では、比較する片方または両方の値が VARCHAR2 データ型か NVARCHAR2 データ型の場合、非空白埋め比較方法を使います。

単一文字

Oracle は、データベースのキャラクタ・セットで与えられた数値に従って各文字を比較します。第 1 の文字の数値が第 2 の文字の数値よりも大きい場合、第 1 の文字は第 2 の文字よりも大きいとみなされます。Oracle は、空白はどの文字よりも小さいとみなします。これは、ほとんどのキャラクタ・セットで言えることです。

次に、一般的なキャラクタ・セットを列挙します。

- 7 ビット ASCII(情報交換用米国標準コード)
- EBCDIC コード（拡張 2 進化 10 進コード）ページ 500
- ISO 8859/1(国際標準化機構)
- JEUC 日本語拡張 UNIX

ASCII と EBCDIC のキャラクタ・セットの一部を表 2–6 と表 2–7 に示します。なお、大文字と小文字は同じではありません。また、キャラクタ・セットの照合順番は、特定の言語に対する言語順序と一致しない場合があります。

表 2–6　ASCII キャラクタ・セット

記号	10 進値	記号	10 進値
空白	32	;	59
!	33	<	60
"	34	=	61
#	35	>	62
$	36	?	63
%	37	@	64
&	38	A-Z	65-90
’	39	[	91
(	40	¥	92
)	41	]	93
*	42	^^	94
+	43	_	95
,	44	’	96
-	45	a-z	97-122
.	46	{	123
/	47	|	124
0-9	48-57	}	125
:	58	~	126

表 2–7　EBCDIC キャラクタ・セット

記号	10 進値	記号	10 進値
空白	64	%	108
¢	74	_	109
.	75	>	110
<	76	?	111
(	77	:	122
+	78	#	123
|	79	@	124
&	80	’	125
!	90	=	126
$	91	"	127
*	92	a-i	129-137
)	93	j-r	145-153
;	94	s-z	162-169
ÿ	95	A-I	193-201
-	96	J-R	209-217
/	97	S-Z	226-233

OBJ オブジェクト値

オブジェクト値は、MAP と ORDER の 2 つの比較方法のいずれかを使って比較されます。どちらの関数でもオブジェクト型インスタンスは比較されますが、両者はきわめて異なります。これらの関数は、オブジェクト型の一部として指定されなければなりません。

MAP 関数は単一のオブジェクトを引数として取り、スカラー値を戻します。2 つのオブジェクトを比較するために、MAP メソッドはそれぞれのオブジェクトに個々に適用されます。その上で結果が比較されます。MAP は、オブジェクト型をスカラーにマップするハッシュ関数と似ています。

ORDER 関数は、2 つのオブジェクト (たとえば、object1 と object2) を引数にとり、単純に 1 つのオブジェクトを他方のオブジェクトと比較します。ORDER 関数は、object1 が object2 より大きい場合は +1、両者が等しい場合は 0、object1 が object2 より小さい場合は -1 を返します。ORDER メソッドによって NULL 値が戻ることはありません。

オブジェクト・インスタンスに対して大量のソートやハッシュ結合の操作を実行する場合には、MAP を使用します。MAP は、オブジェクトをスカラー値にマップするために 1 度だけ

適用され、その後のソートやマージではスカラーが使用されます。MAP メソッドは、オブジェクトの比較が生じるたびに呼び出さなければならない ORDER メソッドよりも効率的です。ハッシュ結合の場合には必ず MAP メソッドを使用します。ハッシュ結合ではオブジェクト値に関してハッシングを行うので、ORDER メソッドは使用できません。

オブジェクト指定には、1 つの比較方法だけを含めることができます。その比較方法は関数でなければなりません。型指定に MAP メソッドまたは ORDER メソッドのいずれか（両方ではない）を定義できます。

2 つのオブジェクト型が等しいかどうかを判断するために、比較方法を指定する必要はありません。

詳細は、「CREATE TYPE コマンド」（4-342 ページ）および『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

VARRAY とネストした表

Oracle8 では、VARRAY とネストした表は比較できません。

データ変換

一般に、式には異なるデータ型の値を含めることができません。たとえば、式では 10 を 5 に乗じた結果に 'JAMES' を加えることはできません。ただし、Oracle では値をあるデータ型から別のデータ型へ変換する場合に、暗黙の変換と明示的な変換がサポートされています。

暗黙のデータ変換

あるデータ型から別のデータ型への変換が意味を持つ場合、Oracle は値を自動的に変換します。Oracle は、次の場合にデータ型を変換します。

- INSERT 文または UPDATE 文で、あるデータ型の値を別のデータ型の列に割り当てる場合。Oracle はその値を列のデータ型に変換します。
- SQL 関数または演算子に不当なデータ型の引数を指定して使用する場合。Oracle はその引数を正当なデータ型に変換します。
- 異なるデータ型の値に比較演算子を使用する場合。Oracle は式の一方を他方のデータ型に変換します。

例 1 テキスト・リテラル '10' は CHAR データ型です。次の文のように数式で使用すると暗黙的に NUMBER データ型に変換されます。

```
SELECT sal + '10'
    FROM emp;
```

例 2 条件で文字値と NUMBER 型の値を比較する場合、NUMBER 型の値は文字値に変換されず、文字値が暗黙的に NUMBER 型の値に変換されます。次の文では、'7936' が暗黙的に 7936 に変換されます。

```
SELECT ename
    FROM emp
    WHERE empno = '7936';
```

例 3 次の文では、Oracle がデフォルトの日付書式 'DD-MON-YYYY' を使用して、'12-MAR-1993' を DATE 値に暗黙的に変換します。

```
SELECT ename
    FROM emp
    WHERE hiredate = '12-MAR-1993';
```

例 4 次の文では、Oracle がテキスト・リテラル 'AAAAZ8AABAAABvlAAA' を ROWID 値に暗黙的に変換します。

```
SELECT ename
    FROM emp
    WHERE ROWID = 'AAAAZ8AABAAABvlAAA';
```

明示的なデータ変換

SQL 変換関数を使用して、データ型の変換を明示的に指定できます。表 2–8 に、あるデータ型から別のデータ型に明示的に値を変換する SQL 関数を示します。

表 2–8 データ型変換の SQL 関数

変換後 変換前	CHAR	NUMBER	DATE	RAW	ROWID
CHAR	—	TO_NUMBER	TO_DATE	HEXTORAW	CHARTOROWID
NUMBER	TO_CHAR	—	TO_DATE (number,'J')		
DATE	TO_CHAR	TO_CHAR (date,'J')	—		
RAW	RAWTOHEX			—	
ROWID	ROWIDTOCHAR				—

これらの関数の詳細は、「変換関数」（3-39 ページ）を参照してください。

注意： 表 2–8 には LONG 型と LONG RAW 型の値からの変換が示されていません。LONG と LONG RAW の値を指定すると、Oracle で暗黙のデータ型変換を行うことができないためです。たとえば、関数や演算子を含む式では LONG と LONG RAW の値を使用できません。LONG データ型および LONG RAW データ型の制限については、「LONG データ型」（2-11 ページ）を参照してください。

暗黙のデータ変換と明示的なデータ変換

次の理由で、暗黙の変換または自動変換に頼るのではなく、明示的な変換を指定することをお薦めします。

- 明示的なデータ型変換関数を使用すると、SQL 文がわかりやすくなる。
- 自動的なデータ型変換（特に列値のデータ型が定数に変換される場合）は、パフォーマンスに悪影響を及ぼす可能性がある。
- 暗黙の変換はその変換が起こるコンテキストに依存し、どんな場合でも同じように機能するとは限らない。
- 暗黙の変換のアルゴリズムは、ソフトウェア・リリースや Oracle 製品の変更によって変更されることがある。明示的な変換を指定しておくと、その動作は将来的にも確実になる。

NULL

行内のある列の値がない場合、その列は *NULL* である、または NULL を含むと言います。NOT NULL 整合性制約または PRIMARY KEY 整合性制約によって制限されていない列であれば、どのデータ型の列でも NULL を含むことができます。実際のデータ値が不定または値に意味がない場合に、NULL を使用してください。

NULL は値ゼロと等価ではないので、ゼロを表すために NULL 値を使用しないでください。(現在 Oracle は長さがゼロの文字値を NULL として処理します。ただし、この処理は Oracle の将来のバージョンでも継続されるとは限りませんので、空の文字列を NULL として処理しないことをお薦めします。)NULL を含む算術式は必ず NULL に評価されます。たとえば、NULL に 10 を加算しても結果は NULL です。実際、オペランドに NULL を指定すると、（連結演算子を除く）すべての演算子は NULL を戻します。

SQL 関数での NULL

引数として NULL を指定すると、(NVL と TRANSLATE を除く) すべてのスカラー関数では NULL が戻されます。NVL 関数を使用すると、NULL が発生したときに値を戻すことができます。たとえば、式 NVL(COMM,0) は、COMM が NULL であれば 0 を戻し、COMM が NULL でなければ COMM の値を戻します。

ほとんどのグループ関数では NULL は無視されます。たとえば、1000 および NULL、NULL、NULL、2000 という 5 つの値の平均を得る問合せを考えます。そのような問合せでは NULL は無視され、平均は (1000+2000)/2=1500 となります。

比較演算子での NULL

NULL を検査するには、比較演算子 IS NULL および IS NOT NULL だけを使用します。NULL を他の演算子で使用して、その結果が NULL の値に依存する場合、結果は UNKNOWN になります。NULL はデータの欠落を表すため、任意の値や別の NULL との関係で等号や不等号は成り立ちません。ただし、Oracle は DECODE 式を評価するときに 2 つの NULL を等しい値とみなすので注意してください。DECODE の構文については、「式」（3-73 ページ）を参照してください。

複合キーの場合、2 つの NULL は等しいと判断されます。言い換えると、NULL を含んだ 2 つの複合キーは、そのキーの NULL でないコンポーネントのすべてが等しい場合、同一であると判断されます。

条件での NULL

UNKNOWN として評価される条件は、FALSE と評価される場合とほとんど同じ働きをします。たとえば、UNKNOWN と評価される条件を WHERE 句に持つ SELECT 文からは、行が戻されません。しかし、UNKNOWN に評価される条件は FALSE 条件とは異なり、UNKNOWN 条件をさらに評価しても UNKNOWN に評価されます。したがって、NOT FALSE は TRUE に評価されますが、NOT UNKNOWN は UNKNOWN に評価されます。

表 2–9 は、条件に NULL を含んだ評価の例です。SELECT 文の WHERE 句で UNKNOWN と評価される条件が使用された場合、その問合せに対して行は戻されません。

表 2–9　NULL を含む条件

A の値	条件	評価結果
10	a IS NULL	FALSE
10	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
10	a = NULL	UNKNOWN
10	a != NULL	UNKNOWN
NULL	a = NULL	UNKNOWN
NULL	a != NULL	UNKNOWN
NULL	a = 10	UNKNOWN
NULL	a != 10	UNKNOWN

NULL を含む論理式の結果を示した真理値表は、表 3–6（3-11 ページ）および 表 3–7 と 表 3–8 を参照してください。

疑似列

疑似列は表の列のように使用できますが、実際に表に格納されているわけではありません。疑似列から値を選択できますが、疑似列に対して値の挿入、更新、削除はできません。この項では次の疑似列について説明します。

- CURRVAL と NEXTVAL
- LEVEL
- ROWID
- ROWNUM

CURRVAL と NEXTVAL

「順序」は一意の連続値を生成できるスキーマ・オブジェクトです。これらの値は主キーや一意のキーによく使用されます。次の疑似列を使用した SQL 文で、順序値を参照できます。

CURRVAL　　順序の現在の値を戻します。

NEXTVAL　　順序を増加させて次の値を戻します。

CURRVAL と NEXTVAL は、順序の名前で修飾しなければなりません。

```
sequence.CURRVAL
sequence.NEXTVAL
```

別のユーザーのスキーマ内での順序の現在値または次にくる値を参照するには、その順序に対する SELECT オブジェクト権限または SELECT ANYSEQUENCE システム権限のどちらかが必要です。さらに、その順序は次に示すように順序を含むスキーマで修飾しなければなりません。

```
schema.sequence.CURRVAL
schema.sequence.NEXTVAL
```

リモート・データベース上の順序の値を参照するには、次のように、データベース・リンクの完全な名前または部分的な名前で順序を修飾しなければなりません。

```
schema.sequence.CURRVAL@dblink
schema.sequence.NEXTVAL@dblink
```

データベース・リンクの参照については、「リモート・データベース内のオブジェクトを参照」（2-50 ページ）を参照してください。

Trusted Oracle を DBMS MAC モードで使用している場合、順序を参照するには、DBMS ラベルが順序の作成ラベルより上位であるか、次のいずれかの基準を満たしていなければなりません。

- 順序の作成ラベルが DBMS ラベルよりも高い場合、READUP システム権限と WRITEUP システム権限を持つ。
- 順序の作成ラベルと DBMS ラベルが一致していない場合、READUP、WRITEUP、WRITEDOWN の各システム権限を持つ。

Trusted Oracle を OS MAC モードで使用している場合、DBMS ラベルよりも低い作成ラベルを持つ順序は参照できません。

順序値の使用場所

次の場所で CURRVAL と NEXTVAL を使用できます。

- 副問合せまたはスナップショット、ビューに含まれていない SELECT 文の SELECT リスト
- INSERT 文内の副問合せの SELECT リスト
- INSERT 文の VALUES 句
- UPDATE 文の SET 句

次の場所では CURRVAL と NEXTVAL を使用できません。

- DELETE 文または SELECT 文、UPDATE 文内の副問合せ
- ビューの問合せまたはスナップショットの問合せ

- DISTINCT 演算子を持つ SELECT 文
- GROUP BY 句または ORDER BY 句を持つ SELECT 文
- 集合演算子 UNION または INTERSECT、MINUS によって別の SELECT 文と結合されている SELECT 文
- SELECT 文の WHERE 句
- CREATE TABLE 文または ALTER TABLE 文の列の DEFAULT 値
- CHECK 制約の条件

また、CURVAL または NEXTVAL を使用する単一の SQL 文では、参照対象の LONG 列、更新対象の表、ロック対象の表がすべて同じデータベース上になければなりません。

順序値の使用方法

順序を作成するときに、初期値と増分値を定義できます。NEXTVAL の最初の参照によって、順序の初期値が戻されます。その後の参照によって、定義された増分値で順序が増加され、その新しい値が戻されます。CURRVAL を参照すると、NEXTVAL への最後の参照で戻された値である、順序の現在の値が常に戻されます。なお、セッションの順序に対して CURRVAL を使用する前に、まず NEXTVAL で順序を初期化してください。

単一の SQL 文の中で、Oracle は一度だけ順序を増加させます。1 つの文の中で順序に対して複数回 NEXTVAL を参照している場合、Oracle は一度だけ順序を増加させ、NEXTVAL が現れる度にすべて同じ値を戻します。1 つの文の中で CURRVAL と NEXTVAL の両方を参照している場合、Oracle は順序を増加させ、それらが文の中で現れる順序にかかわらず、CURRVAL と NEXTVAL の両方に対して同じ値を戻します。

順序には、待機またはロックすることなく数多くのユーザーが同時にアクセスできます。順序については、「CREATE SEQUENCE コマンド」（4-278 ページ）を参照してください。

例 1　この例は従業員順序の現在の値を選択します。

```
SELECT empseq.currval
    FROM DUAL;
```

例 2　この例は従業員順序を増加させ、従業員表に挿入される新しい従業員のためにその値を使用します。

```
INSERT INTO emp
    VALUES (empseq.nextval, 'LEWIS', 'CLERK',
             7902, SYSDATE, 1200, NULL, 20);
```

例 3　この例は次の注文番号を使用して新しい注文をマスター注文表に追加します。次にこの番号を使用して関連する注文をディテール注文表に追加します。

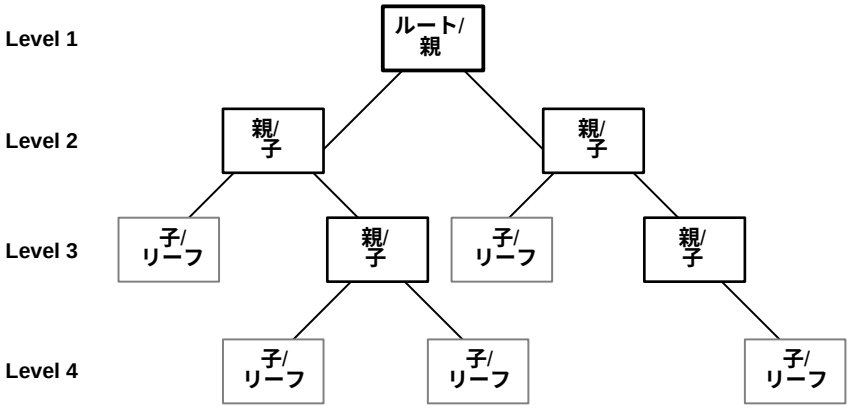
```
INSERT INTO master_order(orderno, customer, orderdate)
    VALUES (orderseq.nextval, 'Al''s Auto Shop', SYSDATE);
```

```
INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'SPARKPLUG', 4);

INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'FUEL PUMP', 1);

INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'TAILPIPE', 2);
```

LEVEL

階層問合せによって戻される各行について、LEVEL 疑似列は、ルート・ノードに 1 を戻し、ルートの子には 2 を戻します（以下同様です）。ルート・ノードは逆ツリー構造の最上位ノードです。子ノードはルートではない任意のノードです。親ノードは子を持つ任意のノードです。リーフ・ノードは子を持たない任意のノードです。図 2–1 に逆ツリーのノードとそれらの LEVEL 値を示します。

図 2–1　階層ツリー



問合せで階層型の関連を規定するためには、START WITH 句と CONNECT BY 句を使用しなければなりません。LEVEL 疑似列の使用については、「SELECT コマンド」（4-486 ページ）を参照してください。

ROWID

データベース内の各行について、ROWID 疑似列により行のアドレスが戻されます。Oracle8 の ROWID 値には、行を捜し出すために必要な次の情報が含まれています。

- オブジェクトのデータ・オブジェクト番号

- データ・ファイル内のデータ・ブロック
- データ・ブロック内の行 (最初の行は 0)
- データ・ファイル (最初のファイルは 1)。ファイル番号は表領域に対して相対的です。

ほとんどの場合、ROWID 値ではデータベース内の行は一意に識別されます。ただし、同じクラスタに格納されている異なる表の行には、同じ ROWID を持つことができます。

ROWID 疑似列の値はデータ型 ROWID を持ちます。ROWID データ型については、「ROWID データ型」（2-16 ページ）を参照してください。

ROWID 値には、次の重要な用途があります。

- 単一の行をアクセスする最も速い方法。
- 表の行が格納されている様子を示すことができる。
- 表の行に対する一意の識別子。

表の主キーとして ROWID を使用しないでください。たとえば、Import ユーティリティと Export ユーティリティで行を削除してから再挿入する場合、ROWID は変わる場合があります。行を削除した場合、Oracle は、後から新たに挿入される行にその ROWID を再度割り当てる可能性があります。

問合せの SELECT 句と WHERE 句で ROWID 疑似列を使用できますが、これらの疑似列の値が実際にデータベースに格納されるわけではありません。ROWID 疑似列の値に対して挿入および更新、削除はできません。

例 次の文は、部門 20 の従業員のデータを含むすべての行のアドレスを選択します。

```
SELECT ROWID, ename
    FROM emp
    WHERE deptno = 20;

ROWID              ENAME
------------------ ----------
AAAAfSAABAAAClaAAA SMITH
AAAAfSAABAAAClaAAD JONES
AAAAfSAABAAAClaAAH SCOTT
AAAAfSAABAAAClaAAK ADAMS
AAAAfSAABAAAClaAAM FORD
```

ROWNUM

問合せによって戻される各行について、ROWNUM 疑似列により表や結合処理された行の集合から Oracle が行を選択する順序を示す番号が戻されます。つまり、選択される最初の行の ROWNUM は 1、2 番目の行の ROWNUM は 2 です（以下同様です）。

次の例のように、ROWNUM を使用して問合せによって戻される行数を制限できます。

```
SELECT *
    FROM emp
    WHERE ROWNUM < 10;
```

比較条件「ROWNUM 値＞正の整数」は常に偽となるので注意してください。たとえば、次の問合せでは行は戻されません。

```
SELECT * FROM emp
    WHERE ROWNUM > 1;
```

最初にフェッチされる行の ROWNUM には 1 が割り当てられるため、条件は偽と判定されます。さらに、2 番目にフェッチされる予定であった行は最初の行になるため、この ROWNUM にも 1 が割り当てられ、条件も偽と判定されます。このように、後続するすべての行が条件を満たさないため、行はまったく戻されません。

また、次の例のように、ROWNUM を使用して表の各行に一意の値を割り当てることもできます。

```
UPDATE tabx
    SET col1 = ROWNUM;
```

Oracle は検索した各行に ROWNUM 値を割り当ててから、ORDER BY 句に従って行をソートします。そのため、ORDER BY 句が各行の ROWNUM に影響を及ぼすことはありません。ただし、ORDER BY 句の指定によって Oracle が索引を使用してデータにアクセスする場合、索引なしの場合とは異なる順序で行が取り出されることがあります。そのため、ROWNUM は ORDER BY 句なしの場合とは異なる値になることもあります。

注意： 問合せで ROWNUM を使用すると、ビューの最適化に影響を及ぼすことがあります。詳細は、『Oracle8 概要』を参照してください。

コメント

SQL 文とスキーマ・オブジェクトに対してコメントを付けることができます。

SQL 文中のコメント

SQL 文中のコメントは、文の実行には影響を及ぼしませんが、アプリケーションを読みやすく、メンテナンスしやすくできます。文にはアプリケーションでのその文の目的を記述したコメントを含めることができます。

コメントは、文中のキーワードまたはパラメータ、句読点の間に入れることができます。次のどちらかの方法を使用します。

- /* を使用してコメントを開始する。コメントのテキストを続けます。このテキストは複数行に及んでもかまいません。*/ でコメントを終了します。開始文字と終了文字は、空白や改行によってテキストから切り離す必要はありません。

- --(ハイフン 2 個) を使用してコメントを開始する。コメントのテキストを続けます。このテキストは複数行にまたがることはできません。改行によってコメントを終了します。

SQL 文の中に両方のスタイルのコメントが複数あってかまいません。コメントのテキストには、使用しているデータベースのキャラクタ・セットの印字可能文字を含めることができます。

注意： SQL スクリプト内の SQL 文では、このようなスタイルのコメントを使用できません。この場合、Server Manager または SQL*Plus の REMARK コマンドを使用してください。これらのコマンドの説明は、『Oracle Server Manager User's Guide』または『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

例 次の文には多くのコメントが含まれています。

```
SELECT ename, sal + NVL(comm, 0), job, loc
/* Select all employees whose compensation is
greater than that of Jones.*/
  FROM emp, dept
       /*The DEPT table is used to get the department name.*/
  WHERE emp.deptno = dept.deptno
    AND sal + NVL(comm,0) >   /* Subquery: */
   (SELECT sal + NLV(comm,0)
                              /* total compensation is sal + comm */
      FROM emp
      WHERE ename = 'JONES')

SELECT ename, -- select the name
    sal + NVL(comm, 0),           -- total compensation
    job,                           -- job
    loc -- and city containing the office
  FROM emp, -- of all employees
       dept
  WHERE emp.deptno = dept.deptno
    AND sal + NVL(comm, 0) >     -- whose compensation
                                -- is greater than
      (SELECT sal + NVL(comm,0)  -- the compensation
    FROM emp
    WHERE ename = 'JONES')       -- of Jones.
```

スキーマ・オブジェクトに関するコメント

このマニュアルの第 4 章「コマンド」に記述されている COMMENT コマンドを使用して、表またはビュー、スナップショット、列にコメントを付けることができます。スキーマ・オブジェクトに付けたコメントは、データ・ディクショナリに格納されます。

ヒント

Oracle オプティマイザに指示（すなわちヒント）を引き渡すために、SQL 文中でコメントを使用できます。オプティマイザは、これらのヒントを使用して文の実行計画を選択します。

文ブロックには、ヒントを含んだコメントは 1 つしか指定できません。このコメントは、SELECT または UPDATE、INSERT、DELETE のいずれかのキーワードの後でだけ指定できます。次の構文は、Oracle が文ブロック内でサポートする両方のスタイルのコメントに含まれるヒントの構文です。

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */
```

または

```
{DELETE|INSERT|SELECT|UPDATE} --+ hint [text] [hint[text]]...
```

ここで、それぞれの意味は次のとおりです。

DELETE INSERT SELECT UPDATE	文ブロックを始める DELETE または INSERT、SELECT、UPDATE のいずれかのキーワードです。ヒントを含むコメントは、これらのキーワードの後でだけ指定できます。
+	Oracle にコメントをヒントのリストとして解釈させるようにするためのプラス記号です。プラス記号は、コメント・デリミタの直後（空白は認められない）になければなりません。
hint	この項と『Oracle8 Server チューニング』で説明するヒントの 1 つです。プラス記号とヒントの間の空白は入れても入れなくてもかまいません。コメントに複数のヒントが含まれている場合は、ヒントの対ごとに 1 つ以上の空白で区切る必要があります。
text	ヒントに含めることができるその他のコメント・テキストです。

表 2–10 に、ヒントの構文と説明を示します。ヒントの詳細は、『Oracle8 Server チューニング』および『Oracle8 Parallel Server 概要および管理』、『Oracle8 Server 概要』を参照してください。

表 2–10　ヒントの構文と説明

ヒントの構文	説明
最適化方法と目標	
`/*+ ALL_ROWS */`	最高のスループットを実現する（つまり、リソース使用の合計を最小限にする）という目標で文ブロックを最適化するために、コストベース方法を明示的に選択する。
`/*+ CHOOSE */`	各 SQL 文について、ルールベース方法とコストベース方法のいずれかをオプティマイザが選択する。どちらを選択するかは、この文による表アクセスの統計情報を利用する。
`/*+ FIRST_ROWS */`	最良の応答時間を実現する（つまり、最初の行を戻すために使用するリソースを最小限にする）という目標で文ブロックを最適化するために、コストベース方法を明示的に選択する。
`/*+ RULE */`	文ブロックのためにルールベース最適化を明示的に選択する。
アクセス方法のヒント	
`/*+ AND_EQUAL(table index) */`	複数の単一列索引の走査結果をマージするアクセス・パスを使う実行計画を明示的に選択する。
`/*+ CLUSTER(table) */`	指定された表にアクセスするためにクラスタ走査を明示的に選択する。
`/*+ FULL(table) */`	指定された表に対する全表走査を明示的に選択する。
`/*+ HASH(table) */`	指定された表にアクセスするためにハッシュ走査を明示的に選択する。
`/*+ HASH_AJ(table) */`	指定された表にアクセスするために、NOT IN 副問合せをハッシュ非結合に変換する。
`/*+ HASH_SJ (table) */`	指定された表にアクセスするために、NOT IN 副問合せをハッシュ非結合に変換する。
`/*+ INDEX(table index) */`	指定された表に対する索引走査を明示的に選択する。
`/*+ INDEX_ASC(table index) */`	指定された表に対する昇順範囲索引走査を明示的に選択する。
`/*+ INDEX_COMBINE(table index) */`	索引が INDEX_COMBINE ヒントの引数として指定されていない場合は、推定コストが最良であるビットマップ索引の組合せをオプティマイザが使用する。特定の索引が引数として指定されている場合は、オプティマイザは、指定されたビットマップ索引の組合せを次々試す。
`/*+ INDEX_DESC(table index) */`	指定された表に対する降順範囲索引走査を明示的に選択する。
`/*+ INDEX_FFS(table index) */`	全表走査ではなく、高速全索引走査を実行する。

表 2–10　ヒントの構文と説明

ヒントの構文	説明
`/*+ MERGE_AJ (table) */`	指定された表にアクセスするために、NOT IN 副問合せをマージ非結合に変換する。
`/*+ MERGE_SJ (table) */`	指定された表にアクセスするために、相関 EXISTS 副問合せをマージ半結合に変換する。
`/*+ ROWID(table) */`	指定された表に対する ROWID での表走査を明示的に選択する。
`/*+ USE_CONCAT */`	問合せの WHERE 句内の結合 OR 条件を、UNION ALL 集合演算子を使って複合問合せに変換する。
結合順序のヒント	
`/*+ ORDERED */`	表が FROM 句に指定されている順序で結合されるようにする。
`/*+ STAR */`	索引のネストされたループの結合を使うことにより、大きな表が最後に結合されるようにする。
結合操作のヒント	
`/*+ DRIVING_SITE (table) */`	問合せ処理が Oracle により選択されたサイトとは異なるサイトで実行されるようにする。
`/*+ USE_HASH (table) */`	ハッシュ結合を使って、指定された各表を別の行ソースに結合させる。
`/*+ USE_MERGE (table) */`	ソート・マージ結合を使って、指定された各表を別の行ソースに結合させる。
`/*+ USE_NL (table) */`	ネストされたループの結合を使って、指定された各表を別の行ソースに結合させる。この場合、指定された表を内部表として使用する。
パラレル実行のヒント	
`/*+ APPEND */` `/*+ NOAPPEND */`	データが単に表に追加される（または追加されない）ように指定する。これにより、既存の空き領域は使用されない。これらのヒントは INSERT キーワードの後でだけ使用する。
`/*+ NOPARALLEL(table) */`	表が PARALLEL 句を使って作成された場合でも、表のパラレル走査を使用できないようにする
`/*+ PARALLEL(table, instances) */`	操作に使用できる同時発生のスレーブ・プロセスの数を希望の数に指定できるようにする。 DELETE および INSERT、UPDATE の各操作は、セッションが PARALLEL DML 使用可能モードであるときにだけ、パラレル化とみなされる。（このモードに入るには、ALTER SESSION PARALLEL DML を使用する。）

表 2–10　ヒントの構文と説明

ヒントの構文	説明
/*+ PARALLEL_INDEX */	PARALLEL 属性を持つ分割された索引および分割されていない索引に対して高速全索引操作をパラレル化できるようにする。
/*+ NOPARALLEL_INDEX */	索引に対する PARALLEL 属性の設定を上書きする。
その他のヒント	
/*+ CACHE */	全表操作が実行されると、ヒント内の表のために取り出されたブロックは、バッファ・キャッシュ内の LRU リストの最大使用頻度の場所に置かれる。
/*+ NOCACHE */	全表操作が実行されると、この表のために取り出されたブロックは、バッファ・キャッシュ内の LRU リストの最低使用頻度の場所に置かれる。
/*+ MERGE (table) */	周囲の問合せの前に複雑なビューまたは副問合せを評価するようにする。
/*+ NO_MERGE (table) */	マージ可能なビューをマージしないようにする。
/*+ PUSH_JOIN_PRED (table) */	オプティマイザが、個々の結合述語をビューにプッシュするかどうかを、コスト・ベースで評価するようにする。
/*+ NO_PUSH_JOIN_PRED (table) */	結合述語のビューへのプッシュをしないようにする。
/*+ PUSH_SUBQ */	マージされていない副問合せを、実行計画の中で可能な限り最も早く評価する。
/*+ STAR_TRANSFORMATION */	型変換が適用された最適な計画がオプティマイザにより使用されるようにする。

データベース・オブジェクト

スキーマ・オブジェクト

スキーマは、論理的なデータの構造またはスキーマ・オブジェクトの集まりです。スキーマは、データベース・ユーザーによって所有され、そのユーザーと同じ名前を持ちます。各ユーザーは、1 つのスキーマを所有します。スキーマ・オブジェクトは、SQL を使って作成したり操作したりできます。スキーマ・オブジェクトには次のタイプのオブジェクトがあります。

- クラスタ
- データベース・リンク
- データベース・トリガー
- 外部プロシージャ・ライブラリ

- 索引構成表
- 索引
- パッケージ
- 順序
- ストアド・ファンクション
- ストアド・プロシージャ
- シノニム
- 表
- ビュー

さらに、Oracle の分散オプションを使っている場合には、次のスキーマ・オブジェクトが利用できます。

- スナップショット
- スナップショット・ログ

さらに、データベース・サーバーにオブジェクト・オプションがインストールされている場合には、次のスキーマ・オブジェクトが利用できます。

- オブジェクト表
- オブジェクト型
- オブジェクト・ビュー

非スキーマ・オブジェクト

また、次のタイプのオブジェクトもデータベースに格納され、SQL で作成、操作されますが、スキーマには含まれません。

- ディレクトリ
- プロファイル
- ロール
- ロールバック・セグメント
- 表領域
- ユーザー

各タイプのオブジェクトは、このマニュアルの第 4 章「コマンド」にある、データベース・オブジェクトを作成するコマンドの項で簡単に定義されています。これらのコマンドは、キーワード CREATE で始まります。たとえば、クラスタの定義については、「CREATE

CLUSTER コマンド」（4-202 ページ）を参照してください。データベース・オブジェクトの概要については、『Oracle8 概要』を参照してください。

大部分のスキーマ・オブジェクトでは、作成時に名前を指定する必要があります。名前は、以降の項に示す規則に従って付けてください。

スキーマ・オブジェクトの部分

スキーマ・オブジェクトの中には、次に示すように名前を付ける必要のある部分から構成されるものもあります。

- 表やビューの中の列
- 索引と表のパーティション
- 表に対する整合性制約
- パッケージ・プロシージャおよびパッケージ・ストアド・ファンクション、パッケージ内に格納されるその他のオブジェクト

パーティション表と索引

表と索引はパーティション化できます。パーティション化されたスキーマ・オブジェクトは、パーティションと呼ばれる多数の部分で構成され、各パーティションの論理属性はすべて同じです。たとえば、表のパーティションはすべて同じ列定義と制約定義を共有し、索引のパーティションはすべて同じ索引列を共有します。

拡張パーティション表名

パーティションは表として使用できます。このため、一部のパーティション・レベルの操作を簡単に実行できます。他の方法では WHERE 句の述語が必要となります。

パーティションを表として使用するには、単一のパーティションからデータを選択してビューを作成し、そのビューを表として使用します。この方法の利点は、これらのビューに対する権限を他のユーザーやロールに付与する（または取り消す）ことによって、パーティション・レベルのアクセス制御機構を構築できる点です。

パーティション・レベルの大量データの操作（たとえば、パーティションからすべての行を削除する操作など）は、1 つのパーティションに対する操作だけに限られます。このようなタイプの操作は、表名の構文で簡単に表されます。同じ操作を WHERE 句の述語で表そうとすると、特にレンジ・パーティション・キーで複数の列を使用しているときに、とても複雑になる可能性があります。

次の DML 文では、表指定の構文が拡張され、リモートでないパーティション表に対するオプションのパーティション指定ができるようになりました。

- DELETE
- INSERT
- LOCK TABLE

- SELECT
- UPDATE

注意： アプリケーションの移植性と ANSI 構文準拠を考慮し、この Oracle 特有の拡張部分からアプリケーションを切り離すためにはビューを使用することをお薦めします。

現在、拡張パーティション表名を使用するときには、次の制限があります。

- リモート表なし : 拡張パーティション表名には、データベース・リンク (dblink)、または dblink を使って表に変換するシノニムを含めることはできません。リモート・パーティションを使用するには、拡張パーティション表名の構文を使ってリモート・サイトにビューを作成し、そのリモート・ビューを参照します。
- ダイレクト PL/SQL サポートなし : 拡張パーティション表名の構文を使用する SQL 文は、DBMS_SQL パッケージを使って動的 SQL を介して使用できますが、PL/SQL ブロック内では使用できません。PL/SQL ブロック内のパーティションを参照するには、拡張パーティション表名の構文を使用するビューを使用します。
- シノニムなし： パーティション拡張は、実表を使って指定する必要があります。シノニムまたはビュー、その他のオブジェクトは使用できません。

構文 拡張パーティション表名を使用する基本構文は次のとおりです。

```
[schema.]{table | view} [@dblink | PARTITION (partition_name)]
```

例 次の文では、SALES がパーティション JAN97 を持つパーティション表です。シングル・パーティション JAN97 のビューを作成でき、それをあたかも表のように使用できます。この例では、パーティションから行が削除されます。

```
CREATE VIEW sales_jan97 AS
     SELECT * FROM sales PARTITION (jan97);
DELETE FROM sales_jan97 WHERE amount < 0;
```

スキーマ・オブジェクト名および修飾子

この項では、次の内容について説明します。

- スキーマ・オブジェクトとスキーマ・オブジェクト位置修飾子の命名規則
- スキーマ・オブジェクトと修飾子の命名ガイドライン

スキーマ・オブジェクトの命名規則

スキーマ・オブジェクトの命名には次の規則が適用されます。

1. 名前は 1 文字から 30 文字までの長さでなければなりません。ただし、次の 2 つは例外です。
   - データベースの名前は、8 文字までに制限されています。
   - データベース・リンクの名前は、128 文字まで指定できます。
2. 名前には引用符を含むことができません。
3. 名前は大文字と小文字を区別しません。
4. 名前は二重引用符で囲まれていない限り、使用しているデータベースのキャラクタ・セットのアルファベット文字で開始しなければなりません。
5. 名前は使用しているデータベースのキャラクタ・セットの英数字の文字と記号 (_、$、#) だけを含むことができます。データベース・リンクの名前は、ピリオド (.) とアットマーク (@) を含むこともできます。$ と # はできるだけ使用しないでください。

   データベースのキャラクタ・セットがマルチバイト文字を含む場合、ユーザーまたはロールの名前にはシングルバイト文字を最低 1 つ含めることをお薦めします。

   注意： データベース名またはグローバル・データベース名、データベース・リンク名にはヨーロッパまたはアジアのキャラクタ・セットの中の特殊文字は使用できません。たとえば、ウムラートを使用することはできません。

6. 名前には、Oracle の予約語は使用できません。Oracle の予約語の全リストは、付録 C「Oracle の予約語とキーワード」を参照してください。

   名前は、データベース・オブジェクトにアクセスするために使用する Oracle 製品固有のその他の予約語によって、さらに制限されることもあります。製品の予約語のリストについては、『PL/SQL ユーザーズ・ガイドおよびリファレンス』などの各製品のマニュアルを参照してください。
7. DUAL という語をオブジェクトまたはオブジェクトの部分の名前として使用しないでください。DUAL は、ダミー表の名前です。
8. Oracle の SQL 言語には、特別な意味を持つキーワードが含まれています。これらのキーワードは予約語ではないため、オブジェクトおよびオブジェクトの部分の名前として使用できます。しかし、キーワードを名前として使用すると、SQL 文が読みにくいものになります。

   Oracle のキーワードのリストは、付録 C「Oracle の予約語とキーワード」を参照してください。
9. 名前領域内では、2 つのオブジェクトに同じ名前を付けることはできません。

   図 2–2 は、スキーマ・オブジェクトの名前領域を示します。表とビューは同じ名前領域に存在するため、同じスキーマ内の表とビューには同じ名前を付けることはできませ

ん。しかし、表と索引は異なる名前領域に存在するため、同じスキーマ内の表と索引には同じ名前を付けることができます。

データベース内の各スキーマには、その中のオブジェクトのために固有の名前領域があります。たとえば、異なるスキーマ内の 2 つの表は異なる名前領域に存在し、同じ名前を付けることができます。

図 2–2　スキーマ・オブジェクトの名前領域

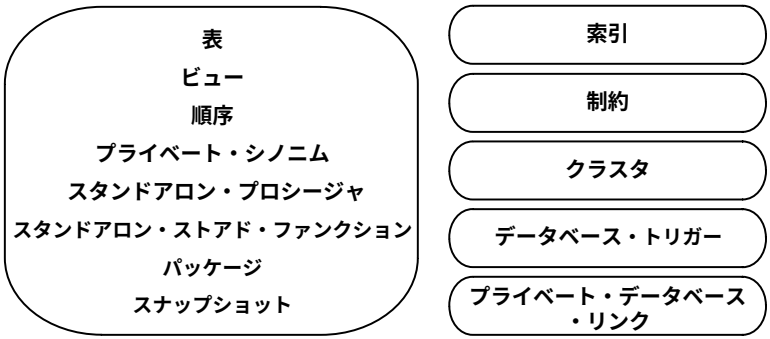
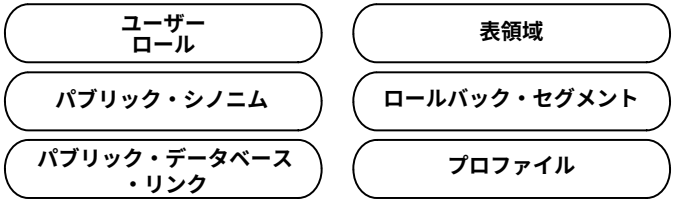


図 2–3 は、非スキーマ・オブジェクトの名前領域を示します。これらの名前領域内のオブジェクトはスキーマに含まれないため、これらの名前領域はデータベース全体で使用されます。

図 2–3　非スキーマ・オブジェクトの名前領域



10. 同じ表やビューでは、複数の列に同じ名前を付けることはできません。しかし、異なる表やビューでは、複数の列に同じ名前を付けることができます。
11. 同じパッケージに含まれるプロシージャや関数には、引数の数やデータ型を変えることによって、同じ名前を付けることができます。異なる引数を持つ、同じ名前のプロシージャや関数を同じパッケージ内に複数作成することを、オーバーロードと言います。
12. 名前は、二重引用符で囲むことができます。その場合、このリストの規則 3 から規則 7 にとらわれずに、名前には空白も含めた任意の文字の組合せを使用できます。移植性を

持たせるためにこのような例外が認められていますが、規則 3 から規則 7 に違反しないことをお薦めします。

二重引用符で囲んだスキーマ・オブジェクト名を使用した場合、そのオブジェクトを参照するときには常に二重引用符を使用しなければなりません。

次に示すような場合には、名前を二重引用符で囲みます。

- 空白を含める。
- 大文字と小文字を区別する。
- 数字のようなアルファベット以外の文字で名前を開始する。
- 英数字および _、$、# 以外の文字を名前に含める。
- 予約語を名前として使用する。

名前を二重引用符で囲むことによって、同じ名前領域内の異なるオブジェクトに対して次の名前を指定できます。

```
emp
"emp"
"Emp"
"EMP "
```

しかし、Oracle は次の名前を同じ名前として解釈するため、同じ名前領域内の異なるオブジェクトには次の名前を使用できません。

```
emp
EMP
"EMP"
```

ユーザーまたはパスワードに引用符で囲んだ名前を付ける場合、その名前に小文字を含めることはできません。

データベース・リンク名を引用符で囲むことはできません。

スキーマ・オブジェクトの命名例

次に、有効なスキーマ・オブジェクト名の例を示します。

```
ename
horse
scott.hiredate
"EVEN THIS & THAT!"
a_very_long_and_valid_name
```

列別名および表別名、ユーザー名、パスワードはオブジェクトやオブジェクトの部分ではありませんが、同様にこれらの命名規則に従わなければなりません。ただし、次のような例外があります。

- 列別名と表別名は、単一の SQL 文の実行時に存在するだけで、データベースには格納されないため、規則 12 は適用されません。
- パスワードには名前領域がないので、規則 9 は適用されません。
- ユーザー名とパスワードで大 / 小文字を区別するために引用符を使用しないでください。ユーザー名とパスワードの命名規則の詳細は、「CREATE USER コマンド」（4-353 ページ）を参照してください。

スキーマ・オブジェクト名の命名ガイドライン

オブジェクトとその部分に名前を付ける場合に役立つガイドラインを次に示します。

- わかりやすい名前（またはよく知られている省略形）を使用する。
- 一貫した命名規則を使用する。
- 複数の表にまたがる同一のエンティティや属性を記述するためには、同一の名前を使用する。

オブジェクトに名前を付けるときには、短く使いやすい名前とわかりやすい名前のバランスを考えてください。迷ったときには、わかりやすい名前を選択してください。データベース内のオブジェクトは、多くの人々が長期間にわたって使用する可能性があるからです。PAYMENT_DUE_DATE のかわりに PMDD という名前を使用すると、10 年後の担当者はデータベースを理解するのに苦労することになるでしょう。

一貫した命名規則を使用すると、アプリケーション上の各表の働きが理解しやすくなります。そのような規則の 1 つの例として、FINANCE アプリケーションに属している表の名前をすべて FIN_ で始めるような場合が考えられます。

同一のエンティティや属性に対しては、複数の表にまたがっていても同じ名前を使用してください。たとえば、EMP 表と DEPT 表の部門番号列には、どちらにも DEPTNO という名前を付けます。

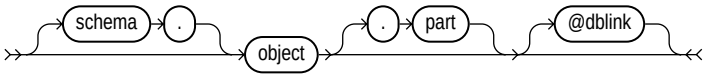
スキーマ・オブジェクトと部分を参照する

SQL 文のコンテキストでスキーマ・オブジェクトとそれらの部分を参照する方法について説明します。次の項目について説明します。

- オブジェクトを参照するための一般的な構文
- Oracle がオブジェクトへの参照を解決する方法
- 自分のスキーマ以外のスキーマ内のオブジェクトを参照する方法
- リモート・データベース内のオブジェクトを参照する方法

次の構文図は、オブジェクトや部分を参照するための一般的な構文を示しています。

schema_object ::=



ここで、それぞれの意味は次のとおりです。

object オブジェクトの名前です。

schema オブジェクトを含むスキーマです。この修飾子を指定することによって、自分のスキーマ以外のスキーマ内のオブジェクトを参照できます。その場合には、自分のスキーマ以外のスキーマ内のオブジェクトを参照するための権限が付与されていなければなりません。この修飾子を指定しないと、自分自身のスキーマ内のオブジェクトを参照するものとみなされます。

スキーマ・オブジェクトだけが *schema* で修飾できます。スキーマ・オブジェクトを図 2–2（2-45 ページ）に示します。図 2–3（2-45 ページ）に示す非スキーマ・オブジェクトはスキーマ・オブジェクトではないため、*schema* では修飾できません。ただし、パブリック・シノニムは例外で、"PUBLIC"(引用符が必要) で修飾できます。

part オブジェクトの部分です。この識別子によって、スキーマ・オブジェクト の部分（たとえば、表の列またはパーティション）を参照できます。なお、すべてのタイプのオブジェクトが部分を持っているわけではありません。

dblink Oracle の分散オプションを使用している場合にだけ適用されます。オブジェクトを含んでいるデータベースの名前です。この修飾子 *dblink* を指定することによって、ローカル・データベース以外のデータベース内のオブジェクトを参照できます。この *dblink* を指定しないと、自分自身のローカル・データベース内のオブジェクトを参照するものとみなされます。なお、すべての SQL 文でリモート・データベースのオブジェクトにアクセスできるとは限りません。

オブジェクトを参照するコンポーネントを区切っているピリオドの前後には、空白を入れることができます。しかし、通常は入れません。

Oracle がスキーマ・オブジェクト参照を解決する方法

SQL 文内のオブジェクトが参照される場合、Oracle はその SQL 文のコンテキストを検討して、該当する名前領域内でそのオブジェクトを突き止めます。そのオブジェクトを突き止め

てから、そのオブジェクトに対して文の操作を実行します。指定した名前のオブジェクトが適切な名前領域内に存在しない場合、Oracle はエラー・メッセージを戻します。

次の例で、Oracle が SQL 文内のオブジェクト参照を解決する方法について説明します。名前 DEPT で識別される表にデータ行を追加する次の文を考えます。

```
INSERT INTO dept
   VALUES (50, 'SUPPORT', 'PARIS');
```

文のコンテキストに基づいて、Oracle は DEPT が次のようなオブジェクトであると判断します。

- 自分のスキーマ内の表
- 自分のスキーマ内のビュー
- 表またはビューに対するプライベート・シノニム
- パブリック・シノニム

Oracle は、文を発行したユーザーのスキーマ外の名前領域を考慮する前に、そのユーザーのスキーマ内の名前領域からオブジェクト参照を解決しようとします。この例では、Oracle は次の方法で名前 DEPT を解決しようとします。

1. Oracle では最初に、表およびビュー、プライベート・シノニムを含む、文を発行したユーザーのスキーマ内の名前領域でオブジェクトを突き止めようとします。オブジェクトがプライベート・シノニムの場合、Oracle はそのシノニムが表すオブジェクトを突き止めます。このオブジェクトは、ユーザー自身のスキーマまたは他のスキーマ、他のデータベースにあることもあります。このオブジェクトが別のシノニムであることもあります。その場合、Oracle はそのシノニムが表すオブジェクトを突き止めます。
2. オブジェクトが名前領域に存在する場合、Oracle はそのオブジェクトに対して文を実行しようとします。この場合、Oracle はデータの行を DEPT に追加しようとします。オブジェクトがその処理にとって正しい型でない場合、Oracle はエラー・メッセージを戻します。この場合、DEPT は、表またはビュー、あるいは表またはビューとなるプライベート・シノニムでなければなりません。DEPT が順序であると、Oracle はエラー・メッセージを戻します。
3. 上記の処理で検索された名前領域にオブジェクトが存在しない場合、Oracle はパブリック・シノニム (図 2–3（2-45 ページ）を参照) を含む名前領域を検索します。オブジェクトがその名前領域に存在すると、Oracle はそのオブジェクトに対して文を実行しようとします。オブジェクトがその処理にとって正しい型でない場合、Oracle はエラー・メッセージを戻します。この例では、DEPT が順序のパブリック・シノニムである場合、Oracle はエラー・メッセージを戻します。

他のスキーマ内のオブジェクトを参照する

自分が所有するスキーマ以外のスキーマ内のオブジェクトを参照するには、次のように、オブジェクト名の前にスキーマ名を付けます。

```
schema.object
```

たとえば、次の文は、スキーマ SCOTT 内の EMP 表を削除します。

```
DROP TABLE scott.emp
```

リモート・データベース内のオブジェクトを参照

ローカル・データベース以外のデータベース内のオブジェクトを参照するには、オブジェクト名にそのデータベースへのデータベース・リンクの名前を続けます。データベース・リンクはスキーマ・オブジェクトであり、これによって Oracle がリモート・データベースに接続され、そこのオブジェクトにアクセスします。この項では次の項目について説明します。

- データベース・リンクを作成する方法
- SQL 文でデータベース・リンクを使用する方法

データベース・リンクを作成する

第 4 章「コマンド」で説明する CREATE DATABASE LINK コマンドを使用して、データベース・リンクを作成できます。このコマンドでは、データベース・リンクに関する次の情報を指定できます。

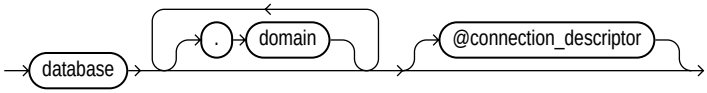
- データベース・リンクの名前
- リモート・データベースにアクセスするためのデータベース接続文字列
- リモート・データベースへ接続するためのユーザー名とパスワード

これらの情報はデータ・ディクショナリに格納されます。

データベース・リンク名 データベース・リンクを作成するとき、データベース・リンク名を指定する必要があります。データベース・リンク名は 128 バイト以内の長さで指定し、ピリオド (.) とアットマーク (@) を使用できます。このように、データベース・リンク名は、他のオブジェクト型の名前とは異なります。

データベース・リンクに付ける名前は、データベース・リンクが参照するデータベースの名前、およびデータベース名の階層内でそのデータベースの位置に一致しなければなりません。データベース・リンク名の書式を次の構文図に示します。

dblink::=



ここで、それぞれの意味は次のとおりです。

database　データベース・リンクが接続するリモート・データベースの名前を指定します。リモート・データベースの名前は、初期化パラメータ DB_NAME によって指定されます。

domain　データベース・リンクが接続するリモート・データベースのドメインを指定します。データベース・リンクの名前にドメインを指定しないと、Oracle は現在データ・ディクショナリに存在しているローカル・データベースのドメイン名をデータベースに付加し、そのリンク名をデータ・ディクショナリに格納します。

connect_descriptor　データベース・リンクをさらに修飾できます。接続修飾子を使用すると、同じデータベースに複数のデータベース・リンクを作成できます。たとえば、接続修飾子を使用して、同じデータベースにアクセスする Oracle パラレル・サーバーの異なるインスタンスに複数のデータベース・リンクを作成できます。

database.domain の組合せは、「サービス名」と呼ばれることもあります。詳細は、『Net8 管理者ガイド』を参照してください。

ユーザー名とパスワード　リモート・データベースへ接続するためにユーザー名とパスワードを使用します。データベース・リンクでは、ユーザー名とパスワードはオプションです。

データベース接続文字列　データベース接続文字列は、Net8 によりリモート・データベースにアクセスするために使用される仕様です。データベース接続文字列の記述方法については、使用しているネットワーク・プロトコル用の Net8 のマニュアルを参照してください。データベース・リンク用のデータベース文字列はオプションです。

データベース・リンクを参照する

データベース・リンクは、分散オプションを指定して Oracle を使用している場合しか利用できません。データベース・リンクを含む SQL 文の発行時に、次のいずれかの書式でデータベース・リンク名を指定します。

完全指定　データ・ディクショナリ内に格納される、データベースおよびドメイン、オプションの接続修飾子を含む完全なデータベース・リンク名を指定します。

部分指定　データベースとオプションの接続修飾子コンポーネントを含むが、ドメイン・コンポーネントを含まないように指定します。

Oracle は、リモート・データベースに接続する前に次のタスクを実行します。

1. 文中に指定されているデータベース・リンク名が部分指定の場合、Oracle は、データ・ディクショナリ内に格納されているグローバル・データベース名に見られるように、そのリンク名にローカル・データベースのドメイン名を付加します。現在のグローバル・データベース名は、GLOBAL_NAME データ・ディクショナリ・ビューで見ることができます。

2. Oracle は最初に、文を発行したユーザーのスキーマ内で、文の中のデータベース・リンクと同じ名前を持つプライベート・データベース・リンクを検索し、必要であれば同じ名前を持つパブリック・データベース・リンクを検索します。

   - Oracle は必ず、最初に一致したデータベース・リンク（プライベートまたはパブリック）のユーザー名とパスワードを採用します。最初に一致したデータベース・リンクに対応付けられているユーザー名およびパスワードがあると、Oracle はそれを使用します。対応付けられているユーザー名およびパスワードがない場合、Oracle は現在のユーザー名とパスワードを使用します。

   - 最初に一致したデータベース・リンクに対応付けられたデータベース文字列が存在する場合、Oracle はそのデータベース文字列を使用します。データベース文字列がない場合、Oracle は一致する次の（パブリック）データベース・リンクを検索します。一致するデータベース・リンクが存在しない場合、または一致するリンクに対応付けられているデータベース文字列が存在しない場合、Oracle はエラー・メッセージを戻します。

3. Oracle は、リモート・データベースにアクセスするためにデータベース文字列を使用します。リモート・データベースにアクセスした後で、Oracle は次の条件が両方とも満たされているか確認します。

   - リモート・データベースの名前（初期化パラメータ DB_NAME によって指定される）がデータベース・リンク名のデータベース・コンポーネントと一致する。

   - リモート・データベースのドメイン（初期化パラメータ DB_DOMAIN によって指定される）が、データベース・リンク名のドメイン・コンポーネントと一致する。

   これらの条件が両方とも満たされている場合、Oracle はステップ 2 で選択したユーザー名とパスワードを使用して接続を続行します。それ以外の場合は、Oracle はエラー・メッセージを戻します。

4. データベース文字列およびユーザー名、パスワードを使用した接続が成功すると、Oracle はリモート・データベース上の指定されたオブジェクトにアクセスしようとします。このとき、この項の前半で説明した、オブジェクト参照を解決するための規則および他のスキーマ内のオブジェクトを参照するための規則が使用されます。

リモート・オブジェクトに対する Oracle の名前の解決は、初期化パラメータ GLOBAL_NAMES、および ALTER SYSTEM コマンドと ALTER SESSION コマンドの GLOBAL_NAMES パラメータを使用して、使用可能または使用禁止にできます。

リモートでの名前の解決の詳細は、『Oracle8 Server 分散システム』を参照してください。

オブジェクト型の属性とメソッドの参照

SQL 文でオブジェクト型の属性とメソッドを参照するには、参照を表別名で完全に修飾しなければなりません。次に例を示します。

```
CREATE TYPE person AS OBJECT
   (ssno VARCHAR(20),
    name VARCHAR (10));

CREATE TABLE emptab (pinfo person);
```

次に示すように、SQL 文では、SSNO 属性への参照は表別名を使用して完全に修飾しなければなりません。

```
SELECT e.pinfo.ssno FROM emptab e;

UPDATE emptab e SET e.pinfo.ssno = '510129980'
   WHERE e.pinfo.name = 'Mike';
```

引数を取らないオブジェクト型のメンバー・メソッドを参照する場合は、空のカッコを付けなければなりません。たとえば、AGE が個人型の中のメソッドで、引数を取らないとします。SQL 文でこのメソッドを呼び出すには、次の例のように、空のカッコを付けなければなりません。

```
SELECT e.pinfo.age() FROM emptab e
   WHERE e.pinfo.name = 'Mike';
```

詳細は、『Oracle8 概要』のユーザー定義データ型の項を参照してください。

3

演算子、関数、式、条件

個々のデータ項目を操作する方法について説明します。加算や減算などの標準的な算術演算子に加え、絶対値や文字列の長さを戻すようなあまり一般的でない関数についても説明します。説明する内容は次のとおりです。

- 演算子
- SQL 関数
- ユーザー関数
- 書式モデル
- 式
- 条件

注意： OBJが前についている関数および式、説明は、Oracle Object Option がデータベース・サーバーにインストールされている場合にだけ有効です。

演算子

演算子は、個々のデータ項目を操作し、結果を戻すために使用されます。これらのデータ項目をオペランドまたは引数と呼びます。演算子は特殊な文字またはキーワードで表します。たとえば、乗算演算子はアスタリスク (*) で表し、NULL を検査する演算子はキーワード IS NULL で表します。以降の表に SQL 演算子を記載します。

単項演算子と 2 項演算子

演算子には一般的に次の 2 つのクラスがあります。

単項　　単項演算子はただ 1 つのオペランドについて操作します。単項演算子は次の書式で表されます。

```
operator operand
```

2 項　　2 項演算子は 2 つのオペランドについて操作します。2 項演算子は次の書式で表されます。

```
operand1 operator operand2
```

このほか、特別な書式を持ち、3 つ以上のオペランドを認める演算子もあります。演算子のオペランドに NULL が指定された場合、結果は常に NULL となります。この規則に従わない唯一の演算子が連結演算子 (||) です。

優先順位

優先順位とは、同じ式の中の異なる演算子を Oracle が評価する順序のことです。複数の演算子を含む式を評価するとき、Oracle は優先順位の高い演算子を評価した後で優先順位の低い演算子を評価します。優先順位の等しい演算子は、式の中で左から右に評価されます。

表 3–1 に、SQL 演算子を優先順位のレベルの高い方から順に示します。同じ行にリストされている演算子には同じ優先順位が付けられています。

表 3–1　SQL 演算子の優先順位

演算子	演算
+, -	同一、否定
*, /	乗算、除算
+, -, ||	加算、減算、連結
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	比較
NOT	指数、論理否定
AND	論理積
OR	論理和

例　次の式では、乗算は加算よりも優先順位が高いため、2 と 3 を掛けた結果に 1 が加算されます。

```
1+2*3
```

式の中でカッコを使用して演算子の優先順位を置き換えることができます。Oracle はカッコの内側の式を評価した後で外側の式を評価します。

SQL では集合演算子 (UNION および UNION ALL、INTERSECT、MINUS) もサポートされます。集合演算子により結合されるのは、問合せによって戻される行のセットであり、個々のデータ項目ではありません。集合演算子の優先順位はすべて同じです。

算術演算子

算術演算子を式の中で使用することにより、数値を否定（正負を反転）および加算、減算、乗算、除算できます。演算の結果も数値になります。これらの演算子の中には日付算術で使用されるものもあります。表 3–2 は算術演算子のリストです。

表 3–2　算術演算子

演算子	用途	例
+ -	式の正負を示す。これらは単項演算子である。	`SELECT * FROM orders WHERE qtysold = -1;` `SELECT * FROM emp WHERE -sal < 0;`
* /	乗算、除算する。これらは 2 項演算子である。	`UPDATE emp SET sal = sal * 1.1;`
+ -	加算、減算する。これらは 2 項演算子である。	`SELECT sal + comm FROM emp WHERE SYSDATE - hiredate > 365;`

二重否定や負の数の減算を表現する場合に、算術式の中で、セパレータのない連続した負の符号 (--) は使用しないでください。文字 -- は、SQL 文ではコメントの開始を示すのに使用されるからです。連続した負の符号は空白やカッコで区切ってください。SQL 文の中のコメントについては、「コメント」（2-35 ページ）を参照してください。

連結演算子

連結演算子は文字列を操作するのに使用します。表 3–3 に連結演算子の説明を示します。

表 3–3　連結演算子

演算子	用途	例
||	文字列を連結する。	`SELECT 'Name is ' || ename FROM emp;`

2 つの文字列を連結した結果は別の文字列になります。両方の文字列が CHAR データ型である場合、結果は CHAR データ型の文字列となり、その最大文字数は 2000 です。どちらかの

文字列が VARCHAR2 データ型である場合、結果は VARCHAR2 データ型の文字列となり、その最大文字数は 4000 です。文字列のデータ型にかかわりなく、文字列に後続する空白は連結後も残ります。CHAR データ型と VARCHAR2 データ型の違いについては、「文字データ型」（2-7 ページ）を参照してください。

多くのプラットフォームで、連結演算子には表 3–3 に示すように 2 本の実線垂直バーで表されます。ただし、IBM 社のプラットフォームの中には、この演算子として破線垂直バーを使用するものもあります。異なるキャラクタ・セットを持つシステム間（たとえば ASCII と EBCDIC 間）で SQL スクリプト・ファイルを移動する場合、垂直バーがターゲットの Oracle 環境で必要な垂直バーに変換されない場合があります。オペレーティング・システムやネットワーク・ユーティリティによる変換を制御するのが困難であったり不可能であったりする場合に備えて、Oracle では垂直バー演算子にかわるものとして CONCAT 文字 関数が提供されています。異なるキャラクタ・セットを持つ環境にアプリケーションを移動する場合は、アプリケーションでこの文字関数を使用することをお薦めします。

Oracle は長さがゼロの文字列を NULL として処理しますが、長さがゼロの文字列を別のオペランドと連結すると、その結果は常にもう一方のオペランドになります。結果が NULL になるのは、2 つの NULL 文字列を連結したときだけです。ただし、この処理は Oracle の将来のバージョンでも継続されるとは限りません。NULL となる可能性がある式を連結する場合には、NVL 関数を使用して明示的に式を長さがゼロの文字列に変換してください。

例 次の例では、CHAR 列と VARCHAR2 列を持つ表を作成し、後続する空白のある値とない値を挿入してから、これらの値を連結して選択しています。なお、CHAR 列と VARCHAR2 列では、ともに後続する空白が保存されます。

```
CREATE TABLE tab1 (col1 VARCHAR2(6), col2 CHAR(6),
        col3 VARCHAR2(6), col4 CHAR(6) );

Table created.

INSERT INTO tab1 (col1, col2, col3, col4)
        VALUES (’abc’, ’def ’, ’ghi ’, ’jkl’);

1 row created.

SELECT col1||col2||col3||col4 "Concatenation"
        FROM tab1;

Concatenation
------------------------
abcdef ghi jkl
```

比較演算子

比較演算子は、2 つの式を比較します。比較の結果は、真 (TRUE) または偽 (FALSE)、不定 (UNKNOWN) になります。条件については、「条件」（3-84 ページ）を参照してください。表 3–4 は比較演算子のリストです。

表 3–4　比較演算子

演算子	用途	例
=	等価性の検査。	SELECT * FROM emp WHERE sal = 1500;
!= ^= <> ¬=	不等性の検査。プラットフォームによっては、一部の不等号演算子の書式を使用できない場合もある。	SELECT * FROM emp WHERE sal != 1500;
>	大 / 小の検査。	SELECT * FROM emp WHERE sal > 1500;
<		SELECT * FROM emp WHERE sal < 1500;
>=	以上 / 以下の検査。	SELECT * FROM emp WHERE sal >= 1500;
<=		SELECT * FROM emp WHERE sal <= 1500;
IN	メンバーとの等価性の検査。"= ANY" と等価。	SELECT * FROM emp WHERE job IN ('CLERK','ANALYST'); SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30);
NOT IN	"!=ALL" と等価。メンバーのいずれかが NULL である場合には、FALSE と評価される。	SELECT * FROM emp WHERE sal NOT IN (SELECT sal FROM emp WHERE deptno = 30); SELECT * FROM emp WHERE job NOT IN ('CLERK', ANALYST');

表 3–4　比較演算子

演算子	用途	例
ANY SOME	リスト内の各値または問合せによって戻される各値と値を比較する。=、!=、>、<、<=、>= のいずれかを先に指定しなければならない。 問合せにより行が戻されない場合には FALSE と評価される。	SELECT * FROM emp WHERE sal = ANY (SELECT sal FROM emp WHERE deptno = 30);
ALL	リスト内のすべての値または問合せによって戻されるすべての値と値を比較する。=、!=、>、<、<=、>= のいずれかを先に指定しなければならない。 問合せにより行が戻されない場合には TRUE と評価される。	SELECT * FROM emp WHERE sal >= ALL (1400, 3000);
[NOT] BETWEEN x AND y	*x* 以上 *y* 以下の範囲にある [ない] ことを検査する。	SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000;
EXISTS	副問合せにより行が 1 行以上戻される場合に TRUE と評価される。	SELECT ename, deptno FROM dept WHERE EXISTS (SELECT * FROM emp WHERE dept.deptno = emp.deptno);
x [NOT] LIKE y [ESCAPE 'z']	*x* がパターン *y* と一致する [しない] 場合に TRUE と評価される。*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する。"_" 文字は任意の 1 文字に一致する。パーセント (%) とアンダースコア (_) を除く任意の文字を ESCAPE の後に指定できる。ワイルドカード文字は、エスケープ文字に指定された文字が前についている場合はリテラルとして扱われる。	「LIKE 演算子」（3-7 ページ）を参照。 SELECT * FROM tab1 WHERE col1 LIKE 'A_C/%E%' ESCAPE '/';
IS [NOT] NULL	NULL を検査する。NULL を検査するために使用すべき唯一の演算子。「NULL」（2-28 ページ）を参照。	SELECT ename, deptno FROM emp WHERE comm IS NULL;

NOT IN 演算子および LIKE 演算子については、次の項で詳しく説明します。

NOT IN 演算子

NOT IN 演算子に続くリストの中のいずれかの項目が NULL である場合、すべての行は (UNKNOWN) 不定と評価されます（行はまったく戻されない）。たとえば、次の文ではそれぞれの行に対して文字列 'TRUE' が戻されます。

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15);
```

ただし、次の文では行は戻りません。

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15,null);
```

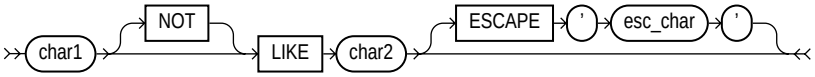
この例で行が戻らないのは、WHERE 句の条件が次のように評価されるからです。

```
deptno != 5 AND deptno != 15 AND deptno != null
```

NULL を比較する条件はすべて NULL になるため、式全体の結果は NULL になります。特に NOT IN 演算子が副問合せを参照するときに、このような作用を見逃してしまう可能性があることに注意してください。

LIKE 演算子

LIKE 演算子は、パターン・マッチングによる文字列の比較で使用されます。次の図は LIKE 演算子を使用する条件の構文を示しています。



ここで、それぞれの意味は次のとおりです。

char1 パターンと比較される値です。この値は CHAR データ型または VARCHAR2 データ型を持ちます。

NOT 条件の結果を論理的に反転させます。つまり、条件が TRUE に評価された場合には FALSE を戻し、FALSE に評価された場合には TRUE を戻します。

char2 char1 の比較対象となるパターンです。パターンは CHAR データ型または VARCHAR2 データ型の値であり、特殊なパターン・マッチング文字 % と _ を含むことができます。

ESCAPE　エスケープ文字として単一の文字を識別します。エスケープ文字を使用して、パターン中で % や _ を特殊文字としてではなくリテラルとして解釈させることができます。

エスケープ文字を含む文字列を検索する場合、そのエスケープ文字を 2 回指定する必要があります。たとえば、エスケープ文字が '/' で、文字列 'client/server' を検索する場合、'client//server' と指定してください。

等号 (=) は、ある文字値全体を別の文字値と適合しますが、LIKE 演算子は、ある文字値の一部を別の文字値と適合します（ある値が指定したパターンの検索を、もう一方の値に対して行う）。LIKE 比較では空白埋めが使用されないので注意してください。

LIKE 演算子では、値を定数ではなくパターンと比較できます。必ず LIKE キーワードの直後に、パターンを指定してください。たとえば、次の問合せを発行することにより、名前が 'SM' で始まる従業員すべての給与を検索できます。

```
SELECT sal
    FROM emp
    WHERE ename LIKE 'SM%';
```

次の問合せは LIKE 演算子のかわりに等号を使用しているため、名前が 'SM%' の従業員すべての給与を検索することになってしまいます。

```
SELECT sal
    FROM emp
    WHERE ename = 'SM%';
```

次の問合せでは名前が 'SM%' の従業員すべての給与が検索されます。この場合、'SM%' が LIKE 演算子の前にあるため、Oracle は 'SM%' をパターンとしてではなく、テキスト・リテラルとして解釈します。

```
SELECT sal
    FROM emp
    WHERE 'SM%' LIKE ename;
```

パターンでは、値の中の異なる文字の代用となる特殊文字がよく使用されます。

- パターン中のアンダースコア (_) は、値の中の 1 文字（マルチバイトのキャラクタ・セットでの 1 バイトとは異なる）の代用となります。
- パターン中のパーセント記号 (%) は、値の中のゼロを含む任意の数の文字（マルチバイトのキャラクタ・セットでの 1 バイトとは異なる）の代用となります。しかし、パターン '%' は NULL と一致しないので注意してください。

大文字 / 小文字の区別とパターン・マッチング　LIKE 演算子と等号 (=) 演算子を含む文字式を比較するすべての条件において、大文字と小文字は区別されます。次の例のように、UPPER() 関数を使用すると、大文字と小文字を区別しないでマッチングできます。

```
UPPER(ename) LIKE 'SM%'
```

索引付きの列でのパターン・マッチング LIKE を使用して索引付きの列をパターン検索する場合、パターンの先頭文字が "%" または "_" でなければ、Oracle では索引を利用して文のパフォーマンスを向上できます。この場合、Oracle はこの先頭文字によって索引を走査できます。パターンの先頭文字が "%" または "_" であると、Oracle は索引を走査できないので、問合せのパフォーマンスは向上しません。

例 1 次の条件は "MA" で始まるすべての ENAME 値について真となります。

```
ename LIKE 'MA%'
```

次の ENAME 値のすべてが条件を満たすと TRUE(真) となります。

```
MARTIN, MA, MARK, MARY
```

大文字と小文字は区別されるので、"Ma" および "ma"、"mA" で始まる ENAME 値では条件が FALSE(偽) になります。

例 2 次の条件を考えます。

```
ename LIKE 'SMITH_'
```

この条件は以下の ENAME 値について真となります。

```
SMITHE, SMITHY, SMITHS
```

特殊文字「_」は ENAME 値の 1 つの文字の代用であるため、この条件は 'SMITH' について偽となります。

ESCAPE オプション ESCAPE オプションを使用すると、パターン中に "%" や "_" を実際の文字として含めることができます。ESCAPE オプションはエスケープ文字を識別します。エスケープ文字がパターンの中で文字 "%" や "_" の前に指定されている場合、Oracle はこの文字を特殊なパターン・マッチング文字としてではなく実際の文字として解釈します。

例 : 次の文は、名前の中に文字列 'A_B' を持つ従業員を検索します。

```
SELECT ename
    FROM emp
    WHERE ename LIKE '%A¥_B%' ESCAPE '¥';
```

ESCAPE オプションはエスケープ文字として円記号 (¥) を識別します。パターンの中でエスケープ文字はアンダースコア (_) に先行します。これによって、Oracle はアンダースコアを特殊なパターン・マッチング文字としてではなくリテラルとして解釈します。

% なしのパターン パターンに文字 "%" が含まれていない場合、両方のオペランドの長さが同じ場合にだけ、条件が TRUE(真) になることが可能です。

例： この表の定義と挿入される値について考えてみます。

```
CREATE TABLE freds (f CHAR(6), v VARCHAR2(6));
INSERT INTO freds VALUES ('FRED', 'FRED');
```

Oracle は CHAR 値に空白埋めを行うので、F の値は空白埋めによって 6 バイトになります。V の値は空白埋めされず、4 文字長のままです。

論理演算子

論理演算子は、2 つのコンポーネントの条件の結果を結合して、両者に基づいた単一の結果を生成するため、または単一の条件の結果を反転させるために使用します。表 3–5 は論理演算子のリストです。

表 3–5　論理演算子

演算子	機能	例
NOT	後続する条件が FALSE の場合に TRUE を戻す。TRUE の場合には FALSE を戻す。UNKNOWN の場合には UNKNOWN を戻す。	`SELECT * FROM emp WHERE NOT (job IS NULL);` `SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000);`
AND	コンポーネントの条件が両方とも TRUE である場合に TRUE を戻す。どちらかが FALSE の場合には FALSE を戻す。それ以外の場合は UNKNOWN を戻す。	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10;`
OR	コンポーネントの条件のどちらかが TRUE である場合に TRUE を戻す。両方とも FALSE の場合には FALSE を戻す。それ以外の場合は UNKNOWN を戻す。	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10;`

たとえば、次の SELECT 文の WHERE 句は、AND 論理演算子を使用して、1984 年より前に入社し、さらに月給が 1,000 ドルを超える従業員だけが選択されることを指定しています。

```
SELECT *
    FROM emp
    WHERE hiredate < TO_DATE('01-JAN-1984', 'DD-MON-YYYY')
        AND sal > 1000;
```

NOT 演算子

表 3–6 は、条件に NOT 演算子を適用した結果を示しています。

表 3–6　NOT の真理値表

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

AND 演算子

表 3–7 は、AND 演算子で 2 つの式を結合した結果を示しています。

表 3–7　AND の真理値表

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR 演算子

表 3–8 は、OR 演算子で 2 つの式を結合した結果を示しています。

表 3–8　OR の真理値表

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

集合演算子

集合演算子は 2 つのコンポーネントの問合せ結果を 1 つの結果にまとめます。集合演算子を含む問合せを複合問合せと呼びます。表 3–9 は、SQL の集合演算子を示しています。

表 3–9　集合演算子

演算子	戻る結果
UNION	各問合せによって戻るすべての行（重複行は含まない）
UNION ALL	各問合せによって戻るすべての行（重複行も含む）
INTERSECT	両方の問合せによって戻るすべての行（重複行は含まない）
MINUS	最初の問合せによって戻る行で、2 番目の問合せでは戻されない行（重複行は含まない）

集合演算子の優先順位はすべて同じです。SQL 文に複数の集合演算子がある場合、カッコによって明示的に別の順序が指定されていない限り、Oracle は左から右の順に評価します。今後の SQL の規格に準拠するために、Oracle の将来のバージョンでは、INTERSECT 演算子には他の集合演算子よりも高い優先順位を与える予定です。そのため、INTERSECT 演算子を他の集合演算子とともに使用する問合せでは、カッコを使用して評価の順序を明示的に指定することをお薦めします。

複合問合せを構成する各問合せと、それに対応する選択リスト内の各式は、数値とデータ型が一致している必要があります。集合演算子によって結合された 2 つの問合せが文字データを選択する場合、戻される値のデータ型は次のようにして決定されます。

- 両方の問合せが CHAR データ型の値を選択する場合、戻される値のデータ型は CHAR となる。
- 問合せのどちらか一方または両方が、VARCHAR2 データ型の値を選択する場合、戻される値のデータ型は VARCHAR2 となる。

例　次の 2 つの問合せとその結果を想定します。

```
SELECT part
    FROM orders_list1;

PART
----------
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE
```

```
SELECT part
    FROM orders_list2;

PART
----------
CRANKSHAFT
TAILPIPE
TAILPIPE
```

次に、各集合演算子でこの 2 つの問合せの結果を結合する例を示します。

UNION の例 次の文は、UNION 演算子によって 2 つの結果を結合しています。結果に重複行は含まれません。次の文は、他方の表に存在していない列がある場合に、データ型を一致させる方法を示しています。

```
SELECT part, partnum, to_date(null) date_in
    FROM orders_list1
UNION
SELECT part, to_null(null), date_in
    FROM orders_list2;

PART       PARTNUM  DATE_IN
------     -------  --------
SPARKPLUG  3323165
SPARKPLUG           10/24/98
FUEL PUMP  3323162
FUEL PUMP           12/24/99
TAILPIPE   1332999
TAILPIPE            01/01/01
CRANKSHAFT 9394991
CRANKSHAFT          09/12/02

SELECT part
    FROM orders_list1
UNION
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
TAILPIPE
CRANKSHAFT
```

UNION ALL の例 次の文は、UNION ALL 演算子を使用して 2 つの結果を結合しています。結果には重複行が含まれることもあります。

```
SELECT part
    FROM orders_list1
UNION ALL
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE
CRANKSHAFT
TAILPIPE
TAILPIPE
```

UNION ALL 演算子がすべての行を戻すのに対して、UNION 演算子は重複しない行だけを戻すことに注目してください。問合せで複数回戻される PART 値（「FUEL PUMP」など）は、UNION 演算子では一度だけ戻されますが、UNION ALL 演算子では複数回戻されています。

INTERSECT の例 次の文は、INTERSECT 演算子によって 2 つの結果を結合しています。この場合、両方の問合せによって共通に戻される行だけが戻されます。

```
SELECT part
    FROM orders_list1
INTERSECT
SELECT part
    FROM orders_list2;

PART
----------
TAILPIPE
```

MINUS の例 次の文は、MINUS 演算子を使用して 2 つの結果を結合します。この場合、最初の問合せでは戻されるが 2 番目の問合せでは戻されない行だけが戻されます。

```
SELECT part
    FROM orders_list1
MINUS
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
```

その他の演算子

表 3–10 に、その他の SQL 演算子を示します。

表 3–10　その他の SQL 演算子

演算子	用途	例

(+)	(+) の直前の列が結合で外部結合列であることを示す。「外部結合」（4-504 ページ）を参照。	`SELECT ename, dname`
		`  FROM emp, dept`
		`  WHERE dept.deptno =`
		`  emp.deptno(+);`

PRIOR	階層型（ツリー構造）の問合せで、PRIOR の次の式を現行の行の親行に対して評価する。このような問合せでは、行の親子関連を定義するために CONNECT BY 句でこの演算子を使用しなければならない。また、階層問合せを実行する SELECT 文の他の部分でこの演算子を使用できる。PRIOR 演算子は単項演算子であり、単項算術演算子の + や - と同じ優先順位を持っている。「階層問合せ」（4-492 ページ）を参照。	`SELECT empno, ename, mgr`
		`  FROM emp`
		`  CONNECT BY`
		`  PRIOR empno = mgr;`

SQL 関数

SQL 関数は、データ項目を操作し結果を戻すという点で演算子と似ています。SQL 関数と演算子は引数を指定する書式が異なります。次の書式によって、関数ではゼロ以上の引数を操作できます。

```
function(argument, argument, ...)
```

SQL 関数が必要とするデータ型以外のデータ型の引数で SQL 関数を呼び出すと、Oracle は SQL 関数を実行する前に、その引数を必要なデータ型に暗黙的に変換します。「データ変換」（2-26 ページ）を参照してください。

NULL を引数として SQL 関数を呼び出すと、SQL 関数により自動的に NULL が戻されます。この規則に従わない SQL 関数は、CONCAT および DECODE、DUMP、NVL、REPLACE の 5 つだけです。

SQL 関数と、PL/SQL で書かれたユーザー関数を混同しないでください。ユーザー関数については、「ユーザー関数」（3-57 ページ）を参照してください。

SQL 関数の構文図では、このマニュアルの「はじめに」にある「構文図と表記法」で説明した規則に従って、データ型とともに引数が示されています。

一般に、SQL 関数には次のタイプがあります。

- 単一行（またはスカラー）関数
- グループ（または集合体）関数

これら 2 つの SQL 関数は、操作の対象とする行数が異なります。単一行関数は、問合せ対象の表やビューの各行に対して 1 つの結果行を戻します。グループ関数は、問合せ対象の行グループに対して 1 つの結果行を戻します。

単一行関数は、(GROUP BY 句を含まない SELECT 文の) 選択リスト、WHERE 句、START WITH 句、CONNECT BY 句に指定できます。

グループ関数は、選択リストと HAVING 句に指定できます。SELECT 文で GROUPBY 句が使用されている場合、Oracle は問合せ対象の表やビューの行をグループに分けます。GROUP BY 句を含む問合せでは、選択リストのすべての要素は、GROUP BY 句からの式またはグループ関数を含む式、定数のいずれかでなければなりません。Oracle は、選択リスト内のグループ関数を行の各グループに適用し、各グループに単一の結果行を戻します。

GROUP BY 句を指定しないと、Oracle は選択リスト内のグループ関数を問合せ対象の表またはビューのすべての行に適用します。HAVING 句でグループ関数を指定して、グループを出力しないこともできます。このとき、出力は、問合せ対象の表またはビューの各行の値ではなく、グループ関数の結果に基づきます。GROUP BY 句と HAVING 句の詳細は、「GROUP BY 句」（4-496 ページ）および「HAVING 句」（4-497 ページ）を参照してください。

以降の項では、関数は引数のデータ型と戻り値によってグループ化されています。

数値関数

数値関数は入力として数値を受け取り、結果として数値を戻します。この項では、SQL の数値関数を説明します。これらの関数の大部分は 38 桁 (10 進) の値を戻します。超越関数の COS、COSH、EXP、LN、LOG、SIN、SINH、SQRT、TAN、TANH は 36 桁 (10 進) の値を戻します。超越関数の ACOS、ASIN、ATAN、ATAN2 は 30 桁 (10 進) の値を戻します。

ABS

構文　`ABS(n)`

用途　*n* の絶対値を戻します。

例

```
SELECT ABS(-15) "Absolute" FROM DUAL;

  Absolute
----------
        15
```

ACOS

構文

```
ACOS(n)
```

用途

n の逆コサインを戻します。入力は -1 ～ 1 の範囲で、出力は 0 ～ p(ラジアン) の範囲です。

例

```
SELECT ACOS(.3)"Arc_Cosine" FROM DUAL;

Arc_Cosine
----------
1.26610367
```

ASIN

構文

```
ASIN(n)
```

用途

n の逆サインを戻します。入力は -1 ～ 1 の範囲で、出力は -p/2 ～ p/2(ラジアン) の範囲です。

例

```
SELECT ASIN(.3) "Arc_Sine" FROM DUAL;

Arc_Sine
----------
.304692654
```

ATAN

構文

```
ATAN(n)
```

用途

n の逆タンジェントを戻します。入力の範囲に制限はなく、出力は -p/2 ～ p/2(ラジアン) の範囲です。

例

```
SELECT ATAN(.3) "Arc_Tangent" FROM DUAL;

Arc_Tangent
 ----------
.291456794
```

ATAN2

構文

```
ATAN2(n, m)
```

用途

n と m の逆タンジェントを戻します。入力の範囲に制限はなく、出力は n と m の符号により -p ～ p(ラジアン) の範囲です。ATAN2(n,m) は ATAN2(n/m) と同じです。

例

```
SELECT ATAN2(.3, .2) "Arc_Tangent2" FROM DUAL;

Arc_Tangent2
------------
.982793723
```

CEIL

構文

```
CEIL(n)
```

用途

n 以上の最も小さい整数を戻します。

例

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

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

COS

構文

```
COS(n)
```

用途

n(ラジアンで表された角度) のコサインを戻します。

例

```
SELECT COS(180 * 3.14159265359/180)
"Cosine of 180 degrees" FROM DUAL;

Cosine of 180 degrees
---------------------
                   -1
```

COSH

構文 COSH(n)

用途 *n* の双曲線コサインを戻します。

例

```
SELECT COSH(0) "Hyperbolic cosine of 0" FROM DUAL;

Hyperbolic cosine of 0
----------------------
                     1
```

EXP

構文 EXP(n)

用途 e を *n* 乗した値を戻します (e = 2.71828183 ...)。

例

```
SELECT EXP(4) "e to the 4th power" FROM DUAL;

e to the 4th power
------------------
          54.59815
```

FLOOR

構文 FLOOR(n)

用途 *n* 以下の最も大きい整数を戻します。

例

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

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

LN

構文 LN(n)

用途 *n* の自然対数を戻します (*n* は正の数)。

例

```
SELECT LN(95) "Natural log of 95" FROM DUAL;

Natural log of 95
-----------------
       4.55387689
```

LOG

構文

```
LOG(m,n)
```

用途

m を底とする *n* の対数を戻します。底 *m* は 0 および 1 以外の任意の正の数、*n* は任意の正の数です。

例

```
SELECT LOG(10,100) "Log base 10 of 100" FROM DUAL;

Log base 10 of 100
------------------
                 2
```

MOD

構文

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

用途

m を *n* で除した余りを戻します。*n* が 0 の場合は、*m* を戻します。

例

```
SELECT MOD(11,4) "Modulus" FROM DUAL;

   Modulus
----------
         3
```

この関数は、*m* が負の場合には古典数学のモジュール関数とは異なる動作をします。古典数学のモジュール関数は、次の公式を用いた MOD 関数で表すことができます。

```
m - n * FLOOR(m/n)
```

次の文は、MOD 関数と古典数学のモジュール関数の相違を示します。

```
SELECT m, n, MOD(m, n),
m - n * FLOOR(m/n) "Classical Modulus"
  FROM test_mod_table;

          M          N  MOD (M,N) Classical Modulus
---------- ---------- ---------- -----------------
        11          4          3                 3
        11         -4          3                -1
       -11          4         -3                 1
       -11         -4         -3                -3
```

POWER

構文

```
POWER(m, n)
```

用途

m を *n* 乗した値を戻します。底 *m* と指数 *n* は任意の数です。ただし、*m* が負の場合、*n* は整数でなければなりません。

例

```
SELECT POWER(3,2) "Raised" FROM DUAL;

    Raised
----------
         9
```

ROUND

構文

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

用途

n を小数点以下 *m* 桁に丸めた値を戻します。*m* が省略されると小数点以下が丸められます。*m* が負の場合は小数点の左 *m* 桁が丸められます。*m* は整数でなければなりません。

例 1

```
SELECT ROUND(15.193,1) "Round" FROM DUAL;

     Round
----------
      15.2
```

例 2

```
SELECT ROUND(15.193,-1) "Round" FROM DUAL;

     Round
----------
        20
```

SIGN

構文

```
SIGN(n)
```

用途

n が 0 より小さい場合は -1、*n* が 0 の場合は 0、*n* が 0 より大きい場合は 1 を戻します。

例

```
SELECT SIGN(-15) "Sign" FROM DUAL;

      Sign
----------
        -1
```

SIN

構文 SIN(n)

用途 *n*(ラジアンで表された角度) のサインを戻します。

例

```
SELECT SIN(30 * 3.14159265359/180)
"Sine of 30 degrees" FROM DUAL;

Sine of 30 degrees
------------------
                 .5
```

SINH

構文 SINH(n)

用途 *n* の双曲線サインを戻します。

例

```
SELECT SINH(1) "Hyperbolic sine of 1" FROM DUAL;

Hyperbolic sine of 1
--------------------
          1.17520119
```

SQRT

構文 SQRT(n)

用途 *n* の平方根を戻します。*n* には負の値は指定できません。SQRT は " 実数 " を戻します。

例

```
 SELECT SQRT(26) "Square root" FROM DUAL;

Square root
-----------
 5.09901951
```

TAN

構文 TAN

用途 *n*(ラジアンで表された角度) のタンジェントを戻します。

例

```
SELECT TAN(135 * 3.14159265359/180)
 "Tangent of 135 degrees"  FROM DUAL;

 Tangent of 135 degrees
 ----------------------
                    - 1
```

TANH

構文 TANH(n)

用途 *n* の双曲線タンジェントを戻します。

例

```
 SELECT TANH(.5) "Hyperbolic tangent of .5"
FROM DUAL;

Hyperbolic tangent of .5
------------------------
               .462117157
```

TRUNC

構文 TRUNC(n[,m])

用途 *n* を小数点以下 *m* 桁に切り捨てた値を戻します。*m* が省略されたときは、小数点以下が切り捨てられます。*m* が負の場合は、小数点の左 *m* 桁がゼロになります。

例

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

  Truncate
----------
      15.7

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

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

文字関数

単一行文字関数は、入力として文字を受け取り、文字または数値のどちらかを戻します。

文字値を戻す文字関数

この項では文字値を戻す文字関数を説明します。特に断りのない限り、これらの関数はすべて VARCHAR2 データ型を持つ値を戻し、長さは 4000 バイトに制限されます。データ型が CHAR の値を戻す関数は、長さは 2000 バイトに制限されます。戻り値の長さが制限を超えた場合、Oracle は戻り値から制限を超えた部分を切り捨てて、エラー・メッセージを出力せずにその結果を戻します。

CHR

構文

```
CHR(n [USING NCHAR_CS])
```

用途

データベース・キャラクタ・セットまたは各国キャラクタ・セットの中の *n* に等しい 2 進数を持つ文字を戻します。

USING NCHAR_CS 句が指定されていない場合は、この関数はデータベース・キャラクタ・セットの中の *n* に等しい 2 進数を持つ文字を VARCHAR2 値として戻します。

USING NCHAR_CS 句が指定されている場合は、この関数は各国キャラクタ・セットの中の *n* に等しい 2 進数を持つ文字を NVARCHAR2 値として戻します。

例 1

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog"
  FROM DUAL;
Dog
---
CAT
```

例 2

```
SELECT CHR(16705 USING NCHAR_CS) FROM DUAL;

C
-
A
```

CONCAT

構文

```
CONCAT(char1, char2)
```

用途

char1 を *char2* と連結して戻します。この関数は連結演算子 (||) と等価です。この演算子の詳細は、「連結演算子」（3-3 ページ）を参照してください。

例

次の例は、ネストを使用して 3 つの文字列を連結しています。

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

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

INITCAP

構文

```
INITCAP(char)
```

用途

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* を戻します。単語は空白または英数字以外の文字で区切ります。

例

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

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

LOWER

構文

```
LOWER(char)
```

用途

すべての文字を小文字にして *char* を戻します。戻り値は引数 *char* と同じデータ型 (CHAR または VARCHAR2) になります。

例

```
SELECT LOWER('MR. SCOTT MCMILLAN') "Lowercase"
FROM DUAL;

Lowercase
--------------------
mr. scott mcmillan
```

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* 桁にして戻します。*char2* のデフォルト値は単一の空白です。*char1* が *n* 文字より長い場合、この関数は *n* に収まる *char1* の一部を戻します。

引数 *n* は端末画面に表示される場合の戻り値の全体の長さです。多くのキャラクタ・セットでは、これは戻り値の文字数でもあります。ただし、マルチバイトのキャラクタ・セットでは、文字列の表示長が文字列の文字数と異なる場合もあります。

例

```
SELECT LPAD('Page 1',15,'*.') "LPAD example"
FROM DUAL;

LPAD example
---------------
*.*.*.*.*Page 1
```

LTRIM

構文

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

用途

char の左側にあって *set* に指定された文字を削除します。*set* のデフォルト値は単一の空白です。Oracle は *char* の先頭文字から走査し始め、set に指定された文字をすべて削除します。*set* に指定された以外の文字が見つかった時点で結果を戻します。

例

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

 LTRIM exampl
 ------------
 XxyLAST WORD
```

NLS_INITCAP

構文

```
NLS_INITCAP(char [, 'nlsparams'] )
```

用途

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* を戻します。単語は空白または英数字以外の文字で区切ります。'*nlsparams*' の値は次の書式で指定します。

```
'NLS_SORT = sort'
```

sort は言語ソート順序または BINARY のどちらかです。言語のソート順序は、大文字と小文字の変換のために特別な言語要件を処理します。なお、これらの要件によって、*char* と異なる長さの値が戻される可能性があります。'*nlsparams*' を指定しないと、セッションのデフォルト・ソート順序が使用されます。ソート順序の説明は、『Oracle8 Server リファレンス』を参照してください。

例

```
SELECT NLS_INITCAP
   ('ijsland', 'NLS_SORT = XDutch') "Capitalized"
   FROM DUAL;

Capital
-------
IJsland
```

NLS_LOWER

構文

```
NLS_LOWER(char [, 'nlsparams'] )
```

用途

すべての文字を小文字にして *char* を戻します。'*nlsparams*' の書式と用途は NLS_INITCAP 関数と同じです。

例

```
SELECT NLS_LOWER
    ('CITTA''', 'NLS_SORT = XGerman') "Lowercase"
    FROM DUAL;

Lower
-----
citta
```

NLS_UPPER

構文

```
NLS_UPPER(char [, 'nlsparams'] )
```

用途

すべての文字を大文字にして *char* を戻します。'*nlsparams*' の書式と用途は NLS_INITCAP 関数と同じです。

例

```
SELECT NLS_UPPER
    ('groβe', 'NLS_SORT = XGerman') "Uppercase"
      FROM DUAL;

Upper
-----
GROSS
```

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* を戻します。*replacement_string* を指定しない場合または NULL の場合、*search_string* がすべて削除されます。*search_string* が NULL の場合、*char* が戻されます。この関数は TRANSLATE 関数を拡張したものです。TRANSLATE 関数は、単一の文字を 1 対 1 で置き換えます。REPLACE 関数では、文字列の置換または削除を実行できます。

例

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

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

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* 桁にして戻します。*char2* のデフォルト値は単一の空白です。*char1* が *n* 文字より長い場合、この関数は n に収まる *char1* の一部を戻します。

引数 *n* は端末画面に表示される場合の戻り値の全体の長さです。多くのキャラクタ・セットでは、これは戻り値の文字数でもあります。ただし、マルチバイトのキャラクタ・セットでは、文字列の表示長が文字列の文字数と異なる場合もあります。

例

```
SELECT RPAD('MORRISON',12,'ab') "RPAD example"
     FROM DUAL;

RPAD example
-----------------
MORRISONabab
```

RTRIM

構文

```
RTRIM(char [,set]
```

用途

char の右側にあって set に指定されたすべての文字を削除し、*char* を戻します。*set* のデフォルト値は単一の空白です。RTRIM は LTRIM と似た働きをします。

例

```
SELECT RTRIM('BROWNINGyxXxy','xy') "RTRIM e.g."
     FROM DUAL;

RTRIM e.g
-------------
BROWNINGyxX
```

SOUNDEX

構文

```
SOUNDEX(char)
```

用途

char と同じ発音を持つ文字列を戻します。この関数によって、綴りは異なるが発音の似た単語（英語）を比較できます。

音声表現については、『The Art of Computer Programming, Volume3: Sorting and Searching(Donald E.Knuth 著)』で規定されています。

- 文字列の最初の文字を残し、以降 a、e、h、i、o、u、w、y が出てきた場合にはすべて削除します。
- 残った文字の 2 文字目以降に対し、次のように数字を割り当てます。

```
b, f, p, v = 1
c, g, j, k, q, s, x, z = 2
d, t = 3
l = 4
m, n = 5
r = 6
```

- 割り当てられた数字が同じ文字が 2 つ以上並んでいる場合は、最初の文字だけを残して削除します。
- 残りを 0 で埋めて、最初の 4 バイトだけを戻します。

例

```
SELECT ename
     FROM emp
     WHERE SOUNDEX(ename)
         = SOUNDEX('SMYTHE');

ENAME
----------
SMITH
```

SUBSTR

構文

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

用途

char の *m* 番目の文字から *n* 文字分の文字列を抜き出して戻します。*m* が 0 である場合、1 として扱われます。*m* が正の数である場合、Oracle は *char* の始めから数えて最初の文字を見つけます。*m* が負の数である場合、Oracle は *char* の終わりから数えます。*n* を指定しないと、Oracle は *char* の終わりまでのすべての文字を戻します。*n* が 1 より小さい場合、NULL を戻します。

引数として *substr* に渡された浮動小数点数は、自動的に整数に変換されます。

例 1

```
SELECT SUBSTR('ABCDEFG',3.1,4) "Subs"
     FROM DUAL;

Subs
----
CDEF
```

例 2

```
SELECT SUBSTR('ABCDEFG',-5,4) "Subs"
     FROM DUAL;

Subs
----
CDEF
```

SUBSTRB

構文

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

用途

引数の *m* と *n* が文字ではなくバイトで表される以外、SUBSTR と同じです。データベースのキャラクタ・セットがシングルバイトの場合、SUBSTRB は SUBSTR と等価です。

引数として substrb に渡された浮動小数点数は、自動的に整数に変換されます。

例

データベースのキャラクタ・セットがダブルバイトの場合を想定します。

```
SELECT SUBSTRB('ABCDEFG',5,4.2)
      "Substring with bytes"
      FROM DUAL;

Substring with bytes
--------------------
CD
```

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* を戻します。*from* 内に存在しない *char* 内の文字は置換されません。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* 内に対応する文字を持ちません。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取り除かれます。戻り値から *from* 内の文字をすべて削除するために *to* に空の文字列は使用できません。Oracle は空の文字列を NULL と解釈し、NULL の引数がある場合、この関数は NULL を戻します。

例 1

次の文は、ライセンス番号を置換します。文字 ’ABC...Z’ はすべて ’X’ に置換され、数 ’012...9’ はすべて ’9’ に置換されます。

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

License
--------
9XXX999
```

例 2

次の文は、文字を取り除いて数値だけになったライセンス番号を戻します。

```
SELECT TRANSLATE('2KRW229',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
"Translate example"
     FROM DUAL;

Translate example
-----------------
2229
```

UPPER

構文

```
UPPER(char)
```

用途

すべての文字を大文字にして *char* を戻します。戻り値は引数 *char* と同じデータ型を持ちます。

例

```
SELECT UPPER('Large') "Uppercase"
     FROM DUAL;

Upper
-----
LARGE
```

数値を戻す文字関数

この項では数値を戻す文字関数を説明します。

ASCII

構文

```
ASCII(char)
```

用途

char の最初の文字の、データベースのキャラクタ・セットでの 10 進表現を戻します。データベースのキャラクタ・セットが 7 ビットの ASCII である場合には ASCII 値を戻します。データベースのキャラクタ・セットが EBCDIC コード・ページ 500 である場合には EBCDIC 値を戻します。なお、この関数と同じような EBCDIC 文字関数は存在しません。

例

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

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

INSTR

構文

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

用途

char1 の *n* 番目の文字から *char2* の検索を開始し、*char2* が *m* 番目に出現した位置を戻します (*char2* が文字列であればその最初の文字の位置を戻します)。*n* が負である場合、Oracle は *char1* の終わりから逆方向に検索します。*m* の値は正でなければなりません。*n* と *m* のデフォルト値は 1 です。この場合、Oracle は *char1* の最初の文字から検索を開始します。検索対象は *char2* の最初の出現です。戻り値は *n* の値にかかわらず、*char1* の先頭から数えた文字位置です。検索が失敗した (*char1* の *n* 番目の文字の後に *char2* が *m* 回出現しない) 場合、戻り値は 0 となります。

例 1

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

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

例 2

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

Reversed Instring
-----------------
                2
```

INSTRB

構文

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

用途

n と戻り値が文字ではなくバイトで表されること以外は INSTR と同じです。データベースのキャラクタ・セットがシングルバイトの場合、INSTRB は INSTR と等価です。

例

この例では、データベースのキャラクタ・セットがダブルバイトの場合を想定しています。

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

Instring in bytes
-----------------
               27
```

LENGTH

構文

```
LENGTH(char)
```

用途

char の長さを文字単位で戻します。*char* のデータ型が CHAR の場合には、その長さには後続する空白がすべて含まれます。*char* が NULL である場合には、NULL が戻されます。

例

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

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

LENGTHB

構文

```
LENGTHB(char)
```

用途

char の長さをバイト単位で戻します。*char* が NULL である場合には、NULL が戻されます。データベースのキャラクタ・セットがシングルバイトの場合、LENGTHB は LENGTH と等価です。

例

この例では、データベースのキャラクタ・セットがダブルバイトの場合を想定しています。

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

Length in bytes
---------------
             14
```

NLSSORT

構文

```
NLSSORT(char [, 'nlsparams'])
```

用途

char をソートするために使用される文字列のバイトを戻します。'*nlsparams*' の値は次の書式で指定します。

```
'NLS_SORT = sort'
```

sort は言語ソート順序または BINARY のどちらかです。'*nlsparams*' を指定しないと、セッションのデフォルト・ソート順序が使用されます。BINARY を指定すると、この関数は *char* を戻します。ソート順序については、『Oracle8 Server リファレンス』の「各国語サポート」の章を参照してください。

例

この関数を使って、文字列の 2 進値を基にした比較ではなく、言語ソート順序を基にした比較を指定できます。

```
SELECT ename FROM emp
   WHERE NLSSORT (ename, 'NLS_SORT = German')
   > NLSSORT ('S', 'NLS_SORT = German') ORDER BY ename;

ENAME
----------
SCOTT
SMITH
TURNER
WARD
```

日付関数

日付関数は DATE データ型の値を操作します。数を戻す MONTHS_BETWEEN 関数を除いた日付関数はすべて DATE データ型の値を戻します。

ADD_MONTHS

構文

```
ADD_MONTHS(d,n)
```

用途

日付 *d* に *n* か月を加えて戻します。引数 *n* には整数を指定します。*d* が月の最終日の場合、または結果の月の最終日が *d* の日付コンポーネントよりも小さい場合、戻される値は結果の月の最終日となります。それ以外の場合、結果は *d* と同じ日付コンポーネントを持ちます。

例

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

Next Month
-----------
17-JAN-1981
```

LAST_DAY

構文

```
LAST_DAY(d)
```

用途

d を含む月の最後の日付を戻します。その月の残りの日数を求めるためにこの関数を使用できます。

例 1

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

SYSDATE   Last       Days Left
--------- --------- ----------
23-OCT-97 31-OCT-97          8
```

例 2

```
SELECT TO_CHAR(
 ADD_MONTHS(
   LAST_DAY(hiredate),5),
    'DD-MON-YYYY') "Five months"
 FROM emp
 WHERE ename = 'MARTIN';

Five months
-----------
28-FEB-1982
```

MONTHS_BETWEEN

構文

```
MONTHS_BETWEEN(d1, d2)
```

用途

日付 *d1* と *d2* の間の月数を戻します。*d1* が *d2* 以降の日付であれば結果は正の値になります。*d1* が *d2* 以前の日付であれば結果は負の値になります。*d1* と *d2* が月の同じ日または月の最終日の場合、結果は整数になります。それ以外の場合、Oracle は結果の小数部を 1 か月、31 日として計算します。また、*d1* と *d2* の時間要素の相違も考慮されます。

例

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

    Months
----------
1.03225806
```

NEW_TIME

構文

```
NEW_TIME(d, z1, z2)
```

用途

時間帯 *z1* の日時が *d* である時点の時間帯 *z2* の日時を戻します。引数 *z1* と *z2* には次のテキスト文字列の 1 つを指定します。

AST	大西洋地区の標準時間と夏時間
ADT	
BST	ベーリング地区の標準時間と夏時間
BDT	
CST	中央地区の標準時間と夏時間
CDT	
EST	東地区の標準時間と夏時間
EDT	
GMT	グリニッジ標準時間
HST	アラスカ、ハワイ地区の標準時間と夏時間
HDT	
MST	山岳地区の標準時間と夏時間
MDT	
NST	ニューファンドランド地区の標準時間

PST	太平洋地区の標準時間と夏時間
PDT	
YST	ユーコン地区の標準時間と夏時間
YDT	

NEXT_DAY

構文

```
NEXT_DAY(d, char)
```

用途

char で指定した曜日で、日付 *d* 以降の最初の日付を戻します。引数 *char* は、セッションで使用している言語での曜日でなければなりません（フルネームでも省略形でも構いません）。必要最小限の文字数は、省略形の文字数です。有効な省略形の後に続けて文字が入力されていても、それらの文字は無視されます。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

次の例では、1992 年 3 月 15 日以降の最初の火曜日の日付を戻します。

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

NEXT DAY
---------
17-MAR-92
```

ROUND

構文

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

用途

d を書式モデル *fmt* で指定した単位まで近似した結果を戻します。*fmt* を指定しないと、*d* は最も近い日が戻ります。*fmt* で使用できる書式モデルについては、「ROUND と TRUNC」（3-38 ページ）を参照してください。

例

```
SELECT ROUND (TO_DATE ('27-OCT-92'),'YEAR')
   "New Year" FROM DUAL;

New Year
---------
01-JAN-93
```

SYSDATE

構文

```
SYSDATE
```

用途

現在の日時を戻します。引数は必要としません。分散 SQL 文では、ローカル・データベース上の日時が戻ります。CHECK 制約の条件でこの関数は使用できません。

例

```
SELECT TO_CHAR
    (SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"NOW"
    FROM DUAL;
NOW
--------------------
10-29-1993 20:27:11
```

TRUNC

構文

```
1TRUNC(d,[fmt])
```

用途

時刻部分を書式モデル *fmt* で指定された単位まで近似した *d* を戻します。*fmt* を指定しないと、*d* は最も近い日が戻ります。*fmt* で使用できる書式モデルについては、「ROUND と TRUNC」（3-38 ページ）を参照してください。

例

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

New Year
---------
01-JAN-92
```

ROUND と TRUNC

表 3–11 に、日付関数 ROUND および TRUNC で使用できる書式モデル、および日付の丸めと切捨ての単位を示します。デフォルト・モデル 'DD' では、午前 0 時（真夜中）を基準に近似を行い、日付を戻します。

表 3–11　日付関数 ROUND と TRUNC の日付書式モデル

書式モデル	丸めと切捨ての単位
CC SCC	4 桁の年の上 2 桁より 1 大きい数
SYYYY YYYY YEAR SYEAR YYY YY Y	年 (7 月 1 日に切り上げ)
IYYY IY IY I	ISO 年
Q	四半期（その四半期の 2 番目の月の 16 日に切上げ）

表 3–11　日付関数 ROUND と TRUNC の日付書式モデル

書式モデル	丸めと切捨ての単位
MONTH MON MM RM	月 (16 日に切上げ)
WW	年の最初の日と同じ曜日
IW	ISO 年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	週の開始日
HH HH12 HH24	時
MI	分

書式モデル DAY および DY、D によって使用される週の開始日は、初期化パラメータ NLS_TERRITORY によって暗黙的に指定されています。このパラメータについては、『Oracle8 Server リファレンス』を参照してください。

変換関数

変換関数は、あるデータ型から他のデータ型に値を変換します。一般的に関数名は*データ型* TO *データ型*の形で与えられます。最初 (TO の前) のデータ型が入力データ型、後のデータ型が出力データ型です。この項では、SQL の変換関数を説明します。

CHARTOROWID

構文　`CHARTOROWID(char)`

用途　CHAR データ型または VARCHAR2 データ型の値を ROWID データ型の値に変換します。

例

```
SELECT ename FROM emp
   WHERE ROWID = CHARTOROWID('AAAAfZAABAAACp8AAO');

ENAME
----------
LEWIS
```

CONVERT

構文

```
CONVERT(char, dest_char_set [,source_char_set] )
```

用途

あるキャラクタ・セットの文字列を別のキャラクタ・セットの文字列に変換します。

引数 char は変換する値です。

引数 *dest_char_set* は char が変換されるキャラクタ・セットの名前です。

引数 *source_char_set* は char をデータベースに格納しているキャラクタ・セットの名前です。デフォルト値はデータベースのキャラクタ・セットです。

変換先キャラクタ・セットと変換元キャラクタ・セットの引数として、リテラルまたはキャラクタ・セットの名前を含んでいる列を指定できます。

完全に文字を変換するには、変換先キャラクタ・セットが変換元キャラクタ・セットで定義されているすべての文字を表現できなければなりません。文字が変換先キャラクタ・セットに存在しないと置換文字が使用されます。置換文字はキャラクタ・セット定義の一部として定義できます。

例

```
SELECT CONVERT('Gro&#223;', 'US7ASCII', 'WE8HP')
"Conversion"
  FROM DUAL;

Conversion
----------
Gross
```

一般的なキャラクタ・セットを次に示します。

US7ASCII	US7 ビット ASCII キャラクタ・セット
WE8DEC	DEC 西ヨーロッパ 8 ビット・キャラクタ・セット
WE8HP	HP 西ヨーロッパ Laserjet 8 ビット・キャラクタ・セット
F7DEC	DEC フランス 7 ビット・キャラクタ・セット
WE8EBCDIC500	IBM 西ヨーロッパ EBCDIC コード・ページ 500
WE8PC850	IBM PC コード・ページ 850
WE8ISO8859P1	ISO 8859-1 西ヨーロッパ 8 ビット・キャラクタ・セット

HEXTORAW

構文

```
HEXTORAW(char)
```

用途

16 進数を含む *char* を RAW 値に変換します。

例

```
INSERT INTO graphics (raw_column)
  SELECT HEXTORAW('7D') FROM DUAL;
```

RAWTOHEX

構文

```
RAWTOHEX(raw)
```

用途

raw を 16 進で表した文字の値に変換します。

例

```
SELECT RAWTOHEX(raw_column) "Graphics"
   FROM graphics;

Graphics
--------
7D
```

ROWIDTOCHAR

構文

```
ROWIDTOCHAR(rowid)
```

用途

ROWID 値を VARCHAR2 データ型に変換します。この変換の結果は常に 18 文字です。

例

```
SELECT ROWID
     FROM offices
     WHERE
     ROWIDTOCHAR(ROWID) LIKE '%Br1AAB%';

ROWID
------------------
AAAAZ6AABAAABr1AAB
```

TO_CHAR(日付変換)

構文

```
TO_CHAR(d [, fmt [, 'nlsparams'] ])
```

用途

DATE データ型の値 *d* を日付書式 *fmt* で指定した書式の VARCHAR2 データ型の値に変換します。*fmt* を指定しないと、*d* がデフォルトの日付書式の VARCHAR2 値に変換されます。日付書式については、「書式モデル」（3-60 ページ）を参照してください。

'nlsparams' には、月と日の名前や略称が戻される言語を指定します。この引数は次の書式で指定します。

```
'NLS_DATE_LANGUAGE = language'
```

nlsparams を指定しないと、この関数はセッションのデフォルト日付言語を使用します。

例

```
SELECT TO_CHAR(HIREDATE, 'Month DD, YYYY')
    "New date format" FROM emp
    WHERE ename = 'BLAKE';

New date format
------------------
May       01, 1981
```

TO_CHAR(数値変換)

構文

```
TO_CHAR(n [, fmt [, 'nlsparams'] ])
```

用途

オプションの数値書式 fmt を使用して、NUMBER データ型の値 *n* を VARCHAR2 データ型の値に変換します。*fmt* を指定しないと、*n* の有効桁数を保持するために十分な長さの VARCHAR2 値に変換されます。数値書式については、「書式モデル」（3-60 ページ）を参照してください。

'*nlsparams*' は数の書式要素によって戻される次の文字を指定します。

- 小数点文字
- グループ・セパレータ
- 各国通貨記号
- 国際通貨記号

この引数は次の書式で指定します。

```
'NLS_NUMERIC_CHARACTERS = "dg"
NLS_CURRENCY = "text"
NLS_ISO_CURRENCY = territory '
```

文字 *d* と *g* はそれぞれ小数点文字とグループ・セパレータを表します。それらは異なるシングルバイト文字でなければなりません。引用符で囲んだ文字列の中では、パラメータ値を囲む引用符のために単一引用符を 2 つ使用しなければならないことに注意してください。通貨記号には 10 文字使用できます。

'*nlsparams*' やパラメータのどれか 1 つを指定しないと、この関数はセッションのデフォルト・パラメータ値を使用します。

例 1

この例では、出力で通貨記号の左に空白埋めが行われます。

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount"
     FROM DUAL;

Amount
--------------
   $10,000.00-
```

例 2

```
SELECT TO_CHAR(-10000,'L99G999D99MI',
'NLS_NUMERIC_CHARACTERS = ",."
NLS_CURRENCY = ''AusDollars'' ') "Amount"
     FROM DUAL;

Amount
--------------------
AusDollars10.000,00-
```

注意:

- オプションの数値書式 fmt では、L は各国通貨記号を、MI は後に付くマイナス記号 (-) を表します。数値書式要素の全リストは、表 3–13（3-63 ページ）を参照してください。
- Oracle で数値を文字列に変換していて、Oracle の NUMBER データ型の範囲を超過または不足するような丸め処理が発生した場合、無限大を表す "~" またはマイナス無限大を表す "-~" が戻されることがあります。通常、このイベントは、TO_CHAR() 関数を制限的な数値書式文字列で使用して、丸め処理が行われた場合に発生します。

TO_DATE

構文

```
TO_DATE(char [, fmt [, 'nlsparams'] ])
```

用途

CHAR データ型または VARCHAR2 データ型の値 *char* を DATE データ型の値に変換します。*fmt* は *char* の書式を指定する日付書式です。*fmt* を指定しない場合、char はデフォルト日付書式でなければなりません。*fmt* が 'J'(ユリウス暦) である場合、*char* は整数でなければなりません。日付書式については、「書式モデル」（3-60 ページ）を参照してください。

'*nlsparams*' は日付変換の TO_CHAR 関数の場合と同じ用途で使用されます。

引数 *char* として DATE 値を持つ TO_DATE 関数を使用してはなりません。戻される DATE 値は、*fmt* またはデフォルト値に依存して、元の *char* とは異なる世紀の値を持つことがあります。

日付書式については、「日付書式モデル」（3-65 ページ）を参照してください。

例

```
INSERT INTO bonus (bonus_date)
  SELECT TO_DATE(
    'January 15, 1989, 11:00 A.M.',
    'Month dd, YYYY, HH:MI A.M.',
     'NLS_DATE_LANGUAGE = American')
     FROM DUAL;
```

TO_MULTI_BYTE

構文

```
TO_MULTI_BYTE(char)
```

用途

シングルバイト文字のすべてを、対応するマルチバイト文字に変換して *char* を戻します。*char* 内に等価なマルチバイト文字がないシングルバイト文字は、シングルバイト文字として出力されます。この関数は、使用しているデータベースのキャラクタ・セットに、シングルバイト文字とマルチバイト文字の両方が含まれている場合にだけ有効です。

TO_NUMBER

構文

```
TO_NUMBER(char [,fmt [, 'nlsparams'] ])
```

用途

オプションの書式モデル *fmt* によって指定した書式の数を含む CHAR データ型または VARCHAR2 データ型の値 *char* を NUMBER データ型の値に変換します。

例 1

```
UPDATE emp SET sal = sal +
   TO_NUMBER('100.00', '9G999D99')
  WHERE ename = 'BLAKE';
```

この関数内の *nlsparams* 文字列は、数値変換用の TO_CHAR 関数内で同じ用途に使用されます。

例 2

```
SELECT TO_NUMBER('-AusDollars100','L9G999D99',
 ' NLS_NUMERIC_CHARACTERS = ',.'
   NLS_CURRENCY = "           AusDollars"
 ') "Amount"
   FROM DUAL;

    Amount
----------
      -100
```

TO_SINGLE_BYTE

構文

```
TO_SINGLE_BYTE(char)
```

用途

マルチバイト文字のすべてを、対応するシングルバイト文字に変換して *char* を戻します。*char* 内に等価なシングルバイト文字がないマルチバイト文字は、マルチバイト文字として出力されます。この関数は、使用しているデータベースのキャラクタ・セットに、シングルバイト文字とマルチバイト文字の両方が含まれている場合にだけ有効です。

TRANSLATE USING

構文

```
TRANSLATE(text USING {CHAR_CS | NCHAR_CS })
```

用途

text を、データベース・キャラクタ・セットと各国キャラクタ・セットの変換のために指定されたキャラクタ・セットに変換します。

引数 *text* は変換する式です。

USING CHAR_CS 引数を指定すると、*text* がデータベース・キャラクタ・セットに変換されます。出力データ型は VARCHAR2 です。

USING NCHAR_CS 引数を指定すると、*text* が各国キャラクタ・セットに変換されます。出力データ型は NVARCHAR2 です。

この関数は、Oracle の CONVERT 関数と似ていますが、入力または出力のデータ型に NCHAR または NVARCHAR2 を使用する場合は、CONVERT ではなくこの関数を使う必要があります。

例 1

```
CREATE TABLE t1 (char_col CHAR(20),
                 nchar_col nchar(20));
INSERT INTO t1
  VALUES ('Hi', N'Bye');
SELECT * FROM t1;

CHAR_COL     NCHAR_COL
--------     ---------
Hi           Bye
```

例 2

```
UPDATE t1 SET
  nchar_col = TRANSLATE(char_col USING NCHAR_CS);
UPDATE t1 SET
  char_col = TRANSLATE(nchar_col USING CHAR_CS);
SELECT * FROM t1;

CHAR_COL     NCHAR_COL
--------     ---------
Hi           Hi
```

例 3

```
UPDATE t1 SET
  nchar_col = TRANSLATE('deo' USING NCHAR_CS);
UPDATE t1 SET
  char_col = TRANSLATE(N'deo' USING CHAR_CS);

CHAR_COL     NCHAR_COL
--------     ---------
deo          deo
```

その他の単一行関数

DUMP

構文

```
DUMP(expr[,return_format[,start_position[,length]] ] )
```

用途

expr のデータ型コード、および長さ（単位はバイト）、内部表現を含む VARCHAR2 値を戻します。戻される結果は常にデータベース・キャラクタ・セットの文字です。各コードに対応するデータ型については、表 2–1（2-5 ページ）を参照してください。

引数 *return_format* には戻り値の書式として、次の値のいずれかを指定します。

デフォルトでは、戻り値にキャラクタ・セット情報が含まれません。*expr* のキャラクタ・セット名を取り出すには、下記のいずれかの書式値に 1000 を加えて指定します。たとえば、*return_format* に 1008 を指定すると、8 進表記で結果が戻り、さらに expr のキャラクタ・セット名が得られます。

8　結果は 8 進表記で戻されます。

10　結果は 10 進表記で戻されます。

16　結果は 16 進表記で戻されます。

17　結果は単一文字として戻されます。

引数 *start_position* と *length* を組み合わせて、戻される内部表現の部分を指定します。何も指定しないと 10 進表記で全体の内部表現が戻されます。

expr が NULL の場合、この関数は 'NULL' を戻します。

例 1

```
SELECT DUMP('abc', 1016)
 FROM DUAL;

DUMP('ABC' ,1016)
-----------------------------------------
Typ=96 Len=3 CharacterSet=WE8DEC: 61,62,63
```

例 2

```
SELECT DUMP(ename, 8, 3, 2) "OCTAL"
 FROM emp
 WHERE ename = 'SCOTT';

OCTAL
----------------------------
Type=1 Len=5: 117,124
```

例 3

```
SELECT DUMP(ename, 10, 3, 2) "ASCII"
FROM emp
WHERE ename = 'SCOTT';

ASCII
----------------------------
Type=1 Len=5: 79,84
```

EMPTY_[B | C]LOB

構文

```
EMPTY_[B|C]LOB()
```

用途

LOB 変数を初期化するため、あるいは INSERT または UPDATE 文で LOB 列または属性を EMPTY に初期化するために使用できる空の LOB ロケータを戻します。EMPTY とは、LOB は初期化されているが、データが挿入されていない状態を言います。

この関数から戻されるロケータは、DBMS_LOB パッケージまたは OCI へのパラメータとして使用できません。

例

```
INSERT INTO lob_tab1 VALUES (EMPTY_BLOB());
UPDATE lob_tab1
  SET clob_col = EMPTY_BLOB();
```

BFILENAME

構文

```
BFILENAME ('directory', 'filename')
```

用途

サーバーのファイル・システムの物理 LOB ファイルに対応付けられている BFILE ロケータを戻します。'directory' には、ファイルが実際に格納されているサーバー・ファイル・システム上での完全パス名の別名を指定します。'filename' には、サーバー・ファイル・システムでのファイル名を指定します。

BFILENAME を指定する時点では、'directory' および 'filename' はファイル・システムに存在しているオブジェクトを指している必要はありません。ただし、後続の SQL、PL/SQL、DBMS_LOB パッケージ、または OCI の操作を実行する前には、BFILE 値を物理ファイルに関連付けておく必要があります。詳細は、「CREATE DIRECTORY コマンド」（4-226 ページ）を参照してください。

注意 : この関数は、指定されたディレクトリやファイルが実際に存在しているかどうかを確認はしません。したがって、BFILENAME の後で CREATE DIRECTORY コマンドを呼び出すこともできます。だだし、BFILE ロケータを実際に使用するときには、そのオブジェクトが存在している必要があります（たとえば、OCILobFileOpen() や DBMS_LOB.FILEOPEN() のような OCILob または DBMS_LOB の操作の 1 つに対するパラメータとして）。

LOB の詳細は、『Oracle8 Server アプリケーション開発者ガイド』および『Oracle8 コール・インタフェース・プログラマーズ・ガイド』を参照してください。

例

```
INSERT INTO file_tbl
   VALUES (BFILENAME ('lob_dir1', 'image1.gif'));
```

GREATEST

構文

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

用途

リストされた *expr* 内の最大値を戻します。2 番目以降のすべての *expr* は、比較の前に最初の *expr* のデータ型に暗黙的に変換されます。Oracle は非空白埋め比較方法で *expr* を比較します。文字の比較はデータベースのキャラクタ・セットの文字値に基づいて行われます。文字値の大きい文字が、別の文字より大きい文字と見なされます。この関数によって戻される値が文字データである場合、そのデータ型は常に VARCHAR2 となります。

例

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

Great
-----
HARRY
```

LEAST

構文

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

用途

リストされた *expr* 内の最少値を戻します。比較の前に、2 番目以降のすべての *expr* は最初の *expr* のデータ型に暗黙的に変換されます。Oracle は非空白埋め比較方法で *expr* を比較します。この関数によって戻される値が文字データである場合、そのデータ型は常に VARCHAR2 となります。

例

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

LEAST
------
  HAROLD
```

NLS_CHARSET_DECL_LEN

構文

```
NLS_CHARSET_DECL_LEN(bytecnt, csid)
```

用途

NCHAR 列の宣言の長さ（文字数）を戻します。引数 *bytecnt* は、列の長さを指定します。引数 *csid* は、列のキャラクタ・セット ID を指定します。

例

```
SELECT NLS_CHARSET_DECL_LEN
  (200, nls_charset_id('ja16eucfixed'))
   FROM DUAL;

NLS_CHARSET_DECL_LEN(200,NLS_CHARSET_ID(' JA16EUCFIXED'))
---------------------------------------------------------
                      100
```

NLS_CHARSET_ID

構文

```
NLS_CHARSET_ID(text)
```

用途

NLS キャラクタ・セット名 *text* に対応する NLS キャラクタ・セットの ID 番号を戻します。引数 *text* には、実行時の VARCHAR2 値を指定します。*text* の値を 'CHAR_CS' に指定すると、サーバーのデータベース・キャラクタ・セットの ID 番号が戻されます。*text* の値を 'NCHAR_CS' に指定すると、サーバーの各国キャラクタ・セットの ID 番号が戻されます。

無効なキャラクタ・セット名を指定すると、NULL が戻されます。

キャラクタ・セット名のリストについては、『Oracle8 Server リファレンス』を参照してください。

例 1

```
SELECT NLS_CHARSET_ID('ja16euc ')
  FROM DUAL;

NLS_CHARSET_ID('JA16EUC ')
--------------------------
                       830
```

例 2

```
SELECT NLS_CHARSET_ID('char_cs ')
  FROM DUAL;

 NLS_CHARSET_ID('CHAR_CS ')
 ----x---------------------
                          2
```

例 3

```
SELECT NLS_CHARSET_ID('nchar_cs ')
  FROM DUAL;

NLS_CHARSET_ID('NCHAR_CS ')
---------------------------
                          2
```

NLS_CHARSET_NAME

構文

```
NLS_CHARSET_NAME(n)
```

用途

ID 番号 *n* に対応する NLS キャラクタ・セット名を戻します。キャラクタ・セット名は、データベース・キャラクタ・セットの VARCHAR2 値として戻されます。

n が有効なキャラクタ・セット ID として認識されない場合は、この関数によって NULL が戻されます。

キャラクタ・セット ID のリストについては、『Oracle8 Server リファレンス』を参照してください。

例

```
SELECT NLS_CHARSET_NAME(2)
  FROM DUAL;

NLS_CH
------
WE8DEC
```

NVL

構文

```
NVL(expr1, expr2)
```

用途

expr1 が NULL の場合には *expr2* を戻します。*expr1* が NULL でない場合には *expr1* を戻します。引数 *expr1* と *expr2* は任意のデータ型を持つことができます。2 つのデータ型が異なる場合、Oracle は *expr2* を *expr1* のデータ型に変換した後で比較を行います。戻り値のデータ型は、常に *expr1* のデータ型と同じになります。ただし、*expr1* が文字データの場合は、戻り値のデータ型は VARCHAR2 となります。

例

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

ENAME      COMMISSION
---------- ------------------------------------
ALLEN      300
WARD       500
MARTIN     1400
BLAKE      NOT APPLICABLE
TURNER     0
JAMES      NOT APPLICABLE
```

UID

構文

```
UID
```

用途

現行のユーザーを一意に識別する整数を戻します。

USER

構文

```
USER
```

用途

現行の Oracle ユーザーの名前をデータ型 VARCHAR2 で戻します。Oracle は、空白埋め比較方法でこの関数の値を比較します。

分散 SQL 文では、UID 関数と USER 関数は、ローカル・データベース上のユーザーを識別します。CHECK 制約の条件でこれらの関数は使用できません。

例

```
SELECT USER, UID FROM DUAL;

USER                                UID
------------------------------ ----------
SCOTT                                  19
```

USERENV

構文

```
USERENV(option)
```

用途

現行のセッションに関する VARCHAR2 データ型の情報を戻します。この情報は、アプリケーション固有の監査証跡表を書き込む場合や現行セッションで使用されている言語固有の文字を判定する場合に有効です。CHECK 制約の条件で USERENV は使用できません。引数 option には次の値のいずれかを指定できます。

'ISDBA'	現在使用可能な ISDBA ロールを持っている場合は 'TRUE' を、持っていない場合は 'FALSE' を戻します。
'LANGUAGE'	現行セッションで使用している言語 (language) と地域 (territory) をデータベースのキャラクタ・セット (character set) も含めて次の形で戻します。 language_territory.characterset
'TERMINAL'	現行セッションの端末に対するオペレーティング・システム識別子を戻します。分散 SQL 文では、このオプションはローカル・セッションの識別子を戻します。分散環境では、リモートの SELECT に対してだけこのオプションを使用でき、リモートの INSERT または UPDATE、DELETE には使用できません。
'SESSIONID'	監査セッション識別子を戻します。このオプションは分散 SQL 文で使用できません。USERENV でこのキーワードを使用するには、初期化パラメータ AUDIT_TRAIL を TRUE に設定しなければなりません。
'ENTRYID'	使用可能な監査エントリ識別子を戻します。このオプションは分散 SQL 文で使用できません。USERENV でこのキーワードを使用するには、初期化パラメータ AUDIT_TRAIL を TRUE に設定しなければなりません。
'LANG'	言語名の ISO 略称を戻します。これは、既存の 'LANGUAGE' パラメータを短縮したものです。
'INSTANCE'	現行のインスタンスのインスタンス ID 番号を戻します。

例

```
SELECT USERENV('LANGUAGE') "Language" FROM DUAL;

Language
-----------------------------------
AMERICAN_AMERICA.WE8DEC
```

VSIZE

構文

```
VSIZE(expr)
```

用途

expr の内部表現でのバイト数を戻します。*expr* が NULL である場合には NULL を戻します。

例

```
SELECT ename, VSIZE (ename) "BYTES"
  FROM emp
  WHERE deptno = 10;

ENAME           BYTES
---------- ----------
CLARK               5
KING                4
MILLER              6
```

オブジェクト参照関数

オブジェクト参照関数は、REF(指定されたオブジェクト型のオブジェクトへの参照) を操作します。REF の詳細は、『Oracle8 概要』および『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

DEREF

構文

```
DEREF(e)
```

用途

引数 *e* のオブジェクト参照を戻します。引数 *e* は、オブジェクトに対する REF を戻す式でなければなりません。

例

```
CREATE TABLE tb1(c1 NUMBER, c2 REF t1);
SELECT DEREF(c2) FROM tb1;
```

REFTOHEX

構文

```
REFTOHEX(r)
```

用途

引数 *r* を 16 進で表した文字の値に変換します。

例

```
CREATE TABLE tb1(c1 NUMBER, c2 REF t1);
SELECT REFTOHEX(c2) FROM tb1;
```

MAKE_REF

構文

```
MAKE_REF(table, key [,key...])
```

用途

key を主キーとして使用して、オブジェクト・ビューの行に対する REF を作成します。オブジェクト・ビューの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

例

```
CREATE TYPE t1 AS OBJECT(a NUMBER, b NUMBER);

CREATE TABLE tb1
   (c1 NUMBER, c2 NUMBER, PRIMARY KEY(c1, c2));

CREATE VIEW v1 OF t1 WITH OBJECT OID(a, b) AS
   SELECT * FROM tb1;

SELECT MAKE_REF(v1, 1, 3) FROM DUAL;
```

グループ関数

グループ関数は、単一行に基づく結果を戻すのではなく、行のグループに基づく結果を戻します。この点で、グループ関数は単一行関数と異なります。グループ関数と単一行関数の違いについては、「SQL 関数」（3-15 ページ）を参照してください。

多くのグループ関数は次のオプションを取ります。

DISTINCT　このオプションを指定すると、グループ関数は異なる値を持つ引数式だけに作用します。

ALL　このオプションを指定すると、グループ関数は重複を含めてすべての値に作用します。

たとえば、1、1、1、3 の平均値は DISTINCT では 2 となり、ALL では 1.5 となります。どちらのオプションも指定しないと ALL が使用されます。

COUNT(*) 以外のすべてのグループ関数で NULL は無視されます。グループ関数に対する引数で NVL を使用して、NULL を値で置き換えることができます。

グループ関数を持つ問合せが、グループ関数への引数として行を戻さなかったり、NULL を持つ行だけを戻したりすると、グループ関数は NULL を戻します。

AVG

構文

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

用途

n の平均値を戻します。

例

```
SELECT AVG(sal) "Average"
   FROM emp;

   Average
----------
2077.21429
```

COUNT

構文

```
COUNT({* | [DISTINCT|ALL] expr})
```

用途

問合せ内の行数を戻します。

expr を指定すると、この関数は *expr* が NULL でない行数を戻します。*expr* の行をすべて数えることもできますし、異なる値だけを数えることもできます。

アスタリスク (*) を指定すると、この関数は重複値や NULL 値も含めたすべての行数を戻します。

例 1

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

   Total
----------
      18
```

例 2

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

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

例 3

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

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

MAX

構文

```
MAX([DISTINCT|ALL] expr)
```

用途

expr の最大値を戻します。

例

```
SELECT MAX(sal) "Maximum" FROM emp;

   Maximum
----------
      5000
```

MIN

構文

```
MIN([DISTINCT|ALL] expr)
```

用途

expr の最小値を戻します。

例

```
SELECT MIN(hiredate) "Earliest" FROM emp;

Earliest
---------
17-DEC-80
```

STDDEV

構文

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

用途

x の標準偏差を戻します。Oracle は、VARIANCE グループ関数に対して定義された分散の平方根として標準偏差を計算します。

例

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

 Deviation
----------
1182.50322
```

SUM

構文

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

用途

n の合計値を戻します。

例

```
SELECT SUM(sal) "Total"
     FROM emp;

     Total
----------
     29081
```

VARIANCE

構文

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

用途

x の分散を戻します。Oracle は次の公式を使用して、*x* の分散を計算します。

$$\frac{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left[\sum_{i=1}^{n} x_i\right]^2}{n-1}$$

ここで、それぞれの意味は次のとおりです。

x_i は、*x* の要素の 1 つです。

n は、セット x の要素の数です。n が 1 の場合、分散は 0 となります。

例

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

Variance
----------
1389313.87
```

ユーザー関数

PL/SQL で独自のユーザー関数を作成し、SQL や SQL 関数にはない機能を持たせることができます。ユーザー関数は、SQL 関数が使用可能なすべての SQL 文で使えます。つまり、式を置くことができる場所であればどこでも使用できます。

たとえば、次の箇所でユーザー関数を使用できます。

- SELECT コマンドの選択リスト
- WHERE 句の条件
- CONNECT BY 句、および START WITH 句、ORDER BY 句、GROUP BY 句
- INSERT コマンドの VALUES 句
- UPDATE コマンドの SET 句

ユーザー関数の作成と使用方法の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

前提条件

ユーザー関数を作成する場合、トップ・レベルの PL/SQL 関数として作成するか、これらの関数を SQL 文で使用する前にパッケージ仕様部で宣言する必要があります。「CREATE FUNCTION コマンド」（4-228 ページ）で説明されている CREATEFUNCTION 文を使用すると、ユーザー関数はトップ・レベルの PL/SQL 関数として作成されます。パッケージ関数は、「CREATE PACKAGE コマンド」（4-246 ページ）で説明されている CREATE PACKAGE 文を含むパッケージで指定します。

パッケージ・ユーザー関数を呼び出すには、パッケージ仕様部で RESTRICT_REFERENCES プラグマを宣言する必要があります。

必要な権限

SQL の式の中でユーザー関数を使用するには、ユーザーがユーザー関数の EXECUTE 権限の所有者であるか、その権限を与えられている必要があります。ユーザー関数で定義したビューを問い合せるには、そのビューに対する SELECT 権限がなければなりません。ビューから選択するには、個別の EXECUTE 権限は必要ありません。

ユーザー関数での制約

不変の定義が必要な場合には、ユーザー関数を使用できません。したがって、ユーザー関数には次の制約があります。

- CREATE TABLE コマンドまたは ALTER TABLE コマンドの CHECK 制約句では使用できない。
- CREATE TABLE コマンドまたは ALTER TABLE コマンドの DEFAULT 句では使用できない。
- OUT または IN OUT パラメータを含めることはできない。
- データベースは更新できない。
- 関数がリモート関数の場合は、パッケージ状態の読取りまたは書込みはできない。
- 関数がパッケージ状態を変更する場合は、関数内の SQL コマンドで parallelism_clause は使用できない。
- 関数内で定義された変数は更新できない。ただし、関数がローカル関数であり、しかも SELECT リストまたは INSERT コマンドの VALUES 句、UPDATE コマンドの SET 句で使用されている場合は例外です。

名前の優先順位

PL/SQL では、データベースの列名は、パラメータなしの関数名よりも優先順位が高くなります。たとえば、ユーザー SCOTT が自分のスキーマ内で次の 2 つのオブジェクトを作成するとします。

```
CREATE TABLE emp(new_sal NUMBER, ...);
```

```
CREATE FUNCTION new_sal RETURN NUMBER IS BEGIN ... END;
```

その後、次の 2 つの文のように NEW_SAL を参照すると列 EMP.NEW_SAL を参照することになります。

```
SELECT new_sal FROM emp;
SELECT emp.new_sal FROM emp;
```

関数 NEW_SAL にアクセスするには、次のように入力します。

```
SELECT scott.new_sal FROM emp;
```

SQL の式内で使用可能なユーザー関数の呼出し例を次に示します。

```
circle_area (radius)
payroll.tax_rate (empno)
scott.payroll.tax_rate (dependent, empno)@ny
```

例 たとえば、スキーマ SCOTT から TAX_RATE ユーザー関数を呼び出し、TAX_TABLE 内の SS_NO 列と SAL 列に対してこの関数を実行し、その結果を変数 INCOME_TAX に入れるには、次のように指定します。

```
SELECT scott.tax_rate (ss_no, sal)
    INTO income_tax
    FROM tax_table
    WHERE ss_no = tax_id;
```

命名規則

オプションのスキーマまたはパッケージの名前の中から 1 つだけを指定すると、最初の識別子はスキーマ名またはパッケージ名のどちらかになります。たとえば、PAYROLL.TAX_RATE という参照内の PAYROLL がスキーマ名かパッケージ名かを判定するには、Oracle は次の手順を実行します。

- 現行スキーマ内の PAYROLL パッケージをチェックする。
- PAYROLL パッケージが見つからない場合は、トップ・レベルに TAX_RATE 関数を含むスキーマ名 PAYROLL を検索する。そのような関数が見つからない場合には、エラー・メッセージを戻します。
- 現行スキーマ内で PAYROLL パッケージが見つかると、その中で TAX_RATE 関数を検索する。そのような関数が見つからない場合には、エラー・メッセージを戻します。

また、ユーザーが定義したシノニムを使用して、ストアド・トップ・レベル・ファンクションを参照できます。

書式モデル

書式モデルは、文字列に格納される DATE データや NUMBER データの書式を記述する文字リテラルです。書式モデルは、次の目的で、TO_CHAR 関数や TO_DATE 関数の引数として使用できます。

- Oracle がデータベースから値を戻す場合に使用する書式を指定する。
- Oracle がデータベースに格納するために指定した値の書式を指定する。

なお、書式モデルによってデータベース内に格納された値の内部表現は変更されません。

この項では次の使用方法について説明します。

- 数値書式モデル
- 日付書式モデル
- 書式モデルの修飾子

戻り値の書式の変更

書式モデルを使用して、データベースから値を戻す場合に Oracle が使用する書式を指定できます。

例 1 次の文は、部門 30 の従業員のコミッション値を選択し、TO_CHAR 関数を使用して、そのコミッション値を数値書式モデル ’$9,990.99’ で指定した書式の文字値に変換します。

```
SELECT ename employee, TO_CHAR(comm, '$9,990.99') commission
   FROM emp
   WHERE deptno = 30;

EMPLOYEE   COMMISSION
---------- ----------
ALLEN         $300.00
WARD          $500.00
MARTIN      $1,400.00
BLAKE
TURNER          $0.00
JAMES
```

Oracle はこの書式モデルによって、ドル記号を先頭に付け、3 桁ごとにカンマで区切り、小数点以下 2 桁を持つコミッションを戻します。COMM 列が NULL のすべての従業員に対して TO_CHAR 関数が NULL を戻していることに注目してください。

例 2 次の文は、部門 20 の各従業員の入社した日付を選択し、TO_CHAR 関数を使用して、その日付を日付書式モデル ’fmMonth DD, YYYY’ で指定した書式の文字列に変換します。

```
 SELECT ename, TO_CHAR(Hiredate,'fmMonth DD, YYYY') hiredate
```

```
    FROM emp
    WHERE deptno = 20;

ENAME      HIREDATE
---------- ------------------
SMITH      December 17, 1980
JONES      April 2, 1981
SCOTT      April 19, 1987
ADAMS      May 23, 1987
FORD       December 3, 1981
LEWIS      October 23, 1997
```

Oracle はこの書式モデルによって、省略しない月の名前 (「書式モデルの修飾子」（3-70 ページ）で説明する "fm" で指定)、2 桁の日、世紀も含めた 4 桁の年で示された入社日付を戻します。

正しい書式の付与

書式モデルを使用して、あるデータ型の値を列が必要とする別のデータ型の値に変換する書式を指定できます。列の値を挿入したり、更新したりするとき、指定する値のデータ型は列のデータ型に一致しなければなりません。たとえば、DATE 列に挿入する値は、DATE データ型の値か、デフォルト日付書式の文字列値でなければなりません (Oracle は暗黙的にデフォルト日付書式の文字列を DATE データ型に変換する)。値が別の書式で与えられる場合、TO_DATE 関数を使用して値を DATE データ型に変換しなければなりません。また、文字列の書式を指定するためにも書式モデルを使用しなければなりません。

例 次の文は、TO_DATE 関数を使用して BAKER の入社日を更新します。文字列 ’1992 05 20’ を DATE 値に変換するために、書式マスク ’YYYY MM DD’ を指定します。

```
UPDATE emp
  SET hiredate = TO_DATE('1992 05 20','YYYY MM DD')
  WHERE ename = 'BLAKE';
```

数値書式モデル

次の場所で数値書式モデルを使用できます。

- NUMBER データ型の値を VARCHAR2 データ型の値に変換する TO_CHAR 関数
- CHAR データ型または VARCHAR2 データ型の値を NUMBER データ型の値に変換する TO_NUMBER 関数

すべての数値書式モデルでは、数値が指定された有効桁数に丸められます。小数点左の有効桁数が書式で指定された桁数よりも多い場合、ポンド記号 (#) が値のかわりに戻されます。正の値が非常に大きく、指定の書式で表せない場合、無限大記号 (~) が値のかわりに戻され

ます。同様に、負の値が非常に小さく、指定の書式で表せない場合、負の無限大記号 (-~) が値のかわりに戻されます。

数値書式の要素

数値書式モデルは、1 つ以上の数値書式の要素から成り立ちます。表 3–12 に数値書式モデルの要素を示します。例を表 3–13 に示します。

- 数値書式モデルに書式要素 MI または S、PR が指定されていない場合、負の戻り値には自動的に負の符号が先頭に付加され、正の戻り値には空白が先頭に付加されます。
- 数値書式モデルには単一の小数点文字 (D) またはピリオド (.) しか指定できませんが、複数のグループ・セパレータ (G) やカンマ (,) は指定できます。
- 数値書式モデルの先頭文字にカンマ (,) は指定できません。
- 数値書式モデルにおいて、グループ・セパレータ (G) やカンマは小数点文字やピリオドの右側に指定できません。

表 3–12　数値書式の要素

要素	例	説明
9	9999	正の値の場合、先頭に空白を埋め込んで指定の桁数にしてから値を戻す。
		負の値の場合、先頭に負の符号を埋め込んで指定の桁数にしてから値を戻す。
		固定小数点数の整数部分の場合、先行ゼロに対して空白を戻す。ただし、値ゼロに対してはゼロを戻す。
0	0999	先行ゼロを戻す。
	9990	後続ゼロを戻す。
$	$9999	値の前にドル記号を付けて戻す。
B	B9999	整数部がゼロの場合、書式モデル内の "0" にかかわらず、固定小数点数の整数部に対して空白を戻す。
MI	9999MI	負の値の後に負の符号 "-" を戻す。
		正の値の後に空白を戻す。
S	S9999	負の値の前に負の符号 "-" を戻す。
		正の値の前に正の符号 "+" を戻す。
	9999S	負の値の後に負の符号 "-" を戻す。
		正の値の後に正の符号 "+" を戻す。
PR	9999PR	山カッコ <> の中に負の値を戻す。
		正の値の前後に空白を付けて戻す。

表 3–12　数値書式の要素

要素	例	説明
D	99D99	指定した位置に小数点文字 (すなわちピリオド ".") を戻す。
G	9G999	指定した位置にグループ・セパレータを戻す。
C	C999	指定した位置に ISO 通貨記号を戻す。
L	L999	指定した位置に各国通貨記号を戻す。
,(カンマ)	9,999	指定した位置にカンマを戻す。
. (ピリオド)	99.99	指定した位置に小数点 (すなわちピリオド ".") を戻す。
V	999V99	値を 10 の *n* 乗倍にして戻す（必要に応じて数値を丸める）。この例では *n* は "V" の後の "9" の数。
EEEE	9.9EEEE	科学表記法で値を戻す。
RN	RN	大文字のローマ数字で値を戻す。
rn		小文字のローマ数字で値を戻す。
		値は 1 ～ 3999 の整数となる。
FM	FM90.9	前後に空白を付けずに値を戻す。

例　表 3–13 は、異なる number 値と *'fmt'* に対する次の問合せの結果を示しています。

```
SELECT TO_CHAR(number, 'fmt')
  FROM DUAL
```

表 3–13　数値変換例の結果

number	'fmt'	結果
-1234567890	9999999999S	'1234567890-'
0	99.99	' .00'
+0.1	99.99	' 0.10'
-0.2	99.99	' -.20'
0	90.99	' 0.00'
+0.1	90.99	' 0.10'
-0.2	90.99	' -0.20'
0	9999	' 0'
1	9999	' 1'

表 3–13　数値変換例の結果

number	'fmt'	結果
0	B9999	'　　'
1	B9999	' 1'
0	B90.99	'　　'
+123.456	999.999	' 123.456'
-123.456	999.999	'-123.456'
+123.456	FM999.009	'123.456'
+123.456	9.9EEEE	' 1.2E+02'
+1E+123	9.9EEEE	' 1.0E+123'
+123.456	FM9.9EEEE	'1.23E+02'
+123.45	FM999.009	'123.45'
+123.0	FM999.009	'123.00'
+123.45	L999.99	' $123.45'
+123.45	FML99.99	'$123.45'
+1234567890	9999999999S	'1234567890+'

書式要素 MI および PR は、数値書式モデルの最後の位置にだけ指定できます。書式要素 S は、数値書式モデルの最初か最後の位置にだけ指定できます。

これらの書式要素によって戻される文字には、初期化パラメータによって指定されるものもあります。表 3–14 に、これらの要素とパラメータを示します。

表 3–14　初期化パラメータによって決定される数値書式要素の値

要素	説明	初期化パラメータ
D	小数点文字	NLS_NUMERIC_CHARACTER
G	グループ・セパレータ	NLS_NUMERIC_CHARACTER
C	ISO 通貨記号	NLS_ISO_CURRENCY
L	各国通貨記号	NLS_CURRENCY

これらの書式要素によって戻される文字は、初期化パラメータ NLS_TERRITORY を使用して暗黙的に指定することもできます。これらのパラメータの詳細は、『Oracle8 Server リファレンス』を参照してください。

これらの書式要素によって戻される文字をセッションごとに変更するには、ALTER SESSION コマンドを使用します。また、デフォルト日付書式をセッションごとに変更する場

合も、ALTER SESSION コマンドを使用します。詳細は、「ALTER SESSION コマンド」（4-58 ページ）を参照してください。

日付書式モデル

次の場所で日付書式モデルを使用できます。

- デフォルト日付書式以外の書式の DATE 値を変換する TO_CHAR 関数
- デフォルト日付書式以外の書式の文字値を変換する TO_DATE 関数

デフォルト日付書式

デフォルト日付書式は、初期化パラメータ NLS_DATE_FORMAT で明示的に指定することも、初期化パラメータ NLS_TERRITORY で暗黙的に指定することもできます。これらのパラメータの詳細は、『Oracle8 Server リファレンス』を参照してください。

デフォルト日付書式をセッションごとに変更するには、ALTER SESSION コマンドを使用します。詳細は、「ALTER SESSION コマンド」（4-58 ページ）を参照してください。

最大長

日付書式モデルの合計長は最大 22 文字です。

日付書式要素

日付書式モデルは、表 3–15 に示す、1 つ以上の日付書式要素から構成されます。入力書式モデルの場合、同じ書式項目は 2 回指定できません。また、類似した情報を表す書式項目を組み合せることもできません。たとえば、’SYYYY’ と ’BC’ は同一の書式文字列内で使用できません。表 3–15 に示すように、日付書式要素には、TO_DATE 関数で使えるものと使えないものがあります。

表 3–15　日付書式要素

要素	TO_DATE で指定できるか	意味
- / , . ; : ’text’	はい	結果に取り込まれる句読点とテキスト。
AD A.D.	はい	ピリオド付き / なしで西暦を示す。

表 3–15　日付書式要素

要素	TO_DATE で指定できるか	意味
AM A.M.	はい	ピリオド付き / なしで午前を示す。
BC B.C.	はい	ピリオド付き / なしで紀元前を示す。
CC SCC	いいえ	4 桁の年の最初の 2 桁より 1 大きい数。"S" を指定すると紀元前の日付の先頭に "-" が付けられる。たとえば、’1900’ の場合は ’20’ となる。
D	はい	曜日 (1 ～ 7)。
DAY	はい	曜日。9 文字分の長さまで空白が詰められる。
DD	はい	月における日 (1 ～ 31)。
DDD	はい	年における日 (1 ～ 366)。
DY	はい	曜日の省略形。
E	いいえ	時代名の略称 (日本、韓国、タイ)。
EE	いいえ	時代名の完全名称 (日本、韓国、タイ)。
HH	はい	時間 (1 ～ 12)。
HH12	いいえ	時間 (1 ～ 12)。
HH24	はい	時間 (0 ～ 23)。
IW	いいえ	ISO 規格に基づく、年における週 (1 ～ 52 または 1 ～ 53)。
IYY IY I	いいえ	それぞれ ISO 年の下 3 桁、2 桁、1 桁。
IYYY	いいえ	ISO 規格に基づいた 4 桁の年。
J	はい	ユリウス暦。紀元前 4712 年 1 月 1 日から経過した日数。’J’ を付けて指定する数値は、整数でなければならない。
MI	はい	分 (0 ～ 59)。
MM	はい	月 (01 ～ 12;1 月 =01)。
MON	はい	月の名前の省略形。
MONTH	はい	月の名前。9 文字分の長さまで空白が詰められる。

表 3–15　日付書式要素

要素	TO_DATE で指定できるか	意味
PM P.M.	いいえ	ピリオド付き / なしで午前を示す。
Q	いいえ	年の四半期 (1、2、3、4;1 月～ 3 月 =1)。
RM	はい	ローマ数字で表した月 (I ～ XII;1 月 =I)。
RR	はい	年を 2 桁に丸める。指定した年が <50 で現在の年の下 2 桁が >=50 の場合は、次の世紀の年を戻す。指定した年が >=50 で現在の年の下 2 桁が <50 の場合は、前の世紀の年を戻す。
RRRR	はい	年を丸める。4 桁または 2 桁で入力できる。2 桁の場合、RR の場合と同様の結果が戻る。年を 4 桁で入力すれば、この処理は行われない。
SS	はい	秒 (0 ～ 59)。
SSSSS	はい	午前 0 時から経過した秒 (0 ～ 86399)。
WW	いいえ	年における週 (1 ～ 53)。第 1 週はその年の 1 月 1 日で始まり、1 月 7 日で終了する。
W	いいえ	月における週 (1 ～ 5)。第 1 週はその月の 1 日で始まり、7 日で終了する。
Y,YYY	はい	指定した位置にカンマを付けた年。
YEAR SYEAR	いいえ	綴りで表した年。"S" を指定すると紀元前の日付の先頭に "-" が付けられる。
YYYY SYYYY	はい	4 桁の年。"S" を指定すると紀元前の日付の先頭に "-" が付けられる。
YYY YY Y	はい	それぞれ年の下 3 桁、2 桁、1 桁。

書式文字列では句読点文字がある場所に、日付文字列の中では英数字がある場合、Oracle はエラーを戻します。次に例を示します。

```
TO_CHAR (TO_DATE('0297','MM/YY'), 'MM/YY')
```

この場合、エラーが戻ります。

日付書式要素と各国語サポート

いくつかの日付書式要素の機能は、Oracle を使用している国および言語に依存します。たとえば、以下の日付書式要素は綴りで値が戻されます。

- MONTH
- MON
- DAY
- DY
- BC または AD、B.C.、A.D.
- AM または PM、A.M.、P.M.。

これらの値を戻す言語は、初期化パラメータ NLS_DATE_LANGUAGE によって明示的に指定することも、初期化パラメータ NLS_LANGUAGE によって暗黙的に指定することもできます。日付書式要素 YEAR と SYEAR によって戻される値は常に英語です。

日付書式要素 D は曜日の数 (1 ～ 7) を戻します。この数が 1 である曜日は初期化パラメータ NLS_TERRITORY によって暗黙的に指定されます。

これらの初期化パラメータの詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

ISO 標準日付書式要素

Oracle は、ISO 規格に従った日付書式要素 IYYY および IYY、IY、I、IW によって戻される値を計算します。これらの値と日付書式要素 YYYY および YYY、YY、Y、WW によって戻される値の相違点については、『Oracle8 Server リファレンス』の「各国語サポート」の章を参照してください。

RR 日付書式要素

日付書式要素 RR は日付書式要素 YY に類似していますが、20 世紀以外の日付値を格納する上でより優れた柔軟性をもたらします。日付書式要素 RR によって、現在が 20 世紀でも、年の下 2 桁を指定するだけで、21 世紀の日付を格納できます。また、必要であれば、同じ方法により、21 世紀の時点でも 20 世紀の日付を格納できます。

TO_DATE 関数で日付書式要素 YY を使用すると、日付値は常にその時点の世紀で戻されます。そのかわりに日付書式要素 RR を使用すると、年に指定した 2 桁の数とその年の下 2 桁の数によって戻り値の世紀が変化します。表 3–16 に RR 日付書式要素の性質をまとめます。

表 3–16　RR 日付書式要素

		指定された 2 桁の年	
		0 ～ 49	50 ～ 99
現在の年の下 2 桁	0 ～ 49	戻る日付は現在の世紀	戻る日付は前の世紀
	50-99	戻る日付は次の世紀	戻る日付は現在の世紀

次の例によって日付書式要素 RR の性質を具体的に説明します。

例 1　これらの問合せが、1950 年～ 1999 年の間に発行されると想定します。

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR') ,'YYYY') "Year"
     FROM DUAL;

Year
----
1995

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY') "Year";
     FROM DUAL;

Year
----
2017
```

例 2　これらの問合せが、2000 年～ 2049 年の間に発行されると想定します。

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR') ,'YYYY') "Year";
  FROM DUAL;

Year
----
1995

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY') "Year";
     FROM DUAL;

Year
----
2017
```

発行される年 (2000 年の前または後) にかかわらず、問合せが同じ値を戻していることに注目してください。日付書式要素 RR によって、世紀が変わっても同じ値を戻す SQL 文を記述できます。

日付書式要素の接尾辞

表 3–17 に日付書式要素に付加できる接尾辞を示します。

表 3–17　日付書式要素の接尾辞

接尾辞	意味	要素の例	値の例
TH	序数	DDTH	4TH
SP	綴りで表した数	DDSP	FOUR
SPTH または THSP	綴りで表した序数	DDSPTH	FOURTH

これらの接尾辞を 1 つでも日付書式要素に付加すると、戻り値は常に英語です。

注意： 日付の接尾辞は出力でだけ有効です。データベースに日付を挿入するためには使用できません。

日付書式要素での先頭文字の大文字化

戻される日付値ではその大文字と小文字は対応する書式要素の表記に従います。たとえば、日付書式モデル 'DAY' は 'MONDAY'、'Day' は 'Monday'、'day' は 'monday' を生成します。

日付書式モデルにおける句読点と文字リテラル

日付書式モデルでは次の文字も指定できます。

- ハイフン、スラッシュ、カンマ、ピリオド、コロンなどの句読点
- 文字リテラル (二重引用符で囲みます)

これらの文字は、戻り値の中で書式モデルに指定された同じ位置に現れます。

書式モデルの修飾子

TO_CHAR 関数の書式モデルで修飾子 FM と FX を使用して、空白の埋め方および書式検査を制御できます。

修飾子は書式モデルに複数指定できます。この場合、後の修飾子は前の修飾子の効果を逆にします。第 1 の修飾子の効果は、それに後続するモデル部分に対して有効になり、第 2 の修飾子が指定されると、その後のモデル部分で無効になります。第 3 の修飾子が指定されると、その後のモデル部分で再び有効になります。以下、同様に続きます。

FM "Fill mode(埋込みモード)" です。この修飾子は TO_CHAR 関数の戻り値における空白の埋込みを抑制します。

- TO_CHAR 関数の日付書式要素では、この修飾子は後続の文字要素 (MONTH など) では空白を抑制し、日付書式モデルの後続の数要素 (MI など) では前後のゼロを抑制します。FM を指定していない場合、文字要素の結果は常に右に空白を埋め込んだ固定長となり、数要素に対しては常に先行ゼロが戻されます。FM を指定した場合、空白の埋込みがないために、戻り値の長さが異なることもあります。
- TO_CHAR 関数の数値書式要素では、この修飾子によって数値の左に加えられた空白が抑制されます。これにより、その結果は出力バッファ中で左揃えになります。FM を指定していない場合、結果は常に右揃えとなり、数の左に空白が埋め込まれます。

FX "Format exact(厳密な書式一致)" です。この修飾子は TO_DATE 関数の文字引数と日付書式モデルに対して、厳密な一致を指定します。

- 文字引数における句読点と引用符で囲まれたテキストは書式モデルの対応する部分と（大文字小文字の違いを除いて）厳密に一致しなければなりません。
- 文字引数には余分な空白を含めることはできません。FX を指定していない場合、Oracle は余分な空白を無視します。
- 文字引数における数値データの桁数は、書式モデルの対応する要素と同じ桁数でなければなりません。FX を指定していない場合、文字引数における数値により先行ゼロが省略されます。

  FX が使用可能になっているとき、FM 修飾子を指定して、この先行ゼロの検査を使用禁止にできます。

文字引数の位置がこれらの条件に違反する場合、Oracle はエラー・メッセージを戻します。

例 1 次の文は日付書式モデルを使用して文字式を戻します。

```
SELECT TO_CHAR(SYSDATE, 'fmDDTH')||' of '||TO_CHAR
   (SYSDATE, 'Month')||', '||TO_CHAR(SYSDATE, 'YYYY') "Ides"
    FROM DUAL;

Ides
------------------
3RD of April, 1995
```

この文は FM 修飾子も使用していることに注目してください。FM を指定しないと、月は次のように空白を埋め込んで 9 文字にして戻されます。

```
SELECT TO_CHAR(SYSDATE, ’DDTH’)||’ of ’||
   TO_CHAR(Month, YYYY’) "Ides"
   FROM DUAL;
```

```
Ides
--------------------
03RD of April , 1995
```

例 2 次の文は、2 つの連続した単一引用符を含む日付書式モデルを使用することによって、戻り値に単一引用符を含めます。

```
SELECT TO_CHAR(SYSDATE, 'fmDay')||'''s Special') "Menu"
     FROM DUAL;

Menu
-----------------
Tuesday's Special
```

書式モデル内の文字リテラルにおいても、同じ目的で単一引用符を 2 つ連続して使用できます。

例 3 表 3–18 は、*char* 値と *'fmt'* の様々な組合せに対し、次の文が FX を使用したマッチング条件を満たしているかどうかを示しています。

```
UPDATE table
  SET date_column = TO_DATE(char, 'fmt');
```

表 3–18 FX 書式モデル修飾子によるマッチング文字データと書式モデル

char	'fmt'	一致とエラー
'15/ JAN /1993'	'DD-MON-YYYY'	一致
' 15! JAN % /1993'	'DD-MON-YYYY'	エラー
'15/JAN/1993'	'FXDD-MON-YYYY'	エラー
'15-JAN-1993'	'FXDD-MON-YYYY'	一致
'1-JAN-1993'	'FXDD-MON-YYYY'	エラー
'01-JAN-1993'	'FXDD-MON-YYYY'	一致
'1-JAN-1993'	'FXFMDD-MON-YYYY'	一致

文字列から日付への変換に関する規則

次の追加の形式化の規則は、文字列値を日付値に変換する場合に適用されます。

- 先行ゼロを含む数値書式要素の桁がすべて指定されている場合は、日付文字列から書式文字列に含まれる句読点を省略できます。つまり、MM、DD、YY などの 2 桁の書式要素については、2 のかわりに 02 を指定してください。
- 日付文字列から、書式文字列の最後にある時間フィールドを省略できます。

- 日付書式要素と日付文字列内の対応する文字のマッチングに失敗した場合、表 3–19 に示すように、元の書式要素のかわりに、別の書式要素の適用が試みられます。

表 3–19　Oracle 形式マッチング

元の書式要素	元の書式要素のかわりに試行する書式要素
'MM'	'MON' および 'MONTH'
'MON	'MONTH'
'MONTH'	'MON'
'YY'	'YYYY'
'RR'	'RRRR'

式

式は、1 つ以上の値および演算子、値を評価する SQL 関数の組合せです。一般的に式のデータ型は、そのコンポーネントのデータ型となります。

次の単純式は、値が 4 になり、データ型は NUMBER データ型（コンポーネントと同じデータ型）になります。

```
2*2
```

次の例は、関数と演算子を使用した複雑な式です。この式は今日の日付に 7 日を加算し、その合計から時間コンポーネントを削除し、結果を CHAR データ型に変換します。

```
TO_CHAR(TRUNC(SYSDATE+7))
```

次の場所で式を使用できます。

- SELECT コマンドの選択リスト
- WHERE 句と HAVING 句の条件
- CONNECT BY 句および START WITH 句、ORDER BY 句
- INSERT コマンドの VALUES 句
- UPDATE コマンドの SET 句

たとえば、次の UPDATE 文の SET 句で引用符で囲まれた文字列 'smith' のかわりに式を使用することもできます。

```
SET ename = 'smith';
```

この SET 句では、引用符で囲まれた文字列 'smith' のかわりに、LOWER(ename) を使用しています。

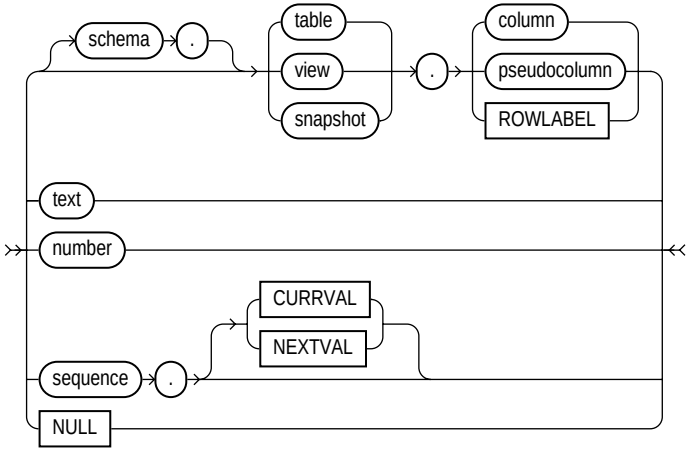
```
SET ename = LOWER(ename);
```

式にはいくつかの書式があります。Oracle は、SQL 全コマンドのすべての部分で、式の書式を全部受け入れるわけではありません。このマニュアルの他の箇所で、条件または SQL 関数、SQL コマンドに *expr* が示されている場合は、必ず適切な式の表記法を使用してください。このマニュアルの第 4 章「コマンド」にある各コマンドの項では、コマンドに指定する式の制限について説明してあります。次の項では、いくつかの例をあげて様々な式の形式を説明します。

書式 1

列または疑似列、定数、順序番号、NULL。

expr_form_I ::=



スキーマは各自用の他に、"PUBLIC"(二重引用符が必要) にもなり得ます。その場合、スキーマは表またはビュー、スナップショットのパブリック・シノニムを修飾しなければなりません。"PUBLIC" でのパブリック・シノニムの修飾は、データ操作言語 (DML) のコマンドでだけサポートされています。データ定義言語 (DDL) のコマンドではサポートされていません。

疑似列は、LEVEL、ROWID、ROWNUM のいずれかです。疑似列は、表でだけ使用でき、ビューやスナップショットでは使用できません。NCHAR と NVARCHAR2 は、有効な疑似列または ROWLABEL データ型ではありません。疑似列については、「疑似列」（2-30 ページ）を参照してください。

Trusted Oracle を使用していない場合、式 ROWLABEL は常に NULL を戻します。ラベルと ROWLABEL の使用方法については、使用している Trusted Oracle のマニュアルを参照してください。

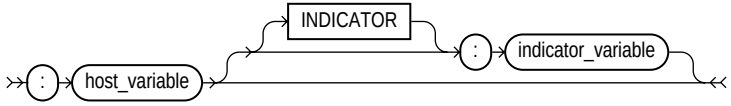
有効な書式 1 の式の例を次に示します。

```
emp.ename
'this is a text string'
10
N'this is an NCHAR string'
```

書式 2

オプションの標識変数を持つホスト変数。この書式の式は埋込み SQL 文または Oracle コール・インタフェース (OCI)・プログラムで処理される SQL 文に限り指定できます。

expr_form_II ::=



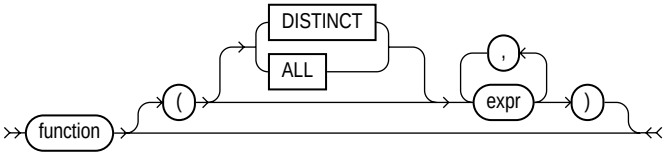
有効な書式 2 の式の例を次に示します。

```
:employee_name INDICATOR :employee_name_indicator_var
:department_location
```

書式 3

単一行を操作する SQL 関数の呼出し。

expr_form_III ::=



有効な書式 3 の式の例を次に示します。

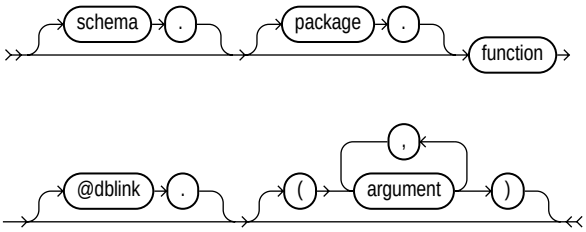
```
LENGTH('BLAKE')
ROUND(1234.567*43)
SYSDATE
```

SQL 関数については、「SQL 関数」（3-15 ページ）を参照してください。

書式 4

ユーザー関数の呼出し。

expr_form_IV ::=



有効な書式 4 の式の例を次に示します。

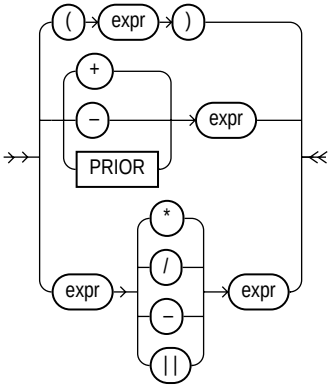
```
circle_area (radius)
payroll.tax_rate (empno)
scott.payrol.tax_rate(dependents, empno)@ny
```

ユーザー関数については、「ユーザー関数」（3-57 ページ）を参照してください。

書式 5

その他の式の組合せ。

expr_form_V ::=



関数の組合せによっては、適切でないものや拒否されるものもありますので、注意してください。たとえば、LENGTH 関数はグループ関数内では使用できません。

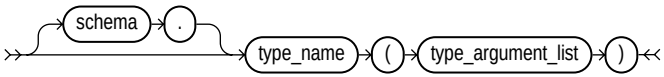
有効な書式 5 の式の例を次に示します。

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

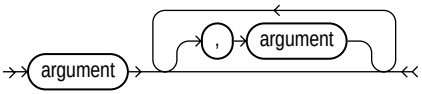
OBJ 書式 6

型コンストラクタの呼出し。

expr_form_VI ::=



type_argument_list ::=



type_name がオブジェクト型である場合、型引数のリストは、最初の引数の型がオブジェクト型の最初の属性に一致する値を取り、2 番目の引数の型がオブジェクト型の 2 番目の属性に一致するというように、順序付けられた引数のリストになっていなければなりません。コンストラクタの引数の合計は、オブジェクト型の属性の合計と一致しなければなりません。引数の最大数は 999 です。

type_name が VARRAY またはネストされた表型である場合、引数のリストには 0 個以上の引数を含めることができます。0 の引数は、空集合の構造であることを示します。それ以外の場合は、各引数が、型が集合型の要素型である要素値に対応します。

type_name がオブジェクト型、VARRAY、ネストされた表型のいずれであっても、引数の最大数は 999 です。

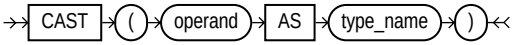
例

```
CREATE TYPE address_t AS OBJECT
  (no NUMBER, street CHAR(31), city CHAR(21), state CHAR(3), zip NUMBER);
CREATE TYPE address_book_t AS TABLE OF address_t;
DECLARE
  /* Object Type variable initialized via Object Type Constructor */
  myaddr address_t = address_t(500, 'Oracle Parkway', 'Redwood Shores',
                                'CA', 94065);
  /* nested table variable initialized to an empty table via a
     constructor*/
```

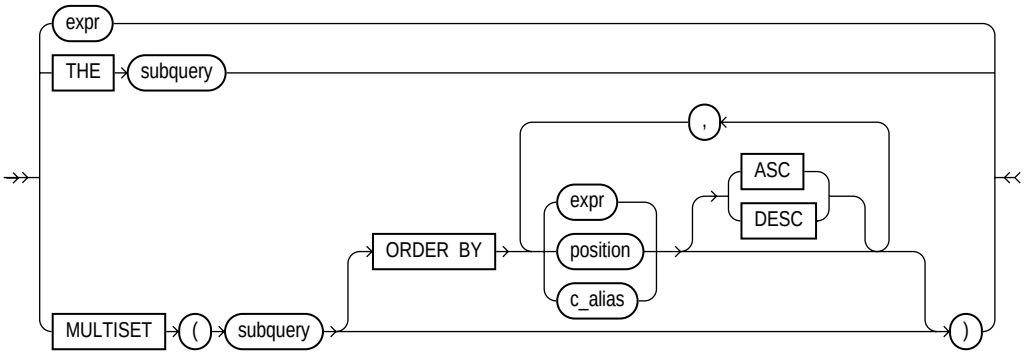
```
  alladdr address_book_t = address_book_t();
BEGIN
  /* below is an example of a nested table constructor with two elements
     specified, where each element is specified as an object type
     constructor. */
  insert into employee values (666999, address_book_t(address_t(500,
     'Oracle Parkway', 'Redwood Shores', 'CA', 94065), address_t(400,
     'Mission Street', 'Fremont', 'CA', 94555)));
END;
```

OBJ 書式 7

集合型の値を別の集合型の値に変換します。



operand ::=



CAST を使用すると、ある型の集合型値を別の集合型に変換できます。名前のない集合（副問合せの結果集合など）または名前付きの集合 (VARRAY またはネストされた表など) を型互換の名前付き集合にキャストできます。*type_name* は集合型の名前でなければならず、*operand* の値は集合値でなければなりません。

名前付き集合型を別の名前付き集合型にキャストするには、両方の集合の要素が同じ型でなければなりません。

副問合せの結果集合が複数行に評価される可能性がある場合は、MULTISET キーワードを指定する必要があります。副問合せの結果である行は、それらの行がキャストされた集合値の要素を形成します。MULTISET キーワードを指定しないと、副問合せは、CAST 式ではサポートされないスカラー副問合せとして扱われます。つまり、Oracle8 ではスカラー副問合せを CAST 演算子の引数にすることは無効です。

この項の CAST の例では、次のユーザー定義の型と表を使用します。

```
CREATE TYPE address_t AS OBJECT
      (no NUMBER, street CHAR(31), city CHAR(21), state CHAR(2));
CREATE TYPE address_book_t AS TABLE OF address_t;
CREATE TYPE address_array_t AS VARRAY(3) OF address_t;
CREATE TABLE emp_address (empno NUMBER, no NUMBER, street CHAR(31),
                              city CHAR(21), state CHAR(2));
CREATE TABLE employees (empno NUMBER, name CHAR(31));
CREATE TABLE dept (dno NUMBER, addresses address_array_t);
```

例 1

副問合せを CAST します。

```
SELECT e.empno, e.name, CAST(MULTISET(SELECT ea.no, ea.street,
                                            ea.city, ea.state
                                       FROM emp_address ea,
                                       WHERE ea.empno = e.empno)
                        AS address_book_t)
  FROM employees e;
```

例 2

CAST では、VARRAY 型の列を、ネストされた表に変換します。表の値は、フラット化した副問合せによって生成されます。「フラット化した副問合せの使用」（4-528 ページ）を参照してください。

```
SELECT *
    FROM THE(SELECT CAST(d.addresses AS address_book_t)
                FROM dept d
                  WHERE d.dno = 111) a
     WHERE a.city = 'Redwood Shores';
```

例 3

次の例は、ORDER BY 句で MULTISET 式を割り当てます。

```
CREATE TABLE projects (empid NUMBER, projname VARCHAR2(10));
CREATE TABLE employees (empid NUMBER, ename VARCHAR2(10));
CREATE TYPE projname_table_type AS TABLE OF VARCHAR2(10);
```

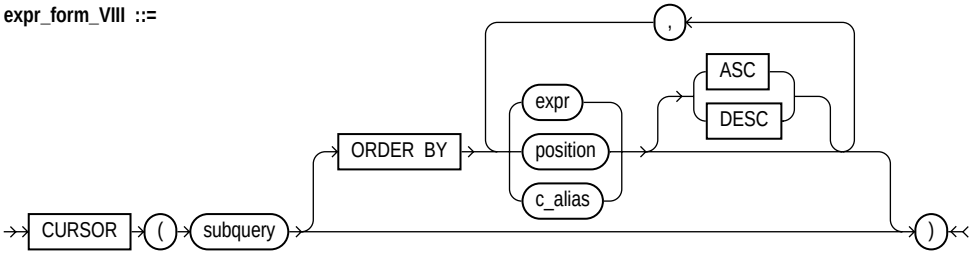
上記のスキーマでの MULTISET 式の例を次にあげます。

```
SELECT e.name, CAST(MULTISET(SELECT p.projname
                              FROM projects p
                              WHERE p.empid=e.empid
                              ORDER BY p.projname)
  AS projname_table_type)
   FROM employees e;
```

OBJ 書式 8

ネストされた CURSOR(カーソル) を戻します。この書式の式は、PL/SQL REF カーソルと同様です。

expr_form_VIII ::=



ネストされたカーソルは、含まれている行が親カーソルからフェッチされると、暗黙的にオープンされます。ネストされたカーソルは、次の場合にはクローズしています。

- ユーザーによって明示的にクローズされたとき
- 親カーソルが再実行されたとき
- 親カーソルがクローズされたとき
- 親カーソルが取り消されたとき
- 親カーソルの 1 つでのフェッチ時にエラーが発生したとき（クリーン・アップの一部としてクローズされる）

CURSOR 式には次の制限が適用されます。

- ネストされたカーソルは、他の問合せ式の中でネストされていない SELECT 文の中でだけ適用できる。ただし、CURSOR 式自体の副問合せの場合を除きます。
- ネストされたカーソルは、問合せ指定の最も外側の SELECT リストだけで表示できる。
- ネストされたカーソルはビューに表示できない。
- ネストされたカーソルに対して BIND 操作と EXECUTE 操作は実行できない。

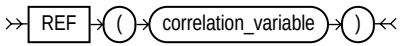
例

```
SELECT d.deptno, CURSOR(SELECT e.empno, CURSOR(SELECT p.projnum,
                                                      p.projname
                                          FROM projects p
                                          WHERE p.empno = e.empno)
                        FROM TABLE(d.employees) e)
  FROM dept d
  WHERE d.dno = 605;
```

OBJ書式 9

オブジェクトに対する参照のコンストラクタ。

expr_form_IX ::=



SQL 文では、REF の引数として、オブジェクト表またはオブジェクト・ビューの行に対応付けられている表別名がとられます。変数または行にバインドされたオブジェクト・インスタンスについての REF 値が戻ります。REF の詳細は、『Oracle8 概要』を参照してください。

例 1

```
SELECT REF(e)
FROM employee_t e
WHERE e.empno = 10000;
```

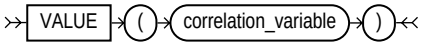
例 2 この例は REF を述語の中で使用します。

```
SELECT e.name
FROM employee_t
  e INTO :x
WHERE REF(e) = empref1;
```

OBJ書式 10

行オブジェクトを戻します。

expr_form_X ::=



SQL 文では、VALUE の引数として、オブジェクト表の行に対応付けられている相関変数（表別名）がとられます。

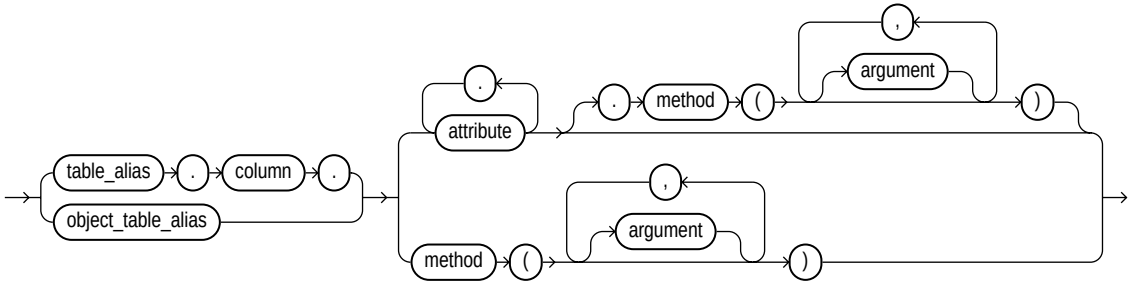
例

```
SELECT VALUE(e)
  FROM employee e
  WHERE e.name = 'John Smith';
```

OBJ書式 11

属性参照およびメソッドの起動。

expr_form_XI::=



column パラメータはオブジェクトまたは REF 列です。この項の例では、次のユーザー定義の型と表を使用します。

```
CREATE OR REPLACE TYPE employee_t AS OBJECT
    (empid NUMBER,
     name CHAR(31),
     birthdate DATE,
     MEMBER FUNCTION age RETURN NUMBER,
     PRAGMA RESTRICT_REFERENCES(age, RNPS, WNPS, WNDS)
    );
CREATE OR REPLACE TYPE BODY employee_t AS
 MEMBER FUNCTION age RETURN NUMBER IS
   var NUMBER;
   BEGIN
     var := months_between(ROUND(SYSDATE, 'YEAR'),
            ROUND(birthdate, 'YEAR'))/12;
     RETURN var ;
   END;
 END; /
CREATE TABLE department (dno NUMBER, manager EMPLOYEE_T);
```

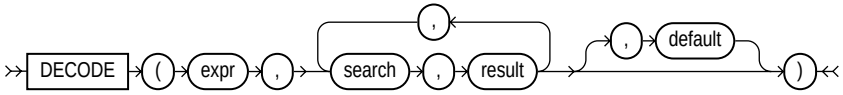
例 次の例では、上記の例で定義したオブジェクト列とメソッドを更新し、そこから選択します。

```
UPDATE department d
  SET d.manager.empid = 100;

SELECT d.manager.name, d.manager.age()
  FROM department d;
```

デコード式

特別の DECODE 構文を使用する式。

decode_expr ::=



この式を評価するために、Oracle は各 *search* 値と *expr* を 1 つずつ比較します。*expr* が *search* に等しい場合に Oracle は対応する *result* を戻します。一致しない場合には *default* が戻されます。*default* が指定されていない場合は NULL が戻されます。*expr* と *search* が文字データを含む場合、Oracle は非空白埋め比較方法で比較します。これらの比較方法については、「データ型の比較規則」（2-22 ページ）を参照してください。

search、*result*、*default* の値を式から導出できます。Oracle は *expr* と比較する前に各 *search* 値を評価します。*expr* と比較する前にすべての search 値を評価するわけではありません。その結果、*expr* に等しい *search* が見つかると、Oracle はその後の *search* を評価しません。

比較する前に、Oracle 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ます。Oracle は戻り値を最初の *result* 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 である場合、あるいは最初の *result* が NULL である場合、Oracle は戻り値を VARCHAR2 データ型の値に変換します。データ型の変換については、「データ変換」（2-26 ページ）を参照してください。

DECODE 式では、Oracle は 2 つの NULL を等価と見なします。*expr* が NULL の場合、Oracle は最初の *search* 値の *result* も NULL として戻します。

DECODE 式のコンポーネントの最大数は、*expr*、*search*、*result*、*default* を含めて 255 です。

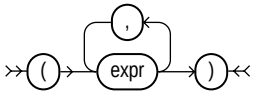
例 次の式は値 DEPTNO をデコードします。DEPTNO が 10 である場合、式は ’ACCOUNTING’ を戻します。同様に、DEPTNO が 20 である場合、式は ’RESEARCH’ を戻します。DEPTNO が 10 または 20、30、40 のどれでもない場合、式は ’NONE’ を戻します。

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

式のリスト

カッコで囲まれ、カンマで区切られた一連の式。

expr_list ::=



選択リストとして 1000 以内の式を指定できます。有効な式のリストの例を次に示します。

```
10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
(LENGTH('MOOSE') * 57, -SQRT(144) + 72, 69)
```

条件

条件は、1 つ以上の式と論理演算子の組合せで指定し、TRUE(真) または FALSE(偽)、UNKNOWN(不定) のいずれかに評価されます。このマニュアルの第 4 章「コマンド」で説明する SQL コマンドの中に *condition* が示されている場合には、必ずこの構文を使用してください。

次の文の WHERE 句で条件を使用できます。

- DELETE
- SELECT
- UPDATE

SELECT コマンドの次の任意の句で、条件を使用できます。

- WHERE
- START WITH
- CONNECT BY
- HAVING

条件は「論理」データ型であると言うこともできます。ただし、Oracle で、正式にこのようなデータ型をサポートするわけではありません。

次のような単純な条件は、常に TRUE に評価されます。

```
1 = 1
```

次のやや複雑な条件は、SAL 値を COMM 値に加算し (NULL は 0 で置き換える)、その合計が定数 2500 よりも大きいかどうかを判定します。

```
NVL(sal, 0) + NVL(comm, 0) > 2500
```

論理演算子を使用すると、複数の条件を単一の条件に結合できます。たとえば、次のように AND 演算子を使用して 2 つの条件を結合できます。

```
(1 = 1) AND (5 < 7)
```

有効な条件の例を次に示します。

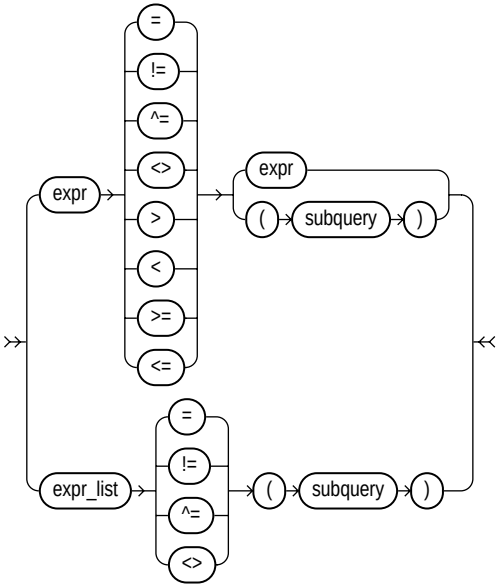
```
name = 'SMITH'
emp.deptno = dept.deptno
hiredate > '01-JAN-88'
job IN ('PRESIDENT', 'CLERK', 'ANALYST')
sal BETWEEN 500 AND 1000
comm IS NULL AND sal = 2000
```

条件には次に示すような複数の書式があります。このマニュアルの第 4 章「コマンド」にある各コマンドの説明では、コマンドに指定する条件の制限について説明しています。以降の項で様々な形式の条件を説明します。

書式 1

式または副問合せの結果との比較。

condition_form_I ::=

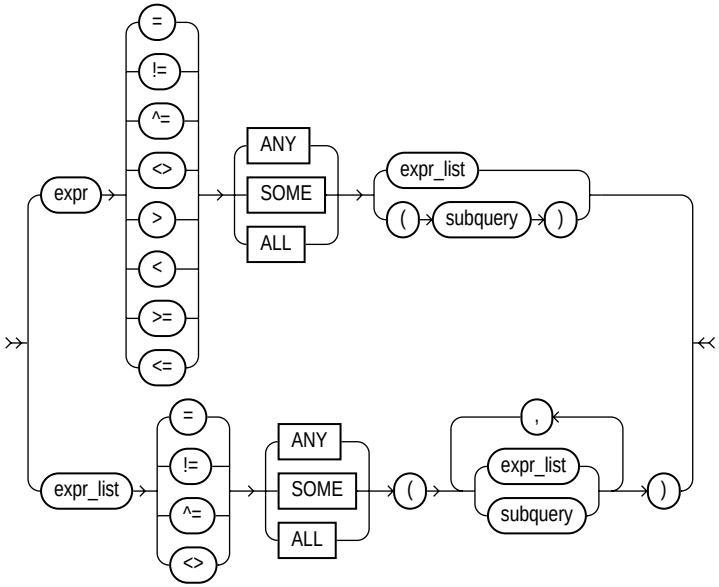


比較演算子については、「比較演算子」（3-5 ページ）を参照してください。

書式 2

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。

condition_form_II ::=

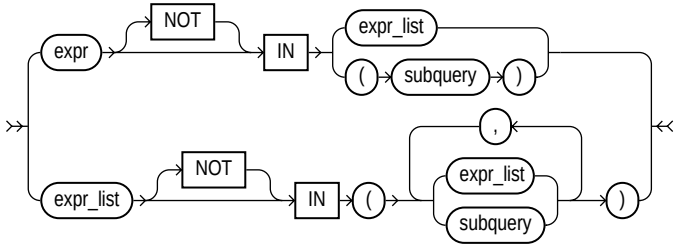


「副問合せ」（4-525 ページ）を参照してください。

書式 3

リストまたは副問合せ内のメンバーシップの検査。

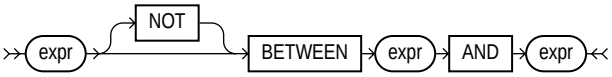
condition_form_III ::=



書式 4

範囲に含まれていることの検査。

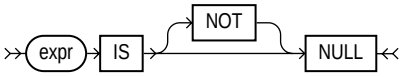
condition_form_IV ::=



書式 5

NULL の検査。

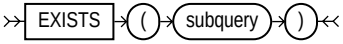
condition_form_V ::=



書式 6

副問合せにおける行の存在の検査。

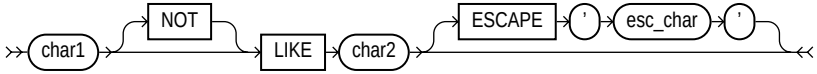
condition_form_VI ::=



書式 7

パターン・マッチングを含む検査。

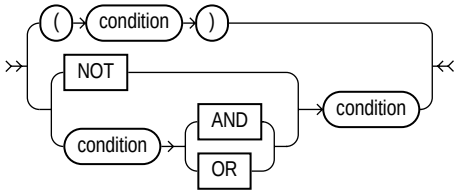
condition_form_VII ::=



書式 8

その他の条件の組合せ。

condition_form_VIII ::=



索引

記号

数字

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

ね

は

ひ

ふ

へ

ま

め

も

ゆ

よ

り

れ

ろ

わ