

Oracle9i

SQL リファレンス

リリース 2 (9.2)

2003 年 2 月

部品番号 : J06261-02

ORACLE®

Oracle9i SQL リファレンス , リリース 2 (9.2)

部品番号 : J06261-02

原本名 : Oracle9i SQL Reference, Release 2 (9.2)

原本部品番号 : A96540-02

原著者 : Diana Lorentz

原本協力者 : Joan Gregoire, Sundeep Abraham, Nipun Agarwal, Dave Alpern, Angela Amor, Patrick Amor, Rick Anderson, Vikas Arora, Lance Ashdown, Hermann Baer, Subhransu Basu, Ruth Baylis, Paula Bingham, Rae Burns, Yujie Cao, Larry Carpenter, Sivasankaran Chandrasekar, Thomas Chang, Tim Chorma, Lex de Haan, Norbert Debes, George Eadon, Bill Gietz, Ray Guzman, John Haydu, Lilian Hobbs, Jiansheng Huang, Ken Jacobs, Archana Johnson, Vishy Karra, Thomas Keefe, Susan Kotsovolos, Muralidhar Krishnaprasad, Goutam Kulkarni, Paul Lane, Shilpa Lawande, Geoff Lee, Yunrui Li, Lenore Luscher, Kevin MacDowell, Anand Manikutty, Vineet Marwah, Steve McGee, Bill McGuirk, Bill McKenna, Meghna Mehta, Tony Morales, Sujatha Muthulingam, Michael Orlowski, Jennifer Polk, Dmitry Potapov, Rebecca Reitmeyer, Kathy Rich, John Russell, Vivian Schupmann, Shrikanth Shankar, Vikram Shukla, Mike Stewart, Sankar Subramanian, Seema Sundara, Hal Takahara, Ashish Thusoo, Anh-Tuan Tran, Randy Urbano, Guhan Viswanathan, David Wang, Jim Warner, Andy Witkowski, Daniel Wong, Jianping Yang, Adiel Yoaz, Qin Yu, Tim Yu, Mohamed Zait, Fred Zemke

Copyright © 2000, 2002, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム (ソフトウェアおよびドキュメントを含む) の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、**Oracle Corporation** (米国オラクル) または日本オラクル株式会社 (日本オラクル) を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である **Oracle Corporation** (米国オラクル) およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『**Restricted Rights**』と共に提供してください。この場合次の **Notice** が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

目次

はじめに	xv

SQL リファレンスの新機能	xxiii

Vol.1

1 概要

SQL の歴史	1-2
SQL 規格	1-2
SQL の特長	1-2
すべてのリレーショナル・データベースに共通の言語	1-3
埋込み SQL	1-3
字句規則	1-4
Oracle のツール製品のサポート	1-4

2 Oracle SQL の基本要素

データ型	2-2
Oracle の組込みデータ型	2-7
文字データ型	2-9
NUMBER データ型	2-12
日時および期間データ型	2-16
ラージ・オブジェクト (LOB) データ型	2-26
ROWID データ型	2-31
ANSI、DB2、SQL/DS のデータ型	2-34

ユーザー定義型	2-35
Oracle が提供する型	2-38
任意型	2-38
XML 型	2-39
空間型	2-41
メディア型	2-41
データ型の比較規則	2-42
データ変換	2-46
リテラル	2-51
テキスト・リテラル	2-51
整数リテラル	2-52
数値リテラル	2-53
期間リテラル	2-54
書式モデル	2-58
数値書式モデル	2-60
日付書式モデル	2-64
書式モデルの修飾子	2-71
文字列から日付への変換に関する規則	2-74
XML 書式モデル	2-74
NULL	2-76
SQL ファンクションでの NULL	2-76
比較条件での NULL	2-76
条件での NULL	2-77
疑似列	2-77
CURRVAL と NEXTVAL	2-78
LEVEL	2-81
ROWID	2-82
ROWNUM	2-83
XMLDATA	2-84
コメント	2-84
SQL 文中のコメント	2-84
スキーマ・オブジェクトに関するコメント	2-86
ヒント	2-86
データベース・オブジェクト	2-101
スキーマ・オブジェクト	2-101

非スキーマ・オブジェクト	2-102
スキーマ・オブジェクトの部分	2-103
スキーマ・オブジェクト名および修飾子	2-105
スキーマ・オブジェクトのネーミング規則	2-105
スキーマ・オブジェクトのネーミング例	2-109
スキーマ・オブジェクト名のネーミング計画	2-109
スキーマ・オブジェクトの構文および SQL 文の構成要素	2-110
Oracle によるスキーマ・オブジェクト参照の変換方法	2-111
他のスキーマ内のオブジェクトの参照	2-112
リモート・データベース内のオブジェクトの参照	2-112
オブジェクト型の属性とメソッドの参照	2-115

3 演算子

SQL 演算子	3-2
単項演算子およびバイナリ演算子	3-2
演算子の優先順位	3-2
算術演算子	3-3
連結演算子	3-4
集合演算子	3-6
ユーザー定義演算子	3-6

4 式

SQL 式	4-2
単純式	4-4
複合式	4-5
CASE 式	4-6
CURSOR 式	4-8
日時式	4-10
ファンクション式	4-11
期間式	4-12
オブジェクト・アクセス式	4-12
スカラー副問合せ式	4-13
型コンストラクタ式	4-14
変数式	4-15
式のリスト	4-16

5 条件

SQL 条件	5-2
条件の優先順位	5-4
比較条件	5-4
単純比較条件	5-6
グループ比較条件	5-7
論理条件	5-8
メンバーシップ条件	5-10
範囲条件	5-12
NULL 条件	5-13
EQUALS_PATH	5-13
EXISTS 条件	5-14
LIKE 条件	5-14
IS OF <i>type</i> 条件	5-19
UNDER_PATH	5-20
複合条件	5-21

6 ファンクション

SQL ファンクション	6-2
単一行ファンクション	6-3
集計ファンクション	6-7
分析ファンクション	6-9
オブジェクト参照ファンクション	6-15
SQL ファンクションのリスト (アルファベット順)	6-15
ABS	6-15
ACOS	6-16
ADD_MONTHS	6-16
ASCII	6-17
ASCIISTR	6-18
ASIN	6-19
ATAN	6-19
ATAN2	6-20
AVG	6-20
BFILENAME	6-22
BIN_TO_NUM	6-23
BITAND	6-24

CAST	6-25
CEIL	6-28
CHARTOROWID	6-28
CHR	6-29
COALESCE	6-31
COMPOSE	6-32
CONCAT	6-33
CONVERT	6-34
CORR	6-35
COS	6-37
COSH	6-37
COUNT	6-38
COVAR_POP	6-40
COVAR_SAMP	6-42
CUME_DIST	6-44
CURRENT_DATE	6-46
CURRENT_TIMESTAMP	6-46
DBTIMEZONE	6-48
DECODE	6-48
DECOMPOSE	6-50
DENSE_RANK	6-51
DEPTH	6-52
DEREF	6-53
DUMP	6-55
EMPTY_BLOB、EMPTY_CLOB	6-56
EXISTSNODE	6-57
EXP	6-58
EXTRACT (日時)	6-58
EXTRACT (XML)	6-60
EXTRACTVALUE	6-61
FIRST	6-62
FIRST_VALUE	6-64
FLOOR	6-66
FROM_TZ	6-66
GREATEST	6-67
GROUP_ID	6-67
GROUPING	6-69
GROUPING_ID	6-70

HEXTORAW	6-71
INITCAP	6-72
INSTR	6-73
LAG	6-75
LAST	6-76
LAST_DAY	6-78
LAST_VALUE	6-79
LEAD	6-81
LEAST	6-82
LENGTH	6-82
LN	6-83
LOCALTIMESTAMP	6-84
LOG	6-85
LOWER	6-86
LPAD	6-86
LTRIM	6-87
MAKE_REF	6-88
MAX	6-89
MIN	6-91
MOD	6-92
MONTHS_BETWEEN	6-93
NCHR	6-94
NEW_TIME	6-94
NEXT_DAY	6-96
NLS_CHARSET_DECL_LEN	6-96
NLS_CHARSET_ID	6-97
NLS_CHARSET_NAME	6-98
NLS_INITCAP	6-98
NLS_LOWER	6-100
NLSSORT	6-100
NLS_UPPER	6-102
NTILE	6-103
NULLIF	6-104
NUMTODSINTERVAL	6-105
NUMTOYMINTERVAL	6-106
NVL	6-107
NVL2	6-108
PATH	6-109

PERCENT_RANK	6-110
PERCENTILE_CONT	6-112
PERCENTILE_DISC	6-114
POWER	6-115
RANK	6-116
RATIO_TO_REPORT	6-118
RAWTOHEX	6-119
RAWTONHEX	6-119
REF	6-120
REFTOHEX	6-121
REGR_ (線形リグレーション) ファンクション	6-122
REPLACE	6-129
ROUND (数値)	6-130
ROUND (日付)	6-131
ROW_NUMBER	6-131
ROWIDTOCHAR	6-133
ROWIDTONCHAR	6-133
RPAD	6-134
RTRIM	6-135
SESSIONTIMEZONE	6-135
SIGN	6-136
SIN	6-137
SINH	6-137
SOUNDEX	6-138
SQRT	6-139
STDDEV	6-140
STDDEV_POP	6-141
STDDEV_SAMP	6-142
SUBSTR	6-144
SUM	6-145
SYS_CONNECT_BY_PATH	6-147
SYS_CONTEXT	6-148
SYS_DBURIGEN	6-153
SYS_EXTRACT_UTC	6-154
SYS_GUID	6-154
SYS_TYPEID	6-155
SYS_XMLAGG	6-156
SYS_XMLGEN	6-157

SYSDATE	6-159
SYSTIMESTAMP	6-159
TAN	6-160
TANH	6-160
TO_CHAR (文字)	6-161
TO_CHAR (日時)	6-162
TO_CHAR (数値)	6-164
TO_CLOB	6-166
TO_DATE	6-166
TO_DSINTERVAL	6-168
TO_LOB	6-169
TO_MULTI_BYTE	6-170
TO_NCHAR (文字)	6-171
TO_NCHAR (日時)	6-172
TO_NCHAR (数値)	6-172
TO_NCLOB	6-173
TO_NUMBER	6-174
TO_SINGLE_BYTE	6-175
TO_TIMESTAMP	6-175
TO_TIMESTAMP_TZ	6-176
TO_YMINTERVAL	6-178
TRANSLATE	6-178
TRANSLATE ...USING	6-180
TREAT	6-181
TRIM	6-183
TRUNC (数値)	6-184
TRUNC (日付)	6-185
TZ_OFFSET	6-185
UID	6-186
UNISTR	6-187
UPDATEXML	6-188
UPPER	6-189
USER	6-190
USERENV	6-190
VALUE	6-192
VAR_POP	6-193
VAR_SAMP	6-194
VARIANCE	6-196

VSIZE	6-197
WIDTH_BUCKET	6-198
XMLAGG	6-200
XMLCOLATTVAL	6-201
XMLCONCAT	6-202
XMLELEMENT	6-203
XMLFOREST	6-206
XMLSEQUENCE	6-207
XMLTRANSFORM	6-209
ROUND および TRUNC 日付ファンクション	6-210
ユーザー定義ファンクション	6-211
前提条件	6-213
名前の優先順位	6-213

Vol.2

7 共通の SQL DDL 句

<i>allocate_extent_clause</i>	7-2
<i>constraints</i>	7-5
<i>deallocate_unused_clause</i>	7-34
<i>file_specification</i>	7-36
<i>logging_clause</i>	7-42
<i>parallel_clause</i>	7-45
<i>physical_attributes_clause</i>	7-47
<i>storage_clause</i>	7-51

8 SQL 問合せおよび副問合せ

問合せおよび副問合せ	8-2
単純な問合せの作成	8-2
階層問合せ	8-3
UNION [ALL]、INTERSECT および MINUS 演算子	8-7
問合せ結果のソート	8-9
結合	8-9
副問合せの使用方法	8-12
ネストされた副問合せのネスト解除	8-14
DUAL 表からの選択	8-14

分散問合せ	8-15

9 SQL 文 : ALTER CLUSTER ~ ALTER SEQUENCE

様々な種類の SQL 文	9-3
SQL 文の構成	9-5
ALTER CLUSTER	9-7
ALTER DATABASE	9-12
ALTER DIMENSION	9-56
ALTER FUNCTION	9-59
ALTER INDEX	9-61
ALTER INDEXTYPE	9-83
ALTER JAVA	9-85
ALTER MATERIALIZED VIEW	9-88
ALTER MATERIALIZED VIEW LOG	9-108
ALTER OPERATOR	9-115
ALTER OUTLINE	9-116
ALTER PACKAGE	9-118
ALTER PROCEDURE	9-122
ALTER PROFILE	9-125
ALTER RESOURCE COST	9-129
ALTER ROLE	9-132
ALTER ROLLBACK SEGMENT	9-134
ALTER SEQUENCE	9-138

10 SQL 文 : ALTER SESSION ~ ALTER SYSTEM

ALTER SESSION	10-2
初期化パラメータおよび ALTER SESSION	10-7
セッション・パラメータおよび ALTER SESSION	10-11
ALTER SYSTEM	10-20
初期化パラメータおよび ALTER SYSTEM	10-34

Vol.3

11 SQL 文 : ALTER TABLE ~ ALTER TABLESPACE

ALTER TABLE	11-2
ALTER TABLESPACE	11-97

12 SQL 文 : ALTER TRIGGER ~ COMMIT

ALTER TRIGGER	12-2
ALTER TYPE	12-6
ALTER USER	12-21
ALTER VIEW	12-29
ANALYZE	12-32
ASSOCIATE STATISTICS	12-46
AUDIT	12-50
CALL	12-63
COMMENT	12-67
COMMIT	12-70

13 SQL 文 : CREATE CLUSTER ~ CREATE JAVA

CREATE CLUSTER	13-2
CREATE CONTEXT	13-11
CREATE CONTROLFILE	13-14
CREATE DATABASE	13-22
CREATE DATABASE LINK	13-35
CREATE DIMENSION	13-41
CREATE DIRECTORY	13-46
CREATE FUNCTION	13-49
CREATE INDEX	13-61
CREATE INDEXTYPE	13-88
CREATE JAVA	13-90

14 SQL 文 : CREATE LIBRARY ~ CREATE SPFILE

CREATE LIBRARY	14-2
CREATE MATERIALIZED VIEW	14-4
CREATE MATERIALIZED VIEW LOG	14-31

CREATE OPERATOR	14-40
CREATE OUTLINE	14-44
CREATE PACKAGE	14-48
CREATE PACKAGE BODY	14-53
CREATE PFILE	14-58
CREATE PROCEDURE	14-60
CREATE PROFILE	14-66
CREATE ROLE	14-73
CREATE ROLLBACK SEGMENT	14-76
CREATE SCHEMA	14-80
CREATE SEQUENCE	14-82
CREATE SPFILE	14-87

Vol.4

15 SQL 文 : CREATE SYNONYM ~ CREATE TRIGGER

CREATE SYNONYM	15-2
CREATE TABLE	15-7
CREATE TABLESPACE	15-75
CREATE TEMPORARY TABLESPACE	15-86
CREATE TRIGGER	15-89

16 SQL 文 : CREATE TYPE ~ DROP ROLLBACK SEGMENT

CREATE TYPE	16-3
CREATE TYPE BODY	16-25
CREATE USER	16-31
CREATE VIEW	16-38
DELETE	16-53
DISASSOCIATE STATISTICS	16-61
DROP CLUSTER	16-64
DROP CONTEXT	16-66
DROP DATABASE LINK	16-67
DROP DIMENSION	16-69
DROP DIRECTORY	16-71
DROP FUNCTION	16-72
DROP INDEX	16-74

DROP INDEXTYPE	16-76
DROP JAVA	16-78
DROP LIBRARY	16-80
DROP MATERIALIZED VIEW	16-81
DROP MATERIALIZED VIEW LOG	16-83
DROP OPERATOR	16-85
DROP OUTLINE	16-87
DROP PACKAGE	16-88
DROP PROCEDURE	16-90
DROP PROFILE	16-92
DROP ROLE	16-94
DROP ROLLBACK SEGMENT	16-95

17 SQL 文 : DROP SEQUENCE ~ ROLLBACK

DROP SEQUENCE	17-2
DROP SYNONYM	17-4
DROP TABLE	17-6
DROP TABLESPACE	17-9
DROP TRIGGER	17-12
DROP TYPE	17-14
DROP TYPE BODY	17-17
DROP USER	17-19
DROP VIEW	17-21
EXPLAIN PLAN	17-23
GRANT	17-27
INSERT	17-52
LOCK TABLE	17-71
MERGE	17-75
NOAUDIT	17-79
RENAME	17-84
REVOKE	17-86
ROLLBACK	17-97

18 SQL 文 : SAVEPOINT ~ UPDATE

SAVEPOINT	18-2
SELECT	18-4
SET CONSTRAINT[S]	18-42

SET ROLE	18-44
SET TRANSACTION	18-47
TRUNCATE	18-51
UPDATE	18-56

A 構文図の読み方

必須キーワードとパラメータ	A-3
オプションのキーワードとパラメータ	A-3
構文のループ	A-4
複数の部分に分割された構文図	A-4
データベース・オブジェクト	A-5

B Oracle と標準 SQL

ANSI 規格	B-2
ISO 規格	B-2
Oracle の規格準拠	B-3
FIPS の規格準拠	B-10
標準 SQL に対する Oracle 拡張機能	B-11
キャラクタ・セットのサポート	B-12

C Oracle の予約語

D 例

拡張索引作成機能の使用方法	D-2
SQL 文での XML の使用方法	D-11

索引

はじめに

このマニュアルでは、Oracle のデータベースの情報を管理するために使用される Structured Query Language (SQL) について説明します。Oracle SQL は、ANSI (米国規格協会) および ISO (国際標準化機構) の SQL99 規格にさらに多くの内容を盛り込んでいます。

この章では、次の内容を説明します。

- [対象読者](#)
- [このマニュアルの構成](#)
- [関連文書](#)
- [表記規則](#)

対象読者

このマニュアルは、すべての Oracle SQL ユーザーを対象としています。

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。

Vol.1

第1章「概要」

SQL の歴史およびリレーショナル・データベースへの SQL を使用したアクセスのメリットについて説明します。

第2章「Oracle SQL の基本要素」

Oracle データベースと Oracle SQL 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」

SQL の演算子について説明します。

第4章「式」

SQL の式について説明します。

第5章「条件」

SQL の条件について説明します。

第6章「ファンクション」

SQL ファンクションの使用方法について説明します。

Vol.2

第7章「共通の SQL DDL 句」

複数の SQL 文で頻繁に使用される様々な DDL 句について説明します。

第8章「SQL 問合せおよび副問合せ」

様々な型の SQL 問合せについて説明し、様々な型の SQL 文を示します。

第9章「SQL 文：ALTER CLUSTER ～ ALTER SEQUENCE」

第10章「SQL 文：ALTER SESSION ～ ALTER SYSTEM」

Vol.3

第 11 章「SQL 文 : ALTER TABLE ~ ALTER TABLESPACE」

第 12 章「SQL 文 : ALTER TRIGGER ~ COMMIT」

第 13 章「SQL 文 : CREATE CLUSTER ~ CREATE JAVA」

第 14 章「SQL 文 : CREATE LIBRARY ~ CREATE SPFILE」

Vol.4

第 15 章「SQL 文 : CREATE SYNONYM ~ CREATE TRIGGER」

第 16 章「SQL 文 : CREATE TYPE ~ DROP ROLLBACK SEGMENT」

第 17 章「SQL 文 : DROP SEQUENCE ~ ROLLBACK」

第 18 章「SQL 文 : SAVEPOINT ~ UPDATE」

9 章から 18 章では、すべての SQL 文をアルファベット順にリストし、説明します。

付録 A「構文図の読み方」

このマニュアルでの構文図の読み方について説明します。

付録 B「Oracle と標準 SQL」

ANSI および ISO 規格に対する Oracle の準拠性について説明します。

付録 C「Oracle の予約語」

Oracle 内部で使用する予約語を示します。

付録 D「例」

複数の SQL 文を使用する様々な例を記載しています。各 SQL 文の詳細は、該当する章を参照してください。

Oracle9i リリース 2 (9.2) マニュアルでの構成変更

次の使用頻度の高い DDL 句は、第 7 章「共通の SQL DDL 句」として、個別に説明しています。

- `allocate_extent_clause` (7-2 ページ)
- `constraints` (7-5 ページ)
- `deallocate_unused_clause` (7-34 ページ)
- `file_specification` (7-36 ページ)
- `logging_clause` (7-42 ページ)
- `parallel_clause` (7-45 ページ)
- `physical_attributes_clause` (7-47 ページ)
- `storage_clause` (7-51 ページ)

以前のリリースでは、`autoextend_clause` は多数の SQL 文に含まれていました。今回のリリースでは、この属性がデータ・ファイルおよび一時ファイルに関連することを明示するために、`file_specification` の `datafile_tempfile_spec` 書式の一部として記載されています。

Oracle9i リリース 1 (9.0.1) マニュアルでの構成変更

前回のマニュアルの式、条件および問合せの章は分割されました。条件および式については 2 つの章に分割され、問合せについては、第 8 章「SQL 問合せおよび副問合せ」で説明しています。

前回のマニュアルでは、式の書式として記載されていた `CAST`、`DECODE` および `EXTRACT` (日時) は、SQL 組込みファンクションとして記載されています。

`LIKE` および、前回のマニュアルで「比較演算子」および「論理演算子」と呼ばれていた要素は、SQL の条件として記載されています。

すべての SQL 文を含む章 (前回のマニュアルの第 7 章～第 10 章) は、10 個の章に分割されました。

関連文書

詳細は、次の Oracle ドキュメントを参照してください。

- SQL に対する Oracle 手続き型言語拡張機能である PL/SQL の詳細は、『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 埋込み SQL の詳細は、『Pro*C/C++Precompiler プログラマーズ・ガイド』、『SQL*Module for Ada Programmer's Guide』および『Pro*COBOL Precompiler プログラマーズ・ガイド』を参照してください。

このマニュアルで示すほとんどの例は、Oracle のインストール時にデフォルトでインストールされる、シード・データベースのサンプル・スキーマを使用します。これらのスキーマの作成方法および使用方法については、『Oracle9i サンプル・スキーマ』を参照してください。

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J (Oracle Technology Network Japan) に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

<http://otn.oracle.co.jp/membership/>

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

<http://otn.oracle.co.jp/document/>

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

- 本文中の表記規則
- コード例中の表記規則

本文中の表記規則

本文では、特別な用語をより迅速に識別できるように、様々な表記規則を使用します。次の表に、それらの表記規則を説明し、その使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語または用語集に記載されている用語（またはその両方）を示します。	この句を指定すると、 索引構成表 (IOT) が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを取ることができます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合わせます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。

規則	意味	例
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディクショナリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名、データベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクトとデータベース構造、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニットおよびパラメータ値も含まれます。 注意：大文字と小文字を組み合わせて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	parallel_clause を指定できます。 Uold_release.SQL を実行します。ここで、old_release とはアップグレードの前にインストールしたリリースを示します。

コード例中の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文を説明します。コード例は、固定幅フォントで表示され、この例に示すとおり通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用される表記規則を説明し、その使用例を示します。

規則	意味	例
[]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	DECIMAL (<i>digits</i> [, <i>precision</i>])
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	{ENABLE DISABLE}
	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。選択項目から 1 つを入力します。縦線は入力しないでください。	{ENABLE DISABLE} [COMPRESS NOCOMPRESS]

規則	意味	例
...	水平の省略記号は、次のいずれかを示します。 ■ 例に直接関連しないコードの一部が省略されている。 ■ コードの一部を繰り返すことができる。	<pre>CREATE TABLE...AS subquery; SELECT col1, col2, ... , coln FROM employees;</pre>
. : .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	<pre>SQL> SELECT NAME FROM V\$DATAFILE; NAME ----- /fsl/dbs/tbs_01.dbf /fsl/dbs/tbs_02.dbf . . /fsl/dbs/tbs_09.dbf 9 rows selected.</pre>
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおりに入力する必要があります。	<pre>acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;</pre>
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。	<pre>CONNECT SYSTEM/system_password DB_NAME = database_name</pre>
大文字	大文字は、システムが提供する要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で示されます。用語が大カッコ内にかぎりと、表示されているとおりの順序および綴りで入力します。ただし、これらの用語は大 / 小文字を区別されないため、小文字でも入力できます。	<pre>SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;</pre>
小文字	小文字は、ユーザー定義のプログラム要素を示します。たとえば、表名、列名、ファイル名などです。 注意：大文字と小文字を組み合わせるプログラム要素もあります。これらの要素は、記載されているとおりに入力してください。	<pre>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</pre>

SQL リファレンスの新機能

この項では、Oracle9i リリース 2 (9.2) の新機能について説明し、追加情報の参照先を示します。今回のリリースへアップグレードするユーザーのために、旧リリースの新機能についても説明しています。

この項では、次の内容を説明します。

- [Oracle9i リリース 2 \(9.2\) の新機能](#)
- [Oracle9i リリース 1 \(9.0.1\) の新機能](#)
- [Oracle8i の新機能](#)

Oracle9i リリース 2 (9.2) の新機能

今回のリリースでは、次の組込み条件が新しく追加されています。

- [EQUALS_PATH](#) (5-13 ページ)
- [UNDER_PATH](#) (5-20 ページ)

今回のリリースでは、次の組込み式が拡張されています。

- 型コンストラクタ式の構文では、新しいユーザー定義コンストラクタを作成できます (4-14 ページの「[型コンストラクタ式](#)」を参照)。

今回のリリースでは、次の組込みファンクションが新しく追加されています。

- [DEPTH](#) (6-52 ページ)
- [EXTRACTVALUE](#) (6-61 ページ)
- [PATH](#) (6-109 ページ)
- [UPDATEXML](#) (6-188 ページ)
- [XMLAGG](#) (6-200 ページ)
- [XMLCONCAT](#) (6-202 ページ)
- [XMLCOLATTVAL](#) (6-201 ページ)
- [XMLELEMENT](#) (6-203 ページ)
- [XMLFOREST](#) (6-206 ページ)
- [XMLSEQUENCE](#) (6-207 ページ)
- [XMLTRANSFORM](#) (6-209 ページ)

今回のリリースでは、次の権限が新しく追加または拡張されています。

- [DEBUG CONNECT SESSION](#) システム権限 (新規) (17-35 ページ)
- [DEBUG ANY PROCEDURE](#) システム権限 (新規) (17-36 ページ)
- [DEBUG](#) オブジェクト権限 (新規) (17-45 ページ)
- プロシージャ、パッケージ、ライブラリおよびオブジェクト型の [EXECUTE](#) オブジェクト権限 (拡張)
- [GRANT ANY OBJECT PRIVILEGE](#) システム権限 (新規) (17-43 ページ)

今回のリリースでは、次のトップレベル SQL 文が新しく追加または拡張されています。

- **ALTER DATABASE** (9-12 ページ) に、スタンバイ・データベースを管理するための新しい構文、管理スタンバイ・リカバリおよび論理スタンバイ用の新しい構文が追加されました。
- **ALTER OPERATOR** (9-115 ページ) によって、既存のユーザー定義演算子が再コンパイル可能になりました。
- **ALTER TABLE** (11-2 ページ) に、列名または制約名を変更できる新しい句が追加されました。
- **CREATE DATABASE** (13-22 ページ) :
 - SYS および SYSTEM ユーザーにパスワードを割り当てるための新しい句が 2 つ追加されました。
 - ローカル管理 SYSTEM 表領域が作成可能になりました。
- **CREATE SYNONYM** (15-2 ページ) によって、オブジェクト型のシノニムが作成可能になりました。
- **CREATE TABLE** (15-7 ページ) :
 - 他のリスト・パーティションのいずれにも属さない列を取得するためのデフォルトのリスト・パーティションが作成可能になりました。
 - レンジ・リスト・コンポジット・パーティション表が作成可能になりました。
 - XMLType 型の表を作成するための構文およびレンジ・リスト・パーティション表を作成するための構文が追加されました。
 - 表のデータおよびパーティション・セグメントのデータ圧縮が可能になりました。
- **CREATE TYPE** (16-3 ページ) では、NCHAR、NVARCHAR2 および NCLOB 属性を持つオブジェクト型が作成可能になりました。
- **CREATE VIEW** (16-38 ページ) に、XMLType 型の表を変換することによって XML ビューを作成するための構文が追加されました。
- **SELECT** (18-4 ページ) には、過去の特定のシステム変更番号または時刻でデータを問い合わせる「フラッシュバック問合せ」のための構文が追加されました。

Oracle9i リリース 1 (9.0.1) の新機能

今回のリリースで新しく追加または変更された組込みデータ型は、次のとおりです。

- Oracle は、組込み型または ANSI がサポートする型が不十分な場合に、新しい型を定義するための SQL に基づくインタフェースを提供します。2-38 ページの「**Oracle が提供する型**」を参照してください。

- キャラクタ・セマンティクスを示す CHAR パラメータおよびバイト・セマンティクスを示す BYTE パラメータについて、2-9 ページの「[CHAR データ型](#)」で説明します。
- 日時機能の追加機能について、2-22 ページの「[INTERVAL YEAR TO MONTH データ型](#)」および 2-23 ページの「[INTERVAL DAY TO SECOND データ型](#)」で説明します。
- 日時機能の追加機能について、2-20 ページの「[TIMESTAMP データ型](#)」で説明します。
- キャラクタ・セマンティクスを示す CHAR パラメータおよびバイト・セマンティクスを示す BYTE パラメータについて、2-11 ページの「[VARCHAR2 データ型](#)」で説明します。

今回のリリースでは、次の式の書式が新しく追加または拡張されています。

- 「[CASE 式](#)」(拡張: 検索 CASE 式) (4-6 ページ)
- 「[CURSOR 式](#)」(拡張: REF CURSOR 引数としてファンクションに渡すことが可能) (4-8 ページ)
- 「[日時式](#)」(新規) (4-10 ページ)
- 「[期間式](#)」(新規) (4-12 ページ)
- 「[スカラー副問合せ式](#)」(新規) (4-13 ページ)

今回のリリースでは、次の条件が新しく追加されています。

- [ISO type 条件](#) (5-19 ページ)

今回のリリースでは、次の組込みファンクションが新しく追加されています。

- [ASCIISTR](#) (6-18 ページ)
- [BIN_TO_NUM](#) (6-23 ページ)
- [COALESCE](#) (6-31 ページ)
- [COMPOSE](#) (6-32 ページ)
- [CURRENT_DATE](#) (6-46 ページ)
- [CURRENT_TIMESTAMP](#) (6-46 ページ)
- [DBTIMEZONE](#) (6-48 ページ)
- [DECOMPOSE](#) (6-50 ページ)
- [EXISTSNODE](#) (6-57 ページ)
- [EXTRACT \(日時\)](#) (6-58 ページ)
- [EXTRACT \(XML\)](#) (6-60 ページ)
- [FIRST](#) (6-62 ページ)
- [FROM_TZ](#) (6-66 ページ)
- [GROUP_ID](#) (6-67 ページ)

- [GROUPING_ID](#) (6-70 ページ)
- [LAST](#) (6-76 ページ)
- [LOCALTIMESTAMP](#) (6-84 ページ)
- [NULLIF](#) (6-104 ページ)
- [PERCENTILE_CONT](#) (6-112 ページ)
- [PERCENTILE_DISC](#) (6-114 ページ)
- [RAWTONHEX](#) (6-119 ページ)
- [ROWIDTONCHAR](#) (6-133 ページ)
- [SESSIONTIMEZONE](#) (6-135 ページ)
- [SYS_CONNECT_BY_PATH](#) (6-147 ページ)
- [SYS_DBURIGEN](#) (6-153 ページ)
- [SYS_EXTRACT_UTC](#) (6-154 ページ)
- [SYS_XMLAGG](#) (6-156 ページ)
- [SYS_XMLGEN](#) (6-157 ページ)
- [SYSTIMESTAMP](#) (6-159 ページ)
- [TO_CHAR](#) (文字) (6-161 ページ)
- [TO_CLOB](#) (6-166 ページ)
- [TO_DSINTERVAL](#) (6-168 ページ)
- [TO_NCHAR](#) (文字) (6-171 ページ)
- [TO_NCHAR](#) (日時) (6-172 ページ)
- [TO_NCHAR](#) (数値) (6-172 ページ)
- [TO_NCLOB](#) (6-173 ページ)
- [TO_TIMESTAMP](#) (6-175 ページ)
- [TO_TIMESTAMP_TZ](#) (6-176 ページ)
- [TO_YMINTERVAL](#) (6-178 ページ)
- [TREAT](#) (6-181 ページ)
- [TZ_OFFSET](#) (6-185 ページ)
- [UNISTR](#) (6-187 ページ)
- [WIDTH_BUCKET](#) (6-198 ページ)

今回のリリースでは、次のファンクションが拡張されています。

- [INSTR](#) (6-73 ページ)
- [LENGTH](#) (6-82 ページ)
- [SUBSTR](#) (6-144 ページ)

今回のリリースでは、次の権限が追加されています。

- [EXEMPT ACCESS POLICY](#) システム権限 (17-43 ページ)
- [RESUMABLE](#) システム権限 (17-43 ページ)
- [SELECT ANY DICTIONARY](#) システム権限 (17-43 ページ)
- [UNDER ANY TYPE](#) システム権限 (17-41 ページ)
- [UNDER ANY VIEW](#) システム権限 (17-42 ページ)
- [UNDER](#) オブジェクト権限 (17-45 ページ)

今回のリリースでは、次の SQL 文が新しく追加されています。

- [CREATE PFILE](#) (14-58 ページ)
- [CREATE SPFILE](#) (14-87 ページ)
- [MERGE](#) (17-75 ページ)

次の SQL 文には、新しい構文が追加されています。

- [ALTER DATABASE](#) (9-12 ページ) には、データベースのマウント中にホット・バックアップ処理を終了するための新しい構文が追加されています。また、スタンバイ・データベースに関連する構文も追加されています。
- [ALTER INDEX](#) 文 (9-61 ページ) の構文では、索引を使用して統計を収集できます。
- [ALTER OUTLINE](#) (9-116 ページ) では、パブリック・アウトラインおよびプライベート・アウトラインの両方を変更できます。
- [ALTER ROLE](#) (9-132 ページ) では、アプリケーション固有のパッケージを使用したロールを識別できます。
- [ALTER SESSION](#) (10-2 ページ) では、セッション中に発行された文を、いくつかの条件において一時停止可能にするかどうかを指定できます。
- [ALTER SYSTEM](#) (10-20 ページ) では、SET 句が拡張され、新しい RESET 句が追加されています。これによって、データベースを停止状態にできます。
- [ALTER TABLE](#) (11-2 ページ) では、指定した値リストによって分割できます。
- [ALTER TYPE](#) (12-6 ページ) では、オブジェクト型の属性またはメソッドの定義を変更できます。

- [ALTER VIEW](#) (12-29 ページ) では、ビューの制約を追加できます。
- [ANALYZE](#) (12-32 ページ) には、`VALIDATE STRUCTURE` 構文の一部として、`ONLINE` および `OFFLINE` 句が追加されています。また、システム統計またはユーザー定義の統計 (またはその両方) を選択できます。
- [constraints](#) (7-5 ページ) は、制約を削除または使用禁止にするときの索引操作が簡単になっています。
- [CREATE CONTEXT](#) (13-11 ページ) には、LDAP ディレクトリまたは Oracle Call Interface (OCI) インタフェースからコンテキストを初期化する構文、およびインスタンスからコンテキストへのアクセスを可能にする構文が追加されています。
- [CREATE CONTROLFILE](#) (13-14 ページ) では、Oracle 管理ファイルを作成できます。
- [CREATE DATABASE](#) (13-22 ページ) では、データベース作成時に、デフォルトの一時表領域を作成できます。また、`UNDO` 表領域を作成することもできます。
- [CREATE FUNCTION](#) (13-49 ページ) では、パイプライン表ファンクション、パラレル表ファンクションおよびユーザー定義の集計ファンクションを作成できます。
- [CREATE OUTLINE](#) (14-44 ページ) では、パブリック・アウトラインおよびプライベート・アウトラインの両方を作成できます。
- [CREATE ROLE](#) (14-73 ページ) では、アプリケーション固有のパッケージを使用したロールを識別できます。
- [CREATE TABLE](#) (15-7 ページ) では、外部表 (データベースの外部にデータを持つ表) および Oracle 管理ファイルを作成できます。また、指定した値リストの分割もできます。
- [CREATE TABLESPACE](#) (15-75 ページ) では、空きリストと同様に、ビットマップを使用したセグメント領域管理ができます。また、Oracle 管理ファイルおよび `UNDO` 表領域を作成できます。
- [CREATE TEMPORARY TABLESPACE](#) (15-86 ページ) では、Oracle 管理ファイルを作成できます。
- [CREATE TYPE](#) (16-3 ページ) では、サブタイプを作成できます。
- [CREATE VIEW](#) (16-38 ページ) では、オブジェクト・ビューのサブビューを作成できます。また、ビューに制約を定義できます。
- [DROP TABLESPACE](#) (17-9 ページ) では、削除された表領域の内容を削除するときに、オペレーティング・システム・ファイルを削除する構文が追加されています。
- [file_specification](#) (7-36 ページ) では、Oracle 管理ファイルを作成できます。
- [INSERT](#) (17-52 ページ) では、列のデフォルト値を挿入する構文が追加されています。

- [SELECT](#) (18-4 ページ) では、GROUP BY 句の中で複数のグループ化を指定して、複数のディメンションにまたがる選択性の高い分析ができます。また、副問合せブロックに名前を割り当てることができます。ANSI に準拠した結合構文が追加されています。
- [SET TRANSACTION](#) (18-47 ページ) では、トランザクション名を指定できます。
- [UPDATE](#) (18-56 ページ) では、列をデフォルト値に更新する構文が追加されています。

Oracle8i の新機能

このバージョンでは、次の SQL ファンクションが追加されています。

- [BITAND](#) (6-24 ページ)
- [CORR](#) (6-35 ページ)
- [COVAR_POP](#) (6-40 ページ)
- [COVAR_SAMP](#) (6-42 ページ)
- [CUME_DIST](#) (6-44 ページ)
- [DENSE_RANK](#) (6-51 ページ)
- [FIRST_VALUE](#) (6-64 ページ)
- [LAG](#) (6-75 ページ)
- [LAST_VALUE](#) (6-79 ページ)
- [LEAD](#) (6-81 ページ)
- [NTILE](#) (6-103 ページ)
- [NUMTOYMINTERVAL](#) (6-106 ページ)
- [NUMTODSINTERVAL](#) (6-105 ページ)
- [NVL2](#) (6-108 ページ)
- [PERCENT_RANK](#) (6-110 ページ)
- [RANK](#) (6-116 ページ)
- [RATIO_TO_REPORT](#) (6-118 ページ)
- [REGR_](#) (線形リグレッション) ファンクション (6-122 ページ)
- [STDDEV_POP](#) (6-141 ページ)
- [STDDEV_SAMP](#) (6-142 ページ)
- [VAR_POP](#) (6-193 ページ)
- [VAR_SAMP](#) (6-194 ページ)

リリース 8.1.5 では、次の SQL 文が追加されています。

- [ALTER DIMENSION](#) (9-56 ページ)
- [ALTER JAVA](#) (9-85 ページ)
- [ALTER OUTLINE](#) (9-116 ページ)
- [ASSOCIATE STATISTICS](#) (12-46 ページ)
- [CALL](#) (12-63 ページ)
- [CREATE CONTEXT](#) (13-11 ページ)
- [CREATE DIMENSION](#) (13-41 ページ)
- [CREATE INDEXTYPE](#) (13-88 ページ)
- [CREATE JAVA](#) (13-90 ページ)
- [CREATE OPERATOR](#) (14-40 ページ)
- [CREATE OUTLINE](#) (14-44 ページ)
- [CREATE TEMPORARY TABLESPACE](#) (15-86 ページ)
- [DISASSOCIATE STATISTICS](#) (16-61 ページ)
- [DROP CONTEXT](#) (16-66 ページ)
- [DROP DIMENSION](#) (16-69 ページ)
- [DROP INDEXTYPE](#) (16-76 ページ)
- [DROP JAVA](#) (16-78 ページ)
- [DROP OPERATOR](#) (16-85 ページ)
- [DROP OUTLINE](#) (16-87 ページ)

さらに、次の機能が拡張されています。

- 集計ファンクションは機能が拡張されました。6-7 ページの「[集計ファンクション](#)」を参照してください。
- LOB 領域記憶パラメータを指定するとき、読み込み専用で LOB キャッシュを指定できます。15-7 ページの「[CREATE TABLE](#)」を参照してください。
- 式に関する項に、新しい式が追加されました。4-6 ページの「[CASE 式](#)」を参照してください。
- 副問合せのネスト解除が可能になりました。8-14 ページの「[ネストされた副問合せのネスト解除](#)」を参照してください。

Structured Query Language (SQL) とは、プログラムおよびユーザーが、Oracle データベースのデータにアクセスするために使用する一連の文です。アプリケーション・プログラムや Oracle のツール製品を使用すると、SQL を直接使用せずにデータベースにアクセスできます。ただし、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、ほとんどのデータベース・システムで使用される SQL の背景について説明します。

この章では、次の内容を説明します。

- [SQL の歴史](#)
- [SQL 規格](#)
- [埋込み SQL](#)
- [字句規則](#)
- [Oracle のツール製品のサポート](#)

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認められています。Structured English Query Language (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。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 言語として認められています。

SQL 規格

オラクル社は、業界標準に準拠するよう努力し、SQL 標準化委員会にも積極的に参加しています。業界で認知されている委員会には、ANSI (米国規格協会)、および IEC (国際電気標準会議) が電気・電子部門を担当している ISO (国際標準化機構) 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付けられますが、技術的な詳細は同じです。

1999 年 6 月に採用された最新の SQL 規格を SQL:99 といいます。新規格の正式名称は、次のとおりです。

- ANSI X3.135-1999、『Database Language SQL』、Part 1「Framework」、Part 2「Foundation」、Part 5「Bindings」
- ISO/IEC 9075:1999、『Database Language SQL』、Part 1「Framework」、Part 2「Foundation」、Part 5「Bindings」

参照： SQL: 99 規格への Oracle の規格準拠については、[付録 B「Oracle と標準 SQL」](#)で説明します。

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。技術的ないい方をする、SQL はデータ副言語です。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します。
- データへの自動的なナビゲーション・アクセス (経路設定) を実行します。

- 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す。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、最近認められた SQL のオプション部分 ISO/IEC 9075-5:1996 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 の拡張機能である Oracle の PL/SQL は、PSM に似ています。

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ことは、データの細部を操作する場合のみで済みます。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。この条件を満たすすべての行が 1 つの手順で検索され、ユーザー、別の SQL 文または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 の問合せオブティマイザが働きます。この機能によって、指定したデータに最も速くアクセスする方法が決定されます。Oracle には、オブ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 文を使用して、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、前述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

すべての主なリレーショナル・データベース管理システムは、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 で記述されたプログラムは移植性に優れているため、わずかな変更のみで他のデータベースに移行できます。

埋込み SQL

埋込み SQL は、手続き型プログラミング言語に埋め込んで使用する標準の SQL 文です。埋込み SQL 文は、Oracle プリコンパイラ関連のマニュアルに記載されています。

埋込み SQL は、次のコマンドの集まりです。

- 対話形式の Oracle のツール製品を使用した、SQL で使用できるすべての SQL コマンド (SELECT、INSERT など)。
- 手続き型プログラミング言語に標準 SQL コマンドを統合する動的 SQL 実行コマンド (PREPARE、OPEN など)。

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 文が解析され、手続き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手続き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

参照： Oracle プリコンパイラおよび埋込み SQL 文の定義については、『Pro*C/C++Precompiler プログラマーズ・ガイド』および『Pro*COBOL Precompiler プログラマーズ・ガイド』を参照してください。

字句規則

SQL 文の記述に関する次の字句規則は、Oracle の SQL 実装に対してのみ適用されますが、他のすべての SQL 実装にも一般的に適用されます。

SQL 文では、文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、空白または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 last_name,salary*12,MONTHS_BETWEEN(hire_date, SYSDATE)
      FROM employees;
```

```
SELECT last_name,
      salary * 12,
      MONTHS_BETWEEN( hire_date, SYSDATE )
FROM employees;
```

予約語、キーワード、識別子およびパラメータは、大文字と小文字を区別せずに記述できます。ただし、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。

参照： 構文の説明は、2-51 ページの「[テキスト・リテラル](#)」を参照してください。

Oracle のツール製品のサポート

ほとんどの Oracle のツール製品では、Oracle SQL のすべての機能をサポートしています。このマニュアルでは、SQL のすべての機能について説明しています。ご使用の Oracle のツール製品でサポートしていない機能がある場合は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』など、その Oracle のツール製品について記述しているマニュアルで、制限事項を確認してください。

Oracle SQL の基本要素

この章では、Oracle SQL の基本要素に関する参照情報を説明します。これらの要素は、SQL 文の最も単純な構成ブロックです。したがって、[第 9 章](#)～[第 18 章](#)で説明されている文を使用する前に、この章で説明する次の概念を理解し、[第 3 章「演算子」](#)、[第 4 章「式」](#)、[第 6 章「ファンクション」](#) および [第 8 章「SQL 問合せおよび副問合せ」](#) をよく読んでおく必要があります。

この章では、次の内容を説明します。

- [データ型](#)
- [リテラル](#)
- [書式モデル](#)
- [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・オブジェクトの構文および SQL 文の構成要素](#)

データ型

Oracle が処理する値は、それぞれ**データ型**を持ちます。値のデータ型は、固定されたプロパティの集合をその値に対応付けます。このプロパティに応じて、Oracle は、あるデータ型の値を別のデータ型の値と区別して扱います。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。

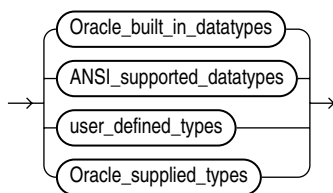
表またはクラスタを作成する場合、各列にデータ型を指定する必要があります。プロシージャまたはストアド・ファンクションを作成する場合は、その各引数にデータ型を指定する必要があります。データ型によって、各列が含むことができる値のドメイン、または各引数が持つことができる値のドメインが決まります。たとえば、DATE 列は、2 月 29 日（うるう年を除く）、2 または 'SHOE' という値を格納できません。列に入る値は、その列のデータ型を受け継ぎます。たとえば、DATE 列に '01-JAN-98' を挿入すると、Oracle はそれが有効な日付に変換されることを確認してから、文字列 '01-JAN-98' を DATE 値として扱います。

Oracle には、多くの組込みデータ型、およびデータ型として使用できるいくつかのユーザー定義型のカテゴリがあります。Oracle のデータ型の構文については、次の構文図で示します。この項は、次の 4 つの項にわかれています。

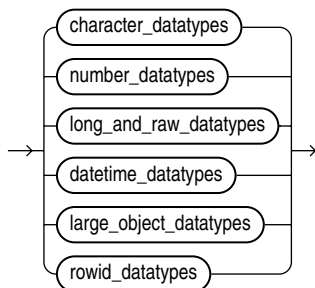
- [Oracle の組込みデータ型](#)
- [ANSI、DB2、SQL/DS のデータ型](#)
- [ユーザー定義型](#)
- [Oracle が提供する型](#)

注意： Oracle プリコンパイラによって、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。組込みデータ型およびユーザー定義型を外部データ型と混同しないでください。Oracle による外部データ型と組込み型またはユーザー定義データ型間の変換など、外部データ型の詳細は、『Pro*COBOL Precompiler プログラマーズ・ガイド』および『Pro*C/C++Precompiler プログラマーズ・ガイド』を参照してください。

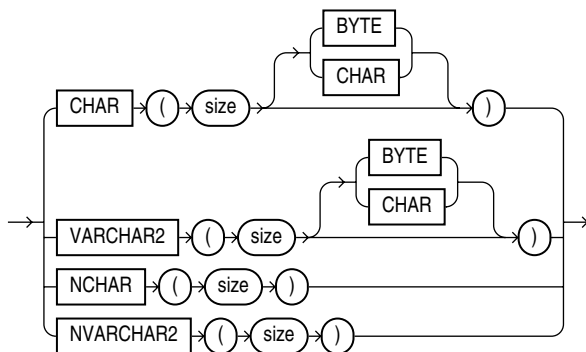
datatypes::=



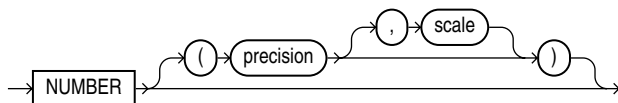
Oracle_built_in_datatypes::=



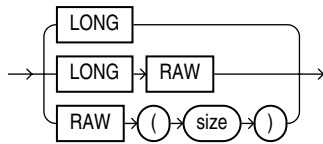
character_datatypes::=



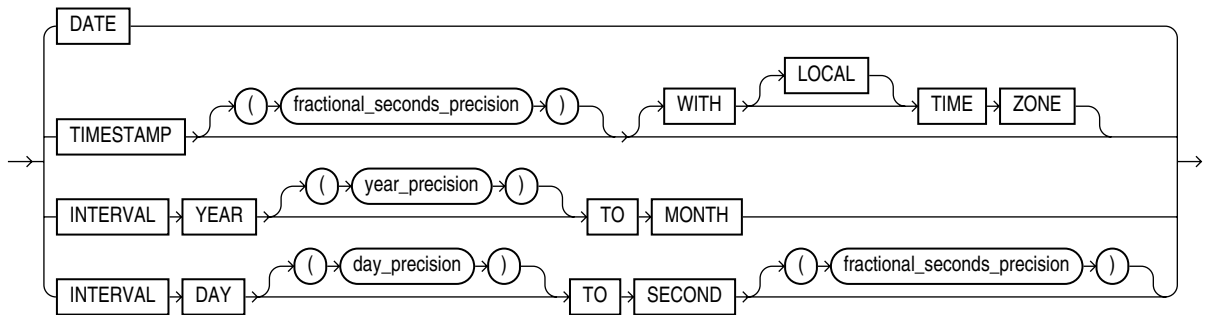
number_datatypes::=



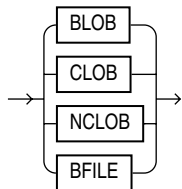
long_and_raw_datatypes::=



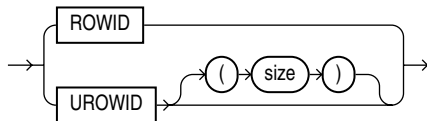
datetime_datatypes::=



large_object_datatypes::=

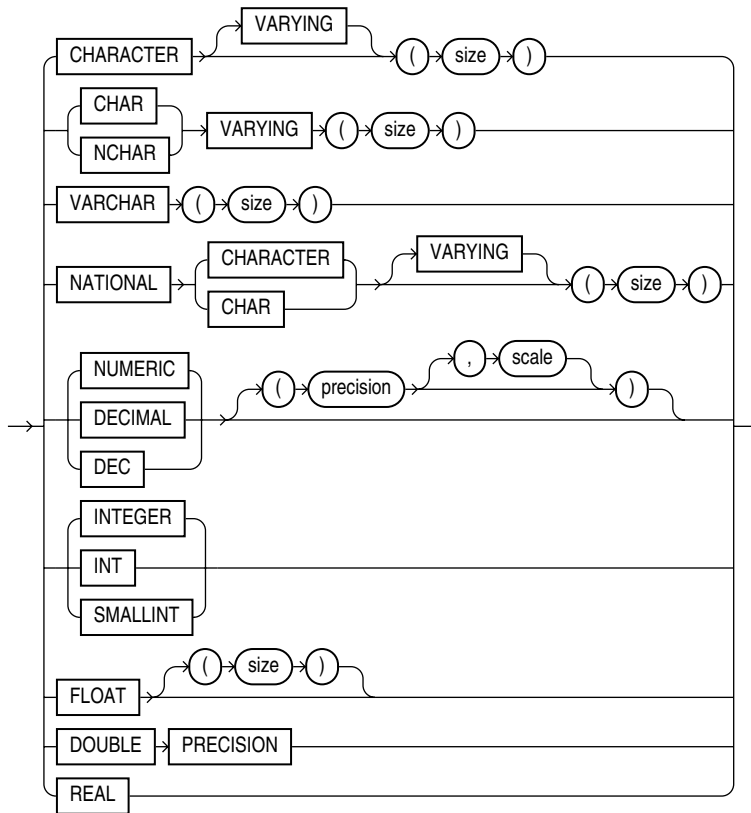


rowid_datatypes::=

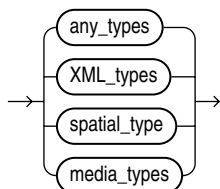


次に、ANSI がサポートしているデータ型について示します。また、2-34 ページの表 2-6 に、ANSI がサポートしているデータ型を Oracle 組込みデータ型にマップする方法を示します。

ANSI_supported_datatypes::=



Oracle_supplied_types::=



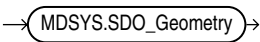
any_types::=



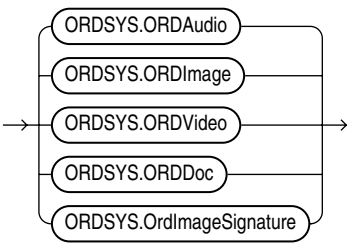
XML_types::=



spatial_type::=



media_types::=



Oracle の組込みデータ型

表 2-1 に Oracle 組込みデータ型の概要を示します。

表 2-1 組込みデータ型の概要

コード ^a	組込みデータ型	説明
1	VARCHAR2(<i>size</i>) [BYTE CHAR]	最大長が <i>size</i> バイトまたは <i>size</i> 文字の変長文字列。最大サイズは 4000、最小サイズは 1 です。VARCHAR2 には、 <i>size</i> を指定する必要があります。 BYTE は、列がバイト長セマンティクスを持つことを示し、CHAR は、列がキャラクタ・セマンティクスを持つことを示します。
1	NVARCHAR2(<i>size</i>)	最大長が <i>size</i> 文字の変長文字列。最大サイズは、各国語キャラクタ・セット定義（上限 4000 バイト）によって決定されます。NVARCHAR2 の <i>size</i> を指定する必要があります。
2	NUMBER(<i>p</i> , <i>s</i>)	精度 <i>p</i> 、位取り <i>s</i> を持つ数。精度 <i>p</i> には 1 ～ 38 の値を指定できます。位取り <i>s</i> には -84 ～ 127 の値を指定できます。
8	LONG	最大 2GB (2 ³¹ から 1 を引いたバイト数) の変長文字データ。
12	DATE	紀元前 4712 年 1 月 1 日～紀元 9999 年 12 月 31 日までの日付を指定します。
180	TIMESTAMP (<i>fractional_seconds_precision</i>)	日付の年、月、日および時刻の時、分、秒の値。 <i>fractional_seconds_precision</i> は、SECOND 日時フィールドの小数部の桁数です。 <i>fractional_seconds_precision</i> の有効範囲は、0 ～ 9 です。デフォルトは 6 です。
181	TIMESTAMP (<i>fractional_seconds_precision</i>) WITH TIME ZONE	タイムゾーンによる時差などのすべての TIMESTAMP の値。 <i>fractional_seconds_precision</i> は、SECOND 日時フィールドの小数部の桁数です。有効範囲は 0 ～ 9 です。デフォルトは 6 です。
231	TIMESTAMP (<i>fractional_seconds_precision</i>) WITH LOCAL TIME ZONE	TIMESTAMP WITH TIME ZONE のすべての値。ただし、次に示す例外があります。 ■ データベースへの格納時、データがデータベースのタイムゾーンに正規化されている場合。 ■ データの検索時、ユーザーがセッションのタイムゾーンでデータを検索する場合。

^a データ型のコードは、Oracle が内部的に使用します。DUMP ファンクションによって、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。

表 2-1 組み込みデータ型の概要（続き）

コード ^a	組み込みデータ型	説明
182	INTERVAL YEAR (<i>year_precision</i>) TO MONTH	年および月で期間を格納。 <i>year_precision</i> は、YEAR 日時フィールドの桁数です。有効範囲は 0 ～ 9 です。デフォルトは 2 です。
183	INTERVAL DAY (<i>day_precision</i>) TO SECOND (<i>fractional_seconds_precision</i>)	日、時、分および秒で期間を格納。 <i>day_precision</i> は、DAY 日時フィールドの最大桁数です。有効範囲は 0 ～ 9 です。デフォルトは 2 です。 <i>fractional_seconds_precision</i> は、SECOND フィールドの小数部の桁数です。有効範囲は 0 ～ 9 です。デフォルトは 6 です。
23	RAW(<i>size</i>)	長さ <i>size</i> バイトのバイナリ・データ。最大サイズは 2000 バイトです。RAW 値には、 <i>size</i> を指定する必要があります。
24	LONG RAW	最大 2GB の可変長バイナリ・データ。
69	ROWID	表の行のアドレスを一意に表す BASE64 文字列。主に、ROWID 疑似列によって戻される値のためのデータ型です。
208	UROWID [(<i>size</i>)]	索引構成表の行の論理アドレスを表す BASE64 文字列。オプションの <i>size</i> は、UROWID 型の列のサイズです。最大サイズおよびデフォルトは 4000 バイトです。
96	CHAR(<i>size</i>) [BYTE CHAR]	長さ <i>size</i> バイトの固定長文字データ。最大サイズは 2000 バイトです。デフォルトおよび最小サイズは 1 です。 BYTE および CHAR は、VARCHAR2 と同じセマンティクスを持ちます。
96	NCHAR(<i>size</i>)	長さ <i>size</i> 文字の固定長文字データ。最大サイズは、各国語キャラクタ・セット定義（上限 2000 バイト）によって決定されます。デフォルトおよび最小サイズは 1 です。
112	CLOB	シングルのバイト・キャラクタを含むキャラクタ・ラージ・オブジェクト。固定幅および可変幅のキャラクタ・セットがサポートされます。両方のキャラクタ・セットで CHAR データベース・キャラクタ・セットを使用します。最大サイズは 4GB です。

^a データ型のコードは、Oracle が内部的に使用します。DUMP ファンクションによって、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。

表 2-1 組み込みデータ型の概要（続き）

コード ^a	組み込みデータ型	説明
112	NCLOB	Unicode キャラクタを含むキャラクタ・ラージ・オブジェクト。固定幅および可変幅のキャラクタ・セットがサポートされます。両方のキャラクタ・セットで NCHAR データベース・キャラクタ・セットを使用します。最大サイズは 4GB です。各国語キャラクタ・セットのデータを格納します。
113	BLOB	バイナリ・ラージ・オブジェクト。最大サイズは 4GB です。
114	BFILE	データベース外に保存された大きなバイナリ・ファイルヘロケータを格納。データベース・サーバー上に存在する外部 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 です。

^a データ型のコードは、Oracle が内部的に使用します。DUMP ファンクションによって、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・キャラクタ・セットまたは各国語キャラクタ・セットの文字（英数字）を格納できます。文字データ型は、他のデータ型より制限が少ないため、プロパティも少なくなります。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 型の列が格納できるのは数値のみです。

文字データは、7 ビット ASCII や EBCDIC 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ます。

次のデータ型が、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。Oracle では、その列に格納される値がすべて size で指定した長さを持つように調整されます。列の長さより短い値を挿入すると、列の長さにあわせるため、その値の後に空白が埋め込まれます。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 バイトで、この許容最大値は 2000 バイトです。CHAR(10) 列には、1 バイトの文字列を挿入できますが、この文字列は 10 バイトまで空白埋めされてから格納されます。

CHAR 列を持つ表を作成する場合、デフォルトでは列の長さはバイト単位になります。BYTE 修飾子は、デフォルトと同じです。CHAR 修飾子（たとえば、CHAR(10 CHAR)）を使用すると、列の長さは文字単位になります。技術的ないい方をする、文字は、データベース・キャラクタ・セットのコードポイントです。サイズは、データベース・キャラクタ・セットによって異なりますが、1 バイトから 4 バイトです。BYTE および CHAR 修飾子は、NLS_LENGTH_SEMANTICS パラメータ（デフォルトはバイト・セマンティクス）で指定したセマンティクスを上書きします。

注意： 異なるキャラクタ・セットを持つデータベース間で適切にデータを変換するには、CHAR データが正しい書式の文字列で構成されていることを確認してください。キャラクタ・セット・サポートの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

参照： 比較セマンティクスについては、2-42 ページの「[データ型の比較規則](#)」を参照してください。

NCHAR データ型

Oracle9i から、NCHAR データ型は Unicode のみのデータに再定義されています。NCHAR 列を持つ表を作成する場合、列の長さを文字単位で指定します。使用する各国語キャラクタ・セットは、データベースを作成するときに指定します。

列の最大長は、各国語キャラクタ・セットの定義によって決まります。NCHAR 文字データ型の幅指定は、文字数を示します。許容最大列サイズは 2000 バイトです。

列の長さより短い値を挿入すると、列の長さにあわせるため、その値の後に空白が埋め込まれます。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、product_descriptions の translated_name 列と各国語キャラクタ・セットの文字列 LCD Monitor 11/PM を比較します。

```
SELECT translated_description from product_descriptions
WHERE translated_name = N'LCD Monitor 11/PM';
```

参照： Unicode データ型のサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NVARCHAR2 データ型

Oracle9i から、NVARCHAR2 データ型は Unicode のみのデータ型に再定義されています。NVARCHAR2 列を持つ表を作成する場合、保持できる最大の文字数を指定します。Oracle では、列の最大長を超えないかぎり、各値を指定されたとおりに正確に列に格納します。

列の最大長は、各国語キャラクタ・セットの定義によって決まります。NVARCHAR2 文字データ型の幅指定は、文字数を示します。許容最大列サイズは 4000 バイトです。

参照： Unicode データ型のサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

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、保持できるデータの最大バイト数または最大文字数を指定します。Oracle では、列の最大長を超えないかぎり、各値を指定されたとおりに正確に列に格納します。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 列には最大長を指定する必要があります。保存される文字列の実際の長さは 0（ゼロ）にできますが、最大長は 1 バイト以上にする必要があります。CHAR 修飾子（たとえば、VARCHAR2(10 CHAR)）を使用すると、バイトではなく、文字で最大長を指定できます。技術的ないい方をする、文字は、データベース・キャラクタ・セットのコードポイントです。CHAR および BYTE 修飾子は、NLS_LENGTH_SEMANTICS パラメータ（デフォルトはバイト）の設定を上書きします。VARCHAR2 データの最大長は 4000 バイトです。Oracle は、非空白埋め比較セマンティクスを使用して VARCHAR2 値を比較します。

注意： 異なるキャラクタ・セットを持つデータベース間で適切にデータを変換するには、VARCHAR2 データが正しい書式の文字列で構成されていることを確認してください。キャラクタ・セット・サポートの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

参照： 比較セマンティクスについては、2-42 ページの「[データ型の比較規則](#)」を参照してください。

VARCHAR データ型

現在、VARCHAR データ型は、VARCHAR2 データ型と同じ意味で使用されています。VARCHAR より VARCHAR2 を使用することをお勧めします。今後、VARCHAR データ型が変更され、異なる比較セマンティクスで比較される別の可変長文字列の型になる可能性があります。

NUMBER データ型

NUMBER データ型は、0（ゼロ）、および精度が 38 桁で、絶対値が $1.0 \times 10^{-130} \sim 9.9...9 \times 10^{125}$ （38 個の 9 の後に 0 が 88 個続く）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します。 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)
```

それぞれの意味は、次のとおりです。

- *p* は、精度（**precision**）、つまり全体の桁数です。Oracle は、38 桁までの精度で数の移植性を保証します。
- *s* は、位取り（**scale**）、つまり小数点の右側にある桁数です。位取りの有効範囲は -84 ～ 127 です。

次の書式で整数を指定できます。

```
NUMBER (p)
```

これは、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
```

精度および位取りを指定しない場合、最大の範囲および精度を Oracle の数値に指定したことになります。

参照： 2-13 ページの「浮動小数点数」を参照してください。

位取りと精度

入力に対する特別な整合性チェックとして、固定小数点数列の位取りと精度を指定します。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。

表 2-2 に、異なる精度および位取りを使用した Oracle のデータの格納方法を示します。

表 2-2 精度および位取りの格納

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

表 2-2 精度および位取りの格納（続き）

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 (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、実際のデータは整数部分の右から指定された桁数のみ丸められます。たとえば、(10,-2) と指定すると 100 の位まで丸められます。

精度より大きい位取り

通常はありえないことですが、精度より大きい位取りを指定することもできます。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 データ型と同様に、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）である必要があり、小数点以下 5 桁を超える値はすべて丸められます。表 2-3 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。

表 2-3 精度より大きい位取り

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。指数は、数字の後に増加する桁数を表すときに、オプションで使用されます（たとえば、 1.777 e^{-20} ）。小数点以下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。

2-12 ページの「**NUMBER データ型**」で説明する範囲で、浮動小数点数を指定できます。書式については、2-53 ページの「**数値リテラル**」を参照してください。Oracle は ANSI のデータ型である 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 では、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) は、浮動小数点数をバイナリ精度 *b* に指定します。精度 *b* は、1 ～ 126 の範囲で指定します。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 進精度の 126 桁は、10 進精度の 38 桁とほぼ等しくなります。

LONG データ型

LONG 列には、最大 2GB (2^{31} 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 列と同じ特長があります。LONG 列を使用すると、長いテキスト列を格納できます。LONG 値の長さは、ご使用のコンピュータで利用できるメモリによって制限される場合もあります。

注意： LONG 列を、できるだけ早く LOB 列へ変換することをお勧めします。今後は、新しく LONG 列を作成できなくなる予定です。

LOB 列は、LONG 列ほど制限は多くありません。LOB 機能はリリースごとに拡張されていますが、LONG 機能は最近のリリースでは変更されていません。LONG 列から LOB 列への変換については、11-2 ページの「[ALTER TABLE](#)」の「`modify_col_properties`」および 6-169 ページの「[TO_LOB](#)」を参照してください。

SQL 文の中の次の場所で LONG 列を参照できます。

- SELECT 構文のリスト
- 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ません。
- LONG 属性を持つオブジェクトは作成できません。
- LONG 列は、WHERE 句または整合性制約では指定できません (NULL および NOT NULL 制約は除く)。
- LONG 列に索引を付けることはできません。
- ストアド・ファンクションは LONG 値を戻すことはできません。
- LONG データ型を使用して、PL/SQL プログラム単位の変数または引数を宣言できます。ただし、そのプログラムは、SQL からコールできません。
- 単一の SQL 文に指定する、すべての LONG 列、更新された表、ロックされた表は、同一データベース上にある必要があります。

- LONG 列および LONG RAW 列は、分散型の SQL 文で使用できません。また、レプリケートできません。
- LONG と LOB の両方の列を持つ表では、1 つの SQL 文で両方に 4000 バイトより大きいデータをバインドすることはできません。ただし、いずれか片方にバインドすることは可能です。
- LONG 列を持つ表は、自動セグメント領域管理の表領域に格納できません。

また、LONG 列は SQL 文の次のような部分では使用できません。

- GROUP BY 句、ORDER BY 句、CONNECT BY 句または SELECT 文にある DISTINCT 演算子
- SELECT 文の一意演算子
- CREATE CLUSTER 文の列リスト
- CREATE MATERIALIZED VIEW 文の CLUSTER 句
- SQL 組み込みファンクション、式または条件
- GROUP BY 句を含む問合せの SELECT 構文のリスト
- UNION、INTERSECT または MINUS 集合演算子によって結合されている副問合せまたは問合せの SELECT 構文のリスト
- CREATE TABLE ...AS SELECT 文の SELECT 構文のリスト
- ALTER TABLE ... MOVE 文
- INSERT 文の副問合せの SELECT 構文のリスト

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 列に挿入できます。
- 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 列を参照できます。
- トリガー内の変数は、LONG データ型を使用して宣言できません。
- :NEW と :OLD は LONG 列で使用できません。

Oracle Call Interface 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。

参照：『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

日時および期間データ型

日時データ型には、DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE および TIMESTAMP WITH LOCAL TIME ZONE があります。日時データ型の値は、「日時」とも呼ばれます。期間データ型には、INTERVAL YEAR TO MONTH および INTERVAL DAY TO SECOND があります。期間データ型の値は、「期間」とも呼ばれます。

日時および期間はいずれもフィールドで構成されます。これらのフィールドの値は、データ型の値によって決まります。表 2-4 に、日時フィールド、および日時と期間の有効な値を示します。

表 2-4 日時フィールド、および日時と期間の値

日時フィールド	日時に有効な値	期間に有効な値
YEAR	-4712 ～ 9999 (0 を除きます)	正または負のすべての整数
MONTH	01 ～ 12	0 ～ 11
DAY	01 ～ 31 (現在の NLS カレンダ・パラメータに従った MONTH および YEAR の値の範囲内)	正または負のすべての整数
HOURL	00 ～ 23	0 ～ 23
MINUTE	00 ～ 59	0 ～ 59
SECOND	00 ～ 59.9(n) (「9(n)」は秒の小数部の精度) DATE には適用されません。	00 ～ 59.9(n) (「9(n)」は秒の小数部の期間の精度)
TIMEZONE_HOUR (次の注意を参照)。	-12 ～ 14 (この範囲は夏時間の変更を保存します) DATE には適用されません。	適用なし
TIMEZONE_MINUTE (次の注意を参照)。	00 ～ 59 DATE には適用されません。	適用なし
注意： TIMEZONE_HOUR および TIMEZONE_MINUTE の両方を指定すると、+ - hh:mm の形式 (-12:59 から +14:00 までの値を使用) で、エンティティとして解釈されます。		
TIMEZONE_REGION	V\$TIMEZONE_NAMES データ・ディクショナリ・ビュー内の TZNAME 列を問い合わせます。DATE には適用されません。	適用なし
TIMEZONE_ABBR	V\$TIMEZONE_NAMES データ・ディクショナリ・ビュー内の TZABBREV 列を問い合わせます。DATE には適用されません。	適用なし

注意：日時データの DML 操作で正しい結果を得るには、組み込み SQL ファンクション DBTIMEZONE および SESSIONTIMEZONE で問い合わせることによって、データベースおよびセッションのタイムゾーンを確認します。タイムゾーンを手動で設定していない場合、Oracle は、オペレーティング・システムのタイムゾーンをデフォルトで使用します。オペレーティング・システムのタイムゾーンが Oracle で有効でない場合は、Oracle は、協定世界時（UTC）（以前のグリニッジ標準時）をデフォルトの値として使用します。

DATE データ型

DATE データ型は、日付および時刻の情報を格納するために使用します。日付および時刻の情報は、文字データ型および数値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プロパティがあります。各 DATE 値には、世紀、年、月、日、時、分および秒の情報が格納されます。

日付値をリテラルに指定するか、文字値や数値を TO_DATE ファンクションによって日付値に変換できます。日付値をリテラルに指定する場合は、グレゴリオ暦を使用する必要があります。次のように、ANSI の日付リテラルを指定できます。

```
DATE '1998-12-25'
```

ANSI の日付リテラルは、時刻部分を含みません。また、'YYYY-MM-DD' という書式で指定する必要があります。また、次のように、Oracle の日付リテラルを指定できます。

```
TO_DATE('98-DEC-25:17:30','YY-MON-DD:HH24:MI')
```

Oracle の日付リテラルのデフォルトの日付書式は、初期化パラメータ NLS_DATE_FORMAT で指定します。この例は、日付としての 2 桁の数、月の名前の省略形、年の下 2 桁および 24 時間表記の時刻を含む日付書式です。

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。

日付値を指定する場合に時刻コンポーネントを指定しないと、デフォルト時刻の午前 12:00:00（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。

Oracle の DATE 列には、常に、日付フィールドと時刻フィールドが含まれます。時刻部分を除いた日付書式を使用する問合せを発行する場合は、DATE 列の時刻フィールドが 0（真夜中）に設定されていることを確認してください。そうでない場合、Oracle は、正しい結果を返さない場合があります。次の例では、数値列 row_num および DATE 列 datecol を持つ表 my_table があると想定します。

```
INSERT INTO my_table VALUES (1, SYSDATE);  
INSERT INTO my_table VALUES (2, TRUNC(SYSDATE));
```

```
SELECT * FROM my_table;
```

```
      ROW_NUM DATECOL
-----
1 04-OCT-00
2 04-OCT-00
```

```
SELECT * FROM my_table
      WHERE datecol = TO_DATE('04-OCT-00','DD-MON-YY');
```

```
      ROW_NUM DATECOL
-----
2 04-OCT-00
```

DATE 列の時刻フィールドが 0 に設定されている場合は、前述の例に示すように、DATE 列に対して問い合わせるか、DATE リテラルを使用して問い合わせることができます。

```
SELECT * FROM my_table WHERE datecol = DATE '2000-10-04';
```

ただし、DATE 列が 0（ゼロ）以外の時刻フィールドを含む場合、正しい結果を得るためには、問合せで時刻フィールドを排除する必要があります。たとえば、次のようにします。

```
SELECT * FROM my_table WHERE TRUNC(datecol) = DATE '2000-10-04';
```

Oracle は、問合せの各行に TRUNC ファンクションを適用します。これによって、データの時刻フィールドが 0 である場合のパフォーマンスが向上します。時刻フィールドが 0 に設定されていることを確認するには、挿入および更新時に次のいずれかの操作を行います。

- TO_DATE ファンクションを使用して、時刻フィールドをマスクします。

```
INSERT INTO my_table VALUES
      (3, TO_DATE('4-APR-2000','DD-MON-YYYY'));
```

- DATE リテラルを使用します。

```
INSERT INTO my_table VALUES (4, '04-OCT-00');
```

- TRUNC ファンクションを使用します。

```
INSERT INTO my_table VALUES (5, TRUNC(SYSDATE));
```

日付ファンクション SYSDATE は、現在のシステムの日付および時刻を戻します。CURRENT_DATE ファンクションは、現在のセッションの日付を戻します。SYSDATE、TO_* 日時ファンクションおよびデフォルト日付書式の詳細は、第 6 章「ファンクション」を参照してください。

日付算術 日付に対しては、日付の加算や減算のみでなく、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析します。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 employees サンプル表の 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のファンクションを用意しています。たとえば、ADD_MONTHS ファンクションを使用すると、日付に月を加算したり、減算できます。MONTHS_BETWEEN ファンクションは、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ます。

各日付には時刻コンポーネントが含まれるため、ほぼすべての日付操作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。

参照：

- 日付ファンクションの詳細は、6-5 ページの「[日時ファンクション](#)」を参照してください。
 - 算術を含むその他の日時および期間データ型については、2-23 ページの「[日時および期間の演算](#)」を参照してください。
-
-

ユリウス日の使用方法 ユリウス日は、紀元前 4712 年 1 月 1 日から経過した日数です。ユリウス日によって共通の基準で日付を算定できます。日付ファンクション TO_DATE と TO_CHAR で日付書式モデル「J」を使用して、Oracle の DATE 値とユリウス日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 日をユリウス日で戻します。

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

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

参照： DUAL 表については、8-14 ページの「[DUAL 表からの選択](#)」を参照してください。

TIMESTAMP データ型

TIMESTAMP データ型は、DATE データ型の拡張機能です。DATE データ型の年、月および日に加えて、時、分および秒の値を格納します。このデータ型は、正確な時刻の値の格納に有効です。TIMESTAMP データ型は、次のように指定します。

```
TIMESTAMP [(fractional_seconds_precision)]
```

fractional_seconds_precision には、オプションで、Oracle が格納する桁数を、SECOND 日時フィールドの小数部まで指定します。このデータ型の列を作成する場合、0 ～ 9 の値を指定できます。デフォルトは 6 です。TIMESTAMP をリテラルに指定する場合、*fractional_seconds_precision* 値には最大 9 桁を指定できます。次に例を示します。

```
TIMESTAMP'1997-01-31 09:26:50.124'
```

参照： 文字データの TIMESTAMP データへの変換については、6-175 ページの「[TO_TIMESTAMP](#)」を参照してください。

TIMESTAMP WITH TIME ZONE データ型

TIMESTAMP WITH TIME ZONE は、**タイムゾーンによる時差**を値に含む TIMESTAMP の変形です。タイムゾーンによる時差は、ローカルの時刻と UTC（時および分）との差異です。このデータ型は複数の地域にまたがる日時情報の収集および評価に有効です。

TIMESTAMP WITH TIME ZONE データ型は、次のように指定します。

```
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE
```

fractional_seconds_precision には、オプションで、Oracle が格納する桁数を、SECOND 日時フィールドの小数部まで指定します。このデータ型の列を作成する場合、0 ～ 9 の値を指定できます。デフォルトは 6 です。TIMESTAMP WITH TIME ZONE をリテラルに指定する場合、*fractional_seconds_precision* 値には最大 9 桁を指定できます。次に例を示します。

```
TIMESTAMP '1997-01-31 09:26:56.66 +02:00'
```

2 つの TIMESTAMP WITH TIME ZONE 値が UTC で同じ時刻を表す場合は、データに格納された TIME ZONE オフセットにかかわらず、同一であるとみなされます。次に例を示します。

```
TIMESTAMP '1999-04-15 8:00:00 -8:00'
```

前述の例文は次の例文と同じです。

```
TIMESTAMP '1999-04-15 11:00:00 -5:00'
```

つまり、太平洋標準時の午前 8 時は、東部標準時の午前 11 時と同じです。

UTC オフセットを TZR (タイムゾーン地域) 書式要素に置換できます。次の例では、前述の例と同じ値を持ちます。

```
TIMESTAMP '1999-04-15 8:00:00 US/Pacific'
```

夏時間に切り替えられる境界のあいまいさを排除するには、TZR および対応する TZD 書式要素の両方を使用します。次の例では、前述の例が確実に夏時間の値を戻します。

```
TIMESTAMP '1999-10-29 01:30:00 US/Pacific PDT'
```

ERROR_ON_OVERLAP_TIME セッション・パラメータを TRUE に設定しておく、TZD 書式要素を追加しなかったために日時値があいまいな場合、Oracle はエラーを戻します。パラメータを FALSE に設定しておく、Oracle はあいまいな日時を標準時刻として解析します。

注意： Oracle のタイムゾーン・データの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

参照：

- 夏時間のサポートについては、2-24 ページの「[夏時間のサポート](#)」および 2-65 ページの表 2-15「[日時書式要素](#)」を参照してください。
- 文字データの TIMESTAMP WITH TIME ZONE データへの変換は、6-176 ページの「[TO_TIMESTAMP_TZ](#)」を参照してください。
- ERROR_ON_OVERLAP_TIME セッション・パラメータについては、10-2 ページの「[ALTER SESSION](#)」を参照してください。

TIMESTAMP WITH LOCAL TIME ZONE データ型

TIMESTAMP WITH LOCAL TIME ZONE は、タイムゾーンによる時差を値に含む TIMESTAMP のもう 1 つの変形です。これは、TIMESTAMP WITH TIME ZONE と異なり、データベースに格納されるデータはデータベース・タイムゾーンに対して正規化され、タイムゾーンによる時差は列データの一部として格納されません。ユーザーがデータを検索すると、Oracle はユーザーのローカル・セッション・タイムゾーンのデータを戻します。タイムゾーンによる時差は、ローカルの時刻と UTC (時および分) との差異です。このデータ型は、2 層アプリケーションのクライアント・システムのタイムゾーンへの日時情報を表示する場合に有効です。

TIMESTAMP WITH LOCAL TIME ZONE データ型は、次のように指定します。

TIMESTAMP [(*fractional_seconds_precision*)] WITH LOCAL TIME ZONE

fractional_seconds_precision には、オプションで、Oracle が格納する桁数を、SECOND 日時フィールドの小数部まで指定します。このデータ型の列を作成する場合、0 ～ 9 の値を指定できます。デフォルトは 6 です。

TIMESTAMP WITH LOCAL TIME ZONE には、リテラルはありません。

注意： Oracle のタイムゾーン・データの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

参照：

- データ型の使用例は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 文字データの TIMESTAMP WITH LOCAL TIME ZONE データへの変換は、6-25 ページの「[CAST](#)」を参照してください。

INTERVAL YEAR TO MONTH データ型

INTERVAL YEAR TO MONTH は、YEAR および MONTH 日時フィールドを使用して期間を格納します。このデータ型は、2 つの日時の値の正確な違いを表す場合に有効です。

INTERVAL YEAR TO MONTH は、次のように指定します。

INTERVAL YEAR [(*year_precision*)] TO MONTH

year_precision は、YEAR 日時フィールドの桁数です。*year_precision* のデフォルト値は 2 です。

注意： 期間値をリテラルとして指定すると、高い柔軟性が得られます。期間値をリテラルとして指定する方法の詳細は、2-54 ページの「[期間リテラル](#)」を参照してください。

INTERVAL DAY TO SECOND データ型

INTERVAL DAY TO SECOND は、日付、時、分および秒で期間を格納します。このデータ型は、年および月の値のみが重要な場合に、2つの日時の値の正確な違いを表す場合に有効です。

このデータ型は、次のように指定します。

```
INTERVAL DAY [(day_precision)]
            TO SECOND [(fractional_seconds_precision)]
```

それぞれの意味は、次のとおりです。

- `day_precision` は、DAY 日時フィールドの桁数です。有効範囲は 0 ～ 9 です。デフォルトは 2 です。
- `fractional_seconds_precision` は、SECOND 日時フィールドの小数部の桁数です。有効範囲は 0 ～ 9 です。デフォルトは 6 です。

注意： 期間値をリテラルとして指定すると、高い柔軟性が得られます。期間値をリテラルとして指定する方法の詳細は、2-54 ページの「[期間リテラル](#)」を参照してください。

日時および期間の演算

Oracle では、日時および期間の値の式を導出できます。日時の値の式から、日時データ型の値を得ることができます。期間の値の式から、期間データ型の値を得ることができます。[表 2-5](#) に、これらの式で利用できる演算子を示します。

表 2-5 日時および期間の値の式の演算子

オペランド 1	演算子	オペランド 2	結果タイプ
日時	+	期間	日時
日時	-	期間	日時
期間	+	日時	日時
日時	-	日時	期間 ^a
期間	+	期間	期間
期間	-	期間	期間
期間	*	数値	期間
数値	*	期間	期間
期間	/	数値	期間

^a この操作は、DATE 値には無効です。

たとえば、期間値の式を開始時間に追加できます。order_date 列を持つサンプル表 oe.orders について考えます。次の文は、order_date 列の値に 30 日を加算します。

```
SELECT order_id, order_date + INTERVAL '30' DAY FROM orders;
```

Oracle は、すべてのタイムスタンプの演算を UTC 時間で実行します。TIMESTAMP WITH LOCAL TIME ZONE では、Oracle は、日時の値をデータベース・タイムゾーンから UTC に変換し、演算を実行した後にデータベース・タイムゾーンに変換しなおします。TIMESTAMP WITH TIME ZONE では、日時の値は常に UTC であるため、変換は必要ありません。

夏時間のサポート

Oracle は、指定したタイムゾーン地域に、夏時間が適用されているかを自動的に判断し、それに基づくローカル時刻の値を戻します。日時の値は、**境界**を除いたすべての指定した地域において、夏時間が適用されているかを Oracle が判断するために有効です。夏時間の開始または終了時に、境界が発生します。たとえば、米国の太平洋地域では、夏時間の開始時、時刻は午前 2 時から午前 3 時に変更されます。午前 2 時と午前 3 時の間の 1 時間は存在しません。夏時間の終了時、時刻は午前 2 時から午前 1 時に変更されます。午前 1 時と午前 2 時の間の 1 時間は繰り返されます。

このような境界を解決するために、Oracle は TZR および TZD 書式要素を使用します。詳細は、2-65 ページの表 2-15 を参照してください。TZR は、日時の入力文字列でタイムゾーン地域を表します。たとえば、Australia/North、UTC、Singapore などです。TZD は、夏時間情報を含むタイムゾーン地域の略称書式です。たとえば、米国 / 太平洋標準時は PST、米国 / 太平洋夏時間は PDT などです。TZR および TZD 書式要素の値を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューの TZNAME および TZABBREV 列に問合せを実行してください。

注意： 夏時間機能には、タイムゾーン地域名が必要です。地域名は、2 つのタイムゾーン・ファイルに格納されます。デフォルトのタイムゾーン・ファイルは、パフォーマンスを最大にするために一般的なタイムゾーンのみの小さなファイルです。タイムゾーンがデフォルトのファイルに存在しない場合は、環境変数 ORA_TZFILE を使用して完全な（大きい）ファイルへのパスを指定するまで、夏時間はサポートされません。環境変数 ORA_TZFILE の設定方法の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

注意： Oracle のタイムゾーン・データの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

参照：

- 書式要素の詳細は、2-64 ページの「[日付書式モデル](#)」を参照してください。
- 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

日時および期間の例

次の例では、日時および期間データ型の宣言方法を示します。

```
CREATE TABLE time_table (  
    start_time      TIMESTAMP,  
    duration_1      INTERVAL DAY (6) TO SECOND (5),  
    duration_2      INTERVAL YEAR TO MONTH);
```

start_time 列は、TIMESTAMP 型です。TIMESTAMP の暗黙的な小数部の精度は 6 です。

duration_1 列は、INTERVAL DAY TO SECOND 型です。DAY フィールドの最大桁数は 6 です。また、小数部の最大桁数は 5 です（その他のすべての日時フィールドの最大桁数は 2 です）。

duration_2 列は、INTERVAL YEAR TO MONTH 型です。各フィールド（YEAR および MONTH）の値の最大桁数は 2 です。

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 によって解析されません（異なるシステム間でデータを移動するときには変換されません）。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、音声、文書、またはバイナリ・データの配列の格納に使用できますが、解析方法は用途によって異なります。

注意： LONG RAW 列をバイナリ LOB (BLOB) へ変換することをお勧めします。LOB 列は、LONG 列ほど制限は多くありません。詳細は、6-169 ページの「[TO_LOB](#)」を参照してください。

RAW は、VARCHAR2 と同様に可変長データ型ですが、Oracle Net（ユーザー・セッションとインスタンスを接続します）およびインポート / エクスポート・ユーティリティは、RAW または LONG RAW データの転送時に文字変換を行いません。これに対し、Oracle Net およびインポート / エクスポートは、データベース・キャラクタ・セットとユーザー・セッションのキャラクタ・セット（ALTER SESSION 文の NLS_LANGUAGE パラメータで設定します）が異なる場合に、CHAR、VARCHAR2 および 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 データ間で、データを自動的に変換するときに、バイナリ・データは 16 進数で表されます。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」として表示または入力されます。

ラージ・オブジェクト (LOB) データ型

組込み LOB データ型の BLOB、CLOB、NCLOB (内部ファイルに格納) および BFILE (外部ファイルに格納) には、構造化されていない大きいデータ (text、image、video、spatial data など) を最大 4GB まで格納できます。

表を作成するときに、LOB 列または 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および記憶特性を指定できます。

LOB 列には、アウトライン 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 Oracle Call Interface (OCI) の操作は、これらのロケータを介して行われます。

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) の属性に指定できます。
- LOB ロケータは、表の列に格納されます。実際の LOB 値は、表の列に格納される場合と格納されない場合があります。BLOB、NCLOB および CLOB の値は、別々の表領域に格納されます。BFILE データは、サーバー上の外部ファイルに格納されます。
- LOB 列にアクセスしたときに戻されるのはロケータです。
- LOB の最大サイズは 4GB です。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りますが、4GB を超えることはありません。
- LOB では、効果的かつランダムなピース単位のデータ・アクセスおよび操作が可能です。
- 1 つの表内に 2 つ以上の LOB 列を定義できます。
- NCLOB の例外を除いて、1 つのオブジェクトに 1 つ以上の LOB 属性を定義できます。
- LOB バインド変数を宣言できます。
- LOB 列と LOB 属性を選択できます。
- 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、既存の行を更新できます (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られます。BFILE は、NULL に設定したり、別のファイルを指すように設定できます)。

- 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 属性を使用して更新できます。
- LOB 列または LOB 属性が含まれている行を削除できます（これによって LOB 値も削除されます）。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ません。

INSERT 文または UPDATE 文を発行するのみで、内部 LOB 列（データベースに格納されている LOB 列）の行にアクセスし、行を挿入できます。ただし、オブジェクト型の一部である LOB 属性にアクセスし、それを移入するには、EMPTY_CLOB ファンクションまたは EMPTY_BLOB ファンクションを使用して、LOB 属性を初期化してください。その後、DBMS_LOB パッケージまたはその他の適切なインタフェースを使用して、空の LOB 属性を選択し、それを移入できます。

LOB 列の制限 LOB 列には、次の制限事項があります。

- 分散された LOB はサポートされていません。そのため、問合せの SELECT または WHERE 句、または DBMS_LOB パッケージのファンクションで、リモート・ロケータを使用できません。

LOB では、次の構文はサポートされていません。

```
SELECT lobcol FROM table1@remote_site;
INSERT INTO lobtable SELECT type1.lobattr FROM table1@remote_site;
SELECT DBMS_LOB.getlength(lobcol) FROM table1@remote_site;
```

ただし、LOB を参照する別の問合せの一部では、リモート・ロケータを使用できます。リモート LOB 列では、次の構文がサポートされています。

```
CREATE TABLE t AS SELECT * FROM table1@remote_site;
INSERT INTO t SELECT * FROM table1@remote_site;
UPDATE t SET lobcol = (SELECT lobcol FROM table1@remote_site);
INSERT INTO table1@remote_site ...
UPDATE table1@remote_site ...
DELETE table1@remote_site ...
```

副問合せを含む最初の 3 つの文では、SELECT 構文のリストでスタンドアロン LOB 列のみが指定できます。SQL ファンクションまたは DBMS_LOB API は、LOB ではサポートされていません。たとえば、次の文はサポートされています。

```
CREATE TABLE AS SELECT clob_col FROM tab@dbs2;
```

ただし、次の文はサポートされていません。

```
CREATE TABLE AS SELECT dbms_lob.substr(clob_col) from tab@dbs2;
```

- クラスタには、キー列または非キー列として LOB を含めることができません。
- LOB の VARRAY は作成できません。

- 問合せの ORDER BY 句、GROUP BY 句または集計ファンクションで LOB 列を指定できません。
- SELECT ... DISTINCT 文、SELECT ... UNIQUE 文または結合で LOB 列を指定できません。ただし、列のオブジェクト型に MAP または ORDER ファンクションが定義されている場合は、SELECT ... DISTINCT 文、または UNION または MINUS 集合演算子を使用する問合せで、オブジェクト型列の LOB 属性を指定することはできます。
- ANALYZE ... COMPUTE 文または ANALYZE ... ESTIMATE 文で LOB 列を指定できません。
- LOB セグメントの初期 (INITIAL) エクステンツは、3 つ以上のデータベース・ブロックを含んでいる必要があります。
- UPDATE DML トリガーを作成する場合、LOB 列は UPDATE OF 句で指定できません。
- LOB 列は主キー列として指定できません。
- LOB 列は索引キーの一部として指定できません。ただし、ファンクション索引のファンクションまたはドメイン索引の索引タイプ指定では、LOB 列を指定できます。また、Oracle Text では、CLOB 列に索引を定義できます。
- INSERT または UPDATE 操作では、任意のサイズのデータを LOB 列にバインドできますが、オブジェクト型の LOB 属性にはデータをバインドできません。INSERT ... AS SELECT 操作では、4000 バイトのデータまで LOB 列にバインドできます。

参照： LOB の使用に関するその他のセマンティクスについては、このマニュアルの各 SQL 文の「キーワードとパラメータ」の項を参照してください。

- LONG と LOB の両方の列を持つ表では、1 つの SQL 文で両方に 4000 バイトより大きいデータをバインドすることはできません。ただし、いずれか片方にバインドすることは可能です。

注意：

- Oracle8i リリース 8.1.6 以上では、LOB の CACHE READS 設定をサポートしています。このような LOB を以前のリリースにダウングレードする場合は、Oracle は警告を表示し、LOB を CACHE READS から CACHE LOGGING に変換します。この後で、NOCACHE LOGGING または NOCACHE NOLOGGING のいずれかに LOB を変更できます。
 - DML トリガーを指定した表で、OCI 関数または DBMS_LOB ルーチンを使用して LOB 列の値またはオブジェクト型列の LOB 属性を変更する場合、Oracle は DML トリガーを起動しません。
-

参照：

- LONG から LOB への移行、CACHE READS の設定など、LOB の詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。
- 6-56 ページの「[EMPTY_BLOB](#)、[EMPTY_CLOB](#)」を参照してください。
- イメージ、オーディオ、ビデオおよび空間データのその他の格納方法については、2-38 ページの「[Oracle が提供する型](#)」を参照してください。

次の例では、サンプル表 pm.print_media を作成しています。この例では、ネストした表である textdoc_tab オブジェクト表が print_media 表に存在することを想定しています。

```
CREATE TABLE print_media
( product_id      NUMBER(6)
, ad_id          NUMBER(6)
, ad_composite    BLOB
, ad_sourcetext   CLOB
, ad_finalextn   CLOB
, ad_fltextn     NCLOB
, ad_textdocs_ntab textdoc_tab
, ad_photo       BLOB
, ad_graphic     BFILE
, ad_header      adheader_typ
, press_release  LONG
) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;
```

参照：

- これらのインタフェースおよび LOB の詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』および『Oracle Call Interface プログラマーズ・ガイド』を参照してください。
- LONG 列から LOB 列への変換については、11-2 ページの「[ALTER TABLE](#)」の「[modify_col_properties](#)」および 6-169 ページの「[TO_LOB](#)」を参照してください。

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。

BFILENAME ファンクションを使用すると、実表のデータに影響を与えずに BFILE のファイル名およびパスを変更できます。

参照： この組み込み SQL ファンクションの詳細は、6-22 ページの「[BFILENAME](#)」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関係なく、リカバリができません。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、ファイルが存在し、Oracle のプロセスがファイルに対するオペレーティング・システムの読み込み権限を持ってい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、サイズが大きいバイナリ・ファイルの読み込み専用のサポートが有効になります。この場合、ファイルを修正またはレプリケートすることはできません。Oracle では、ファイル・データにアクセスするための 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なインタフェースは、DBMS_LOB パッケージと OCI です。

参照：

- LOB の詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』および『Oracle Call Interface プログラマーズ・ガイド』を参照してください。
- 13-46 ページの「[CREATE DIRECTORY](#)」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・セットのセマンティクス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、DBMS_LOB パッケージまたは 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、コミットおよびロールバックできます。ただし、1 つのトランザクションの 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、シングルバイトおよびマルチバイト・キャラクタ・データを格納するために使用します。固定幅および可変幅のキャラクタ・セットがサポートされます。両方のキャラクタ・セットで CHAR データベース・キャラクタ・セットを使用します。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、DBMS_LOB パッケージまたは 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。CLOB 値の操作は、コミットおよびロールバックできます。ただし、1 つのトランザクションの 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 データ型は、各国語キャラクタ・セットを使用する Unicode データを格納するために使用します。固定幅および可変幅のキャラクタ・セットがサポートされます。NCLOB には、4GB までの文字テキスト・データを格納できます。

NCLOB では、トランザクションが完全にサポートされます。SQL、DBMS_LOB パッケージまたは 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、コミットおよびロールバックできます。ただし、1 つのトランザクションの 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。

参照： Unicode データ型のサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

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 を問い合わせることによって、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文字列で、文字列のデータ型は ROWID です。また、ROWID データ型を持つ実際の列を含む表やクラストを作成することもできます。Oracle では、こ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。

参照： ROWID 疑似列の詳細は、2-77 ページの「[疑似列](#)」を参照してください。

制限 ROWID

Oracle8 以降、Oracle SQL では、パーティション表、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**制限 ROWID**と呼ばれます。制限 ROWID の書式は次のとおりです。

block.row.file

それぞれの意味は、次のとおりです。

- *block* は、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 進文字列です。この文字列の長さは、オペレーティング・システムによって異なります。
- *row* は、データ・ブロック内の行を識別する 4 桁の 16 進文字列です。ブロック内の最初の行は 0 になります。
- *file* は、行を含むデータ・ファイルを識別する 16 進文字列です。最初のデータ・ファイルは 1 になります。この文字列の長さは、オペレーティング・システムによって異なります。

拡張 ROWID

ユーザー列に格納される**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、DBA_OBJECTS および 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、基本となる 64 という値で格納され、文字 A～Z、a～z、0～9、プラス記号 (+) およびスラッシュ (/) を含めることができます。拡張 ROWID は直接利用できません。拡張 ROWID の内容を解析するには、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・ファンクションを使用すると、制限 ROWID から直接利用できる情報、および拡張 ROWID に固有の情報が取り出され、提供されます。

参照： DBMS_ROWID パッケージで利用できるファンクション、およびこのファンクションの使用方法については、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

互換性と移行

制限形式の ROWID は、旧リリースとの互換性を保つために Oracle9i 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 が戻されます。

参照： 互換性と移行の問題については、『Oracle9i データベース移行ガイド』を参照してください。

UROWID 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。ただし、物理アドレスまたは永続アドレス以外のアドレス、または Oracle が生成したものでないアドレスがある行を持つ表もあります。たとえば、索引構成表の行のアドレスは索引リーフに格納され、移動できます。外部キー表の ROWID（たとえば、ゲートウェイを介してアクセスされる DB2）は、標準の Oracle ROWID ではありません。

Oracle では、ユニバーサル ROWID（**UROWID**）を使用して索引構成表および外部キー表のアドレスを格納します。索引構成表には、論理 UROWID があり、外部キー表には外部キー UROWID があります。いずれのタイプの UROWID も、（ヒープ構成表の物理 ROWID のように）ROWID 疑似列に格納されます。

Oracle は、表の主キーに基づいて論理 ROWID を作成します。論理 ROWID は、主キーが変更されないかぎり、変更されません。索引構成表の ROWID 疑似列は UROWID データ型です。この疑似列には、ヒープ構成の ROWID 疑似列と同様に（SELECT ROWID 文を使用して）アクセスできます。索引構成表の ROWID を格納する場合、表に UROWID 型の列を定義し、この列に ROWID 疑似列の値を取り込みます。

注意： ヒープ構成表には物理 ROWID があります。データ型が UROWID の列をヒープ構成表に指定しないことをお勧めします。

参照：

- ユニバーサル ROWID の詳細は、『Oracle9i データベース概要』を参照してください。
- データベースの行のアドレスの詳細は、2-31 ページの「[ROWID データ型](#)」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 文では、ANSI データ型、および IBM 社の製品 SQL/DS と DB2 のデータ型も使用できます。Oracle では、Oracle データ型の名前と異なる 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 2-6 および表 2-7 に示す変換に基づいて、Oracle のデータ型で列データを格納します。

表 2-6 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、固定小数点数のみを指定できます。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 2-7 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、固定小数点数のみを指定できます。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、バイナリ精度 *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 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。次のデータ型を持つ列は定義しないでください。

- GRAPHIC
- LONG VARGRAPHIC
- VARGRAPHIC
- TIME

データ型が TIME であるデータは、Oracle の DATE データとしても表現できます。

ユーザー定義型

ユーザー定義のデータ型は、Oracle 組込みデータ型とその他のユーザー定義のデータ型を、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します。

次の項で、ユーザー定義型の様々なカテゴリを説明します。

参照：

- Oracle 組み込みデータ型の詳細は、『Oracle9i データベース概要』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、16-3 ページの「[CREATE TYPE](#)」および 16-25 ページの「[CREATE TYPE BODY](#)」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、『Oracle9i アプリケーション開発者ガイド - 基礎編』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、次の3種類のコンポーネントを持つスキーマ・オブジェクトです。

- 名前 - 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 - 組み込み型またはその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 - PL/SQL で記述され、データベースに格納されるファンクションまたはプロシージャ、あるいは C や Java などの言語で記述され、外部に格納されるファンクション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実装します。

REF

オブジェクト識別子 (OID) を使用することによって、オブジェクトを一意に識別し、他のオブジェクトまたはリレーショナル表からそのオブジェクトを参照できます。REF と呼ばれるデータ型のカテゴリが、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）状態であるといえます。DANGLING 状態の REF は、NULL である REF と異なります。REF が DANGLING 状態であるかどうかを確認するには、述語 IS [NOT] DANGLING を使用します。たとえば、列型が customer_typ（属性 cust_email を持つ）を指している REF である customer_ref 列があるとします。この customer_ref 列を含むオブジェクト・ビュー oc_orders（サンプル・スキーマ oe 内）は、次のように指定します。

```
SELECT o.customer_ref.cust_email
FROM oc_orders o
WHERE o.customer_ref 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であるため、**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。

- リレーショナル表の列のデータ型
- オブジェクト型属性
- PL/SQL の変数、パラメータ、またはファンクションの戻り型

通常、Oracle は、1 つの配列オブジェクトをインライン形式でその行の他のデータと同じ表領域に、またはアウトライン形式で LOB に格納します。この形式は配列オブジェクトのサイズによって決まります。ただし、VARRAY に個別の記憶特性を指定する場合、Oracle はこれをサイズに関係なくアウトライン形式で格納します。

参照： 詳細は、15-38 ページの「[CREATE TABLE](#)」の「[varray_col_properties](#)」を参照してください。

ネストした表

ネストした表は、順序付けられていない要素の集合を表現します。その要素は、組込み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。

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

- リレーショナル表の列
- オブジェクト型属性
- PL/SQL の変数、パラメータ、およびファンクションの戻り値

ネストした表が、リレーショナル表内の列型として使用される場合、またはオブジェクト表の基礎となるオブジェクト型の属性として使用される場合、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リレーショナル表またはオブジェクト表に対応付けます。

Oracle が提供する型

オラクル社は、組込み型または ANSI がサポートする型が不十分な場合に、新しい型を定義するための SQL に基づくインタフェースを提供します。これらの型の動作は、C/C++、Java または PL/ SQL で実装されます。Oracle は、入出力、異機種間でのクライアント側の新しいデータ型へのアクセス、およびアプリケーションとデータベース間のデータ転送のための最適化に必要な下位レベルのインフラストラクチャ・サービスを自動的に提供します。

これらのインタフェースは、ユーザー定義（またはオブジェクト）型の作成に使用できます。また、Oracle が有効なデータ型を作成するときに使用します。このようなデータ型のいくつかはサーバーで提供され、横方向の幅広いアプリケーション領域（任意型など）および縦方向の固有アプリケーション領域（空間型など）の両方に役立ちます。

Oracle が提供する型については、次の項で説明します。また、それらの型の実装および使用に関するドキュメントの参照先も示します。

- [任意型](#)
- [XML 型](#)
- [空間型](#)
- [メディア型](#)

任意型

任意型は、実際の型が不明なプロシージャ・パラメータおよび表の列の柔軟性が高いモデリングを提供します。これらのデータ型によって、型の記述、データ・インスタンスおよびその他の SQL 型の一連のデータ・インスタンスを動的にカプセル化し、アクセスできます。これらの型を構成およびアクセスするには、OCI および PL/SQL インタフェースを使用します。

SYS.ANYTYPE

この型には、任意の名前のある SQL 型または名前のない一時型の型の記述を含めることができます。

SYS.ANYDATA

この型には、指定した型のインスタンスをデータおよび型の記述とともに含めることができます。ANYDATA は、表の列のデータ型として使用できます。また、このデータ型によって、単一列に異機種間の値を格納できます。ユーザー定義型と同様に、SQL 組込み型の値も格納できます。

SYS.ANYDATASET

この型には、指定した型の記述およびその型の一連のデータ・インスタンスを含めることができます。ANYDATASET は、柔軟性が必要なプロシージャ・パラメータのデータ型として使用できます。ユーザー定義型と同様に、SQL 組込み型のデータ・インスタンスの値も格納できます。

参照： これらの型の実装および使用のガイドラインは、『Oracle Call Interface プログラマーズ・ガイド』、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

XML 型

eXtensible Markup Language (XML) は、World Wide Web Consortium (W3C) によって開発された、構造化されたデータおよび構造化されていないデータを Web で表示するための標準書式です。Universal Resource Identifiers (URI) は、Web ページなどの Web 上のリソースを識別します。Oracle は、データベース自体に格納されるデータにアクセスするために DBUriRef と呼ばれる URI のクラスと同様に、XML および URI データを処理するための型を提供します。また、データベースから外部および内部 URI に格納およびアクセスする新しい型のセットを提供します。

XMLType

Oracle が提供するこの型は、データベースでの XML データの格納および問合せに使用します。XMLType は、XPath 式を使用した XML データへのアクセス、抽出および問合せに使用するメンバー・ファンクションを持ちます。XPath は、XML 文書をトラバースするために W3C によって開発された別の規格です。Oracle の XMLType ファンクションは、W3C の多数の XPath 式をサポートします。また、Oracle は、既存のリレーショナルまたはオブジェクト・リレーショナル・データから XMLType 値を作成するための SQL ファンクションおよび PL/SQL パッケージのセットを提供します。

XMLType はシステム定義型であるため、ファンクションの引数として、あるいは表またはビューの列のデータ型として使用できます。また、XMLType の表およびビューも作成できます。表に XMLType 列を作成する場合は、XML データを CLOB 列またはオブジェクトと関連付けて格納するように選択できます。

また、スキーマを登録し (DBMS_XMLSCHEMA パッケージを使用)、登録したスキーマに適合する表または列を作成できます。この場合、デフォルトでは、XML データは基礎となるオブジェクト・リレーショナル列に格納されますが、スキーマ・ベースのデータであっても、CLOB 列に格納するように指定できます。

XMLType 列の問合せおよび DML は、格納メカニズムとは関係なく、同様に動作します。

URI データ型

Oracle は、継承階層で関連する URI 型のファミリー (URIType、DBURIType、XDBURIType および HTTPURIType) を提供します。URIType はオブジェクト型であり、その他は URIType のサブタイプです。URIType はスーパータイプであるため、この型の列を作成し、DBURIType または HTTPURIType 型のインスタンスをこの列に格納できます。

HTTPURIType HTTPURIType を使用すると、外部の Web ページまたはファイルの URL を格納できます。Oracle は、Hypertext Transfer Protocol (HTTP プロトコル) を使用して、これらのファイルにアクセスします。

XDBURIType XDBURIType を使用すると、表の任意の URIType 列に埋込み可能な URI として、ドキュメントを XML データベース階層に展開できます。XDBURIType は、URL で構成されています。URL は、参照先の XML 文書の階層名と、XPath 構文を表すオプションのフラグメントで構成されています。フラグメントと URL 部分は、シャープ記号 (#) で区切ります。次に、XDBURIType の例を示します。

```
/home/oe/doc1.xml
/home/oe/doc1.xml#/orders/order_item
```

DBURIType DBURIType を使用すると、データベース内のデータを参照する DBURIType を格納できます。この場合、データベースの内部または外部に格納されたデータを参照し、常にデータにアクセスできます。

DBURIType は、データベース内のデータを参照するために、XPath のような表現を使用します。データベースを XML ツリーに想定すると、表、行および列が XML 文書の要素です。たとえば、サンプルの人材のユーザー hr は次のように XML ツリーで表します。

```
<HR>
  <EMPLOYEES>
    <ROW>
      <EMPLOYEE_ID>205</EMPLOYEE_ID>
      <LAST_NAME>Higgins</LAST_NAME>
      <SALARY>12000</SALARY>
      .. <!-- other columns -->
    </ROW>
    ... <!-- other rows -->
  </EMPLOYEES>
  <!-- other tables...-->
</HR>
<!-- other user schemas on which you have some privilege on...-->
```

DBURIType は、この仮想 XML 文書の単純な XPath 式です。したがって、従業員番号 205 の従業員の SALARY 値を EMPLOYEES 表で参照するには、DBURIType を次のように記述します。

```
/HR/EMPLOYEES/ROW[EMPLOYEE_ID=205]/SALARY
```

このモデルを使用すると、CLOB 列またはその他の列に格納されたデータを参照し、それらのデータを外部の URL として外部の世界へ公開できます。

URIFactory パッケージ

Oracle は、URIType の様々なサブタイプのインスタンスを作成し、返すことができる URIFactory パッケージも提供します。このパッケージは、URL 文字列を分析し、URL の種類 (HTTP、DBUri など) を識別し、サブタイプのインスタンスを作成します。DBUri インスタンスを作成する場合は、URL の先頭に接尾辞 /oradb を付ける必要があります。たとえば、URIFactory.getURI('/oradb/HR/EMPLOYEES') により DBUriType インスタンスが作成され、URIFactory.getUri('/sys/schema') により XDBUriType インスタンスが作成されます。

参照：

- オブジェクト型および型の継承については、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。
- 提供される型およびその実装については、『Oracle9i XML Developer's Kit ガイド-XDK』を参照してください。
- Oracle Advanced Queuing での XMLType の使用については、『Oracle9i アプリケーション開発者ガイド - アドバンスド・キューイング』を参照してください。

空間型

Oracle Spatial のオブジェクト・リレーショナル実装は、一連のオブジェクト・データ型、索引メソッド型およびこれらの型の演算子によって構成されます。

MDSYS.SDO_GEOMETRY

空間オブジェクトのジオメトリの記述は、ユーザー定義の表の単一行およびオブジェクト型 SDO_GEOMETRY の単一列に格納されます。SDO_GEOMETRY 型の列を含むすべての表は、表に対して一意の主キーを定義する別の列を持つ必要があります。このような表は、ジオメトリ表と呼ばれる場合があります。

参照： この型の実装および使用のガイドラインについては、『Oracle Spatial 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。

メディア型

Oracle *interMedia* は、マルチメディア・データを記述するために、Java または C++ クラスに類似したオブジェクト型を使用します。これらのオブジェクト型のインスタンスは、メタデータおよびメディア・データを含む属性とメソッドで構成されます。Oracle *interMedia* 型は次のとおりです。

ORDSYS.ORDAudio

ORDAUDIO オブジェクト型は、オーディオ・データの格納および管理をサポートします。

ORDSYS.ORDImage

ORDIMAGE オブジェクト型は、イメージ・データの格納および管理をサポートします。

ORDSYS.ORDImageSignature

ORDSYS.ORDImageSignature オブジェクト型は、イメージ・データの色、テクスチャ、および形の情報の簡潔な表現をサポートします。

ORDSYS.ORDVideo

ORDVIDEO オブジェクト型は、ビデオ・データの格納および管理をサポートします。

ORDSYS.ORDDoc

ORDDOC オブジェクト型は、オーディオ、イメージ、ビデオ・データなどすべての種類のメディア・データの格納および管理をサポートします。この型は、すべてのメディアを単一行に格納する場合に使用します。

参照： これらの型の実装および使用のガイドラインは、『Oracle *interMedia* 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。

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

大きい値は小さい値よりも大きいとみなされます。すべての負の数は、0（ゼロ）および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-1997（1997年3月29日）に相当する日付は 05-JAN-1998（1998年1月5日）に相当する日付よりも小さく、05-JAN-1998 1:35pm（1998年1月5日午後1時35分）に相当する日付は 05-JAN-1998 10:09am（1998年1月5日午前10時9分）に相当する日付よりも大きいとみなされます。

文字列値

文字値は、次のいずれかの比較セマンティクスを使用して比較されます。

- 空白埋め比較セマンティクス
- 非空白埋め比較セマンティクス

ここでは、これらの比較セマンティクスについて説明します。

空白埋め比較セマンティクス 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、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。最初に異なる文字の位置で、大きい方の文字を持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つの値は等しいとみなされます。この規則では、2つの値の後続空白数のみが異なる場合、その2つの値は等しいとみなされます。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 ファンクションの戻り値の場合のみ空白埋め比較セマンティクスを使用します。

非空白埋め比較セマンティクス 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。最初に異なる文字の位置で、大きい方の文字を持つ値の方が大きいとみなされます。長さが異なる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つの値は等しいとみなされます。Oracle では、比較する片方または両方の値が VARCHAR2 データ型または NVARCHAR2 データ型の場合、非空白埋め比較セマンティクスを使用します。

これらの異なる比較セマンティクスを使用して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。次の表に、それぞれの比較セマンティクスを使用して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。

空白埋め比較	非空白埋め比較
'ac' > 'ab'	'ac' > 'ab'
'ab' > 'a '	'ab' > 'a '
'ab' > 'a'	'ab' > 'a'
'ab' = 'ab'	'ab' = 'ab'
'a ' = 'a'	'a ' > 'a'

単一文字

Oracle は、データベース・キャラクタ・セットで指定さ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、ほぼすべてのキャラクタ・セットでいえることです。

次に、一般的なキャラクタ・セットを示します。

- 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)
- 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 2-8 と表 2-9 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 2-8 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

表 2-8 ASCII キャラクタ・セット（続き）

記号	10 進値	記号	10 進値
.	46	{	123
/	47		124
0-9	48-57	}	125
:	58	~	126

表 2-9 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

オブジェクト値

オブジェクト値は、MAP と ORDER の 2 つの比較関クションのいずれかを使用して比較されます。どちらの関クションでもオブジェクト型インスタンスは比較されますが、両者は別のものです。これらの関クションは、オブジェクト型の一部として指定される必要があります。

参照： MAP メソッドと ORDER メソッド、およびこれらのメソッドが戻す値の詳細は、16-3 ページの「[CREATE TYPE](#)」および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

VARRAY とネストした表

Oracle9i では、VARRAY とネストした表を比較することはできません。

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 に 5 を掛けた値に 'JAMES' を加えることはできません。ただし、Oracle では値のあるデータ型から別のデータ型へ変換する場合の、暗黙的な変換と明示的な変換をサポートしています。

暗黙的なデータ変換と明示的なデータ変換

次の理由から、暗黙的な変換または自動変換ではなく、明示的な変換を指定することをお勧めします。

- 明示的なデータ型変換関クションを使用すると、SQL 文がわかりやすくなります。
- 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ります。
- 暗黙的な変換はその変換が行われるコンテキストに依存し、どんな場合でも同様に機能するとはかぎりません。たとえば、日付値から VARCHAR2 型へ値を暗黙的に変換すると、NLS_DATE_FORMAT パラメータに指定されている値によって、予期しない年が戻される場合があります。
- 暗黙的な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ります。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ります。

暗黙的なデータ変換

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。表 2-10 に、Oracle の暗黙的な変換のマトリックスについて示します。この表は、変換の方向または変換されるコンテキストにかかわらず、すべての可能な変換を示します。詳細は、表の後の説明を参照してください。

表 2-10 暗黙的な型変換のマトリックス

	CHAR	VARCHAR2	DATE	DATETIME/ INTERVAL	LONG	NUMBER	RAW	ROWID	CLOB	BLOB	NCHAR	NVARCHAR	NCLOB
CHAR	—	X	X	X	X	X	X	—	X	—	X	X	—
VARCHAR2	X	—	X	X	X	X	X	X	X	—	X	X	—
DATE	X	X	—	—	—	—	—	—	—	—	X	X	—
DATETIME/ INTERVAL	X	X	—	—	X	—	—	—	—	—	X	X	—
LONG	X	X	—	X	—	—	X	—	X	—	X	X	X
NUMBER	X	X	—	—	—	—	—	—	—	—	X	X	—
RAW	X	X	—	—	X	—	—	—	—	X	X	X	—
ROWID	X	X	—	—	—	—	—	—	—	—	X	X	—
CLOB	X	X	—	—	X	—	—	—	—	—	—	—	—
BLOB	—	—	—	—	—	—	X	—	—	—	—	—	—
NCHAR	X	X	X	X	X	X	X	X	—	—	—	X	X
NVARCHAR2	X	X	X	X	X	X	X	X	—	—	X	—	X
NCLOB	—	—	—	—	X	—	—	—	—	—	X	X	—

次に示す規則に従って、Oracle は、暗黙的なデータ型変換を実行する方向を確立します。

- INSERT および UPDATE 操作中に、Oracle は変更する列のデータ型に値を変換します。
- SELECT FROM 操作中に、Oracle は列からターゲット変数の型にデータを変換します。
- NUMBER の値と文字の値を比較する場合、Oracle は文字データを NUMBER に変換します。
- DATE の値と文字の値を比較する場合、Oracle は文字データを DATE に変換します。
- SQL ファンクション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。
- 割当てを実行する場合、Oracle は等号 (=) の右側の値を左側の割当てターゲットのデータ型に変換します。
- 連結中に、Oracle は非文字データ型を CHAR または NCHAR に変換します。
- 演算の処理および文字データ型と非文字データ型の比較中に、Oracle はすべての文字データ型を数値、日付または ROWID のいずれかの適切なデータ型に変換します。
CHAR/VARCHAR2 と NCHAR/NVARCHAR2 の演算処理では、Oracle は数値に変換します。
- CHAR/VARCHAR2 と NCHAR/NVARCHAR2 の比較では、異なるキャラクタ・セットが必要な場合があります。このような場合のデフォルトの変換の方向は、データベース・キャラクタ・セットから各国語キャラクタ・セットです。表 2-11 に、異なるキャラクタ・タイプ間での暗黙的な変換の方向を示します。
- ほとんどの SQL 文字ファンクションは、CLOB をパラメータとして指定できます。また、Oracle は CLOB と CHAR 型間で暗黙的な変換を実行します。このため、CLOB を使用できないファンクションは、暗黙的な変換を使用して CLOB を受け入れます。このような場合、Oracle はファンクションが起動される前に CLOB を CHAR または VARCHAR2 に変換します。CLOB が 4000 バイトより大きい場合、Oracle は最初の 4000 バイトのみを CHAR に変換します。

表 2-11 異なるキャラクタ・タイプの変換方向

	CHAR へ	VARCHAR2 へ	NCHAR へ	NVARCHAR2 へ
CHAR から	--	VARCHAR2	NCHAR	NVARCHAR2
VARCHAR2 から	VARCHAR2	--	NVARCHAR2	NVARCHAR2
NCHAR から	NCHAR	NCHAR	--	NVARCHAR2
NVARCHAR2 から	NVARCHAR2	NVARCHAR2	NVARCHAR2	--

暗黙的なデータ変換の例

テキスト・リテラルの例 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 salary + '10'
FROM employees;
```

文字値および数値の例 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、'200' が暗黙的に 200 に変換されます。

```
SELECT last_name
FROM employees
WHERE employee_id = '200';
```

日付の例 次の文では、Oracle がデフォルトの日付書式 'DD-MON-YY' を使用して、'03-MAR-97' を DATE 値に暗黙的に変換します。

```
SELECT last_name
FROM employees
WHERE hire_date = '03-MAR-97';
```

ROWID の例 次の文では、Oracle がテキスト・リテラル 'AAAFYmAAFAAAAFGAH' を ROWID 値に暗黙的に変換します（ROWID はデータベース内で一意のため、この例を使用するには、データベースの実際の ROWID を知る必要があります）。

```
SELECT last_name
FROM employees
WHERE ROWID = 'AAAFd1AAFAAAABSAH';
```

明示的なデータ変換

また、SQL 変換ファンクションを使用すると、データ型の変換を明示的に指定できます。次の表に、値をあるデータ型から別のデータ型に明示的に変換する SQL ファンクションを示します。

表 2-12 明示的な型の変換

	CHAR、 VARCHAR2、 NCHAR、 NVARCHAR2 へ	NUMBER へ	Datetime/ Interval へ	RAW へ	ROWID へ	LONG、 LONG RAW へ	CLOB、 NCLOB、 BLOB へ
CHAR、 VARCHAR2、 NCHAR、 NVARCHAR2 から	TO_CHAR (文字) TO_NCHAR (文字)	TO_NUMBER	TO_DATE TO_TIMESTAMP TO_TIMESTAMP_TZ TO_YMINTERVAL TO_DSINTERVAL	HEXTORAW	CHARTOROWID	—	TO_CLOB TO_NCLOB
NUMBER から	TO_CHAR (数値) TO_NCHAR (数値)	—	TO_DATE NUMTOYMINTERVAL NUMTODSINTERVAL	—	—	—	—
Datetime/ Interval から	TO_CHAR (日付) TO_NCHAR (日時)	—	—	—	—	—	—
RAW から	RAWTOHEX RAWTONHEX	—	—	—	—	—	TO_BLOB
ROWID から	ROWIDTOCHAR	—	—	—	—	—	—
LONG/LONG RAW から	—	—	—	—	—	—	TO_LOB
CLOB、 NCLOB、 BLOB から	TO_CHAR TO_NCHAR	—	—	—	—	—	TO_CLOB TO_NCLOB

注意： LONG および LONG RAW の値を指定すると、Oracle で暗黙的なデータ型変換を行うことができません。たとえば、ファンクション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、2-14 ページの「[LONG データ型](#)」を参照してください。

参照： すべての明示的な変換ファンクションの詳細は、6-6 ページの「[変換ファンクション](#)」を参照してください。

リテラル

リテラルと**定数値**という用語の意味は同じで、固定データ値のことです。たとえば、`'JACK'`、`'BLUE ISLAND'` および `'101'` はすべて文字リテラルです。文字リテラルは、一重引用符で囲みます。一重引用符を付けることで、**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。

この項では、次の内容を説明します。

- テキスト・リテラル
- 整数リテラル
- 数値リテラル
- 期間リテラル

多くの SQL 文とファンクションでは、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。

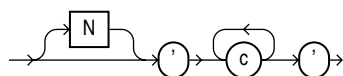
Datetime（日時）または **Interval**（期間）のデータ型をリテラルとして指定する場合は、データ型に含まれているオプションの精度を考慮する必要があります。**Datetime**（日時）および **Interval**（期間）のデータ型をリテラルとして指定する場合の例は、2-2 ページの「[データ型](#)」の関連する項を参照してください。

テキスト・リテラル

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、条件、SQL ファンクションおよび SQL 文に示されている `'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）の構文は次のとおりです。

text::=



それぞれの意味は、次のとおりです。

- **N** は、各国語キャラクタ・セットを使用してリテラルを指定します。この表記法を使用して入力したテキストは、使用時に **Oracle** によって各国語キャラクタ・セットに変換されます。
- **c** は、一重引用符 (') を除く、データベース・キャラクタ・セットの任意の要素です。

- ' ' は、テキスト・リテラルの始まりと終わりを示す 2 つの一重引用符です。リテラル内で一重引用符を表すには、一重引用符を 2 つ使用します。

テキスト・リテラルは、一重引用符で囲みます。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 データ型の両方のプロパティを持ちます。

- 式と条件の中のテキスト・リテラルは、Oracle によって CHAR データ型として扱われ、空白埋め比較セマンティクスで比較されます。
- テキスト・リテラルの最大長は 4000 バイトです。

有効なテキスト・リテラルの例を次に示します。

```
'Hello'  
'ORACLE.dbs'  
'Jackie''s raincoat'  
'09-MAR-98'  
N'nchar literal'
```

参照：

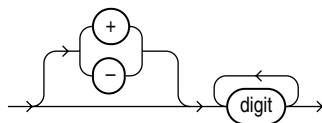
- `expr` の構文の説明は、4-2 ページの「[SQL 式](#)」を参照してください。
- 2-43 ページの「[空白埋め比較セマンティクス](#)」を参照してください。

整数リテラル

このマニュアルの他の箇所で、式、条件、SQL ファンクションおよび SQL 文に示されている *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::=



digit は、0、1、2、3、4、5、6、7、8、9 のいずれかです。

整数は最大 38 桁の精度を記憶できます。

有効な **integer**（整数）の例を次に示します。

7
+255

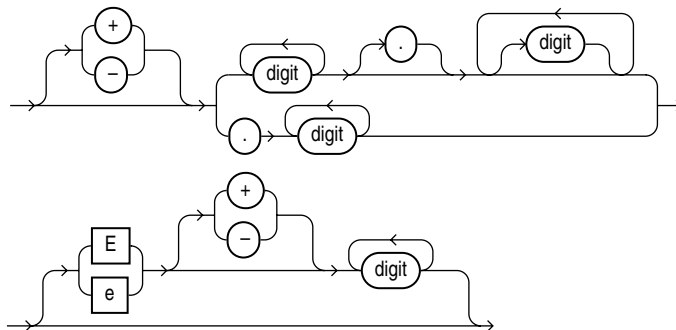
参照： *expr* の構文の説明は、4-2 ページの「SQL 式」を参照してください。

数値リテラル

このマニュアルの他の箇所で、式、条件、SQL ファンクションおよび 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::=



それぞれの意味は、次のとおりです。

- **+** または **-** は、正の値または負の値を示します。符号を指定しない場合、デフォルトは正の値です。
- **digit** は、0、1、2、3、4、5、6、7、8、9 のいずれかです。
- **e** または **E** は、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示します。指数は -130 ～ 125 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** を使用してピリオド (.) 以外的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'
```

参照： 10-2 ページの「[ALTER SESSION](#)」および『Oracle9i データベース・リファレンス』を参照してください。

有効な *number* (数) の例を次に示します。

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

参照： *expr* の構文の説明は、4-2 ページの「[SQL 式](#)」を参照してください。

期間リテラル

期間リテラルは期間を指定します。年および月、または日付、時間、分および秒の違いを指定できます。Oracle は、`YEAR TO MONTH` および `DAY TO SECOND` の 2 種類の期間リテラルをサポートします。各リテラルは先行フィールドを含み、後続フィールドを含むこともあります。先行フィールドは、計測する日付または時刻の基本単位を定義します。後続フィールドは、考慮する基本単位の最小増分値を定義します。たとえば、`YEAR TO MONTH` 期間では、最も近い月に対する年との期間が考慮されます。`DAY TO MINUTE` 期間では、最も近い分に対する日との期間が考慮されます。

数値形式の日付データがある場合、`NUMTOYMINTERVAL` または `NUMTODSINTERVAL` 変換関数を使用して、数値データを期間リテラルへ変換できます。

期間リテラルは、主に分析関数とともに使用します。

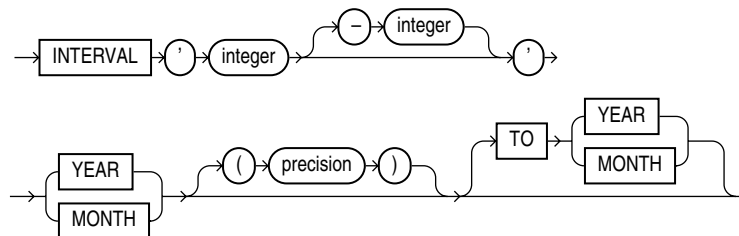
参照：

- 6-9 ページの「[分析関数](#)」および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- 6-105 ページの「[NUMTODSINTERVAL](#)」および 6-106 ページの「[NUMTOYMINTERVAL](#)」を参照してください。

INTERVAL YEAR TO MONTH

次の構文を使用して、YEAR TO MONTH 期間リテラルを指定します。

interval_year_to_month::=



それぞれの意味は、次のとおりです。

- `'integer [-integer]'` には、リテラルの先行フィールドおよびオプションの後続フィールドの整数値を指定します。先行フィールドが `YEAR` で、後続フィールドが `MONTH` の場合、`MONTH` フィールドの整数値の範囲は 0 ～ 11 です。
- `precision` は、先行フィールドの桁数です。先行フィールド精度の有効範囲は 0 ～ 9 で、デフォルトは 2 です。

先行フィールドの制限事項： 先行フィールドの桁数は、後続フィールドの桁数より多く設定する必要があります。たとえば、`INTERVAL '0-1' MONTH TO YEAR` は無効です。

次の `INTERVAL YEAR TO MONTH` リテラルは、`interval` が 123 年 2 か月であることを示しています。

```
INTERVAL '123-2' YEAR(3) TO MONTH
```

このリテラルの他の書式の例を次に示します。省略バージョンも含まれます。

期間リテラルの書式	説明
<code>INTERVAL '123-2' YEAR(3) TO MONTH</code>	123 年 2 か月の <code>interval</code> を示します。先行フィールド精度がデフォルトの 2 桁より大きい場合、その精度を指定してください。
<code>INTERVAL '123' YEAR(3)</code>	123 年 0 か月の <code>interval</code> を示します。
<code>INTERVAL '300' MONTH(3)</code>	300 か月の <code>interval</code> を示します。
<code>INTERVAL '4' YEAR</code>	<code>INTERVAL '4-0' YEAR TO MONTH</code> へマップし、4 年を示します。

期間リテラルの書式	説明
INTERVAL '5' MONTH	INTERVAL '4-2' YEAR TO MONTH へマップし、50 か月または 4 年 2 か月を示します。
INTERVAL '123' YEAR	デフォルト精度は 2 ですが、123 が 3 桁のため、エラーを戻します。

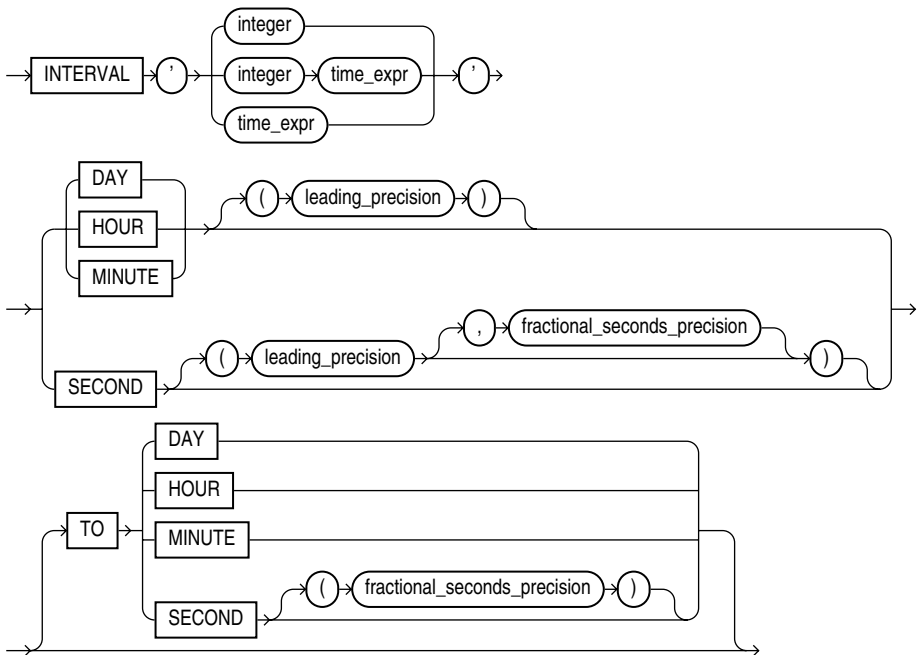
ある INTERVAL YEAR TO MONTH リテラルを別のリテラルに加算または減算して、新しい INTERVAL YEAR TO MONTH リテラルを作成できます。たとえば、次のようにします。

INTERVAL '5-3' YEAR TO MONTH + INTERVAL'20' MONTH =
INTERVAL '6-11' YEAR TO MONTH

INTERVAL DAY TO SECOND

次の構文を使用して、DAY TO SECOND 期間リテラルを指定します。

interval_day_to_second::=



それぞれの意味は、次のとおりです。

- `integer` は日数を指定します。ここで指定した値の桁数が先行精度で指定した桁数より大きい場合、Oracle はエラーを戻します。
- `time_expr` には、HH[:MI[:SS[.n]]]、MI[:SS[.n]] または SS[.n] 書式で時間を指定します (ここで、`n` は秒の小数部を指定します)。 `n` の桁数が、`fractional_seconds_precision` で指定した数より多い場合、`n` は `fractional_seconds_precision` 値で指定した数に丸められます。先行フィールドが DAY の場合にのみ、1 桁の整数および 1 つの空白が後に続く `time_expr` を指定できます。
- `leading_precision` は、先行フィールドの桁数です。有効範囲は 0 ～ 9 です。デフォルトは 2 です。
- `fractional_seconds_precision` は、SECOND 日時フィールドの小数部の桁数です。有効範囲は 1 ～ 9 です。デフォルトは 6 です。

先行フィールドの制限事項： 先行フィールドの桁数は、後続フィールドの桁数より多く設定する必要があります。たとえば、INTERVAL MINUTE TO DAY は無効です。このため、SECOND が先行フィールドの場合、期間リテラルは後続フィールドを持つことができません。

後続フィールドの有効範囲は次のとおりです。

- HOUR: 0 ～ 23
- MINUTE: 0 ～ 59
- SECOND: 0 ～ 59.999999999

様々な INTERVAL DAY TO SECOND リテラルの書式の例を次に示します。省略バージョンも含まれます。

期間リテラルの書式	説明
INTERVAL '4 5:12:10.222' DAY TO SECOND (3)	4 日 5 時間 12 分 10.222 秒を示します。
INTERVAL '4 5:12' DAY TO MINUTE	4 日 5 時間 12 分を示します。
INTERVAL '400 5' DAY (3) TO HOUR	400 日 5 時間を示します。
INTERVAL '400' DAY (3)	400 日を示します。
INTERVAL '11:12:10.2222222' HOUR TO SECOND (7)	11 時間 12 分 10.2222222 秒を示します。
INTERVAL '11:20' HOUR TO MINUTE	11 時間 20 分を示します。
INTERVAL '10' HOUR	10 時間を示します。

期間リテラルの書式	説明
INTERVAL '10:22' MINUTE TO SECOND	10 分 22 秒を示します。
INTERVAL '10' MINUTE	10 分を示します。
INTERVAL '4' DAY	4 日を示します。
INTERVAL '25' HOUR	25 時間を示します。
INTERVAL '40' MINUTE	40 分を示します。
INTERVAL '120' HOUR(3)	120 時間を示します。
INTERVAL '30.12345' SECOND(2,4)	30.1235 秒を示します。精度は 4 のため、秒の小数部 '12345' は '1235' に丸められます。

DAY TO SECOND 期間リテラルを別の DAY TO SECOND 期間リテラルに加算または減算できます。次に例を示します。

INTERVAL'20' DAY - INTERVAL'240' HOUR = INTERVAL'10-0' DAY TO SECOND

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。文字列を日付または数値に変換する場合、書式モデルは文字列の変換方法を問い合わせます。SQL 文では、書式モデルは、次の目的で、TO_CHAR ファンクションや TO_DATE ファンクション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します。
- 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します。

注意： 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。

次に例を示します。

- '17:45:29' の日付書式モデルは、'HH24:MI:SS' です。
- '11-Nov-1999' の日付書式モデルは、'DD-Mon-YYYY' です。
- '\$2,304.25' の数値書式モデルは、'\$9,999.99' です。

日付および数値書式モデルの要素のリストは、2-61 ページの表 2-13「数値書式の要素」および 2-65 ページの表 2-15「日時書式要素」を参照してください。

いくつかの書式の値は、初期化パラメータの値によって決まります。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。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` 文を使用します。

参照：

- これらのパラメータの詳細は、『Oracle9i データベース・リファレンス』および『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。
- これらのパラメータの値を変更する場合の詳細は、10-2 ページの「[ALTER SESSION](#)」を参照してください。

戻り値の書式例 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。

次の文は、部門 80 の従業員の給与を選択し、`TO_CHAR` ファンクションを使用して、その給与を数値書式モデル '\$99,990.99' で指定した書式の文字値に変換します。

```
SELECT last_name employee, TO_CHAR(salary, '$99,990.99')
      FROM employees
      WHERE department_id = 80;
```

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 桁を持つ給与を戻します。

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` ファンクションを使用して、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 last_name employee,
      TO_CHAR(hire_date, 'fmMonth DD, YYYY') hiredate
      FROM employees
      WHERE department_id = 20;
```

Oracle はこの書式モデルによって、2 桁の日、世紀も含めた 4 桁の年で示された入社日付（'fm' で指定）を空白で埋めないで戻します。

参照： fm 書式要素の説明については、2-71 ページの「[書式モデルの修飾子](#)」を参照してください。

正しい書式モデルの付与例 列の値を挿入または更新する場合、指定する値のデータ型は列のデータ型と一致している必要があります。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。

たとえば、DATE 列に挿入する値は、DATE データ型の値か、またはデフォルト日付書式の文字列値である必要があります (Oracle は、暗黙的にデフォルト日付書式の文字列を DATE データ型に変換します)。値が別の書式で与えられる場合、TO_DATE ファンクションを使用して値を DATE データ型に変換する必要があります。また、文字列の書式を指定する場合にも、書式モデルを使用する必要があります。

次の文は、TO_DATE ファンクションを使用して Hunold の入社日を更新します。文字列 '1998 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 employees
  SET hire_date = TO_DATE('1998 05 20','YYYY MM DD')
  WHERE last_name = 'Hunold';
```

この項の後半では、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](#)

参照： 詳細は、6-162 ページの「[TO_CHAR \(日時\)](#)」、6-164 ページの「[TO_CHAR \(数値\)](#)」および 6-166 ページの「[TO_DATE](#)」を参照してください。

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 ファンクション
- 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 ファンクション

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多い場合、シャープ記号 (#) 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 (-~) が戻されます。通常、このイベントは、TO_CHAR ファンクション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。

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で構成されます。表 2-13 に数値書式モデルの要素を示します。表 2-14 に例を示します。

数値書式モデルに書式要素 MI、S または PR が指定されないかぎり、負の戻り値の先頭には自動的に負の符号が付けられ、正の戻り値の先頭には空白が付けられます。

表 2-13 数値書式の要素

要素	例	説明
, (カンマ)	9,999	指定した位置にカンマを戻します。1 つの数値書式モデルに複数のカンマを指定できます。 制限事項： ■ 数値書式モデルは、カンマ要素で始めることはできません。 ■ 数値書式モデルでは、カンマを小数点文字やピリオドの右側に指定できません。
. (ピリオド)	99.99	指定した位置に小数点 (ピリオド (.)) を戻します。 制限事項： 1 つの数値書式には、1 つのピリオドのみ指定できます。
\$	\$9999	値の前にドル記号を付けて戻します。
0	0999 9990	先行 0 (ゼロ) を戻します。 後続 0 (ゼロ) を戻します。
9	9999	正の値の場合は先頭に空白を埋め込み、負の値の場合は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します。 固定小数点数の整数部分の場合、先行 0 (ゼロ) に対して空白を戻します。ただし、値 0 (ゼロ) に対しては 0 (ゼロ) を戻します。
B	B9999	整数部が 0 (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します。
C	C999	指定した位置に ISO 通貨記号 (NLS_ISO_CURRENCY パラメータの現在の値) を戻します。
D	99D99	指定した位置に小数点文字 (NLS_NUMERIC_CHARACTER パラメータの現在の値) を戻します。デフォルトはピリオド (.) です。 制限事項： 1 つの数値書式モデルには 1 つの小数点文字のみ指定できます。
EEEE	9.9EEEE	科学表記法で値を戻します。
FM	FM90.9	前後に空白を付けずに値を戻します。

表 2-13 数値書式の要素（続き）

要素	例	説明
G	9G999	指定した位置に桁区切り（NLS_NUMERIC_CHARACTER パラメータの現在の値）を戻します。単一の数値書式モデルに複数の桁区切りを指定できます。 制限事項： 数値書式モデルにおいて、桁区切りは小数点文字やピリオドの右側に指定できません。
L	L999	指定した位置にローカル通貨記号（NLS_CURRENCY パラメータの現在の値）を戻します。
MI	9999MI	負の値の後に負の符号（-）を戻します。 正の値の後に空白を戻します。 制限事項： 書式要素 MI は、数値書式モデルの最後の位置にのみ指定できます。
PR	9999PR	山カッコ <> の中に負の値を戻します。 正の値の前後に空白を付けて戻します。 制限事項： 書式要素 PR は、数値書式モデルの最後の位置にのみ指定できます。
RN	RN	大文字のローマ数字で値を戻します。
rn	rn	小文字のローマ数字で値を戻します。 値は 1 ～ 3999 の整数となります。
S	S9999 9999S	負の値の前に負の符号（-）を戻します。 正の値の前に正の符号（+）を戻します。 負の値の後に負の符号（-）を戻します。 正の値の後に正の符号（+）を戻します。 制限事項： 書式要素 S は、数値書式モデルの最初または最後の位置にのみ指定できます。
TM	TM	テキスト最小値。できるだけ少ない文字数を 10 進数で戻します。この要素は、大文字と小文字を区別しません。 デフォルトは TM9 です。デフォルトでは、戻り値が 64 文字を超えないかぎり、固定表記法で文字数を戻します。戻り値が 64 文字を超える場合、Oracle は自動的に科学表記法で文字数を戻します。 制限事項： - この要素を他の要素に優先させることはできません。 - この要素の後には、9、E（1 つ）または e（1 つ）のみを指定できます。

表 2-13 数値書式の要素（続き）

要素	例	説明
U	U99999	指定した位置にユーロまたは他の第 2 通貨記号 (NLS_DUAL_CURRENCY パラメータの現在の値) を戻します。
V	999V99	値を 10 の <i>n</i> 乗にして戻します (必要に応じて数値を丸めます)。この例では <i>n</i> は V の後の 9 の数です。
X	XXXX xxxx	指定の桁数の 16 進数値を戻します。指定した値が整数でない場合、Oracle はその値を整数に丸めます。 制限事項： ■ この要素は、正の値または 0 (ゼロ) のみを受け入れます。負の値の場合、エラーを戻します。 ■ この要素は、0 (先行 0 (ゼロ) を戻す) または FM にのみ優先させることができます。その他の要素の場合、エラーを戻します。0 (ゼロ) または FM を X と同時に指定しないと、戻り値の前に常に空白が 1 つ追加されます。

表 2-14 に、異なる `number` と '`fmt`' に対して次の問合せを行った場合の結果を示します。

```
SELECT TO_CHAR(number, 'fmt')
FROM DUAL;
```

表 2-14 数値変換例の結果

number	'fmt'	結果
-1234567890	9999999999S	'1234567890 - '
0	99.99	' .00 '
+0.1	99.99	' .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 '
0	B9999	' '
1	B9999	' 1 '
0	B90.99	' '

表 2-14 数値変換例の結果（続き）

number	'fmt'	結果
+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	' 1.2E+02 '
+123.45	FM999.009	'123.45 '
+123.0	FM999.009	'123.00 '
+123.45	L999.99	' \$123.45 '
+123.45	FML999.99	'\$123.45 '
+1234567890	9999999999S	'1234567890+ '

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文字値を DATE 値に変換する TO_* 日時ファンクション（TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_YMINTERVAL および TO_DSINTERVAL）
- デフォルト日付書式以外の書式の DATE 値を文字列に変換する（たとえば、アプリケーションから日付を出力する）TO_CHAR ファンクション

日付書式モデルの合計長は最大 22 文字です。

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。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 文を使用します。

参照：

- NLS パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 10-2 ページの「ALTER SESSION」を参照してください。

日付書式要素

日時書式モデルは、2-65 ページの表 2-15 に示す、1 つ以上の日付書式要素で構成されます。

- 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わせることもできません。たとえば、'SYYYY' と 'BC' を同一の書式文字列内で使用することはできません。
- 表 2-15 に示すとおり、日時書式要素には、TO_* 日時ファンクションで使用できないものもあります。
- 次の日時書式要素は、期間およびタイムスタンプ書式モデルでは使用できますが、元の日付書式モデル（FF、TZD、TZh、TZM および TZR）では使用できません。

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位置と同じ位置に現れます。

表 2-15 日時書式要素

要素	TO_* 日時書式で指定できるか ^a	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 / なしで西暦を示します。
AM A.M.	はい	ピリオド付き / なしで午前を示します。
BC B.C.	はい	ピリオド付き / なしで紀元前を示します。

^aTO_* 日時ファンクションとは、TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_YMINTERVAL および TO_DSINTERVAL です。

表 2-15 日時書式要素（続き）

要素	TO_* 日時書式で 指定できるか ^a	意味
CC SCC	いいえ	世紀。 ■ 4桁で表した年の下2桁が01～99（01および99を含む）の場合、世紀はその年の上2桁より1つ大きくなります。 ■ 4桁で表した年の下2桁が00の場合、世紀はその年の上2桁と同一になります。 たとえば、2002は21を戻し、2000は20を戻します。
D	はい	曜日（1～7）。
DAY	はい	曜日。9文字分の長さまで空白が埋め込まれます。
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）。
FF [1..9]	はい	小数部。基数は出力されません（基数文字の追加には、X書式要素を使用します）。戻される日時の値の小数部の桁数を指定するには、FFの後に、1～9の数字を使用します。数字を指定しない場合は、日時データ型に指定された精度またはデータ型のデフォルトの精度が使用されます。 例： 'HH:MI:SS.FF' SELECT TO_CHAR(SYSTIMESTAMP, 'SS.FF3') from dual;
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）。

^aTO_*日時ファンクションとは、TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_YMINTERVALおよびTO_DSINTERVALです。

表 2-15 日時書式要素（続き）

要素	TO * 日時書式で 指定できるか ^a	意味
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 規格に基づく 4 桁の年。
J	はい	ユリウス日。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、整数にしてください。
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埋め込まれます。
PM P.M.	いいえ	ピリオド付き / なしで午後を示します。
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	2 桁のみを使用して、21 世紀に 20 世紀の日付を格納できます。詳細は、2-69 ページの「 RR 日付要素書式 」を参照してください。
RRRR	はい	年を丸めます。4 桁または 2 桁で入力できます。2 桁の場合、RR の場合と同様の結果が戻ります。年を 4 桁で入力すると、この処理は行われません。
SS	はい	秒（0 ～ 59）。
SSSSS	はい	午前 0 時から経過した秒（0 ～ 86399）。
TZD	はい	夏時間の情報。TZD の値は、夏時間の情報を持つタイムゾーン文字列の省略形です。TZR で指定した地域と対応している必要があります。 例： PST（米国 / 太平洋標準時）、PDT（米国 / 太平洋夏時間）

^a TO_* 日時ファンクションとは、TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_YMINTERVAL および TO_DSINTERVAL です。

表 2-15 日時書式要素（続き）

要素	TO_* 日時書式で 指定できるか ^a	意味
TZH	はい	タイムゾーンの時間（TZM 書式要素を参照） 例：'HH:MI:SS.FFTZH:TZM'
TZM	はい	タイムゾーンの分（TZH 書式要素を参照） 例：'HH:MI:SS.FFTZH:TZM'
TZR	はい	タイムゾーン地域の情報。値は、データベースでサ ポートされるタイムゾーン地域である必要があります。 例：US/Pacific
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 日で終了します。
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 日で終了します。
X	はい	ローカル基数文字 例：'HH:MI:SSXFF'
Y,YYY	はい	指定した位置にカンマを付けた年。
YEAR SYEAR	いいえ	フルスペルで表した年。「S」を指定すると紀元前の 日付の先頭に「-」が付けられます。
YYYY SYYYYY	はい	4 桁で表した年。「S」を指定すると紀元前の日付の先 頭に「-」が付けられます。
YYY YY Y	はい	それぞれ年の下 3 桁、2 桁、1 桁。

^aTO_* 日時ファンクションとは、TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、
TO_YMINTERVAL および TO_DSINTERVAL です。

書式文字列に句読点文字がある日付文字列に英数字がある場合、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。

日付書式要素およびグローバリゼーション・サポート

いくつかの日時書式要素の機能は、Oracle を使用している国および言語に依存します。たとえば、次の日時書式要素は、フルスペルで値が戻されます。

- MONTH
- MON
- DAY
- DY
- BC、AD、B.C. または A.D.
- AM、PM、A.M. または 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。日時書式要素 `YEAR` と `SYEAR` によって戻される値は常に英語です。

日時書式要素 `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。

参照： グローバリゼーション・サポートの初期化パラメータの詳細は、『Oracle9i データベース・リファレンス』および『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

ISO 標準日付書式要素

Oracle は、ISO 規格に従った日時書式要素 `IYYYY`、`IYY`、`IY`、`I` および `IW` によって戻される値を計算します。これらの値と日時書式要素 `YYYY`、`YYY`、`YY`、`Y` および `WW` によって戻される値の相違点については、『Oracle9i Database グローバリゼーション・サポート・ガイド』の「グローバリゼーション・サポート」の章を参照してください。

RR 日付要素書式

RR 日時書式要素は、YY 日時書式要素に似ていますが、20 世紀以外の日付値を格納する場合の柔軟性が優れています。RR 日時書式要素によって、現在が 21 世紀でも、年の下 2 桁を指定するだけで、20 世紀の日付を格納できます。

YY 日時書式要素で `TO_DATE` ファンクションを使用した場合、日付値は常に現在の年と同じ上 2 桁で戻されます。そのかわりに RR 日時書式要素を使用した場合、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 2-16 に、RR 日時書式要素の特長を示します。

表 2-16 RR 日付要素書式

		指定された 2 桁の年	
—		00 ~ 49	50 ~ 99
現在の年の 下 2 桁	00 ~ 49	戻される年は、現在の年と同じ上 2 桁を持ちます。	戻される年の上 2 桁は、現在の年の上 2 桁より 1 少なくなります。
	50 ~ 99	戻される年の上 2 桁は、現在の年の上 2 桁より 1 多くなります。	戻される年は、現在の年と同じ上 2 桁を持ちます。

次に、RR 日時書式要素の特長を具体的に説明します。

RR 日付書式の例

次の問合せが、1950 年～ 1999 年の間に発行されるとします。

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR'), 'YYYY') "Year"
FROM DUAL;
```

Year

1998

```
SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR'), 'YYYY') "Year"
FROM DUAL;
```

Year

2017

次の問合せが、2000 年～ 2049 年の間に発行されるとします。

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR'), 'YYYY') "Year"
FROM DUAL;
```

Year

1998

```
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。RR 日時書式要素によって、年の上 2 桁が異なっても同じ値を戻す SQL 文を記述できます。

日時書式要素の接尾辞

表 2-17 に、日時書式要素に付加できる接尾辞を示します。

表 2-17 日時書式要素の接尾辞

接尾辞	意味	要素の例	値の例
TH	序数	DDTH	4TH
SP	フルスペルで表した数	DDSP	FOUR
SPTH または THSP	フルスペルで表した序数	DDSPTH	FOURTH

注意：

- これらの接尾辞を 1 つでも日時書式要素に付加した場合、戻り値は常に英語です。
- 日付の接尾辞は出力書式設定にのみ有効です。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。

書式モデルの修飾子

TO_CHAR ファンクション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。以降、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 ファンクションの戻り値における空白の埋込みを回避します。

- TO_CHAR ファンクションの日時書式要素では、この修飾子は後続の文字要素（MONTH など）では空白を回避し、日時書式モデルの後続の数要素（MI など）では先行 0（ゼロ）を回避します。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 0（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 ファンクションの数値書式要素では、この修飾子は数値の左に加えられた空白を回避します。これによって、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 ファンクション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／小文字の違いを除いて）厳密に一致する必要があります。
- 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ある必要があります。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 0（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 0（ゼロ）のチェック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。

フォーマット修飾子の例

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, 'fmMonth')||', '||TO_CHAR(SYSDATE, 'YYYY') "Ides"
FROM DUAL;
```

```
Ides
-----
3RD of April, 1998
```

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(SYSDATE, 'Month')||', '||
      TO_CHAR(SYSDATE, 'YYYY') "Ides"
FROM DUAL;
```

```
Ides
-----
03RD of April      , 1998
```

次の文は、2つの連続した一重引用符を含む日付書式モデルを使用することによって、戻り値に一重引用符を含めます。

```
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、同じ目的で一重引用符を2つ連続して使用できます。

表 2-18 に、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ます (table という名前の表には DATE データ型の列 date_column があります)。

```
UPDATE table
SET date_column = TO_DATE(char, 'fmt');
```

表 2-18 FX 書式モデル修飾子による文字データと書式モデルのマッチング

char	'fmt'	一致とエラー
'15/JAN/1998'	'DD-MON-YYYY'	一致
' 15! JAN % /1998'	'DD-MON-YYYY'	エラー
'15/JAN/1998'	'FXDD-MON-YYYY'	エラー
'15-JAN-1998'	'FXDD-MON-YYYY'	一致
'1-JAN-1998'	'FXDD-MON-YYYY'	エラー
'01-JAN-1998'	'FXDD-MON-YYYY'	一致
'1-JAN-1998'	'FXFMDD-MON-YYYY'	一致

文字列から日付への変換に関する規則

次の追加の書式化規則は、文字列値を日付値に変換する場合に適用されます（ただし、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した場合は適用できません）。

- 先行 0（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。たとえば、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 を指定した場合です。
- 日付文字列から、書式文字列の最後にある時刻フィールドを省略できます。
- 日時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 2-19 に示すとおり、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 2-19 Oracle 書式マッチング

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'

XML 書式モデル

SYS_XMLGEN ファンクションは、XML 文書を含む XMLType 型のインスタンスを戻します。Oracle では、SYS_XMLGEN ファンクションの結果をフォーマットする XMLFormat オブジェクトを提供します。

表 2-20 に、XMLFormat オブジェクトの属性を示します。表に示すように、ファンクションはこの型を実装します。

参照：

- SYS_XMLGEN ファンクションの詳細は、6-157 ページの「SYS_XMLGEN」を参照してください。
- XMLFormat オブジェクトの実装およびその使用の詳細は、『Oracle9i XML API リファレンス -XDK および Oracle XML DB』および『Oracle9i XML Developer's Kit ガイド -XDK』を参照してください。

表 2-20 XMLFormat オブジェクトの属性

属性	データ型	用途
enclTag	VARCHAR2 (100)	SYS_XMLGEN ファンクションの結果の囲みタグ名です。ファンクションへの入力が列名である場合、デフォルトはその列名です。それ以外の場合、デフォルトは ROW です。schemaType を USE_GIVEN_SCHEMA に設定すると、この属性によって XML Schema の要素名を指定することもできます。
schemaType	VARCHAR2 (100)	出力された XML 文書のスキーマ生成の型です。有効な値は、'NO_SCHEMA' および 'USE_GIVEN_SCHEMA' です。デフォルトは、'NO_SCHEMA' です。
schemaName	VARCHAR2 (4000)	schemaType の値が 'USE_GIVEN_SCHEMA' のときに、Oracle が使用するターゲット・スキーマの名前です。schemaName を指定すると、囲みタグが要素名として使用されます。
processingIns	VARCHAR2 (4000)	ファンクションの出力の最上位に、要素の前に追加されるユーザーの処理命令です。

XMLFormat オブジェクトを実装するファンクションは、次のとおりです。

```

    STATIC FUNCTION createFormat (
        enclTag IN varchar2 := 'ROWSET',
        schemaType IN varchar2 := 'NO_SCHEMA',
        schemaName IN varchar2 := null,
        targetNameSpace IN varchar2 := null,
        dburlPrefix IN varchar2 := null,
        processingIns IN varchar2 := null) RETURN XMLGenFormatType,
    MEMBER PROCEDURE genSchema (spec IN varchar2),
    MEMBER PROCEDURE setSchemaName(schemaName IN varchar2),
    MEMBER PROCEDURE setTargetNameSpace(targetNameSpace IN varchar2),
    MEMBER PROCEDURE setEnclosingElementName(enclTag IN varchar2),
    MEMBER PROCEDURE setDbUrlPrefix(prefix IN varchar2),
    MEMBER PROCEDURE setProcessingIns(pi IN varchar2),
    CONSTRUCTOR FUNCTION XMLGenFormatType (
        enclTag IN varchar2 := 'ROWSET',
        schemaType IN varchar2 := 'NO_SCHEMA',
        schemaName IN varchar2 := null,
        targetNameSpace IN varchar2 := null,
        dburlPrefix IN varchar2 := null,
        processingIns IN varchar2 := null) RETURN SELF AS RESULT

```

NULL

行のある列の値がない場合、その列は **NULL** である、または **NULL** を含むといいます。**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の場合は、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 0（ゼロ）と同じではないため、0（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 0（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** の今後のバージョンでも継続されるとはかぎらないため、空の文字列を **NULL** として処理しないことをお勧めします。**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** を指定した場合、（連結演算子を除く）すべての演算子は **NULL** を戻します。

SQL ファンクションでの NULL

引数として **NULL** を指定した場合、（**REPLACE**、**NVL** および **CONCAT** を除く）すべてのスカラー・ファンクションでは **NULL** が戻されます。**NVL** ファンクションを使用した場合、**NULL** が発生したときに値を戻すことができます。たとえば、**NVL (COMM, 0)** は、**COMM** が **NULL** の場合は 0（ゼロ）を戻し、**COMM** が **NULL** でなければ **COMM** の値を戻します。

ほぼすべての集計ファンクションでは、**NULL** は無視されます。たとえば、1000、**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。

比較条件での NULL

NULL を検査するには、比較条件 **IS NULL** および **IS NOT NULL** のみを使用します。**NULL** を他の条件で使用して、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** ファンクションを評価するときに 2 つの **NULL** を等しい値とみなします。

参照： 構文および追加情報については、6-48 ページの「[DECODE](#)」を参照してください。

コンポジット・キーの場合、2 つの **NULL** は等しいと判断されます。**NULL** を含む 2 つのコンポジット・キーは、そのキーの **NULL** 以外の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。ただし、UNKNOWN と評価される条件は FALSE 条件とは異なり、UNKNOWN 条件をさらに評価しても UNKNOWN と評価されます。したがって、NOT FALSE は TRUE と評価されますが、NOT UNKNOWN は UNKNOWN と評価されます。

表 2-21 は、条件に NULL を含む評価の例です。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 2-21 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 を含む論理条件の結果を示した真理値表は、5-9 ページの表 5-4、5-9 ページの表 5-5 および 5-9 ページの表 5-6 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
- XMLDATA

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。

- 順序の現在の値を戻します。
- 順序を増加させて次の値を戻します。

CURRVAL と NEXTVAL は、順序の名前で修飾する必要があります。

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

別のユーザーのスキーマ内での順序の現在の値または次の値を参照するには、その順序に対する SELECT オブジェクト権限または SELECT ANY SEQUENCE システム権限のどちらかが必要です。さらに、その順序は、次に示すとおり、順序を含むスキーマで修飾する必要があります。

```
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も順序を修飾する必要があります。

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

参照： データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。

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、マテリアライズド・ビューまたはビューに含まれていない SELECT 文の SELECT 構文のリスト
- INSERT 文内の副問合せの SELECT 構文のリスト
- INSERT 文の VALUES 句
- UPDATE 文の SET 句

順序値の制限事項： 次の構造体では、CURRVAL および NEXTVAL は使用できません。

- DELETE 文、SELECT 文または UPDATE 文内の副問合せ
- ビューの問合せ、またはマテリアライズド・ビューの問合せ
- 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、INTERSECT または 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、CURRVAL または NEXTVAL を使用する単一の SQL 文では、参照された LONG 列、更新された表、ロックされた表がすべて同じデータベース上にある必要があります。

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 NEXTVAL 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。

NEXTVAL への参照が含まれる単一の SQL 文の中では、Oracle は、次の各行につき 1 回のみ順序を増加させます。

- SELECT 文の外部問合せブロックにより戻される行。このような問合せブロックは、次の場所に指定できます。
 - トップレベルの SELECT 文
 - INSERT ... SELECT 文（単一表または複数表）。マルチテーブル・インサートの場合、NEXTVAL への参照が VALUES 句内に存在する必要がある、マルチテーブル・インサートの複数のブランチで NEXTVAL が参照される場合でも、副問合せによって戻される行ごとに、順序が 1 回更新されます。
 - CREATE TABLE ... AS SELECT 文
 - CREATE MATERIALIZED VIEW ... AS SELECT 文
- UPDATE 文で更新される行
- VALUES 句が含まれる INSERT 文
- MERGE 文でマージ（挿入または更新）される行。NEXTVAL への参照は、`merge_insert_clause` または `merge_update_clause` に指定できます。

これらの場所のいずれかが、NEXTVAL を複数回参照している場合、Oracle は 1 回のみ順序を増加させ、NEXTVAL が検出されるたびにすべて同じ値を戻します。

これらの場所のいずれかが、CURRVAL と NEXTVAL の両方を参照している場合、Oracle は CURRVAL と NEXTVAL の両方について順序を増加させ同じ値を戻します。

順序には、待機またはロックすることなく多数のユーザーが同時にアクセスできます。

参照： 順序の詳細は、14-82 ページの「[CREATE SEQUENCE](#)」を参照してください。

順序の次の値の検索例 次の例では、サンプル・スキーマ hr の従業員順序の次の値を検索します。

```
SELECT employees_seq.nextval
FROM DUAL;
```

表への順序値の挿入例 次の例では、従業員順序を増加させ、サンプル表 hr.employees 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 employees
VALUES (employees_seq.nextval, 'John', 'Doe', 'jdoe',
       '555-1212', TO_DATE(SYSDATE), 'PU_CLERK', 2500, null, null,
       30);
```

順序の現在の値の再利用例 次の例では、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。その後、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 orders (order_id, order_date, customer_id)
VALUES (orders_seq.nextval, TO_DATE(SYSDATE), 106);

INSERT INTO order_items (order_id, line_item_id, product_id)
VALUES (orders_seq.currval, 1, 2359);

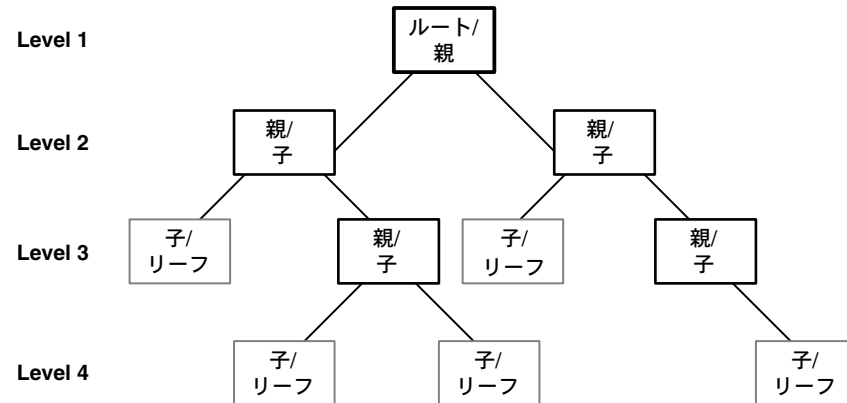
INSERT INTO order_items (order_id, line_item_id, product_id)
VALUES (orders_seq.currval, 2, 3290);

INSERT INTO order_items (order_id, line_item_id, product_id)
VALUES (orders_seq.currval, 3, 2381);
```

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、ルート行に 1 を戻し、ルートの子には 2 を戻します（以降同様に続きます）。**ルート行**は逆ツリー構造の最上位行です。**子である行**は任意の非ルート行です。**親である行**は子を持つ任意の行です。**リーフ行**は子を持たない任意の行です。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



問合せの中で階層型の関連を定義するには、START WITH 句および CONNECT BY 句を使用する必要があります。

WHERE 句の LEVEL の制限事項： WHERE 句の [NOT] IN 条件の右辺が副問合せになっている場合、条件の左辺で LEVEL は使用できません。ただし、FROM 句の副問合せで LEVEL を指定すると、同じ結果が得られます。たとえば、次の文は無効です。

```

SELECT employee_id, last_name FROM employees
  WHERE (employee_id, LEVEL)
        IN (SELECT employee_id, 2 FROM employees)
  START WITH employee_id = 2
  CONNECT BY PRIOR employee_id = manager_id;

```

ただし、次の文は、LEVEL 情報を含む問合せを FROM 句にカプセル化するため有効です。

```

SELECT v.employee_id, v.last_name, v.lev
  FROM
    (SELECT employee_id, last_name, LEVEL lev
     FROM employees v
     START WITH employee_id = 100
     CONNECT BY PRIOR employee_id = manager_id) v

```

```
WHERE (v.employee_id, v.lev) IN  
(SELECT employee_id, 2 FROM employees);
```

参照： 階層問合せの概要については、8-3 ページの「[階層問合せ](#)」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 疑似列によって行のアドレスが戻されます。Oracle9i の ROWID 値には、行を検索する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 ROWID または UROWID データ型を持ちます。

参照： 2-31 ページの「[ROWID データ型](#)」および 2-33 ページの「[UROWID データ型](#)」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。

- 単一の行に最も速くアクセスする方法です。
- 表の行が格納されている様子を示すことができます。
- 表の行に対する一意の識別子です。

表の主キーとして ROWID を使用しないでください。たとえば、インポート・ユーティリティとエクスポート・ユーティリティで行を削除してから再挿入する場合、ROWID が変わる場合があります。行を削除した場合、Oracle は、後から新しく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、更新および削除はできません。

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, last_name  
FROM employees  
WHERE department_id = 20;
```

ROWNUM

ROWNUM 疑似列は、問合せによって戻される各行について、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を戻します。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（以降同様に続きます）。

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 employees WHERE ROWNUM < 10;
```

同じ問合せで ROWNUM に ORDER BY 句が続く場合、ORDER BY 句によって行が再び順序付けられます。結果は、行がアクセスされる方法によって異なります。たとえば、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、後続の文には前述の例と同じ効果はありません。

```
SELECT * FROM employees WHERE ROWNUM < 11 ORDER BY last_name;
```

ORDER BY 句を副問合せに埋め込んで ROWNUM 条件をトップレベル問合せに置いた場合、行の順序付けの後で ROWNUM 条件を強制的に適用させることができます。たとえば、次の問合せは、小さい順から 10 個の従業員数を戻します。これは、上位 N 番までの問合せと呼ばれることがあります。

```
SELECT * FROM  
  (SELECT * FROM employees ORDER BY employee_id)  
 WHERE ROWNUM < 11;
```

前述の例では、ROWNUM 値はトップレベルの SELECT 文の値です。これらの値は、副問合せ内の employee_id によって行が順序付けられた後で生成されます。

参照： 上位 N 番までの問合せの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

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

```
SELECT * FROM employees  
 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、条件は偽と判断されます。2 番目にフェッチされる予定だ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、条件も偽と判断されます。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 my_table
SET column1 = ROWNUM;
```

注意： 問合せで ROWNUM を使用した場合、ビューの最適化に影響することがあります。詳細は、『Oracle9i データベース概要』を参照してください。

XMLDATA

Oracle は、XMLSchema 情報および STORAGE 句の指定方法に基づいて、XMLType データを LOB またはオブジェクト・リレーショナル列に格納します。XMLDATA 疑似列を使用すると、基礎となる LOB またはオブジェクト・リレーショナル列にアクセスし、追加の STORAGE 句のパラメータ、制約、索引などを指定できます。

例 次の文は、この疑似列の使用方を示しています。XMLType の簡単な表を作成とします。

```
CREATE TABLE xml_lob_tab of XMLTYPE;
```

デフォルト記憶域は、CLOB 列にあります。基礎となる LOB 列の記憶特性を変更する場合は、次の文を使用できます。

```
ALTER TABLE xml_lob_tab MODIFY LOB (XMLDATA)
(STORAGE (BUFFER_POOL DEFAULT) CACHE);
```

D-11 ページの「[SQL 文での XML の使用方法](#)」で作成した xwarehouses 表のような XML Schema ベースの表を作成したとします。その後、次の文に示すように、XMLDATA 列を使用して、基礎となる列のプロパティを設定できます。

```
ALTER TABLE xwarehouses ADD (UNIQUE(XMLDATA."WarehouseId"));
```

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なります。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、パラメータまたは句読点の間に入れることができます。次のいずれかの方法を使用します。

- スラッシュとアスタリスク (/*) を使用してコメントを開始します。コメントのテキストを続けます。このテキストは複数行にまたがってもかまいません。アスタリスクとスラッシュ (*/) を使用して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します。コメントのテキストを続けます。このテキストは複数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もかまいません。コメントのテキストには、使用しているデータベース・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。

例 次の文には多くのコメントが含まれています。

```
SELECT last_name, salary + NVL(commission_pct, 0),
       job_id, e.department_id
/* Select all employees whose compensation is
greater than that of Pataballa.*/
FROM employees e, departments d
      /*The DEPARTMENTS table is used to get the department name.*/
WHERE e.department_id = d.department_id
      AND salary + NVL(commission_pct,0) > /* Subquery:      */
      (SELECT salary + NVL(commission_pct,0)
       /* total compensation is salar + commission_pct */
       FROM employees
       WHERE last_name = 'Pataballa');

SELECT last_name,           -- select the name
       salary + NVL(commission_pct, 0), -- total compensation
       job_id,              -- job
       e.department_id      -- and department
FROM employees e,          -- of all employees
     departments d
WHERE e.department_id = d.department_id
      AND salary + NVL(commission_pct, 0) > -- whose compensation
                                           -- is greater than
      (SELECT salary + NVL(commission_pct,0) -- the compensation
       FROM employees
       WHERE last_name = 'Pataballa')      -- of Pataballa.

;
```

スキーマ・オブジェクトに関するコメント

COMMENT コマンドを使用して、表、ビュー、マテリアライズド・ビューまたは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。

参照： コメントについては、12-67 ページの「[COMMENT](#)」を参照してください。

ヒント

Oracle オプティマイザに指示（**ヒント**）を与えるために、SQL 文中でコメントを使用できます。オプティマイザは、これらのヒントを提案として使用し、文の実行計画を選択します。

文ブロックには、ヒントを含むコメントは 1 つのみ指定できます。このコメントは、SELECT、UPDATE、INSERT または DELETE のいずれかのキーワードの後でのみ指定できます。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]]...
```

それぞれの意味は、次のとおりです。

- DELETE、INSERT、SELECT または UPDATE は、文ブロックを始める DELETE、INSERT、SELECT または 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のみ指定できます。
- +（プラス記号）は、コメントをヒントのリストとして、Oracle に解析させます。プラス記号は、コメント・デリミタの直後に置く必要があります（空白を入れてはいけません）。
- hint は、この項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、1 つ以上の空白で区切る必要があります。
- text は、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 2-22 に、機能のカテゴリに分類したヒントを示します。表の後には、ヒントをアルファベット順に示し、その構文および説明を示します。

注意： Oracle は、誤ったスペルのヒントを標準のコメントとして処理し、エラーを戻しません。

参照： ヒントの詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』および『Oracle9i データベース概要』を参照してください。

表 2-22 機能のカテゴリに分類したヒント

カテゴリ	ヒント
最適化目標と方法	ALL_ROWS および FIRST_ROWS
	CHOOSE
	RULE
アクセス方法のヒント	AND_EQUAL
	CLUSTER
	FULL
	HASH
	INDEX および NO_INDEX
	INDEX_ASC および INDEX_DESC
	INDEX_COMBINE
	INDEX_FFS
結合順序のヒント	ROWID
	ORDERED
	STAR
結合操作のヒント	DRIVING_SITE
	HASH_SJ、MERGE_SJ および NL_SJ
	LEADING
	USE_HASH および USE_MERGE
	USE_NL
パラレル実行のヒント	PARALLEL および NOPARALLEL
	PARALLEL_INDEX
	PQ_DISTRIBUTE
	NOPARALLEL_INDEX

表 2-22 機能のカテゴリに分類したヒント（続き）

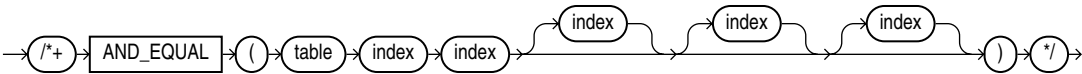
カテゴリ	ヒント
問合せ変換のヒント	EXPAND_GSET_TO_UNION
	FACT および NOFACT
	MERGE
	NO_EXPAND
	NO_MERGE
	REWRITE および NOREWRITE
	STAR_TRANSFORMATION
	USE_CONCAT
その他のヒント	APPEND および NOAPPEND
	CACHE および NOCACHE
	CURSOR_SHARING_EXACT
	DYNAMIC_SAMPLING
	NESTED_TABLE_GET_REFS
	UNNEST および NO_UNNEST
	ORDERED_PREDICATES
	PUSH_PRED および NO_PUSH_PRED
	PUSH_SUBQ および NO_PUSH_SUBQ

all_rows_hint::=



ALL_ROWS ヒントによって、最高のスループットを実現する（リソース使用の合計を最小限にする）という目的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します。

and_equal_hint::=



AND_EQUAL ヒントによって、複数の単一列索引のスキャン結果をマージするアクセス・パスを使用する実行計画を明示的に選択します。

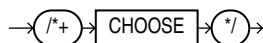
append_hint::=

データベースがシリアル・モードで実行中の場合、APPEND ヒントによって、ダイレクト・パス・インサートを使用可能にできます。Enterprise Edition を使用していない場合、データベースはシリアル・モードです。シリアル・モードでは、従来の INSERT がデフォルトです。パラレル・モードでは、ダイレクト・パス・インサートがデフォルトです。

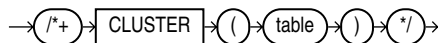
ダイレクト・パス・インサートでは、データは、表に割り当てられている既存の領域ではなく、表の最後に追加されます。その結果、ダイレクト・パス・インサートの処理速度は、従来の INSERT の処理速度よりも非常に速くなります。

cache_hint::=

CACHE ヒントによって、全表スキャンの実行時に、この表に取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最高使用頻度側に入れます。このオプションは、小規模な参照表で有効です。

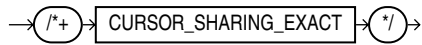
choose_hint::=

CHOOSE ヒントによって、SQL 文に対して、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します。オプティマイザは、文によってアクセスされる表の統計情報に基づいて選択します。データ・ディクショナリが、これらの表の 1 つ以上に統計表を持つ場合、オプティマイザはコストベース方法を使用し、最適なスループットを実現するために最適化します。データ・ディクショナリがこれらの表の統計情報を持たない場合、ルールベース方法を使用します。

cluster_hint::=

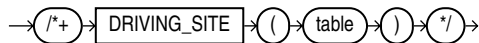
CLUSTER ヒントによって、指定された表にアクセスするために、クラスタ・スキャンを明示的に選択します。このヒントは、クラスタ化されているオブジェクトでのみ適用されます。

cursor_sharing_exact_hint::=



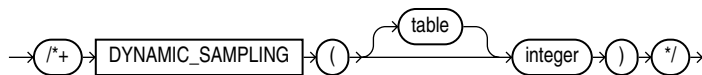
セキュリティ上の問題がなければ、Oracle は、SQL 文のリテラルをバインド変数で置換できます。これは、CURSOR_SHARING 起動パラメータによって制御されます。CURSOR_SHARING_EXACT ヒントによって、この置換処理をオフにできます。つまり、Oracle は、バインド変数によってリテラルを置換せずに SQL 文を実行します。

driving_site_hint::=



DRIVING_SITE ヒントによって、問合せ処理が、Oracle によって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します。このヒントは、ルールベースまたはコストベースの最適化で使用可能です。

dynamic_sampling_hint::=



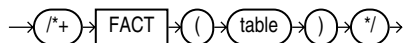
DYNAMIC_SAMPLING ヒントによって、動的サンプリングを制御でき、より正確な選択性およびカーディナリティの推定値を判断することで、サーバーのパフォーマンスが向上します。DYNAMIC_SAMPLING の値は、0 ～ 10 の値に設定できます。レベルが高くなると、コンパイラは動的サンプリングを頻繁に行い、動的サンプリングの適用範囲が広がります。表を指定しないかぎり、サンプリングはデフォルトでカーソル・レベルに設定されます。

expand_gset_to_union_hint::=

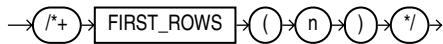


EXPAND_GSET_TO_UNION ヒントは、グルーピング・セットが含まれている問合せ (GROUP BY GROUPING SET または GROUP BY ROLLUP が指定された問合せなど) に使用します。このヒントによって、個々のグルーピングの UNION ALL を使用して、問合せが対応する問合せに変換されるようにします。

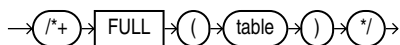
fact_hint::=



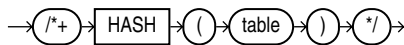
FACT ヒントは、スター型変換のコンテキストで使用され、ヒントに指定された表をファクト表であるとみなすようにスター型変換に指示します。

first_rows_hint::=

FIRST_ROWS(*n*) (*n* は正の整数) ヒントまたは **FIRST_ROWS** によって、Oracle に個々の SQL 文に対する応答を高速化するための最適化を行うように指示します。**FIRST_ROWS(*n*)** ヒントは、Oracle に、最初の *n* 行を最も効果的に戻す計画を選択するように指示するため、高い精度を実現します。**FIRST_ROWS** ヒントは、最初の 1 行を戻す計画を最適化し、下位互換性およびブラン・スタビリティを保持します。

full_hint::=

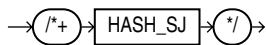
FULL ヒントによって、指定された表に対する全表スキャンを明示的に選択します。

hash_hint::=

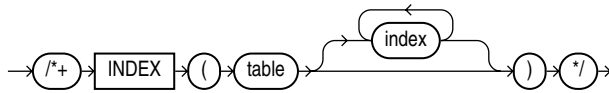
HASH ヒントによって、指定された表にアクセスするために、ハッシュ・スキャンを明示的に選択します。このヒントは、クラスタに格納されている表のみで適用されます。

hash_aj_hint::=

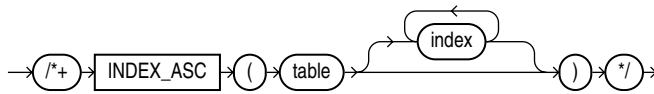
特定の問合せの場合に、**EXISTS** 副問合せに **HASH_AJ**、**MERGE_AJ** または **NL_AJ** ヒントを指定します。**HASH_AJ** はハッシュ・アンチ結合を、**MERGE_AJ** はソート / マージ・アンチ結合を、**NL_AJ** はネストされたループ・アンチ結合を使用します。

hash_sj_hint::=

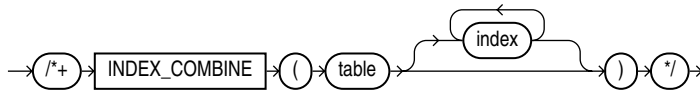
特定の問合せの場合に、**EXISTS** 副問合せに **HASH_SJ**、**MERGE_SJ** または **NL_SJ** ヒントを指定します。**HASH_SJ** はハッシュ・セミ結合を、**MERGE_SJ** はソート / マージ・セミ結合を、**NL_SJ** はネストされたループ・セミ結合を使用します。

index_hint::=

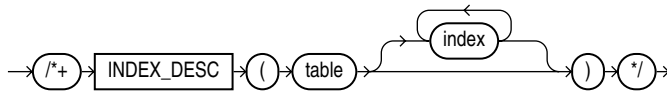
INDEX ヒントによって、指定された表に対する索引スキャンを明示的に選択します。INDEX ヒントは、ドメイン、B ツリー、ビットマップ索引およびビットマップ・ジョイン・インデックスに対して使用できます。ただし、ビットマップ索引に対しては、INDEX ヒントではなく、適応性の高い INDEX_COMBINE ヒントを使用することをお勧めします。

index_asc_hint::=

INDEX_ASC ヒントによって、指定された表に対する索引スキャンを明示的に選択します。文が索引レンジ・スキャンを使用する場合、索引付けされた値の昇順に索引エントリをスキャンします。

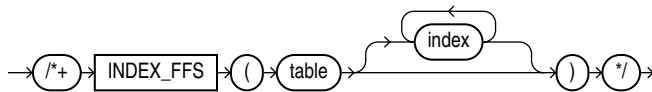
index_combine_hint::=

INDEX_COMBINE ヒントによって、表に対するビットマップ・アクセス・パスを明示的に選択します。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、表にアクセスするための推定コストが最適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します。特定の索引が引数として指定されている場合、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に試します。

index_desc_hint::=

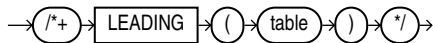
INDEX_DESC ヒントによって、指定された表に対する索引スキャンを明示的に選択します。文が索引レンジ・スキャンを使用する場合、索引付けされた値の降順に索引エントリをスキャンします。パーティション索引では、結果はパーティションごとに降順に戻されます。

index_ffs_hint::=



INDEX_FFS ヒントによって、全表スキャンではなく、高速全索引スキャンを実行します。

leading_hint::=



LEADING ヒントによって、結合順序で最初の表として指定された表を使用できるようにします。

異なる表で LEADING ヒントを指定する場合、すべての LEADING ヒントは無視されます。ORDERED ヒントを指定すると、すべての LEADING ヒントは上書きされます。

merge_hint::=



MERGE ヒントによって、問合せベースでビューをマージできます。

ビューに対する問合せの SELECT 構文のリストに GROUP BY 句または DISTINCT 演算子が含まれる場合、複合ビューがマージ可能であれば、オプティマイザはビューの問合せをアクセス文にマージできます。また、副問合せと相関関係がなければ、IN 副問合せをアクセス文にマージする場合にも複合マージを使用できます。

複合マージはコストベースでは実行されません。アクセス問合せブロックには、MERGE ヒントを含める必要があります。このヒントを含めない場合、オプティマイザは別の方法を使用します。

merge_aj_hint::=



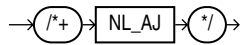
HASH_AJ ヒントを参照してください。

merge_sj_hint::=



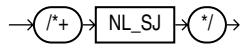
HASH_SJ ヒントを参照してください。

nl_aj_hint::=



HASH_AJ ヒントを参照してください。

nl_sj_hint::=



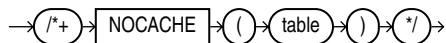
HASH_SJ ヒントを参照してください。

noappend_hint::=



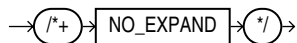
INSERT 文の実行中のパラレル・モードを使用禁止にすると、NOAPPEND ヒントによって従来の INSERT を使用可能にできます（シリアル・モードでは、従来の INSERT がデフォルトです。パラレル・モードでは、ダイレクト・パス・インサートがデフォルトです）。

nocache_hint::=



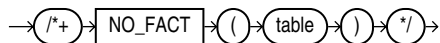
NOCACHE ヒントによって、全表スキャンの実行時に、この表に取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最低使用頻度側に入れます。これは、バッファ・キャッシュ内のブロックの通常の動作です。

no_expand_hint::=



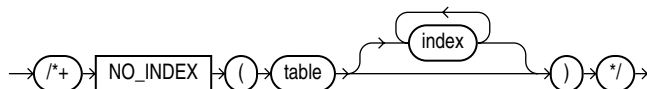
NO_EXPAND ヒントによって、WHERE 句に OR 条件または IN リストが指定されている問合せに対して、コストベースのオプティマイザが OR 拡張を検討しないようにします。通常、オプティマイザは、OR 拡張の使用を検討し、使用しない場合よりコストが低いと判断した場合は、この方法を使用します。

no_fact_hint::=



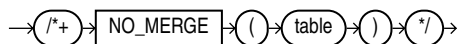
NO_FACT ヒントは、スター型変換のコンテキストの中で使用され、ヒントに指定された表をファクト表とみなさないように指示します。

no_index_hint::=



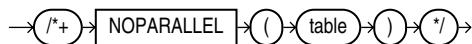
NO_INDEX ヒントによって、指定された表に対する一連の索引を明示的に禁止します。

no_merge_hint::=



NO_MERGE ヒントによって、マージ可能なビューをマージしないようにします。

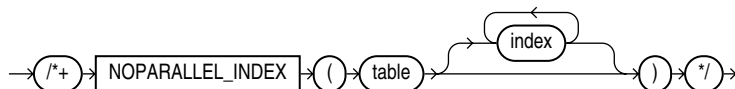
noparallel_hint::=



NOPARALLEL ヒントによって、表内の句の PARALLEL 指定を上書きします。一般的に、ヒントは表内の句より優先されます。

NOPARALLEL での制限事項： ネストした表を含む問合せは、パラレル化できません。

noparallel_index_hint::=



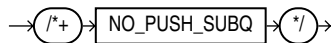
NOPARALLEL_INDEX ヒントによって、パラレル索引スキャン操作を回避するために、索引の PARALLEL 属性を上書きします。

no_push_pred_hint::=



NO_PUSH_PRED ヒントによって、ビューへ述語結合を送信しないようにします。

no_push_subq_hint::=



NO_PUSH_SUBQ ヒントによって、マージされていない副問合せを、実行計画の最後の手順として評価するようにします。比較的成本が高く、行数を大幅に削減しない副問合せの場合、副問合せを最後に評価することによってパフォーマンスを向上できます。

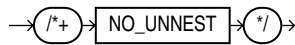
norewrite_hint::=



NOREWRITE ヒントによって、QUERY_REWRITE_ENABLED パラメータの設定を上書きして、問合せブロックのクエリー・リライトを使用禁止にします。問合せブロック要求には、NOREWRITE ヒントを使用してください。

注意： NOREWRITE ヒントによって、ファンクション索引を使用禁止にします。

no_unnest_hint::=



NO_UNNEST ヒントによって、特定の副問合せブロックに対するネスト解除が使用禁止になります。

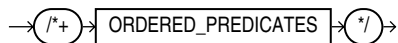
ordered_hint::=



ORDERED ヒントによって、表が、FROM 句に指定されている順序で結合されるようにします。

結合を実行する SQL 文で ORDERED ヒントを省略した場合、オプティマイザが、表の結合順序を選択します。オプティマイザが選択しない表から選択される行数がわかる場合は、ORDERED ヒントを使用すると、結合順序を指定できる場合があります。この情報によって、オプティマイザより効率的に内部表および外部表を選択できます。

ordered_predicates_hint::=



ORDERED_PREDICATES ヒントによって、索引キーとして使用される述語を除く述語の評価順序をオプティマイザが保持できるようにします。SELECT 文の WHERE 句で、このヒントを使用します。

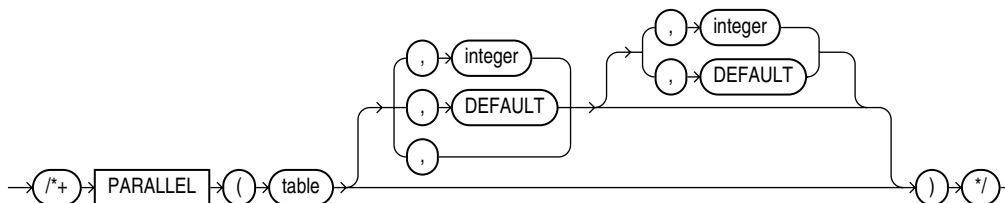
ORDERED_PREDICATES ヒントを使用しない場合、Oracle では、次の順序ですべての述語を評価します。

1. 最初に、ユーザー定義ファンクション、型メソッドまたは副問合せのない述語は、WHERE 句に指定されている順序で評価されます。

2. 次に、ユーザーが計算したコストを持つユーザー定義ファンクションおよび型メソッドのある述語が、そのコストの昇順で評価されます。
3. 次に、ユーザーが計算したコストを持たないユーザー定義ファンクションおよび型メソッドのある述語が、WHERE 句に指定された順序で評価されます。
4. 次に、WHERE 句に指定されていない述語（オプティマイザが中間的に生成した述語など）が評価されます。
5. 最後に、副問合せのある述語が、WHERE 句に指定されている順序で評価されます。

注意： 前述のとおり、ORDERED_PREDICATES ヒントを使用して、索引キーの述語評価の順序を保持することはできません。

parallel_hint::=



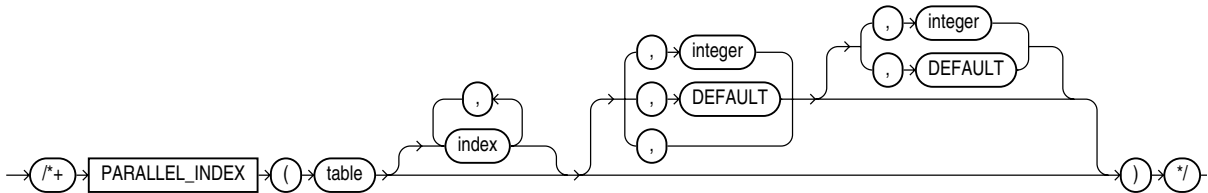
PARALLEL ヒントによって、パラレル操作に必要な同時サーバーの数を指定できます。ヒントは、SELECT、INSERT、UPDATE および DELETE 文の一部および表スキャン部分に適用されます。

注意： ソートまたはグループ化操作では、使用できるサーバーの数は、PARALLEL ヒントの 2 倍です。

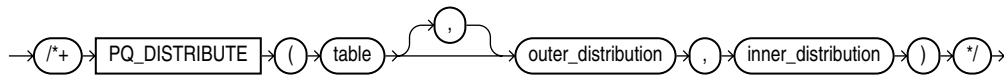
パラレル制限に違反した場合、ヒントは無視されます。

注意： Oracle は、一時表に関するパラレル・ヒントを無視します。

参照： 15-7 ページの [CREATE TABLE](#) および『Oracle9i データベース概要』を参照してください。

parallel_index_hint::=

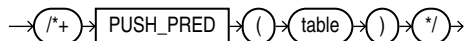
PARALLEL_INDEX ヒントによって、パーティション索引の索引範囲スキャンの平行化に必要な同時サーバー数を指定します。

pq_distribute_hint::=

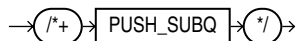
PQ_DISTRIBUTE ヒントによって、平行結合操作のパフォーマンスを向上させます。結合された表の行を、生成側の問合せサーバーと受取側の問合せサーバーの間でどのように分散するかを指定します。このヒントによって、オプティマイザが通常行う決定が上書きされます。

EXPLAIN PLAN 文を使用すると、オプティマイザが指定した分散を確認できます。両方の表がシリアルの場合、オプティマイザは分散ヒントを無視します。

参照： 外部および内部結合表への分散で使用可能な組合せについては、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

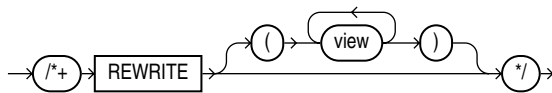
push_pred_hint::=

PUSH_PRED ヒントによって、ビューへの述語結合を送信するようにします。

push_subq_hint::=

PUSH_SUBQ ヒントによって、マージされていない副問合せを、実行計画の中でできるだけ早く評価するようにします。一般に、マージされていない副問合せは、実行計画の最後に実行されます。比較的成本が低く、行数を大幅に削減する副問合せの場合、副問合せを早く評価することによってパフォーマンスを向上できます。

副問合せがリモート表に適用される場合、または副問合せがマージ結合を使用して結合される場合は、このヒントを使用しても効果はありません。

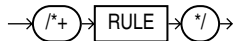
rewrite_hint::=

REWRITE ヒントによって、コストを考慮する必要がない場合、コストベースのオプティマイザに、マテリアライズド・ビューの間合せをリライトさせます。ビュー・リストの指定にかかわらず、REWRITE ヒントを使用してください。ビュー・リストを指定して REWRITE ヒントを使用し、そのリストに適切なマテリアライズド・ビューが含まれる場合、Oracle はコストに関係なくそのビューを使用します。

Oracle では、リスト外のビューを考慮しません。ビュー・リストを指定しない場合、Oracle は適切なマテリアライズド・ビューを検索し、コストに関係なくそのビューを使用します。

rowid_hint::=

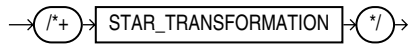
ROWID ヒントによって、指定された表に対する ROWID による表スキャンを明示的に選択します。

rule_hint::=

RULE ヒントによって、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します。これによって、文ブロックに指定されている別のヒントは無視されます。

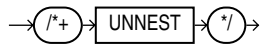
star_hint::=

STAR ヒントによって、スター間合せ計画を強制的に使用します。スター計画の間合せでは、結合順序の最後に最大表があり、その最大表とネストされたループ結合を連結索引上で結合します。3 つ以上の表があり、大規模表の連結索引に 3 つ以上の列があり、アクセス方法のヒントまたは結合方法のヒントに矛盾がない場合に、STAR ヒントを適用します。また、オプティマイザは、小規模表の異なる順列についても考慮します。

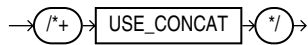
star_transformation_hint::=

STAR_TRANSFORMATION ヒントによって、型変換が適用された最適な計画をオプティマイザが使用するようにします。ヒントを使用しない場合、オプティマイザは、型変換を適用した問合せ用の最適な計画ではなく、型変換を適用せずに生成した最適な計画を使用することを、コストベースで判断します。

ヒントを指定した場合でも、型変換が適用されるとはかぎりません。オプティマイザは、適切であると判断した場合にのみ、副問合せを生成します。副問合せが生成されない場合、型変換が適用された問合せが存在しないため、型変換が適用されない問合せに対する最適な計画が、ヒントの有無にかかわらず使用されます。

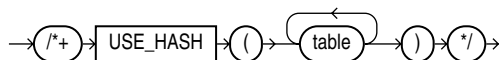
unnest_hint::=

UNNEST ヒントによって、Oracle は副問合せブロックの妥当性のみを確認するように指示します。有効な副問合せブロックであれば、副問合せのネスト解除は使用可能になります。

use_concat_hint::=

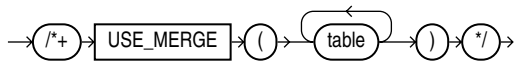
USE_CONCAT ヒントによって、問合せの WHERE 句に指定されている結合条件 OR を、UNION ALL 集合演算子を使用して複合問合せに変換します。一般的に、この型変換は、連結を使用する問合せのコストが、連結を使用しないコストより少ない場合にのみ発生します。

USE_CONCAT ヒントは、IN リストの処理および IN リストを含むすべての論理和に対する OR 拡張をオフにします。

use_hash_hint::=

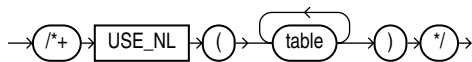
USE_HASH ヒントによって、ハッシュ結合を使用して、指定された各表を別の行ソースに結合させます。

use_merge_hint::=



USE_MERGE ヒントによって、ソート / マージ結合を使用して、指定された各表を別の行ソースに結合させます。

use_nl_hint::=



USE_NL ヒントによって、ネストされたループ結合を使用して、指定された各表を別の行ソースに結合させます。この場合、指定された表を内部表として使用します。

データベース・オブジェクト

次の項で説明するとおり、Oracle は、特定のスキーマに対応付けられたオブジェクトと、特定のスキーマに対応付けられていないオブジェクトを認識します。

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 を使用して作成および操作できます。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
- 制約
- データベース・リンク
- データベース・トリガー
- ディメンション
- 外部プロシージャ・ライブラリ
- 索引構成表
- 索引
- 索引タイプ

- Java クラス、Java リソース、Java ソース
- マテリアライズド・ビュー
- マテリアライズド・ビュー・ログ
- オブジェクト表
- オブジェクト型
- オブジェクト・ビュー
- 演算子
- 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

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 で作成および操作されますが、スキーマには含まれません。

コンテキスト

ディレクトリ

パラメータ・ファイル (PFILE) およびサーバー・パラメータ・ファイル (SPFILE)

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 9 章～第 18 章のデータベース・オブジェクトを作成する文の項で簡単に定義されています。これらの文は、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、13-2 ページの「[CREATE CLUSTER](#)」を参照してください。

参照： データベース・オブジェクトの概要については、『Oracle9i データベース概要』を参照してください。

ほとんどのデータベース・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、この後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、次に示すとおり名前を付けることができる、または名前を付ける必要のある部分で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およびサブパーティション
- 表に対する整合性制約
- パッケージ・プロシージャ、パッケージ・ストアド・ファンクション、および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

表および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すべての論理属性は同じです。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。

レンジ・メソッドを使用して表または索引をパーティション化する場合、各パーティションのパーティション・キー列の最大値を指定します。リスト・メソッドを使用して表または索引をパーティション化する場合、各パーティションのパーティション・キー列の実際の値を指定します。ハッシュ・メソッドを使用して表または索引をパーティション化する場合、パーティション・キー列に関してシステムが定義するハッシュ・ファンクションに基づいて、Oracle が表の行をパーティションに分割するようにします。コンボジット・パーティション・メソッドを使用して表または索引をパーティション化する場合、パーティションの範囲を指定すると、Oracle はハッシュ・ファンクションに基づいて、各パーティションにある行を1つ以上のハッシュ・サブパーティションに分割します。コンボジット・メソッドを使用してパーティション化された表または索引の各サブパーティションは、同じ論理属性を持ちます。

拡張パーティションおよび拡張サブパーティション名

拡張パーティションおよび拡張サブパーティション名を使用した場合、1つのパーティションまたはサブパーティションのみ、パーティション・レベルおよびサブパーティション・レベルの操作（あるパーティションまたはサブパーティションからのすべての行の削除など）ができます。拡張された名前がない場合、そのような操作には述語（WHERE 句）を指定する必要があります。レンジおよびリスト・パーティション表では、パーティション・レベル操作を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は）、非常に複雑になる可能性があります。ハッシュ・パーティションおよびサブパーティションの場合、述語の使用はより難しくなります。これは、これらのパーティションおよびサブパーティションが、システムが定義するハッシュ・ファンクションに基づいているためです。

拡張パーティション名を使用した場合、パーティションを表のように使用できます。この方法のメリットは、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ことです。このメリットは、レンジ・パーティション表に最も有効です。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。

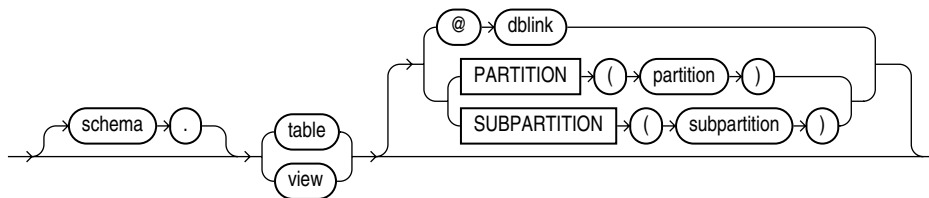
次のデータ操作言語（DML）文では、拡張パーティションまたは拡張サブパーティションの表名を指定できます。

- 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には、ビューを使用することをお勧めします。

構文 拡張パーティション表名および拡張サブパーティション表名を使用する場合の基本的な構文は次のとおりです。

partition_extended_name::=



拡張パーティション表名の制限事項： 現在、拡張パーティション表名および拡張サブ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。

- リモート表は使用できません。拡張パーティション表名または拡張サブ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 を使用し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・パーティションおよびリモート・サブパーティションを使用するには、拡張表名の構文を使用してリモート・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。

- シノニムは使用できません。拡張パーティションまたは拡張サブパーティションは、実表を使用して指定する必要があります。シノニム、ビューまたはその他のオブジェクトは使用できません。

例 次の文の `sales` は、パーティション `sales_q1_2000` を持つパーティション表です。単一パーティション `sales_q1_2000` のビューを作成でき、それを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 Q1_2000_sales AS
  SELECT * FROM sales PARTITION (SALES_Q1_2000);

DELETE FROM Q1_2000_sales WHERE amount_sold < 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・オブジェクトの位置修飾子のネーミング規則
- スキーマ・オブジェクトと修飾子のネーミング計画

スキーマ・オブジェクトのネーミング規則

すべてのデータベース・オブジェクトには、名前があります。SQL 文では、引用識別子または非引用識別子を使用して、オブジェクトの名前を表します。

- 引用識別子は、二重引用符 (") で囲みます。引用識別子を使用してスキーマ・オブジェクトを指定した場合、そのオブジェクトを参照するときは、必ず二重引用符を使用します。
- 非引用識別子の前後に句読点は付けません。

データベース・オブジェクトを指定する場合、引用識別子または非引用識別子を使用できます。ただし、データベース・リンク名には、非引用識別子を使用する必要があります。ユーザー名とパスワードで大 / 小文字を区別するには、引用符を使用しないことをお勧めします。

参照： ユーザー名とパスワードのネーミング規則の詳細は、16-31 ページの「[CREATE USER](#)」を参照してください。

特に指定がないかぎり、次の規則は、引用識別子と非引用識別子の両方に適用されます。

1. 名前は、1 ～ 30 バイトの長さで指定する必要があります。ただし、次の 2 つは例外です。
 - データベースの名前は、8 バイト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 バイトまで指定できます。

2. 非引用識別子に **Oracle** の予約語は使用できません。引用識別子には、予約語を使用できますが、お薦めしません。

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。

注意： 予約語 ROWID は、この規則の例外です。二重引用符で囲んでも、大文字の ROWID を名前として使用することはできません。ただし、単語に 1 つ以上の小文字が含まれる場合は使用できます（たとえば、"Rowid"、"rowid" など）。

参照：

- **Oracle** の予約語のすべてのリストは、[付録 C「Oracle の予約語」](#) を参照してください。
 - 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。
3. **Oracle** の SQL 言語には、特別な意味を持つ文字が含まれています。これらの文字には、データ型、ファンクション名、ダミーのシステム表 DUAL およびキーワード（DIMENSION、SEGMENT、ALLOCATE、DISABLE など、SQL 文中の大文字の単語）が含まれます。これらの文字は予約語ではありません。ただし、**Oracle** は固有の方法でこれらの文字を内部的に使用します。したがって、これらの文字を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した場合、使用している SQL 文が読みにくくなり、予期しない結果になることがあります。

特に、「SYS_」で始まる文字をスキーマ・オブジェクト名として使用しないでください。また、SQL 組込みファンクションの名前を、スキーマ・オブジェクトまたはユーザー定義ファンクションの名前として使用しないでください。

参照：

- 2-2 ページの「[データ型](#)」および 6-2 ページの「[SQL ファンクション](#)」を参照してください。
 - 8-14 ページの「[DUAL 表からの選択](#)」を参照してください。
4. 異なるプラットフォームおよびオペレーティング・システム間では、ASCII 文字を使用することで最適な互換性を得ることができます。データベース名、グローバル・データベース名、データベース・リンク名には ASCII 文字を使用してください。

注意： ユーザー名およびパスワードは、ご使用のプラットフォームに応じて、ASCII または EBCDIC 文字のみでエンコードすることをお勧めします。詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

5. 非引用識別子は、データベース・キャラクタ・セットのアルファベット文字で開始する必要があります。引用識別子の開始文字には、任意の文字を使用できます。
6. 非引用識別子には、データベース・キャラクタ・セットの英数字、アンダースコア (_)、ドル記号 (\$) およびシャープ記号 (#) のみ含めることができます。データベース・リンクの名前には、ピリオド (.) とアットマーク (@) を含めることもできます。\$ と # はできるだけ使用しないでください。

引用識別子には、すべての文字、句読点および空白を使用できます。ただし、引用識別子と非引用識別子のいずれにも、二重引用符は使用できません。

7. ネームスペース内では、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。

次のスキーマ・オブジェクトは、1 つのネームスペースを共有します。

- 表
- ビュー
- 順序
- プライベート・シノニム
- スタンドアロン・プロシージャ
- スタンドアロン・ストアド・ファンクション
- パッケージ
- マテリアライズド・ビュー
- ユーザー定義型

次の各スキーマ・オブジェクトは、固有のネームスペースを持ちます。

- 索引
- 制約
- クラスタ
- データベース・トリガー
- プライベート・データベース・リンク
- ディメンション

表およびビューが同じネームスペースにあるため、同じスキーマの表およびビューが同じ名前を持つことはできません。ただし、表と索引は異なるネームスペースに存在します。このため、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ネームスペースがあります。たとえば、異なるスキーマ内の2つの表は異なるネームスペースに存在し、同じ名前を付けることができます。

次の各非スキーマ・オブジェクトは、固有のネームスペースを持ちます。

- ユーザー・ロール
- パブリック・シノニム
- パブリック・データベース・リンク
- 表領域
- ロールバック・セグメント
- プロファイル
- パラメータ・ファイル (PFILE) およびサーバー・パラメータ・ファイル (SPFILE)

これらのネームスペース内のオブジェクトはスキーマに含まれないため、これらのネームスペースはデータベース全体で使用されます。

8. 非引用識別子は、大 / 小文字が区別されず、すべて大文字として解析されます。引用識別子は、大 / 小文字が区別されます。

名前を二重引用符で囲むことによって、同じネームスペース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。

```
employees  
"employees"  
"Employees"  
"EMPLOYEES"
```

ただし、Oracle は次の名前を同じ名前として解析するため、同じネームスペース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。

```
employees  
EMPLOYEES  
"EMPLOYEES"
```

9. ユーザーまたはパスワードの名前に引用識別子を使用する場合、その名前に小文字を含めることはできません。
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。ただし、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. 引数の数およびデータ型が異なる場合、同じパッケージに含まれるプロシージャやファンクションに同じ名前を付けることができます。異なる引数を持ち、同じ名前のプロシージャやファンクションを同じパッケージ内に複数作成することを、オーバーロードといいます。

スキーマ・オブジェクトのネーミング例

次に、有効なスキーマ・オブジェクト名の例を示します。

```
last_name
horse
hr.hire_date
"EVEN THIS & THAT!"
a_very_long_and_valid_name
```

これらのすべての例は、2-105 ページの「スキーマ・オブジェクトのネーミング規則」に示す規則に従っています。次の例は、30 文字を超えているため、無効となります。

```
a_very_very_long_and_valid_name
```

列別名、表別名、ユーザー名およびパスワードは、オブジェクトまたはオブジェクトの部分ではありませんが、特に指定がないかぎり、同様にこれらのネーミング規則に従う必要があります。

スキーマ・オブジェクト名のネーミング計画

オブジェクトとその部分に名前を付ける場合に有効な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）を使用します。
- 一貫したネーミング規則を使用します。
- 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、同一の名前を使用します。

オブジェクトに名前を付ける場合は、短くて簡単な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、わかりやすい名前にしてください。これは、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ためです。
payment_due_date のかわりに pmd といふ名前を使用すると、10 年後の担当者は表の列の理解に苦労することになります。

一貫したネーミング規則を使用すると、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、employees サンプル表と departments サンプル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。

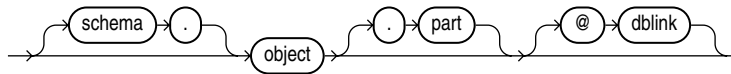
スキーマ・オブジェクトの構文および SQL 文の構成要素

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変換する方法
- 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

次に、オブジェクトやそれらの部分を参照するための一般的な構文を示します。

object_part::=



それぞれの意味は、次のとおりです。

- *object* は、オブジェクトの名前です。
- *schema* は、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必要です。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・オブジェクトのみが *schema* で修飾できます。スキーマ・オブジェクトについては、2-107 ページの規則 7 を参照してください。2-107 ページの規則 7 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」で修飾できます。この場合、引用符が必要です。）

- *part* は、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とは限りません。
- *dblink* は、Oracle の分散オプションを使用している場合にのみ適用されます。オブジェクトを含む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際の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。ただし、通常は入れません。

Oracle によるスキーマ・オブジェクト参照の変換方法

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、該当するネームスペース内でそのオブジェクトの位置を確認します。そのオブジェクトの位置を確認してから、そのオブジェクトに対して文の実行を実行します。指定した名前のオブジェクトが適切なネームスペース内に存在しない場合、Oracle はエラーを戻します。

次の例で、Oracle が SQL 文内のオブジェクト参照を変換する方法について説明します。名前 `departments` 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 departments VALUES (  
    280, 'ENTERTAINMENT_CLERK', 206, 1700);
```

文のコンテキストに基づいて、Oracle は、`departments` 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ネームスペースを考慮する前に、そのユーザーのスキーマ内のネームスペースからオブジェクト参照を変換しようとします。この例では、Oracle は次の方法で名前 `departments` を変換しようとします。

1. Oracle は、最初に、表、ビューおよびプライベート・シノニムを含む文を発行したユーザーのスキーマ内のネームスペースで、オブジェクトの位置を確認しようとします。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の位置を確認します。このオブジェクトは、ユーザー自身のスキーマ、他のスキーマ、または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場合もあります。その場合、Oracle はそのシノニムが表すオブジェクトの位置を確認します。
2. オブジェクトがネームスペース内に存在する場合、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。この例では、Oracle はデータ行を `departments` 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 はエラーを戻します。この場合、`departments` 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・シノニムである必要があります。`departments` が順序である場合、Oracle はエラーを戻します。
3. 前述の処理で検索されたネームスペースにオブジェクトが存在しない場合、Oracle はパブリック・シノニムを含むネームスペースを検索します。オブジェクトがそのネームスペースに存在する場合、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。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 はエラーを戻します。この例では、`departments` が順序のパブリック・シノニムである場合、Oracle はエラーを戻します。

注意： オブジェクト型のパブリック・シノニムに、依存表またはユーザー定義型がある場合は、依存オブジェクトと同じスキーマに、シノニムと同じ名前で作成することはできません。

オブジェクト型のパブリック・シノニムに、依存表またはユーザー定義型がない場合は、依存オブジェクトと同じスキーマに、依存オブジェクトと同じ名前で、オブジェクトを作成できます。すべての依存オブジェクトが無効になり、次のアクセス時に妥当性チェックが再実行されます。

他のスキーマ内のオブジェクトの参照

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、サンプル・スキーマ `hr` 内の `employees` 表を削除します。

`DROP TABLE hr.employees`

リモート・データベース内のオブジェクトの参照

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、オブジェクト名の後に、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、そこにある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・リンクの作成

13-35 ページの「[CREATE DATABASE LINK](#)」を使用して、データベース・リンクを作成します。この文では、データベース・リンクに関する次の情報を指定できます。

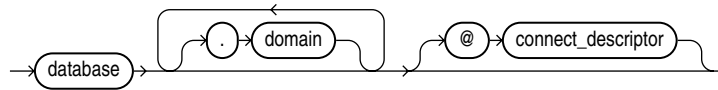
- データベース・リンク名
- リモート・データベースにアクセスするためのデータベース接続文字列
- リモート・データベースに接続するためのユーザー名およびパスワード

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、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。

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のそのデータベースの位置に一致している必要があります。次に、データベース・リンク名の書式を示します。

dblink::=



それぞれの意味は、次のとおりです。

- **database** には、データベース・リンクが接続するリモート・データベースのグローバル名の名前の部分を指定します。このグローバル名は、リモート・データベースのデータ・ディクショナリに格納されます。この名前は、GLOBAL_NAME ビューで見ることができます。
- **domain** には、データベース・リンクが接続するリモート・データベースのグローバル名のドメイン部分を指定します。データベース・リンクの名前に **domain** 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に、データベース・リンク名を付加します。
- **connect_descriptor** によって、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場合、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 Real Application Clusters 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。

参照：『Oracle9i Net Services 管理者ガイド』を参照してください。

ユーザー名およびパスワード リモート・データベースに接続するために、ユーザー名およびパスワードを使用します。データベース・リンクでは、ユーザー名および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、Oracle Net がリモート・データベースにアクセスするために使用する仕様です。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 Oracle Net のドキュメント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・リンクの参照

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にのみ利用できます。データベース・リンクを含む SQL 文の発行時に、次のいずれかの方法でデータベース・リンク名を指定します。

- `complete` は、データ・ディクショナリ内に格納される、`database`、`domain`、およびオプションの `connect_descriptor` コンポーネントを含む完全なデータベース・リンク名を指定します。
- `partial` は、`database`、およびオプションの `connect_descriptor` コンポーネントを含むが、`domain` 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とおり、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・リンクを検索します。必要に応じて、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）のユーザー名および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、現在のユーザー名および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ている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、`GLOBAL_NAMES` パラメータの値が `true` の場合は、Oracle は、データベース・リンク名の `database.domain` 部分がリモート・データベースの完全なグローバル名に一致しているかどうかを確認します。この条件が満たされている場合、Oracle は手順 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、Oracle はエラーを戻します。

4. データベース文字列、ユーザー名およびパスワードを使用した接続が成功した場合、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て、このとき、この項の前半で説明した、オブジェクト参照を変換するための規則、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・データベースの完全なグローバル名が、データベース・リンクの `database.domain` 部分と一致する必要があるという要件を無効にするには、初期化パラメータ `GLOBAL_NAMES` か、`ALTER SYSTEM` または `ALTER SESSION` 文の `GLOBAL_NAMES` パラメータに `false` を設定します。

参照： リモート・データベースの名前の変換の詳細は、『Oracle9i データベース管理者ガイド』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 文のオブジェクト型の属性とメソッドを参照するには、参照を表の別名で完全に修飾する必要があります。次の例では、`cust_address_typ` 型、および `cust_address_typ` に基づく `cust_address` 列を持つ表 `customers` を含むサンプル・スキーマ `oe` について考えます。

```
CREATE TYPE cust_address_typ AS OBJECT
( street_address    VARCHAR2(40)
, postal_code       VARCHAR2(10)
, city              VARCHAR2(30)
, state_province    VARCHAR2(10)
, country_id        CHAR(2)
);
/

CREATE TABLE customers
( customer_id        NUMBER(6)
, cust_first_name    VARCHAR2(20) CONSTRAINT cust_fname_nn NOT NULL
, cust_last_name     VARCHAR2(20) CONSTRAINT cust_lname_nn NOT NULL
, cust_address       cust_address_typ
.
.
.
```

次に示すとおり、SQL 文では、`postal_code` 属性への参照は表別名を使用して完全に修飾する必要があります。

```
SELECT c.cust_address.postal_code FROM customers c;

UPDATE customers c SET c.cust_address.postal_code = 'GU13 BE5'
WHERE c.cust_address.city = 'Fleet';
```

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、空のカッコを付ける必要があります。たとえば、サンプル・スキーマ `oe` には、メンバー・ファンクション `getCatalogName` を含む `catalog_typ` に基づくオブジェクト表 `categories_tab` が含まれます。SQL 文でこのメソッドをコールするには、次の例のように、空のカッコを付ける必要があります。

```
SELECT c.getCatalogName() FROM categories_tab c
WHERE category_id = 90;
```

参照： ユーザー定義データ型の詳細は、『Oracle9i データベース概要』を参照してください。

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します。

この章では、次の内容を説明します。

- [SQL 演算子](#)
- [算術演算子](#)
- [連結演算子](#)
- [集合演算子](#)
- [ユーザー定義演算子](#)

この章では、非論理（非ブール）演算子について説明します。これらの演算子は、問合せまたは副問合せ内で WHERE または HAVING 句の条件として、単独では使用できません。条件として使用できる論理演算子の詳細は、[第 5 章「条件」](#)を参照してください。

SQL 演算子

演算子は、**オペランド**または**引数**と呼ばれる個々のデータ項目を操作します。演算子は、特殊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 (*) で表します。

注意： Oracle Text がインストールされている場合は、Oracle Text 問合せで、この製品に含まれる SCORE 演算子を使用できます。この演算子の詳細は、『Oracle Text リファレンス』を参照してください。

単項演算子およびバイナリ演算子

一般に、演算子には次の 2 つのクラスがあります。

- 単項: 単項演算子は、1 つのみのオペランドについて操作します。単項演算子の書式は次のとおりです。

operator operand

- バイナリ: バイナリ演算子は、2 つのオペランドについて操作します。バイナリ演算子の書式は次のとおりです。

operand1 operator operand2

この他、特別な書式を持ち、3 つ以上のオペランドについて操作可能な演算子もあります。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 になります。この規則に従わない唯一の演算子が連結演算子 (||) です。

演算子の優先順位

優先順位とは、同じ式の中の異なる演算子を Oracle が評価する順序を意味します。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の優先順位は同じです。

表 3-1 SQL 演算子の優先順位

演算子	操作
+, - (単項演算子)、PRIOR	同一、否定、階層内の位置
*, /	乗算、除算
+, - (バイナリ演算子)、	加算、減算、連結
SQL 条件は、SQL 演算子の後で評価されます。	5-4 ページの「 条件の優先順位 」を参照してください。

優先順位の例 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上書きできます。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、UNION ALL、INTERSECT および MINUS）もサポートされます。集合演算子によって結合されるのは、問合せによって戻される行の集まりで、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。

参照：

- 3-6 ページの「[集合演算子](#)」を参照してください。
- 階層問合せのみで使用する PRIOR 演算子については、8-3 ページの「[階層問合せ](#)」を参照してください。

算術演算子

算術演算子を式の中で使用することによって、数値を否定（正負を反転）、加算、減算、乗算および除算できます。演算の結果も数値になります。これらの演算子の中には、日付算術で使用するものもあります。[表 3-2](#) に、算術演算子を示します。

表 3-2 算術演算子

演算子	用途	例
+ -	式の正負を示す場合、これらは単項演算子です。	SELECT * FROM order_items WHERE quantity = -1; SELECT * FROM employees WHERE -salary < 0;
	加算、減算を行う場合、これらはバイナリ演算子です。	SELECT hire_date FROM employees WHERE SYSDATE - hire_date > 365;
* /	乗算、除算を行います。これらはバイナリ演算子です。	UPDATE employees SET salary = salary * 1.1;

二重否定や負の数の減算を表現する場合に、算術式で、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場合に使用します。連続した負の符号は、空白またはカッコで区切ってください。

参照： SQL 文中のコメントの詳細は、2-84 ページの「[コメント](#)」を参照してください。

連結演算子

連結演算子は、文字列および CLOB データを操作する場合に使用します。[表 3-3](#) に、連結演算子を示します。

表 3-3 連結演算子

演算子	用途	例
	文字列および CLOB データを連結します。	SELECT 'Name is ' last_name FROM employees;

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 データ型の場合、結果は CHAR データ型の文字列になり、その最大文字数は 2000 です。どちらかの文字列が VARCHAR2 データ型の場合、結果は VARCHAR2 データ型の文字列になり、最大文字数は 4000 です。どちらかの引数が CLOB データ型の場合、結果は、一時 CLOB になります。データ型が文字列型か CLOB 型かにかかわらず、後続空白は連結後も文字列に残ります。

多くのプラットフォームでは、連結演算子は、表 3-3 に示すとおり 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、移動先の 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・システムまたはネットワーク・ユーティリティによる変換の制御が困難または不可能である場合に備えて、Oracle では、垂直バー演算子にかわるものとして CONCAT 文字ファンクションが提供されています。異なるキャラクタ・セットを持つ環境間でアプリケーションを移動する場合は、この文字ファンクションを使用することをお勧めします。

Oracle は、長さが 0（ゼロ）の文字列を NULL として処理しますが、長さが 0（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のみです。ただし、この処理は Oracle の今後のバージョンでも継続されるとはかぎりません。NULL になる可能性がある式を連結する場合は、NVL ファンクションを使用して、その式を長さが 0（ゼロ）の文字列に明示的に変換してください。

参照：

- CHAR データ型と VARCHAR2 データ型の違いの詳細は、2-9 ページの「[文字データ型](#)」を参照してください。
- CLOB データ型の詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。
- 6-33 ページの「[CONCAT](#)」ファンクションおよび 6-107 ページの「[NVL](#)」ファンクションを参照してください。

例 次の例では、CHAR 列および VARCHAR2 列を持つ表を作成し、後続空白のある値とない値を挿入してから、これらの値を選択し、連結します。なお、CHAR 列および VARCHAR2 列では、ともに後続空白が保存されます。

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

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

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

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

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 3-4 に、SQL の集合演算子を示します。これらの演算子の詳細および制限については、8-7 ページの「[UNION \[ALL\]](#)、[INTERSECT](#) および [MINUS 演算子](#)」を参照してください。

表 3-4 集合演算子

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

ユーザー定義演算子

ユーザー定義演算子は、組込み演算子のように、一連のオペランドを入力として受け取り、結果を返します。ユーザー定義演算子は、ユーザーが CREATE OPERATOR 文で作成し、名前で識別されます。これらは、表、ビュー、型およびスタンドアロン・ファンクションとして同じネームスペースに存在します。

新規の演算子を定義すると、他の組込み演算子のように SQL 文で使用できます。たとえば、SELECT 文の SELECT 構文のリスト、WHERE 句の条件、ORDER BY 句および GROUP BY 句でユーザー定義演算子を使用できます。ただし、これはユーザー定義オブジェクトであるため、演算子に対する EXECUTE 権限が必要です。

たとえば、指定したキーワードが行に含まれる場合に、入力としてテキスト列およびキーワードを受け取り、1 を返す INCLUDES 演算子を定義する場合は、次のような SQL 問合せを作成します。

```
SELECT * FROM product_descriptions
WHERE includes (translated_description, 'Oracle and UNIX') = 1;
```

参照： ユーザー定義演算子の詳細は、14-40 ページの「[CREATE OPERATOR](#)」および『Oracle9i Data Cartridge Developer's Guide』を参照してください。

4

式

この章では、値、演算子およびファンクションを式の中で組み合わせて使用方法について説明します。

この章では、次の内容を説明します。

- SQL 式
- 単純式
- 複合式
- CASE 式
- CURSOR 式
- 日時式
- ファンクション式
- 期間式
- オブジェクト・アクセス式
- スカラー副問合せ式
- 型コンストラクタ式
- 変数式
- 式のリスト

SQL 式

式は、1 つ以上の値、演算子、および値を評価する SQL ファンクションの組合せです。一般に、式のデータ型は、そのコンポーネントのデータ型になります。

次の単純式は、値が 4 になり、データ型は NUMBER（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、ファンクションと演算子を使用した複雑な式です。この式は、現在の日付に 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 文の SELECT 構文のリスト
- WHERE 句および HAVING 句の条件
- CONNECT BY 句、START WITH 句および ORDER BY 句
- INSERT 文の VALUES 句
- UPDATE 文の 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。

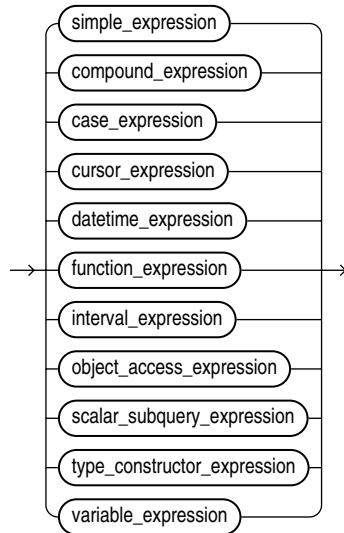
```
SET last_name = 'Smith';
```

この SET 句では、引用符で囲まれた文字列 'Smith' のかわりに、INITCAP(last_name) を使用しています。

```
SET last_name = INITCAP(last_name);
```

次の構文に示すとおり、式にはいくつかの書式があります。

expr::=



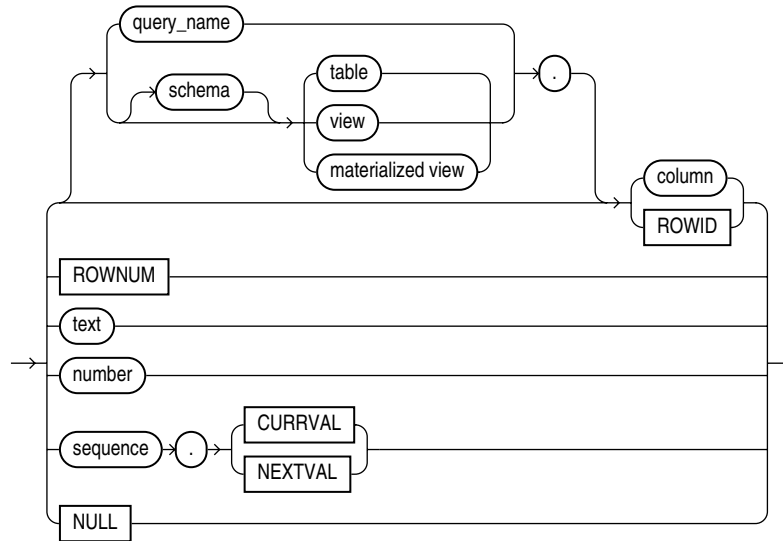
Oracle は、すべての SQL 文のすべての部分で、式のすべての書式を受け入れるわけではありません。このマニュアルの他の箇所で、条件、SQL ファンクションまたは SQL 文に `expr` が示されている場合は、必ず適切な式の表記法を使用してください。次の項では、いくつかの例をあげて、様々な式の書式を説明します。

参照： 文に指定する式の制限については、[第 9 章～第 18 章](#)の個々の SQL 文の説明を参照してください。

単純式

単純式は、列、疑似列、定数、順序番号または NULL を指定します。

simple_expression::=



スキーマは各ユーザー用の他に、"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、ビューまたはマテリアライズド・ビューのパブリック・シノニムを修飾する必要があります。"PUBLIC"でのパブリック・シノニムの修飾は、データ操作言語（DML）文でのみサポートされています。データ定義言語（DDL）文ではサポートされていません。

疑似列は、LEVEL、ROWID または ROWNUM のいずれかです。疑似列は、表でのみ使用でき、ビューまたはマテリアライズド・ビューでは使用できません。NCHAR および NVARCHAR2 は、有効な疑似列データ型ではありません。

参照：

- 疑似列の詳細は、2-77 ページの「[疑似列](#)」を参照してください。
- `query_name` については、18-10 ページの「[subquery_factoring_clause](#)」を参照してください。

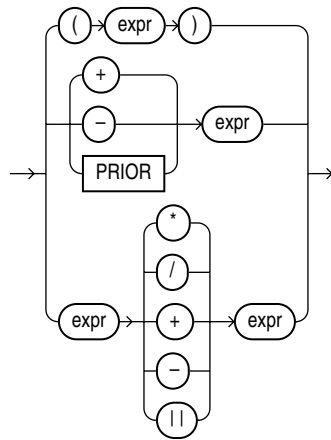
有効な単純式の例を次に示します。

```
employees.last_name
'this is a text string'
10
N'this is an NCHAR string'
```

複合式

複合式は、その他の式の組合せを指定します。

compound_expression::=



注意： 組込みファンクションを式として使用できます (4-11 ページの「[ファンクション式](#)」を参照してください)。ただし、複合式では、ファンクション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。たとえば、LENGTH ファンクションは集計ファンクション内では使用できません。

PRIOR 演算子は、階層問合せの CONNECT BY 句で使用されます。

参照： 3-2 ページの「[演算子の優先順位](#)」および 8-3 ページの「[階層問合せ](#)」を参照してください。

有効な複合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'))
```

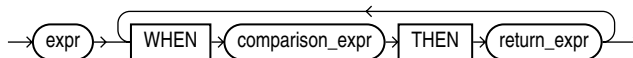
CASE 式

CASE 式を使用すると、プロシージャを起動せずに、SQL 文で IF ... THEN ... ELSE 論理を使用できます。構文は次のとおりです。

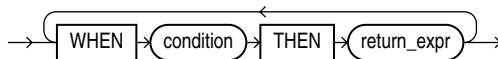
case_expression::=



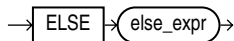
simple_case_expression::=



searched_case_expression::=



else_clause::=



単純 CASE 式では、Oracle は、*expr* と *comparison_expr* が一致する最初の WHEN ... THEN の組合せを検索し、*return_expr* を戻します。WHEN ... THEN の組合せが条件に一致せず、ELSE 句が存在する場合、Oracle は *else_expr* を戻します。それ以外の場合、Oracle は NULL を戻します。*return_expr* および *else_expr* には、リテラル NULL を指定できません。

すべての式 (*expr*、*comparison_expr* および *return_expr*) は、同じデータ型 (CHAR、VARCHAR2、NCHAR または NVARCHAR2) である必要があります。

検索 CASE 式では、Oracle は、*condition* が真である項目を左から右へ検索し、*return_expr* を戻します。真である *condition* がなく、ELSE 句が存在する場合、Oracle は *else_expr* を戻します。それ以外の場合、Oracle は NULL を戻します。

注意： CASE 式の引数の最大数は 255 です。WHEN ... THEN の各組は、2 つの引数として数えます。128 種類の制限を超えないようにするために、CASE 式をネストできます。つまり、`return_expr` 自体を CASE 式にできます。

参照：

- その他の CASE 式の書式については、6-31 ページの「[COALESCE](#)」および 6-104 ページの「[NULLIF](#)」を参照してください。
- CASE 式の様々な書式を使用した例は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

単純 CASE 式の例 次の文は、サンプル表 `oe.customers` のそれぞれの顧客について、クレジット利用限度額を、100 ドルの場合は「Low」、5,000 ドルの場合は「High」、それ以外の場合は「Medium」で表示します。

```
SELECT cust_last_name,
       CASE credit_limit WHEN 100 THEN 'Low'
       WHEN 5000 THEN 'High'
       ELSE 'Medium' END
FROM customers;
```

CUST_LAST_NAME	CASECR
...	
Bogart	Medium
Nolte	Medium
Loren	Medium
Gueney	Medium

検索 CASE 式の例 次の文は、2,000 ドルを最少額の給与として、サンプル表 `oe.employees` の従業員の給与の平均を検出します。

```
SELECT AVG(CASE WHEN e.salary > 2000 THEN e.salary
               ELSE 2000 END) "Average Salary" from employees e;
```

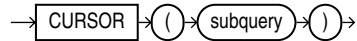
Average Salary

6461.68224

CURSOR 式

CURSOR 式は、ネステッド・カーソルを戻します。この式の書式は、PL/SQL の REF CURSOR と同じで、REF CURSOR 引数としてファンクションに渡すことができます。

cursor_expression::=



カーソル式が評価されるときに、ネステッド・カーソルは暗黙的にオープンされます。たとえば、カーソル式が SELECT 構文のリストにある場合、問合せによってフェッチされた各行に対して、ネステッド・カーソルがオープンされます。ネステッド・カーソルは、次の場合にのみクローズされます。

- 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 式の制限事項：

- 囲まれる文が SELECT 文以外の文である場合、ネステッド・カーソルは、プロシージャの REF CURSOR 引数としてのみ表示されます。
- 囲まれる文が SELECT 文である場合、ネステッド・カーソルは、問合せ指定の一番外側の SELECT 構文のリストまたは別のネステッド・カーソルの一番外側の SELECT 構文のリストに表示されます。
- ネステッド・カーソルはビューに表示できません。
- ネステッド・カーソルに対して、BIND 操作および EXECUTE 操作は実行できません。

例 問合せの SELECT 構文のリストでの CURSOR 式の使用方法について、次に例を示します。

```

SELECT department_name, CURSOR(SELECT salary, commission_pct
    FROM employees e
    WHERE e.department_id = d.department_id)
FROM departments d;
  
```

ファンクションの引数としての CURSOR の使用方法について、次に例を示します。例では、まず、サンプル・スキーマ OE に REF CURSOR 引数を許可するファンクションを作成します。

```
CREATE FUNCTION f (cur SYS_REFCURSOR, mgr_hiredate DATE)
RETURN NUMBER IS
    emp_hiredate DATE;
    before number :=0;
    after number:=0;
begin
    loop
        fetch cur into emp_hiredate;
        exit when cur%NOTFOUND;
        if emp_hiredate > mgr_hiredate then
            after:=after+1;
        else
            before:=before+1;
        end if;
    end loop;
    close cur;
    if before > after then
        return 1;
    else
        return 0;
    end if;
end;
/
```

ファンクションには、カーソルおよび日付を指定できます。ファンクションは、カーソルが日付セットを戻す問合せであることを想定します。次の問合せでは、ファンクションを使用して、サンプル表 employees から、ほとんどの従業員がマネージャよりも前に雇用されているマネージャを検索します。

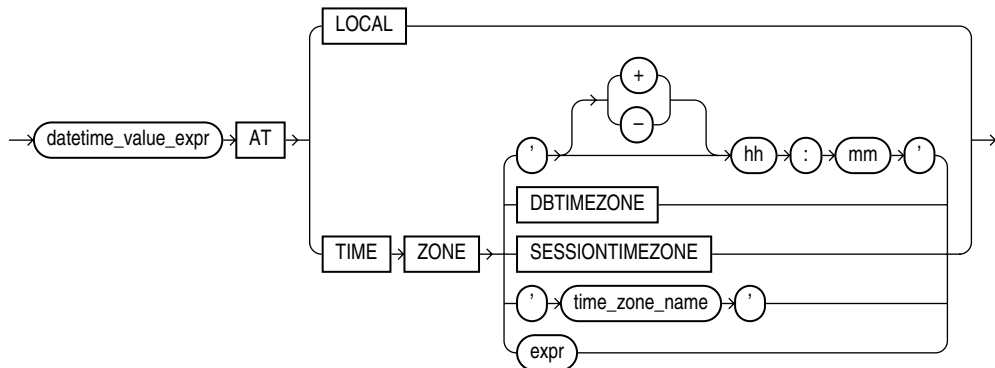
```
SELECT e1.last_name FROM employees e1
WHERE f(
    CURSOR(SELECT e2.hire_date FROM employees e2
    WHERE e1.employee_id = e2.manager_id),
    e1.hire_date) = 1;
```

```
LAST_NAME
-----
De Haan
Mourgos
Cambrault
Zlotkey
Higgins
```

日時式

日時式は、日時データ型の値を戻します。

datetime_expression::=



datetime_value_expression は、日時の値を戻す日時列または複合式です。2-23 ページの表 2-5 で定義される規則に従って、日時および期間を組み合わせたことができます。日時の値を戻す 3 つの組合せは、日時式で有効です。

AT LOCAL を指定すると、Oracle は現行のセッションのタイムゾーンを使用します。

AT TIME ZONE の設定は、次のように解析されます。

- 文字列 '(+|-)HH:MM': UTC のオフセットとしてタイムゾーンを指定します。
- DBTIMEZONE: Oracle は、データベースの作成中に（明示的またはデフォルトで）構築されたデータベース・タイムゾーンを使用します。
- SESSIONTIMEZONE: Oracle は、デフォルトで構築されたセッション・タイムゾーン、または最新の ALTER SESSION 文で構築されたセッション・タイムゾーンを使用します。
- *time_zone_name*: Oracle は、*time_zone_name* で指定されたタイムゾーンの *datetime_value_expr* を戻します。有効なタイムゾーン名を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューに問合せを実行してください。

参照： 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

- *expr*: *expr* が有効なタイムゾーン書式で文字列を戻す場合、Oracle は、そのタイムゾーンで入力を戻します。そうでない場合は、エラーが戻ります。

例 次の例は、タイムゾーンの日時の値を別のタイムゾーンに変換します。

```
SELECT FROM_TZ(CAST(TO_DATE('1999-12-01 11:00:00',
    'YYYY-MM-DD HH:MI:SS') AS TIMESTAMP), 'America/New_York')
    AT TIME ZONE 'America/Los_Angeles' "West Coast Time"
FROM DUAL;
```

West Coast Time

01-DEC-99 08.00.00.000000 AM AMERICA/LOS_ANGELES

ファンクション式

組込み SQL ファンクションまたはユーザー定義ファンクションを式として使用できます。有効な組込みファンクション式の例を次に示します。

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

参照： 組込みファンクションの詳細は、6-2 ページの「[SQL ファンクション](#)」および 6-7 ページの「[集計ファンクション](#)」を参照してください。

ユーザー定義ファンクション式は、次のものへのコールを指定します。

- オラクル社が提供するパッケージにあるファンクション（『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照）
- ユーザー定義パッケージにあるファンクションまたはスタンドアロン・ユーザー定義ファンクション（6-211 ページの「[ユーザー定義ファンクション](#)」を参照）
- ユーザー定義ファンクションおよび演算子（14-40 ページの「[CREATE OPERATOR](#)」、13-49 ページの「[CREATE FUNCTION](#)」および『Oracle9i Data Cartridge Developer's Guide』を参照）

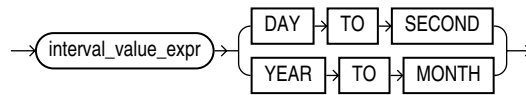
有効なユーザー定義ファンクション式の例を次に示します。

```
circle_area(radius)
payroll.tax_rate(empno)
hr.employees.comm_pct(dependents, empno)@remote
DBMS_LOB.getlength(column_name)
my_function(DISTINCT a_column)
```

期間式

期間式は、INTERVAL YEAR TO MONTH または INTERVAL DAY TO SECOND の値を戻します。

interval_expression ::=



`interval_value_expression` は、期間の値を戻す期間列または複合式の値です。2-23 ページの表 2-5 で定義される規則に従って、日時および時間を組み合わせることができます。期間の値を戻す 6 つの組合せは、期間式で有効です。

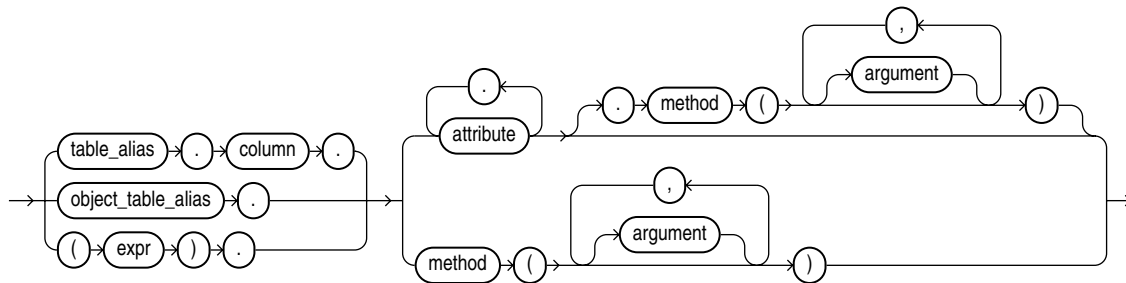
たとえば、次の文は、システム・タイムスタンプ（日時の値）からサンプル表 `orders` の `order_date` 列の値（別の日時の値）を減算して、期間値の式を戻します。

```
SELECT (SYSTIMESTAMP - order_date) DAY TO SECOND from orders;
```

オブジェクト・アクセス式

オブジェクト・アクセス式は、属性の参照およびメソッドの起動を指定します。

object_access_expression ::=



column パラメータはオブジェクトまたは REF 列です。expr を指定する場合、オブジェクト型に変換する必要があります。

ある型のメンバー・ファンクションが SQL 文のコンテキストでコールされると、SELF 引数が NULL の場合に、Oracle は NULL を戻し、ファンクションは起動されません。

例 次の例では、サンプル・オブジェクト型 `oe.order_item_typ` に基づく表を作成し、オブジェクト列属性から更新および検索する方法を示します。

```
CREATE TABLE short_orders (  
    sales_rep VARCHAR2(25), item order_item_typ);  
  
UPDATE short_orders s SET sales_rep = 'Unassigned';  
  
SELECT o.item.line_item_id, o.item.quantity FROM short_orders o;
```

スカラー副問合せ式

スカラー副問合せ式は、1つの行から1つの列値のみを戻す副問合せです。スカラー副問合せ式の値は、副問合せの **SELECT** 構文リスト項目の値です。副問合せが0行を戻す場合、スカラー副問合せ式の値は **NULL** です。副問合せが2つ以上の行を戻す場合、**Oracle** はエラーを戻します。

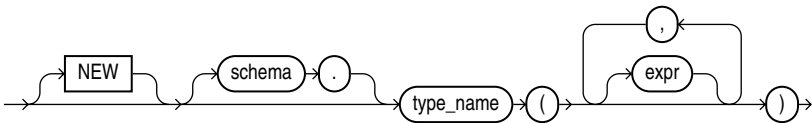
スカラー副問合せ式は、式 (*expr*) をコールするほとんどの構文で使用できます。ただし、次の場所では、スカラー副問合せは無効です。

- 列のデフォルト値
- クラスタのハッシュ式
- DML 文 **RETURNING** 句
- ファンクション索引の基礎
- **CHECK** 制約
- **CASE** 式の **WHEN** 条件
- **GROUP BY** 句および **HAVING** 句
- **START WITH** 句および **CONNECT BY** 句
- **CREATE PROFILE** などの問合せに関連しない文

型コンストラクタ式

型コンストラクタ式は、型コンストラクタへのコールを指定します。型コンストラクタの引数は、任意の式です。

type_constructor_expression::=



NEW キーワードによって、コレクション型ではなくオブジェクトに、コンストラクタが適用されます。これによって、適切なコンストラクタを起動して新しいオブジェクトを作成するように Oracle に指示します。NEW キーワードはオプションですが、指定することをお勧めします。

type_name が**オブジェクト型**の場合、最初の引数の値の型がオブジェクト型の最初の属性と一致する値を取り、2 番目の引数の値の型がオブジェクト型の 2 番目の属性と一致するというように、式は順序付けられたリストになっている必要があります。コンストラクタの引数の合計数は、オブジェクト型の属性の合計数と一致する必要があります。

type_name が**VARRAY 型**またはネストした表型の場合、式のリストには 0 個以上の引数を含めることができます。引数が 0 個の場合は、空コレクション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、型がコレクション型の要素型である要素値に対応します。

型コンストラクタは、ファンクションが起動されるすべての場所で起動できます。型コンストラクタにも、引数の最大数など同様の制限事項があります。

参照： 型コンストラクタの詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

式の例 次の例では、サンプル・スキーマ oe の cust_address_typ 型を使用して、型コンストラクタへのコールに含まれる式の使用方法を示します（PL/SQL 文はイタリック体の部分です）。

```
CREATE TYPE address_book_t AS TABLE OF cust_address_typ;
DECLARE
    myaddr cust_address_typ := cust_address_typ(
        '500 Oracle Parkway', 94065, 'Redwood Shores', 'CA','USA');
    alladdr address_book_t := address_book_t();
BEGIN
    INSERT INTO customers VALUES (
        666999, 'Smith', 'Joe', myaddr, NULL, NULL, NULL, NULL,
        NULL, NULL, NULL);
END;
/
```

副問合せ例 次の例では、サンプル・スキーマ `oe` の `warehouse_typ` 型を使用して、型コ
ンストラクタへのコールに含まれる副問合せの使用方法を示します。

```
CREATE TABLE warehouse_tab OF warehouse_typ;

INSERT INTO warehouse_tab
VALUES (warehouse_typ(101, 'new_wh', 201));

CREATE TYPE facility_typ AS OBJECT (
    facility_id NUMBER,
    warehouse_ref REF warehouse_typ);

CREATE TABLE buildings (b_id NUMBER, building facility_typ);

INSERT INTO buildings VALUES (10, facility_typ(102,
(SELECT REF(w) FROM warehouse_tab w
WHERE warehouse_name = 'new_wh')));

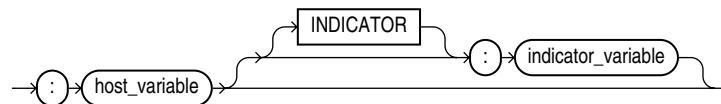
SELECT b.b_id, b.building.facility_id "FAC_ID",
DEREF(b.building.warehouse_ref) "WH" FROM buildings b;

      B_ID      FAC_ID WH(WAREHOUSE_ID, WAREHOUSE_NAME, LOCATION_ID)
-----
      10          102 WAREHOUSE_TYP(101, 'new_wh', 201)
```

変数式

変数式は、オプションのインジケータ変数を持つホスト変数を指定します。この書式の式
は、埋込み SQL 文または Oracle Call Interface (OCI) プログラムで処理される SQL 文のみ
で指定できます。

variable_expression::=



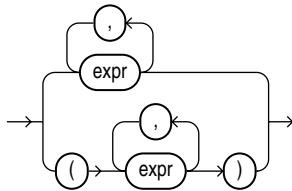
有効な変数式の例を次に示します。

```
:employee_name INDICATOR :employee_name_indicator_var
:department_location
```

式のリスト

式のリストは、その他の式の組合せです。

expression_list::=



式のリストは、比較条件、メンバーシップ条件、および問合せと副問合せの GROUP BY 句に指定できます。

比較条件およびメンバーシップ条件は、WHERE 句の条件として指定します。これらの条件には、カンマで区切られた 1 つ以上の式、または 1 つ以上の式の集合（各集合には、カンマで区切られた式が 1 つ以上含まれる）を含めることができます。式の集合が複数存在する場合は、次のように指定します。

- 各集合をカッコで区切ります。
- 各集合に含まれる式の数と同じにします。
- 各集合内の式の数、比較条件の演算子の前またはメンバーシップ条件の IN キーワードの前にある式の数と同じにします。

カンマで区切られた式のリストには、最大 1000 個の式を指定できます。カンマで区切られた式の集合のリストには、任意数の式の集合を含めることができますが、各集合に指定できる式は最大 1000 個です。

次に、条件に含める有効な式のリストを示します。

```
(10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
(( 'Guy', 'Himuro', 'GHIMURO'), ('Karen', 'Colmenares', 'KCOLMENA'))
```

3 番目の例の場合、各集合の式の数、条件の最初の部分の式の数と同じである必要があります。次に例を示します。

```
SELECT * FROM employees
WHERE (first_name, last_name, email) IN
(( 'Guy', 'Himuro', 'GHIMURO'), ('Karen', 'Colmenares', 'KCOLMENA'))
```

参照： 5-4 ページの「[比較条件](#)」および 5-10 ページの「[メンバーシップ条件](#)」を参照してください。

単純な GROUP BY 句では、式のリストの書式として、次のいずれかを使用できます。

```
SELECT department_id, MIN(salary), MAX(salary)
FROM employees
GROUP BY department_id, salary;
```

```
SELECT department_id, MIN(salary), MAX(salary)
FROM employees
GROUP BY (department_id, salary);
```

ROLLUP、CUBE、および GROUP BY 句の GROUPING SETS 句では、個々の式と同じリストにある式の集合を組み合わせることができます。次に、1 つの SQL 文に含まれている有効なグルーピング・セットと式のリストを示します。

```
SELECT
prod_category, prod_subcategory, country_id, cust_city, count(*)
FROM products, sales, customers
WHERE sales.prod_id = products.prod_id
AND sales.cust_id=customers.cust_id
AND sales.time_id = '01-oct-00'
AND customers.cust_year_of_birth BETWEEN 1960 and 1970
GROUP BY GROUPING SETS
(
(prod_category, prod_subcategory, country_id, cust_city),
(prod_category, prod_subcategory, country_id),
(prod_category, prod_subcategory),
country_id
);
```

参照： 18-4 ページの「[SELECT](#)」を参照してください。

5

条件

条件は、1 つ以上の式と論理（ブール）演算子の組合せで指定し、TRUE、FALSE または UNKNOWN のいずれかの値を返します。

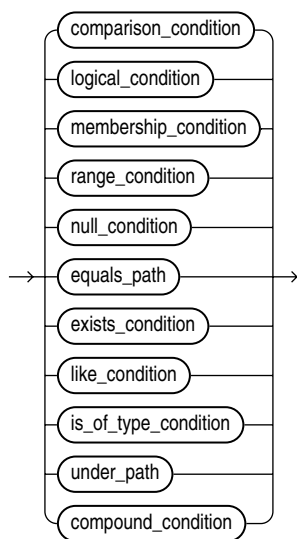
この章では、次の内容を説明します。

- [SQL 条件](#)
- [比較条件](#)
- [論理条件](#)
- [メンバーシップ条件](#)
- [範囲条件](#)
- [NULL 条件](#)
- [EQUALS_PATH](#)
- [EXISTS 条件](#)
- [LIKE 条件](#)
- [IS OF type 条件](#)
- [UNDER_PATH](#)
- [複合条件](#)

SQL 条件

条件には、次の構文で示すとおり、複数の書式があります。

condition::=



注意： Oracle Text がインストールされている場合、CONTAINS、CATSEARCH および MATCHES など、この製品に含まれる組込み条件を使用できます。Oracle Text 要素の詳細は、『Oracle Text リファレンス』を参照してください。

次の項では、様々な書式の条件を説明します。SQL 文に *condition* が含まれる場合は、適切な条件構文を使用する必要があります。

条件は、次の文の WHERE 句で使用できます。

- DELETE
- SELECT
- UPDATE

また、SELECT 文の次の句で 사용할 수도 있습니다。

- WHERE
- START WITH
- CONNECT BY
- HAVING

조건은, 논리 데이터형이라고도 할 수 있습니다.ただし、Oracle で、正式にこのようなデータ型をサポートしているわけではありません。

次のような単純な 조건은,常に TRUE に 평가됩니다。

```
1 = 1
```

次のやや複雑な 조건은, salary の値を salary*commission_pct の値に加算し (NULL は 0 で置き換える)、その合計が定数 25000 より大きいかどうかを判断します。

```
NVL(salary, 0) + NVL(salary*commission_pct, 0) > 25000
```

論理 조건을 사용하면,複数の 조건을 단일의 조건에 결합할 수 있습니다。たとえば,次のように AND 条件を使用して 2 つの 조건을 결합할 수 있습니다。

```
(1 = 1) AND (5 < 7)
```

有効な 조건의 예를次に示します。

```
name = 'SMITH'
employees.department_id = departments.department_id
hire_date > '01-JAN-88'
job_id IN ('SA_MAN', 'SA_REP')
salary BETWEEN 5000 AND 10000
commission_pct IS NULL AND salary = 2100
```

参照： 文に指定する条件の制限については、第 9 章～第 18 章にある各文の説明を参照してください。

条件の優先順位

優先順位とは、同じ式の中の異なる条件を Oracle が評価する順序を意味します。複数の条件を含む式を評価するとき、Oracle は優先順位の高い条件を評価した後で、優先順位の低い条件を評価します。優先順位の等しい条件は、式の中で左から右に評価されます。

表 5-1 に、SQL 条件を優先順位の高い方から順に示します。同じ行に示されている条件の優先順位は同じです。表に示すとおり、Oracle は条件の前に演算子を評価します。

表 5-1 SQL 条件の優先順位

条件の種類	用途
SQL 演算子は、SQL 条件の前で評価されます。	3-2 ページの「 演算子の優先順位 」を参照してください。
=、!=、<、>、<=、>=	比較
IS [NOT] NULL、LIKE、[NOT] BETWEEN、[NOT] IN、EXISTS、IS OF type	比較
NOT	指数、論理否定
AND	論理積
OR	論理和

比較条件

比較条件は、2 つの式を比較します。比較の結果は、真 (TRUE)、偽 (FALSE) または不明 (UNKNOWN) になります。

注意： ラージ・オブジェクト (LOB) は、比較条件ではサポートされていません。ただし、CLOB データの比較では、PL/SQL プログラムを使用できます。

表 5-2 に、比較条件を示します。

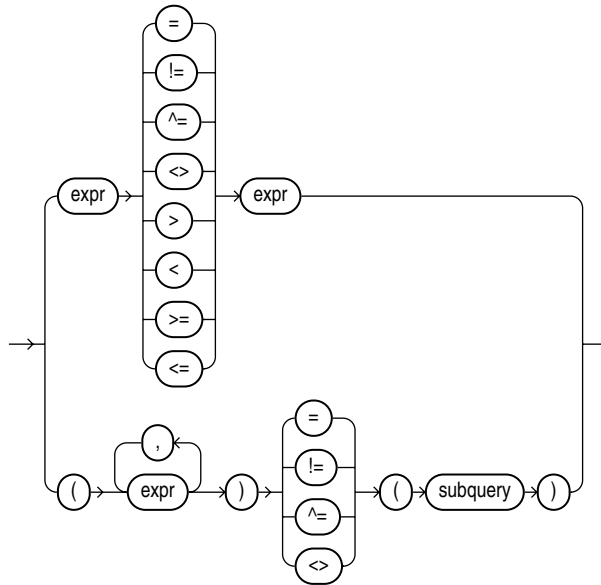
表 5-2 比較条件

条件の種類	用途	例
=	等価性をテストします。	<pre>SELECT * FROM employees WHERE salary = 2500;</pre>
!= ^= <> ¬=	不等性をテストします。プラットフォームによっては、一部の不等号条件を使用できない場合があります。	<pre>SELECT * FROM employees WHERE salary != 2500;</pre>
> <	大 / 小をテストします。	<pre>SELECT * FROM employees WHERE salary > 2500; SELECT * FROM employees WHERE salary < 2500;</pre>
>= <=	以上 / 以下をテストします。	<pre>SELECT * FROM employees WHERE salary >= 2500; SELECT * FROM employees WHERE salary <= 2500;</pre>
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、ある値を比較します。=、!=、>、<、<=、>= のいずれかを先に指定する必要があります。 問合せによって行が戻されない場合には、FALSE と評価されます。	<pre>SELECT * FROM employees WHERE salary = ANY (SELECT salary FROM employees WHERE department_id = 30);</pre>
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、ある値を比較します。=、!=、>、<、<=、>= のいずれかを先に指定する必要があります。 問合せによって行が戻されない場合には、TRUE と評価されます。	<pre>SELECT * FROM employees WHERE salary >= ALL (1400, 3000);</pre>

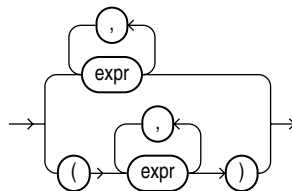
単純比較条件

単純比較条件は、式または副問合せの結果の比較方法を指定します。

simple_comparison_condition::=



expression_list::=



この条件の下の方の書式（演算子の左辺に複数の式を指定する書式）を使用する場合は、*expression_list* の下の方の書式を使用し、副問合せによって戻される値は、*expression_list* 内の式と同じ数とデータ型で構成されている必要があります。

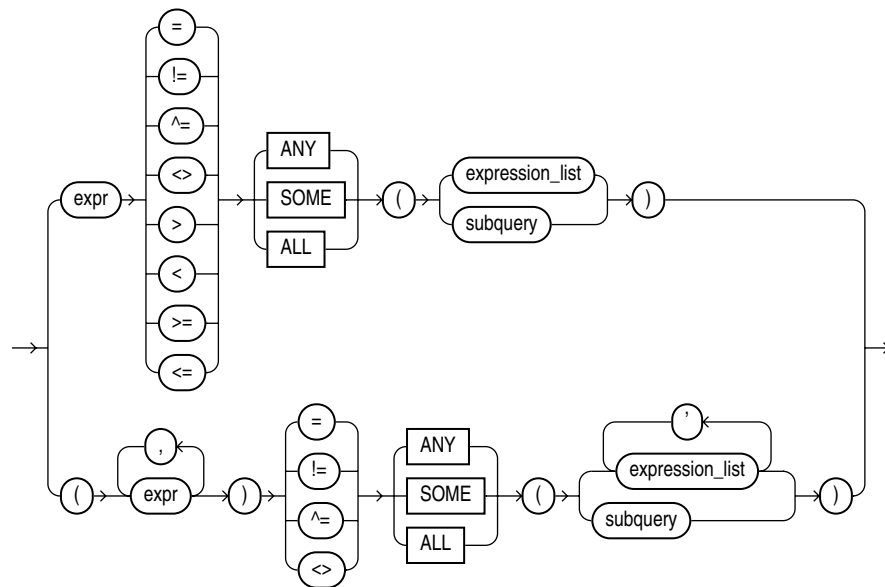
参照：

- 式の組合せの詳細は、4-16 ページの「[式のリスト](#)」を参照してください。
- 副問合せの詳細は、18-4 ページの「[SELECT](#)」を参照してください。

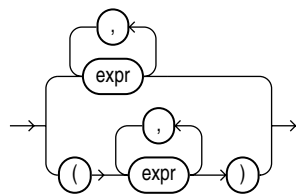
グループ比較条件

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

group_comparison_condition::=



expression_list::=



この条件の上の方の書式（演算子の左辺に 1 つの式を指定する書式）を使用する場合は、*expression_list* の上の方の書式を使用する必要があります。この条件の下の方の書式（演算子の左辺に複数の式を指定する書式）を使用する場合は、*expression_list* の下の方の書式を使用し、各 *expression_list* 内の式は、演算子の左辺にある式と同じ数とデータ型で構成されている必要があります。

参照：

- 4-16 ページの「[式のリスト](#)」を参照してください。
- 18-4 ページの「[SELECT](#)」を参照してください。

論理条件

論理条件は、2 つのコンポーネント条件の結果を組み合わせ、それらに基づいて単一の結果を生成したり、単一の条件の結果を反転させます。[表 5-3](#) に、論理条件を示します。

表 5-3 論理条件

条件の種類	操作	例
NOT	後続する条件が FALSE の場合に TRUE を返します。TRUE の場合には FALSE を返します。UNKNOWN の場合には UNKNOWN を返します。	<pre>SELECT * FROM employees WHERE NOT (job_id IS NULL); SELECT * FROM employees WHERE NOT (salary BETWEEN 1000 AND 2000);</pre>
AND	コンポーネントの条件が両方とも TRUE の場合に TRUE を返します。どちらかが FALSE の場合には FALSE を返します。それ以外の場合は UNKNOWN を返します。	<pre>SELECT * FROM employees WHERE job_id = 'PU_CLERK' AND department_id = 30;</pre>
OR	コンポーネントの条件のどちらかが TRUE の場合に TRUE を返します。両方とも FALSE の場合は FALSE を返します。それ以外の場合は UNKNOWN を返します。	<pre>SELECT * FROM employees WHERE job_id = 'PU_CLERK' OR department_id = 10;</pre>

表 5-4 に、式に NOT 条件を適用した結果を示します。

表 5-4 NOT 真理値表

—	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

表 5-5 に、2 つの式に AND 条件を組み合わせた結果を示します。

表 5-5 AND 真理値表

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

たとえば、次の SELECT 文の WHERE 句は、AND 論理条件を使用して、1984 年より前に入社し、給与が 1,000 ドルを超える従業員のみを戻します。

```
SELECT * FROM employees
WHERE hire_date < TO_DATE('01-JAN-1989', 'DD-MON-YYYY')
      AND salary > 2500;
```

表 5-6 に、2 つの式に OR を適用した結果を示します。

表 5-6 OR 真理値表

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

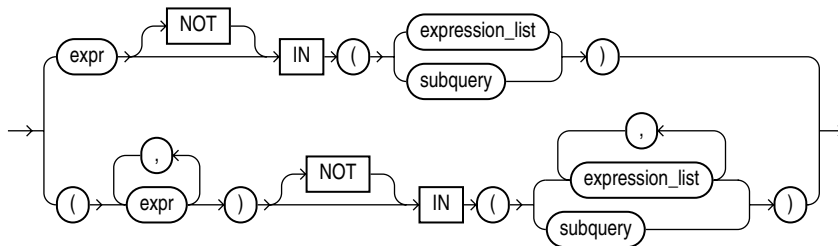
たとえば、次の問合せは、歩合率が 40% または給与が 20,000 ドルを超える従業員を戻します。

```
SELECT employee_id FROM employees
WHERE commission_pct = .4 OR salary > 20000;
```

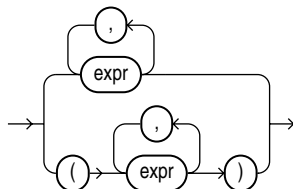
メンバーシップ条件

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

membership_condition::=



expression_list::=



この条件の上の方の書式（演算子の左辺に 1 つの式を指定する書式）を使用する場合は、*expression_list* の上の方の書式を使用する必要があります。この条件の下の方の書式（演算子の左辺に複数の式を指定する書式）を使用する場合は、*expression_list* の下の方の書式を使用し、各 *expression_list* 内の式は、演算子の左辺にある式と同じ数とデータ型で構成されている必要があります。

参照： 4-16 ページの「[式のリスト](#)」を参照してください。

表 5-7 に、メンバーシップ条件を示します。

表 5-7 メンバーシップ条件

条件の種類	操作	例
IN	メンバーとの等価性をテストします。「= ANY」と同じです。	<pre>SELECT * FROM employees WHERE job_id IN ('PU_CLERK','SH_CLERK'); SELECT * FROM employees WHERE salary IN (SELECT salary FROM employees WHERE department_id =30);</pre>
NOT IN	「!=ALL」と同じです。メンバーのいずれかが NULL の場合には、FALSE と評価されます。	<pre>SELECT * FROM employees WHERE salary NOT IN (SELECT salary FROM employees WHERE department_id = 30); SELECT * FROM employees WHERE job_id NOT IN ('PU_CLERK', 'SH_CLERK');</pre>

NOT IN 演算子に続くリストの中のいずれかの項目が NULL の場合は、すべての行は FALSE または不明 (UNKNOWN) と評価されます (行は戻されません)。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 employees
WHERE department_id NOT IN (10, 20);
```

ただし、次の文では行は戻されません。

```
SELECT 'True' FROM employees
WHERE department_id NOT IN (10, 20, NULL);
```

この例で行が戻されないのは、WHERE 句の条件が次のように評価されるためです。

```
department_id != 10 AND department_id != 20 AND department_id != null
```

3 番目の条件で department_id と NULL の比較が行われるため、この条件の結果が UNKNOWN となり、式全体の結果が FALSE (department_id を持つ行が 10 または 20 と等しいため) となります。特に、NOT IN 演算子が副問合せを参照するときは、このような動作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。

また、NOT IN 条件が、行を戻さない副問合せを参照する場合は、次のように、すべての行が戻されます。

```
SELECT 'True' FROM employees
WHERE department_id NOT IN (SELECT 0 FROM dual WHERE 1=2);
```

WHERE 句の LEVEL の制限事項： WHERE 句の [NOT] IN 条件の右边が副問合せになっている場合、条件の左辺で LEVEL は使用できません。ただし、FROM 句の副問合せで LEVEL を指定すると、同一の結果が得られます。たとえば、次の文は無効です。

```
SELECT employee_id, last_name FROM employees
WHERE (employee_id, LEVEL)
      IN (SELECT employee_id, 2 FROM employees)
START WITH employee_id = 2
CONNECT BY PRIOR employee_id = manager_id;
```

ただし、次の文は、LEVEL 情報を含む問合せを FROM 句にカプセル化するため有効です。

```
SELECT v.employee_id, v.last_name, v.lev
FROM
  (SELECT employee_id, last_name, LEVEL lev
   FROM employees v
   START WITH employee_id = 100
   CONNECT BY PRIOR employee_id = manager_id) v
WHERE (v.employee_id, v.lev) IN
      (SELECT employee_id, 2 FROM employees);
```

範囲条件

範囲条件は、範囲に含まれているかどうかをテストします。

range_condition::=

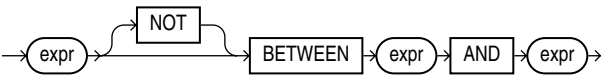


表 5-8 に、範囲条件を示します。

表 5-8 範囲条件

条件の種類	操作	例
[NOT] BETWEEN x AND y	x 以上 y 以下の範囲である [な い] ことをテストします。	SELECT * FROM employees WHERE salary BETWEEN 2000 AND 3000;

NULL 条件

NULL 条件は、NULL かどうかをテストします。

null_condition::=

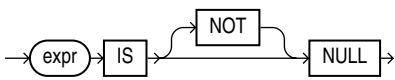


表 5-9 に、NULL 条件を示します。

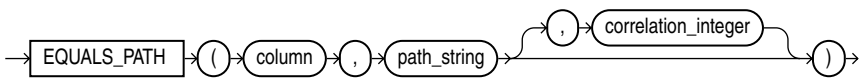
表 5-9 NULL 条件		
条件の種類	操作	例
IS [NOT] NULL	NULL をテストします。NULL のテストに使用する必要がある唯一の条件です。 参照 ：2-76 ページの「NULL」を参照してください。	SELECT last_name FROM employees WHERE commission_pct IS NULL;

EQUALS_PATH

EQUALS_PATH 条件は、Oracle XML DB のリソースが、指定されたパスのデータベースに存在するかを判断します。

この条件は、RESOURCE_VIEW および PATH_VIEW の問合せで使用します。これらのパブリック・ビューは、XML データベース・リポジトリ内に格納されているデータに対し、SQL でアクセスするためのメカニズムを提供します。RESOURCE_VIEW には、リポジトリ内のリソースごとに行が 1 行あり、PATH_VIEW には、リポジトリ内の一意パスごとに行が 1 行あります。

equals_path::=



この条件は、指定どおりパスにのみ適用されます。この条件は、UNDER_PATH と似ていますが、制限は多くなります。

オプションの correlation_integer 引数は、EQUALS_PATH 条件と補助ファンクション PATH および DEPTH を関連付けます。

参照： 5-20 ページの「UNDER_PATH」、6-52 ページの「DEPTH」および 6-109 ページの「PATH」を参照してください。

例

ビュー RESOURCE_VIEW は、データベース・リポジトリ内の (res 列にある) すべての XML リソースへの (any_path 列にある) パスを計算します。次の例は、RESOURCE_VIEW ビューに問い合せて、サンプル・スキーマ oe にあるリソースへのパスを検索します。EQUALS_PATH 条件により、問合せは指定されたパスのみを戻します。

```
SELECT ANY_PATH FROM RESOURCE_VIEW
       WHERE EQUALS_PATH(res, '/sys/schemas/OE/www.oracle.com')=1;
```

```
ANY_PATH
-----
/sys/schemas/OE/www.oracle.com
```

この例と、5-20 ページの「[UNDER_PATH](#)」の例を比較してください。

EXISTS 条件

EXISTS 条件は、副問合せに行が存在するかどうかをテストします。

exists_condition::=

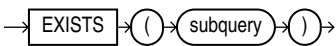


表 5-10 に、EXISTS 条件を示します。

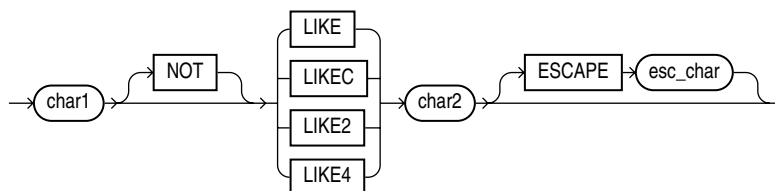
表 5-10 EXISTS 条件

条件の種類	操作	例
EXISTS	副問合せによって行が 1 行以上戻される場合には、TRUE と評価されます。	<pre>SELECT department_id FROM departments d WHERE EXISTS (SELECT * FROM employees e WHERE d.department_id = e.department_id);</pre>

LIKE 条件

LIKE 条件は、パターン一致を含むかどうかをテストします。等号演算子 (=) は、ある文字値を別の文字値と一致させますが、LIKE 条件は、ある文字値の一部を別の文字値と一致させます (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います)。LIKE は、入力キャラクタ・セットによって定義された文字を使用して、文字列を算出します。LIKEC は、完全な Unicode キャラクタを使用します。LIKE2 は、UCS2 コードポイントを使用します。LIKE4 は、UCS4 コードポイントを使用します。

like_condition::=



この構文は、次の特長があります。

- `char1` は、キャラクタ列などの文字式で、**検索値**と呼ばれます。
- `char2` は、通常はリテラルの文字式で、**パターン**と呼ばれます。
- `esc_char` は、通常はリテラルの文字式で、**エスケープ文字**と呼ばれます。

`esc_char` が指定されていない場合、デフォルトのエスケープ文字はありません。`char1`、`char2` または `esc_char` のいずれかが `NULL` である場合、結果は不明になります。それ以外の場合は、エスケープ文字（指定されている場合）は、長さが 1 の文字列です。

すべての文字式（`char1`、`char2` および `esc_char`）は、`CHAR`、`VARCHAR2`、`NCHAR` または `NVARCHAR2` データ型です。文字式が異なる場合、Oracle はすべての文字式を `char1` のデータ型に変換します。

パターンは、特殊パターン一致文字を含むことができます。

- パーセント（%）は、任意の長さ（0 を含む）のすべての文字列を一致させます。
- アンダースコア（_）は、すべての単一文字を一致させます。

パーセント（%）およびアンダースコア（_）の文字を検索するには、それらの文字の前にエスケープ文字を指定してください。たとえば、エスケープ文字がアットマーク（@）である場合、`@%` を使用してパーセント（%）を、`@_` を使用してアンダースコア（_）を検索できます。

エスケープ文字を検索する場合は、その文字を続けて入力してください。たとえば、アットマーク（@）がエスケープ文字である場合、`@@` を使用してアットマーク（@）を検索できます。

パターンでは、エスケープ文字に、パーセント（%）、アンダースコア（_）またはエスケープ文字自体が続く必要があります。

表 5-11 に、LIKE 条件を示します。

表 5-11 LIKE 条件

条件の種類	操作	例
x [NOT] LIKE y [ESCAPE 'z']	x がパターン y に一致しない場合は TRUE と評価されます。y 内で、文字「%」は 0 以上の NULL 以外の文字を含む文字列と一致します。「_」文字は、任意の 1 文字に一致します。パーセント (%) およびアンダースコア () を除く任意の文字を ESCAPE の後に指定できます。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付いている場合はリテラルとして扱われます。	<pre>SELECT last_name FROM employees WHERE last_name LIKE '%A_B%' ESCAPE '\';</pre>

LIKE 条件を処理するために、Oracle は、パターンを 1 つまたは 2 つの文字で構成されるサブパターンに分割します。2 文字のサブパターンは、エスケープ文字で始まり、もう 1 つの文字はパーセント (%)、アンダースコア () またはエスケープ文字です。

P₁、P₂、...、P_n を、このようなサブパターンと想定します。検査値を部分文字列 S₁、S₂、...、S_n に分割でき、すべての i が 1 ～ n の場合、LIKE 条件は真です。

- P_i がアンダースコア () の場合、S_i は単一文字です。
- P_i がパーセント (%) の場合、S_i は文字列です。
- P_i がエスケープ文字で始まる 2 文字の場合、S_i は P_i の 2 番目の文字です。
- それ以外の場合は、P_i と S_i は同じです。

LIKE 条件では、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って、名前が「R」で始まるすべての従業員の給与を検索できます。

```
SELECT salary
FROM employees
WHERE last_name LIKE 'R%';
```

次の問合せは、LIKE 条件ではなく = 演算子を使用しているため、名前が「R%」のすべての従業員の給与が検索されます。

```
SELECT salary
FROM employees
WHERE last_name = 'R%';
```

次の問合せでは、名前が「SM%」のすべての従業員の給与が検索されます。この場合、'SM%' が LIKE キーワードの前にあるため、Oracle は、'SM%' をパターンとしてではなく、テキスト・リテラルとして解析します。

```
SELECT salary
FROM employees
WHERE 'SM%' LIKE last_name;
```

パターンでは、値の中の異なる文字に置き換えることができ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）に置き換えることができます。
- パターン中のパーセント記号 (%) は、値の中の 0（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）に置き換えることができます。ただし、パターン「%」は、NULL と一致しません。

英大文字または英小文字の区別

LIKE 条件および等号 (=) 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 ファンクションを使用すると、大文字と小文字を区別せずに一致させることができます。

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

索引列でのパターン一致

LIKE を使用して索引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 は索引を利用して文のパフォーマンスを向上させることができます。この場合、Oracle はこの先頭文字によって索引をスキャンできます。パターンの先頭文字が「%」または「_」の場合、Oracle は索引をスキャンできないため、問合せのパフォーマンスは向上しません。

一般的な例

次の条件は、「Ma」で始まるすべての last_name 値について TRUE（真）となります。

```
last_name LIKE 'Ma%'
```

次の last_name 値はすべて、条件を TRUE（真）にします。

```
Mallin, Markle, Marlow, Marvins, Marvis, Matos
```

大文字と小文字は区別されるため、「MA」、「ma」および「mA」で始まる last_name 値では条件が FALSE（偽）になります。

次の条件を考えます。

```
last_name LIKE 'SMITH_'
```

この条件は、次の `last_name` 値について TRUE（真）となります。

```
SMITHE, SMITHY, SMITHS
```

特殊文字「`_`」は、`last_name` 値の 1 つの文字に置き換えることができるため、この条件は「SMITH」について偽となります。

ESCAPE 句の例

エスケープ文字を識別する `ESCAPE` 句を使用すると、パターン中に「`%`」または「`_`」を実際の文字として含めることができます。エスケープ文字がパターンの中で文字「`%`」または「`_`」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一致文字としてではなく、リテラル文字として解析します。

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 last_name
FROM employees
WHERE last_name LIKE '%A\_B%' ESCAPE '\';
```

`ESCAPE` 句は、エスケープ文字としてバックスラッシュ（`\`）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一致文字としてではなく、リテラルとして解析します。

パーセント（%）なしのパターンの例

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のみ、条件が TRUE（真）になります。この表の定義および挿入される値について考えます。

```
CREATE TABLE ducks (f CHAR(6), v VARCHAR2(6));
INSERT INTO ducks VALUES ('DUCK', 'DUCK');
SELECT '*' || f || '*' "char",
       '*' || v || '*' "varchar"
FROM ducks;
```

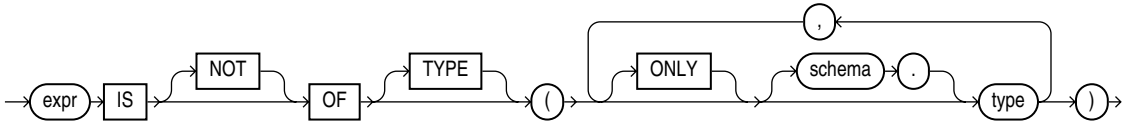
```
char      varchar
-----
*DUCK    *DUCK*
```

Oracle は、`CHAR` 値に空白埋めを行うため、`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。

IS OF type 条件

IS OF type 条件を使用すると、固有の型情報に基づくオブジェクト・インスタンスをテストできます。

is_of_type_condition::=



type によって参照されるすべての型に対する EXECUTE 権限が必要です。また、すべての type は、同じ型ファミリに属している必要があります。

expr が NULL である場合、この条件は NULL と評価されます。expr が NULL でない場合、次に示す環境では、この条件は TRUE (NOT キーワードを指定した場合は FALSE) と評価されます。

- expr の型が、type リストで指定された型のサブタイプであり、ONLY を指定していない場合
- expr の型が type リストで明示的に指定されている場合

expr は、相関変数を伴う VALUE ファンクションの書式をとる場合があります。

次の例では、15-63 ページの「置換可能な表および列のサンプル:」の型階層に基づいて作成されたサンプル表 oe.persons を使用します。この例は、IS OF type 条件を使用して、問合せを特定のサブタイプに制限します。

```
SELECT * FROM persons p
WHERE VALUE(p) IS OF TYPE (employee_t);
```

NAME	SSN

Joe	32456
Tim	5678

```
SELECT * FROM persons p
WHERE VALUE(p) IS OF (ONLY part_time_emp_t);
```

NAME	SSN

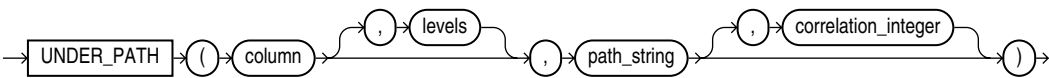
Tim	5678

UNDER_PATH

UNDER_PATH 条件は、列内に指定されたリソースが、Oracle XML DB リポジトリ内の *path_string* で指定された特定のパスに存在するかを判断します。パス情報は、この条件を使用する場合に問い合わせる RESOURCE_VIEW ビューによって計算されます。

この条件は、RESOURCE_VIEW および PATH_VIEW の問合せで使用します。これらのパブリック・ビューは、XML データベース・リポジトリ内に格納されているデータに対して SQL でアクセスするためのメカニズムを提供します。RESOURCE_VIEW には、リポジトリ内のリソースごとに行が 1 行あり、PATH_VIEW には、リポジトリ内の一意パスごとに行が 1 行あります。

under_path::=



オプションの *levels* 引数は、検索対象となる *path_string* 以下のレベル数を示します。0 未満の値は 0 として扱われます。

オプションの *correlation_integer* 引数は、UNDER_PATH 条件と補助関数 *PATH* および *DEPTH* を関連付けます。

参照：

- 5-13 ページの「[EQUALS_PATH](#)」を参照してください。
- 6-52 ページの「[DEPTH](#)」および 6-109 ページの「[PATH](#)」を参照してください。

例

ビュー RESOURCE_VIEW は、データベース・リポジトリ内の (res 列にある) すべての XML リソースへの (any_path 列にある) パスを計算します。次の例は、RESOURCE_VIEW ビューに問い合わせ、サンプル・スキーマ oe にあるリソースへのパスを検索します。この問合せは、15-67 ページの「XMLType 表の例 :」で作成された XML Schema のパスを戻します。

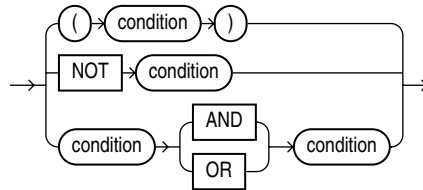
```
SELECT ANY_PATH FROM RESOURCE_VIEW
      WHERE UNDER_PATH(res, '/sys/schemas/OE/www.oracle.com')=1;

ANY_PATH
-----
/sys/schemas/OE/www.oracle.com/xwarehouses.xsd
```

複合条件

複合条件は、異なる条件の組合せを指定します。

compound_condition ::=



参照： NOT、AND および OR 条件の詳細は、5-8 ページの「[論理条件](#)」を参照してください。

ファンクション

ファンクションは、データ項目を操作し、結果を戻すという点で演算子に似ていますが、ファンクション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、ファンクションでは0（ゼロ）以上の引数を操作できます。

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

この章では、次の内容を説明します。

- [SQL ファンクション](#)
- [ユーザー定義ファンクション](#)

SQL ファンクション

SQL ファンクションは、Oracle に組み込まれており、適切な SQL 文で使用できます。SQL ファンクションと、PL/SQL で記述されたユーザー・ファンクションを混同しないでください。

SQL ファンクションが戻す値のデータ型以外のデータ型の引数で SQL ファンクションをコールすると、Oracle は SQL ファンクションを実行する前に、その引数を必要なデータ型に暗黙的に変換します。NULL を引数として SQL ファンクションをコールすると、SQL ファンクションは自動的に NULL を戻します。この規則に従わない SQL ファンクションは、CONCAT、NVL および REPLACE のみです。

SQL ファンクションの構文図では、データ型とともに引数が示されています。SQL 構文にパラメータ「function」が指定されている場合は、この項で説明するファンクションの1つに置き換えます。ファンクションは、引数のデータ型および戻り値によってグループ化されています。

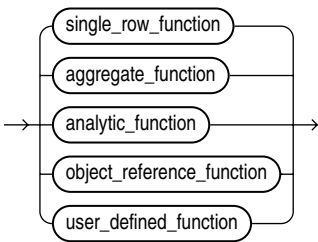
注意： LOB 列に SQL ファンクションを適用すると、Oracle は、SQL および PL/SQL の処理中に一時 LOB を作成します。ご使用のアプリケーションの一時 LOB を格納するために、十分な一時表領域が割り当てられていることを確認してください。

参照：

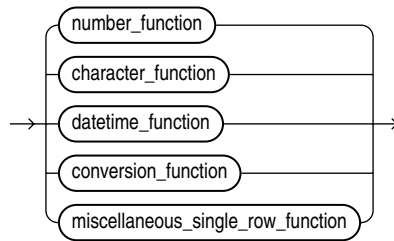
- ユーザー・ファンクションの詳細は、6-211 ページの「[ユーザー定義ファンクション](#)」を参照してください。
- Oracle Text で使用するファンクションの詳細は、『Oracle Text リファレンス』を参照してください。
- データ型の暗黙的な変換については、2-46 ページの「[データ変換](#)」を参照してください。

次に、ファンクションのカテゴリを表す構文を示します。

function::=



single_row_function::=



次の項に、前述の図のユーザー定義ファンクション以外の各グループにおける組込み SQL ファンクションを示します。すべての組込み SQL ファンクションを、アルファベット順に説明します。ユーザー定義ファンクションについては、この章の最後で説明します。

単一行ファンクション

単一行ファンクションは、問合せ対象の表またはビューの各行に対して 1 つの結果行を返します。これらのファンクションは、SELECT 構文のリスト、WHERE 句、START WITH 句、CONNECT BY 句および HAVING 句に指定できます。

数値ファンクション

数値ファンクションは入力として数値を受け取り、結果として数値を返します。これらのファンクションのほとんどは、38 桁（10 進）の値を返します。超越関数（COS、COSH、EXP、LN、LOG、SIN、SINH、SQRT、TAN および TANH）は、36 桁（10 進）の値を返します。超越関数の ACOS、ASIN、ATAN、ATAN2 は、30 桁（10 進）の値を返します。数値ファンクションを次に示します。

ABS
ACOS
ASIN
ATAN
ATAN2
BITAND
CEIL
COS
COSH
EXP
FLOOR
LN
LOG
MOD
POWER
ROUND（数値）

SIGN
SIN
SINH
SQRT
TAN
TANH
TRUNC (数値)
WIDTH_BUCKET

文字値を戻す文字ファンクション

文字値を戻す文字ファンクションは、入力した引数と同じデータ型の値を戻します。

- CHAR 型の値を戻すファンクションは、値の長さが 2000 バイトに制限されます。
- VARCHAR2 型の値を戻すファンクションは、値の長さが 4000 バイトに制限されます。

これらのファンクションでは、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・メッセージを表示せずにその結果を戻します。

- CLOB 型の値を戻すファンクションは、値の長さが 4GB に制限されます。

CLOB ファンクションの戻り値が制限を超える場合、Oracle はエラーを表示し、データを戻しません。

文字値を戻す文字ファンクションを次に示します。

CHR
CONCAT
INITCAP
LOWER
LPAD
LTRIM
NLS_INITCAP
NLS_LOWER
NLSSORT
NLS_UPPER
REPLACE
RPAD
RTRIM
SOUNDEX
SUBSTR
TRANSLATE
TREAT
TRIM
UPPER

数値を戻す文字ファンクション

数値を戻す文字ファンクションの引数には、すべての文字データ型を指定できます。

数値を戻す文字ファンクションを次に示します。

ASCII
INSTR
LENGTH

日時ファンクション

日時ファンクションは、DATE データ型の値を操作します。数値を戻す MONTHS_BETWEEN ファンクションを除いて、すべての日時ファンクションは DATE データ型の日時または期間の値を戻します。日時ファンクションを次に示します。

ADD_MONTHS
CURRENT_DATE
CURRENT_TIMESTAMP
DBTIMEZONE
EXTRACT (日時)
FROM_TZ
LAST_DAY
LOCALTIMESTAMP
MONTHS_BETWEEN
NEW_TIME
NEXT_DAY
NUMTODSINTERVAL
NUMTOYMINTERVAL
ROUND (日付)
SESSIONTIMEZONE
SYS_EXTRACT_UTC
SYSDATE
SYSTEMTIMESTAMP
TO_DSINTERVAL
TO_TIMESTAMP
TO_TIMESTAMP_TZ
TO_YMINTERVAL
TRUNC (日付)
TZ_OFFSET

変換ファンクション

変換ファンクションは、あるデータ型から他のデータ型に値を変換します。一般に、ファンクション名は `datatype TO datatype` の書式で指定されます。最初のデータ型は入力データ型です。2 番目のデータ型は出力データ型です。SQL 変換ファンクションを次に示します。

ASCIISTR
BIN_TO_NUM
CAST
CHARTOROWID
COMPOSE
CONVERT
DECOMPOSE
HEXTORAW
NUMTODSINTERVAL
NUMTOYMINTERVAL
RAWTOHEX
RAWTONHEX
ROWIDTOCHAR
ROWIDTONCHAR
TO_CHAR (文字)
TO_CHAR (日時)
TO_CHAR (数値)
TO_CLOB
TO_DATE
TO_DSINTERVAL
TO_LOB
TO_MULTI_BYTE
TO_NCHAR (文字)
TO_NCHAR (日時)
TO_NCHAR (数値)
TO_NCLOB
TO_NUMBER
TO_SINGLE_BYTE
TO_YMINTERVAL
TRANSLATE ...USING
UNISTR

その他の単一行ファンクション

次の単一行ファンクションは、他の単一行ファンクションのカテゴリのいずれにも入りません。

BFILENAME
COALESCE
DECODE

DEPTH
DUMP
EMPTY_BLOB、EMPTY_CLOB
EXISTSNODE
EXTRACT (XML)
EXTRACTVALUE
GREATEST
LEAST
NLS_CHARSET_DECL_LEN
NLS_CHARSET_ID
NLS_CHARSET_NAME
NULLIF
NVL
NVL2
PATH
SYS_CONNECT_BY_PATH
SYS_CONTEXT
SYS_DBURIGEN
SYS_EXTRACT_UTC
SYS_GUID
SYS_TYPEID
SYS_XMLAGG
SYS_XMLGEN
UID
UPDATEXML
USER
USERENV
VSIZE
XMLAGG
XMLCOLATTVAL
XMLCONCAT
XMLFOREST
XMLSEQUENCE
XMLTRANSFORM

集計ファンクション

集計ファンクションは、単一行に基づく結果行を戻すのではなく、行のグループに基づく単一結果行を戻します。集計ファンクションは、**SELECT** 構文のリスト、**ORDER BY** および **HAVING** 句に指定できます。集計ファンクションは、通常、**SELECT** 文の **GROUP BY** 句で使用され、この句の中で問合せ対象の表またはビューの行がグループ化されます。**GROUP BY** 句を含む問合せでは、**SELECT** 構文のリストの要素は、集計ファンクション、**GROUP BY** 式、定数またはこれらのいずれかを含む式になります。**Oracle** は、集計ファンクション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、SELECT 構文のリスト内の集計ファンクションは、問合せ対象の表またはビューのすべての行に適用されます。HAVING 句で集計ファンクションを使用して、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、集計ファンクションの結果に基づきます。

参照： 問合せおよび副問合せにおける GROUP BY 句および HAVING 句の詳細は、18-29 ページの「[GROUP BY 句の使用例:](#)」および 18-22 ページの「[HAVING 句](#)」を参照してください。

単一の引数を指定する多くの集計ファンクションには、次の句があります。

- DISTINCT 句を指定すると、集計ファンクションは引数式の重複行を排除した値のみを考慮します。
- ALL 句を指定すると、集計ファンクションは重複行を含むすべての値を考慮します。

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。どの句も指定しない場合、デフォルトで ALL が使用されます。

COUNT(*) および GROUPING を除くすべての集計ファンクションは、NULL を無視します。集計ファンクションに対する引数に NVL ファンクションを使用して、NULL をある値で置き換えることができます。COUNT は NULL を戻しません。数字または 0（ゼロ）を戻します。その他の集計ファンクションでは、データ・セットに行がない場合、または集計ファンクションに対する引数として NULL を持つ行のみがある場合は NULL を戻します。

集計ファンクションはネストできます。たとえば、次の例では、サンプル・スキーマ hr のすべての部門における最高額の給与の平均を計算します。

```
SELECT AVG(MAX(salary)) FROM employees GROUP BY department_id;
```

```
AVG(MAX(SALARY))
-----
          10925
```

この計算では、GROUP BY 句 (department_id) で定義されている各グループごとの内部集計 (MAX (salary)) を評価し、その結果をもう一度集計しています。

集計ファンクションを次に示します。

```
AVG
CORR
COUNT
COVAR_POP
COVAR_SAMP
CUME_DIST
DENSE_RANK
FIRST
GROUP_ID
GROUPING
```

GROUPING_ID
 LAST
 MAX
 MIN
 PERCENTILE_CONT
 PERCENTILE_DISC
 PERCENT_RANK
 RANK
 REGR_ (線形リグレーション) ファンクション
 STDDEV
 STDDEV_POP
 STDDEV_SAMP
 SUM
 VAR_POP
 VAR_SAMP
 VARIANCE

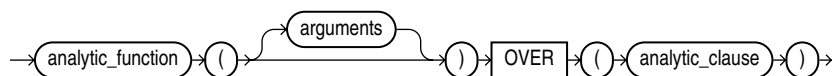
分析ファンクション

分析ファンクションは、行のグループに基づいて集計値を計算します。各グループに対して複数の行を戻す点で、集計ファンクションと異なります。行のグループを **window** といい、分析句で定義されます。各行に対して、行の「スライディング」ウィンドウが定義されます。この **window** によって、「カレント行」の計算に使用される行の範囲が決定されます。**window** の大きさは、行の物理数値または時間などのロジカル・インターバルに基づきます。

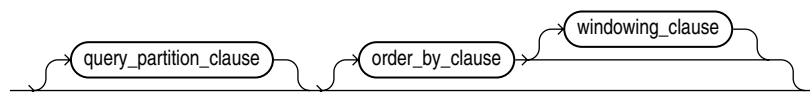
分析ファンクションは、問合せで最後に実行される演算（最後の ORDER BY 句を除く）の集合です。すべての結合およびすべての WHERE、GROUP BY および HAVING 句は、分析ファンクションが処理される前に実行されます。そのため、分析ファンクションは、SELECT 構文のリストまたは ORDER BY 句のみに指定できます。

通常、分析ファンクションは、累積集計、移動集計、センター集計およびレポート集計の実行に使用されます。

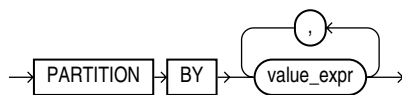
analytic_function::=



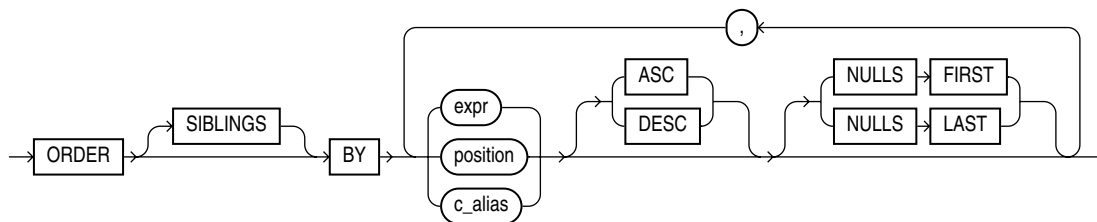
analytic_clause::=



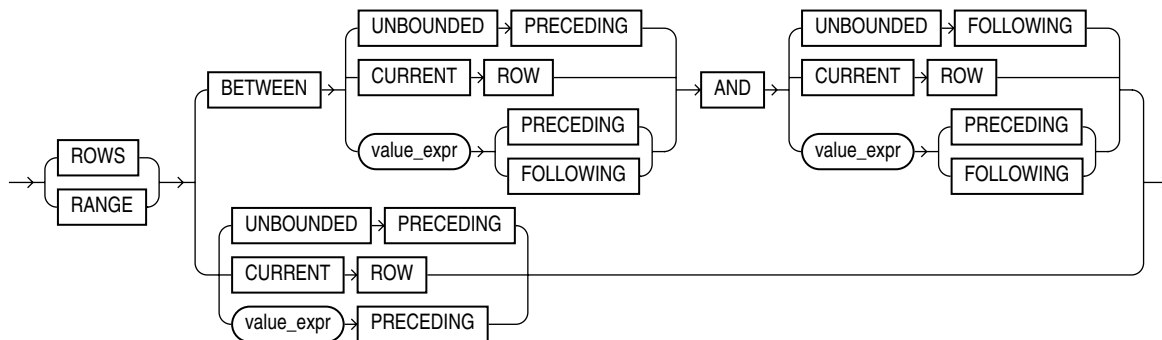
query_partition_clause::=



order_by_clause::=



windowing_clause::=



次に、この構文のセマンティクスを示します。

analytic_function

分析ファンクションの名前を指定します（6-14 ページの表を参照）。

arguments

分析ファンクションには引数を 0 ～ 3 個指定します。

analytic_clause

OVER *analytic_clause* 句は、ファンクションが問合せ結果セットを操作することを示します。FROM、WHERE、GROUP BY および HAVING 句の後に計算されます。SELECT 構文のリストのこの句または ORDER BY 句に分析ファンクションを指定できます。分析ファンクションに基づいて、問合せの結果をフィルタするには、これらのファンクションを親問合せ内でネストした後、ネストされた副問合せの結果をフィルタします。

注意：

- *analytic_clause* のどの部分にも、分析ファンクションを指定できません。つまり、分析ファンクションはネストできません。ただし、副問合せで分析ファンクションを指定して、別の分析ファンクションを計算することはできます。
 - OVER *analytic_clause* には、組込み分析ファンクションと同様に、ユーザー定義の分析ファンクションを指定できます。13-49 ページの [CREATE FUNCTION](#) を参照してください。
-
-

query_partition_clause

PARTITION BY 句を使用すると、1 つ以上の *value_expr* に基づいて、問合せ結果セットをグループに分割できます。この句を省略すると、ファンクションは問合せ結果セットのすべての行を単一のグループとして扱います。

同じまたは異なる PARTITION BY キーで、同じ問合せに複数の分析ファンクションを指定できます。

注意： 問い合わせているオブジェクトにパラレル属性があり、*query_partition_clause* で分析ファンクションを指定する場合は、ファンクションの計算もパラレル化されます。

有効な値の *value_expr* は、定数、列、非分析ファンクション、ファンクション式、またはこれらのいずれかを含む式です。

order_by_clause

order_by_clause を使用すると、パーティション内でのデータの順序付け方法を指定できます。PERCENTILE_CONT および（単一キーのみを適用する）PERCENTILE_DISC 以外の分析ファンクションでは、各キーが *value_expr* で定義され、順序付けシーケンスで修飾された複数キーのパーティションの値を順序付けできます。

各ファンクションには、複数の順序式を指定できます。これは、2 番目の式が最初の式にある同一値との間の関連性を変換できるため、値をランク付けするファンクションを使用する場合に特に有効です。

注意： `order_by_clause` の結果が複数行の個々の値である場合、ファンクションは各行の同じ値を戻します。この動作の詳細は、6-145 ページの「[SUM](#)」の分析例を参照してください。

ORDER BY 句の制限事項： `order_by_clause` を分析ファンクションで使用する場合、式 (`expr`) が必要です。SIBLINGS キーワードは無効です（これは、階層問合せでのみ有効です）。位置 (`position`) および列別名 (`c_alias`) は無効です。それ以外で使用する場合、この `order_by_clause` は、問合せまたは副問合せ全体を順序付ける場合に使用するものと同じです。

ASC | DESC 順序付けシーケンス（昇順または降順）を指定します。デフォルトは ASC です。

NULLS FIRST | NULLS LAST NULL 値を含む戻された行が順序の最初にくるか、最後にくるかを指定します。

NULLS LAST は昇順のデフォルトで、NULLS FIRST は降順のデフォルトです。

注意： 分析ファンクションは、常に、ファンクションの `order_by_clause` で指定された順序で行を操作します。ただし、ファンクションの `order_by_clause` は結果の順序を保証しません。最終結果の順序を保証するには、問合せの `order_by_clause` を使用してください。

参照： この句の詳細は、18-23 ページの「[SELECT](#)」の「[order_by_clause](#)」を参照してください。

windowing_clause

一部の分析ファンクションでは、`windowing_clause` を使用できます。6-18 ページにある分析ファンクションの表では、`windowing_clause` を使用できるファンクションにアスタリスク (*) が付いています。

ROWS | RANGE これらのキーワードは、各行に対して、ファンクションの結果の計算に使用される「window」（行の物理集合または論理集合）を定義します。ファンクションは、window のすべての行に適用されます。window は、問合せ結果セット内またはパーティションの上から下までスライドします。

- ROWS は、物理単位（行）で window を指定します。
- RANGE は、論理オフセットとして window を指定します。

`order_by_clause` を指定しないと、この句を指定できません。

注意： 分析ファンクションが論理オフセットで戻す値は、常に決定的なものです。ただし、分析ファンクションが物理オフセットで戻す値は、順序式の結果が一意的順序にならないかぎり、非決定的な結果を生成することがあります。 `order_by_clause` に複数の列を指定して、結果の順序を一意的にする必要があります。

BETWEEN ...AND BETWEEN ... AND 句を使用すると、window にスタート・ポイントおよびエンド・ポイントを指定できます。最初の式 (AND の前) はスタート・ポイントを定義し、2 番目の式 (AND の後) はエンド・ポイントを定義します。

BETWEEN を省略してエンド・ポイントを 1 つのみ指定すると、Oracle はそれをスタート・ポイントとみなし、デフォルトでカレント行をエンド・ポイントに指定します。

UNBOUNDED PRECEDING UNBOUNDED PRECEDING を指定すると、パーティションの最初の行で、window が開始することを指定できます。これはスタート・ポイントの指定で、エンド・ポイントの指定としては使用できません。

UNBOUNDED FOLLOWING UNBOUNDED FOLLOWING を指定して、パーティションの最後の行で、window が終了することを指定できます。これはエンド・ポイントの指定で、スタート・ポイントの指定としては使用できません。

CURRENT ROW スタート・ポイントとして、window がカレント行または値 (それぞれ ROW または RANGE を指定したかどうかに基づく) で開始することを指定します。この場合、`value_expr PRECEDING` をエンド・ポイントにできません。

エンド・ポイントとして、window がカレント行または値 (それぞれ ROW または RANGE を明示的に指定したかどうかに基づく) で終了することを指定します。この場合、`value_expr FOLLOWING` をスタート・ポイントにできません。

`value_expr PRECEDING` または `value_expr FOLLOWING` RANGE または ROW に対して、次のことがいえます。

- `value_expr FOLLOWING` がスタート・ポイントの場合、エンド・ポイントは `value_expr FOLLOWING` である必要があります。
- `value_expr PRECEDING` がエンド・ポイントの場合、スタート・ポイントは `value_expr PRECEDING` である必要があります。

数値形式の時間間隔で定義されている論理ウィンドウを定義する場合、変換ファンクションを使用する必要があります。

参照： 数値時間から間隔への変換の詳細は、6-106 ページの「`NUMTOYMINTERVAL`」および 6-105 ページの「`NUMTODSINTERVAL`」を参照してください。

ROWS を指定した場合、次のことがいえます。

- `value_expr` は物理オフセットになります。これは定数または式であり、正数値に評価する必要があります。
- `value_expr` がスタート・ポイントの一部の場合、エンド・ポイントの前にある行に評価する必要があります。

RANGE を指定した場合、次のことがいえます。

- `value_expr` は論理オフセットになります。これは、正数値または期間リテラルに評価する定数または式である必要があります。

参照： 期間リテラルの詳細は、2-51 ページの「[リテラル](#)」を参照してください。

- `order_by_clause` には、式を 1 つのみ指定できます。
- `value_expr` が数値に対して評価を行う場合、ORDER BY `expr` は NUMBER または DATE データ型である必要があります。
- `value_expr` が間隔値に対して評価を行う場合、ORDER BY `expr` は DATE データ型である必要があります。

`windowing_clause` を完全に省略した場合、デフォルトで RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW になります。

分析ファンクションは、通常、データ・ウェアハウス環境で使用されます。分析ファンクションを次に示します。`windowing_clause` を含む完全な構文を使用できるファンクションには、アスタリスク (*) が付いています。

AVG *
CORR *
COVAR_POP *
COVAR_SAMP *
COUNT *
CUME_DIST
DENSE_RANK
FIRST
FIRST_VALUE *
LAG
LAST
LAST_VALUE *
LEAD
MAX *
MIN *
NTILE
PERCENT_RANK
PERCENTILE_CONT

PERCENTILE_DISC
 RANK
 RATIO_TO_REPORT
 REGR_ (線形リグレーション) ファンクション *
 ROW_NUMBER
 STDDEV *
 STDDEV_POP *
 STDDEV_SAMP *
 SUM *
 VAR_POP *
 VAR_SAMP *
 VARIANCE *

参照： これらのファンクションおよびその使用方法の詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

オブジェクト参照ファンクション

オブジェクト参照ファンクションは、指定されたオブジェクト型のオブジェクトへの参照となる REF を操作します。オブジェクト参照ファンクションを次に示します。

Deref
 MAKE_REF
 REF
 REFTOHEX
 VALUE

参照： REF の詳細は、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

SQL ファンクションのリスト（アルファベット順）

ABS

構文

abs::=



用途

ABS は、 n の絶対値を戻します。

例

次の例では、-15 の絶対値を戻します。

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

Absolute

15

ACOS

構文

acos::=



用途

ACOS は、*n* のアーク・コサインを戻します。引数 *n* は -1 ～ 1 の範囲で、ファンクションによって戻される値は 0 ～ π (ラジアン) の範囲です。

例

次の例では、.3 のアーク・コサインを戻します。

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

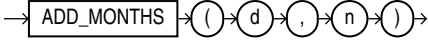
Arc_Cosine

1.26610367

ADD_MONTHS

構文

add_months::=



用途

ADD_MONTHS は、日付 *d* に *n* か月を加えて戻します。引数 *n* には、任意の整数を指定できます。*d* が月の最終日の場合、または結果の月の日数が *d* の日付コンポーネントよりも少ない場合、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

次の例では、サンプル表 employees の hire_date 後の月を戻します。

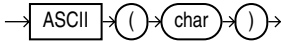
```
SELECT TO_CHAR(  
    ADD_MONTHS(hire_date,1),  
    'DD-MON-YYYY') "Next month"  
FROM employees  
WHERE last_name = 'Baer';
```

```
Next Month  
-----  
07-JUL-1994
```

ASCII

構文

ascii::=



用途

ASCII は、char の最初の文字の、データベース・キャラクタ・セットでの 10 進表記を戻します。

char のデータ型は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 です。戻り値のデータ型は NUMBER です。データベース・キャラクタ・セットが 7 ビットの ASCII の場合、このファンクションは ASCII 値を戻します。データベース・キャラクタ・セットが EBCDIC コードの場合、このファンクションは EBCDIC 値を戻します。このファンクションと一致する EBCDIC 文字ファンクションは存在しません。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、文字「Q」を ASCII の 10 進表記で戻します。

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

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

ASCIISTR

構文

asciistr::=



用途

ASCIISTR は、任意のキャラクタ・セットでの文字列を引数として、データベース・キャラクタ・セットでの ASCII 文字列を戻します。戻り値は、SQL で表示される文字およびスラッシュ (/) のみを含みます。ASCII 文字以外は、Unicode (UTF-16) のバイナリ・コード値に変換されます。

参照： Unicode キャラクタ・セットおよびキャラクタ・セマンティクスの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

例

次の例では、テキスト文字列「ABÄCDE」を ASCII 文字列で戻します。

```
SELECT ASCIISTR('ABÄCDE') FROM DUAL;

ASCIIS
-----
ABDCDE
```

ASIN

構文

asin::=



用途

ASIN は、 n のアーク・サインを戻します。引数 n は $-1 \sim 1$ の範囲で、ファンクションによって戻される値は $-\pi/2 \sim \pi/2$ (ラジアン) の範囲です。

例

次の例では、.3 のアーク・サインを戻します。

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

```

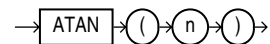
Arc_Sine
-----
.304692654

```

ATAN

構文

atan::=



用途

ATAN は、 n のアーク・タンジェントを戻します。引数 n の範囲に制限はなく、ファンクションによって戻される値は $-\pi/2 \sim \pi/2$ (ラジアン) の範囲です。

例

次の例では、.3 のアーク・タンジェントを戻します。

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

```

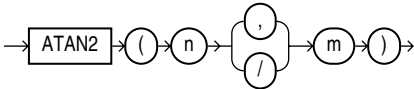
Arc_Tangent
-----
.291456794

```

ATAN2

構文

atan2::=



用途

ATAN2 は、 n および m のアーク・タンジェントを戻します。引数 n の範囲に制限はなく、 n と m の符号により、ファンクションによって戻される値は $-\pi \sim \pi$ (ラジアン) の範囲です。ATAN2(n, m) は、ATAN2(n/m) と同じです。

例

次の例では、.3 および .2 のアーク・タンジェントを戻します。

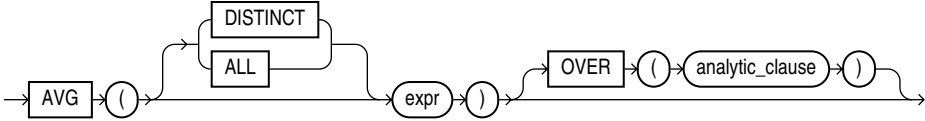
```
SELECT ATAN2(.3, .2) "Arc_Tangent2" FROM DUAL;
```

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

AVG

構文

avg::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

AVG は、*expr* の平均値を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。 *order_by_clause* および *windowing_clause* は指定できません。

参照：

- 6-7 ページの「集計ファンクション」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「SQL 式」を参照してください。

集計の例

次の例では、hr.employees 表にあるすべての従業員の平均給与を計算します。

```
SELECT AVG(salary) "Average" FROM employees;
```

```
Average
-----
      6425
```

分析の例

次の例では、employees 表の各従業員について、ある期間内に雇用された従業員の平均給与を所属別に計算します。

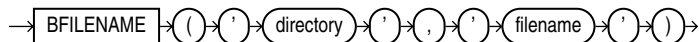
```
SELECT manager_id, last_name, hire_date, salary,
       AVG(salary) OVER (PARTITION BY manager_id ORDER BY hire_date
                          ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS c_mavg
FROM employees;
```

MANAGER_ID	LAST_NAME	HIRE_DATE	SALARY	C_MAVG
100	Kochhar	21-SEP-89	17000	17000
100	De Haan	13-JAN-93	17000	15000
100	Raphaely	07-DEC-94	11000	11966.6667
100	Kaufling	01-MAY-95	7900	10633.3333
100	Hartstein	17-FEB-96	13000	9633.33333
100	Weiss	18-JUL-96	8000	11666.6667
100	Russell	01-OCT-96	14000	11833.3333
.				
.				
.				

BFILENAME

構文

bfilename::=



用途

BFILENAME は、サーバーのファイル・システムの物理 LOB バイナリ・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' は、実際にファイルが存在するサーバーのファイル・システム上のフルパス名に対する別名となるデータベース・オブジェクトです。
- 'filename' は、サーバーのファイル・システムにあるファイルの名前です。

SQL 文、PL/SQL 文、DBMS_LOB パッケージまたは OCI の操作で、これらを BFILENAME への引数として使用するには、まずディレクトリ・オブジェクトを作成し、BFILE 値を物理ファイルと対応付ける必要があります。

次の 2 つの方法で、このファンクションを使用できます。

- DML 文で BFILE 列を初期化する場合
- プログラム・インタフェースで、BFILE ロケータに値を割り当てることによって BFILE データにアクセスする場合

ディレクトリの引数の大 / 小文字は区別されます。そのため、データ・ディクショナリ内に存在する名前と同じ名前ディレクトリ・オブジェクト名を指定しているかを確認する必要があります。たとえば、CREATE DIRECTORY 文で、大 / 小文字を組み合わせた識別子を引用符で囲んで使用して Admin ディレクトリを作成すると、BFILENAME ファンクションを使用する場合、ディレクトリ・オブジェクトを 'Admin' として指定する必要があります。ご使用のオペレーティング・システムの表記規則に従って、ファイル名の引数を完全に指定する必要があります。

参照：

- LOB の詳細および BFILE データの検出例については、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』および『Oracle Call Interface プログラマーズ・ガイド』を参照してください。
- 13-46 ページの「[CREATE DIRECTORY](#)」を参照してください。

例

次の例では、サンプル表 `pm.print_media` に行を挿入します。BFILENAME を使用して、サーバーのファイル・システムのバイナリ・ファイルを識別します。

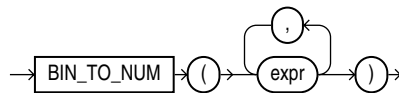
```
CREATE DIRECTORY media_dir AS '/demo/schema/product_media';

INSERT INTO print_media (product_id, ad_id, ad_graphic)
VALUES (3000, 31001,
        bfilename('MEDIA_DIR', 'modem_comp_ad.gif'));
```

BIN_TO_NUM

構文

`bin_to_num::=`



用途

BIN_TO_NUM は、ビット・ベクトルを同等の数値に変換します。このファンクションの各引数は、ビット・ベクトルのビットを表します。各 `expr` は、0 または 1 に評価される必要があります。このファンクションは Oracle の数値を戻します。

BIN_TO_NUM は、データ・ウェアハウスのアプリケーションで、グルーピング・セットを使用して、マテリアライズド・ビューから対象グループを検索する場合に有効です。

参照：

- GROUPING SETS 構文の詳細は、18-21 ページの「[group_by_clause](#)」を参照してください。
- データ集計の詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

例

次の例では、バイナリの値を数値に変換します。

```
SELECT BIN_TO_NUM(1,0,1,0) FROM DUAL;
```

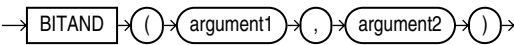
```
BIN_TO_NUM(1,0,1,0)
```

```
-----
```

BITAND

構文

bitand::=



用途

BITAND は、構成要素 *argument1*、*argument2* のビットに対する AND 操作の計算をします。*argument1* および *argument2* は、負以外の整数に変換される必要があり、整数を戻します。次に示すとおり、このファンクションは、一般的に DECODE ファンクションで使用されます。

注意： このファンクションは、戻り値のデータ型を判断しません。このため、SQL*Plus では、TO_NUMBER などのデータ型を戻すラッパーの BITAND を指定する必要があります。

例

次の例は、サンプル表 `oe.orders` の `order_status` をビットごとに表示します。(この例では、オプションの合計が 7 であると指定されているため、`order_status` が 7 を超える行は排除されます。)

```
SELECT order_id, customer_id,
       DECODE(BITAND(order_status, 1), 1, 'Warehouse', 'PostOffice')
         Location,
       DECODE(BITAND(order_status, 2), 2, 'Ground', 'Air') Method,
       DECODE(BITAND(order_status, 4), 4, 'Insured', 'Certified') Receipt
FROM orders
WHERE order_status < 8;
```

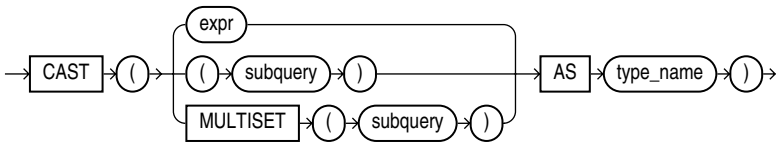
ORDER_ID	CUSTOMER_ID	LOCATION	MET	RECEIPT
2458	101	Postoffice	Air	Certified
2397	102	Warehouse	Air	Certified
2454	103	Warehouse	Air	Certified
2354	104	Postoffice	Air	Certified
2358	105	Postoffice	G	Certified
2381	106	Warehouse	G	Certified
2440	107	Warehouse	G	Certified
2357	108	Warehouse	Air	Insured

2394	109 Warehouse	Air	Insured
2435	144 Postoffice	G	Insured
2455	145 Warehouse	G	Insured
.			
.			
.			

CAST

構文

cast::=



用途

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

名前のないオペランド（日付や副問合せの結果セットなど）または名前付きのコレクション（VARRAY やネストした表など）を型互換の名前付きコレクションにキャストできます。
type_name は、組込みデータ型またはコレクション型の名前である必要があり、オペランドは、組込みデータ型であるか、またはその値がコレクション値である必要があり

オペランドでは、*expr* は組込みデータ型またはコレクション型のいずれかで、*subquery* がコレクション型または組込み型の単一値を戻す必要があります。MULTiset は、副問合せの結果セットをとり、コレクション値を戻すように Oracle に知らせます。表 6-1 に、どの組込みデータ型が、どの組込みデータ型にキャストできるかを示します（CAST は、LONG 型、LONG RAW 型、すべての LOB データ型または Oracle が提供する型をサポートしていません）。

表 6-1 組み込みデータ型のキャスト

	CHAR、 VARCHAR2 から	NUMBER から	DATETIME / INTERVAL ^b から	RAW から	ROWID、 UROWID から	NCHAR、 NVARCHAR2 から
CHAR、 VARCHAR2 へ	X	X	X	X	X	—
NUMBER へ	X	X	—	—	—	—
DATE、 TIMESTAMP、 INTERVAL へ	X	—	X	—	—	—
RAW へ	X	—	—	X	—	—
ROWID、 UROWID へ	X	—	—	—	X ^a	—
NCHAR、 NVARCHAR2 へ	—	X	X	X	X	X

^a UROWID が索引構成表の ROWID の値を含んでいる場合、UROWID を ROWID にキャストすることはできません。

^b Datetime/Interval には、DATE、TIMESTAMP、TIMESTAMP WITH TIMEZONE、INTERVAL DAY TO SECOND および INTERVAL YEAR TO MONTH が含まれます。

名前付きコレクション型を別の名前付きコレクション型にキャストするには、両方のコレクションの要素が同じ型である必要があります。

副問合せの結果セット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コレクション値の要素を形成します。MULTISET キーワードを省略すると、副問合せはスカラー副問合せとして処理されます。

組み込みデータ型の例

次の例では、CAST ファンクションをスカラー・データ型とともに使用します。

```
SELECT CAST('22-OCT-1997' AS TIMESTAMP WITH LOCAL TIME ZONE)
       FROM dual;

SELECT product_id,
       CAST(ad_sourcetext AS VARCHAR2(30))
       FROM print_media;
```

コレクションの例

後続く CAST の例は、サンプルの注文入力スキーマ oe で使用されている cust_address_typ に基づいています。

```
CREATE TYPE address_book_t AS TABLE OF cust_address_typ;
/
CREATE TYPE address_array_t AS VARRAY(3) OF cust_address_typ;
/
CREATE TABLE cust_address (
    custno          NUMBER,
    street_address  VARCHAR2(40),
    postal_code     VARCHAR2(10),
    city            VARCHAR2(30),
    state_province  VARCHAR2(10),
    country_id      CHAR(2));
```

```
CREATE TABLE cust_short (custno NUMBER, name VARCHAR2(31));
```

```
CREATE TABLE states (state_id NUMBER, addresses address_array_t);
```

次の例では、副問合せをキャストします。

```
SELECT s.custno, s.name,
       CAST(MULTISET(SELECT ca.street_address,
                           ca.postal_code,
                           ca.city,
                           ca.state_province,
                           ca.country_id
                       FROM cust_address ca
                       WHERE s.custno = ca.custno)
           AS address_book_t)
FROM cust_short s;
```

CAST では、VARRAY 型の列をネストした表に変換します。

```
SELECT CAST(s.addresses AS address_book_t)
FROM states s
WHERE s.state_id = 111;
```

次のオブジェクトは、この後に示す例で使用するオブジェクトを作成します。

```
CREATE TABLE projects
    (employee_id NUMBER, project_name VARCHAR2(10));

CREATE TABLE emps_short
    (employee_id NUMBER, last_name VARCHAR2(10));
```

```
CREATE TYPE project_table_typ AS TABLE OF VARCHAR2(10);  
/
```

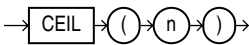
次の MULTISET 式の例は、前述のオブジェクトを使用します。

```
SELECT e.last_name,  
       CAST(MULTISET(SELECT p.project_name  
                     FROM projects p  
                     WHERE p.employee_id = e.employee_id  
                     ORDER BY p.project_name)  
            AS project_table_typ)  
FROM emp_short e;
```

CEIL

構文

ceil::=



用途

CEIL は、*n* 以上の最も小さい整数を戻します。

例

次の例では、15.7 以上である最小の整数を戻します。

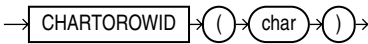
```
SELECT CEIL(15.7) "Ceiling" FROM DUAL;
```

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

CHARTOROWID

構文

chartorowid::=



用途

CHARTOROWID は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の値を ROWID データ型に変換します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「データ型の比較規則」を参照してください。

例

次の例では、文字表記の ROWID を ROWID に変換します。（このファンクションは、異なるデータベースの異なる ROWID を戻します。）

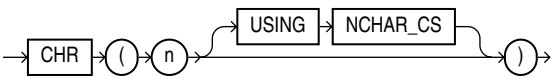
```
SELECT last_name FROM employees
       WHERE ROWID = CHARTOROWID('AAAFd1AAFAAAAABSAA/');
```

```
LAST_NAME
-----
Greene
```

CHR

構文

chr::=



用途

CHR は、データベース・キャラクタ・セットまたは各国語キャラクタ・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 が指定されていない場合、このファンクションは、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 が指定されている場合、このファンクションは、各国語キャラクタ・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。

シングルバイト・キャラクタ・セットの場合、Oracle は、 $n > 256$ に対して $n \bmod 256$ に等しい 2 進数を戻します。マルチバイト・キャラクタ・セットの場合、 n は 1 つのコードポイント全体として解決される必要があります。無効なコードポイントは検証されないため、無効なコードポイントを指定した場合の結果は、予測不能です。

注意：（オプションの USING NCHAR_CS 句の有無にかかわらず）CHR ファンクションを使用すると、ASCII および EBCDIC ベースのマシン・アーキテクチャ間で移植不可能なコードが戻されます。

参照： 6-94 ページの「[NCHR](#)」を参照してください。

例

次の例は、データベース・キャラクタ・セットが WE8ISO8859P1 と定義されている ASCII ベースのマシンで実行されています。

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

```
Dog
---
CAT
```

キャラクタ・セットが WE8EBCDIC1047 の EBCDIC ベースのマシンでも同じ結果を戻すには、前述の例を次のように修正する必要があります。

```
SELECT CHR(195) || CHR(193) || CHR(227) "Dog"
FROM DUAL;
```

```
Dog
---
CAT
```

マルチバイト・キャラクタ・セットの場合、このように連結しても異なる結果となります。たとえば、a1a2（a1 は 1 つ目のバイト、a2 は 2 つ目のバイト）という 16 進数の値を持つマルチバイト文字の場合、 n に対して「a1a2」に等しい 2 進数または 41378 を指定する必要があります。つまり、次のように指定する必要があります。

```
SELECT CHR(41378) FROM DUAL;
```

次の例のように、a2 に等しい 2 進数と連結された a1 に等しい 2 進数を指定することはできません。

```
SELECT CHR(161) || CHR(162) FROM DUAL;
```

ただし、次の例のように、マルチバイト文字を連結するマルチバイトのコードポイント全体を連結することは可能です。この例では、a1a2 と a1a3 の 16 進数を持つマルチバイト文字を連結しています。

```
SELECT CHR(41378) || CHR(41379) FROM DUAL;
```

次の例では、UTF8 キャラクタ・セットを使用します。

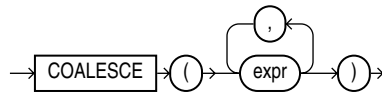
```
SELECT CHR(50052 USING NCHAR_CS) FROM DUAL;
```

```
CH
--
Ä
```

COALESCE

構文

coalesce::=



用途

COALESCE は、式のリストの最初の NULL でない *expr* を戻します。1 つ以上の *expr* が、リテラル NULL 以外である必要があります。すべての *expr* が NULL と評価された場合、このファンクションは NULL を戻します。

このファンクションは NVL ファンクションを一般化したファンクションです。

COALESCE は、CASE 式の変形として使用できます。次に例を示します。

```
COALESCE (expr1, expr2)
```

これは、次の CASE 式と同じです。

```
CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END
```

次の例も同様です。

```
COALESCE (expr1, expr2, ..., exprn), for n>=3
```

これは、次の CASE 式と同じです。

```
CASE WHEN expr1 IS NOT NULL THEN expr1
      ELSE COALESCE (expr2, ..., exprn) END
```

参照： 6-107 ページの「**NVL**」 および 4-6 ページの「**CASE 式**」を参照してください。

例

次の例では、サンプル表 `oe.product_information` を使用して、製品の「クリアランス・セール」を企画します。製品の表示価格から 10% 値引きします。表示価格がない場合は、最小価格はセール価格となります。最小価格がない場合、セール価格は「5」となります。

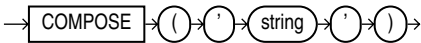
```
SELECT product_id, list_price, min_price,  
       COALESCE(0.9*list_price, min_price, 5) "Sale"  
FROM product_information  
WHERE supplier_id = 102050;
```

PRODUCT_ID	LIST_PRICE	MIN_PRICE	Sale
2382	850	731	765
3355			5
1770		73	73
2378	305	247	274.5
1769	48		43.2

COMPOSE

構文

`compose::=`



用途

COMPOSE は、引数として任意のデータ型の文字列を取り、入力されたものと同じキャラクタ・セットで、完全に正規化された形式の **Unicode** 文字列を戻します。*string* のデータ型は、**CHAR**、**VARCHAR2**、**NCHAR**、**NVARCHAR2**、**CLOB** または **NCLOB** です。たとえば、**o** ウムラウト・コードポイントは、ウムラウト・コードポイントによって修飾された「**o**」コードポイントとして戻されます。

参照： **Unicode** キャラクタ・セットおよびキャラクタ・セマンティクスの詳細は、『**Oracle9i データベース概要**』を参照してください。

例

次の例は、o ウムラウトのコードポイントを戻します。

```
SELECT COMPOSE ( 'o' || UNISTR('\0308') ) FROM DUAL;
```

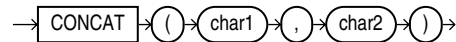
```
CO
--
ö
```

参照： 6-187 ページの「[UNISTR](#)」を参照してください。

CONCAT

構文

concat::=



用途

CONCAT は、*char2* に連結されている *char1* を戻します。*char1* および *char2* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。*char1* と同じキャラクタ・セットの文字列が戻されます。そのデータ型は、引数のデータ型によって決まります。

2 つの異なるデータ型を連結すると、可逆式変換となるデータ型が戻されます。したがって、引数の 1 つが LOB の場合、戻り値は LOB となります。引数の 1 つが各国語データ型の場合は、戻り値は各国語データ型となります。たとえば、次のようになります。

- CONCAT (CLOB,NCLOB) は NCLOB を戻します。
- CONCAT (NCLOB,NCHAR) は NCLOB を戻します。
- CONCAT (NCLOB,CHAR) は NCLOB を戻します。
- CONCAT (NCHAR,CLOB) は NCLOB を戻します。

このファンクションは、連結演算子 (||) と同等です。

参照： CONCAT 演算子の詳細は、3-4 ページの「[連結演算子](#)」を参照してください。

例

次の例では、ネストを使用して 3 つの文字列を連結します。

```
SELECT CONCAT(CONCAT(last_name, ''s job category is '),
              job_id) "Job"
FROM employees
WHERE employee_id = 152;
```

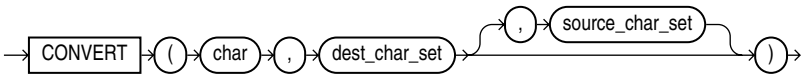
Job

Hall's job category is SA_REP

CONVERT

構文

convert::=



用途

CONVERT は、文字列を、あるキャラクタ・セットから別のキャラクタ・セットに変換します。戻り値のデータ型は VARCHAR2 です。

- 引数 *char* は変換する値です。*char* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。
- 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・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する必要があります。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

次の例では、Latin-1 文字列を ASCII に変換するキャラクタ・セットの変換を示します。これは、同じ文字列を WE8ISO8859P1 データベースから US7ASCII データベースへインポートした場合と同じ結果が得られます。

```
SELECT CONVERT('Ä Ê Í Õ Ø Å B C D E ', 'US7ASCII', 'WE8ISO8859P1')
        FROM DUAL;
```

```
CONVERT('ÄÊÍÕØABCDE'
-----
A E I ? ? A B C D E ?
```

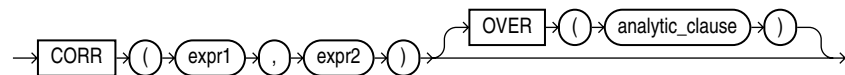
一般的なキャラクタ・セットを次に示します。

- US7ASCII: US7 ビット ASCII キャラクタ・セット
- WE8DEC: 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

CORR

構文

corr::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

CORR は、数値の組の集合に対する相関係数を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

`expr1` および `expr2` は数値式です。Oracle は、`expr1` または `expr2` が NULL である組を排除した後、このファンクションを (`expr1`, `expr2`) の集合に適用します。その後、Oracle は次の計算を行います。

```
COVAR_POP(expr1, expr2) / (STDDEV_POP(expr1) * STDDEV_POP(expr2))
```

ファンクションは、NUMBER 型の値を返します。ファンクションが空の集合に適用されると、NULL を返します。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- `expr` の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル・ビュー `oe.products` の重さクラスごとの製品の表示価格と最小価格の相関係数を計算します。

```
SELECT weight_class, CORR(list_price, min_price)
FROM product_information
GROUP BY weight_class;
```

WEIGHT_CLASS	CORR (LIST_PRICE, MIN_PRICE)
1	.99914795
2	.999022941
3	.998484472
4	.999359909
5	.999536087

分析の例

次の例では、1998 年度の `sh.sales` および `sh.times` サンプル表から、1 か月の売上利益および 1 か月の売上個数の相関累積係数を計算します。

```
SELECT t.calendar_month_number,
CORR (SUM(s.amount_sold), SUM(s.quantity_sold))
OVER (ORDER BY t.calendar_month_number) as CUM_CORR
FROM sales s, times t
WHERE s.time_id = t.time_id AND calendar_year = 1998
GROUP BY t.calendar_month_number
ORDER BY t.calendar_month_number;
```

CALENDAR_MONTH_NUMBER	CUM_CORR
1	
2	1


```

3 .994309382
4 .852040875
5 .846652204
6 .871250628
7 .910029803
8 .917556399
9 .920154356
10 .86720251
11 .844864765
12 .903542662

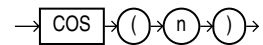
```

相関ファンクションには、操作する行が 1 つ以上必要です。そのため、前述の例にある最初の行には、計算する値はありません。

COS

構文

COS::=



用途

COS は、 n （ラジアンで表された角度）のコサインを戻します。

例

次の例では、180 度のコサインを戻します。

```

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

```

```

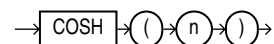
Cosine of 180 degrees
-----
-1

```

COSH

構文

cosh::=



用途

COSH は、*n* の双曲線コサインを戻します。

例

次の例では、0 の双曲線コサインを戻します。

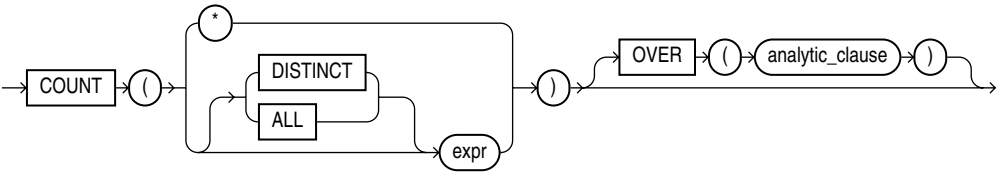
```
SELECT COSH(0) "Hyperbolic cosine of 0" FROM DUAL;
```

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

COUNT

構文

count::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

COUNT は、問合せ内の行数を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。*order_by_clause* および *windowing_clause* は指定できません。

expr を指定すると、COUNT は *expr* が NULL でない行数を戻します。*expr* のすべての行を数えるか、または異なる値のみを数えることができます。

アスタリスク (*) を指定すると、このファンクションは重複値および NULL 値を含むすべての行を戻します。COUNT は NULL を戻しません。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- `expr` の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、COUNT を集計ファンクションとして使用します。

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

```

      Total
-----
      107

```

```
SELECT COUNT(*) "Allstars" FROM employees
      WHERE commission_pct > 0;
```

```

    Allstars
-----
        35

```

```
SELECT COUNT(commission_pct) "Count" FROM employees;
```

```

      Count
-----
        35

```

```
SELECT COUNT(DISTINCT manager_id) "Managers" FROM employees;
```

```

    Managers
-----
        18

```

分析の例

次の例では、employees 表の各従業員について、その従業員の給与より 50 ドル少ない金額から 150 ドル多い金額の範囲の給与を得ている従業員の数を計算します。

```
SELECT last_name, salary,
      COUNT(*) OVER (ORDER BY salary RANGE BETWEEN 50 PRECEDING
      AND 150 FOLLOWING) AS mov_count FROM employees;
```

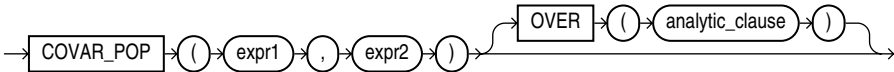
LAST_NAME	SALARY	MOV_COUNT
Olson	2100	3
Markle	2200	2
Philtanker	2200	2

Landry	2400	8
Gee	2400	8
Colmenares	2500	10
Patel	2500	10
.		
.		
.		

COVAR_POP

構文

covar_pop::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

COVAR_POP は、数値の組の集合に対する母集団共分散を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr1 および expr2 は数値式です。Oracle は、expr1 または expr2 が NULL であるすべての組を排除した後、このファンクションを (expr1 , expr2) の集合に適用します。その後、Oracle は次の計算を行います。

$$(SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / n$$

ここで、n は (expr1 , expr2) の組の数です (ただし、expr1 および expr2 の両方が NULL ではない場合です)。

ファンクションは、NUMBER 型の値を戻します。ファンクションが空の集合に適用されると、NULL を戻します。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 sh.sales から各年度における売上利益高および売上個数の母集団共分散を計算します。

```
SELECT t.calendar_month_number,
       COVAR_POP(s.amount_sold, s.quantity_sold) AS covar_pop,
       COVAR_SAMP(s.amount_sold, s.quantity_sold) AS covar_samp
FROM sales s, times t
WHERE s.time_id = t.time_id
AND t.calendar_year = 1998
GROUP BY t.calendar_month_number;
```

CALENDAR_MONTH_NUMBER	COVAR_POP	COVAR_SAMP
1	5437.68586	5437.88704
2	5923.72544	5923.99139
3	6040.11777	6040.38623
4	5946.67897	5946.92754
5	5986.22483	5986.4463
6	5726.79371	5727.05703
7	5491.65269	5491.9239
8	5672.40362	5672.66882
9	5741.53626	5741.80025
10	5050.5683	5050.78195
11	5256.50553	5256.69145
12	5411.2053	5411.37709

分析の例

次の例では、デモ・スキーマ oe の製品の表示価格および最小価格の累積標本共分散を計算します。

```
SELECT product_id, supplier_id,
       COVAR_POP(list_price, min_price)
       OVER (ORDER BY product_id, supplier_id)
       AS CUM_COVP,
       COVAR_SAMP(list_price, min_price)
       OVER (ORDER BY product_id, supplier_id)
       AS CUM_COVS
FROM product_information p
WHERE category_id = 29
ORDER BY product_id, supplier_id;
```

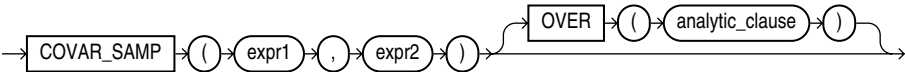
PRODUCT_ID	SUPPLIER_ID	CUM_COVP	CUM_COVS
1774	103088	0	
1775	103087	1473.25	2946.5

1794	103096	1702.77778	2554.16667
1825	103093	1926.25	2568.33333
2004	103086	1591.4	1989.25
2005	103086	1512.5	1815
2416	103088	1475.97959	1721.97619
.			
.			
.			

COVAR_SAMP

構文

covar_samp::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

COVAR_SAMP は、数値の組の集合の標本共分散を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr1 および *expr2* は数値式です。Oracle は、*expr1* または *expr2* が NULL であるすべての組を排除した後、このファンクションを (*expr1*, *expr2*) の集合に適用します。その後、Oracle は次の計算を行います。

$$(\text{SUM}(\text{expr1} * \text{expr2}) - \text{SUM}(\text{expr1}) * \text{SUM}(\text{expr2}) / n) / (n-1)$$

ここで、*n* は (*expr1*, *expr2*) の組の数です (ただし、*expr1* および *expr2* の両方が NULL ではない場合です)。

ファンクションは、NUMBER 型の値を戻します。ファンクションが空の集合に適用されると、NULL を戻します。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 sh.sales から各年度における売上利益高および売上個数の母集団共分散を計算します。

```
SELECT t.calendar_month_number,
       COVAR_POP(s.amount_sold, s.quantity_sold) AS covar_pop,
       COVAR_SAMP(s.amount_sold, s.quantity_sold) AS covar_samp
FROM sales s, times t
WHERE s.time_id = t.time_id
AND t.calendar_year = 1998
GROUP BY t.calendar_month_number;
```

CALENDAR_MONTH_NUMBER	COVAR_POP	COVAR_SAMP
1	5437.68586	5437.88704
2	5923.72544	5923.99139
3	6040.11777	6040.38623
4	5946.67897	5946.92754
5	5986.22483	5986.4463
6	5726.79371	5727.05703
7	5491.65269	5491.9239
8	5672.40362	5672.66882
9	5741.53626	5741.80025
10	5050.5683	5050.78195
11	5256.50553	5256.69145
12	5411.2053	5411.37709

分析の例

次の例では、デモ・スキーマ oe の製品の表示価格および最小価格の累積標本共分散を計算します。

```
SELECT product_id, supplier_id,
       COVAR_POP(list_price, min_price)
       OVER (ORDER BY product_id, supplier_id)
       AS CUM_COVP,
       COVAR_SAMP(list_price, min_price)
       OVER (ORDER BY product_id, supplier_id)
       AS CUM_COVS
FROM product_information p
WHERE category_id = 29
ORDER BY product_id, supplier_id;
```

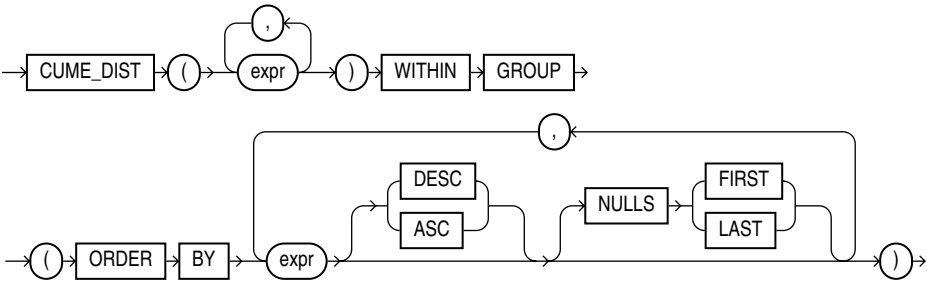
PRODUCT_ID	SUPPLIER_ID	CUM_COVP	CUM_COVS
1774	103088	0	
1775	103087	1473.25	2946.5

1794	103096	1702.77778	2554.16667
1825	103093	1926.25	2568.33333
2004	103086	1591.4	1989.25
2005	103086	1512.5	1815
2416	103088	1475.97959	1721.97619
.			
.			
.			

CUME_DIST

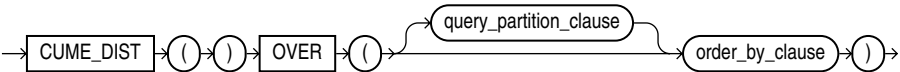
集計の構文

cume_dist_aggregate::=



分析の構文

cume_dist_analytic::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

CUME_DIST は、値のグループにある値の累積分布値を計算します。CUME_DIST が戻す値の範囲は、0 より大きく 1 以下です。連結値は、常に同じ累積分布値に対して評価を行います。

- 集計ファンクションとしての CUME_DIST は、ファンクションの引数および対応するソート指定によって識別される不確定な行 R に対して、集計グループ内の行の中で行 R の関連する位置を計算します。Oracle は、不確定な行 R を集計される行のグループに挿入するように計算します。このファンクションの引数は、各集計グループ内の 1 つの不確定行を識別します。このため、すべての引数は、集計グループ内で定数式と評価される必要があります。定数引数式および集計の ORDER BY 句の式の位置は、一致します。このため、引数の数は同じであり、その型は互換性がある必要があります。
- 分析ファンクションとしての CUME_DIST は、値のグループにある特定の値の相対位置を計算します。行 R について、昇順で順序付けられているとします。R の CUME_DIST は、R の値以下の値の行を、評価される行の数（問合せ結果セットまたはパーティション）で割った数です。

集計の例

次の例では、サンプル表 oe.employees の従業員の中から、給与が 15,500 ドルであり、歩合が 5% の不確定な従業員の累積分布を計算します。

```
SELECT CUME_DIST(15500, .05) WITHIN GROUP
  (ORDER BY salary, commission_pct) "Cume-Dist of 15500"
FROM employees;
```

```
Cume-Dist of 15500
-----
          .972222222
```

分析の例

次の例では、購入地域の各従業員の給与のパーセンタイルを計算します。たとえば、事務員の 40% が、Himuro の給与以下の給与を得ていることがわかります。

```
SELECT job_id, last_name, salary, CUME_DIST()
  OVER (PARTITION BY job_id ORDER BY salary) AS cume_dist
FROM employees
WHERE job_id LIKE 'PU%';
```

JOB_ID	LAST_NAME	SALARY	CUME_DIST
PU_CLERK	Colmenares	2500	.2
PU_CLERK	Himuro	2600	.4
PU_CLERK	Tobias	2800	.6
PU_CLERK	Baida	2900	.8
PU_CLERK	Khoo	3100	1
PU_MAN	Raphaely	11000	1

CURRENT_DATE

構文

current_date::=



用途

CURRENT_DATE は、セッション・タイムゾーンの現在の日付を DATE データ型のグレゴリオ暦の値で戻します。

例

次の例では、CURRENT_DATE がセッション・タイムゾーンによって異なることを示します。

```
ALTER SESSION SET TIME_ZONE = '-5:0';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_DATE
-----	-----
-05:00	29-MAY-2000 13:14:03

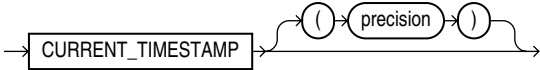
```
ALTER SESSION SET TIME_ZONE = '-8:0';
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_DATE
-----	-----
-08:00	29-MAY-2000 10:14:33

CURRENT_TIMESTAMP

構文

current_timestamp::=



用途

CURRENT_TIMESTAMP は、セッション・タイムゾーンの現在の日付および時刻を TIMESTAMP WITH TIME ZONE データ型の値で戻します。タイムゾーンによる時差は、SQL セッションの現在のローカル時刻を反映します。精度の指定を省略した場合のデフォルトは 6 です。このファンクションと LOCALTIMESTAMP との違いは、CURRENT_TIMESTAMP は、TIMESTAMP WITH TIME ZONE の値を戻し、LOCALTIMESTAMP は TIMESTAMP の値を戻す点です。

オプションの引数では、precision は、戻される時刻の値の小数部の精度を指定します。

例

次の例では、CURRENT_TIMESTAMP がセッション・タイムゾーンによって異なることを示します。

```
ALTER SESSION SET TIME_ZONE = '-5:0';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_TIMESTAMP
-05:00	04-APR-00 01.17.56.917550 PM -05:00

```
ALTER SESSION SET TIME_ZONE = '-8:0';
SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_TIMESTAMP
-08:00	04-APR-00 10.18.21.366065 AM -08:00

CURRENT_TIMESTAMP で書式マスクを使用する場合は、ファンクションが戻す値と書式マスクを一致させてください。たとえば、次の表の場合を考えます。

```
CREATE TABLE current_test (col1 TIMESTAMP WITH TIME ZONE);
```

ファンクションが戻す型の TIME ZONE の部分がマスクに含まれていないため、次の文は正常に実行されません。

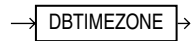
```
INSERT INTO current_test VALUES
  (TO_TIMESTAMP_TZ(CURRENT_TIMESTAMP, 'DD-MON-RR HH.MI.SSXFF PM'));
```

次の文では、CURRENT_TIMESTAMP の戻り値の型と一致する正しい書式マスクが使用されています。

```
INSERT INTO current_test VALUES (TO_TIMESTAMP_TZ
  (CURRENT_TIMESTAMP, 'DD-MON-RR HH.MI.SSXFF PM TZh:TzM'));
```

DBTIMEZONE

構文

dbtimezone::=

用途

DBTIMEZONE ファンクションは、データベース・タイムゾーンの値を戻します。戻り型は、タイムゾーン・オフセット ('[+|-] TZH:TZM' 書式の文字型) またはタイムゾーン地域名です。これは、最近の CREATE DATABASE または ALTER DATABASE 文でユーザーが指定したデータベース・タイムゾーンの値によって異なります。

例

次の例では、データベース・タイムゾーンが **UTC** タイムゾーンに設定されていると想定します。

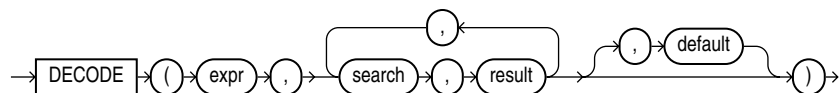
```
SELECT DBTIMEZONE FROM DUAL;
```

```
DBTIME
-----
-08:00
```

DECODE

構文

decode::=



用途

DECODE フังก์ションは、*expr* と各 *search* の値を 1 つずつ比較します。*expr* が *search* と等しい場合、Oracle は対応する *result* を戻します。一致する値が見つからない場合は、*default* を戻します。*default* が省略されている場合は、NULL を戻します。

`expr` および `search` に文字データが含まれている場合、Oracle は、非空白埋め比較セマンティクスを使用してそれらを比較します。`expr`、`search` および `result` のデータ型は、`CHAR`、`VARCHAR2`、`NCHAR` または `NVARCHAR2` です。戻される文字列は、`VARCHAR2` データ型で、最初の `result` パラメータと同じキャラクタ・セットの文字列です。

search、*result* および *default* の値は、式から導出できます。Oracle は、*search* 値のいずれかと *expr* を比較する前にすべての *search* 値を評価するのではなく、各 *search* 値と *expr* を比較する前にのみ、各 *search* 値を評価します。その結果、*expr* と等しい *search* が見つかったと、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 の場合、または最初の *result* が NULL の場合、Oracle は戻り値を VARCHAR2 データ型の値に変換します。

DECODE ファンクションでは、Oracle は 2 つの NULL を同等とみなします。*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 ファンクションのコンポーネントの最大数は、*expr*、*search*、*result*、*default* を含めて 255 です。

参照：

- 比較セマンティクスについては、2-42 ページの「[データ型の比較規則](#)」を参照してください。
- データ型変換については、2-46 ページの「[データ変換](#)」を参照してください。
- 暗黙的な変換のデメリットについては、2-46 ページの「[暗黙的なデータ変換と明示的なデータ変換](#)」を参照してください。

例

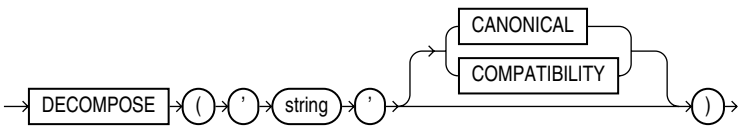
この例では、warehouse_id の値をデコードします。warehouse_id が 1 の場合「Southlake」を、warehouse_id が 2 の場合は「San Francisco」を戻します。warehouse_id が 1、2、3、4 のいずれでもない場合、ファンクションは「Non-domestic」を戻します。

```
SELECT product_id,
       DECODE (warehouse_id, 1, 'Southlake',
               2, 'San Francisco',
               3, 'New Jersey',
               4, 'Seattle',
               'Non-domestic')
       "Location of inventory" FROM inventories
WHERE product_id < 1775;
```

DECOMPOSE

構文

decompose::=



用途

DECOMPOSE は、Unicode キャラクタに対してのみ有効です。DECOMPOSE は、任意のデータ型の文字列を引数として取り、入力と同じキャラクタ・セットで分解された Unicode の文字列を返します。たとえば、**o** ウムラウト・コードポイント、**o** ウムラウト・コードポイントが続く「**o**」コードポイントとして戻されます。

- `string` は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。
- CANONICAL を使用すると標準的な分解が行われ、（たとえば、COMPOSE ファンクションを使用して）元の文字列への再構成が可能になります。これはデフォルト値です。
- COMPATIBILITY を使用すると互換モードで分解が行われます。このモードでは、再構成はできません。このモードは、半角および全角のカタカナ文字を分解する場合など、外的な書式またはスタイル情報なしに再構成を行うことが望ましくない場合に有効です。

参照： Unicode キャラクタ・セットおよびキャラクタ・セマンティクス
の詳細は、『Oracle9i データベース概要』を参照してください。

例

次の例では、文字列「Châteaux」をその要素のコードポイントに分解します。

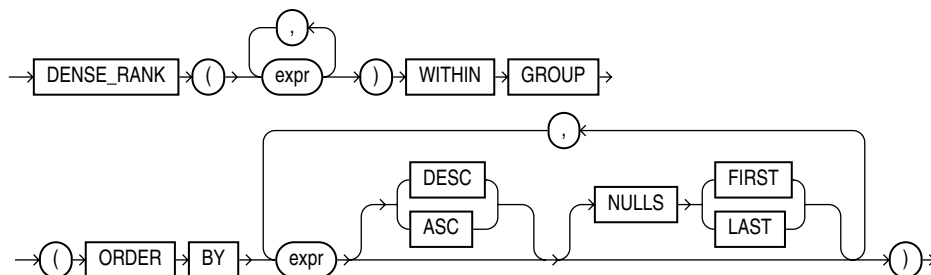
```
SELECT DECOMPOSE ('Châteaux') FROM DUAL;
```

```
DECOMPOSE
-----
Cha^teaux
```

DENSE_RANK

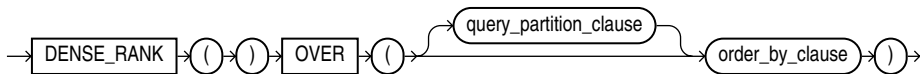
集計の構文

dense_rank_aggregate::=



分析の構文

dense_rank_analytic::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

DENSE_RANK は、順序付けされた行のグループの行のランクを計算します。ランクは 1 から始まる連続した整数です。ランクの最大値は、問合せが戻す一意の数値です。ランクの値は、連続した整数です。ランク付け基準と同じ値を持つ行は、同じランクになります。

- 集計ファンクションとして使用する DENSE_RANK は、ファンクションの引数によって識別される不確定な行の稠密ランクを、与えられたソート指定で計算します。ファンクションの引数は、各集計グループの単一行を識別するため、すべての引数は各集計グループ内で定数式に評価される必要があります。定数引数式および集計の *order_by_clause* の式の位置は、一致します。このため、引数の数は同じであり、型は互換性がある必要があります。
- 分析ファンクションとして使用する DENSE_RANK は、他の行について、問合せで戻される各行のランクを計算します。この計算は、*order_by_clause* にある *value_exprs* の値に基づいて行われます。

集計の例

次の例では、サンプル表 `oe.employees` から、給与が 15,500 ドルであり、歩合が 5% の仮定の従業員のランクを計算します。

```
SELECT DENSE_RANK(15500, .05) WITHIN GROUP
      (ORDER BY salary DESC, commission_pct) "Dense Rank"
FROM employees;
```

Dense Rank

3

分析の例

次の文は、HUMAN RESOURCES または PURCHASING 部門で働くすべての従業員の部門名、従業員名および給与を選択します。その後、この 2 つの部門それぞれについて、一意の各給与に対するランクを計算します。同じ給与は同じランクになります。この例と、6-116 ページの「[RANK](#)」の例を比較してください。

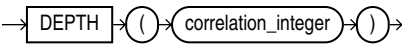
```
SELECT d.department_name, e.last_name, e.salary, DENSE_RANK()
      OVER (PARTITION BY e.department_id ORDER BY e.salary) as drank
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND d.department_id IN ('30', '40');
```

DEPARTMENT_NAME	LAST_NAME	SALARY	DRANK
-----	-----	-----	-----
Purchasing	Colmenares	2500	1
Purchasing	Himuro	2600	2
Purchasing	Tobias	2800	3
Purchasing	Baida	2900	4
Purchasing	Khoo	3100	5
Purchasing	Raphaely	11000	6
Human Resources	Marvis	6500	

DEPTH

構文

depth::=



用途

DEPTH は、UNDER_PATH および EQUALS_PATH 条件でのみ使用される補助ファンクションです。このファンクションは、同じ相関変数を持つ UNDER_PATH 条件によって指定されるパスのレベル数を返します。

correlation_integer は任意の整数です。文に複数の一次条件が含まれている場合に、この補助ファンクションをその一次条件と関連付けるために使用します。1 未満の値は 1 として扱われます。

参照：

- 5-13 ページの「EQUALS_PATH」および 5-20 ページの「UNDER_PATH」を参照してください。
- 6-109 ページの「PATH」を参照してください。

例

EQUALS_PATH および UNDER_PATH 条件は、2 つの補助ファンクションを取ることができ、その 1 つが DEPTH です。次に、その 2 つの補助ファンクションの使用方法を示します。この例では、XML Schema である warehouses.xsd (D-11 ページの「SQL 文での XML の使用方法」で作成) が存在することを前提としています。

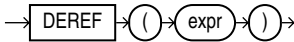
```
SELECT PATH(1), DEPTH(2)
FROM RESOURCE_VIEW
WHERE UNDER_PATH(res, '/sys/schemas/OE', 1)=1
      AND UNDER_PATH(res, '/sys/schemas/OE', 2)=1;
```

PATH(1)	DEPTH(2)
-----	-----
/www.oracle.com	1
/www.oracle.com/xwarehouses.xsd	2

DEREF

構文

deref::=



用途

DEREF は、引数 *expr* のオブジェクト参照を戻します。この場合、*expr* はオブジェクトに REF を戻す必要があります。問合せでこのファンクションを使用しない場合、次の例で示すとおり、かわりに REF のオブジェクト ID を戻します。

参照： 6-88 ページの「[MAKE_REF](#)」を参照してください。

例

サンプル・スキーマ oe は、オブジェクト型 cust_address_typ（作成方法は、この後の例で繰り返します）を含みます。次の例では、cust_address_typ_new 表を作成し、cust_address_typ への REF である 1 つの列を含む別の表を作成します。

```
CREATE TYPE cust_address_typ_new AS OBJECT
( street_address      VARCHAR2(40)
, postal_code         VARCHAR2(10)
, city                VARCHAR2(30)
, state_province      VARCHAR2(10)
, country_id          CHAR(2)
);
/
CREATE TABLE address_table OF cust_address_typ_new;

CREATE TABLE customer_addresses (
  add_id NUMBER,
  address REF cust_address_typ_new
  SCOPE IS address_table);

INSERT INTO address_table VALUES
  ('1 First', 'G45 EU8', 'Paris', 'CA', 'US');

INSERT INTO customer_addresses
  SELECT 999, REF(a) FROM address_table a;

SELECT address FROM customer_addresses;

ADDRESS
-----
000022020876B2245DBE325C5FE03400400B40DCB176B2245DBE305C5FE03400400B40DCB1

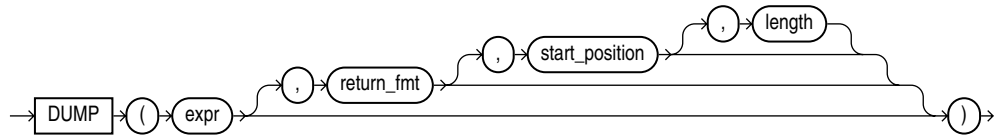
SELECT Deref(address) FROM customer_addresses;

Deref(ADDRESS) (STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID)
-----
CUST_ADDRESS_TYP('1 First', 'G45 EU8', 'Paris', 'CA', 'US')
```

DUMP

構文

`dump::=`



用途

DUMP は、`expr` のデータ型コード、長さ（バイト単位）および内部表現を含む VARCHAR2 値を返します。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、2-7 ページの表 2-1 を参照してください。

引数 `return_fmt` 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。

- 8 は、結果を 8 進表記で返します。
- 10 は、結果を 10 進表記で返します。
- 16 は、結果を 16 進表記で返します。
- 17 は、結果を単一文字として返します。

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、前述の書式のいずれかの値に 1000 を加えて指定します。たとえば、`return_fmt` 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。

引数 `start_position` と `length` を組み合わせて、内部表現の戻す部分を指定します。デフォルトでは、10 進表記で全体の内部表現が戻されます。

`expr` が NULL の場合、このファンクションは NULL を返します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、文字列式および列からダンプ情報を抽出する方法を示します。

```
SELECT DUMP('abc', 1016)
      FROM DUAL;
```

```
DUMP('ABC',1016)
-----
Typ=96 Len=3 CharacterSet=WE8DEC: 61,62,63
```

```
SELECT DUMP(last_name, 8, 3, 2) "OCTAL"
      FROM employees
      WHERE last_name = 'Hunold';
```

```
OCTAL
-----
Typ=1 Len=6: 156,157
```

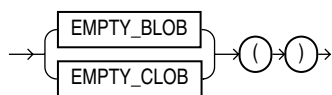
```
SELECT DUMP(last_name, 10, 3, 2) "ASCII"
      FROM employees
      WHERE last_name = 'Hunold';
```

```
ASCII
-----
Typ=1 Len=6: 110,111
```

EMPTY_BLOB、EMPTY_CLOB

構文

empty_lob::=



用途

EMPTY_BLOB および EMPTY_CLOB は、LOB 変数を初期化したり、INSERT または UPDATE 文で LOB 列または属性を EMPTY に初期化できる空の LOB ロケータを戻します。EMPTY とは、LOB は初期化されていても、データが移入されていない状態をいいます。

LOB ロケータの制限 このファンクション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することはできません。

例

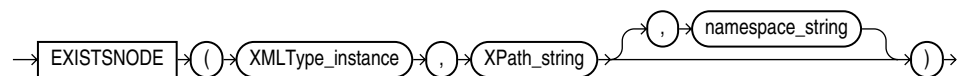
次の例では、サンプル表 `pm.print_media` の `ad_photo` 列を `EMPTY` に初期化します。

```
UPDATE print_media SET ad_photo = EMPTY_BLOB();
```

EXISTSNODE

構文

existsnode::=



用途

EXISTSNODE は、パスを使用してドキュメントをトラバースした後にノードが存在するかを判断します。XML 文書、およびパスを指定する XPath 文字列 VARCHAR2 を含む XMLType インスタンスを引数として指定します。オプションの `namespace_string` は、接頭辞のデフォルト・マッピングまたはネームスペース・マッピング（Oracle が XPath 式を評価する場合に使用）を指定する VARCHAR2 値に解決される必要があります。

戻り値は、NUMBER です。

- ドキュメントに XPath トラバースを適用した後にノードが存在しない場合は、0（ゼロ）が戻ります。
- ノードが存在する場合は、1 が戻ります。

例

次の例では、サンプル表 `oe.warehouses` の `warehouse_spec` 列の XML パスに `/Warehouse/Dock` ノードが存在するかを判断します。

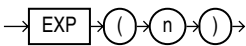
```
SELECT warehouse_id, warehouse_name
  FROM warehouses
 WHERE EXISTSNODE(warehouse_spec, '/Warehouse/Docks') = 1;
```

```
WAREHOUSE_ID WAREHOUSE_NAME
-----
1 Southlake, Texas
2 San Francisco
4 Seattle, Washington
```

EXP

構文

exp::=



用途

EXP は、e を *n* 乗した値（e = 2.71828183 ...）を戻します。

例

次の例では、e を 4 乗した値を戻します。

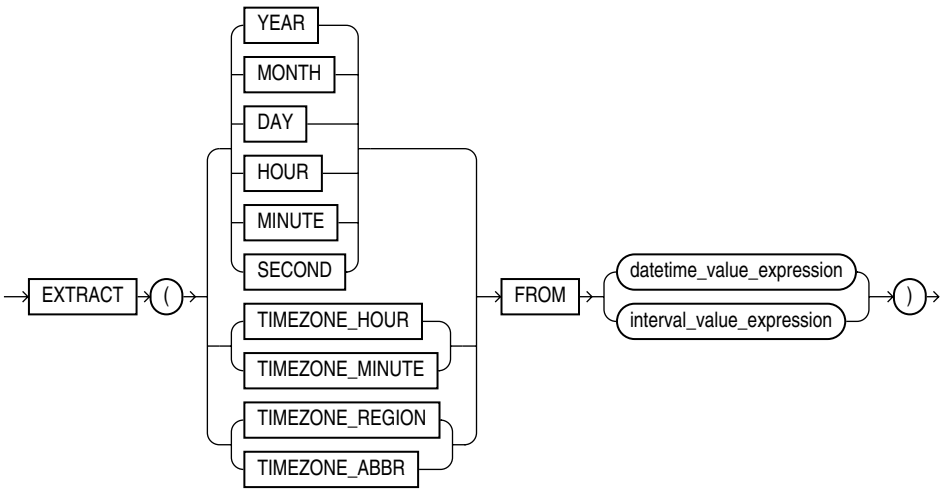
```
SELECT EXP(4) "e to the 4th power" FROM DUAL;
```

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

EXTRACT（日時）

構文

extract_datetime::=



用途

EXTRACT は、日時または期間値の式から、指定された日時フィールドの値を抽出し、戻します。TIMEZONE_REGION を抽出する場合、戻り値は、適切なタイムゾーン名を含む文字列です。また、TIMEZONE_ABBR (略称) を抽出する場合、戻り値は、適切な略称を含む文字列です。その他の値を抽出する場合、戻り値はグレゴリオ暦です。タイムゾーン値を持つ日時から抽出する場合、戻り値は UTC です。有効なタイムゾーン名およびそれに対する略称を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューに問合せを実行してください。

日時フィールドと日時または期間値の式を組み合わせると、あいまいな結果になる場合があります。この場合、Oracle は、'UNKNOWN' を戻します (詳細は、次の例を参照してください)。

注意： 抽出を実行するフィールドは、*datetime_value_expr* または *interval_value_expr* フィールドである必要があります。たとえば、DATE 値からは、YEAR、MONTH および DAY のみを抽出できます。同様に、TIMESTAMP WITH TIME ZONE データ型からは、TIMEZONE_HOUR および TIMEZONE_MINUTE のみを抽出できます。

参照：

- *datetime_value_expr* および *interval_value_expr* の詳細は、2-23 ページの「[日時および期間の演算](#)」を参照してください。
- 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

例

次の例では、1998 年を戻します。

```
SELECT EXTRACT(YEAR FROM DATE '1998-03-07') FROM DUAL;
```

```
EXTRACT(YEARFROMDATE'1998-03-07')
-----
1998
```

次の例では、サンプル表 hr.employees から、1998 以降に雇用されたすべての従業員を選択します。

```
SELECT last_name, employee_id, hire_date
       FROM employees
      WHERE EXTRACT(YEAR FROM
                    TO_DATE(hire_date, 'DD-MON-RR')) > 1998
      ORDER BY hire_date;
```

LAST_NAME	EMPLOYEE_ID	HIRE_DATE
Landry	127	14-JAN-99
Lorentz	107	07-FEB-99
Cabrio	187	07-FEB-99
.		
.		
.		

次の例では、結果があいまいになるため、Oracle は、'UNKNOWN' を戻します。

```
SELECT EXTRACT(TIMEZONE_REGION
              FROM TIMESTAMP '1999-01-01 10:00:00 -08:00')
       FROM DUAL;

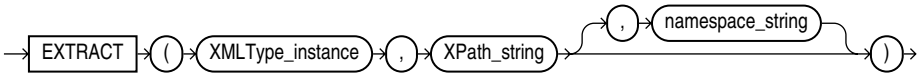
EXTRACT(TIMEZONE_REGIONFROMTIMESTAMP'1999-01-0110:00:00-08:00')
-----
UNKNOWN
```

タイムゾーン数値オフセットが式に指定され、その数値オフセットが複数のタイムゾーン地域をマップする場合、結果があいまいになります。

EXTRACT (XML)

構文

extract_xml::=



用途

EXTRACT (XML) は、EXISTSNODE ファンクションに似ています。VARCHAR2 の XPath 文字列を適用し、XML のフラグメントを含む XMLType インスタンスを戻します。オプションの namespace_string は、接頭辞のデフォルト・マッピングまたはネームスペース・マッピング（Oracle が XPath 式を評価する場合に使用）を指定する VARCHAR2 値に解決される必要があります。

例

次の例では、サンプル表 `oe.warehouses` の `warehouse_spec` 列の XML パスの `/Warehouse/Dock` ノードの値を抽出します。

```
SELECT warehouse_name, EXTRACT(warehouse_spec, '/Warehouse/Docks')
      "Number of Docks"
FROM warehouses
WHERE warehouse_spec IS NOT NULL;
```

WAREHOUSE_NAME	Number of Docks
-----	-----
Southlake, Texas	<Docks>2</Docks>
San Francisco	<Docks>1</Docks>
New Jersey	<Docks/>
Seattle, Washington	<Docks>3</Docks>

この例と、XML のフラグメントのスカラー値を戻す、6-61 ページの「[EXTRACTVALUE](#)」の例を比較してみてください。

EXTRACTVALUE

構文

`extractvalue::=`



EXTRACTVALUE は、引数として XMLType インスタンスと XPath 式を取り、結果として得られるノードのスカラー値を戻します。結果はシングルノードであり、テキスト・ノード、属性または要素のいずれかである必要があります。結果が要素である場合、その要素はシングル・テキスト・ノードを子ノードとして持つ必要があります、このファンクションが戻す値は、その子ノードの値になります。指定された XPath が複数の子ノードを持つノードを指す場合、または指定されたノードが非テキスト・ノードの子を持つ場合は、エラーが戻されます。オプションの `namespace_string` は、接頭辞のデフォルト・マッピングまたはネームスペース・マッピング（Oracle が XPath 式を評価する場合に使用）を指定する VARCHAR2 値に解決される必要があります。

XML Schema に基づくドキュメントで戻り値の型が推測できる場合は、適切な型のスカラー値が戻ります。その他の場合は、VARCHAR2 型の結果が戻ります。XML Schema に基づかないドキュメントの場合、戻り値は常に VARCHAR2 です。

例

次の例では、入力として、6-60 ページの「[EXTRACT \(XML\)](#)」の例と同じ引数を取ります。このファンクションは、`EXTRACT` ファンクションとは異なり、XML のフラグメントを戻すのではなく、XML のフラグメントのスカラー値を戻します。

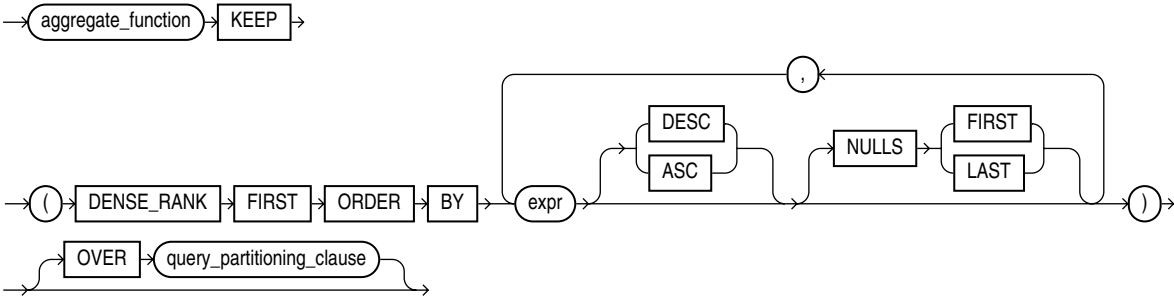
```
SELECT warehouse_name,  
       EXTRACTVALUE(e.warehouse_spec, '/Warehouse/Docks')  
       "Docks"  
FROM warehouses e  
WHERE warehouse_spec IS NOT NULL;
```

WAREHOUSE_NAME	Docks
-----	-----
Southlake, Texas	2
San Francisco	1
New Jersey	
Seattle, Washington	3

FIRST

構文

first::=



参照： `ORDER BY` 句および `OVER` 句の構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

`FIRST` ファンクションと `LAST` ファンクションは類似しています。両方のファンクションとも、与えられたソート指定に対して、`FIRST` または `LAST` としてランク付けされた一連の行の一連の値を操作する集計ファンクションおよび分析ファンクションです。1 つの行のみが `FIRST` または `LAST` としてランク付けされている場合、集計は、1 つの要素のみを持つセットを操作します。

ソートされたグループの最初または最後の行の値が必要であり、その値がソート・キーでない場合、FIRST および LAST ファンクションは、自己結合またはビューの必要性を排除し、パフォーマンスを向上させます。

- `aggregate_function` は、MIN、MAX、SUM、AVG、COUNT、VARIANCE または STDDEV ファンクションのいずれかです。FIRST または LAST のいずれかにランク付けされた行の値を操作します。1つの行のみが FIRST または LAST としてランク付けされている場合、集計は、単一（集計ではない）のセットを操作します。
- `DENSE_RANK FIRST` または `DENSE_RANK LAST` は、最小（FIRST）または最大（LAST）稠密ランク（「オリンピック・ランク」）を持つ行のみを集計することを示します。

OVER 句を指定すると、FIRST および LAST ファンクションを分析ファンクションとして使用できます。OVER 句のうち、`query_partitioning_clause` の部分のみがこれらのファンクションで有効です。

集計の例

次の例では、サンプル表 `hr.employees` の各部門で、歩合が最低である従業員の中での最低給与、および歩合が最高の従業員の中での最高給与を戻します。

```
SELECT department_id,
MIN(salary) KEEP (DENSE_RANK FIRST ORDER BY commission_pct) "Worst",
MAX(salary) KEEP (DENSE_RANK LAST ORDER BY commission_pct) "Best"
  FROM employees
 GROUP BY department_id;
```

DEPARTMENT_ID	Worst	Best
10	4400	4400
20	6000	13000
30	2500	11000
40	6500	6500
50	2100	8200
60	4200	9000
70	10000	10000
80	6100	14000
90	17000	24000
100	6900	12000
110	8300	12000
	7000	7000

分析の例

次の例では、前述の例と同じ計算をしますが、当該部門の各従業員の結果を戻します。

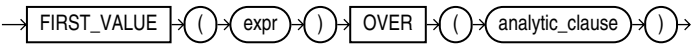
```
SELECT last_name, department_id, salary,
       MIN(salary) KEEP (DENSE_RANK FIRST ORDER BY commission_pct)
         OVER (PARTITION BY department_id) "Worst",
       MAX(salary) KEEP (DENSE_RANK LAST ORDER BY commission_pct)
         OVER (PARTITION BY department_id) "Best"
FROM employees
ORDER BY department_id, salary;
```

LAST_NAME	DEPARTMENT_ID	SALARY	Worst	Best
Whalen	10	4400	4400	4400
Fay	20	6000	6000	13000
Hartstein	20	13000	6000	13000
.				
.				
.				
Gietz	110	8300	8300	12000
Higgins	110	12000	8300	12000
Grant		7000	7000	7000

FIRST_VALUE

構文

first_value::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

FIRST_VALUE は分析ファンクションです。これは、順序付けられた値の集合にある最初の値を戻します。

expr には、FIRST_VALUE または他の分析ファンクションを使用できません。他の組込みファンクション式は expr で使用できますが、分析ファンクションはネストできません。

参照： expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、部門 90 の各従業員について、その部門で給与が一番少ない従業員の名前を選択します。

```
SELECT department_id, last_name, salary, FIRST_VALUE(last_name)
  OVER (ORDER BY salary ASC ROWS UNBOUNDED PRECEDING) AS lowest_sal
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY employee_id);
```

DEPARTMENT_ID	LAST_NAME	SALARY	LOWEST_SAL
90	Kochhar	17000	Kochhar
90	De Haan	17000	Kochhar
90	King	24000	Kochhar

例では、FIRST_VALUE ファンクションの非決定的な性質が示されています。Kochhar と De Haan の給与は同じであるため、Kochhar の次の行に De Haan があります。Kochhar が最初に表示されているのは、副問合せが戻す行が employee_id で順序付けられているためです。ただし、副問合せが戻す行が employee_id で降順に順序付けられている場合は、次の例に示すとおり、ファンクションは異なる値を戻します。

```
SELECT department_id, last_name, salary, FIRST_VALUE(last_name)
  OVER (ORDER BY salary ASC ROWS UNBOUNDED PRECEDING) AS fv
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY employee_id DESC);
```

DEPARTMENT_ID	LAST_NAME	SALARY	FV
90	De Haan	17000	De Haan
90	Kochhar	17000	De Haan
90	King	24000	De Haan

次の例では、一意キーで順序付けることによって、FIRST_VALUE ファンクションを決定的にする方法を示します。

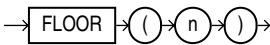
```
SELECT department_id, last_name, salary, hire_date,
  FIRST_VALUE(last_name) OVER
    (ORDER BY salary ASC, hire_date ROWS UNBOUNDED PRECEDING) AS fv
FROM (SELECT * FROM employees
      WHERE department_id = 90 ORDER BY employee_id DESC);
```

DEPARTMENT_ID	LAST_NAME	SALARY	HIRE_DATE	FV
90	Kochhar	17000	21-SEP-89	Kochhar
90	De Haan	17000	13-JAN-93	Kochhar
90	King	24000	17-JUN-87	Kochhar

FLOOR

構文

`floor::=`



用途

FLOOR は *n* 以下の最大整数を返します。

例

次の例では、15.7 以下である最大の整数を返します。

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

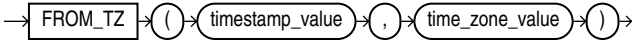
Floor

15

FROM_TZ

構文

`from_tz::=`



用途

FROM_TZ ファンクションは、タイムゾーンのタイムスタンプ値を `TIMESTAMP WITH TIME ZONE` 値に変換します。*time_zone_value* は、`'TZH:TZM'` 書式の文字列、またはオプションの `TZD` 書式を持つ `TZR` 書式で文字列を返す文字式です。

例

次の例では、タイムスタンプ値を `TIMESTAMP WITH TIME ZONE` に返します。

```
SELECT FROM_TZ(TIMESTAMP '2000-03-28 08:00:00', '3:00')
FROM DUAL;
```

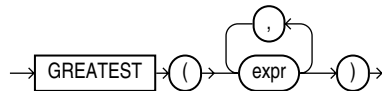
FROM_TZ(TIMESTAMP'2000-03-2808:00:00','3:00')

28-MAR-00 08.00.00 AM +03:00

GREATEST

構文

greatest::=



用途

GREATEST は、*expr* リストの最大値を返します。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、非空白埋め比較セマンティクスで *expr* を比較します。文字の比較は、データベース・キャラクタ・セットの文字値に基づいて行われます。キャラクタ・セット値の大きい文字が、別の文字より大きい文字とみなされます。このファンクションによって戻される値が文字データの場合、そのデータ型は常に VARCHAR2 になります。

参照： 2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の文では、最大値を持つ文字列を検索します。

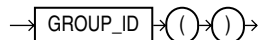
```
SELECT GREATEST ('HARRY', 'HARRIOT', 'HAROLD')
       "Greatest" FROM DUAL;
```

```
Greatest
-----
HARRY
```

GROUP_ID

構文

group_id::=



用途

GROUP_ID は、GROUP BY 指定の結果から、重複するグループを識別します。このファンクションは、問合せ結果から重複グループを除外する場合に有効です。このファンクションは、重複グループを一意に識別するために、Oracle の数値を返します。このファンクションは、1 つの GROUP BY 句を含む SELECT 文のみに適用できます。

特定のグループ化で n 個の重複が存在する場合、GROUP_ID は 0 ～ n-1 の範囲の数値を返します。

例

次の例では、サンプル表 sh.countries および sh.sales の問合せの結果、重複する co.country_region グループ化に値「1」を割り当てます。

```
SELECT co.country_region, co.country_subregion,
       SUM(s.amount_sold) "Revenue",
       GROUP_ID() g
FROM sales s, customers c, countries co
WHERE s.cust_id = c.cust_id AND
      c.country_id = co.country_id AND
      s.time_id = '1-JAN-00' AND
      co.country_region IN ('Americas', 'Europe')
GROUP BY co.country_region,
         ROLLUP (co.country_subregion);
```

COUNTRY_REGION	COUNTRY_SUBREGION	Revenue	G
Americas	Northern America	220844	0
Americas	Southern America	10872	0
Europe	Eastern Europe	12751	0
Europe	Western Europe	558686	0
Americas		231716	0
Europe		571437	0
Americas		231716	1
Europe		571437	1

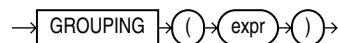
文の最後に次の HAVING 句を追加すると、GROUP_ID が 1 より小さい行のみを戻すことができます。

```
HAVING GROUP_ID() < 1
```


GROUPING

構文

grouping::=



用途

GROUPING は、通常のグループ化された行と超集合行を区別します。ROLLUP や CUBE などの GROUP BY の拡張機能は、すべての値の集合が NULL で表される超集合行を生成します。GROUPING ファンクションを使用すると、通常の行に含まれる NULL と超集合行ですべての値の集合を表す NULL を区別できます。

GROUPING ファンクションの *expr* は、GROUP BY 句の式のいずれかと一致する必要があります。行の *expr* の値がすべての値の集合を表す NULL の場合、このファンクションは 1 の値を返します。それ以外の場合は、0 (ゼロ) を返します。GROUPING ファンクションは、Oracle の 数値を返します。

参照： これらの用語の詳細は、18-21 ページの SELECT 文の「[group_by_clause](#)」を参照してください。

例

次の例では、サンプル表 `hr.departments` および `hr.employees` で、GROUPING ファンクションが 1 (表の通常の行ではなく超集合行) を返す場合、それ以外の場合に表示される NULL のかわりに、文字列「All Jobs」が「JOB」列に表示されます。

```

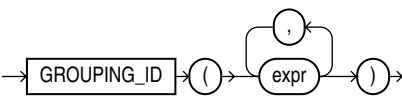
SELECT DECODE(GROUPING(department_name), 1, 'All Departments',
              department_name) AS department,
       DECODE(GROUPING(job_id), 1, 'All Jobs', job_id) AS job,
       COUNT(*) "Total Empl", AVG(salary) * 12 "Average Sal"
FROM   employees e, departments d
WHERE  d.department_id = e.department_id
GROUP BY ROLLUP (department_name, job_id);
  
```

DEPARTMENT	JOB	Total Empl	Average Sal
Accounting	AC_ACCOUNT	1	99600
Accounting	AC_MGR	1	144000
Accounting	All Jobs	2	121800
Administration	AD_ASST	1	52800
Administration	All Jobs	1	52800
Executive	AD_PREP	1	288000
Executive	AD_VP	2	204000
Executive	All Jobs	3	232000

Finance	FI_ACCOUNT	5	95040
Finance	FI_MGR	1	144000
Finance	All Jobs	6	103200
.			
.			
.			

GROUPING_ID

構文
grouping_id::=



用途

GROUPING_ID は、行に関連する GROUPING ビット・ベクトルに対応する数値を戻します。GROUPING_ID は、ROLLUP や CUBE など、GROUP BY の拡張機能および GROUPING ファンクションを含む SELECT 文にのみ適用できます。多くの GROUP BY 式を含む問合せでは、多くの GROUPING ファンクションを必要とする特定の行の GROUP BY レベルを指定するため、非常に複雑な SQL になります。GROUPING_ID は、このような場合に有効です。

GROUPING_ID は、複数の GROUPING ファンクションの結果をビット・ベクトル（1 と 0 を組み合わせた文字列）に連結したものと同じです。GROUPING_ID を使用すると、複数の GROUPING ファンクションを使用する必要がなくなり、行のフィルタ条件の表記が簡単になります。GROUPING_ID を使用すると、要求する行が単一の条件（GROUPING_ID = n）によって識別されるため、行のフィルタ処理が簡単になります。このファンクションは、1 つの表に複数のレベルの集計を格納する場合に、特に有効です。

例

次の例では、サンプル表 sh.sales の問合せからグループ化 ID を抽出する方法を示します。

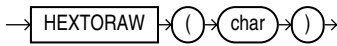
```
SELECT channel_id, promo_id, sum(amount_sold) s_sales,
       GROUPING(channel_id) gc,
       GROUPING(promo_id) gp,
       GROUPING_ID(channel_id, promo_id) gcp,
       GROUPING_ID(promo_id, channel_id) gpc
FROM sales
WHERE promo_id > 496
GROUP BY CUBE(channel_id, promo_id);
```

C	PROMO_ID	S_SALES	GC	GP	GCP	GPC
-	-	-	-	-	-	-
C	497	26094.35	0	0	0	0
C	498	22272.4	0	0	0	0
C	499	19616.8	0	0	0	0
C	9999	87781668	0	0	0	0
C		87849651.6	0	1	1	2
I	497	50325.8	0	0	0	0
I	498	52215.4	0	0	0	0
I	499	58445.85	0	0	0	0
I	9999	169497409	0	0	0	0
I		169658396	0	1	1	2
P	497	31141.75	0	0	0	0
P	498	46942.8	0	0	0	0
P	499	24156	0	0	0	0
P	9999	70890248	0	0	0	0
P		70992488.6	0	1	1	2
S	497	110629.75	0	0	0	0
S	498	82937.25	0	0	0	0
S	499	80999.15	0	0	0	0
S	9999	267205791	0	0	0	0
S		267480357	0	1	1	2
T	497	8319.6	0	0	0	0
T	498	5347.65	0	0	0	0
T	499	19781	0	0	0	0
T	9999	28095689	0	0	0	0
T		28129137.3	0	1	1	2
	497	226511.25	1	0	2	1
	498	209715.5	1	0	2	1
	499	202998.8	1	0	2	1
	9999	623470805	1	0	2	1
		624110031	1	1	3	3

HEXTORAW

構文

hextoraw::=



用途

HEXTORAW は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 キャラクタ・セットで 16 進数を含む *char* を RAW 値に変換します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、RAW 列を含む簡単な表を作成し、RAW に変換された 16 進数値を挿入します。

```
CREATE TABLE test (raw_col RAW(10));

INSERT INTO test VALUES (HEXTORAW('7D'));
```

参照： 2-25 ページの「[RAW データ型と LONG RAW データ型](#)」および 6-119 ページの「[RAWTOHEX](#)」を参照してください。

INITCAP

構文

initcap::=



用途

INITCAP は、各単語の最初の文字を大文字、残りの文字を小文字にして *char* を戻します。単語は空白または英数字以外の文字で区切ります。

char は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型です。戻り値は、*char* と同じデータ型です。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、文字列にある各単語を大文字で始めます。

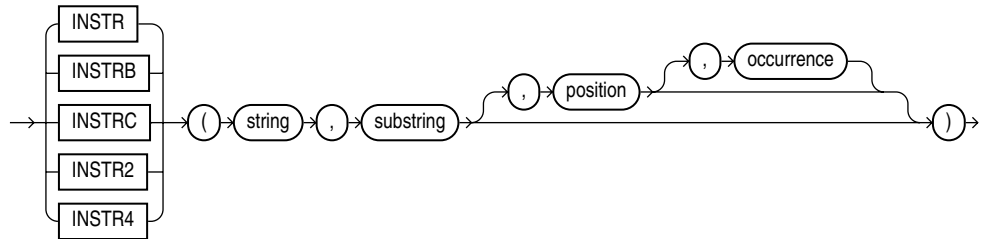
```
SELECT INITCAP('the soap') "Capitals" FROM DUAL;
```

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

INSTR

構文

instr::=



用途

INSTR ファンクションは、*string* の *substring* を検索します。このファンクションは、最初に現れた文字 *string* の位置を示す整数を戻します。INSTR は、入力キャラクタ・セットによって定義された文字を使用して、文字列を算出します。INSTRB は、文字のかわりにバイトを使用します。INSTRC は、完全な Unicode キャラクタを使用します。INSTR2 は、UCS2 コードポイントを使用します。INSTR4 は、UCS4 コードポイントを使用します。

- *position* は、Oracle が検索を開始する文字 *string* の位置を示す 0（ゼロ）以外の整数です。*position* が負の場合、Oracle は *string* の終わりから逆方向にカウントおよび検索します。
- *occurrence* は、検索する *string* が現れたことを示す整数です。*occurrence* の値は正である必要があります。

string および *substring* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻り値のデータ型は NUMBER です。

position および *occurrence* のデフォルト値は 1 です。この場合、Oracle は *string* の最初の文字から検索を開始します。検索対象は *substring* が最初に現れる位置です。戻り値は、*position* の値にかかわらず、*string* の先頭に関係し、文字で表されます。検索が失敗した (*string* の *position* 番目の文字の後に *substring* が *occurrence* 回現れない) 場合、戻り値は 0 となります。

例

次の例では、文字列「CORPORATE FLOOR」で文字列「OR」の検索を 3 番目の文字から開始します。文字列 CORPORATE FLOOR で 2 回目に現れる「OR」の開始位置を戻します。

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

```

Instring
-----
          14
```

次の例では、最後の文字から、最後から 3 番目の文字まで、つまり「FLOOR」の最初の「O」まで逆方向にカウントします。次に、2 回目に現れる「OR」を逆方向に検索し、2 回目に現れるこの文字列が、検索文字列の 2 番目の文字で始まることを認識します。

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

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

この例では、データベース・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。

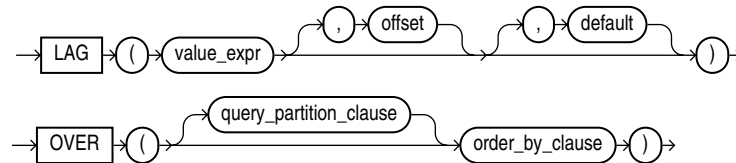
```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2) "Instring in bytes"
       FROM DUAL;
```

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

LAG

構文

lag::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

LAG は分析ファンクションです。これは、自己結合せずに、表の 1 つ以上の行へ同時アクセスを行います。問合せから戻される一連の行およびカーソル位置を指定すると、LAG は、その位置より前にある指定された物理オフセットにある行へアクセスします。

offset を指定しない場合、デフォルト値は 1 です。オフセットがウィンドウの有効範囲を超えた場合、オプションの *default* 値が戻されます。*default* を指定しない場合、デフォルト値は NULL です。

value_expr には、LAG または他の分析ファンクションを使用できません。他の組み込みファンクション式は *value_expr* で使用できますが、分析ファンクションはネストできません。

参照： *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、employees 表の各販売員について、その販売員の直前に雇用された従業員の給与を示します。

```

SELECT last_name, hire_date, salary,
       LAG(salary, 1, 0) OVER (ORDER BY hire_date) AS prev_sal
FROM employees
WHERE job_id = 'PU_CLERK';

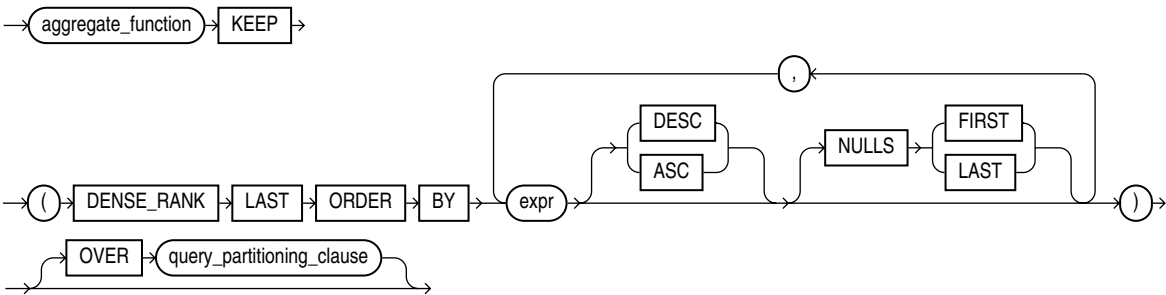
```

LAST_NAME	HIRE_DATE	SALARY	PREV_SAL
Khoo	18-MAY-95	3100	0
Tobias	24-JUL-97	2800	3100
Baida	24-DEC-97	2900	2800
Himuro	15-NOV-98	2600	2900
Colmenares	10-AUG-99	2500	2600

LAST

構文

last::=



参照： `query_partitioning_clause` の構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

FIRST ファンクションと LAST ファンクションは類似しています。両方のファンクションとも、与えられたソート指定に対して、FIRST または LAST としてランク付けされた一連の行の一連の値を操作する集計ファンクションおよび分析ファンクションです。1 つの行のみが FIRST または LAST としてランク付けされている場合、集計は、1 つの要素のみを持つセットを操作します。

ソートされたグループの最初または最後の行の値が必要であり、その値がソート・キーでない場合、FIRST および LAST ファンクションは、自己結合またはビューの必要性を排除し、パフォーマンスを向上させます。

- `aggregate_function` は、MIN、MAX、SUM、AVG、COUNT、VARIANCE または STDDEV ファンクションのいずれかです。FIRST または LAST のいずれかにランク付けされた行の値を操作します。1 つの行のみが FIRST または LAST としてランク付けされている場合、集計は、単一（集計ではない）のセットを操作します。
- `DENSE_RANK FIRST` または `DENSE_RANK LAST` は、最小（FIRST）または最大（LAST）稠密ランク（「オリンピック・ランク」）を持つ行のみを集計することを示します。

OVER 句を指定すると、FIRST および LAST ファンクションを分析ファンクションとして使用できます。OVER 句のうち、`query_partitioning_clause` の部分のみがこれらのファンクションで有効です。

集計の例

次の例では、サンプル表 `hr.employees` の各部門で、歩合が最低である従業員の中での最低給与、および歩合が最高の従業員の中での最高給与を戻します。

```
SELECT department_id,  
MIN(salary) KEEP (DENSE_RANK FIRST ORDER BY commission_pct) "Worst",  
MAX(salary) KEEP (DENSE_RANK LAST ORDER BY commission_pct) "Best"  
  FROM employees  
 GROUP BY department_id;
```

DEPARTMENT_ID	Worst	Best
10	4400	4400
20	6000	13000
30	2500	11000
40	6500	6500
50	2100	8200
60	4200	9000
70	10000	10000
80	6100	14000
90	17000	24000
100	6900	12000
110	8300	12000
	7000	7000

分析の例

次の例では、前述の例と同じ計算をしますが、当該部門の各従業員の結果を戻します。

```
SELECT last_name, department_id, salary,  
MIN(salary) KEEP (DENSE_RANK FIRST ORDER BY commission_pct)  
  OVER (PARTITION BY department_id) "Worst",  
MAX(salary) KEEP (DENSE_RANK LAST ORDER BY commission_pct)  
  OVER (PARTITION BY department_id) "Best"  
  FROM employees  
 ORDER BY department_id, salary;
```

LAST_NAME	DEPARTMENT_ID	SALARY	Worst	Best
Whalen	10	4400	4400	4400
Fay	20	6000	6000	13000
Hartstein	20	13000	6000	13000
.				
.				
.				
Gietz	110	8300	8300	12000
Higgins	110	12000	8300	12000
Grant		7000	7000	7000

LAST_DAY

構文

last_day::=



用途

LAST_DAY は、date を含む月の最終日の日付を戻します。

例

次の文は、現在の月の残りの日数を確認します。

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

SYSDATE	Last	Days Left
-----	-----	-----
30-MAY-01	31-MAY-01	1

次の例では、各従業員の雇用開始日に 5 か月を加えて、評価日付を戻します。

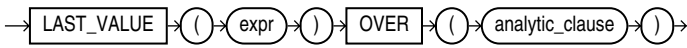
```
SELECT last_name, hire_date, TO_CHAR(
       ADD_MONTHS(LAST_DAY(hire_date), 5)) "Eval Date"
FROM employees;
```

LAST_NAME	HIRE_DATE	Eval Date
-----	-----	-----
King	17-JUN-87	30-NOV-87
Kochhar	21-SEP-89	28-FEB-90
De Haan	13-JAN-93	30-JUN-93
Hunold	03-JAN-90	30-JUN-90
Ernst	21-MAY-91	31-OCT-91
Austin	25-JUN-97	30-NOV-97
Pataballa	05-FEB-98	31-JUL-98
Lorentz	07-FEB-99	31-JUL-99
.		
.		
.		

LAST_VALUE

構文

last_value::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析関クション](#)」を参照してください。

用途

LAST_VALUE は分析関クションです。これは、順序付けられた値の集合にある最後の値を戻します。

expr には、LAST_VALUE または他の分析関クションを使用できません。他の組込み関クション式は expr で使用できますが、分析関クションはネストできません。

参照： expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、給与が一番高い従業員の雇用開始日を各行に戻します。

```

SELECT last_name, salary, hire_date, LAST_VALUE(hire_date) OVER
  (ORDER BY salary
   ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY hire_date);
  
```

LAST_NAME	SALARY	HIRE_DATE	LV
Kochhar	17000	21-SEP-89	17-JUN-87
De Haan	17000	13-JAN-93	17-JUN-87
King	24000	17-JUN-87	17-JUN-87

この例では、LAST_VALUE 関クションの非決定的な性質を示しています。Kochhar と De Haan の給与は同じであるため、Kochhar の次の行に De Haan があります。Kochhar が最初に表示されているのは、副問合せの行が hire_date で順序付けられているためです。ただし、行が hire_date で降順に順序付けられている場合は、次の例に示すとおり、関クションは異なる値を戻します。

```

SELECT last_name, salary, hire_date, LAST_VALUE(hire_date) OVER
  (ORDER BY salary
   ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
  
```

```
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY hire_date DESC);
```

LAST_NAME	SALARY	HIRE_DATE	LV
De Haan	17000	13-JAN-93	17-JUN-87
Kochhar	17000	21-SEP-89	17-JUN-87
King	24000	17-JUN-87	17-JUN-87

次の2つの例では、一意キーで順序付けることによって、LAST_VALUE ファンクションを決定的にする方法を示しています。salary および hire_date でファンクション内を順序付けると、副問合せの順序付けにかかわらず、同じ結果が戻されます。

```
SELECT last_name, salary, hire_date, LAST_VALUE(hire_date) OVER
      (ORDER BY salary, hire_date
       ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY hire_date);
```

LAST_NAME	SALARY	HIRE_DATE	LV
Kochhar	17000	21-SEP-89	17-JUN-87
De Haan	17000	13-JAN-93	17-JUN-87
King	24000	17-JUN-87	17-JUN-87

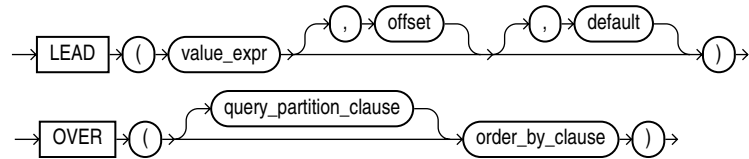
```
SELECT last_name, salary, hire_date, LAST_VALUE(hire_date) OVER
      (ORDER BY salary, hire_date
       ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM (SELECT * FROM employees WHERE department_id = 90
      ORDER BY hire_date DESC);
```

LAST_NAME	SALARY	HIRE_DATE	LV
Kochhar	17000	21-SEP-89	17-JUN-87
De Haan	17000	13-JAN-93	17-JUN-87
King	24000	17-JUN-87	17-JUN-87

LEAD

構文

lead::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

LEAD は分析ファンクションです。これは、自己結合せずに、表の 1 つ以上の行へ同時アクセスを行います。問合せから戻される一連の行およびカーソル位置を指定すると、LEAD は、その位置より後にある指定された物理オフセットの行へアクセスします。

offset を指定しない場合、デフォルト値は 1 です。オフセットが表の有効範囲を超えた場合、オプションの *default* 値が戻されます。*default* を指定しない場合、デフォルト値は NULL です。

value_expr には、LEAD または他の分析ファンクションを使用できません。他の組込みファンクション式は *value_expr* で使用できますが、分析ファンクションはネストできません。

参照： *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、employees 表の各従業員について、その従業員の直後に雇用された従業員の雇用開始日を示します。

```

SELECT last_name, hire_date,
       LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "NextHired"
FROM employees WHERE department_id = 30;
  
```

LAST_NAME	HIRE_DATE	NextHired
Raphaely	07-DEC-94	18-MAY-95
Khoo	18-MAY-95	24-JUL-97
Tobias	24-JUL-97	24-DEC-97
Baida	24-DEC-97	15-NOV-98

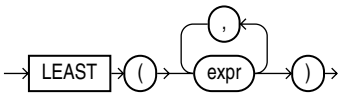
Himuro15-NOV-98 10-AUG-99

Colmenares10-AUG-99

LEAST

構文

least::=



用途

LEAST は、リストされた *expr* 内の最小値を戻します。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、非空白埋め比較セマンティクスで *expr* を比較します。このファンクションによって戻される値が文字データの場合、そのデータ型は常に VARCHAR2 になります。

例

次の文は、LEAST ファンクションの使用例です。

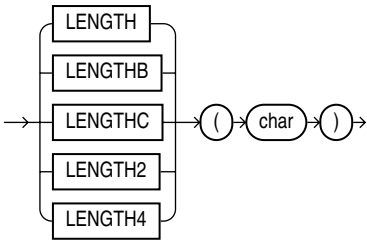
```
SELECT LEAST('HARRY', 'HARRIOT', 'HAROLD') "LEAST"
FROM DUAL;
```

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

LENGTH

構文

length::=



用途

LENGTH の各ファンクションは、*char* の長さを戻します。LENGTH は、入力キャラクタ・セットによって定義された文字を使用して、長さを算出します。LENGTHB は、文字のかわりにバイトを使用します。LENGTHC は、完全な **Unicode** キャラクタを使用します。LENGTH2 は、UCS2 コードポイントを使用します。LENGTH4 は、UCS4 コードポイントを使用します。

char は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻り値のデータ型は NUMBER です。*char* のデータ型が CHAR の場合、その長さにはすべての後続空白が含まれます。*char* が NULL の場合、このファンクションは NULL を戻します。

例

次の例では、シングルバイトおよびマルチバイトのデータベース・キャラクタ・セットを使用する LENGTH ファンクションを使用します。

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

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

この例では、データベース・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。

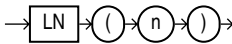
```
SELECT LENGTHB ('CANDIDE') "Length in bytes"
      FROM DUAL;
```

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

LN

構文

ln::=



用途

LN は、*n* の自然対数を戻します (*n* は正の数)。

例

次の例では、95 の自然対数を返します。

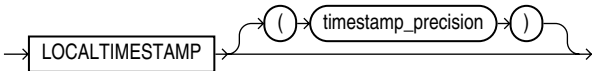
```
SELECT LN(95) "Natural log of 95" FROM DUAL;

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

LOCALTIMESTAMP

構文

localtimestamp::=



用途

LOCALTIMESTAMP は、セッションのタイムゾーンの現在の日付および時刻を `TIMESTAMP` データ型の値で返します。これに対して、`CURRENT_TIMESTAMP` は、`TIMESTAMP WITH TIME ZONE` 値を返します。

参照： 6-46 ページの「[CURRENT_TIMESTAMP](#)」を参照してください。

例

次の例では、`LOCALTIMESTAMP` と `CURRENT_TIMESTAMP` の相違を示します。

```
ALTER SESSION SET TIME_ZONE = '-5:00';
SELECT CURRENT_TIMESTAMP, LOCALTIMESTAMP FROM DUAL;
```

CURRENT_TIMESTAMP	LOCALTIMESTAMP

04-APR-00 01.27.18.999220 PM -05:00	04-APR-00 01.27.19 PM

```
ALTER SESSION SET TIME_ZONE = '-8:00';
SELECT CURRENT_TIMESTAMP, LOCALTIMESTAMP FROM DUAL;
```

CURRENT_TIMESTAMP	LOCALTIMESTAMP

04-APR-00 10.27.45.132474 AM -08:00	04-APR-00 10.27.451 AM

LOCALTIMESTAMP で書式マスクを使用する場合は、ファンクションが戻す値と書式マスクを一致させてください。たとえば、次の表の場合を考えます。

```
CREATE TABLE local_test (col1 TIMESTAMP WITH LOCAL TIME ZONE);
```

ファンクションが戻す型の TIME ZONE の部分がマスクに含まれていないため、次の文は正常に実行されません。

```
INSERT INTO local_test VALUES
  (TO_TIMESTAMP(LOCALTIMESTAMP, 'DD-MON-RR HH.MI.SSXFF'));
```

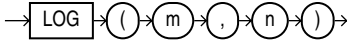
次の文では、LOCALTIMESTAMP の戻り値の型と一致する正しい書式マスクが使用されています。

```
INSERT INTO local_test VALUES
  (TO_TIMESTAMP(LOCALTIMESTAMP, 'DD-MON-RR HH.MI.SSXFF PM'));
```

LOG

構文

log::=



用途

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

例

次の例では、100 の対数を戻します。

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

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

LOWER

構文

lower::=



用途

LOWER は、すべての文字を小文字にして *char* を戻します。*char* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻り値は、*char* と同じデータ型です。

例

次の例では、文字列を小文字にして戻します。

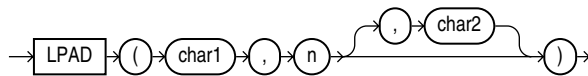
```
SELECT LOWER('MR. SCOTT MCMILLAN') "Lowercase"
   FROM DUAL;
```

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

LPAD

構文

lpad::=



用途

LPAD は、*char1* の左側に *char2* に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* のデフォルト値は空白 1 個です。*char1* が *n* より長い場合、このファンクションは *n* に収まる *char1* の一部を戻します。

char1 および *char2* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char1* と同じキャラクター・セットです。

引数 *n* は、戻り値が画面に表示される場合の全体の長さです。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、表示される文字列の長さが文字列の文字数と異なる場合もあります。

例

次の例では、文字「*」を左側に埋め込みます。

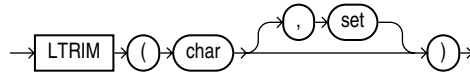
```
SELECT LPAD('Page 1',15,'*.') "LPAD example"
      FROM DUAL;
```

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

LTRIM

構文

ltrim::=



用途

LTRIM は、*char* の左側にあつて *set* に指定された文字を削除します。*set* のデフォルト値は空白 1 個です。*char* が文字リテラルの場合、一重引用符で囲む必要があります。Oracle は *char* の先頭文字からスキャンし始め、*set* に指定された文字をすべて削除します。*set* に指定された文字以外の文字が見つかった時点で結果を戻します。

char および *set* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

例

次の例では、文字列の左端にある *x* および *y* をすべて削除します。

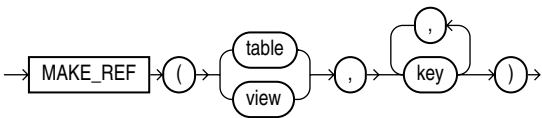
```
SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example"
      FROM DUAL;
```

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

MAKE_REF

構文

make_ref::=



用途

MAKE_REF は、オブジェクト識別子が主キーに基づいている、オブジェクト・ビューの行またはオブジェクト表の行に対する REF を作成します。

参照：

- オブジェクト・ビューの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 6-53 ページの「DEREF」を参照してください。

例

サンプル・スキーマ oe には、inventory_typ に基づくオブジェクト・ビュー oc_inventories が格納されています。オブジェクト識別子は product_id です。次の例では、3003 という product_id を含むオブジェクト・ビュー oc_inventories にある行への REF を作成します。

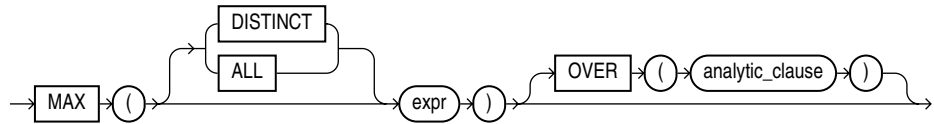
```
SELECT MAKE_REF (oc_inventories, 3003) FROM DUAL;

MAKE_REF(OC_INVENTORIES,3003)
-----
00004A038A0046857C14617141109EE03408002082543600000014260100010001
00290090606002A00078401FE0000000B03C21F040000000000000000000000
0000000000
```

MAX

構文

max::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

MAX は *expr* の最大値を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。*order_by_clause* および *windowing_clause* は指定できません。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、hr.employees 表の最高給与を検索します。

```
SELECT MAX(salary) "Maximum" FROM employees;
```

```

Maximum
-----
      24000
  
```

分析の例

次の例では、各従業員について、その従業員と所属が同じ従業員のうち、一番高い給与を計算します。

```
SELECT manager_id, last_name, salary,
       MAX(salary) OVER (PARTITION BY manager_id) AS mgr_max
FROM employees;
```

```
MANAGER_ID LAST_NAME          SALARY    MGR_MAX
```

100	Kochhar	17000	17000
100	De Haan	17000	17000
100	Raphaely	11000	17000
100	Kaufling	7900	17000
100	Fripp	8200	17000
100	Weiss	8000	17000
...			

この問合せを述語のある親問合せで囲むと、各部門で給与の一番高い従業員がわかります。

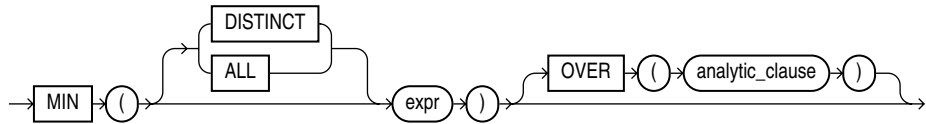
```
SELECT manager_id, last_name, salary
FROM (SELECT manager_id, last_name, salary,
      MAX(salary) OVER (PARTITION BY manager_id) AS rmax_sal
      FROM employees) WHERE salary = rmax_sal;
```

MANAGER_ID	LAST_NAME	SALARY
100	Kochhar	17000
100	De Haan	17000
101	Greenberg	12000
101	Higgins	12000
102	Hunold	9000
103	Ernst	6000
108	Faviet	9000
114	Khoo	3100
120	Nayer	3200
120	Taylor	3200
121	Sarchand	4200
122	Chung	3800
123	Bell	4000
124	Rajs	3500
145	Tucker	10000
146	King	10000
147	Vishney	10500
148	Ozer	11500
149	Abel	11000
201	Goyal	6000
205	Gietz	8300
	King	24000

MIN

構文

min::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

MIN は、*expr* の最小値を返します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。*order_by_clause* および *windowing_clause* は指定できません。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、hr.employees 表の最初の雇用開始日を返します。

```
SELECT MIN(hire_date) "Earliest" FROM employees;
```

```
Earliest
```

```
-----
```

```
17-JUN-87
```

分析の例

次の例では、各従業員について、その従業員が雇用された日以前に雇用された従業員を検索します。その従業員と所属が同じ従業員のサブセットを決定し、そのサブセット内で一番低い給与を返します。

```
SELECT manager_id, last_name, hire_date, salary,
       MIN(salary) OVER(PARTITION BY manager_id ORDER BY hire_date
                        RANGE UNBOUNDED PRECEDING) as p_cmin
FROM employees;
```

MANAGER_ID	LAST_NAME	HIRE_DATE	SALARY	P_CMIN
100	Kochhar	21-SEP-89	17000	17000
100	De Haan	13-JAN-93	17000	17000
100	Raphaely	07-DEC-94	11000	11000
100	Kaufling	01-MAY-95	7900	7900
100	Hartstein	17-FEB-96	13000	7900
100	Weiss	18-JUL-96	8000	7900
100	Russell	01-OCT-96	14000	7900
100	Partners	05-JAN-97	13500	7900
100	Errazuriz	10-MAR-97	12000	7900
.				
.				
.				

MOD

構文

mod::=



用途

MOD は m を n で割った余りを戻します。 n が 0 の場合は、 m を戻します。

例

次の例では、 11 を 4 で割ったときの余りを戻します。

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

Modulus

3

このファンクションは、 m が負の場合には古典数学のモジュール・ファンクションとは異なる動作をします。 古典数学のモジュール・ファンクションは、 次の公式を用いた MOD ファンクションで表すことができます。

$$m - n * \text{FLOOR}(m/n)$$

次の表に、MOD ファンクションと古典数学のモジュール・ファンクションの相違を示します。

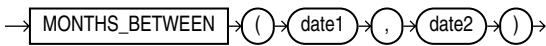
m	n	MOD(m,n)	古典数学のモジュール・ファンクション
11	4	3	3
11	-4	3	-1
-11	4	-3	1
-11	-4	-3	-3

参照： 6-66 ページの「FLOOR」を参照してください。

MONTHS_BETWEEN

構文

months_between::=



用途

MONTHS_BETWEEN は、日付 *date1* と *date2* の間の月数を戻します。*date1* が *date2* より後の日付の場合、結果は正の値になります。*date1* が *date2* より前の日付の場合、結果は負の値になります。*date1* および *date2* が、月の同じ日または月の最終日の場合、結果は常に整数になります。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、*date1* および *date2* の差を割り出します。

例

次の例では、2 つの日付間の月数を計算します。

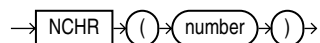
```
SELECT MONTHS_BETWEEN
       (TO_DATE('02-02-1995', 'MM-DD-YYYY'),
        TO_DATE('01-01-1995', 'MM-DD-YYYY')) "Months"
FROM   DUAL;

Months
-----
1.03225806
```

NCHR

構文

nchr::=



用途

NCHR は、各国語キャラクタ・セットの *number* と同等のバイナリを持つ文字を戻します。このファンクションは、CHR ファンクションに USING NCHAR_CS 句を指定して使用した場合と同じ結果を戻します。

参照： 6-29 ページの「[CHR](#)」を参照してください。

例

次の例では、NCHAR 文字 187 を戻します。

```
SELECT NCHR(187) FROM DUAL;
```

```
NC
--
>
```

```
SELECT CHR(187 USING NCHAR_CS) FROM DUAL;
```

```
CH
--
>
```

NEW_TIME

構文

new_time::=



用途

NEW_TIME は、タイムゾーン *zone1* の日時が *date* の時点のタイムゾーン *zone2* の日時を戻します。このファンクションを使用する前に、24 時間で表示されるように、NLS_DATE_FORMAT パラメータを設定する必要があります。

注意： このファンクションが入力として取ることのできるタイムゾーンの数には制限があります。FROM_TZ ファンクションと日時式を組み合わせることによって、より多くのタイムゾーンにアクセスできます。6-66 ページの「[FROM_TZ](#)」および 4-10 ページの「[日時式](#)」の例を参照してください。

引数 `zone1` および `zone2` には、次のテキスト列のいずれかを指定できます。

- AST、ADT: 大西洋標準時および大西洋夏時間
- BST、BDT: ベーリング標準時およびベーリング夏時間
- CST、CDT: 中央標準時および中央夏時間
- EST、EDT: 東部標準時および東部夏時間
- GMT: グリニッジ標準時
- HST、HDT: アラスカ - ハワイ標準時およびアラスカ - ハワイ夏時間
- MST、MDT: 山岳部標準時および東部夏時間
- NST: ニューファンドランド標準時
- PST、PDT: 太平洋標準時および太平洋夏時間
- YST、YDT: ユーコン標準時およびユーコン夏時間

例

次の例では、指定された太平洋標準時と同等の大西洋標準時を戻します。

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

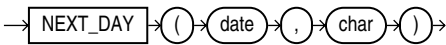
SELECT NEW_TIME(TO_DATE(
  '11-10-99 01:23:45', 'MM-DD-YY HH24:MI:SS'),
  'AST', 'PST') "New Date and Time" FROM DUAL;
```

```
New Date and Time
-----
09-NOV-1999 21:23:45
```

NEXT_DAY

構文

next_day::=



用途

NEXT_DAY は、*char* で指定した曜日で、日付 *date* より後の最初の日付を戻します。引数 *char* は、セッションの日付言語での曜日である必要があります（フルネームでも省略形で可）。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 *date* と同じ時、分および秒のコンポーネントを持っています。

例

次の例では、2001 年 2 月 2 日以降の最初の火曜日の日付を戻します。

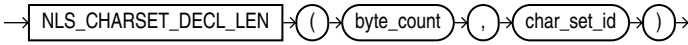
```
SELECT NEXT_DAY('02-FEB-2001','TUESDAY') "NEXT DAY"
      FROM DUAL;

NEXT DAY
-----
06-FEB-2001
```

NLS_CHARSET_DECL_LEN

構文

nls_charset_decl_len::=



用途

NLS_CHARSET_DECL_LEN は、NCHAR 列の宣言の幅を（文字数で）戻します。*byte_count* 引数は、列の幅です。*char_set_id* 引数は、列のキャラクタ・セット ID です。

例

次の例では、マルチバイト・キャラクタ・セットを使用している場合に、列の幅が 200 バイトである文字の数を戻します。

```
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::=



用途

NLS_CHARSET_ID は、キャラクタ・セット名 *text* に対応するキャラクタ・セットの ID 番号を戻します。引数 *text* は、実行時の VARCHAR2 値です。*text* 値 'CHAR_CS' は、サーバーのデータベース・キャラクタ・セットの ID 番号を戻します。*text* 値 'NCHAR_CS' は、サーバーの各国語キャラクタ・セットの ID 番号を戻します。

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。

例

次の例では、キャラクタ・セットの ID を戻します。

```
SELECT NLS_CHARSET_ID('ja16euc')
FROM DUAL;
```

```
NLS_CHARSET_ID('JA16EUC')
-----
830
```

参照： キャラクタ・セット名のリストについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NLS_CHARSET_NAME

構文

nls_charset_name::=



用途

NLS_CHARSET_NAME は、ID 番号 *number* に対応するキャラクタ・セット名を返します。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。

number 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、このファンクションは NULL を返します。

例

次の例では、キャラクタ・セットの ID 番号 2 に対応するキャラクタ・セットを返します。

```
SELECT NLS_CHARSET_NAME(2)
FROM DUAL;
```

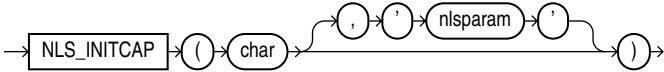
```
NLS_CH
-----
WE8DEC
```

参照： キャラクタ・セット ID のリストについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NLS_INITCAP

構文

nls_initcap::=



用途

NLS_INITCAP は、各単語の最初の文字を大文字、残りの文字を小文字にして *char* を返します。単語は空白または英数字以外の文字で区切ります。

char および '*nlsparam*' は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

'*nlsparam*' の値は次の書式で指定します。

```
'NLS_SORT = sort'
```

sort は、言語ソート基準または BINARY のいずれかです。言語ソート基準は、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。これらの要件によって、*char* と異なる長さの値が戻される場合があります。'*nlsparam*' を省略すると、このファンクションはセッションに対してデフォルトのソート基準を使用します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次に、ファンクションによって言語ソート基準がどのように異なる値を戻すかを示します。

```
SELECT NLS_INITCAP
       ('ijsland') "InitCap" FROM DUAL;
```

```
InitCap
-----
Ijsland
```

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

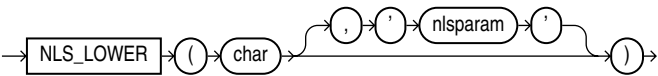
```
InitCap
-----
IJIsland
```

参照： ソート基準の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NLS_LOWER

構文

nls_lower::=



用途

NLS_LOWER は、すべての文字を小文字にして *char* を戻します。

char および '*nlsparam*' は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

'*nlsparam*' の書式および用途は、NLS_INITCAP ファンクションと同じです。

例

次の例では、XGerman 言語ソート順序を使用する文字列 'citta' を戻します。

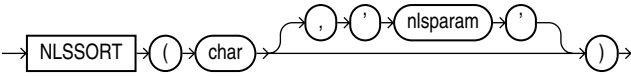
```
SELECT NLS_LOWER
      ('CITTA', 'NLS_SORT = XGerman') "Lowercase"
FROM DUAL;

Lowerc
-----
citta'
```

NLSSORT

構文

nlsort::=



用途

NLSSORT は、*char* のソートに使用される文字列のバイトを戻します。

char および '*nlsparam*' は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型です。戻される文字列は RAW データ型です。

'*nlsparam*' の値は次の書式で指定します。

```
'NLS_SORT = sort'
```

sort は、言語ソート基準または BINARY のいずれかです。'*nlsparam*' を省略すると、このファンクションはセッションに対してデフォルトのソート基準を使用します。BINARY を指定すると、このファンクションは *char* を戻します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

このファンクションを使用すると、文字列の 2 進値を基にした比較ではなく、言語ソート基準を基にした比較を指定できます。次の例では、2 つの値を含むテスト表を作成し、戻される値が NLSSORT ファンクションによってどのように順序付けられるかを示します。

```
CREATE TABLE test (name VARCHAR2(15));
INSERT INTO test VALUES ('Gaardiner');
INSERT INTO test VALUES ('Gaberd');

SELECT * FROM test ORDER BY name;

NAME
-----
Gaardiner
Gaberd

SELECT * FROM test
      ORDER BY NLSSORT(name, 'NLS_SORT = XDanish');

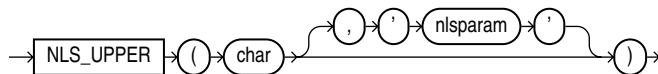
NAME
-----
Gaberd
Gaardiner
```

参照： ソート基準の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NLS_UPPER

構文

nls_upper::=



用途

NLS_UPPER は、すべての文字を大文字にして *char* を戻します。

char および '*nlsparam*' は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

'*nlsparam*' の書式および用途は、NLS_INITCAP ファンクションと同じです。

例

次の例では、すべての文字を大文字に変換して文字列を戻します。

```
SELECT NLS_UPPER ('große') "Uppercase"
      FROM DUAL;
```

Upper

GROßE

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

Upperc

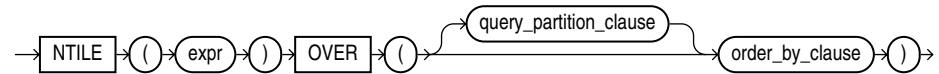
GROSSE

参照： 6-98 ページの「[NLS_INITCAP](#)」を参照してください。

NTILE

構文

ntile::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析関クション](#)」を参照してください。

用途

NTILE は分析関クションです。これは、順序付けられたデータセットを *expr* に指定した数のバケットに分割し、適切なバケット番号を各行に割り当てます。バケットには 1 ～ *expr* の番号が付けられます。ここで、*expr* には、パーティションごとに、正の定数を指定する必要があります。

バケット内の行数は、最大で 1 異なります。残りの値（バケットで割った行数の余り）は、バケット 1 から順に、1 行ずつ分割されます。

expr が行数より大きい場合、行数と等しい数のバケットに行が入れられ、余りのバケットは空になります。

expr には、NTILE または他の分析関クションを使用できません。他の組込み関クション式は *expr* で使用できますが、分析関クションはネストできません。

参照： *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、部門 100 の `oe.employees` 表の `salary` 列の値を 4 つのバケットに分割します。`salary` 列は、この部門に 6 つの値を持つため、2 つの余分な値（6 を 4 で割った余り）は、バケット 1 および 2 に割り当てられます。そのため、これらのバケットは、バケット 3 または 4 より値が 1 つ多くなります。

```
SELECT last_name, salary, NTILE(4) OVER (ORDER BY salary DESC)
       AS quartile FROM employees
WHERE department_id = 100;
```

LAST_NAME	SALARY	QUARTILE
Greenberg	12000	1
Faviet	9000	1
Chen	8200	2

Urman	7800	2
Sciarra	7700	3
Popp	6900	4

NULLIF

構文

nullif::=



用途

NULLIF は、*expr1* と *expr2* を比較します。同じである場合は、NULL を戻します。異なる場合は、*expr1* を戻します。*expr1* には、リテラル NULL を指定できません。

NULLIF ファンクションは、次の CASE 式と論理的に同じです。

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

参照： 4-6 ページの「CASE 式」を参照してください。

例

次の例では、サンプル・スキーマ hr から、雇用後に職種が変更した従業員を検索します。これは、employees 表の現行の job_id が job_history 表の job_id と異なるかどうかによって識別されます。

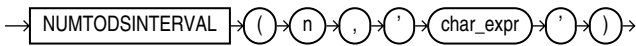
```
SELECT e.last_name, NULLIF(e.job_id, j.job_id) "Old Job ID"
FROM employees e, job_history j
WHERE e.employee_id = j.employee_id
ORDER BY last_name;
```

LAST_NAME	Old Job ID
De Haan	AD_VP
Hartstein	MK_MAN
Kaufling	ST_MAN
Kochhar	AD_VP
Kochhar	AD_VP
Raphaely	PU_MAN
Taylor	SA_REP
Taylor	
Whalen	AD_ASST
Whalen	

NUMTODSINTERVAL

構文

numtodsinterval::=



用途

NUMTODSINTERVAL は、*n* を INTERVAL DAY TO SECOND リテラルに変換します。*n* は、数値、または結果が数値になる式になります。*char_expr* のデータ型は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 です。*char_expr* の値は *n* の単位を指定し、結果が次の文字列値のいずれかになる必要があります。

- 'DAY'
- 'HOUR'
- 'MINUTE'
- 'SECOND'

char_expr は、大文字と小文字を区別しません。カッコ内の先行値および後続値は無視されます。デフォルトでは、戻り値の精度は 9 です。

例

次の例では、各従業員について、雇用開始日から 100 日以内に同じマネージャの部下として配属された従業員の人数を計算します。

```
SELECT manager_id, last_name, hire_date,
       COUNT(*) OVER (PARTITION BY manager_id ORDER BY hire_date
                      RANGE NUMTODSINTERVAL(100, 'day') PRECEDING) AS t_count
FROM employees;
```

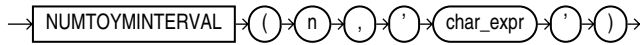
MANAGER_ID	LAST_NAME	HIRE_DATE	T_COUNT
100	Kochhar	21-SEP-89	1
100	De Haan	13-JAN-93	1
100	Raphaely	07-DEC-94	1
100	Kaufling	01-MAY-95	1
100	Hartstein	17-FEB-96	1
.			
.			
.			
149	Grant	24-MAY-99	1
149	Johnson	04-JAN-00	1
201	Goyal	17-AUG-97	1

205 Gietz	07-JUN-94	1
King	17-JUN-87	1

NUMTOYMINTERVAL

構文

numtoyminterval::=



用途

NUMTOYMINTERVAL は、数値 *n* を INTERVAL YEAR TO MONTH リテラルに変換します。*n* は、数値、または結果が数値になる式になります。*char_expr* のデータ型は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 です。*char_expr* の値は *n* の単位を指定し、結果が次の文字列値のいずれかになる必要があります。

- 'YEAR'
- 'MONTH'

char_expr は、大文字と小文字を区別しません。カッコ内の先行値および後続値は無視されます。デフォルトでは、戻り値の精度は 9 です。

例

次の例では、各従業員について、その従業員の雇用日から 1 年の間に雇用された従業員の給与の合計を計算します。

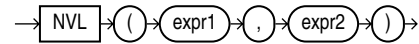
```
SELECT last_name, hire_date, salary, SUM(salary)
      OVER (ORDER BY hire_date
            RANGE NUMTOYMINTERVAL(1,'year') PRECEDING) AS t_sal
FROM employees;
```

LAST_NAME	HIRE_DATE	SALARY	T_SAL
-----	-----	-----	-----
King	17-JUN-87	24000	24000
Whalen	17-SEP-87	4400	28400
Kochhar	21-SEP-89	17000	17000
.			
.			
.			
Markle	08-MAR-00	2200	112400
Ande	24-MAR-00	6400	106500
Banda	21-APR-00	6200	109400
Kumar	21-APR-00	6100	109400

NVL

構文

nvl::=



用途

NVL を使用すると、NULL（空白）を問合せの結果に含まれる文字列で置換できます。
`expr1` が NULL の場合、NVL は `expr2` を返します。`expr1` が NULL でない場合、NVL は `expr1` を返します。引数 `expr1` および `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、戻り値のデータ型は VARCHAR2 になり、`expr1` のキャラクタ・セットになります。

例

次の例では、従業員が歩合を受け取らない場合、従業員名および従業員の歩合「Not Applicable」を表示します。

```

SELECT last_name, NVL(TO_CHAR(commission_pct), 'Not Applicable')
       "COMMISSION" FROM employees
WHERE last_name LIKE 'B%'
ORDER BY last_name;

```

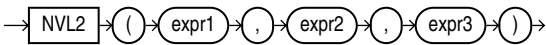
LAST_NAME	COMMISSION

Baer	Not Applicable
Baida	Not Applicable
Banda	.1
Bates	.15
Bell	Not Applicable
Bernstein	.25
Bissot	Not Applicable
Bloom	.2
Bull	Not Applicable

NVL2

構文

nvl2::=



用途

NVL2を使用すると、指定された式が NULL かどうかに基づく問合せによって戻される値を判断できます。 *expr1* が NULL でない場合、NVL2 は *expr2* を戻します。 *expr1* が NULL の場合、NVL2 は *expr3* を戻します。 引数 *expr1* は、任意のデータ型を持つことができます。 引数 *expr2* および *expr3* は、LONG 以外の任意のデータ型を持つことができます。

expr2 と *expr3* のデータ型が異なり、*expr3* が NULL 定数でない場合、Oracle は比較前に *expr3* を *expr2* のデータ型に変換します。 この場合、データ型の変換は必要ありません。

戻り値のデータ型は、常に *expr2* のデータ型と同じになります。 ただし、*expr2* が文字データの場合、戻り値のデータ型は VARCHAR2 になります。

例

次の例では、employees の commission_pct 列が NULL かどうかによって、従業員の収入が給与と歩合か、または給与のみかを示します。

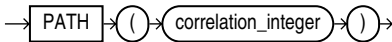
```
SELECT last_name, salary, NVL2(commission_pct,
    salary + (salary * commission_pct), salary) income
FROM employees WHERE last_name like 'B%'
ORDER BY last_name;
```

LAST_NAME	SALARY	INCOME
-----	-----	-----
Baer	10000	10000
Baida	2900	2900
Banda	6200	6882
Bates	7300	8468
Bell	4000	4000
Bernstein	9500	11970
Bissot	3300	3300
Bloom	10000	12100
Bull	4100	4100

PATH

構文

path::=



用途

PATH は、UNDER_PATH および EQUALS_PATH 条件でのみ使用される補助ファンクションです。一次条件で指定されたリソースへの相対パスを戻します。

correlation_integer は任意の数値で、この補助ファンクションをその一次条件と関連付けるために使用します。1 未満の値は 1 として扱われます。

参照：

- 5-13 ページの「EQUALS_PATH」および 5-20 ページの「UNDER_PATH」を参照してください。
- 6-52 ページの「DEPTH」を参照してください。

例

EQUALS_PATH および UNDER_PATH 条件は、2 つの補助ファンクションを取ることができ、その 1 つが PATH です。次に、その 2 つの補助ファンクションの使用法を示します。この例では、XML Schema である warehouses.xsd (D-11 ページの「SQL 文での XML の使用方法」で作成) が存在することを前提としています。

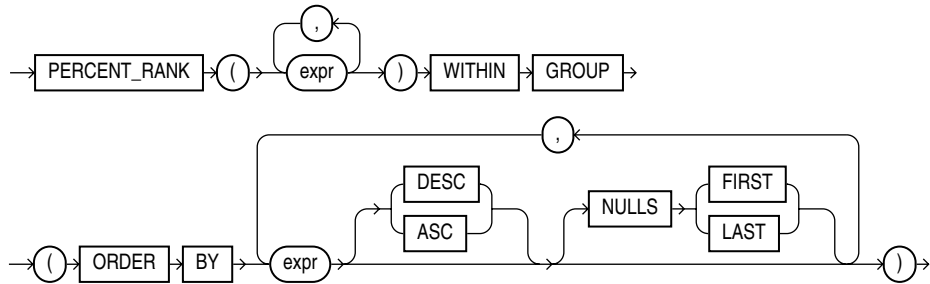
```
SELECT PATH(1), DEPTH(2)
FROM RESOURCE_VIEW
WHERE UNDER_PATH(res, '/sys/schemas/OE', 1)=1
      AND UNDER_PATH(res, '/sys/schemas/OE', 2)=1;
```

PATH(1)	DEPTH(2)
/www.oracle.com	1
/www.oracle.com/xwarehouses.xsd	2

PERCENT_RANK

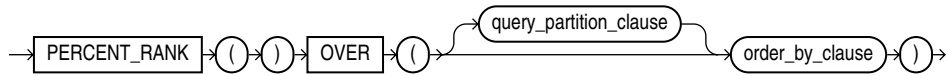
集計の構文

percent_rank_aggregate::=



分析の構文

percent_rank_analytic::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

PERCENT_RANK は、CUME_DIST (累積分布) ファンクションと似ています。
PERCENT_RANK が戻す値の範囲は、0 ～ 1 (0 および 1 を含む) です。すべての集合の最初の行の PERCENT_RANK は 0 (ゼロ) になります。

- 集計ファンクションとしての PERCENT_RANK は、ファンクションの引数および対応するソート指定によって識別される不確定な行 R を計算し、行 R のランクから 1 を引いて、集計グループ内の行の数で割ります。不確定な行 R を Oracle が集計する行のグループに挿入するように計算します。このファンクションの引数は、各集計グループ内の 1 つの不確定行を識別します。このため、すべての引数は、集計グループ内で定数式と評価される必要があります。定数引数式および集計の ORDER BY 句の式の位置は、一致します。このため、引数の数は同じであり、その型は互換性がある必要があります。
- 分析ファンクションとしての PERCENT_RANK は、行 R に対して、R のランクから 1 引いた数を評価される行数（問合せ結果セット全体またはパーティション）より 1 少ない数で割ります。

- 分析関数としての PERCENT_RANK は、行 R に対して、R のランクから 1 引いた数を評価される行数（問合せ結果セット全体またはパーティション）より 1 少ない数で割ります。

集計の例

次の例では、サンプル表 hr.employees から、給与が 15,500 ドルであり、歩合が 5% である不確定な従業員のパーセント・ランクを計算します。

```
SELECT PERCENT_RANK(15000, .05) WITHIN GROUP
  (ORDER BY salary, commission_pct) "Percent-Rank"
FROM employees;
```

Percent-Rank

.971962617

分析の例

次の例では、従業員ごとに、部門内での給与のパーセント・ランクを計算します。

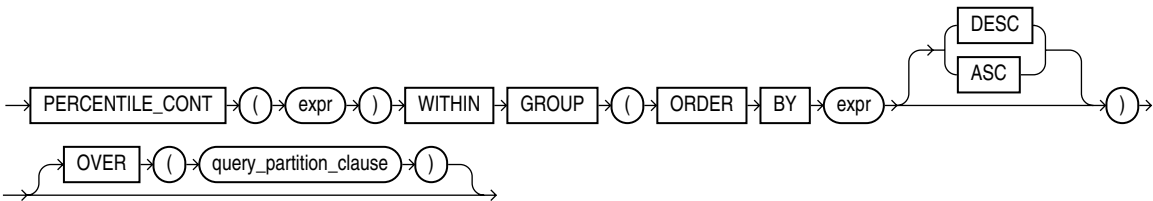
```
SELECT department_id, last_name, salary,
  PERCENT_RANK()
    OVER (PARTITION BY department_id ORDER BY salary DESC) AS pr
FROM employees
ORDER BY pr, salary;
```

DEPARTMENT_ID	LAST_NAME	SALARY	PR
10	Whalen	4400	0
40	Marvis	6500	0
.			
.			
.			
80	Vishney	10500	.176470588
50	Everett	3900	.181818182
30	Khoo	3100	.2
.			
.			
.			
80	Johnson	6200	.941176471
50	Markle	2200	.954545455
50	Philtanker	2200	.954545455
50	Olson	2100	1
.			
.			
.			

PERCENTILE_CONT

構文

percentile_cont::=



参照： OVER 句の構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

PERCENTILE_CONT は、連続分散モデルを想定する逆分散関数です。このファンクションは、パーセンタイル値およびソート指定を用い、そのソート指定に従ってそのパーセンタイル値に該当する補間された値を戻します。計算では、NULL は無視されます。

最初の `expr` は、パーセンタイル値であるため、0 ～ 1 の数値で評価します。この `expr` は、各集計グループ内の定数である必要があります。ORDER BY 句には、Oracle が補間を実行できる型である数値または日時値の単一式を指定します。

PERCENTILE_CONT の結果は、順序付けされた後の値間の直線補間によって計算されます。集計グループで、パーセンタイル値 (P) および行数 (N) を使用すると、ソート指定に従って行を順序付けた後の行数を計算します。この行数 (RN) は、計算式 $RN = (1 + (P * (N - 1)))$ に従って計算されます。集計ファンクションの最終結果は、行番号が $CRN = CEILING(RN)$ および $FRN = FLOOR(RN)$ の行の値間の直線補間によって計算されます。

最終結果は次のとおりです。

```
if (CRN = FRN = RN) then
  (value of expression from row at RN)
else
  (CRN - RN) * (value of expression for row at FRN) +
  (RN - FRN) * (value of expression for row at CRN)
end if
```

PERCENTILE_CONT ファンクションは、分析ファンクションとしても使用できます。その場合、OVER 句には、`query_partitioning_clause` のみを指定できます。各行に対して、各パーティション内の一連の値から、指定されたパーセンタイルに該当する値を戻します。

集計の例

次の例では、各部門の給与の中央値を計算します。

```
SELECT department_id,  
       PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary DESC)  
       "Median cont",  
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY salary DESC)  
       "Median disc"  
FROM employees GROUP BY department_id;
```

DEPARTMENT_ID	Median-cont	Median-disc
10	4400	4400
20	9500	13000
30	2850	2900
40	6500	6500
50	3100	3100
60	4800	4800
70	10000	10000
80	8800	8800
90	17000	17000
100	8000	8200
110	10150	12000

PERCENTILE_CONT および PERCENTILE_DISC は、異なる結果を返す場合があります。
PERCENTILE_CONT は、直線補間後の結果を計算します。PERCENTILE_DISC は、集計された一連の値から値のみを返します。この例に示すように、パーセンタイル値が 0.5 の場合、PERCENTILE_CONT は、偶数の要素を持つグループの中間の 2 つの値の平均を返します。それに対して、PERCENTILE_DISC は、中間の 2 つの値の最初の値を返します。奇数の要素を持つ集計グループでは、どちらのファンクションも中間の要素の値を返します。

分析の例

次の例では、0.5 のパーセンタイル（Percent_Rank）に対応する部門 60 の中央値は 4800 です。部門 30 の給与にパーセンタイル 0.5 がないため、2900（パーセンタイル 0.4）～2800（パーセンタイル 0.6）の範囲で 2850 に評価される中央値が補間される必要があります。

```
SELECT last_name, salary, department_id,  
       PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary DESC)  
       OVER (PARTITION BY department_id) "Percentile_Cont",  
       PERCENT_RANK()  
       OVER (PARTITION BY department_id ORDER BY salary DESC) "Percent_Rank"  
FROM employees WHERE department_id IN (30, 60);
```

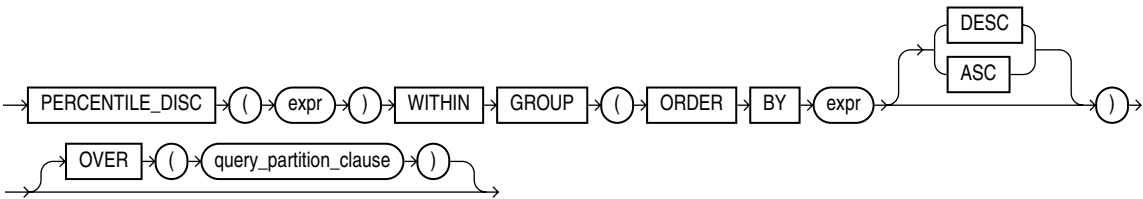
LAST_NAME	SALARY	DEPARTMENT_ID	Percentile_Cont	Percent_Rank
Raphaely	11000	30	2850	0

Khoo	3100	30	2850	.2
Baida	2900	30	2850	.4
Tobias	2800	30	2850	.6
Himuro	2600	30	2850	.8
Colmenares	2500	30	2850	1
Hunold	9000	60	4800	0
Ernst	6000	60	4800	.25
Austin	4800	60	4800	.5
Pataballa	4800	60	4800	.5
Lorentz	4200	60	4800	1

PERCENTILE_DISC

構文

percentile_disc::=



参照： OVER 句の構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

PERCENTILE_DISC は、不連続分散モデルを想定する逆分散関数です。このファンクションでは、パーセンタイル値およびソート指定を指定し、そのセットから要素を戻します。計算では、NULL は無視されます。

最初の *expr* は、パーセンタイル値であるため、0 ～ 1 の数値で評価します。この *expr* は、各集計グループ内の定数である必要があります。ORDER BY 句には、ソート可能な型の単一式を指定します。

指定されたパーセンタイル値 P に対して、PERCENTILE_DISC ファンクションは、ORDER BY 句の式の値をソートし、P 以上である（同じソート指定に従う）最小 CUME_DIST 値を持つ値を戻します。

集計の例

6-112 ページの「[PERCENTILE_CONT](#)」にある例を参照してください。

分析の例

次の例では、サンプル表 `hr.employees` の各従業員の給与の中央値不連続パーセンタイルを計算します。

```
SELECT last_name, salary, department_id,
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY salary DESC)
       OVER (PARTITION BY department_id) "Percentile_Disc",
       CUME_DIST() OVER (PARTITION BY department_id
       ORDER BY salary DESC) "Cume_Dist"
FROM employees where department_id in (30, 60);
```

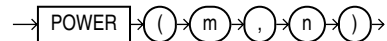
LAST_NAME	SALARY	DEPARTMENT_ID	Percentile_Disc	Cume_Dist
Raphaely	11000	30	2900	.166666667
Khoo	3100	30	2900	.333333333
Baida	2900	30	2900	.5
Tobias	2800	30	2900	.666666667
Himuro	2600	30	2900	.833333333
Colmenares	2500	30	2900	1
Hunold	9000	60	4800	.2
Ernst	6000	60	4800	.4
Austin	4800	60	4800	.8
Pataballa	4800	60	4800	.8
Lorentz	4200	60	4800	1

部門 30 の中央値の値は 2900 です。この値の対応するパーセンタイル (Cume_Dist) は、0.5 以上の最小値です。部門 60 の中央値の値は 4800 です。この値の対応するパーセンタイルは、0.5 以上の最小値です。

POWER

構文

power::=



用途

POWER は、 m を n 乗した値を返します。底 m および指数 n は任意の数です。ただし、 m が負の場合、 n は整数である必要があります。

例

次の例では、3 の 2 乗を戻します。

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

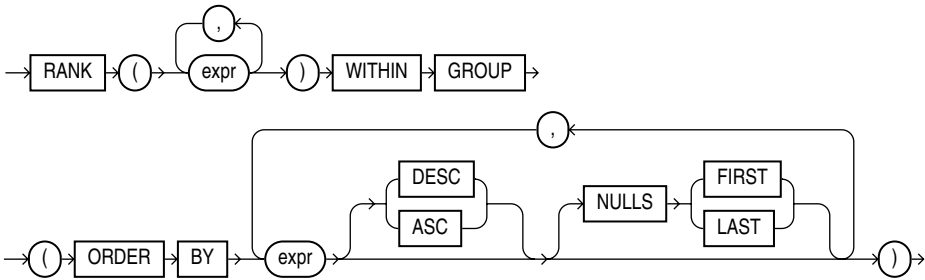
Raised

9

RANK

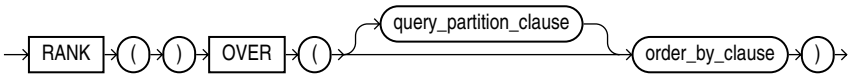
集計の構文

rank_aggregate::=



分析の構文

rank_analytic::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析関数](#)」を参照してください。

用途

RANK は、一連の値における値のランクを計算します。ランク付け基準と同じ値を持つ行は、同じランクになります。Oracle は連結行の数を連結ランクに追加して、次のランクを計算します。そのため、ランクは連続した数値でない場合があります。

- 集計ファンクションとして使用する RANK は、ファンクションの引数として識別される不確定な行のランクを、指定されたソート指定に従って計算します。ファンクションの引数は、各集計グループの単一行を識別するため、すべての引数は各集計グループ内で定数式に評価される必要があります。定数引数式および集計の ORDER BY 句の式の位置は、一致します。このため、引数の数は同じであり、その型は互換性がある必要があります。
- 分析ファンクションとして使用する RANK は、ある問合せが戻す他の行について、その問合せが戻す各行のランクを計算します。この計算は、*order_by_clause* にある *value_exprs* の値に基づいて行われます。

集計の例

次の例では、サンプル表 `hr.employees` から、給与が 15,500 ドルであり、歩合が 5% である不確定な従業員のランクを計算します。

```
SELECT RANK(15500, .05) WITHIN GROUP
       (ORDER BY salary, commission_pct) "Rank"
FROM employees;
```

```
Rank
-----
105
```

同様に、次の問合せは、従業員の給与から給与が 15,500 ドルのランクを戻します。

```
SELECT RANK(15500) WITHIN GROUP
       (ORDER BY salary DESC) "Rank of 15500"
FROM employees;
```

```
Rank of 15500
-----
4
```

分析の例

次の例では、サンプル・スキーマ `hr` の各部門内の従業員を、その給与および歩合に基づいてランク付けします。給与が同じ場合は同じランクになるため、連続しないランクになります。この例と、6-51 ページの「[DENSE_RANK](#)」の例を比較してください。

```
SELECT department_id, last_name, salary, commission_pct,
       RANK() OVER (PARTITION BY department_id
                    ORDER BY salary DESC, commission_pct) "Rank"
FROM employees;
```

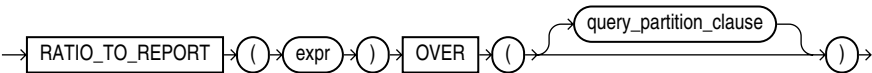
DEPARTMENT_ID	LAST_NAME	SALARY	COMMISSION_PCT	Rank
10	Whalen	4400		1

20	Hartstein	13000	1
20	Fay	6000	2
30	Raphaely	11000	1
30	Khoo	3100	2
30	Baida	2900	3
30	Tobias	2800	4
.			
.			
.			

RATIO_TO_REPORT

構文

ratio_to_report::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

RATIO_TO_REPORT は分析ファンクションです。このファンクションは、ある値の集合の合計に対する、その値の比率を計算します。expr が NULL の場合は、値も NULL になります。

値の集合は、query_partition_clause によって決まります。この句を省略すると、比率は、問合せによって戻されるすべての行で計算されます。

expr には、RATIO_TO_REPORT または他の分析ファンクションを使用できません。他の組込みファンクション式は expr で使用できますが、分析ファンクションはネストできません。

参照： expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、すべての事務員の給与の合計に対する各事務員の給与の割合の値を計算します。

```
SELECT last_name, salary, RATIO_TO_REPORT(salary) OVER () AS rr
FROM employees
WHERE job_id = 'PU_CLERK';
```

LAST_NAME	SALARY	RR
Khoo	3100	.223021583
Baida	2900	.208633094
Tobias	2800	.201438849
Himuro	2600	.18705036
Colmenares	2500	.179856115

RAWTOHEX

構文

rawtohex::=



用途

RAWTOHEX は、*raw* を 16 進で表した文字値に変換します。*raw* 引数は、RAW または BLOB のいずれかのデータ型です。

例

次の例は、RAW 列の値に等しい 16 進を戻します。

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

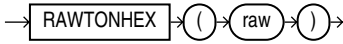
```
Graphics
-----
7D
```

参照： 2-25 ページの「[RAW データ型と LONG RAW データ型](#)」および 6-71 ページの「[HEXTORAW](#)」を参照してください。

RAWTONHEX

構文

rawtonhex::=



用途

RAWTONHEX は、raw を 16 進で表した NVARCHAR2 文字値に変換します。

例

次の例は、RAW 列の値に等しい 16 進を戻します。

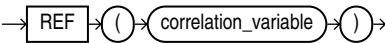
```
SELECT RAWTONHEX (raw_column),
       DUMP ( RAWTONHEX (raw_column) ) "DUMP"
FROM graphics;
```

RAWTONHEX (RA)	DUMP
7D	Typ=1 Len=4: 0,55,0,68

REF

構文

ref::=



用途

REF は、引数としてオブジェクト表またはオブジェクト・ビューの行に関連付けられた相関変数（表別名）を取ります。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。

例

サンプル・スキーマ oe には、次のように定義された cust_address_typ という型が含まれます。

Attribute	Type
STREET_ADDRESS	VARCHAR2 (40)
POSTAL_CODE	VARCHAR2 (10)
CITY	VARCHAR2 (30)
STATE_PROVINCE	VARCHAR2 (10)
COUNTRY_ID	CHAR (2)

次の例では、サンプル型 oe.cust_address_typ に基づく表（addresses 表）を作成した後、その表に行を挿入し、その表からその型のオブジェクト・インスタンスの REF 値を取り出します。

```

CREATE TABLE addresses OF cust_address_typ;

INSERT INTO addresses VALUES (
    '123 First Street', '4GF HlJ', 'Our Town', 'Ourcounty', 'US');

SELECT REF(e) FROM addresses e;

REF(E)
-----
00002802097CD1261E51925B60E0340800208254367CD1261E51905B60E034080020825436010101820000

```

参照：『Oracle9i データベース概要』を参照してください。

REFTOHEX

構文

reftohex::=

→ **REFTOHEX** → () → *expr* → () →

用途

REFTOHEX は、引数 *expr* を 16 進で表した文字値に変換します。*expr* は、REF を戻す必要があります。

例

サンプル・スキーマ *oe* には *warehouse_typ* が含まれています。次の例では、その型を基に、ある列の REF 値をそれに等しい 16 進を含む文字値に変換する方法を示します。

```

CREATE TABLE warehouse_table OF warehouse_typ
    (PRIMARY KEY (warehouse_id));

CREATE TABLE location_table
    (location_number NUMBER, building REF warehouse_typ
    SCOPE IS warehouse_table);

INSERT INTO warehouse_table VALUES (1, 'Downtown', 99);

INSERT INTO location_table SELECT 10, REF(w) FROM warehouse_table w;

SELECT REFTOHEX(building) FROM location_table;

REFTOHEX (BUILDING)
-----
0000220208859B5E9255C31760E034080020825436859B5E9255C21760E034080020825436

```

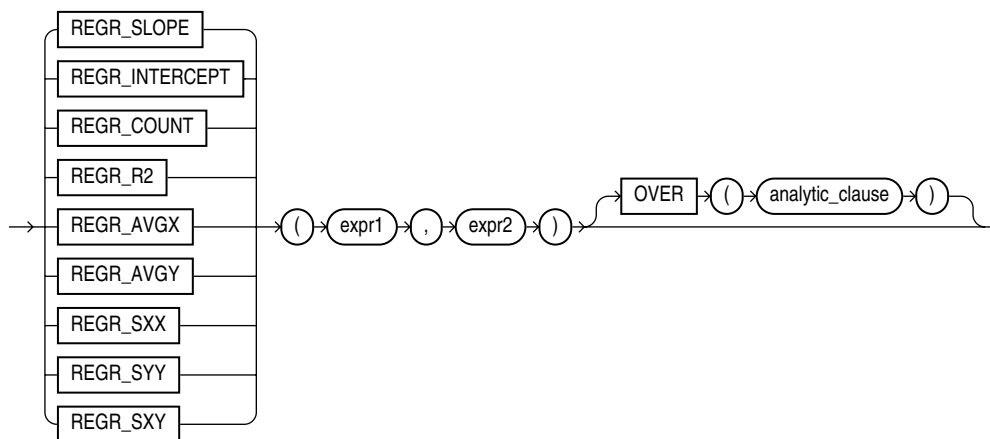
REGR_（線形リグレッション）ファンクション

線形リグレッション・ファンクションは次のとおりです。

- REGR_SLOPE
- REGR_INTERCEPT
- REGR_COUNT
- REGR_R2
- REGR_AVGX
- REGR_AVGY
- REGR_SXX
- REGR_SYY
- REGR_SXY

構文

linear_regr::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

線形リグレッション・ファンクションは、微分最小 2 乗法で求めたリグレッション直線を数値の組の集合に対応付けます。これは、集計ファンクションまたは分析ファンクションとして使用できます。

参照:

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- `expr` の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

Oracle は、`expr1` または `expr2` が NULL であるすべての組を排除した後、このファンクションを (`expr1`, `expr2`) の集合に適用します。Oracle は、データでの引渡し中、同時にすべてのリグレーション・ファンクションを計算します。

`expr1` は、従属変数の値 (y 値) として解析されます。`expr2` は、独立変数の値 (x 値) として解析されます。これらの式は、両方とも数値である必要があります。

- `REGR_SLOPE` は、直線の傾きを戻します。戻り値は数値で、NULL になる場合もあります。NULL (`expr1`, `expr2`) の組を排除した後、このファンクションは次の計算を行います。

```
COVAR_POP(expr1, expr2) / VAR_POP(expr2)
```

- `REGR_INTERCEPT` は、リグレーション・ファンクションの y 切片を戻します。戻り値は数値で、NULL になる場合もあります。NULL (`expr1`, `expr2`) の組を排除した後、このファンクションは次の計算を行います。

```
AVG(expr1) - REGR_SLOPE(expr1, expr2) * AVG(expr2)
```

- `REGR_COUNT` は整数を戻します。この整数は、リグレーション・ファンクションに適応させるために使用される NULL でない数値の組です。
- `REGR_R2` は、リグレーションに対する確定係数 (「R の 2 乗」または「適合度」とも呼ぶ) を戻します。戻り値は数値で、NULL になる場合もあります。VAR_POP (`expr1`) および VAR_POP (`expr2`) は、NULL の組が排除された後に評価されます。戻り値は次のとおりです。

```
NULL if VAR_POP(expr2) = 0
```

```
1 if VAR_POP(expr1) = 0 and  
   VAR_POP(expr2) != 0
```

```
POWER(CORR(expr1,expr),2) if VAR_POP(expr1) > 0 and  
   VAR_POP(expr2) != 0
```

これ以外のすべてのリグレーション・ファンクションの戻り値は数値で、NULL になる場合もあります。

- `REGR_AVGX` は、リグレーション直線の独立変数 (`expr2`) の平均を求めます。NULL (`expr1`, `expr2`) の組を排除した後、このファンクションは次の計算を行います。

```
AVG(expr2)
```

- REGR_AVGY は、リグレッション直線の従属変数（*expr1*）の平均を求めます。NULL（*expr1*、*expr2*）の組を排除した後、このファンクションは次の計算を行います。

```
AVG(expr1)
```

REGR_SXY、REGR_SXX、REGR_SYY は補助ファンクションです。これらは、様々な診断統計の計算に使用されます。

- NULL（*expr1*、*expr2*）の組を排除した後、REGR_SXX は次の計算を行います。

```
REGR_COUNT(expr1, expr2) * VAR_POP(expr2)
```

- NULL（*expr1*、*expr2*）の組を排除した後、REGR_SYY は次の計算を行います。

```
REGR_COUNT(expr1, expr2) * VAR_POP(expr1)
```

- NULL（*expr1*、*expr2*）の組を排除した後、REGR_SXY は次の計算を行います。

```
REGR_COUNT(expr1, expr2) * COVAR_POP(expr1, expr2)
```

次の例は、サンプル表 `sh.sales` および `sh.products` に基づいています。

一般的な線形リグレッションの例

次の例では、様々な線形リグレッション・ファンクションを比較します。

```
SELECT
s.channel_id,
REGR_SLOPE(s.quantity_sold, p.prod_list_price) SLOPE ,
REGR_INTERCEPT(s.quantity_sold, p.prod_list_price) INTCP ,
REGR_R2(s.quantity_sold, p.prod_list_price) RSQR ,
REGR_COUNT(s.quantity_sold, p.prod_list_price) COUNT ,
REGR_AVGX(s.quantity_sold, p.prod_list_price) AVGLISTP ,
REGR_AVGY(s.quantity_sold, p.prod_list_price) AVGQSOLD
FROM sales s, products p
WHERE s.prod_id=p.prod_id AND
p.prod_category='Men' AND
s.time_id=to_DATE('10-OCT-2000')
GROUP BY s.channel_id
;
```

C	SLOPE	INTCP	RSQR	COUNT	AVGLISTP	AVGQSOLD
C	-.03529838	16.4548382	.217277422	17	87.8764706	13.3529412
I	-.0108044	13.3082392	.028398018	43	116.77907	12.0465116
P	-.01729665	11.3634927	.026191191	33	80.5818182	9.96969697
S	-.01277499	13.488506	.000473089	71	52.571831	12.8169014
T	-.01026734	5.01019929	.064283727	21	75.2	4.23809524

REGR_SLOPE および REGR_INTERCEPT の例

次の例では、各会計年度の売上高および売上利益について、リグレーション直線の傾きおよび切片を計算します。

```
SELECT t.fiscal_year,
       REGR_SLOPE(s.amount_sold, s.quantity_sold) "Slope",
       REGR_INTERCEPT(s.amount_sold, s.quantity_sold) "Intercept"
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.fiscal_year;
```

FISCAL_YEAR	Slope	Intercept
1998	49.3934247	71.6015479
1999	49.3443482	70.1502601
2000	49.2262135	75.0287476

次の例では、1998 年の最後の 3 週間 (fiscal_week_number が 50、51 または 52) の各週末トランザクション (day_number_in_week が 6 または 7) のうち、2 つの製品 (270 および 260) の売上高および売上利益について、リグレーション直線の傾きおよび切片を計算します。

```
SELECT t.fiscal_month_number "Month", t.day_number_in_month "Day",
       REGR_SLOPE(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_month_desc, t.day_number_in_month) AS CUM_SLOPE,
       REGR_INTERCEPT(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_month_desc, t.day_number_in_month) AS CUM_ICPT
FROM sales s, times t
WHERE s.time_id = t.time_id
      AND s.prod_id IN (270, 260)
      AND t.fiscal_year=1998
      AND t.fiscal_week_number IN (50, 51, 52)
      AND t.day_number_in_week IN (6,7)
ORDER BY t.fiscal_month_desc, t.day_number_in_month;
```

Month	Day	CUM_SLOPE	CUM_ICPT
12	12	-68	1872
12	12	-68	1872
12	13	-20.244898	1254.36735
12	13	-20.244898	1254.36735
12	19	-18.826087	1287
12	20	62.4561404	125.28655
12	20	62.4561404	125.28655
12	20	62.4561404	125.28655
12	20	62.4561404	125.28655
12	26	67.2658228	58.9712313

```
12          26 67.2658228 58.9712313
12          27 37.5245541 284.958221
12          27 37.5245541 284.958221
12          27 37.5245541 284.958221
```

REGR_COUNT の例

次の例では、customers 表のアカウント・マネージャを持つ顧客の数（合計は 319）を戻します。

```
SELECT REGR_COUNT(customer_id, account_mgr_id) FROM customers;

REGR_COUNT(CUSTOMER_ID,ACCOUNT_MGR_ID)
-----
231
```

次の例では、1998 年 4 月のトランザクションの累積数を 1 日ごとに計算します。

```
SELECT UNIQUE t.day_number_in_month,
  REGR_COUNT(s.amount_sold, s.quantity_sold)
  OVER (PARTITION BY t.fiscal_month_number
  ORDER BY t.day_number_in_month) "Regr_Count"
FROM sales s, times t
WHERE s.time_id = t.time_id
AND t.fiscal_year = 1998 AND t.fiscal_month_number = 4;

DAY_NUMBER_IN_MONTH  Reqr_Count
-----
1          825
2         1650
3         2475
4         3300
.
.
.
26        21450
30        22200
```

REGR_R2 の例

次の例では、5,000 より多い売上高および売上利益に対する、リグレーション直線の確定係数を計算します。

```
SELECT REGR_R2(amount_sold, quantity_sold) FROM sales
  WHERE amount_sold > 5000;

REGR_R2 (AMOUNT_SOLD,QUANTITY_SOLD)
-----
.024087453
```

次の例では、1998 年の各月の 1 か月の売上高および利益について、リグレーション直線の累積確定係数を計算します。

```
SELECT t.fiscal_month_number,
       REGR_R2(SUM(s.amount_sold), SUM(s.quantity_sold))
       OVER (ORDER BY t.fiscal_month_number) "Regr_R2"
FROM sales s, times t
WHERE s.time_id = t.time_id
AND t.fiscal_year = 1998
GROUP BY t.fiscal_month_number
ORDER BY t.fiscal_month_number;
```

FISCAL_MONTH_NUMBER	Regr_R2
1	
2	1
3	.927372984
4	.807019972
5	.932745567
6	.94682861
7	.965342011
8	.955768075
9	.959542618
10	.938618575
11	.880931415
12	.882769189

REGR_AVGY および REGR_AVGX の例

次の例では、各年度の売上高および売上利益について、リグレーション平均を計算します。

```
SELECT t.fiscal_year,
       REGR_AVGY(s.amount_sold, s.quantity_sold) "Regr_AvgY",
       REGR_AVGX(s.amount_sold, s.quantity_sold) "Regr_AvgX"
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.fiscal_year;
```

FISCAL_YEAR	Regr_AvgY	Regr_AvgX
1998	716.602044	13.0584283
1999	714.910831	13.0665536
2000	717.331304	13.0479781

次の例では、1998 年 12 月の最後の 2 週間の製品 260 の売上高および売上利益の累積平均を計算します。

```
SELECT t.day_number_in_month,
       REGR_AVGY(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_month_desc, t.day_number_in_month)
         "Regr_AvgY",
       REGR_AVGX(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_month_desc, t.day_number_in_month)
         "Regr_AvgX"
FROM sales s, times t
WHERE s.time_id = t.time_id
      AND s.prod_id = 260
      AND t.fiscal_month_desc = '1998-12'
      AND t.fiscal_week_number IN (51, 52)
ORDER BY t.day_number_in_month;
```

DAY_NUMBER_IN_MONTH	Regr_AvgY	Regr_AvgX
-----	-----	-----
14	882	24.5
14	882	24.5
15	801	22.25
15	801	22.25
16	777.6	21.6
18	642.857143	17.8571429
18	642.857143	17.8571429
20	589.5	16.375
21	544	15.1111111
22	592.363636	16.4545455
22	592.363636	16.4545455
24	553.846154	15.3846154
24	553.846154	15.3846154
26	522	14.5
27	578.4	16.0666667

REGR_SXY、REGR_SXX および REGR_SYY の例

次の例では、サンプル表 sh.sales の各年度の売上高および売上利益のリグレーション解析について、REGR_SXY、REGR_SXX および REGR_SYY の値を計算します。

```
SELECT t.fiscal_year,
       REGR_SXY(s.amount_sold, s.quantity_sold) "Regr_sxy",
       REGR_SYY(s.amount_sold, s.quantity_sold) "Regr_syy",
       REGR_SXX(s.amount_sold, s.quantity_sold) "Regr_sxx"
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.fiscal_year;
```

FISCAL_YEAR	Regr_sxy	Regr_syy	Regr_sxx
1998	1620591607	2.3328E+11	32809865.2
1999	1955866724	2.7695E+11	39637097.2
2000	2127877398	3.0630E+11	43226509.7

次の例では、1998 年の各月の製品 260 および 270 の週末の売上高および売上利益に対する、REGR_SXY、REGR_SXX および REGR_SYY の累積統計を計算します。

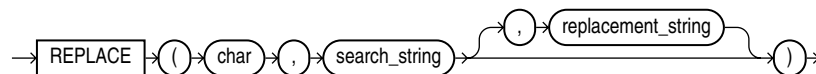
```
SELECT t.day_number_in_month,
       REGR_SXY(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_year, t.fiscal_month_desc) "Regr_sxy",
       REGR_SYY(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_year, t.fiscal_month_desc) "Regr_syy",
       REGR_SXX(s.amount_sold, s.quantity_sold)
         OVER (ORDER BY t.fiscal_year, t.fiscal_month_desc) "Regr_sxx"
FROM sales s, times t
WHERE s.time_id = t.time_id
      AND prod_id IN (270, 260)
      AND t.fiscal_month_desc = '1998-02'
      AND t.day_number_in_week IN (6,7)
ORDER BY t.day_number_in_month;
```

DAY_NUMBER_IN_MONTH	Regr_sxy	Regr_syy	Regr_sxx
1	130973783	1.8916E+10	2577797.94
.			
.			
.			
30	130973783	1.8916E+10	2577797.94

REPLACE

構文

replace::=



用途

REPLACE は、*replacement_string* ですべての *search_string* を変換して *char* を戻します。*replacement_string* を指定しない場合または NULL の場合、すべての *search_string* が削除されます。*search_string* が NULL の場合、*char* が戻されます。

char と同様に、*search_string* および *replacement_string* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

このファンクションは、TRANSLATE ファンクションに関連する機能を提供します。TRANSLATE は、単一文字を 1 対 1 で置き換えます。REPLACE ファンクションでは、文字列の置換または削除を実行できます。

参照： 6-178 ページの「[TRANSLATE](#)」を参照してください。

例

次の例では、「J」を「BL」に置換します。

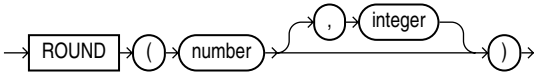
```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
      FROM DUAL;
```

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

ROUND（数値）

構文

round_number::=



用途

ROUND は、*number* を小数点以下 *integer* 桁に丸めた値を戻します。*integer* が省略されると、*number* は小数点以下が丸められます。*integer* が負の場合は小数点の左桁が丸められます。*integer* は整数である必要があります。

例

次の例では、数値を小数点以下 1 桁に丸めます。

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

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

次の例では、数値の小数点の左 1 桁を丸めます。

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

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

ROUND (日付)

構文

round_date::=



用途

ROUND は、*date* を書式モデル *fmt* で指定した単位に丸めた結果を戻します。*fmt* を省略すると、*date* は最も近い日に丸められます。

参照： *fmt* で使用できる書式モデルについては、6-210 ページの「ROUND および TRUNC 日付ファンクション」を参照してください。

例

次の例では、次の年の最も近い日に丸めます。

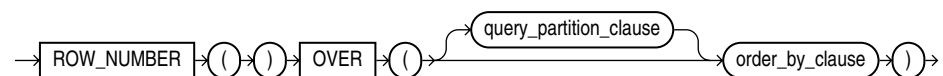
```
SELECT ROUND (TO_DATE ('27-OCT-00'), 'YEAR')
       "New Year" FROM DUAL;
```

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

ROW_NUMBER

構文

row_number::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析関クション](#)」を参照してください。

用途

ROW_NUMBER は分析ファンクションです。このファンクションは、*order_by_clause* に指定された行の、1 から始まる順序シーケンスで、このファンクションが適用される各行（パーティションの各行、または問合せが戻す各行）に一意の数値を割り当てます。

expr には、ROW_NUMBER または他の分析ファンクションを使用できません。他の組込みファンクション式は *expr* で使用できますが、分析ファンクションはネストできません。

参照： *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

例

次の例では、サンプル表 *oe.employees* 表の各部門について、従業員の雇用日順に数値を各行に割り当てます。

```
SELECT department_id, last_name, employee_id, ROW_NUMBER()
  OVER (PARTITION BY department_id ORDER BY employee_id) AS emp_id
FROM employees;
```

DEPARTMENT_ID	LAST_NAME	EMPLOYEE_ID	EMP_ID
10	Whalen	200	1
20	Hartstein	201	1
20	Fay	202	2
30	Raphaely	114	1
30	Khoo	115	2
30	Baida	116	3
30	Tobias	117	4
30	Himuro	118	5
30	Colmenares	119	6
40	Mavris	203	1
.			
.			
.			
100	Popp	113	6
110	Higgins	205	1
110	Gietz	206	2

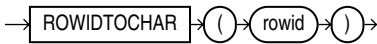
ROW_NUMBER は非決定的なファンクションです。ただし、employee_id は一意キーであるため、この場合のファンクションの結果は決定的になります。

参照： 非決定的な動作の例は、6-64 ページの「FIRST_VALUE」および 6-79 ページの「LAST_VALUE」を参照してください。

ROWIDTOCHAR

構文

rowidtochar::=



用途

ROWIDTOCHAR は、ROWID の値を 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

次の例では、employees 表の ROWID 値を文字値に変換します。(結果はサンプル・データベースのビルドごとに異なります。)

```
SELECT ROWID FROM employees
      WHERE ROWIDTOCHAR(ROWID) LIKE '%SAAb%';

ROWID
-----
AAAFfIAAFAAAABSAAb
```

ROWIDTONCHAR

構文

rowidtonchar::=



用途

ROWIDTONCHAR は、ROWID 値を NVARCHAR2 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

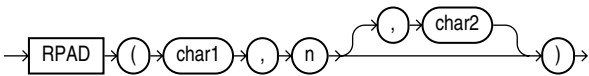
```
SELECT LENGTHB( ROWIDTONCHAR(ROWID) ), ROWIDTONCHAR(ROWID)
FROM employees;
```

```
LENGTHB(ROWIDTONCHAR(ROWID)) ROWIDTONCHAR(ROWID)
-----
36 AAAffIAAFAAAABSAAA
.
.
.
```

RPAD

構文

rpad::=



用途

RPAD は、*char1* の右に *char2* で指定した文字を必要に応じて連続的に埋め込み、長さ *n* にして戻します。*char2* のデフォルト値は 1 個の空白です。*char1* が *n* より長い場合、このファンクションは *n* に収まる *char1* の一部を戻します。

char1 および *char2* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char1* と同じキャラクタ・セットです。

引数 *n* は、戻り値が画面に表示される場合の全体の長さです。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、表示される文字列の長さが文字列の文字数と異なる場合もあります。

例

次の例では、12 文字になるまで、名前の右側に文字「ab」を埋め込みます。

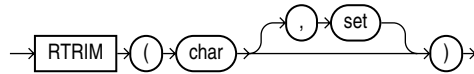
```
SELECT RPAD('MORRISON',12,'ab') "RPAD example"
FROM DUAL;
```

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

RTRIM

構文

rtrim::=



用途

RTRIM は、*char* の右側にあつて *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 1 個の空白です。*char* が文字リテラルの場合、一重引用符で囲む必要があります。RTRIM は LTRIM と同様の働きをします。

char および *set* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*char* と同じキャラクタ・セットです。

例

次の例では、文字列の右側から文字「xy」を削除します。

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

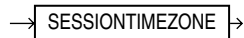
```
RTRIM examp
-----
BROWNINGyxX
```

参照： 6-87 ページの「[LTRIM](#)」を参照してください。

SESSIONTIMEZONE

構文

sessiontimezone::=



用途

SESSIONTIMEZONE は、現行のセッションのタイムゾーンの値を戻します。戻り型は、タイムゾーン・オフセット（'[+|]TZh:TzM' 書式の文字型）またはタイムゾーン地域名です。これは、最近の ALTER SESSION 文でユーザーが指定したセッション・タイムゾーンの値によって異なります。

注意： 環境変数 `ORA_SDTZ` を使用して、デフォルトのクライアント・セッションのタイムゾーンを設定できます。この変数の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

例

次の例では、現行のセッションのタイムゾーンを戻します。

```
SELECT SESSIONTIMEZONE FROM DUAL;

SESSION
-----
-08:00
```

SIGN

構文

`sign::=`



用途

$n < 0$ の場合、SIGN は -1 を戻します。 $n = 0$ の場合は 0 を戻します。 $n > 0$ の場合は 1 を戻します。

例

次の例では、ファンクションの引数 (-15) が 0 より小さいことを示します。

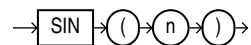
```
SELECT SIGN(-15) "Sign" FROM DUAL;

Sign
-----
-1
```

SIN

構文

sin::=



用途

SIN は、 n （ラジアンで表された角度）のサインを返します。

例

次の例では、30 度のサインを返します。

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

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

SINH

構文

sinh::=



用途

SINH は、 n の双曲線サインを返します。

例

次の例では、1 の双曲線サインを返します。

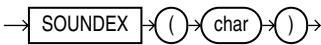
```
SELECT SINH(1) "Hyperbolic sine of 1" FROM DUAL;
```

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

SOUNDEX

構文

soundex::=



用途

SOUNDEX は、*char* と同じ音声表現を持つ文字列を戻します。このファンクションによって、綴りが異なっても発音が類似した英単語を比較できます。

音声表現については、『The Art of Computer Programming, Volume 3: Sorting and Searching』(Donald E. Knuth 著) で次のように定義されています。

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

b, f, p, v = 1
c, g, j, k, q, s, x, z = 2
d, t = 3
l = 4
m, n = 5
r = 6
- 元の名前（1 つ目の定義を行う前）で、同じ数値を持つ 2 つ以上の文字が並んでいるか、または h と w の間の文字以外と並んでいる場合は、最初の文字以外のすべての文字を削除します。
- 最初の 4 バイトを 0（ゼロ）で埋めて戻します。

char は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型です。戻り値は、*char* と同じデータ型です。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、「Smyth」と同じ音声表現を持つ従業員を戻します。

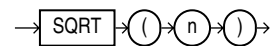
```
SELECT last_name, first_name
       FROM hr.employees
       WHERE SOUNDEX(last_name)
              = SOUNDEX('SMYTHE');
```

```
LAST_NAME  FIRST_NAME
-----
Smith      Lindsey
Smith      William
```

SQRT

構文

sqrt::=

**用途**

SQRT は、 n の平方根を戻します。値 n は負の値にはできません。SQRT は実数を戻します。

例

次の例では、26 の平方根を戻します。

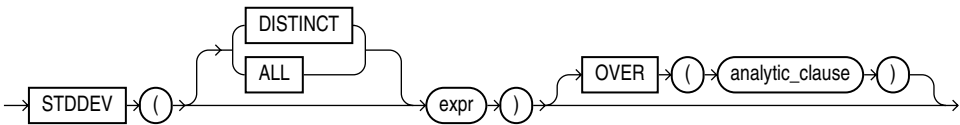
```
SELECT SQRT(26) "Square root" FROM DUAL;
```

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

STDDEV

構文

stddev::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

STDDEV は、数値の集合である `expr` の標本標準偏差を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。このファンクションは、STDDEV_SAMP が NULL を戻すことに対して、入力データが 1 行のみの場合に STDDEV が 0 (ゼロ) を戻すという点で、STDDEV_SAMP と異なります。

Oracle は、VARIANCE 集計ファンクション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。

DISTINCT を指定する場合は、`analytic_clause` の `query_partition_clause` のみ指定できます。`order_by_clause` および `windowing_clause` は指定できません。

参照：

- 詳細は、6-7 ページの「[集計ファンクション](#)」、6-196 ページの「[VARIANCE](#)」および 6-142 ページの「[STDDEV_SAMP](#)」を参照してください。
- `expr` の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 `hr.employees` の給与値の標本標準偏差を戻します。

```
SELECT STDDEV(salary) "Deviation"
FROM employees;
```

```
Deviation
-----
3909.36575
```


分析の例

次の例では、hire_date で順序付けられたサンプル表 hr.employees の部門 80 の給与の値の累積標準偏差を戻します。

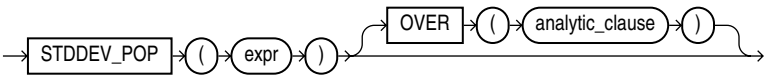
```
SELECT last_name, salary,
       STDDEV(salary) OVER (ORDER BY hire_date) "StdDev"
FROM employees
WHERE department_id = 30;
```

LAST_NAME	SALARY	StdDev
Raphaely	11000	0
Khoo	3100	5586.14357
Tobias	2800	4650.0896
Baida	2900	4035.26125
Himuro	2600	3649.2465
Colmenares	2500	3362.58829

STDDEV_POP

構文

stddev_pop::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

STDDEV_POP は母集団標準偏差を計算し、母集団分散の平方根を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr は数値式であり、このファンクションは NUMBER 型の値を戻します。このファンクションは、VAR_POP ファンクションの平方根と同じです。VAR_POP が NULL を戻す場合、このファンクションも NULL を戻します。

参照：

- 6-7 ページの「[集計ファンクション](#)」および 6-193 ページの「[VAR_POP](#)」を参照してください。
- expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 sh.sales にある売上高の母集団標準偏差および標本標準偏差を返します。

```
SELECT STDDEV_POP(amount_sold) "Pop",
       STDDEV_SAMP(amount_sold) "Samp"
FROM sales;
```

Pop	Samp
-----	-----
896.355151	896.355592

分析の例

次の例では、サンプル表 hr.employees の部門ごとの給与の母集団標準偏差を返します。

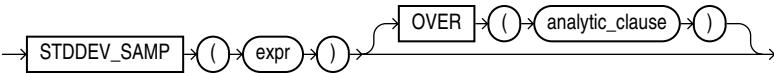
```
SELECT department_id, last_name, salary,
       STDDEV_POP(salary) OVER (PARTITION BY department_id) AS pop_std
FROM employees;
```

DEPARTMENT_ID	LAST_NAME	SALARY	POP_STD
-----	-----	-----	-----
10	Whalen	4400	0
20	Hartstein	13000	3500
20	Goyal	6000	3500
.			
.			
.			
100	Sciarra	7700	1644.18166
100	Urman	7800	1644.18166
100	Popp	6900	1644.18166
110	Higgins	12000	1850
110	Gietz	8300	1850

STDDEV_SAMP

構文

stddev_samp::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

STDDEV_SAMP は標本累積標準偏差を計算し、標本分散の平方根を返します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr は数値式であり、このファンクションは NUMBER 型の値を返します。このファンクションは、VAR_SAMP ファンクションの平方根と同じです。VAR_SAMP が NULL を返す場合、このファンクションも NULL を返します。

参照：

- 6-7 ページの「[集計ファンクション](#)」および 6-194 ページの「[VAR_SAMP](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 sh.sales にある売上高の母集団標準偏差および標本標準偏差を返します。

```
SELECT STDDEV_POP(amount_sold) "Pop",
       STDDEV_SAMP(amount_sold) "Samp"
FROM sales;
```

Pop	Samp
896.355151	896.355592

分析の例

次の例では、employees 表の部門ごとの給与の標本標準偏差を返します。

```
SELECT department_id, last_name, hire_date, salary,
       STDDEV_SAMP(salary) OVER (PARTITION BY department_id
                                ORDER BY hire_date
                                ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sdev
FROM employees;
```

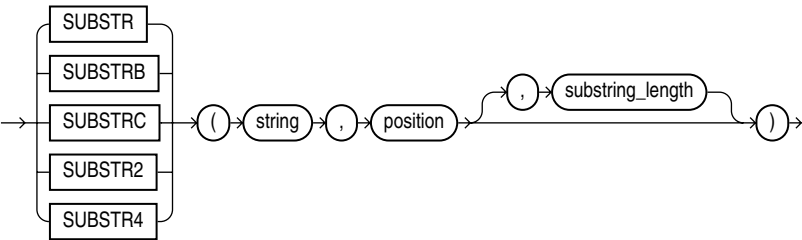
DEPARTMENT_ID	LAST_NAME	HIRE_DATE	SALARY	CUM_SDEV
10	Whalen	17-SEP-87	4400	
20	Hartstein	17-FEB-96	13000	
20	Goyal	17-AUG-97	6000	4949.74747
30	Raphaely	07-DEC-94	11000	
30	Khoo	18-MAY-95	3100	5586.14357
30	Tobias	24-JUL-97	2800	4650.0896
30	Baida	24-DEC-97	2900	4035.26125

.			
.			
.			
	100	Chen	28-SEP-97 8200 2003.33056
	100	Sciarra	30-SEP-97 7700 1925.91969
	100	Urman	07-MAR-98 7800 1785.49713
	100	Popp	07-DEC-99 6900 1801.11077
	110	Higgins	07-JUN-94 12000
	110	Gietz	07-JUN-94 8300 2616.29509

SUBSTR

構文

substr::=



用途

SUBSTR の各ファンクションは、*string* の *position* の文字から *substring_length* 文字分の文字列を抜き出して戻します。SUBSTR は、入力キャラクタ・セットによって定義された文字を使用して、長さを計算します。SUBSTRB は、文字のかわりにバイトを使用します。SUBSTRC は、完全な Unicode キャラクタを使用します。SUBSTR2 は、UCS2 コードポイントを使用します。SUBSTR4 は、UCS4 コードポイントを使用します。

- *position* が 0 の場合、1 として処理されます。
- *position* が正の数の場合、Oracle は *string* の始めから数えて最初の文字を検索します。
- *position* が負の場合、Oracle は *string* の終わりから逆方向にカウントします。
- *substring_length* を指定しないと、Oracle は *string* の終わりまでのすべての文字を戻します。*substring_length* が 1 より小さい場合、NULL を戻します。

string は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻り値は、*string* と同じデータ型です。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

次の例では、「ABCDEFGH」の指定されたサブストリングを戻します。

```
SELECT SUBSTR('ABCDEFGH',3,4) "Substring"
      FROM DUAL;
```

Substring

CDEF

```
SELECT SUBSTR('ABCDEFGH',-5,4) "Substring"
      FROM DUAL;
```

Substring

CDEF

データベース・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('ABCDEFGH',5,4.2) "Substring with bytes"
      FROM DUAL;
```

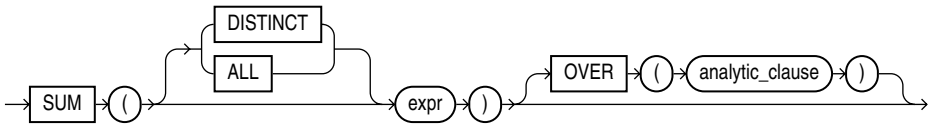
Substring with bytes

CD

SUM

構文

sum::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

SUM は、*expr* の値の合計を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。*order_by_clause* および *windowing_clause* は指定できません。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 `hr.employees` にあるすべての給与の合計を計算します。

```
SELECT SUM(salary) "Total"
FROM employees;
```

Total

691400

分析の例

次の例では、サンプル表 `hr.employees` の各マネージャについて、そのマネージャの下で働く従業員の現在の給与以下の給与の累積合計を計算します。Raphaely および Cambrault が同じ累積を持っています。これは、Raphaely および Cambrault が同じ給与であるため、Oracle が給与の値を同時に追加し、同じ累積合計を両方の行に対して適用したためです。

```
SELECT manager_id, last_name, salary,
       SUM(salary) OVER (PARTITION BY manager_id ORDER BY salary
       RANGE UNBOUNDED PRECEDING) l_csum
FROM employees;
```

MANAGER_ID	LAST_NAME	SALARY	L_CSUM
-----	-----	-----	-----
100	Mourgos	5800	5800
100	Vollman	6500	12300
100	Kaufling	7900	20200
100	Weiss	8000	28200
100	Fripp	8200	36400
100	Zlotkey	10500	46900
100	Raphaely	11000	68900
100	Cambrault	11000	68900
100	Errazuriz	12000	80900

```

.
.
.
149 Taylor          8600      30200
149 Hutton          8800      39000
149 Abel            11000     50000
201 Fay              6000      6000
205 Gietz            8300      8300
      King           24000     24000

```

SYS_CONNECT_BY_PATH

構文

sys_connect_by_path::=

→ SYS_CONNECT_BY_PATH → (→ column → , → char →) →

用途

SYS_CONNECT_BY_PATH は、階層問合せのみで有効です。このファンクションは、ルートからノードへの列の値のパスを、CONNECT BY 条件によって戻された各行を *char* で区切った列の値とともに戻します。

column および *char* は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型です。戻される文字列は、VARCHAR2 データ型であり、*column* と同じキャラクタ・セットです。

参照： 階層問合せおよび CONNECT BY 条件の詳細は、8-3 ページの「[階層問合せ](#)」を参照してください。

例

次の例では、従業員 Kochhar から Kochhar のすべての従業員（およびそれらの従業員の従業員）への従業員名のパスを戻します。

```

SELECT LPAD(' ', 2*level-1) || SYS_CONNECT_BY_PATH(last_name, '/') "Path"
FROM employees
START WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id;

```

Path

```

-----
/Kochhar
/Kochhar/Greenberg
/Kochhar/Greenberg/Faviet
/Kochhar/Greenberg/Chen

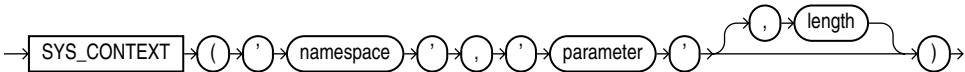
```

```
/Kochhar/Greenberg/Sciarra  
/Kochhar/Greenberg/Urman  
/Kochhar/Greenberg/Popp  
/Kochhar/Whalen  
/Kochhar/Mavris  
/Kochhar/Baer  
/Kochhar/Higgins  
/Kochhar/Higgins/Gietz
```

SYS_CONTEXT

構文

sys_context::=



用途

SYS_CONTEXT は、コンテキスト *namespace* に関連付けられた *parameter* の値を戻します。このファンクションは、SQL 文および PL/SQL 文で使用できます。

namespace および *parameter* には、文字列（定数）、またはネームスペースまたは属性を指定する文字列に変換する式を指定できます。コンテキスト *namespace* はすでに作成されている必要があり、関連付けられた *parameter* およびその値は、DBMS_SESSION.set_context プロシージャを使用して設定されている必要があります。*namespace* は有効な SQL 識別子である必要があります。*parameter* 名には、すべての文字列を指定できます。大 / 小文字を識別しますが、長さは 30 バイト以下です。

戻り値のデータ型は VARCHAR2 です。戻り値のデフォルトの最大サイズは、256 バイトです。オプションの *length* パラメータを指定して、このデフォルトを上書きできます。値の有効範囲は 1 ～ 4000 バイトです。（無効な値を指定すると、Oracle はその値を無視してデフォルト値を使用します。）

Oracle9i では、現行のセッションを記述する USERENV という組込みネームスペースを提供しています。ネームスペース USERENV の事前定義パラメータおよびその戻り文字列の長さについては、6-149 ページの表 6-2 を参照してください。

SYS_CONTEXT の制限事項： この機能は、パラレル問合せでは指定できません。

参照：

- アプリケーション開発でのアプリケーション・コンテキスト機能の使用
方法については、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- ユーザー定義のコンテキスト・ネームスペースの作成方法について
は、13-11 ページの「CREATE CONTEXT」を参照してください。
- DBMS_SESSION.set_context プロシージャについては、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

例

次の例では、データベースにログインしたユーザー名を戻します。

```
CONNECT OE/OE
SELECT SYS_CONTEXT ('USERENV', 'SESSION_USER')
FROM DUAL;
```

```
SYS_CONTEXT ('USERENV', 'SESSION_USER')
-----
OE
```

次の例では、hr_apps の作成時に、コンテキスト hr_apps に関連付けられた PL/SQL パッケージの属性 group_no に対する値として設定されたグループ番号を戻します。

```
SELECT SYS_CONTEXT ('hr_apps', 'group_no') "User Group"
FROM DUAL;
```

表 6-2 ネームスペース USERENV の事前定義パラメータ

パラメータ	戻り値	戻り値の長さ (バイト)
AUDITED_CURSORID	監査をトリガーによって実行した SQL のカーソル ID を戻します。	指定なし
AUTHENTICATION_DATA	ログイン・ユーザーの認証に使用されるデータを戻します。X.503 認証セッションでは、このフィールドは HEX2 形式での認証のコンテキストを戻します。 注意： 構文の length パラメータを使用して、AUTHENTICATION_DATA 属性の戻り値を変更できます。最大 4000 までの値を指定できます。この属性は、Oracle がこのような変更を実行する USERENV の唯一の属性です。	256

表 6-2 ネームスペース USERENV の事前定義パラメータ (続き)

パラメータ	戻り値	戻り値の長さ (バイト)
AUTHENTICATION_TYPE	次に示すとおり、ユーザーの認証方法を戻します。 ■ DATABASE: ユーザー名 / パスワード認証 ■ OS: オペレーティング・システムの外部ユーザー認証 ■ NETWORK: ネットワーク・プロトコルまたは ANO 認証 ■ PROXY: OCI プロキシ接続認証	30
BG_JOB_ID	現行のセッションが Oracle のバックグラウンド・プロセスで確立された場合、そのセッションのジョブ ID を戻します。セッションがバックグラウンド・プロセスで確立されていない場合は、NULL を戻します。	64
CLIENT_IDENTIFIER	グローバル・コンテキスト (グローバルにアクセスされたアプリケーション・コンテキスト、または OCI コンテキストの OCI_ATTR_CLIENT_IDENTIFIER 属性) のクライアントのセッション識別子を戻します。関連する識別子がグローバルに設定されていない場合は、NULL を戻します。	指定なし
CLIENT_INFO	DBMS_APPLICATION_INFO パッケージを使用するアプリケーションが格納できる 64 バイトまでのユーザー・セッション情報を戻します。	64
CURRENT_SCHEMA	カレント・スキーマで使用されているデフォルトのスキーマ名を戻します。この値は、セッション中に ALTER SESSION SET CURRENT_SCHEMA 文を使用して変更できます。	30
CURRENT_SCHEMAID	現行のセッションで使用されているデフォルトのスキーマ ID を戻します。	30
CURRENT_SQL	ファイングレイン監査イベントをトリガーによって実行した現行の SQL を戻します。この属性は、ファイングレイン監査機能のイベント・ハンドラ内のみで指定できます。	64
CURRENT_USER	現行のセッションで権限のあるユーザーのユーザー名を戻します。	30
CURRENT_USERID	現行のセッションで権限のあるユーザーのユーザー ID を戻します。	30

表 6-2 ネームスペース USERENV の事前定義パラメータ（続き）

パラメータ	戻り値	戻り値の長さ (バイト)
DB_DOMAIN	DB_DOMAIN 初期化パラメータで指定されたデータベースのドメインを戻します。	256
DB_NAME	DB_NAME 初期化パラメータで指定されたデータベース名を戻します。	30
ENTRY_ID	使用可能な監査エントリ識別子を戻します。この属性を分散 SQL 文で使用することはできません。USERENV でこのキーワードを使用するには、AUDIT_TRAIL 初期化パラメータに TRUE を設定する必要があります。	30
EXTERNAL_NAME	データベース・ユーザーの外部名を戻します。v.503 認証を使用する SSL 認証セッションでは、このフィールドは、ユーザー認証に格納されている識別名 (DN) を戻します。	256
FG_JOB_ID	現行のセッションが Oracle のフォアグラウンド・プロセスで確立された場合、そのセッションのジョブ ID を戻します。セッションがフォアグラウンド・プロセスで確立されていない場合は、NULL を戻します。	30
GLOBAL_CONTEXT_MEMORY	コンテキストへのグローバルなアクセスによって、システム・グローバル領域で使用された数値を戻します。	指定なし
HOST	接続中のクライアントのホスト・マシン名を戻します。	54
INSTANCE	現行のインスタンスのインスタンス識別番号を戻します。	30
IP_ADDRESS	接続中のクライアントのマシンの IP アドレスを戻します。	30
ISDBA	オペレーティング・システムまたはパスワード・ファイルによって、ユーザーが DBA 権限を持っていると認証された場合、TRUE を戻します。	30
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。	62
LANGUAGE	現行のセッションで使用している言語 (language) および地域 (territory) を、データベース・キャラクタ・セット (character set) も含めて次の書式で戻します。 language_territory.characterset	52

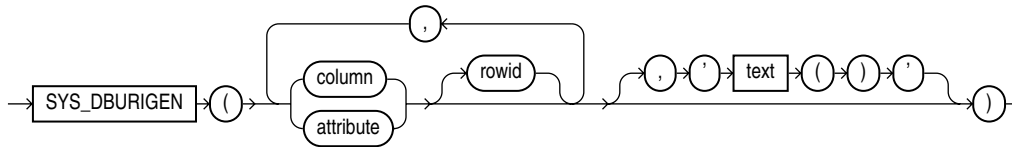
表 6-2 ネームスペース USERENV の事前定義パラメータ（続き）

パラメータ	戻り値	戻り値の長さ (バイト)
NETWORK_PROTOCOL	接続文字列の 'PROTOCOL=protocol' の部分で指定された、通信に使用されるネットワーク・プロトコルを戻します。	256
NLS_CALENDAR	現行のセッションの現行のカレンダを戻します。	62
NLS_CURRENCY	現行のセッションの通貨を戻します。	62
NLS_DATE_FORMAT	セッションの日付書式を戻します。	62
NLS_DATE_LANGUAGE	日付の表示に使用される言語を戻します。	62
NLS_SORT	BINARY または言語ソート基準を戻します。	62
NLS_TERRITORY	現行のセッションの地域を戻します。	62
OS_USER	データベース・セッションを初期化するクライアント・プロセスのオペレーティング・システム・ユーザー名を戻します。	30
PROXY_USER	SESSION_USER のかわりに現行のセッションを開いたデータベース・ユーザー名を戻します。	30
PROXY_USERID	SESSION_USER のかわりに現行のセッションを開いたデータベース・ユーザーの ID を戻します。	30
SESSION_USER	現行のユーザーが認証されているデータベース・ユーザー名を戻します。この値は、セッションの存続期間中は同じです。	30
SESSION_USERID	現行のユーザーが認証されているデータベース・ユーザーの ID を戻します。	30
SESSIONID	監査セッション識別子を戻します。この属性を分散 SQL 文で使用することはできません。	30
TERMINAL	現行のセッションのクライアントに対するオペレーティング・システムの識別子を戻します。分散 SQL 文では、この属性はローカル・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 に対してのみこのオプションを使用でき、リモートの INSERT、UPDATE または DELETE には使用できません。（このパラメータの戻り値の長さはオペレーティング・システムによって異なります。）	10

SYS_DBURIGEN

構文

sys_dburigen::=



用途

SYS_DBURIGen は、引数として 1 つ以上の列または属性、およびオプションで ROWID を取り、特定の列または行オブジェクトへの DBUriType データ型の URL を生成します。これによって、データベースから XML 文書を検索するための URL を使用できるようになります。

参照されるすべての列または属性は、サンプル表に存在する必要があります。これらは、主キーの役割を果たす必要があります。つまり、実際に表の主キーに一致する必要はありませんが、一意の値を参照する必要があります。複数の列を指定すると、最後の列以外のすべての列はデータベースの行を識別し、指定された最後の列は行にある列を識別します。

デフォルトでは、URL はフォーマットされた XML 文書を指します。ドキュメントのテキストのみを指す場合は、オプションの 'text()' を指定します（この XML コンテキストでは、小文字の 'text' はキーワードであり、構文のプレースホルダではありません）。

列または属性を含む表またはビューは、問合せのコンテキストで指定されるスキーマを持たない場合、Oracle は、表名またはビュー名をパブリック・シノニムとして解析します。

参照： データベースの URIType データ型および XML 文書の詳細は、『Oracle9i XML API リファレンス -XDK および Oracle XML DB』および『Oracle9i XML Developer's Kit ガイド -XDK』を参照してください。

例

次の例では、SYS_DBURIGen ファンクションを使用して、サンプル表 hr.employees の employee_id = 206 である行の email 列への DBUriType データ型の URL を生成します。

```
SELECT SYS_DBURIGEN(employee_id, email)
FROM employees
WHERE employee_id = 206;
```

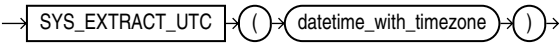
```
SYS_DBURIGEN (EMPLOYEE_ID,EMAIL) (URL, SPARE)
```

```
-----
DBURITYPE ('/PUBLIC/EMPLOYEES/ROW[EMPLOYEE_ID='206']/EMAIL', NULL)
```

SYS_EXTRACT_UTC

構文

sys_extract_utc::=



用途

SYS_EXTRACT_UTC は、タイムゾーンによる時差のある日時から協定世界時（UTC）（以前のグリニッジ標準時）を抽出します。

例

次の例では、指定された日時から UTC を抽出します。

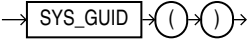
```
SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00 -08:00')
FROM DUAL;
```

```
SYS_EXTRACT_UTC(TIMESTAMP'2000-03-2811:30:00.00-08:00')
-----
28-MAR-00 07.30.00 PM
```

SYS_GUID

構文

sys_guid::=



用途

SYS_GUID は、16 バイトで構成されたグローバルな一意の識別子（RAW 値）を生成して戻します。多くのプラットフォームでは、生成された識別子は、ホスト識別子とプロセス、またはファンクションをコールするプロセスやスレッドのスレッド識別子、およびそのプロセスやスレッドに対する非反復値（バイトの順序）で構成されています。

例

次の例では、サンプル表 `hr.locations` に列を追加後、一意の識別子を各行に挿入し、グローバルな一意識別子の 16 バイトの RAW 値を 32 文字の 16 進表記で戻します。

```
ALTER TABLE locations ADD (uid_col RAW(32));
```

```
UPDATE locations SET uid_col = SYS_GUID();
```

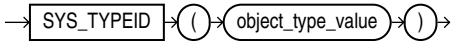
```
SELECT location_id, uid_col FROM locations;
```

LOCATION_ID	UID_COL
1000	7CD5B7769DF75CEFE034080020825436
1100	7CD5B7769DF85CEFE034080020825436
1200	7CD5B7769DF95CEFE034080020825436
1300	7CD5B7769DFA5CEFE034080020825436
.	.
.	.
.	.

SYS_TYPEID

構文

`sys_typeid::=`



用途

`SYS_TYPEID` は、オペランドで最も指定される型の型 ID を戻します。この値は、主に、置換可能な列の基礎となる型判別式の列を識別するために使用されます。たとえば、型判別式の列に索引を構築するために、`SYS_TYPEID` によって戻される値を使用できます。

注意：

- このファンクションは、オブジェクト型のオペランドのみで使用してください。
- すべての最終ルート・オブジェクト型（型階層に属さない最終型）は、NULL 型の ID を持ちます。Oracle は、型階層に属するすべての型に、NULL 以外の一意の型 ID を割り当てます。

参照： 型 ID の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

例

次の例では、15-63 ページの「置換可能な表および列のサンプル:」で作成された表 `persons` および `books` を使用します。これらの表は、どちらも 16-22 ページの「型の階層例:」で作成された `person_t` 型を使用します。最初の間合せは、`persons` 表に格納されたオブジェクト・インスタンスの最も指定される型を戻します。

```
SELECT name, SYS_TYPEID(VALUE(p)) "Type_id" FROM persons p;
```

NAME	Type_id
-----	-----
Bob	01
Joe	02
Tim	03

次の間合せは、`books` 表に格納された作者の最も指定される型を戻します。

```
SELECT b.title, b.author.name, SYS_TYPEID(author)
       "Type_ID" FROM books b;
```

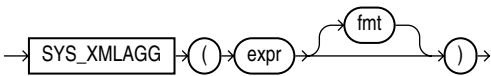
TITLE	AUTHOR.NAME	Type_ID
-----	-----	-----
An Autobiography	Bob	01
Business Rules	Joe	02
Mixing School and Work	Tim	03

`SYS_TYPEID` ファンクションを使用すると、表の型判別式の列に索引を作成できます。13-86 ページの「置換可能な列の索引の例:」の例を参照してください。

SYS_XMLAGG

構文

`sys_xmlagg::=`



用途

SYS_XMLAGG は、*expr* によって表されるすべての XML 文書または XML フラグメントを集約し、単一の XML 文書を生成します。このファンクションは、デフォルト名 ROWSET の新しい囲み要素を追加します。XML 文書を別の方法でフォーマットする場合は、XMLFormat オブジェクトのインスタンスである *fmt* を指定します。

参照：

- SYS_XMLAGG の結果をフォーマットするための XMLFormat 型の属性の使用については、2-74 ページの「XML 書式モデル」を参照してください。
- 6-157 ページの「SYS_XMLGEN」を参照してください。
- XML 型およびそれらの使用については、『Oracle9i XML API リファレンス -XDK および Oracle XML DB』および『Oracle9i XML Developer's Kit ガイド -XDK』を参照してください。

例

次の例では、SYS_XMLGEN ファンクションを使用して、従業員名の最初の文字が R であるサンプル表 employees の各行に対して、XML 文書を生成した後、デフォルトの囲み要素 ROWSET の 1 つの XML 文書にすべての行を集約します。

```
SELECT SYS_XMLAGG(SYS_XMLGEN(last_name))
       FROM employees
       WHERE last_name LIKE 'R%';
```

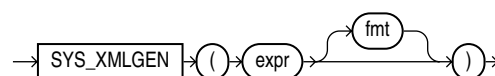
```
SYS_XMLAGG(SYS_XMLGEN(LAST_NAME))
```

```
-----
<ROWSET>
  <LAST_NAME>Raphaely</LAST_NAME>
  <LAST_NAME>Rogers</LAST_NAME>
  <LAST_NAME>Rajs</LAST_NAME>
  <LAST_NAME>Russell</LAST_NAME>
</ROWSET>
```

SYS_XMLGEN

構文

sys_xmlgen::=



用途

SYS_XMLGEN は、データベースの特定の行および列を評価する式を取り、XML 文書を含む XMLType 型のインスタンスを戻します。expr は、スカラー値、ユーザー定義型または XMLType インスタンスです。

- expr がスカラー値である場合、ファンクションはスカラー値を含む XML 要素を戻します。
- expr が型である場合、ファンクションは XML 要素へユーザー定義型の属性をマップします。
- expr が XMLType インスタンスである場合、ファンクションはデフォルトのタグ名が ROW である XML 要素でドキュメントを囲みます。

デフォルトでは、XML 文書の要素は expr の要素と一致します。たとえば、expr が列名に変換される場合、XML の囲み要素は同じ列名になります。XML 文書を別の方法でフォーマットする場合は、XMLFormat オブジェクトのインスタンスである fmt を指定します。

参照：

- XMLFormat 型の詳細および SYS_XMLGEN の結果を書式化するための属性の使用方法については、2-74 ページの「XML 書式モデル」を参照してください。
- XML 型およびそれらの使用については、『Oracle9i XML API リファレンス -XDK および Oracle XML DB』および『Oracle9i XML Developer's Kit ガイド -XDK』を参照してください。

例

次の例では、サンプル表 oe.employees から employee_id 値が 205 の従業員の電子メール ID を検出し、EMAIL 要素を持つ XML 文書を含む XMLType のインスタンスを生成します。

```
SELECT SYS_XMLGEN(email)
       FROM employees
       WHERE employee_id = 205;
```

```
SYS_XMLGEN (EMAIL)
```

```
-----
<EMAIL>SHIGGINS</EMAIL>
```

SYSDATE

構文

sysdate::=

→ SYSDATE →

用途

SYSDATE は現在の日時を戻します。戻り値のデータ型は **DATE** です。このファンクションに引数は不要です。分散 SQL 文では、このファンクションはローカル・データベース上の日時を戻します。**CHECK** 制約の条件でこのファンクションは使用できません。

例

次の例では、現在の日付および時刻を戻します。

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

```
NOW
-----
04-13-2001 09:45:51
```

SYSTIMESTAMP

構文

systimestamp::=

→ SYSTIMESTAMP →

用途

SYSTIMESTAMP は、データベースが存在するシステムのタイムゾーン（小数部を含む）を戻します。戻り型は、**TIMESTAMP WITH TIME ZONE** です。

例

次の例では、システムの日付を戻します。

```
SELECT SYSTIMESTAMP FROM DUAL;
```

```
SYSTIMESTAMP
```

28-MAR-00 12.38.55.538741 PM -08:00

次の例では、小数部を明示的に指定する方法を示します。

SELECT TO_CHAR(SYSTIMESTAMP, 'SSSS.FF') FROM DUAL;

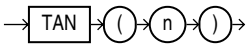
TO_CHAR(SYSTIM

5050.105900

TAN

構文

tan::=



用途

TAN は、*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

次の例では、135 度のタンジェントを戻します。

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

Tangent of 135 degrees

- 1

TANH

構文

tanh::=



用途

TANH は、*n* の双曲線タンジェントを戻します。

例

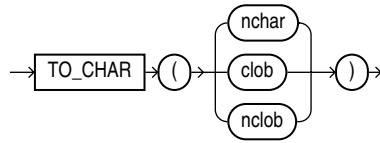
次の例では、5 の双曲線タンジェントを戻します。

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

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

TO_CHAR (文字)**構文**

to_char_char::=

**用途**

TO_CHAR (文字) は、NCHAR、NVARCHAR2、CLOB または NCLOB データをデータベース・キャラクタ・セットに変換します。

例

次の例では、単純な文字列を文字データとして解析します。

```
SELECT TO_CHAR('01110') FROM DUAL;
```

```
TO_CH
-----
01110
```

この例と、6-164 ページの「[TO_CHAR \(数値\)](#)」の最初の例を比較してください。

次の例では、pm.print_media 表の CLOB データをデータベース・キャラクタ・セットに変換します。

```
SELECT TO_CHAR(ad_sourcetext) FROM print_media
       WHERE product_id = 2268;
```

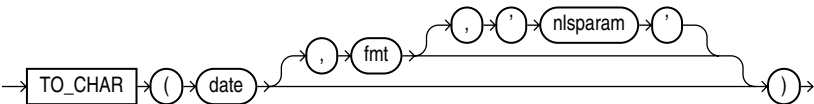
```
TO_CHAR (AD_SOURCETEXT)
-----
```

```
*****
TIGER2 2268...Standard Hayes Compatible Modem
Product ID: 2268
The #1 selling modem in the universe! Tiger2's modem includes call management and
Internet voicing. Make real-time full duplex phone calls at the same time you're
online.
*****
```

TO_CHAR (日時)

構文

to_char_date::=



用途

TO_CHAR (日時) は、DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE または TIMESTAMP WITH LOCAL TIME ZONE データ型の *date* を日付書式 *fmt* で指定された書式の VARCHAR2 データ型に変換します。 *fmt* を省略すると、次のように、*date* は VARCHAR2 値に変換されます。

- DATE は、デフォルトの日付書式の値に変換されます。
- TIMESTAMP および TIMESTAMP WITH LOCAL TIME ZONE は、デフォルトのタイムスタンプ書式の値に変換されます。
- TIMESTAMP WITH TIME ZONE は、タイムゾーン書式のデフォルトのタイムスタンプの値に変換されます。

'*nlsparam*' には、月と日の名前および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'
```

nlsparam を指定しないと、このファンクションはセッションのデフォルト日付言語を使用します。

参照： 日付書式については、2-58 ページの「書式モデル」を参照してください。

例

次の例で使用する表は、次のとおりです。

```
CREATE TABLE date_tab (
    ts_col      TIMESTAMP,
    tsltz_col   TIMESTAMP WITH LOCAL TIME ZONE,
    tstz_col    TIMESTAMP WITH TIME ZONE);
```

次の例では、TO_CHAR を別の TIMESTAMP データ型に適用した結果を示します。
TIMESTAMP WITH LOCAL TIME ZONE 列の結果は、セッションのタイムゾーンを識別します。
これに対して、TIMESTAMP および TIMESTAMP WITH TIME ZONE 列の結果は、セッションのタイムゾーンを識別しません。

```
ALTER SESSION SET TIME_ZONE = '-8:00';
INSERT INTO date_tab VALUES (
    TIMESTAMP'1999-12-01 10:00:00',
    TIMESTAMP'1999-12-01 10:00:00',
    TIMESTAMP'1999-12-01 10:00:00');
INSERT INTO date_tab VALUES (
    TIMESTAMP'1999-12-02 10:00:00 -8:00',
    TIMESTAMP'1999-12-02 10:00:00 -8:00',
    TIMESTAMP'1999-12-02 10:00:00 -8:00');

SELECT TO_CHAR(ts_col, 'DD-MON-YYYY HH24:MI:SSxFF'),
       TO_CHAR(tstz_col, 'DD-MON-YYYY HH24:MI:SSxFF TZh:TZM')
FROM date_tab;

TO_CHAR(TS_COL, 'DD-MON-YYYYHH24:MI:SS')  TO_CHAR(TSTZ_COL, 'DD-MON-YYYYHH24:MI:SS')
-----
01-DEC-1999 10:00:00                      01-DEC-1999 10:00:00.000000 -08:00
02-DEC-1999 10:00:00                      02-DEC-1999 10:00:00.000000 -08:00

SELECT SESSIONTIMEZONE,
       TO_CHAR(tsltz_col, 'DD-MON-YYYY HH24:MI:SSxFF')
FROM date_tab;

SESSIONTIMEZONE  TO_CHAR(TSLTZ_COL, 'DD-MON-YYYYHH24:MI:SS')
-----
-08:00           01-DEC-1999 10:00:00.000000
-08:00           02-DEC-1999 10:00:00.000000

ALTER SESSION SET TIME_ZONE = '-5:00';
SELECT TO_CHAR(ts_col, 'DD-MON-YYYY HH24:MI:SSxFF'),
       TO_CHAR(tstz_col, 'DD-MON-YYYY HH24:MI:SSxFF TZh:TZM')
FROM date_tab;

TO_CHAR(TS_COL, 'DD-MON-YYYYHH24:MI:SS')  TO_CHAR(TSTZ_COL, 'DD-MON-YYYYHH24:MI:SS')
```

```
-----
01-DEC-1999 10:00:00.000000      01-DEC-1999 10:00:00.000000 -08:00
02-DEC-1999 10:00:00.000000      02-DEC-1999 10:00:00.000000 -08:00

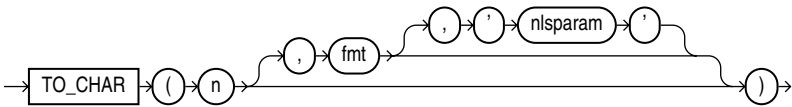
SELECT SESSIONTIMEZONE,
       TO_CHAR(tsltz_col, 'DD-MON-YYYY HH24:MI:SSxFF')
FROM date_tab;

SESSIONTIMEZONE      TO_CHAR(TSLTZ_COL, 'DD-MON-YYYY
-----
-05:00                01-DEC-1999 13:00:00.000000
-05:00                02-DEC-1999 13:00:00.000000
```

TO_CHAR（数値）

構文

to_char_number::=



用途

TO_CHAR（数値）は、NUMBER データ型の *n* を、オプションの数値書式 *fmt* を使用して 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。

'nlsparam' には、数値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
- 桁区切り
- 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* および *g* は、それぞれ小数点文字および桁区切りを表します。これらは、異なるシングルバイト文字である必要があります。引用符付き文字列の中では、パラメータ値を囲む一重引用符を 2 つ使用する必要があります。通貨記号には 10 文字使用できます。

'*nlsparam*' またはパラメータのいずれか 1 つを省略すると、このファンクションはセッションのデフォルト・パラメータ値を使用します。

参照： 日付書式については、2-58 ページの「[書式モデル](#)」を参照してください。

例

次の文は、暗黙的な変換を使用して文字列および数値を数値に解析します。

```
SELECT TO_CHAR('01110' + 1) FROM dual;
```

```
TO_C
----
1111
```

この例と、6-161 ページの「[TO_CHAR \(文字\)](#)」の最初の例を比較してください。

次の例では、出力で通貨記号の左側に空白埋めが行われます。

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

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

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

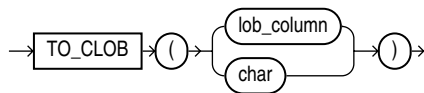
```
Amount
-----
AusDollars10.000,00-
```

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。数値書式要素のすべてのリストは、2-61 ページの[表 2-13](#) を参照してください。

TO_CLOB

構文

to_clob::=



用途

TO_CLOB は、LOB 列またはその他の文字列の NCLOB 値を CLOB 値に変換します。*char* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。Oracle は、基礎となる LOB データを各国語キャラクタ・セットからデータベース・キャラクタ・セットに変換することによって、このファンクションを実行します。

例

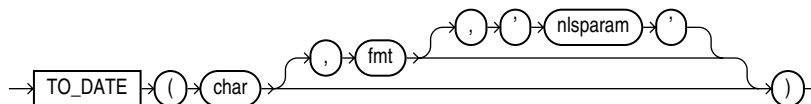
次の文は、サンプル表 pm.print_media の NCLOB データを CLOB に変換し、CLOB 列に挿入します。

```
UPDATE PRINT_MEDIA
  SET AD_FINALTEXT = TO_CLOB (AD_FLTEXTN);
```

TO_DATE

構文

to_date::=



用途

TO_DATE は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の *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* はデフォルトの日付書式である必要があります。*fmt* が「J」（ユリウス日）の場合、*char* は整数である必要があります。

注意： このファンクションは、データを別の日時データ型へは変換しません。他の日時変換の詳細は、6-175 ページの「[TO_TIMESTAMP](#)」、6-176 ページの「[TO_TIMESTAMP_TZ](#)」、6-168 ページの「[TO_DSINTERVAL](#)」および 6-178 ページの「[TO_YMINTERVAL](#)」を参照してください。

デフォルトの日付書式は、NLS_TERRITORY 初期化パラメータによって暗黙的に決まります。NLS_DATE_FORMAT パラメータによって明示的に設定することもできます。

'nlsparam' は、日付変換の TO_CHAR ファンクションの場合と同じ用途で使用されます。

引数 *char* に DATE 値を持つ TO_DATE ファンクションは使用しないでください。戻される DATE 値は、*fmt* またはデフォルトの日付書式によって、元の *char* とは異なる世紀の値を持つことがあります。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

参照： 2-64 ページの「[日付書式モデル](#)」を参照してください。

例

次の例では、文字列をタイムスタンプに変換します。

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

```
TO_DATE('
-----
15-JAN-89
```

戻された値は、NLS_TERRITORY パラメータが 'AMERICA' に設定されている場合は、デフォルトの日付書式を反映します。異なる NLS_TERRITORY の値が設定されている場合は、異なるデフォルトの日付書式となります。

```
ALTER SESSION SET NLS_TERRITORY = 'KOREAN';
```

```
SELECT TO_DATE(
    'January 15, 1989, 11:00 A.M.',
    'Month dd, YYYY, HH:MI A.M.',
```

```
'NLS_DATE_LANGUAGE = American')
FROM DUAL;

TO_DATE(
-----
89/01/15
```

TO_DSINTERVAL

構文

to_dsinterval::=



用途

TO_DSINTERVAL は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の文字列を、INTERVAL DAY TO SECOND データ型の値に変換します。

- char は、変換する文字列です。
- このファンクションで指定できる有効な nlsparam パラメータは、NLS_NUMERIC_CHARACTERS のみです。この引数は、次の書式で指定します。

```
NLS_NUMERIC_CHARACTERS = "dg"
```

d および g は、それぞれ小数点文字および桁区切りを表します。

例

次の例では、employees 表から 1990 年 1 月 1 日から 100 日以上勤務している従業員を検索します。

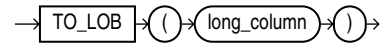
```
SELECT employee_id, last_name FROM employees
WHERE hire_date + TO_DSINTERVAL('100 10:00:00')
<= DATE '1990-01-01';
```

```
EMPLOYEE_ID LAST_NAME
-----
100 King
101 Kochhar
200 Whalen
```

TO_LOB

構文

to_lob::=



用途

TO_LOB は、*long_column* 列の LONG または LONG RAW 値を LOB 値に変換します。このファンクションは LONG または LONG RAW 列に対してのみ、および INSERT 文における副問合せの SELECT 構文のリストにおいてのみ適用できます。

このファンクションを使用する前に、LOB 列を作成して変換された LONG 値を受け取る必要があります。LONG に変換するには、CLOB 列を作成します。LONG RAW に変換するには、BLOB 列を作成します。

注意： 索引構成表を作成する場合は、CREATE TABLE ...AS SELECT 文の副問合せで、TO_LOB ファンクションを使用して LONG 列を LOB 列に変換することはできません。LONG 列を含まない索引構成表を作成し、INSERT ... AS SELECT 文で TO_LOB ファンクションを使用してください。

参照：

- LONG 列を LOB に変換する別の方法については、11-2 ページの「[ALTER TABLE](#)」の「*modify_col_properties*」を参照してください。
- INSERT 文の副問合せについては、17-52 ページの「[INSERT](#)」を参照してください。

例

サンプル表 pm.print_media は、LONG 型の列 *press_release* を持ちます。次の例では、*press_release* 列の LOB データを持つ表の一部を再作成します。

```

CREATE TABLE new_print_media (
    product_id    NUMBER(6),
    ad_id         NUMBER(6),
    press_release CLOB);

INSERT INTO new_print_media
    (SELECT p.product_id, p.ad_id, TO_LOB(p.press_release)
     FROM print_media p);
  
```

TO_MULTI_BYTE

構文

to_multi_byte::=

→ TO_MULTI_BYTE → ((char)) →

用途

TO_MULTI_BYTE は、シングルスバイト文字を、対応するマルチバイト文字に変換して *char* を戻します。*char* のデータ型は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 です。戻り値は、*char* と同じデータ型です。

char 内に同等のマルチバイト文字がないシングルスバイト文字は、シングルスバイト文字として出力されます。このファンクションは、ご使用のデータベース・キャラクタ・セットに、シングルスバイト文字およびマルチバイト文字の両方が含まれている場合にのみ有効です。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、シングルスバイト 'A' から UTF8 のマルチバイト 'A' への変換を示します。

```
SELECT dump (TO_MULTI_BYTE ( 'A' )) FROM DUAL;
```

```
DUMP (TO_MULTI_BYTE ( 'A' ))
```

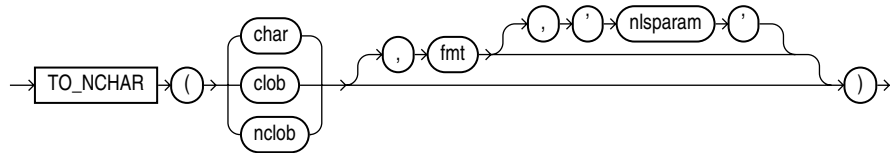
```
-----
```

```
Typ=1 Len=3: 239,188,161
```

TO_NCHAR (文字)

構文

to_nchar_char::=



用途

TO_NCHAR (文字) は、文字列、CLOB または NCLOB をデータベース・キャラクタ・セットから各国語キャラクタ・セットに変換します。このファンクションは、各国語キャラクタ・セットで USING 句を指定した TRANSLATE ... USING ファンクションと同じです。

参照： 2-46 ページの「[データ変換](#)」および 6-180 ページの「[TRANSLATE ...USING](#)」を参照してください。

例

次の例では、pm.print_media 表の NCLOB データを各国語キャラクタ・セットに変換します。

```
SELECT TO_NCHAR(ad_fltextn) FROM print_media
       WHERE product_id = 3106;
```

```
TO_NCHAR(AD_FLTEXTN)
```

```
-----
TIGER2 Tastaturen...weltweit fuehrend in Computer-Ergonomie.
```

```
TIGER2 3106 Tastatur
```

```
Product Nummer: 3106
```

```
Nur 39 EURO!
```

```
Die Tastatur KB 101/CH-DE ist eine Standard PC/AT Tastatur mit 102 Tasten. Tasta  
turbelegung: Schweizerdeutsch.
```

```
. NEU: Kommt mit ergonomischer Schaumstoffunterlage.
```

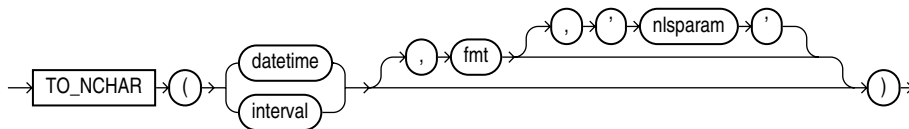
```
. Extraflache und ergonomisch-geknickte Versionen verfugbar auf Anfrage.
```

```
. Lieferbar in Elfenbein, Rot oder Schwarz.
```

TO_NCHAR (日時)

構文

to_nchar_date::=



用途

`TO_NCHAR` (日時) は、`DATE`、`TIMESTAMP`、`TIMESTAMP WITH TIME ZONE`、`TIMESTAMP WITH LOCAL TIME ZONE`、`INTERVAL MONTH TO YEAR` または `INTERVAL DAY TO SECOND` データ型の文字列をデータベース・キャラクタ・セットから各国語キャラクタ・セットに変換します。

例

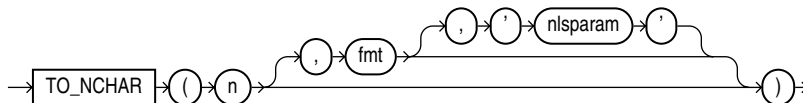
```
SELECT TO_NCHAR(order_date) FROM orders
WHERE order_status > 9;
```

```
TO_NCHAR(ORDER_DATE)
-----
14-SEP-99 08.53.40.223345 AM
13-SEP-99 09.19.00.654279 AM
27-JUN-00 08.53.32.335522 PM
26-JUN-00 09.19.43.190089 PM
06-DEC-99 01.22.34.225609 PM
```

TO_NCHAR (数値)

構文

to_nchar_number::=



用途

TO_NCHAR (数値) は、数値を NVARCHAR2 キャラクタ・セットの文字列に変換します。オプションの *fmt*、*n* に対応する '*nlsparam*' は、DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE、INTERVAL MONTH TO YEAR または INTERVAL DAY TO SECOND データ型です。

例

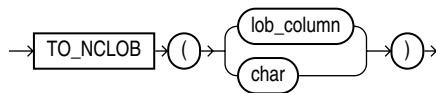
```
SELECT TO_NCHAR(customer_id) "NCHAR_Customer_ID" FROM orders
       WHERE order_status > 9;
```

```
NCHAR_Customer_ID
-----
102
103
148
149
148
```

TO_NCLOB

構文

to_nclob::=



用途

TO_NCLOB は、LOB 列またはその他の文字列の CLOB 値を NCLOB 値に変換します。 *char* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。Oracle は、LOB 列のキャラクタ・セットをデータベース・キャラクタ・セットから各国語キャラクタ・セットに変換することによって、このファンクションを実行します。

例

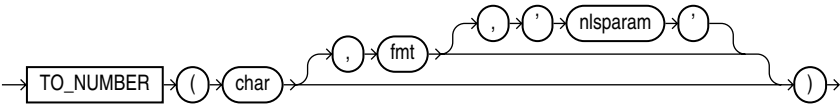
次の例では、TO_NCLOB ファンクションでデータを変換後、pm.print_media 表の NCLOB 列に文字データを挿入します。

```
INSERT INTO print_media (product_id, ad_id, ad_fltextrn)
VALUES (3502, 31001,
       TO_NCLOB('Placeholder for new product description'));
```

TO_NUMBER

構文

to_number::=



用途

TO_NUMBER は、オプションの書式モデル *fmt* によって指定された書式の数値を含む CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の値 *char* を、NUMBER データ型の値に変換します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、文字列データを数値に変換します。

```
UPDATE employees SET salary = salary +
  TO_NUMBER('100.00', '9G999D99')
WHERE last_name = 'Perkins';
```

このファンクションの '*nlsparam*' 文字列は、数値変換の TO_CHAR ファンクションの場合と同じ用途に使用されます。

参照： 6-164 ページの「[TO_CHAR \(数値\)](#)」を参照してください。

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

Amount
-----
-100
```

TO_SINGLE_BYTE

構文

to_single_byte::=



用途

TO_SINGLE_BYTE は、マルチバイト文字を、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* のデータ型は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 です。戻り値は、*char* と同じデータ型です。

char 内に同等の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。このファンクションは、ご使用のデータベース・キャラクタ・セットに、シングルバイト文字およびマルチバイト文字の両方が含まれている場合にのみ有効です。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、UTF8 のマルチバイト 'A' からシングルバイトの ASCII の 'A' への変換を示します。

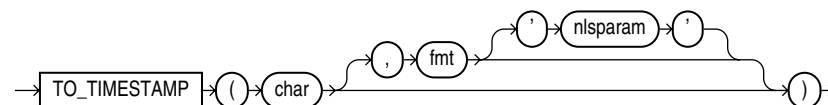
```
SELECT TO_SINGLE_BYTE( CHR(15711393)) FROM DUAL;
```

```
T
-
A
```

TO_TIMESTAMP

構文

to_timestamp::=



用途

TO_TIMESTAMP は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の *char* を、TIMESTAMP データ型の値に変換します。

オプションの *fmt* は、*char* の書式を指定します。*fmt* を指定しない場合、*char* はデフォルト書式の TIMESTAMP データ型である必要があります。オプションの '*nlsparam*' は、日付変換の TO_CHAR ファンクションの場合と同じ用途で使用されます。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできません。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、文字列をタイムスタンプに変換します。

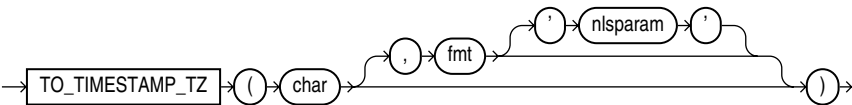
```
SELECT TO_TIMESTAMP ('1999-12-01 11:00:00', 'YYYY-MM-DD HH:MI:SS')
      FROM DUAL;

TO_TIMESTAMP('1999-12-0111:00:00','YYYY-MM-DDHH:MI:SS')
-----
01-DEC-99 11.00.00.000000000 AM
```

TO_TIMESTAMP_TZ

構文

to_timestamp_tz::=



用途

TO_TIMESTAMP_TZ は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の *char* を、TIMESTAMP WITH TIME ZONE データ型の値に変換します。

注意： このファンクションは、文字列を TIMESTAMP WITH LOCAL TIME ZONE に変換しません。変換するには、6-25 ページの「[CAST](#)」で示すように、CAST ファンクションを使用してください。

オプションの *fmt* は、*char* の書式を指定します。*fmt* を指定しない場合、*char* はデフォルト書式の `TIMESTAMP WITH TIME ZONE` データ型である必要があります。オプションの '*nlsparam*' は、日付変換の `TO_CHAR` ファンクションの場合と同じ用途で使用されます。

例

次の例では、文字列を `TIMESTAMP WITH TIME ZONE` 値に変換します。

```
SELECT TO_TIMESTAMP_TZ('1999-12-01 11:00:00 -8:00',
    'YYYY-MM-DD HH:MI:SS TZh:TZM') FROM DUAL;

TO_TIMESTAMP_TZ('1999-12-0111:00:00-08:00','YYYY-MM-DDHH:MI:SSTZh:TZM')
-----
01-DEC-99 11.00.00.0000000000 AM -08:00
```

次の例では、サンプル表 `oe.order_items` および `oe.orders` を使用して、`UNION` 操作の `NULL` 列を `TIMESTAMP WITH LOCAL TIME ZONE` としてキャストします。

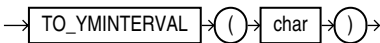
```
SELECT order_id, line_item_id,
    CAST(NULL AS TIMESTAMP WITH LOCAL TIME ZONE) order_date
    FROM order_items
UNION
SELECT order_id, to_number(null), order_date
    FROM orders;
```

ORDER_ID	LINE_ITEM_ID	ORDER_DATE
2354	1	
2354	2	
2354	3	
2354	4	
2354	5	
2354	6	
2354	7	
2354	8	
2354	9	
2354	10	
2354	11	
2354	12	
2354	13	
2354		14-JUL-00 05.18.23.234567 PM
2355	1	
2355	2	
...		

TO_YMINTERVAL

構文

to_yminterval::=



用途

TO_YMINTERVAL は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の文字列を INTERVAL YEAR TO MONTH 型に変換します。char は、変換する文字列です。

例

次の例では、サンプル表 hr.employees の各従業員の雇用された後から 1 年と 2 か月後の日付を計算します。

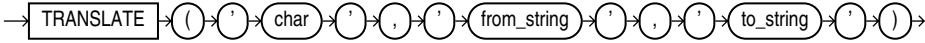
```
SELECT hire_date, hire_date + TO_YMINTERVAL('01-02') "14 months"
FROM employees;
```

```
HIRE_DATE 14 months
-----
17-JUN-87 17-AUG-88
21-SEP-89 21-NOV-90
13-JAN-93 13-MAR-94
03-JAN-90 03-MAR-91
21-MAY-91 21-JUL-92
.
.
.
```

TRANSLATE

構文

translate::=



用途

TRANSLATE は、*from_string* 内のすべての文字を *to_string* 内の対応する文字に置換して *char* を戻します。*from_string* 内に存在しない *char* 内の文字は置換されません。引数 *from_string* には、*to_string* より多い文字を指定できます。この場合、*from_string* の終わりにある余分な文字には、*to_string* 内に対応する文字がありません。これらの余分な文字が *char* 内にある場合、それらの文字は戻り値から削除されます。

戻り値から *from_string* 内の文字をすべて削除するために、*to_string* に空の文字列を使用することはできません。Oracle は空の文字列を NULL と解析し、NULL の引数がある場合、このファンクションは NULL を戻します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、ライセンス番号を変換します。「ABC...Z」のすべての文字は「X」に変換され、「012 ... 9」のすべての数字は「9」に変換されます。

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

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

次の例では、文字を削除して数値のみになったライセンス番号を戻します。

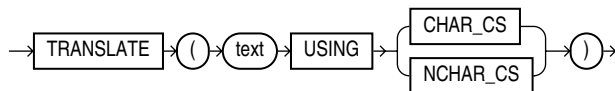
```
SELECT TRANSLATE('2KRW229',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
"Translate example"
      FROM DUAL;
```

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

TRANSLATE ...USING

構文

`translate_using::=`



用途

TRANSLATE ... USING は、`text` をデータベース・キャラクタ・セットと各国語キャラクタ・セット間の変換に指定されたキャラクタ・セットに変換します。

注意： TRANSLATE ... USING は、主に ANSI との互換性のためにサポートされているファンクションです。TO_CHAR および TO_NCHAR ファンクションを使用してデータをデータベース・キャラクタ・セットまたは各国語キャラクタ・セットに適切に変換することをお薦めします。TO_CHAR および TO_NCHAR は、文字データのみを受け入れる TRANSLATE ... USING に比べ、より多くのデータ型を引数として取ることができます。

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` が各国語キャラクタ・セットに変換されます。出力データ型は NVARCHAR2 です。

このファンクションは、Oracle の CONVERT ファンクションに似ていますが、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 ではなくこのファンクションを使用する必要があります。入力に UCS2 コードポイントまたはバックslash シュ (\) が含まれる場合は、UNISTR ファンクションを使用してください。

参照： 6-34 ページの「[CONVERT](#)」および 6-187 ページの「[UNISTR](#)」を参照してください。

例

次の文は、サンプル表 `oe.product_descriptions` のデータを使用して、TRANSLATE ... USING ファンクションの使用方法を示しています。

```
CREATE TABLE translate_tab (char_col VARCHAR2(100),
                             nchar_col NVARCHAR2(50));

INSERT INTO translate_tab
```



```
SELECT NULL, translated_name
FROM product_descriptions
WHERE product_id = 3501;

SELECT * FROM translate_tab;

CHAR_COL                                NCHAR_COL
-----
...
C per a SPNIX4.0 - Sys
C pro SPNIX4.0 - Sys
C for SPNIX4.0 - Sys
C til SPNIX4.0 - Sys
...

UPDATE translate_tab
SET char_col = TRANSLATE (nchar_col USING CHAR_CS);

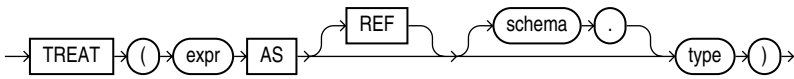
SELECT * FROM translate_tab;

CHAR_COL                                NCHAR_COL
-----
...
C per a SPNIX4.0 - Sys  C per a SPNIX4.0 - Sys
C pro SPNIX4.0 - Sys   C pro SPNIX4.0 - Sys
C for SPNIX4.0 - Sys   C for SPNIX4.0 - Sys
C til SPNIX4.0 - Sys   C til SPNIX4.0 - Sys
...
```

TREAT

構文

treat::=



用途

TREAT は式の宣言型を変更します。

このファンクションを使用する場合は、type に対する EXECUTE オブジェクト権限が必要です。

- `expr` の宣言された型が `source_type` である場合、`type` はスーパータイプまたは `source_type` のサブタイプである必要があります。`expr` の最も指定される型が `type` (または `type` のサブタイプ) である場合、TREAT は `expr` を戻します。`expr` の最も指定される型が `type` (または `type` のサブタイプ) ではない場合、TREAT は NULL を戻します。
- `expr` の宣言された型が REF `source_type` である場合、`type` はサブタイプまたは `source_type` のスーパータイプである必要があります。DEREF(`expr`) の最も指定される型が `type` (または `type` のサブタイプ) である場合、TREAT は `expr` を戻します。DEREF(`expr`) の最も指定される型が `type` (または `type` のサブタイプ) ではない場合、TREAT は NULL を戻します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、15-63 ページの「[置換可能な表および列のサンプル:](#)」で作成された表 `oe.persons` を使用します。この表は、16-22 ページの「[型の階層例:](#)」で作成された `person_t` 型に基づいています。次の例では、`persons` 表のすべての人物の給与属性を検索します。従業員ではない人物のインスタンスの値は NULL です。

```
SELECT name, TREAT(VALUE(p) AS employee_t).salary salary
FROM persons p;
```

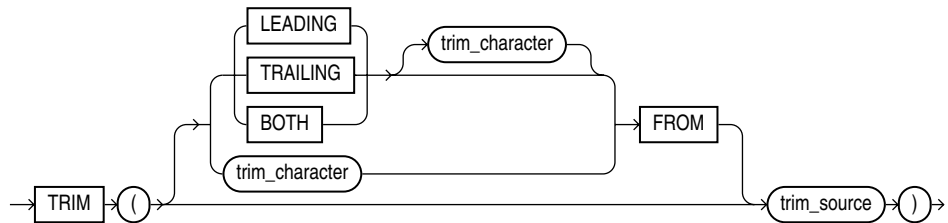
NAME	SALARY
-----	-----
Bob	
Joe	100000
Tim	1000

TREAT ファンクションを使用すると、置換可能列のサブタイプ属性に索引を作成できます。13-86 ページの「[置換可能な列の索引の例:](#)」の例を参照してください。

TRIM

構文

trim::=



用途

TRIMによって、文字列の先行または後続文字（またはその両方）を切り捨てることができます。*trim_character* または *trim_source* が文字リテラルの場合、引用符で囲む必要があります。

- LEADING を指定すると、Oracle は *trim_character* と等しい先行文字を削除します。
- TRAILING を指定すると、Oracle は *trim_character* と等しい後続文字を削除します。
- BOTH を指定するか、またはいずれも指定しない場合、Oracle は *trim_character* と等しい先行および後続文字を削除します。
- *trim_character* を指定しないと、デフォルト値は空白になります。
- *trim_source* のみを指定すると、Oracle は先行および後続空白を削除します。
- このファンクションは、値を VARCHAR2 データ型で戻します。値の最大長は、*trim_source* の長さです。
- *trim_source* または *trim_character* のいずれかが NULL の場合、TRIM は NULL を戻します。

trim_character および *trim_source* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻される文字列は、VARCHAR2 データ型であり、*trim_source* と同じキャラクタ・セットです。

例

この例では、先行および後続 0（ゼロ）を数値から切り捨てます。

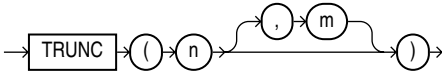
```
SELECT TRIM (0 FROM 0009872348900) "TRIM Example"
FROM DUAL;

TRIM Example
-----
          98723489
```

TRUNC（数値）

構文

trunc_number::=



用途

TRUNC（数値）ファンクションは、*n* を小数第 *m* 位までに切り捨てた値を戻します。*m* を指定しない場合、*n* の小数点以下を切り捨てます。*m* が負の場合は、小数点の左 *m* 桁を切り捨て、0（ゼロ）にします。

例

次の例では、数値を切り捨てます。

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

Truncate
-----
        15.7

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

Truncate
-----
         10
```

TRUNC（日付）

構文

trunc_date::=



用途

TRUNC（日付）ファンクションは、時刻部分を書式モデル *fmt* で指定された単位まで切り捨てた *date* を戻します。*fmt* を省略すると、*date* は最も近い日に切り捨てられます。

参照： *fmt* で使用できる書式モデルについては、6-210 ページの「[ROUND および TRUNC 日付ファンクション](#)」を参照してください。

例

次の例では、日付を切り捨てます。

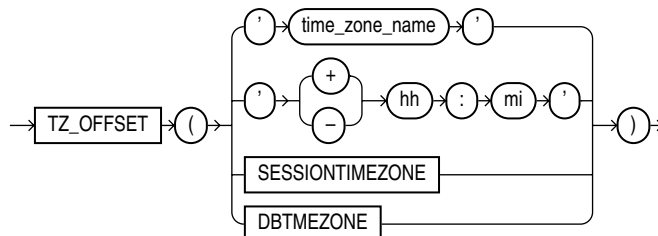
```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR')
       "New Year" FROM DUAL;
```

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

TZ_OFFSET

構文

tz_offset::=



用途

TZ_OFFSET は、文が実行された日付に基づいて入力された値に対応するタイムゾーン・オフセットを戻します。有効なタイムゾーン名、UTC からのタイムゾーン・オフセット（それ自体を戻します）、またはキーワード SESSIONTIMEZONE または DBTIMEZONE を入力できます。有効な値を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューの TZNAME 列を問い合わせます。

参照： 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

例

次の例では、UTC からの米国 / 東部タイムゾーンのタイムゾーン・オフセットを戻します。

```
SELECT TZ_OFFSET('US/Eastern') FROM DUAL;
```

```
TZ_OFFS
-----
-04:00
```

UID

構文

uid::=



用途

UID は、セッション・ユーザー（ログインしているユーザー）を一意に識別する整数を戻します。

例

次の例では、現行のユーザーの UID を戻します。

```
SELECT UID FROM DUAL;
```

UNISTR

構文

unistr::=



用途

UNISTR は、文字列を引数として指定すると、その文字列を各国語キャラクタ・セットで戻します。データベースの各国語キャラクタ・セットは、AL16UTF16 または UTF8 です。UNISTR では、文字列への文字の **Unicode** エンコーディング値の指定を可能にすることによって、**Unicode** 文字列リテラルがサポートされます。これは、NCHAR 列にデータを挿入する場合などに有効です。

Unicode エンコーディング値は、\xxxx という形式です。xxxx は、文字の UCS-2 エンコーディング形式での 16 進数値です。バックスラッシュ自体を文字列に含めるには、バックスラッシュをもう 1 つ追加します (\\)。

移植性およびデータ保護のために、UNISTR 文字列の引数には ASCII 文字および **Unicode** エンコーディング値のみを指定することをお勧めします。

参照： **Unicode** および各国語キャラクタ・セットの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

例

次の例では、ASCII 文字および **Unicode** エンコーディング値を UNISTR ファンクションに渡した後、その UNISTR ファンクションによって各国語キャラクタ・セットで文字列が戻されます。

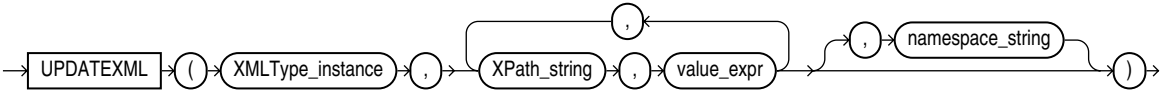
```
SELECT UNISTR('abc\00e5\00f1\00f6') FROM DUAL;
```

```
UNISTR
-----
abcåñö
```

UPDATEXML

構文

updatexml::=



用途

UPDATEXML は、引数として XMLType インスタンスおよび XPath 値の組を取り、更新された値を持つ XMLType インスタンスを戻します。XPath_string が XML 要素の場合、対応する value_expr は XMLType インスタンスです。XPath_string が属性またはテキスト・ノードの場合、value_expr は任意のスカラー・データ型となります。各 XPath_string およびこれに対応する value_expr のターゲットのデータ型は一致する必要があります。オプションの namespace_string は、接頭辞のデフォルト・マッピングまたはネームスペース・マッピング（Oracle が XPath 式を評価する場合に使用）を指定する VARCHAR2 値に解決される必要があります。

XML 要素を更新して NULL にした場合、属性および子要素が削除されるため、要素は空になります。要素のテキスト・ノードを更新して NULL にした場合、その要素のテキスト値が削除されるため、要素自体は残りますが空になります。

多くの場合、このファンクションはメモリー内の XML 文書を実体化して、その値を更新します。ただし、UPDATEXML は、直接オブジェクト・リレーショナル列の値を更新するように、その列での UPDATE 文に対して最適化されます。この最適化には、次の条件があります。

- XMLType_instance は、UPDATE ... SET 句の列と同じである必要があります。
- XPath_string は、スカラー・コンテンツとなる必要があります。

例

次の例では、サンプル・スキーマ OE（XMLType 型の warehouse_spec 列を持つ）にある San Francisco 倉庫のドック数を 4 に更新します。

```
SELECT warehouse_name,
       EXTRACT(warehouse_spec, '/Warehouse/Docks')
       "Number of Docks"
FROM warehouses
WHERE warehouse_name = 'San Francisco';
```

WAREHOUSE_NAME	Number of Docks
San Francisco	<Docks>1</Docks>


```
UPDATE warehouses SET warehouse_spec =
    UPDATEXML(warehouse_spec,
        '/Warehouse/Docks/text()',4)
    WHERE warehouse_name = 'San Francisco';
```

1 row updated.

```
SELECT warehouse_name,
    EXTRACT(warehouse_spec, '/Warehouse/Docks')
    "Number of Docks"
FROM warehouses
    WHERE warehouse_name = 'San Francisco';
```

WAREHOUSE_NAME	Number of Docks
San Francisco	<Docks>4</Docks>

UPPER

構文

upper::=



用途

UPPER は、すべての文字を大文字にして *char* を戻します。*char* は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB または NCLOB データ型です。戻り値は、*char* と同じデータ型です。

例

次の例では、文字列を大文字で戻します。

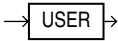
```
SELECT UPPER('Large') "Uppercase"
FROM DUAL;
```

```
Upper
-----
LARGE
```

USER

構文

user::=



用途

USER は、セッション・ユーザー（ログインしているユーザー）の名前を VARCHAR2 データ型で戻します。Oracle は、空白埋め比較セマンティクスでこのファンクションの値を比較します。

分散 SQL 文では、UID ファンクションおよび USER ファンクションは、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ファンクションは使用できません。

例

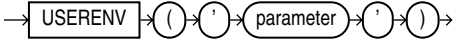
次の例では、現行のユーザーおよびユーザーの UID を戻します。

```
SELECT USER, UID FROM DUAL;
```

USERENV

構文

userenv::=



用途

注意： USERENV は、下位互換用に保持されるレガシー・ファンクションです。現行の機能に対して組込み USERENV ネームスペースとともに SYS_CONTEXT ファンクションを使用することをお勧めします。詳細は、6-148 ページの「[SYS_CONTEXT](#)」を参照してください。

USERENV は現行のセッションについての情報を戻します。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、またはセッションで現在使用されている言語固有の文字を判断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。表 6-3 に、parameter 引数の値を示します。

SESSIONID、ENTRYID および COMMITSCN パラメータを使用するコール（NUMBER を戻す）以外の USERENV へのすべてのコールは、VARCHAR2 データを戻します。

表 6-3 USERENV ファンクションのパラメータ

パラメータ	戻り値
CLIENT_INFO	DBMS_APPLICATION_INFO パッケージを使用するアプリケーションが格納できる 64 バイトまでのユーザー・セッション情報を戻します。 注意： 商業用のアプリケーションによっては、このコンテキスト値を使用する可能性があります。このコンテキスト領域の使用に対する制限については、これらのアプリケーションのドキュメントを確認してください。 参照： ■ アプリケーション・コンテキストの詳細は、『Oracle9i データベース概要』を参照してください。 ■ 13-11 ページの「CREATE CONTEXT」および 6-148 ページの「SYS_CONTEXT」を参照してください。
ENTRYID	使用可能な監査エントリ識別子を戻します。この属性を分散 SQL 文で使用することはできません。USERENV でこのキーワードを使用するには、AUDIT_TRAIL 初期化パラメータに TRUE を設定する必要があります。
ISDBA	ISDBA は、オペレーティング・システムまたはパスワード・ファイルによって、ユーザーが DBA 権限を持っていると認証された場合、'TRUE' を戻します。
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。
LANGUAGE	セッションで現在使用している言語（language）および地域（territory）を、データベース・キャラクタ・セット（character set）も含めた次の書式で戻します。 language_territory.characterset
SESSIONID	監査セッション識別子を戻します。この属性を分散 SQL 文で使用することはできません。
TERMINAL	現行の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、この属性はローカル・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 に対してのみこのオプションを使用でき、リモートの INSERT、UPDATE または DELETE には使用できません。

例

次の例では、現行のセッションの LANGUAGE パラメータを戻します。

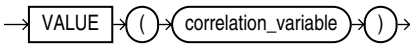
```
SELECT USERENV('LANGUAGE') "Language" FROM DUAL;

Language
-----
AMERICAN_AMERICA.WE8DEC
```

VALUE

構文

value::=



用途

VALUE は、引数としてオブジェクト表の行に関連付けられた相関変数（表別名）を取り、オブジェクト表に格納されたオブジェクト・インスタンスを戻します。オブジェクト・インスタンスの型は、オブジェクト表と同じ型です。

例

次の例では、15-63 ページの「置換可能な表および列のサンプル:」で作成するサンプル表 oe.persons を使用します。

```
SELECT VALUE(p) FROM persons p;

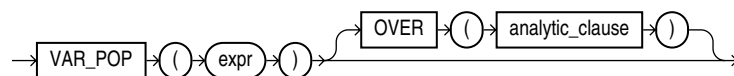
VALUE(P) (NAME, SSN)
-----
PERSON_T('Bob', 1234)
EMPLOYEE_T('Joe', 32456, 12, 100000)
PART_TIME_EMP_T('Tim', 5678, 13, 1000, 20)
```

参照： VALUE ファンクションでの IS OF type の使用方法については、5-19 ページの「IS OF type 条件」を参照してください。

VAR_POP

構文

var_pop::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

VAR_POP は、数値の集合にある NULL を削除した後、この集合の母集団分散を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr は数値式であり、このファンクションは NUMBER 型の値を戻します。ファンクションが空の集合に適用されると、NULL を戻します。このファンクションは、次の計算を行います。

$$(\text{SUM}(\text{expr}^2) - \text{SUM}(\text{expr})^2 / \text{COUNT}(\text{expr})) / \text{COUNT}(\text{expr})$$

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- expr の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、employees 表にある給与の人口分散を計算します。

```
SELECT VAR_POP(salary) FROM employees;
```

```
VAR_POP(SALARY)
-----
      15140307.5
```

分析の例

次の例では、1998 年における月ごとの売上について、累積の母集団および標本分散を計算します。

```
SELECT t.calendar_month_desc,
       VAR_POP(SUM(s.amount_sold))
         OVER (ORDER BY t.calendar_month_desc) "Var_Pop",
       VAR_SAMP(SUM(s.amount_sold))
```

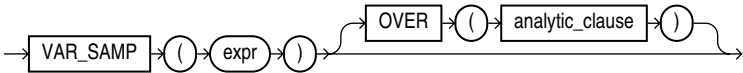
```
OVER (ORDER BY t.calendar_month_desc) "Var_Samp"
FROM sales s, times t
WHERE s.time_id = t.time_id AND t.calendar_year = 1998
GROUP BY t.calendar_month_desc;
```

CALENDAR	Var_Pop	Var_Samp
-----	-----	-----
1998-01	0	
1998-02	6.1321E+11	1.2264E+12
1998-03	4.7058E+11	7.0587E+11
1998-04	4.6929E+11	6.2572E+11
1998-05	1.5524E+12	1.9405E+12
1998-06	2.3711E+12	2.8453E+12
1998-07	3.7464E+12	4.3708E+12
1998-08	3.7852E+12	4.3260E+12
1998-09	3.5753E+12	4.0222E+12
1998-10	3.4343E+12	3.8159E+12
1998-11	3.4245E+12	3.7669E+12
1998-12	4.8937E+12	5.3386E+12

VAR_SAMP

構文

var_samp::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

VAR_SAMP は、数値の集合にある NULL を削除した後、この集合の標本分散を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr は数値式であり、このファンクションは NUMBER 型の値を戻します。このファンクションが空の集合に適用されると、NULL を戻します。このファンクションは、次の計算を行います。

$$\left(\text{SUM}(\text{expr}^2) - \text{SUM}(\text{expr})^2 / \text{COUNT}(\text{expr}) \right) / \left(\text{COUNT}(\text{expr}) - 1 \right)$$

このファンクションは、1 つの要素の集合を入力すると VARIANCE は 0 を返し、VAR_SAMP は NULL を戻すという点を除いては、VARIANCE に似ています。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- `expr` の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 `employees` にある給与の標本分散を計算します。

```
SELECT VAR_SAMP(salary) FROM employees;
```

```
VAR_SAMP(SALARY)
-----
15283140.5
```

分析の例

次の例では、1998 年における月ごとの売上について、累積の母集団および標本分散を計算します。

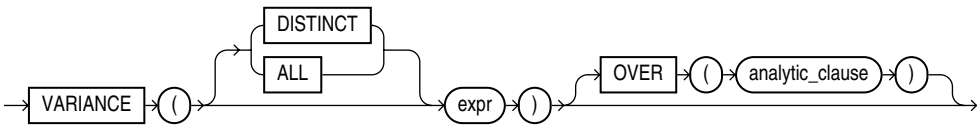
```
SELECT t.calendar_month_desc,
       VAR_POP(SUM(s.amount_sold))
         OVER (ORDER BY t.calendar_month_desc) "Var_Pop",
       VAR_SAMP(SUM(s.amount_sold))
         OVER (ORDER BY t.calendar_month_desc) "Var_Samp"
FROM sales s, times t
WHERE s.time_id = t.time_id AND t.calendar_year = 1998
GROUP BY t.calendar_month_desc;
```

CALENDAR	Var_Pop	Var_Samp
-----	-----	-----
1998-01	0	
1998-02	6.1321E+11	1.2264E+12
1998-03	4.7058E+11	7.0587E+11
1998-04	4.6929E+11	6.2572E+11
1998-05	1.5524E+12	1.9405E+12
1998-06	2.3711E+12	2.8453E+12
1998-07	3.7464E+12	4.3708E+12
1998-08	3.7852E+12	4.3260E+12
1998-09	3.5753E+12	4.0222E+12
1998-10	3.4343E+12	3.8159E+12
1998-11	3.4245E+12	3.7669E+12
1998-12	4.8937E+12	5.3386E+12

VARIANCE

構文

variance::=



参照： 構文、セマンティクスおよび制限事項の詳細は、6-9 ページの「[分析ファンクション](#)」を参照してください。

用途

VARIANCE は *expr* の分散を戻します。これは、集計ファンクションまたは分析ファンクションとして使用できます。

expr の分散の計算結果は、次のようになります。

- *expr* の行数が 1 の場合は 0 (ゼロ)
- *expr* の行数が 1 を超える場合は VAR_SAMP

DISTINCT を指定する場合は、*analytic_clause* の *query_partition_clause* のみ指定できます。*order_by_clause* および *windowing_clause* は指定できません。

参照：

- 6-7 ページの「[集計ファンクション](#)」を参照してください。
- *expr* の書式の詳細は、4-2 ページの「[SQL 式](#)」を参照してください。

集計の例

次の例では、サンプル表 `employees` にあるすべての給与の合計を計算します。

```
SELECT VARIANCE(salary) "Variance"
FROM employees;

Variance
-----
15283140.5
```


分析の例

次の例では、雇用日で順序付けられた部門 30 の給与の値の累積分散を戻します。

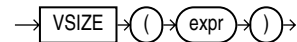
```
SELECT last_name, salary, VARIANCE(salary)
       OVER (ORDER BY hire_date) "Variance"
FROM employees
WHERE department_id = 30;
```

LAST_NAME	SALARY	Variance
Raphaely	11000	0
Khoo	3100	31205000
Tobias	2800	21623333.3
Baida	2900	16283333.3
Himuro	2600	13317000
Colmenares	2500	11307000

VSIZE

構文

vsize::=



用途

VSIZE は、*expr* の内部表現でのバイト数を戻します。*expr* が NULL の場合は NULL を戻します。

注意： このファンクションは、CLOB データを直接的にサポートしていません。ただし、暗黙的なデータ変換を使用して CLOB を引数として渡すことはできます。詳細は、2-42 ページの「[データ型の比較規則](#)」を参照してください。

例

次の例では、部門 10 の従業員の last_name のバイト数を戻します。

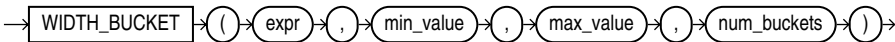
```
SELECT last_name, VSIZE (last_name) "BYTES"
FROM employees
WHERE department_id = 10;
```

LAST_NAME	BYTES
-----	-----
Whalen	6

WIDTH_BUCKET

構文

width_bucket::=



用途

WIDTH_BUCKET を使用すると、ヒストグラムの幅が同じサイズに分割された等幅ヒストグラムを作成できます。このファンクションと、等高ヒストグラムを作成する NTILE を比較してください。各バケットが実際の数値線幅の「closed-open」間隔であることが理想です。たとえば、10 が間隔に含まれ 20 が排除されることを示すために、バケットは 10.00 ～ 19.999... のスコアに割り当てられます。これは、[10, 20] と表すこともできます。

指定された式に対して、WIDTH_BUCKET は、この式の値が評価された後に該当するバケット数を戻します。

- `expr` は、ヒストグラムが作成される式です。この式は、数値または日時値に評価される必要があります。`expr` が NULL と評価された場合、式は NULL を戻します。
- `min_value` および `max_value` は、`expr` の許容域のエンド・ポイントに解決される式です。これらの式は両方とも数値または日付値に評価される必要があります、いずれも NULL に評価されません。
- `num_buckets` は、バケット数を示す定数に解決される式です。この式は、正の整数に評価される必要があります。

Oracle は、必要に応じて、0（ゼロ）の下位バケットおよび `num_buckets+1` の下位バケットを作成します。これらのバケットは、`min_value` 未満および `max_value` より大きい値を処理し、エンド・ポイントの妥当性チェックに有効です。

例

次の例では、サンプル表 `oe.customers` のスイスの顧客の `credit_limit` 列に 10 バケットのヒストグラムを作成し、各顧客のバケット数（「クレジット・グループ」）を戻します。最大値を超えるクレジット利用限度額を持つ顧客は、下位バケット 11 に割り当てられます。

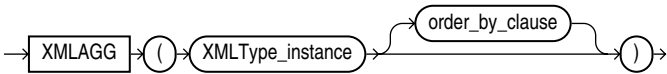
```
SELECT customer_id, cust_last_name, credit_limit,
       WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit Group"
FROM customers WHERE nls_territory = 'SWITZERLAND'
ORDER BY "Credit Group";
```

CUSTOMER_ID	CUST_LAST_NAME	CREDIT_LIMIT	Credit Group
825	Dreyfuss	500	1
826	Barkin	500	1
853	Palin	400	1
827	Siegel	500	1
843	Oates	700	2
844	Julius	700	2
835	Eastwood	1200	3
840	Elliott	1400	3
842	Stern	1400	3
841	Boyer	1400	3
837	Stanton	1200	3
836	Berenger	1200	3
848	Olmos	1800	4
849	Kaurusmdki	1800	4
828	Minnelli	2300	5
829	Hunter	2300	5
852	Tanner	2300	5
851	Brown	2300	5
850	Finney	2300	5
830	Dutt	3500	7
831	Bel Geddes	3500	7
832	Spacek	3500	7
838	Nicholson	3500	7
839	Johnson	3500	7
833	Moranis	3500	7
834	Idle	3500	7
845	Fawcett	5000	11
846	Brando	5000	11
847	Streep	5000	11

XMLAGG

構文

xmlagg::=



用途

XMLAGG は集計ファンクションです。XML フラグメントのコレクションを取り、集計された XML 文書を戻します。NULL を戻す引数は結果から排除されます。

XMLAGG は、ノードのコレクションを戻す点を除いて、SYS_XMLAGG に似ていますが、XMLFormat オブジェクトを使用した書式設定は受け入れません。また、XMLAGG は、SYS_XMLAGG とは異なり、出力を要素タグで囲みません。

注意： この *order_by_clause* にかぎり、他の使用方法とは異なり、数値リテラルは列の位置として解釈されず、単に数値リテラルとして解釈されます。

参照： 6-203 ページの「[XMLELEMENT](#)」および 6-156 ページの「[SYS_XMLAGG](#)」を参照してください。

例

次の例では、従業員のジョブ ID と姓を要素の内容とする Employee 要素を含む Department 要素を生成します。

```
SELECT XMLELEMENT("Department",
  XMLAGG(XMLELEMENT("Employee",
    e.job_id||' '||e.last_name)
  ORDER BY last_name))
  as "Dept_list"
FROM employees e
WHERE e.department_id = 30;
```

```
Dept_list
-----
<Department>
  <Employee>PU_CLERK Baida</Employee>
  <Employee>PU_CLERK Colmenares</Employee>
  <Employee>PU_CLERK Himuro</Employee>
  <Employee>PU_CLERK Khoo</Employee>
```

```

    <Employee>PU_MAN Raphaely</Employee>
    <Employee>PU_CLERK Tobias</Employee>
  </Department>

```

XMLAGG が行を集計するため、結果は単一行となります。GROUP BY 句を使用して、戻された行の集合を複数のグループにまとめることができます。

```

SELECT XMLELEMENT("Department",
    XMLAGG(XMLELEMENT("Employee", e.job_id||' '||e.last_name)))
  AS "Dept_list"
  FROM employees e
  GROUP BY e.department_id;

```

Dept_list

```

-----
<Department>
  <Employee>AD_ASST Whalen</Employee>
</Department>

<Department>
  <Employee>MK_MAN Hartstein</Employee>
  <Employee>MK_REP Fay</Employee>
</Department>

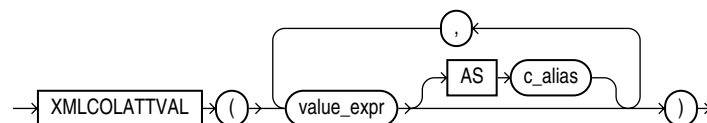
<Department>
  <Employee>PU_MAN Raphaely</Employee>
  <Employee>PU_CLERK Khoo</Employee>
  <Employee>PU_CLERK Tobias</Employee>
  <Employee>PU_CLERK Baida</Employee>
  <Employee>PU_CLERK Colmenares</Employee>
  <Employee>PU_CLERK Himuro</Employee>
</Department>
...

```

XMLCOLATTVAL

構文

xmlcolattval::=



用途

XMLCOLATTVAL は、XML フラグメントを作成し、各 XML フラグメントの名前が「column」、属性が「name」となるように、結果としてできた XML を展開します。AS *c_alias* 句を使用して、name 属性の値を列名以外の値に変更できます。

value_expr の値を指定する必要があります。 *value_expr* が NULL の場合、要素は戻されません。

XMLColAttVal の制限事項： *value_expr* にオブジェクト型の列を指定することはできません。

例

次の例では、従業員のサブセットに対して Emp 要素を作成します。このとき、Emp の内容として、employee_id、last_name および salary の各要素がネストされます。ネストされた各要素には column という名前が付き、属性値として列名を持つ name 属性が与えられます。

```
SELECT XMLELEMENT("Emp",
  XMLCOLATTVAL(e.employee_id, e.last_name, e.salary)) "Emp Element"
FROM employees e
WHERE employee_id = 204;
```

Emp Element

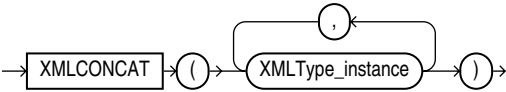
```
<Emp>
  <column name="EMPLOYEE_ID">204</column>
  <column name="LAST_NAME">Baer</column>
  <column name="SALARY">10000</column>
</Emp>
```

参照： これらの 2 つのファンクションの出力の比較については、6-206 ページの「XMLFOREST」を参照してください。

XMLCONCAT

構文

XMLConcat::=



用途

XMLCONCAT は、入力として一連の XMLType インスタンスを取り、各行について一連の要素を連結し、連結した結果を戻します。XMLCONCAT は XMLSEQUENCE の逆の処理をします。

NULL 式は結果から排除されます。値のすべての式が NULL の場合、このファンクションは NULL を戻します。

参照： 6-207 ページの「XMLSEQUENCE」を参照してください。

例

次の例では、従業員のサブセットの名前と姓に対して XML 要素を作成し、作成した要素を連結して戻します。

```
SELECT XMLCONCAT(XMLELEMENT("First", e.first_name),
  XMLELEMENT("Last", e.last_name)) AS "Result"
FROM employees e
WHERE e.employee_id > 202;
```

Result

```
-----
<First>Susan</First>
<Last>Mavris</Last>

<First>Hermann</First>
<Last>Baer</Last>

<First>Shelley</First>
<Last>Higgins</Last>

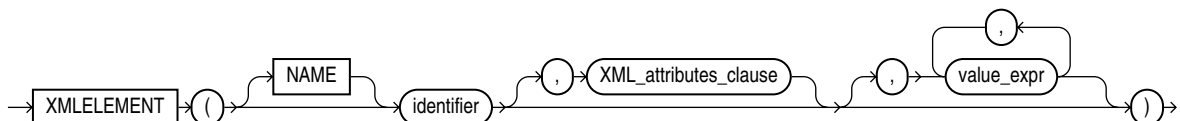
<First>William</First>
<Last>Gietz</Last>
```

4 rows selected.

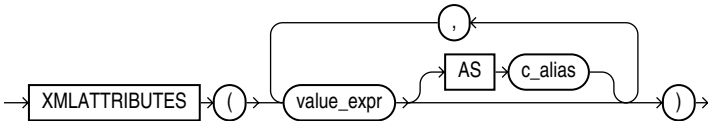
XMLELEMENT

構文

xmlelement::=



xml_attributes_clause::=



用途

XMLELEMENT は、*identifier* に対する要素名、要素に対する属性のオプションのコレクションおよびその要素の内容を構成する引数を取ります。このファンクションは `XMLType` 型のインスタンスを戻します。XMLELEMENT は、戻された XML に属性を格納できる点を除いて `SYS_XMLGEN` に似ていますが、`XMLFormat` オブジェクトを使用した書式設定は受け入れません。

次の項に示すとおり、通常は、XMLELEMENT ファンクションはネストされており、ネスト構造の XML 文書を作成します。

Oracle が囲みタグとして使用する *identifier* の値を指定する必要があります。この識別子は、列名または列の参照である必要はありません。式または NULL は指定できません。

`XML_attributes_clause` では、`value_expr` が NULL の場合、その値の式に対する属性は作成されません。`value_expr` の型に、オブジェクト型またはコレクションは指定できません。

要素の内容を構成するオブジェクトは、XMLATTRIBUTES キーワードの後に指定します。

- `value_expr` がスカラー式である場合、AS 句は省略できます。この場合、Oracle は列名を要素名として使用します。
- `value_expr` がオブジェクト型またはコレクションである場合、AS 句は必須です。この場合、Oracle は指定された `c_alias` を囲みタグとして使用します。
- `value_expr` が NULL の場合、その値の式に対する要素は作成されません。

参照： 6-157 ページの「[SYS_XMLGEN](#)」を参照してください。

例

次の例では、一連の従業員について、従業員の名前と雇用日を指定する、ネストされた要素を持つ Emp 要素を作成します。

```
SELECT XMLELEMENT('Emp', XMLELEMENT('Name',
    e.job_id||' '||e.last_name),
    XMLELEMENT('Hiredate', e.hire_date)) as "Result"
FROM employees e WHERE employee_id > 200;
```

Result

```
-----
<Emp>
```



```
<Name>MK_MAN Hartstein</Name>
<Hiredate>17-FEB-96</Hiredate>
</Emp>

<Emp>
  <Name>MK_REP Fay</Name>
  <Hiredate>17-AUG-97</Hiredate>
</Emp>

<Emp>
  <Name>HR_REP Mavris</Name>
  <Hiredate>07-JUN-94</Hiredate>
</Emp>

<Emp>
  <Name>PR_REP Baer</Name>
  <Hiredate>07-JUN-94</Hiredate>
</Emp>

<Emp>
  <Name>AC_MGR Higgins</Name>
  <Hiredate>07-JUN-94</Hiredate>
</Emp>

<Emp>
  <Name>AC_ACCOUNT Gietz</Name>
  <Hiredate>07-JUN-94</Hiredate>
</Emp>
```

6 rows selected.

次の例では、XMLELEMENT ファンクションに `XML_attributes_clause` を使用して、トップレベル要素に対する属性値を持つ、ネストされた XML 要素を作成します。

```
SELECT XMLELEMENT("Emp",
  XMLATTRIBUTES(e.employee_id AS "ID", e.last_name),
  XMLELEMENT("Dept", e.department_id),
  XMLELEMENT("Salary", e.salary)) AS "Emp Element"
FROM employees e
WHERE e.employee_id = 206;
```

Emp Element

```
-----
<Emp ID="206" LAST_NAME="Gietz">
  <Dept>110</Dept>
  <Salary>8300</Salary>
</Emp>
```

`last_name` には *AS identifier* 句が指定されていません。その結果、戻された XML は、デフォルトで列名 `last_name` を使用します。

最後に、次の例では、*XML_attributes_clause* に副問合せを使用して、別の表から要素の属性に情報を取り出します。

```
SELECT XMLELEMENT("Emp", XMLATTRIBUTES(e.employee_id, e.last_name),
  XMLELEMENT("Dept", XMLATTRIBUTES(e.department_id,
    (SELECT d.department_name FROM departments d
     WHERE d.department_id = e.department_id) as "Dept_name")),
  XMLELEMENT("salary", e.salary),
  XMLELEMENT("Hiredate", e.hire_date)) AS "Emp Element"
FROM employees e
WHERE employee_id = 205;
```

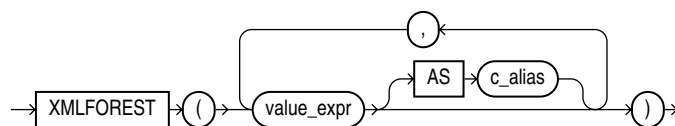
Emp Element

```
<Emp EMPLOYEE_ID="205" LAST_NAME="Higgins">
  <Dept DEPARTMENT_ID="110" Dept_name="Accounting"/>
  <salary>12000</salary>
  <Hiredate>07-JUN-94</Hiredate>
</Emp>
```

XMLFOREST

構文

xmlforest::=



用途

XMLFOREST は、各引数パラメータを XML に変換し、変換された引数を連結した XML フラグメントを戻します。

- `value_expr` がスカラー式である場合、*AS* 句は省略できます。この場合、Oracle は列名を要素名として使用します。
- `value_expr` がオブジェクト型またはコレクションである場合、*AS* 句は必須です。この場合、Oracle は指定された `c_alias` を囲みタグとして使用します。
- `value_expr` が NULL の場合、その `value_expr` に対する要素は作成されません。

例

次の例では、従業員のサブセットに対して Emp 要素を作成します。このとき、Emp の内容として、employee_id、last_name および salary の各要素がネストされます。

```
SELECT XMLELEMENT("Emp",
  XMLFOREST(e.employee_id, e.last_name, e.salary))
  "Emp Element"
FROM employees e WHERE employee_id = 204;
```

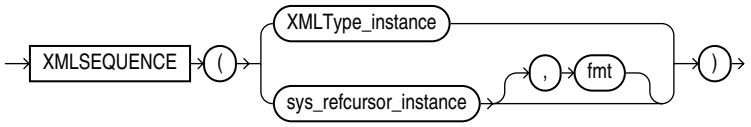
```
Emp Element
-----
<Emp>
  <EMPLOYEE_ID>204</EMPLOYEE_ID>
  <LAST_NAME>Baer</LAST_NAME>
  <SALARY>10000</SALARY>
</Emp>
```

参照： これらの 2 つのファンクションの出力の比較については、6-201 ページの「XMLCOLATTVAL」を参照してください。

XMLSEQUENCE

構文

xmlsequence::=



用途

XMLSEQUENCE には 2 つの書式があります。

- 1 つ目の書式は、入力として XMLType インスタンスを取り、XMLType にあるトップレベルのノードの VARRAY を戻します。
- 2 つ目の書式は、入力として REFCURSOR インスタンスおよびオプションの XMLFormat オブジェクトのインスタンスを取り、カーソルの各行に対して、XMLSequence 型として XML 文書を戻します。

XMLSEQUENCE は XMLType のコレクションを戻すため、このファンクションを TABLE 句で使用して、コレクション値をネスト解除することで複数行にし、SQL 問合せでの処理をさらに進めることができます。

参照： このファンクションの詳細は、『Oracle9i XML API リファレンス-XDK および Oracle XML DB』を参照してください。

例

次の例では、XMLSEQUENCE が複数の要素を持つ XML 文書を VARRAY 型の単一要素ドキュメントに分割する方法を示しています。この例では、TABLE キーワードが、コレクションを副問合せの FROM 句で利用できる表の値とみなすように、Oracle に指示しています。

```
SELECT EXTRACT(warehouse_spec, '/Warehouse') as "Warehouse"
      FROM warehouses WHERE warehouse_name = 'San Francisco';
```

Warehouse

```
-----
<Warehouse>
  <Building>Rented</Building>
  <Area>50000</Area>
  <Docks>1</Docks>
  <DockType>Side load</DockType>
  <WaterAccess>Y</WaterAccess>
  <RailAccess>N</RailAccess>
  <Parking>Lot</Parking>
  <VClearance>12 ft</VClearance>
</Warehouse>
```

1 row selected.

```
SELECT VALUE(p)
      FROM warehouses w,
      TABLE(XMLSEQUENCE(EXTRACT(warehouse_spec, '/Warehouse/*')) p
      WHERE w.warehouse_name = 'San Francisco';
```

VALUE(P)

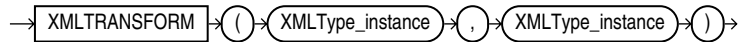
```
-----
<Building>Rented</Building>
<Area>50000</Area>
<Docks>1</Docks>
<DockType>Side load</DockType>
<WaterAccess>Y</WaterAccess>
<RailAccess>N</RailAccess>
<Parking>Lot</Parking>
<VClearance>12 ft</VClearance>
```

8 rows selected.

XMLTRANSFORM

構文

xmltransform::=



用途

XMLTRANSFORM は、引数として XMLType インスタンスおよび XSL スタイルシート（それ自体が XMLType インスタンス）を取ります。このファンクションは、スタイルシートをインスタンスに適用して、XMLType を戻します。

このファンクションは、データをデータベースから取得するように、スタイルシートに従ってデータを編成する場合に有効です。

参照： このファンクションの詳細は、『Oracle9i XML API リファレンス -XDK および Oracle XML DB』を参照してください。

例

XMLTRANSFORM ファンクションを使用するには、XSL スタイルシートが必要です。次に、ノード内の要素をアルファベット順に並べる単純なスタイルシートの例を示します。

```

CREATE TABLE xsl_tab (col1 XMLTYPE);

INSERT INTO xsl_tab VALUES (
XMLTYPE.createxml (
'<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output encoding="utf-8"/>
  <!-- alphabetizes an xml tree -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates select="*|text()">
        <xsl:sort select="name(.)" data-type="text" order="ascending"/>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space(.)"/>
  </xsl:template>
</xsl:stylesheet> ');

1 row created.

```

次の例では、XSL スタイルシート `xsl_tab` を使用して、サンプル表 `oe.warehouses` にある `warehouse_spec` の要素をアルファベット順に並べます。

```
SELECT XMLTRANSFORM(w.warehouse_spec, x.col1).GetClobVal()
FROM warehouses w, xsl_tab x
WHERE w.warehouse_name = 'San Francisco';

XMLTRANSFORM(W.WAREHOUSE_SPEC,X.COL1).GETCLOBVAL()
-----
<Warehouse>
  <Area>50000</Area>
  <Building>Rented</Building>
  <DockType>Side load</DockType>
  <Docks>1</Docks>
  <Parking>Lot</Parking>
  <RailAccess>N</RailAccess>
  <VClearance>12 ft</VClearance>
  <WaterAccess>Y</WaterAccess>
</Warehouse>
```

ROUND および TRUNC 日付ファンクション

表 6-4 に、ROUND および TRUNC 日付ファンクション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。デフォルトのモデル DD では、午前 0 時（真夜中）を基準に丸めおよび切捨てを行い、日付を戻します。

表 6-4 ROUND および TRUNC 日付ファンクションの日付書式モデル

書式モデル	丸め単位または切捨て単位
CC SCC	4 桁の年号の上 2 桁より 1 大きい数
SYYY YYYY YEAR SYEAR YYY YY Y	年（7 月 1 日に切上げ）
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 6-4 ROUND および TRUNC 日付ファンクションの日付書式モデル（続き）

書式モデル	丸め単位または切捨て単位
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、DY および D によって使用される週の開始日は、NLS_TERRITORY 初期化パラメータによって暗黙的に指定されています。

参照： このパラメータの詳細は、『Oracle9i データベース・リファレンス』および『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

ユーザー定義ファンクション

PL/SQL または Java でユーザー定義ファンクションを作成し、SQL または SQL 組込みファンクションにはない機能を持たせることができます。ユーザー定義ファンクションは、SQL ファンクションを指定できるすべての SQL 文に指定できます。式を指定できる場所であれば、どこでも使用できます。

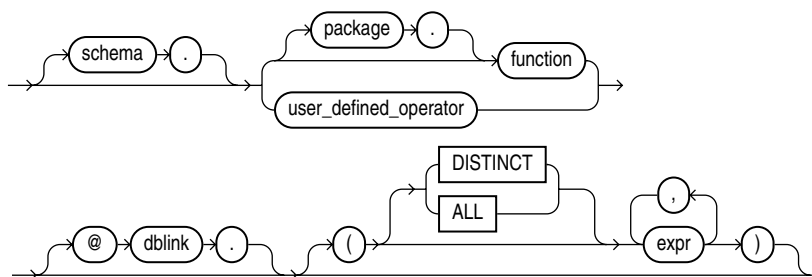
たとえば、SQL 文の次の場所でユーザー定義ファンクションを使用できます。

- SELECT 文の SELECT 構文のリスト
- WHERE 句の条件
- CONNECT BY 句、START WITH 句、ORDER BY 句および GROUP BY 句

- INSERT 文の VALUES 句
- UPDATE 文の SET 句

注意： Oracle SQL は、ブール値のパラメータまたは戻り値を持つファンクションのコールをサポートしていません。したがって、ユーザー定義ファンクションが SQL 文からコールされる場合は、数値 (0 または 1) または文字列 ('TRUE' または 'FALSE') を戻すように設計する必要があります。

user_defined_function::=



オプションの式のリストは、ファンクション、パッケージまたは演算子の属性と一致する必要があります。

ユーザー定義ファンクションの制限事項： DISTINCT および ALL キーワードは、ユーザー定義集計ファンクションのみで有効です。

参照：

- ファンクションの作成 (ユーザー定義ファンクションの制限を含む) の詳細は、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- ユーザー・ファンクションの作成および使用方法の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

前提条件

ユーザー定義ファンクションを SQL 文で使用するには、トップレベル・ファンクションとして作成するか、またはパッケージ仕様部で宣言する必要があります。

SQL の式の中でユーザー・ファンクションを使用するには、ユーザーがユーザー・ファンクションの EXECUTE 権限を持っている必要があります。ユーザー・ファンクションで定義したビューを問い合わせるには、そのビューに対する SELECT 権限が必要です。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。

参照：

- トップレベル・ファンクションの詳細は、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- パッケージ・ファンクションの詳細は、14-48 ページの「[CREATE PACKAGE](#)」を参照してください。

名前の優先順位

SQL 文内では、データベースの列名は、パラメータなしのファンクション名より優先順位が高くなります。たとえば、**Human Resources** のマネージャが、hr スキーマに次の 2 つのオブジェクトを作成する場合は、次のようにします。

```
CREATE TABLE new_emps (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 new_emps.new_sal 列を参照することになります。

```
SELECT new_sal FROM new_emps;  
SELECT new_emps.new_sal FROM new_emps;
```

new_sal ファンクションにアクセスするには、次のように入力します。

```
SELECT hr.new_sal FROM new_emps;
```

SQL の式内で使用できるユーザー・ファンクションのコール例を次に示します。

```
circle_area (radius)  
payroll.tax_rate (empno)  
hr.employees.tax_rate (dependent, empno)@remote
```

例 スキーマ `hr` から `tax_rate` ユーザー・ファンクションをコールし、`tax_table` 内の `sal` 列と `SAL` 列に対してこのファンクションを実行し、その結果を `income_tax` 変数に入れるには、次のように指定します。

```
SELECT hr.tax_rate (ss_no, sal)
       INTO income_tax
       FROM tax_table
       WHERE ss_no = tax_id;
```

ネーミング規則

オプションのスキーマ名またはパッケージ名を 1 つのみ指定すると、最初の識別子はスキーマ名またはパッケージ名のいずれかになります。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断するには、Oracle は次の手順を実行します。

1. カレント・スキーマ内の `PAYROLL` パッケージをチェックします。
2. `PAYROLL` パッケージが検出されない場合は、トップレベルの `TAX_RATE` ファンクションを含むスキーマ名 `PAYROLL` を検索します。このようなファンクションが検出されない場合は、エラーを戻します。
3. カレント・スキーマ内で `PAYROLL` パッケージが検出されると、`PAYROLL` パッケージの中で `TAX_RATE` ファンクションを検索します。このようなファンクションが検出されない場合は、エラーを戻します。

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することもできます。

共通の SQL DDL 句

この章では、複数の SQL 文で使用される SQL データ定義句について説明します。

この章では、次の内容を説明します。

- *allocate_extent_clause*
- *constraints*
- *deallocate_unused_clause*
- *file_specification*
- *logging_clause*
- *parallel_clause*
- *physical_attributes_clause*
- *storage_clause*

allocate_extent_clause

用途

allocate_extent_clause 句を使用すると、データベース・オブジェクトの新しいエクステントを明示的に割り当てることができます。

この句を使用してエクステントを明示的に割り当てた場合、NEXT および PCTINCREASE 記憶域パラメータの値は変更されません。したがって、Oracle で暗黙的に割り当てられる次のエクステントのサイズには影響ありません。

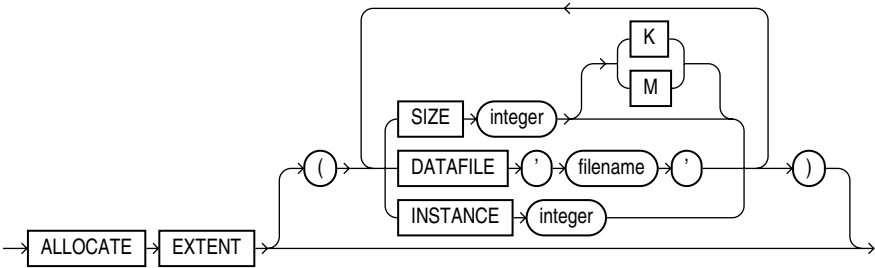
参照： NEXT および PCTINCREASE 記憶域パラメータの詳細は、7-51 ページの「storage_clause」を参照してください。

エクステントは次の SQL 文で割り当てることができます。

- ALTER CLUSTER (9-7 ページの「ALTER CLUSTER」を参照)
- ALTER INDEX: エクステントを索引、索引パーティションまたは索引サブパーティションに割り当てる場合 (9-61 ページの「ALTER INDEX」を参照)
- ALTER MATERIALIZED VIEW: エクステントをマテリアライズド・ビュー、そのパーティションまたはサブパーティションの 1 つ、あるいは索引構成マテリアライズド・ビューのオーバーフロー・セグメントに割り当てる場合 (9-88 ページの「ALTER MATERIALIZED VIEW」を参照)
- ALTER MATERIALIZED VIEW LOG (9-108 ページの「ALTER MATERIALIZED VIEW LOG」を参照)
- ALTER TABLE: エクステントを表、表パーティション、表サブパーティション、索引構成表のマッピング表、索引構成表のオーバーフロー・セグメントまたは LOB 記憶域セグメントに割り当てる場合 (11-2 ページの「ALTER TABLE」を参照)

構文

allocate_extent_clause::=



セマンティクス

この項では、`allocate_extent_clause`のパラメータについて説明します。詳細は、特定のデータベース・オブジェクトに対してこれらのパラメータを設定または再設定する SQL 文の説明を参照してください。

注意： 同じ文で `allocate_extent_clause` と `deallocate_unused_clause` を指定することはできません。

SIZE

エクステント・サイズをバイト単位で指定します。K または M を使用して、KB または MB 単位でエクステント・サイズを指定することもできます。

表、索引、マテリアライズド・ビューまたはマテリアライズド・ビュー・ログでは、SIZE を指定しないと、Oracle ではオブジェクトの記憶域パラメータの値に基づいてサイズが決定されます。ただし、クラスタの場合、クラスタの記憶域パラメータは評価されないため、デフォルト値を使用しない場合は SIZE を指定する必要があります。

DATAFILE '*filename*'

新しいエクステントを割り当てるデータ・ファイルを、表、クラスタ、索引、マテリアライズド・ビューまたはマテリアライズド・ビュー・ログの表領域から 1 つ指定します。DATAFILE を指定しないと、Oracle がデータ・ファイルを選択します。

INSTANCE *integer*

Oracle を Real Application Clusters とあわせて使用する場合のみ、このパラメータを使用します。

INSTANCE *integer* を指定すると、指定したインスタンスに対応付けられた空きリスト・グループが新しいエクステントを使用できるようになります。インスタンス数が空きリスト・グループの最大数を超えた場合、指定したインスタンス数 / 最大数という除算が行われ、その余りから、使用する空きリスト・グループが識別されます。インスタンスは初期化パラメータ INSTANCE_NUMBER の値で識別されます。

このパラメータを指定しない場合、表、クラスタ、索引、マテリアライズド・ビューまたはマテリアライズド・ビュー・ログに領域が割り当てられますが、特定の空きリスト・グループからは割り当てられません。かわりにマスター空きリストが使用され、必要に応じて領域が割り当てられます。

注意： 自動セグメント領域管理の使用時は、`allocate_extent_clause` の `INSTANCE` パラメータを設定しても、指定したインスタンスに新しく割り当てられた領域が確保されない場合があります。自動セグメント領域管理は、エクステンとインスタンス間の固定したアフィニティを保持しないためです。

参照： `allocate_extent_clause` の `INSTANCE` パラメータの設定の詳細は、『Oracle9i Real Application Clusters 管理』を参照してください。

constraints

用途

constraints を使用すると、**整合性制約**（データベース内の値を制限する規則）を定義できます。Oracle では、6 つの制約を作成し、それを 2 つの方法で宣言することができます。

次に、6 つの整合性制約について簡単に説明します。詳細は、7-10 ページの「[セマンティクス](#)」を参照してください。

- **NOT NULL 制約**は、データベースの値が NULL になることを禁止します。
- **一意制約**は、複数の行が同じ列または列の組合せで同じ値を持つことを禁止しますが、一部の値が NULL になることを許可します。
- **主キー制約**は、1 つの宣言で NOT NULL 制約と一意制約を組み合わせたものです。つまり、複数の行が同じ列または列の組合せで同じ値を持つこと、および値が NULL になることを禁止します。
- **外部キー制約**は、ある表の値が別の表の値と一致することを必要とします。
- **CHECK 制約**は、データベースの値が、指定された条件を満たすことを必要とします。
- REF 列は、定義上は別のオブジェクト型またはリレーショナル表の中のオブジェクトを参照します。**REF 制約**を使用すると、REF 列と参照先のオブジェクトの関係をさらに詳しく指定できます。

制約は、次の 2 つの構文で定義できます。

- 個々の列または属性の定義の一部として定義できます。これを、**表内指定**といいます。
- 表定義の一部として定義できます。これを、**表外指定**といいます。

NOT NULL 制約は、表内に宣言する必要があります。その他のすべての制約は、表内または表外に宣言できます。

制約句は、次の文に指定できます。

- CREATE TABLE（15-7 ページの「[CREATE TABLE](#)」を参照）
- ALTER TABLE（11-2 ページの「[ALTER TABLE](#)」を参照）
- CREATE VIEW（16-38 ページの「[CREATE VIEW](#)」を参照）
- ALTER VIEW（12-29 ページの「[ALTER VIEW](#)」を参照）

ビュー制約 Oracle では、ビュー制約を適用していません。ただし、実表に対する制約によってビューに制約を適用できます。

ビューには、一意制約、主キー制約および外部キー制約のみを指定でき、これらの制約は `DISABLE NOVALIDATE` モードのみでサポートされています。オブジェクト列の属性にビュー制約を定義することはできません。

参照：

- ビュー制約の詳細は、7-24 ページの「[ビュー制約](#)」を参照してください。
- `DISABLE NOVALIDATE` モードの詳細は、7-20 ページの「[DISABLE 句](#)」を参照してください。

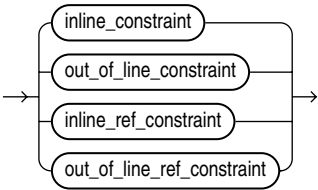
前提条件

制約を定義する文を発行できる権限が必要です。

外部キー制約を作成する場合は、この条件に加えて、親表またはビューが自分のスキーマ内に設定されている必要があります。設定されていないかぎり、親表またはビューの参照キー列に対する `REFERENCES` 権限が必要です。

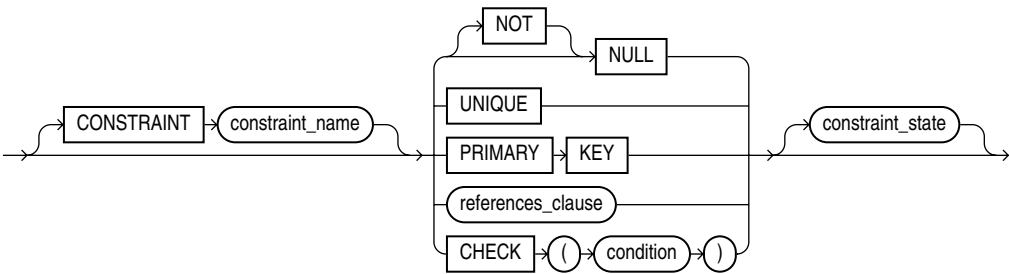
構文

`constraints::=`



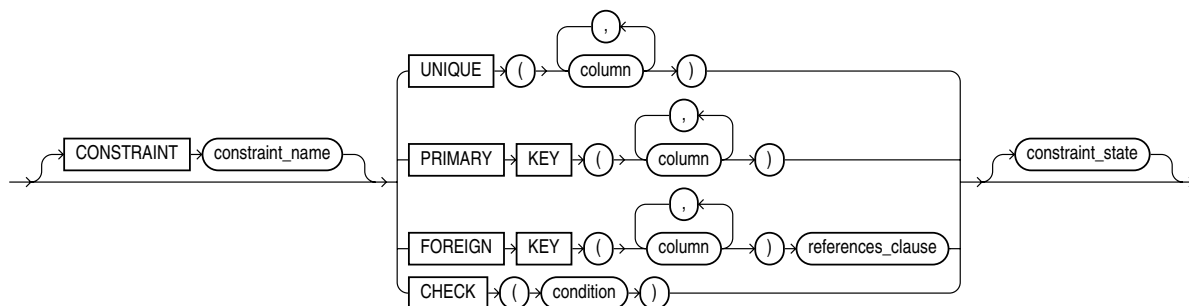
(7-6 ページの「[inline_constraint::=](#)」、7-7 ページの「[out_of_line_constraint::=](#)」、7-7 ページの「[inline_ref_constraint::=](#)」、7-7 ページの「[out_of_line_ref_constraint::=](#)」を参照)

`inline_constraint::=`



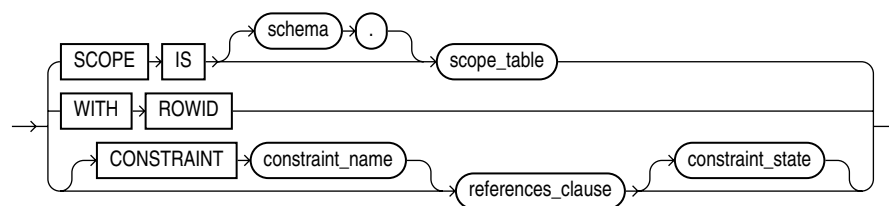
(7-8 ページの「[references_clause::=](#)」を参照)

out_of_line_constraint::=



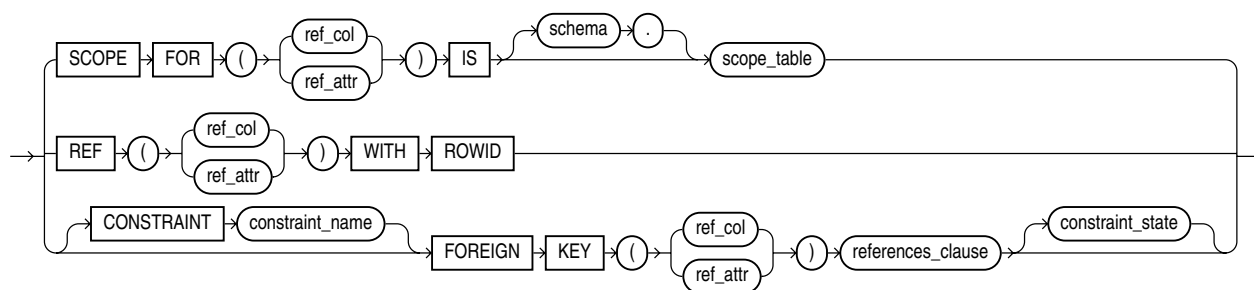
(7-8 ページの「[references_clause::=](#)」、7-8 ページの「[constraint_state::=](#)」を参照)

inline_ref_constraint::=



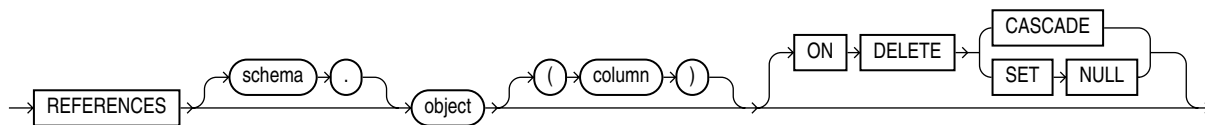
(7-8 ページの「[references_clause::=](#)」、7-8 ページの「[constraint_state::=](#)」を参照)

out_of_line_ref_constraint::=

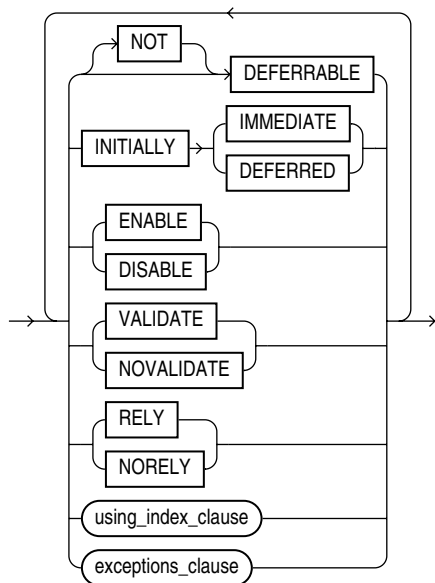


(7-8 ページの「[references_clause::=](#)」、7-8 ページの「[constraint_state::=](#)」を参照)

references_clause::=

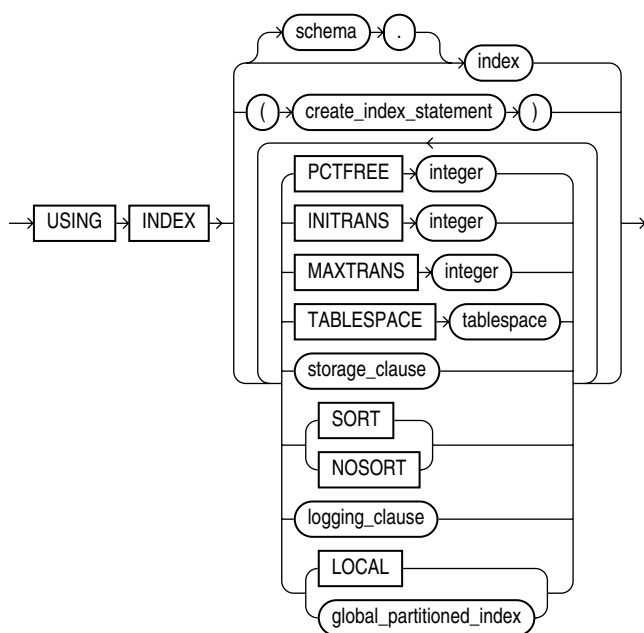


constraint_state::=



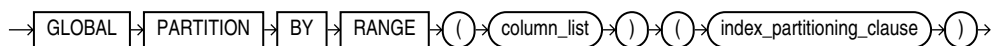
(7-9 ページの「**using_index_clause::=**」、7-10 ページの「**exceptions_clause::=**」を参照)

using_index_clause::=

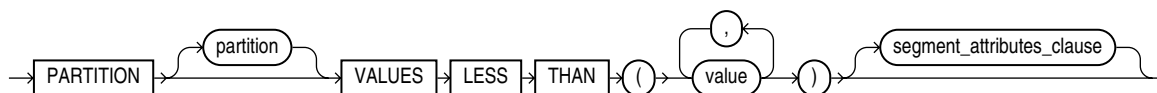


(13-62 ページの「[create_index::=](#)」、7-51 ページの「[storage_clause](#)」、7-42 ページの「[logging_clause](#)」、7-9 ページの「[global_partitioned_index::=](#)」を参照)

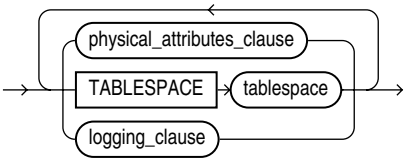
global_partitioned_index::=



index_partitioning_clause::=

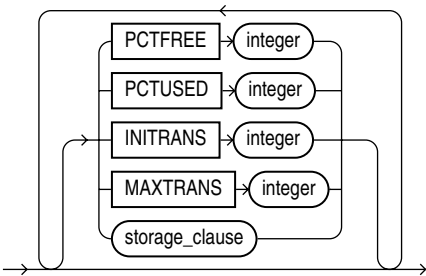


segment_attributes_clause::=



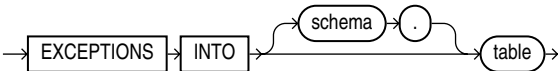
(7-48 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause](#)」を参照)

physical_attributes_clause::=



(7-51 ページの「[storage_clause](#)」を参照)

exceptions_clause::=



セマンティクス

この項では、`constraints` のセマンティクスについて説明します。詳細は、表またはビューの制約を定義または再定義する `SQL` 文の説明を参照してください。

- ---
- 注意：** Oracle では、ユーザー定義オブジェクト、ネストした表、`VARRAY`、`REF` または `LOB` 型の列または属性に対して制約を使用することはできません。ただし、次の 2 つの例外があります。

 - `NOT NULL` 制約は、ユーザー定義オブジェクト、`VARRAY`、`REF` または `LOB` 型の列や属性に使用できます。
 - `NOT NULL`、外部キーおよび `REF` 制約は、`REF` 型の列に使用できます。
- ---

CONSTRAINT *constraint_name* 制約名を指定します。この識別子を指定しない場合、SYS_Cn の形式で名前が生成されます。整合性制約の名前と定義は、USER_、ALL_ および DBA_CONSTRAINTS データ・ディクショナリ・ビュー（それぞれ CONSTRAINT_NAME 列および SEARCH_CONDITION 列）に格納されます。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

NOT NULL 制約

NOT NULL 制約は、列に NULL が含まれることを禁止します。NULL キーワードは、実際に整合性制約を定義するものではありませんが、これを指定すると、列に NULL が含まれることが許可されます。NOT NULL および NULL は、表内指定で定義する必要があります。NOT NULL または NULL を指定しない場合、NULL がデフォルトになります。

NOT NULL 制約は、XMLType 列および VARRAY 列で表内指定できる唯一の制約です。

NOT NULL 制約を満たすには、表のすべての行がその列の値を持つ必要があります。

注意： Oracle では、すべてのキー列が NULL の表の行には索引を付けません（ただし、ビットマップ索引の場合を除きます）。このため、表のすべての行に索引を付けるには、1 つ以上の索引キー列に NOT NULL 制約を指定するか、ビットマップ索引を作成する必要があります。

NOT NULL 制約の制限事項：

- ビュー制約には NULL または NOT NULL を指定できません。
- オブジェクト属性には NULL または NOT NULL を指定できません。そのかわりに、IS [NOT] NULL 条件で CHECK 制約を使用してください。

参照： 7-31 ページの「[属性レベル制約の例](#)」および 7-27 ページの「[NOT NULL の例](#)」を参照してください。

一意制約

一意制約は、列を一意キーとして指定します。**複合一意キー**は、列の組合せを一意キーとして指定します。一意制約を表内に定義する場合に必要なのは、UNIQUE キーワードのみです。一意制約を表外に定義する場合は、1 つ以上の列も指定する必要があります。複合一意キーは、表外に定義する必要があります。

一意制約を満たすには、表の中の 2 つの行が一意キーに対して同じ値を持たないようにする必要があります。ただし、単一の列で構成される一意キーの場合は、複数の NULL を持つことができます。複合一意キーを満たすには、表またはビューの 2 つの行がキー列に対して同じ組合せの値を持たないようにする必要があります。すべてのキー列に対して NULL を持つ行は、自動的にその制約を満たすことになります。ただし、1 つ以上のキー列に対して NULL を持ち、その他のキー列に対して同じ組合せの値を持つ 2 つの行は、制約に違反します。

注意： Oracle では、1 つ以上の列に一意制約を指定すると、暗黙的に一意キーに索引が作成されます。問合せのパフォーマンスを目的に一意性を定義する場合は、かわりに `CREATE UNIQUE INDEX` 文を使用して明示的に一意索引を作成することをお勧めします。詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。

一意制約の制限事項：

- 表またはビューには、一意キーを 1 つのみ指定できます。
- 一意キー列は、LOB、LONG、LONG RAW、VARRAY、NESTED TABLE、OBJECT、BFILE、REF または `TIMESTAMP WITH TIME ZONE` データ型を含むことができません。ただし、一意キーは `TIMESTAMP WITH LOCAL TIME ZONE` の列を含むことができます。
- 1 つの複合一意キーは、33 以上の列を持つことはできません。
- 同一の列または列の組合せを一意キーと主キーの両方には指定できません。
- 継承階層でサブテーブルまたはサブビューを作成する場合は、一意キーを指定できません。一意キーは、トップレベル（ルート）の表またはビューのみに指定できます。

参照： 7-25 ページの「[一意キーの例](#)」および 7-26 ページの [複合一意キーの例](#)を参照してください。

主キー制約

主キー制約は、列を表またはビューの主キーとして指定します。**複合主キー**は、列の組合せを主キーとして指定します。主キー制約を表内に定義する場合に必要なのは、`PRIMARY KEY` キーワードのみです。主キー制約を表外に定義する場合は、1 つ以上の列も指定する必要があります。複合主キーは表外に定義する必要があります。

主キー制約は、1 つの宣言で `NOT NULL` 制約と一意制約を組み合わせたものです。このため、主キー制約を満たすには次の条件があります。

- 主キーの値が、表の中の複数行に存在することはできません。
- 主キーを構成する列に、NULL を持たせることはできません。

主キー制約の制限事項：

- 表またはビューには、主キーを 1 つのみ指定できます。
- 主キー列は、LOB、LONG、LONG RAW、VARRAY、NESTED TABLE、OBJECT、BFILE、REF または TIMESTAMP WITH TIME ZONE データ型を含むことができません。ただし、主キーは TIMESTAMP WITH LOCAL TIME ZONE の列を含むことができます。
- 主キーのサイズは、1 データベース・ブロックを超えることはできません。
- 1 つの複合主キーは、33 以上の列を持つことはできません。
- 同一の列または列の組合せを一意キーと主キーの両方には指定できません。
- 継承階層でサブテーブルまたはサブビューを作成する場合は、主キーを指定できません。主キーは、トップレベル（ルート）の表またはビューのみに指定できます。

参照： 7-26 ページの「[主キーの例](#)」および 7-27 ページの「[複合主キーの例](#)」を参照してください。

外部キー制約

外部キー制約（**参照整合性制約**ともいう）は、列を外部キーとして指定し、その外部キーと指定した主キーまたは一意キー（**参照キー**）との関係を設定します。**複合外部キー**は、列の組合せを外部キーとして指定します。

外部キーを持つ表またはビューを**子オブジェクト**といい、参照キーを持つ表またはビューを**親オブジェクト**といいます。外部キーと参照キーを、同一の表またはビューに設定することができます。この場合、親表と子表は同一の表になります。親表または親ビューのみを指定して、列名を指定しない場合、外部キーは自動的に親表または親ビューの主キーを参照します。外部キーと参照キーの対応する列は、同じ順序とデータ型で構成されている必要があります。

単一キー列の外部キー制約は、表内または表外に定義できます。複合外部キーと属性の外部キーは、表外に指定します。

複合外部キー制約を満たすには、複合外部キーが親表または親ビューの複合一意キーまたは複合主キーを参照するか、外部キーの 1 つ以上の列の値が NULL である必要があります。

外部キーと主キーの両方、または外部キーと一意キーの両方に、同一の列または列の組合せを指定できます。また、外部キーとクラスタ・キーの両方にも同じ列または列の組合せを指定できます。

1 つの表またはビューで複数の外部キーを定義できます。また、1 つの列が複数の外部キーを構成することもできます。

外部キー制約の制限事項：

- 外部キー列は、LOB、LONG、LONG RAW、VARRAY、NESTED TABLE、OBJECT、BFILE、REF または TIMESTAMP WITH TIME ZONE データ型を含むことができません。ただし、外部キーは TIMESTAMP WITH LOCAL TIME ZONE の列を含むことができます。
- 親表または親ビューの参照一意制約または参照主キー制約が事前に定義されている必要があります。
- 1 つの複合外部キーは、33 以上の列を持つことはできません。
- 子表と親表は、同一データベース上に設定されている必要があります。分散データベースのノード間の参照整合性制約を使用可能にする場合、データベース・トリガーを使用する必要があります。15-89 ページの「[CREATE TRIGGER](#)」を参照してください。
- 子オブジェクトと親オブジェクトのどちらかがビューの場合、ビュー制約のすべての制限事項が制約に適用されます。7-24 ページの「[ビュー制約](#)」を参照してください。
- AS *subquery* 句を含む CREATE TABLE 文には、外部キー制約を定義できません。そのかわりに、制約を指定せずに表を作成し、後で ALTER TABLE 文を使用してその制約を追加できます。

参照：『Oracle9i アプリケーション開発者ガイド - 基礎編』、7-27 ページの「[外部キー制約の例](#)」および 7-29 ページの「[複合外部キー制約の例](#)」を参照してください。

references_clause 外部キー制約は *references_clause* 構文を使用します。外部キー制約を表内に指定する場合に必要なのは、*references_clause* のみです。外部キー制約を表外に指定する場合は、FOREIGN KEY キーワードと 1 つ以上の列も指定する必要があります。

ON DELETE 句 ON DELETE 句を指定すると、参照主キーまたは参照一意キーの値を削除した場合に参照整合性がどのように自動処理されるかを指定できます。この句を指定しない場合、子表に依存する行を持つ親表の中の参照キーの値は削除できません。

- 依存する外部キーの値を削除する場合は、CASCADE を指定します。
- 依存する外部キーの値を NULL に変換する場合は、SET NULL を指定します。

ON DELETE の制限事項： この句は、ビュー制約には指定できません。

参照： 7-28 ページの「[ON DELETE の例](#)」を参照してください。

CHECK 制約

CHECK 制約によって、表の各行に必要な条件を指定できます。制約を満たすためには、表のそれぞれの行が、その条件に対して TRUE または不明 (NULL のため) のいずれかである必要があります。特定の行に対する CHECK 制約条件が評価される場合、条件にある列名に、その行の列値が適用されます。

CHECK 制約の表内指定と表外指定の構文は同じです。ただし、表内指定では現在定義されている列 (または、オブジェクト列の場合は列の属性) のみを参照でき、表外指定では複数の列または属性を参照できます。

注意： Oracle では、CHECK 制約の条件が相互に排他的かどうかは検証しません。このため、1 つの列に対して複数の CHECK 制約を作成する場合は、制約の用途が矛盾しないように注意する必要があります。また、条件の評価について特別な順序を想定しないでください。

参照：

- その他の詳細および構文は、[第 5 章「条件」](#)を参照してください。
- 7-29 ページの「[CHECK 制約の例](#)」および 7-31 ページの「[属性レベル制約の例](#)」を参照してください。

CHECK 制約の制限事項：

- ビューに対しては、CHECK 制約を指定できません。ただし、WITH CHECK OPTION 句を使用してビューを定義することは可能です。これは、ビューに CHECK 制約を指定することと同じです。
- CHECK 制約の条件では、その表の中のすべての列を参照できますが、他の表の列は参照できません。
- CHECK 制約の条件は、次の構造を持つことができません。
 - 副問合せおよびスカラー副問合せ式
 - 決定的でないファンクションへのコール (CURRENT_DATE、CURRENT_TIMESTAMP、DBTIMEZONE、LOCALTIMESTAMP、SESSIONTIMEZONE、SYSDATE、SYSTIMESTAMP、UID、USER および USERENV)
 - ユーザー定義ファンクションへのコール
 - REF 列の参照解除 (DEREF ファンクションを使用する場合など)
 - ネストした表の列または属性

- 疑似列 CURRVAL、NEXTVAL、LEVEL または ROWNUM
- 完全に指定されていない日付定数

REF 制約

REF 制約を使用すると、REF 型の列とそれが参照するオブジェクトとの関係を指定できます。

ref_constraint REF 制約は、*ref_constraint* 構文を使用します。REF 制約は表内または表外に定義します。表外指定の場合は、指定している REF 列または属性を指定する必要があります。

- *ref_column* には、オブジェクトまたはリレーショナル表の REF 列の名前を指定します。
- *ref_attribute* には、リレーショナル表のオブジェクト列内の埋込み REF 属性を指定します。

表内指定と表外指定のどちらでも、有効範囲制約、ROWID 制約または参照整合性制約を REF 列に定義できます。

REF 列の有効範囲表または参照表が、主キーに基づくオブジェクト識別子を持っている場合、その REF 列は**ユーザー定義 REF 列**です。

参照： REF の詳細は、『Oracle9i データベース概要』を参照してください。また、7-13 ページの「[外部キー制約](#)」および 7-31 ページの「[REF 制約の例](#)」も参照してください。

REF 列の有効範囲制約

表が REF 列を持つ場合は、この列のそれぞれの REF 値が別のオブジェクト表内の行を参照する場合があります。SCOPE 句は、参照の有効範囲を 1 つの表 *scope_table* に制限します。REF 列または属性の値は *scope_table* 内のオブジェクトを指し、その場所に（REF 列と同じ型の）オブジェクト・インスタンスが格納されます。

REF 列内の参照の有効範囲を 1 つの表に制限する場合に、SCOPE 句を指定します。この句を指定するには、*scope_table* が自分のスキーマ内にあるか、または *scope_table* に対する SELECT 権限または SELECT ANY TABLE システム権限が必要です。REF 列ごとに有効範囲表を 1 つのみ指定できます。

有効範囲制約の制限事項：

- 表が空でない場合は、有効範囲制約を既存の列に追加できません。
- VARRAY 列の REF 要素に対しては有効範囲制約を指定できません。
- AS *subquery* を指定し、この副問合せがユーザー定義 REF を戻す場合、この句を指定する必要があります。

- 有効範囲制約を、後で REF 列から削除することはできません。

REF 列の ROWID 制約

WITH ROWID を指定して、*ref_column* または *ref_attribute* の REF の値とともに ROWID を格納します。ROWID とともに REF 値を格納した場合、参照解除操作のパフォーマンスは向上しますが、使用する領域も多くなります。デフォルトの REF 値の記憶域では、ROWID は格納されません。

参照： 参照解除の例は、6-53 ページの「[DEREF](#)」ファンクションを参照してください。

ROWID 制約の制限事項：

- VARRAY 列の REF 要素に対しては ROWID 制約を定義できません。
- ROWID 制約を、後で REF 列から削除することはできません。
- REF 列または属性の範囲が限定される場合、この句は無視され、ROWID は REF とともに格納されません。

REF 列の参照整合性制約

ref_constraint 構文の *references_clause* を使用すると、外部キー制約を REF 列に定義できます。また、この句は REF 列や属性の有効範囲を参照表に暗黙的に制限します。ただし、REF 列以外の外部キー制約が親表の実際の列を参照することに対し、REF 列の外部キー制約は親表の暗黙のオブジェクト識別子 (OID) を参照します。

CONSTRAINT を指定しない場合、制約に対するシステム名が *SYS_Cn* という形式で生成されます。

範囲が限定されている既存の REF 列に参照整合性制約を追加する場合、参照表は、REF 列の有効範囲表と同じである必要があります。後で参照整合性制約を削除する場合、REF 列の範囲が参照表に限定されたままになります。

他の型の列に対する外部キー制約と同様に、表内で宣言する場合に必要なのは *references_clause* のみです。表外で宣言する場合は、FOREIGN KEY キーワード、および 1 つ以上の REF 列または属性も指定する必要があります。

参照： オブジェクト識別子の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

REF 列の参照整合性制約の制限事項：

- 範囲が限定されていない既存の REF 列に参照整合性制約を追加する場合、有効範囲制約が暗黙的に追加されます。したがって、有効範囲制約に適用されるすべての制限事項は、この場合にも適用されます。
- *references_clause* 内のオブジェクト名の後には列を指定できません。

制約状態の指定

制約定義の一部として、制約が適用される方法およびタイミングを指定できます。

constraint_state *constraint_state* は、表内指定と表外指定の両方に使用できます。*constraint_state* の句はどのような順序でも指定できますが、各句を指定できるのは1回のみです。

DEFERRABLE 句 DEFERRABLE および NOT DEFERRABLE パラメータは、後続のトランザクションで、SET CONSTRAINT(S) 文を使用して、トランザクションが終わるまで制約のチェックを遅延できるかどうかを指定します。この句を指定しない場合、NOT DEFERRABLE がデフォルトになります。

- NOT DEFERRABLE を指定すると、後続のトランザクションで、SET CONSTRAINT[S] 句を使用して、トランザクションがコミットされるまでこの制約のチェックを遅延させることができます。NOT DEFERRABLE 制約のチェックは、トランザクションの終わりまで遅延させることはできません。

注意： 新しい NOT DEFERRABLE 制約を宣言する場合、CREATE TABLE 文のコミット時にその制約が有効である必要があります。有効でない場合、CREATE 文は正常に実行されません。

- DEFERRABLE を指定すると、後続のトランザクションで、SET CONSTRAINT[S] 句を使用して、トランザクションがコミットされるまでこの制約のチェックを遅延させることができます。この設定により、制約に違反する可能性のある変更をデータベースに行っている場合に、すべての変更が完了するまで、制約を一時的に使用禁止にすることが可能になります。

制約の遅延可能状態は変更できません。つまり、これらのパラメータのいずれかを指定したか、どちらも指定せずに NOT DEFERRABLE 制約を暗黙的に有効にしたかによって、この句を ALTER TABLE 文に指定することができなくなります。制約を削除してから再作成する必要があります。

参照：

- トランザクションに対する制約のチェックの設定の詳細は、18-42 ページの「[SET CONSTRAINT\[S\]](#)」を参照してください。
- 遅延制約の詳細は、『Oracle9i データベース管理者ガイド』および『Oracle9i データベース概要』を参照してください。
- 7-33 ページの「[DEFERRABLE 制約の例](#)」を参照してください。

[NOT] DEFERRABLE の制限事項： ビュー制約にはこれらのパラメータを指定できません。

INITIALLY 句 INITIALLY 句は、DEFERRABLE 句が指定されている制約に対するデフォルトのチェック動作を指定します。INITIALLY 設定は、後続のトランザクションに SET CONSTRAINT(S) 文を指定することで上書きできます。

- INITIALLY IMMEDIATE を指定すると、この制約は後続の各 SQL 文の終わりでチェックされます。INITIALLY を指定しない場合、INITIALLY IMMEDIATE がデフォルトになります。

注意： 新しい INITIALLY IMMEDIATE 制約を宣言する場合、CREATE TABLE 文のコミット時にその制約が有効である必要があります。有効でない場合、CREATE 文は正常に実行されません。

- INITIALLY DEFERRED を指定すると、この制約は後続のトランザクションの終わりでチェックされます。

制約を NOT DEFERRABLE として宣言した場合、この句は無効です。NOT DEFERRABLE 制約は自動的に INITIALLY IMMEDIATE になり、INITIALLY DEFERRED に変更することはできないためです。

VALIDATE | NOVALIDATE VALIDATE および NOVALIDATE の動作は、制約が明示的にまたはデフォルトで使用可能 / 使用禁止のどちらになっているかで異なります。詳細は、7-19 ページの「[ENABLE 句](#)」および 7-20 ページの「[DISABLE 句](#)」を参照してください。

ENABLE 句 制約を表のデータに適用する場合は、ENABLE を指定します。

注意： 一意制約または主キー制約を使用可能にした場合、キーに索引が存在しないと、一意索引が作成されます。その後で制約が使用禁止になった場合、この索引は削除されます。そのため、制約が使用可能になるたびに索引が再作成されます。

索引の再作成を避け、余分な索引を削除するために、最初に使用禁止にした主キー制約および一意制約を新しく作成します。その後、一意でない索引を作成して（または、既存の一意でない索引を使用して）制約を適用してください。制約が使用禁止の場合、一意でない索引は削除されず、後続の ENABLE 操作が容易になります。

- ENABLE VALIDATE を使用すると、すべての旧データと新データが制約に従うことを指定できます。制約が使用可能で、妥当性チェック済の場合は、すべてのデータが現在有効で、今後も有効であることが保証されます。

整合性制約に違反する行が表にある場合、制約は使用禁止のままエラーを戻します。すべての行が制約に従っている場合、Oracle は制約を使用可能にします。新規データが整合性制約に違反する場合、その後の文は実行されず、整合性制約の違反を示すエラーが戻されます。

注意： 主キー制約を **ENABLE VALIDATE** モードに設定すると、妥当性チェック・プロセスによって主キー列に **NULL** が含まれないことが検証されます。これによるオーバーヘッドを回避するには、データを列に入力する前、およびこの表の主キー制約を使用可能にする前に、主キーの各列に **NOT NULL** というマークを付けます。

- **ENABLE NOVALIDATE** を使用すると、制約データに対して新しく行うすべての **DML** 操作が制約に従うことが保証されます。この句は、表の既存データが制約に従っていることを保証しないため、表ロックは必要ありません。

VALIDATE も **NOVALIDATE** も指定しない場合、**VALIDATE** がデフォルトになります。

ENABLE NOVALIDATE から **ENABLE VALIDATE** に単一制約状態を変更すると、パラレルで操作が実行できるため、読み込み、書き込みまたはその他の **DDL** 操作が中断されません。

ENABLE 句の制限事項： 使用禁止になっている一意キーまたは主キーを参照する外部キーを使用可能にすることはできません。

DISABLE 句 整合性制約を使用禁止にする場合は、**DISABLE** を指定します。データ・ディクショナリでは、使用禁止になっている整合性制約は、使用可能な制約とともに表示されます。この句を指定せずに制約を作成した場合は、その制約は自動的に使用可能になります。

- **DISABLE VALIDATE** は、制約を使用禁止にして制約の索引を削除しますが、制約は有効のままです。この機能を使用すると、大量のデータをロードでき、また索引用の領域が削減されるため、データ・ウェアハウスで非常に有効です。この設定を使用することで、**ALTER TABLE** 文の *exchange_partition_clause* または **SQL*Loader** を使用して、データを非パーティション表からパーティション表にロードすることもできます。他の **SQL** 文を使用した表に対するすべての変更（挿入、更新および削除）は禁止されます。

参照： この設定の使用の詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

- **DISABLE NOVALIDATE** は、Oracle によって制約がメンテナンスされないこと（使用禁止になっているため）、および制約が真であると保証されないこと（妥当性チェックが行われていないため）を示します。

外部キー制約が **DISABLE NOVALIDATE** 状態であっても、外部キーが参照している主キーを持つ表を削除できません。また、オブティマイザは、**DISABLE NOVALIDATE** 状態でも制約を使用できます。

参照： この設定の使用方法の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

VALIDATE も NOVALIDATE も指定しない場合、NOVALIDATE がデフォルトになります。

一意索引を使用している一意制約または主キー制約を使用禁止にすると、一意索引は削除されます。

参照： その他の注意事項や制限事項は、15-52 ページの「CREATE TABLE」の「[enable_disable_clause](#)」を参照してください。

RELY 句 RELY および NORELY は、既存の制約を変更する場合のみ（つまり、ALTER TABLE ... MODIFY 制約構文内でのみ）有効です。これらのパラメータは、クエリー・リライトで NOVALIDATE モードの制約を考慮するかどうかを指定します。RELY を指定すると、適用されていないクエリー・リライトに対する NOVALIDATE モードの既存の制約は、アクティブになります。その制約は NOVALIDATE モードであるため、適用されません。デフォルト値は NORELY です。

適用されない制約は、通常、マテリアライズド・ビューおよびクエリー・リライトにのみ有効です。QUERY_REWRITE_INTEGRITY モード（10-2 ページの「[ALTER SESSION](#)」を参照）に従って、クエリー・リライトは、VALIDATE モードの制約、または RELY パラメータが設定された NOVALIDATE モードの制約を使用して、結合情報を確認します。

RELY 句の制限事項： RELY には、遅延不可の NOT NULL 制約を設定できません。

参照： マテリアライズド・ビューおよびクエリー・リライトの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

索引による制約の適用

一意制約または主キー制約の状態を定義する場合は、制約の適用に使用する索引を指定すること、または制約の適用に使用する索引を Oracle で自動作成することが可能です。

using_index_clause *using_index_clause* は、一意制約または主キー制約を使用可能にしている場合にのみ指定できます。*using_index_clause* の句はどのような順序でも指定できますが、各句を指定できるのは 1 回のみです。

- *schema.index* を指定すると、指定した索引を使用して制約が適用されます。索引がない、または制約の適用に索引が使用できない場合、エラーが戻されます。
- *create_index_statement* を指定すると、索引が作成され、制約の適用に使用されます。索引が作成できない、または制約の適用に索引が使用できない場合、エラーが戻されます。
- 既存の索引の指定も、新しい索引の作成も行わない場合は、索引が自動で作成されます。この場合、次のようになります。

- 索引には制約と同じ名前が割り当てられます。
- 索引には、INITRANS、MAXTRANS、TABLESPACE、PCTFREE および STORAGE の各パラメータ値を選択できます。PCTUSED または *logging_clause* を索引に指定することはできません。
- 表がパーティション化されている場合、一意制約または主キー制約にローカル・パーティション索引またはグローバル・パーティション索引を指定できます。

using_index_clause の制限事項：

- この句は、ビュー制約には指定できません。
- この句は、NOT NULL、外部キーまたは CHECK 制約には指定できません。
- 索引構成表の主キーが使用可能な場合は、索引の指定 (*schema.index*) および作成 (*create_index_statement*) はできません。

参照：

- INITRANS、MAXTRANS、TABLESPACE および PCTFREE パラメータの詳細は 7-47 ページの「[physical_attributes_clause](#)」を、記憶域パラメータの詳細は 7-51 ページの「[storage_clause](#)」を参照してください。
- LOCAL と *global_partitioned_index* 句、および索引に関連する NOSORT と *logging_clause* の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。
- 7-32 ページの「[明示的な索引の制御例](#)」を参照してください。

NOSORT 句 NOSORT によって、データベースに表の行が昇順で格納されているため、索引を作成する場合に行をソートする必要がないことを指定します。

logging_clause *logging_clause* を使用すると、索引の作成を REDO ログ・ファイルに記録するかどうかを指定できます。

参照： 7-42 ページの「[logging_clause](#)」を参照してください。

global_partitioned_index *global_partitioned_index* によって、索引のパーティション化がユーザー定義であり、基礎となる表と同一レベルでパーティション化されないことを指定できます。デフォルトでは、非パーティション索引はグローバル索引です。グローバル索引は、*column_list* に指定された表の列の値の範囲に基づいてパーティション化されます。ローカル索引にこの句は指定できません。

`column_list` では、索引の列リストの左の接頭辞を指定する必要があります。つまり、索引が列 `a`、`b` および `c` に定義されている場合は、`column_list` に `(a,b,c)`、`(a,b)` または `(a,c)` は指定できますが、`(b,c)`、`(c)` または `(b,a)` は指定できません。

グローバル・パーティション索引キーの制限事項：

- `column_list` に指定できるのは 32 列以下です。
- これらの列には、ROWID 類似列または ROWID 型の列は指定できません。

注意： 異なるキャラクタ・セットを使用してデータベースを使用しているか、使用する予定がある場合は、キャラクタ列を分割する際に注意してください。文字のソート順序は、すべてのキャラクタ・セットで同一ではありません。

参照： キャラクタ・セットのサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

index_partitioning_clause この句は、個々の索引パーティションを示すために使用します。この句が繰り返される数によってパーティションの数が決まります。`partition` を指定しない場合、名前は `SYS_Pn` の形式で生成されます。

`VALUES LESS THAN (value_list)` には、グローバル索引の現在のパーティションの上限（境界は含まない）を指定します。値のリストは、`global_partitioned_index` 句の列リストに対応するリテラル値を含む、カンマで区切られた順序リストです。最後のパーティションの値としては、必ず `MAXVALUE` を指定します。

注意： 索引が `DATE` 列でパーティション化されている場合、および日付書式で年の最初の 2 桁の数字が指定されていない場合、年の 4 文字書式マスクを指定して `TO_DATE` ファンクションを使用する必要があります。日付書式は、`NLS_TERRITORY` によって暗黙的に決定されるか、`NLS_DATE_FORMAT` によって明示的に決定されます。

制約の例外の処理

制約の状態を定義する場合は、制約に違反するすべての行の ROWID を格納する表を指定できます。

exceptions_clause `exceptions_clause` 構文を使用すると、例外の処理を定義できます。`schema` を指定しない場合、自分のスキーマ内に例外表があるとみなされます。この句自体を指定しない場合、表の名前は `EXCEPTIONS` になります。`EXCEPTIONS` 表または指定した表は、ローカル・データベースに存在する必要があります。

次のいずれかのスクリプトを使用して、EXCEPTIONS 表を作成できます。

- UTLEXCPT.SQL は、物理 ROWID を使用します。そのため、行は、索引構成表からではなく従来表から収集されます（次の注意を参照）。
- UTLEXPT1.SQL は、ユニバーサル ROWID を使用します。そのため、行は、従来表と索引構成表の両方から収集されます。

独自の例外表を作成する場合、これら 2 つのスクリプトのいずれかで規定される形式に従う必要があります。

exceptions_clause の制限事項：

- この句は、ビュー制約には指定できません。
- この文が正常に終了されるまで ROWID は存在しないため、この句を CREATE TABLE 文に指定することはできません。

注意： ユニバーサル ROWID ではなく、主キーに基づく索引構成表から例外を収集する場合、索引構成表ごとに別の例外表を作成し、主キー記憶域を確保する必要があります。スクリプトを変更および再発行することによって、別の名前の例外表を複数作成できます。

参照：

- これらのスクリプトの使用に関する互換性については、『Oracle9i データベース移行ガイド』を参照してください。
- SQL スクリプトの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_IOT パッケージを参照してください。
- 移行行および連鎖行の削除については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

ビュー制約

Oracle では、ビュー制約を適用していません。ただし、ビューに対する操作には、基となる実表に定義されている整合性制約が適用されます。つまり、実表に対する制約によって、ビューに制約を適用できます。

ビュー制約の制限事項： ビュー制約は表制約のサブセットで、次の制限事項があります。

- ビューには、一意制約、主キー制約および外部キー制約のみ指定できます。ただし、WITH CHECK OPTION 句を使用してビューを定義することは可能です。これは、ビューに CHECK 制約を指定することと同じです。

- ビュー制約は直接適用されないため、INITIALLY DEFERRED または DEFERRABLE は指定できません。
- ビュー制約は、DISABLE NOVALIDATE モードのみでサポートされています。ビュー制約を宣言する場合は、キーワード DISABLE NOVALIDATE を指定する必要がある、他のモードは指定できません。
- *using_index_clause*、*exceptions_clause* 句、または *references_clause* の ON DELETE 句は指定できません。
- オブジェクト列の属性にビュー制約を定義することはできません。

例

一意キーの例 次の文は、サンプル表 sh.promotions を作成する文の例です。この文は制約を表内に定義し、promo_id 列で一意キーを暗黙的に使用可能にします（この例では、他の制約は省略されています）。

```
CREATE TABLE promotions_var1
( promo_id          NUMBER(6)
                                CONSTRAINT promo_id_u  UNIQUE
, promo_name        VARCHAR2(20)
, promo_category    VARCHAR2(15)
, promo_cost        NUMBER(10,2)
, promo_begin_date  DATE
, promo_end_date    DATE
) ;
```

制約 promo_id_u は、一意キーとして promo_id 列を識別します。この制約によって、表に同じ ID を持つ複数の販売促進の行が存在しないことが保証されます。ただし、識別子のない行は許可されます。

この制約を表外に定義して使用可能にすることもできます。

```
CREATE TABLE promotions_var2
( promo_id          NUMBER(6)
, promo_name        VARCHAR2(20)
, promo_category    VARCHAR2(15)
, promo_cost        NUMBER(10,2)
, promo_begin_date  DATE
, promo_end_date    DATE
, CONSTRAINT promo_id_u UNIQUE (promo_id)
USING INDEX PCTFREE 20
      TABLESPACE stocks
      STORAGE (INITIAL 8K NEXT 6K) );
```

この文には *using_index_clause* も含まれています。この句は、制約を使用可能にするために作成される索引の記憶特性を指定します。

複合一意キーの例 次の文は、oe.warehouses 表の warehouse_id 列と warehouse_name 列を組み合わせで複合一意キーを定義し、使用可能にします。

```
ALTER TABLE warehouses
  ADD CONSTRAINT wh_unq UNIQUE (warehouse_id, warehouse_name)
  USING INDEX PCTFREE 5
  EXCEPTIONS INTO wrong_id;
```

wh_unq 制約によって、warehouse_id と warehouse_name を組み合わせた値が表の中に複数存在しないことが保証されます。

この ADD CONSTRAINT 句には、制約以外のプロパティも指定できます。

- USING INDEX 句は、制約を使用可能にするために作成される索引の記憶特性を指定します。
- EXCEPTIONS INTO 句によって、制約に違反する行が warehouses 表に含まれている場合、その行に関する情報が wrong_id 表に書き込まれます。wrong_id 例外表が存在しない場合、この文は正常に実行されません。

主キーの例 次の文は、サンプル表 hr.locations を作成する文の例です。locations_demo 表を作成し、location_id 列に主キーを定義して使用可能にします（この例では、hr.locations 表の他の制約は省略されています）。

```
CREATE TABLE locations_demo
( location_id    NUMBER(4) CONSTRAINT loc_id_pk PRIMARY KEY
, street_address VARCHAR2(40)
, postal_code    VARCHAR2(12)
, city           VARCHAR2(30)
, state_province VARCHAR2(25)
, country_id     CHAR(2)
) ;
```

表内に指定されている loc_id_pk 制約は、location_id 列を locations_demo 表の主キーとして識別します。この制約によって、表の中の複数の所在地が同一の所在地識別子を持つことはなく、かつ所在地識別子が NULL にならないことが保証されます。

この制約を表外に定義して使用可能にすることもできます。

```
CREATE TABLE locations_demo
( location_id    NUMBER(4)
, street_address VARCHAR2(40)
, postal_code    VARCHAR2(12)
, city           VARCHAR2(30)
, state_province VARCHAR2(25)
, country_id     CHAR(2)
, CONSTRAINT loc_id_pk PRIMARY KEY (location_id));
```

NOT NULL の例 次の文は、(7-26 ページの「主キーの例」で作成した) `locations_demo` 表を変更し、`country_id` 列に NOT NULL 制約を定義して使用可能にします。

```
ALTER TABLE locations_demo
  MODIFY (country_id CONSTRAINT country_nn NOT NULL);
```

制約 `country_nn` によって、`country_id` が NULL の所在地の行が表にないことが保証されます。

複合主キーの例 次の文は、`sh.sales` サンプル表の `prod_id` 列および `cust_id` 列を組み合わせて複合主キーを定義します。

```
ALTER TABLE sales
  ADD CONSTRAINT sales_pk PRIMARY KEY (prod_id, cust_id) DISABLE;
```

この制約は、`sales` 表の主キーとして `prod_id` 列と `cust_id` 列の組合せを識別します。この制約によって、表の中の複数の行が `prod_id` 列と `cust_id` 列の両方に対して同じ値を持たないことが保証されます。

この制約句 (PRIMARY KEY) では、次の制約のプロパティも指定しています。

- 制約定義によって制約名が指定されていないため、この制約に対する名前が Oracle によって自動的に生成されます。
- DISABLE 句によって制約が定義されますが、使用可能にはなりません。

外部キー制約の例 次の文は、`dept_20` 表を作成し、`departments` 表の `department_id` 列の主キーを参照する `department_id` 列に、外部キーを定義して使用可能にします。

```
CREATE TABLE dept_20
  (employee_id    NUMBER(4),
   last_name      VARCHAR2(10),
   job_id         VARCHAR2(9),
   manager_id     NUMBER(4),
   hire_date      DATE,
   salary         NUMBER(7,2),
   commission_pct NUMBER(7,2),
   department_id  CONSTRAINT fk_deptno
                 REFERENCES departments(department_id) );
```

`fk_deptno` 制約によって、`dept_20` 表の従業員が属しているすべての部門が `departments` 表に含まれることが保証されます。ただし、部門番号が NULL 値の従業員 (どの部門にも属さない従業員) がいてもかまいません。すべての従業員がいずれかの部門に割り当てられるようにするには、`dept_20` 表の `department_id` 列に対して、REFERENCES 制約の他に NOT NULL 制約を作成します。

この制約を定義して使用可能にする前に、departments 表の department_id 列を主キーまたは一意キーとして指定する制約を定義して使用可能にする必要があります。

制約が表内に定義されているため、この外部キー制約定義は FOREIGN KEY 句を使用しません。また、この列には参照キーのデータ型が自動的に割り当てられるため、department_id 列のデータ型は必要ありません。

制約定義によって親表と参照キーの列の両方が指定されます。参照キーは親表の主キーであるため、参照キーの列名の指定は任意です。

この外部キー制約を表外に定義することもできます。

```
CREATE TABLE dept_20
  (employee_id      NUMBER(4),
   last_name        VARCHAR2(10),
   job_id           VARCHAR2(9),
   manager_id       NUMBER(4),
   hire_date        DATE,
   salary           NUMBER(7,2),
   commission_pct   NUMBER(7,2),
   department_id,
  CONSTRAINT fk_deptno
    FOREIGN KEY (department_id)
      REFERENCES departments(department_id) );
```

これらの外部キー定義は、両方とも ON DELETE 句を指定していないため、従業員がいる部門は削除できません。

ON DELETE の例 次の文では、dept_20 表を作成し、2 つの参照整合性制約を定義して使用可能にした後、ON DELETE 句を使用します。

```
CREATE TABLE dept_20
  (employee_id      NUMBER(4) PRIMARY KEY,
   last_name        VARCHAR2(10),
   job_id           VARCHAR2(9),
   manager_id       NUMBER(4) CONSTRAINT fk_mgr
      REFERENCES employees ON DELETE SET NULL,
   hire_date        DATE,
   salary           NUMBER(7,2),
   commission_pct   NUMBER(7,2),
   department_id    NUMBER(2)  CONSTRAINT fk_deptno
      REFERENCES departments(department_id)
      ON DELETE CASCADE );
```

最初の ON DELETE 句によって、従業員番号 2332 の上司が employees 表から削除された場合、その上司の部下だった dept_20 表のすべての従業員の manager_id 値が NULL になります。

次の ON DELETE 句によって、departments 表の department_id 値が削除されると、これに依存する dept_20 表の行の department_id 値も同時に削除されます。たとえば、部門番号 20 が departments 表から削除されると、その部門の全従業員が dept_20 表から削除されます。

複合外部キー制約の例 次の文は、dept_20 表の employee_id 列および last_name 列を組み合わせて外部キーを定義して使用可能にします。

```
ALTER TABLE dept_20
  ADD CONSTRAINT fk_empid_empname
  FOREIGN KEY (employee_id, last_name)
  REFERENCES hr.employees(employee_id, last_name)
  EXCEPTIONS INTO wrong_id;
```

fk_empid_empname 制約によって、dept_20 表の中すべての従業員が、employees 表に存在する employee_id と last_name の組合せを持つことが保証されます。この制約を定義して使用可能にする前に、employees 表の employee_id 列と last_name 列の組合せを主キーまたは一意キーとして指定する制約を定義して使用可能にする必要があります。

EXCEPTIONS INTO 句によって、制約に違反する行が dept_20 表に含まれている場合、その行に関する情報が wrong_id 表に書き込まれます。wrong_id 例外表が存在しない場合、この文は常に行われません。

CHECK 制約の例 次の文は、divisions 表を作成し、その表の各列に CHECK 制約を定義します。

```
CREATE TABLE divisions
  (div_no    NUMBER CONSTRAINT check_divno
    CHECK (div_no BETWEEN 10 AND 99)
    DISABLE,
   div_name  VARCHAR2(9)  CONSTRAINT check_divname
    CHECK (div_name = UPPER(div_name))
    DISABLE,
   office    VARCHAR2(10) CONSTRAINT check_office
    CHECK (office IN ('DALLAS','BOSTON',
                     'PARIS','TOKYO'))
    DISABLE);
```

列に定義されている各制約によって、列の値が次のように制限されます。

- check_divno によって、部門番号は必ず 10 ～ 99 の範囲内になります。
- check_divame によって、部門名はすべて大文字になります。
- check_office によって、部門の所在地は Dallas、Boston、Paris または Tokyo のいずれかに制限されます。

それぞれの CONSTRAINT 句に DISABLE 句が指定されているため、これらの制約は定義されるのみで、使用可能にはされません。

次の文は、dept_20 表を作成し、CHECK 制約を表外に定義して暗黙的に使用可能にします。

```
CREATE TABLE dept_20
  (employee_id    NUMBER(4) PRIMARY KEY,
   last_name      VARCHAR2(10),
   job_id         VARCHAR2(9),
   manager_id     NUMBER(4),
   salary         NUMBER(7,2),
   commission_pct NUMBER(7,2),
   department_id  NUMBER(2),
   CONSTRAINT check_sal CHECK (salary * commission_pct <= 5000));
```

この制約は、不等式の条件を使用して、従業員の歩合総額 (salary と commission_pct を掛けた金額) を 5,000 ドルに制限します。

- 従業員の給与と歩合が NULL 以外の値の場合、制約を満たすには、この金額が 5,000 ドルを超えてはいけません。
- 従業員の給与または歩合が NULL 値の場合、条件の結果は不明となり、その従業員は自動的に制約を満たします。

この例の制約句には制約名が指定されていないため、制約の名前が自動的に生成されます。

次の文は、1 つの主キー制約、2 つの外部キー制約、1 つの NOT NULL 制約および 2 つの CHECK 制約を定義して使用可能にします。

```
CREATE TABLE order_detail
  (CONSTRAINT pk_od PRIMARY KEY (order_id, part_no),
   order_id    NUMBER
     CONSTRAINT fk_oid
       REFERENCES oe.orders(order_id),
   product_id  NUMBER
     CONSTRAINT fk_pno
       REFERENCES oe.product_information(product_id),
   quantity    NUMBER
     CONSTRAINT nn_qty NOT NULL
     CONSTRAINT check_qty CHECK (quantity > 0),
   cost        NUMBER
     CONSTRAINT check_cost CHECK (cost > 0) );
```

この制約によって、表のデータに対して次の規則を適用できます。

- pk_od は、order_id 列と product_id 列の組合せを表の主キーとして指定します。この制約を満たすためには、order_id 列および product_id 列の組合せが同じである行が、表内に 2 つあってはいけません。また、order_id 列および product_id 列では、行に NULL を入れることはできません。

- `fk_oid` は、サンプル・スキーマ `oe` 内の `orders` 表にある `order_id` 列を参照する外部キーとして、`order_id` 列を指定します。`order_detail.order_id` 列に追加されるすべての新しい値は、`oe.orders.order_id` 列にあらかじめ存在する必要があります。
- `fk_pno` は、`oe` が所有する `product_information` 表にある `product_id` 列を参照する外部キーとして、`product_id` 列を指定します。`order_detail.product_id` 列に追加されるすべての新しい値は、`oe.product_information.product_id` 列にあらかじめ存在する必要があります。
- `nn_qty` は、`quantity` 列に対して `NULL` を禁止します。
- `check_qty` によって、`quantity` 列の値は必ず 0（ゼロ）より大きくなります。
- `check_cost` によって、`cost` 列の値は必ず 0（ゼロ）より大きくなります。

この例は、制約句と列定義について、次の点についても示しています。

- 表外制約定義は、列定義の前と後のどちらにも指定できます。この例では、`pk_od` 制約の表外定義が、列定義の前にあります。
- 列定義には、複数の表内制約定義を含めることができます。この例では、`quantity` 列の定義は `nn_qty` 制約と `check_qty` 制約の両方の定義を含んでいます。
- 表には、複数の `CHECK` 制約を指定できます。複数のビジネス・ルールを適用する複雑な条件を持つ 1 つの `CHECK` 制約よりも、それぞれ 1 つのビジネス・ルールのみを適用する単純な条件を持つ複数の `CHECK` 制約を使用してください。矛盾している制約があると、その制約を識別するエラー・メッセージが戻されます。エラーが検出された制約が 1 つのビジネス・ルールのみを有効にする場合、このようなエラー・メッセージの方が、矛盾のあるビジネス・ルールを正確に識別できます。

属性レベル制約の例 次の文は、`students` 表の `name` 列の `first_name` 属性と `last_name` 属性の両方に対して値が存在することを保証します。

```
CREATE TYPE person_name AS OBJECT
    (first_name VARCHAR2(30), last_name VARCHAR2(30));
/
```

```
CREATE TABLE students (name person_name, age INTEGER,
    CHECK (name.first_name IS NOT NULL AND
           name.last_name IS NOT NULL));
```

REF 制約の例 次の例は、サンプル・スキーマのオブジェクト型である `cust_address_typ` の複製を作成し、`SCOPE` 制約が指定された `REF` 列を含む表を作成します。

```
CREATE TYPE cust_address_typ_new AS OBJECT
    ( street_address    VARCHAR2(40)
      , postal_code      VARCHAR2(10)
```

```
        , city                VARCHAR2(30)
        , state_province      VARCHAR2(10)
        , country_id          CHAR(2)
      );
/
CREATE TABLE address_table OF cust_address_typ_new;

CREATE TABLE customer_addresses (
  add_id NUMBER,
  address REF cust_address_typ_new
  SCOPE IS address_table);
```

次の例は、同じ表を作成しますが、親表の `OID` 列を参照する `REF` 列に参照整合性制約が指定されています。

```
CREATE TABLE customer_addresses (
  add_id NUMBER,
  address REF cust_address_typ REFERENCES address_table);
```

次の例では、`department_typ` 型および `departments_obj_t` 表（15-73 ページの「[オブジェクト表の作成例:](#)」で作成）を使用します。この文を実行すると、有効範囲付き `REF` を持つ表が作成されます。

```
CREATE TABLE employees_obj
( e_name  VARCHAR2(100),
  e_number NUMBER,
  e_dept  REF department_typ SCOPE IS departments_obj_t );
```

次の文は、参照整合性制約が定義されている `REF` 列を含む表を作成します。

```
CREATE TABLE employees_obj
( e_name  VARCHAR2(100),
  e_number NUMBER,
  e_dept  REF department_typ REFERENCES departments_obj_t);
```

明示的な索引の制御例 次の文は、Oracle が制約を適用する場合に使用する索引を、明示的に制御する一意制約または主キー制約を作成する別の方法を示します。

```
CREATE TABLE promotions_var3
( promo_id      NUMBER(6)
, promo_name    VARCHAR2(20)
, promo_category VARCHAR2(15)
, promo_cost    NUMBER(10,2)
, promo_begin_date DATE
, promo_end_date DATE
, CONSTRAINT promo_id_u UNIQUE (promo_id, promo_cost)
  USING INDEX (CREATE UNIQUE INDEX promo_ix1
    ON promotions_var3 (promo_id, promo_cost))
```

```
, CONSTRAINT promo_id_u2 UNIQUE (promo_cost, promo_id)
  USING INDEX promo_ix1);
```

この例は、1つの制約に対して1つの索引を作成し、その索引を使用して同じ文に別の制約を作成して使用可能にできることも示しています。

DEFERRABLE 制約の例 次の文は、scores 列に CHECK 制約 NOT DEFERRABLE INITIALLY IMMEDIATE (デフォルト) を使用した games 表を作成します。

```
CREATE TABLE games (scores NUMBER CHECK (scores >= 0));
```

次の文は、列に対する一意制約を INITIALLY DEFERRED DEFERRABLE として定義します。

```
CREATE TABLE games
(scores NUMBER, CONSTRAINT unq_num UNIQUE (scores)
INITIALLY DEFERRED DEFERRABLE);
```

deallocate_unused_clause

用途

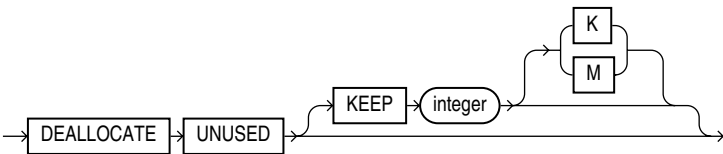
deallocate_unused_clauseを使用すると、データベース・オブジェクト・セグメントの終わりにある未使用領域の割当てを明示的に解除し、解放された領域を表領域の他のセグメントで使えるようになります。

未使用領域の割当ては、次の文で解除できます。

- ALTER CLUSTER (9-7 ページの「ALTER CLUSTER」を参照)
- ALTER INDEX: 未使用領域の割当てを索引、索引パーティションまたは索引サブパーティションから解除する場合 (9-61 ページの「ALTER INDEX」を参照)
- ALTER MATERIALIZED VIEW: 未使用領域の割当てを索引構成マテリアライズド・ビューのオーバーフロー・セグメントから解除する場合 (9-88 ページの「ALTER MATERIALIZED VIEW」を参照)
- ALTER TABLE: 未使用領域の割当てを表、表パーティション、表サブパーティション、索引構成表のマッピング表、索引構成表のオーバーフロー・セグメントまたは LOB 記憶域セグメントから解除する場合 (11-2 ページの「ALTER TABLE」を参照)

構文

deallocate_unused_clause::=



セマンティクス

この項では、deallocate_unused_clause のセマンティクスについて説明します。詳細は、特定のデータベース・オブジェクトに対してこの句を設定または再設定する SQL 文の説明を参照してください。

注意： 同じ文で deallocate_unused_clause と allocate_extent_clause の両方を指定することはできません。

最高水位標を超える（データベース・ブロックがデータを受け取るためにフォーマットされていない）未使用領域のみ解放できます。未使用領域の割当て解除は、オブジェクトの終わりから開始し、オブジェクトの先頭にある最高水位標に向かって進行します。

エクステンツが割当て解除の範囲内に完全に含まれる場合、エクステンツ全体が解放されて再利用可能となります。エクステンツの一部が含まれる場合、最高水位標までのすでに使用されている部分がエクステンツになり、残りの未使用領域が解放されて再利用可能になります。

割当てが解除される表領域のユーザー割当て制限は、解放される領域の量のみ増加します。

解放される領域の実際の量は、INITIAL、MINEXTENTS および NEXT 記憶域パラメータによって異なります。

参照： これらのパラメータについては、7-51 ページの「[storage_clause](#)」を参照してください。

KEEP integer

割当て解除後に、データベース・オブジェクトのセグメントが保持する、最高水位標を超えるバイト数を指定します。

- KEEP を指定しなかった場合に、最高水位標が INITIAL および MINEXTENTS のサイズを超えると、最高水位標を超えるすべての未使用領域が解放されます。最高水位標が INITIAL または MINEXTENTS のサイズより小さい場合は、MINEXTENTS をを超えるすべての未使用領域が解放されます。
- KEEP オプションを指定した場合、指定した量の領域が保持され、残りの領域のみが解放されます。このとき、残りのエクステンツ数が MINEXTENTS より小さくなった場合、MINEXTENTS はそのエクステンツ数に変更されます。また、初期エクステンツが INITIAL より小さくなった場合、INITIAL がそのサイズに変更されます。
- どちらの場合にも、NEXT 記憶域パラメータの値は、割当て解除された最後のエクステンツのサイズに設定されます。

file_specification

用途

file_specification 構文のいずれかを使用すると、ファイルをデータ・ファイルまたは一時ファイルとして指定できます。また、1つ以上のファイルのグループを1つの REDO ログ・ファイル・グループとして指定することもできます。

file_specification は次の文に指定できます。

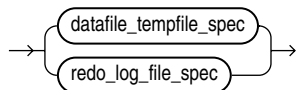
- CREATE CONTROLFILE (13-14 ページの「[CREATE CONTROLFILE](#)」を参照)
- CREATE DATABASE (13-22 ページの「[CREATE DATABASE](#)」を参照)
- ALTER DATABASE (9-12 ページの「[ALTER DATABASE](#)」を参照)
- CREATE TABLESPACE (15-75 ページの「[CREATE TABLESPACE](#)」を参照)
- CREATE TEMPORARY TABLESPACE (15-86 ページの「[CREATE TEMPORARY TABLESPACE](#)」を参照)
- ALTER TABLESPACE (11-97 ページの「[ALTER TABLESPACE](#)」を参照)

前提条件

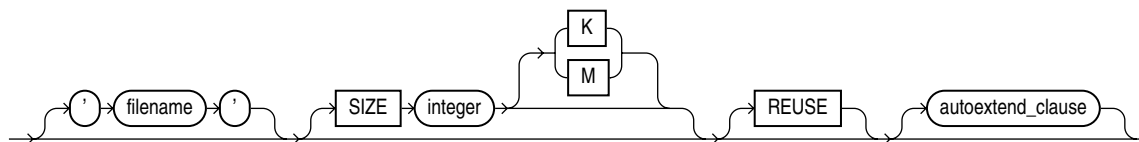
前述のいずれかの文を発行できる権限が必要です。

構文

file_specification::=



datafile_tempfile_spec::=



```

graph LR
    Entry(( )) --> L1(( ))
    L1 --> F1(filename)
    F1 --> L2(( ))
    L2 --> F2(filename)
    F2 --> L3(( ))
    L3 --> S1(SIZE)
    S1 --> I1(integer)
    I1 --> L4(( ))
    L4 --> K1(K)
    K1 --> M1(M)
    M1 --> L5(( ))
    L5 --> R1(REUSE)
    R1 --> L1
  
```

```

graph LR
    Input(( )) --> AUTOEXTEND[AUTOEXTEND]
    AUTOEXTEND --> OFF[OFF]
    OFF --> Input
    AUTOEXTEND --> ON[ON]
    ON --> NEXT[NEXT]
    NEXT --> integer(integer)
    integer --> K[K]
    integer --> M[M]
    K --> M
    M --> maxsize_clause(maxsize_clause)
    maxsize_clause --> Input
  
```

この項では、`file_specification` のセマンティクスについて説明します。詳細は、データ・ファイル、一時ファイルまたは REDO ログ・ファイルを指定する SQL 文の説明を参照してください。

新しいファイルの場合、`filename`はそのファイルの名前になります。Oracle 管理ファイルを使用していない場合は、`filename`を指定しなければ、文は正常に実行されません。ただし、Oracle 管理ファイルを使用している場合は、`filename`や残りの句は必須ではありません。この場合、ファイルに一意の名前が作成され、そのファイルが `DB_CREATE_FILE_DEST` 初期化パラメータ（すべてのファイル・タイプに有効）または `DB_CREATE_ONLINE_LOG_DEST_n` 初期化パラメータ（ログ・ファイルに対しては `DB_CREATE_FILE_DEST` より優先される）で指定されるディレクトリに保存されます。

参照： Oracle 管理ファイルの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。また、7-40 ページの「[データ・ファイルの指定例](#)」および 7-40 ページの「[ログ・ファイルの指定例](#)」も参照してください。

既存のファイルの場合は、ファイル名を指定する必要があります。データ・ファイル、一時ファイルまたは REDO ログ・ファイル・メンバーの名前を指定します。*filename* には、7 ビット ASCII キャラクタ・セットまたは EBCDIC キャラクタ・セットのシングルバイト文字のみを使用できます。マルチバイト文字は無効です。

REDO ログ・ファイル・グループは、1 つ以上のメンバー（コピー）で構成されます。*filename* は、使用するオペレーティング・システムの表記規則に従って、完全な名前で指定する必要があります。

SIZE 句

ファイルのサイズをバイト単位で指定します。K または M を使用して、KB または MB 単位で指定することもできます。

- UNDO 表領域の場合、各データ・ファイルに対して SIZE 句を指定する必要があります。その他の表領域の場合、ファイルがすでに存在するとき、または Oracle 管理ファイルを作成するときには、このパラメータを省略できます。
- Oracle 管理ファイルの作成時にこの句を指定しないと、100MB のファイルが作成されます。
- 表領域は、中に含まれるオブジェクトのサイズの合計より 1 ブロック大きいサイズである必要があります。

参照： 自動 UNDO 管理および UNDO 表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。また、7-40 ページの「[ログ・ファイルの追加例](#)」も参照してください。

REUSE

REUSE は、既存ファイルの再利用を可能にします。すでに存在している *filename* を指定する場合は、REUSE を指定する必要があります。

- ファイルがすでに存在する場合、ファイル名が再利用され、新しいサイズが適用されるか（SIZE を指定する場合）、または元のサイズのままとなります。
- ファイルが存在していない場合、この句は無視され、ファイルが作成されます。

REUSE 句の制限事項： *filename* を指定しないかぎり、REUSE は指定できません。

注意： 既存のファイルが使用される場合は、そのファイルに入っていた内容は失われます。

参照： 7-41 ページの「データ・ファイルの追加例」および7-40 ページの「ログ・ファイルの追加例」を参照してください。

autoextend_clause

autoextend_clause は、新しいデータ・ファイルまたは一時ファイルの自動拡張を使用可能または使用禁止にします。この句を指定しない場合、次のようになります。

- Oracle 管理ファイルの場合
 - SIZE を指定すると、指定したサイズのファイルが作成され、AUTOEXTEND が使用禁止になります。
 - SIZE を指定しないと、100MB のファイルが作成され、AUTOEXTEND が使用可能になり、MAXSIZE が無制限になります。
- ユーザー管理ファイルの場合、SIZE の指定 / 未指定にかかわらず、AUTOEXTEND が使用禁止にされたファイルが作成されます。

ON ON を指定すると、自動拡張を使用可能にします。

OFF OFF を指定すると、自動拡張を使用禁止にします。

注意： 自動拡張を使用禁止にする場合は、NEXT および MAXSIZE の値を 0（ゼロ）に設定します。後続の文で自動拡張を使用可能に戻す場合は、これらの値を設定しなおす必要があります。

NEXT NEXT 句を使用すると、エクステン트가さらに必要になった場合にデータ・ファイルに自動的に割り当てられるディスク領域の増分サイズを、バイト単位で指定できます。K または M を使用すると、KB または MB 単位で指定できます。デフォルトのサイズは 1 データ・ブロックです。

MAXSIZE MAXSIZE 句を使用すると、データ・ファイルの自動拡張で使用されるディスク領域の最大サイズを指定できます。

UNLIMITED データ・ファイルまたは一時ファイルに割り当てられるディスク領域を制限しない場合は、UNLIMITED 句を使用します。

***autoextend_clause* の制限事項：** この句は CREATE CONTROLFILE 文の *datafile tempfile_spec* の一部、または ALTER DATABASE CREATE DATAFILE 句には指定できません。

例

ログ・ファイルの指定例 次の文は、payable という名前のデータベースを作成します。このデータベースには各グループにメンバーを 2 つ持つ REDO ログ・ファイル・グループが 2 つと、データ・ファイルが 1 つ設定されています。

```
CREATE DATABASE payable
  LOGFILE GROUP 1 ('diska:log1.log', 'diskb:log1.log') SIZE 50K,
  GROUP 2 ('diska:log2.log', 'diskb:log2.log') SIZE 50K
  DATAFILE 'diskc:dbone.dat' SIZE 30M;
```

LOGFILE 句の最初のファイル指定は、REDO ログ・ファイル・グループに GROUP 1 の値を指定します。このグループには、'diska:log1.log' および 'diskb:log1.log' というメンバーがあり、サイズはそれぞれ 50KB です。

LOGFILE 句の 2 番目のファイル指定は、REDO ログ・ファイル・グループに GROUP 2 の値を指定します。このグループには、'diska:log2.log' および 'diskb:log2.log' というメンバーがあり、サイズはそれぞれ 50KB です。

DATAFILE 句のファイル指定では、'diskc:dbone.dat' という名前のデータ・ファイルが指定されています。このファイルのサイズは 30MB です。

各ファイル指定は SIZE パラメータの値を指定し、REUSE 句は指定していないため、指定のファイルがすでに定義されていることはありません。Oracle が新しくファイルを作成します。

ログ・ファイルの追加例 次の文は、2 つのメンバーで構成される新しい REDO ログ・ファイル・グループを、payable データベースに追加します。

```
ALTER DATABASE payable
  ADD LOGFILE GROUP 3 ('diska:log3.log', 'diskb:log3.log')
  SIZE 50K REUSE;
```

ADD LOGFILE 句のファイル指定は、REDO ログ・ファイル・グループに GROUP 3 の値を指定します。このグループには、'diska:log3.log' および 'diskb:log3.log' というメンバーがあり、サイズはそれぞれ 50KB です。このファイル指定では REUSE 句が指定されているため、各メンバーはすでに存在している可能性があります（ただし、存在していなくてもかまいません）。

データ・ファイルの指定例 次の文は、stocks という名前の表領域を作成します。また、この表領域にはデータ・ファイルが 3 つあります。

```
CREATE TABLESPACE stocks
  DATAFILE 'stock1.dat' SIZE 10M,
  'stock2.dat' SIZE 10M,
  'stock3.dat' SIZE 10M;
```

このデータ・ファイルのファイル指定は、'diskc:stock1.dat'、
'diskc:stock2.dat' および 'diskc:stock3.dat' という名前のファイルを指定します。

データ・ファイルの追加例 次の文は、stocks 表領域を変更し、新しいデータ・ファイルを追加します。

```
ALTER TABLESPACE stocks  
  ADD DATAFILE 'stock4.dat' SIZE 10M REUSE;
```

このファイル指定は、'diskc:stock4.dat' という名前のデータ・ファイルを指定します。このファイル名が存在しない場合、REUSE キーワードは無視されます。

logging_clause

用途

`logging_clause` を使用すると、データベース・オブジェクトの作成が REDO ログ・ファイルに記録されるか (LOGGING)、記録されないか (NOLOGGING) を指定できます。

この句は、オブジェクトに対して後で実行されるダイレクト・ローダー (SQL*Loader) 操作およびダイレクト・パス・インサート操作を記録するか (LOGGING)、または記録しないか (NOLOGGING) も指定します。

`logging_clause` は次の文に指定できます。

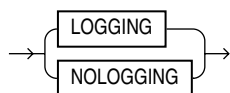
- CREATE TABLE および ALTER TABLE: 表、表のパーティションの 1 つ、LOB セグメント、または索引構成表のオーバーフロー・セグメントのログを記録する場合 (15-7 ページの「[CREATE TABLE](#)」および 11-2 ページの「[ALTER TABLE](#)」を参照)
- CREATE INDEX および ALTER INDEX: 索引または索引のパーティションの 1 つのログを記録する場合 (13-61 ページの「[CREATE INDEX](#)」および 9-61 ページの「[ALTER INDEX](#)」を参照)
- CREATE MATERIALIZED VIEW および ALTER MATERIALIZED VIEW: マテリアライズド・ビュー、マテリアライズド・ビューのパーティションの 1 つまたは LOB セグメントのログを記録する場合 (14-4 ページの「[CREATE MATERIALIZED VIEW](#)」および 9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照)
- CREATE MATERIALIZED VIEW LOG および ALTER MATERIALIZED VIEW LOG: マテリアライズド・ビューまたはマテリアライズド・ビュー・ログのパーティションの 1 つのログを記録する場合 (14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」および 9-108 ページの「[ALTER MATERIALIZED VIEW LOG](#)」を参照)
- CREATE TABLESPACE および ALTER TABLESPACE: 表領域に作成されたすべてのオブジェクトに対するデフォルトのロギング特性を設定または変更する場合 (15-75 ページの「[CREATE TABLESPACE](#)」および 11-97 ページの「[ALTER TABLESPACE](#)」を参照)

次の操作にも、LOGGING または NOLOGGING を指定できます。

- 索引の再構築 (CREATE INDEX ... REBUILD を使用)
- 表の移動 (ALTER TABLE ... MOVE を使用)

構文

`logging_clause ::=`



セマンティクス

この項では、*logging_clause* のセマンティクスについて説明します。詳細は、特定のデータベース・オブジェクトに対してロギング特性を設定または再設定する SQL 文の説明を参照してください。

データベース・オブジェクトの作成、およびその後のオブジェクトへの挿入を REDO ログ・ファイルに記録する場合は、LOGGING を指定します。

これらの操作を記録しない場合は、NOLOGGING を指定します。

- **非パーティション・オブジェクト**の場合、この句に指定する値は、オブジェクトに関連付けられたセグメントの実際の物理属性となります。
- **パーティション・オブジェクト**の場合、この句に指定する値は、PARTITION 記述でロギング属性を指定しないかぎり、CREATE 文（および後続の ALTER ... ADD PARTITION 文）で指定するすべてのパーティションに関連付けられたセグメントのデフォルトの物理属性となります。

ロギング属性を指定しているオブジェクトが強制ロギング・モードのデータベースまたは表領域に存在している場合、そのデータベースまたは表領域が強制ロギング・モードから別のモードに変わるまで、NOLOGGING 設定は無視されます。

データベースを ARCHIVELOG モードで運用する場合、LOGGING 操作の前に取ったバックアップからのメディア・リカバリによって、オブジェクトが再作成されます。ただし、NOLOGGING 操作の前に取ったバックアップからのメディア・リカバリでは、オブジェクトは再作成されません。

NOLOGGING モードでの操作で生成される REDO ログのサイズは、LOGGING モードで生成されるログより非常に小さくなります。

NOLOGGING モードでは、データの変更時に、（新しいエクステン트를 INVALID としてマーク設定し、ディクショナリの変更を記録するために）最小限のログが記録されます。メディア・リカバリ中に NOLOGGING が適用された場合、REDO データのログ記録が中断されるため、エクステン트無効化レコードでは、一定のブロック範囲に「論理的に無効」というマークが付きます。このため、損失してはならないデータベース・オブジェクトの場合は、NOLOGGING 操作の後にバックアップを取る必要があります。

NOLOGGING は、LOGGING をサポートする句のサブセットのみでサポートされます。次の操作でのみ、NOLOGGING モードがサポートされます。

DML

- ダイレクト・パス・インサート（シリアルまたはパラレル）
- ダイレクト・ローダー（SQL*Loader）

DDL

- CREATE TABLE ... AS SELECT

- CREATE TABLE ... *LOB_storage_clause* ... *LOB_parameters* ... NOCACHE | CACHE READS
- ALTER TABLE ... *LOB_storage_clause* ... *LOB_parameters* ... NOCACHE | CACHE READS (新しく作成された LOB 列のロギングを指定する場合)
- ALTER TABLE ... *modify_LOB_storage_clause* ... *modify_LOB_parameters* ... NOCACHE | CACHE READS (既存の LOB 列のロギングを変更する場合)
- ALTER TABLE ... MOVE
- ALTER TABLE ... [データ移動を伴うすべてのパーティション操作]
 - ALTER TABLE ... ADD PARTITION (ハッシュ・パーティションのみ)
 - ALTER TABLE ... MERGE PARTITIONS
 - ALTER TABLE ... SPLIT PARTITION
 - ALTER TABLE ... MOVE PARTITION
 - ALTER TABLE ... MODIFY PARTITION ... ADD SUBPARTITION
 - ALTER TABLE ... MODIFY PARTITION ... COALESCE SUBPARTITION
 - ALTER TABLE ... MODIFY PARTITION ... REBUILD UNUSABLE INDEXES
- CREATE INDEX
- ALTER INDEX ... REBUILD
- ALTER INDEX ... REBUILD PARTITION
- ALTER INDEX ... SPLIT PARTITION

LOB 以外のオブジェクトの場合、この句を指定しないと、オブジェクトが存在する表領域のロギング属性がオブジェクトのデフォルトのロギング属性になります。

LOB に対してこの句を省略した場合は、次のようになります。

- **CACHE** を指定した場合は、**LOGGING** が使用されます (**CACHE NOLOGGING** は指定できないため)。
- **NOCACHE** または **CACHE READS** を指定した場合は、表が存在する表領域の属性がデフォルトのロギング属性として使用されます。

NOLOGGING は、行データとともにインラインに格納された **LOB** に適用されません。つまり、**LOB** に対する **NOLOGGING** を 4000 バイト未満の値に指定し、**STORAGE IN ROW** を使用禁止にしていなかった場合、**NOLOGGING** の指定は無視され、**LOB** データは他の表データと同様に扱われます。

参照： ロギングおよびパラレル DML の詳細は、『Oracle9i データベース概要』および『Oracle9i データベース管理者ガイド』を参照してください。

parallel_clause

用途

`parallel_clause`を使用すると、データベース・オブジェクト作成の平行化、およびその後のオブジェクトに対する問合せのデフォルトの並列度と、オブジェクトに対するDML操作の設定が可能になります。

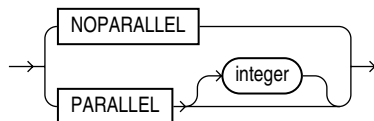
`parallel_clause`は次の文に指定できます。

- CREATE TABLE: 表の平行化を設定する場合（15-7 ページの「[CREATE TABLE](#)」を参照）
- ALTER TABLE（11-2 ページの「[ALTER TABLE](#)」を参照）
 - 表の平行化を変更する場合
 - 表パーティションの追加、結合、交換、マージ、分割、切捨て、削除または移動の操作を平行化する場合
- CREATE CLUSTER および ALTER CLUSTER: クラスタの平行化を設定または変更する場合（13-2 ページの「[CREATE CLUSTER](#)」および 9-7 ページの「[ALTER CLUSTER](#)」を参照）
- CREATE INDEX: 索引の平行化を設定する場合（13-61 ページの「[CREATE INDEX](#)」を参照）
- ALTER INDEX（9-61 ページの「[ALTER INDEX](#)」を参照）
 - 表の平行化を変更する場合
 - 索引の再構築または索引パーティションの分割を平行化する場合
- CREATE MATERIALIZED VIEW: マテリアライズド・ビューの平行化を設定する場合（14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照）
- ALTER MATERIALIZED VIEW（9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照）
 - マテリアライズド・ビューの平行化を変更する場合
 - マテリアライズド・ビュー・パーティションの追加、結合、交換、マージ、分割、切捨て、削除または移動の操作を平行化する場合
 - マテリアライズド・ビュー・サブパーティションの追加または移動の操作を平行化する場合
- CREATE MATERIALIZED VIEW LOG: 表の平行化を設定する場合（14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」を参照）

- ALTER MATERIALIZED VIEW LOG (9-108 ページの「[ALTER MATERIALIZED VIEW LOG](#)」を参照)
 - マテリアライズド・ビューの平行化を変更する場合
 - マテリアライズド・ビュー・パーティションの追加、結合、交換、マージ、分割、切捨て、削除または移動の操作を平行化する場合
- ALTER DATABASE ... RECOVER: データベースをリカバリする場合 (9-12 ページの「[ALTER DATABASE](#)」を参照)
- ALTER DATABASE ... *standby_database_clauses*: スタンバイ・データベースに対する操作を平行化する場合 (9-12 ページの「[ALTER DATABASE](#)」を参照)

構文

parallel_clause::=



セマンティクス

この項では、`parallel_clause` のセマンティクスについて説明します。詳細は、特定のデータベース・オブジェクトまたは操作に対して平行化を設定または再設定する SQL 文の説明を参照してください。

注意： `parallel_clause` 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL `NOPARALLEL` を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、`PARALLEL` を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ `PARALLEL_THREADS_PER_CPU` の値を掛けたものです。

PARALLEL integer *integer* には、平行操作で使用する平行・スレッド数である **並列度** を指定します。各平行・スレッドは、1、2 個の平行実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

physical_attributes_clause

用途

`physical_attributes_clause`を使用すると、PCTFREE、PCTUSED、INITTRANS および MAXTRANS パラメータの値、および表、クラスタ、索引またはマテリアライズド・ビューの記憶特性を指定できます。

`physical_attributes_clause` は次の文に指定できます。

- CREATE CLUSTER および ALTER CLUSTER: クラスタおよびクラスタ内のすべての表の物理属性を設定または変更する場合（13-2 ページの [CREATE CLUSTER](#) および 9-7 ページの [ALTER CLUSTER](#)」を参照）
- CREATE TABLE: 表、表パーティション、オブジェクト表の OIDINDEX または索引構成表のオーバーフロー・セグメントの物理属性を設定する場合（15-7 ページの [CREATE TABLE](#)」を参照）
- ALTER TABLE: 表の物理属性、将来追加される表パーティションのデフォルトの物理属性、または既存の表パーティションの物理属性を変更する場合（11-2 ページの [ALTER TABLE](#)」を参照）

注意：

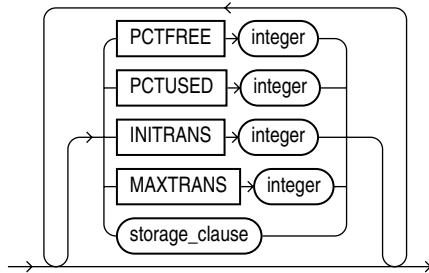
- 物理属性は、一時表には指定できません。
 - 物理属性は、クラスタ化表には指定できません。クラスタ内の表はクラスタの物理属性を継承します。
-

- CREATE INDEX: 索引または索引パーティションの物理属性を設定する場合（13-61 ページの [CREATE INDEX](#)」を参照）
- ALTER INDEX: 索引の物理属性、将来追加されるパーティションのデフォルトの物理属性、または既存の索引パーティションの物理属性を変更する場合（9-61 ページの [ALTER INDEX](#)」を参照）
- CREATE MATERIALIZED VIEW: マテリアライズド・ビュー、そのパーティションの 1 つ、またはマテリアライズド・ビューを保持するために生成される索引の物理属性を設定する場合（14-4 ページの [CREATE MATERIALIZED VIEW](#)」を参照）
- ALTER MATERIALIZED VIEW: マテリアライズド・ビューの物理属性、将来追加されるパーティションのデフォルトの物理属性、および既存のパーティション、またはマテリアライズド・ビューを保持するために作成される索引の物理属性を変更する場合（9-88 ページの [ALTER MATERIALIZED VIEW](#)」を参照）

- CREATE MATERIALIZED VIEW LOG および ALTER MATERIALIZED VIEW LOG: マテリアライズド・ビュー・ログの物理属性を設定または変更する場合 (14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」および 9-108 ページの「[ALTER MATERIALIZED VIEW LOG](#)」を参照)

構文

physical_attributes_clause::=



(7-51 ページの「[storage_clause](#)」を参照)

セマンティクス

この項では、*physical_attributes_clause* のパラメータについて説明します。詳細は、特定のデータベース・オブジェクトに対してこれらのパラメータを設定または再設定する SQL 文の説明を参照してください。

PCTFREE integer

データベース・オブジェクトの各データ・ブロック内で、オブジェクトの行を将来更新するために確保しておく領域の割合を表す整数値を指定します。PCTFREE の値は、0 ～ 99 の値にする必要があります。値に 0 を指定した場合は、ブロック全体が一杯になるまで新しい行を挿入できます。デフォルト値は 10 です。10 を指定した場合、既存の行に対して行われる更新用に各ブロックの 10% が確保されるため、各ブロックでは最大 90% まで表に新しい行を挿入できます。

PCTFREE は、表、パーティション、クラスタ、索引、マテリアライズド・ビューおよびマテリアライズド・ビュー・ログを作成および変更する文の中で同様に機能します。PCTFREE と PCTUSED の組合せによって、新しい行を既存のデータ・ブロックと新しいデータ・ブロックのどちらに挿入するかが決まります。

PCTFREE 句の制限事項： 索引を変更する場合、このパラメータは、*modify_index_default_attrs* 句および *split_partition_clause* にのみ指定できます。

PCTUSED integer

使用済領域のうち、データベース・オブジェクトのデータ・ブロックごとに確保される最小限の割合を表す整数値を指定します。ブロックは、使用済領域が PCTUSED の値を下回ると、行挿入の対象となります。PCTUSED は 0 ～ 99 までの正の整数で指定し、デフォルト値は 40 です。

PCTUSED は、表、パーティション、クラスタ、マテリアライズド・ビューおよびマテリアライズド・ビュー・ログを作成および変更する文の中で同様に機能します。

PCTUSED は、索引構成表 (ORGANIZATION INDEX) には無効な表記憶特性です。

PCTFREE および PCTUSED の合計は 100 以下である必要があります。PCTFREE と PCTUSED をともに使用して、表内の領域を効果的に利用できます。

PCTUSED 句の制限事項： このパラメータは、索引または索引構成表の索引セグメントには指定できません。

参照： PCTUSED および PCTFREE の各値によるパフォーマンスへの効果については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

INITRANS integer

データベース・オブジェクトに割り当てられた各データ・ブロックに割り当てられる、同時実行トランザクション・エントリの初期数を指定します。この値の範囲は 1 ～ 255 で、デフォルト値は 1 ですが、次の例外があります。

- クラスタまたは索引の INITRANS のデフォルト値は、クラスタの表領域の INITRANS のデフォルト値と 2 のいずれか大きい方の値です。
- 索引のデフォルト値は 2 です。

通常、INITRANS 値は、変更せずにデフォルトのまま使用してください。

ブロックを更新するトランザクションごとに、ブロックのトランザクション・エントリが必要です。トランザクション・エントリのサイズは、ご使用のオペレーティング・システムによって異なります。

このパラメータを指定した場合、最小数の同時実行トランザクションでブロックを更新することができます。さらに、トランザクション・エントリを動的に割り当てるときのオーバーヘッドを回避できます。

INITRANS パラメータは、表、パーティション、クラスタ、索引、マテリアライズド・ビューおよびマテリアライズド・ビュー・ログを作成および変更する文の中で同様に機能します。

MAXTRANS integer

データベース・オブジェクトに割り当てられたデータ・ブロックを更新する同時実行トランザクションの最大数を指定します。この制限は問合せには適用されません。この値の範囲は 1 ～ 255 で、デフォルト値はデータ・ブロック・サイズにより異なります。MAXTRANS 値は、変更せずにデフォルトのまま使用してください。

ブロックを同時に更新するトランザクション数が INITTRANS 値を超えると、MAXTRANS 値を超えるまで、またはブロックの空き領域がなくなるまで、そのブロックにトランザクション・エントリが動的に割り当てられます。

MAXTRANS パラメータは、PARTITION 記述、クラスタ、索引、マテリアライズド・ビューおよびマテリアライズド・ビュー・ログの中では、表の場合と同じ働きをします。

storage_clause *storage_clause* によって、表、オブジェクト表 OID 索引、パーティション、LOB のデータ・セグメント、LOB の索引セグメントまたは索引構成表のオーバーフロー・データ・セグメントの記憶特性を指定できます。この句は、大規模な表のパフォーマンスに影響します。記憶域は、追加領域の動的割当てを最小限に抑えるように割り当てられます。

参照： 7-51 ページの「[storage_clause](#)」を参照してください。

storage_clause

用途

storage_clause を使用すると、データベース・オブジェクトの格納方法を指定できます。記憶域パラメータは、データベースのデータへのアクセス時間およびデータベース内での領域の効率的な利用に影響します。

参照： 記憶域パラメータの影響の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

クラスタ、索引、ロールバック・セグメント、マテリアライズド・ビュー、マテリアライズド・ビュー・ログ、表またはパーティションを作成する場合、これらのオブジェクトに割り当てられるセグメントに、記憶域パラメータの値を指定できます。記憶域パラメータを指定しない場合、オブジェクトが存在する表領域に指定されている記憶域パラメータの値が使用されます。

クラスタ、索引、ロールバック・セグメント、マテリアライズド・ビュー、マテリアライズド・ビュー・ログ、表またはパーティションを変更する場合、記憶域パラメータの値を変更できます。この値の変更は、それ以降のエクステンツの割当てにのみ影響します。

storage_clause は *physical_attributes_clause* の一部であるため、この句は、物理属性句を指定できる任意の文に指定できます (7-47 ページの「[physical_attributes_clause](#)」を参照)。

storage_clause は次の文にも指定できます。

- CREATE CLUSTER および ALTER CLUSTER: クラスタおよびクラスタ内のすべての表の記憶特性を設定または変更する場合 (13-2 ページの「[CREATE CLUSTER](#)」および 9-7 ページの「[ALTER CLUSTER](#)」を参照)
- CREATE INDEX および ALTER INDEX: 索引または索引パーティションの記憶特性を設定または変更する場合 (13-61 ページの「[CREATE INDEX](#)」および 9-61 ページの「[ALTER INDEX](#)」を参照)
- CREATE MATERIALIZED VIEW および ALTER MATERIALIZED VIEW: マテリアライズド・ビュー、そのパーティションの 1 つ、またはマテリアライズド・ビューを保持するために生成される索引の記憶特性を設定または変更する場合 (14-4 ページの「[CREATE MATERIALIZED VIEW](#)」および 9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照)

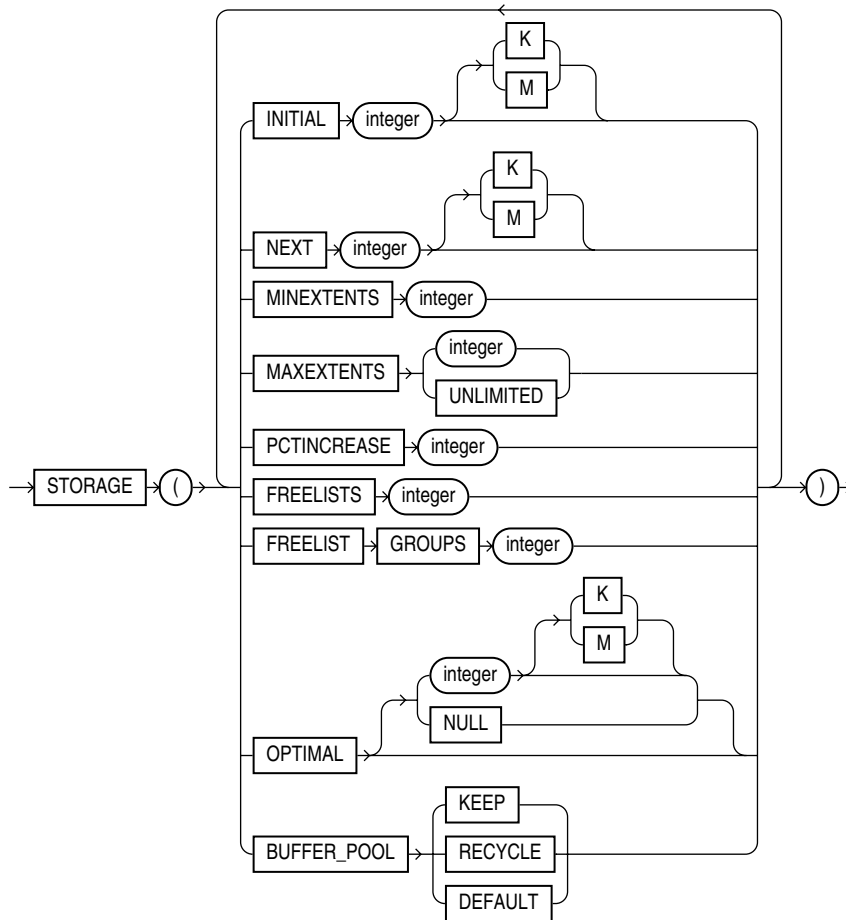
- CREATE MATERIALIZED VIEW LOG および ALTER MATERIALIZED VIEW LOG: マテリアライズド・ビュー・ログの記憶特性を設定または変更する場合 (14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」および 9-108 ページの「[ALTER MATERIALIZED VIEW LOG](#)」を参照)
- CREATE ROLLBACK SEGMENT および ALTER ROLLBACK SEGMENT: ロールバック・セグメントの記憶域属性を設定または変更する場合 (14-76 ページの「[CREATE ROLLBACK SEGMENT](#)」および 9-134 ページの「[ALTER ROLLBACK SEGMENT](#)」を参照)
- CREATE TABLE および ALTER TABLE: 表、表のパーティションまたはサブパーティションの 1 つの LOB データ・セグメントの記憶特性を設定する場合 (15-7 ページの「[CREATE TABLE](#)」および 11-2 ページの「[ALTER TABLE](#)」を参照)
- CREATE TABLESPACE および ALTER TABLESPACE: 表領域に作成されたオブジェクトに対するデフォルトの記憶特性を設定または変更する場合 (15-75 ページの「[CREATE TABLESPACE](#)」および 11-97 ページの「[ALTER TABLESPACE](#)」を参照)
- constraints: 制約の適用に使用される索引 (パーティション索引の場合は索引のパーティション) の記憶域を指定する場合 (7-5 ページの「[constraints](#)」を参照)

前提条件

STORAGE パラメータの値を変更する場合は、適切な CREATE 文または ALTER 文を使用するための権限が必要です。

構文

storage_clause::=



セマンティクス

この項では、*storage_clause* のパラメータについて説明します。詳細は、特定のデータベース・オブジェクトに対してこれらのパラメータを設定または再設定する SQL 文の説明を参照してください。

注意： ローカル管理表領域の場合、*storage_clause* の解析方法は異なります。作成時には、MAXEXTENTS は無視され、他のパラメータ値を使用してセグメントの初期サイズが計算されます。詳細は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

参照： 7-59 ページの「[表の記憶域属性の指定例](#)」および 7-59 ページの「[ロールバック・セグメントの記憶域属性の指定例](#)」を参照してください。

INITIAL

オブジェクトの第 1 エクステントのサイズをバイト単位で指定します。スキーマ・オブジェクトの作成時に、このエクステントに領域が割り当てられます。K または M を使用すると、KB または MB 単位で指定できます。

デフォルトのサイズは 5 データ・ブロックです。セグメントを手動で領域管理する表領域の場合、最小値は 2 データ・ブロック・サイズに指定した各空きリスト・グループの 1 データ・ブロック・サイズを加えた値になります。セグメントを自動的に領域管理する表領域の場合、最小値は 5 データ・ブロック・サイズになります。最大値は、ご使用のオペレーティング・システムによって異なります。

ディクショナリ管理表領域では、表領域の作成時に MINIMUM EXTENT が指定されている場合、INITIAL の値は、必要に応じて指定した MINIMUM EXTENT のサイズに丸められます。MINIMUM EXTENT が指定されていない場合、その表領域に作成されるセグメントの INITIAL エクステント・サイズは最小値（前述の説明を参照）、または要求されたサイズが 5 ブロックより大きい場合は、5 ブロックの倍数に丸められます。

ローカル管理表領域では、INITIAL の値と表領域に指定されたエクステント・サイズの両方に基づいてオブジェクトの第 1 エクステントが決定されます。たとえば、エクステントが 5M の均一なローカル管理表領域の場合、INITIAL の値を 1M に指定すると、1M のエクステントが 5 つ作成されます。

INITIAL の制限事項： ALTER 文では、INITIAL を指定できません。

参照： 空きリスト・グループの詳細は、7-56 ページの「[FREELIST GROUPS](#)」を参照してください。

NEXT

オブジェクトに割り当てる次のエクステント・サイズをバイト単位で指定します。K または M を使用して、KB または MB 単位で指定することもできます。デフォルトのサイズは 5 データ・ブロックです。最小のサイズは 1 データ・ブロックです。最大値は、ご使用のオペレーティング・システムによって異なります。5 データ・ブロックより小さい値が指定された場合、データ・ブロック・サイズの一番近い倍数に丸められます。5 データ・ブロックを超える値の場合は、断片化を最小限に抑える値に切り上げられます。詳細は、『[Oracle9i データベース管理者ガイド](#)』を参照してください。

NEXT パラメータの値を変更した場合（ALTER 文で指定した場合）、次に割り当てられるエクステントのサイズは、直前に割り当てられたエクステントのサイズおよび PCTINCREASE パラメータの値とは関係なく、指定したサイズになります。

参照： 断片化の最小化の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

PCTINCREASE

3 番目以降の各エクステントが、直前のエクステントに対して増加する割合（パーセント）を指定します。デフォルト値は 50 です。この場合、3 番目以降のエクステントは、それぞれその直前のエクステントより 50% ずつ大きくなります。最小値は 0 で、この場合、第 2 エクステント以降のエクステントのサイズはすべて同じになります。最大値は、ご使用のオペレーティング・システムによって異なります。

計算された各エクステントのサイズは、データ・ブロック・サイズの一番近い倍数に丸められます。

また、PCTINCREASE パラメータの値を変更した場合（ALTER 文で指定した場合）、この変更した値と直前に割り当てられたエクステントのサイズを使用して、次に割り当てるエクステントのサイズが計算されます。

提案： すべてのエクステントを同じサイズにする場合は、PCTINCREASE の設定を 0（ゼロ）にして、SMON によるエクステントの結合を回避します。通常、設定は 0（ゼロ）にすることをお薦めします。そうすることで、断片化を最小限に抑え、処理中に一時セグメントの極端な拡大化を防ぐことができます。

PCTINCREASE の制限事項： ロールバック・セグメントに PCTINCREASE は指定できません。ロールバック・セグメントでは、PCTINCREASE の値は常に 0（ゼロ）です。

MINEXTENTS

オブジェクトの作成時に割り当てられる合計エクステント数を指定します。このパラメータを使用した場合、使用可能な領域が連続していない場合でも、オブジェクト作成時にたくさんの領域を割り当てることができます。最小値（デフォルト）は 1 です。この場合、第 1 エクステントのみが割り当てられます。ただし、ロールバック・セグメントの場合、最小値（デフォルト）は 2 です。最大値は、ご使用のオペレーティング・システムによって異なります。

MINEXTENTS の値が 1 より大きい場合、INITIAL、NEXT および PCTINCREASE 記憶域パラメータの値に基づいて、次のエクステントのサイズが計算されます。

ALTER 文で MINEXTENTS の値を変更する場合、現行の値より小さくすることはできますが、大きくすることはできません。MINEXTENTS をより小さい値に再設定すると便利な場合があります。たとえば、TRUNCATE ... DROP STORAGE 文の前で、TRUNCATE 操作の後もセグメントがエクステントの最小数を変更しない場合です。

MINEXTENTS の制限事項： ローカル管理表領域に存在するオブジェクトに対する MINEXTENTS の値は変更できません。

MAXEXTENTS

第 1 エクステントを含めて、Oracle がオブジェクトに割り当てることができるエクステントの総数を指定します。最小値は 1 です（最小値が常に 2 のロールバック・セグメントは除きます）。デフォルト値は、データ・ブロックのサイズによって異なります。

MAXEXTENTS の制限事項： ローカル管理表領域に存在するオブジェクトに対する MAXEXTENTS の値は変更できません。

UNLIMITED 必要に応じてエクステントが自動的に割り当てられるようにする場合に、UNLIMITED を指定します。断片化を最小限に抑えるため、この設定をお勧めします。

ただし、ロールバック・セグメントにこの句は使用しないでください。長時間にわたって挿入、更新または削除を続ける特殊なトランザクションでは、ディスクが一杯になるまで新規エクステントの作成を続けます。

注意： *storage_clause* を指定せずに作成したロールバック・セグメントは、ロールバック・セグメントを作成した表領域の記憶域パラメータと同じ設定になります。MAXEXTENTS UNLIMITED を指定して表領域を作成する場合、ロールバック・セグメントのデフォルトは同じ設定になります。

FREELIST GROUPS

作成するデータベース・オブジェクトに対する空きリスト・グループ数を指定します。このパラメータの最小値（デフォルト）は 1 です。Oracle は、Real Application Clusters インスタンスのインスタンス番号を使用して、各インスタンスを空きリスト・グループにマップします。

1 つの空きリスト・グループに、それぞれデータベース・ブロックを 1 つずつ使用します。したがって、次のことがいえます。

- 各空きリスト・グループの最小値に、1 データ・ブロックを加えた値を格納できるだけの十分な大きさの値を INITIAL の値に指定していない場合、INITIAL の値は必要な分だけ引き上げられます。

- 均一なローカル管理表領域にオブジェクトを作成する場合、空きリスト・グループ数に適応するだけの十分なエクステンツ・サイズがないと、作成操作は正常に実行されません。

注意： オブジェクトが存在する表領域が自動セグメント領域管理モードの場合、FREELIST GROUPS の設定は無視されます。データベースがこのモードで実行されている場合は、15-36 ページの「[LOB_storage_clause](#)」の FREEPOOLS パラメータを参照してください。

FREELIST GROUPS の制限事項： FREELIST GROUPS パラメータは、CREATE TABLE、CREATE CLUSTER、CREATE MATERIALIZED VIEW、CREATE MATERIALIZED VIEW LOG および CREATE INDEX 文でのみ指定できます。

参照：『Oracle9i Real Application Clusters 管理』を参照してください。

FREELISTS

表領域以外のオブジェクトについて、表、パーティション、クラスタまたは索引の各空きリスト・グループの空きリスト数を指定します。このパラメータの最小値（デフォルト）は1です。各空きリスト・グループには、空きリストが1つ割り当てられます。最大値は、データ・ブロックのサイズによって異なります。FREELISTS に指定した値が大きすぎた場合、最大値を示すエラーが戻ります。

注意： オブジェクトが存在する表領域が自動セグメント領域管理モードの場合、FREELIST の設定は無視されます。データベースがこのモードで実行されている場合は、15-36 ページの「[LOB_storage_clause](#)」の FREEPOOLS パラメータを参照してください。

FREELISTS の制限事項： 表領域またはロールバック・セグメントを作成または変更するとき以外は、すべての文の *storage_clause* に FREELISTS を指定できます。

OPTIMAL

OPTIMAL キーワードは、ロールバック・セグメントのみに指定します。 ロールバック・セグメントの最適なサイズをバイト単位で指定します。K または M を使用すると、KB または MB 単位で指定できます。エクステンツのデータがアクティブ・トランザクションで不要になった場合、Oracle は、そのエクステンツの割当てを動的に解除することによって、指定されたロールバック・セグメントのサイズを維持します。ロールバック・セグメントのサイズの合計を OPTIMAL 値より小さくせずに、できるだけ多くのエクステンツの割当てを解除します。

OPTIMAL には、MINEXTENTS、INITIAL、NEXT および PCTINCREASE パラメータで最初に割り当てた領域より小さい値を指定できません。最大値は、ご使用のオペレーティング・システムによって異なります。値は、データ・ブロック・サイズの一番近い倍数に丸められます。

NULL ロールバック・セグメントに対する最適なサイズがないことを示す場合に NULL を指定します。これは、ロールバック・セグメントのエクステンツの割当てが解除されないことを示します。これはデフォルトの動作です。

BUFFER_POOL

BUFFER_POOL 句では、スキーマ・オブジェクト用のデフォルトのバッファ・プール（キャッシュ）を定義します。オブジェクトのすべてのブロックは、指定されたキャッシュに格納されます。

- バッファ・プールがパーティション表またはパーティション索引用に定義されている場合は、パーティションは、パーティション・レベル定義で変更されないかぎり、表定義または索引定義のバッファ・プールを継承します。
- 索引構成表の場合、索引セグメントおよびオーバーフロー・セグメントに対して個々にバッファ・プールを指定できます。

BUFFER_POOL の制限事項：

- この句は、クラスタ化表には指定できません。ただし、クラスタには指定できます。
- この句は、表領域またはロールバック・セグメントには指定できません。

KEEP KEEP を指定すると、ブロックがセグメントから KEEP バッファ・プールへ移されます。KEEP バッファ・プールに適切なサイズを維持すると、メモリーのスキーマ・オブジェクトが保持され、I/O 操作を避けることができます。KEEP は、表、クラスタ、マテリアライズド・ビューまたはマテリアライズド・ビュー・ログに指定する NOCACHE 句より優先されます。

RECYCLE RECYCLE を指定すると、ブロックがセグメントから RECYCLE プールへ移されます。RECYCLE プールに適切なサイズを指定すると、不要なキャッシュ領域が利用されず、デフォルト・プールが RECYCLE プールであるオブジェクト数が削減されます。

DEFAULT デフォルトのバッファ・プールを識別する場合に、DEFAULT を指定します。これは、KEEP も RECYCLE も指定しないオブジェクトのデフォルトです。

参照： 複数のバッファ・プールの使用方法については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

例

表の記憶域属性の指定例 次の文は、表の作成時に記憶域パラメータの値を指定します。

```
CREATE TABLE divisions
  (div_no      NUMBER(2),
   div_name    VARCHAR2(14),
   location    VARCHAR2(13) )
  STORAGE ( INITIAL 100K NEXT      50K
            MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 5);
```

STORAGE パラメータに指定した値に基づいて、次に示すとおり表に領域が割り当てられます。

- MINEXTENTS の値は 1 のため、表作成時にエクステントが 1 つ割り当てられます。
- INITIAL の値は 100KB のため、第 1 エクステントのサイズは 100KB になります。
- 表データが第 1 エクステントを超えた場合、第 2 エクステントが割り当てられます。NEXT の値が 50KB のため、第 2 エクステントのサイズは 50KB になります。
- 表データが増加して最初の 2 つのエクステントを超えた場合、第 3 エクステントが割り当てられます。PCTINCREASE の値は 5 のため、第 3 エクステントの値は第 2 エクステントの 5% 増の 52.5KB になります。このとき、データ・ブロック・サイズが 2KB の場合は、値は丸められて 52KB になります。

これ以降、表データの増加に応じて、直前に割り当てられたエクステントのサイズより 5% 大きいサイズのエクステントが割り当てられます。

- MAXEXTENTS の値は 50 のため、表に割り当てられるエクステントの最大数は 50 になります。

ロールバック・セグメントの記憶域属性の指定例 次の文は、ロールバック・セグメントを作成し、記憶域パラメータの値を指定します。

```
CREATE ROLLBACK SEGMENT rs_store
  STORAGE ( INITIAL 10K NEXT 10K
            MINEXTENTS 2 MAXEXTENTS 25
            OPTIMAL 50K );
```

STORAGE パラメータの値に基づいて、ロールバック・セグメントに次のように領域が割り当てられます。

- MINEXTENTS の値は 2 のため、ロールバック・セグメント作成時にエクステントが 2 つ割り当てられます。
- INITIAL の値は 10KB のため、第 1 エクステントのサイズは 10KB となります。
- NEXT の値は 10KB のため、第 2 エクステントのサイズは 10KB となります。

- ロールバック・データが2つのエクステントを超えた場合、第3エクステントが割り当てられます。ロールバック・セグメントの PCTINCREASE の値は常に 0（ゼロ）のため、第3エクステント以降のエクステントのサイズは第2エクステントのサイズと同じ 10KB です。
- MAXEXTENTS の値は 25 のため、ロールバック・セグメントに割り当てられるエクステントの最大数は 25 になります。
- OPTIMAL の値は 50KB のため、ロールバック・セグメントが 50KB を超えた場合、エクステントの割当てが解除されます。割当てが解除されるエクステントは、アクティブでなくなったトランザクションのデータが入っているエクステントのみです。

SQL 問合せおよび副問合せ

この章では、SQL 問合せおよび副問合せについて説明します。

この章では、次の内容を説明します。

- 問合せおよび副問合せ
- 単純な問合せの作成
- 階層問合せ
- UNION [ALL]、INTERSECT および MINUS 演算子
- 問合せ結果のソート
- 結合
- 副問合せの使用方法
- ネストされた副問合せのネスト解除
- DUAL 表からの選択
- 分散問合せ

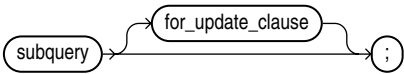
問合せおよび副問合せ

問合せとは、1 つ以上の表またはビューからデータを検索する操作のことです。このマニュアルでは、トップレベルの SELECT 文を**問合せ**といい、他の SQL 文の中でネストされた問合せを**副問合せ**といいます。

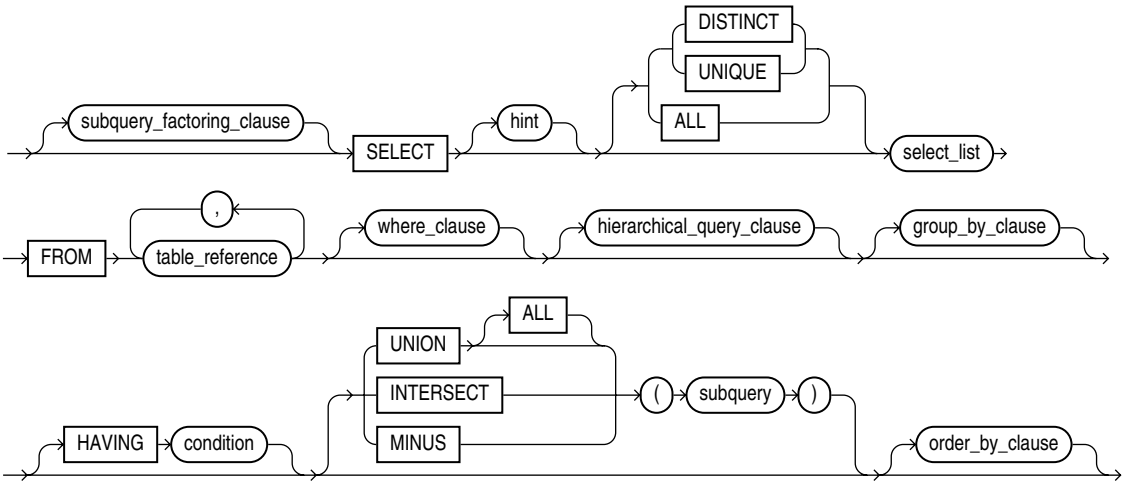
この項では、問合せおよび副問合せの種類およびその使用方法について説明します。この章では、トップレベルの構文について説明します。

参照： すべての句のすべての構文およびこの文のセマンティクスについては、18-4 ページの「[SELECT](#)」を参照してください。

select::=



subquery::=



単純な問合せの作成

SELECT キーワードの後、FROM 句の前にある式のリストを、**SELECT 構文のリスト**といいます。SELECT 構文のリストに、1 つ以上の表、ビューおよびマテリアライズド・ビューから Oracle が戻す行に含まれる 1 つ以上の列を指定します。SELECT 構文のリストの要素によって、列のデータ型、長さおよび数が決定されます。

複数の表に同じ名前の列がある場合、表の名前でその列名を修飾する必要があります。それ以外の場合は、完全に修飾した列名はオプションとなります。ただし、明示的に表および列の参照を修飾することをお勧めします。表および列名を完全に修飾することで、Oracle の作業が少なくなります。

列の別名 `c_alias` を使用して、SELECT 構文のリストの前の式にラベルを付けると、列が新しい見出し付きで表示されます。別名によって、問合せ中に SELECT 構文のリストの項目名を効果的に変更できます。別名は ORDER BY 句の中で使用できますが、問合せ内のその他の句には使用できません。

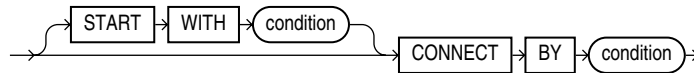
Oracle オプティマイザに指示（ヒント）を与えるために、SELECT 文中でコメントを使用できます。オプティマイザは、これらのヒントを使用して文の実行計画を選択します。

参照： ヒントの詳細は、2-86 ページの「[ヒント](#)」および『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

階層問合せ

表に階層データが含まれる場合、階層問合せ句を使用して階層順に行を選択することができます。

hierarchical_query_clause::=



- START WITH 句では、階層のルート行を指定します。
- CONNECT BY 句では、階層の親 / 子の行の関連を指定します。階層問合せでは、**condition** 内の 1 つの式を、親である行を参照するために PRIOR 演算子で修飾する必要があります。次に例を示します。

```
... PRIOR expr = expr
```

または

```
... expr = PRIOR expr
```

CONNECT BY *condition* が複合条件の場合、1 つの条件のみに PRIOR 演算子が必要です。次に例を示します。

```
CONNECT BY last_name != 'King' AND PRIOR employee_id = manager_id
```

また、CONNECT BY *condition* に副問合せを含めることはできません。

PRIOR は単項演算子であり、単項算術演算子の + および - と同じ優先順位を持っています。この演算子は、階層問合せ内でカレント行の親である行の直後にある式を評価します。

PRIOR は、等価演算子を使用して列の値を比較する場合に最もよく使用されます (PRIOR キーワードは演算子の前後のどちらにも指定できます)。PRIOR を指定すると、列の親である行の値が使用されます。等号 (=) 以外の演算子は、理論上は CONNECT BY 句に指定できます。ただし、これらの他の演算子の組合せによっては、作成される条件は無限ループを発生させる場合があります。この場合、実行時にループが検出され、エラーが戻されます。

参照： 8-5 ページの「例」を参照してください。

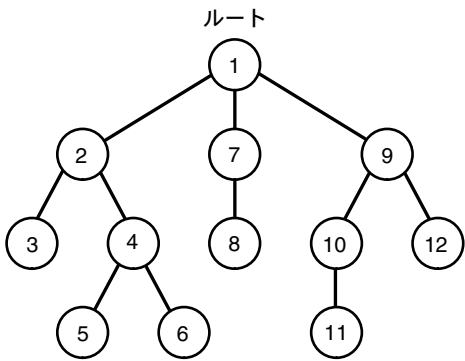
階層問合せでの WHERE 句の Oracle の処理は、WHERE 句が結合を含むかどうかによって異なります。

- WHERE 句が結合を含む場合、Oracle は、CONNECT BY 処理の前に述語結合を適用します。
- WHERE 句に結合が含まれない場合、Oracle は、CONNECT BY 処理の後に、CONNECT BY 述語以外のすべての述語を適用します。この場合、階層の他の行に影響はありません。

Oracle は階層問合せ句からの情報を使用して、次の手順で階層を形成します。

1. Oracle は、CONNECT BY 句の前または後に WHERE 句を処理します。これは、WHERE 句が述語結合を含むかどうかによって決定されます。
2. Oracle は、階層のルート行を選択します。これらの行は、START WITH 条件を満たすものです。
3. Oracle は、各ルート行の子である行を選択します。子である各行は、1 つのルート行に関して CONNECT BY 条件を満たす必要があります。
4. Oracle は、子である行の連続生成を選択します。まず、手順 3 で戻された子である行を選択し、その行にある子を選択します (以降同様に続きます)。現在の親である行に関する CONNECT BY 条件を評価することによって、常に子を選択します。
5. 問合せに結合を含まない WHERE 句が含まれる場合、Oracle は、階層から WHERE 句の条件を満たさないすべての行を排除します。条件を満たさない子である行をすべて排除するのではなく、各行に関してこの条件をそれぞれ評価します。
6. Oracle は、[図 8-1](#) に示す順序で行を戻します。この図では、親である行の下に子である行が表示されます。階層ツリーの詳細は、2-81 ページの [図 2-1 「階層ツリー」](#) を参照してください。

図 8-1 階層問合せ



親である行に対する子を検索するために、Oracle は、親である行の CONNECT BY 条件の PRIOR 式、および各行の他の式を表の中で評価します。条件が TRUE となる行が、その親である行の子です。CONNECT BY 条件に、問合せによって選択された行をさらにフィルタ処理するための他の条件を含めることができます。CONNECT BY 条件に、副問合せを含めることはできません。

CONNECT BY 条件が階層のループになった場合、Oracle はエラーを戻します。1 つの行が別の行の親（または親の親または祖先）および子（または子の子または子孫）の場合、ループが発生します。

注意： 階層問合せでは、ORDER BY または GROUP BY を指定しないでください。指定すると、CONNECT BY の結果の階層順序が壊れます。同じ親の兄弟である行を順序付ける場合は、ORDER SIBLINGS BY 句を使用します。18-23 ページの「[order_by_clause](#)」を参照してください。

例

次の階層問合せは、CONNECT BY を使用して従業員とマネージャの関係を定義しています。

```
SELECT employee_id, last_name, manager_id
FROM employees
CONNECT BY PRIOR employee_id = manager_id;
```

EMPLOYEE_ID	LAST_NAME	MANAGER_ID

101	Kochhar	100
108	Greenberg	101
109	Faviet	108
110	Chen	108

111	Sciarra	108
112	Urman	108
113	Popp	108
200	Whalen	101
.		
.		
.		

次の例は、前述の例と似ていますが、LEVEL 疑似列を使用して、親および子である行を表示しています。

```
SELECT employee_id, last_name, manager_id, LEVEL
FROM employees
CONNECT BY PRIOR employee_id = manager_id;
```

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	LEVEL
101	Kochhar	100	1
108	Greenberg	101	2
109	Faviet	108	3
110	Chen	108	3
111	Sciarra	108	3
112	Urman	108	3
113	Popp	108	3
...			

最後に、次の例は、START WITH 句を追加して階層にルート行を指定し、SIBLINGS キーワードを使用した ORDER BY 句を追加して階層の順序を保持しています。

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

LAST_NAME	EMPLOYEE_ID	MANAGER_ID	LEVEL
King	100		1
Cambrault	148	100	2
Bates	172	148	3
Bloom	169	148	3
Fox	170	148	3
Kumar	173	148	3
Ozer	168	148	3
Smith	171	148	3
De Haan	102	100	2
Hunold	103	102	3
Austin	105	103	4
Ernst	104	103	4

Lorentz	107	103	4
Pataballa	106	103	4
Errazuriz	147	100	2
Ande	166	147	3
Banda	167	147	3
...			

参照：

- 階層問合せでの LEVEL 疑似列の処理方法については、2-81 ページの「[LEVEL](#)」を参照してください。
- ルートからノードへの列の値のパスの検索については、6-147 ページの「[SYS_CONNECT_BY_PATH](#)」を参照してください。
- ORDER BY 句の SIBLINGS キーワードについては、18-23 ページの「[order_by_clause](#)」を参照してください。

UNION [ALL]、INTERSECT および MINUS 演算子

集合演算子 UNION、UNION ALL、INTERSECT および MINUS を使用して、複数の問合せを組み合わせることができます。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ないかぎり、Oracle は左から右の順に評価します。

複合問合せを構成する各問合せと、それに対応する SELECT 構文のリスト内の各式は、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 になります。
- 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 になります。

集合演算子の制限事項：

- 集合演算子は、データ型が BLOB、CLOB、BFILE、VARRAY またはネストした表である列に対しては無効になります。
- UNION、INTERSECT および MINUS 演算子は、LONG 列に対しては無効になります。
- 集合演算子の前の SELECT 構文のリストに式が含まれている場合、[order_by_clause](#) でその式を参照するには、式に列の別名を指定する必要があります。
- [for_update_clause](#) は、これらの集合演算子とともに指定できません。
- これらの演算子の副問合せには、[order_by_clause](#) を指定できません。
- TABLE コレクション式を含む SELECT 文では、これらの演算子を使用できません。

注意： SQL 規格に準拠するために、Oracle の今後のリリースでは、他の集合演算子より優先順位の高い INTERSECT 演算子が提供されます。したがって、INTERSECT 演算子と他の集合演算子を使用する問合せでは、カッコを使用して評価順序を指定してください。

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、他の表に存在していない列がある場合に、(TO_CHAR ファンクションを使用して) データ型を一致させる必要があることを示しています。

```
SELECT location_id, department_name "Department",
       TO_CHAR(NULL) "Warehouse" FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department", warehouse_name
FROM warehouses;
```

LOCATION_ID	Department	Warehouse
1400	IT	
1400		Southlake, Texas
1500	Shipping	
1500		San Francisco
1600		New Jersey
1700	Accounting	
1700	Administration	
1700	Benefits	
1700	Construction	
...		

UNION ALL の例 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のみを戻します。UNION ALL 演算子は、重複行も対象に含めます。

```
SELECT product_id FROM order_items
UNION
SELECT product_id FROM inventories;

SELECT location_id FROM locations
UNION ALL
SELECT location_id FROM departments;
```

問合せで複数回戻される location_id 値 (1700 など) は、UNION 演算子では 1 回のみ戻されますが、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返される行のみが返されます。

```
SELECT product_id FROM inventories
INTERSECT
SELECT product_id FROM order_items;
```

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、最初の間合せでは返されるが、2 番目の間合せでは戻されない行のみが返されます。

```
SELECT product_id FROM inventories
MINUS
SELECT product_id FROM order_items;
```

問合せ結果のソート

ORDER BY 句を使用して、問合せによって選択された行を順序付けます。位置のソートは次のような場合に有効です。

- 長い SELECT 構文のリストの式によって順序付けるためには、ORDER BY 句で全体の式を複製するのではなく、その位置を指定することができます。
- 複合問合せ（集合演算子 UNION、INTERSECT、MINUS または UNION ALL を含む）では、ORDER BY 句に明示的な式ではなく、位置を使用する必要があります。また ORDER BY 句は、最後のコンポーネントの間合せにのみ使用できます。ORDER BY 句は、複合問合せ全体によって戻されたすべての行を順序付けます。

ORDER BY 句による値のソートは、NLS_SORT 初期化パラメータによって明示的に指定するか、NLS_LANGUAGE 初期化パラメータによって暗黙的に指定します。ALTER SESSION 文を使用すると、ソート方法を 1 つの言語ソート基準から別の言語ソート基準に動的に変更できます。ORDER BY 句の NLS_SORT パラメータと NLSSORT ファンクションを使用して、1 つの間合せに特定のソート基準を指定することもできます。

参照： NLS パラメータの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

結合

結合とは、2 つ以上の表、ビューまたはマテリアライズド・ビューの行を結合する問合せです。複数の表が問合せの FROM 句に指定される場合、Oracle は結合を実行します。問合せの SELECT 構文のリストは、これらの表のいずれかの任意の列を選択することができます。これらの表のいずれか 2 つに共通の列名を持つものがある場合、問合せの間、これらの列に対してすべての参照を明確にするために表の名前を付けて修飾する必要があります。

結合条件

結合問合せの大部分は、表ごとに異なる 2 つの列を比較する WHERE 句条件を含んでいます。このような条件を**結合条件**と呼びます。結合を実行するために、Oracle は各表に 1 つずつ含まれている列を結合し、結合条件が TRUE になるようにします。結合条件の列を SELECT 構文のリストに表示する必要はありません。

3 つ以上の表を結合するために、Oracle はまず列を比較する結合条件に基づいて 2 つの表を結合し、結合された表と新規の表の列を含む結合条件に基づいて、さらにもう 1 つの表を結合します。すべての表が結果に結合されるまで、このプロセスを継続します。オプティマイザは、Oracle が結合条件に基づいて表を結合する順序、表の索引、およびコストベースの最適化アプローチを行う場合の表の統計を決定します。

結合条件の他に、結合問合せの WHERE 句にも、1 つの表のみの列を示す他の条件を指定することができます。これらの条件は、結合問合せによって戻された列をさらに制限することができます。

注意： 結合を含む WHERE 句には、LOB 列を指定できません。WHERE 句での LOB の使用については、他にも制限事項があります。詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

等価結合

等価結合とは、等価演算子を含む結合条件での結合のことです。等価結合は、指定した列に同等の値を持つ行を結合します。オプティマイザが結合の実行を選択する内部アルゴリズムによって、1 つの表の等価結合条件における列の合計サイズは、データ・ブロックのサイズ以下に制限される可能性があります。データ・ブロックのサイズは、初期化パラメータ DB_BLOCK_SIZE によって指定されます。

参照： 18-33 ページの「[結合問合せの使用例 :](#)」を参照してください。

自己結合

自己結合とは、自己の表結合のことです。この表は FROM 句に 2 回指定され、結合条件の列名を修飾する表の別名が続きます。自己結合を実行するために、Oracle は結合条件を満たす表の行を結合して戻します。

参照： 18-34 ページの「[自己結合の使用例 :](#)」を参照してください。

デカルト積

結合問合せの2つの表に結合条件がない場合、Oracleは**デカルト積**を戻します。この場合、1つの表の各行が別の表の各行に結合されます。デカルト積は常に多数の行を生成するため、有効ではありません。たとえば、それぞれが100行を持つ2つの表のデカルト積は10,000行を生成します。特にデカルト積を必要としないかぎり、必ず結合条件を指定してください。問合せが3つ以上の表を結合し、特定の組に対して結合条件を指定しない場合、オペティマイザは、中間のデカルト積を生成しないように結合順序を選択する可能性があります。

内部結合

内部結合（単純結合）とは、結合条件を満たす行のみを戻す、複数の表の結合です。

外部結合

外部結合は、単純結合の結果を拡張します。外部結合は、結合条件を満たすすべての行と、ある特定の表の結合条件を満たす行を除いたすべての行を戻します。

- 表AおよびBの外部結合を行い、すべての行をAから戻す問合せ（**左側外部結合**）を記述するには、FROM句でLEFT [OUTER] JOIN構文を使用するか、WHERE句の結合条件で外部結合演算子(+)をBのすべての列に適用します。Bに一致する行のないAのすべての行に関して、Oracleは、Bの列を含む任意のSELECT構文のリストの式にNULLを戻します。
- 表AおよびBの外部結合を行い、すべての行をBから戻す問合せ（**右側外部結合**）を記述するには、FROM句でRIGHT [OUTER] JOIN構文を使用するか、WHERE句の結合条件で外部結合演算子(+)をAのすべての列に適用します。Aに一致する行のないBのすべての行に関して、Oracleは、Aの列を含む任意のSELECT構文のリストの式にNULLを戻します。
- 外部結合を行い、結合条件を満たさない場合にすべての行をAおよびBからNULLで拡張して戻す問合せ（**完全外部結合**）を記述するには、FROM句でFULL [OUTER] JOIN構文を使用します。

Oracleの結合演算子よりも、FROM句のOUTER JOIN構文を使用することをお勧めします。Oracleの結合演算子(+)を使用した外部結合問合せには、次の規則と制限事項があります（これらの規則や制限事項は、FROM句の結合構文にはありません）。

- FROM句の結合構文を含む問合せブロックには結合演算子(+)を指定できません。
- 結合演算子(+)は、WHERE句またはFROM句の左相関のコンテキスト（TABLE句を指定する場合）にのみ指定でき、表またはビューの列にのみ適用されます。
- AおよびBが複数の結合条件によって結合される場合、これらの条件のすべてにおいて結合演算子(+)を使用する必要があります。使用しない場合、Oracleは単純結合の結果である行のみを戻しますが、外部結合の結果がないことを示す警告やエラーは出力しません。

- ある表を外部問合せに指定して別の表を内部問合せに指定した場合、結合演算子 (+) は外部結合を生成しません。
- 自己結合が有効であっても、結合演算子 (+) を使用して表を自己に外部結合することはできません。たとえば、次の文は無効です。

```
-- The following statement is not valid:
SELECT employee_id, manager_id
FROM employees
WHERE employees.manager_id(+) = employees.employee_id;
```

ただし、次の自己結合は有効です。

```
SELECT e1.employee_id, e1.manager_id, e2.employee_id
FROM employees e1, employees e2
WHERE e1.manager_id(+) = e2.employee_id;
```

- 結合演算子 (+) は任意の式ではなく、列にのみ適用することができます。ただし、任意の式には結合演算子 (+) でマークされた 1 つ以上の列を含めることができます。
- 結合演算子 (+) を含む条件は、OR 論理演算子を使用する他の条件と結合できません。
- 条件は、IN 比較条件を使用して、結合演算子 (+) でマークされた列と式を比較できません。
- 条件は、結合演算子 (+) でマークされた列を副問合せと比較できません。

WHERE 句に表 B の列と定数を比較する条件が含まれる場合、Oracle がこの列に対して NULL を生成する表 A の列を戻すように、結合演算子 (+) をこの列に適用する必要があります。それ以外の場合、Oracle は単純結合の結果のみを戻します。

2 組以上の表の外部結合を行う問合せにおいて、単一表は他の 1 つの表のみにに対して NULL 生成された表になることができます。そのため、A と B の結合条件および B と C の結合条件における B の列に、結合演算子 (+) を適用することはできません。

参照： 外部結合の構文については、18-4 ページの「[SELECT](#)」を参照してください。

副問合せの使用法

副問合せは、複数部分の問合せに応答します。たとえば、Taylor の部門で働いている人を判断するには、まず Taylor が働く部門を判断する副問合せを使用できます。その後、親 SELECT 文で元の問合せに応答することができます。SELECT 文の FROM 句の副問合せは、**インライン・ビュー**とも呼ばれます。また、SELECT 文の WHERE 句の副問合せは、**ネストした副問合せ**とも呼ばれます。

副問合せは、別の副問合せを含むことができます。トップレベル問合せの FROM 句内の副問合せレベルの数には、制限がありません。WHERE 句には、最大 255 レベルの副問合せをネストできます。

副問合せにある列が、含まれる文の列と同じ名前を持つ場合、含まれる文の表の列に表名または別名で参照の接頭辞を付ける必要があります。文をさらに読みやすくするには、常に、表、ビューまたはマテリアライズド・ビューの名前または別名で副問合せの列を修飾します。

副問合せが親である文で参照する表の列を参照する場合、Oracle は**相関副問合せ**を行います。相関副問合せは、親である文によって処理された各行を 1 回評価します。親である文は、SELECT、UPDATE または DELETE 文のいずれかです。

相関副問合せは、応答が親である文によって処理された各列の値に依存する複数部分の問合せに応答します。たとえば、相関副問合せを使用して、部門内で給与が平均給与以上の従業員を判断することができます。この場合、相関副問合せは独自で各部門の平均給与を計算します。

参照： 18-40 ページの「[相関副問合せの使用例 :](#)」を参照してください。

副問合せは、次の用途に使用します。

- INSERT または CREATE TABLE 文のターゲット表に挿入する一連の行を定義します。
- CREATE VIEW または CREATE MATERIALIZED VIEW 文のビューまたはマテリアライズド・ビューに含める一連の行を定義します。
- UPDATE 文の既存の行に割り当てる 1 つ以上の値を定義します。
- SELECT、UPDATE および DELETE 文の WHERE 句、HAVING 句または START WITH 句における条件に対する値を定義します。
- 含まれる問合せによって操作される表を定義します。

表名を指定する場合と同様に、問合せを含む FROM 句に副問合せを指定することによってこれらのことを行います。INSERT、UPDATE および DELETE 文においても、このようにして表のかわりに副問合せを使用することができます。

このように使用された副問合せには、相関変数を指定できますが、外部参照ではなく、その副問合せ内に定義された相関変数のみを指定できます。外部参照（左相関副問合せ）は、SELECT 文の FROM 句でのみ使用できます。

参照： 18-16 ページの「[table_collection_expression](#)」を参照してください。

1 つの行から 1 つの列の値を戻すスカラー副問合せは、有効な書式の式です。構文で `expr` をコールするほとんどの場合に、スカラー副問合せ式を使用できます。

参照： 4-13 ページの「[スカラー副問合せ式](#)」を参照してください。

ネストされた副問合せのネスト解除

副問合せは、親である文の WHERE 句内にあるときはネストされています。ネストされた副問合せを持つ文を評価する場合、Oracle は、副問合せ部分を複数回評価する必要があり、効果的なアクセス・パスまたは結合を見逃してしまう可能性があります。

副問合せのネスト解除によって、副問合せの本体がネスト解除され、その副問合せを含む文の本体に結合されます。これによって、アクセス・パスおよび結合の評価時に、オブティマイザが副問合せと文を 1 つのものと判断します。オブティマイザは、ほぼすべての副問合せをネスト解除できますが、いくつか例外があります。これらの例外としては、階層副問合せ、および ROWNUM 疑似列、集合演算子の 1 つ、ネストした集計ファンクション、副問合せの直接的な外部問合せブロックではない問合せブロックへの参照を含む副問合せなどがあります。

制約がない場合、オブティマイザは、次のネストされた副問合せを自動的にネスト解除します（ただし、ネスト解除しない場合もあります）。

- 相関関係のない IN 副問合せ
- IN および EXISTS 相関副問合せ（集計ファンクションまたは GROUP BY 句を含まない場合）

ネスト解除された拡張副問合せを行うには、次のタイプの副問合せをネスト解除するようにオブティマイザに指示します。

- 副問合せに HASH_AJ または MERGE_AJ ヒントを指定して、NOT IN 副問合せをネスト解除します。
- 副問合せに UNNEST ヒントを指定して、その他の副問合せをネスト解除します。

参照： ヒントの詳細は、[第 2 章「Oracle SQL の基本要素」](#)を参照してください。

DUAL 表からの選択

DUAL は、データ・ディクショナリとともに Oracle によって自動的に作成された表です。DUAL は、ユーザー SYS のスキーマにありますが、すべてのユーザーが DUAL という名前でアクセスすることができます。DUAL は、VARCHAR2(1) として定義されている DUMMY 列を持ち、「X」値を持つ行を含みます。DUAL 表から選択することは、定数式を SELECT 文で計算する場合に便利です。DUAL には 1 行以外存在しないため、定数は 1 回のみ戻されます。一方で、任意の表から定数、疑似列または式を選択できますが、値は表の行の数のみ戻されます。

参照： DUAL から定数値を選択する例は、6-2 ページの「[SQL ファンクション](#)」を参照してください。

分散問合せ

Oracle の分散データベース管理システム・アーキテクチャによって、Oracle Net および Oracle サーバーを使用するリモート・データベースにアクセスできます。名前の最後に @dblink を追加して、リモート表、ビューまたはマテリアライズド・ビューを識別できます。dblink は、リモート表、ビューまたはマテリアライズド・ビューを含むデータベースへのデータベース・リンクの完全な名前または部分的な名前である必要があります。

参照：

- データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。
- リモート・データベースへのアクセスについては、『Oracle9i Net Services 管理者ガイド』を参照してください。

分散問合せの制限事項： 現在、分散問合せには、FOR UPDATE 句によってロックされたすべての表、および問合せによって選択された LONG 列を持つすべての表が、同じデータベース上に位置している必要があるという制限があります。たとえば、次の文は、remote データベースの print_media 表から press_release (LONG 値) を選択し、local データベースの print_media 表をロックするため、エラーになります。

```
SELECT r.product_id, l.ad_id, r.press_release
FROM pm.print_media@remote r, pm.print_media l
FOR UPDATE OF l.ad_id;
```

また、Oracle は現在、リモート表にあるユーザー定義型またはオブジェクト REF を選択する分散問合せをサポートしていません。

SQL 文 : ALTER CLUSTER ~ ALTER SEQUENCE

この章では、様々な SQL 文についてアルファベット順に説明します。その他の SQL 文については、[第 10 章](#)～[第 18 章](#)を参照してください。

この章では、次の内容を説明します。

- [様々な種類の SQL 文](#)
- [SQL 文の構成](#)
- [ALTER CLUSTER](#)
- [ALTER DATABASE](#)
- [ALTER DIMENSION](#)
- [ALTER FUNCTION](#)
- [ALTER INDEX](#)
- [ALTER INDEXTYPE](#)
- [ALTER JAVA](#)
- [ALTER MATERIALIZED VIEW](#)
- [ALTER MATERIALIZED VIEW LOG](#)
- [ALTER OPERATOR](#)
- [ALTER OUTLINE](#)
- [ALTER PACKAGE](#)

-
- ALTER PROCEDURE
 - ALTER PROFILE
 - ALTER RESOURCE COST
 - ALTER ROLE
 - ALTER ROLLBACK SEGMENT
 - ALTER SEQUENCE

様々な種類の SQL 文

次の項にある表は、SQL 文の機能の概要について、次のカテゴリに分類して説明しています。

- データ定義言語（DDL）文
- データ操作言語（DML）文
- トランザクション制御文
- セッション制御文
- システム制御文

データ定義言語（DDL）文

データ定義言語（DDL）文によって、次のタスクを実行できます。

- スキーマ・オブジェクトの作成、変更および削除
- 権限およびロールの付与および取消し
- 表、索引またはクラスタ上の情報の分析
- 監査オプションの構築
- データ・ディクショナリへのコメントの追加

CREATE、ALTER および DROP コマンドは、特定のオブジェクトに対して排他的アクセスを必要とします。たとえば、別のユーザーが特定の表でトランザクションをオープンしている場合、ALTER TABLE 文は実行できません。

GRANT、REVOKE、ANALYZE、AUDIT および COMMENT コマンドは、特定のオブジェクトに対する排他的アクセスを必要としません。たとえば、他のユーザーが表を更新しているときでも、その表を分析できます。

Oracle は、暗黙的にすべての DDL 文の前後で現在のトランザクションをコミットします。

DDL 文の多くは、Oracle にスキーマ・オブジェクトを再コンパイルまたは再認可させることができます。Oracle がスキーマ・オブジェクトを再コンパイルまたは再認可する方法、および DDL 文によってそれを実行する環境については、『Oracle9i データベース概要』を参照してください。

DDL 文は、DBMS_SQL パッケージを使用した PL/SQL によってサポートされます。

参照：『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

次に、DDL 文を示します。

```
ALTER ... (ALTER で始まるすべての文)
ANALYZE
ASSOCIATE STATISTICS
AUDIT
COMMENT
CREATE ... (CREATE で始まるすべての文)
DISASSOCIATE STATISTICS
DROP ... (DROP で始まるすべての文)
GRANT
NOAUDIT
RENAME
REVOKE
TRUNCATE
```

データ操作言語 (DML) 文

データ操作言語 (DML) 文は、既存スキーマ・オブジェクトのデータを問い合わせ、操作します。次の文は、現在のトランザクションを暗黙的にコミットしません。次に、データ操作言語文を示します。

```
CALL
DELETE
EXPLAIN PLAN
INSERT
LOCK TABLE
MERGE
SELECT
UPDATE
```

CALL および EXPLAIN PLAN 文は、動的に実行されるときにのみ PL/SQL でサポートされます。他のすべての DML 文は、PL/SQL で完全にサポートされます。

トランザクション制御文

トランザクション制御文は、DML 文で行った変更を管理します。次に、トランザクション制御文を示します。

```
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION
```

COMMIT および ROLLBACK コマンドの特定書式以外のトランザクション制御文は、PL/SQL でサポートされます。制限については、12-70 ページの「[COMMIT](#)」および 17-97 ページの「[ROLLBACK](#)」を参照してください。

セッション制御文

セッション制御文は、ユーザー・セッションのプロパティを動的に管理します。次の文は、現在のトランザクションを暗黙的にコミットしません。

PL/SQL は、セッション制御文をサポートしません。次に、セッション制御文を示します。

```
ALTER SESSION  
SET ROLE
```

システム制御文

単一システム制御文 ALTER SYSTEM は、Oracle インスタンスのプロパティを動的に管理します。この文は、現在のトランザクションを暗黙的にコミットしません。また、PL/SQL ではサポートされません。

埋込み SQL 文

埋込み SQL 文は、DDL、DML およびトランザクション制御文を手続き型言語プログラム内に入れます。埋込み SQL は、Oracle プリコンパイラでサポートされており、次のマニュアルに記載されています。

- 『Pro*COBOL Precompiler プログラマーズ・ガイド』
- 『Pro*C/C++Precompiler プログラマーズ・ガイド』
- 『SQL*Module for Ada Programmer's Guide』

SQL 文の構成

この章および第 10 章～第 18 章のすべての SQL 文は、次の項で編成されています。

構文 構文図には、文を構成するキーワードおよびパラメータを示します。

注意： すべてのキーワードおよびパラメータがあらゆる環境において有効なわけではありません。構文の制限事項については、文および句の「セマンティクス」を参照してください。

用途 文の基本的な使用方法を説明します。

前提条件 文の実行に必要な権限と、文を使用する前に実行する手順を示します。特に指定がないかぎり、ご使用のインスタンスでデータベースがオープンされている必要があります。

セマンティクス 構文を構成するキーワード、パラメータおよび句の用途について説明します。また、制限事項およびその他の注意事項についても説明します。(この章および第 10 章～第 18 章で使用するキーワードおよびパラメータの表記規則の詳細は、「はじめに」を参照してください。)

例 文の様々な句およびパラメータの使用方法を示します。

ALTER CLUSTER

用途

ALTER CLUSTER 文を使用すると、クラスタの記憶特性および並列特性を再定義できます。

注意： クラスタ・キーの列番号および列名を変更するためにこの文を使用することはできません。また、クラスタを格納する表領域を変更することはできません。

参照：

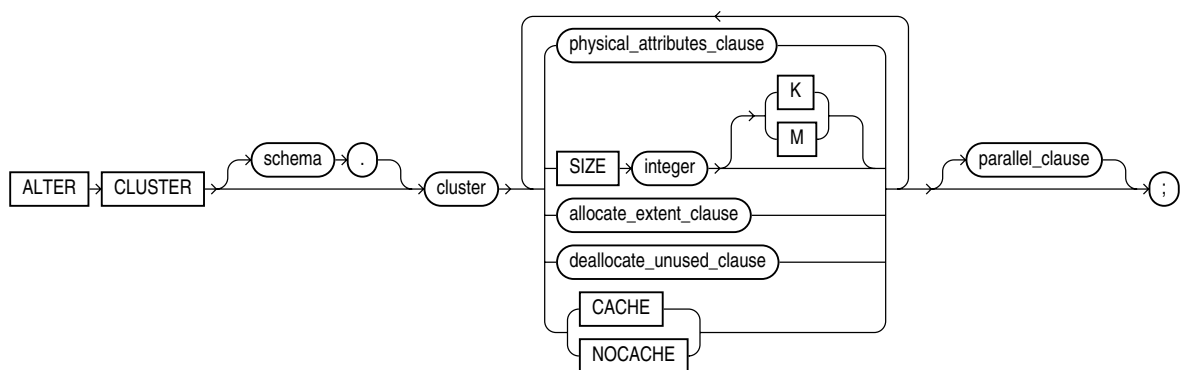
- クラスタの作成については、13-2 ページの「[CREATE CLUSTER](#)」を参照してください。
- クラスタからの表の削除については、16-64 ページの「[DROP CLUSTER](#)」および 17-6 ページの「[DROP TABLE](#)」を参照してください。

前提条件

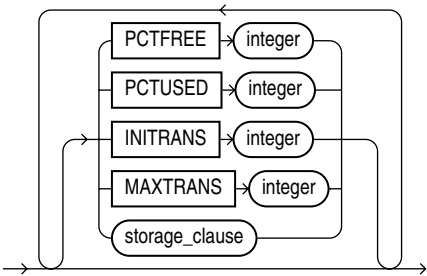
クラスタが自分のスキーマ内にあるか、または ALTER ANY CLUSTER システム権限が必要です。

構文

alter_cluster::=

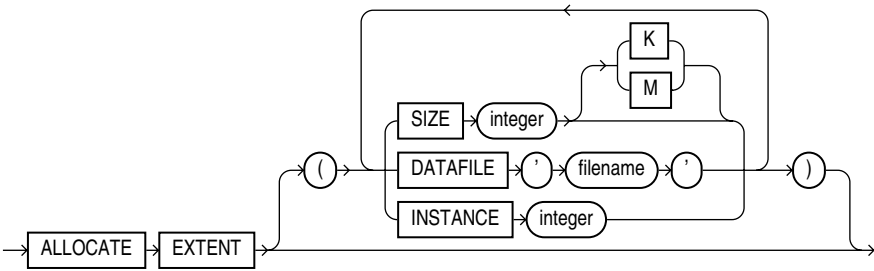


physical_attributes_clause::=

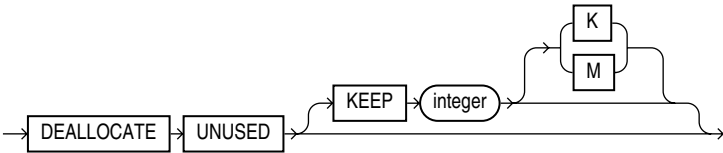


(7-51 ページの「`storage_clause`」を参照)

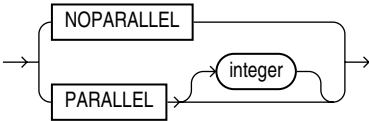
allocate_extent_clause::=



deallocate_unused_clause::=



parallel_clause::=



セマンティクス

schema

クラスタが含まれているスキーマを指定します。*schema* を指定しない場合、そのクラスタが自分のスキーマにあるとみなされます。

cluster

変更するクラスタの名前を指定します。

physical_attributes_clause

クラスタの PCTUSED パラメータ、PCTFREE パラメータ、INITTRANS パラメータおよび MAXTRANS パラメータの値を変更します。

クラスタの記憶特性を変更するには、**STORAGE** 句を使用します。

物理属性の制限事項： クラスタの記憶域パラメータ **INITIAL** および **MINEXTENTS** の値は変更できません。

参照：

- この句の詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- 記憶域パラメータの詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

SIZE integer

SIZE 句を使用すると、クラスタに割り当てられたデータ・ブロック中に格納されるクラスタ・キーの数を指定できます。

SIZE の制限事項： ハッシュ・クラスタではなく、索引クラスタの **SIZE** パラメータのみを変更できます。

参照： **SIZE** パラメータについては、13-2 ページの「[CREATE CLUSTER](#)」を参照してください。9-11 ページの「[クラスタの変更例：](#)」も参照してください。

allocate_extent_clause

allocate_extent_clause を指定すると、クラスタの新しいエクステンツを明示的に割り当てることができます。

この句で明示的にエクステントを割り当てる場合、Oracle は、クラスタの記憶域パラメータを評価しません。割り当てられる新しいエクステントの新しいサイズも決定しません（表を作成する際に行います）。したがって、Oracle がデフォルト値を使用しないようにするには、SIZE を指定してください。

エクステントの割当ての制限事項： ハッシュ・クラスタではなく、索引クラスタのみに新しいエクステントを割り当てることができます。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。また、9-11 ページの「[未使用領域の解放例:](#)」を参照してください。

deallocate_unused_clause

deallocate_unused_clause 句を使用すると、クラスタの終わりの未使用領域の割当てを明示的に解除し、解放された領域が他のセグメントで使用可能になります。

参照： この句の詳細は、7-34 ページの「[deallocate_unused_clause](#)」を参照してください。

CACHE | NOCACHE

CACHE 全表スキャンの実行時に、このクラスタに対して取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最高使用頻度側に入れる場合は、CACHE を指定します。この句は、小規模な参照表で有効です。

NOCACHE 全表スキャンの実行時に、このクラスタに対して取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最低使用頻度側に入れる場合は、NOCACHE を指定します。これはデフォルトの動作です。

parallel_clause

parallel_clause を指定すると、クラスタの DML および問合せのデフォルト並列度を変更できます。

並列化するクラスタの制限事項： クラスタの表が LOB 型またはユーザー定義オブジェクト型の列を含む場合、クラスタでその後に行う INSERT、UPDATE または DELETE 操作と同様、この文は通知なしに逐次実行されます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer integer には、パラレル操作で使用するパラレル・スレッド数である **並列度** を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、integer に値を指定する必要はありません。

参照： 15-51 ページの「CREATE TABLE」の「parallel_clause の注意事項 :」を参照してください。

例

次に、13-9 ページの「CREATE CLUSTER」の「例」で作成したクラスタを変更する例を示します。

クラスタの変更例： 次の文は、personnel クラスタを変更します。

```
ALTER CLUSTER personnel
    SIZE 1024 CACHE;
```

この結果、各クラスタ・キー値に 1024 バイトが割り当てられ、CACHE 属性が指定されます。データ・ブロックのサイズを 2KB と想定した場合、このクラスタ内の今後のデータ・ブロックには、各ブロックに 2 つのクラスタ・キー（2KB を 1024 バイトで割った値）が含まれます。

未使用領域の解放例： 次の文は、language クラスタから未使用領域の割当てを解除し、後で使用できるように 30KB の未使用領域を保持します。

```
ALTER CLUSTER language
    DEALLOCATE UNUSED KEEP 30 K;
```

ALTER DATABASE

用途

ALTER DATABASE 文を使用すると、既存のデータベースを変更、メンテナンスまたはリカバリできます。

参照：

- メディア・リカバリの実行例については、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』および『Oracle9i Recovery Manager ユーザーズ・ガイド』を参照してください。
- スタンバイ・データベースをメンテナンスするために ALTER DATABASE 文を使用する場合の詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。
- データベース作成の詳細は、13-22 ページの「[CREATE DATABASE](#)」を参照してください。

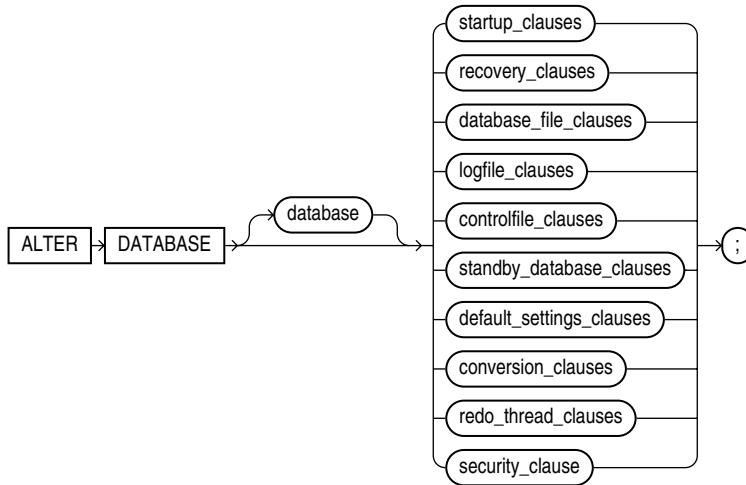
前提条件

ALTER DATABASE システム権限が必要です。

RECOVER 句を指定する場合は、SYSDBA システム権限が必要です。

構文

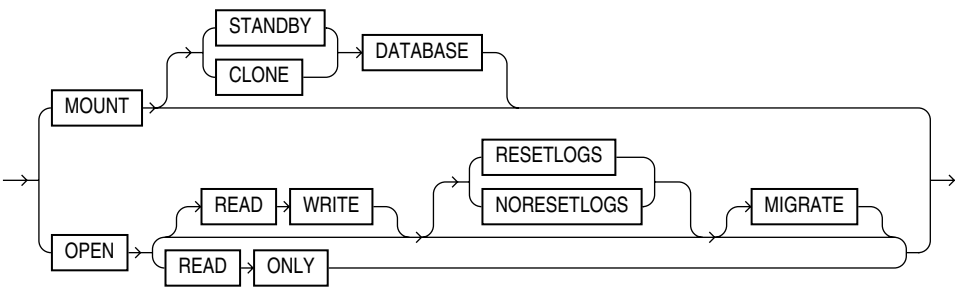
alter_database::=



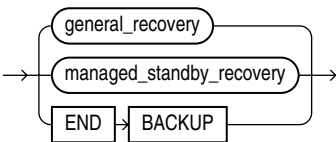
ALTER DATABASE 構文のグループは、次のとおりです。

- **startup_clauses::=** (9-14 ページ)
- **recovery_clauses::=** (9-14 ページ)
- **database_file_clauses::=** (9-18 ページ)
- **logfile_clauses::=** (9-20 ページ)
- **controlfile_clauses::=** (9-21 ページ)
- **standby_database_clauses::=** (9-22 ページ)
- **default_settings_clauses::=** (9-23 ページ)
- **conversion_clauses::=** (9-23 ページ)
- **redo_thread_clauses::=** (9-23 ページ)
- **security_clause::=** (9-23 ページ)

startup_clauses::=

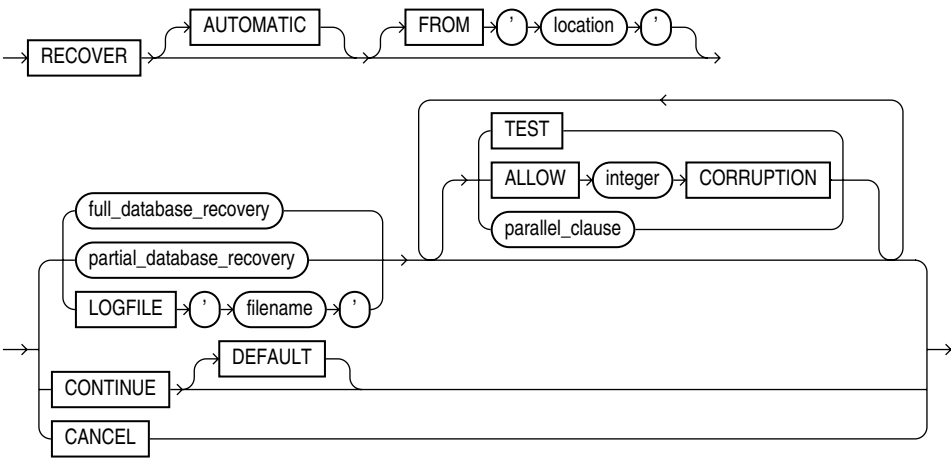


recovery_clauses::=



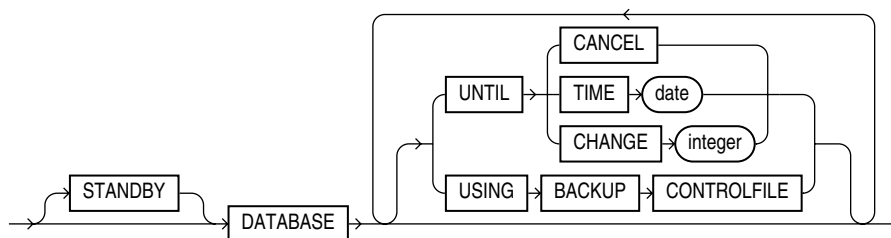
(9-14 ページの「[general_recovery::=](#)」、9-16 ページの「[managed_standby_recovery::=](#)」を参照)

general_recovery::=

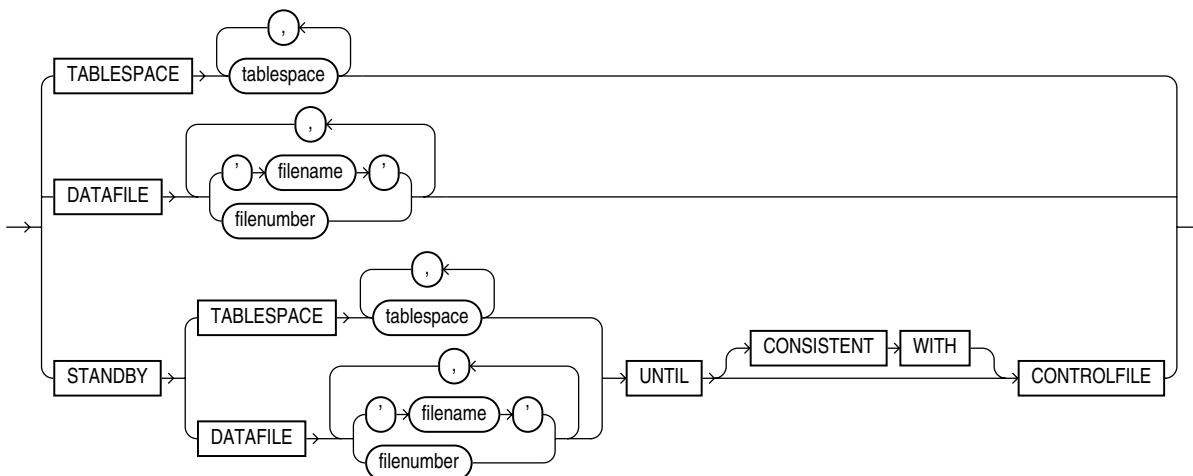


(9-15 ページの「[full_database_recovery::=](#)」、9-15 ページの「[partial_database_recovery::=](#)」、9-15 ページの「[parallel_clause::=](#)」を参照)

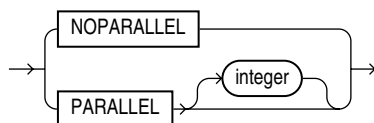
full_database_recovery::=

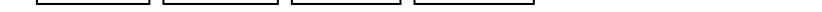


partial_database_recovery::=

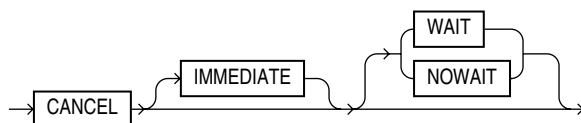


parallel_clause::=

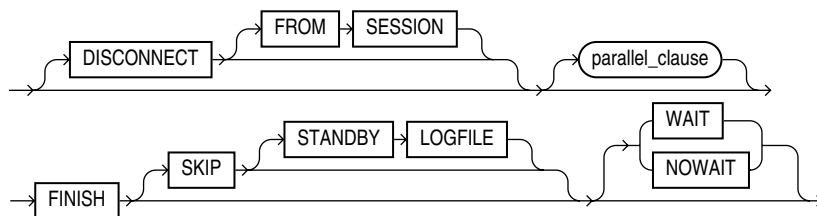




cancel_clause::=



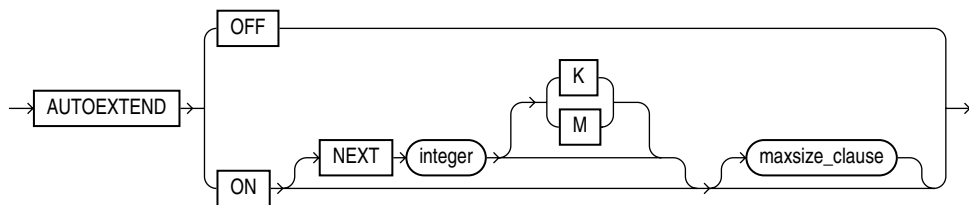
finish_clause::=



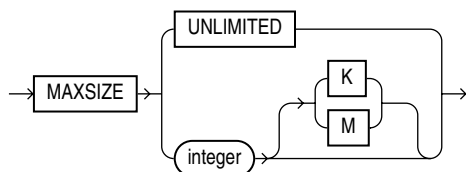
(9-15 ページの「[parallel_clause::=](#)」を参照)



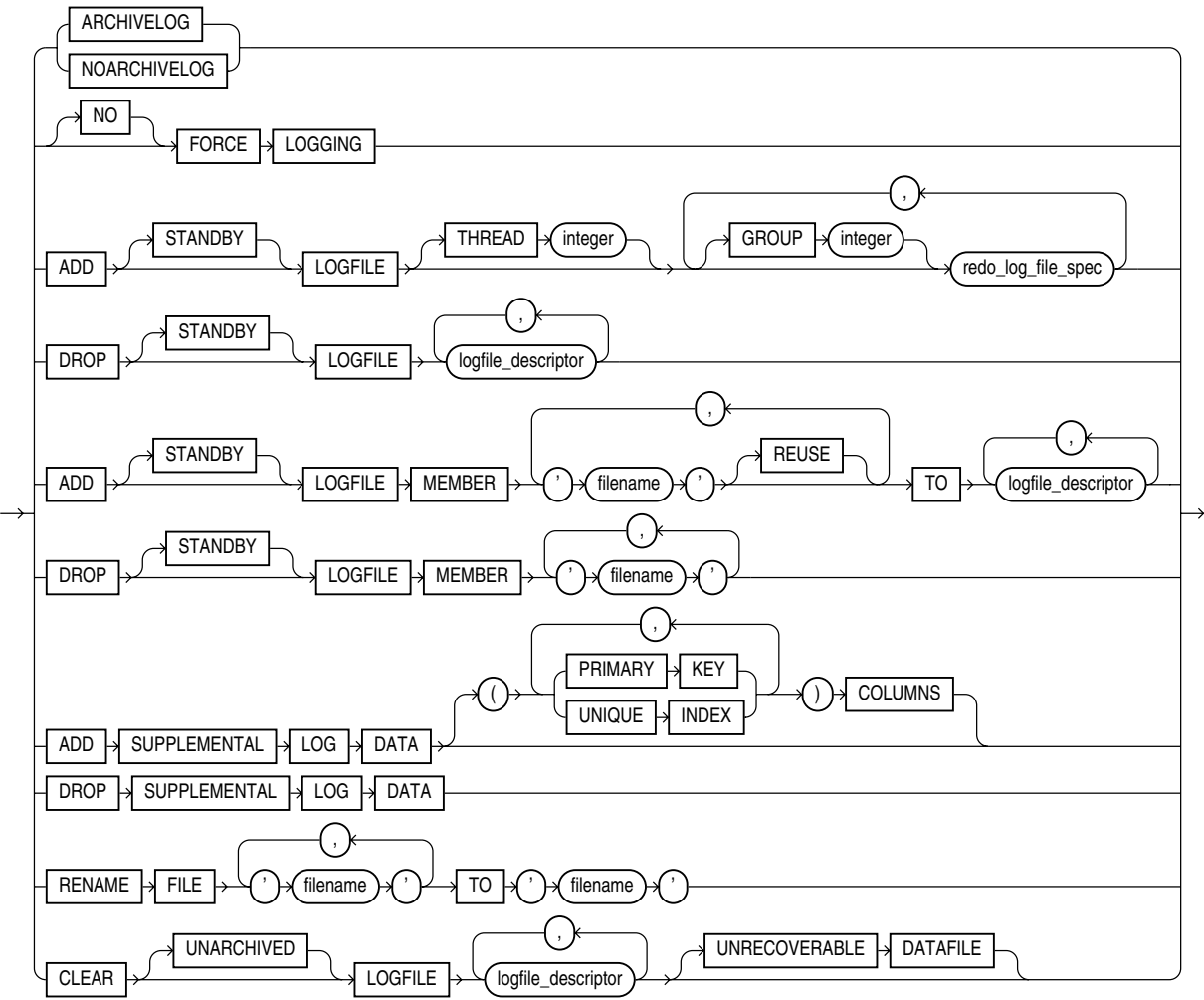
autoextend_clause::=



maxsize_clause::=

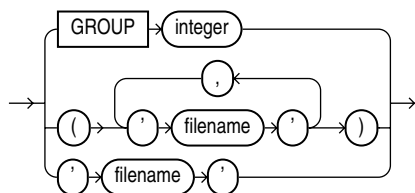


logfile_clauses::=

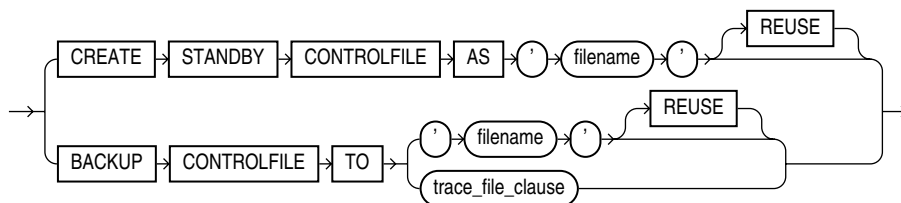


(7-37 ページの「redo_log_file_spec::=」を参照)

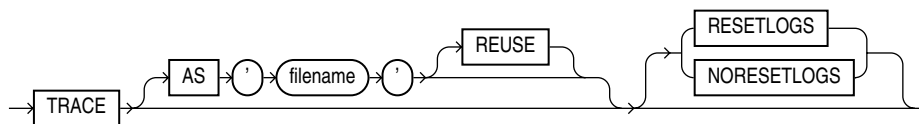
logfile_descriptor::=



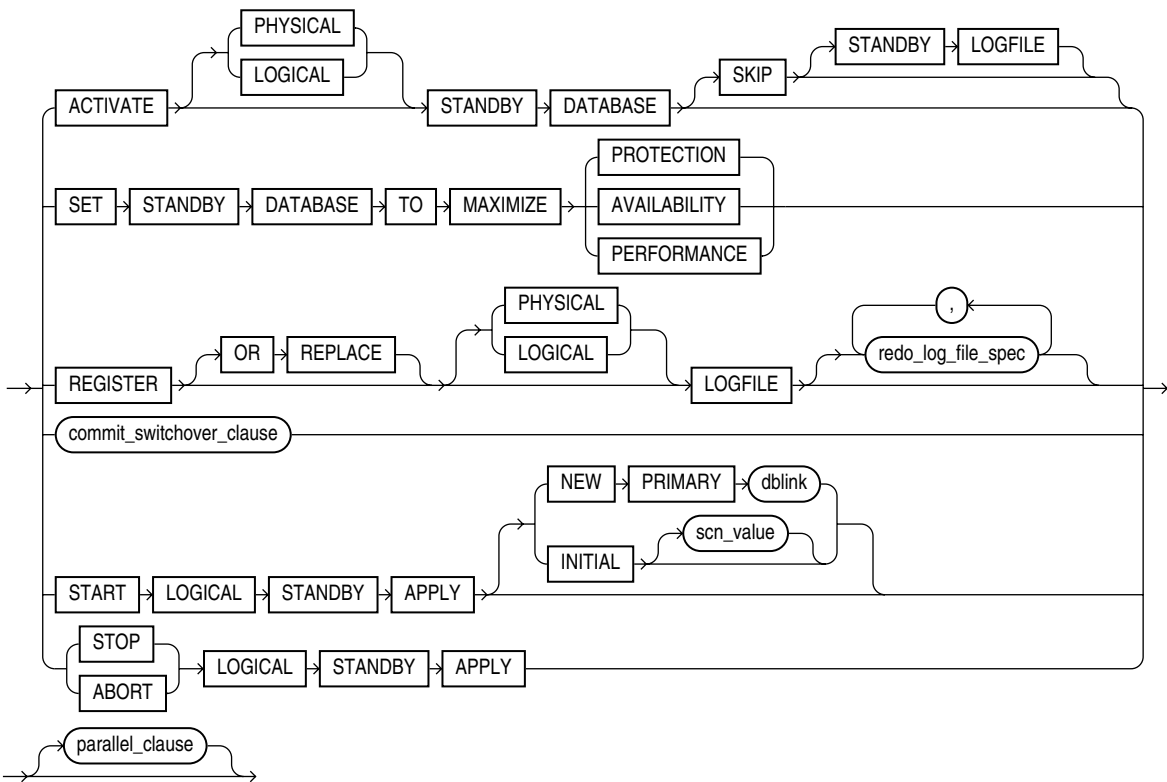
controlfile_clauses::=



trace_file_clause::=

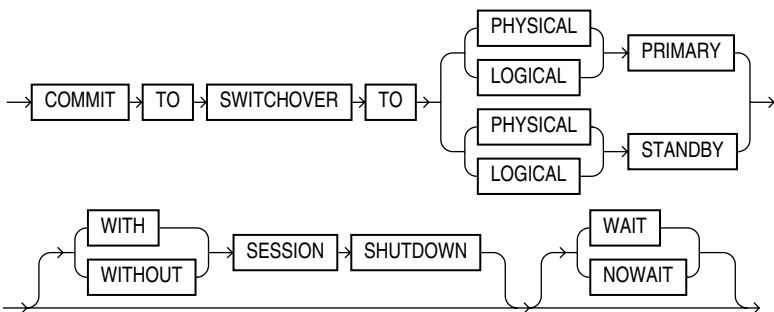


standby_database_clauses::=

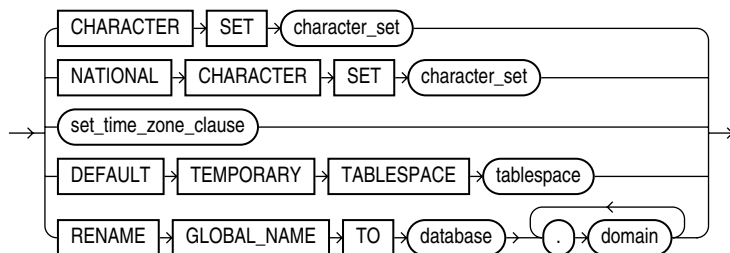


(7-37 ページの「redo_log_file_spec::=」を参照)

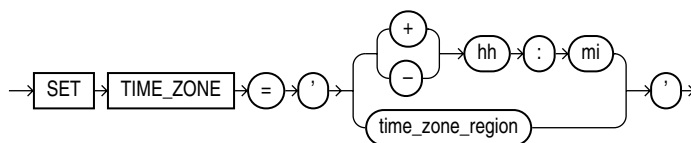
commit_switchover_clause::=



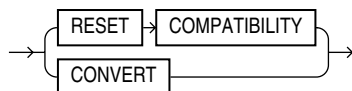
default_settings_clauses::=



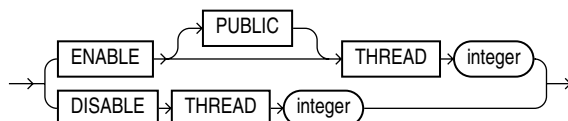
set_time_zone_clause::=



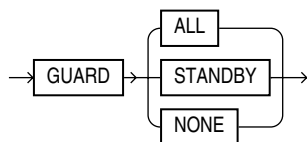
conversion_clauses::=



redo_thread_clauses::=



security_clause::=



セマンティクス

database

変更するデータベースの名前を指定します。データベース名には ASCII 文字のみが使用できます。データベース名を指定しないと、初期化パラメータ `DB_NAME` に指定されているデータベースが変更されます。なお、データベースの制御ファイルが初期化パラメータ `CONTROL_FILES` に指定されている場合にのみ、そのデータベースを変更できます。データベース識別子は、Oracle Net のデータベース指定とは関係ありません。

startup_clauses

`startup_clauses` を使用すると、データベースがマウントおよびオープンされ、ユーザーからのアクセスが可能になります。

MOUNT 句

MOUNT 句を使用すると、データベースがマウントされます。データベースがマウントされている場合、この句は使用できません。

MOUNT STANDBY DATABASE MOUNT STANDBY DATABASE を指定すると、物理スタンバイ・データベースがマウントされます。この文の実行直後、スタンバイ・インスタンスはプライマリ・インスタンスからアーカイブ REDO ログを受信し、ログを STANDBY_ARCHIVE_DEST で指定された位置にアーカイブできるようになります。

参照：『Oracle9i Data Guard 概要および管理』を参照してください。

MOUNT CLONE DATABASE MOUNT CLONE DATABASE を指定すると、クローン・データベースがマウントされます。

参照：クローン・データベースの詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

OPEN 句

OPEN 句を使用すると、データベースが使用可能な状態になります。データベースをオープンするには、マウントしておく必要があります。

OPEN を他のキーワードを指定せずに単独で指定した場合のデフォルトは、OPEN READ WRITE NORESETLOGS です。

READ WRITE READ WRITE を指定すると、読み込み / 書き込みモードでデータベースがオープンされ、ユーザーは REDO ログを生成できるようになります。これはデフォルト値です。

参照：9-51 ページの「[READ ONLY | READ WRITE の例:](#)」を参照してください。

RESETLOGS RESETLOGS を指定すると、現行のログ順序番号を 1 にリセットし、リカバリ時に適用されなかった REDO 情報が今後適用されないように破棄されます。これによって、REDO ログに記録されている変更は、すべて破棄されますが、データベース内の変更は破棄されません。

RECOVER 句による不完全リカバリや、バックアップ制御ファイルによるメディア・リカバリを行った後は、RESETLOGS を指定してデータベースをオープンしてください。この句を使用してデータベースをオープンした場合は、データベース全体のバックアップを行ってください。

NORESETLOGS NORESETLOGS を指定すると、ログの順序番号および REDO ログ・ファイルの現在の状態が保持されます。

ログのリセットの制限事項： バックアップ制御ファイルによる完全メディア・リカバリまたは不完全メディア・リカバリを行った後でのみ RESETLOGS および NORESETLOGS を指定できます。それ以外の場合は、NORESETLOGS が自動的に適用されます。

MIGRATE Oracle リリース 7.3.4 からこのリリースへアップグレードする場合にかぎり、MIGRATE 句を使用します。この句は、Oracle に対して、アップグレードに必要なシステム・パラメータを動的に変更するように指示します。7.3.4 以外のリリースからのアップグレードの場合は、SQL*Plus の STARTUP MIGRATE コマンドが使用できます。

参照：

- データベースをあるリリースから別のリリースに移行する手順については、『Oracle9i データベース移行ガイド』を参照してください。
- SQL*Plus の STARTUP コマンドの詳細は、『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。

READ ONLY READ ONLY を指定すると、トランザクションが読み専用で制限され、ユーザーは REDO ログを生成できなくなります。この句を使用すると、プライマリ・データベース・サイトからのアーカイブ・ログのコピー中でも、物理スタンバイ・データベースが問合せに使用可能になります。

OPEN 句の制限事項：

- データベースが他のインスタンスで READ WRITE でオープンされている場合、READ ONLY ではオープンできません。
- データベースをリカバリする必要がある場合、READ ONLY ではオープンできません。
- データベースが READ ONLY でオープンされている間は、表領域をオフラインに切り替えることはできません。ただし、データ・ファイルのオフラインとオンラインの切替えはできます。また、データベースが READ ONLY でオープンされている間は、オフラインのデータ・ファイルおよび表領域をリカバリできます。

recovery_clauses

recovery_clauses には、バックアップ後の操作が含まれます。

参照： データベースのバックアップの詳細は、『Oracle9i バックアップおよびリカバリ概要』および『Oracle9i Recovery Manager ユーザーズ・ガイド』を参照してください。9-54 ページの「[データベースのリカバリ例:](#)」も参照してください。

general_recovery

general_recovery 句を指定すると、データベース、スタンバイ・データベース、または指定した表領域やファイルのメディア・リカバリを制御できます。インスタンスで、データベースがマウント済、オープン状態、クローズ状態のいずれの場合でも、関連ファイルが使用中でなければ、この句を使用できます。

一般的なデータベース・リカバリの制限事項：

- データベースがクローズ状態の場合にのみ、データベース全体をリカバリできます。
- インスタンスは、排他モードでデータベースがマウントされている必要があります。
- リカバリ対象の表領域またはデータ・ファイルがオフラインの場合、データベースがオープン状態でもクローズ状態でも、表領域またはデータ・ファイルをリカバリできません。
- 共有サーバー・アーキテクチャで Oracle に接続している場合、メディア・リカバリは実行できません。

注意： メディアについて特別な要件がない場合は、*general_recovery* 句ではなく、SQL*Plus の RECOVER コマンドを使用することをお勧めします。

参照：

- メディア・リカバリの詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。
- SQL*Plus の RECOVER コマンドの詳細は、『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。

AUTOMATIC

AUTOMATIC を指定すると、リカバリ操作を続けるために必要な新規アーカイブ REDO ログ・ファイルの名前が自動的に生成されます。LOG_ARCHIVE_DEST_*n* パラメータが定義されている場合、最初のローカル接続先に対して有効で使用可能なパラメータがスキャンされます。その接続先を LOG_ARCHIVE_FORMAT と結合させて使用し、ターゲットの REDO ログ・ファイル名を生成します。LOG_ARCHIVE_DEST_*n* が定義されていない場合は、かわりに LOG_ARCHIVE_DEST パラメータ値が使用されます。

ファイルが検索されると、そのファイルに含まれている REDO が適用されます。ファイルが検索されない場合は、ファイル名の入力を求めるプロンプトが表示されます。この場合、提案として生成されたファイル名が表示されます。

AUTOMATIC も LOGFILE も指定しない場合、ファイル名の入力を求めるプロンプトが表示されます。この場合、提案として生成されたファイル名が表示されます。次に、生成されたファイル名を受け入れるか、または完全なファイル名に置き換えます。生成されたファイル名とアーカイブ済のファイル名が異なることを認識しておく、LOGFILE 句を使用することによって時間を節約できます。

FROM 'location'

FROM 'location' を指定すると、アーカイブ REDO ログ・ファイル・グループを読み込む位置を指定できます。location には、使用するオペレーティング・システムの表記規則に従って、ファイルの位置を完全に指定する必要があります。このパラメータを指定しないと、そのアーカイブ REDO ログ・グループは、初期化パラメータ LOG_ARCHIVE_DEST または LOG_ARCHIVE_DEST_1 に指定された位置にあるとみなされます。

full_database_recovery

full_database_recovery 句を指定すると、データベース全体がリカバリされます。

DATABASE DATABASE 句を指定すると、データベース全体がリカバリされます。これはデフォルト値です。データベースがクローズされている場合にのみ、このオプションを使用できます。

STANDBY DATABASE STANDBY DATABASE 句を指定すると、プライマリ・データベースからコピーされたアーカイブ REDO ログ・ファイルおよび制御ファイルを使用して、物理スタンバイ・データベースを手動でリカバリできます。スタンバイ・データベースは、マウントされているがオープンされていない状態である必要があります。

注意： この句は、オンライン・データ・ファイルのみをリカバリします。

- UNTIL を使用すると、リカバリ操作の存続期間を指定できます。
 - CANCEL は、取消しベースのリカバリを示します。RECOVER CANCEL 句を指定した ALTER DATABASE 文を発行するまで、データベース・リカバリが続行されます。
 - TIME は、時間ベースのリカバリを示します。このパラメータは、**date** に指定した時点までデータベースをリカバリします。**date** は、'YYYY-MM-DD:HH24:MI:SS' の書式の文字リテラルである必要があります。
 - CHANGE は、変更ベースのリカバリを示します。*integer* に指定したシステム変更番号 (SCN) の直前の、トランザクションの一貫性が保たれるところまでデータベースをリカバリします。
- 現行の制御ファイルのかわりにバックアップ制御ファイルを使用する場合、USING BACKUP CONTROLFILE を指定します。

partial_database_recovery

partial_database_recovery 句を指定すると、個々の表領域およびデータ・ファイルがリカバリされます。

TABLESPACE TABLESPACE 句を指定すると、指定した表領域のみがリカバリされます。リカバリの対象となる表領域がオフラインの場合、データベースがオープン状態でもクローズ状態でも、この句を使用できます。

参照： 9-52 ページの「[パラレル・リカバリ処理の使用例:](#)」を参照してください。

DATAFILE DATAFILE 句を指定すると、指定したデータ・ファイルがリカバリされます。リカバリ対象のデータ・ファイルがオフラインの場合、データベースがオープン状態でもクローズ状態でも、この句を使用できます。

データ・ファイルは、名前または番号で識別できます。番号で識別した場合、*filenumber* は、V\$DATAFILE 動的パフォーマンス・ビューの FILE# 列、または DBA_DATA_FILES データ・ディクショナリ・ビューの FILE_ID 列の数を表す整数です。

STANDBY TABLESPACE STANDBY TABLESPACE を指定すると、プライマリ・データベースおよび制御ファイルからコピーしたアーカイブ REDO ログ・ファイルを使用して、スタンバイ・データベース内に損失または破損した表領域が再構成されます。

STANDBY DATAFILE STANDBY DATAFILE を指定すると、プライマリ・データベースおよび制御ファイルからコピーされたアーカイブ REDO ログ・ファイルを使用して、損失または破損したデータ・ファイルを物理スタンバイ・データベース内で手動で再構成できます。[DATAFILE](#) 句と同様、ファイルは、名前または番号で識別できます。

- 旧スタンバイ・データ・ファイルまたは表領域のリカバリに、現行のスタンバイ・データベースの制御ファイルを使用する場合、UNTIL [CONSISTENT WITH] CONTROLFILE を指定します。ただし、スタンバイ制御ファイルに反映されていない REDO ログ・ファイルの内容は適用されません。キーワード CONSISTENT WITH はオプションであり、意味を明確にするためのものです。

LOGFILE

LOGFILE '*filename*' を指定すると、指定した REDO ログ・ファイルを使用して、メディア・リカバリが続行されます。

TEST

TEST 句を使用すると、試行リカバリが実行されます。試行リカバリは、通常のリカバリ手順に問題があった場合に有効です。REDO ストリームを見ると、続いて発生する可能性のある問題を検出することができます。試行リカバリは、通常のリカバリと同様に REDO を適用しますが、ディスクに変更を書き込みません。変更は、試行リカバリの最後にロールバックされます。

ALLOW ... CORRUPTION

ALLOW *integer* CORRUPTION 句を指定すると、ログ・ファイルが破損した場合に、許容リカバリの続行で許容する破損ブロックの数を指定することができます。

試行リカバリでこの句を使用する（TEST 句と結合する）場合は、*integer* に 2 以上の数値を指定できます。通常のリカバリでこの句を使用する場合は、*integer* に 2 以上の数値を指定できません。

参照：

- 一般的なデータベースのリカバリの詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。
- スタンバイ・データベースの管理リカバリの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

parallel_clause

parallel_clause を使用すると、メディア・リカバリをパラレル化するかどうかを指定できます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、**PARALLEL** を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ **PARALLEL_THREADS_PER_CPU** の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である **並列度** を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

参照： 詳細は、15-51 ページの「CREATE TABLE」の「[parallel_clause](#) の注意事項:」を参照してください。

CONTINUE

CONTINUE を指定すると、スレッドを使用禁止にするために中断されていた複数インスタンス・リカバリが再開されます。

CONTINUE DEFAULT を指定すると、他に指定されたログ・ファイルがない場合、自動的に生成された REDO ログ・ファイルを使用して、リカバリが再開されます。ファイル名の入力を求めるプロンプトが表示されないこと以外は、AUTOMATIC の指定と同じです。

CANCEL

CANCEL を指定すると、取消しベースのリカバリが終了します。

managed_standby_recovery

managed_standby_recovery 句は、物理スタンバイのみに適用されます。この句は、管理スタンバイ・リカバリ・モードの指定に使用します。このモードでは、管理スタンバイ・データベースは、スタンバイ・データベース・アーキテクチャ全体のアクティブ・コンポーネントであるとみなされます。プライマリ・データベースは、その REDO ログ・ファイルをスタンバイ・サイトにアクティブにアーカイブします。これらのアーカイブ REDO ログは、スタンバイ・サイトにアーカイブされると、管理スタンバイ・リカバリ操作で使用可能になります。管理スタンバイ・リカバリは、メディア・リカバリに制限されます。インスタンスで、データベースがマウント済、オープン状態、クローズ状態のいずれの場合でも、関連ファイルが使用中でなければ、この句を使用できます。

管理スタンバイ・リカバリの制限事項： 9-26 ページの「[general_recovery](#)」と同じ制限があります。

参照： この句のパラメータの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。9-54 ページの「[管理スタンバイ・データベースのリカバリ例:](#)」も参照してください。

DISCONNECT DISCONNECT を指定すると、バックグラウンド・プロセスの 1 つの管理 REDO プロセス (Managed Redo Process: MRP) が、分離されたバックグラウンド・プロセスとしてアーカイブ REDO ファイルを適用します。これによって、現行のセッションが他のタスクで使用可能になります。(FROM SESSION キーワードはオプションであり、意味を明確にするためのものです。)

DISCONNECT の制限事項：

- DISCONNECT は、管理スタンバイ・リカバリを開始する場合のみ指定できます。ただし、操作が開始した後は、指定できません。
- TIMEOUT および DISCONNECT[FROM SESSION] は同時に指定できません。DISCONNECT 句はバックグラウンドのリカバリ操作を開始しますが、TIMEOUT はフォアグラウンドのリカバリ操作のみを適用対象とします。

TIMEOUT integer 管理リカバリ操作の待ち時間を分単位で指定します。管理スタンバイ・データベースに書き込むために、要求されたアーカイブ REDO ログ・ファイルが使用可能になるまで、リカバリ処理は *integer* 分待ちます。REDO ログ・ファイルがその時間内に使用可能にならなかった場合、リカバリ処理は終了し、エラー・メッセージが表示されます。管理スタンバイ・リカバリ・モードに戻る場合は、この文を再発行します。

TIMEOUT を指定しないか、または NOTIMEOUT を指定すると、RECOVER CANCEL 句付きでこの文を再発行するか、またはインスタンスが停止したり障害を発生させないかぎり、データベースは管理スタンバイ・リカバリ・モードのままです。

TIMEOUT の制限事項：

- TIMEOUT を指定した場合、FINISH は指定できません。
- TIMEOUT および DISCONNECT[FROM SESSION] は同時に指定できません。DISCONNECT 句はバックグラウンドのリカバリ操作を開始しますが、TIMEOUT はフォアグラウンドのリカバリ操作のみを適用対象とします。

NODELAY | DEFAULT DELAY | DELAY integer DELAY を指定すると、指定された間隔 (分単位) 待機してからアーカイブ REDO ログを適用するように Oracle に指示できます。遅延間隔は、リカバリに使用するアーカイブ REDO ログが選択された後、開始します。

- 遅延アーカイブ・ログをスタンバイ・データベースにすぐに適用する必要がある場合は、NODELAY を指定します。
- プライマリ・データベースの初期化パラメータ LOG_ARCHIVE_DEST_n に指定されている分数に戻す場合は、DEFAULT DELAY を指定します。

これらのパラメータによって、プライマリ・データベースの LOG_ARCHIVE_DEST_n パラメータの DELAY の設定が上書きされます。これらのパラメータを省略した場合、アーカイブ・ログの適用は LOG_ARCHIVE_DEST_n の設定に従って遅延されます。パラメータに DELAY が指定されていない場合、アーカイブ・ログはすぐに適用されます。

DELAY の制限事項：

- NONDELAY および DELAY は同時に指定できません。
- DELAY を指定した場合、FINISH は指定できません。

参照： LOG_ARCHIVE_DEST_ *n* パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

NEXT integer NEXT パラメータを使用すると、指定された数のアーカイブ REDO ログを、アーカイブ後できるだけ早く適用できます。このパラメータによって、プライマリ・データベースの初期化パラメータ LOG_ARCHIVE_DEST_ *n* の遅延設定および以前の ALTER DATABASE ... *managed_standby_recovery* 文に指定された DELAY 値が一時的に上書きされます。integer アーカイブ REDO ログが処理されると、これらの遅延は再び有効になります。

NEXT の制限事項： NEXT を指定した場合、FINISH は指定できません。

参照： LOG_ARCHIVE_DEST_ *n* パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

EXPIRE integer 管理リカバリ操作が自動終了してから、現在の時刻までの時間を分単位で指定します。期限切れ時に処理中だったアーカイブ REDO ログの処理は、Oracle によって自動終了されるため、指定された間隔後、事実上、処理が期限切れになることもあります。

あらかじめ定義された EXPIRE オプションを無効にする場合は、NOEXPIRE を指定します。

期限切れは、管理リカバリ処理の開始時間ではなく、常に、現行の文が発行された時間に関連します。既存の管理リカバリ操作を終了するには、CANCEL パラメータを使用します。

EXPIRE の制限事項： EXPIRE を指定した場合、FINISH は指定できません。

THROUGH 句 この句を使用すると、どの時点で管理リカバリを終了するかを Oracle に指示できます。

- THROUGH ... SEQUENCE: この句を指定すると、Oracle は、アーカイブ・ログのスレッド番号および順序番号に基づき、管理リカバリを終了します。対応するアーカイブ・ログが適用されると、管理リカバリは終了します。THREAD 句を省略した場合、Oracle は、スレッド 1 を想定します。
- THROUGH ALL ARCHIVELOG: この句を指定すると、Oracle は、すべてのアーカイブ・ログがリカバリされるまで、管理スタンバイ・プロセスを続行します。この文を使用すると、以前 THROUGH ... SEQUENCE を指定した文を上書きできます。THROUGH 句を省略した場合、これがデフォルト値となります。

- **THROUGH ... SWITCHOVER:** スイッチオーバー操作が発生すると、これらの操作が「END-OF-REDO アーカイブ」インジケータを生成するため、管理スタンバイ・リカバリ・プロセスは、通常停止します。この句は、1 つ以上のスタンバイ・データベースがある場合に有効です。そのスタンバイ・データベースのうち、1 つのみがスイッチオーバー後にスタンバイ・ロールに残ります。この句によって、管理スタンバイ・リカバリ・プロセスの操作可能な状態が保存されます。これによって、リカバリ・プロセスが停止し、新しいプライマリ・データベースがアクティブになった後、そのリカバリ・プロセスを再開するのではなく、新しいプライマリ・データベースから REDO ストリームを受信するまで、これらの「セカンダリ」スタンバイ・データベースは待機します。
 - スイッチオーバー操作が実行される間、管理スタンバイ・リカバリを操作できるよう維持する場合は、ALL を指定します。
 - 最後の END-OF-REDO アーカイブ・インジケータの後、管理スタンバイ・リカバリ操作を取り消す場合は、LAST を指定します。
 - 次に生成された END-OF-REDO アーカイブ・インジケータのリカバリ後、管理スタンバイ・リカバリ操作を取り消す場合は、NEXT を指定します。これはデフォルト値です。

CANCEL CANCEL 句を指定すると、現在のアーカイブ REDO ファイルのすべての REDO が適用された後、管理スタンバイ・リカバリ操作が終了します。CANCEL キーワードのみを指定した場合、セッション制御はリカバリ処理が実際に終了した時点で戻ります。

- CANCEL IMMEDIATE を指定すると、現在のアーカイブ REDO ファイルのすべての REDO が適用された後、または次の REDO ログ・ファイルが読み込まれた後の、いずれか早い方の処理の後に、管理リカバリ操作が終了します。セッション制御は、リカバリ処理が実際に終了した時点で戻ります。

CANCEL IMMEDIATE の制限事項: CANCEL IMMEDIATE 句は、RECOVER MANAGED STANDBY DATABASE 文を発行したセッションからは発行できません。

- CANCEL IMMEDIATE NOWAIT は、セッション制御が、リカバリ処理の終了後ではなくすぐに戻されることを除いて、CANCEL IMMEDIATE と同じです。
- CANCEL NOWAIT を指定すると、次の REDO ログ・ファイルが読み込まれた後に管理リカバリ操作が終了し、セッション制御がすぐに戻ります。

FINISH FINISH 句は、物理スタンバイ・データベースのみに適用されます。FINISH を指定すると、スタンバイ・データベースの現行のスタンバイ・オンライン REDO ログ・ファイルがリカバリされます。この句は、ログ・ライター (LGWR) プロセスが現行のスタンバイ・ログに REDO を転送するときに、プライマリ・データベースに障害が発生した場合にのみ使用します。この句は、アーカイブ・ログ用に指定したすべての遅延間隔を上書きし、すぐにログが適用されます。

FINISH 操作の後、スタンバイ・データベースをプライマリ・データベースとしてオープンする必要があります。

NOWAIT を指定すると、制御は、リカバリ処理の完了後ではなく、すぐに戻されます。

FINISH の制限事項： TIMEOUT、DELAY、EXPIRE または NEXT を指定した場合、FINISH は指定できません。

parallel_clause parallel_clause を使用すると、Oracle が管理リカバリ処理をパラレル化するかどうかを指定できます。NOPARALLEL を指定するか、またはこの句を省略すると、管理スタンバイ・リカバリ操作がシリアルに実行されます。

参照：

- この句の詳細は、7-45 ページの「[parallel_clause](#)」を参照してください。
- 管理スタンバイ・リカバリのパラレル実行によるパフォーマンス向上に関するガイドラインについては、『Oracle9i Data Guard 概要および管理』を参照してください。

END BACKUP 句

END BACKUP 指定すると、オンライン・バックアップ・モードの現行のデータベースにあるすべてのデータ・ファイルが、オンライン・バックアップ・モードから戻されます。この操作を実行するときデータベースはマウントされ、オープンされていない状態である必要があります。

3 つの方法でオンライン・バックアップ（ホット・バックアップ）操作を終了できます。通常の操作では、ALTER TABLESPACE ... END BACKUP 文を使用して、表領域をオンライン・バックアップ・モードから戻します。これによって、表領域がオンライン・バックアップ・モードであることによるオーバーヘッドの増加を回避できます。

システム障害、インスタンス障害、または SHUTDOWN ABORT 操作の後は、オンライン・バックアップ・モードのファイルがシステム・クラッシュ時のファイルと一致するかどうかは識別されません。ファイルに一貫性がある場合、個々のデータ・ファイル、またはすべてのデータ・ファイルをオンライン・バックアップ・モードから戻すことができます。これによって、起動時にファイルのメディア・リカバリを回避できます。

- 個々のデータ・ファイルをオンライン・バックアップ・モードから戻すには、ALTER DATABASE DATAFILE ... END BACKUP 文を使用します。9-35 ページの「[database_file_clauses](#)」を参照してください。
- 表領域のすべてのデータ・ファイルをオンライン・バックアップ・モードから戻すには、ALTER TABLESPACE ... END BACKUP 文を使用します。

参照： オンラインの表領域バックアップの終了については、11-97 ページの「[ALTER TABLESPACE](#)」を参照してください。

database_file_clauses

`database_file_clauses` を指定すると、データ・ファイルおよび一時ファイルを変更できます。インスタンスで、データベースがマウント済、オープン状態、クローズ状態のいずれの場合でも、関連ファイルが使用中でなければ、次の句はどれでも使用できます。

CREATE DATAFILE

CREATE DATAFILE 句を使用すると、元のデータ・ファイルのかわりに新しい空のデータ・ファイルが作成されます。この句を使用すると、バックアップを取らずに失われたデータ・ファイルを再作成できます。`filename` または `filenumber` には、データベースの一部であるファイルまたは以前一部であったファイルを指定します。番号で識別した場合、`filenumber` は、V\$DATAFILE 動的パフォーマンス・ビューの FILE# 列、または DBA_DATA_FILES データ・ディクショナリ・ビューの FILE_ID 列の数を表す整数です。

- AS NEW を指定すると、データ・ファイル用のデフォルトのファイル・システム位置に、システムが生成するファイル名で、置き換えるファイルと同じサイズの Oracle 管理データ・ファイルが作成されます。
- AS `datafile_tempfile_spec` を指定すると、新しいデータ・ファイルにファイル名（およびオプションでサイズ）を割り当てることができます。

元のファイル（`filename` または `filenumber`）が既存の Oracle 管理データ・ファイルの場合、Oracle は新しいファイルを作成した後、元のファイルを削除しようとします。元のファイルが既存のユーザー管理データ・ファイルの場合、Oracle は元のファイルは削除しません。

AS 句を指定しない場合、Oracle によって、`filename` または `filenumber` に指定したファイルと同じ名前およびサイズのファイルが新しく作成されます。

リカバリ時には、元のデータ・ファイルの作成後に書き込まれたアーカイブ REDO ログを、失われたデータ・ファイルに代わる新しい空のデータ・ファイルに適用する必要があります。

新しいファイルは、元のファイルの作成時と同じ状態で作成されます。新しいファイルを元のファイルが失われた時点の状態に戻すには、メディア・リカバリを行ってください。

データ・ファイルの新規作成の制限事項：

- SYSTEM 表領域の最初のデータ・ファイルに基づいて新しいファイルを作成することはできません。
- この CREATE DATAFILE 句には、`datafile_tempfile_spec` の `autoextend_clause` は指定できません。

参照： ファイル仕様（`datafile_tempfile_spec`）の詳細は、7-36 ページの「[file_specification](#)」を参照してください。9-53 ページの「[新しいデータ・ファイルの作成例](#)」も参照してください。

DATAFILE 句

DATAFILE 句を使用すると、名前または番号で識別するファイル进行操作できます。番号で識別した場合、*filenumber* は、V\$DATAFILE 動的パフォーマンス・ビューの FILE# 列、または DBA_DATA_FILES データ・ディクショナリ・ビューの FILE_ID 列の数を表す整数です。DATAFILE 句を使用すると、次のようにデータベース・ファイルを変更できます。

ONLINE ONLINE を指定すると、データ・ファイルがオンラインになります。

OFFLINE OFFLINE を指定すると、データ・ファイルがオフラインになります。データベースがオープンされている場合、データ・ファイルをオンラインに戻す前に、データ・ファイルのメディア・リカバリを行う必要があります。これは、データ・ファイルがオフラインになる前に、チェックポイントが実行されないためです。

DROP データベースが NOARCHIVELOG モードの場合は、DROP 句を指定して、データ・ファイルをオフラインにする必要があります。ただし、DROP 句は、データベースからデータ・ファイルを削除しません。そのため、データ・ファイルが存在する表領域を削除する必要があります。削除するまで、データ・ファイルは RECOVER または OFFLINE の状態でデータ・ディクショナリに残ります。

データベースが ARCHIVELOG モードの場合は、DROP キーワードは無視されます。

RESIZE RESIZE を指定すると、データ・ファイルのサイズを指定した絶対サイズ（バイト単位）を増やしたり減らすことができます。K または M を使用すると、KB または MB 単位で指定できます。デフォルト値はないため、必ずサイズを指定してください。

増やしたサイズに対して十分なディスク領域がない場合、または減らしたサイズを超えるデータがファイルに含まれる場合、エラー・メッセージが戻されます。

参照： 9-54 ページの「[データ・ファイルのサイズの変更例：](#)」を参照してください。

END BACKUP END BACKUP を指定すると、データ・ファイルがオンライン・バックアップ・モードから戻されます。END BACKUP 句は、ALTER DATABASE 構文の最上位に完全に記述されます。9-34 ページの「[END BACKUP 句](#)」を参照してください。

TEMPFILE 句

TEMPFILE 句を使用すると、一時データ・ファイルのサイズを変更できます。また、*autoextend_clause* を指定すると、永続データ・ファイルと同じ効果を持たせることができます。一時ファイルは、名前または番号で識別します。番号で識別した場合、*filenumber* は、V\$TEMPFILE 動的パフォーマンス・ビューの FILE# 列の数を表す整数です。

注意： オペレーティング・システムによっては、一時ファイルのブロックが実際にアクセスされるまで、一時ファイル用の領域が割り当てられない場合があります。領域の割当ての遅延のため、一時ファイルの作成およびサイズ変更が速くなります。ただし、後で一時ファイルが使用されるときに、十分なディスク領域を使用可能にする必要があります。発生する可能性がある問題を回避するには、一時ファイルを作成またはサイズ変更の前に、ディスク領域が、新しく作成する一時ファイルまたはサイズ変更後の一時ファイルのサイズより大きいことを確認してください。ディスク領域に余裕を持たせることによって、関連のない操作が原因で増加が予想されるディスク使用量に対応できます。十分な領域があることを確認した後で、作成またはサイズ変更操作を実行してください。

一時ファイル変更の制限事項： データベースをオープンしないと、TEMPFILE は指定できません。

DROP DROP を指定すると、データベースから一時ファイルが削除されます。表領域はそのまま残ります。

INCLUDING DATAFILES を指定すると、対応付けられたオペレーティング・システム・ファイルは削除され、削除された各ファイルのメッセージがアラート・ログに書き込まれます。

autoextend_clause

autoextend_clause は、新しいデータ・ファイルまたは一時ファイルの自動拡張を使用可能または使用禁止にします。

参照： *autoextend_clause* の詳細は、7-36 ページの「[file_specification](#)」を参照してください。

RENAME FILE 句

RENAME FILE 句を使用すると、データ・ファイル、一時ファイルまたは REDO ログ・ファイル・メンバーの名前を変更できます。この句を指定する前に、オペレーティング・システムのファイル名の表記規則に従って、各ファイル名を指定してください。

- データ・ファイルおよび一時ファイルにこの句を使用するには、データベースをマウントしておく必要があります。データベースはオープンしていてもかまいませんが、名前を変更するデータ・ファイルおよび一時ファイルは、オフラインにしておく必要があります。
- ログ・ファイル用にこの句を使用するには、データベースはマウントされているがオープンされていない状態である必要があります。

この句は、制御ファイルのファイル名のみを変更します。オペレーティング・システムのファイルの名前は、実際には変更されません。オペレーティング・システムのファイル名は存在したままですが、使用されません。制御ファイルでは旧ファイルをデータ・ファイル、一時ファイルまたは REDO ログ・ファイルとして指していないため、旧ファイルが Oracle 管理の場合、オペレーティング・システムの旧ファイルはこの句の実行後に削除されます。

参照： 9-52 ページの「[ログ・ファイル・メンバーの名前の変更例:](#)」を参照してください。

logfile_clauses

logfile_clauses を指定すると、ログ・ファイルを追加、削除または変更できます。

ARCHIVELOG | NOARCHIVELOG

ARCHIVELOG 句および NOARCHIVELOG 句は、インスタンスでデータベースがマウントされているがオープンされていない、かつ Real Application Clusters が使用禁止の場合のみに使用できます。

ARCHIVELOG REDO ログ・ファイル・グループを再利用する前に、グループの内容をアーカイブする場合は、ARCHIVELOG を指定します。このモードでは、メディア・リカバリができるようになります。この句は、エラーなしの通常終了または即時終了でインスタンスを停止し再起動した後に、Real Application Clusters を使用禁止にしてデータベースをマウントした場合のみに指定できます。

NOARCHIVELOG REDO ログ・ファイル・グループを再利用する前に、内容をアーカイブする必要がない場合は、NOARCHIVELOG を指定します。このモードでは、メディア障害後のリカバリはできません。

[NO] FORCE LOGGING

この句を使用すると、データベースを FORCE LOGGING モードにしたり、このモードから戻すことができます。データベースは、マウントまたはオープンされている必要があります。

FORCE LOGGING モードでは、一時表領域および一時セグメントの変更を除くデータベースのすべての変更が記録されます。この設定は、個々の表領域に指定する NOLOGGING または FORCE LOGGING 設定、および個々のデータベース・オブジェクトに指定する NOLOGGING 設定より優先され、これらのいずれにも依存しません。

FORCE LOGGING を指定すると、Oracle は、まだ記録されていない実行中のすべての操作が終了するまで待機します。

参照： FORCE LOGGING モードの使用の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

ADD [STANDBY] LOGFILE 句

ADD LOGFILE 句を使用すると、指定したスレッドに1つ以上の REDO ログ・ファイル・グループが追加され、そのスレッドに割り当てられているインスタンスが、そのグループを使用できるようになります。STANDBY を指定すると、作成された REDO ログ・ファイルは、物理スタンバイ・データベースのみで使用可能になります。

ログ・ファイルがオンライン用、スタンバイ・データベース用のどちらに設計されたかを確認するには、V\$LOGFILE 動的パフォーマンス・ビューの TYPE 列を問い合わせます。

参照： 9-52 ページの「[REDO ログ・ファイル・グループの追加例:](#)」を参照してください。

THREAD THREAD 句は、パラレル・モードの Real Application Clusters で Oracle を使用している場合にのみ適用可能です。integer にはスレッド番号を指定します。作成可能なスレッドの数は、CREATE DATABASE 文で指定した MAXINSTANCES パラメータ値によって制限されます。

THREAD パラメータを指定しないと、REDO ログ・ファイル・グループは、インスタンスに割り当てられたスレッドに追加されます。

GROUP GROUP 句を指定すると、すべてのスレッドのすべてのグループ間で、REDO ログ・ファイル・グループを一意に識別できます。この値は、1 ～ MAXLOGFILES の値までの範囲で設定できます。同一の GROUP 値を持つ REDO ログ・ファイル・グループは、複数追加できません。このパラメータを指定しない場合、値が自動的に生成されます。REDO ログ・ファイル・グループの GROUP 値は、動的パフォーマンス・ビュー V\$LOG で調べることができます。

redo_log_file_spec 各 redo_log_file_spec には、1 つ以上のメンバー（1 つ以上のコピー）で構成される REDO ログ・ファイル・グループを指定します。

参照：

- 7-36 ページの「[file_specification](#)」を参照してください。
- 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

DROP LOGFILE 句

DROP LOGFILE 句を使用すると、REDO ログ・ファイル・グループのすべてのメンバーが削除されます。ADD LOGFILE MEMBER 句と同様に、REDO ログ・ファイル・グループを指定します。

- 現行のログ・ファイル・グループを削除する場合、最初に ALTER SYSTEM SWITCH LOGFILE 文を発行する必要があります。
- アーカイブが必要な REDO ログ・ファイル・グループは、削除できません。

- REDO ログ・ファイル・グループを削除すると、その REDO スレッドの REDO ログ・ファイル・グループが 2 つ未満になる場合は、削除できません。

参照： 10-20 ページの「[ALTER SYSTEM](#)」および 9-52 ページの「[ログ・ファイル・メンバーの削除例 :](#)」を参照してください。

ADD [STANDBY] LOGFILE MEMBER 句

ADD LOGFILE MEMBER 句を使用すると、既存の REDO ログ・ファイル・グループに新しいメンバーを追加できます。新しいメンバーをそれぞれ '*filename*' に指定します。すでにファイルが存在する場合、追加するメンバーはグループ内の他のメンバーと同じサイズである必要があり、REUSE を指定する必要があります。ファイルが存在しない場合、適切なサイズのファイルが作成されます。メディア障害によってグループのすべてのメンバーを失った場合は、そのグループにメンバーを追加することはできません。

STANDBY を指定すると、ログ・ファイル・メンバーが物理スタンバイ・データベースのみで使用可能になります。ただし、このキーワードは必須ではありません。グループ *integer* がスタンバイ・データベース用に追加された場合は、すべてのメンバーも同様にスタンバイ・データベースのみに使用されます。

次のいずれかの方法で、既存の REDO ログ・ファイル・グループを指定できます。

GROUP *integer* *integer* には REDO ログ・ファイル・グループを識別する GROUP パラメータの値を指定します。

filename(s) REDO ログ・ファイル・グループのすべてのメンバーを指定します。ご使用のオペレーティング・システムの表記規則に従って、ファイル名を完全に指定する必要があります。

参照： 9-52 ページの「[REDO ログ・ファイル・グループ・メンバーの追加例 :](#)」を参照してください。

DROP LOGFILE MEMBER 句

DROP LOGFILE MEMBER 句を使用すると、1 つ以上の REDO ログ・ファイル・メンバーが削除されます。各 '*filename*' には、ご使用のオペレーティング・システムのファイル名の表記規則に従って、メンバーを完全に指定する必要があります。

- 現行のログのログ・ファイルを削除する場合、最初に ALTER SYSTEM SWITCH LOGFILE 文を発行する必要があります。

参照： 10-20 ページの「[ALTER SYSTEM](#)」を参照してください。

- この句では、有効なデータを含む REDO ログ・ファイル・グループのすべてのメンバーを削除できません。このような操作には、DROP LOGFILE 句を使用してください。

参照： 9-52 ページの「ログ・ファイル・メンバーの削除例:」を参照してください。

ADD SUPPLEMENTAL LOG DATA 句

ADD SUPPLEMENTAL LOG DATA 句を指定すると、更新操作実行中にログ・ストリームに列データを追加できます。これらの4つのキーワードによって、最小サブリメンタル・ロギングは、使用可能になります。デフォルトでは使用可能ではありません。

最小サブリメンタル・ロギングによって、LogMiner（および LogMiner の技術に基づいた製品）には、連鎖行および様々な記憶域構成（クラスタ表など）をサポートするために十分な情報が確保されます。

サブリメンタル・ログ・データが、別のデータベースで変更対象（論理スタンバイなど）となる場合、ログ・データは更新された各行を一意に識別する必要があります。この場合は、PRIMARY KEY COLUMNS および UNIQUE KEY COLUMNS を指定して、識別キー（フル）・サブリメンタル・ロギングを使用可能にする必要があります。

PRIMARY KEY COLUMNS PRIMARY KEY COLUMNS を指定すると、主キーを持つすべての表において、主キーのすべての列は更新操作が実行されるたびに REDO ログに置かれます。主キーを定義していない場合は、行を一意に識別する列の集合が REDO ログに置かれます。この集合には、最大サイズが固定長のすべての列が含まれます。

UNIQUE INDEX COLUMNS UNIQUE INDEX COLUMNS を指定すると、一意キーを持つすべての表において、一意キー列が変更された場合、一意キーに属するその他のすべての列も REDO ログに置かれます。

注意： データベースのオープン時にこの文を発行できます。ただし、パフォーマンスに影響するカーソル・キャッシュのすべての DML カーソルが、キャッシュが再移入されるまで無効になります。

参照： サブリメンタル・ロギングの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

DROP SUPPLEMENTAL LOG DATA 句

DROP SUPPLEMENTAL LOG DATA 句を使用すると、更新操作が発生するたびに REDO ログ・ストリームに追加のログ情報が置かれることを停止することができます。この文は、前述の ADD SUPPLEMENTAL LOG DATA 文によって得られた効果を終了します。

参照： サブリメンタル・ロギングの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

CLEAR LOGFILE 句

CLEAR LOGFILE 句を使用すると、オンライン REDO ログが初期化しなおされます。REDO ログをアーカイブしないオプションもあります。CLEAR LOGFILE は、REDO ログの追加および削除と似ていますが、スレッドに 2 つ以外ログがない場合や、クローズ状態のスレッドの現行の REDO ログ・ファイルに対しても発行できます。

- アーカイブされていない REDO ログを再利用する場合は、UNARCHIVED を指定する必要があります。

注意： リカバリのために REDO ログが必要な場合に UNARCHIVED を指定すると、バックアップが使用できなくなります。

- ARCHIVELOG モードのデータベースでデータ・ファイルをオフラインにする（DROP キーワードを使用せずに ALTER DATABASE ... DATAFILE OFFLINE を指定する）場合、およびそのデータ・ファイルをオンラインに戻す前に、消去するアーカイブされていないログがデータ・ファイルのリカバリに必要な場合は、UNRECOVERABLE DATAFILE を指定する必要があります。この場合、一度 CLEAR LOGFILE 文を実行して、データ・ファイルおよび表領域全体を削除する必要があります。

メディア・リカバリに必要なログを、CLEAR LOGFILE を使用して消去しないでください。データベースのチェックポイント後の REDO を含むログを消去する必要がある場合は、不完全メディア・リカバリを最初に実行する必要があります。オープンしているスレッドの現行の REDO ログは消去できます。クローズしているスレッドの現行のログは、そのスレッド内でログを切り替えれば消去できます。

CLEAR LOGFILE 文が、システム障害またはインスタンス障害による割込みを受けると、データベースがハングする場合があります。このような状況になった場合は、データベースの再起動後に、この文を再発行します。ログ・グループのあるメンバーにアクセスしようとした際、I/O エラーによる障害が発生した場合は、そのメンバーを削除して他のメンバーを追加できます。

参照： 9-54 ページの「[ログ・ファイルの消去例:](#)」を参照してください。

controlfile_clauses

controlfile_clauses を指定すると、制御ファイルを作成またはバックアップできます。

CREATE STANDBY CONTROLFILE 句

CREATE STANDBY CONTROLFILE 句は、物理スタンバイ・データベースのみに適用されます。この句を使用すると、物理スタンバイ・データベースを管理するための制御ファイルを作成できます。ファイルがすでに存在している場合は、REUSE を指定してください。

参照： 『Oracle9i Data Guard 概要および管理』を参照してください。

BACKUP CONTROLFILE 句

BACKUP CONTROLFILE 句を使用すると、現行の制御ファイルのバックアップが取られます。この句を使用するとき、データベースをオープンまたはマウントしておく必要があります。

TO 'filename' filename には制御ファイルのバックアップ先ファイルを指定します。ご使用のオペレーティング・システムの表記規則に従って、ファイル名を完全に指定する必要があります。指定したファイルがすでに存在している場合は、REUSE を指定します。

TO TRACE TO TRACE を指定すると、制御ファイルの物理バックアップが取られるのではなく、トレース・ファイルに SQL 文が書き込まれます。トレース・ファイルに記述された SQL 文を使用すると、データベースの起動、制御ファイルの再作成、データベースのリカバリやオープンなどの操作を、作成した制御ファイルに基づいて正しく実行できます。

制御ファイルのコピーがすべて失われた場合（または、制御ファイルのサイズを変更する場合）は、トレース・ファイルからスクリプト・ファイルに文をコピーし、必要に応じて文を編集できます。

- スクリプトを標準トレース・ファイルではなく、*filename* と呼ばれるファイルに格納する場合は、**AS filename** を指定します。
- REUSE を指定すると、*filename* と呼ばれる既存ファイルを上書きできます。
- RESETLOGS は、データベースの起動用としてトレース・ファイルに書き込まれた SQL 文が、ALTER DATABASE OPEN RESETLOGS であることを示します。オンライン・ログが使用不可能な場合のみ、この設定は有効です。
- NORESETLOGS は、データベースの起動用としてトレース・ファイルに書き込まれた SQL 文が、ALTER DATABASE OPEN NORESETLOGS であることを示します。オンライン・ログが使用可能な場合のみ、この設定は有効です。

オンライン・ログの今後の状態が予測できない場合は、RESETLOGS も NORESETLOGS も指定しないでください。この場合、両方のバージョンのスクリプトがトレース・ファイルに入れられるため、スクリプトが必要になった時点で、適切なバージョンを選択できます。

standby_database_clauses

この句を使用すると、スタンバイ・データベースをアクティブにする、またはプロテクト・モードと非プロテクト・モードのいずれかを指定することができます。

参照： 物理および論理スタンバイ・データベース、およびスタンバイ・データベースの管理と使用の詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

ACTIVATE STANDBY DATABASE 句

ACTIVATE STANDBY DATABASE 句を指定すると、スタンバイ・データベースの状態がアクティブ・データベースに変更され、プライマリ・データベースに設定することができます。この句を指定する前に、データベースをマウントしておく必要があります。

PHYSICAL PHYSICAL を指定すると、物理スタンバイ・データベースをアクティブにできます。これはデフォルト値です。

LOGICAL LOGICAL を指定すると、論理スタンバイ・データベースをアクティブにできます。論理スタンバイ・データベースが複数ある場合は、すべてのスタンバイ・システムで同じログ・データが使用可能であることを確認する必要があります。

SKIP [STANDBY LOGFILE] この句は、物理スタンバイ・データベースのみに適用されます。この句を使用すると、スタンバイ REDO ログ・ファイルに、RECOVER MANAGED STANDBY DATABASE FINISH コマンドを使用してリカバリできるデータが含まれている場合でも、操作が強制処理されます。

注意： スタンバイ REDO ログ・ファイルを使用しない場合でも、物理スタンバイに対して RECOVER MANAGED STANDBY DATABASE FINISH 文を使用することをお勧めします。スタンバイ REDO ログの内容を廃棄する場合のみ、SKIP 句を使用してください。

SET STANDBY [DATABASE] 句

この句を使用すると、データベース環境でのデータの保護レベルを指定できます。この句は、プライマリ・データベースから指定します。この場合、プライマリ・データベースはマウントされている状態で、オープンされていない状態である必要があります。

注意： PROTECTED および UNPROTECTED キーワードは、説明上置き換えられていますが、サポートされています。PROTECTED は、TO MAXIMIZE PROTECTION と同等です。UNPROTECTED は、TO MAXIMIZE PERFORMANCE と同等です。

TO MAXIMIZE PROTECTION この設定は、最大保護モードを確立し、最高レベルのデータ保護を実現します。トランザクションのリカバリに必要なすべてのデータが、SYNC ログ転送モードを使用するように構成された 1 つ以上の物理スタンバイ・データベースに書き込まれるまで、そのトランザクションはコミットされません。1 つ以上のスタンバイ・データベースに REDO レコードを書き込めない場合、プライマリ・データベースは停止します。このモードでは、データ非消失は保証されますが、プライマリ・データベースのパフォーマンスおよび可用性に最大の影響を与える可能性があります。

TO MAXIMIZE AVAILABILITY この設定は、最大可用性モードを確立し、2 番目に高いレベルのデータ保護を実現します。トランザクションのリカバリに必要なすべてのデータが、SYNC ログ転送モードを使用するように構成された 1 つ以上の物理または論理スタンバイ・データベースに書き込まれるまで、そのトランザクションはコミットされません。最大保護モードとは異なり、1 つ以上のスタンバイ・データベースに REDO レコードを書き込めない場合でも、プライマリ・データベースは停止しません。障害が修正され、スタンバイ・データベースがプライマリ・データベースと同一になるまで、データ保護レベルは最大パフォーマンス・モードまで下げられます。最大パフォーマンス・モードのプライマリ・データベースに障害が発生しないかぎり、このモードによってデータ非消失が保証されます。最大可用性モードは、プライマリ・データベースの可用性に影響を与えずに、最大レベルのデータ保護を実現します。

TO MAXIMIZE PERFORMANCE この設定は、最大パフォーマンス・モードを確立します。これがデフォルトの設定です。トランザクションのリカバリに必要なデータが、スタンバイ・データベースにすべて書き込まれる前に、トランザクションはコミットされます。そのため、プライマリ・データベースに障害が発生し、プライマリ・データベースから REDO レコードをリカバリできない場合は、一部のトランザクションが失われることがあります。このモードは、プライマリ・データベースのパフォーマンスに影響を与えずに、最大レベルのデータ保護を実現します。

データベースの現行モードを確認するには、V\$DATABASE 動的パフォーマンス・ビューの PROTECTION_MODE 列を問い合わせます。

参照： スタンバイ・データベースのこれらの設定については、『Oracle9i Data Guard 概要および管理』を参照してください。

REGISTER LOGFILE 句

スタンバイ・データベースで REGISTER LOGFILE 句を指定すると、障害が発生したプライマリ・データベースからログ・ファイルを手動で登録することができます。

論理スタンバイ・データベースの場合は、このコマンドを使用して新しい論理スタンバイ・データベースの最初の開始点をシードできます。その後、ALTER DATABASE START LOGICAL STANDBY APPLY INITIAL コマンドを発行すると、最下位の登録済ログ・ファイルが開始点として使用されます。

OR REPLACE OR REPLACE を指定すると、たとえば位置やファイル仕様が変更された場合に、スタンバイ・データベースの既存のアーカイブ・ログ・エントリが更新されます。エントリの SCN は、正確に一致している必要があります。また、元のエントリは、管理スタンバイ・ログの送信メカニズムによって作成される必要があります。

COMMIT TO SWITCHOVER 句

この句を使用すると、正常なスイッチオーバーを実行できます。このスイッチオーバーでは、現行のプライマリ・データベースがスタンバイ状態になり、1 つのスタンバイ・データベースがプライマリ・データベースになります。Real Application Clusters 環境では、この文を発行するインスタンス以外のすべてのインスタンスが正常に停止する必要があります。

- **PHYSICAL** を指定すると、プライマリ・データベースが物理スタンバイ・データベースのロールで実行できる状態になります。
- **LOGICAL** を指定すると、プライマリ・データベースが論理スタンバイ・データベースのロールで実行できる状態になります。**LOGICAL** を指定した場合は、**ALTER DATABASE START LOGICAL STANDBY APPLY** 文を発行する必要があります。
- プライマリ・データベースで **COMMIT TO SWITCHOVER TO STANDBY** を指定すると、正常なスイッチオーバーが実行され、プライマリ・データベースがスタンバイ・データベース状態になります。プライマリ・データベースは、オープンされている必要があります。
- スタンバイ・データベースで **COMMIT TO SWITCHOVER TO PRIMARY** 文を発行すると、正常なスイッチオーバーが実行され、このスタンバイ・データベースがプライマリ状態になります。スタンバイ・データベースは、マウントされているか、または **READ ONLY** モードでオープンされている必要があります。

WITH | WITHOUT SESSION SHUTDOWN **WITH SESSION SHUTDOWN** を指定すると、開いているすべてのアプリケーション・セッションは停止し、コミットされていないトランザクションが、この文の実行の一部としてロールバックされます。この句を省略するか、または **WITHOUT SESSION SHUTDOWN**（デフォルト）を指定した場合、開いているアプリケーション・セッションがあると、文は実行されません。

WITH SESSION SHUTDOWN の制限事項： この句は、論理データベースには不要です。論理データベースではサポートされません。

WAIT | NOWAIT **WAIT** を指定すると、**SWITCHOVER** コマンドが完了した後、制御が戻ります。**NOWAIT** を指定すると、スイッチオーバー操作が完了する前に、制御が戻ります。デフォルトは **WAIT** です。

参照： プライマリ・データベースとスタンバイ・データベース間の正常なスイッチオーバーの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

START LOGICAL STANDBY APPLY 句

START LOGICAL STANDBY APPLY 句を指定すると、論理スタンバイ・データベースへの **REDO** ログの適用が開始します。

- 初めてスタンバイ・データベースにログを適用する場合は、**INITIAL** を指定します。
- **ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY** 文の後、またはスタンバイ・データベースが1つのプライマリ・データベースからのログの処理を完了し、新しいデータベースがプライマリ・データベースになる場合は、**NEW PRIMARY** を指定します。

STOP | ABORT LOGICAL STANDBY 句

この句を使用すると、ログ適用サービスが停止されます。この句は、物理スタンバイ・データベースではなく、論理スタンバイ・データベースのみに適用されます。STOP 句を使用すると、適用が正常に停止されます。

default_settings_clauses

データベースのデフォルト設定を変更できます。

CHARACTER SET、NATIONAL CHARACTER SET

CHARACTER SET は、データベースでデータを格納する場合に使用するキャラクタ・セットを変更します。NATIONAL CHARACTER SET は、NCHAR、NCLOB または NVARCHAR2 と定義された列にデータを格納する場合に使用する各国語キャラクタ・セットを変更します。*character_set* は、引用符なしで指定します。データベースは、オープンされている必要があります。

注意：

- ALTER DATABASE CHARACTER SET 文または ALTER DATABASE NATIONAL CHARACTER SET 文はロールバックできません。そのため、それらの文を発行する前に、データベース全体のバックアップを取ってください。
- 既存データベースのキャラクタ・セットを新しいデータベースのキャラクタ・セットへ移行する前に、キャラクタ・セット・スキャナ (CSSCAN) を使用して、データを分析することをお勧めします。これによって、データベースで認識されない非 ASCII データの損失を防ぐことができます。CSSCAN の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

キャラクタ・セット変更の注意：

Oracle9i では、マルチバイト・データベース・キャラクタ・セットの場合、CLOB データは、UCS-2 (2 バイト固定幅の Unicode) として格納されます。シングルバイト・データベース・キャラクタ・セットの場合、CLOB データはデータベース・キャラクタ・セットに格納されます。ALTER DATABASE 文を使用して、データベースまたは各国語キャラクタ・セットを変更する場合は、データ変換は行われません。そのため、この文を使用してシングルバイトからマルチバイトへデータベース・キャラクタ・セットを変更しても、CLOB 列は元のデータベース・キャラクタ・セットのままです。これは CLOB 列のデータの非一貫性です。同様に、各国語キャラクタ・セットを Unicode からその他に変更すると、SQL の NCHAR 列 (NCHAR, NVARCHAR2 および NCLOB) は破損します。

次の手順でデータベース・キャラクタ・セットを変更することをお薦めします。

1. CLOB および SQL の NCHAR データ型の列をエクスポートします。
2. CLOB および SQL の NCHAR 列を含む表を削除します。
3. ALTER DATABASE 文を使用して、キャラクタ・セットおよび各国語キャラクタ・セットを変更します。
4. CLOB および SQL の NCHAR 列を再インポートします。

キャラクタ・セット変更の制限事項：

- SYSDBA システム権限が必要です。また、データベースは、制限モードで（たとえば、SQL*Plus の STARTUP RESTRICT コマンドを使用して）起動してください。
- 現在のキャラクタ・セットは、変更するキャラクタ・セットの厳密なサブセットである必要があります。ソース・キャラクタ・セットのコードポイント値で表される各キャラクタは、ターゲット・キャラクタ・セットの同一コードポイント値で表される必要があります。

参照： データベース・キャラクタ・セットの移行の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。9-53 ページの「[キャラクタ・セットの変更例:](#)」も参照してください。

set_time_zone_clause

SET TIME_ZONE 句を使用すると、データベースのタイムゾーンを設定できます。次の 2 つの方法でタイムゾーンを設定します。

- UTC からの時差を指定。hh:mm の有効範囲は、-12:00 ～ +14:00 です。
- タイムゾーン地域を指定。有効な地域名を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューの TZNAME 列を問い合わせます。

注意： データベースのタイムゾーンを UTC ("0:00") に設定することをお薦めします。これによって、タイムゾーンの変換が不要になるため、特にデータベース間のパフォーマンスを向上できます。

参照： 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

すべての新しい TIMESTAMP WITH LOCAL TIME ZONE データは、ディスクに格納されるときにデータベースのタイムゾーンに正規化されます。データベースの既存のデータは、自動的に新しいタイムゾーンに更新されません。データベースのタイムゾーンを決定するには、組み込み関数 DBTIMEZONE を問い合わせます（6-48 ページの「[DBTIMEZONE](#)」を参照）。

この句を使用してタイムゾーンを設定または変更した後、新しいタイムゾーンを有効にするためにデータベースを再起動する必要があります。

DEFAULT TEMPORARY TABLESPACE 句

データベースのデフォルトの一時表領域を変更します。この操作の完了後、以前のデフォルトの一時表領域が割り当てられているすべてのユーザーに、新しいデフォルトの一時表領域が再度割り当てられます。その後、必要に応じて以前のデフォルトの一時表領域を削除することができます。

現行のデフォルトの一時表領域の名前を確認するには、DATABASE_PROPERTIES データ・ディクショナリ表の PROPERTY_VALUE 列を、「PROPERTY_NAME = 'DEFAULT_TEMP_TABLESPACE'」という条件で問い合わせます。

デフォルトの一時表領域の制限事項：

- デフォルトの一時表領域として割り当てる表領域、または再度割り当てる表領域は、標準的なブロック・サイズである必要があります。
- SYSTEM 表領域がローカル管理される場合、デフォルトの一時表領域として指定する表領域も、ローカル管理される必要があります。

参照： 9-53 ページの「[デフォルトの一時表領域の定義例](#)」を参照してください。

conversion_clauses

RESET COMPATIBILITY 句

RESET COMPATIBILITY を指定すると、次に再起動する際、Oracle の以前のバージョンに再設定する必要があるデータベースにマークが付けられます。データベースは、オープンされている必要があります。

注意： RESET COMPATIBILITY は、下位互換性に影響する Oracle の機能を使用禁止にしている場合にのみ有効です。

参照： Oracle の以前のバージョンへのダウングレードの詳細は、『Oracle9i データベース移行ガイド』を参照してください。

CONVERT 句

CONVERT 句を使用すると、Oracle7 のデータ・ディクショナリが変換されます。この句を使用すると、Oracle7 のデータ・ディクショナリは、Oracle データベース内に存在しなくなります。

注意： この句は、Oracle9i へ移行する場合にのみ使用し、データベースがマウントされている場合は使用しないでください。

参照： 『Oracle9i データベース移行ガイド』を参照してください。

redo_thread_clauses

REDO ログ・ファイル・グループのスレッドを使用可能または使用禁止にできます。

ENABLE THREAD 句

Oracle Real Application Clusters 環境において、ENABLE THREAD を指定すると、REDO ログ・ファイル・グループの指定スレッドが使用可能になります。使用可能にできるスレッドは、2 つ以上の REDO ログ・ファイルを持つスレッドのみです。データベースは、オープンされている必要があります。

PUBLIC PUBLIC を指定すると、初期化パラメータ THREAD で特定のスレッドを明示的に要求していないインスタンスに対して、使用可能になったスレッドを使用できるようにします。PUBLIC を指定しないと、初期化パラメータ THREAD を使用して明示的に要求するインスタンスのみがこのスレッドを使用できます。

参照： スレッドを使用可能または使用禁止にする場合の詳細は、『Oracle9i Real Application Clusters 管理』を参照してください。

DISABLE THREAD 句

DISABLE THREAD を指定すると、指定スレッドをすべてのインスタンスで使用禁止にできます。データベースは、オープンされている必要がありますが、指定したスレッドを使用しているインスタンスが、データベースをマウント済の場合は、そのスレッドを使用禁止にできません。

参照： スレッドを使用可能または使用禁止にする場合の詳細は、『Oracle9i Real Application Clusters 管理』を参照してください。9-53 ページの「[Real Application Clusters スレッドを使用可能および使用禁止にする例](#)」も参照してください。

RENAME GLOBAL_NAME 句

RENAME GLOBAL_NAME を指定すると、データベースのグローバル名を変更できます。database には、データベースの新しい名前を 8 バイト以内の長さで指定します。オプションの domain には、ネットワーク階層におけるデータベースの有効な位置を指定します。データベースがマウントされている場合、この句は使用できません。

注意： データベース名を変更しても、リモート・データベース内の既存のデータベース・リンク、シノニム、ストアド・プロシージャ、ストアド・ファンクションからのユーザーのデータベースに対するグローバル参照は変更されません。これらの参照を変更するのは、リモート・データベース管理者の責任です。

参照： グローバル名の詳細は、『Oracle9i Heterogeneous Connectivity Administrator's Guide』を参照してください。9-53 ページの「[グローバル・データベース名の変更例](#)」を参照してください。

security_clause

security_clause (GUARD) を使用すると、データベースのデータが変更されないように保護できます。

ALL ALL を指定すると、SYS 以外のすべてのユーザーは、データベースのデータを変更できなくなります。

STANDBY STANDBY を指定すると、SYS 以外のすべてのユーザーは、論理スタンバイで管理されるデータベース・オブジェクトを変更できなくなります。この設定は、論理スタンバイによってレプリケートされる前に、データをレポート操作によって変更できるようにする場合に便利です。

参照： 論理スタンバイの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

NONE NONE を指定すると、データベースのすべてのデータについて、通常のセキュリティを設定できます。

例

READ ONLY | READ WRITE の例： 次の最初の文は、データベースを読み込み専用モードでオープンします。2 番目の文は、データベースを読み書き両用モードに戻し、オンライン REDO ログを消去します。

```
ALTER DATABASE OPEN READ ONLY;
```

```
ALTER DATABASE OPEN READ WRITE RESETLOGS;
```

パラレル・リカバリ処理の使用例： 次の文は、パラレル・リカバリ処理を使用して表領域のリカバリを行います。

```
ALTER DATABASE
  RECOVER TABLESPACE tbs_03
  PARALLEL;
```

REDO ログ・ファイル・グループの追加例： 次の文は、2つのメンバーを含む REDO ログ・ファイル・グループを追加し、GROUP パラメータの値に 3 を指定してこのグループを識別します。

```
ALTER DATABASE
  ADD LOGFILE GROUP 3
    ('diska:log3.log' ,
     'diskb:log3.log') SIZE 50K;
```

次の文は、2つのメンバーを含む REDO ログ・ファイル・グループをスレッド 5 (Real Application Clusters 環境内) に追加して、このグループに GROUP パラメータ値 4 を割り当てます。

```
ALTER DATABASE
  ADD LOGFILE THREAD 5 GROUP 4
    ('diska:log4.log',
     'diskb:log4:log');
```

ログ・ファイル・メンバーの削除例： 次の文は、前述の例で追加した REDO ログ・ファイル・メンバーの 1 つを削除します。

```
ALTER DATABASE
  DROP LOGFILE MEMBER 'diskb:log3.log';
```

次の文は、REDO ログ・ファイル・グループ 3 のすべてのメンバーを削除します。

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

REDO ログ・ファイル・グループ・メンバーの追加例： 次の文は、前述の例で追加した REDO ログ・ファイル・グループに 1 つのメンバーを追加します。

```
ALTER DATABASE
  ADD LOGFILE MEMBER 'diskc:log3.log'
  TO GROUP 3;
```

ログ・ファイル・メンバーの名前の変更例： 次の文は、REDO ログ・ファイル・メンバーの名前を変更します。

```
ALTER DATABASE
  RENAME FILE 'diskc:log3.log' TO 'diskb:log3.log';
```

この例では、REDO ログ・グループ・メンバーのファイルが、別のファイル名に変更されただけです。ファイル名が、実際に 'diskc:log3.log' から 'diskb:log3.log' に変更されたわけではありません。実際のファイル名を変更するには、オペレーティング・システムから操作する必要があります。

デフォルトの一時表領域の定義例： 次の文は、データベースのデフォルトの一時表領域 tbs_5 を作成します。この文は、作成時に何も指定されていない場合にデフォルトの一時表領域を作成するか、既存のデフォルトの一時表領域を temp で置き換えます。

```
ALTER DATABASE
  DEFAULT TEMPORARY TABLESPACE tbs_5;
```

Real Application Clusters スレッドを使用可能および使用禁止にする例： 次の文は、Real Application Clusters 環境のスレッド 5 を使用禁止にします。

```
ALTER DATABASE
  DISABLE THREAD 5;
```

次の文は、Real Application Clusters 環境のスレッド 5 を使用可能にして、特定のスレッドを明示的に要求しない任意の Oracle インスタンスが、このスレッドを使用できるようにします。

```
ALTER DATABASE
  ENABLE PUBLIC THREAD 5;
```

新しいデータ・ファイルの作成例： 次の文は、ファイル 'tabspace_file03.dbf' を基に新しいデータ・ファイル 'tabspace_file04.dbf' を作成します。新しいデータ・ファイルを作成する前に、既存のデータ・ファイル（またはこのデータ・ファイルが存在する表領域）をオフラインにする必要があります。

```
ALTER DATABASE
  CREATE DATAFILE 'tabspace_file03.dbf'
  AS 'tabspace_file04.dbf';
```

グローバル・データベース名の変更例： 次の文は、データベースのグローバル名を変更し、データベース名とドメインの両方を指定します。

```
ALTER DATABASE
  RENAME GLOBAL_NAME TO demo.world.oracle.com;
```

キャラクタ・セットの変更例： 次の文は、データベース・キャラクタ・セットおよび各国語キャラクタ・セットを UTF8 キャラクタ・セットに変更します。

```
ALTER DATABASE CHARACTER SET UTF8;
ALTER DATABASE NATIONAL CHARACTER SET UTF8;
```

データベース名はオプションです。キャラクタ・セット名は引用符を付けずに指定します。

データ・ファイルのサイズの変更例： 次の文は、データ・ファイル 'disk1:db1.dat' のサイズを変更します。

```
ALTER DATABASE
    DATAFILE 'disk1:db1.dat' RESIZE 10 M;
```

ログ・ファイルの消去例： 次の文は、ログ・ファイルを消去します。

```
ALTER DATABASE
    CLEAR LOGFILE 'diskc:log3.log';
```

データベースのリカバリ例： 次の文は、データベース全体を完全にリカバリし、必要な新しいアーカイブ REDO ログ・ファイル名を生成します。

```
ALTER DATABASE
    RECOVER AUTOMATIC DATABASE;
```

次の文は、Oracle が適用する REDO ログ・ファイル名を明示的に指定します。

```
ALTER DATABASE
    RECOVER LOGFILE 'diskc:log3.log';
```

次の文は、元のスタンバイ・データベース内の対応するデータ・ファイル、関連するアーカイブ・ログおよび現行のスタンバイ・データベース制御ファイルを使用して、スタンバイ・データ・ファイル /finance/stbs_21.f をリカバリします。

```
ALTER DATABASE
    RECOVER STANDBY DATAFILE '/finance/stbs_21.f'
    UNTIL CONTROLFILE;
```

次の文は、時間ベースのデータベース・リカバリを実行します。

```
ALTER DATABASE
    RECOVER AUTOMATIC UNTIL TIME '2001-10-27:14:00:00';
```

データベースは、2001 年 10 月 27 日の午後 2 時の状態にリカバリされます。

表領域のリカバリ例は、9-52 ページの「[パラレル・リカバリ処理の使用例:](#)」を参照してください。

管理スタンバイ・データベースのリカバリ例： 次の文は、管理スタンバイ・リカバリ・モードでスタンバイ・データベースをリカバリします。

```
ALTER DATABASE
    RECOVER MANAGED STANDBY DATABASE;
```

次の文は、データベースを管理スタンバイ・リカバリ・モードにします。管理リカバリ処理は、次のアーカイブ・ログまで最長 60 分間待ち状態になります。

```
ALTER DATABASE
```

```
    RECOVER MANAGED STANDBY DATABASE TIMEOUT 60;
```

後続のログがその前のログから 60 分以内に作成される場合、手動で終了しないかぎり、リカバリが継続されます。

次の文は、管理リカバリ操作を終了します。

```
ALTER DATABASE
```

```
    RECOVER MANAGED STANDBY DATABASE CANCEL IMMEDIATE;
```

管理リカバリ操作は、現行の REDO ログ・ファイルから REDO の次のグループが読み込まれる前に終了します。メディア・リカバリは、現行の REDO ログ・ファイルから REDO を適用している間に終了します。

ALTER DIMENSION

用途

ALTER DIMENSION 文を使用すると、ディメンションの階層関係またはディメンション属性を変更できます。

参照： ディメンションの詳細は、13-41 ページの「[CREATE DIMENSION](#)」を参照してください。

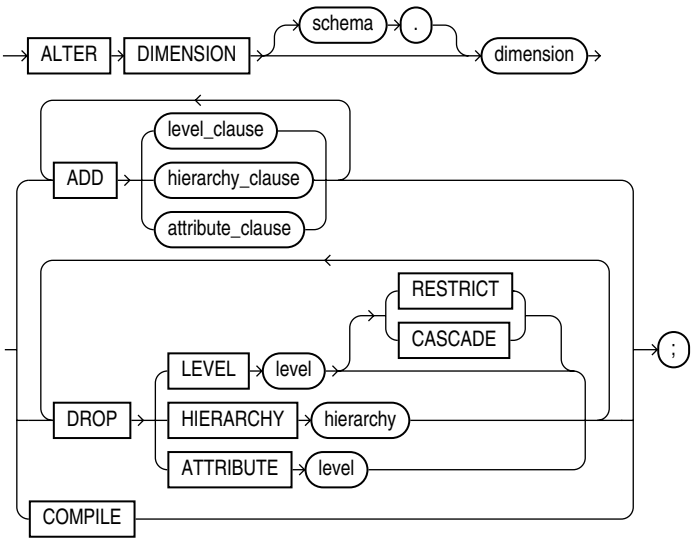
前提条件

この文を使用する場合、ディメンションが自分のスキーマ内に設定されているか、または ALTER ANY DIMENSION システム権限が必要です。

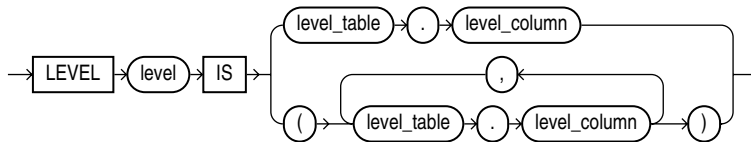
所有者の権限があれば、ディメンションはいつでも変更されます。

構文

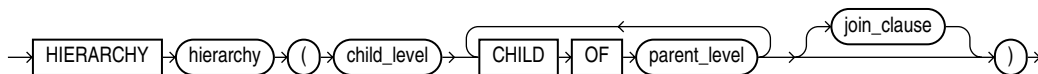
alter_dimension::=



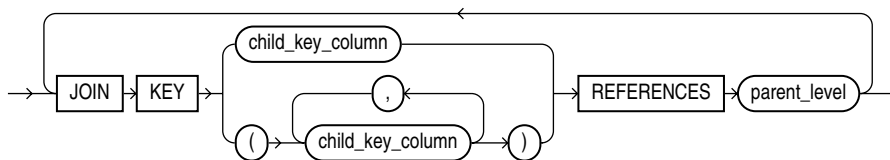
level_clause::=



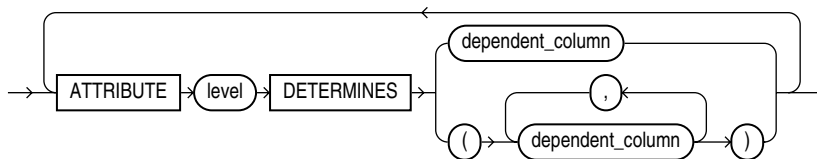
hierarchy_clause::=



join_clause::=



attribute_clause::=



セマンティクス

次のキーワード、パラメータおよび句の意味は、ALTER DIMENSION に対してのみ有効です。その他のキーワード、パラメータおよび句には、CREATE DIMENSION 文での機能と同じ機能があります。

参照： 13-41 ページの「[CREATE DIMENSION](#)」を参照してください。

schema

変更するディメンションのスキーマを指定します。*schema* を指定しない場合、ディメンションが自分のスキーマ内にあるとみなされます。

dimension

ディメンション名を指定します。ディメンションは、すでに存在する必要があります。

ADD

ADD 句を使用すると、ディメンションにレベル、階層または属性を追加できます。これらの要素の 1 つを追加しても、既存のマテリアライズド・ビューは無効になりません。

ADD LEVEL 句は、他の ADD 句より前に処理されます。

DROP

DROP 句を使用すると、ディメンションからレベル、階層または属性を削除できます。指定するすべてのレベル、階層または属性は、すでに存在している必要があります。

DROP の制限事項： 属性または階層がレベルを参照する場合は、すべての参照している属性および階層を削除するか、または CASCADE を指定しないかぎり、このレベルを削除することはできません。

CASCADE レベルを参照する属性または階層をレベルとともに削除する場合は、CASCADE を指定します。

RESTRICT 属性または階層が参照するレベルを削除できないようにする場合は、RESTRICT を指定します。これはデフォルト値です。

COMPILE

COMPILE を指定すると、無効のディメンションを明示的に再コンパイルできます。ADD 句または DROP 句が発行されると、ディメンションは自動的にコンパイルされます。ただし、ディメンションが参照するオブジェクトを変更する（たとえば、ディメンションで参照された表を削除した後再作成する）と、ディメンションが無効になるため、これを明示的に再コンパイルする必要があります。

例

ディメンションの変更例： 次の例は、サンプル・スキーマ sh の customers_dim ディメンションを変更します。

```
ALTER DIMENSION customers_dim
  DROP ATTRIBUTE country;
```

```
ALTER DIMENSION customers_dim
  ADD LEVEL zone IS customers.cust_postal_code
  ADD ATTRIBUTE zone DETERMINES (cust_city);
```


ALTER FUNCTION

用途

ALTER FUNCTION 文を使用すると、無効なスタンドアロンのストアド・ファンクションを再コンパイルできます。明示的に再コンパイルすることによって、実行時に暗黙的に再コンパイルする必要がなくなり、また、実行時のコンパイル・エラーとパフォーマンス上のオーバーヘッドもなくなります。

ALTER FUNCTION 文は、9-122 ページの「[ALTER PROCEDURE](#)」と似ています。ファンクションおよびプロシージャの再コンパイルの詳細は、『Oracle9i データベース概要』を参照してください。

注意： この文を使用して既存のファンクションの宣言や定義を変更することはできません。ファンクションを再宣言または再定義する場合は、OR REPLACE 句を指定して CREATE FUNCTION 文を使用します。詳細は、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。

前提条件

ファンクションが自分のスキーマ内にあるか、ALTER ANY PROCEDURE システム権限が必要です。

構文

alter_function::=



セマンティクス

schema

ファンクションが含まれているスキーマを指定します。*schema* を指定しない場合、ファンクションが自分のスキーマ内に定義されているものとみなされます。

function

再コンパイルするファンクション名を指定します。

COMPILE

COMPILE を指定すると、ファンクションが再コンパイルされます。COMPILE キーワードは必須です。ファンクションが正常にコンパイルされない場合、SQL*Plus コマンド SHOW ERRORS を使用すると、関連するコンパイラ・エラー・メッセージが表示されます。

再コンパイル中、Oracle はすべての永続コンパイラのスイッチ設定を削除し、セッションからそれらを再び取得してコンパイルの終了時に格納します。この手順を回避するには、REUSE SETTINGS 句を指定します。

DEBUG

DEBUG を指定すると、PL/SQL コンパイラに対して、PL/SQL デバッガ用のコードを生成および格納するように指示できます。

REUSE SETTINGS

REUSE SETTINGS を指定すると、Oracle によるコンパイラ・スイッチ設定の削除および再取得を回避できます。この句では既存の設定が持続され、その設定で再コンパイルされます。

DEBUG と REUSE SETTINGS の両方を指定する場合は、PLSQL_COMPILER_FLAGS パラメータの永続格納値が INTERPRETED, DEBUG に設定されます。その他のコンパイラ・スイッチ値は変更されません。

参照： PLSQL_COMPILER_FLAGS パラメータと COMPILE 句の相互作用の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

例

ファンクションの再コンパイル例： 次の文は、サンプル・ユーザー oe が所有するファンクション get_bal を明示的に再コンパイルします。

```
ALTER FUNCTION oe.get_bal  
    COMPILE;
```

get_bal の再コンパイル時にエラーが発生しなければ、get_bal は有効になります。その後、Oracle は、実行時にそれを再コンパイルしなくても実行できます。get_bal の再コンパイル時にコンパイラ・エラーが発生した場合はエラーが戻り、get_bal は無効のままです。

また、get_bal に依存しているすべてのオブジェクトが無効になります。その後、明示的に再コンパイルせずに、これらのオブジェクトを参照した場合、Oracle は、実行時にそれらを暗黙的に再コンパイルします。

ALTER INDEX

用途

ALTER INDEX 文を使用すると、既存の索引を変更または再作成できます。

参照： 索引の作成については、13-61 ページの「[CREATE INDEX](#)」を参照してください。

前提条件

索引が自分のスキーマ内にあるか、または ALTER ANY INDEX システム権限が必要です。

MONITORING USAGE 句を実行する場合は、索引は自分のスキーマ内に存在する必要があります。

ドメイン索引を変更する場合は、索引の索引タイプに対して EXECUTE オブジェクト権限が必要です。

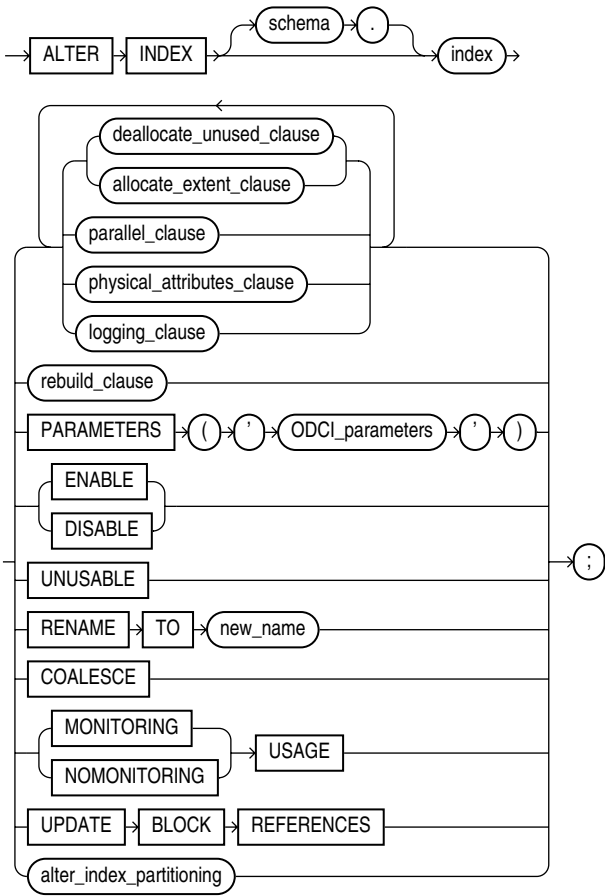
スキーマ・オブジェクト権限は、個々の索引パーティションまたはサブパーティションではなく、親索引に付与されている必要があります。

索引パーティションの変更、再作成または分割、索引サブパーティションの変更または再作成を行う場合は、表領域割当て制限が必要です。

参照： ドメイン索引については、13-61 ページの「[CREATE INDEX](#)」および『Oracle9i Data Cartridge Developer's Guide』を参照してください。

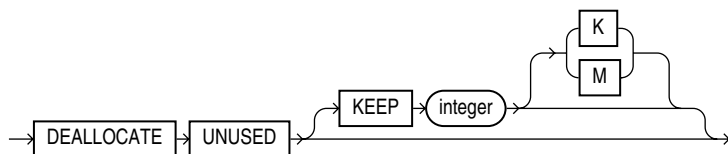
構文

alter_index::=

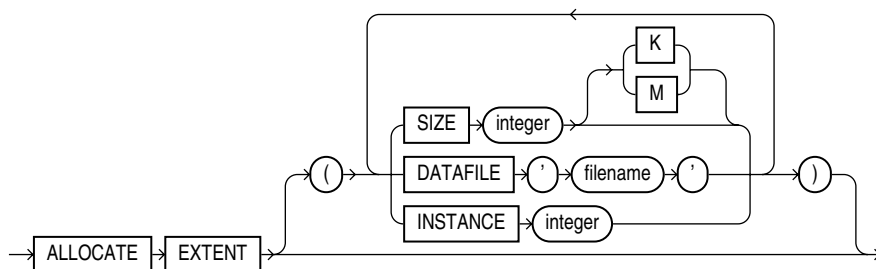


(9-63 ページの「[deallocate_unused_clause::=](#)」、9-63 ページの「[allocate_extent_clause::=](#)」、9-63 ページの「[parallel_clause::=](#)」、9-63 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、9-64 ページの「[rebuild_clause::=](#)」、9-65 ページの「[alter_index_partitioning::=](#)」を参照)

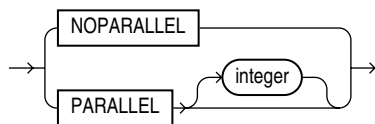
deallocate_unused_clause::=



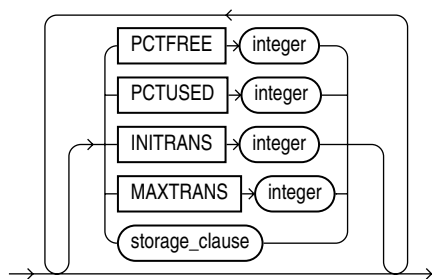
allocate_extent_clause::=



parallel_clause::=

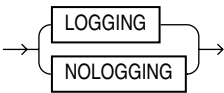


physical_attributes_clause::=

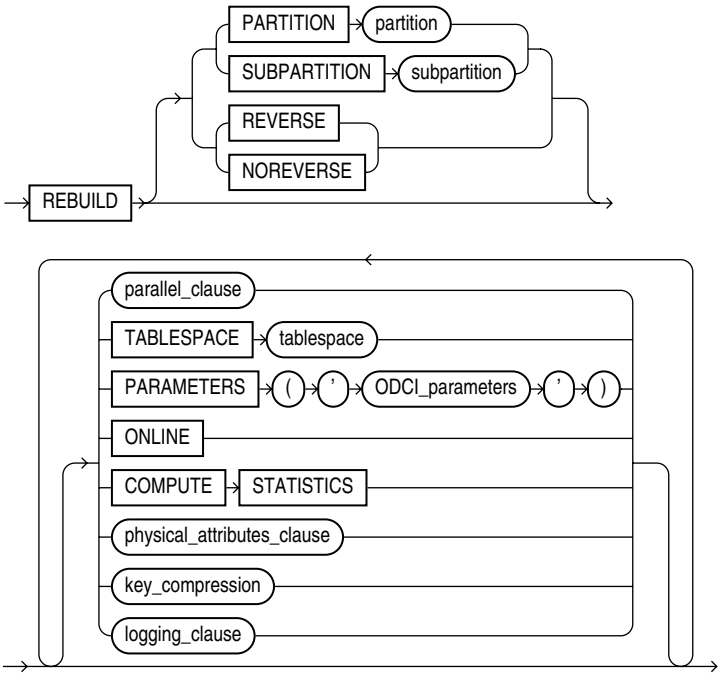


(7-53 ページの「[storage_clause::=](#)」を参照)

logging_clause::=

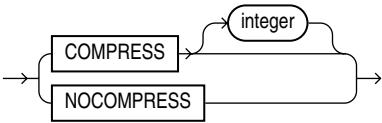


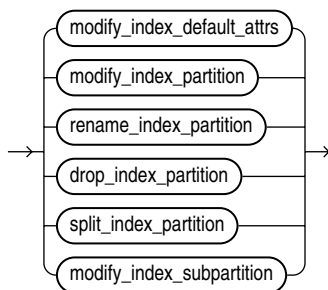
rebuild_clause::=



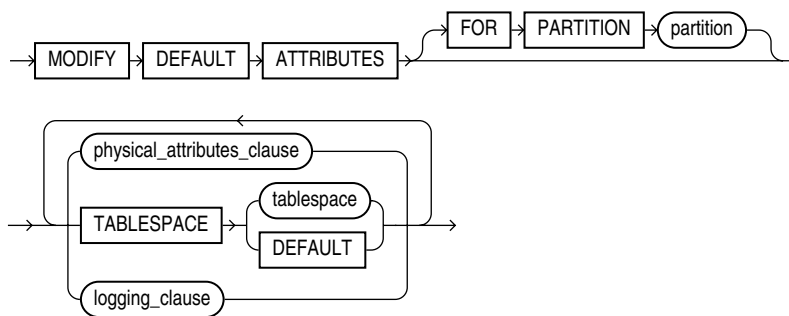
(9-63 ページの「[physical_attributes_clause::=](#)」、9-64 ページの「[key_compression::=](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

key_compression::=



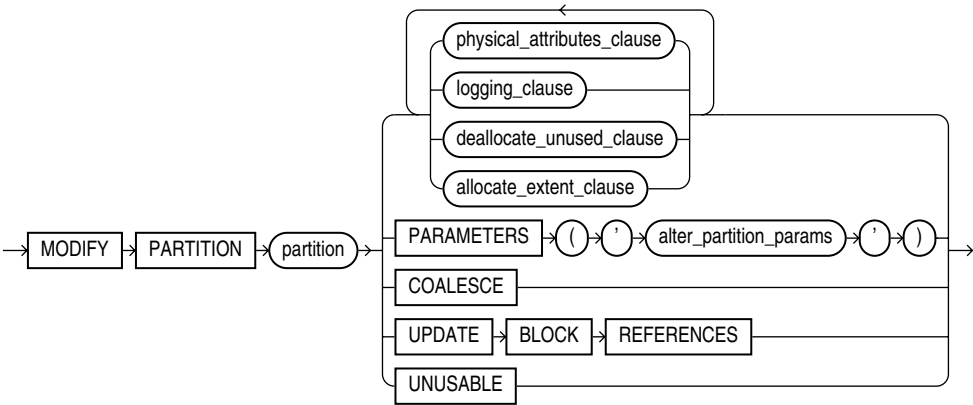
alter_index_partitioning::=

(9-65 ページの「[modify_index_default_attrs::=](#)」、9-66 ページの「[modify_index_partition::=](#)」、9-66 ページの「[rename_index_partition::=](#)」、9-66 ページの「[drop_index_partition::=](#)」、9-66 ページの「[split_index_partition::=](#)」、9-67 ページの「[modify_index_subpartition::=](#)」を参照)

modify_index_default_attrs::=

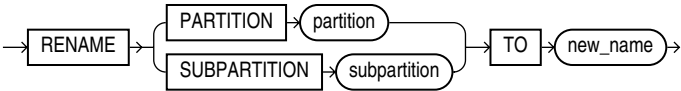
(9-63 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

modify_index_partition::=



(9-63 ページの「`physical_attributes_clause::=`」、7-42 ページの「`logging_clause::=`」、9-63 ページの「`allocate_extent_clause::=`」、9-63 ページの「`deallocate_unused_clause::=`」を参照)

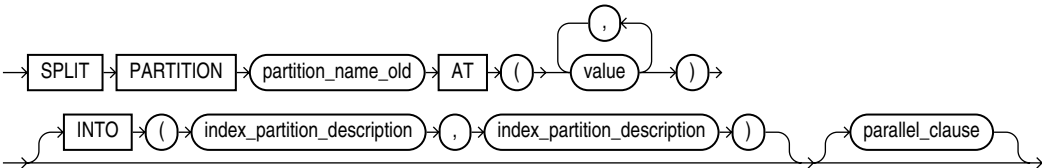
rename_index_partition::=



drop_index_partition::=

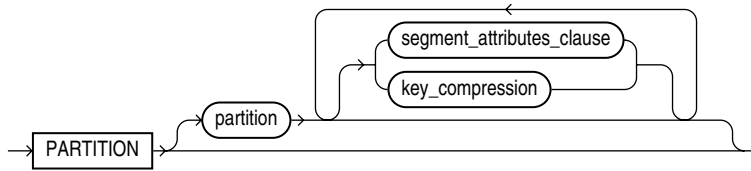


split_index_partition::=



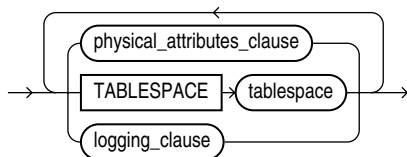
(9-63 ページの「`parallel_clause::=`」を参照)

index_partition_description::=



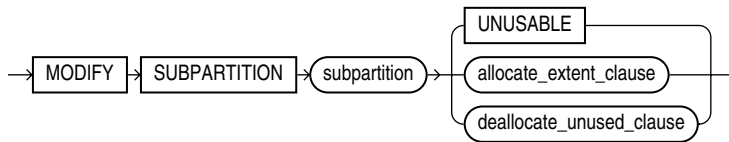
(9-64 ページの「[key_compression::=](#)」を参照)

segment_attributes_clause::=



(9-63 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

modify_index_subpartition::=



(9-63 ページの「[allocate_extent_clause::=](#)」、9-63 ページの「[deallocate_unused_clause::=](#)」を参照)

セマンティクス

schema

索引が含まれているスキーマを指定します。*schema* を指定しない場合、索引が自分のスキーマ内に定義されているものとみなされます。

index

変更する索引の名前を指定します。

索引変更の制限事項：

- `index` がドメイン索引である場合は、`PARAMETERS` 句、`RENAME` 句、`rebuild_clause` (`PARAMETERS` 句の有無に関係なく)、`parallel_clause` または `UNUSABLE` 句のみ指定できます。その他のすべての句は無効です。
- `LOADING` または `FAILED` のマークが付いているドメイン索引は、変更または名前の変更ができません。索引に `FAILED` のマークが付いている場合、`REBUILD` 句のみ指定できます。

参照： ドメイン索引の `LOADING` および `FAILED` 状態については、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

`deallocate_unused_clause`

`deallocate_unused_clause` 句を使用すると、索引の終わりの未使用領域の割当てを明示的に解除し、解放された領域が表領域内の他のセグメントで使用可能になります。

`index` がレンジ・パーティションまたはハッシュ・パーティションである場合、各索引パーティションの未使用領域の割当てが解除されます。`index` がコンポジット・パーティション表のローカル索引である場合、各索引サブパーティションの未使用領域の割当てが解除されます。

領域の割当て解除の制限事項：

- この句は、一時表の索引に対して指定できません。
- この句および `rebuild_clause` は、指定できません。

参照： この句の詳細は、7-34 ページの「[deallocate_unused_clause](#)」を参照してください。

KEEP integer `KEEP` 句は、割当てを解除した後に索引に残す、最高水位標を超えるバイト数を指定できます。残りのエクステント数が `MINEXTENTS` より少ない場合、`MINEXTENTS` は現行のエクステント数に設定されます。初期エクステントが `INITIAL` より小さくなると、`INITIAL` は初期エクステントの現行の値に設定されます。`KEEP` を指定しないと、すべての未使用領域が解放されます。

参照： この句の詳細は、11-2 ページの「[ALTER TABLE](#)」を参照してください。

`allocate_extent_clause`

`allocate_extent_clause` は、索引の新しいエクステントを明示的に割り当てます。ハッシュ・パーティション表のローカル索引に対して、新規エクステントが索引の各パーティションに割り当てられます。

エクステントの割当ての制限事項： この句は、一時表の索引、レンジ・パーティションまたはコンポジット・パーティション索引に対して指定できません。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。

parallel_clause

PARALLEL 句を使用すると、索引の間合せおよび DML に対するデフォルトの並列度を変更できます。

並列化する索引の制限事項： この句は、一時表の索引に対して指定できません。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である**並列度**を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

参照： 15-51 ページにある CREATE TABLE の「[parallel_clause の注意事項](#)」を参照してください。9-81 ページの「[パラレル間合せを使用可能にする例](#)」も参照してください。

physical_attributes_clause

physical_attributes_clause を使用すると、非パーティション索引、パーティション索引のすべてのパーティションおよびサブパーティション、指定されたパーティション、または指定されたパーティションのすべてのサブパーティションに対するパラメータの値を変更できます。

参照：

- この句のパラメータの詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。
- 9-81 ページの「[実属性の変更例](#)」および 9-82 ページの「[MAXEXTENTS の変更例](#)」を参照してください。

索引の物理属性の制限事項：

- この句は、一時表の索引に対して指定できません。
- 索引の変更中は、PCTUSED パラメータを指定できません。
- PCTFREE パラメータは、*rebuild_clause*、*modify_index_default_attrs* 句または *split_partition_clause* の一部としてのみ指定できます。

storage_clause

storage_clause を使用すると、非パーティション索引、索引パーティション、またはパーティション索引のすべてのパーティションの記憶域パラメータ、あるいはパーティション索引の記憶域パラメータのデフォルト値を変更できます。

参照： 7-51 ページの「[storage_clause](#)」を参照してください。

logging_clause

logging_clause を使用すると、非パーティション索引、レンジまたはハッシュ索引パーティション、コンポジット・パーティション索引のすべてのパーティションまたはサブパーティションに対する後続のダイレクト・ローダー (SQL*Loader) およびダイレクト・パス・インサート操作のログを、REDO ログ・ファイルに記録するか (LOGGING)、記録しないか (NOLOGGING) を指定します。

索引セグメントには、実表の属性と異なるロギング属性、および同じ実表の他の索引セグメントと異なるロギング属性を指定できます。

索引のログの制限事項： この句は、一時表の索引に対して指定できません。

参照：

- この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。
- LOGGING およびパラレル DML の詳細は、『Oracle9i データベース概要』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

RECOVERABLE | UNRECOVERABLE

これらのキーワードは以前のリリースのもので、それぞれ LOGGING および NOLOGGING に置き換えられています。RECOVERABLE および UNRECOVERABLE は、下位互換用のためにサポートされていますが、LOGGING および NOLOGGING キーワードを使用することをお勧めします。

RECOVERABLE は、パーティション表または LOB の記憶特性の作成時には無効なキーワードです。UNRECOVERABLE は、パーティション表または索引構成表の作成時には無効なキーワードです。また、CREATE INDEX の AS 副問合せ句を使用してのみ指定できます。

rebuild_clause

rebuild_clause を使用すると、既存の索引、あるいはパーティションまたはサブパーティションのいずれかを再構築します。索引に UNUSABLE のマークが付いている場合、正常に再構築すると USABLE になります。ファンクション索引も使用可能にします。索引の基になるファンクションが存在しない場合、再構築された文は正確に実行されません。

索引の再構築の制限事項：

- 一時表の索引は、再構築できません。
- INVALID のマークが付いているビットマップ索引は再構築できません。制約を削除してからそれを再作成する必要があります。
- パーティション索引全体は、再構築できません。PARTITION 句で説明するとおり、各パーティションまたはサブパーティションを再構築する必要があります。
- この文の deallocate_unused_clause は指定できません。
- 索引全体 (ALTER INDEX) またはパーティション (ALTER INDEX ... MODIFY PARTITION) に対して、PCTFREE パラメータ値を変更できません。ALTER INDEX 文の他のすべての形式では、PCTFREE を指定できます。
- ドメイン索引の場合
 - 指定できるのは、PARAMETERS 句 (索引または索引のパーティション用) または parallel_clause のみです。その他の再構築の句は無効です。
 - 索引に IN_PROGRESS のマークが付いていない場合にのみ、索引を再構築できます。
 - 索引に IN_PROGRESS または FAILED のマーク、パーティションに IN_PROGRESS のマークが付いていない場合にのみ、索引パーティションを再構築できます。
- ローカル索引は再構築できませんが、ALTER INDEX ... REBUILD PARTITION でローカル索引のパーティションを再構築できます。
- ハッシュ・パーティションまたはハッシュ・サブパーティションのローカル索引に指定できるパラメータは、TABLESPACE のみです。

PARTITION 句

PARTITION 句を使用すると、索引のパーティションが 1 つ再構築されます。この句は、索引パーティションを別の表領域に移動したり、作成時の物理属性を変更するために使用できます。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

パーティションの再構築の制限事項： この句は、コンポジット・パーティション表のローカル索引に対して指定できません。かわりに、REBUILD SUBPARTITION 句を使用してください。

参照： パーティションのメンテナンス操作については、『Oracle9i データベース管理者ガイド』を参照してください。9-81 ページの「[使用禁止の索引パーティションの再構築例](#)」も参照してください。

SUBPARTITION 句

SUBPARTITION 句を使用すると、索引のサブパーティションが 1 つ再構築されます。この句を使用して、索引パーティションを他の表領域に移動することもできます。TABLESPACE を指定しないと、サブパーティションは同じ表領域に再構築されます。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

索引のサブパーティション変更の制限事項：

- パラメータ TABLESPACE および *parallel_clause* 以外は、サブパーティションに対して指定できません。
- リスト・パーティションのサブパーティションは再構築できません。

REVERSE | NOREVERSE

索引ブロックのバイトを逆順に格納するかどうかを示します。

- REVERSE は、索引ブロックのバイトを逆順で格納します。また、索引を再構築する場合の ROWID を除外します。

- NOREVERSE は、索引を再構築する場合に逆順にせずに索引ブロックのバイトを格納します。NOREVERSE キーワードを指定せずに REVERSE 索引を再構築すると、再構築された索引は、逆キーの索引になります。

逆索引の制限事項：

- ビットマップ索引または索引構成表は逆順には格納できません。
- パーティションまたはサブパーティションに対して、REVERSE または NOREVERSE を指定できません。

参照： 9-80 ページの「[索引ブロックの逆順格納例：](#)」を参照してください。

parallel_clause

parallel_clause を使用すると、索引の再構築をパラレル化できます。

参照： 9-80 ページの「[索引のパラレル再構築例：](#)」を参照してください。

TABLESPACE 句

再構築された索引、索引パーティションまたは索引サブパーティションが格納される表領域を指定します。デフォルトは、索引またはパーティションを再構築する前に格納されていた表領域です。

COMPRESS | NOCOMPRESS

COMPRESS を指定すると、キー列値の繰返し項目を排除するキー圧縮が使用可能になります。integer を使用して、接頭辞の長さ（圧縮する接頭辞列数）を指定します。

- 一意の索引の場合、接頭辞の長さの有効範囲は、1 ～ キー列の数から 1 を引いた数までです。接頭辞の長さのデフォルト値は、キー列の数から 1 を引いた数です。
- 一意でない索引の場合、接頭辞の長さの有効範囲は、1 ～ キー列の数までです。接頭辞の長さのデフォルト値は、キー列の数です。

非パーティション索引（一意でない索引または 2 列以上の一意索引）のみを圧縮します。

NOCOMPRESS を指定すると、キー圧縮が使用禁止になります。これはデフォルト値です。

キー圧縮の制限事項： ビットマップ索引には、COMPRESS を指定できません。

ONLINE 句

ONLINE を指定すると、表またはパーティションの DML 操作を索引の再構築中に可能にするかどうかを指定します。

ONLINE の制限事項：

- オンラインで索引を作成中は、パラレル DML はサポートされません。ONLINE を指定し、パラレル DML 文を発行すると、Oracle はエラーを戻します。
- ビットマップ索引またはクラスタ索引には、ONLINE を指定できません。
- 索引構成表の一意索引の場合、索引構成表内の索引キー列の数と主キー列の数の合計は、32 以下にする必要があります。

COMPUTE STATISTICS 句

索引の再構築中に、比較的低コストで統計情報を収集する場合、COMPUTE STATISTICS を指定します。これらの統計情報は、SQL 文の実行計画を選択する際に、オブティマイザによって使用中のデータ・ディクショナリに格納されます。

収集された統計情報のタイプは、再構築している索引タイプによって異なります。

注意： 表のかわりに別の索引を使用して索引を作成する場合、元の索引は適切な統計情報を提供しない場合があります。このため、一般的に基となる表を使用して統計を計算します。その結果、統計は改善されますが、パフォーマンスが低下することがあります。

PL/SQL パッケージおよびプロシージャでは、その他の方法で統計を収集できます。

参照：『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』および 9-80 ページの「[索引統計の収集例](#)」を参照してください。

logging_clause

ALTER INDEX ... REBUILD 操作をログに記録するかどうかを指定します。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

PARAMETERS 句

PARAMETERS 句は、ドメイン索引のみに適用されます。この句は、未解析のまま適切な ODCI 索引タイプ・ルーチンに渡すパラメータ文字列を指定します。パラメータ文字列の最大長は 1,000 文字です。

索引全体を変更または再構築する場合、文字列は索引レベルのパラメータを参照する必要があります。索引のパーティションを再構築する場合、文字列はパーティション・レベルのパラメータを参照する必要があります。

索引に UNUSABLE のマークが付いている場合、パラメータが変更されるのみで USABLE になりません。UNUSABLE のマークが付いた索引を再構築し、使用可能にする必要があります。

注意： Oracle Text がインストール済の場合、Oracle Text 固有のパラメータを使用する Oracle Text のドメイン索引を再構築することができます。パラメータの詳細は、『Oracle Text リファレンス』を参照してください。

PARAMETERS 句の制限事項：

- この句は、ドメイン索引に対してのみ指定できます。
- 索引に IN_PROGRESS または FAILED のマークが付いておらず、IN_PROGRESS のマークが付いた索引パーティションが存在せず、変更対象のパーティションに FAILED のマークが付いていない場合にのみ、索引パーティションを変更できます。

参照：

- 索引タイプ・ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- ドメイン索引の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。

ENABLE 句

ENABLE は、索引が使用するユーザー定義ファンクションが削除または置換されたために使用禁止になったファンクション索引のみに適用されます。次の条件が該当する場合、この句によって、このような索引が使用可能になります。

- ファンクションが現在有効な場合
- 現行のファンクションの署名が、索引が作成された場合のファンクションの署名と一致する場合
- ファンクションに現在 DETERMINISTIC のマークが付いている場合

ファンクション索引を使用可能にする場合の制限事項： ENABLE と同じ文には、ALTER INDEX の他の句を指定できません。

DISABLE 句

DISABLE は、ファンクション索引のみに適用されます。この句を使用してファンクション索引を使用禁止にします。たとえば、ファンクションの本体を処理する場合に、これを行います。その後、ENABLE キーワードを使用して、索引を再構築、または別の ALTER INDEX 文を指定できます。

UNUSABLE 句

UNUSABLE を指定すると、索引、索引パーティションまたは索引サブパーティションに UNUSABLE のマークを付けます。使用禁止の索引を使用可能にする場合、再構築するか、または削除して再作成する必要があります。1 つのパーティションに UNUSABLE のマークが付いている場合も、同じ索引の他のパーティションは有効です。その索引を必要とする文が使用禁止のパーティションにアクセスしない場合、その文を実行できます。また、使用禁止のパーティションは、分割または名前を変更してから再構築できます。

索引への使用禁止のマーク付けの制限事項： この句は、一時表の索引に対して指定できません。

RENAME 句

この句を指定すると、索引の名前を変更できます。*new_index_name* は単一の識別子で、スキーマ名は含まれません。

索引の名前の変更の制限事項： ドメイン索引の場合、*index* および *index* のパーティションに IN_PROGRESS または FAILED のマークが付いていると無効になります。

参照： 9-81 ページの「[索引の名前の変更例](#)」を参照してください。

COALESCE 句

COALESCE を指定すると、再利用するために、索引ブロックの内容を空きブロックにマージできます（可能な場合）。

索引ブロックの結合の制限事項：

- この句は、一時表の索引に対して指定できません。
- この句は、索引構成表の主キー索引に対して指定できません。かわりに ALTER TABLE の COALESCE 句を使用します。

参照：

- 領域管理および索引の結合の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- 索引構成表の領域の結合の詳細は、11-104 ページの「[COALESCE](#)」を参照してください。

MONITORING USAGE | NOMONITORING USAGE

この句を使用すると、索引の使用を監視するかどうかを決定できます。

- MONITORING USAGE を指定すると、索引の監視が開始されます。まず、索引の使用に関する既存の情報が削除され、以降に ALTER INDEX ... NOMONITORING USAGE 文が実行されるまで、索引の使用が監視されます。

- 索引の監視を終了するには、NOMONITORING USAGE を指定します。

この ALTER INDEX ... NOMONITORING USAGE 文が発行された後、索引が使用されたかどうかを調べるには、V\$OBJECT_USAGE 動的パフォーマンス・ビューの USED 列を問い合わせます。

参照： データ・ディクショナリおよび動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

UPDATE BLOCK REFERENCES 句

UPDATE BLOCK REFERENCES 句は、索引構成表の通常のドメイン索引に対してのみ有効です。この句を指定すると、主キーによって識別されるブロックの適切なデータベース・アドレスとともに索引行の一部として格納され、失効したと推測されるすべてのデータ・ブロック・アドレスを更新することができます。

注意： ドメイン索引では、AlterIndexUpdBlockRefs に設定された alter_option パラメータで ODCIIndexAlter ルーチンが実行されます。このルーチンはカートリッジ・コードを使用可能にし、失効したと推測される索引のデータ・ブロック・アドレスを更新します。

UPDATE BLOCK REFERENCES の制限事項： この句を ALTER INDEX の他の句と組み合わせることはできません。

alter_index_partitioning

ALTER INDEX 文のパーティション化句は、パーティション索引に対してのみ有効です。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

索引パーティションの変更の制限事項：

- これらの句は、一時表の索引に対して指定できません。
- ベース索引に対するいくつかの操作を1つの ALTER INDEX 文にまとめることはできません（RENAME および REBUILD は除く）、パーティション操作を、他のパーティション操作またはベース索引に対する操作と組み合わせることはできません。

modify_index_default_attrs

パーティション索引のデフォルト属性に新しい値を指定します。

パーティションのデフォルト属性の変更の制限事項： ハッシュ・パーティション表またはコンポジット・パーティション表の索引の属性には、TABLESPACE のみ指定できます。

TABLESPACE 索引の新規パーティション、または索引パーティションのサブパーティションに対して、デフォルトの表領域を指定します。

logging_clause パーティション索引または索引パーティションのデフォルトのロギング属性を指定します。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

FOR PARTITION FOR PARTITION 句を使用すると、コンポジット・パーティション表にあるローカル索引のパーティションのサブパーティションに対して、デフォルトの属性を指定します。

FOR PARTITION の制限事項： リスト・パーティションに対して FOR PARTITION を指定することはできません。

参照： 9-82 ページの「[デフォルト属性の変更例:](#)」を参照してください。

modify_index_partition

modify_index_partition 句を使用すると、索引パーティション *partition* またはそのサブパーティションの実物理属性、ロギング属性または記憶特性を変更できます。

UPDATE BLOCK REFERENCES UPDATE BLOCK REFERENCES 句は、索引構成表の通常の索引に対してのみ有効です。この句を使用すると、2 次索引パーティションに格納されている、失効したと推測されるすべてのデータ・ブロック・アドレスを更新することができます。

UPDATE BLOCK REFERENCES の制限事項：

- ハッシュ・パーティション表の索引に対して、*physical_attributes_clause* を指定することはできません。
- ALTER INDEX の他の句とともに、UPDATE BLOCK REFERENCES を指定することはできません。

注意： 索引がコンポジット・パーティション表のローカル索引である場合、ここで指定した変更は、以前に索引のサブパーティションに対して指定したすべての属性を上書きします。また、このパーティションの将来のサブパーティションに対する属性のデフォルト値を設定します。サブパーティションの属性を上書きせずにパーティションのデフォルトの属性を変更する場合は、ALTER TABLE...MODIFY DEFAULT ATTRIBUTES OF PARTITION を使用します。

参照： 9-81 ページの「[索引への使用禁止のマーク付けの例：](#)」を参照してください。

rename_index_partition

rename_index_partition 句を使用すると、索引パーティションまたは索引サブパーティションの名前を *new_name* に変更できます。

索引パーティションの名前の変更の制限事項：

- リスト・パーティションのサブパーティションは名前を変更できません。
- ドメイン索引のパーティションの場合、*index* に IN_PROGRESS または FAILED のマークが付いていると無効になります。また、パーティションに IN_PROGRESS のマークが付くことはなく、名前を変更するパーティションに FAILED のマークが付いていると無効になります。

参照： 9-82 ページの「[索引パーティションの名前の変更例：](#)」を参照してください。

drop_index_partition

drop_index_partition 句を使用すると、グローバル・パーティション索引からパーティションとその中のデータを削除できます。グローバル索引のパーティションを削除する場合、その索引の次のパーティションに UNUSABLE のマークが付けられます。グローバル索引の最上位のパーティションは削除できません。

参照： 9-82 ページの「[索引パーティションの削除例：](#)」を参照してください。

split_index_partition

split_index_partition 句を使用すると、グローバル・パーティション索引のパーティションを 2 つのパーティションに分割し、新しいパーティションを索引に追加できます。

UNUSABLE のマークが付いたパーティションを分割すると、2つのパーティションが生成されますが、その両方に UNUSABLE のマークが付けられます。このようなパーティションは、使用前に再構築する必要があります。

使用可能なパーティションを分割した場合、索引データが入っている 2つのパーティションが生成されます。両方の新規パーティションは使用可能です。

AT 句 *split_partition_1* に新しい上限（境界は含まない）を指定します。
value_list の値は、*partition_name_old* の分割前のパーティション境界より小さく、その次の最小のパーティション（そのようなパーティションがある場合）のパーティション境界より大きい値である必要があります。

INTO 句 分割の結果、生成される 2つのパーティションの名前と物理属性を任意に指定します。

参照： 9-82 ページの「[パーティションの分割例:](#)」を参照してください。

modify_index_subpartition

modify_index_subpartition 句を使用すると、コンポジット・パーティション表にあるローカル索引のサブパーティションに対する領域の UNUSABLE のマーク付け、割当てまたは割当て解除が可能になります。このようなサブパーティションの他のすべての属性は、パーティションレベルのデフォルトの属性から継承されます。

例

索引ブロックの逆順格納例： 次の文は、索引ブロックのバイトが逆順に格納されるように、索引 *ord_customer_ix* (13-81 ページの「[索引の作成例:](#)」で作成) を再構築します。

```
ALTER INDEX ord_customer_ix REBUILD REVERSE;
```

索引統計の収集例： 次の文は、非パーティション索引 *ord_customer_ix* の統計情報を収集します。

```
ALTER INDEX ord_customer_ix REBUILD COMPUTE STATISTICS;
```

収集された統計情報のタイプは、再構築している索引タイプによって異なります。

参照： 『Oracle9i データベース概要』を参照してください。

索引の平行再構築例： 次の文は、平行実行プロセスを使用して、既存の索引から索引を再構築し、既存の索引のスキャンおよび新しい索引の構築を行います。

```
ALTER INDEX ord_customer_ix REBUILD PARALLEL;
```

実属性の変更例： 次の文は、同じ索引に将来追加されるデータ・ブロックが、5つの初期トランザクション・エントリと 100KB の増分エクステントを使用するように、索引 oe.cust_lname_ix を変更します。

```
/* Unless you change the default tablespace of sample user oe,
   or specify different tablespace storage for the index, this
   example fails because the default tablespace originally assigned
   to oe is locally managed.
*/
ALTER INDEX oe.cust_lname_ix
    INITTRANS 5
    STORAGE (NEXT 100K);
```

索引 oe.cust_lname_ix がパーティション化されている場合、この文は将来追加される索引のパーティションのデフォルト属性も変更します。将来追加される新しいパーティションでは、5つの初期トランザクション・エントリと 100KB の増分エクステントが使用されます。

パラレル問合せを使用可能にする例： 次の文は、索引 upper_ix（13-83 ページの「[ファクション索引の作成例](#)」で作成）に対するスキャンがパラレル化されないように、索引のパラレル属性を設定します。

```
ALTER INDEX upper_ix PARALLEL;
```

索引の名前の変更例： 次の文は、索引の名前を変更します。

```
ALTER INDEX upper_ix RENAME TO upper_name_ix;
```

索引への使用禁止のマーク付けの例： 次の例は、cost_ix 索引（13-84 ページの「[グローバル・パーティション索引の作成例](#)」で作成）を使用します。この索引のパーティション p1 は、9-82 ページの「[索引パーティションの削除例](#)」で削除されています。最初の文は、索引パーティション p2 に UNUSABLE のマークを付けます。

```
ALTER INDEX cost_ix
    MODIFY PARTITION p2 UNUSABLE;
```

次の文は、索引 cost_ix 全体に UNUSABLE のマークを付けます。

```
ALTER INDEX cost_ix UNUSABLE;
```

使用禁止の索引パーティションの再構築例： 次の文は、cost_ix 索引のパーティション p2 および p3 を再構築し、索引をもう一度使用可能にします。パーティション p3 の再構築は、ログに記録されません。

```
ALTER INDEX cost_ix
    REBUILD PARTITION p2;
ALTER INDEX cost_ix
```

```
REBUILD PARTITION p3 NOLOGGING;
```

MAXEXTENTS の変更例： 次の文は、パーティション p3 のエクステントの最大数を変更し、ロギング属性を変更します。

```
/* This example will fail if the tablespace in which partition p3
   resides is locally managed.
*/
ALTER INDEX cost_ix MODIFY PARTITION p3
   STORAGE (MAXEXTENTS 30) LOGGING;
```

索引パーティションの名前の変更例： 次の文は、cost_ix 索引（13-84 ページの「[グローバル・パーティション索引の作成例](#)」で作成）の索引パーティションの名前を変更します。

```
ALTER INDEX cost_ix
   RENAME PARTITION p3 TO p3_Q3;
```

パーティションの分割例： 次の文は、索引 cost_ix（13-84 ページの「[グローバル・パーティション索引の作成例](#)」で作成）のパーティション p2 をパーティション p2a および p2b に分割します。

```
ALTER INDEX cost_ix
   SPLIT PARTITION p2 AT (1500)
   INTO ( PARTITION p2a TABLESPACE tbs_01 LOGGING,
         PARTITION p2b TABLESPACE tbs_02);
```

索引パーティションの削除例： 次の文は、cost_ix 索引から索引パーティション p1 を削除します。

```
ALTER INDEX cost_ix
   DROP PARTITION p1;
```

デフォルト属性の変更例： 次の文は、ローカル・パーティション索引 prod_idx（13-84 ページの「[ハッシュ・パーティション表の索引の作成例](#)」で作成）のデフォルトの属性を変更します。将来追加される新しいパーティションでは、5 つの初期トランザクション・エントリと 100KB の増分エクステントが使用されます。

```
ALTER INDEX prod_idx
   MODIFY DEFAULT ATTRIBUTES INITRANS 5 STORAGE (NEXT 100K);
```


ALTER INDEXTYPE

用途

ALTER INDEXTYPE 文を使用すると、索引タイプの演算子を追加または削除したり、実装タイプを変更したり、索引タイプのプロパティを変更することができます。

前提条件

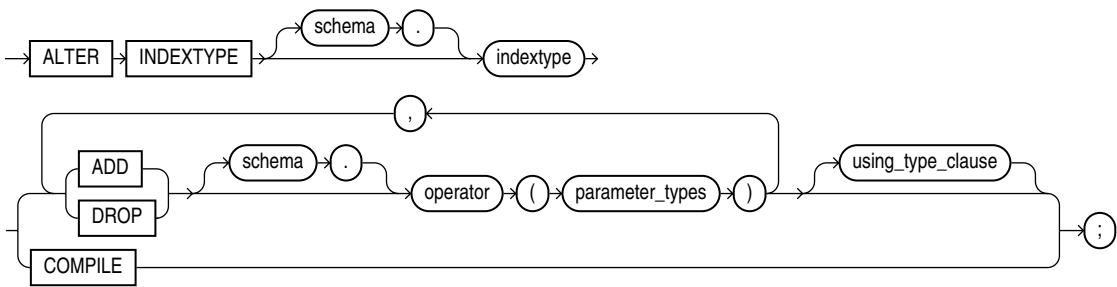
自分のスキーマまたはその他のスキーマの索引タイプを変更する場合は、ALTER ANY INDEXTYPE システム権限が必要です。

新しい演算子を追加する場合は、その演算子に対する EXECUTE オブジェクト権限が必要です。

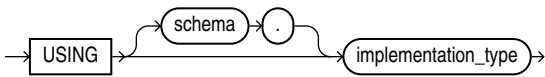
実装タイプを変更する場合は、新しい実装タイプに対する EXECUTE オブジェクト権限が必要です。

構文

alter_indextype::=



using_type_clause::=



セマンティクス

schema

索引タイプが存在するスキーマ名を指定します。*schema* を指定しない場合、索引タイプが自分のスキーマ内に定義されているものとみなされます。

indextype

変更する索引タイプの名前を指定します。

ADD | DROP

ADD または DROP 句を使用すると、演算子を追加または削除できます。

- *schema* には、演算子を含むスキーマを指定します。*schema* を指定しない場合、その演算子が自分のスキーマにあるとみなされます。
- *operator* には、索引タイプによってサポートされる演算子の名前を指定します。
この句に指定されるすべての演算子は有効な演算子である必要があります。
- *parameter_type* には、演算子へのパラメータ・タイプを指定します。

USING 句

USING 句を使用すると、索引タイプを実装する新しいタイプを指定できます。

COMPILE

この句を使用すると、索引タイプを明示的に再コンパイルできます。通常、索引タイプは自動的に再コンパイルされるため、この句は一部のアップグレード操作の後でのみ指定する必要があります。

例

索引タイプの変更例： 次の例では、CREATE INDEXTYPE 文で作成した TextIndexType 索引タイプに他の演算子バインディングを追加します。TextIndexType は、(CLOB, CLOB) バインディングを伴う新しい演算子 lob_contains をサポートします。

```
ALTER INDEXTYPE TextIndexType ADD lob_contains(CLOB, CLOB);
```

ALTER JAVA

用途

ALTER JAVA 文を使用すると、Java クラス・スキーマ・オブジェクトの変換、または Java ソース・スキーマ・オブジェクトのコンパイルを強制実行します（Java 名に対するすべての外部参照を他のクラスと対応付ける前に、Java クラスのメソッドをコールすることはできません）。

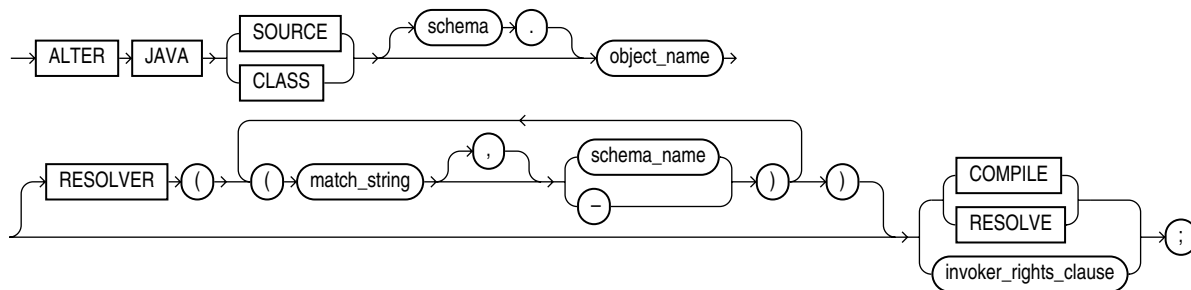
参照： Java クラスの変換および Java ソースのコンパイルの詳細は、『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。

前提条件

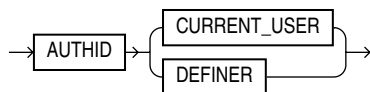
Java ソース、クラスまたはリソースが自分のスキーマ内にあるか、または ALTER ANY PROCEDURE システム権限が必要です。さらに、Java クラスに対する EXECUTE オブジェクト権限も必要です。

構文

alter_java::=



invoker_rights_clause::=



セマンティクス

JAVA SOURCE

ALTER JAVA SOURCE を使用すると、Java ソース・スキーマ・オブジェクトをコンパイルできます。

JAVA CLASS

ALTER JAVA CLASS を使用すると、Java ソース・スキーマ・オブジェクトを変換できます。

object_name

前回作成した Java クラスまたはソース・スキーマ・オブジェクトを指定します。小文字、または大文字と小文字を組み合わせた名前を付けるには、二重引用符を使用してください。

RESOLVER

RESOLVER 句によって、Java クラスまたはソースが作成されたときに指定したマッピング・ペアを使用して、完全に指定された参照用の Java 名に対するスキーマの検索方法を指定できます。

参照： 13-90 ページの「[CREATE JAVA](#)」および 9-87 ページの「[Java クラスの変換例](#)」を参照してください。

RESOLVE | COMPILE

RESOLVE および COMPILE は、同義のキーワードです。これらの句によって、プライマリ Java クラス・スキーマ・オブジェクトの変換を指定できます。

- クラスに適用された場合、他のクラス・スキーマ・オブジェクトに対する参照名に変換されます。
- ソースに適用された場合、ソースがコンパイルされます。

invoker_rights_clause

invoker_rights_clause は、メソッドを定義したユーザーのスキーマ内で、権限を使用してクラスのメソッドを実行するか、または、CURRENT_USER のスキーマ内で、権限を使用してクラスのメソッドを実行するかを指定します。

また、この句は、問合せ、DML 操作、およびその型のメンバー・ファンクションおよびプロシージャ内の動的 SQL 文の外部名の変換方法も定義します。

AUTHID CURRENT_USER クラスのメソッドが **CURRENT_USER** 権限で実行されることを指定する場合は、**CURRENT_USER** を意味します。この句はデフォルトで、実行者権限クラスを作成します。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を **CURRENT_USER** のスキーマで変換することも指定します。他のすべての文における外部名は、メソッドを含むスキーマで変換します。

AUTHID DEFINER メソッドを定義したユーザーの権限を使用して、クラスのメソッドを実行する場合は、**DEFINER** を指定します。

さらに、メソッドのあるスキーマ内で外部名を変換するかどうかを指定します。

参照：

- **CURRENT_USER** の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 『Oracle9i Java Stored Procedures Developer's Guide』も参照してください。

例

Java クラスの変換例： 次の文は、Java クラスを強制変換します。

```
ALTER JAVA CLASS "Agent"  
  RESOLVER (("/home/java/bin/" pm) (* public))  
  RESOLVE;
```

ALTER MATERIALIZED VIEW

用途

マテリアライズド・ビューは、問合せ結果を含むデータベース・オブジェクトです。問合せの FROM 句には、表、ビューおよびその他のマテリアライズド・ビューを指定できます。これらをあわせて、**マスター表**（レプリケーション用語）または**ディテール表**（データ・ウェアハウス用語）といいます。このマニュアルでは、「マスター表」を使用します。マスター表が格納されているデータベースを**マスター・データベース**といいます。

ALTER MATERIALIZED VIEW 文を使用すると、既存のマテリアライズド・ビューを次の方法で変更します。

- 記憶特性を変更します。
- リフレッシュ方法、モードまたは時間を変更します。
- 別のタイプのマテリアライズド・ビューになるように構造を変更します。
- クエリー・リライトを使用可能または使用禁止にします。

注意： キーワード SNAPSHOT は、MATERIALIZED VIEW のかわりに下位互換用にサポートされています。

参照：

- マテリアライズド・ビューの作成の詳細は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。
- レプリケーション環境でのマテリアライズド・ビューの詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウス環境でのマテリアライズド・ビューの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

前提条件

マテリアライズド・ビューを変更するために必要な権限は、次のように直接付与される必要があります。

マテリアライズド・ビューが自分のスキーマ内にあるか、または ALTER ANY MATERIALIZED VIEW システム権限が必要です。

クエリー・リライトでマテリアライズド・ビューを使用可能にする場合、次の条件が必要です。

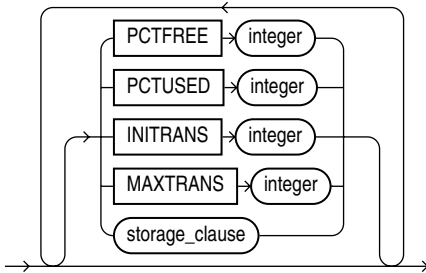
- マテリアライズド・ビュー内のすべてのマスター表が自分のスキーマ内にある場合、QUERY REWRITE 権限が必要です。
- いずれかのマスター表が別のスキーマ内にある場合、GLOBAL QUERY REWRITE 権限が必要です。
- マテリアライズド・ビューが別のユーザーのスキーマ内にある場合、ユーザーおよびそのスキーマ所有者の両方に、前述の適切な QUERY REWRITE 権限が必要です。また、マテリアライズド・ビューの所有者は、マテリアライズド・ビュー所有者が所有しないすべてのマスター表への SELECT 権限を持っている必要があります。

参照：『Oracle9i アドバンスド・レプリケーション』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

alter_materialized_view::=

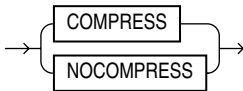
(9-91 ページの「[physical_attributes_clause::=](#)」、9-91 ページの「[data_segment_compression::=](#)」、9-91 ページの「[LOB_storage_clause::=](#)」、9-92 ページの「[modify_LOB_storage_clause::=](#)」、11-58 ページの「[alter_table_partitioning](#)」(ALTER TABLE 構文の一部)、9-93 ページの「[parallel_clause::=](#)」、9-93 ページの「[logging_clause::=](#)」、9-94 ページの「[allocate_extent_clause::=](#)」を参照)

[physical_attributes_clause::=](#)

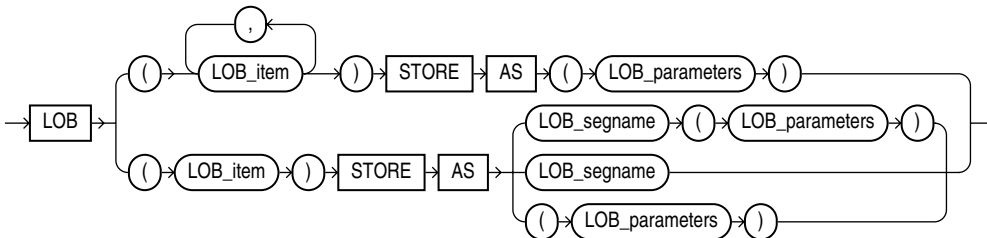


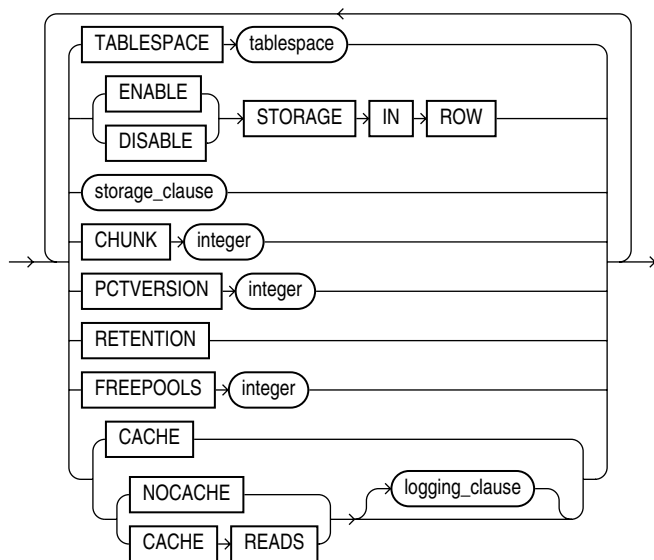
(7-53 ページの「[storage_clause::=](#)」を参照)

[data_segment_compression::=](#)

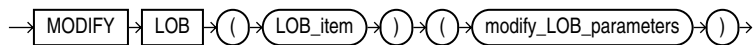


[LOB_storage_clause::=](#)

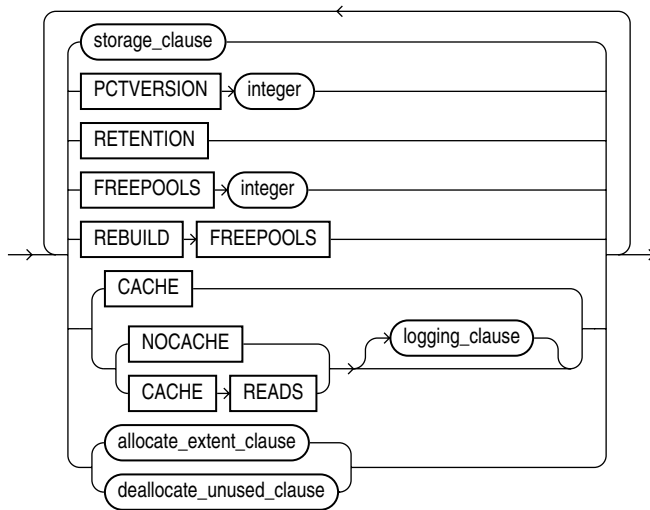


LOB_parameters::=

(7-53 ページの「[storage_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

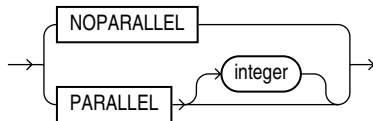
modify_LOB_storage_clause::=

modify_LOB_parameters::=

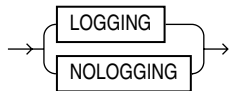


(7-53 ページの「[storage_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、
9-94 ページの「[allocate_extent_clause::=](#)」、9-94 ページの
「[deallocate_unused_clause::=](#)」を参照)

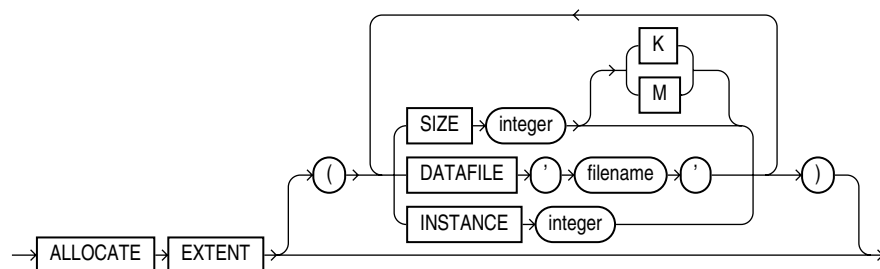
parallel_clause::=



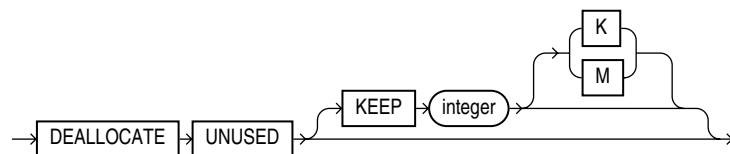
logging_clause::=



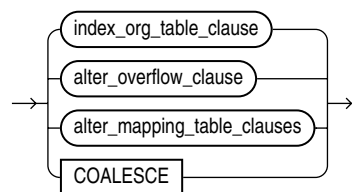
allocate_extent_clause::=



deallocate_unused_clause::=

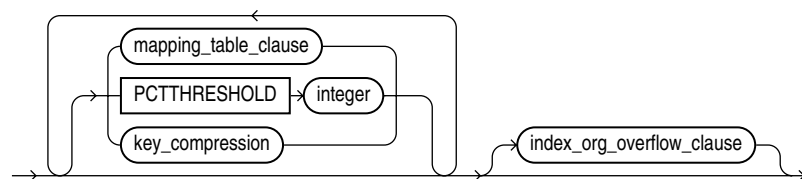


alter_iot_clauses::=



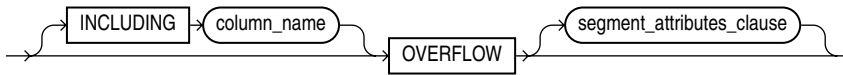
(9-94 ページの「[index_org_table_clause::=](#)」、
 9-95 ページの「[alter_overflow_clause::=](#)」を参照。
[alter_mapping_table_clauses](#): マテリアライズド・ビューではサポートされません。)

index_org_table_clause::=

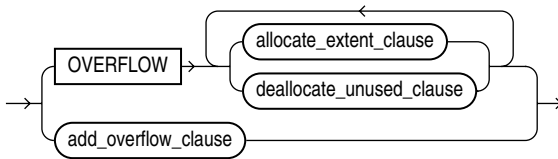


([mapping_table_clause](#): マテリアライズド・ビューではサポートされません。
[key_compression](#): マテリアライズド・ビューではサポートされません。
 9-95 ページの「[index_org_overflow_clause::=](#)」を参照)

index_org_overflow_clause::=

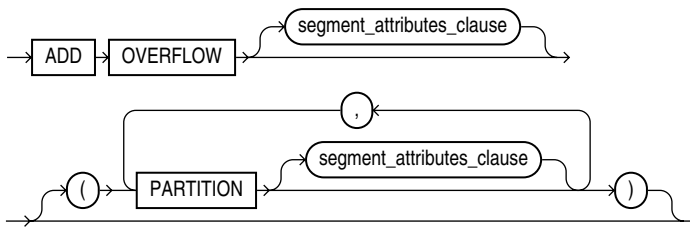


alter_overflow_clause::=

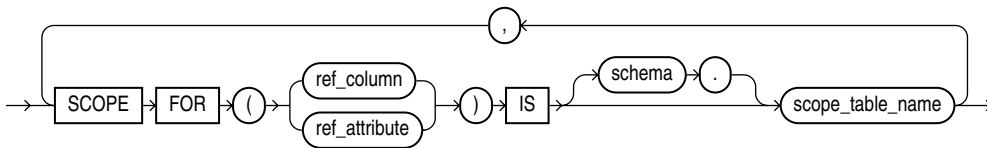


(9-94 ページの「[allocate_extent_clause::=](#)」、9-94 ページの「[deallocate_unused_clause::=](#)」を参照)

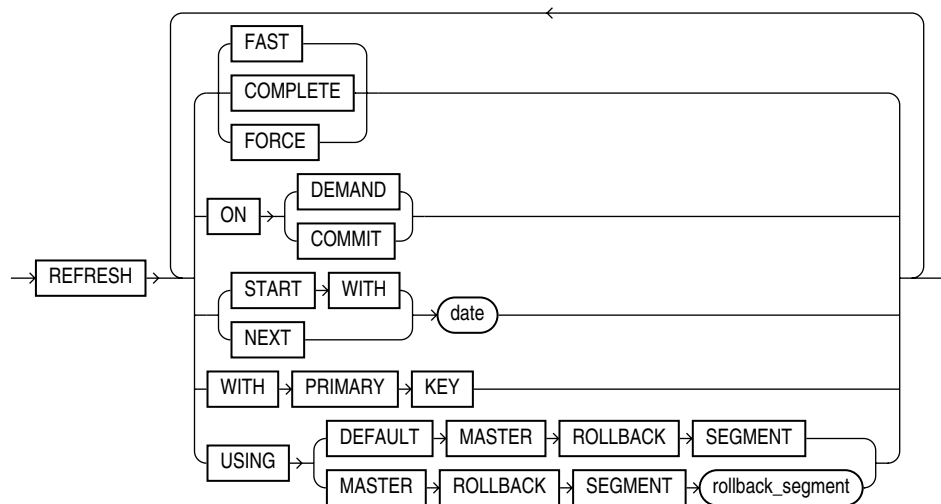
add_overflow_clause::=



scoped_table_ref_constraint::=



`alter_mv_refresh::=`



セマンティクス

schema

マテリアライズド・ビューが含まれているスキーマを指定します。*schema* を指定しない場合、マテリアライズド・ビューは自分のスキーマ内にあるとみなされます。

materialized_view

変更するマテリアライズド・ビューの名前を指定します。

physical_attributes_clause

PCTFREE、PCTUSED、INITRANS および MAXTRANS パラメータの値 (USING INDEX 句で使用する場合は、INITRANS および MAXTRANS パラメータ値のみ)、およびマテリアライズド・ビューの記憶特性を指定します。

参照：

- PCTFREE、PCTUSED、INITRANS および MAXTRANS パラメータの詳細は、11-2 ページの「[ALTER TABLE](#)」を参照してください。
- 記憶特性の詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

data_segment_compression

data_segment_compression 句を使用すると、ディスクおよびメモリの使用量を削減するために、データ・セグメントを圧縮するかどうかを指定できます。COMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用可能になります。NOCOMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用禁止になります。

参照： データ・セグメントの圧縮の詳細は、15-29 ページの「CREATE TABLE」の「[data_segment_compression](#)」を参照してください。

LOB_storage_clause

LOB_storage_clause を使用すると、新しい LOB の記憶特性を指定できます。マテリアライズド・ビューの LOB 記憶域は、表の場合と同様に動作します。

参照： LOB 記憶域パラメータの詳細は、15-36 ページの CREATE TABLE の「[LOB_storage_clause](#)」を参照してください。

modify_LOB_storage_clause

modify_LOB_storage_clause を使用すると、LOB 属性 *lob_item* の物理属性または LOB オブジェクト属性を変更できます。マテリアライズド・ビューの LOB 記憶域の変更は、表の場合と同様に動作します。

参照： 変更可能な LOB 記憶域パラメータの詳細は、11-55 ページの ALTER TABLE の「[modify_LOB_storage_clause](#)」を参照してください。

alter_table_partitioning

マテリアライズド・ビューの *partitioning_clauses* の構文および一般的な機能は、パーティション表と同じです。

参照： 11-58 ページの「[alter_table_partitioning](#)」を参照してください。

表のパーティション変更の制限事項：

- *partitioning_clauses* では、*LOB_storage_clause* または *modify_LOB_storage_clause* を指定できません。
- マテリアライズド・ビュー・パーティションを削除、切捨てまたは交換すると、Oracle はエラーを戻します。

注意： マテリアライズド・ビューの内容をマスター表の内容と同期させて保持するには、表パーティションを削除または切り捨てた後、表に依存しているすべてのマテリアライズド・ビューを手動で完全リフレッシュすることをお勧めします。

MODIFY PARTITION UNUSABLE LOCAL INDEXES この句を使用すると、*partition*に関連付けられたすべてのローカル索引パーティションに、UNUSABLE のマークを付けます。

MODIFY PARTITION REBUILD UNUSABLE LOCAL INDEXES この句を使用すると、*partition*に関連付けられた、使用禁止のローカル索引パーティションを再構築します。

parallel_clause

parallel_clause は、マテリアライズド・ビューのデフォルトの並列度を変更できます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である**並列度**を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

参照： 詳細は、15-51 ページの CREATE TABLE の「[parallel_clause の注意事項](#)」を参照してください。

logging_clause

マテリアライズド・ビューのロギング特性を指定または変更します。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

allocate_extent_clause

allocate_extent_clause は、マテリアライズド・ビューの新しいエクステントを明示的に割り当てます。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。

CACHE | NOCACHE

アクセス頻度の高いデータについて、CACHE は、全表スキャンの実行時にこの表に対して取り出された各ブロックを、バッファ・キャッシュの LRU リストの最高使用頻度側に入れることを指定します。この属性は、小規模な参照表で有効です。NOCACHE は、ブロックを LRU リストの最低使用頻度側に入れることを指定します。

参照： CACHE または NOCACHE の指定については、11-2 ページの「[ALTER TABLE](#)」を参照してください。

alter_iot_clauses

alter_iot_clauses を使用すると、索引構成マテリアライズド・ビューの特性を変更できます。*alter_iot_clauses* コンポーネントのキーワードおよびパラメータは、ALTER TABLE と同じです。また、次の制限事項があります。

索引構成マテリアライズド・ビューの変更の制限事項： *index_org_table_clause* の *mapping_table_clauses* または *key_compression* 句は指定できません。

参照： 索引構成マテリアライズド・ビューの作成については、14-16 ページの CREATE MATERIALIZED VIEW の「[index_org_table_clause](#)」を参照してください。

USING INDEX 句

マテリアライズド・ビューのデータをメンテナンスするために使用される索引の INITRANS、MAXTRANS および STORAGE の各パラメータ値を変更します。

USING INDEX 句の制限事項： この句では、PCTUSED または PCTFREE パラメータは指定できません。

MODIFY *scoped_table_ref_constraint*

MODIFY *scoped_table_ref_constraint* 句を使用すると、新しい表に REF 列または属性の有効範囲を再指定できます。

REF 列の有効範囲の再指定の制限事項： ALTER MATERIALIZED VIEW 文では、1 つの REF 列または属性の有効範囲のみを再指定することができます。この句は、この文以外では使用できません。

REBUILD 句

REBUILD を指定すると、*materialized_view* で参照されるタイプが導出された場合に、リフレッシュ操作が再生成されます。

REBUILD 句の制限事項： 同じ ALTER MATERIALIZED VIEW 文で他の句を指定できません。

alter_mv_refresh

alter_mv_refresh を使用すると、自動リフレッシュの方法、モードおよび回数のデフォルト値を変更できます。マテリアライズド・ビューのマスター表の内容が変更された場合、マテリアライズド・ビューのデータを更新し、現在マスター表にあるデータを正確に反映させる必要があります。この句によって、自動的にマテリアライズド・ビューをリフレッシュする回数をスケジューリングし、リフレッシュの方法およびモードを指定できます。

注意： この句では、デフォルトのリフレッシュ・オプションのみを設定します。リフレッシュを実際にも実装する手順は、『Oracle9i アドバンスド・レプリケーション』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

FAST 句

FAST を指定すると、増分リフレッシュ方法を指定できます。これはマスター表に対して行った変更に従ってリフレッシュを行います。この変更は、マスター表に関連付けられたマテリアライズド・ビュー・ログ（従来型 DML 変更の場合）またはダイレクト・ローダー・ログ（ダイレクト・パス・インサート操作の場合）に格納されます。

従来型 DML の変更の場合も、ダイレクト・パス・インサートの場合も、他の条件によって、高速リフレッシュへのマテリアライズド・ビューの適応性が制限されることがあります。

参照：

- レプリケーション環境における高速リフレッシュの制限については、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウス環境における高速リフレッシュの制限については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- 9-105 ページの「[自動リフレッシュの例:](#)」を参照してください。

FAST リフレッシュの制限事項：

- 作成時に FAST リフレッシュを指定した場合、作成するマテリアライズド・ビューは高速リフレッシュに適応することが検証されています。ALTER MATERIALIZED VIEW 文でリフレッシュ方法を FAST に変更した場合、これは検証されていません。マテリアライズド・ビューが高速リフレッシュに適応しない場合、このビューをリフレッシュするとエラーが戻されます。
- 定義する問合せに分析ファンクションが含まれている場合、マテリアライズド・ビューは高速リフレッシュに適応しません。

参照： 6-9 ページの「[分析ファンクション](#)」を参照してください。

COMPLETE 句

完全リフレッシュを行う場合に、COMPLETE を指定します。これは、マテリアライズド・ビューの定義する問合せを実行することによって実装されます。完全リフレッシュを要求すると、高速リフレッシュが実行可能であっても、完全リフレッシュが実行されます。

参照： 9-106 ページの「[完全リフレッシュの例 :](#)」を参照してください。

FORCE 句

リフレッシュ時、高速リフレッシュが可能な場合は高速リフレッシュを実行し、そうでない場合は完全リフレッシュを実行するときは、FORCE を指定します。

ON COMMIT 句

高速リフレッシュを実行して、マテリアライズド・ビューのマスター表に対するトランザクションをコミットするときは、必ず ON COMMIT を指定します。

ON COMMIT の制限事項： この句は、マテリアライズド結合ビューおよびマテリアライズド集計ビューにのみサポートされます。

参照： 『Oracle9i アドバンスド・レプリケーション』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

ON DEMAND 句

マテリアライズド・ビューを、3 つの DBMS_MVIEW リフレッシュ・プロシージャのいずれかのコールによる要求でリフレッシュする場合は、ON DEMAND を指定します。ON COMMIT および ON DEMAND のどちらも指定しなかった場合、ON DEMAND がデフォルトになります。

参照：

- これらのプロシージャの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- REFRESH ON DEMAND を指定することによって作成できるマテリアライズド・ビューのタイプについては、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

注意： ON COMMIT または ON DEMAND を指定した場合、START WITH または NEXT を同時に指定できません。

START WITH 句

START WITH *date* を指定すると、最初の自動リフレッシュ時間を表す日付を指定できます。

NEXT 句

NEXT を指定すると、自動リフレッシュの間隔を計算するための日付式を指定できます。

START WITH 値および NEXT 値は、将来の時刻に評価される値です。START WITH 値を省略した場合、Oracle はマテリアライズド・ビューの作成時に NEXT 式を評価することによって、最初の自動リフレッシュ時刻を判断します。START WITH 値を指定し、NEXT 値を指定しない場合、Oracle は 1 回のみマテリアライズド・ビューをリフレッシュします。START WITH 値および NEXT 値のどちらも指定しない場合、または `alter_mv_refresh` を指定しない場合、マテリアライズド・ビューは自動リフレッシュされません。

WITH PRIMARY KEY 句

WITH PRIMARY KEY を指定すると、ROWID マテリアライズド・ビューを主キー・マテリアライズド・ビューに変更できます。主キー・マテリアライズド・ビューを使用すると、高速リフレッシュを継続できるマテリアライズド・ビューの機能に影響せずに、マテリアライズド・ビュー・マスター表を再編成できます。

この句を指定するには、マスター表に、使用可能な主キー制約が定義され、この制約に基づき、主キー情報を記録するマテリアライズド・ビュー・ログが定義されている必要があります。

参照：

- 主キー・マテリアライズド・ビューの詳細は、『Oracle9i アドバンスド・レプリケーション』を、ROWID マテリアライズド・ビューを主キー・マテリアライズド・ビューに変更する場合の詳細は、『Oracle9i データベース移行ガイド』を参照してください。
- 9-107 ページの「[主キー・マテリアライズド・ビューの例:](#)」を参照してください。

USING ROLLBACK SEGMENT 句

USING ROLLBACK SEGMENT を指定すると、マテリアライズド・ビューのリフレッシュ中に使用するリモート・ロールバック・セグメントを変更できます。使用するロールバック・セグメント名を `rollback_segment` に指定します。

参照： DBMS_REFRESH パッケージを使用してローカル・マテリアライズド・ビューのロールバック・セグメントを変更する場合の詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。9-106 ページの「マテリアライズド・ビューのロールバック・セグメントの変更例:」も参照してください。

DEFAULT 使用するロールバック・セグメントを Oracle に選択させる場合、DEFAULT を指定します。DEFAULT を指定した場合、`rollback_segment` は指定できません。

MASTER ... rollback_segment リモート・マスターで個々のマテリアライズド・ビュー用に使用されるリモート・ロールバック・セグメントを指定します。（ローカル・マテリアライズド・ビューのロールバック・セグメントを変更する場合は、DBMS_REFRESH パッケージを使用します。詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。）

1 つのマスター・ロールバック・セグメントは、各マテリアライズド・ビューに格納され、マテリアライズド・ビューの作成およびリフレッシュ時にスキャンされます。複合マテリアライズド・ビューの場合、マスター・ロールバック・セグメントの指定は無視されます。

QUERY REWRITE 句

この句を使用して、マテリアライズド・ビューがクエリー・リライトで利用できるかどうかを判断します。

ENABLE 句

ENABLE を指定して、クエリー・リライトでマテリアライズド・ビューを使用可能にします。

参照： 9-106 ページの「クエリー・リライトを使用可能にする例:」を参照してください。

マテリアライズド・ビューを使用可能にする場合の制限事項：

- マテリアライズド・ビューが無効または使用禁止の場合、ENABLE モードでもクエリー・リライトに適応しません。
- マテリアライズド・ビューの全体または一部がビューから作成されている場合、クエリー・リライトを使用可能にできません。

- マテリアライズド・ビューのすべてのユーザー定義ファンクションが DETERMINISTIC である場合のみ、クエリー・リライトを使用可能にできます。

参照： 13-49 ページの「[CREATE FUNCTION](#)」を参照してください。

- 文内の式が反復可能な場合のみ、クエリー・リライトを使用可能にできます。たとえば、CURRENT_TIME または USER を含めることはできません。

参照： クエリー・リライトの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

DISABLE 句

マテリアライズド・ビューがクエリー・リライトで使えないようにする場合は、DISABLE を指定します（マテリアライズド・ビューが無効な場合、使用禁止であるかどうかにかかわらず、クエリー・リライトの使用には適応しません）。ただし、使用禁止にされたマテリアライズド・ビューをリフレッシュすることはできません。

COMPILE

COMPILE を指定すると、マテリアライズド・ビューを明示的に再検証します。マテリアライズド・ビューが依存するオブジェクトを削除または変更した場合、マテリアライズド・ビューはアクセス可能のままですが、クエリー・リライトに対しては無効です。再度、明示的にマテリアライズド・ビューの妥当性チェックを行い、クエリー・リライトの使用に適応させるには、この句を使用します。

マテリアライズド・ビューの再妥当性チェックに失敗すると、リフレッシュできなくなるか、またはクエリー・リライトに使用できなくなります。

参照： 9-107 ページの「[マテリアライズド・ビューのコンパイル例:](#)」を参照してください。

CONSIDER FRESH

この句を使用すると、マスター表が変更された後のマテリアライズド・ビューの失効状態を管理することができます。CONSIDER FRESH は、マテリアライズド・ビューがフレッシュであり、TRUSTED または STALE_TOLERATED モードでのクエリー・リライトに適応するとみなされるように指定します。Oracle は、マテリアライズド・ビューがフレッシュであるかどうかを保証できないため、ENFORCED モードでのクエリー・リライトはサポートしません。また、この句はマテリアライズド・ビューの失効状態を UNKNOWN に設定します。失効状態は、ALL_MVIEWS、DBA_MVIEWS および USER_MVIEWS の各データ・ディクショナリ・ビューの STALENESS 列に表示されます。

注意： いずれかのマスター表の内容が変更された場合、マテリアライズド・ビューは失効します。この句は、Oracle に対して、マテリアライズド・ビューがフレッシュで、変更されていないものと仮定するように指示します。そのため、リフレッシュが保留されているこれらの表に対する実際の更新内容は、マテリアライズド・ビューから削除されます。

参照： クエリー・リライトの詳細、およびマスター表へのパーティション・メンテナンス操作の影響については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。9-106 ページの「[CONSIDER FRESH の例:](#)」も参照してください。

例

自動リフレッシュの例： 次の文は、マテリアライズド・ビュー `sales_by_month_by_state` (14-26 ページの「[マテリアライズド集計ビューの作成例:](#)」で作成) のリフレッシュ方法のデフォルトを FAST に変更します。

```
ALTER MATERIALIZED VIEW sales_by_month_by_state
  REFRESH FAST;
```

これ以降のマテリアライズド・ビューの自動リフレッシュは、高速リフレッシュになります。これは、単純なマテリアライズド・ビューであり、そのマスター表には、マテリアライズド・ビューの作成前または最後のリフレッシュ前に作成されたマテリアライズド・ビュー・ログがあります。

REFRESH 句に START WITH または NEXT の値が指定されていないため、マテリアライズド・ビュー `sales_by_month_by_state` を作成したとき、または最後に変更したときに REFRESH 句で設定されたリフレッシュ間隔がそのまま使用されます。

次の文は、マテリアライズド・ビュー `sales_by_month_by_state` の新しい自動リフレッシュ間隔を格納します。

```
ALTER MATERIALIZED VIEW sales_by_month_by_state
  REFRESH NEXT SYSDATE+7;
```

REFRESH 句に START WITH の値が指定されていないため、マテリアライズド・ビュー `sales_by_month_by_state` が作成されたとき、または最後に変更されたときに、START WITH と NEXT の値によって設定された日時に次の自動リフレッシュが行われます。

このマテリアライズド・ビューは、次に自動リフレッシュが行われる際に、リフレッシュされます。次に自動リフレッシュが行われる日時は、NEXT に設定した式 `SYSDATE+7` が計算されて決まります。その後は、週に 1 回リフレッシュが自動的に行われます。REFRESH 句にリフレッシュ方法が明示的に指定されていないため、直前の CREATE MATERIALIZED VIEW 文または ALTER MATERIALIZED VIEW 文の REFRESH 句で指定されたリフレッシュ方法が引き続き使用されます。

CONSIDER FRESH の例： 次の文は、Oracle がマテリアライズド・ビュー `sales_by_month_by_state` をフレッシュであるとみなすように指定します。この文を使用すると、`sales_by_month_by_state` のマスター表に対してパーティション・メンテナンス操作を実行した後でも、`sales_by_month_by_state` に対して TRUSTED モードでのクエリー・リライトが可能です。

```
ALTER MATERIALIZED VIEW sales_by_month_by_state CONSIDER FRESH;
```

参照： この ALTER MATERIALIZED VIEW の例を必要とするパーティション・メンテナンスの例は、11-89 ページの「[表のパーティションの分割例](#)」を参照してください。

完全リフレッシュの例： 次の文は、マテリアライズド・ビュー `emp_data` (14-28 ページの「[マテリアライズド・ビューの定期的リフレッシュ例](#)」で作成) の新しいリフレッシュ方法、NEXT で示す新しいリフレッシュ日時および新しい自動リフレッシュ間隔を指定します。

```
ALTER MATERIALIZED VIEW emp_data
  REFRESH COMPLETE
  START WITH TRUNC(SYSDATE+1) + 9/24
  NEXT SYSDATE+7;
```

START WITH 句に指定した値によって、このマテリアライズド・ビューの次の自動リフレッシュは翌日の午前 9 時に発生するように設定されます。この日時にマテリアライズド・ビューの完全リフレッシュが実行され、NEXT に設定した式が計算されます。その後は、このマテリアライズド・ビューは毎週リフレッシュされます。

クエリー・リライトを使用可能にする例： 次の文は、マテリアライズド・ビュー `emp_data` のクエリー・リライトを使用可能にし、暗黙的に再妥当性チェックを行います。

```
ALTER MATERIALIZED VIEW emp_data
  ENABLE QUERY REWRITE;
```

マテリアライズド・ビューのロールバック・セグメントの変更例： 次の文は、マテリアライズド・ビューのリフレッシュ時に使用されるリモートのマスター・ロールバック・セグメントを `rbs_two` に変更します。

```
ALTER MATERIALIZED VIEW new_employees
  REFRESH USING MASTER ROLLBACK SEGMENT rbs_two;
```

次の文は、マテリアライズド・ビュー・リフレッシュ時に使用されるリモートのマスター・ロールバック・セグメントを Oracle によって選択されたものに変更します。

```
ALTER MATERIALIZED VIEW new_employees
  REFRESH USING DEFAULT MASTER ROLLBACK SEGMENT;
```


主キー・マテリアライズド・ビューの例： 次の文は、ROWID マテリアライズド・ビュー (14-27 ページの「[ROWID マテリアライズド・ビューの作成例 :](#)」で作成) を主キー・マテリアライズド・ビューに変更します。この例では、`order_data` に主キーを持つマテリアライズド・ビュー・ログを定義している必要があります。

```
ALTER MATERIALIZED VIEW order_data  
  REFRESH WITH PRIMARY KEY;
```

マテリアライズド・ビューのコンパイル例： 次の文は、マテリアライズド・ビュー `store_mv` を再検証します。

```
ALTER MATERIALIZED VIEW order_data COMPILE;
```

ALTER MATERIALIZED VIEW LOG

用途

ALTER MATERIALIZED VIEW LOG 文を使用すると、記憶特性、リフレッシュ・モードまたはリフレッシュ時刻、または既存のマテリアライズド・ビュー・ログのタイプを変更できます。**マテリアライズド・ビュー・ログ**とは、マテリアライズド・ビューのマスター表に関連付けられる表です。

注意： キーワード **SNAPSHOT** は、**MATERIALIZED VIEW** のかわりに下位互換用にサポートされています。

参照：

- マテリアライズド・ビュー・ログの変更については、14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」を参照してください。
- マテリアライズド・ビューのリフレッシュ方法など、マテリアライズド・ビューの詳細は、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- マテリアライズド・ビューの様々なタイプの詳細は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。

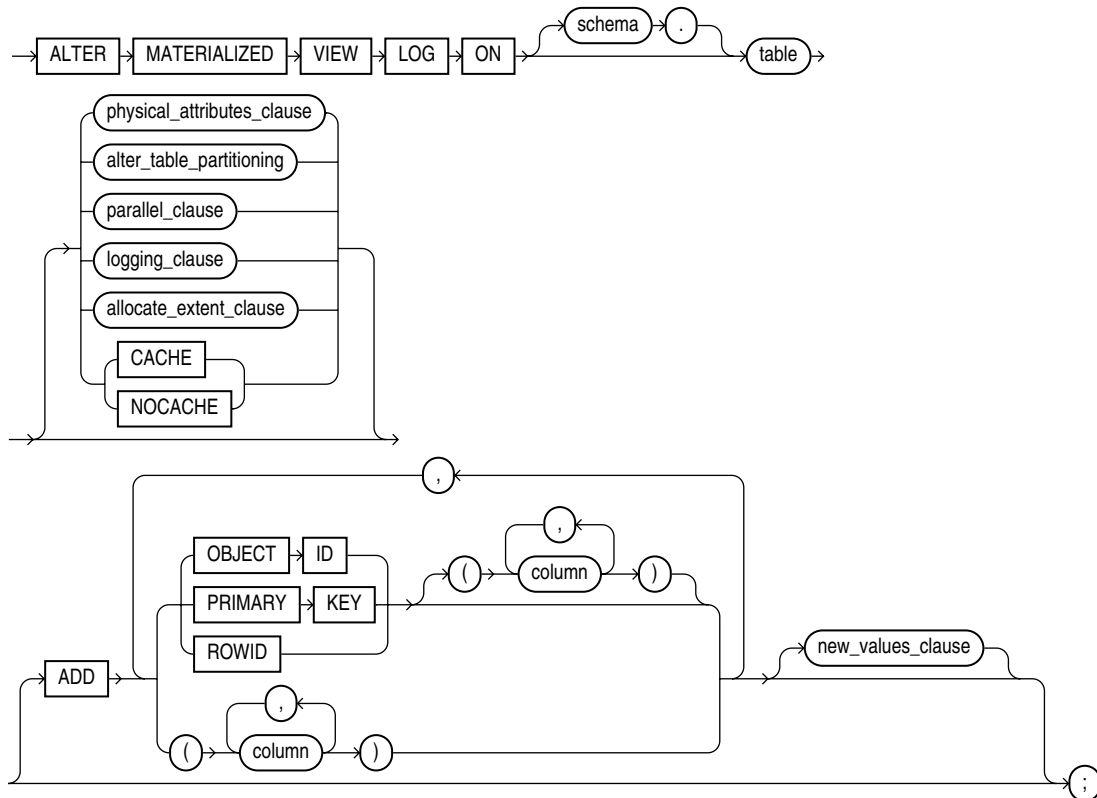
前提条件

マスター表の所有者、またはマスター表に対する **SELECT** 権限およびマテリアライズド・ビュー・ログに対する **ALTER** 権限を持つユーザーのみがマテリアライズド・ビュー・ログを変更できます。

参照： ALTER MATERIALIZED VIEW LOG の前提条件の詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。

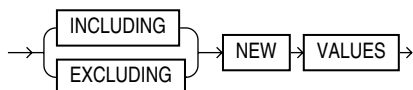
構文

alter_materialized_view_log::=

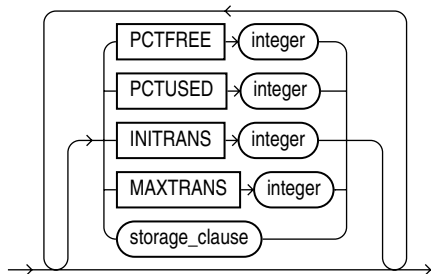


(9-110 ページの「[physical_attributes_clause::=](#)」、11-58 ページの「[alter_table_partitioning](#)」 (ALTER TABLE 構文の一部)、9-110 ページの「[parallel_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、9-110 ページの「[allocate_extent_clause::=](#)」、9-109 ページの「[new_values_clause::=](#)」を参照)

new_values_clause::=

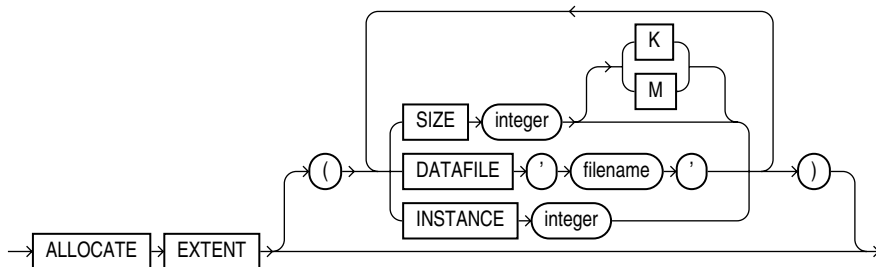


physical_attributes_clause::=

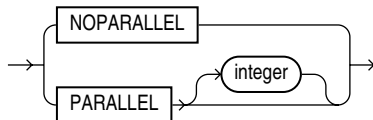


(7-53 ページの「*storage_clause::=*」を参照)

allocate_extent_clause::=



parallel_clause::=



セマンティクス

schema

マスター表が定義されているスキーマを指定します。*schema* を指定しないと、マテリアライズド・ビュー・ログは自分のスキーマ内にあるとみなされます。

table

変更するマテリアライズド・ビュー・ログに関連付けられたマスター表の名前を指定します。

physical_attributes_clause

physical_attributes_clause を使用すると、PCTFREE、PCTUSED、INITRANS および MAXTRANS の各パラメータの値、表、パーティションおよびオーバーフロー・データ・セグメントの記憶特性、またはパーティション表のデフォルト特性を変更できます。

マテリアライズド・ビューの物理属性の制限事項： マテリアライズド・ビュー・ログがローカル管理表領域に存在する場合は、*storage_clause* を使用して、エクステンツ・パラメータを変更することはできません。

参照： これらのパラメータについては、15-7 ページの「[CREATE TABLE](#)」を参照してください。

alter_table_partitioning

partitioning_clauses の構文および一般的な機能は、ALTER TABLE 文と同じです。

マテリアライズド・ビューのパーティション変更の制限事項：

- マテリアライズド・ビュー・ログのパーティションを変更する場合、*LOB_storage_clause* または *modify_LOB_storage_clause* は使用できません。
- マテリアライズド・ビュー・ログ・パーティションを削除、切捨てまたは交換すると、Oracle はエラーを戻します。

参照： 11-58 ページの「[alter_table_partitioning](#)」を参照してください。

parallel_clause

parallel_clause は、マテリアライズド・ビュー・ログへのパラレル操作がサポートされているかどうかを指定します。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL *integer* *integer* には、パラレル操作で使用するパラレル・スレッド数である**並列度**を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

参照： 詳細は、15-51 ページの CREATE TABLE の「[parallel_clause の注意事項:](#)」を参照してください。

logging_clause

マテリアライズド・ビュー・ログに対するロギング属性を指定します。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

allocate_extent_clause

allocate_extent_clause を使用すると、マテリアライズド・ビュー・ログの新しいエクステンツを明示的に割り当てることができます。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。

CACHE | NOCACHE 句

アクセス頻度が高いデータについて、CACHE は、全表スキャンの実行時にこのログ用に取り出された各ブロックを、バッファ・キャッシュ内の LRU リストの最高使用頻度側に入れることを指定します。この属性は、小規模な参照表で有効です。NOCACHE は、ブロックを LRU リストの最低使用頻度側に入れることを指定します。

参照： CACHE または NOCACHE の指定については、15-7 ページの「[CREATE TABLE](#)」を参照してください。

ADD 句

ADD 句を使用すると、マテリアライズド・ビュー・マスター表内の行が変更される際に、主キー値、ROWID 値またはオブジェクト ID 値も記録できるようにマテリアライズド・ビュー・ログを拡張できます。また、この句は、新しく列を記録するためにも使用できます。

これらの情報の記録を停止する場合は、スナップショット・ログを削除してから、再作成する必要があります。マテリアライズド・ビュー・ログを削除した後再作成した場合、マスター表に依存するすべての既存マテリアライズド・ビューが、次のリフレッシュ時に強制的に完全リフレッシュされます。

マテリアライズド・ビュー・ログの拡張の制限事項： 各マテリアライズド・ビュー・ログに指定できるのは、PRIMARY KEY、ROWID、OBJECT ID および列リストを1つずつです。そのため、これら3つの値のいずれかが作成時に（暗黙的または明示的に）指定された場合、この ALTER 文にはそれらの値を指定できません。

OBJECT ID 更新されるすべての行の適切なオブジェクト識別子をマテリアライズド・ビュー・ログに記録する場合は、OBJECT ID を指定します。

OBJECT ID 句の制限事項： OBJECT ID は、オブジェクト表のログ用のみに指定でき、記憶表用には指定できません。

PRIMARY KEY 更新されるすべての行の主キーをマテリアライズド・ビュー・ログに記録する場合は、PRIMARY KEY を指定します。

ROWID 更新されるすべての行の ROWID 値をマテリアライズド・ビュー・ログに記録する場合は、ROWID を指定します。

column 更新されるすべての行に対して、マテリアライズド・ビュー・ログに記録する値を持つ新しい列を指定します。通常、フィルタ列（マテリアライズド・ビューが参照する主キー以外の列）および結合列（副問合せの WHERE 句で結合を定義する主キー以外の列）を指定します。

参照：

- マテリアライズド・ビュー・ログの値の明示的および暗黙的な論理和については、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。
- フィルタ列および結合列については、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- 9-114 ページの「[ROWID マテリアライズド・ビューの例：](#)」を参照してください。

NEW VALUES 句

NEW VALUES 句は、マテリアライズド・ビュー・ログに古い値と新しい値の両方を保存するかを指定できます。ALTER MATERIALIZED VIEW LOG 文で追加した主キー、ROWID または列のみでなく、ログのすべての列にこの句で設定した値を適用します。

INCLUDING INCLUDING を指定すると、古い値と新しい値の両方を保存します。このログが単一表マテリアライズド集計ビューの表用で、マテリアライズド・ビューに高速リフレッシュを実行する場合、INCLUDING を指定してください。

EXCLUDING EXCLUDING を指定すると、ログに対する新しい値の記録を使用禁止にします。この句を使用すると、新しい値の記録によるオーバーヘッドを回避できます。

マテリアライズド・ビューのリフレッシュ・モードを FAST 以外のモードに変更した場合を除き、高速リフレッシュが可能な単一表マテリアライズド集計ビューが定義されている場合は、EXCLUDING NEW VALUES を使用しないでください。

参照： 9-114 ページの「マテリアライズド・ビュー・ログ EXCLUDING NEW VALUES の例：」を参照してください。

例

ROWID マテリアライズド・ビューの例： 次の文は、ROWID 情報も記録するように既存の主キー・マテリアライズド・ビュー・ログを変更します。

```
ALTER MATERIALIZED VIEW LOG ON order_items ADD ROWID;
```

マテリアライズド・ビュー・ログ EXCLUDING NEW VALUES の例： 次の文は、フィルタ列を追加し、新しい値を除外することで、hr.employees のマテリアライズド・ビュー・ログを変更します。このログを使用するマテリアライズド集計ビューは、これ以降高速リフレッシュされません。ただし、高速リフレッシュが不要になる場合は、この処理によって新しい値の記録によるオーバーヘッドを回避できます。

```
ALTER MATERIALIZED VIEW LOG ON employees
  ADD (commission_pct)
  EXCLUDING NEW VALUES;
```


ALTER OPERATOR

用途

ALTER OPERATOR 文を使用すると、既存の演算子をコンパイルできます。

参照： 14-40 ページの「[CREATE OPERATOR](#)」を参照してください。

前提条件

演算子が自分のスキーマ内または別のスキーマ内にあるか、あるいは ALTER ANY OPERATOR システム権限が必要です。

構文

alter_operator ::=



セマンティクス

schema

演算子が含まれているスキーマを指定します。この句を指定しない場合、その演算子が自分のスキーマにあるとみなされます。

operator

再コンパイルする演算子の名前を指定します。

COMPILE

COMPILE を指定すると、演算子が再コンパイルされます。COMPILE キーワードは必須です。

例

ユーザー定義演算子のコンパイル例： 次の例は、演算子 eq_op（14-43 ページの「[ユーザー定義演算子の作成例](#)」で作成）をコンパイルします。

```
ALTER OPERATOR eq_op COMPILE;
```

ALTER OUTLINE

用途

ストアド・アウトラインの名前を変更する場合、ストアド・アウトラインを異なるカテゴリに再度割り当てる場合、または、アウトラインの SQL 文をコンパイルし、古いアウトライン・データを現行の条件で作成したアウトラインと置き換えて、ストアド・アウトラインを再生成する場合に、ALTER OUTLINE 文を使用します。

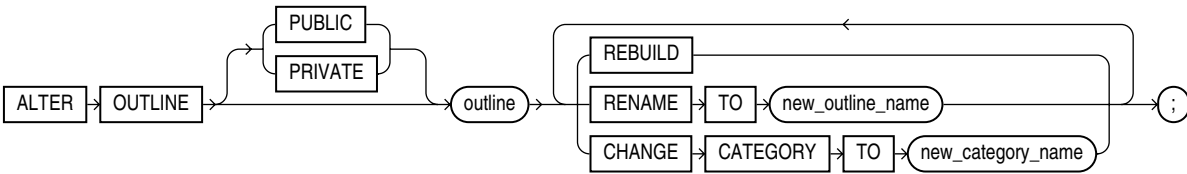
参照： アウトラインの詳細は、14-44 ページの「[CREATE OUTLINE](#)」および『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

前提条件

アウトラインを変更する場合は、ALTER ANY OUTLINE システム権限が必要です。

構文

alter_outline::=



セマンティクス

PUBLIC | PRIVATE

アウトラインのパブリック・バージョンを変更するには、PUBLIC を指定します。これはデフォルト値です。

現行のセッションに対してプライベートで、現行の解析スキーマにデータが格納されているアウトラインを変更する場合は、PRIVATE を指定します。

outline

変更するアウトラインの名前を指定します。

REBUILD

REBUILD を指定すると、現行の条件で、*outline* の実行計画が再生成されます。

参照： 9-117 ページの「[アウトラインの再構築例:](#)」を参照してください。

RENAME TO 句

RENAME TO 句を使用すると、*outline* の値と置き換えるアウトライン名を指定できます。

CHANGE CATEGORY TO 句

CHANGE CATEGORY TO 句を使用すると、*outline* の移動先となるカテゴリ名を指定できます。

例

アウトラインの再構築例： 次の文は、アウトラインのテキストをコンパイルし、古いアウトライン・データを現行の条件で作成したアウトラインと置き換えて、`salaries` というストアド・アウトラインを再生成します。

```
ALTER OUTLINE salaries REBUILD;
```

ALTER PACKAGE

用途

ALTER PACKAGE 文を使用すると、パッケージ仕様部またはパッケージ本体（あるいはその両方）を明示的に再コンパイルします。明示的に再コンパイルすることによって、実行時に暗黙的に再コンパイルする必要がなくなり、また、実行時のコンパイル・エラーとパフォーマンス上のオーバーヘッドもなくなります。

パッケージ中のすべてのオブジェクトは、1つの単位として格納されているため、ALTER PACKAGE 文によって、すべてのパッケージ・オブジェクトがまとめて再コンパイルされます。ALTER PROCEDURE 文または ALTER FUNCTION 文を使用して、パッケージ中の一部のプロシージャまたはファンクションを再コンパイルすることはできません。

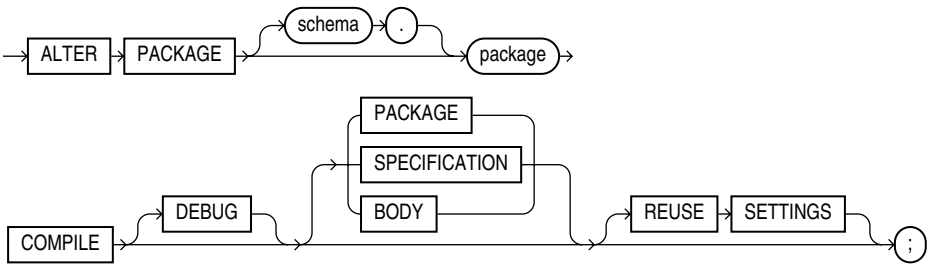
注意： この文を使用して、既存のパッケージの宣言や定義を変更することはできません。パッケージを再宣言または再定義する場合は、14-48 ページの「[CREATE PACKAGE](#)」または「[CREATE PACKAGE BODY](#)」に OR REPLACE 句を指定します。

前提条件

パッケージを変更するには、パッケージが自分のスキーマ内にあるか、または ALTER ANY PROCEDURE システム権限が必要です。

構文

alter_package::=



セマンティクス

schema

パッケージが含まれているスキーマを指定します。*schema* を指定しない場合、パッケージが自分のスキーマ内に定義されているとみなされます。

package

再コンパイルするパッケージの名前を指定します。

COMPILE

パッケージ仕様部およびパッケージ本体を再コンパイルする場合は、COMPILE を指定する必要があります。COMPILE キーワードは必須です。

再コンパイル中、Oracle はすべての永続コンパイラのスイッチ設定を削除し、セッションからそれらを再び取得してコンパイルの終了時に格納します。この手順を回避するには、REUSE SETTINGS 句を指定します。

パッケージの再コンパイル時にコンパイル・エラーが発生した場合は、エラーが戻り、パッケージ本体は無効のままです。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

参照： 9-120 ページの「[パッケージの再コンパイル例:](#)」を参照してください。

SPECIFICATION

SPECIFICATION を指定すると、無効かどうかにかかわらず、パッケージ仕様部のみを再コンパイルします。パッケージ仕様部を変更した場合は、コンパイル・エラーの有無を確認するために再コンパイルします。

その結果、そのパッケージ中のプロシージャまたはファンクションをコールするプロシージャなど、再コンパイルされたパッケージ仕様部に依存するローカル・オブジェクトはすべて無効になります。パッケージ本体もそのパッケージ仕様部に依存します。その後、明示的に再コンパイルせずに、これらの依存オブジェクトを参照した場合、Oracle は実行時にそれらを暗黙的に再コンパイルします。

BODY

BODY を指定すると、無効かどうかにかかわらず、パッケージ本体のみを再コンパイルします。パッケージ本体は、変更後に再コンパイルしてください。なお、パッケージ本体を再コンパイルしても、そのパッケージ仕様部に基づくオブジェクトは無効になりません。

パッケージ本体を再コンパイルした場合、そのパッケージ本体が依存するオブジェクトが無効の場合は、最初にそのオブジェクトが再コンパイルされます。パッケージ本体の再コンパイルが正常に終了した場合、この本体は有効になります。

PACKAGE

PACKAGE を指定すると、有効か無効かにかかわらず、パッケージ仕様部およびパッケージ本体（存在する場合）の両方を再コンパイルします。これはデフォルト値です。パッケージ仕様部およびパッケージ本体を再コンパイルした場合、前述の SPECIFICATION および BODY で説明するとおり、再妥当性チェックおよび再コンパイルが必要になります。

参照： リモート・オブジェクトを含むスキーマ・オブジェクト間の依存性を Oracle が管理する方法については、『Oracle9i データベース概要』を参照してください。

DEBUG

DEBUG を指定すると、PL/SQL コンパイラに対して、PL/SQL デバッグ用のコードを生成および格納するように指示できます。

参照： パッケージのデバッグの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

REUSE SETTINGS

REUSE SETTINGS を指定すると、Oracle によるコンパイラ・スイッチ設定の削除および再取得を回避できます。この句では既存の設定が持続され、その設定で再コンパイルされます。

DEBUG と REUSE SETTINGS の両方を指定する場合は、PLSQL_COMPILER_FLAGS パラメータの永続格納値が INTERPRETED, DEBUG に設定されます。その他のコンパイラ・スイッチ値は変更されません。

参照： PLSQL_COMPILER_FLAGS パラメータと COMPILE 句の相互作用の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

例

パッケージの再コンパイル例： 次の文は、hr.emp_mgmt パッケージの仕様部（14-51 ページの「[パッケージの作成例](#)」で作成）および本体を明示的に再コンパイルします。

```
ALTER PACKAGE emp_mgmt  
    COMPILE PACKAGE;
```

emp_mgmt の仕様部および本体の再コンパイル時に、コンパイル・エラーが発生しなかった場合、emp_mgmt は有効になります。その後、ユーザー hr は、実行時に再コンパイルしなくても、emp_mgmt の仕様部に宣言されたすべてのパッケージ・オブジェクトをコールまたは参照できます。emp_mgmt の再コンパイル時にコンパイル・エラーが発生した場合は、エラーが戻り、emp_mgmt は無効のままです。

また、emp_mgmt に依存しているすべてのオブジェクトが無効になります。その後、明示的に再コンパイルせずに、これらのオブジェクトを参照した場合、Oracle は、実行時にそれらを暗黙的に再コンパイルします。

次の文は、スキーマ hr 内の emp_mgmt パッケージの本体を再コンパイルします。

```
ALTER PACKAGE hr.emp_mgmt  
    COMPILE BODY;
```

パッケージ本体の再コンパイル時にコンパイル・エラーが発生しなければ、この本体は有効になります。その後、ユーザー hr は、実行時に再コンパイルしなくても、emp_mgmt の仕様部に宣言されたすべてのパッケージ・オブジェクトをコールまたは参照できます。本体の再コンパイル時にコンパイル・エラーが発生した場合は、エラー・メッセージが戻り、パッケージ本体は無効のままです。

この文は、emp_mgmt の仕様部ではなく本体を再コンパイルするため、依存するオブジェクトは無効にはなりません。

ALTER PROCEDURE

用途

ALTER PROCEDURE 文を使用すると、スタンドアロンのストアード・プロシージャを明示的に再コンパイルできます。明示的に再コンパイルすることによって、実行時に暗黙的に再コンパイルする必要がなくなり、また、実行時のコンパイル・エラーとパフォーマンス上のオーバーヘッドもなくなります。

パッケージの一部であるプロシージャを再コンパイルする場合、ALTER PACKAGE 文を使用して、そのパッケージ全体を再コンパイルします (9-118 ページの「ALTER PACKAGE」を参照)。

注意： この文を使用して、既存のプロシージャの宣言または定義を変更することはできません。プロシージャを再宣言または再定義する場合は、OR REPLACE 句を指定して CREATE PROCEDURE 文を使用します (14-60 ページの「CREATE PROCEDURE」を参照)。

ALTER PROCEDURE 文は、ALTER FUNCTION 文と似ています。

参照： 9-59 ページの「ALTER FUNCTION」を参照してください。

前提条件

プロシージャは、自分のスキーマ内にある必要があります。自分のスキーマ内にない場合は、ALTER ANY PROCEDURE システム権限が必要です。

構文

alter_procedure::=



セマンティクス

schema

プロシージャが含まれているスキーマを指定します。schema を指定しない場合、プロシージャが自分のスキーマ内に定義されているとみなされます。

procedure

再コンパイルするプロシージャの名前を指定します。

COMPILE

COMPILE を指定すると、プロシージャが再コンパイルされます。COMPILE キーワードは必須です。プロシージャが有効か無効かにかかわらず、プロシージャは再コンパイルされます。

- いずれかのオブジェクトが無効の場合、プロシージャが依存するオブジェクトが最初に再コンパイルされます。
- プロシージャに依存するすべてのローカル・オブジェクト（たとえば、再コンパイルしたプロシージャをコールするプロシージャ、再コンパイルしたプロシージャをコールするプロシージャを定義するパッケージ本体など）が無効になります。
- プロシージャの再コンパイルが正常に終了すると、このプロシージャは有効になります。プロシージャの再コンパイル時にエラーが発生した場合は、エラー・メッセージが戻り、プロシージャは無効のままです。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

再コンパイル中、Oracle はすべての永続コンパイラのスイッチ設定を削除し、セッションからそれらを再び取得してコンパイルの終了時に格納します。この手順を回避するには、REUSE SETTINGS 句を指定します。

参照： リモート・オブジェクトを含むスキーマ・オブジェクト間の依存状態管理については、『Oracle9i データベース概要』を参照してください。
9-124 ページの「[プロシージャの再コンパイル例:](#)」も参照してください。

DEBUG

DEBUG を指定すると、PL/SQL コンパイラに対して、PL/SQL デバッガ用のコードを生成および格納するように指示できます。

参照： これらのプロシージャについては、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

REUSE SETTINGS

REUSE SETTINGS を指定すると、Oracle によるコンパイラ・スイッチ設定の削除および再取得を回避できます。この句では既存の設定が持続され、その設定で再コンパイルされます。

DEBUG と REUSE SETTINGS の両方を指定する場合は、PLSQL_COMPILER_FLAGS パラメータの永続格納値が INTERPRETED, DEBUG に設定されます。その他のコンパイラ・スイッチ値は変更されません。

参照： PLSQL_COMPILER_FLAGS パラメータと COMPILE 句の相互作用の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

例

プロシージャの再コンパイル例： 次の文は、ユーザー hr が所有するプロシージャ remove_emp を明示的に再コンパイルします。

```
ALTER PROCEDURE hr.remove_emp  
    COMPILE;
```

credit の再コンパイル時にエラーが発生しなければ、credit は有効になります。その後、Oracle は、実行時にそれを再コンパイルしなくても実行できます。credit の再コンパイル時にコンパイル・エラーが発生した場合は、エラーが戻り、credit は無効のままです。

依存するオブジェクトもすべて無効になります。これらのオブジェクトとは、credit をコールするプロシージャ、ファンクション、パッケージ本体などです。その後、明示的に再コンパイルせずに、これらのオブジェクトを参照した場合、Oracle は、実行時にそれらを暗黙的に再コンパイルします。

ALTER PROFILE

用途

ALTER PROFILE 文を使用すると、プロファイルのリソース制限またはパスワード管理パラメータを追加、変更または削除できます。

ALTER PROFILE 文を使用すると、プロファイルに対して行った変更は、このプロファイルの現行のセッションのユーザーには影響しません。後続セッションのユーザーのみに影響します。

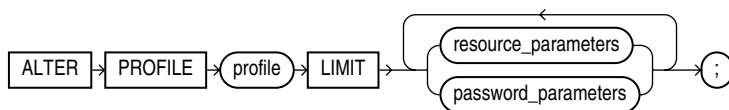
参照： プロファイルの作成の詳細は、14-66 ページの「[CREATE PROFILE](#)」を参照してください。

前提条件

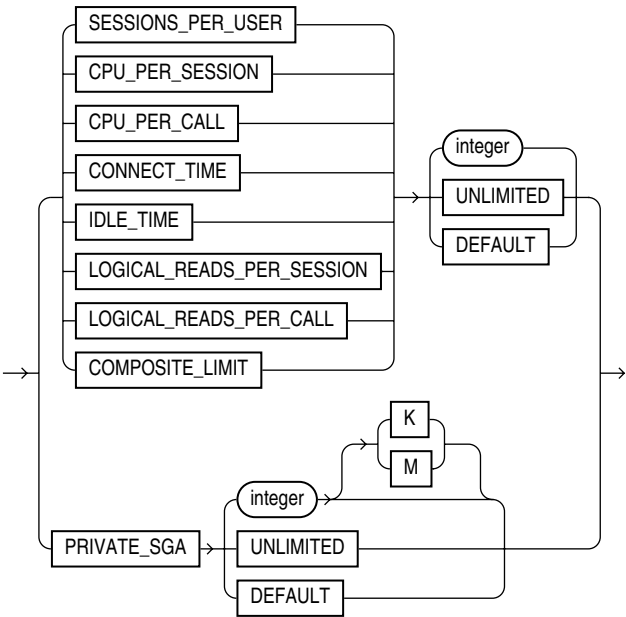
プロファイルのリソース制限を変更する場合は、ALTER PROFILE システム権限が必要です。パスワード制限および保護を変更する場合は、ALTER PROFILE および ALTER USER システム権限が必要です。

構文

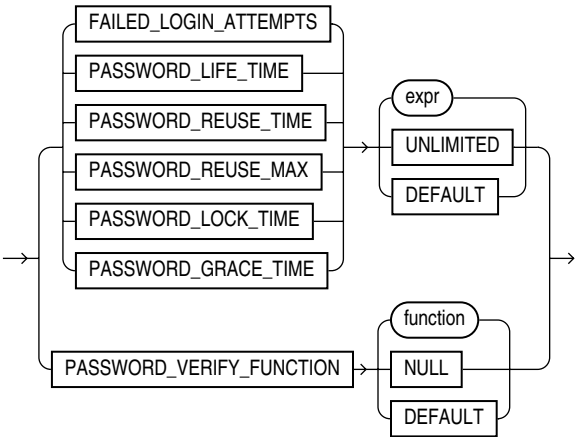
alter_profile::=



resource_parameters::=



password_parameters::=



セマンティクス

ALTER PROFILE 文のキーワード、パラメータおよび句の意味は、すべて CREATE PROFILE 文のキーワード、パラメータおよび句と同じです。

注意： DEFAULT プロファイルから制限を削除することはできません。

参照： 14-66 ページの「[CREATE PROFILE](#)」および次の項の例を参照してください。

例

パスワードを無効にする例： 次の文は、new_profile プロファイル (14-71 ページの「[プロファイルの作成例](#)」で作成) のパスワードを 90 日間再利用できないようにします。

```
ALTER PROFILE new_profile
  LIMIT PASSWORD_REUSE_TIME 90
  PASSWORD_REUSE_MAX UNLIMITED;
```

デフォルトのパスワード値の設定例： 次の文は、app_user プロファイル (14-72 ページの「[プロファイルのパスワード制限の設定例](#)」で作成) の PASSWORD_REUSE_TIME 値を DEFAULT プロファイルに定義された値にデフォルト設定します。

```
ALTER PROFILE app_user
  LIMIT PASSWORD_REUSE_TIME DEFAULT
  PASSWORD_REUSE_MAX UNLIMITED;
```

ログインおよびパスワード・ロック時刻の制限例： 次の文は、プロファイル app_user の FAILED_LOGIN_ATTEMPTS を 5 に PASSWORD_LOCK_TIME を 1 に変更します。

```
ALTER PROFILE app_user LIMIT
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LOCK_TIME 1;
```

この文を使用すると、ログインに 5 回失敗した場合に、app_user のアカウントは 1 日ロックされます。

パスワードの有効期限および猶予期間の変更例： 次の文は、プロファイル app_user2 の PASSWORD_LIFE_TIME を 90 日に、PASSWORD_GRACE_TIME を 5 日に変更します。

```
ALTER PROFILE app_user2 LIMIT
  PASSWORD_LIFE_TIME 90
  PASSWORD_GRACE_TIME 5;
```

同時セッションの制限例： 次の文は、プロファイル `app_user` に同時実行セッションの新しい制限 5 を指定します。

```
ALTER PROFILE app_user LIMIT SESSIONS_PER_USER 5;
```

現在、プロファイル `engineer` に `SESSIONS_PER_USER` の制限が定義されていない場合は、このプロファイルに制限 5 が追加されます。プロファイルに制限が定義されている場合は、前述の文によってその制限が 5 に再定義されます。プロファイル `engineer` が割り当てられているすべてのユーザーは、同時実行のセッションが 5 件に制限されます。

プロファイル制限の削除例： 次の文は、プロファイル `app_user` の `IDLE_TIME` 制限を削除します。

```
ALTER PROFILE app_user LIMIT IDLE_TIME DEFAULT;
```

プロファイル `app_user` が割り当てられているユーザーは、以降のセッションからプロファイル `DEFAULT` に定義された `IDLE_TIME` 制限に従います。

プロファイル・アイドル時間の制限例： 次の文は、プロファイル `DEFAULT` にアイドル時間の制限（2 分）を定義します。

```
ALTER PROFILE default LIMIT IDLE_TIME 2;
```

`IDLE_TIME` の制限は、次のユーザーに適用されます。

- プロファイルが明示的に割り当てられていないユーザー
- `IDLE_TIME` の制限が定義されていないプロファイルが明示的に割り当てられているユーザー

次の文は、プロファイル `app_user2` に無制限のアイドル時間を設定します。

```
ALTER PROFILE app_user2 LIMIT IDLE_TIME UNLIMITED;
```

プロファイル `app_user2` が割り当てられているすべてのユーザーは、以降のセッションから無制限のアイドル時間が許可されます。

ALTER RESOURCE COST

用途

ALTER RESOURCE COST 文を使用すると、セッションで使用するリソース・コストの合計を算出するための式を指定または変更できます。

Oracle は、その他のリソースの使用も監視しますが、セッションに対するリソース・コストの合計は、この構文の 4 種類のリソースに基づいて算出されます。

リソース・コストの合計を算出するための式を指定した場合、CREATE PROFILE 文の COMPOSITE_LIMIT パラメータを使用して、セッションに対するコストを制限できます。セッションのコストが制限を超えた場合、セッションは異常終了し、エラーが戻ります。各リソースに割り当てた重みを変更するために ALTER RESOURCE COST 文を使用した場合、現行のセッション以降のすべてのセッションで、その新しい重みを基にリソース・コストが計算されます。

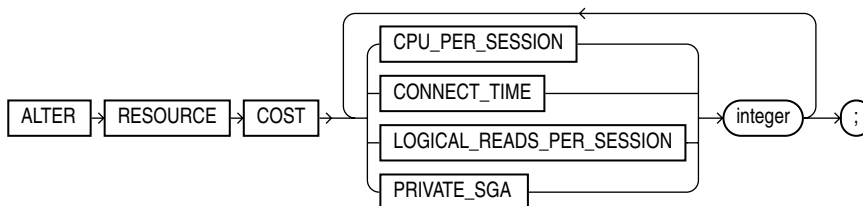
参照： リソースの合計およびリソース制限の設定については、14-66 ページの「[CREATE PROFILE](#)」を参照してください。

前提条件

ALTER RESOURCE COST システム権限が必要です。

構文

alter_resource_cost::=



セマンティクス

CPU_PER_SESSION

セッションによって使用された CPU 時間を、100 分の 1 秒単位で指定します。

CONNECT_TIME

セッションの経過時間を分単位で指定します。

LOGICAL_READS_PER_SESSION

メモリーおよびディスクから読み込まれるブロックなど、セッション中に読み込まれるデータ・ブロックの数を指定します。

PRIVATE_SGA

セッションごとに使用できるシステム・グローバル領域（SGA）内のプライベート領域のバイト数を指定します。共有サーバー・アーキテクチャを使用して、セッション用として SGA 内でプライベート領域を割り当てている場合のみ、この制限が適用されます。

integer

各リソースの重みを指定します。各リソースに割り当てる重みによって、各リソースがリソース・コストの合計に影響する程度が決定されます。リソースに重みを割り当てない場合は、デフォルト値の 0（ゼロ）が適用され、コストへの影響はありません。重みを割り当てた場合は、データベースの次のセッション以降のすべてのセッションで、その重みが適用されます。

最初にセッションで使用された各リソースの量にそのリソースの重みを乗算し、次に、4 種類のリソースの乗算結果を加算することによって、リソース・コストの合計が計算されます。どのセッションについても、このコストは、ユーザーのプロファイル内の COMPOSITE_LIMIT パラメータの値によって制限されます。乗算結果と総コストは、ともに **サービス単位**と呼ばれる単位で表されます。

例

リソース・コストの変更例： 次の文は、リソース CPU_PER_SESSION と CONNECT_TIME に重みを割り当てます。

```
ALTER RESOURCE COST
  CPU_PER_SESSION 100
  CONNECT_TIME      1;
```

この重みによって、セッションごとに次のコスト計算式が設定されます。

$$\text{cost} = (100 * \text{CPU\_PER\_SESSION}) + (1 * \text{CONNECT\_TIME})$$

CPU_PER_SESSION および CONNECT_TIME の値は、DEFAULT プロファイルまたはセッションのユーザーのプロファイルにある値のいずれかです。

ここでは、リソース LOGICAL_READS_PER_SESSION および PRIVATE_SGA に重みを割り当てていないため、これらのリソースは式に含まれません。

プロファイルで COMPOSITE_LIMIT 値として 500 を割り当てた場合、cost が 500 を超えると、必ず、セッションはこの制限を超えます。たとえば、CPU 時間 0.04 秒、経過時間 101 分を使用するセッションは、この制限を超えます。同様に、CPU 時間が 0.0301 秒、経過時間が 200 分のセッションもこの制限を超えます。

一度割り当てた重みは、次のように、別の ALTER RESOURCE 文を発行することによって変更できます。

```
ALTER RESOURCE COST
  LOGICAL_READS_PER_SESSION 2
  CONNECT_TIME 0;
```

新しく割り当てた重みによって、新しいコスト計算式が設定されます。

$$\text{cost} = (100 * \text{CPU\_PER\_SESSION}) + (2 * \text{LOGICAL\_READ\_PER\_SECOND})$$

CPU_PER_SESSION および LOGICAL_READS_PER_SECOND の値は、DEFAULT プロファイルまたはセッションのユーザーのプロファイルにある値のいずれかです。

この ALTER RESOURCE COST 文によって、式は次のように変更されます。

- CPU_PER_SESSION リソースの重みは指定しないが、このリソースにはすでに重みが割り当てられているため、式では先に指定した重みがそのまま使用されます。
- LOGICAL_READS_PER_SESSION リソースに重みを割り当てたため、このリソースが式で使用されます。
- CONNECT_TIME リソースに 0（ゼロ）を割り当てたため、このリソースは式に含まれていません。
- PRIVATE_SGA リソースの重みを指定せず、かつ、このリソースには重みが指定されていなかったため、このリソースは式で使用されません。

ALTER ROLE

用途

ALTER ROLE 文を使用すると、ロールを使用可能にするために必要な許可を変更できます。

参照：

- ロールの作成の詳細は、14-73 ページの「[CREATE ROLE](#)」を参照してください。
- セッションのロールを使用可能または使用禁止にする方法については、18-44 ページの「[SET ROLE](#)」を参照してください。

前提条件

ロールに Admin Option が付与されている必要があります。付与されていない場合は、ALTER ANY ROLE システム権限が必要です。

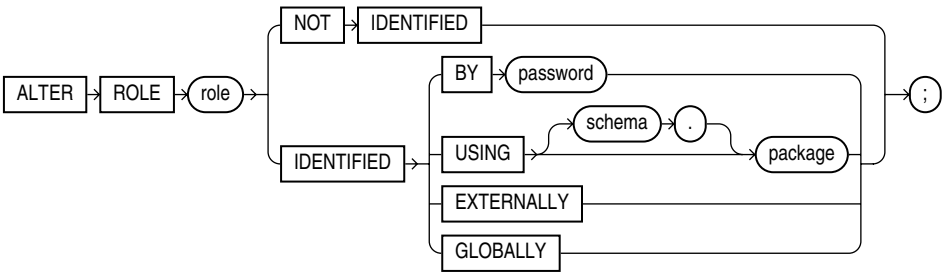
ロールを IDENTIFIED GLOBALLY に変更する前に、次の作業が必要です。

- ロールに対して外部的に識別されたロール権限をすべて取り消します。
- すべてのユーザー、ロールおよび PUBLIC からロールの付与を取り消します。

この規則の唯一の例外として、現在ロールを変更しているユーザーからはそのロールを取り消さないでください。

構文

alter_role::=



セマンティクス

ALTER ROLE 文のキーワード、パラメータおよび句の意味は、すべて CREATE ROLE 文のキーワード、パラメータおよび句と同じです。

注意：

- ロールを変更しても、ロールがすでに使用可能になっているユーザー・セッションには影響しません。
 - パスワードによって識別されるロールをアプリケーション・ロールに変更する場合 (USING *package* 句を使用)、ロールに対応付けられたパスワード情報は失われます。次にロールが使用可能になるときから、新しい認証方式が使用されます。
 - ALTER ANY ROLE システム権限を持つユーザーが、IDENTIFIED GLOBALLY ロールを IDENTIFIED BY *password*、IDENTIFIED EXTERNALLY または NOT IDENTIFIED に変更すると、非グローバルなロールを作成した場合と同様に、そのユーザーに変更されたロールが ADMIN OPTION 付きで付与されます。
-
-

参照： 14-73 ページの「[CREATE ROLE](#)」および次の例を参照してください。

例

ロール識別機能の変更例： 次の文は、ロール warehouse_user (14-75 ページの「[ロールの作成例：](#)」で作成) を NOT IDENTIFIED に変更します。

```
ALTER ROLE warehouse_user NOT IDENTIFIED;
```

ロール・パスワードの変更例： 次の文は、ロール dw_manager (14-75 ページの「[ロールの作成例：](#)」で作成) のパスワードを data に変更します。

```
ALTER ROLE dw_manager  
    IDENTIFIED BY data;
```

パスワードの変更後、ロール dw_manager が付与されているユーザーは、新しいパスワード「data」を入力してこのロールを使用可能にする必要があります。

アプリケーション・ロールの例： 次の例は、ロール dw_manager を hr.admin パッケージを使用してアプリケーション・ロールに変更します。

```
ALTER ROLE dw_manager IDENTIFIED USING hr.admin;
```

ALTER ROLLBACK SEGMENT

用途

ALTER ROLLBACK SEGMENT 文を使用すると、ロールバック・セグメントのオンライン / オフライン切替え、記憶特性の変更、またはロールバック・セグメントの最適サイズまたは指定サイズへの縮小を行います。

ここでは、データベースがロールバック UNDO モードで実行されている（初期化パラメータ UNDO_MANAGEMENT を MANUAL に設定、またはすべて設定しない）ことを前提としています。

データベースが自動 UNDO 管理モードで実行されている場合（初期化パラメータ UNDO_MANAGEMENT を AUTO に設定）、ユーザー作成ロールバック・セグメントを変更することはできません。この場合、CREATE ROLLBACK SEGMENT または ALTER ROLLBACK SEGMENT 文の応答でエラーが戻されます。このエラーを回避するには、UNDO_SUPPRESS_ERRORS パラメータを TRUE に設定します。

参照：

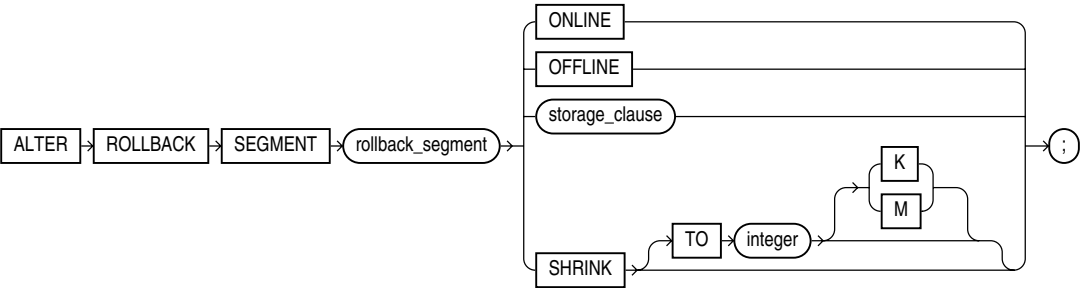
- ロールバック・セグメントの作成については、14-76 ページの「[CREATE ROLLBACK SEGMENT](#)」を参照してください。
- UNDO_MANAGEMENT パラメータおよび UNDO_SUPPRESS_ERRORS パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

前提条件

ALTER ROLLBACK SEGMENT システム権限が必要です。

構文

alter_rollback_segment::=



(7-51 ページの「[storage_clause](#)」を参照)

セマンティクス

rollback_segment

既存のロールバック・セグメントの名前を指定します。

ONLINE

ONLINE を指定すると、ロールバック・セグメントをオンラインにできます。ロールバック・セグメントを作成した場合、最初はオフライン状態になり、トランザクションに使用できなくなります。この句を指定した場合、ロールバック・セグメントはオンラインになり、インスタンスは、トランザクションに対してそのロールバック・セグメントを使用できるようになります。また、初期化パラメータ ROLLBACK_SEGMENTS を使用すると、インスタンスの起動時にロールバック・セグメントをオンラインにできます。

参照： 9-137 ページの「[ロールバック・セグメントをオンラインにする例](#)」を参照してください。

OFFLINE

OFFLINE を指定すると、ロールバック・セグメントをオフラインにできます。

- ロールバック・セグメント内に、アクティブ・トランザクションのロールバックに必要な情報が含まれていない場合は、すぐにオフラインになります。
- ロールバック・セグメントにアクティブ・トランザクションについての情報が含まれている場合、このロールバック・セグメントをその後のトランザクションに対して使用できないようにします。また、そのすべてのアクティブ・トランザクションがコミットまたはロールバックされた後、ロールバック・セグメントはオフラインになります。

一度オフラインにされたロールバック・セグメントは、どのインスタンスからもオンラインにできます。

ロールバック・セグメントがオンラインかオフラインかを調べるには、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS に問い合わせます。オンライン・ロールバック・セグメントの STATUS 値は IN_USE です。オフライン・ロールバック・セグメントの STATUS 値は AVAILABLE です。

ロールバック・セグメントをオフラインにする場合の制限事項： SYSTEM ロールバック・セグメントをオフラインにすることはできません。

参照： ロールバック・セグメントを使用可能および使用禁止にする方法については、『Oracle9i データベース管理者ガイド』を参照してください。

storage_clause

storage_clause を使用すると、ロールバック・セグメントの記憶特性を変更できます。

ロールバック・セグメントの記憶域の制限事項： 既存のロールバック・セグメントに対する INITIAL および MINEXTENTS の値は変更できません。

参照： 構文および追加情報については、7-51 ページの「[storage_clause](#)」を参照してください。9-137 ページの「[ロールバック・セグメントの記憶域の変更例](#)」も参照してください。

SHRINK 句

ロールバック・セグメントを最適サイズまたは指定サイズに縮小する場合に、SHRINK を指定します。縮小されるかどうか、および縮小量は、ロールバック・セグメントの使用可能領域およびアクティブ・トランザクションのロールバック・セグメント内での領域保持状態の状況によって異なります。

K または M で単位を KB または MB に指定しなかった場合、*integer* の値の単位はバイトになります。

TO *integer* に値を指定しなかった場合、ロールバック・セグメントを作成した CREATE ROLLBACK SEGMENT 文の *storage_clause* の OPTIMAL で指定した値が、デフォルトのサイズになります。OPTIMAL 値を指定しなかった場合、CREATE ROLLBACK SEGMENT 文の *storage_clause* の MINEXTENTS で指定した値がデフォルトのサイズになります。

TO *integer* に値を指定するかどうかにかかわらず、次のことがいえます。

- この文の実行時には、ロールバック・セグメントの縮小値が有効です。その後、サイズは CREATE ROLLBACK SEGMENT 文の OPTIMAL 値に戻ります。
- ロールバック・セグメントは、エクステント数 2 未満には縮小できません。

ロールバック・セグメントを縮小した後でロールバック・セグメントの実際のサイズを判断する場合は、DBA_SEGMENTS ビューの BYTES 列、BLOCKS 列および EXTENTS 列に問い合わせます。

ロールバック・セグメントの縮小の制限事項： Real Application Clusters 環境では、インスタンスに対してオンライン状態のロールバック・セグメントのみを縮小できます。

参照： 9-137 ページの「[ロールバック・セグメントのサイズの変更例](#)」を参照してください。

例

ロールバック・セグメントをオンラインにする例： 次の文は、ロールバック・セグメント rbs_one をオンラインにします。

```
ALTER ROLLBACK SEGMENT rbs_one ONLINE;
```

ロールバック・セグメントの記憶域の変更例： 次の文は、rbs_one の STORAGE パラメータを変更します。

```
ALTER ROLLBACK SEGMENT rbs_one  
    STORAGE (NEXT 1000 MAXEXTENTS 20);
```

ロールバック・セグメントのサイズの変更例： 次の文は、ロールバック・セグメントのサイズを 100MB に変更します。

```
ALTER ROLLBACK SEGMENT rbs_one  
    SHRINK TO 100 M;
```

ALTER SEQUENCE

用途

ALTER SEQUENCE 文を使用すると、既存の順序の増分値、最小値および最大値、キャッシュされる数および動作を変更できます。この文は、順序番号に影響します。

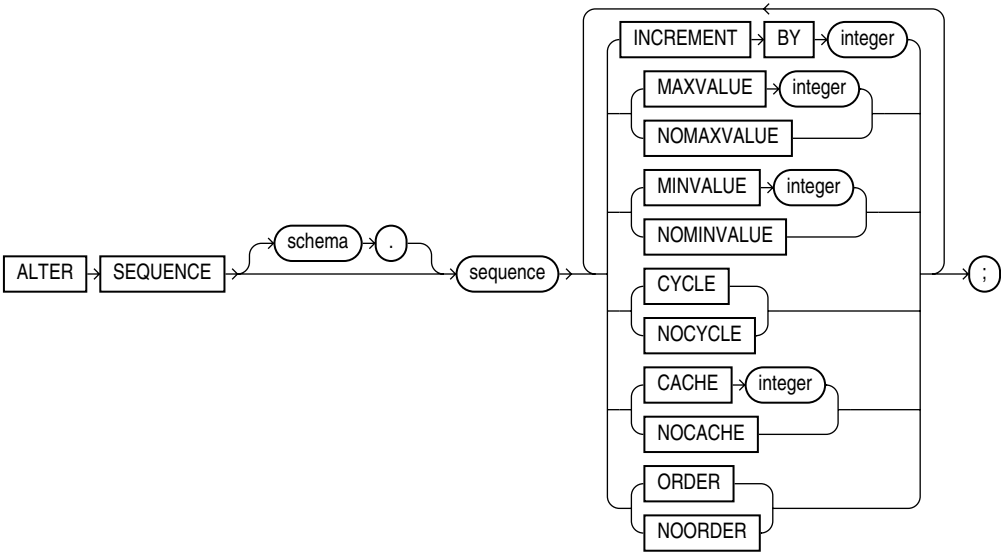
参照： 順序の詳細は、14-82 ページの「[CREATE SEQUENCE](#)」を参照してください。

前提条件

順序が自分のスキーマ内にある必要があります。自分のスキーマ内にはない場合は、順序に対する ALTER オブジェクト権限または ALTER ANY SEQUENCE システム権限が必要です。

構文

alter_sequence::=



セマンティクス

この文のキーワードおよびパラメータの意味は、順序を作成する場合と同じです。

- 異なる順序番号で再開する場合、順序を削除して再作成する必要があります。
- NEXTVAL を最初に呼び出す前に、INCREMENT BY の値を変更する場合、いくつかの順序番号がスキップされます。このため、元の START WITH の値を保持するには、順序を削除し、これを元の START WITH の値および新しい INCREMENT BY の値を使用して再作成する必要があります。
- いくつかの妥当性チェックが行われます。たとえば、MAXVALUE の値に現行の順序番号より小さい値は指定できません。

参照：

- 順序の作成の詳細については、14-82 ページの「[CREATE SEQUENCE](#)」を参照してください。
- 順序の削除および再作成の詳細は、17-2 ページの「[DROP SEQUENCE](#)」を参照してください。

例

順序の変更例： 次の文は、customers_seq 順序に新しい最大値を設定します。

```
ALTER SEQUENCE customers_seq  
    MAXVALUE 1500;
```

次の文は、customers_seq 順序に CYCLE および CACHE オプションを指定します。

```
ALTER SEQUENCE customers_seq  
    CYCLE  
    CACHE 5;
```


10

SQL 文 : ALTER SESSION ~ ALTER SYSTEM

この章では、次の SQL 文について説明します。

- ALTER SESSION
- ALTER SYSTEM

ALTER SESSION

用途

ALTER SESSION 文を使用すると、データベースへの接続に影響するすべての条件またはパラメータを、指定または変更できます。この文は、データベースとの接続を切断するまで有効です。

前提条件

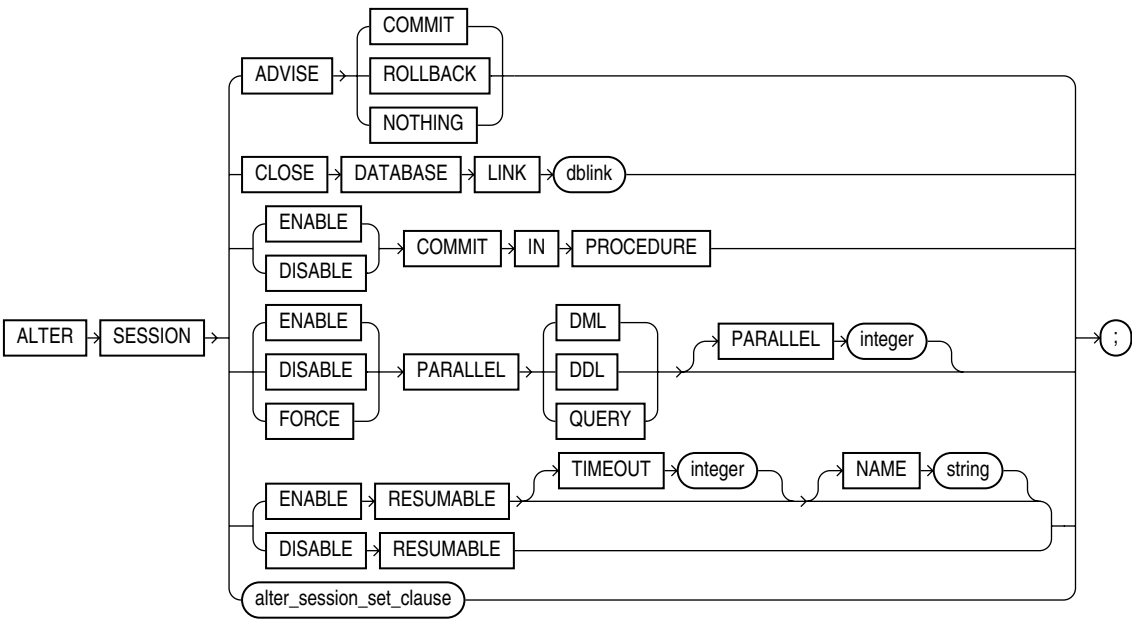
SQL トレース機能を使用可能または使用禁止にするには、ALTER SESSION システム権限が必要です。

再開可能な領域割当てを使用可能または使用禁止にするには、RESUMABLE システム権限が必要です。

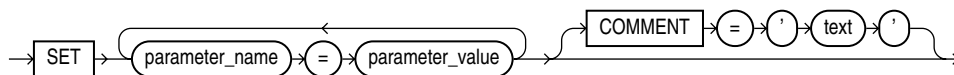
特に指定がないかぎり、これ以外の操作についての権限は不要です。

構文

alter_session::=



`alter_session_set_clause::=`



セマンティクス

ADVISE 句

ADVISE 句は、分散トランザクションを強制処理するためのアドバイスをリモート・データベースに送ります。リモート・データベース内の `DBA_2PC_PENDING` ビューの `ADVISE` 列に、アドバイスが表示されます（値 'C' が `COMMIT`、'R' が `ROLLBACK`、' ' が `NOTHING` を示します）。トランザクションの状態がインダウトになった場合、データベース管理者は、このアドバイスを使用してトランザクションをコミットするか、ロールバックするかを決定できます。

単一トランザクションにおいて、ADVISE 句を指定した `ALTER SESSION` 文を複数発行し、リモート・データベースごとに異なるアドバイスを送ることができます。ADVISE 句を指定した文はそれぞれ、ADVISE 句を指定した別の文が発行されるまで、トランザクション内の後続する文で参照されるデータベースに対してアドバイスを送ります。

参照： 10-16 ページの「[分散トランザクションを強制的に実行する例:](#)」を参照してください。

CLOSE DATABASE LINK 句

`CLOSE DATABASE LINK` を指定すると、データベース・リンク `dblink` がクローズされます。データベース・リンクを使用する `SQL` 文を発行した場合、Oracle は、このデータベース・リンクを使用してリモート・データベース上にセッションを作成します。この接続は、セッションの終了またはデータベース・リンクの数が初期化パラメータ `OPEN_LINKS` の値を超えるまでオープンされています。リンクをオープンしたままにしておくことによって発生するネットワークのオーバーヘッドを減らすには、セッションでデータベース・リンクを再度使用しない場合に、この句を使用してデータベース・リンクを明示的にクローズします。

参照： 10-17 ページの「[データベース・リンクをクローズする例:](#)」を参照してください。

ENABLE | DISABLE COMMIT IN PROCEDURE

プロシージャおよびストアド・ファンクションは PL/SQL で記述されるため、COMMIT 文と ROLLBACK 文を発行できます。アプリケーション自体が直接発行していない COMMIT 文や ROLLBACK 文によって、アプリケーションが中断される場合、DISABLE COMMIT IN PROCEDURE を指定して、セッション中にコールされるプロシージャおよびストアド・ファンクションがこれらの文を発行しないように制御します。

その後、ENABLE | DISABLE COMMIT IN PROCEDURE を指定することによって、セッションでプロシージャおよびストアド・ファンクションが COMMIT および ROLLBACK 文を発行できるようになります。

一部のアプリケーション (SQL*Forms など) は、自動的にプロシージャおよびストアド・ファンクションでの COMMIT 文や ROLLBACK 文を禁止します。詳細は、ご使用のアプリケーションのドキュメントを参照してください。

PARALLEL DML | DDL | QUERY

PARALLEL パラメータは、そのセッションの後続の DML、DDL または問合せ文をパラレル実行とみなすかどうかを指定します。この句は、現行のセッション中に表自体を変更せずに、表の並列度を上書き可能にします。コミットされていないトランザクションは、DML に対してこの句を実行する前に、コミットまたはロールバックされる必要があります。

参照： 10-16 ページの「[パラレル DML を使用可能にする例](#)」を参照してください。

ENABLE 句

ENABLE を指定すると、セッション内の後続文をパラレルで実行します。これは、DDL 文および問合せ文のデフォルトです。

- DML: パラレル・ヒントまたはパラレル句が指定されている場合に、セッションの DML 文をパラレル・モードで実行します。
- DDL: パラレル句が指定されている場合に、セッションの DDL 文をパラレル・モードで実行します。
- QUERY: パラレル・ヒントまたはパラレル句が指定されている場合に、セッションの問合せをパラレル・モードで実行します。

ENABLE 句の制限事項： オプションの PARALLEL *integer* に ENABLE を指定することはできません。

DISABLE 句

DISABLE を指定すると、後続文をシリアルに実行します。これは、DML 文のデフォルトです。

- DML: セッションの DML 文をシリアルに実行します。

- DDL: セッションの DDL 文をシリアルに実行します。
- QUERY: セッションの問合せをシリアルに実行します。

DISABLE 句の制限事項: オプションの `PARALLEL integer` に `DISABLE` を指定することはできません。

FORCE 句

FORCE は、セッションの後続文を強制的にパラレル実行します。パラレル句もパラレル・ヒントも指定されていない場合は、デフォルトの並列度が使用されます。この句は、セッションの後続文に指定されたすべての `parallel_clause` を上書きしますが、パラレル・ヒントにより上書きされます。

- DML: パラレル DML 制限のどれにも違反していない場合、特定の並列度がこの句に指定されていないかぎり、セッションの後続の DML 文は、デフォルトの並列度で実行されます。
- DDL: 特定の並列度がこの句に指定されていないかぎり、セッションの後続の DDL 文は、デフォルトの並列度で実行されます。結果のデータベース・オブジェクトは、通常の並列度に対応します。

FORCE DDL を使用した場合、そのセッションで作成されるすべての表は、自動的にデフォルトのパラレル化レベルで作成されます。結果は、CREATE TABLE 文で（デフォルトの並列度を使用して）`parallel_clause` を指定した場合と同じです。

- QUERY: 特定の並列度がこの句に指定されていないかぎり、後続の問合せはデフォルトの並列度で実行されます。

PARALLEL integer 並列度を明示的に指定する整数を指定します。

- FORCE DDL では、並列度は後続の DDL 文のパラレル句を上書きします。
- FORCE DML および FORCE QUERY では、並列度は、データ・ディクショナリの表に格納されている現行の並列度を上書きします。
- ヒントによって文に指定される並列度は、強制的に実行される並列度を上書きします。

次の DML 操作は、この句に関係なくパラレル化されません。

- クラスタ化表に対する操作
- データベースまたはパッケージの状態を読み書きする埋込みファンクションを使用した操作
- 起動する可能性のあるトリガーを使用した表に対する操作
- オブジェクト型、LONG または LOB データ型が含まれている表またはスキーマ・オブジェクトでの操作

RESUMABLE 句

これらの句を使用すると、再開可能な領域割当てを使用可能および使用禁止にできます。この機能によって、領域不足のエラー条件が発生した場合に操作は停止され、エラー条件が修復されたときに中断したところから自動的に再開されます。

注意： 再開可能な領域割当ては、ローカル管理表領域での操作で、完全にサポートされます。ディクショナリ管理表領域を使用する場合は、制限事項があります。制限事項の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

ENABLE RESUMABLE

この句を使用すると、セッションに対する再開可能な領域割当てが使用可能になります。

TIMEOUT TIMEOUT では、エラー条件が修復されるまで操作を停止する時間を秒単位で指定できます。エラー条件が TIMEOUT で指定した時間までに修復されない場合は、停止操作は異常終了します。

NAME NAME では、ユーザー定義のテキスト文字列を指定することができ、再開可能モードのセッション中に発行される文の識別に有効です。USER_RESUMABLE データ・ディクショナリ・ビューおよび DBA_RESUMABLE データ・ディクショナリ・ビューに、テキスト文字列が挿入されます。NAME を指定しない場合は、デフォルト文字列「User username(userid), Session sessionid, Instance instanceid」が挿入されます。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

DISABLE RESUMABLE

この句を使用すると、セッションに対する再開可能な領域割当てが使用禁止になります。

alter_session_set_clause

alter_session_set_clause を使用すると、ALTER SESSION 文の有効範囲内で動的なセッション・パラメータおよび初期化パラメータを設定できます。同じ *alter_session_set_clause* で複数のパラメータに対する値を設定できます。

COMMENT を使用すると、コメント文字列をパラメータ値の変更に対応付けることができます。

初期化パラメータおよび ALTER SESSION

ALTER SYSTEM 文で設定可能なすべての初期化パラメータについては、10-20 ページの「[ALTER SYSTEM](#)」を参照してください。10-7 ページの表 10-1 に ALTER SESSION の有効範囲内で動的な初期化パラメータおよび「ALTER SYSTEM」での参照先を示します。ALTER SESSION でパラメータを設定した場合は、設定した値が現行のセッション以外では継続されないことのみが ALTER SYSTEM と異なります。

ALTER SESSION で設定可能な一部のパラメータは、初期化パラメータではありません。つまり、初期化パラメータ・ファイルではなく、ALTER SESSION でのみ設定可能です。これらのセッション・パラメータについては、表 10-1 の後に説明します。

注意： 特に指定がないかぎり、ここで説明するパラメータは初期化パラメータを示し、パラメータの概要のみを説明します。初期化パラメータの値を変更する前に、『Oracle9i データベース・リファレンス』または『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

表 10-1 ALTER SESSION で設定可能な初期化パラメータ

パラメータ	コメント
CURSOR_SHARING (10-44 ページ)	様々な環境での設定の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
DB_BLOCK_CHECKING (10-46 ページ)	ALTER SESSION SET DB_BLOCK_CHECKING による設定は、後続の ALTER SYSTEM SET DB_BLOCK_CHECKING 文で上書きされます。
DB_CREATE_FILE_DEST (10-48 ページ)	—
DB_CREATE_ONLINE_LOG_DEST_n (10-48 ページ)	—
DB_FILE_MULTIBLOCK_READ_COUNT (10-49 ページ)	—
FILESYSTEMIO_OPTIONS (10-59 ページ)	—

表 10-1 ALTER SESSION で設定可能な初期化パラメータ（続き）

パラメータ	コメント
GLOBAL_NAMES (10-61 ページ)	グローバル・ネーム解決およびその実行方法の詳細は、2-112 ページの「リモート・データベース内のオブジェクトの参照」を参照してください。
HASH_AREA_SIZE (10-61 ページ)	—
HASH_JOIN_ENABLED (10-62 ページ)	—
LOG_ARCHIVE_DEST_n (10-70 ページ)	—
LOG_ARCHIVE_DEST_STATE_n (10-71 ページ)	—
LOG_ARCHIVE_MIN_SUCCEED_DEST (10-72 ページ)	—
MAX_DUMP_FILE_SIZE (10-77 ページ)	—
グローバリゼーション・サポート (NLS_) パラメータ	
インスタンスの起動時に、「NLS」で始まる初期化パラメータの値に基づいてグローバリゼーション・サポートが提供されます。動的パフォーマンス表 V\$NLS_PARAMETERS に問い合せて、セッションの現在のグローバリゼーション属性を参照できます。NLS パラメータの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。	
NLS_CALENDAR (10-78 ページ)	—
NLS_COMP (10-79 ページ)	—
NLS_CURRENCY (10-79 ページ)	—
NLS_DATE_FORMAT (10-80 ページ)	有効な日付書式モデルの詳細は、2-64 ページの「日付書式モデル」を参照してください。
NLS_DATE_LANGUAGE (10-80 ページ)	—
NLS_DUAL_CURRENCY (10-80 ページ)	数値書式要素の詳細は、2-60 ページの「数値書式モデル」を参照してください。
NLS_ISO_CURRENCY (10-81 ページ)	—
NLS_LANGUAGE (10-81 ページ)	—

表 10-1 ALTER SESSION で設定可能な初期化パラメータ（続き）

パラメータ	コメント
NLS_LENGTH_SEMANTICS (10-81 ページ)	—
NLS_NCHAR_CONV_EXCP (10-82 ページ)	—
NLS_NUMERIC_CHARACTERS (10-82 ページ)	—
NLS_SORT (10-82 ページ)	—
NLS_TERRITORY (10-83 ページ)	—
NLS_TIMESTAMP_FORMAT (10-83 ページ)	—
NLS_TIMESTAMP_TZ_FORMAT (10-83 ページ)	—
OBJECT_CACHE_MAX_SIZE_PERCENT (10-84 ページ)	—
OBJECT_CACHE_OPTIMAL_SIZE (10-84 ページ)	—
OLAP_PAGE_POOL_SIZE (10-85 ページ)	—
OPTIMIZER_DYNAMIC_SAMPLING (10-86 ページ)	このパラメータの設定方法は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
OPTIMIZER_INDEX_CACHING (10-87 ページ)	—
OPTIMIZER_INDEX_COST_ADJ (10-87 ページ)	—
OPTIMIZER_MAX_PERMUTATIONS (10-87 ページ)	—
OPTIMIZER_MODE (10-88 ページ)	アプリケーションの特性に基づく、コストベース方法に対する目標の決定方法は、『Oracle9i データベース概要』および『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
ORACLE_TRACE_ENABLE (10-89 ページ)	—
PARALLEL_INSTANCE_GROUP (10-92 ページ)	—
PARALLEL_MIN_PERCENT (10-93 ページ)	—

表 10-1 ALTER SESSION で設定可能な初期化パラメータ（続き）

パラメータ	コメント
PARTITION_VIEW_ENABLED (10-94 ページ)	パーティション・ビューの詳細は、16-39 ページの「 パーティション・ビュー 」を参照してください。
PLSQL_COMPILER_FLAGS (10-95 ページ)	—
QUERY_REWRITE_ENABLED (10-98 ページ)	—
QUERY_REWRITE_INTEGRITY (10-99 ページ)	—
REMOTE_DEPENDENCIES_MODE (10-100 ページ)	—
SESSION_CACHED_CURSORS (10-105 ページ)	—
SORT_AREA_RETAINED_SIZE (10-109 ページ)	—
SORT_AREA_SIZE (10-109 ページ)	—
STAR_TRANSFORMATION_ENABLED (10-111 ページ)	—
STATISTICS_LEVEL (10-112 ページ)	—
TIMED_OS_STATISTICS (10-113 ページ)	—
TIMED_STATISTICS (10-113 ページ)	—
TRACE_ENABLED (10-113 ページ)	—
UNDO_SUPPRESS_ERRORS (10-116 ページ)	—
WORKAREA_SIZE_POLICY (10-118 ページ)	—

セッション・パラメータおよび ALTER SESSION

次のパラメータは、セッション・パラメータであり、初期化パラメータではありません。

CONSTRAINT[S]

構文：

```
CONSTRAINT[S] = { IMMEDIATE | DEFERRED | DEFAULT }
```

CONSTRAINT[S] は、遅延可能制約によって指定された条件を、いつ適用するかを指定します。

- `immediate` を設定した場合、遅延可能な制約によって指定される条件は、各 DML 文の直後にチェックされます。これは、セッションの各トランザクションの開始時に、SET CONSTRAINTS ALL IMMEDIATE 文を発行することと同じです。
- `deferred` を設定した場合、遅延可能な制約によって指定される条件は、トランザクションのコミット時にチェックされます。これは、セッションの各トランザクションの開始時に、SET CONSTRAINTS ALL DEFERRED 文を発行することと同じです。
- `default` を設定した場合、すべての制約は各トランザクションの開始時に、DEFERRED または IMMEDIATE の初期状態にリストアされます。

CREATE_STORED_OUTLINES

構文：

```
CREATE_STORED_OUTLINES = {TRUE | FALSE | 'category_name'}
```

CREATE_STORED_OUTLINES パラメータは、セッション中に発行された問合せごとに、アウトラインを自動的に作成および格納するかどうかを決定します。

- `TRUE` を設定した場合、同一セッション内で後続の問合せに対するアウトラインは、自動的に作成可能になります。このアウトラインは、一意のシステム生成名を受け取り、DEFAULT カテゴリに格納されます。すでに特定の問合せに定義したアウトラインが DEFAULT カテゴリに存在する場合は、そのアウトラインが保持され、新しいアウトラインは作成されません。
- `FALSE` を設定した場合、セッション中のアウトラインは作成禁止になります。これはデフォルト値です。
- `category_name` を設定した場合、TRUE と同じ動作をしますが、セッション中に作成されたすべてのアウトラインは、`category_name` カテゴリに格納されます。

CURRENT_SCHEMA

構文：

```
CURRENT_SCHEMA = schema
```

CURRENT_SCHEMA は、セッションの現行のスキーマを、指定したスキーマに変更します。セッション中のスキーマ・オブジェクトに対する後続の未修飾の参照は、この指定したスキーマ内でオブジェクトに変換されます。この設定は、現行のセッションの存続期間中、または ALTER SESSION SET CURRENT_SCHEMA 文を再発行するまで保持されます。

この設定によって、現行のユーザーのスキーマ以外にあるオブジェクトに対する操作を、オブジェクトをスキーマ名で修飾することなく簡単に行えます。この設定によって、現行のスキーマは変更されますが、このセッションのユーザーまたは現行のユーザーは変更されません。また、このセッションに対して、追加のシステム権限またはオブジェクト権限は付与されません。

ERROR_ON_OVERLAP_TIME

構文：

```
ERROR_ON_OVERLAP_TIME = {TRUE | FALSE}
```

ERROR_ON_OVERLAP_TIME には、Oracle が不明瞭な境界日時値（日時が標準か夏時間かが明確でない場合）を処理する方法を指定します。

- TRUE を指定すると、不明瞭なオーバラップ・タイムスタンプに対してエラーが戻されます。
- FALSE を指定すると、不明瞭なオーバラップ・タイムスタンプは標準時刻のデフォルトになります。これはデフォルト値です。

FLAGGER

構文：

```
FLAGGER = { ENTRY | INTERMEDIATE | FULL | OFF }
```

FLAGGER パラメータは、FIPS のフラグ付けを指定します。このフラグ付けを使用した場合、ANSI SQL92 の拡張要素である SQL 文が発行されたときに、エラー・メッセージが生成されます。FLAGGER は、セッション・パラメータであり、初期化パラメータではありません。

Oracle では、ENTRY レベル、INTERMEDIATE レベル、FULL レベルのそれぞれのフラグ付けに違いはありません。セッションでフラグ付けが設定された場合、これに続く ALTER SESSION SET FLAGGER 文は成功しますが、ORA-00097 のメッセージが生成されます。このため、セッションを切断しなくても FIPS のフラグ付けを変更できます。OFF を設定した場合、フラグ付けの使用は停止されます。

参照： 現行の ANSI SQL 規格に対する Oracle の準拠については、[付録 B「Oracle と標準 SQL」](#) を参照してください。

INSTANCE

構文：

```
INSTANCE = integer
```

Real Application Clusters 環境で INSTANCE パラメータを設定すると、セッションが integer で指定したインスタンスに接続する場合と同様に、データベース・ファイルにアクセスします。INSTANCE はセッション・パラメータであり、初期化パラメータではありません。パフォーマンスの最適化のため、Real Application Clusters の各インスタンスは、それぞれ専用のロールバック・セグメントや空きリスト・グループなどを使用しています。Real Application Clusters 環境では、通常、特定のインスタンスに接続し、使用するパーティション化データにアクセスします。別のインスタンスに接続する必要がある場合は、データのパーティション分割はなくなります。このパラメータを設定した場合、インスタンスに接続している場合と同様に、別のインスタンスにもアクセスできます。

ISOLATION_LEVEL

構文：

```
ISOLATION_LEVEL = {SERIALIZABLE | READ COMMITTED}
```

ISOLATION_LEVEL パラメータは、データベースを変更するトランザクションがどのように処理されるかを指定します。ISOLATION_LEVEL はセッション・パラメータであり、初期化パラメータではありません。

- SERIALIZABLE を設定した場合、セッション内のトランザクションは、SQL92 に規定されているとおりシリアル化可能トランザクション分離モードを使用します。シリアル化可能トランザクションが行を更新する DML 文を実行する場合、現在の更新対象の行がそのシリアル化可能トランザクションの開始時にコミットされていない別のトランザクションによって更新されていたときは、その DML 文は失敗します。シリアル化可能トランザクションは、同一トランザクション内で行った更新を確認できます。
- READ COMMITTED を設定した場合、セッション内のトランザクションは、Oracle トランザクションのデフォルトの動作を行います。このため、別のトランザクションでかけられた行ロックを必要とする DML がトランザクションに含まれていると、DML 文は、行ロックが解除されるまで待ち状態になります。

PLSQL_DEBUG

構文：

```
PLSQL_DEBUG = { TRUE | FALSE }
```

PLSQL_DEBUG パラメータには、コンパイル操作中にデバッグ情報を含むか含まないかを設定します。このパラメータに TRUE を設定すると、DEBUG キーワードを ALTER {FUNCTION | PROCEDURE | PACKAGE} COMPILE 文に追加したときと同様に動作します。

SKIP_UNUSABLE_INDEXES

構文：

```
SKIP_UNUSABLE_INDEXES = { TRUE | FALSE }
```

SKIP_UNUSABLE_INDEXES パラメータは、使用禁止の索引または索引パーティションが表にある場合、その表の使用およびレポートを制御します。SKIP_UNUSABLE_INDEXES は、セッション・パラメータで、初期化パラメータではありません。

- TRUE に設定した場合、UNUSABLE のマークが付けられた索引および索引パーティションのエラー・レポートを使用禁止にします。使用禁止の索引または索引パーティションを持つ表に対するすべての操作（挿入、削除、更新および選択）が認められます。

注意： 索引を使用して、表に一意制約を適用する場合、表に対する挿入操作および更新操作を可能にすると、制約に違反する場合があります。そのため、この設定を使用しても、使用禁止の一意索引についてのエラー・レポートは使用禁止になりません。

- FALSE に設定した場合、UNUSABLE のマークが付けられた索引のエラー・レポートを使用可能にします。使用禁止の索引または索引パーティションを持つ表に対する挿入、削除および更新は認められません。これはデフォルト値です。

SQL_TRACE

構文：

```
INSTANCE = integer
```

SQL_TRACE は、初期化パラメータです。ただし、ALTER SESSION 文で値を変更しても、V\$PARAMETER ビューに反映されません。そのため、このコンテキストでは、セッション・パラメータのみを考慮します。

参照： 出力の書式設定および解析方法を含む SQL トレース機能の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

TIME_ZONE

構文：

```
TIME_ZONE = '[+ | -] hh:mm'
             | LOCAL
             | DBTIMEZONE
             | 'time_zone_region'
```


TIME_ZONE パラメータには、現行の SQL セッションのデフォルトのローカル・タイムゾーンによる時差を指定します。TIME_ZONE はセッション・パラメータであり、初期化パラメータではありません。現行のセッションのタイムゾーンを決定するには、組込み関数 SESSIONTIMEZONE を問い合わせます（6-135 ページの [SESSIONTIMEZONE](#) を参照）。

- UTC の前または後の時および分を示す書式マスク ('[+|-] hh:mm') を指定します。hh:mm の有効範囲は、-12:00 ～ +14:00 です。
- LOCAL を指定すると、現行の SQL セッションのデフォルトのローカル・タイムゾーンによる時差が、現行の SQL セッションを起動したときに構築された、元のデフォルトのローカル・タイムゾーンによる時差に設定されます。
- DBTIMEZONE を指定すると、現行のセッションのタイムゾーンにデータベースのタイムゾーンと一致する値が設定されます。この設定を指定する場合、DBTIMEZONE ファンクションは、データベースのタイムゾーンを UTC オフセットまたはタイムゾーン地域として戻します。これはデータベースのタイムゾーンの設定に依存します。
- 有効な time_zone_region を指定します。有効な地域名を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューの TZNAME 列を問い合わせます。この設定を指定する場合、SESSIONTIMEZONE ファンクションは地域の名前を戻します。

注意： 環境変数 ORA_SDTZ を使用して、クライアント・セッションのデフォルトのタイムゾーンを設定することもできます。この変数の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

USE_PRIVATE_OUTLINES

構文：

```
USE_PRIVATE_OUTLINES = { TRUE | FALSE | category_name }
```

USE_PRIVATE_OUTLINES パラメータを使用すると、プライベート・アウトラインの使用を制御することができます。このパラメータが使用可能で、アウトライン化された SQL 文が発行された場合、オブティマイザは、USE_STORED_OUTLINES が使用可能なときにはパブリック領域ではなくそのセッションのプライベート領域からアウトラインを検索します。そのセッションのプライベート領域にアウトラインが存在しない場合、オブティマイザは、文のコンパイルにアウトラインを使用しません。USE_PRIVATE_OUTLINES は、初期化パラメータではありません。

- TRUE に設定した場合、要求をコンパイルするときに、オブティマイザは DEFAULT カテゴリにストアド・プライベート・アウトラインを使用します。
- FALSE に設定した場合、オブティマイザはストアド・プライベート・アウトラインを使用しません。これはデフォルト値です。USE_STORED_OUTLINES が使用可能な場合、オブティマイザはストアド・パブリック・アウトラインを使用します。

- `category_name` を設定した場合、要求をコンパイルするときに、オブティマイザは `category_name` カテゴリにストアド・アウトラインを使用します。

USE_PRIVATE_OUTLINES の制限事項： `USE_STORED_OUTLINES` が使用可能な場合は、このパラメータを使用可能にできません。

USE_STORED_OUTLINES

構文：

```
USE_STORED_OUTLINES = { TRUE | FALSE | category_name }
```

`USE_STORED_OUTLINES` パラメータは、オブティマイザが実行計画を生成するために格納されているパブリック・アウトラインを使用するかどうかを決定します。

`USE_STORED_OUTLINES` は、初期化パラメータではありません。

- `TRUE` に設定した場合、要求をコンパイルするときに、オブティマイザが `DEFAULT` カテゴリにストアド・アウトラインを使用します。
- `FALSE` に設定した場合、オブティマイザがストアド・アウトラインを使用しません。これはデフォルト値です。
- `category_name` を設定した場合、要求をコンパイルするときに、オブティマイザは `category_name` カテゴリにストアド・アウトラインを使用します。

USED_STORED_OUTLINES の制限事項： `USE_PRIVATE_OUTLINES` が使用可能な場合は、このパラメータを使用可能にできません。

例

パラレル DML を使用可能にする例： 現行のセッションのパラレル DML モードを使用可能にする場合、次の文を発行します。

```
ALTER SESSION ENABLE PARALLEL DML;
```

分散トランザクションを強制的に実行する例： 次のトランザクションによって、データベース・リンク `remote` によって識別されるデータベース上の `employees` 表に従業員のレコードを挿入し、`local` によって識別されるデータベース上の `employees` 表から従業員のレコードを削除します。

```
ALTER SESSION  
  ADVISE COMMIT;
```

```
INSERT INTO employees@remote  
VALUES (8002, 'Juan', 'Fernandez', 'juanf@hr.com', NULL,  
       TO_DATE('04-OCT-1992', 'DD-MON-YYYY'), 'SA_CLERK', 3000,  
       NULL, 121, 20);
```

```
ALTER SESSION
  ADVISE ROLLBACK;

DELETE FROM employees@local
  WHERE employee_id = 8002;

COMMIT;
```

このトランザクションには、ADVISE 句を指定した ALTER SESSION 文が 2 つあります。このトランザクションが状態不明（インダウト）になった場合、remote には、最初に指定した ALTER SESSION 文によって 'COMMIT' が送信され、local には、2 番目の指定によって 'ROLLBACK' が送信されます。

データベース・リンクをクローズする例： 次の文は、データベース・リンクを使用している local データベース上の jobs 表を更新し、このトランザクションをコミットして、データベース・リンクを明示的にクローズします。

```
UPDATE jobs@local SET min_salary = 3000
  WHERE job_id = 'SH_CLERK';

COMMIT;
```

```
ALTER SESSION
  CLOSE DATABASE LINK local;
```

日付書式の動的な変更例： 次の文は、セッションのデフォルトの日付書式を 'YYYY MM DD-HH24:MI:SS' に動的に変更します。

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

変更後は、新しい日付書式が次のように適用されます。

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

```
TODAY
-----
2001 04 12 12:30:38
```

日付言語の動的な変更例： 次の文は、日付書式要素の言語をフランス語に変更します。

```
ALTER SESSION
  SET NLS_DATE_LANGUAGE = French;

SELECT TO_CHAR(SYSDATE, 'Day DD Month YYYY') Today
```

```
FROM DUAL;

TODAY
-----
Jeudi    12 Avril    2001
```

ISO 通貨の変更例： 次の文では、ISO 通貨記号をアメリカ合衆国の ISO 通貨記号に動的に変更します。

```
ALTER SESSION
  SET NLS_ISO_CURRENCY = America;

SELECT TO_CHAR( SUM(salary), 'C999G999D99') Total
FROM employees;

TOTAL
-----
USD694,900.00
```

小数点文字と桁区切りの変更例： 次の文は、小数点文字をカンマ (,) に、桁区切りをピリオド (.) に動的に変更します。

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ',.' ;

これらの数値書式要素を使用した場合、新しい文字が戻ります。

ALTER SESSION SET NLS_CURRENCY = 'FF';

SELECT TO_CHAR( SUM(salary), 'L999G999D99') Total FROM employees;

TOTAL
-----
FF694.900,00
```

NLS 通貨の変更例： 次の文は、各国通貨記号を 'DM' に動的に変更します。

```
ALTER SESSION
  SET NLS_CURRENCY = 'DM';

SELECT TO_CHAR( SUM(salary), 'L999G999D99') Total
FROM employees;

TOTAL
-----
DM694.900,00
```

NLS 言語の変更例： 次の文は、表示されたエラー・メッセージの言語をフランス語に動的に変更します。

```
ALTER SESSION
  SET NLS_LANGUAGE = FRENCH;

Session modifiee.

SELECT * FROM DMP;

ORA-00942:Table ou vue inexistante
```

言語のソート順の変更例： 次の文は、言語ソート順序をスペイン語に動的に変更します。

```
ALTER SESSION
  SET NLS_SORT = XSpanish;
```

これによって、文字の値はスペイン語のソート順序に基づいてソートされます。

SQL トレースを使用可能にする例： 次の文は、セッションに対して SQL トレース機能を使用可能にします。

```
ALTER SESSION
  SET SQL_TRACE = TRUE;
```

クエリー・リライトを使用可能にする例： 次の文は、明示的に使用禁止にされていないすべてのマテリアライズド・ビューに対する現行のセッションのクエリー・リライトを使用可能にします。

```
ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE;
```

ALTER SYSTEM

用途

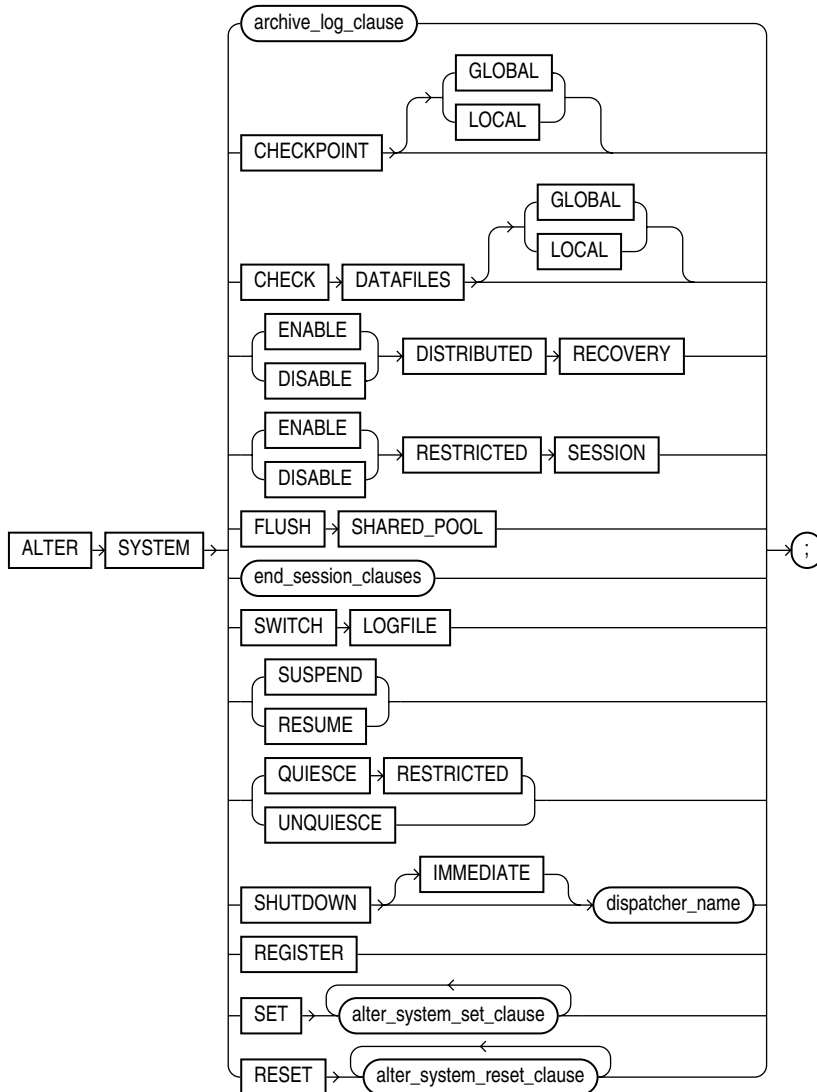
ALTER SYSTEM 文を使用すると、Oracle インスタンスを動的に変更できます。この設定は、データベースがマウントされているかぎり有効です。

前提条件

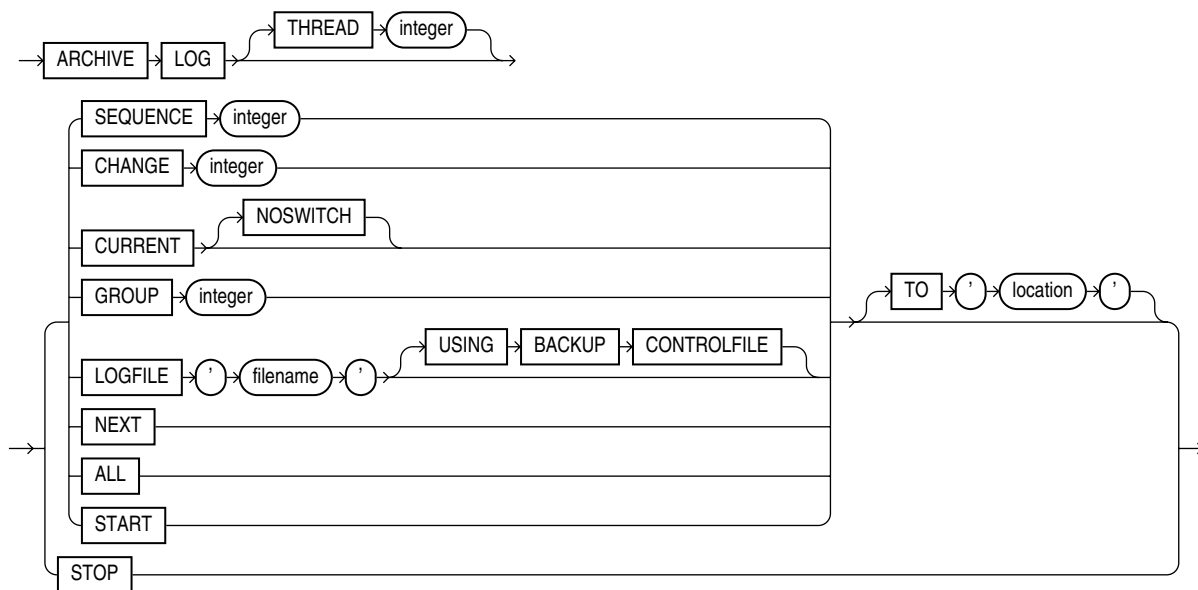
ALTER SYSTEM システム権限が必要です。

構文

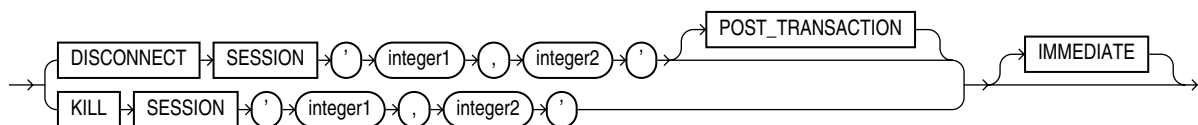
alter_system::=



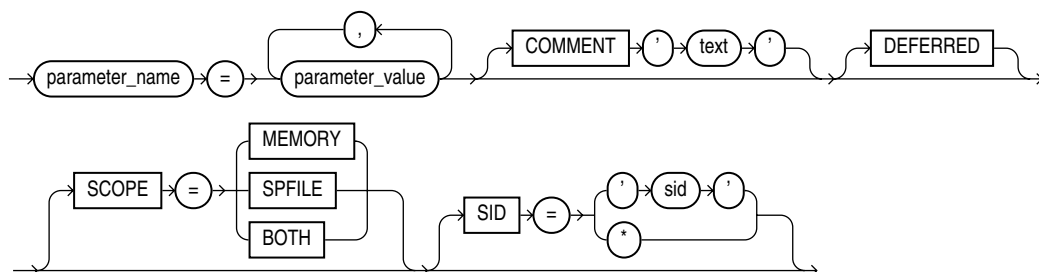
archive_log_clause::=



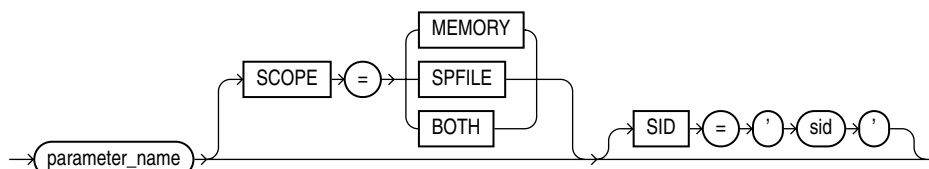
end_session_clauses::=



alter_system_set_clause::=



`alter_system_reset_clause::=`



セマンティクス

archive_log_clause

archive_log_clause は、REDO ログ・ファイルを手動でアーカイブするか、または自動アーカイブを使用可能または使用禁止にします。この句を使用する場合、インスタンスでデータベースをマウントする必要があります。特に指定がないかぎり、データベースはオープンまたはクローズできます。

THREAD 句

THREAD を指定すると、アーカイブする REDO ログ・ファイル・グループを含むスレッドを指定できます。

THREAD 句の制限事項： Real Application Clusters 環境で Oracle を使用する場合のみ、このパラメータを設定します。

参照： 10-118 ページの「[REDO ログの手動アーカイブの例:](#)」を参照してください。

SEQUENCE 句

SEQUENCE を指定すると、指定したスレッド内のログ順序番号 *integer* によって識別されるオンライン REDO ログ・ファイル・グループを手動でアーカイブできます。THREAD パラメータを指定しなかった場合、インスタンスに割り当てられているスレッドから、指定したグループがアーカイブされます。

CHANGE 句

CHANGE を指定すると、オンライン REDO ログ・ファイル・グループを、手動でアーカイブできます。このグループには、指定したスレッド内の *integer* によって識別される SCN を持つ REDO ログ・エントリが含まれます。この SCN が現行の REDO ログ・ファイル・グループ内にある場合、ログ・スイッチが実行されます。THREAD パラメータを指定しない場合、使用可能な状態にあるすべてのスレッドから、この SCN を含むグループがアーカイブされます。

インスタンスでデータベースをオープンしている場合にのみ、この句を使用できます。

CURRENT 句

CURRENT を指定すると、ログ・スイッチを強制的に発生させ、指定したスレッドの現行の REDO ログ・ファイル・グループを手動でアーカイブできます。THREAD パラメータを指定しない場合、すべての使用可能なスレッドから、現行のログ以前のログも含むすべての REDO ログ・ファイル・グループがアーカイブされます。データベースがオープンしているときのみ、CURRENT を指定できます。

NOSWITCH ログ・スイッチの強制実行なしで現行の REDO ログ・ファイル・グループを手動でアーカイブする場合は、NOSWITCH を指定します。この設定は、プライマリ・データベースが停止したときに、データ分岐が発生しないようにするために、主にスタンバイ・データベースで使用されます。データ分岐は、プライマリ・データベースに障害が発生した場合に、データが消失する可能性があることを意味します。

インスタンスでデータベースがマウントされているがオープンされていない場合にのみ、NOSWITCH 句を使用できます。データベースがオープンしている場合は、この操作によってデータベースは自動的にクローズされます。再オープンする前にデータベースを手動で停止する必要があります。

GROUP 句

GROUP を指定すると、オンライン REDO ログ・ファイル・グループを手動でアーカイブできます。このグループには、integer によって識別される GROUP 値が含まれます。この REDO ログ・ファイル・グループの GROUP 値は、データ・ディクショナリ・ビュー DBA_LOG_FILES に問い合わせて確認できます。THREAD パラメータと GROUP パラメータの両方を指定する場合は、指定する REDO ログ・ファイル・グループが、指定するスレッド内に含まれている必要があります。

LOGFILE 句

LOGFILE を指定すると、オンライン REDO ログ・ファイル・グループを手動でアーカイブできます。このグループには、'filename' によって識別される REDO ログ・ファイル・メンバーが含まれます。THREAD パラメータと LOGFILE パラメータの両方を指定する場合は、指定する REDO ログ・ファイル・グループが、指定するスレッド内に含まれている必要があります。

データベースがバックアップ制御ファイルでマウントされている場合は、USING BACKUP CONTROLFILE を指定し、現行のログ・ファイルを含むすべてのオンライン・ログ・ファイルのアーカイブを許可します。

LOGFILE 句の制限事項： REDO ログ・ファイル・グループは、一杯になった順にアーカイブする必要があります。LOGFILE パラメータを使用して REDO ログ・ファイル・グループのアーカイブを指定した場合、それ以前の REDO ログ・ファイル・グループがアーカイブされていないとエラー・メッセージが戻ります。

NEXT 句

NEXT を指定すると、一杯になってもアーカイブされていない次のオンライン REDO ログ・ファイルを、指定したスレッドから手動でアーカイブできます。THREAD パラメータを指定しない場合、使用可能な任意のスレッド上の、アーカイブされていない最初の REDO ログ・ファイル・グループがアーカイブされます。

ALL 句

ALL を指定すると、一杯になってもアーカイブされていないすべてのオンライン REDO ログ・ファイルを、指定したスレッドから手動でアーカイブできます。THREAD パラメータを指定しない場合、使用可能なすべてのスレッドから、一杯でアーカイブされていないすべての REDO ログ・ファイル・グループがアーカイブされます。

START 句

START を指定すると、REDO ログ・ファイル・グループの自動アーカイブを使用可能にできます。

START 句の制限事項： インスタンスに割り当てられているスレッドについてのみ、自動アーカイブを使用可能にできます。

TO location 句

TO 'location' を指定すると、REDO ログ・ファイル・グループがアーカイブされる位置を指定できます。このパラメータの値は、オペレーティング・システムの規則に従って、ファイルの位置を完全に指定する必要があります。このパラメータを指定しない場合、REDO ログ・ファイル・グループは初期化パラメータ LOG_ARCHIVE_DEST または LOG_ARCHIVE_DEST_n に指定された場所に格納されます。

STOP 句

STOP を指定すると、REDO ログ・ファイル・グループの自動アーカイブを使用禁止にできます。インスタンスに割り当てられているスレッドについてのみ、自動アーカイブを使用禁止にできます。

CHECKPOINT 句

CHECKPOINT を指定すると、チェックポイントを明示的に強制処理させ、コミット済のトランザクションによる変更をディスク上のデータ・ファイルに書き込みます。インスタンスでデータベースがオープンしている場合にのみ、この句を指定できます。チェックポイントが完了するまで、ユーザーに制御は戻りません。

GLOBAL Real Application Clusters 環境で、データベースをオープンしているすべてのインスタンスに対してチェックポイントを実行します。これはデフォルト値です。

LOCAL Real Application Clusters 環境で、文を発行するインスタンスの REDO ログ・ファイル・グループのスレッドに対してのみチェックポイントを実行します。

参照： 10-119 ページの「[チェックポイントの発生例:](#)」を参照してください。

CHECK DATAFILES 句

Real Application Clusters 環境などの分散データベース・システムで、データベース・コントロール・ファイルからインスタンスの SGA を更新し、すべてのオンライン・データ・ファイルに情報を反映します。

- GLOBAL を指定すると、データベースをオープンしているすべてのインスタンスに対して、この同期化を実行します。これはデフォルト値です。
- LOCAL を指定すると、ローカル・インスタンスに対してのみこの同期化を実行します。インスタンスでデータベースをオープンしておく必要があります。

end_session_clauses

end_session_clauses を使用すると、現行のセッションを終了することができます。

DISCONNECT SESSION 句

DISCONNECT SESSION 句を使用すると、専用サーバーのプロセス（接続が Shared Server 経由の場合は、仮想回路）を破棄することによって、現行のセッションが切断されます。この句を使用する場合、インスタンスでデータベースをオープンする必要があります。次の両方の値を V\$SESSION ビューで指定して、このセッションを識別する必要があります。

- *integer1* には、SID 列の値を指定します。
- *integer2* には、SERIAL# 列の値を指定します。

システム・パラメータを適切に設定した場合、アプリケーション・フェイルオーバーが有効になります。

- `POST_TRANSACTION` を設定すると、セッションが切断される前に、実行中のトランザクションを完了します。セッションに実行中のトランザクションがない場合、この句は、後述の `KILL SESSION` と同様に動作します。
- `IMMEDIATE` を設定すると、セッションを切断し、実行中のトランザクションの完了を待たずに、すぐにセッション全体の状態をリカバリします。
 - `POST_TRANSACTION` を指定した場合、セッションに実行中のトランザクションがあれば、`IMMEDIATE` キーワードは無視されます。
 - `POST_TRANSACTION` を指定しない場合、または `POST_TRANSACTION` を指定していてもセッションに実行中のトランザクションがない場合、この句は、後述の `KILL SESSION IMMEDIATE` と同様の効果があります。

参照： 10-122 ページの「[セッションの切断例：](#)」を参照してください。

KILL SESSION 句

`KILL SESSION` を指定すると、セッションに終了済のマークが付き、実行中のトランザクションがロールバックされ、すべてのセッション・ロックが解放され、セッション・リソースが一部リカバリされます。この句を使用するには、インスタンスでデータベースをオープンしている必要があります。また、自セッションおよび中断されるセッションは、同じインスタンスにある必要があります。次の両方の値を `V$SESSION` ビューで指定して、このセッションを識別する必要があります。

- `integer1` には、SID 列の値を指定します。
- `integer2` には、`SERIAL#` 列の値を指定します。

リモート・データベースからの応答を待ったり、トランザクションをロールバックするなど、最後まで完了する必要があるアクティビティをセッションが実行している場合、Oracle はこのアクティビティが完了するまで待機し、セッションに終了済のマークを付け、その後、ユーザーに制御を戻します。待ち時間が 1 分以上続く場合は、中断されるセッションにマークが付けられ、マークが付けられたセッションが中断されることを示すメッセージとともにユーザーに制御が戻されます。アクティビティが完了すると、`PMON` バックグラウンド・プロセスは、セッションに終了済のマークを付けます。

セッションに実行中のトランザクションがあるかどうかにかかわらず、セッション・ユーザーがセッションに要求を発行してセッションが中断されたことを示すメッセージを受け取るまで、Oracle は、セッション全体の状態をリカバリしません。

参照： 10-121 ページの「[セッションの中断例：](#)」を参照してください。

IMMEDIATE IMMEDIATE を指定すると、実行中のトランザクションをロールバックしてすべてのセッション・ロックを解放し、セッション全体の状態をリカバリしてから、ユーザーに制御を戻すように Oracle に指示できます。

DISTRIBUTED RECOVERY 句

DISTRIBUTED RECOVERY 句によって、分散リカバリを使用可能または使用禁止にします。この句を使用する場合、インスタンスでデータベースをオープンする必要があります。

ENABLE ENABLE を指定すると、分散リカバリが使用可能になります。シングルプロセス環境では、分散リカバリを開始する場合にこの句を使用する必要があります。

トランザクションに関係があるリモート・ノードにアクセスできない場合、インダウト・トランザクションをリカバリするには、ENABLE DISTRIBUTED RECOVERY 文を複数回発行する必要があります。インダウト・トランザクションは、データ・ディクショナリ・ビュー DBA_2PC_PENDING に表示されます。

参照： 10-121 ページの「[分散リカバリを使用可能にする例：](#)」を参照してください。

DISABLE DISABLE を指定すると、分散リカバリが使用禁止になります。

RESTRICTED SESSION 句

RESTRICTED SESSION を指定すると、Oracle にログインできるユーザーを制限できます。

インスタンスでデータベースがマウントされていてもデスマウントされていても、またはオープン状態でもクローズ状態でも、この句を使用できます。

ENABLE ENABLE を指定すると、RESTRICTED SESSION システム権限が付与されているユーザーのみが Oracle にログインできます。既存のセッションは終了しません。

DISABLE DISABLE を指定すると、ENABLE RESTRICTED SESSION 句とは逆の動作をします。つまり、CREATE SESSION システム権限が付与されているすべてのユーザーが Oracle にログインできます。これはデフォルト値です。

参照： 10-119 ページの「[セッションへのログインを制限する例：](#)」を参照してください。

FLUSH SHARED_POOL 句

FLUSH SHARED_POOL 句を指定すると、SGA の共有プール上のすべてのデータが消去されます。共有プールは次のものを格納します。

- キャッシュされたデータ・ディクショナリ情報
- SQL 文の共有 SQL 領域、共有 PL/SQL 領域、ストアド・プロシージャ、ファンクション、パッケージおよびトリガー

この文は、現在実行中の項目に対する共有 SQL 領域および共有 PL/SQL 領域を消去しません。インスタンスでデータベースがマウントされていてもディスマウントされていても、またはオープン状態でもクローズ状態でも、この句を使用できます。

参照： 10-119 ページの「[共有プールの消去例：](#)」を参照してください。

SWITCH LOGFILE 句

SWITCH LOGFILE 句を指定すると、現行の REDO ログ・ファイル・グループのファイルが一杯であるかどうかにかかわらず、新しい REDO ログ・ファイル・グループへの書込みを明示的に強制開始させます。ログ・スイッチを強制的に発生させると、チェックポイントが実行されますが、チェックポイントが完了する前に、すぐに制御が戻されます。この句を使用する場合、インスタンスでデータベースをオープンする必要があります。

参照： 10-121 ページの「[ログ・スイッチの発生例：](#)」を参照してください。

SUSPEND | RESUME

SUSPEND 句を指定すると、すべての I/O（データ・ファイル、制御ファイルおよびファイル・ヘッダー）を停止し、すべてのインスタンスで実行中のトランザクションを処理せずにデータベースのコピーを作成可能にします。

SUSPEND および RESUME の制限事項：

- ホット・バックアップ・モードでデータベースの表領域を確保するまで、この句は使用できません。
- システムが停止中に新しいインスタンスを起動する場合、この新しいインスタンスは停止しません。

RESUME 句を指定すると、問合せおよび I/O に対して、再度、データベースを使用可能にします。

QUIESCE RESTRICTED | UNQUIESCE

QUIESCE RESTRICTED 句および UNQUIESCE 句を使用すると、データベースを**休止状態**にしたり、休止状態から戻すことができます。この状態では、データベース管理者は、トランザクション、問合せまたは PL/SQL 操作が同時に存在する状態では安全に実行できない管理操作を実行することができます。

注意： QUIESCE RESTRICTED 句は、データベース・リソース・マネージャ機能がインストールされている場合、およびデータベースをオープンしたインスタンスでデータベースが起動された後、リソース・マネージャが継続的にアクティブになっている場合のみ有効です。

複数の QUIESCE RESTRICTED 文または UNQUIESCE 文が異なるセッションまたはインスタンスで同時に発行された場合、1 つを除いた他のすべてのセッションまたはインスタンスにエラーが戻されます。

QUIESCE RESTRICTED

QUIESCE RESTRICTED を指定すると、データベースは休止状態になります。この句は、データベースがオープン中のすべてのインスタンスに次の影響を与えます。

- Oracle は、すべてのインスタンスのデータベース・リソース・マネージャに、アクティブでないすべてのセッション（SYS および SYSTEM 以外）をアクティブにしないように指示します。SYS および SYSTEM 以外のユーザーは、新しいトランザクション、問合せ、フェッチまたは PL/SQL 操作を起動できません。
- Oracle は、SYS または SYSTEM 以外のユーザーが開始した、すべてのインスタンスの既存のトランザクションが終了するまで（コミットまたは異常終了するまで）待機します。Oracle は、SYS または SYSTEM 以外のユーザーが開始した、内部トランザクションにない、すべてのインスタンスで実行中のすべての問合せ、フェッチおよび PL/SQL プロシージャが終了するまで待機します。連続する複数の Oracle Call インタフェース (OCI) のフェッチによって問合せが実行される場合、Oracle はすべてのフェッチが終了するまで待機しません。現行のフェッチが終了するまで待機しますが、次のフェッチは行われません。Oracle は、エンキューなどの共有リソースを保持するすべてのセッション（SYS および SYSTEM 以外のセッション）がリソースを解放するまで待機します。すべての操作が完了した後、Oracle はデータベースを休止状態にし、QUIESCE RESTRICTED 文の実行を終了します。
- 共有サーバー・モードでインスタンスを実行している場合は、Oracle は、データベース・リソース・マネージャに、SYS または SYSTEM 以外のユーザーによるログインを阻止するように指示します。非共有サーバー・モードでインスタンスを実行している場合、そのインスタンスへのユーザー・ログインに制限はありません。

休止状態中、すべてのインスタンスにおいてリソース・マネージャのプランは変更できません。

UNQUIESCE

UNQUIESCE を指定すると、データベースを休止状態から戻します。これによって、SYS または SYSTEM 以外のユーザによって開始された、トランザクション、問合せ、フェッチおよび PL/SQL プロシージャを再開できます。UNQUIESCE 文は、QUIESCE RESTRICTED 文を発行したセッションと同じセッションで起動する必要はありません。

SHUTDOWN 句

SHUTDOWN 句は、システムに Oracle の共有サーバー・アーキテクチャを使用している場合のみに関連します。*dispatcher_name* で識別されたディスパッチャを停止します。*dispatcher_name* は、'Dxxx' という形式の文字列である必要があります。xxx はディスパッチャの数です。ディスパッチャ名のリストを取得するには、V\$DISPATCHER 動的パフォーマンス・ビューの NAME 列を問い合わせます。

- IMMEDIATE を指定した場合、ディスパッチャは新しい接続の受入れをすぐに中止し、そのディスパッチャによる既存の接続はすべて終了されます。すべてのセッションがクリーンアップされてから、ディスパッチャ・プロセスは停止します。
- IMMEDIATE を指定しない場合、ディスパッチャは新しい接続の受入れをすぐに中止しますが、すべてのユーザーが切断し、すべてのデータベース・リンクが終了されるのを待ちます。その後、ディスパッチャは停止されます。

REGISTER 句

REGISTER を指定すると、PMON バックグラウンド・プロセスによってリスナーにインスタンスがすぐに登録されます。この句を指定しない場合、PMON が次に検出ルーチンを実行するまでインスタンスの登録は行われません。その結果、クライアントは、リスナー起動後最大 60 秒間サービスにアクセスできない可能性があります。

参照： PMON バックグラウンド・プロセスおよびリスナーの詳細は、『Oracle9i データベース概要』および『Oracle9i Net Services 管理者ガイド』を参照してください。

alter_system_set_clause

alter_system_set_clause を使用すると、すべての初期化パラメータ値を設定またはリセットできます。パラメータの詳細は、10-34 ページの「[初期化パラメータおよび ALTER SYSTEM](#)」を参照してください。

初期化パラメータ値を変更する権限は、以前のパラメータ・ファイル (pfile) を使用してデータベースを起動したか、サーバー・パラメータ・ファイル (spfile) を使用してデータベースを起動したかによって異なります。特定のパラメータ値を変更する権限を持っているかどうかを確認するには、V\$PARAMETER 動的パフォーマンス・ビューの ISSYS_MODIFIABLE 列を問い合わせます。

パラメータ値を設定するときに、次の設定も行えます。

COMMENT

COMMENT 句を使用すると、コメント文字列をパラメータ値の変更に対応付けることができます。SPFILE をあわせて指定すると、パラメータ・ファイルにコメントが表示され、そのパラメータに対する直前の変更がわかります。

DEFERRED

DEFERRED キーワードを指定すると、データベースに接続するその後のセッションに対するパラメータの値を設定または変更できます。現行のセッションでは変更前の値が残ります。

このパラメータの V\$PARAMETER の ISSYS_MODIFIABLE 列の値が DEFERRED の場合は、DEFERRED を指定する必要があります。この列の値が IMMEDIATE の場合、この句の DEFERRED キーワードはオプションです。この列の値が FALSE の場合、この ALTER SYSTEM 文は DEFERRED を指定できません。

参照： V\$PARAMETER 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

SCOPE

SCOPE 句を使用すると、変更が有効になるタイミングを指定できます。有効範囲は、データベースの起動に使用したファイルがパラメータ・ファイル (pfile) か、サーバー・パラメータ・ファイル (spfile) かによって異なります。

MEMORY MEMORY を指定すると、変更がメモリーで行われ、すぐに有効になり、データベースが停止するまで持続されます。パラメータ・ファイル (pfile) を使用してデータベースを起動すると、指定した範囲のみで有効になります。

SPFILE SPFILE を指定すると、変更がサーバー・パラメータ・ファイルで行われます。新しい設定は、データベースが次に停止し、再起動されたときに有効になります。静的パラメータ値を変更したときには、SPFILE を指定する必要があります。

BOTH BOTH を指定すると、変更がメモリーとサーバー・パラメータ・ファイルの両方で行われます。新しい設定はすぐに有効になり、データベースが停止し、再起動された後も持続します。

データベースの起動でサーバー・パラメータ・ファイルを使用した場合は、BOTH がデフォルトです。データベースの起動でパラメータ・ファイルを使用した場合は、MEMORY がデフォルトで、指定した範囲のみで有効になります。

SID

SID 句は、Real Application Clusters 環境のみに関連します。この句を使用すると、値を有効にするインスタンスの SID を指定することができます。

- すべてのインスタンスに対してパラメータ値を変更する場合、SID = '*' を指定します。
- sid のインスタンスのみでパラメータ値を変更する場合、SID = 'sid' を指定します。この設定は、SID = '*' を指定する前後の ALTER SYSTEM SET 文より優先されます。

この句を指定しない場合は、次のようになります。

- pfile（クライアント側の初期化パラメータ・ファイル）を使用してインスタンスを起動する場合、現行のインスタンスの SID とみなされます。
- spfile（サーバー・パラメータ・ファイル）を使用してインスタンスを起動する場合、SID = '*' を指定したとみなされます。

現行のインスタンス以外のインスタンスを指定すると、そのインスタンスに、メモリーのパラメータ値の変更を通知するメッセージが送信されます。

参照： V\$PARAMETER ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

alter_system_reset_clause

alter_system_reset_clause は、Real Application Clusters 環境で使用されます。サーバー・パラメータ・ファイルですべてのインスタンス用にパラメータが設定されていても、各インスタンスを個々に制御することができます。SCOPE 句は、alter_system_set_clause と同様に動作します。

SID SID 句を指定すると、以前、インスタンスに対して ALTER SYSTEM SET ...SID = 'sid' 文で設定したパラメータを削除できます。インスタンスは、パラメータ値を前後の ALTER SYSTEM SET ...SID = '*' 文で指定されたものとみなします。

参照： Real Application Clusters 環境の各インスタンス用のパラメータ設定の詳細は、『Oracle9i Real Application Clusters 配置およびパフォーマンス』を参照してください。

初期化パラメータおよび ALTER SYSTEM

ここでは、初期化パラメータの概要をアルファベット順に示します。パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

ACTIVE_INSTANCE_COUNT

パラメータ・タイプ	整数
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	1 またはクラスタ内のインスタンスの数以上 > (1 以外の値はインスタンスのアクティブまたはスタンバイ状態に影響しない)
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

注意： このパラメータは、インスタンスを 2 つのみ持つクラスタ内でのみ機能します。

ACTIVE_INSTANCE_COUNT を使用すると、2 インスタンス・クラスタの 1 つのインスタンスをプライマリ・インスタンスとして、別のインスタンスをセカンダリ・インスタンスとして指定できます。このパラメータは 3 つ以上のインスタンスを持つクラスタでは機能しません。

AQ_TM_PROCESSES

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ～ 10

AQ_TM_PROCESSES によって、キュー・メッセージの時間監視が使用可能になります。キュー・メッセージの時間は、遅延および時間切れプロパティを示すメッセージで使用できます。1 ～ 10 までの値を指定すると、メッセージの監視用のキュー・モニター・プロセスがその数だけ作成されます。AQ_TM_PROCESSES に値を設定しないか、または 0 に設定した場合、キュー・モニターは作成されません。

ARCHIVE_LAG_TARGET

パラメータ・タイプ	整数
デフォルト値	0（使用禁止）
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0、または 60 ～ 7200 の整数
Real Application Clusters	複数インスタンスには同じ値を使用する必要がある。

ARCHIVE_LAG_TARGET によって、データ損失量を制限できます。また、指定した時間間隔でログ・スイッチを強制することによって、スタンバイ・データベースの可用性を効果的に高めることができます。

AUDIT_FILE_DEST

パラメータ・タイプ	文字列
構文	AUDIT_FILE_DEST = 'directory'
デフォルト値	ORACLE_HOME/rdbms/audit
パラメータ・クラス	静的

AUDIT_FILE_DEST には、監査ファイルが格納されているディレクトリを指定します。

AUDIT_SYS_OPERATIONS

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

AUDIT_SYS_OPERATIONS によって、ユーザー SYS、および SYSDBA または SYSOPER 権限で接続しているユーザーから発行された操作の監査が使用可能または使用禁止になります。監査レコードは、オペレーティング・システムの監査証跡に書き込まれます。

AUDIT_TRAIL

パラメータ・タイプ 文字列

構文 AUDIT_TRAIL = {NONE | FALSE | DB | TRUE | OS}

デフォルト値 なし

パラメータ・クラス 静的

AUDIT_TRAIL は、監査証跡への行の自動書込みを使用可能または使用禁止にするために使用します。

BACKGROUND_CORE_DUMP

パラメータ・タイプ 文字列

構文 BACKGROUND_CORE_DUMP = {PARTIAL | FULL}

デフォルト値 partial

パラメータ・クラス 静的

BACKGROUND_CORE_DUMP は、Oracle バックグラウンド・プロセスのコア・ファイルに SGA を含めるかどうかを指定します。

BACKGROUND_DUMP_DEST

パラメータ・タイプ 文字列

構文 BACKGROUND_DUMP_DEST = {*pathname* | *directory*}

デフォルト値 オペレーティング・システム依存

パラメータ・クラス 動的:ALTER SYSTEM

値の範囲 有効なローカル・パス、ディレクトリまたはディスク

BACKGROUND_DUMP_DEST には、Oracle の操作中に、バックグラウンド・プロセス (LGWR、DBW*n* など) のデバッグ・トレース・ファイルが書き込まれる (ディレクトリまたはディスクの) パス名を指定します。

BACKUP_TAPE_IO_SLAVES

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SYSTEM ... DEFERRED
値の範囲	true false

BACKUP_TAPE_IO_SLAVES には、Recovery Manager がテープへのデータのバックアップ、コピー、リストアに I/O サーバー・プロセス（スレーブ）を使用するかどうかを指定します。値を true に設定すると、I/O サーバー・プロセスはテープ・デバイスに対する書込みまたは読込みに使用されます。値を false（デフォルト）に設定すると、I/O サーバー・プロセスはバックアップに使用されません。そのかわり、バックアップを行うシャドウ・プロセスがテープ・デバイスにアクセスします。

BITMAP_MERGE_AREA_SIZE

パラメータ・タイプ	整数
デフォルト値	1048576 (1MB)
パラメータ・クラス	静的
値の範囲	オペレーティング・システム依存

注意： インスタンス共有サーバー・オプションで構成されていないかぎり、BITMAP_MERGE_AREA_SIZE パラメータを使用することはお薦めしません。かわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。

BITMAP_MERGE_AREA_SIZE は、下位互換性を保つために残されます。

BITMAP_MERGE_AREA_SIZE は、ビットマップ索引を含むシステムのみに関連します。このパラメータには、索引のレンジ・スキャンによって取り出されたビットマップをマージするために使用するメモリー容量を指定します。デフォルト値は 1MB です。シングル・ビットマップにマージするにはビットマップ・セグメントをソートする必要があるため、通常、より大きい値を指定するとパフォーマンスが向上します。

BLANK_TRIMMING

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

BLANK_TRIMMING には、文字データ型のデータ割当て方法を指定します。

BUFFER_POOL_KEEP

パラメータ・タイプ	文字列
構文	<pre>BUFFER_POOL_KEEP = {integer (BUFFERS:integer, LRU_LATCHES:integer)}</pre> <i>integer</i> は、バッファ件数および LRU ラッチ数 (オプション)
デフォルト値	なし
パラメータ・クラス	静的

注意： DB_KEEP_CACHE_SIZE パラメータが設定されている場合、このパラメータは無視されます。かわりに、DB_KEEP_CACHE_SIZE を使用することをお勧めします。また、BUFFER_POOL_KEEP は、新しい DB_KEEP_CACHE_SIZE 動的パラメータと組み合わせることはできません。同じパラメータ・ファイルでこの 2 つのパラメータを組み合わせる場合、エラーが発生します。BUFFER_POOL_KEEP は、下位互換性を保つために残されます。

BUFFER_POOL_KEEP は、バッファの合計数 (DB_BLOCK_BUFFERS パラメータの値) の近くに KEEP バッファ・プールとして設定することによって、オブジェクトをバッファ・キャッシュに保存できます。また、LRU ラッチの合計数の指定部分を KEEP バッファ・プールに割り当てることができます。

BUFFER_POOL_RECYCLE

パラメータ・タイプ 文字列

構文 `BUFFER_POOL_RECYCLE = {integer | (BUFFERS:integer, LRU_LATCHES:integer)}`
integer は、バッファ件数および LRU ラッチ数（オプション）

デフォルト値 なし

パラメータ・クラス 静的

注意： DB_RECYCLE_CACHE_SIZE パラメータが設定されている場合、このパラメータは無視されます。かわりに、DB_RECYCLE_CACHE_SIZE を使用することをお勧めします。また、BUFFER_POOL_RECYCLE は、新しい DB_RECYCLE_CACHE_SIZE 動的パラメータと組み合わせることはできません。同じパラメータ・ファイルでこの 2 つのパラメータを組み合わせた場合、エラーが発生します。BUFFER_POOL_RECYCLE は、下位互換性を保つために残されます。

BUFFER_POOL_RECYCLE は、バッファの合計数（DB_BLOCK_BUFFERS パラメータの値）の近くに RECYCLE バッファ・プールとして設定することによって、バッファ・キャッシュ内のオブジェクト・サイズを制限できます。また、LRU ラッチの合計数の指定部分を RECYCLE バッファ・プールに割り当てることができます。

CIRCUITS

パラメータ・タイプ 整数

デフォルト値 次の値が導出される：

- SESSIONS 値（共有サーバー・アーキテクチャを使用する場合）
- 0（共有サーバー・アーキテクチャを使用しない場合）

パラメータ・クラス 静的

CIRCUITS には、インバインドおよびアウトバインド・ネットワーク・セッションに使用可能な、バーチャル・サーキットの合計数を指定します。このパラメータは、インスタンスに必要な SGA の総量に影響するパラメータの 1 つです。

CLUSTER_DATABASE

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

CLUSTER_DATABASE は、Oracle9i Real Application Clusters が使用可能かどうかを指定する Oracle9i Real Application Clusters のパラメータです。

CLUSTER_DATABASE_INSTANCES

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	静的
値の範囲	0（ゼロ）以外の任意の値

CLUSTER_DATABASE_INSTANCES は、クラスタ・データベースの一部として構成されたインスタンスの数を指定する Oracle9i Real Application Clusters のパラメータです。すべてのインスタンスにこのパラメータを設定する必要がある。通常、このパラメータを、Oracle9i Real Application Clusters 環境のインスタンスの数に設定してください。このパラメータを適切に設定すると、メモリー使用を改善できます。

CLUSTER_INTERCONNECTS

パラメータ・タイプ	文字列
構文	CLUSTER_INTERCONNECTS = ifn [: ifn ...]
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	コロンで区切られた、1 つ以上の IP アドレス

CLUSTER_INTERCONNECTS は、Oracle9i Real Application Clusters 環境で使用可能な追加のクラスタ・インターコネクトに関する情報を Oracle に提供します。

COMMIT_POINT_STRENGTH

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	静的
値の範囲	0 ～ 255

COMMIT_POINT_STRENGTH は、分散データベース・システムのみに関連しています。このパラメータには、分散トランザクション内の**コミット・ポイント・サイト**を決定する値を指定します。COMMIT_POINT_STRENGTH について最大の値を持つトランザクションのノードが、コミット・ポイント・サイトになります。

COMPATIBLE

パラメータ・タイプ	文字列
構文	COMPATIBLE = <i>release_number</i>
デフォルト値	8.1.0
パラメータ・クラス	静的
値の範囲	今回のリリースに対するデフォルトのリリース
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

COMPATIBLE は、新しいリリースを使用できるようにするとともに、以前のリリースとの下位互換性を保証します。この機能は、以前のリリースに戻す必要がある場合に効果的です。

CONTROL_FILE_RECORD_KEEP_TIME

パラメータ・タイプ	整数
デフォルト値	7 (日)
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	0 ～ 365 (日)

CONTROL_FILE_RECORD_KEEP_TIME には、制御ファイル内の再利用可能なレコードを再利用するまでの最小経過日数を指定します。再利用可能セクションに新規レコードを追加する必要があり、一番古いレコードがまだ再利用可能になってない場合は、このレコード・セクションは拡張されます。このパラメータを 0 に設定すると、再利用可能セクションは拡張されずに、レコードが必要に応じて再利用されます。

CONTROL_FILES

パラメータ・タイプ 文字列

構文 `CONTROL_FILES = filename [, filename [...]]`

注意: 制御ファイル名が、Oracle Managed Files となる場合があります。これは、`CREATE CONTROLFILE REUSE` 文を使用して制御ファイルが再作成された場合に発生します。

デフォルト値 オペレーティング・システム依存

パラメータ・クラス 静的

値の範囲 1 ～ 8 つのファイル名

Real Application Clusters 複数インスタンスには、同じ値を指定する必要がある。

すべてのデータベースには、データベースの構造（構造体の名前、作成のタイムスタンプ、データ・ファイルおよび REDO ファイルの名前や位置など）を記述する**制御ファイル**があります。CONTROL_FILES には、1 つ以上の制御ファイルの名前を指定します。複数指定する場合は、名前をカンマで区切ります。

CORE_DUMP_DEST

パラメータ・タイプ 文字列

構文 `CORE_DUMP_DEST = directory`

デフォルト値 `ORACLE_HOME/DBS`

パラメータ・クラス 動的: ALTER SYSTEM

CORE_DUMP_DEST は、主に UNIX のパラメータであるため、ご使用のプラットフォームではサポートされていない場合があります。CORE_DUMP_DEST には、コア・ファイルをダンプするディレクトリを指定します。

CPU_COUNT

パラメータ・タイプ 整数

デフォルト値 Oracle によって自動的に設定される。

パラメータ・クラス 静的

値の範囲 0 ～ 無制限

注意： ほとんどのプラットフォームでは、CPU_COUNT の値は Oracle のインスタンスに使用できる CPU の数に自動的に設定されます。CPU_COUNT の値は変更しないでください。

CPU_COUNT には、Oracle が使用できる CPU の数を指定します。シングル CPU コンピュータでは、CPU_COUNT の値は 1 です。

CREATE_BITMAP_AREA_SIZE

パラメータ・タイプ	整数
デフォルト値	8388608 (8MB)
パラメータ・クラス	静的
値の範囲	オペレーティング・システム依存

注意： インスタンスが共有サーバー・オブションで構成されていないかぎり、CREATE_BITMAP_AREA_SIZE パラメータを使用することはお薦めしません。かわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。CREATE_BITMAP_AREA_SIZE は、下位互換性を保つために残されます。

CREATE_BITMAP_AREA_SIZE はビットマップ索引を含むシステムのみに関連します。このパラメータには、ビットマップの作成に割り当てるメモリー容量（バイト単位）を指定します。デフォルト値は 8MB です。より大きい値を指定すると、その分、索引を速く作成できます。

CREATE_STORED_OUTLINES

構文：

```
CREATE_STORED_OUTLINES = {TRUE | FALSE | category_name} [NOOVERRIDE]
```

CREATE_STORED_OUTLINES パラメータは、セッション中に発行された問合せごとに、自動的にアウトラインを作成および格納するかどうかを決定します。

CREATE_STORED_OUTLINES は、初期化パラメータではありません。

- TRUE に設定した場合、システム内で後続の問合せに対するアウトラインは、自動的に作成可能になります。このアウトラインは、一意のシステム生成名を受け取り、DEFAULT カテゴリに格納されます。すでに特定の問合せに定義したアウトラインが DEFAULT カテゴリに存在する場合は、そのアウトラインが保持され、新しいアウトラインは作成されません。
- FALSE に設定した場合、システムのアウトラインは作成禁止になります。これはデフォルト値です。
- `category_name` を設定した場合、システムで作成されたアウトラインが `category_name` で指定したカテゴリに格納されることを除き、TRUE と同じ動作をします。
- NOOVERRIDE に設定した場合、パラメータが明示的に設定されたすべてのセッションの設定は上書きされません。NOOVERRIDE に設定しない場合、この設定は、すべてのセッションで有効になります。

CURSOR_SHARING

パラメータ・タイプ	文字列
構文	<code>CURSOR_SHARING = {SIMILAR EXACT FORCE}</code>
デフォルト値	EXACT
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

CURSOR_SHARING では、同じカーソルを共有する SQL 文の種類を判断します。

CURSOR_SPACE_FOR_TIME

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

CURSOR_SPACE_FOR_TIME によって、時間を節約するために、カーソルはより多くの領域を使用できるようになります。このパラメータは、共有 SQL 領域およびクライアントのプライベート SQL 領域の両方に影響します。

DB_nK_CACHE_SIZE

パラメータ・タイプ	大整数
構文	DB_[2 4 8 16 32]K_CACHE_SIZE = integer [K M G]
デフォルト値	0M（追加のブロック・サイズ・キャッシュは、デフォルトでは構成されない）
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	最小値: グラニュル・サイズ 最大値: オペレーティング・システム依存

DB_nK_CACHE_SIZE（n は、2、4、8、16、32 のいずれか）には、nK バッファのキャッシュ・サイズを指定します。DB_BLOCK_SIZE に nK 以外の値が設定されている場合にのみ、このパラメータを設定できます。たとえば、DB_BLOCK_SIZE=4096 の場合、DB_4K_CACHE_SIZE というパラメータの指定は無効です（4 KB ブロックのキャッシュ・サイズは、すでに DB_CACHE_SIZE で指定されているため）。

DB_BLOCK_BUFFERS

パラメータ・タイプ	整数
デフォルト値	次の値が導出される: 48 MB / DB_BLOCK_SIZE
パラメータ・クラス	静的
値の範囲	50 以上。上限は、オペレーティング・システム依存。
Real Application Clusters	複数インスタンスは異なる値を指定でき、必要に応じて値を変更可能。

注意: DB_CACHE_SIZE パラメータが設定されている場合、このパラメータは無視されます。かわりに、DB_CACHE_SIZE を使用することをお勧めします。また、DB_BLOCK_BUFFERS は、新しい DB_CACHE_SIZE 動的パラメータと組み合わせることはできません。同じパラメータ・ファイルでこの 2 つのパラメータを組み合わせた場合、エラーが発生します。DB_BLOCK_BUFFERS は、下位互換性を保つために残されます。

DB_BLOCK_BUFFERS には、バッファ・キャッシュ内のデータベース・バッファの数を指定します。このパラメータは、インスタンスの SGA の必要メモリー総量の決定にかかわるパラメータの 1 つです。

DB_BLOCK_CHECKING

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

DB_BLOCK_CHECKING によって、データ・ブロックに対してブロック・チェックを実行するかどうかを制御します。パラメータ値を true に設定すると、すべてのデータ・ブロックのブロック・チェックを実行します。パラメータ値を false に設定すると、ユーザーの表領域に対するブロック・チェックを実行しません。ただし、SYSTEM 表領域に対するブロック・チェックは常にオンです。

DB_BLOCK_CHECKSUM

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	true false

DB_BLOCK_CHECKSUM では、DBW*n* およびダイレクト・ローダーが、**チェックサム**（ブロック内に格納されているすべてのバイトから計算された数値）を計算し、ディスクにデータ・ブロックを書き込むときに、すべてのデータ・ブロックのキャッシュ・ヘッダーにそのチェックサムを格納するかどうかを決めます。ブロックが読み込まれると、このパラメータが true でかつブロックの最後の書込みにチェックサムが格納された場合のみ、チェックサムが検証されます。また、すべてのログ・ブロックは、現行のログに書き込まれる前に、チェックサムを与えられます。

DB_BLOCK_SIZE

パラメータ・タイプ	整数
デフォルト値	2048
パラメータ・クラス	静的
値の範囲	2048 ~ 32768。ただし、オペレーティング・システムによっては、さらに範囲が狭い場合がある。
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

注意： このパラメータは、データベース作成時に設定してください。
データベース作成後は変更しないでください。

DB_BLOCK_SIZE には、Oracle データベースのブロック・サイズ（バイト単位）を指定します。通常、値は 2048 または 4096 です。データベースを作成した時点で有効な DB_BLOCK_SIZE の値によって、ブロック・サイズが判断されます。値は、初期値に設定しておく必要があります。

DB_CACHE_ADVICE

パラメータ・タイプ 文字列

構文 DB_CACHE_ADVICE = {ON | READY | OFF}

デフォルト値 STATISTICS_LEVEL が TYPICAL または ALL に設定されている場合は ON
STATISTICS_LEVEL が BASIC に設定されている場合は OFF

パラメータ・クラス 動的:ALTER SYSTEM

DB_CACHE_ADVICE では、V\$DB_CACHE_ADVICE パフォーマンス・ビューを介して、様々なキャッシュ・サイズでの動作予測用を使用する統計収集が、使用可能または使用禁止にされます。

DB_CACHE_SIZE

パラメータ・タイプ 大整数

構文 DB_CACHE_SIZE = integer [K | M | G]

デフォルト値 48MB。最も近いグラニュル・サイズに切り上げられる。

パラメータ・クラス 動的:ALTER SYSTEM

DB_CACHE_SIZE には、プライマリ・ブロック・サイズ（DB_BLOCK_SIZE パラメータで定義済みのブロック・サイズ）を持つバッファの DEFAULT バッファ・プール・サイズを指定します。

DB_CREATE_FILE_DEST

パラメータ・タイプ	文字列
構文	DB_CREATE_FILE_DEST = <i>directory</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

DB_CREATE_FILE_DEST は、Oracle 管理データ・ファイルのデフォルトの位置を設定します。DB_CREATE_ONLINE_LOG_DEST_ *n* が指定されていない場合、この位置は、Oracle 管理制御ファイルおよびオンライン REDO ログのデフォルトとしても使用されます。

DB_CREATE_ONLINE_LOG_DEST_ *n*

パラメータ・タイプ	文字列
構文	DB_CREATE_ONLINE_LOG_DEST_ [1 2 3 4 5] = <i>directory</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

DB_CREATE_ONLINE_LOG_DEST_ *n* (*n* は、1、2、3、4、5 のいずれか) は、Oracle 管理制御ファイルおよびオンライン REDO ログのデフォルトの位置を設定します。

DB_DOMAIN

パラメータ・タイプ	文字列
構文	DB_DOMAIN = <i>domain_name</i>
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	ピリオドで区切った名前コンポーネントの有効な文字列で、最大 128 文字 (ピリオドを含む)。NULL は指定できない。
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

分散データベース・システムでは、DB_DOMAIN には、ネットワーク構造内でのデータベースの論理上の位置を指定します。このデータベースが分散システムまたはその一部の場合、このパラメータを設定します。この値は、有効な識別子で構成され、ピリオドで区切られたグローバルなデータベース名の拡張コンポーネントで構成されます。DB_DOMAIN には、ドメイン内の各データベースに一意の文字列を指定することをお勧めします。

DB_FILE_MULTIBLOCK_READ_COUNT

パラメータ・タイプ	整数
デフォルト値	8
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	オペレーティング・システム依存

DB_FILE_MULTIBLOCK_READ_COUNT は、表スキャン中に、I/O を最小化するために使用できるパラメータの 1 つです。このパラメータには、順次スキャン中に 1 回の I/O 操作で読み込まれるブロックの最大数を指定します。全表スキャンに必要な I/O の合計数は、表のサイズ、マルチブロック READ 件数などの条件、およびパラレル実行が操作に対して使用可能な状態になっているかどうかによって異なります。

DB_FILE_NAME_CONVERT

パラメータ・タイプ	文字列
構文	<pre>DB_FILE_NAME_CONVERT = [('string1' , 'string2' , 'string3' , 'string4' , ...)]</pre> それぞれの意味は、次のとおりです。 ■ string1 は、プライマリ・データベース・ファイル名のパターン。 ■ string2 は、スタンバイ・データベース・ファイル名のパターン。 ■ string3 は、プライマリ・データベース・ファイル名のパターン。 ■ string4 は、スタンバイ・データベース・ファイル名のパターン。 必要な数のプライマリ置換文字列とスタンバイ置換文字列の組合せを使用可能。一重または二重引用符を使用可能。カッコはオプション。 設定例: <pre>DB_FILE_NAME_CONVERT = (' /dbs/t1/' , ' /dbs/t1/s_' , 'dbs/t2/' , 'dbs/t2/s_')</pre>
デフォルト値	なし
パラメータ・クラス	静的

DB_FILE_NAME_CONVERT は、リカバリ用のクローン・データベースの作成に役立ちます。このパラメータは、プライマリ・データベース上の新規のデータ・ファイルのファイル名をスタンバイ・データベース上のファイル名に変換します。データ・ファイルをプライマリ・データベースに追加すると、それに対応するファイルをスタンバイ・データベースに追加する必要があります。スタンバイ・データベースを更新するときに、このパラメータは、プライマリ・データベース上のデータ・ファイル名をスタンバイ・データベース上のデータ・ファイル名に変換します。このファイルがスタンバイ・データベース上に書き込み可能な状態で存在している必要があります。そうでない場合、リカバリ・プロセスはエラーによって停止します。

DB_FILES

パラメータ・タイプ	整数
デフォルト値	200
パラメータ・クラス	静的
値の範囲	最小値: データベース内のデータ・ファイルの絶対ファイル番号の中で最も大きい値。 最大値: オペレーティング・システム依存
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

DB_FILES には、このデータベースに対してオープンできるデータベース・ファイルの最大数を指定します。最大有効値は、ADD DATAFILE 文によって追加されるファイルを含め、このデータベースに対して指定されるすべてのファイルの最大数です。この値は、オペレーティング・システムによって異なります。

DB_KEEP_CACHE_SIZE

パラメータ・タイプ	大整数
構文	DB_KEEP_CACHE_SIZE = integer [K M G]
デフォルト値	0 (KEEP キャッシュは、デフォルトでは構成されない)
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	最小値: グラニュール・サイズ 最大値: オペレーティング・システム依存

DB_KEEP_CACHE_SIZE には、KEEP バッファ・プールのサイズを指定します。KEEP バッファ・プール内のバッファのサイズは、プライマリ・ブロック・サイズ (DB_BLOCK_SIZE パラメータで定義済みのブロック・サイズ) です。

DB_NAME

パラメータ・タイプ	文字列
構文	DB_NAME = <i>database_name</i>
デフォルト値	なし
パラメータ・クラス	静的
Real Application Clusters	すべてのインスタンスにこのパラメータを設定する必要がある。そうしない場合は、SQL*Plus 文 STARTUP OPEN または SQL 文 ALTER DATABASE MOUNT に同じ値を指定する必要がある。

DB_NAME には、最大 8 文字のデータベース識別子を指定します。このパラメータを指定する場合は、CREATE DATABASE 文で指定された名前に対応する値である必要があります。DB_NAME の使用は任意ですが、一般的に、DB_NAME は CREATE DATABASE 文を発行する前に設定し、その後、その文の中で参照します。

DB_RECYCLE_CACHE_SIZE

パラメータ・タイプ	大整数
構文	DB_RECYCLE_CACHE_SIZE = <i>integer</i> [K M G]
デフォルト値	0 (RECYCLE キャッシュは、デフォルトでは構成されない)
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	最小値: グラニュル・サイズ 最大値: オペレーティング・システム依存

DB_RECYCLE_CACHE_SIZE には、RECYCLE バッファ・プールのサイズを指定します。RECYCLE プール内のバッファのサイズは、プライマリ・ブロック・サイズ (DB_BLOCK_SIZE パラメータで定義済のブロック・サイズ) です。

DB_WRITER_PROCESSES

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	静的
値の範囲	1 ~ 20

DB_WRITER_PROCESSES は、データを大量に変更するシステムに便利です。このパラメータには、インスタンスに対するデータベース・ライター・プロセス数の初期値を指定します。

DBLINK_ENCRYPT_LOGIN

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

パスワードを使用してデータベースに接続する場合、Oracle は、パスワードをデータベースに送信する前に暗号化します。DBLINK_ENCRYPT_LOGIN には、データベース・リンクを介して他の Oracle サーバーに接続する場合に、暗号化パスワードを使用するかどうかを指定します。

DBWR_IO_SLAVES

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的
値の範囲	0 以上。上限は、オペレーティング・システム依存。

DBWR_IO_SLAVES は、データベース・ライター・プロセス (DBW0) を 1 つのみ持つシステムに関連します。DBW0 プロセスで使用される I/O サーバー・プロセスの数を指定します。DBW0 プロセスおよびそのサーバー・プロセスは、常にディスクに書き込みます。デフォルト値は 0 で、I/O サーバー・プロセスは使用されません。

DG_BROKER_CONFIG_FILE*n*

パラメータ・タイプ	文字列
構文	DG_BROKER_CONFIG_FILE[1 2] = <i>filename</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	1 つのファイル名

DG_BROKER_CONFIG_FILE n (n は、1 または 2) には、Data Guard Broker 構成ファイルの名前を指定します。

DG_BROKER_START

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

DG_BROKER_START によって、Oracle は、DMON (Data Guard Broker) プロセスを開始する必要があるかどうかを判断できます。DMON は、重要度の低い Oracle のバックグラウンド・プロセスであり、このパラメータが true に設定されている場合はインスタンスが存在するかぎり存在します。

DISK_ASYNC_IO

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	静的
値の範囲	true false

DISK_ASYNC_IO は、データ・ファイル、制御ファイル、およびログ・ファイルへの I/O が非同期かどうかを制御します (Real Application Clusters プロセスでは、表スキャン中に、CPU 処理と I/O 要求をオーバーラップできるかどうかを制御します)。ご使用のプラットフォームがディスクへの非同期 I/O をサポートしている場合は、このパラメータをデフォルト値のままにしておくことをお勧めします。ただし、非同期 I/O の実装が安定してない場合は、このパラメータを false に設定することで、非同期 I/O を使用禁止にできます。プラットフォームがディスクへの非同期 I/O をサポートしていない場合、このパラメータは無効です。

DISPATCHERS

パラメータ・タイプ 文字列

構文

```
DISPATCHERS = 'dispatch_clause'

dispatch_clause::=
    (PROTOCOL = protocol) |
    (ADDRESS = address) |
    (DESCRIPTION = description )
[options_clause]

options_clause::=
    (DISPATCHERS = integer |
    SESSIONS = integer |
    CONNECTIONS = integer |
    TICKS = seconds |
    POOL = {1 | ON | YES | TRUE | BOTH |
        ({IN | OUT} = ticks) | 0 | OFF | NO | FALSE |
        ticks} |
    MULTIPLEX = {1 | ON | YES | TRUE |
        0 | OFF | NO | FALSE | BOTH | IN | OUT} |
    LISTENER = tnsname |
    SERVICE = service |
    INDEX = integer)
```

デフォルト値 なし

パラメータ・クラス 動的:ALTER SYSTEM

DISPATCHERS は、共有サーバー・アーキテクチャ内のディスパッチャ・プロセスを構成します。Oracle のパラメータ解析部は、属性を順番にかかわらず大 / 小文字を区別せずに指定できるような名前値の構文をサポートします。次に例を示します。

```
DISPATCHERS = "(PROTOCOL=TCP) (DISPATCHERS=3)"
```

今回のリリースでは、ALTER SYSTEM 文の DISPATCHERS 初期化パラメータで、SCOPE 句がサポートされていません。

現行のインスタンスの DISPATCHERS パラメータを再構成するには、SCOPE 句を指定せずに ALTER SYSTEM SET DISPATCHERS 文を発行します。この新しい構成を永続化するには、データベースを起動する前に初期化パラメータ・ファイルの DISPATCHERS パラメータを指定します。サーバー・パラメータ・ファイル（spfile）を使用してデータベースを起動している場合は、CREATE SPFILE を発行して、初期化パラメータから spfile を再作成する必要があります。

DISTRIBUTED_LOCK_TIMEOUT

パラメータ・タイプ	整数
デフォルト値	60
パラメータ・クラス	静的
値の範囲	0 ～無制限

DISTRIBUTED_LOCK_TIMEOUT には、分散トランザクションがロックされたリソースを待機する時間（秒単位）を指定します。

DML_LOCKS

パラメータ・タイプ	整数
デフォルト値	次の値が導出される : 4 × TRANSACTIONS
パラメータ・クラス	静的
値の範囲	20 ～無制限。0 を設定すると、エンキューが使用禁止になります。
Real Application Clusters	このパラメータは、すべてのインスタンスに設定する必要があり、インスタンスの値は、正の値、またはすべてが 0（ゼロ）である必要がある。

DML ロックは、DML 操作（挿入、更新、削除）を実行中の表に対して取得されるロックです。DML_LOCKS には、DML ロックの最大数（トランザクションで変更された表 1 つにつき 1 個）を指定します。この値は、すべてのユーザーが現在参照している表に対するロック総数と同じである必要があります。たとえば、3 人のユーザーが 1 つの表のデータを変更している場合は、3 つのエントリが必要になり、3 人のユーザーが 2 つの表のデータを変更している場合は、6 つのエントリが必要になります。

注意： サーバー・パラメータ・ファイル（spfile）を使用してデータベースを起動した場合にのみ、ALTER SYSTEM を使用してこのパラメータを設定できます。この場合、SCOPE = SPFILE を指定する必要があります。

DRS_START

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

注意： DG_BROKER_START パラメータが設定されている場合、このパラメータは無視されます。DG_BROKER_START を使用することをお勧めします。DRS_START は、下位互換性を保つために残されます。

DRS_START によって、Oracle は、DMON（Data Guard Broker）プロセスを開始する必要があるかどうかを判断できます。DMON は、重要度の低い Oracle のバックグラウンド・プロセスであり、このパラメータが true に設定されている場合はインスタンスが存在するかぎり存在します。

ENQUEUE_RESOURCES

パラメータ・タイプ	整数
デフォルト値	SESSIONS パラメータから導出される。
パラメータ・クラス	静的
値の範囲	10 ～無制限

ENQUEUE_RESOURCES は、ロック・マネージャによって同時にロックできるリソースの数を設定します。エンキューは、既知のリソースを複数の同時実行プロセスが様々なレベルで共有できるようにする高度なロック・メカニズムです。同時に使用されるオブジェクトは、エンキューによって保護できます。たとえば、Oracle では様々なレベルで表を共有でき、2 つのプロセスが 1 つの表を共有モードまたは共有更新モードでロックすることも可能です。

注意： サーバー・パラメータ・ファイル（spfile）を使用してデータベースを起動した場合にのみ、ALTER SYSTEM を使用してこのパラメータを設定できます。この場合、SCOPE = SPFILE を指定する必要があります。

EVENT

パラメータ・タイプ	文字列
デフォルト値	なし
パラメータ・クラス	静的

EVENT は、システムをデバッグするために使用されるパラメータです。変更する場合は、オラクル社カスタマ・サポート・センターの指示に従ってください。

FAL_CLIENT

パラメータ・タイプ	文字列
構文	<code>FAL_CLIENT = string</code>
デフォルト値	なし
パラメータ・クラス	動的:ALTER SYSTEM

FAL_CLIENT には、フェッチ・アーカイブ・ログ (FAL) ・サービスが使用する FAL クライアント名を指定します。FAL クライアント名は、FAL_SERVER パラメータで構成され、FAL クライアントの参照に使用されます。この値は Oracle Net のサービス名です。このサービス名は、FAL クライアント (スタンバイ・データベース) を指すように、FAL サーバー・システム上で適切に構成されていることを前提とします。

FAL_SERVER

パラメータ・タイプ	文字列
構文	<code>FAL_SERVER = string</code>
デフォルト値	なし
パラメータ・クラス	動的:ALTER SYSTEM

FAL_SERVER には、スタンバイ・データベースの FAL サーバーを指定します。この値は Oracle Net のサービス名です。このサービス名は、必要な FAL サーバーを指すように、スタンバイ・データベース・システムで適切に構成されていることを前提とします。

FAST_START_IO_TARGET

パラメータ・タイプ	整数
デフォルト値	キャッシュ内のすべてのバッファ
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	1000 ～キャッシュ内のすべてのバッファ。0 に設定すると、リカバリ I/O の制限は使用禁止になる。
Real Application Clusters	複数インスタンスには異なる値を指定でき、実行時に値を変更可能。

注意： FAST_START_MTTR_TARGET パラメータが設定されている場合、このパラメータは無視されます。FAST_START_MTTR_TARGET を使用することをお勧めします。FAST_START_IO_TARGET は、下位互換性を保つために残されます。

FAST_START_IO_TARGET (Oracle Enterprise Edition でのみ使用可能) には、クラッシュまたはインスタンスのリカバリ時に必要な I/O の数を指定します。

FAST_START_MTTR_TARGET

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ～ 3600 秒
Real Application Clusters	複数インスタンスには異なる値を指定でき、実行時に値を変更可能。

FAST_START_MTTR_TARGET には、データベースがシングル・インスタンスのクラッシュ・リカバリを実行するまでにかかる秒数を指定します。FAST_START_MTTR_TARGET を指定した場合、次のようになります。

- FAST_START_IO_TARGET で上書きされます。
- LOG_CHECKPOINT_INTERVAL で上書きされます。

FAST_START_PARALLEL_ROLLBACK

パラメータ・タイプ	文字列
構文	FAST_START_PARALLEL_ROLLBACK = {HIGH LOW FALSE}
デフォルト値	LOW
パラメータ・クラス	動的: ALTER SYSTEM

FAST_START_PARALLEL_ROLLBACK は、パラレル・ロールバックを実行するために存在するプロセスの最大数を判断します。このパラメータは一部またはすべてのトランザクションの実行に長時間かかるシステムで役立ちます。

FILE_MAPPING

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	true false

FILE_MAPPING によって、ファイル・マッピングが使用可能または使用禁止になります。ファイル・マッピングが使用可能な場合に、情報のマッピングを管理するために、FMON バックグラウンド・プロセスが起動します。

FILESYSTEMIO_OPTIONS

パラメータ・タイプ	文字列
構文	FILESYSTEMIO_OPTIONS = {none setall directIO asynch}
デフォルト値	なし
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

FILESYSTEMIO_OPTIONS には、ファイル・システムのファイルの I/O 操作を指定します。

FIXED_DATE

パラメータ・タイプ	文字列
構文	FIXED_DATE = YYYY-MM-DD-HH24:MI:SS（またはデフォルトの Oracle 日付書式）
デフォルト値	なし
パラメータ・クラス	動的:ALTER SYSTEM

FIXED_DATE には、SYSDATE が現在の日付のかわりに常に戻す定数の日付を設定します。このパラメータは、主にテストに役立ちます。この値は前述の書式、またはデフォルトの Oracle 日付書式（時刻なし）で設定できます。

GC_FILES_TO_LOCKS

パラメータ・タイプ	文字列
構文	GC_FILES_TO_LOCKS = '{file_list=lock_count[!blocks] [EACH] [:...]}' 引用符の中に空白を入れることはできない。
デフォルト値	なし
パラメータ・クラス	静的
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。値を変更するには、クラスタ内のすべてのインスタンスを停止し、各インスタンスの値を変更してから、各インスタンスを再起動する必要がある。

注意： このパラメータをデフォルト以外の値に設定すると、Oracle9i Real Application Clusters でキャッシュ・フュージョン処理が使用禁止になります。

GC_FILES_TO_LOCKS は、排他モードで実行中のインスタンスには影響しない、Oracle9i Real Application Clusters のパラメータです。データ・ファイルに対するリリース 9.0.1 より前のパラレル・キャッシュ管理（PCM）ロックのマップを制御します。

GLOBAL_CONTEXT_POOL_SIZE

パラメータ・タイプ	文字列
デフォルト値	1MB
パラメータ・クラス	静的
値の範囲	MB 単位の任意の整数値

GLOBAL_CONTEXT_POOL_SIZE には、グローバル・アプリケーション・コンテキストの格納および管理用に、SGA に割り当てるメモリー量を指定します。

GLOBAL_NAMES

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

GLOBAL_NAMES には、データベース・リンクを、接続するデータベースと同じ名前にする必要があるかどうかを指定します。

HASH_AREA_SIZE

パラメータ・タイプ	整数
デフォルト値	次の値が導出される: $2 \times \text{SORT\_AREA\_SIZE}$
パラメータ・クラス	動的: ALTER SESSION
値の範囲	0 以上。上限は、オペレーティング・システム依存。

注意： インスタンスが共有サーバー・オプションで構成されていない場合、HASH_AREA_SIZE パラメータを使用することはお薦めしません。かわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。HASH_AREA_SIZE は、下位互換性を保つために残されます。

`HASH_AREA_SIZE` はパラレル実行操作、および DML または DDL 文の問合せ部分に関連します。このパラメータには、ハッシュ結合で使用するメモリの最大容量（バイト）を指定します。

HASH_JOIN_ENABLED

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	動的:ALTER SESSION
値の範囲	true false

`HASH_JOIN_ENABLED` には、オブティマイザで選択される結合方法にハッシュ結合を使用するかどうかを指定します。`false` に設定すると、ハッシュ結合を結合方法として選択できません。`true` に設定すると、オブティマイザによってハッシュ結合とその他の結合とのコストが比較され、コスト効率が最適な場合、ハッシュ結合が採用されます。データ・ウェアハウス・アプリケーション用に、このパラメータを `true` に設定することをお勧めします。

HI_SHARED_MEMORY_ADDRESS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的

`HI_SHARED_MEMORY_ADDRESS` には、SGA の実行時の開始アドレスを指定します。これらのパラメータは、リンク時に SGA の開始アドレスを指定するプラットフォームでは無視されます。

HS_AUTOREGISTER

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

HS_AUTOREGISTER では、異機種間サービス (HS) エージェントの自動自己登録を使用可能または使用禁止にします。使用可能な場合、サーバーのデータ・ディクショナリに、以前の不明なエージェント・クラスまたは新しいエージェント・バージョンを記述する情報がアップロードされます。

IFILE

パラメータ・タイプ	パラメータ・ファイル
構文	IFILE = <i>parameter_file_name</i>
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	有効なパラメータ・ファイル名
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

IFILE を使用して、現行のパラメータ・ファイルの中に別のパラメータ・ファイルを組み込みます。次に例を示します。

```
IFILE = COMMON.ORA
```

INSTANCE_GROUPS

パラメータ・タイプ	文字列
構文	INSTANCE_GROUPS = <i>group_name</i> [, <i>group_name</i> ...]
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	カンマで区切られた、1 つ以上のインスタンス・グループ名
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

INSTANCE_GROUPS は、パラレル・モードでのみ使用できる Oracle9i Real Application Clusters のパラメータです。PARALLEL_INSTANCE_GROUP とともに使用することによって、インスタンスの制限数にパラレル問合せを制限できます。

INSTANCE_NAME

パラメータ・タイプ 文字列

構文 `INSTANCE_NAME = instance_id`

デフォルト値 インスタンス SID

注意：SID は、ホストでのインスタンスの共有メモリーを識別しますが、他のインスタンスからは一意に識別されません。

パラメータ・クラス 静的

値の範囲 英数字

Oracle9i Real Application Clusters 環境では、複数インスタンスを単一のデータベース・サービスと関連付けることができます。クライアントで、データベースに接続する特定のインスタンスを指定すると、Oracle の接続時ロード・バランスを上書きできます。INSTANCE_NAME には、このインスタンスの一意の名前を指定します。

INSTANCE_NUMBER

パラメータ・タイプ 整数

デフォルト値 使用可能な最小番号：インスタンス起動順序、およびその他のインスタンスの INSTANCE_NUMBER の値から導出。Oracle9i Real Application Clusters 構成でない場合は 0。

パラメータ・クラス 静的

値の範囲 1 ～データベース作成時に指定されたインスタンスの最大数

Real Application Clusters すべてのインスタンスに異なる値を設定する必要がある。

INSTANCE_NUMBER は、Oracle9i Real Application Clusters のパラメータで、パラレル・モードまたは排他モードで指定できます。記憶域パラメータ FREELIST GROUPS で作成されたデータベース・オブジェクトごとの空きリスト・グループの 1 つにインスタンスをマップする一意の番号を指定します。

JAVA_MAX_SESSIONSPACE_SIZE

パラメータ・タイプ 整数

デフォルト値 0

パラメータ・クラス 静的

値の範囲 0 ～ 4GB

Java セッション・スペースは、1 つのデータベースから他のデータベースをコールする Java を保持するメモリーです。JAVA_MAX_SESSIONSPACE_SIZE には、サーバーで Java プログラムを実行できるセッション・スペースの最大量をバイト単位で指定します。ユーザー・セッション中に Java がこの量を超えそうになると、Java Virtual Machine がメモリー不足障害で、セッションを終了します。

JAVA_POOL_SIZE

パラメータ・タイプ	大整数
構文	<code>LARGE_POOL_SIZE = integer [K M G]</code>
デフォルト値	24MB。最も近いグラニュル・サイズに切り上げられる。
パラメータ・クラス	静的
値の範囲	最小値: グラニュル・サイズ 最大値: オペレーティング・システム依存

JAVA_POOL_SIZE には、Java プールのサイズをバイト単位で指定します。この Java プールから、Java メモリー・マネージャは、ランタイム実行時のほとんどの Java を割り当てます。このメモリーには、コールの終わりで Java セッション・スペースに移行される Java オブジェクトの他に、Java メソッドおよびクラス定義のメモリー内での共有表現が含まれます。

JAVA_SOFT_SESSIONSPACE_LIMIT

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的
値の範囲	0 ～ 4GB

Java セッション・スペースは、1 つのデータベースから他のデータベースをコールする Java を保持するメモリーです。JAVA_SOFT_SESSIONSPACE_LIMIT には、セッションでの Java メモリー使用量の**一時的な制限**をバイト単位で指定します。これは、ユーザー・セッション中の Java が、あまりに多くのメモリーを使用している場合の警告手段となります。ユーザー・セッション中の Java がこのサイズを超えた場合、トレース・ファイルに警告が生成されます。

JOB_QUEUE_PROCESSES

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ~ 1000
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

JOB_QUEUE_PROCESSES には、ジョブ実行用に作成できるプロセスの最大数を指定します。インスタンスごとのジョブ・キュー・プロセスの数 (000、...999) を指定します。レプリケーションでは、データ・リフレッシュにジョブ・キューが使用されます。アドバンスト・キューイングでは、メッセージ伝播にジョブ・キューが使用されます。DBMS_JOB パッケージを使用して、ユーザー・ジョブ要求を作成できます。

LARGE_POOL_SIZE

パラメータ・タイプ	大整数
構文	LARGE_POOL_SIZE = integer [K M G]
デフォルト値	次の両方に該当する場合は 0。 ■ パラレル実行によって、プールが要求されない。 ■ DBWR_IO_SLAVES が設定されていない。 それ以外の場合は、PARALLEL_MAX_SERVERS、PARALLEL_THREADS_PER_CPU、CLUSTER_DATABASE_INSTANCES、DISPATCHERS および DBWR_IO_SLAVES の値から導出される。
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	300KB ~ 2GB (正確な最大値はオペレーティング・システム依存)

LARGE_POOL_SIZE には、ラージ・プールの割当てヒープ・サイズをバイト単位で指定します。ラージ・プールの割当てヒープは、セッション・メモリの共有サーバー・システム、メッセージ・バッファのパラレル実行およびディスク I/O バッファのバックアップ・プロセスが使用します (パラレル実行は、PARALLEL_AUTOMATIC_TUNING に TRUE が設定されている場合のみ、ラージ・プールからバッファを割り当てます)。

LICENSE_MAX_SESSIONS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0～セッション・ライセンスの数
Real Application Clusters	複数インスタンスには、異なる値を指定可能。ただし、データベースをマウントするすべてのインスタンスの合計は、そのデータベースにライセンス付与されたセッションの総数以下にする必要がある。

LICENSE_MAX_SESSIONS には、同時ユーザー・セッションの最大数を指定します。最大数に達すると、RESTRICTED SESSION 権限を持つユーザーのみがデータベースに接続できます。接続できないユーザーは、システムが最大容量に達していることを示す警告メッセージを受け取ります。

LICENSE_MAX_USERS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0～ユーザー・ライセンスの数
Real Application Clusters	複数インスタンスには同じ値を指定する必要がある。異なるインスタンスで、このパラメータに異なる値が指定されると、データベースを最初にマウントするインスタンスの値が優先される。

LICENSE_MAX_USERS には、データベースで作成できるユーザーの最大数を指定します。最大数に達すると、それ以上ユーザーを作成できなくなります。ただし、この最大数を引き上げることはできません。

LICENSE_MAX_USERS の制限事項： ユーザー数の最大値は、データベースに現在登録されているユーザー数より小さくすることができません。

参照： 10-120 ページの「[ライセンス・パラメータの変更例:](#)」を参照してください。

LICENSE_SESSIONS_WARNING

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ~ LICENSE_MAX_SESSIONS パラメータの値
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

LICENSE_SESSIONS_WARNING には、同時ユーザー・セッション数に関する警告限度を指定します。この限度に達しても、ユーザーの追加接続はできますが、Oracle は新規の接続のたびにアラート・ファイルにメッセージを書き込みます。この限度に達した後に接続した RESTRICTED SESSION 権限を持つユーザーは、システムがその容量制限に近づいていることを示す警告メッセージを受け取ります。

LOCAL_LISTENER

パラメータ・タイプ	文字列
構文	LOCAL_LISTENER = <i>network_name</i>
デフォルト値	(ADDRESS = (PROTOCOL=TCP) (HOST=) (PORT=1521))
パラメータ・クラス	動的:ALTER SYSTEM

LOCAL_LISTENER には、Oracle Net ローカル・リスナー（このインスタンスと同じマシン上で実行中のリスナー）のアドレスまたはアドレス・リストを解決するネットワーク名を指定します。アドレスまたはアドレス・リストは、TNSNAMES.ORA ファイルまたはご使用のシステム用に構成されている他のアドレス・リポジトリで指定されます。

LOCK_NAME_SPACE

パラメータ・タイプ	文字列
構文	LOCK_NAME_SPACE = <i>namespace</i>
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	最大 8 文字の英数字。特殊文字は使用できない。

LOCK_NAME_SPACE には、分散ロック・マネージャ（DLM）がロック名の生成に使用する名前空間を指定します。スタンバイまたはクローン・データベースが、同じクラスタ上で、プライマリ・データベースと同じデータベース名を持つ場合、このパラメータの設定を検討してください。

LOCK_SGA

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

LOCK_SGA は、SGA 全体を物理メモリーの中にロックします。通常は、SGA を実（物理）メモリーにロックすることをお薦めします。特に、仮想メモリーの使用が、SGA の一部をディスク領域に格納してしまう可能性がある場合はそうしてください。このパラメータをサポートしていないプラットフォームでは、このパラメータは無視されます。

LOG_ARCHIVE_DEST

パラメータ・タイプ	文字列
構文	LOG_ARCHIVE_DEST = <i>filespec</i>
デフォルト値	NULL
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	有効なパス名またはデバイス名（RAW パーティションを除く）
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

注意： Enterprise Edition ユーザーを使用している場合、LOG_ARCHIVE_DEST_n パラメータが設定されていると、このパラメータは無視されます。Oracle Enterprise Edition がインストールされていない場合、またはインストールされているが LOG_ARCHIVE_DEST_n パラメータを指定していない場合は、このパラメータは有効です。

LOG_ARCHIVE_DEST は、データベースを ARCHIVELOG モードで起動している場合またはアーカイブ REDO ログからデータベースをリカバリしている場合のみ適用できます。LOG_ARCHIVE_DEST は LOG_ARCHIVE_DEST_n パラメータと非互換であり、また LOG_ARCHIVE_DEST_n パラメータに NULL 文字列以外の値がある場合、NULL 文字列('') または ('') で定義されている必要があります。REDO ログ・ファイルをアーカイブするとき、ディスク・ファイルまたはテープ・デバイスのデフォルト位置およびルートを指定するには、テキスト文字列を使用してください（テープへのアーカイブは、すべてのオペレーティング・システムでサポートされているわけではありません）。この値には、RAW パーティションは指定できません。

LOG_ARCHIVE_DEST_n

パラメータ・タイプ 文字列

構文

```
LOG_ARCHIVE_DEST_[1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10] =  
  
    { null_string } |  
    { LOCATION=path_name | SERVICE=service_name }  
    [ { MANDATORY | OPTIONAL } ]  
    [ REOPEN[=seconds] | NOREOPEN ]  
    [ DELAY[=minutes] | NODELAY ]  
    [ REGISTER[=template] | NOREGISTER ]  
    [ TEMPLATE=template | NOTEMPLATE ]  
    [ ALTERNATE=destination | NOALTERNATE ]  
    [ DEPENDENCY=destination | NODEPENDENCY ]  
    [ MAX_FAILURE=count | NOMAX_FAILURE ]  
    [ QUOTA_SIZE=blocks | NOQUOTA_SIZE ]  
    [ QUOTA_USED=blocks | NOQUOTA_USED ]  
    [ ARCH | LGWR ]  
    [ SYNC[=PARALLEL|NOPARALLEL] | ASYNC[=blocks] ]  
    [ AFFIRM | NOAFFIRM ]  
    [ NET_TIMEOUT=seconds | NONET_TIMEOUT ]  
  
    }
```

デフォルト値 なし

パラメータ・クラス 動的: ALTER SESSION、ALTER SYSTEM

注意： Oracle Enterprise Edition がインストールされている場合のみ、このパラメータは有効です。Oracle Enterprise Edition がインストールされている場合は、LOG_ARCHIVE_DEST を継続して使用できます。ただし、互換性がないので、LOG_ARCHIVE_DEST_n および LOG_ARCHIVE_DEST の両方を使用することはできません。

LOG_ARCHIVE_DEST_n パラメータ (n= 1、2、3、... 10) は、アーカイブ・ログの宛先を 10 まで定義します。パラメータの整数の接尾辞は、V\$ARCHIVE_DEST 動的パフォーマンス・ビューが表示する **ハンドル** として定義されます。

LOG_ARCHIVE_DEST_STATE_ *n*

パラメータ・タイプ 文字列

構文 LOG_ARCHIVE_DEST_STATE_ *n* = {alternate | reset |
defer | enable}

デフォルト値 enable

パラメータ・クラス 動的: ALTER SESSION、ALTER SYSTEM

LOG_ARCHIVE_DEST_STATE_ *n* パラメータ (*n* = 1、2、3、... 10) には、対応する宛先の可用性状態を指定します。パラメータの接尾辞 (1 ~ 10) は、10 個の対応する LOG_ARCHIVE_DEST_ *n* の宛先パラメータのうちの 1 つを指定します。

LOG_ARCHIVE_DUPLEX_DEST

パラメータ・タイプ 文字列

構文 LOG_ARCHIVE_DUPLEX_DEST = *filespec*

デフォルト値 なし

パラメータ・クラス 動的: ALTER SYSTEM

値の範囲 NULL 文字列、有効なパス名またはデバイス名 (RAW パーティションを除く)

注意: Oracle Enterprise Edition を使用している場合、LOG_ARCHIVE_DEST_ *n* パラメータが設定されていると、このパラメータは無視されます。Oracle Enterprise Edition がインストールされていない場合、またはインストールされているが LOG_ARCHIVE_DEST_ *n* パラメータを指定していない場合は、このパラメータは有効です。

LOG_ARCHIVE_DUPLEX_DEST は、初期化パラメータ LOG_ARCHIVE_DEST に似ています。このパラメータには、2 番目のアーカイブ宛先であるを指定します。複数アーカイブ宛先この多重アーカイブ宛先は、must-succeed モードまたは best-effort アーカイブ宛先のいずれかです。どちらになるかは、LOG_ARCHIVE_MIN_SUCCEED_DEST パラメータで指定する、成功する必要のあるアーカイブ宛先の数によって決まります。

LOG_ARCHIVE_FORMAT

パラメータ・タイプ	文字列
構文	LOG_ARCHIVE_FORMAT = <i>filename</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	静的
値の範囲	有効なファイル名を解決する文字列
Real Application Clusters	複数インスタンスには異なる値を指定できるが、同じ値を推奨する。

LOG_ARCHIVE_FORMAT は、REDO ログを ARCHIVELOG モードで使用している場合のみ適用できます。REDO ログ・ファイルをアーカイブするときにデフォルトのファイル名書式を指定するには、テキスト文字列および変数を使用してください。この書式から生成された文字列は、LOG_ARCHIVE_DEST パラメータで指定した文字列に追加されます。

LOG_ARCHIVE_MAX_PROCESSES

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	1 ～ 10 の任意の整数

LOG_ARCHIVE_MAX_PROCESSES には、最初に起動されるアーカイバ・バックグラウンド・プロセス（ARC0 から ARC9）の数を指定します。

LOG_ARCHIVE_MIN_SUCCEED_DEST

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	LOG_ARCHIVE_DEST_n を使用している場合、1 ～ 10 LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST を使用している場合、1 または 2

LOG_ARCHIVE_MIN_SUCCEED_DEST には、オンライン・ログ・ファイルの再利用を可能にするために必要なアーカイブ先の最小数を定義します。

LOG_ARCHIVE_START

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

LOG_ARCHIVE_START は、REDO ログを ARCHIVELOG モードで使用しているときのみ適用できます。このパラメータには、インスタンスの起動時にアーカイブが自動、手動のどちらであるかを指定します。

LOG_ARCHIVE_TRACE

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0、1、2、4、8、16、32、64、128
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

LOG_ARCHIVE_TRACE は、アーカイブ・ログ・プロセスで生成される出力結果を制御します。

LOG_BUFFER

パラメータ・タイプ	整数
デフォルト値	512KB、または 128KB × CPU_COUNT のいずれか大きい方
パラメータ・クラス	静的
値の範囲	オペレーティング・システム依存

LOG_BUFFER には、REDO エントリを REDO ログ・ファイルにバッファリングするときに使用されるメモリー容量を、バイト単位で指定します。REDO ログ・エントリには、データベース・ブロック・バッファに対する変更の記録が含まれています。LGWR プロセスは、ログ・バッファから REDO ログ・ファイルに REDO ログ・エントリを書き込みます。

LOG_CHECKPOINT_INTERVAL

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	無制限
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

LOG_CHECKPOINT_INTERVAL には、チェックポイントの頻度、つまり増分チェックポイントと REDO ログに書き込まれた最終ブロック間に存在する可能性のある REDO ログ・ファイルのブロックの数を指定します。この数は、データベース・ブロックではなく、オペレーティング・システムの物理ブロックの数です。

LOG_CHECKPOINT_TIMEOUT

パラメータ・タイプ	整数
デフォルト値	1800
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ～無制限
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

LOG_CHECKPOINT_TIMEOUT には、REDO ログへの最後の書込みが発生した場所（**ログの最後尾**とも呼ばれる）での、増分チェックポイント以降の時間（秒単位）を指定します。また、このパラメータは、すべてのバッファが *integer* 秒より長い間、使用済で残らないことを示します。

LOG_CHECKPOINTS_TO_ALERT

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

LOG_CHECKPOINTS_TO_ALERT によって、チェックポイントをアラート・ファイルに記録できます。これによって、指定した頻度でチェックポイントが発生しているかどうかを判断できます。

LOG_FILE_NAME_CONVERT

パラメータ・タイプ 文字列

構文

```
LOG_FILE_NAME_CONVERT = ([('string1' , 'string2' ,  
'string3' , 'string4' , ...[]])
```

それぞれの意味は、次のとおりです。

- string1 は、プライマリ・データベース・ファイル名のパターン。
- string2 は、スタンバイ・データベース・ファイル名のパターン。
- string3 は、プライマリ・データベース・ファイル名のパターン。
- string4 は、スタンバイ・データベース・ファイル名のパターン。

必要な数のプライマリ置換文字列とスタンバイ置換文字列の組合せを使用可能。一重または二重引用符を使用可能。カッコはオプション。

設定例：

```
LOG_FILE_NAME_CONVERT=('/dbs/t1/', '/dbs/t1/s_', 'db  
s/t2/ ', 'dbs/t2/s_')
```

デフォルト値 なし

パラメータ・クラス 静的

値の範囲 文字列

LOG_FILE_NAME_CONVERTによって、プライマリ・データベース上の新規のログ・ファイルのファイル名がスタンバイ・データベース上のログ・ファイルのファイル名に変換されます。ログ・ファイルをプライマリ・データベースに追加すると、それに対応するファイルをスタンバイ・データベースに追加する必要があります。

LOG_PARALLELISM

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	静的
値の範囲	1 ～ 255

LOG_PARALLELISM には、Oracle 内部での REDO 割当ての並行性レベルを指定します。

LOGMNR_MAX_PERSISTENT_SESSIONS

パラメータ・タイプ	整数
デフォルト値	1
パラメータ・クラス	静的
値の範囲	1 ～ LICENSE_MAX_SESSIONS

LOGMNR_MAX_PERSISTENT_SESSIONS には、LogMiner 永続マイニング・セッション（ディスクにバックアップが取られた LogMiner セッション）の最大数を指定します。すべてのセッションが、スタンドアロン・インスタンスによって生成された REDO ログをマイニングしているときに、これらのセッションは同時にアクティブの状態になります。これによって、LogMiner が使用できるように、(2 × LOGMNR_MAX_PERSISTENT_SESSIONS) MB の連続するメモリーが SGA に事前に割り当てられます。

MAX_COMMIT_PROPAGATION_DELAY

パラメータ・タイプ	整数
デフォルト値	700
パラメータ・クラス	静的
値の範囲	0 ～ 90000
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

注意： このパラメータを変更するのは、問合せの実行時に、データベースの最新の状態を見る必要がある場合のみにしてください。

`MAX_COMMIT_PROPAGATION_DELAY` は、Oracle9i Real Application Clusters のパラメータです。この初期化パラメータは、クラスタ・データベースでの特定の状況下以外は変更しないでください。

MAX_DISPATCHERS

パラメータ・タイプ	整数
デフォルト値	5
パラメータ・クラス	静的
値の範囲	5 または構成されるディスパッチャの数のいずれか大きい方

`MAX_DISPATCHERS` には、同時に実行できるディスパッチャ・プロセスの最大数を指定します。デフォルト値は、システムにディスパッチャが構成された場合のみ適用されます。

MAX_DUMP_FILE_SIZE

パラメータ・タイプ	文字列
構文	<code>MAX_DUMP_FILE_SIZE = {integer [K M] UNLIMITED}</code>
デフォルト値	UNLIMITED
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	0 ～ 無制限または UNLIMITED

`MAX_DUMP_FILE_SIZE` には、トレース・ファイル（アラート・ファイルを除く）の最大サイズを指定します。トレース・ファイルが使用する領域が大きくなりすぎるおそれがある場合は、この値を変更してください。

MAX_ENABLED_ROLES

パラメータ・タイプ	整数
デフォルト値	20
パラメータ・クラス	静的
値の範囲	0 ～ 148

`MAX_ENABLED_ROLES` には、他のロールに含まれるロールを含め、ユーザーが使用可能にできるデータベース・ロールの最大数を指定します。

MAX_ROLLBACK_SEGMENTS

パラメータ・タイプ	整数
デフォルト値	MAX(30, TRANSACTIONS/ TRANSACTIONS_PER_ROLLBACK_SEGMENT)
パラメータ・クラス	静的
値の範囲	2 ～ 65535

MAX_ROLLBACK_SEGMENTS には、SGA 内のロールバック・セグメント・キャッシュの最大サイズを指定します。指定する数字は、1 つのインスタンスが同時にオンライン状態 (ONLINE の状態) を維持できるロールバック・セグメントの最大サイズです。

MTS パラメータ

10-107 ページの「[共有サーバーのパラメータ](#)」を参照してください。

MAX_SHARED_SERVERS

パラメータ・タイプ	整数
デフォルト値	SHARED_SERVERS から導出される (20 または 2 × SHARED_SERVERS)
パラメータ・クラス	静的
値の範囲	オペレーティング・システム依存

MAX_SHARED_SERVERS には、同時に実行できる共有サーバー・プロセスの最大数を指定します。ご使用のシステムで人為的なデッドロックが頻繁に発生する場合、MAX_SHARED_SERVERS の値を増やす必要があります。

NLS_CALENDAR

パラメータ・タイプ	文字列
構文	NLS_CALENDAR = "calendar_system"
デフォルト値	なし
パラメータ・クラス	動的: ALTER SESSION
値の範囲	有効な暦の書式名

NLS_CALENDAR には、Oracle が使用する暦法を指定します。このパラメータは、次の値のうち 1 つをとることができます。

- Arabic Hijrah（イスラム紀元）
- English Hijrah（英語版イスラム紀元）
- Gregorian（グレゴリオ暦）
- Japanese Imperial（日本の元号制）
- Persian（ペルシャ暦）
- ROC Official（台湾の暦）
- Thai Buddha（タイ仏教暦）

NLS_COMP

パラメータ・タイプ 文字列

構文 NLS_COMP = {binary | ansi}

デフォルト値 binary

パラメータ・クラス 動的:ALTER SESSION

通常、WHERE 句での比較と PL/SQL ブロックでの比較は、NLSSORT ファンクションを指定しないかぎり、バイナリです。NLS_COMP を ANSI に設定すると、WHERE 句および PL/SQL ブロックでの比較では、NLS_SORT パラメータで指定された言語ソートを使用する必要があります。言語ソートする列で索引を定義する必要があります。

NLS_CURRENCY

パラメータ・タイプ 文字列

構文 NLS_CURRENCY = *currency_symbol*

デフォルト値 NLS_TERRITORY から導出される。

パラメータ・クラス 動的:ALTER SESSION

値の範囲 最大値は 10 バイトまでの有効な文字列（NULL を含まない）

NLS_CURRENCY には、L 数値書式要素について各国通貨記号として使用する文字列を指定します。このパラメータのデフォルト値は、NLS_TERRITORY によって決まります。

NLS_DATE_FORMAT

パラメータ・タイプ	文字列
構文	NLS_DATE_FORMAT = <i>format</i>
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	固定長を超えない、有効な日付書式マスク

NLS_DATE_FORMAT には、TO_CHAR ファンクションおよび TO_DATE ファンクションで使用するデフォルトの日付書式を指定します。このパラメータのデフォルト値は、NLS_TERRITORY によって決まります。

NLS_DATE_LANGUAGE

パラメータ・タイプ	文字列
構文	NLS_DATE_LANGUAGE = <i>language</i>
デフォルト値	NLS_LANGUAGE から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	有効な NLS_LANGUAGE の値

NLS_DATE_LANGUAGE には、TO_DATE ファンクションおよび TO_CHAR ファンクションによって戻される曜日名、月名および日付の略称 (a.m.、p.m.、AD、BC) に使用する言語を指定します。

NLS_DUAL_CURRENCY

パラメータ・タイプ	文字列
構文	NLS_DUAL_CURRENCY = <i>currency_symbol</i>
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	最大 10 文字までの有効な書式名

NLS_DUAL_CURRENCY には、その地域の第二通貨記号 (Euro など) を指定します。デフォルトは、現在の言語環境地域での第二通貨記号です。

NLS_ISO_CURRENCY

パラメータ・タイプ	文字列
構文	NLS_ISO_CURRENCY = <i>territory</i>
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	有効な NLS_TERRITORY の値

NLS_ISO_CURRENCY には、C 数値書式要素について国際通貨記号として使用する文字列を指定します。

NLS_LANGUAGE

パラメータ・タイプ	文字列
構文	NLS_LANGUAGE = <i>language</i>
デフォルト値	オペレーティング・システム依存。環境変数 NLS_LANG から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	有効な言語名

NLS_LANGUAGE には、データベースのデフォルト言語を指定します。この言語は、メッセージ、曜日名、月名、および AD、BC、a.m.、p.m. に対する記号に使用されます。また、デフォルトのソート・メカニズムにも使用されます。このパラメータによって、パラメータ NLS_DATE_LANGUAGE および NLS_SORT のデフォルト値が決まります。

NLS_LENGTH_SEMANTICS

パラメータ・タイプ	文字列
構文	NLS_LENGTH_SEMANTICS = <i>string</i> 例: NLS_LENGTH_SEMANTICS = 'CHAR'
デフォルト値	BYTE
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	BYTE CHAR

NLS_LENGTH_SEMANTICS によって、バイト長または文字長のセマンティクスを使用して、CHAR 列および VARCHAR2 列を作成できます。既存の列には影響しません。

NLS_NCHAR_CONV_EXCP

パラメータ・タイプ	文字列
構文	NLS_NCHAR_CONV_EXCP = {TRUE FALSE}
デフォルト値	FALSE
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

NLS_NCHAR_CONV_EXCP は、キャラクタ・タイプの暗黙的または明示的な変換中に発生するデータ損害によって、エラーが報告されるかどうかを決定します。

NLS_NUMERIC_CHARACTERS

パラメータ・タイプ	文字列
構文	NLS_NUMERIC_CHARACTERS = "decimal_character_group_separator"
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的: ALTER SESSION

NLS_NUMERIC_CHARACTERS には、桁区切りおよび小数点として使用する文字を指定します。このパラメータは、NLS_TERRITORY で暗黙的に定義された文字を上書きします。桁区切りは、整数グループ（千、100 万、10 億など）を区切ります。小数点は、数値の整数部分と小数部分を区切ります。

NLS_SORT

パラメータ・タイプ	文字列
構文	NLS_SORT = {BINARY linguistic_definition}
デフォルト値	NLS_LANGUAGE から導出される。
パラメータ・クラス	動的: ALTER SESSION
値の範囲	BINARY または有効な言語定義名

NLS_SORT には、ORDER BY 問合せの照合順番を指定します。

NLS_TERRITORY

パラメータ・タイプ	文字列
構文	NLS_TERRITORY = <i>territory</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	動的:ALTER SESSION
値の範囲	有効な地域名

NLS_TERRITORY には、日と週の順序付けについて地域別規則に従う場合のその地域の名前を指定します。

NLS_TIMESTAMP_FORMAT

パラメータ・タイプ	文字列
構文	NLS_TIMESTAMP_FORMAT = " <i>format</i> "
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的:ALTER SESSION
値の範囲	有効な日時書式マスク

NLS_TIMESTAMP_FORMAT には、TO_CHAR ファンクションおよび TO_TIMESTAMP ファンクションで使用するデフォルトのタイムスタンプ書式を定義します。

NLS_TIMESTAMP_TZ_FORMAT

パラメータ・タイプ	文字列
構文	NLS_TIMESTAMP_TZ_FORMAT = " <i>format</i> "
デフォルト値	NLS_TERRITORY から導出される。
パラメータ・クラス	動的:ALTER SESSION
値の範囲	有効な日時書式マスク

NLS_TIMESTAMP_TZ_FORMAT には、TO_CHAR ファンクションおよび TO_TIMESTAMP_TZ ファンクションで使用するデフォルトのタイムスタンプを定義します。このタイムスタンプは、タイムゾーン書式を使用します。

07_DICTIONARY_ACCESSIBILITY

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

07_DICTIONARY_ACCESSIBILITY は、Oracle7 から Oracle Security Server への移行に使用されます。このパラメータは、SYSTEM 権限の制限を制御します。このパラメータに true を設定すると、SYSTEM 権限による SYS スキーマ内のオブジェクトへのアクセスが許可されます (Oracle7 の動作)。false に設定すると、すべてのスキーマ内のオブジェクトへのアクセスが許可される SYSTEM 権限では、SYS スキーマ内のオブジェクトへアクセスできなくなります。

OBJECT_CACHE_MAX_SIZE_PERCENT

パラメータ・タイプ	整数
デフォルト値	10
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM ... DEFERRED
値の範囲	0 以上。上限は、オペレーティング・システム依存の最大値。

オブジェクト・キャッシュは、クライアント上のメモリー・ブロックで、アプリケーションがすべてのオブジェクトを格納でき、サーバーへラウンド・トリップせずにオブジェクト間でナビゲートできるようにします。OBJECT_CACHE_MAX_SIZE_PERCENT には、セッション・オブジェクト・キャッシュが、最適なキャッシュ・サイズを、最大で何パーセント超過できるかを指定します。最大サイズは、最適サイズに、このパーセントと最適サイズの積を足したサイズです。キャッシュ・サイズがこの最大サイズを超えると、システムは、キャッシュを最適サイズに縮小しようとします。

OBJECT_CACHE_OPTIMAL_SIZE

パラメータ・タイプ	整数
デフォルト値	102400 (100KB)
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM ... DEFERRED
値の範囲	10KB 以上。上限は、オペレーティング・システム依存の最大値。

オブジェクト・キャッシュは、クライアント上のメモリー・ブロックで、アプリケーションがすべてのオブジェクトを格納でき、サーバーヘラウンド・トリップせずにオブジェクト間でナビゲートできるようにします。OBJECT_CACHE_OPTIMAL_SIZE には、キャッシュ・サイズが最大サイズを超えた場合に、セッション・オブジェクト・キャッシュが縮小されるサイズ（バイト単位）を指定します。

OLAP_PAGE_POOL_SIZE

パラメータ・タイプ	整数
デフォルト値	32MB
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM ... DEFERRED
値の範囲	4MB ～ 2GB

OLAP_PAGE_POOL_SIZE には、OLAP プールのサイズ（バイト単位）を指定します。

OPEN_CURSORS

パラメータ・タイプ	整数
デフォルト値	50
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	1 ～ 4294967295 (4GB-1)

OPEN_CURSORS には、1 つのセッションで同時にオープンできるカーソル（プライベート SQL 領域へのハンドル）の最大数を指定します。このパラメータを使用して、1 つのセッションでカーソルをオープンしすぎないようにできます。このパラメータには、PL/SQL カーソル・キャッシュのサイズも制限します。PL/SQL カーソル・キャッシュは、ユーザーによる文の解析が繰り返されることを防ぐため、PL/SQL が使用します。

OPEN_LINKS

パラメータ・タイプ	整数
デフォルト値	4
パラメータ・クラス	静的
値の範囲	0 ～ 255

OPEN_LINKS には、1 つのセッションでのリモート・データベースに対する同時オープン接続の最大数を指定します。これらの接続には、外部プロシージャおよびカートリッジの他、データベース・リンクが含まれます。これらは、それぞれ個別のプロセスを使用します。

OPEN_LINKS_PER_INSTANCE

パラメータ・タイプ	整数
デフォルト値	4
パラメータ・クラス	静的
値の範囲	0 ～ 4294967295 (4GB-1)
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

OPEN_LINKS_PER_INSTANCE には、データベース・インスタンスごとのグローバルに移行可能なオープン接続の最大数を指定します。XA トランザクションでは、移行可能なオープン接続が使用されるので、トランザクションがコミットされると、接続がキャッシュされます。この接続を別のトランザクションでも使用できます。ただし、その接続を作成したユーザーがそのトランザクションを所有している場合に限定されます。

OPTIMIZER_DYNAMIC_SAMPLING

パラメータ・タイプ	整数
デフォルト値	OPTIMIZER_FEATURES_ENABLE が 9.2.0 以上に設定されている場合は 1 OPTIMIZER_FEATURES_ENABLE が 9.0.1 以下に設定されている場合は 0
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	0 ～ 10

OPTIMIZER_DYNAMIC_SAMPLING は、オブティマイザが実行する動的サンプリングのレベルを制御します。

OPTIMIZER_FEATURES_ENABLE

パラメータ・タイプ	文字列
構文	OPTIMIZER_FEATURES_ENABLE = {8.0.0 8.0.3 8.0.4 8.0.5 8.0.6 8.0.7 8.1.0 8.1.3 8.1.4 8.1.5 8.1.6 8.1.7 9.0.0 9.0.1 9.2.0}
デフォルト値	9.2.0
パラメータ・クラス	静的

OPTIMIZER_FEATURES_ENABLE は、Oracle のリリース番号に基づいて一連のオブティマイザ機能を使用可能にするためのアンブレラ・パラメータとして動作します。

OPTIMIZER_INDEX_CACHING

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SESSION
値の範囲	0 ～ 100

OPTIMIZER_INDEX_CACHING を設定すると、ネストしたループ結合および IN リスト・イテレータを支援するように、コストベースの最適化動作を調整できます。

OPTIMIZER_INDEX_COST_ADJ

パラメータ・タイプ	整数
デフォルト値	100
パラメータ・クラス	動的:ALTER SESSION
値の範囲	1 ～ 10000

OPTIMIZER_INDEX_COST_ADJ を指定すると、オブティマイザがアクセス・パスを決定するときに、索引の使用を選択する傾向の強さを調整できます。つまり、オブティマイザが全表スキャンを経由して索引アクセス・パスを選択するようになります。

OPTIMIZER_MAX_PERMUTATIONS

パラメータ・タイプ	整数
デフォルト値	OPTIMIZER_FEATURES_ENABLE が 9.0.0 以上に設定されている場合は 2000 OPTIMIZER_FEATURES_ENABLE が 8.1.7 以下に設定されている場合は 80000
パラメータ・クラス	動的:ALTER SESSION
値の範囲	4 ～ 80000

OPTIMIZER_MAX_PERMUTATIONS は、オブティマイザが問合せで結合を考慮する表の並べ替えの数を制限します。この制限により、問合せに必要な解析時間を許容できる制限内にすることができます。ただし、この方法には、オブティマイザが別のよい方法を見落とすというデメリットがあります。

OPTIMIZER_MODE

パラメータ・タイプ	文字列
構文	OPTIMIZER_MODE = {first_rows_[1 10 100 1000] first_rows all_rows choose rule}
デフォルト値	choose
パラメータ・クラス	動的:ALTER SESSION

OPTIMIZER_MODE は、インスタンスの最適化方法を選択するためのデフォルトの動作を確立します。

ORACLE_TRACE_COLLECTION_NAME

パラメータ・タイプ	文字列
構文	ORACLE_TRACE_COLLECTION_NAME = <i>collection_name</i>
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	最大 16 文字までの有効なコレクション識別名（ファイル名が 8 文字までのプラットフォームを除く）

コレクションは、インストール製品の実行中に発生したイベントを収集したデータです。ORACLE_TRACE_COLLECTION_NAME には、このインスタンスに対する Oracle Trace のコレクション識別名を指定します。このパラメータは、出力ファイル名（コレクション定義ファイル .cdf およびデータ・コレクション・ファイル .dat）でも使用されます。ORACLE_TRACE_ENABLE が true に設定されている場合、この値に設定した NULL 以外の文字列が、この値に再度 NULL を設定するまで、デフォルトの Oracle Trace のコレクション機能を起動します。

ORACLE_TRACE_COLLECTION_PATH

パラメータ・タイプ	文字列
構文	ORACLE_TRACE_COLLECTION_PATH = <i>pathname</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	静的
値の範囲	ディレクトリのフルパス名

ORACLE_TRACE_COLLECTION_PATH には、Oracle Trace のコレクション定義ファイル (.cdf) およびデータ・コレクション・ファイル (.dat) を格納するディレクトリ・パス名を指定します。デフォルトの場合、Oracle Trace の .cdf および .dat ファイルは、ORACLE_HOME/otrace/admin/cdf に格納されます。

ORACLE_TRACE_COLLECTION_SIZE

パラメータ・タイプ	整数
デフォルト値	5242880
パラメータ・クラス	静的
値の範囲	0 ～ 4294967295

ORACLE_TRACE_COLLECTION_SIZE には、Oracle Trace コレクション・ファイルの最大サイズ（バイト単位）を指定します。このコレクション・ファイルが最大サイズに到達すると、収集は禁止されます。値 0 は、ファイルにサイズ制限がないことを示します。

ORACLE_TRACE_ENABLE

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

サーバーの Oracle Trace のコレクション機能を使用可能にするには、ORACLE_TRACE_ENABLE を true に設定します。この設定のみでは Oracle Trace のコレクション機能は起動されませんが、そのサーバーで Oracle Trace を使用できるようになります。

ORACLE_TRACE_FACILITY_NAME

パラメータ・タイプ	文字列
構文	ORACLE_TRACE_FACILITY_NAME = {ORACLED ORACLEE ORACLESM ORACLEC}
デフォルト値	ORACLED
パラメータ・クラス	静的

ORACLE_TRACE_FACILITY_NAME には、Oracle Trace が収集するイベント・セットを指定します。このパラメータ値の後に .fdf 拡張子が付いて、Oracle Trace 製品定義ファイルの名前になります。このファイルは、ORACLE_TRACE_FACILITY_PATH パラメータで指定されたディレクトリに格納する必要があります。製品定義ファイルには、Oracle Trace データ・コレクション API を使用する製品のために収集されるすべてのイベントおよびデータ項目の定義情報が格納されています。

ORACLE_TRACE_FACILITY_PATH

パラメータ・タイプ	文字列
構文	ORACLE_TRACE_FACILITY_PATH = <i>pathname</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	静的
値の範囲	ディレクトリのフルパス名

ORACLE_TRACE_FACILITY_PATH には、Oracle Trace の機能定義ファイルが格納されるディレクトリ・パス名を指定します。Solaris の場合、デフォルトのパス名は、ORACLE_HOME/otrace/admin/fdf/ です。NT の場合、デフォルトのパス名は、%OTRACE80%\ADMIN\FDF\ です。

OS_AUTHENT_PREFIX

パラメータ・タイプ	文字列
構文	OS_AUTHENT_PREFIX = <i>authentication_prefix</i>
デフォルト値	OPS\$
パラメータ・クラス	静的

OS_AUTHENT_PREFIX には、サーバーに接続しようとするユーザーの認証に使用される接頭辞を指定します。このパラメータの値は、各ユーザーのオペレーティング・システム・アカウント名およびパスワードの先頭に連結されます。接続が要求されると、接頭辞の付いたユーザー名が、データベース内の Oracle ユーザー名と比較されます。

OS_ROLES

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

OS_ROLES は、Oracle またはオペレーティング・システムのどちらが、各ユーザー名のロールを識別および管理するかを判断します。

PARALLEL_ADAPTIVE_MULTI_USER

パラメータ・タイプ	ブール
デフォルト値	PARALLEL_AUTOMATIC_TUNING の値から導出される。
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

PARALLEL_ADAPTIVE_MULTI_USER を true に設定すると、パラレル実行を使用するマルチユーザー環境での、パフォーマンス向上を目的とした適応アルゴリズムが使用可能になります。問合せ起動時のシステム・ロードに基づき、アルゴリズムでは、要求された並列度が自動的に減少します。有効な並列度は、デフォルトの並列度、または表またはヒントから得られる並列度を減少要因で割った値に基づきます。

PARALLEL_AUTOMATIC_TUNING

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

注意： このパラメータは、Oracle9i Real Application Clusters 環境および排他モードでのパラレル実行に適用されます。

PARALLEL_AUTOMATIC_TUNING を true に設定した場合、パラレル実行を制御するパラメータのデフォルト値が判断されます。このパラメータの設定に加えて、システムのターゲット表に PARALLEL 句を指定する必要があります。その後、すべての連続するパラレル操作が自動的にチューニングされます。

PARALLEL_EXECUTION_MESSAGE_SIZE

パラメータ・タイプ	整数
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	静的
値の範囲	2148 ～ 65535 (64 KB-1)
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

PARALLEL_EXECUTION_MESSAGE_SIZE には、パラレル実行（以前は、パラレル問合せ、PDML、パラレル・リカバリ、レプリケーションに適用）用のメッセージのサイズを指定します。

PARALLEL_INSTANCE_GROUP

パラメータ・タイプ	文字列
構文	PARALLEL_INSTANCE_GROUP = <i>group_name</i>
デフォルト値	現在アクティブであるすべてのインスタンスで構成されるグループ
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	すべてのアクティブ・インスタンスの INSTANCE_GROUPS パラメータに指定されたすべてのグループ名
Real Application Clusters	異なるインスタンスには異なる値を指定可能

PARALLEL_INSTANCE_GROUP は、Oracle9i Real Application Clusters のパラメータで、パラレル・モードでのみ使用できます。INSTANCE_GROUP パラメータとともに使用すると、インスタンスの制限数にパラレル問合せを制限できます。

PARALLEL_MAX_SERVERS

パラメータ・タイプ	整数
デフォルト値	CPU_COUNT、PARALLEL_AUTOMATIC_TUNING および PARALLEL_ADAPTIVE_MULTI_USER の値から導出される。
パラメータ・クラス	静的
値の範囲	0 ～ 3599
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

注意： このパラメータは、Oracle9i Real Application Clusters 環境および排他モードでのパラレル実行に適用されます。

PARALLEL_MAX_SERVERS には、インスタンスに関するパラレル実行プロセスおよびパラレル・リカバリ・プロセスの最大数を指定します。増加が要求されると、Oracle は、プロセス数を、インスタンスの始動時に作成された数からこの指定された数以内の範囲で増やします。

PARALLEL_MIN_PERCENT

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的: ALTER SESSION
値の範囲	0 ～ 100
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

PARALLEL_MIN_PERCENT は、PARALLEL_MAX_SERVERS および PARALLEL_MIN_SERVERS と組み合わせて処理されます。パラレル実行に必要な、(PARALLEL_MAX_SERVERS の値の) パラレル実行プロセスの最小の割合を指定できます。このパラメータを設定することで、適切なリソースが使用可能でないかぎり、パラレル操作が順次実行されないようにします。デフォルト値 0 は、プロセスの最小の割合が設定されていないことを意味します。

PARALLEL_MIN_SERVERS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的
値の範囲	0 ～ PARALLEL_MAX_SERVERS の値
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

注意： このパラメータは、Oracle9i Real Application Clusters 環境および排他モードでのパラレル実行に適用されます。

PARALLEL_MIN_SERVERS には、インスタンスに対するパラレル実行プロセスの最小数を指定します。この値は、インスタンスの起動時に作成されるパラレル実行プロセスの数です。

PARALLEL_THREADS_PER_CPU

パラメータ・タイプ	整数
デフォルト値	オペレーティング・システム依存。通常は 2。
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0（ゼロ）以外の任意の数値

注意： このパラメータは、Oracle9i Real Application Clusters 環境および排他モードでのパラレル実行に適用されます。

PARALLEL_THREADS_PER_CPU には、インスタンスのデフォルト並列度を指定し、パラレル適応およびロード・バランス・アルゴリズムを判断します。このパラメータには、パラレル実行中に CPU が処理できるパラレル実行プロセスまたはスレッドの数を記述します。

PARTITION_VIEW_ENABLED

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的:ALTER SESSION
値の範囲	true false

注意： パーティション・ビューではなく、パーティション表（Oracle8 以上で使用可能）を使用することをお勧めします。パーティション・ビューは、下位互換性のためにサポートされています。

PARTITION_VIEW_ENABLED には、オブティマイザがパーティション・ビューを使用するかどうかを指定します。このパラメータを true に設定すると、オブティマイザはパーティション・ビューでの不要な表アクセスを排除（つまりスキップ）し、基礎となる表についての統計からパーティション・ビューについての統計を算出する方法を変更します。

PGA_AGGREGATE_TARGET

パラメータ・タイプ	大整数
構文	<code>PGA_AGGREGATE_TARGET = integer [K M G]</code>
デフォルト値	0 (デフォルトでは、自動メモリー管理はOFF)
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	最小値: 10MB 最大値: 4096 GB - 1

PGA_AGGREGATE_TARGET には、インスタンスに接続されたすべてのサーバー・プロセスが使用できるターゲット集計 PGA メモリーを指定します。メモリー集中型の SQL 演算子（ソート、グループ化、ハッシュ結合、ビットマップ・マージ、ビットマップ作成など）が使用する SQL 作業領域の自動サイズ指定を有効にするには、このパラメータを設定する必要があります。

PLSQL_COMPILER_FLAGS

パラメータ・タイプ	文字列
構文	<code>PLSQL_COMPILER_FLAGS = { [DEBUG NON_DEBUG] [INTERPRETED NATIVE] }</code>
デフォルト値	INTERPRETED、NON_DEBUG
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

PLSQL_COMPILER_FLAGS は、PL/SQL コンパイラが使用するパラメータです。このパラメータには、カンマで区切った文字列のリストとして、コンパイラ・フラグを指定します。

PLSQL_NATIVE_C_COMPILER

パラメータ・タイプ	文字列
構文	<code>PLSQL_NATIVE_C_COMPILER = pathname</code>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	有効なパス名

PLSQL_NATIVE_C_COMPILER には、生成された C ファイルをオブジェクト・ファイルにコンパイルするために使用する C コンパイラのフルパス名を指定します。

PLSQL_NATIVE_LIBRARY_DIR

パラメータ・タイプ	文字列
構文	PLSQL_NATIVE_LIBRARY_DIR = <i>directory</i>
デフォルト値	なし
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	有効なディレクトリ・パス

PLSQL_NATIVE_LIBRARY_DIR は、PL/SQL コンパイラが使用するパラメータです。このパラメータには、システム固有のコンパイラが生成した共有オブジェクトが格納されるディレクトリ名を指定します。

PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	0 ～ $2^{32}-1$ (32 ビットで表される最大値)

PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT には、PLSQL_NATIVE_LIBRARY_DIR で指定したディレクトリに、データベース管理者が作成するサブディレクトリ数を指定します。

PLSQL_NATIVE_LINKER

パラメータ・タイプ	文字列
構文	PLSQL_NATIVE_LINKER = <i>pathname</i>
デフォルト値	なし
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	有効なパス名

PLSQL_NATIVE_LINKER には、リンカー (UNIX での ld、またはオブジェクト・ファイルを共有オブジェクトまたは DLL にリンクするために使用される GNU ld など) のフルパス名を指定します。

PLSQL_NATIVE_MAKE_FILE_NAME

パラメータ・タイプ	文字列
構文	PLSQL_NATIVE_MAKE_FILE_NAME = <i>pathname</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	有効なパス名

PLSQL_NATIVE_MAKE_FILE_NAME には、Make ファイルのフルパス名を指定します。
make ユーティリティ (PLSQL_NATIVE_MAKE_UTILITY で指定) では、Make ファイルを使用して、共有オブジェクトまたは DLL を作成します。

PLSQL_NATIVE_MAKE_UTILITY

パラメータ・タイプ	文字列
構文	PLSQL_NATIVE_MAKE_UTILITY = <i>pathname</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	有効なパス名

PLSQL_NATIVE_MAKE_UTILITY には、make ユーティリティ (UNIX の make または gmake (GNU make)) のフルパス名を指定します。生成された C ソースから、共有オブジェクトまたは DLL を生成するには、make ユーティリティが必要です。

PLSQL_V2_COMPATIBILITY

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

PL/SQL バージョン 2 により、バージョン 8 で禁止されている異常動作が使用できるようになります。下位互換性のためにその動作を保持する場合は、PLSQL_V2_COMPATIBILITY を true に設定します。false に設定すると、PL/SQL バージョン 8 の動作は実行されますが、バージョン 2 の動作は実行されません。

PRE_PAGE_SGA

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

PRE_PAGE_SGA は、インスタンス起動時に SGA 全体がメモリーに読み込まれるかどうかを判断します。オペレーティング・システムのページ表エントリは、SGA のページごとに事前作成されます。この設定によって、インスタンスの起動に必要な時間が長くなる場合がありますが、Oracle のパフォーマンスが最高になるまでの時間が短縮されます。

PROCESSES

パラメータ・タイプ	整数
デフォルト値	PARALLEL_MAX_SERVERS から導出される。
パラメータ・クラス	静的
値の範囲	6 以上。上限は、オペレーティング・システム依存。
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

PROCESSES には、Oracle に同時に接続できるオペレーティング・システムのユーザー・プロセスの最大数を指定します。この値は、ロック、ジョブ・キュー・プロセス、パラレル実行プロセスなどのすべてのバックグラウンド・プロセスを考慮して設定する必要があります。

QUERY_REWRITE_ENABLED

パラメータ・タイプ	文字列
構文	QUERY_REWRITE_ENABLED = {force true false}
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

QUERY_REWRITE_ENABLED を使用すると、データベースへのクエリー・リライトを、グローバルに使用可能または使用禁止にすることができます。

参照： 10-119 ページの「クエリー・リライトを使用可能にする例 :」を参照してください。

QUERY_REWRITE_INTEGRITY

パラメータ・タイプ	文字列
構文	<pre>QUERY_REWRITE_INTEGRITY = {stale_tolerated trusted enforced}</pre>
デフォルト値	enforced
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

QUERY_REWRITE_INTEGRITY は、クエリー・リライトを行う程度を判断します。最も安全なレベルでは、適用されていないリレーションシップに依存するクエリー・リライト変換は使用されません。

RDBMS_SERVER_DN

パラメータ・タイプ	X.500 識別名
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	X.500 識別名形式の値すべて

RDBMS_SERVER_DN には、Oracle サーバーの識別名 (DN) を指定します。エンタープライズ・ディクショナリ・サービスからエンタープライズ・ロールを取り出すために使用されます。

READ_ONLY_OPEN_DELAYED

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

READ_ONLY_OPEN_DELAYED は、読み込み専用表領域内のデータ・ファイルがアクセスされる場合を判断します。

RECOVERY_PARALLELISM

パラメータ・タイプ	整数
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	静的
値の範囲	オペレーティング・システム依存。上限は、 PARALLEL_MAX_SERVERS。

RECOVERY_PARALLELISM には、インスタンス・リカバリまたはクラッシュ・リカバリに関与するプロセスの数を指定します。0 または 1 は、リカバリが 1 つのプロセスによって順次実行されることを示します。

REMOTE_ARCHIVE_ENABLE

パラメータ・タイプ	文字列
構文	REMOTE_ARCHIVE_ENABLE = {receive [, send] false true}
デフォルト値	true
パラメータ・クラス	静的

REMOTE_ARCHIVE_ENABLE によって、リモートの宛先への REDO アーカイブの送信およびリモートでアーカイブされた REDO の受信が使用可能または使用禁止になります。

REMOTE_DEPENDENCIES_MODE

パラメータ・タイプ	文字列
構文	REMOTE_DEPENDENCIES_MODE = {TIMESTAMP SIGNATURE}
デフォルト値	TIMESTAMP
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

REMOTE_DEPENDENCIES_MODE には、リモートの PL/SQL ストアド・プロシージャに対する依存性の処理方法を指定します。

REMOTE_LISTENER

パラメータ・タイプ	文字列
構文	REMOTE_LISTENER = <i>network_name</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SYSTEM

REMOTE_LISTENER には、Oracle Net リモート・リスナー（このインスタンスと異なるマシン上で実行中のリスナー）のアドレスまたはアドレス・リストを解決するネットワーク名を指定します。アドレスまたはアドレス・リストは、TNSNAMES.ORA ファイルまたはご使用のシステム用に構成されている他のアドレス・リポジトリで指定されます。

REMOTE_LOGIN_PASSWORDFILE

パラメータ・タイプ	文字列
構文	REMOTE_LOGIN_PASSWORDFILE= {NONE SHARED EXCLUSIVE}
デフォルト値	NONE
パラメータ・クラス	静的
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

REMOTE_LOGIN_PASSWORDFILE には、Oracle がパスワード・ファイルを確認するかどうか、およびそのパスワード・ファイルを使用できるデータベースの数を指定します。

REMOTE_OS_AUTHENT

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

REMOTE_OS_AUTHENT には、OS_AUTHENT_PREFIX パラメータ値でリモート・クライアントが認証されるかどうかを指定します。

REMOTE_OS_ROLES

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

REMOTE_OS_ROLES には、オペレーティング・システム・ロールがリモート・クライアントに与えられるかどうかを指定します。デフォルト値 false では、Oracle によって、リモート・クライアントに対するロールが識別および管理されます。

REPLICATION_DEPENDENCY_TRACKING

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	静的
値の範囲	true false

REPLICATION_DEPENDENCY_TRACKING によって、データベースへの読込み / 書込み操作に対する依存性追跡を使用可能または使用禁止にできます。依存性追跡は、レプリケート環境でパラレルに変更を伝播するときに重要になります。

RESOURCE_LIMIT

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	true false

RESOURCE_LIMIT は、リソース制限がデータベース・プロファイルで行われるかどうかを判断します。

参照： 10-119 ページの「リソース制限を使用可能にする例:」を参照してください。

RESOURCE_MANAGER_PLAN

パラメータ・タイプ	文字列
構文	RESOURCE_MANAGER_PLAN = <i>plan_name</i>
デフォルト値	なし
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	有効な任意の文字列

RESOURCE_MANAGER_PLAN には、インスタンスに使用するトップレベルのリソース・プランを指定します。リソース・マネージャは、このトップレベルのプランをすべての子孫（サブプラン、ディレクティブおよびコンシューマ・グループ）にロードします。このパラメータを指定しない場合、リソース・マネージャはデフォルトのオフです。

ROLLBACK_SEGMENTS

パラメータ・タイプ	文字列
構文	ROLLBACK_SEGMENTS = (segment_name [, segment_name] ...)
デフォルト値	このパラメータを指定しない場合、インスタンスでは、パブリック・ロールバック・セグメントがデフォルトで使用される。
パラメータ・クラス	静的
値の範囲	DBA_ROLLBACK_SEGS に示されたすべてのロールバック・セグメント名（SYSTEM を除く）
Real Application Clusters	複数インスタンスには、異なる値を指定可能

ROLLBACK_SEGMENTS には、インスタンスに割り当てる 1 つ以上のロールバック・セグメントの名前を指定します。このパラメータが設定されると、ロールバック・セグメント数がインスタンスに必要な最小数（TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT から算出）を超えていても、インスタンスは、このパラメータで指定されたロールバック・セグメントをすべて取得します。

ROW_LOCKING

パラメータ・タイプ	文字列
構文	ROW_LOCKING = {ALWAYS DEFAULT INTENT}
デフォルト値	ALWAYS
パラメータ・クラス	静的
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

ROW_LOCKING には、UPDATE 操作中に行ロックを取得するかどうかを指定します。

SERIAL_REUSE

パラメータ・タイプ	文字列
構文	SERIAL_REUSE = {DISABLE SELECT DML PLSQL ALL}
デフォルト値	DISABLE
パラメータ・クラス	静的

SERIAL_REUSE には、メモリー・シリアル再利用機能を使用するカーソルの種類を指定します。この機能は、SGA 内のプライベート・カーソル・メモリーを割り当て、同じカーソルを実行するセッションによってそのメモリーが（同時ではなく、シリアルに）再利用できるようになります。

SERVICE_NAMES

パラメータ・タイプ	文字列
構文	SERVICE_NAMES = db_service_name [, db_service_name [...]]
デフォルト値	DB_NAME.DB_DOMAIN（定義されている場合）
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	ASCII 文字列、またはカンマで区切られた文字列名のリスト

SERVICE_NAMES には、このインスタンスに接続する 1 つ以上のデータベース・サービスの名前を指定します。同じデータベースの異なる使用を区別するために、複数サービス名を指定できます。

SESSION_CACHED_CURSORS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	動的:ALTER SESSION
値の範囲	0 以上。上限は、オペレーティング・システム依存。
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

SESSION_CACHED_CURSORS を使用すると、キャッシュするセッション・カーソル数を指定できます。同じ SQL 文に対して解析コールを繰り返し行くと、その文のセッション・カーソルをセッション・カーソル・キャッシュに移動できるようになります。後続の解析コールでは、カーソルはキャッシュ内にあるので、カーソルを再オープンする必要がなくなります。新規のエントリのために領域を空ける必要がある場合、Oracle は、最近使用されたアルゴリズムを使用して、セッション・カーソル・キャッシュ内のエントリを削除します。

SESSION_MAX_OPEN_FILES

パラメータ・タイプ	整数
デフォルト値	10
パラメータ・クラス	静的
値の範囲	1 ～ 50 またはオペレーティング・システムのレベルで定義された MAX_OPEN_FILES 値 (いずれか小さい方)

SESSION_MAX_OPEN_FILES には、すべてのセッションでオープンできる BFILE の最大数を指定します。ここで設定された数に達すると、その後は DBMS_LOB.FILEOPEN() または OCILobFileOpen() を使用してセッションでさらにファイルをオープンしようとしても失敗します。このパラメータの最大値は、使用しているオペレーティング・システムで定義されている対応パラメータに依存します。

SESSIONS

パラメータ・タイプ	整数
デフォルト値	次の値が導出される: $(1.1 \times \text{PROCESSES}) + 5$
パラメータ・クラス	静的
値の範囲	1 ～ 2^{31}

SESSIONS には、システムに作成できるセッションの最大数を指定します。すべてのログインにはセッションが必要なため、このパラメータにはシステムでの同時ユーザーの最大数を実地的な値で指定します。このパラメータには、同時ユーザーの推定最大値に、バックグラウンド・プロセスの数を加えて、さらに再帰セッションの約 10% を加えた値を常に明示的に設定しておく必要があります。

SGA_MAX_SIZE

パラメータ・タイプ	大整数
構文	SGA_MAX_SIZE = integer [K M G]
デフォルト値	起動時の SGA サイズは、SGA 内の個々のプール（バッファ・キャッシュ、共有プール、ラージ・プールなど）のサイズによって異なる。
パラメータ・クラス	静的
値の範囲	0 以上。上限は、オペレーティング・システム依存。

SGA_MAX_SIZE には、インスタンスの存続期間を通しての SGA の最大サイズを指定します。

SHADOW_CORE_DUMP

パラメータ・タイプ	文字列
構文	SHADOW_CORE_DUMP = {partial full none}
デフォルト値	partial
パラメータ・クラス	静的

SHADOW_CORE_DUMP には、フォアグラウンド（クライアント）・プロセスのコア・ファイルに SGA を含めるかどうかを指定します。

SHARED_MEMORY_ADDRESS

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的

SHARED_MEMORY_ADDRESS および HI_SHARED_MEMORY_ADDRESS には、システム・グローバル領域（SGA）の実行時の開始アドレスを指定します。これらのパラメータは、リンク時の SGA の開始アドレスを指定する多くのプラットフォームでは無視されます。

SHARED_POOL_RESERVED_SIZE

パラメータ・タイプ	大整数
構文	SHARED_POOL_RESERVED_SIZE = integer [K M G]
デフォルト値	SHARED_POOL_SIZE の値 5%
パラメータ・クラス	静的
値の範囲	最小値: 5000 最大値: SHARED_POOL_SIZE の値の 50%

SHARED_POOL_RESERVED_SIZE には、大きな共有プール・メモリー領域の要求が連続したときのために確保する共有プール領域（バイト単位）を指定します。

SHARED_POOL_SIZE

パラメータ・タイプ	大整数
構文	SHARED_POOL_SIZE = integer [K M G]
デフォルト値	32 ビット・プラットフォーム: 8MB。最も近いグラニュル・サイズに切り上げられる。 64 ビット・プラットフォーム: 64MB。最も近いグラニュル・サイズに切り上げられる。
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	最小値: グラニュル・サイズ 最大値: オペレーティング・システム依存

SHARED_POOL_SIZE には、共有プールのサイズ（バイト単位）を指定します。共有プールには、共有カーソル、ストアド・プロシージャ、制御構造体、およびその他の構造体が含まれます。PARALLEL_AUTOMATIC_TUNING を false に設定している場合、共有プールからパラレル実行メッセージ・バッファが割り当てられます。値を大きくするほど、マルチユーザー・システムでのパフォーマンスが改善されます。値が小さいほど、使用されるメモリーは少なくなります。

共有サーバーのパラメータ

Oracle9i から、マルチスレッド・サーバー・アーキテクチャは、共有サーバー・アーキテクチャといえます。

インスタンスを起動すると、Oracle は SHARED_SERVERS および DISPATCHERS 初期化パラメータの値に基づく共有サーバー・アーキテクチャの共有サーバー・プロセスおよびディスパッチャ・プロセスを作成します。ALTER SYSTEM 文で SHARED_SERVERS および

DISPATCHERS パラメータを設定し、インスタンスの実行中に次の操作のいずれかを実行できます。

- 共有サーバー・プロセスの最小値を増やして、追加の共有サーバー・プロセスを作成します。
- 現行のコールが処理を終了した後、既存の共有サーバー・プロセスを終了します。
- 特定のプロトコルに対するディスパッチャ・プロセスをより多く作成します。プロトコル全体で、初期化パラメータ MAX_DISPATCHERS によって指定される数まで作成できます。
- 現行のユーザー・プロセスがインスタンスから切断した後、特定のプロトコルに対する既存のディスパッチャ・プロセスを終了します。

SHARED_SERVERS

パラメータ・タイプ 整数

デフォルト値 1 (共有サーバー・アーキテクチャを使用する場合)
0 (共有サーバー・アーキテクチャを使用しない場合)

パラメータ・クラス 動的: ALTER SYSTEM

値の範囲 オペレーティング・システム依存

SHARED_SERVERS には、インスタンスの起動時に作成するサーバー・プロセスの数を指定します。システム負荷が減少する場合は、このサーバーの最小値が保持されます。そのため、SHARED_SERVERS をシステムの起動時にあまり高く設定しないように注意してください。

参照: 10-119 ページの「[共有サーバーの設定の変更例:](#)」を参照してください。

SHARED_SERVER_SESSIONS

パラメータ・タイプ 整数

デフォルト値 次の値が導出される: CIRCUITS または SESSIONS - 5 (いずれか小さい方)

パラメータ・クラス 静的

値の範囲 0 ～ SESSIONS - 5

SHARED_SERVER_SESSIONS には、ユーザー・セッションにより使用可能な、共有サーバー・アーキテクチャの合計数を指定します。このパラメータを設定すると、専用サーバーのユーザー・セッションを確保することが可能になります。

SORT_AREA_RETAINED_SIZE

パラメータ・タイプ	整数
デフォルト値	SORT_AREA_SIZE から導出される。
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM ... DEFERRED
値の範囲	2 個のデータベース・ブロックに相当する値から SORT_AREA_SIZE の値まで。

注意： インスタンスが共有サーバーのオプションで構成されていない場合、SORT_AREA_RETAINED_SIZE パラメータを使用することはお薦めしません。かわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。SORT_AREA_RETAINED_SIZE は、下位互換性を保つために残されます。

SORT_AREA_RETAINED_SIZE には、ソート完了後にそのまま保持されるユーザー・グローバル領域 (UGA) メモリーの最大量 (バイト単位) を指定します。保持されたメモリー・サイズにより、メモリー内のソート領域の保持に使用される読み込みバッファのサイズが制御されます。このメモリーは、最後の行がソート領域から取り出されると解放され、オペレーティング・システムではなく、UGA に戻されます。

SORT_AREA_SIZE

パラメータ・タイプ	整数
デフォルト値	65536
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM ... DEFERRED
値の範囲	最小値: 6 個のデータベース・ブロックに相当する値 最大値: オペレーティング・システム依存

注意： インスタンスが共有サーバーのオプションで構成されていない場合、SORT_AREA_SIZE パラメータを使用することはお薦めしません。かわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。SORT_AREA_SIZE は、下位互換性を保つために残されます。

`SORT_AREA_SIZE` には、ソートに使用するメモリの最大量をバイトで指定します。ソート完了後、行が戻される前に、ソート用に割り当てられたすべてのメモリ（`SORT_AREA_RETAINED_SIZE` パラメータの指定量を除く）が解放されます。最後の行が戻されると、残りのメモリが解放されます。

SPFILE

パラメータ・タイプ	文字列
構文	<code>SPFILE = spfile_name</code>
デフォルト値	<code>ORACLE_HOME/dbs/spfile.ora</code>
パラメータ・クラス	静的
値の範囲	任意の有効な SPFILE
Real Application Clusters	複数インスタンスには同じ値を指定する必要がある。

このパラメータには、使用中の現行のサーバー・パラメータ・ファイル（SPFILE）名を指定します。このパラメータは、クライアント側の PFILE で定義し、使用するサーバー・パラメータ・ファイル名を指定できます。

SQL92_SECURITY

パラメータ・タイプ	ブール
デフォルト値	<code>false</code>
パラメータ・クラス	静的
値の範囲	<code>true</code> <code>false</code>

SQL92 規格は、WHERE または SET 句で表列値を参照する UPDATE または DELETE 文を実行する場合に、ユーザーが表の SELECT 権限を持つことを、セキュリティ管理者が要求できることを指定します。SQL92_SECURITY により、ユーザーが、UPDATE または DELETE 文などを実行するために、SELECT オブジェクト権限が付与されるべきかどうかを指定できます。

SQL_TRACE

パラメータ・タイプ	ブール
デフォルト値	<code>false</code>
パラメータ・クラス	静的
値の範囲	<code>true</code> <code>false</code>

SQL_TRACE の値によって、SQL トレース機能が使用禁止または使用可能になります。このパラメータを true に設定すると、パフォーマンスを改善するためのチューニングについての情報が提供されます。この値は、DBMS_SYSTEM パッケージを使用しても変更できます。

STANDBY_ARCHIVE_DEST

パラメータ・タイプ	文字列
構文	STANDBY_ARCHIVE_DEST = <i>filespec</i>
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	動的:ALTER SYSTEM
値の範囲	RAW 以外の有効なパス名またはデバイス名

STANDBY_ARCHIVE_DEST は、管理リカバリ・モードのスタンバイ・データベースのみに関連します。このパラメータには、プライマリ・データベースからのアーカイブ・ログの位置を指定します。STANDBY_ARCHIVE_DEST および LOG_ARCHIVE_FORMAT は、完全に修飾されたスタンバイ・ログ・ファイル名を作成するため、およびスタンバイ制御ファイルにファイル名を格納するために使用されます。

STANDBY_FILE_MANAGEMENT

パラメータ・タイプ	文字列
構文	STANDBY_FILE_MANAGEMENT = {MANUAL AUTO}
デフォルト値	MANUAL
パラメータ・クラス	動的:ALTER SYSTEM

STANDBY_FILE_MANAGEMENT によって、スタンバイ・ファイルの自動管理が使用可能または使用禁止になります。スタンバイ・ファイルの自動管理が使用可能になると、プライマリ・データベースでのオペレーティング・システム・ファイルの追加および削除が、スタンバイ・データベースにレプリケートされます。

STAR_TRANSFORMATION_ENABLED

パラメータ・タイプ	文字列
構文	STAR_TRANSFORMATION_ENABLED = {TEMP_DISABLE TRUE FALSE}
デフォルト値	FALSE
パラメータ・クラス	動的:ALTER SESSION

STAR_TRANSFORMATION_ENABLED によって、コストベースの問合せ変換を、スター問合せに適用するかどうかが判別されます。

STATISTICS_LEVEL

パラメータ・タイプ	文字列
構文	STATISTICS_LEVEL = {ALL TYPICAL BASIC}
デフォルト値	TYPICAL
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

STATISTICS_LEVEL には、データベースの統計収集レベルを設定します。

TAPE_ASYNC_IO

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	静的
値の範囲	true false

TAPE_ASYNC_IO は、シーケンシャル・デバイスへの I/O（たとえば、テープとの間で行う Oracle データのバックアップまたはリストア）が非同期かどうか、つまり、Real Application Clusters プロセスで、表スキャン中に、CPU 処理と I/O 要求をオーバーラップできるかどうかを制御します。プラットフォームがシーケンシャル・デバイスへの非同期 I/O をサポートしている場合は、このパラメータをデフォルト値のままにしておくことをお勧めします。プラットフォームがシーケンシャル・デバイスへの非同期 I/O をサポートしている場合は、このパラメータをデフォルト値のままにしておくことをお勧めします。ただし、非同期 I/O の実装が安定してない場合は、TAPE_ASYNC_IO を false に設定することで、非同期 I/O を使用禁止にできます。プラットフォームがシーケンシャル・デバイスへの非同期 I/O をサポートしていない場合、このパラメータは無効です。

THREAD

パラメータ・タイプ	整数
デフォルト値	0
パラメータ・クラス	静的
値の範囲	0 ～使用可能スレッドの最大数
Real Application Clusters	このパラメータを指定する場合、複数インスタンスには、異なる値を指定する必要がある。

THREAD は、Oracle9i Real Application Clusters のパラメータで、インスタンスで使用される REDO スレッドの番号を指定します。

TIMED_OS_STATISTICS

パラメータ・タイプ	整数
デフォルト値	STATISTICS_LEVEL が ALL に設定されている場合は 5 STATISTICS_LEVEL が BASIC または TYPICAL に設定されている場合は 0
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	無制限

TIMED_OS_STATISTICS には、クライアントからサーバーへの要求が発生したとき、または要求が完了したときに、オペレーティング・システムの統計情報を収集する間隔（秒単位）を指定します。

TIMED_STATISTICS

パラメータ・タイプ	ブール
デフォルト値	STATISTICS_LEVEL が TYPICAL または ALL に設定されている場合は true STATISTICS_LEVEL が BASIC に設定されている場合は false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

TIMED_STATISTICS には、時刻に関連する統計情報を収集するかどうかを指定します。

TRACE_ENABLED

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	true false
Real Application Clusters	すべてのインスタンスに、このパラメータを設定する必要がある。 また、複数インスタンスには、同じ値を指定する必要がある。

TRACE_ENABLED は、Oracle の実行履歴またはコード・パスのトレースを制御します。オラクル社カスタマ・サポート・センターでは、この情報をデバッグに使用します。

TRACEFILE_IDENTIFIER

パラメータ・タイプ	文字列
構文	TRACEFILE_IDENTIFIER = "traceid"
デフォルト値	なし
パラメータ・クラス	動的: ALTER SESSION
値の範囲	ご使用のプラットフォーム上で、ファイル名に使用される可能性があるすべての文字

TRACEFILE_IDENTIFIER には、Oracle Trace のファイル名となるユーザー定義の識別子を指定します。このようなユーザー定義の識別子を使用すると、単純に名前からトレース・ファイルを識別できます。その場合、トレース・ファイルをオープンしたり、内容を確認する必要はありません。

TRANSACTION_AUDITING

パラメータ・タイプ	ブール
デフォルト値	true
パラメータ・クラス	動的: ALTER SYSTEM ... DEFERRED
値の範囲	true false

TRANSACTION_AUDITING を true に設定すると、ユーザー・ログイン名、ユーザー名、セッション ID、一部のオペレーティング・システムの情報およびクライアント情報を含む特別な REDO レコードが生成されます。逐次トランザクションごとに、セッション ID のみを含むレコードが生成されます。これらの後続のレコードは、セッション ID を含む最初のレコードにリンクされます。

TRANSACTIONS

パラメータ・タイプ	整数
デフォルト値	次の値が導出される: (1.1 × SESSIONS)
パラメータ・クラス	静的
値の範囲	4 ~ 2 ³²
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

TRANSACTIONS には、同時実行トランザクションの最大数を指定します。大きい値を指定すると、SGA のサイズが大きくなり、割り当てるロールバック・セグメント数が増えます。再帰的トランザクションが実行できるように、デフォルト値は SESSIONS（つまり PROCESSES）より大きい値です。

注意： サーバー・パラメータ・ファイル（spfile）を使用してデータベースを起動した場合にのみ、ALTER SYSTEM を使用してこのパラメータを設定できます。この場合、SCOPE = SPFILE を指定する必要があります。

TRANSACTIONS_PER_ROLLBACK_SEGMENT

パラメータ・タイプ	整数
デフォルト値	5
パラメータ・クラス	静的
値の範囲	1 以上。上限は、オペレーティング・システム依存。
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

TRANSACTIONS_PER_ROLLBACK_SEGMENT には、各ロールバック・セグメントを処理する同時トランザクションの数を指定します。起動時に取得されるロールバック・セグメントの最小数は、TRANSACTIONS をこのパラメータの値で割った数です。たとえば、TRANSACTIONS が 101 でこのパラメータが 10 の場合、取得されるロールバック・セグメントの最小数は 101/10 となり、11 に切り上げられます。

UNDO_MANAGEMENT

パラメータ・タイプ	文字列
構文	UNDO_MANAGEMENT = {MANUAL AUTO}
デフォルト値	MANUAL
パラメータ・クラス	静的
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

UNDO_MANAGEMENT には、システムが使用する必要がある UNDO 領域管理モードを指定します。AUTO に設定すると、インスタンスは、自動 UNDO 管理モードで起動します。手動 UNDO 管理モードでは、UNDO 領域がロールバック・セグメントとして外部的に割り当てられます。

UNDO_RETENTION

パラメータ・タイプ	整数
デフォルト値	900
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	0 ~ 2 ³² -1 (32 ビットで表される最大値)
Real Application Clusters	複数インスタンスには、同じ値を指定する必要がある。

UNDO_RETENTION には、データベースに保存するコミットされたロールバック情報の量 (秒単位) を指定します。UNDO_RETENTION は、データ・ブロックの以前のイメージを作成するために変更をロールバックする場合に、以前のロールバック情報を必要とする問合せで使用します。この値は、インスタンスの起動時に設定できます。

UNDO_SUPPRESS_ERRORS

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM
値の範囲	true false

UNDO_SUPPRESS_ERRORS によって、自動 UNDO 管理モードで手動 UNDO 管理モードの操作 (ALTER ROLLBACK SEGMENT ONLINE など) を実行しているときのエラーを回避できます。このパラメータを設定すると、すべてのアプリケーション・プログラムおよびスクリプトが自動 UNDO 管理モードに変換される前に、UNDO 表領域の特性を使用できます。たとえば、SET TRANSACTION USE ROLLBACK SEGMENT 文を使用するツールの場合、そのツールに「ALTER SESSION SET UNDO_SUPPRESS_ERRORS = TRUE」文を追加して、ORA-30019 エラーを回避できます。

UNDO_TABLESPACE

パラメータ・タイプ	文字列
構文	UNDO_TABLESPACE = <i>undoname</i>
デフォルト値	データベース内の最初に使用可能な UNDO 表領域
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	既存の有効な UNDO 表領域名
Real Application Clusters	複数インスタンスには、異なる値を指定可能。

UNDO_TABLESPACE には、インスタンスの起動時に使用する UNDO 表領域を指定します。インスタンスが手動 UNDO 管理モードのときにこのパラメータを指定すると、エラーが発生して起動できません。

USE_INDIRECT_DATA_BUFFERS

パラメータ・タイプ	ブール
デフォルト値	false
パラメータ・クラス	静的
値の範囲	true false

USE_INDIRECT_DATA_BUFFERS は、システム・グローバル領域 (SGA) のメモリーの使用方法を制御します。このパラメータは、4GB を超える物理メモリーをサポートする 32 ビット・プラットフォーム用の拡張バッファ・キャッシュ・メカニズムを使用可能または使用禁止にします。この物理メモリーをサポートしないプラットフォームでは、このパラメータは無視されます。

USE_STORED_OUTLINES

構文: USE_STORED_OUTLINES = {TRUE | FALSE | *category_name*}

USE_STORED_OUTLINES パラメータは、オブティマイザが実行計画を生成するために格納されているパブリック・アウトラインを使用するかどうかを決定します。

USE_STORED_OUTLINES は、初期化パラメータではありません。

- TRUE に設定した場合、要求をコンパイルするときに、オブティマイザが DEFAULT カテゴリにストアド・アウトラインを使用します。
- FALSE に設定した場合、オブティマイザがストアド・アウトラインを使用しません。これはデフォルト値です。
- *category_name* を設定した場合、要求をコンパイルするときに、オブティマイザは *category_name* カテゴリにストアド・アウトラインを使用します。

USER_DUMP_DEST

パラメータ・タイプ	文字列
構文	USER_DUMP_DEST = { <i>pathname</i> <i>directory</i> }
デフォルト値	オペレーティング・システム依存
パラメータ・クラス	動的: ALTER SYSTEM
値の範囲	有効なローカル・パス、ディレクトリまたはディスク

USER_DUMP_DEST には、ユーザー・プロセスのかわりに、サーバーがデバッグ・トレース・ファイルを書き込むディレクトリのパス名を指定します。

UTL_FILE_DIR

パラメータ・タイプ	文字列
構文	UTL_FILE_DIR = <i>pathname</i>
デフォルト値	なし
パラメータ・クラス	静的
値の範囲	有効なディレクトリ・パス

UTL_FILE_DIR により、PL/SQL によるファイル I/O を使用するディレクトリを 1 つ以上指定できます。複数のディレクトリを指定する場合、初期化パラメータ・ファイルの別々の行で各ディレクトリに対する UTL_FILE_DIR パラメータを繰り返す必要があります。

WORKAREA_SIZE_POLICY

パラメータ・タイプ	文字列
構文	WORKAREA_SIZE_POLICY = { <i>AUTO</i> <i>MANUAL</i> }
デフォルト値	<i>AUTO</i> (PGA_AGGREGATE_TARGET が設定されている場合) <i>MANUAL</i> (PGA_AGGREGATE_TARGET が設定されていない場合)
パラメータ・クラス	動的: ALTER SESSION、ALTER SYSTEM

WORKAREA_SIZE_POLICY には、作業領域のサイズ指定方法を指定します。このパラメータは、作業領域のチューニング時に使用するモードを制御します。

例

REDO ログの手動アーカイブの例： 次の文は、スレッド番号 3、ログ順序番号 4 の REDO ログ・ファイル・グループを手動でアーカイブします。

```
ALTER SYSTEM ARCHIVE LOG THREAD 3 SEQUENCE 4;
```

次の文は、SCN 9356083 の REDO ログ・エントリを含む REDO ログ・ファイル・グループを手動でアーカイブします。

```
ALTER SYSTEM ARCHIVE LOG CHANGE 9356083;
```

次の文は、メンバー 'disk1:log6.log' を含む REDO ログ・ファイル・グループを、'diska:[arch\$]' という場所にあるアーカイブ REDO ログ・ファイルに手動でアーカイブします。


```
ALTER SYSTEM ARCHIVE LOG  
    LOGFILE 'disk1:log6.log'  
    TO 'diska:[arch$]';
```

クエリー・リライトを使用可能にする例： 次の文は、明示的に使用禁止にされていないすべてのマテリアライズド・ビューに対するすべてのセッションにおけるクエリー・リライトを使用可能にします。

```
ALTER SYSTEM SET QUERY_REWRITE_ENABLED = TRUE;
```

セッションへのログインを制限する例： たとえば、アプリケーションのメンテナンス中は、RESTRICTED SESSION システム権限が付与されているアプリケーション開発者のみがログインできるように制限できます。このためには、次の文を発行します。

```
ALTER SYSTEM  
    ENABLE RESTRICTED SESSION;
```

次に、ALTER SYSTEM 文の KILL SESSION 句を使用すると、既存のセッションをどれでも終了できます。

アプリケーションのメンテナンスが終了した後で、次の文を発行することによって、CREATE SESSION システム権限が付与されているユーザーもログインできるようになります。

```
ALTER SYSTEM  
    DISABLE RESTRICTED SESSION;
```

共有プールの消去例： 共有プールを消去してから、パフォーマンス分析を開始します。共有プールを消去する場合、次の文を発行します。

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

チェックポイントの発生例： 次の文は、チェックポイントを強制的に発生させます。

```
ALTER SYSTEM CHECKPOINT;
```

リソース制限を使用可能にする例： 次の ALTER SYSTEM 文は、リソース制限を動的に使用可能にします。

```
ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;
```

共有サーバーの設定の変更例： 次の文は、共有サーバー・プロセスの最小数を 25 に変更します。

```
ALTER SYSTEM SET SHARED_SERVERS = 25;
```

共有サーバー・プロセスの数が 25 より少ない場合は追加作成されます。共有サーバー・プロセスが 25 より多く、25 あれば負荷を管理できる場合は、現行のコールによる処理が終了した時点で、25 を超える分の共有サーバー・プロセスは終了します。

次の文は、TCP/IP プロトコルのディスパッチャ・プロセス数を 5 に、DECNet プロトコルのディスパッチャ・プロセス数を 10 に動的に変更します。

```
ALTER SYSTEM
  SET DISPATCHERS =
    ' (INDEX=0) (PROTOCOL=TCP) (DISPATCHERS=5) ',
    ' (INDEX=1) (PROTOCOL=DECNet) (DISPATCHERS=10) ';
```

TCP のディスパッチャ・プロセスの数が 5 より少ない場合、ディスパッチャ・プロセスが新しく作成されます。5 より多い場合は、接続されているユーザーが接続を切断した後に、5 を超える分のディスパッチャ・プロセスは終了します。

DECNet のディスパッチャ・プロセスの数が 10 より少ない場合は、ディスパッチャ・プロセスが新しく作成されます。10 より多い場合は、接続されているユーザーが接続を切断した後に、10 を超える分のディスパッチャ・プロセスは終了します。

これ以外のプロトコル用として既存ディスパッチャがある場合、この文は、そのディスパッチャの数に影響しません。

ライセンス・パラメータの変更例： 次の文は、インスタンスにおけるセッションの最大数を 64 に、セッションの警告しきい値を 54 に動的に変更します。

```
ALTER SYSTEM
  SET LICENSE_MAX_SESSIONS = 64
  LICENSE_SESSIONS_WARNING = 54;
```

セッション数が 54 に達した場合、後続の各セッションの ALERT ファイルに警告メッセージが書き込まれます。RESTRICTED SESSION システム権限を持つユーザーも、後続セッションを開始する場合に、警告メッセージを受け取ります。

セッション数が 64 に達した場合、セッション数が再び 64 を下回るまでは、RESTRICTED SESSION システム権限を持つユーザー以外は新しいセッションを開始できません。

次の文は、インスタンスのセッションの最大数を動的に使用禁止にします。この文の実行後は、インスタンスのセッション数に制限がなくなります。

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 0;
```

次の文は、データベースのユーザー数の制限を 200 に動的に変更します。この文の実行後は、データベース・ユーザー数が 200 を超えることはありません。

```
ALTER SYSTEM SET LICENSE_MAX_USERS = 200;
```

ログ・スイッチの発生例： 書き込み中のファイルの削除および名前の変更はできません。ただし、ログ・スイッチを強制的に発生させることによって、現行の REDO ログ・ファイル・グループまたはそのメンバーの 1 つを削除または名前の変更ができます。強制的に発生したログ・スイッチは、インスタンスの REDO ログ・スレッドのみに影響します。次の文は、ログ・スイッチを強制的に発生させます。

```
ALTER SYSTEM SWITCH LOGFILE;
```

分散リカバリを使用可能にする例： 次の文は、分散リカバリを使用可能にします。

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

デモンストレーションまたはテストの目的で、分散リカバリを使用禁止にする場合があります。次の文を使用すると、シングルプロセス・モードまたはマルチプロセス・モードのどちらの分散リカバリでも使用禁止にできます。

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

デモンストレーションまたはテストが終了した場合、ALTER SYSTEM 文に ENABLE DISTRIBUTED RECOVERY 句を指定して実行すると、分散リカバリを再び使用可能にできます。

セッションの中断例： あるユーザーのセッションで、他のユーザーが必要とするリソースを使用している場合、そのユーザーのセッションを中断させる必要があります。このユーザーは、セッションが中断したことを示すエラー・メッセージを受け取ります。このユーザーは、新しいセッションを開始しないかぎり、このデータベースをコールできません。次の V\$SESSION 動的パフォーマンス表のデータについて考えます。

```
SELECT sid, serial#, username
FROM v$session;
```

SID	SERIAL#	USERNAME
1	1	
2	1	
3	1	
4	1	
5	1	
7	1	
8	28	OPS\$BQUIGLEY
10	211	OPS\$SWIFT
11	39	OPS\$OBRIEN
12	13	SYSTEM
13	8	SCOTT

次の例では、V\$SESSION の SID 値と SERIAL# 値を使用して、ユーザー scott のセッションを中断します。

```
ALTER SYSTEM KILL SESSION '13, 8';
```

セッションの切断例： 次の文は、V\$SESSION の SID と SERIAL# の値を使用して、ユーザー scott のセッションを切断します。

```
ALTER SYSTEM DISCONNECT SESSION '13, 8' POST_TRANSACTION;
```

SQL 文 : ALTER TABLE ～ ALTER TABLESPACE

この章では、次の SQL 文について説明します。

- ALTER TABLE
- ALTER TABLESPACE

ALTER TABLE

用途

ALTER TABLE を使用すると、非パーティション表、パーティション表、表パーティションおよび表サブパーティションの定義を変更できます。オブジェクト表またはオブジェクト列を含むリレーショナル表の場合は、ALTER TABLE を使用して型が変更された後に、表を参照する型の最新の定義に変換します。

参照：

- 表の作成については、15-7 ページの「[CREATE TABLE](#)」を参照してください。
- Oracle Text とともに使用する ALTER TABLE については、『Oracle Text リファレンス』を参照してください。

前提条件

表が自分のスキーマ内にある必要があります。自分のスキーマ内でない場合は、その表に対する ALTER 権限または ALTER ANY TABLE システム権限が必要です。また、CREATE ANY INDEX 権限が必要な操作もあります。

パーティション化操作におけるその他の前提条件 表の所有者でない場合、`drop_table_partition` または `truncate_table_partition` 句を使用するには、DROP ANY TABLE 権限が必要です。

`add_table_partition`、`modify_table_partition`、`move_table_partition` および `split_table_partition` 句を使用する場合、領域を確保する表領域に領域割当て制限が必要です。

制約およびトリガーにおけるその他の前提条件 一意キー制約または主キー制約を有効にする場合は、表に索引を作成するための権限が必要です。Oracle は、その表を含むスキーマにおいて、一意または主キー列に索引を作成するため、この権限が必要になります。

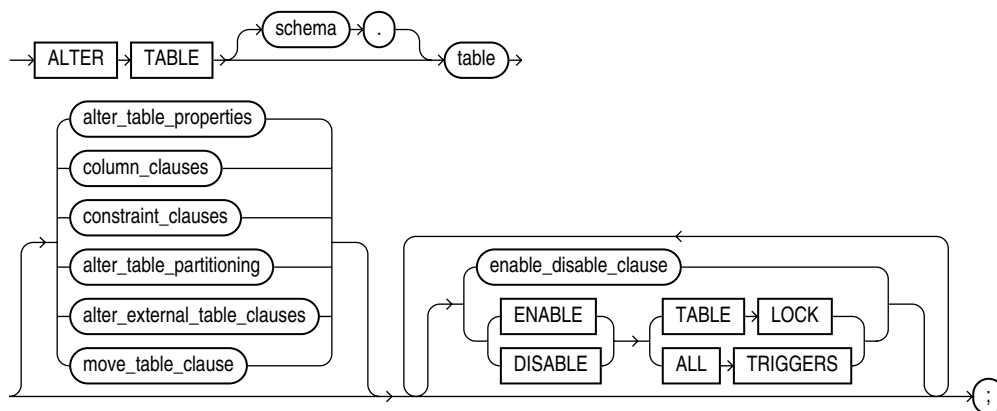
トリガーを使用可能または使用禁止にする場合、トリガーが自分のスキーマ内にある必要があります。自分のスキーマ内でない場合は、ALTER ANY TRIGGER システム権限が必要です。

オブジェクト型を使用する場合のその他の前提条件 表を変更するときに列定義でオブジェクト型を使用する場合、そのオブジェクトが、変更する表と同じスキーマに属している必要があります。または、EXECUTE ANY TYPE システム権限またはそのオブジェクト型に対する EXECUTE スキーマ・オブジェクト権限が必要です。

参照： 索引を作成する場合に必要な権限については、13-61 ページの「[CREATE INDEX](#)」を参照してください。

構文

alter_table::=



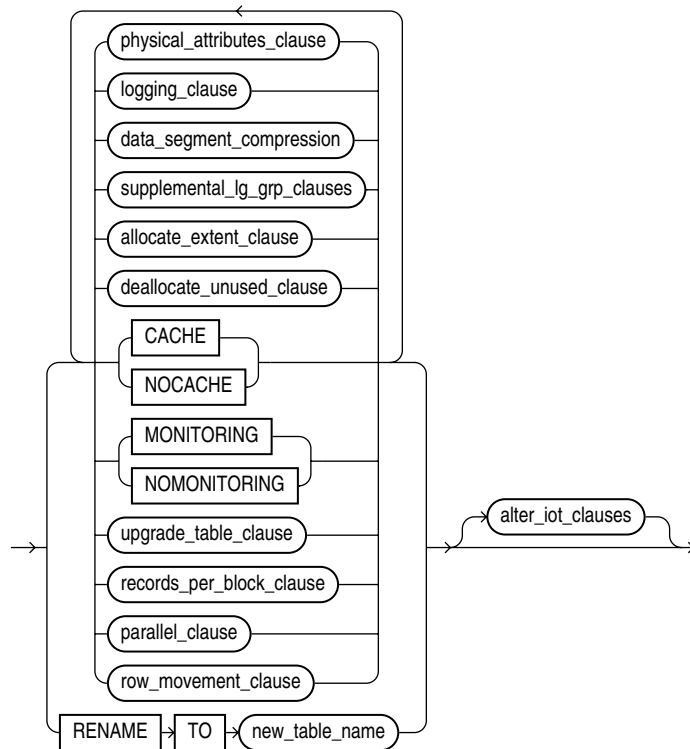
注意： *table* の後に句を指定する必要があります。必須の句はありませんが、1 つ以上の句を指定する必要があります。

ALTER TABLE 構文のグループは、次のとおりです。

- **alter_table_properties::=** (11-4 ページ)
- **column_clauses::=** (11-9 ページ)
- **constraint_clauses::=** (11-11 ページ)
- **alter_table_partitioning::=** (11-18 ページ)
- **alter_external_table_clause::=** (11-17 ページ)
- **move_table_clause::=** (11-30 ページ)
- **enable_disable_clause::=** (11-30 ページ)

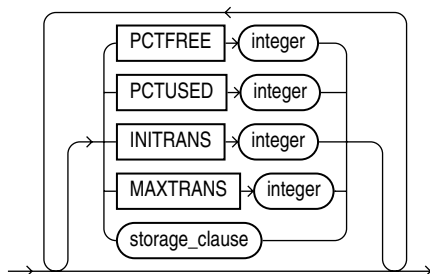
各句の後には、そのコンポーネントの副次句の参照先が記載されています。

alter_table_properties::=



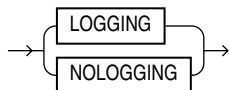
(11-5 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、11-5 ページの「[data_segment_compression::=](#)」、11-5 ページの「[supplemental_lg_grp_clauses::=](#)」、11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」、11-6 ページの「[upgrade_table_clause::=](#)」、11-6 ページの「[records_per_block_clause::=](#)」、11-6 ページの「[parallel_clause::=](#)」、11-6 ページの「[row_movement_clause::=](#)」、11-7 ページの「[alter_iot_clauses::=](#)」を参照)

physical_attributes_clause::=

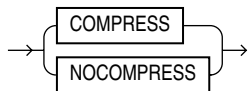


(7-51 ページの「[storage_clause](#)」を参照)

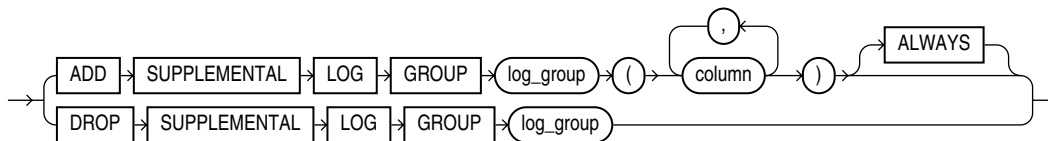
logging_clause::=



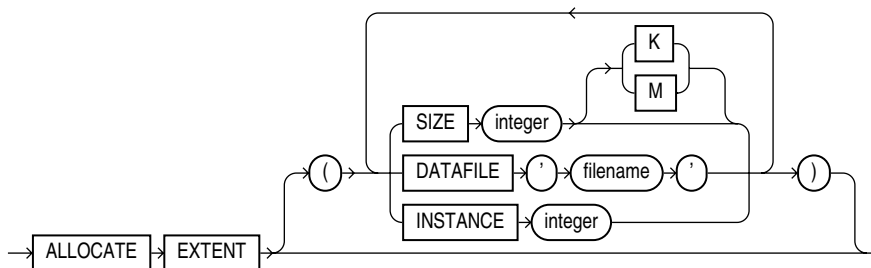
data_segment_compression::=



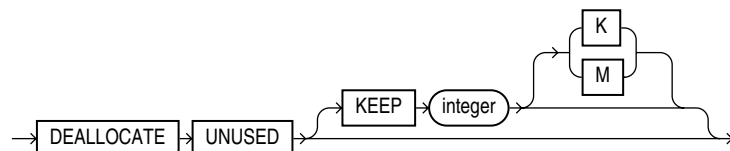
supplemental_lg_grp_clauses::=



allocate_extent_clause::=



```
deallocate_unused_clause::=
```

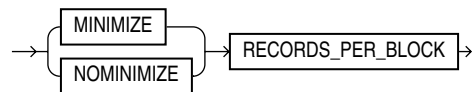


upgrade_table_clause::=

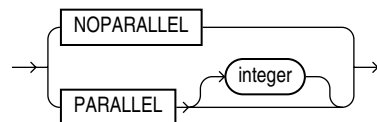


(11-12 ページの「[column_properties::=](#)」、11-15 ページの「[modify_LOB_storage_clause::=](#)」を参照)

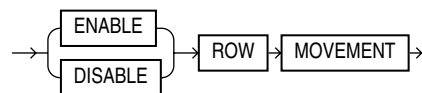
```
records_per_block_clause ::=
```



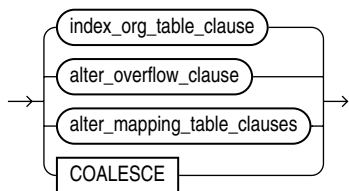
parallel_clause ::=



row_movement_clause::=

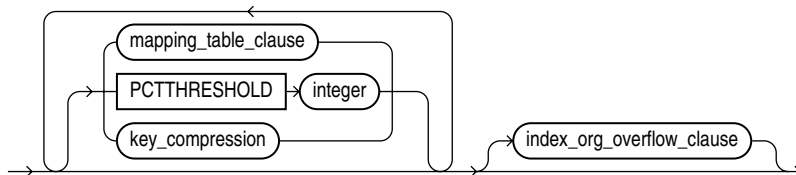


alter_iot_clauses::=

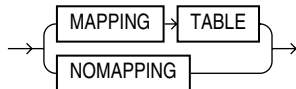


(11-8 ページの「[alter_overflow_clause::=](#)」、11-8 ページの「[alter_mapping_table_clauses::=](#)」を参照)

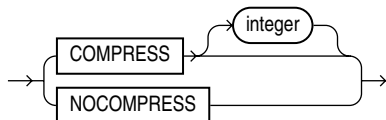
index_org_table_clause::=



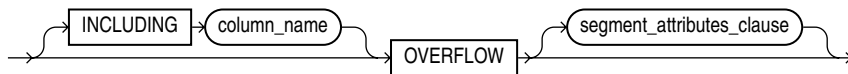
mapping_table_clause::=



key_compression::=

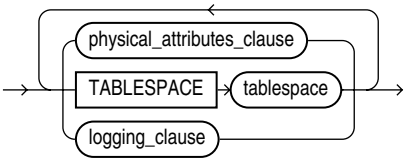


index_org_overflow_clause::=



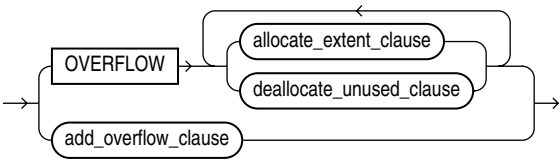
(11-8 ページの「[segment_attributes_clause::=](#)」を参照)

segment_attributes_clause::=



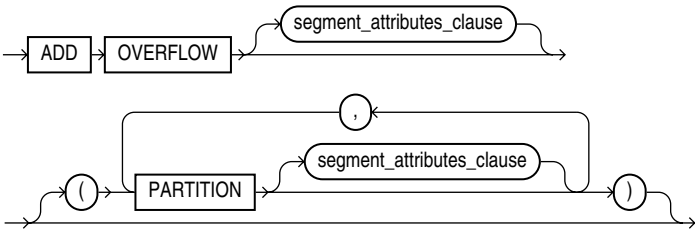
(11-5 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause](#)」を参照)

alter_overflow_clause::=



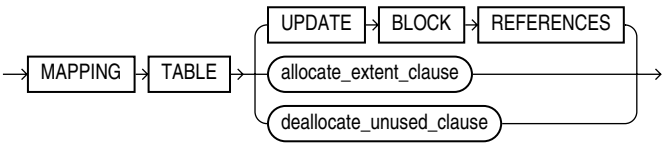
(11-8 ページの「[segment_attributes_clause::=](#)」、11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」を参照)

add_overflow_clause::=



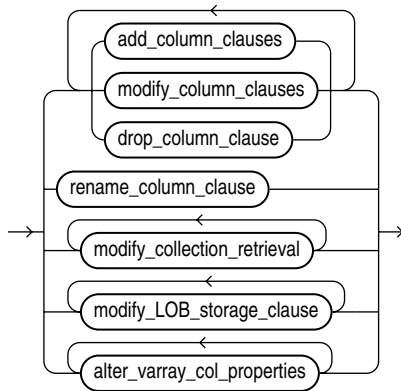
(11-8 ページの「[segment_attributes_clause::=](#)」を参照)

alter_mapping_table_clauses::=



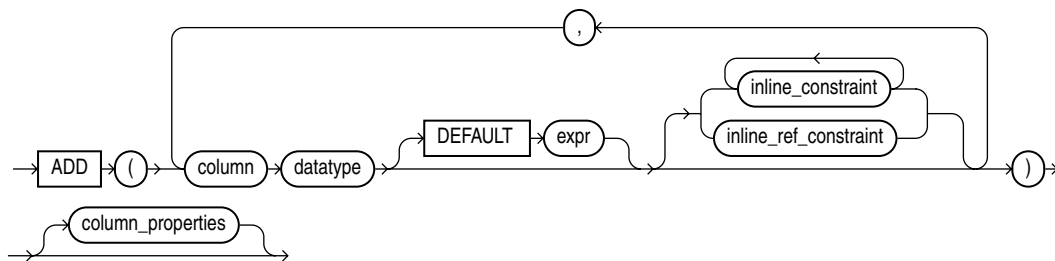
(11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」を参照)

column_clauses::=



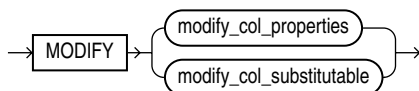
(11-9 ページの「[add_column_clause::=](#)」、11-9 ページの「[modify_column_clauses::=](#)」、11-10 ページの「[drop_column_clause::=](#)」、11-10 ページの「[rename_column_clause::=](#)」、11-11 ページの「[modify_collection_retrieval::=](#)」、11-15 ページの「[modify_LOB_storage_clause::=](#)」、11-16 ページの「[alter_varray_col_properties::=](#)」を参照)

add_column_clause::=

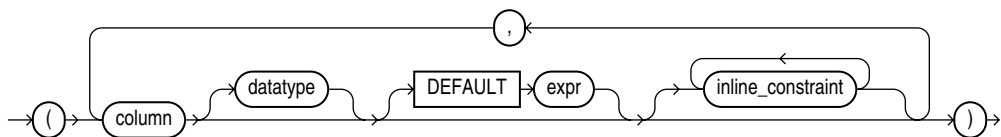


(7-5 ページの [constraints](#) の「[inline_constraint](#)」および「[inline_ref_constraint](#)」、11-12 ページの「[column_properties::=](#)」を参照)

modify_column_clauses::=



modify_col_properties::=

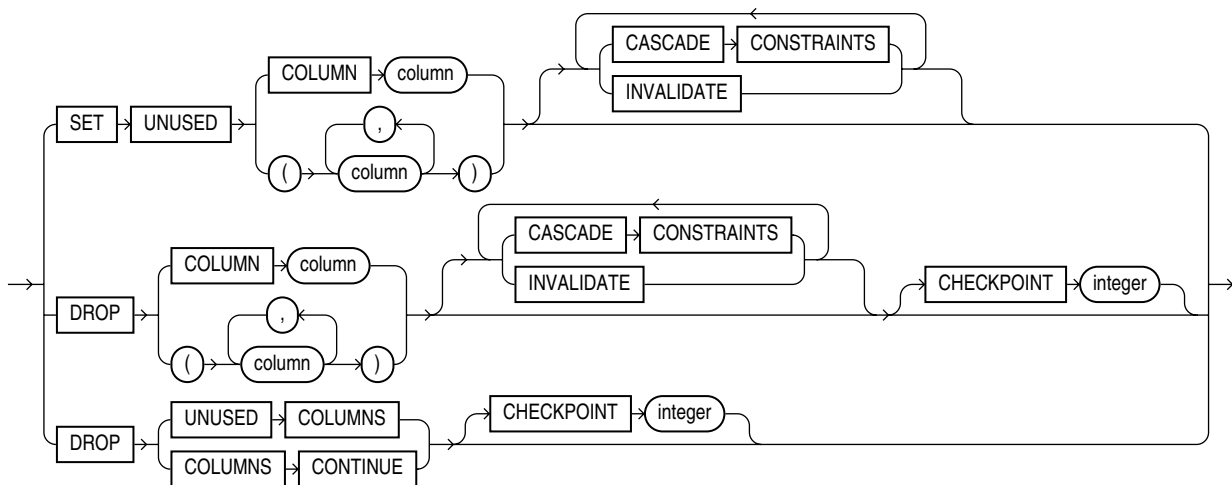


(7-5 ページの *constraints* の「*inline_constraint*」を参照)

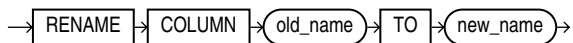
modify_col_substitutable::=



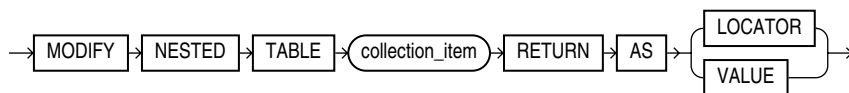
drop_column_clause::=



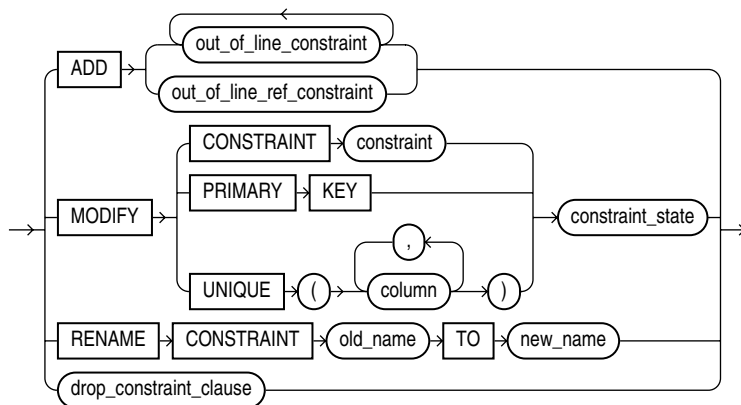
rename_column_clause::=



modify_collection_retrieval::=

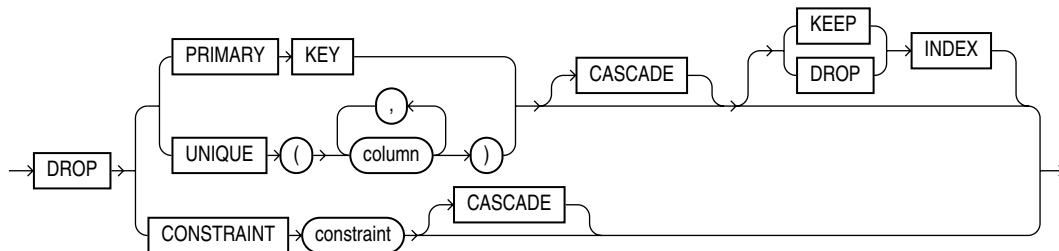


constraint_clauses::=

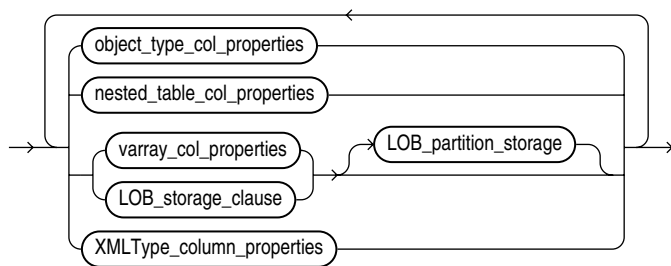


(7-5 ページの *constraints* の「*constraint_state:*」を参照)

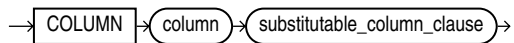
drop_constraint_clause::=



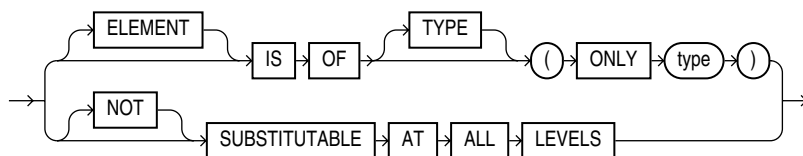
column_properties::=



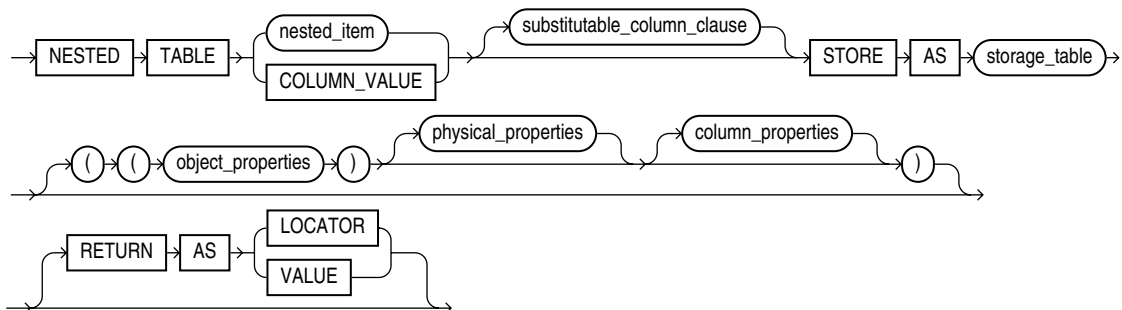
object_type_col_properties::=



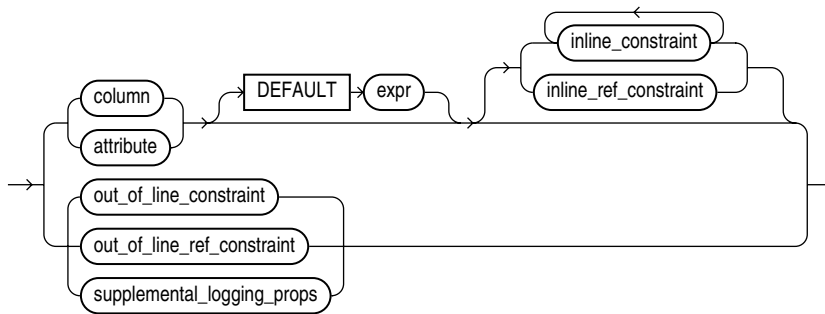
substitutable_column_clause::=



nested_table_col_properties::=

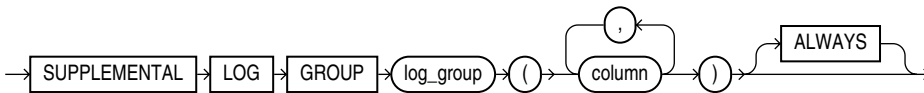


object_properties::=

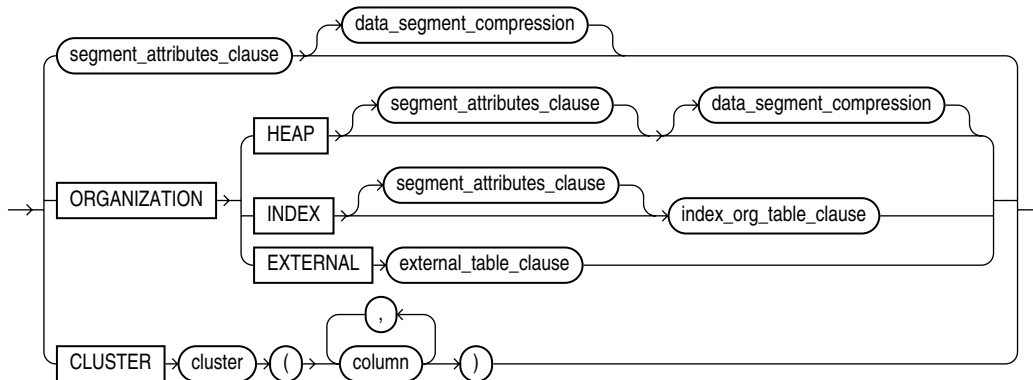


(7-5 ページの [constraints](#) の「[inline_constraint](#)」、「[inline_ref_constraint](#)」、「[out_of_line_constraint](#)」および「[out_of_line_ref_constraint](#)」を参照)

supplemental_logging_props::=

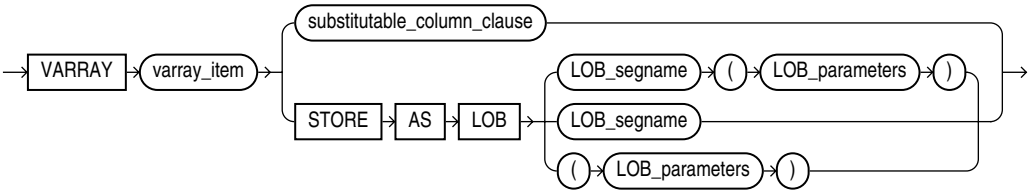


physical_properties::=



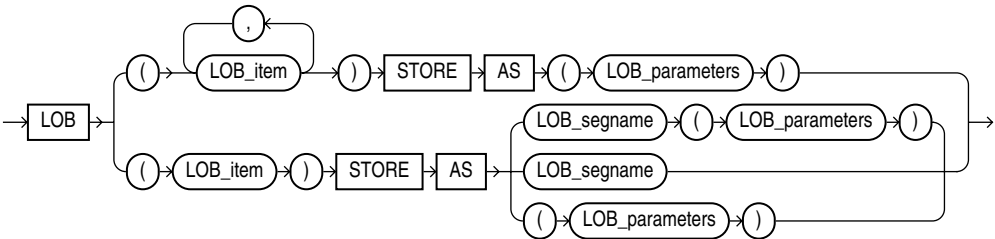
(11-8 ページの「[segment_attributes_clause::=](#)」、11-7 ページの「[index_org_table_clause::=](#)」、11-18 ページの「[external_data_properties::=](#)」を参照)

varray_col_properties::=

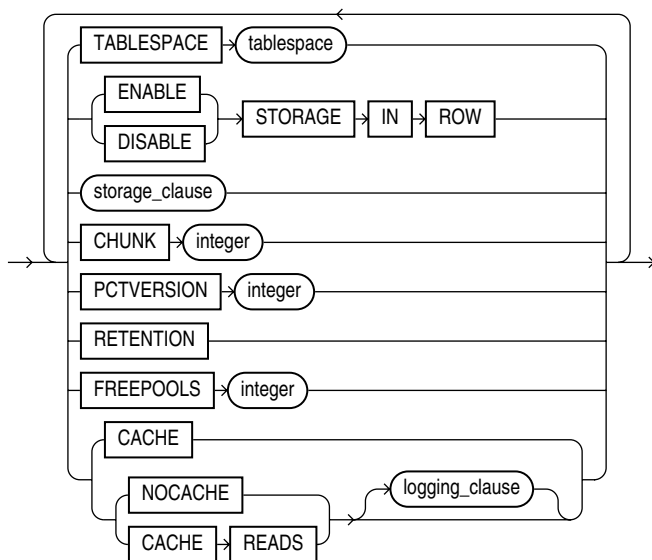


(11-12 ページの「`substitutable_column_clause::=`」を参照)

LOB_storage_clause::=

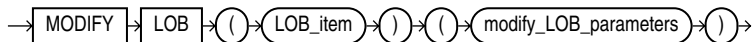


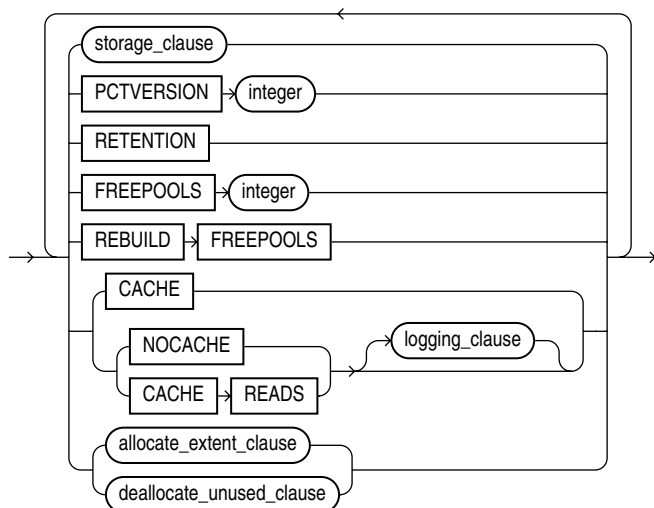
LOB_parameters::=



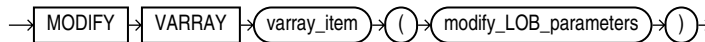
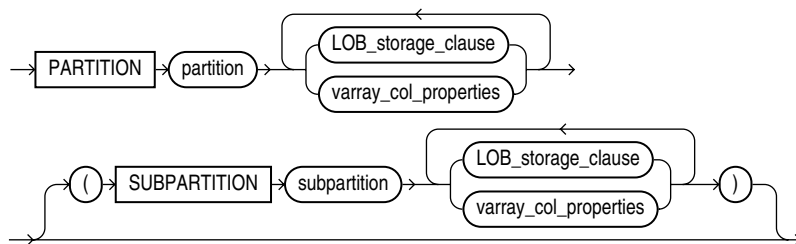
(7-53 ページの「[storage_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

modify_LOB_storage_clause::=

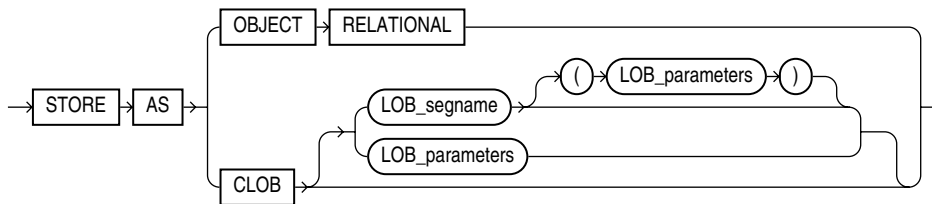
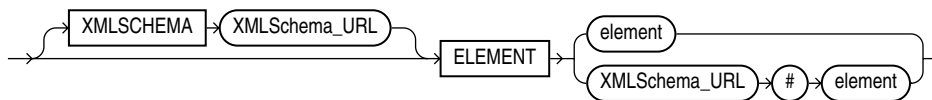
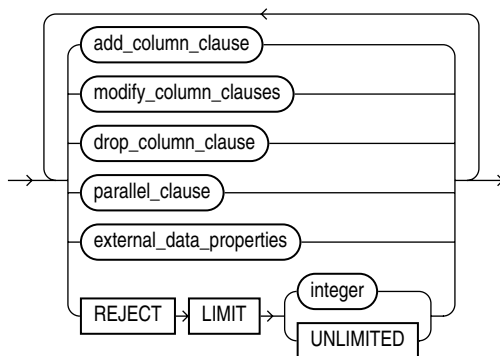


modify_LOB_parameters::=

(7-53 ページの「[storage_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、
11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの
「[deallocate_unused_clause::=](#)」を参照)

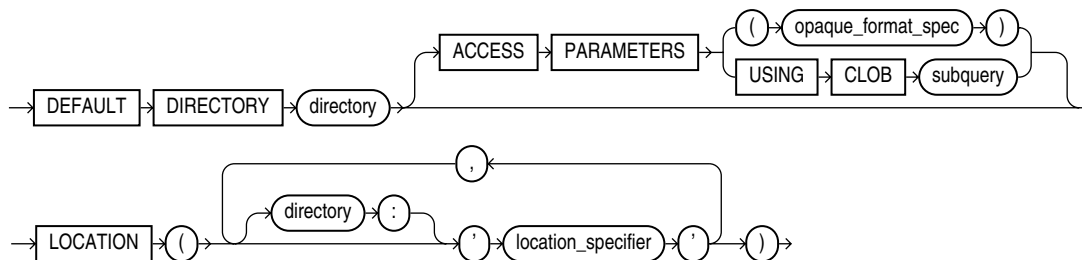
alter_varray_col_properties::=**LOB_partition_storage::=**

(11-14 ページの「[LOB_storage_clause::=](#)」、11-14 ページの
「[varray_col_properties::=](#)」を参照)

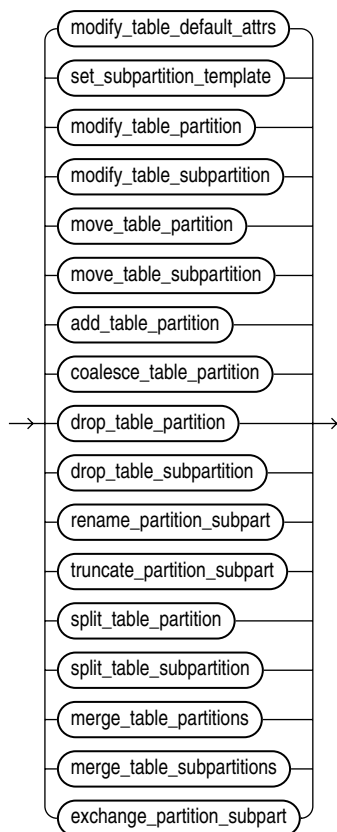
XMLType_column_properties::=**XMLType_storage::=****XMLSchema_spec::=****alter_external_table_clause::=**

(11-9 ページの「[add_column_clause::=](#)」、11-9 ページの「[modify_column_clauses::=](#)」、11-10 ページの「[drop_column_clause::=](#)」、11-11 ページの「[drop_constraint_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

external_data_properties::=

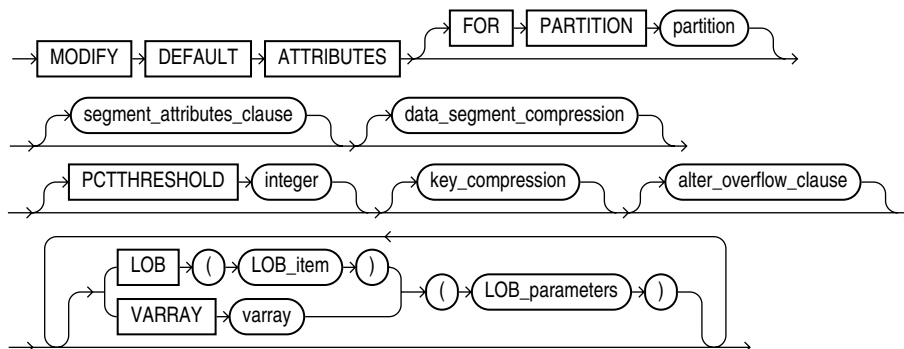


alter_table_partitioning::=



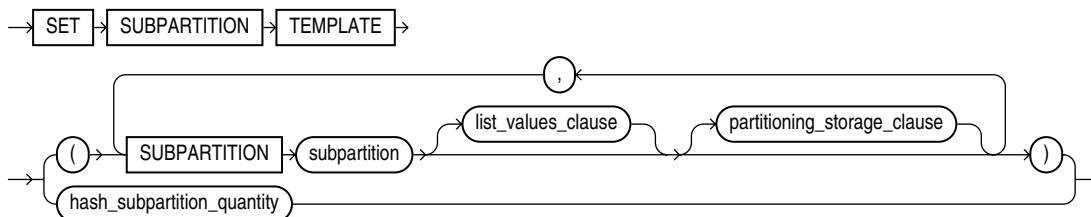
(11-19 ページの「[modify_table_default_attrs::=](#)」、11-19 ページの「[set_subpartition_template::=](#)」、11-20 ページの「[modify_table_partition::=](#)」、11-21 ページの「[modify_table_subpartition::=](#)」、11-21 ページの「[move_table_partition::=](#)」、11-21 ページの「[move_table_subpartition::=](#)」、11-22 ページの「[add_table_partition::=](#)」、11-22 ページの「[coalesce_table_partition::=](#)」、11-23 ページの「[drop_table_partition::=](#)」、11-23 ページの「[drop_table_subpartition::=](#)」、11-23 ページの「[rename_partition_subpart::=](#)」、11-23 ページの「[truncate_partition_subpart::=](#)」、11-24 ページの「[split_table_partition::=](#)」、11-24 ページの「[split_table_subpartition::=](#)」、11-24 ページの「[merge_table_partitions::=](#)」、11-25 ページの「[merge_table_subpartitions::=](#)」、11-25 ページの「[exchange_partition_subpart::=](#)」を参照)

[modify_table_default_attrs::=](#)



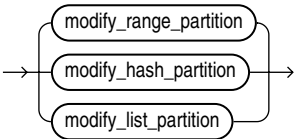
(11-8 ページの「[segment_attributes_clause::=](#)」、11-7 ページの「[key_compression::=](#)」、11-15 ページの「[LOB_parameters::=](#)」、11-8 ページの「[alter_overflow_clause::=](#)」を参照)

[set_subpartition_template::=](#)



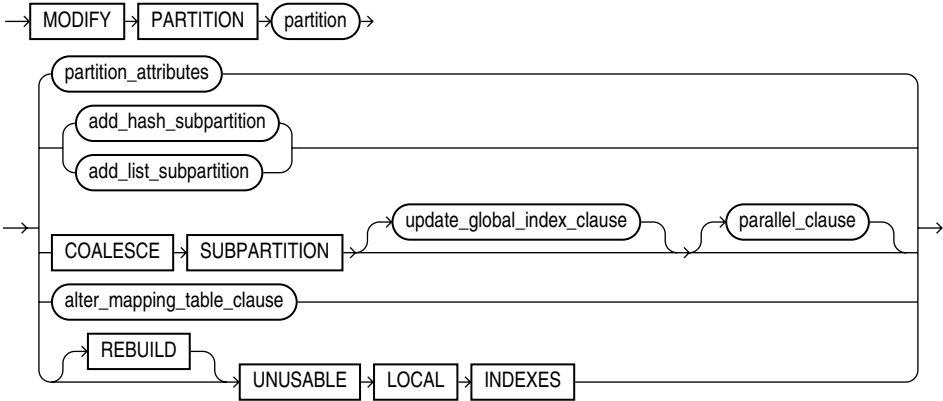
(11-25 ページの「[list_values_clause::=](#)」、11-26 ページの「[partitioning_storage_clause::=](#)」を参照)

modify_table_partition::=



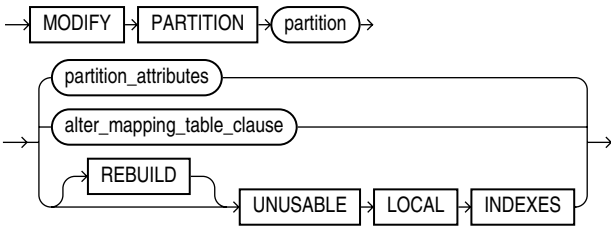
(11-20 ページの「[modify_range_partition::=](#)」、11-20 ページの「[modify_hash_partition::=](#)」、11-21 ページの「[modify_list_partition::=](#)」を参照)

modify_range_partition::=

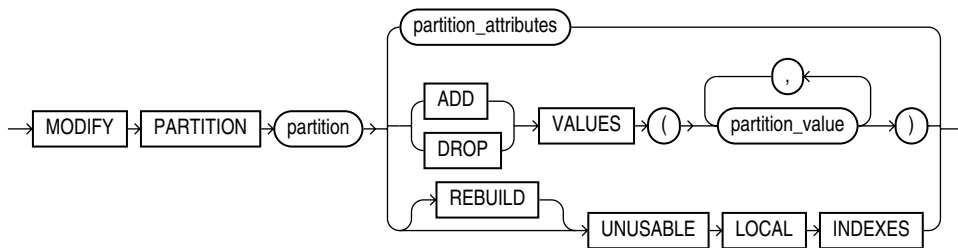


(11-26 ページの「[partition_attributes::=](#)」、11-27 ページの「[add_hash_subpartition::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」、11-8 ページの「[alter_mapping_table_clauses::=](#)」を参照)

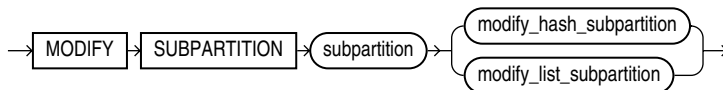
modify_hash_partition::=



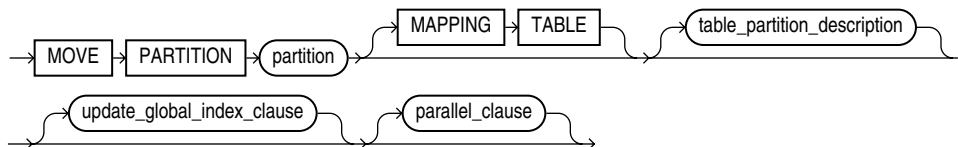
(11-26 ページの「[partition_attributes::=](#)」、11-8 ページの「[alter_mapping_table_clauses::=](#)」を参照)

modify_list_partition::=

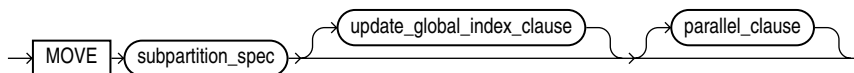
(11-26 ページの「[partition_attributes::=](#)」、11-27 ページの「[add_list_subpartition::=](#)」を参照)

modify_table_subpartition::=

(11-27 ページの「[modify_hash_subpartition::=](#)」、11-28 ページの「[modify_list_subpartition::=](#)」を参照)

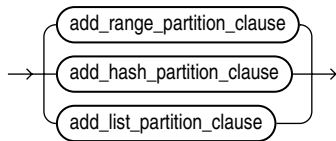
move_table_partition::=

(11-28 ページの「[table_partition_description::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

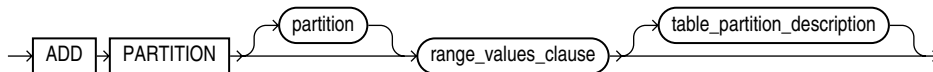
move_table_subpartition::=

(11-29 ページの「[subpartition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

add_table_partition::=

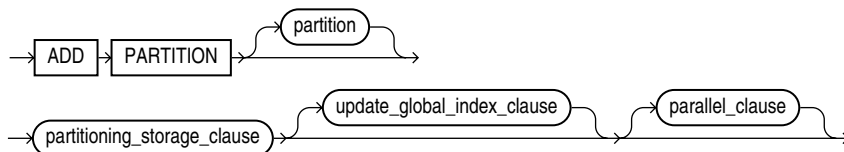


add_range_partition_clause::=



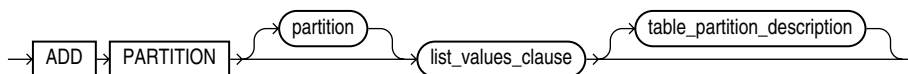
(11-26 ページの「[range_values_clause::=](#)」、11-28 ページの「[table_partition_description::=](#)」を参照)

add_hash_partition_clause::=



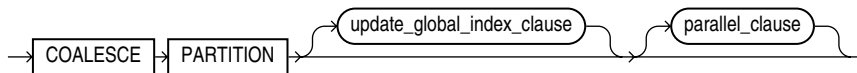
(11-26 ページの「[partitioning_storage_clause::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

add_list_partition_clause::=



(11-25 ページの「[list_values_clause::=](#)」、11-28 ページの「[table_partition_description::=](#)」を参照)

coalesce_table_partition::=



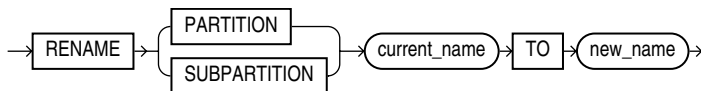
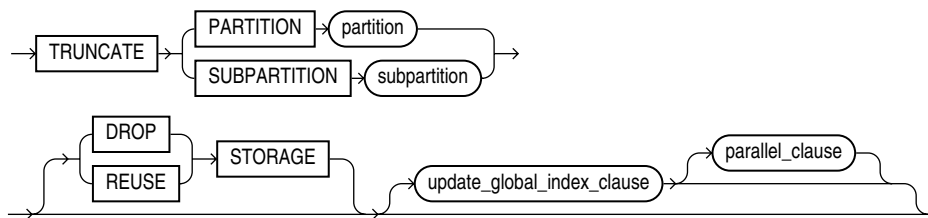
(11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

drop_table_partition::=

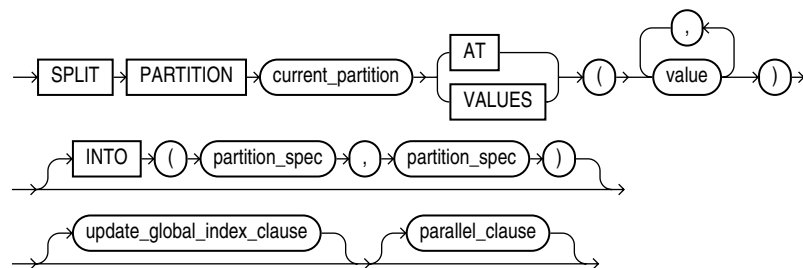
(11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

drop_table_subpartition::=

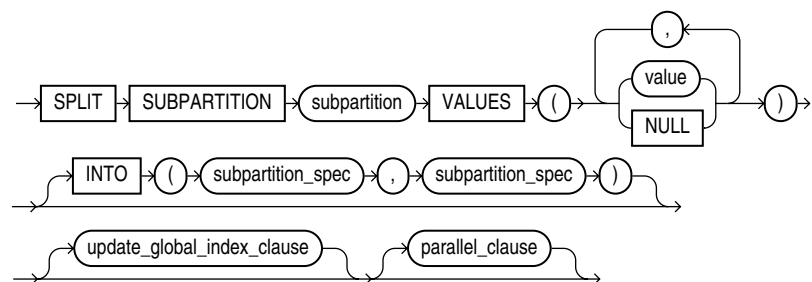
(11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

rename_partition_subpart::=**truncate_partition_subpart::=**

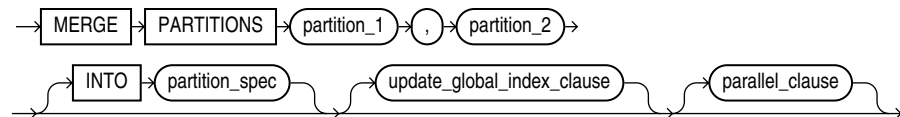
(11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

split_table_partition::=

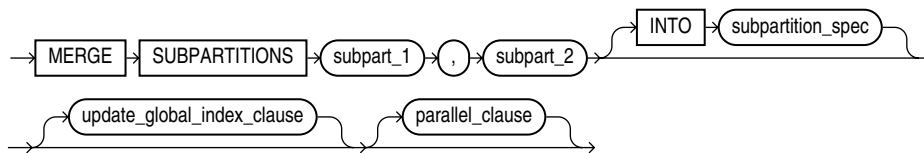
(11-29 ページの「[partition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

split_table_subpartition::

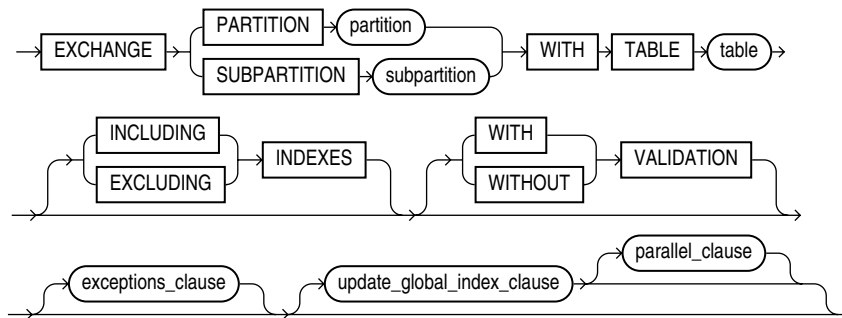
(11-29 ページの「[subpartition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

merge_table_partitions::=

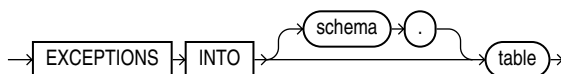
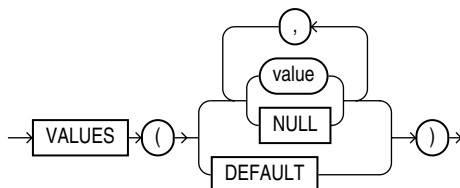
(11-29 ページの「[partition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

merge_table_subpartitions::=

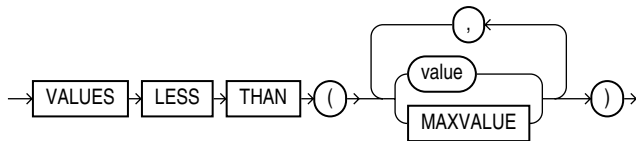
(11-29 ページの「[subpartition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

exchange_partition_subpart::=

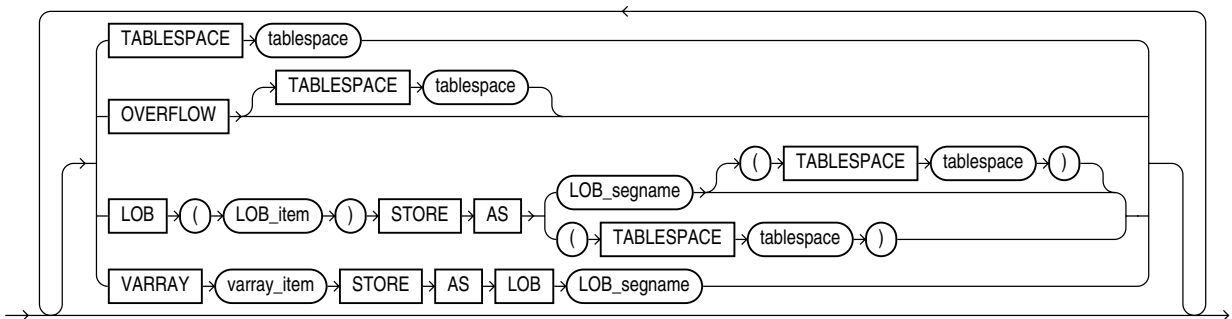
(11-29 ページの「[update_global_index_clause::=](#)」、11-29 ページの「[parallel_clause::=](#)」を参照)

exceptions_clause::=**list_values_clause::=**

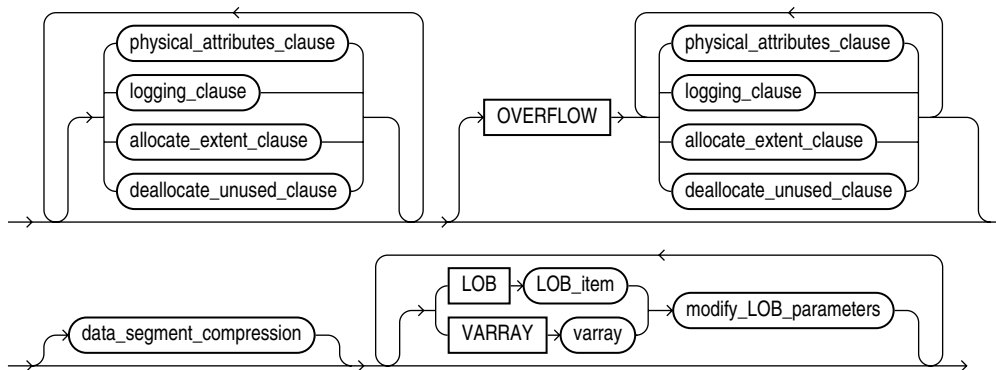
range_values_clause::=



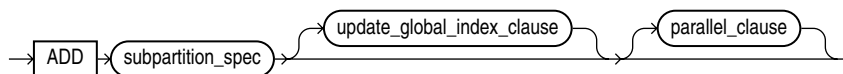
partitioning_storage_clause::=



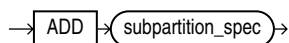
partition_attributes::=



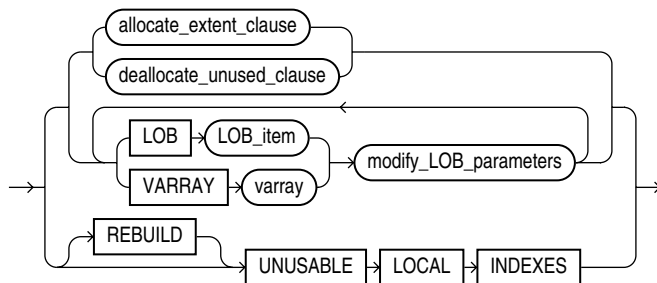
(11-5 ページの「[physical_attributes_clause::=](#)」、7-42 ページの「[logging_clause::=](#)」、11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」、11-5 ページの「[data_segment_compression::=](#)」、11-16 ページの「[modify_LOB_parameters::=](#)」を参照)

add_hash_subpartition::=

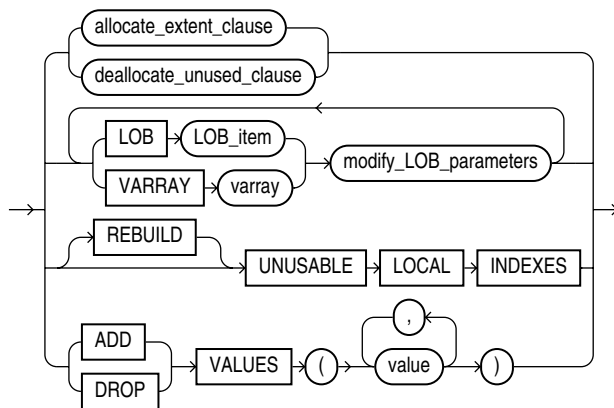
(11-29 ページの「[subpartition_spec::=](#)」、11-29 ページの「[update_global_index_clause::=](#)」を参照)

add_list_subpartition::=

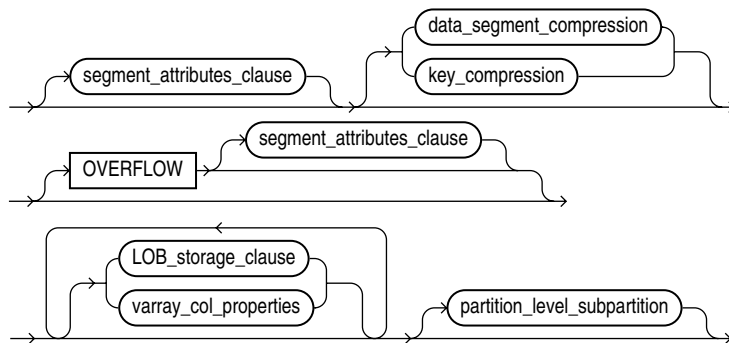
(11-29 ページの「[subpartition_spec::=](#)」を参照)

modify_hash_subpartition::=

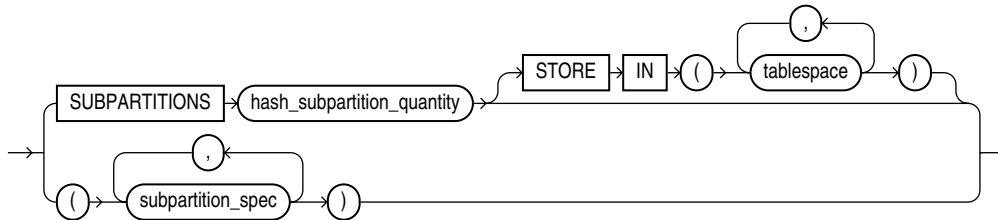
(11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」、11-16 ページの「[modify_LOB_parameters::=](#)」を参照)

modify_list_subpartition::=

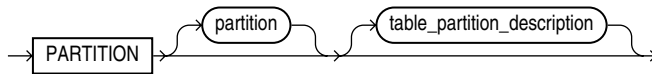
(11-5 ページの「[allocate_extent_clause::=](#)」、11-6 ページの「[deallocate_unused_clause::=](#)」、11-16 ページの「[modify_LOB_parameters::=](#)」を参照)

table_partition_description::=

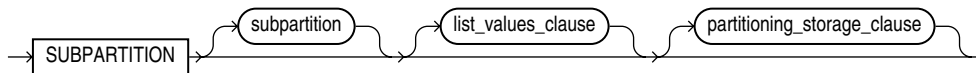
(11-8 ページの「[segment_attributes_clause::=](#)」、11-7 ページの「[key_compression::=](#)」、11-14 ページの「[LOB_storage_clause::=](#)」、11-14 ページの「[varray_col_properties::=](#)」を参照)

partition_level_subpartition::=

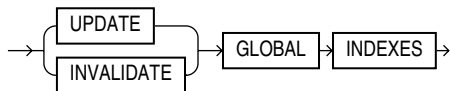
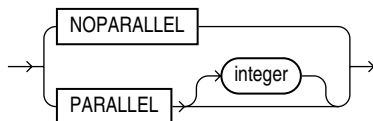
(11-29 ページの「[subpartition_spec::=](#)」を参照)

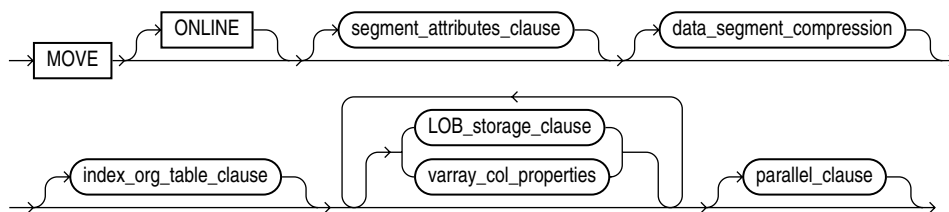
partition_spec::=

(11-28 ページの「[table_partition_description::=](#)」を参照)

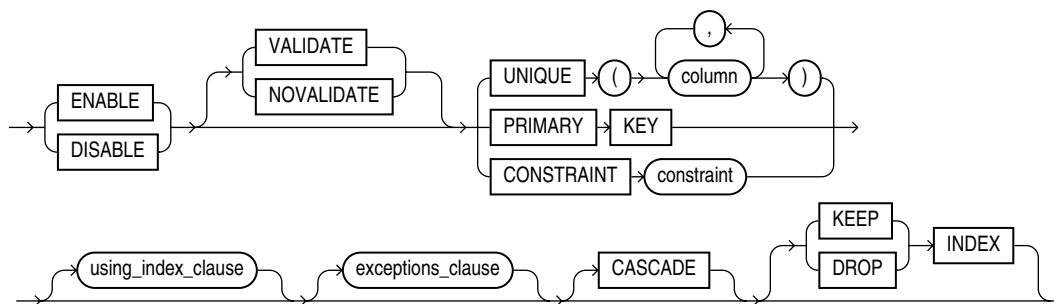
subpartition_spec::=

(11-25 ページの「[list_values_clause::=](#)」、11-26 ページの「[partitioning_storage_clause::=](#)」を参照)

update_global_index_clause::=**parallel_clause::=**

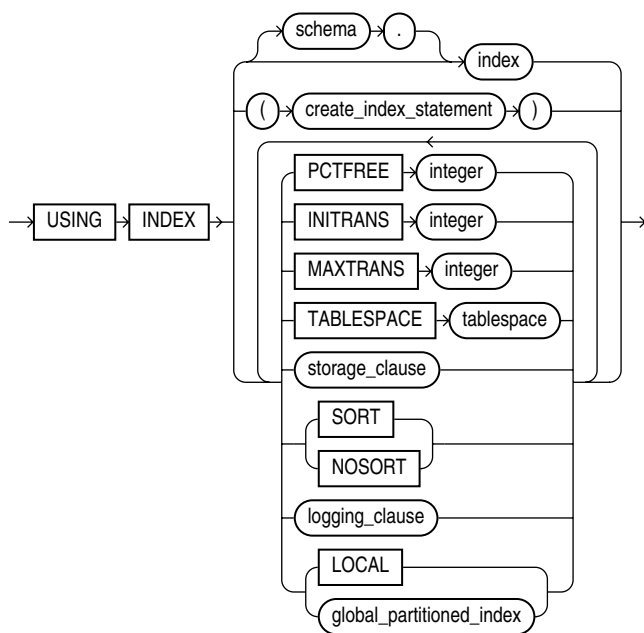
move_table_clause::=

(11-8 ページの「[segment_attributes_clause::=](#)」、11-7 ページの「[index_org_table_clause::=](#)」、11-14 ページの「[LOB_storage_clause::=](#)」、11-14 ページの「[varray_col_properties::=](#)」を参照)

enable_disable_clause::=

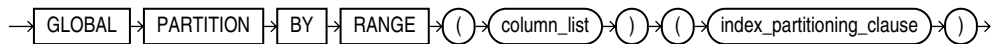
(11-31 ページの「[using_index_clause::=](#)」、11-25 ページの「[exceptions_clause::=](#)」を参照)

using_index_clause::=

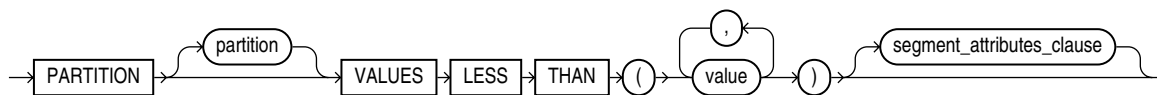


(13-62 ページの「[create_index::=](#)」、7-51 ページの「[storage_clause](#)」、7-42 ページの「[logging_clause::=](#)」を参照)

global_partitioned_index::=



index_partitioning_clause::=



セマンティクス

ALTER TABLE 文の多くの句の機能は、CREATE TABLE 文での機能と同じです。これらの句の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

注意： ALTER TABLE 文を実行すると、変更した表にアクセスするプロシージャおよびストアド・ファンクションが無効になる場合があります。無効になる条件および具体的な状況については、『Oracle9i データベース概要』を参照してください。

schema

変更する表が定義されているスキーマを指定します。 *schema* を指定しない場合、この表が自分のスキーマに存在するとみなされます。

table

変更する表の名前を指定します。

一時表の制限事項：

一時表の変更、一時表内の列の削除または一時表の名前の変更を実行できます。ただし、一時表に対して次のことはできません。

- ネストした表または VARRAY 型の列の追加、その他の型の列の追加
- 追加または変更された列の参照整合性（外部キー）制約の指定
- 追加または変更された LOB 列に対する次の *LOB_storage_clause* の句の指定：
TABLESPACE、*storage_clause*、*logging_clause*、*LOB_index_clause*
- *physical_attributes_clause*、*nested_table_col_properties*、*parallel_clause*、*allocate_extent_clause*、*deallocate_unused_clause* または索引構成表（IOT）句の指定
- パーティション表と一時表の間でのパーティションの交換
- *logging_clause* の指定
- MOVE の指定

外部表の制限事項：

外部表の列を追加、削除または変更できます。ただし、外部表に対して次のことはできません。

- LONG、LOB またはオブジェクト型の列の追加、または外部表の列のデータ型をこれらのデータ型のいずれかに変換

- 外部表への制約の追加
- 外部表の記憶域パラメータの変更
- `logging_clause` の指定
- MOVE の指定

注意： 1つ以上のマテリアライズド・ビューのマスター表となっている表を変更した場合、マテリアライズド・ビューには `INVALID` のマークが付けられます。無効なマテリアライズド・ビューは、クエリー・リライトによって使用できません。また、リフレッシュすることもできません。マテリアライズド・ビューを再検証する場合は、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。

参照： マテリアライズド・ビューに関する一般的な情報については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

alter_table_properties

`alter_table_clauses` を使用すると、データベースの表を変更できます。

physical_attributes_clause

`physical_attributes_clause` は、`PCTFREE`、`PCTUSED`、`INITRANS` および `MAXTRANS` パラメータの値と記憶特性を変更します。

表の物理属性の変更の制限事項：

- `PCTUSED` パラメータは、索引構成表（IOT）の索引セグメントに対して指定できません。
- ローカル管理表領域の表の記憶域属性を変更しようとすると、エラーが発生します。ただし、ローカル管理表領域にパーティション表のセグメントがあり、ディクショナリ管理表領域に他のセグメントがある場合、ディクショナリ管理表領域のセグメントの記憶域属性は変更されますが、ローカル管理表領域のセグメントの記憶域属性は変更されません。また、エラーも発生しません。
- 自動セグメント領域管理のセグメントの場合は、`PCTUSED` 設定の変更は無視されます。`PCTFREE` 設定を変更した場合、その変更をセグメントに割当て済のブロックに実装するには、`DBMS_REPAIR.segment_fix_status` プロシージャを実行する必要があります。

注意：

- 非パーティション表の場合、作成時に表に指定した値は新しく指定した値によって上書きされます。
 - レンジ・パーティション表、リスト・パーティション表またはハッシュ・パーティション表の場合、新たに指定した値がその表のデフォルト値およびすべての既存パーティションに対する実際の値となり、そのパーティションにすでに設定されていた値は上書きされます。既存のパーティション値を上書きせずにデフォルトの表属性を変更する場合は、`modify_table_default_attrs` を使用してください。
 - コンポジット・パーティション表の場合、新しく指定した値がその表とその表のすべてのパーティションのデフォルト値、およびその表のすべてのサブパーティションに対する実際の値となり、そのサブパーティションにすでに設定されていた値は上書きされます。既存のサブパーティションの値を上書きせずにデフォルトのパーティション属性を変更する場合は、FOR PARTITION 句とともに `modify_table_default_attrs` を使用してください。
-

参照：

- 物理属性パラメータの詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- 記憶域パラメータの詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

data_segment_compression

`data_segment_compression` 句は、ヒープ構成表にのみ有効です。この句を使用すると、ディスクおよびメモリの使用量を削減するために、データ・セグメントを圧縮するかどうかを指定できます。COMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用可能になります。NOCOMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用禁止になります。

注意： 初めて、圧縮データが追加されるように表を変更する場合は、表のすべてのビットマップ索引およびビットマップ索引パーティションに UNUSABLE のマークを付ける必要があります。

参照：

- 圧縮率の計算の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。また、データ圧縮の使用例は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- データ・セグメントの圧縮を使用したオブジェクトの作成方法の詳細は、15-29 ページの CREATE TABLE の「[data_segment_compression](#)」を参照してください。

logging_clause

非パーティション表、表パーティション、パーティション表のすべてのパーティションまたはパーティションのすべてのサブパーティションに対する後続のダイレクト・ローダー (SQL*Loader) およびダイレクト・パス・インサート操作について、REDO ログ・ファイルにログを記録するか (LOGGING)、記録しないか (NOLOGGING) を指定します。

modify_table_default_attrs 句とともにこの句を使用した場合、パーティション表のロギング属性が影響を受けます。

また、*logging_clause* は、ALTER TABLE ... MOVE および ALTER TABLE ... SPLIT 操作のログを記録するかどうかも指定します。

参照：

- この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。
- *logging_clause* およびパラレル DML の詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

supplemental_lg_grp_clauses

supplemental_lg_grp_clauses を使用すると、補助 REDO ログ・グループを追加および削除できます。

- ADD LOG GROUP 句を使用すると、REDO ログ・グループを追加できます。
- DROP LOG GROUP 句を使用すると、必要でなくなったときに REDO ログ・グループを削除できます。

参照： 補助 REDO ログ・グループの詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

allocate_extent_clause

allocate_extent_clause を使用すると、表、パーティション、サブパーティション、オーバーフロー・データ・セグメント、LOB データ・セグメントまたは LOB 索引に新しいエクステントを明示的に割り当てることができます。

表のエクステントの割当ての制限事項： 一時表、レンジ・パーティション表またはコンポジット・パーティション表にはエクステントを割り当てることはできません。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。また、11-92 ページの「[エクステントの割当て例:](#)」を参照してください。

deallocate_unused_clause

deallocate_unused_clause を使用すると、表、パーティション、サブパーティション、オーバフロー・データ・セグメント、LOB データ・セグメントまたは LOB 索引の最後にある未使用領域の割当てを明示的に解除し、解放された領域を表領域の他のセグメントから利用できます。

参照： この句の詳細は、7-34 ページの「[deallocate_unused_clause](#)」を参照してください。また、11-87 ページの「[未使用領域の解放例:](#)」を参照してください。

CACHE | NOCACHE

CACHE 句を使用して、バッファ・キャッシュ内でのブロックの格納方法を指定します。次に、CACHE または NOCACHE も指定しない場合の例を示します。

- CREATE TABLE 文では、NOCACHE がデフォルトです。
- ALTER TABLE 文では、既存の値は変更されません。

CACHE 句 この句は、アクセス頻度の高いデータについて、全表スキャンの実行時に、この表のために取り出されたブロックを、バッファ・キャッシュ内の最低使用頻度 (LRU) リストの最高使用頻度側に置く場合に指定します。この属性は、小規模な参照表で有効です。

LOB_storage_clause のパラメータとして CACHE を使用すると、より高速なアクセスのために、LOB データ値がバッファ・キャッシュに配置されます。

CACHE の制限事項： CACHE は、索引構成表に対して指定できません。ただし、索引構成表は暗黙的に CACHE 動作を提供します。

NOCACHE 句 この句は、アクセス頻度の低いデータについて、全表スキャンの実行時に、この表のために取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最低使用頻度側に置く場合に指定します。

LOB_storage_clause のパラメータとして NOCACHE を指定すると、LOB 値がバッファ・キャッシュに入れられないか、またはバッファ・キャッシュに入れられ、LRU リストの最低使用頻度側に置かれるかのいずれかを指定できます。(デフォルトでは後者です)。

NOCACHE の制限事項： NOCACHE は、索引構成表に対して指定できません。

MONITORING | NOMONITORING

MONITORING 句 表の変更統計を収集する場合は、MONITORING を指定します。これらの統計表は、一定の期間に DML 文の影響を受ける行数の推定値です。これらは、オブティマイザが使用またはユーザーが分析する場合に使用できます。

参照： この句の使用の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

NOMONITORING 句 表の変更統計を収集しない場合は、NOMONITORING を指定します。

MONITORING の制限事項： MONITORING または NOMONITORING は、一時表に対して指定できません。

upgrade_table_clause

upgrade_table_clause は、オブジェクト表およびオブジェクト列を含むリレーショナル表に関連します。この句を使用すると、ターゲットとなる表のメタデータが参照される各型の最新バージョンに準拠する型に変換されます。表が有効な場合、表のメタデータは変更されないままとなります。

オブジェクト表およびオブジェクト列のアップグレードの制限事項： この句の中では、*object_type_col_properties* を *column_properties* 句として指定できません。

INCLUDING DATA 表のデータを型の最新バージョンの形式に変更する場合は、INCLUDING DATA を指定します（型が変更されたときに変換されなかった場合）。*column_properties* および *LOB_partition_storage* を使用して表をアップグレードする間に、新しいすべての列の記憶域を定義できます。これはデフォルト値です。

表が古いバージョンの型に基づくデータを含むかどうかを確認するには、USER_TAB_COLUMNS データ・ディクショナリ・ビューの DATA_UPGRADED 列を参照します。

NOT INCLUDING DATA 列データを変更しないままにする場合は、NOT INCLUDING DATA を指定します。

NOT INCLUDING DATA の制限事項： 表が Oracle8 リリース 8.0.x のイメージ・フォーマットを含む場合、NOT INCLUDING DATA を指定できません。表がこのような列を含むかどうかを確認するには、USER_TAB_COLUMNS データ・ディクショナリ・ビューの V80_FMT_IMAGE 列を参照します。

参照：

- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 表が依存する型を変更する際に、依存表のデータを変換する場合の詳細は、12-6 ページの「[ALTER TYPE](#)」を参照してください。
- 表のデータを型の最新バージョンの形式に変換しないことによる影響の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

records_per_block_clause

1 ブロックに格納できるレコード数を制限するかどうかを指定する場合は、*records_per_block_clause* を使用します。この句によって、この後、表に作成されるビットマップ索引はできるだけ小さくなり（圧縮され）ます。

ブロック内のレコードの制限事項：

- 表にすでにビットマップ索引が定義されている場合は、MINIMIZE または NOMINIMIZE のいずれも指定できません。まず、ビットマップ索引を削除する必要があります。
- この句は、索引構成表またはネストした表に対して指定できません。

MINIMIZE 表の各ブロックの最大レコード数を計算し、ブロックに含まれるレコード数がその数を超えないように挿入操作を制限するために、MINIMIZE を指定します。

MINIMIZE を指定する前に、表にデータのサンプル・セットを定義しておくことをお薦めします。データ・セグメント圧縮を使用している場合（11-34 ページの「[data_segment_compression](#)」を参照）、圧縮データのサンプル・セットは、すでに表に存在している必要があります。

MINIMIZE の制限事項： MINIMIZE は、空の表に対して指定できません。

NOMINIMIZE MINIMIZE 機能を無効にするために、NOMINIMIZE を指定します。これはデフォルト値です。

RENAME TO

表の名前を *new_table_name* に変更する場合は、RENAME 句を使用します。

表名の変更の制限事項： マテリアライズド・ビューの名前は変更できません。

注意： この句を使用した場合、依存するすべてのマテリアライズド・ビューは無効になります。マテリアライズド・ビューの詳細は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

row_movement_clause

row_movement_clause を使用して、表の列を移動できます。データ・セグメントの圧縮時や、パーティション・データの更新操作時などに、列を移動できます。

注意： データ・アクセスで静的な ROWID が必要な場合は、行移動を有効にしないでください。通常の表（ヒープ構成表）の場合は、行を移動すると、その行の ROWID が変更されます。索引構成表での行の移動の場合は、論理 ROWID の物理推測コンポーネントは不正確になりますが、論理 ROWID は有効のままです。

- 行を移動するには、ENABLE を指定します。このとき、ROWID は変更されます。
- 行を移動しない場合は、DISABLE を指定します。このとき、ROWID は変更されません。

行の移動の制限事項： この句は、パーティション化された索引構成表に対して指定できません。

alter_iot_clauses

index_org_table_clause

詳細は、15-30 ページの「CREATE TABLE」の *index_org_table_clause* を参照してください。

参照： 11-88 ページの「[索引構成表の変更例:](#)」を参照してください。

alter_overflow_clause

alter_overflow_clause を使用して、索引構成表の定義を変更します。索引構成表は、主キーによってソートされたデータを保持する表であり、主キーに基づくアクセスおよび操作には最適です。

注意： 索引構成表に列を追加した場合、各列の最大サイズが評価され、行の最大値が計算されます。オーバーフロー・セグメントが必要で、OVERFLOW を指定していない場合は、エラーが発生し ALTER TABLE 文は実行されません。このチェック機能によって、索引構成表に対する後続の DML 操作が、オーバーフロー・セグメントがないために失敗することを防げます。

PCTTHRESHOLD integer 索引ブロック内で、索引構成表の行を格納するために確保されている領域の割合（パーセント）を指定します。PCTTHRESHOLD は、主キーを保持するために十分な大きさである必要があります。指定したしきい値を超える列から始まる行の後続列はすべて、オーバーフロー・セグメントに格納されます。PCTTHRESHOLD は 1 ～ 50 の値を取る必要があります。PCTTHRESHOLD を指定しない場合のデフォルト値は 50 です。

PCTTHRESHOLD の制限事項： PCTTHRESHOLD は、索引構成表の個別パーティションに対して指定できません。

INCLUDING column_name 索引構成表の行を索引部分とオーバーフロー部分に分割する列を指定します。主キー列は常に索引に格納されます。column_name は、最後の主キー列でもその他の主キー以外の列でもかまいません。column_name に続くすべての主キー以外の列は、オーバーフロー・データ・セグメントに格納されます。

INCLUDING 句の制限事項： 索引構成表の個々のパーティションにこの句は指定できません。

注意： column_name で行を分割しようとした場合に、行の索引部分のサイズが、PCTTHRESHOLD の値（指定またはデフォルト）を超えると、Oracle は PCTTHRESHOLD の値に基づいて、行を切り離します。

overflow_attributes 索引構成表に対して、変更するオーバーフロー・データ・セグメントの物理記憶域属性およびロギング属性を指定する場合に、overflow_attributes を使用します。この句に指定するパラメータは、オーバーフロー・データ・セグメントにのみ適用できます。

参照： 15-7 ページの「[CREATE TABLE](#)」を参照してください。

add_overflow_clause 指定した索引構成表にオーバーフロー・データ・セグメントを追加する場合に、add_overflow_clause を使用します。また、この句を使用すると、エクステンツを明示的に割り当てたり、既存のオーバーフロー・セグメントから未使用領域の割当てを解除することができます。

STORE IN *tablespace* 句を使用すると、オーバーフロー・セグメント全体に対する表領域の記憶域を指定できます。PARTITION 句を使用すると、パーティションによるセグメントに対する表領域の記憶域を指定できます。

パーティション化された索引構成表の場合、次の点に注意してください。

- PARTITION を指定しない場合、それぞれのパーティションに自動的にオーバーフロー・セグメントが割り当てられます。これらのセグメントの物理属性は表のレベルから継承されます。
- 1 つ以上のパーティションに別々の物理属性を指定する場合、表の中のパーティションそれぞれに個別の属性を指定する必要があります。パーティションの名前を指定する必要はありませんが、パーティションが作成された順番で属性を指定する必要があります。

パーティションの順番は、USER_IND_PARTITIONS ビューの PARTITION_NAME および PARTITION_POSITION 列の問合せから得ることができます。

TABLESPACE を指定していないパーティションがある場合、表に対して指定された表領域が使用されます。表レベルで TABLESPACE を指定していない場合は、そのパーティションの主キー索引セグメントの表領域が使用されます。

参照： `add_overflow_clause` 句の詳細は、7-2 ページの「[allocate_extent_clause](#)」および 7-34 ページの「[deallocate_unused_clause](#)」を参照してください。

alter_mapping_table_clauses

`alter_mapping_table_clauses` は、*table* が索引構成されており、マッピング表を持つ場合にのみ有効です。

UPDATE BLOCK REFERENCES UPDATE BLOCK REFERENCES を指定すると、主キーによって識別されるブロックの適切なデータベース・アドレスを持つマッピング表の論理 ROWID 列の一部として格納され、失効したと推測されるすべてのデータ・ブロック・アドレスを更新することができます。

allocate_extent_clause `allocate_extent_clause` を使用して、新しいエクステントを索引構成表のマッピング表の終わりに割り当てます。

参照： この句の詳細は、7-2 ページの「[allocate_extent_clause](#)」を参照してください。

deallocate_unused_clause `deallocate_unused_clause` を指定すると、索引構成表のマッピング表の終わりにある未使用領域の割当てを解除できます。

参照： この句の詳細は、7-34 ページの「[deallocate_unused_clause](#)」を参照してください。

COALESCE

このキーワードは *table* が索引構成されている場合のみに関連します。COALESCE を指定すると、空きブロックの再利用が可能な索引構成表の主キー索引ブロックを結合できます。この句は *parallel_clause* とあわせて指定できます。

column_clauses

add_column_clause

add_column_clause を使用すると、表に列を追加できます。

参照： この句のキーワードとパラメータの詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。また、11-92 ページの「[表の列の追加例 :](#)」を参照してください。

列を追加した場合、DEFAULT 句を指定しないかぎり、新しい列の各行には初期値として NULL が設定されます。この場合、新しい列の各行は、DEFAULT で指定した値で更新されます。この更新操作によって、表に定義された AFTER UPDATE トリガーが起動します。

注意： 列にデフォルト値がある場合、DEFAULT 句を使用してデフォルトを NULL に変更することができますが、デフォルト値を完全に削除することはできません。つまり、列に割り当てられたデフォルト値がある場合、USER_TAB_COLUMNS データ・ディクショナリ・ビューの DATA_DEFAULT 列にはデフォルト値または NULL のいずれかが表示されます。

パーティション化された索引構成表の各パーティションには、オーバーフロー・データ・セグメントを追加できます。

非パーティションおよびパーティション表に LOB 列を追加できます。表、およびパーティションまたはサブパーティションのレベルで LOB 記憶域を指定できます。

「SELECT *」構文を使用して、表からすべての列を選択するように指定した問合せを使用してビューを作成した場合、*table* に列を追加しても、新しい列がビューに自動的に追加されることはありません。ビューに新しい列を追加する場合、CREATE VIEW 文に OR REPLACE 句を指定してビューを再作成してください。

参照： 16-38 ページの「[CREATE VIEW](#)」を参照してください。

列の追加の制限事項：

- クラスタ化表には、LOB 列を追加できません
- LOB 列をハッシュ・パーティション表に追加する場合、新しいパーティションに対して指定できる属性は、TABLESPACE のみです。

- *table* に行がある場合、DEFAULT 句を指定しないかぎり、NOT NULL 制約のある列を追加できません。
- 索引構成表にこの句を指定した場合、同じ文では他の句を指定できません。

DEFAULT

新しい列にデフォルト値を指定する場合、または既存の列に新しいデフォルト値を指定する場合は、DEFAULT 句を使用します。後続の INSERT 文で列に値を指定しない場合、この値が自動的に割り当てられます。表に新しい列を追加する場合に、デフォルト値を指定した場合、その表のすべての行にデフォルトの列値が挿入されます。

デフォルト値のデータ型は、列に対して指定したデータ型と一致している必要があります。また、列には、このデフォルト値を保持できる長さが必要です。

列のデフォルト値の制限事項：

- DEFAULT 式に、他の列、つまり疑似列 CURRVAL、NEXTVAL、LEVEL および ROWNUM、または完全には指定されていない日付定数に対する参照を指定することはできません。
- 式には、スカラー副問合せ式を除くすべての書式が可能です。

参照： 11-93 ページの「[デフォルト列値の指定例：](#)」を参照してください。

inline_constraint

inline_constraint を使用すると、新しい列に制約を追加できます。

inline_ref_constraint

inline_ref_constraint を使用すると、新しい REF 型の列を定義できます。

参照： 制約の型の構文や制限などの詳細は、7-5 ページの「[constraints](#)」を参照してください。

column_properties

column_properties を使用して、オブジェクト、ネストした表、VARRAY または LOB 列の記憶特性を指定します。

object_type_col_properties この句は新しいオブジェクト型列や属性を追加する場合にのみ有効です。既存のオブジェクト型列のプロパティを変更するには、*modify_column_clauses* を使用します。

object_type_col_properties を使用すると、新しいオブジェクト列、オブジェクト属性、コレクション列およびコレクション属性の要素に対する記憶特性を指定できます。

column *column* には、オブジェクト列または属性を指定します。

substitutable_column_clause *substitutable_column_clause* を使用すると、同じ階層のオブジェクト列または属性が互いに置換可能かどうかを指定できます。列が特定の型であるか、サブタイプのインスタンスを含むものであるか、またはその両方を指定できます。

- **ELEMENT** を指定すると、コレクションの要素型が宣言した型のサブタイプに制約されます。
- **IS OF [TYPE] (ONLY type)** 句を指定すると、オブジェクト列の型が宣言した型のサブタイプに制約されます。
- **NOT SUBSTITUTABLE AT ALL LEVELS** を指定すると、オブジェクト列がサブタイプに対応するインスタンスを持つことはできません。また、置換は、埋込みオブジェクト属性、埋込みのネストした表および **VARRAY** の要素には使用できません。デフォルトは、**SUBSTITUTABLE AT ALL LEVELS** です。

***substitutable_column_clause* の制限事項：**

- この句は、オブジェクト列の属性には指定できません。ただし、オブジェクト表自体の代替性が設定されていない場合、リレーショナル表におけるオブジェクト型の列およびオブジェクト表のオブジェクト列に対してこの句を指定できます。
- コレクション型の列の場合、この句で指定できる部分は **[NOT] SUBSTITUTABLE AT ALL LEVELS** のみです。

nested_table_col_properties *nested_table_col_properties* 句を使用すると、ネストした表に対して別の記憶特性を指定し、そのネストした表を索引構成表として定義できます。ネストした表の型を持つ列または列属性付きで表を作成する場合は、この句を挿入する必要があります。（この句の中で、親オブジェクト表に対する場合と同じ働きをする句は、ここでは繰り返されません）。

- *nested_item* には、型がネストした表である列（または、その表のオブジェクト型の最上位の属性）の名前を指定します。
ネストした表がマルチレベル・コレクションの場合、内部のネストした表には名前が割り当てられていない場合があります。この場合、*nested_item* 名のかわりに **COLUMN_VALUE** を指定します。
- *storage_table* には、*nested_item* の行を含む表の名前を指定します。記憶表は、親表と同じスキーマ、および親表と同じ表領域内に作成されます。

ネストした表の列属性の制限事項：

- *parallel_clause* は指定できません。
- (*segment_attributes_clause* の一部として) **TABSPACE** をネストした表に指定できません。表領域は常に親表の表領域です。
- *physical_properties* 句の一部として **CLUSTER** を指定できません。

参照： 11-94 ページの「ネストした表の例:」を参照してください。

varray_col_properties *varray_col_properties* を使用すると、VARRAY 型のデータが格納されている LOB に対して、別の記憶特性を指定できます。この句を指定する場合、インラインに格納できるほど小さい値でも、VARRAY は必ず LOB に格納されます。*varray_item* がマルチレベル・コレクションの場合、*varray_item* 内にネストされたすべてのコレクション項目は、常に *varray_item* と同じ LOB に格納されます。

VARRAY 列の属性の制限事項： VARRAY 列に対し、*LOB_parameters* の一部として *TABLESPACE* を指定することはできません。VARRAY に対する LOB 表領域は、デフォルトではそれを格納する表の表領域になります。

LOB_storage_clause *LOB_storage_clause* を使用すると、新しく追加した LOB 列、パーティションまたはサブパーティションの LOB 記憶特性を指定できます。この句では、既存の LOB 列は変更できません。その場合は、*modify_LOB_storage_clause* を使用してください。

CACHE READS 句 CACHE READS は LOB 記憶域にのみ適用されます。LOB 値が書き込み操作中ではなく読み込み操作中にバッファ・キャッシュに入れられることを指定します。

- *LOB_item* には、表とは異なる表領域および記憶特性を明示的に定義する、LOB 列名または LOB オブジェクト属性を指定します。
- *LOB_segname* には、LOB データ・セグメントの名前を指定します。*LOB_item* が複数指定されている場合は、*LOB_segname* を使用できません。

新しい LOB 列を追加するときに、CACHE READS でロギング属性を指定できます。作成時に LOB 列を定義するときにも指定できます。

CACHE または NOCACHE から CACHE READS へ、または CACHE READS から CACHE または NOCACHE へ LOB 列を変更するときに、ロギング属性を変更できます。LOGGING または NOLOGGING を指定しない場合、LOB 列の現行ロギング属性がデフォルトになります。

既存の LOB に対し、CACHE、NCACHE または CACHE READS を指定しない場合、Oracle は、LOB 属性の既存値を保持します。

LOB パラメータの制限事項：

- ハッシュ・パーティションまたはハッシュ・サブパーティションに指定できる *LOB_parameters* のパラメータは、*TABLESPACE* のみです。
- *table* がパーティション化されている場合、*LOB_index_clause* を指定できません。

ENABLE | DISABLE STORAGE IN ROW LOB 値をインライン（行内）またはアウトライン（行外）のどちらに格納するかを指定します（LOB 値が格納されている場所にかかわらず、LOB ロケータは、常に、インラインに格納されます）。

- **ENABLE** は、LOB 値の長さが、約 4000 バイトからシステム制御情報分を引いた長さより小さい場合、LOB 値をインラインに格納することを指定します。これはデフォルト値です。
- **DISABLE** を指定すると、LOB 値の長さにかかわらず、LOB 値はアウトラインに格納されます。

行の記憶域を使用可能にする場合の制限事項： **STORAGE IN ROW** は、一度設定すると変更できません。したがって、この句は *modify_col_properties* 句の一部には指定できません。ただし、新しい列を追加するとき (*add_column_clause*)、または表を移動するとき (*move_table_clause*) に、この設定を変更することはできます。

CHUNK integer LOB の操作用に割り当てるバイト数を指定します。*integer* にデータベースのブロック・サイズの倍数を指定しなかった場合、自動的に次に大きい倍数（バイト単位）に切り上げられます。たとえば、データベース・ブロック・サイズが 2048 の場合、*integer* に 2050 を指定すると、4096 バイト（2 ブロック）が割り当てられます。最大値は 32768（32KB）で、これが Oracle で許容されるブロック・サイズの最大値です。デフォルトの CHUNK サイズは、Oracle での 1 データベース・ブロックです。

CHUNK の制限事項：

- **CHUNK** の値は、一度設定すると変更できません。
- **CHUNK** の値は、**NEXT** の値（デフォルト値または記憶域句で指定した値）以下である必要があります。**CHUNK** の値が **NEXT** の値を超えると、エラーが戻ります。

PCTVERSION integer LOB の記憶域全体のうち、旧バージョンの LOB の更新に使用される最大記憶域の割合（パーセント）を指定します。デフォルト値は 10 です。これは、LOB の記憶域全体の 10% が使用されるまで以前のバージョンの LOB データが上書きされないことを意味します。

RETENTION データベースが自動 UNDO モードで稼働している場合、旧バージョンの LOB を保持するには、**PCTVERSION** ではなく **RETENTION** を指定します。この句によって、**PCTVERSION** のこれまでの設定が上書きされます。

RETENTION の制限事項： データベースが手動 UNDO モードで稼働している場合は、**RETENTION** を指定できません。

参照： **RETENTION** パラメータの詳細は、15-36 ページの「[LOB_parameters](#)」を参照してください。

FREEPOOLS integer データベースが自動 UNDO モードで稼働している場合、この句を使用すると LOB の空きリスト・グループ数を指定できます。この句によって、**FREELIST GROUPS** のこれまでの設定が上書きされます。

FREEPOOLS の制限事項： データベースが手動 UNDO モードで稼働している場合は、FREEPOOLS を指定できません。

参照： FREEPOOLS パラメータの詳細は、15-36 ページの「[LOB_parameters](#)」を参照してください。

LOB_index_clause この句は、Oracle8i 以降は使用されません。Oracle は各 LOB 列に対して索引を生成します。LOB 索引は、システムによって名前が付けられ管理されており、LOB データ・セグメントと同じ表領域に格納されます。

この句の指定が可能な場合もありますが、指定しないことをお勧めします。どのような場合でも、LOB 索引を LOB データと異なる表領域に置かないでください。

参照： 以前のバージョンから移行された表の LOB 索引の管理方法については、『Oracle9i データベース移行ガイド』を参照してください。

LOB_partition_storage

LOB_partition_storage 句を使用すると、各パーティションに、別の LOB_storage_clause または varray_col_properties を指定できます。パーティションは、その位置の順に指定してください。パーティションの順番は、USER_IND_PARTITIONS ビューの PARTITION_NAME および PARTITION_POSITION 列の間合せから得ることができます。

特定のパーティションに、LOB_storage_clause または varray_col_properties 句を指定しなかった場合、表レベルで LOB 項目に指定された記憶特性が設定されます。表のレベルでも LOB 項目に記憶特性を指定しなかった場合、LOB データ・パーティションは、対応する表パーティションと同じ表領域に格納されます。

LOB パーティションの記憶域の制限事項： 1 つの ALTER TABLE 文では、LOB_partition_storage のリストを 1 つのみ指定できます。すべての LOB_storage_clauses および varray_col_properties 句は、LOB_partition_storage 句のリストの前に指定する必要があります。

XMLType_column_properties XMLType_column_properties を指定すると、XMLType 列に対する記憶域属性を指定できます。

XMLType_storage XMLType 列は、LOB 列またはオブジェクト・リレーショナル列に格納できます。

- オブジェクト・リレーショナル列に XMLType データを格納するには、STORE AS OBJECT RELATIONAL を指定します。データ・オブジェクトをリレーショナルに格納すると、リレーショナル列に索引を定義できるため、間合せのパフォーマンスが向上します。

オブジェクト・リレーショナル形式での格納を指定する場合、XMLSchema_spec 句も指定する必要があります。

- CLOB 列に XMLType データを格納する場合、STORE AS CLOB を指定します。CLOB 列にデータを格納すると、元の内容が保持されるため、検索時間が短縮されます。

LOB 記憶域を定義する場合、LOB パラメータと *XMLSchema_spec* 句のいずれかを指定できますが、両方は指定できません。特定のスキーマ・ベースの XML インスタンスに表や列を制限する場合は、*XMLSchema_spec* 句を指定します。

XMLSchema_spec この句を使用すると、登録した XMLSchema (XMLSCHEMA 句、または ELEMENT 句の一部として登録) の URL および XML 要素名を指定できます。XMLSchema の URL は省略可能ですが、要素は必ず指定します。XMLSchema の URL を指定する場合は、DBMS_XMLSCHEMA パッケージを使用して XMLSchema をあらかじめ登録しておく必要があります。

参照：

- *LOB_segname* および *LOB_parameters* 句の詳細は、11-45 ページの「*LOB_storage_clause*」を参照してください。
- オブジェクト・リレーショナル表における XML Type 列の例は、15-68 ページの「XMLType 列の例:」を参照してください。XMLSchema の作成例は、D-11 ページの「SQL 文での XML の使用方法」を参照してください。
- XMLType 列と表および XMLSchema の作成の詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。

modify_column_clauses

modify_column_clauses を使用すると、既存の列のプロパティや既存のオブジェクト型の列の代替性を変更できます。

参照： 11-92 ページの「[表の列の変更例:](#)」を参照してください。

modify_col_properties

この句を使用すると、列のプロパティを変更できます。この句で省略した列定義のオプション部分 (データ型、デフォルト値、制約) は、変更されません。

datatype 列に対するすべての行が NULL の場合、列のデータ型を変更できます。ただし、マテリアライズド・ビューの記憶表にある列のデータ型を変更した場合、それに対応するマテリアライズド・ビューが無効になります。

参照整合性制約の外部キーの一部として、文で列が指定されている場合のみ、データ型を省略できます。参照整合性制約の参照キーに対応する列のデータ型が、その列に自動的に割り当てられます。

すべての列が NULL 値であるかどうかにかかわらず、文字列と未処理列のサイズまたは数値列の精度は、いつでも大きくすることができます。データの変換が必要ない範囲内で、列のデータ型のサイズを縮小できます。既存のデータがスキャンされ、データが新しい最大長を超える場合はエラーが戻されます。

DATE 列を TIMESTAMP または TIMESTAMP WITH LOCAL TIME ZONE に変更できます。すべての TIMESTAMP WITH LOCAL TIME ZONE を DATE 列に変更できます。

注意： TIMESTAMP WITH LOCAL TIME ZONE 列を DATE 列に変更すると、秒の小数部およびタイムゾーンで調整されたデータが失われます。

- TIMESTAMP WITH LOCAL TIME ZONE データに秒の小数部がある場合、秒の小数部を丸めて列の行データが更新されます。
 - TIMESTAMP WITH LOCAL TIME ZONE データに 60 以上の分フィールドがある場合（夏時間の規則が切り換えられた場合に境界で発生）、その分フィールドから 60 を引いて列の行データが更新されます。
-
-

表が空の場合、日時列または期間列の先行フィールドまたは秒の小数部を増やすことも減らすこともできます。表が空でない場合、日時列または期間列の先行フィールドまたは秒の小数部を増やすことのみできます。

LONG 列は CLOB または NCLOB 列に、LONG RAW 列は BLOB 列に変更できます。

- 変更された LOB 列は、元の LONG 列で定義されたすべての制約およびトリガーを継承します。すべての制約を変更する場合、後続の ALTER TABLE 文で変更する必要があります。
- ドメイン索引が LONG 列で定義されている場合、列を LOB に変更する前に削除する必要があります。
- 変更後、表のすべての列にある他のすべての索引を再構築する必要があります。

参照：

- LONG および LOB 列の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- LONG から LOB への移行の詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。
- 索引の削除および再構築の詳細は、9-61 ページの「[ALTER INDEX](#)」を参照してください。
- 11-92 ページの「[LONG 列から LOB への変換例：](#)」を参照してください。

CHAR および VARCHAR2 列の場合、CHAR（元々バイトで指定されていた列に対するキャラクタ・セマンティクス）または BYTE（元々文字で指定されていた列に対するバイト・セマンティクス）を指定すると、長さセマンティクスを変更できます。既存の列の長さを確認するには、ALL_TAB_COLUMNS、USER_TAB_COLUMNS または DBA_TAB_COLUMNS データ・ディクショナリ・ビューの CHAR_USED 列を問い合わせます。

参照：

- バイト・セマンティクスおよびキャラクタ・セマンティクスの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

inline_constraint MODIFY 句を使用して既存の列に追加できる整合性制約は NOT NULL 制約のみで、その列に NULL 値がない場合に限定されます。これ以外の整合性制約（一意制約、主キー制約、参照整合性制約および CHECK 制約）を既存の列に定義する場合は、add_column_clause を使用します。既存の列の制約を変更する場合は、constraint_clauses を使用します。

列のプロパティの変更の制限事項：

- 列にドメイン索引が定義されている場合は、表の列を変更できません。最初にドメイン索引を削除してから列を変更する必要があります。
- 索引構成表に対して ROWID データ型の列は指定できませんが、UROWID 型の列は指定できます。
- 列のデータ型を REF に変更できません。

参照： マテリアライズド・ビューの再検証の詳細は、9-88 ページの「ALTER MATERIALIZED VIEW」を参照してください。

modify_col_substitutable

この句を使用すると、既存のオブジェクト型列の代替性を設定または変更できます。

FORCE キーワードを指定すると、型 ID 情報またはサブタイプ属性に関するデータが含まれる非表示列が削除されます。オブジェクト型の列または属性が FINAL でない場合、FORCE を指定する必要があります。

列の代替性の変更の制限事項：

- ALTER TABLE 文には、この句を一度だけ指定できます。
- オブジェクト表自体の代替性が設定されている場合、オブジェクト表の列の代替性は変更できません。

- IS OF TYPE 構文を使用して列を作成または追加した場合は、この句を指定できません (11-44 ページの「[substitutable_column_clause](#)」を参照)。この構文によって、オブジェクト列や属性で使えるサブタイプの範囲が、特定のサブタイプに制限されます。
- ネストしたオブジェクト型の属性に FINAL がない場合、FORCE を指定しても VARRAY 列を NOT SUBSTITUTABLE に変更できません。

drop_column_clause

drop_column_clause は、不要になった列を削除したり、将来、システム・リソースへの要求が少なくなったときに削除するようにマークを付けることによって、データベースの領域を解放します。

- ネストした表の列を削除すると、その記憶表も削除されます。
- LOB 列を削除すると、LOB データおよび対応する LOB 索引セグメントも削除されます。
- BFILE 列を削除すると、その列に格納されたロケータのみ削除され、ロケータによって参照されるファイルは削除されません。
- INCLUDING 列として定義した列を削除（または未使用とマーク）すると、この列の直前に格納された列が新しい INCLUDING 列になります。

SET UNUSED 句

SET UNUSED を使用すると、1 つ以上の列が未使用としてマーク付けされます。この句を指定した場合でも、実際に目的の列が表の各行から削除されるわけではありません（これらの列が使用しているディスク領域がリストアされるわけではありません）。このため、応答時間は DROP 句を使用した場合よりも速くなります。

UNUSED のマークが付いた列を持つすべて表は、データ・ディクショナリ・ビュー `USER_UNUSED_COL_TABS`、`DBA_UNUSED_COL_TABS` および `ALL_UNUSED_COL_TABS` で表示できます。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

未使用列は削除されたように処理されますが、その列データは表の行に残っています。UNUSED のマークが付けられた列にはアクセスできなくなります。「SELECT *」問合せでも、未使用列からはデータを取り出すことができません。また、UNUSED のマークが付けられた列の名前および型は、DESCRIBE コマンドでは表示されず、未使用列と同じ名前の新しい列を表に追加できます。

注意： これらの列を実際に削除するまでは、表当たり 1000 列の制限に対して、これらの列もカウント対象になります。ただし、DLL 文でこの句の結果をロールバックすることはできません。つまり、SET USED に相当するものを発行しても SET UNUSED 列を取り出すことはできません。

また、LONG 型の列に UNUSED のマークを付けた場合、この未使用の LONG 列を実際に削除しないかぎり、その表には別の LONG 列を追加できません。

参照： 1000 列の制限の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

DROP 句

表のそれぞれの行から、対象となる列に関連付けられた列記述子およびデータを削除する場合は、DROP を指定します。特定の列を明示的に削除した場合、対象の表で UNUSED のマークが付いている列もすべて同時に削除されます。

列データを削除した場合、次のものが削除されます。

- 対象の列に定義されているすべての索引。
- 対象の列を参照しているすべての制約。
- 対象の列に統計タイプが関連付けられている場合、FORCE オプションによって、関連付けは解除され、その統計タイプを使用して収集したすべての統計情報は削除されます。

注意： 対象の列が他の列の親キーである場合または CHECK 制約が対象である列とその他の列を参照している場合は、Oracle はエラーを返し、CASCADE CONSTRAINTS 句を指定しないかぎり、列を削除しません。この句を指定した場合、対象である列を参照しているすべての制約が削除されます。

参照： 統計タイプの関連付けの解除方法の詳細は、16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

DROP UNUSED COLUMNS 句

表から未使用にマークされている列をすべて削除する場合は、DROP UNUSED COLUMNS を指定します。表の未使用の列からディスク領域を再利用する場合に、この文を使用します。表に未使用の列がない場合でも、エラーは戻されません。

column 未使用として設定または削除する 1 つ以上の列を指定します。列を 1 つのみ指定する場合にかぎり、COLUMN キーワードを使用します。列リストを指定する場合、リストには重複する列を指定できません。

CASCADE CONSTRAINTS 削除する列に定義されている主キーおよび一意キーを参照する外部キー制約をすべて削除する場合、および削除する列に定義されている複数列制約をすべて削除する場合は、CASCADE CONSTRAINTS を指定します。他の表の列、または対象である表の他の列が参照している制約がある場合は、CASCADE CONSTRAINTS を指定します。CASCADE CONSTRAINTS を指定しない場合、その文は異常終了し、エラーが戻されます。

INVALIDATE INVALIDATE キーワードはオプションです。ビュー、トリガー、ストアド・プログラム・ユニットなどのすべての依存オブジェクトが自動的に無効になります。オブジェクトの無効化は再帰的プロセスです。したがって、すべての直接的な依存オブジェクトおよび間接的な依存オブジェクトが無効になります。ただし、Oracle では、リモート依存性をローカル依存性と別に管理しているため、無効になるのはローカル依存性のみです。

この文によって無効となったオブジェクトは、次に参照された際に自動的に有効になります。オブジェクトを参照する前に、オブジェクトに存在するエラーは、すべて修正しておく必要があります。

参照： 依存性の詳細は、『Oracle9i データベース概要』を参照してください。

CHECKPOINT CHECKPOINT を指定すると、integer 行を処理した後に DROP COLUMN 操作のチェックポイントが適用されます。このとき integer は任意の値であり、1 以上である必要があります。integer が表の行数より大きい場合、すべての行が処理された後にチェックポイントが適用されます。integer を指定しない場合、デフォルトの 512 が設定されます。チェックポイントは、DROP COLUMN 操作中に蓄積された UNDO ログの量を削減し、ロールバック・セグメント領域外での実行を防ぎます。ただし、チェックポイントが適用された後にこの文が中断された場合、表は使用禁止の状態のままになります。表が使用できない間、その表に対して実行可能な操作は、DROP TABLE、TRUNCATE TABLE および ALTER TABLE DROP COLUMNS CONTINUE（後述）のみです。

この句は列データを削除しないため、SET UNUSED と同時に使用できません。

DROP COLUMNS CONTINUE 句

中断されたところから列削除操作を続ける場合は、DROP COLUMNS CONTINUE を指定します。表が有効な状態にあるときにこの文を発行すると、エラーになります。

列の削除の制限事項：

- この句の各部分は、文の中で 1 回のみ指定でき、他の ALTER TABLE 句と同時に使用することはできません。たとえば、次のような文は許可されません。

```
ALTER TABLE t1 DROP COLUMN f1 DROP (f2);
ALTER TABLE t1 DROP COLUMN f1 SET UNUSED (f2);
```

```
ALTER TABLE t1 DROP (f1) ADD (f2 NUMBER);
ALTER TABLE t1 SET UNUSED (f3)
  ADD (CONSTRAINT ck1 CHECK (f2 > 0));
```

- オブジェクト型の列は、エンティティとしてのみ削除できます。オブジェクト型の列から属性を削除するには、ALTER TYPE ... DROP ATTRIBUTE 文を CASCADE INCLUDING TABLE DATA 句とともに使用します。属性の削除は、すべての依存オブジェクトに影響することに注意してください。詳細は、12-16 ページの「[DROP ATTRIBUTE](#)」を参照してください。
- 索引構成表からは、主キー列でない場合にかぎり、列を削除できます。索引構成表の主キー制約は絶対に削除できないため、CASCADE CONSTRAINTS を指定しても主キー列は削除できません。
- 削除した列または未使用の列で表をエクスポートできます。ただし、エクスポート・ファイルに指定されたすべての列が表に存在する（これらの列のいずれも削除または未使用のマークを付けられていない）場合のみ、その表をインポートできます。そうでない場合は、エラーが戻ります。
- ドメイン索引が構築されている列は削除できません。
- REF 列からは SCOPE 表制約および WITH ROWID 制約を削除できません。
- 次のものは、この句を使用して削除できません。
 - 擬似列、クラスタ列、またはパーティション列（パーティションが作成されたすべての表領域がオンラインで読み込み / 書き込みモードである場合、パーティション表から非パーティション列を削除できます）。
 - ネストした表の列、オブジェクト表の列または SYS が所有する表の列。

参照： 11-88 ページの「[列の削除例](#)」を参照してください。

rename_column_clause

rename_column_clause を使用すると、*table* の列名を変更できます。新しい列は、*table* 内の他の列と同じ名前を指定しないでください。

列名を変更すると、依存オブジェクトは次のように処理されます。

- 名前が変更された列に依存するファンクション・ベース索引および CHECK 制約は、引き続き有効です。
- 依存ビュー、トリガー、ドメイン索引、ファンクション、プロシージャおよびパッケージには、INVALID のマークが付けられます。これらのオブジェクトが次にアクセスされる際に、妥当性チェックが実行されます。妥当性チェックが正常に実行されない場合は、新しい列名を使用してこれらのオブジェクトを変更する必要があります。

列名の変更の制限事項：

- 同じ文の中で、この句を他の *column_clauses* と同時に使用することはできません。
- 結合索引の定義に使用される列名は変更できません。列名を変更する場合、索引を削除し、列名を変更してから、索引を再作成する必要があります。

参照： 11-88 ページの「[列名の変更例：](#)」を参照してください。

modify_collection_retrieval

modify_collection_retrieval を使用すると、データベースからコレクション型の項目が取り出されたときの戻り値を変更できます。

collection_item ネストした表型または VARRAY 型の列修飾属性の名前を指定します。

RETURN AS 問合せの結果として何を戻り値とするかを指定します。

- LOCATOR は、ネストした表に対して一意のロケータを戻すことを指定します。
- VALUE は、ネストした表のコピーをそのまま戻すことを指定します。

参照： 11-85 ページの「[コレクションの取出し例：](#)」を参照してください。

modify_LOB_storage_clause

modify_LOB_storage_clause を使用すると、*LOB_item* の物理属性を変更できます。各 *modify_LOB_storage_clause* に対して、*LOB_item* を 1 つのみ指定できます。

REBUILD FREEPOOLS 句を指定すると、LOB 列から古いデータがすべて削除されます。この句は、LOB の管理目的で PCTVERSION に戻す場合にのみ有効です。古いデータ・ブロックを管理する場合、この処理を実行します。また、Oracle9i リリース 1 (9.0.1) 以下にダウングレードする場合、この処理を実行する必要があります。

LOB 記憶域の変更の制限事項：

- LOB 記憶域属性を変更する場合、*storage_clause* の INITIAL パラメータの値は変更できません。
- 同じ文で *allocate_extent_clause* と *deallocate_unused_clause* の両方を指定することはできません。
- 両方は指定できません。

参照： LOB パラメータの設定の詳細は、15-36 ページの「CREATE TABLE」の「[LOB_storage_clause](#)」を参照してください。また、11-94 ページの「[LOB 列の例：](#)」を参照してください。

alter_varray_col_properties

alter_varray_col_properties を使用すると、VARRAY が格納されている既存の LOB の記憶特性を変更できます。

VARRAY 列の属性変更の制限事項： *LOB_parameters* の TABLESPACE 句は、この句の一部として指定できません。VARRAY に対する LOB 表領域のデフォルトは、表を含む表領域になります。

constraint_clauses

constraint_clauses を使用すると、アウトライン宣言を使用して新しい制約を追加、既存の制約の状態を変更、および制約を削除できます。

参照： アウトライン制約および *constraint_state* のすべてのキーワードとパラメータの詳細は、7-5 ページの「[constraints](#)」を参照してください。

制約の追加

ADD 句を使用すると、アウトライン制約またはアウトライン REF 制約を表に追加できます。

参照： 11-87 ページの「[CHECK 制約を使用禁止にする例:](#)」、11-91 ページの「[オブジェクト識別子の指定例:](#)」および 11-95 ページの「[REF 列の例:](#)」を参照してください。

制約の変更

MODIFY CONSTRAINT 句を使用すると、既存の制約の状態を変更できます。

制約変更の制限事項：

- NOT DEFERRABLE 制約は、INITIALLY DEFERRED には変更できません。
- 表索引のパーティション化キーまたはサブパーティション化キーの一部である列の長さまたはデータ型は変更できません。
- CHAR 型の列を VARCHAR2（または VARCHAR）型に変更または VARCHAR2（または VARCHAR）型の列を CHAR 型に変更できるのは、列のすべての行が NULL 値である場合、または列のサイズを変更しない場合のみです。
- クラスタの一部である場合は、LONG または LONG RAW 列を LOB に変更できません。LONG または LONG RAW 列を LOB に変更する場合は、同じ ALTER TABLE 文では、DEFAULT 句および *LOB_storage_clause* 以外は指定できません。
- LONG または LONG RAW 列を LOB に変更する場合のみ、*LOB_storage_clause* を *modify_col_properties* の一部として指定できます。
- 索引構成表に対してこの句を指定する場合、同じ文では他の句を指定できません。

参照： 11-86 ページの「[制約状態の変更例 :](#)」を参照してください。

制約名の変更

RENAME CONSTRAINT 句を使用すると、*table* の既存の制約名を変更できます。新しい制約名は、同スキーマ内にあるオブジェクトの既存の制約と同じ名前にはできません。制約に依存するすべてのオブジェクトは、引き続き有効です。

参照： 11-93 ページの「[制約名の変更例 :](#)」を参照してください。

drop_constraint_clause

drop_constraint_clause を使用すると、データベースの整合性制約を削除できます。制約の適用を中止し、データ・ディクショナリから制約が削除されます。各 *drop_constraint_clause* には、制約を 1 つのみ指定できますが、1 つの文の中では、複数の *drop_constraint_clauses* を指定できます。

主キー PRIMARY KEY を指定すると、表の主キー制約を削除できます。

UNIQUE UNIQUE を指定すると、指定した列の一意制約を削除できます。

注意： ビットマップ・ジョイン・インデックスが定義されている列から主キー制約または一意制約を削除すると、索引は無効になります。ビットマップ・ジョイン・インデックスの詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。

CONSTRAINT CONSTRAINT *constraint* を指定すると、主キー制約または一意制約以外の整合性制約を削除できます。

CASCADE 削除する整合性制約に依存するその他の整合性制約もすべて削除する場合は、CASCADE を指定します。

KEEP | DROP INDEX KEEP または DROP INDEX を指定すると、主キーまたは一意制約の適用に使用する索引を残すか削除するかを指定できます。

制約削除の制限事項：

- 参照整合性制約の一部の主キー制約または一意キー制約は、外部キーを削除しないと削除できません。参照されたキーと外部キーをともに削除する場合は、CASCADE 句を使用してください。CASCADE を省略すると、外部キーによって参照される主キー制約および一意制約は削除されません。
- 主キーをオブジェクト識別子 (OID) として使用している表では、主キー制約は (CASCADE 句を使用しても) 削除できません。

- REF 列の参照整合性制約を削除した場合、REF 列の有効範囲には参照先の表が含まれたままになります。
- 列の有効範囲は削除できません。

参照： 11-94 ページの「[制約の削除例:](#)」を参照してください。

alter_external_table_clause

alter_external_table_clause を使用すると、外部表の特性を変更できます。この句は、外部データには影響しません。 *parallel_clause*、*enable_disable_clause*、*external_data_properties* および REJECT LIMIT 句の構文およびセマンティクスは、CREATE TABLE の内容と同じです。15-32 ページの「CREATE TABLE」の「[external_table_clause](#)」を参照してください。

外部表の変更の制限事項：

- この句以外の句を使用して外部表を変更できません。
- LONG、LOB またはオブジェクト型の列は外部表に追加できません。また、外部表の列のデータ型を、これらの型に変更できません。
- 制約は外部表に追加できません。
- 外部表の記憶域パラメータは変更できません。

alter_table_partitioning

ここで説明する句は、パーティション表にのみ適用されます。1 つの ALTER TABLE 文の中では、パーティション操作を他のパーティション操作または実表での操作と組み合わせて使用することはできません。

表パーティションの変更の制限事項：

- 関連する CONTEXT ドメイン索引が含まれる表におけるパーティション操作の詳細は、『Oracle Text リファレンス』を参照してください。
- 1 つ以上のマテリアライズド・ビューのマスター表で、パーティションを削除、交換、切捨て、移動、変更または分割した場合、その表に関する大量の既存ロード情報が削除されます。したがって、前述の操作のいずれかを行う前に、必ず依存するマテリアライズド・ビューをリフレッシュしてください。
- *table* にビットマップ・ジョイン・インデックスが定義されている場合、*table* のパーティションの変更操作を実行すると、索引に UNUSABLE のマークが付けられます。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

modify_table_default_attrs

modify_table_default_attrs を使用すると、*table* の属性に対する新しいデフォルト値を指定できます。その後に作成するパーティションおよび LOB パーティションは、パーティションまたは LOB パーティションの作成時に明示的に上書きしないかぎり、この値を継承します。既存のパーティションおよび LOB パーティションは、この句の影響を受けません。

文の中で指定した属性のみが影響を受け、指定されたデフォルト値は、個々のパーティション・レベルで指定された属性で上書きされます。

- FOR PARTITION は、コンポジット・パーティション表にのみ適用されます。*partition* の属性に新しいデフォルト値を指定します。その後に作成する *partition* のサブパーティションおよび LOB パーティションは、サブパーティションまたは LOB パーティションの作成時に明示的に上書きしないかぎり、この値を継承します。既存のサブパーティションは、この句の影響を受けません。
- PCTTHRESHOLD、*key_compression* および *alter_overflow_clause* は、パーティション化された索引構成表にのみ有効です。ただし、*key_compression* 句では、COMPRESS キーワードの後に int 型を指定できません。キー圧縮の長さは、表の作成時にのみ指定できます。
- 索引構成表の索引セグメントに対しては、*segment_attributes* 句で PCTUSED パラメータを指定できません。
- *key_compression_clause* は、キー圧縮が表レベルで定義されている場合にのみ指定できます。

set_subpartition_template

set_subpartition_template 句を使用すると、表パーティションに対するデフォルト・リスト定義またはハッシュ・サブパーティション定義を作成したり、既存の定義を置き換えることができます。この句は、コンポジット・パーティション表に対してのみ有効です。この句によって、既存のサブパーティション・テンプレートは置き換えられます。既存のテンプレートがない場合は新規のテンプレートが作成されます。既存のサブパーティション、ローカル索引およびグローバル索引は影響を受けません。ただし、その後のパーティション化操作（追加操作やマージ操作）では、新規テンプレートが使用されます。

既存のサブパーティション・テンプレートを削除するには、ALTER TABLE *table* SET SUBPARTITION TEMPLATE () を指定します。

サブパーティションのテンプレートの制限事項：

- レンジ・ハッシュ・コンポジット・パーティション表の場合、`list_values_clause`を指定できません。
- レンジ・リスト・コンポジット・パーティション表の場合、`hash_subpartition_quantity`は指定できません。
- レンジ・ハッシュ・パーティション表およびレンジ・リスト・パーティション表の両方で、サブパーティションに対して指定できる `partitioning_storage_clause` の句は、`TABSPACE` のみです。

`modify_table_partition`

`modify_table_partition` 句を使用すると、レンジ・パーティション、ハッシュ・パーティションおよびリスト・パーティションの実際の物理属性を変更できます。そのパーティションの1つ以上のLOB項目の記憶域属性を任意に変更できます。物理属性（制限事項については後述）、ロギングおよび記憶域パラメータに対して、新しい値を指定できます。

また、パーティションを変更した結果、使用不可能になったローカル索引の処理方法も指定できます。11-80 ページの「[UNUSABLE LOCAL INDEXES 句](#)」を参照してください。

パーティション化された索引構成表の場合、パーティションの変更時にマッピング表も同時に更新できます。11-41 ページの「[alter_mapping_table_clauses](#)」を参照してください。

参照： 11-90 ページの「[表のパーティションの変更例：](#)」を参照してください。

`modify_range_partition`

`table` がコンポジット・パーティションで、レンジ・パーティションを変更する場合、この句を使用します。

- `allocate_extent_clause` を指定した場合、`partition` のそれぞれのサブパーティションにエクステントが割り当てられます。
- `deallocate_unused_clause` を指定した場合、`partition` のそれぞれのサブパーティションから未使用の記憶域の割当てが解除されます。
- この句で変更された他の属性は、`partition` のサブパーティションでも変更され、既存の値は上書きされます。既存のサブパーティションの属性が変更されないようにするには、`modify_table_default_attrs` の `FOR PARTITION` 句を使用します。

レンジ・パーティションの変更の制限事項： `UNUSABLE LOCAL INDEXES` を指定すると、`modify_range_partition` のその他の句は指定できません。

add_hash_subpartition この句は、レンジ・ハッシュ・コンポジット・パーティションに対してのみ有効です。add_hash_subpartition 句を使用すると、ハッシュ・サブパーティションを partition に追加できます。Oracle は、ハッシュ関数によって partition の他のサブパーティションから再ハッシュされた行を、新しいサブパーティションに入れます。オプションのロード・バランシングの場合、サブパーティションの合計数は偶数にする必要があります。

- subpartition を指定しなかった場合、SYS_SUBPn という形式の名前が割り当てられます。
- この操作では、list_values_clause は有効ではありません。
- partitioning_storage_clause でサブパーティションに対して指定できる句は、TABLESPACE 句のみです。TABLESPACE を指定しなかった場合、新しいサブパーティションは partition のデフォルトの表領域に格納されます。

表のすべてのグローバル索引が無効になります。update_global_index_clause を使用し、操作中にこれらの索引を更新できます。

選択したパーティションに対応するローカル索引パーティションが追加されます。追加したパーティションに対応するローカル索引パーティションに UNUSABLE のマークが付けられ、再構築する必要があります。

add_list_subpartition add_list_subpartition を使用すると、パーティションにリスト・サブパーティションを追加できます。この句は、レンジ・リスト・コンポジット・パーティションにデフォルト・サブパーティションを作成していない場合にのみ有効です。

- subpartition を指定しなかった場合、SYS_SUBPn という形式の名前が割り当てられます。
- この操作には list_values_clause が必要です。また、list_values_clause には、partition のその他のサブパーティションには存在していない値を指定する必要があります。ただし、他のパーティションのサブパーティションに登録された値は指定できます。
- partitioning_storage_clause でサブパーティションに対して指定できる句は、TABLESPACE 句のみです。TABLESPACE を指定しなかった場合、新しいサブパーティションは partition のデフォルトの表領域に格納されます。パーティションにデフォルト表領域が定義されていない場合は、表のデフォルト表領域が使用されます。表にもデフォルト表領域が定義されていない場合は、ユーザーのデフォルト表領域が使用されます。

表のすべてのローカル索引パーティションには、同じ値リストを持つサブパーティションも追加されます。表の既存のローカル索引パーティションおよびグローバル索引パーティションは影響を受けません。

リスト・サブパーティションの変更の制限事項： パーティションにデフォルト・サブパーティションがすでに作成されている場合は、この句を指定できません。その場合、*split_list_subpartition* 句を使用して、デフォルト・パーティションを分割する必要があります。

modify_hash_partition

partition_attributes でハッシュ・パーティションを変更する場合、*allocate_extent_clause* および *deallocate_unused_clause* のみを指定できます。パーティションのその他の属性は、TABLESPACE 以外の表レベルのデフォルトによって継承されます。TABLESPACE は作成時と同じ状態です。

COALESCE SUBPARTITION COALESCE SUBPARTITION は、ハッシュ・サブパーティションにのみ適用されます。ハッシュ・サブパーティションが選択され、その内容が 1 つ以上の他のサブパーティション（ハッシュ・ファンクションが決定する）に分割された後、選択されたサブパーティションを削除する場合は、COALESCE SUBPARTITION を使用します。

表のすべてのグローバル索引が無効になります。*update_global_index_clause* を使用し、操作中にこれらの索引を更新できます。

選択したパーティションに対応するローカル索引パーティションが削除されます。1 つ以上の吸収パーティションに対応するローカル索引パーティションに、UNUSABLE のマークが付けられます。この索引パーティションは、再構築する必要があります。

ハッシュ・パーティションの変更の制限事項： UNUSABLE LOCAL INDEXES を指定すると、*modify_hash_partition* のその他の句は指定できません。

modify_list_partition

リスト・パーティションを変更する場合、次の句も使用できます。

リスト・パーティションの変更の制限事項： UNUSABLE LOCAL INDEXES を指定すると、*modify_list_partition* のその他の句は指定できません。

ADD | DROP VALUES 句 これらの句は、リスト・パーティションを変更する場合のみ有効です。これらの句によって、表のローカル索引とグローバル索引が影響を受けることはありません。

- ADD VALUES 句を使用すると、*partition* の *partition_value* リストが追加した値まで拡張されます。追加するパーティションの値に、15-46 ページの「CREATE TABLE」の *list_partitioning* でリストしているすべての規則および制限事項に準拠する値を設定する必要があります。
- DROP VALUES 句を使用すると、1 つ以上の *partition_value* の排除によって *partition* の *partition_value* リストが削除されます。この句を指定すると、Oracle はこの値の行が存在しないことを検証します。そのような行が存在する場合は、エラーが戻されます。

注意： 表にローカル同一キー索引が定義されている場合、デフォルトのリスト・パーティションのある表では ADD VALUES 操作が改善されます。このような索引によって、DROP VALUES 操作も改善されます。

リスト値の追加と削除の制限事項：

- デフォルトのリスト・パーティションには、値を追加できません。table にデフォルト・パーティションが定義されている場合、デフォルト・パーティション以外に値を追加すると、その値がデフォルト・パーティションにすでに存在しているかどうかを確認されます。デフォルト・パーティションにその値が存在している場合、エラーが戻されます。
- デフォルト・パーティションからは値を削除できません。

modify_table_subpartition

この句は、コンポジット・パーティション表にのみ適用されます。

modify_hash_subpartition

modify_hash_subpartition 句を使用すると、table の個々のサブパーティションに対して記憶域を割当てまたは割当て解除を行うことができます。この句は、レンジ・ハッシュ・コンポジット・パーティション表に対してのみ有効です。

また、パーティションを変更した結果、使用不可能になったローカル索引の処理方法も指定できます。11-80 ページの「[UNUSABLE LOCAL INDEXES 句](#)」を参照してください。

ハッシュ・サブパーティションの変更の制限事項： サブパーティションに指定できる *modify_LOB_parameters* は、*allocate_extent_clause* および *deallocate_unused_clause* のみです。

modify_list_subpartition

modify_list_subpartition 句を指定すると、リスト・サブパーティションをハッシュ・サブパーティションと同様に変更できます。また、リスト・サブパーティションの値リストに値を追加したり、この値リストから値を削除することができます。この句は、レンジ・リスト・コンポジット・パーティション表に対してのみ有効です。

ADD VALUES ADD VALUES を指定すると、*subpartition* の値リストを拡張できます。

- *subpartition* および同じパーティション内のその他のサブパーティションの値リストにすでに存在する値は指定できません。ただし、他のパーティションのサブパーティションの値リストに定義されている値は指定できます。

- デフォルト・サブパーティションが定義されている場合、追加した値がデフォルト・パーティションの行にすでに存在していないかが確認されます。追加した値がデフォルト・パーティションに存在する場合、指定した文は実行されません。
- ローカル索引サブパーティションの値リストには、対応する値が追加されます。この操作によって、ローカル索引パーティションとグローバル索引パーティションのステータスが影響を受けることはありません。

DROP VALUES DROP VALUES を指定すると、*subpartition* の値リストから 1 つ以上の値を削除できます。

- 指定する値は、*subpartition* の既存値のサブセットにする必要があります。
- この句を使用して、サブパーティションのすべての値は削除できません。すべての値を削除するには、ALTER TABLE ... DROP SUBPARTITION 文を使用する必要があります。
- *subpartition* 内の行に削除対象の値が含まれる場合、この操作は実行されずに、エラーが戻されます。最初に削除対象の値が含まれる行を削除してから、この文を再発行する必要があります。
- 対応するローカル索引サブパーティションの値リストは、自動的に更新されます。この操作によって、ローカル索引パーティションとグローバル索引パーティションのステータスが影響を受けることはありません。

また、パーティションを変更した結果、使用不可能になったローカル索引の処理方法も指定できます。11-80 ページの「[UNUSABLE LOCAL INDEXES 句](#)」を参照してください。

リスト・サブパーティションの変更の制限事項： サブパーティションに指定できる *modify_LOB_parameters* は、*allocate_extent_clause* および *deallocate_unused_clause* のみです。

move_table_partition

move_table_partition 句を使用すると、*partition* を別のセグメントへ移動できます。パーティション・データの別の表領域への移動、断片化を削減するためのデータの再クラスト化、および作成時の物理属性の変更ができます。

表に LOB 列が含まれている場合、*LOB_storage_clause* を使用して、このパーティションに関連付けられた LOB データおよび LOB 索引セグメントを移動できます。この場合、指定した LOB のみが影響を受けます。特定の LOB 列に *LOB_storage_clause* を指定しなかった場合、その列の LOB データおよび LOB 索引セグメントは移動されません。

ヒープ構成表のすべてのグローバル索引が無効になります。

[update_global_index_clause](#) を使用し、操作中にこれらの索引を更新できます。索引構成表のグローバル索引は主キー・ベースであるため、使用禁止になりません。

選択したパーティションに対応するローカル索引パーティションが移動されます。移動したパーティションが空でない場合に、UNUSABLE のマークが付けられます。このパーティションは再構築する必要があります。

LOB データ・セグメントを移動する場合、古いデータ・セグメントおよび対応する索引セグメントが削除され、新しい表領域を指定しない場合でも、新しいセグメントが作成されます。

移動操作では、`parallel_clause` (指定されている場合) からパラレル属性が取得されます。`parallel_clause` が指定されていない場合は、表のデフォルトのパラレル属性があれば、これが使用されます。どちらも指定されていない場合は、パラレル化を使用せずに移動が行われます。

MOVE PARTITION の `parallel_clause` を指定すると、`table` のデフォルトのパラレル属性は変更されません。

注意： 索引構成表の場合、主キーのアドレスおよびその値を使用して、論理 ROWID が構成されます。論理 ROWID は、表の 2 次索引に格納されます。索引構成表のパーティションを移動した場合、ROWID のアドレス部分に変更され、パフォーマンスの障害になる場合があります。最適なパフォーマンスを維持するには、移動したパーティションの 2 次を再構築し、ROWID を更新してください。

参照： 論理 ROWID の詳細は、『Oracle9i データベース概要』を参照してください。また、11-91 ページの「[表のパーティションの移動例 :](#)」も参照してください。

MAPPING TABLE MAPPING TABLE 句は、索引構成表用に定義されたマッピング表を持つ索引構成表のみに関連します。索引パーティションとともにマッピング表は移動され、対応するすべてのビットマップ索引パーティションに UNUSABLE のマークが付けられます。

参照： 15-31 ページの CREATE TABLE の「[mapping_table_clause](#)」を参照してください。

表パーティションの移動の制限事項：

- `partition` がハッシュ・パーティションである場合、この句に TABLESPACE の属性以外は指定できません。
- この句は、サブパーティションを含むパーティションに対して指定できません。ただし、`move_table_subpartition_clause` を使用してサブパーティションを移動できます。

`move_table_subpartition`

`move_table_subpartition` 句を使用すると、`subpartition` を別のセグメントへ移動できます。TABLESPACE を指定しない場合、サブパーティションは同じ表領域に残ります。

[update_global_index_clause](#) を使用し、操作中に `table` のグローバル索引を更新できます。サブパーティションが空でない場合、UNUSABLE のマークが付けられます。移動したサブパーティションに対応するすべてのローカル索引サブパーティションは再構築する必要があります。

表に LOB 列が含まれている場合、[LOB_storage_clause](#) を使用して、このサブパーティションに関連付けられた LOB データおよび LOB 索引セグメントを移動できます。この場合、指定した LOB のみが影響を受けます。特定の LOB 列に [LOB_storage_clause](#) を指定しなかった場合、その列の LOB データおよび LOB 索引セグメントは移動されません。

LOB データ・セグメントを移動する場合、古いデータ・セグメントおよび対応する索引セグメントが削除され、新しい表領域を指定しない場合でも、新しいセグメントが作成されます。

表サブパーティションの移動の制限事項： [subpartition_spec](#) で指定できる [partitioning_storage_clause](#) の句は、TABLESPACE のみです。

`add_table_partition`

`add_table_partition` 句を使用すると、`table` にハッシュ・パーティション、レンジ・パーティションまたはリスト・パーティションを追加できます。

`table` で定義されているローカル索引に、基本の表パーティションと同じ名前で新しいパーティションが追加されます。索引に同じ名前のパーティションがすでに存在する場合、SYS_Pn という形式でパーティションの名前が生成されます。

`table` が索引構成されている場合、表で定義されたすべてのマッピング表およびオーバーフロー領域に、パーティションが追加されます。

参照： 11-89 ページの「[LOB を持つ表パーティションの追加例](#) :」を参照してください。

`add_range_partition_clause`

`add_range_partition_clause` を使用すると、新しいレンジ・パーティションをパーティション表の一番上（最後の既存のパーティションの後）に追加できます。新しいパーティションに作成時の物理属性を指定できます。表に LOB 列が含まれている場合、1 つ以上の LOB 項目にパーティション・レベルの属性を指定することもできます。

新しい `partition_name` を指定しなかった場合、SYS_Pn という形式でパーティション名が割り当てられます。コンポジット・パーティション表にレンジ・パーティションを追加してサブパーティションを定義しなかった場合、11-67 ページの「[partition_level_subpartition](#)」で説明するとおり、サブパーティション名は自動的に割り当てられます。

ドメイン索引が `table` で定義されている場合、IN_PROGRESS または FAILED のマークが付いていると無効になります。

1 つの表当たりのパーティションの上限は 64KB-1 です。

レンジ・パーティションの追加の制限事項：

- 既存の上位パーティションにある各パーティション化キーのパーティションの上限が MAXVALUE の場合、表にパーティションを追加できません。そのかわり、`split_table_partition` 句を使用して、表の始めまたは中間にパーティションを追加します。
- `key_compression` および OVERFLOW は、パーティション化された索引構成表に対してのみ有効です。パーティション表にすでにオーバーフロー・セグメントが存在する場合にかぎり、OVERFLOW を指定できます。表レベルでキー圧縮が使用可能な場合にかぎり、キー圧縮を指定できます。
- PCTUSED パラメータは、索引構成表の索引セグメントに対して指定できません。

range_values_clause 新しいパーティションの上限を指定します。value_list は、column_list に対応するリテラル値を順序どおりにカンマで区切ったリストです。value_list の値は、表内にある既存の最上位パーティションのパーティション境界より大きくする必要があります。

partition_level_subpartition partition_level_subpartition 句 (table_partition_description 内) は、コンポジット・パーティション表に対してのみ使用できます。この句を使用すると、新しいレンジ・ハッシュ・コンポジット・パーティションまたはレンジ・リスト・コンポジット・パーティションのハッシュ・サブパーティションまたはリスト・サブパーティションを指定できます。この句によって、表レベルで subpartition_template に定義されたすべてのサブパーティションの設定は上書きされます。

コンポジット・パーティション：

- サブパーティションの数を指定できます（または、サブパーティションが格納される 1 つ以上の表領域を指定することもできます）。この場合、SYS_SUBPn という形式でサブパーティション名が割り当てられます。表領域の数は、サブパーティションの数と等しくなくてもかまいません。パーティション数が表領域数より多い場合、表領域名を循環させます。
- または、subpartition_spec を使用して、個々のパーティション名を指定できます。また、オプションで各パーティションが格納される表領域を指定することもできます。
- partition_level_subpartition を省略すると、サブパーティションのテンプレートが作成されている場合は、そのテンプレートに基づいてサブパーティションが作成されます。サブパーティションのテンプレートが作成されていない場合、1 つのハッシュ・サブパーティションまたは 1 つのデフォルト・リスト・サブパーティションが作成されます。
- partition_level_subpartition 句全体を指定しない場合、次のようにサブパーティション名が割り当てられます。

- サブパーティションのテンプレートを定義し、サブパーティション名を指定した場合、「`partition_name` アンダースコア (`_`) `subpartition_name`」(たとえば、`P1_SUB1`) という形式でサブパーティション名が生成されます。
- サブパーティションのテンプレートを定義していない場合、またはサブパーティションのテンプレートは定義したが、パーティション名を指定していない場合、`SYS_SUBP n` という形式でサブパーティション名が生成されます。
- `partition_spec` で指定できる `partitioning_storage_clause` の句は、`TABLESPACE` のみです。

レンジ-ハッシュ・コンポジット・パーティションの場合、`subpartition_spec` の `list_values_clause` は無効です。

レンジ-リスト・コンポジット・パーティションは、次の方法で指定できます。

- `hash_subpartition_quantity` は適切でないため、`partition_level_subpartition` の下位ブランチを使用する必要があります。
- `subpartition_spec` の中では、各サブパーティションに対して `list_values_clause` を指定する必要があります。このとき、同じパーティションのその他のサブパーティション内に存在しない値を指定する必要があります。

参照： 15-7 ページの「[CREATE TABLE](#)」を参照してください。

`table` に定義されたすべてのローカル索引に対して、同じサブパーティションを持つ新規の索引パーティションが追加されます。`table` のグローバル索引が影響を受けることはありません。

`add_hash_partition_clause`

`add_hash_partition_clause` を使用すると、パーティション表の一番上に新しいハッシュ・パーティションを追加できます。Oracle は、ハッシュ関数によって `table` の他のパーティションから再ハッシュされた行を新しいサブパーティションに入れます。オプションのロード・バランシングの場合、パーティションの合計数は偶数にする必要があります。

パーティション名を指定します。また、パーティションが格納される表領域を指定することもできます。名前を指定しない場合、`SYS_Pn` という形式でパーティション名が割り当てられます。`TABLESPACE` を指定しなかった場合、新しいパーティションは表のデフォルトの表領域に格納されます。他の属性は、常に、表レベルのデフォルトから継承されます。

`update_global_index_clause` を使用し、操作中に `table` のグローバル索引を更新できます。ヒープ構成表の場合、この操作がパーティション間でデータを再ハッシュすると `UNUSABLE` のマークが付けられ、対応するローカル索引パーティションを再構築する必要があります。索引構成表の索引は主キー・ベースであるため、使用禁止になりません。

`parallel_clause` を使用すると、新しいパーティションの作成をパラレル化するかどうかを指定できます。

参照： ハッシュ・パーティション化の詳細は、15-7 ページの「[CREATE TABLE](#)」および『Oracle9i データベース概要』を参照してください。

ハッシュ・パーティションの追加の制限事項： `partition_level_subpartition` は、`table_partition_description` に対して指定できません。

add_list_partition_clause

`add_list_partition_clause` を使用すると、パーティションの新しい一連の値を使用して、`table` に新しいパーティションを追加できます。新しいパーティションに作成時の物理属性を指定できます。表に LOB 列が含まれている場合、1 つ以上の LOB 項目にパーティション・レベルの属性を指定することもできます。

表にリスト・パーティションを追加すると、表に定義されているすべてのローカル索引に、対応する索引パーティションが同じ値のリストで追加されます。グローバル索引には影響がありません。

リスト・パーティションの追加の制限事項：

- `partition_level_subpartition` は、`table_partition_description` に対して指定できません。
- 表のデフォルト・パーティションがすでに定義されている場合は、リスト・パーティションを追加できません。その場合、`split_table_partition` 句を使用して、デフォルト・パーティションを分割する必要があります。

参照：

- リスト・パーティションの詳細および制限事項については、15-46 ページの `CREATE TABLE` の「[list_partitioning](#)」を参照してください。
- 11-90 ページの「[デフォルト・リスト・パーティションの使用例:](#)」を参照してください。

coalesce_table_partition

COALESCE は、ハッシュ・パーティションにのみ適用されます。

`coalesce_table_partition` 句を使用すると、最後のハッシュ・パーティションが選択され、その内容が 1 つ以上の残りのパーティション（ハッシュ・ファンクションが決定）に分割された後、選択されたパーティションが削除されます。

ヒープ構成表のすべてのグローバル索引が無効になります。

[update_global_index_clause](#) を使用し、操作中にこれらの索引を更新できます。索引構成表のグローバル索引は主キー・ベースであるため、使用禁止になりません。

選択したパーティションに対応するローカル索引パーティションが削除されます。1 つ以上の吸収パーティションに対応するローカル索引パーティションに、UNUSABLE のマークが付けられます。この索引パーティションは、再構築する必要があります。

drop_table_partition

drop_table_partition 句を使用すると、パーティション表から、*partition* およびそのパーティション内のデータが削除されます。データを表に残したままパーティションを削除する場合は、そのパーティションを隣接するパーティションにマージする必要があります。

参照： 11-75 ページの「[merge_table_partitions](#)」を参照してください。

表に LOB 列が存在する場合、*partition* に対応する LOB データ、および LOB 索引パーティション（および存在する場合はサブパーティション）も削除されます。

table が索引構成されており、定義されたマッピング表を持つ場合、対応するマッピング表のパーティションも同様に削除されます。

partition に対応するローカル索引パーティションおよびサブパーティションは、UNUSABLE のマークが付いている場合でも削除されます。

[update_global_index_clause](#) を使用し、操作中にヒープ構成表のグローバル索引を更新できます。[update_global_index_clause](#) とともに [parallel_clause](#) を指定すると、削除操作ではなく、索引の更新がパラレル化されます。

レンジ・パーティションを削除し、その後、削除したパーティションに属していた行を挿入した場合、1 つ上位のパーティションに行が格納されます。ただし、そのパーティションが最上位のパーティションである場合、削除したパーティションが表していた値の範囲が表に対して無効になるため、挿入は失敗します。

表パーティションの削除の制限事項：

- ハッシュ・パーティション表のパーティションは削除できません。
- *table* にパーティションが 1 つのみ存在する場合は、このパーティションを削除できません。その表を削除する必要があります。

参照： 11-90 ページの「[表のパーティションの削除例：](#)」を参照してください。

drop_table_subpartition

この句を使用すると、レンジ・リスト・コンポジット・パーティション表からリスト・サブパーティションを削除できます。削除対象のサブパーティション内の行はすべて削除されます。

ローカル索引の対応するサブパーティションは、自動的に削除されます。その他の索引サブパーティションには影響がありません。`update_global_index_clause` を指定しないかぎり、グローバル索引には UNUSABLE のマークが付けられます。

表サブパーティションの削除の制限事項：

- ハッシュ・サブパーティションは削除できません。かわりに、MODIFY PARTITION ... COALESCE SUBPARTITION 構文を使用してください。
- パーティションの最後のサブパーティションは削除できません。かわりに、`drop_table_partition` 句を使用してください。

rename_partition_subpart

`rename_table_partition` を使用すると、表パーティションまたは表サブパーティションの名前を `current_name` から `new_name` に変更できます。パーティションおよびサブパーティションのどちらの場合も、`new_name` は同じ表に存在するすべてのパーティションおよびサブパーティションと異なる値である必要があります。

`table` が索引構成されている場合、対応する主キー索引パーティションに、既存のオーバーフロー・パーティションおよびマッピング表パーティションと同じ名前が割り当てられます。

参照： 11-91 ページの「[表のパーティション名の変更例:](#)」を参照してください。

truncate_partition_subpart

TRUNCATE PARTITION を指定すると、`partition` からすべての行が削除されます。表がコンポジット・パーティション化されている場合は、`partition` のサブパーティションからすべての行が削除されます。TRUNCATE SUBPARTITION を指定すると、`subpartition` からすべての行が削除されます。`table` が索引構成されている場合、対応するすべてのマッピング表のパーティションおよびオーバーフロー領域のパーティションが切り捨てられます。

切り捨てるパーティションまたはサブパーティションにデータが含まれている場合は、まず、その表の参照整合性制約を使用禁止にする必要があります。また、別の方法として、行を削除してからパーティションを切り捨てる方法もあります。

表に LOB 列がある場合、このパーティションの LOB データおよび LOB 索引セグメントも切り捨てられます。表がコンポジット・パーティション化されている場合、このパーティションのサブパーティションの LOB データおよび LOB 索引セグメントは切り捨てられません。

`table` でドメイン索引が定義されている場合、索引に IN_PROGRESS または FAILED のマークが付いていると無効になります。また、切り捨てられる表パーティションに対応する索引パーティションに IN_PROGRESS のマークが付いていると無効になります。

切り捨てられるそれぞれのパーティションまたはサブパーティションでは、対応するローカル索引パーティションおよびサブパーティションも切り捨てられます。これらの索引パーティションまたはサブパーティションに UNUSABLE のマークが付いている場合、これらは切り捨てられ、UNUSABLE のマークは VALID にリセットされます。

`update_global_index_clause` を使用し、操作中に `table` のグローバル索引を更新できます。`update_global_index_clause` とともに `parallel_clause` を指定すると、切捨て操作ではなく、索引の更新がパラレル化されます。

DROP STORAGE 削除した行が占有していた領域の割当てを解除し、表領域内の別のスキーマ・オブジェクトがその領域を使用できるように、DROP STORAGE を指定します。

REUSE STORAGE パーティションまたはサブパーティションに割り当てられ、削除された行から領域を確保するように、REUSE STORAGE を指定します。この領域は、そのパーティションまたはサブパーティションに対する後続の挿入および更新のためにのみ使用できます。

参照： 11-91 ページの「[表のパーティションの切捨て例：](#)」を参照してください。

split_table_partition

`split_table_partition` 句を使用すると、`current_partition` から新しいセグメント、物理属性および初期エクステントをそれぞれ含む、2 つの新しいパーティションが作成されます。`current_partition` に対応付けられたセグメントは、廃棄されます。

新しいパーティションは、指定されていないすべての物理属性を `current_partition` から継承します。

注意： 一定の条件が満たされると、SPLIT PARTITION および SPLIT SUBPARTITION を最適化して処理速度を上げることができます。これらの操作の最適化の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

デフォルト・リスト・パーティションを分割すると、分割された 1 つ目のパーティションには分割値が格納され、2 つ目のパーティションにはデフォルト値が格納されます。

`table` が索引構成されている場合、対応するマッピング表のパーティションが分割され、親の索引構成表のパーティションと同じ表領域に配置されます。対応するオーバーフロー領域も分割されます。OVERFLOW 句を使用すると、新しいオーバーフロー領域にセグメント属性を指定できます。

対応するローカル索引パーティションに `UNUSABLE` のマークが付いている場合でも、それらは分割されます。分割したパーティションに対応するローカル索引パーティションに `UNUSABLE` のマークが付けられ、再構築する必要があります。新しい索引パーティションの属性は、分割されたパーティションから継承されます。新しい索引パーティションは、分割された索引パーティションのデフォルト表領域に格納されます。索引パーティションにデフォルト表領域が定義されていない場合、下位層にある表のパーティションの表領域が使用されます。

`table` に LOB 列がある場合、`LOB_storage_clause` を使用して、分割の結果生成された LOB データ・セグメントに対して個々の LOB 記憶域属性を指定できます。
`current_partition` の LOB データおよび LOB 索引セグメントが削除され、新しい表領域を指定しなくても、各 LOB 列または各パーティションに新しいセグメントが作成されます。

AT 句 `AT` 句は、レンジ・パーティションのみに適用されます。新しい 2 つのパーティションのうちの最初の方に、新しい上限（境界を含まない）を指定します。値リストは、`current_partition` の元のパーティション境界より小さく、その次に小さいパーティション（そのようなパーティションがある場合）のパーティション境界より大きい値にする必要があります。

VALUES 句 `VALUES` 句は、リスト・パーティションのみに適用されます。新しい 2 つのパーティションのうちの最初に含まれるパーティションの値を指定します。指定したパーティションの値を使用して新しい最初のパーティションが作成され、`current_partition`に残っているパーティションの値を使用して新しい 2 番目のパーティションが作成されます。このため、値リストには `current_partition` のすべてのパーティションの値を含むことはできません。また、`current_partition` に存在していないすべてのパーティションの値も含むことはできません。

INTO 句 `INTO` 句によって、分割の結果生成された 2 つのパーティションを記述します。分割の結果生成される 2 つのパーティションにオプションの名前および物理属性を指定しない場合でも、`function_spec` にはキーワード `PARTITION` を指定する必要があります。新しいパーティション名を指定しない場合、`SYS_Pn` という形式の名前が割り当てられます。指定しないすべての属性は、`current_partition` から継承されます。

レンジ・ハッシュ・コンポジット・パーティション表の場合、新しいパーティションにサブパーティションを指定するとき、サブパーティションに対して `TABLESPACE` のみを指定できます。他のすべての属性は、`current_partition` から継承されます。新しいパーティションに対してサブパーティションを指定しなかった場合、パーティションの表領域も `current_partition` から継承されます。

レンジ・リスト・コンポジット・パーティション表の場合、新しいパーティションにはサブパーティションを指定できません (`table_partition_description` の `partition_level_subpartition` を使用)。分割されたパーティションのサブパーティションの属性 (サブパーティションと属性リストの番号) は、`current_partition` から継承されます。

すべてのレンジ・リスト・コンポジット・パーティション表およびレンジ・ハッシュ・コンポジット・パーティション表において、新しく作成したサブパーティションに対して名前を指定しなかった場合、次の親パーティションから名前が継承されます。

- 親パーティション内のサブパーティションが「`partition_name` アンダースコア (`_`) `subpartition_name`」という形式の名前（たとえば、`P1_SUBP1`）を持つ場合、新しいパーティション名に基づいて、サブパーティションの名前が生成されます（たとえば、`P1A_SUB1`、`P1B_SUB1`）。
- 親パーティション内のサブパーティションがその他の形式の名前を持つ場合、`SYS_SUBPn` という形式のサブパーティション名が生成されます。

索引に `UNUSABLE` のマークが付いている場合でも、`table` で定義されている各ローカル索引の対応するパーティションが分割されます。

ヒープ構成表のすべてのグローバル索引が無効になります。

[update_global_index_clause](#) を使用し、操作中にこれらの索引を更新できます。索引構成表のグローバル索引は主キー・ベースであるため、使用禁止になりません。

分割操作をパラレル化しても、表のデフォルトのパラレル属性を変更しない場合は、`parallel_clause` を使用します。

表パーティションの分割の制限事項：

- この句は、ハッシュ・パーティション表に対して指定できません。
- `partition_spec` では、`key_compression` および `OVERFLOW` 句は、パーティション化された索引構成表に対してのみ指定できます。また、`PCTUSED` パラメータは、索引構成表の索引セグメントに対して指定できません。

参照： 11-89 ページの「[表のパーティションの分割例 :](#)」 および 11-90 ページの「[デフォルト・リスト・パーティションの使用例 :](#)」を参照してください。

`split_table_subpartition`

この句を使用すると、重複しない値リストを持つ 2 つのサブパーティションにリスト・サブパーティションを分割できます。

注意： 一定の条件が満たされると、`SPLIT PARTITION` および `SPLIT SUBPARTITION` を最適化して処理速度を上げることができます。これらの操作の最適化の詳細は、『[Oracle9i データベース管理者ガイド](#)』を参照してください。

- `VALUES` 句では、新しい 2 つのサブパーティションのうち、1 番目のサブパーティションに含める値を指定します。同じパーティション内の別のサブパーティションに対して `NULL` を指定していない場合は、`NULL` を指定できます。指定したサブパーティションの値リストに基づいて新しい 1 番目のサブパーティションが作成されます。また、現行のサブパーティションのその他のパーティション値に基づいて 2 番目のパーティションが作成されます。このため、値リストには現行のサブパーティションのすべてのパーティション値を含めることができません。また、現行のサブパーティションに存在しないパーティション値も含めることができません。
- `INTO` 句を使用すると、分割の結果生成された 2 つのサブパーティションを定義できます。2 つのサブパーティションにオプションの名前および属性を指定しない場合でも、`subpartition_spec` ではキーワード `PARTITION` を指定する必要があります。新しいサブパーティション名を指定しない場合、またはこの句全体を指定しない場合、`SYS_SUBPn` という形式の名称が割り当てられます。指定しないすべての属性は、現行のサブパーティションから継承されます。

対応するローカル索引パーティションに `UNUSABLE` のマークが付いている場合でも、それらは分割されます。新しい索引サブパーティションに名前があらかじめ指定されていないかぎり、新しい表のサブパーティションの名称が継承されます。この場合、`SYS_SUBPn` という形式の新しい索引サブパーティション名が割り当てられます。新しい索引サブパーティションの物理属性は、親サブパーティションから継承されます。親サブパーティションにデフォルトの `TABLESPACE` 属性が定義されていない場合、対応する新しい表のサブパーティションの記憶域が継承されます。

表のグローバル索引には `UNUSABLE` のマークが付けられます。

`update_global_index_clause` も指定した場合、グローバル索引の再構築が実行されます。

表サブパーティションの分割の制限事項：

- この句は、ハッシュ・サブパーティションに対して指定できません。
- `subpartition_spec` で指定できる `partitioning_storage_clause` の句は、`TABLESPACE` のみです。

merge_table_partitions

`merge_table_partitions` 句を使用すると、`table` の 2 つのパーティションの内容が 1 つの新しいパーティションにマージされ、元の 2 つのパーティションが削除されます。

- レンジ・パーティションの場合、マージされる 2 つのパーティションは、隣接している必要があります。新しいパーティションは、元の 2 つのパーティションのうち、上位のパーティションのパーティション境界を継承します。

- リスト・パーティションをマージする場合は、隣接している必要はありません。2つのリスト・パーティションをマージする場合、結果のパーティションの値リストは、マージされる2つのパーティションの値リストの集合をあわせたものです。デフォルトのリスト・パーティションを別のリスト・パーティションとマージすると、結果のパーティションはデフォルト・パーティションになり、デフォルト値が含まれます。
- 2つのレンジ・リスト・コンポジット・パーティションをマージする場合、`partition_level_subpartition` は指定できません。サブパーティション・テンプレートからサブパーティションの情報が取得されます。サブパーティションのテンプレートを定義していない場合、1つのデフォルト・サブパーティションが作成されます。

`segment_attributes_clause` に指定されていない属性はすべて、表レベルのデフォルトから継承されます。

新しい `partition_name` を指定しなかった場合、SYS_Pn という形式でパーティション名が割り当てられます。新しいパーティションにサブパーティションがある場合、11-67 ページの「`partition_level_subpartition`」で説明するとおりサブパーティション名が割り当てられます。

ヒープ構成表のすべてのグローバル索引には、UNUSABLE のマークが付けられます。`update_global_index_clause` を使用し、操作中にこれらの索引を更新できます。索引構成表のグローバル索引は主キー・ベースであるため、使用禁止になりません。

選択したパーティションに対応するローカル索引パーティションが削除されます。マージされたパーティションに対応するローカル索引パーティションに UNUSABLE のマークが付けられ、再構築する必要があります。

表パーティションのマージの制限事項： この句は、ハッシュ・パーティション表に対して指定できません。かわりに、`coalesce_table_partition` 句を使用してください。

参照： 11-90 ページの「2つの表パーティションのマージ例：」および
11-90 ページの「デフォルト・リスト・パーティションの使用例：」を参照
してください。

`partition_level_subpartition` `partition_level_subpartition` 句は、レンジ・ハッシュ・コンポジット・パーティションをマージする場合にのみ有効です。この句を使用すると、新しくマージしたパーティションに対してサブパーティション属性を指定できます。この句に指定されていないすべての属性は、表レベルの値から継承されます。この句を指定しない場合、マージされた新しいパーティションは、表レベルのデフォルトからサブパーティション属性を継承します。

この句を指定しない場合、新しいパーティションでは、定義したサブパーティションのテンプレートからサブパーティションの定義が継承されます。サブパーティションのテンプレートを定義していない場合、新しくマージされたパーティションに1つのサブパーティションが作成されます。

`parallel_clause` を指定すると、マージ操作がパラレル化されます。

partition_level_subpartition 句の制限事項： この句は、レンジ・リスト・コンポジット・パーティションに対して指定できません。

merge_table_subpartitions

`merge_table_subpartitions` 句を使用すると、`table` の 2 つのリスト・パーティションの内容が 1 つの新しいサブパーティションにマージされ、元の 2 つのサブパーティションが削除されます。マージ対象の 2 つのサブパーティションは同一パーティションに属している必要がありますが、隣接している必要はありません。マージの結果生成されるサブパーティション内のデータは、マージされたサブパーティションのデータが結合されたものです。

- 新しいサブパーティション名を指定しない場合、または `INTO` 句全体を指定しない場合、`SYS_SUBPn` という形式の名前が割り当てられます。
- `INTO` 句を指定しない場合、`subpartition_spec` でキーワード `SUBPARTITION` を指定する必要があります。`list_values_clause` は指定できません。`partitioning_storage_clause` で指定できるのは、`TABLESPACE` 句のみです。

新しいサブパーティションに対して明示的に指定しなかった属性は、パーティション・レベルの値から継承されます。新しいサブパーティションに対してサブパーティション名を再利用すると、新しいサブパーティションでは、パーティション・レベルのデフォルト値ではなく、名前が再利用されたサブパーティションの値が継承されます。

対応するローカル索引のサブパーティションがマージされ、結果として生成される索引サブパーティションには `UNUSABLE` のマークが付けられます。また、`table` のパーティション化されたグローバル索引とパーティション化されていないグローバル索引の両方に対して、`UNUSABLE` のマークが付けられます。

表サブパーティションのマージの制限事項： この句は、ハッシュ・サブパーティションに対して指定できません。

exchange_partition_subpart

`EXCHANGE PARTITION` 句または `EXCHANGE SUBPARTITION` 句を使用して、次のデータおよび索引セグメントを交換します。

- 1 つのハッシュ・パーティション、リスト・パーティションまたはレンジ・パーティション（または 1 つのハッシュ・サブパーティション、リスト・サブパーティション）を含む 1 つの非パーティション表
- レンジ・ハッシュ・コンポジット・パーティション表のレンジ・パーティションのハッシュ・サブパーティションを持つハッシュ・パーティション表
- レンジ・リスト・コンポジット・パーティション表のレンジ・パーティションのリスト・サブパーティションを持つリスト・パーティション表

交換対象の表、パーティションおよびサブパーティションの構造は、パーティション・キーを含め常に同じである必要があります。リスト・パーティションとリスト・サブパーティションの場合、対応する値リストも一致している必要があります。

この句をトランスポータブル表領域とともに使用すると、高速データ・ロードが容易になります。

参照： トランスポータブル表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

`table` に LOB 列がある場合、各 LOB 列の LOB データ、および LOB 索引パーティション・セグメントまたは LOB 索引サブパーティション・セグメントは、`table` の対応する LOB データおよび LOB 索引セグメントと交換されます。

2 つのオブジェクトのすべてのセグメント属性（表領域およびロギングを含む）も交換されます。

表およびパーティションのすべての統計情報（表、列、索引統計およびヒストグラムを含む）が交換されます。新しいパーティション表を受け取る表の集計統計は再計算されます。

交換されるオブジェクトのすべてのグローバル索引は、無効になります。

`update_global_index_clause` をこの句とともに指定すると、交換されるパーティションを含む表のグローバル索引は更新されます。交換される表のグローバル索引は、無効のままになります。`update_global_index_clause` とともに `parallel_clause` を指定すると、交換操作ではなく、索引の更新がパラレル化されます。

参照： 11-80 ページの「パーティションまたはサブパーティションの交換の制限事項 :」を参照してください。

WITH TABLE `table` パーティションまたはサブパーティションを交換する表を指定します。

INCLUDING INDEXES ローカル索引パーティションまたはサブパーティションを（非パーティション表の）対応する表索引または（ハッシュ・パーティション化表の）対応するローカル索引と交換する場合は、INCLUDING INDEXES を指定します。

EXCLUDING INDEXES 交換された表のすべての標準索引、索引パーティションおよびパーティションに対応するすべての索引パーティションまたはサブパーティションに UNUSABLE のマークを付ける場合に、EXCLUDING INDEXES を指定します。

WITH VALIDATION WITH VALIDATION を指定すると、交換された表にあるすべての行が交換されたパーティションまたはサブパーティションにマップされない場合にエラーが戻されます。

WITHOUT VALIDATION 交換された表にある行が正しくマップされたかどうかのチェックが必要ない場合は、WITHOUT VALIDATION を指定します。

exceptions_clause 制約に違反するすべての列の ROWID を格納する表を指定します。
schema を指定しない場合、自分のスキーマ内に例外表があるとみなされます。この句自体
 を指定しない場合、表の名前は EXCEPTIONS になります。例外表は、ローカル・デー
 タベース内にある必要があります。

次のいずれかのスクリプトを使用して、EXCEPTIONS 表を作成できます。

- UTLEXCPT.SQL は、物理 ROWID を使用します。そのため、行は、索引構成表からで
 なく従来表から収集されます（次の注意を参照）。
- UTLEXPT1.SQL は、ユニバーサル ROWID を使用します。そのため、行は、ヒープ構成
 表と索引構成表の両方から収集されます。

独自の例外表を作成する場合、これら 2 つのスクリプトのいずれかで規定される形式に従う
 必要があります。

注意： ユニバーサル ROWID ではなく、主キーに基づく索引構成表から
 例外を収集する場合、索引構成表ごとに別の例外表を作成し、主キー記憶
 域を確保する必要があります。スクリプトを変更および再発行すること
 によって、別の名前の例外表を複数作成できます。

参照：

- SQL スクリプトの詳細は、『Oracle9i PL/SQL パッケージ・プロシー
 ジャおよびタイプ・リファレンス』の DBMS_IOT パッケージを参照し
 てください。
- 移行行および連鎖行の削除については、『Oracle9i データベース・パ
 フォーマンス・チューニング・ガイドおよびリファレンス』を参照し
 てください。
- これらのスクリプトを使用する場合の互換性は、『Oracle9i データベ
 ース移行ガイド』を参照してください。

例外表の指定の制限事項：

- この句はサブパーティションでは無効です。
- パーティション表は、一意制約で定義する必要があり、その制約は DISABLE
 VALIDATE の状態である必要があります。

これらの条件が満たされない場合、この句は無視されます。

参照： 制約の確認の詳細は、7-5 ページの「[constraints](#)」を参照して
 ください。また、11-87 ページの「[索引構成表の例外表の作成例:](#)」も参照
 してください。

パーティションまたはサブパーティションの交換の制限事項：

- 交換される両方の表は同じ主キーを含む必要があり、参照表が空でないかぎり、どちらの表も有効な外部キーを参照できません。
- パーティション表とコンポジット・パーティション表のレンジ・パーティションの交換時には、次の制限事項があります。
 - パーティション表のパーティション・キーは、コンポジット・パーティション表のサブパーティション・キーと同一である必要があります。
 - パーティション表のパーティション数は、コンポジット・パーティション表のレンジ・パーティションのサブパーティション数と同一である必要があります。
 - リスト・パーティション表をコンポジット・パーティション表のレンジ・リスト・パーティションと交換する場合、リスト・パーティションの値リストは、レンジ・リスト・サブパーティションの値リストと同一である必要があります。
- パーティション化された索引構成表に対して、次の追加の制限事項があります。
 - ソースおよびターゲットの表およびパーティションは、その主キーが同じ列に同じ順序で設定されている必要があります。
 - 圧縮が使用可能な場合は、ソースおよびターゲットの両方で使用可能で、接頭辞の長さは同じである必要があります。
 - ソースおよびターゲットの両方は、索引構成されている必要があります。
 - ソースおよびターゲットの両方に、オーバーフロー・セグメントが必要です。または、ソースおよびターゲットの両方がオーバーフロー・セグメントを持ってはいけません。また、ソースおよびターゲットの両方ともマッピング表を含むか、または両方とも含まない必要があります。
 - ソースおよびターゲットの両方は、LOB 列に対して記憶域属性が同一である必要があります。

参照： 11-90 ページの「[表パーティションの交換例：](#)」を参照してください。

UNUSABLE LOCAL INDEXES 句

この 2 つの句を使用すると、*partition* に対応するローカル索引パーティションおよび索引サブパーティションの属性を変更できます（使用する句は、パーティションとサブパーティションのどちらを変更するかによって決まります）。

- **UNUSABLE LOCAL INDEXES** を指定すると、*partition* に関連付けられたすべてのローカル索引パーティションに、UNUSABLE のマークが付けられます。
- **REBUILD UNUSABLE LOCAL INDEXES** を指定すると、*partition* に関連付けられたすべての UNUSABLE のローカル索引パーティションまたはサブパーティションが再構築されます。

UNUSABLE LOCAL INDEXES 句の制限事項：

- この句を *modify_table_partition* の他の句と同時に指定することはできません。
- サブパーティションが含まれるパーティションに対しては、*modify_table_partition* でこの句を指定できません。ただし、*modify_hash_subpartition* または *modify_list_subpartition* 句では指定できます。

update_global_index_clause

表パーティションで DDL 文を実行するときに、グローバル索引が *table* で定義されている場合は、DDL 文が実行中のパーティションを除き、索引全体が無効になります。この句を使用すると、DDL 操作中に変更するグローバル索引パーティションを更新でき、DDL 文の後に索引を再構築する必要がなくなります。

UPDATE GLOBAL INDEXES *UPDATE GLOBAL INDEXES* を指定すると、*table* で定義したグローバル索引を更新することができます。

INVALIDATE GLOBAL INDEXES *INVALIDATE GLOBAL INDEXES* を指定すると、*table* で定義したグローバル索引を無効にすることができます。

どちらも指定しない場合、グローバル索引は無効になります。

グローバル索引の無効化の制限事項： この句はグローバル索引のみをサポートしています。ドメイン索引および索引構成表はサポートされません。また、この句では *USABLE* および *VALID* の索引のみが更新されます。*UNUSABLE* の索引は使用禁止のままになり、*INVALID* のグローバル索引は無視されます。

参照： 11-91 ページの「[グローバル索引の更新例:](#)」を参照してください。

parallel_clause

parallel_clause を使用すると、表の間合せおよび DML に対するデフォルトの並列度を変更できます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL *NOPARALLEL* を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、**PARALLEL** を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ **PARALLEL_THREADS_PER_CPU** の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である **並列度** を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

表の並列化の変更の制限事項：

- *table* に LOB 型またはユーザー定義オブジェクト型の列が含まれている場合、この *table* での INSERT、UPDATE および DELETE は、通知なしに逐次実行されます。ただし、後続の間合せはパラレルで実行されます。
- *parallel_clause* を *move_table_clause* と組み合わせて指定する場合、このパラレル化は移動のみに適用され、後続の表での DML 操作および間合せには適用されません。

参照： 15-51 ページの CREATE TABLE の「[parallel_clause の注意事項：](#)」を参照してください。また、11-86 ページの「[パラレル処理の指定例：](#)」を参照してください。

move_table_clause

move_table_clause によって、非パーティション表のデータを新しいセグメントに再配置できます。オプションとして、別の表領域への配置および記憶域属性の変更を行うこともできます。

LOB_storage_clause および *varray_col_properties* 句を使用して、その表に関連付けられた LOB データ・セグメントを移動することもできます。この句で指定していない LOB 項目は移動できません。

index_org_table_clause

索引構成表の場合は、*index_org_table_clause* 構文を使用すると、オーバーフロー・セグメント属性も指定できます。*move_table_clause* を使用すると、索引構成表の主キー索引が再構築されます。オーバーフロー・データ・セグメントは、キーワード OVERFLOW が明示的に指定されていないかぎり、再構築されません。ただし、次の場合は例外です。

- ALTER TABLE 文の一部として PCTTHRESHOLD の値または INCLUDING 列を変更する場合は、オーバーフロー・データ・セグメントが再構築されます。
- 索引構成表内のアウトラインの列（LOB 列、VARRAY 列、ネストした表の列）のいずれかを明示的に移動する場合は、オーバーフロー・データ・セグメントも再構築されます。

LOB 列の索引およびデータ・セグメントは、LOB 列を ALTER TABLE 文の一部として明示的に指定しないかぎり、再構築されません。

ONLINE 句 その表の主キー索引の再構築中に、索引構成表に対する DML 操作を可能にする場合は、ONLINE を指定します。

ONLINE 句の制限事項：

- 同じ文でこの句と他の句は結合できません。
- この句は、パーティション化されていない索引構成表に対してのみ指定できます。
- オンライン MOVE 中のパラレル DML はサポートされていません。ONLINE を指定し、パラレル DML 文を発行すると、Oracle はエラーを戻します。

mapping_table_clauses マッピング表が存在していない場合に Oracle にマッピング表を作成させるには、MAPPING TABLE を指定します。マッピング表がすでに存在する場合、マッピング表は索引構成表とともに移動され、すべてのビットマップ索引には UNUSABLE のマークが付けられます。新しいマッピング表は、親表と同じ表領域に作成されます。

NOMAPPING を指定すると、既存のマッピング表が削除されます。

マッピング表の制限事項： 表でビットマップ索引が定義されている場合は、NOMAPPING を指定できません。

参照： 15-31 ページの CREATE TABLE の「[mapping_table_clause](#)」を参照してください。

key_compression *key_compression* 句を使用すると、索引構成表のキー圧縮を使用可能または使用禁止にできます。

- COMPRESS は、キー圧縮を使用可能にします。これによって、索引構成表の主キー列の値が重複しなくなります。*integer* を使用して、接頭辞の長さ（圧縮する接頭辞列数）を指定します。

接頭辞の長さの有効範囲は、1 ～（主キー列数 -1）までです。デフォルトでは（主キー列数 -1）になります。

- NOCOMPRESS は、索引構成表でのキー圧縮を使用禁止にします。これはデフォルト値です。

TABLESPACE *tablespace* 再構築した索引構成表を格納する表領域を指定します。

表の移動の制限事項：

- MOVE を指定する場合は、最初の句にする必要があります。この句以外では、*physical_attributes_clause*、*parallel_clause* および *LOB_storage_clause* のみが指定できます。
- LONG または LONG RAW 列を含む表は、移動できません。
- パーティション表（ヒープ表または索引構成表）全体の移動はできません。個々のパーティションまたはサブパーティションを移動してください。

参照： 11-64 ページの「[move_table_partition](#)」および 11-65 ページの「[move_table_subpartition](#)」を参照してください。

LOB についての注意：

move_table_clause で指定するすべての LOB 列については、次のことに注意してください。

- 新しい表領域が指定されていない場合でも、古い LOB データ・セグメントとこれに対応する索引セグメントは削除され、新しいセグメントが作成されます。
 - *table* の LOB 索引がその LOB データと異なる表領域にある場合、移動の後、LOB 索引は LOB データとともに LOB データの表領域にまとめて格納されます。
-
-

enable_disable_clause

enable_disable_clause を使用すると、Oracle が整合性制約を適用するかどうか、およびその方法を指定できます。DROP および KEEP 句は、一意制約または主キー制約を使用禁止にする場合のみに有効です。

参照：

- この文に関連する注意および制限事項については、15-52 ページの CREATE TABLE の「[enable_disable_clause](#)」を参照してください。
- 索引の使用方法与制約の適用方法の詳細は、7-21 ページの「[索引による制約の適用](#)」を参照してください。

TABLE LOCK

DDL 操作中にロックされた表にのみ DDL 操作を実行できます。このような表ロックは、DML 操作中には必要ありません。

注意： 一時表に表ロックを適用することはできません。

ENABLE TABLE LOCK `ENABLE TABLE LOCK` を指定して表ロックを有効にすることによって、表に対する DDL 操作が可能になります。

DISABLE TABLE LOCK `DISABLE TABLE LOCK` を指定して表ロックを無効にすることによって、表に対する DDL 操作が不可能になります。

ALL TRIGGERS

`ALL TRIGGERS` 句を使用すると、表に関連するすべてのトリガーを使用可能または使用禁止にできます。

ENABLE ALL TRIGGERS `ENABLE ALL TRIGGERS` を指定して、表に関連するすべてのトリガーを使用可能にします。トリガー条件が満たされた場合に、トリガーが起動されます。

単一トリガーを使用可能にする場合は、`ALTER TRIGGER` の `enable_clause` を使用してください。

参照： 15-89 ページの「[CREATE TRIGGER](#)」、12-2 ページの「[ALTER TRIGGER](#)」および 11-87 ページの「[トリガーを使用可能にする例:](#)」を参照してください。

DISABLE ALL TRIGGERS `DISABLE ALL TRIGGERS` を指定して、表に関連するすべてのトリガーを使用禁止にします。トリガー条件が満たされた場合でも、使用禁止のトリガーは起動されません。

例

コレクションの取出し例： 次の文は、サンプル表 `sh.print_media` のネストした表の列 `ad_textdocs_ntab` を変更し、問合せ時にロケータのかわりに実値を戻します。

```
ALTER TABLE print_media MODIFY NESTED TABLE ad_textdocs_ntab
RETURN AS VALUE;
```

パラレル処理の指定例： 次の文は、サンプル表 `oe.customers` への問合せに対してパラレル処理を指定します。

```
ALTER TABLE customers
  PARALLEL;
```

制約状態の変更例： 次の文は、`employees` 表の `emp_manager_fk` という名前の整合性制約を `ENABLE VALIDATE` 状態にします。

```
ALTER TABLE employees
  ENABLE VALIDATE CONSTRAINT emp_manager_fk
  EXCEPTIONS INTO exceptions;
```

Oracle が制約を使用可能にするためには、`employees` 表の各行がこの制約を満たしている必要があります。制約に違反する行があれば、制約は使用禁止のままになります。すべての例外は、`exceptions` 表に記述されます。次の文で、`employees` 表の例外を検出することもできます。

```
SELECT employees.*
  FROM employees e, exceptions ex
 WHERE e.row_id = ex.row_id
       AND ex.table_name = 'EMPLOYEES'
       AND ex.constraint = 'EMP_MANAGER_FK';
```

次の文は、`employees` 表の 2 つの制約を `ENABLE NOVALIDATE` 状態にします。

```
ALTER TABLE employees
  ENABLE NOVALIDATE PRIMARY KEY
  ENABLE NOVALIDATE CONSTRAINT emp_last_name_nn;
```

この文には、次の 2 つの `ENABLE` 句が含まれています。

- 1 番目の `ENABLE` 句は、表の主キー制約を `ENABLE NOVALIDATE` 状態にします。
- 2 番目の `ENABLE` 句は、`emp_last_name_nn` という制約を `ENABLE NOVALIDATE` 状態にします。

この例では、表のそれぞれの行が 2 つの制約を満たす場合にかぎり、その制約が使用可能になります。どちらかの制約に違反する行があった場合、エラーが戻され、どちらの制約も使用禁止のままになります。

`phone_calls` 表の `areaco` 列および `phoneno` 列の組合せに対する外部キーを使用した参照整合性制約があるとします。外部キーは `customers` 表の `areaco` 列および `phoneno` 列にある一意キーを参照します。次の文は、`customers` 表の `areaco` 列および `phoneno` 列にある一意キーを使用禁止にします。

```
ALTER TABLE phone_calls
  DISABLE UNIQUE (areaco, phoneno) CASCADE;
```

customers 表の一意キーは、phone_calls 表の外部キーによって参照されるため、この一意キーを使用禁止にする場合は、CASCADE 句を使用します。この句によって、外部キーも使用禁止になります。

索引構成表の例外表の作成例： 次の例は、主キー制約に違反する索引構成表 hr.countries からの行を保持する except_table 表を作成します。

```
EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE ('hr', 'countries', 'except_table');
```

```
ALTER TABLE countries
  ENABLE PRIMARY KEY
  EXCEPTIONS INTO except_table;
```

例外表を指定する場合は、この表に行を挿入する権限が必要です。検出された例外を調べる場合、例外表を問い合わせる権限が必要です。

参照：

- 17-52 ページの「[INSERT](#)」を参照してください。
- 表に行を挿入するために必要な権限の詳細は、18-4 ページの「[SELECT](#)」を参照してください。

CHECK 制約を使用禁止にする例： 次の文は、employees 表に CHECK 制約を定義し、その制約を使用禁止にします。

```
ALTER TABLE employees ADD CONSTRAINT check_comp
  CHECK (salary + (commission_pct*salary) <= 5000)
  DISABLE;
```

check_comp 制約は、給与総額が 5000 ドルを超える従業員がいないことを保証します。ただし、この制約が使用禁止になっているため、従業員の給与をこの制限以上に増やすことができます。

トリガーを使用可能にする例： 次の文は、employees 表に対応付けられているすべてのトリガーを使用可能にします。

```
ALTER TABLE employees
  ENABLE ALL TRIGGERS;
```

未使用領域の解放例： 次の文は、employees 表で再利用できるように最高水位標が MINEXTENTS を超えるすべての未使用領域を解放します。

```
ALTER TABLE employees
  DEALLOCATE UNUSED;
```

列名の変更例： 次の文は、サンプル表 `oe.customers` の列名を `credit_limit` から `credit_amount` に変更します。

```
ALTER TABLE customers
    RENAME COLUMN credit_limit TO credit_amount;
```

列の削除例： 次の文は、`CASCADE CONSTRAINTS` が指定されている `drop_column_clause` です。表 `t1` が次のように作成されているとします。

```
CREATE TABLE t1 (
    pk NUMBER PRIMARY KEY,
    fk NUMBER,
    c1 NUMBER,
    c2 NUMBER,
    CONSTRAINT ri FOREIGN KEY (fk) REFERENCES t1,
    CONSTRAINT ck1 CHECK (pk > 0 and c1 > 0),
    CONSTRAINT ck2 CHECK (c2 > 0)
);
```

次の文に対してエラーが戻されます。

```
/* The next two statements return errors:
ALTER TABLE t1 DROP (pk); -- pk is a parent key
ALTER TABLE t1 DROP (c1); -- c1 is referenced by multicolumn
                        -- constraint ck1
```

次の文を発行すると、列 `pk`、主キー制約、外部キー制約 `ri` および `CHECK` 制約 `ck1` が削除されます。

```
ALTER TABLE t1 DROP (pk) CASCADE CONSTRAINTS;
```

削除された列に定義した制約が参照する列もすべて削除される場合、`CASCADE CONSTRAINTS` は必要ありません。たとえば、他の表から列 `pk` を参照する他の参照制約が存在していないとします。この場合は、`CASCADE CONSTRAINTS` 句を指定しない次の文が有効になります。

```
ALTER TABLE t1 DROP (pk, fk, c1);
```

索引構成表の変更例： 次の文は、索引構成表 `hr.countries` の索引セグメントの `INITRANS` パラメータを変更します。

```
ALTER TABLE countries INITRANS 4;
```

次の文では、オーバーフロー・データ・セグメントを索引構成表 `countries` に追加します。

```
ALTER TABLE countries ADD OVERFLOW;
```

次の文は、索引構成表 `countries` のオーバーフロー・データ・セグメントの `INITTRANS` パラメータを変更します。

```
ALTER TABLE countries OVERFLOW INITTRANS 4;
```

表のパーティションの分割例： 次の文は、サンプル表 `sh.sales` の古いパーティション `sales_q4_2000` を分割して 2 つの新しいパーティションを作成し、1 つには `sales_q4_2000b` という名前を付け、もう 1 つには旧パーティションの名前を再利用します。

```
ALTER TABLE sales SPLIT PARTITION SALES_Q4_2000
  AT (TO_DATE('15-NOV-2000','DD-MON-YYYY'))
  INTO (PARTITION SALES_Q4_2000, PARTITION SALES_Q4_2000b);
```

サンプル表 `pm.print_media` は、パーティション `p1` とパーティション `p2` にレンジ・パーティション化されます (表をパーティション化する前に、`print_media` の `LONG` 列を `LOB` に変換する必要があります)。次の文は、表のパーティション `p2` をパーティション `p2a` と `p2b` に分割します。

```
ALTER TABLE print_media_part
  SPLIT PARTITION p2 AT (150) INTO
  (PARTITION p2a TABLESPACE omf_ts1
    (LOB ad_photo, ad_composite) STORE AS (TABLESPACE omf_ts2),
  PARTITION p2b
    (LOB (ad_photo, ad_composite) STORE AS (TABLESPACE omf_ts2)));
```

`p2a` と `p2b` のパーティションでは、列 `ad_photo` と `ad_composite` に対する `LOB` セグメントが表領域 `omb_ts2` 内に作成されます。パーティション `p2a` のその他の列に対する `LOB` セグメントは、表領域 `omf_ts1` に保存されます。パーティション `p2b` のその他の列に対する `LOB` セグメントは、この `ALTER` 文の実行前の表領域で保持されます。ただし、`LOB` データおよび `LOB` 索引セグメントが新しい表領域に移動されない場合でも、これらの新しいセグメントが作成されます。

LOB を持つ表パーティションの追加例： 次の文は、パーティション `p3` を `print_media_part` 表に追加し (前述の例を参照)、表の `BLOB` と `CLOB` 列の記憶特性を指定します。

```
ALTER TABLE print_media_part ADD PARTITION p3 VALUES LESS THAN (MAXVALUE)
  (LOB (ad_photo, ad_composite) STORE AS (TABLESPACE omf_ts2)
  (LOB (ad_sourcetext, ad_finaltext) STORE AS (TABLESPACE omf_ts1));
```

パーティション `p3` の列 `ad_photo` および `ad_composite` に対する `LOB` データと `LOB` 索引セグメントは、表領域 `omf_ts2` に格納されます。`LOB` 列の他の属性は、まず表レベルのデフォルトから継承され、次に表領域のデフォルトから継承されます。

列 `ad_source_text` および `ad_finaltext` の LOB データ・セグメントは、`omf_ts1` 表領域に格納され、他のすべての属性は、表レベルのデフォルト値から継承され、次に表領域のデフォルト値から継承されます。

デフォルト・リスト・パーティションの使用例： 次の文は、リスト・パーティション表 (15-69 ページの「[リスト・パーティション化の例 :](#)」で作成) を使用します。最初の文は、既存のデフォルト・パーティションを新規の `south` パーティションとデフォルト・パーティションに分割します。

```
ALTER TABLE list_customers SPLIT PARTITION rest
VALUES ('MEXICO', 'COLOMBIA')
INTO (PARTITION south, PARTITION rest);
```

次の文は、結果のデフォルト・パーティションを `asia` パーティションとマージします。

```
ALTER TABLE list_customers
MERGE PARTITIONS asia, rest INTO PARTITION rest;
```

次の文は、デフォルト・パーティションを分割して、`asia` パーティションを再作成します。

```
ALTER TABLE list_customers SPLIT PARTITION rest
VALUES ('CHINA', 'THAILAND')
INTO (PARTITION east, partition rest);
```

2 つの表パーティションのマージ例： 次の文は、パーティション (11-89 ページの「[表のパーティションの分割例 :](#)」で作成) をマージして、1 つのパーティションに戻します。

```
ALTER TABLE sales
MERGE PARTITIONS sales_q4_2000, sales_q4_2000b
INTO PARTITION sales_q4_2000;
```

表のパーティションの削除例： 次の文は、パーティション `p3` (11-89 ページの「[LOB を持つ表パーティションの追加例 :](#)」で作成) を削除します。

```
ALTER TABLE print_media_part DROP PARTITION p3;
```

表パーティションの交換例： 次の文は、パーティション `feb97` を表 `sales_feb97` に変換します。ローカル索引パーティションと `sales_feb97` に対応する索引との交換、および `sales_feb97` 内のデータがパーティション `feb97` の範囲内かどうかの検証は行われません。

```
ALTER TABLE sales
EXCHANGE PARTITION feb97 WITH TABLE sales_feb97
WITHOUT VALIDATION;
```

表のパーティションの変更例： 次の文は、`sales` 表のパーティション `nov96` に対応するすべてのローカル索引パーティションに、`UNUSABLE` のマークを付けます。

```
ALTER TABLE sales MODIFY PARTITION nov96
UNUSABLE LOCAL INDEXES;
```

次の文は、UNUSABLE のマークが付けられたすべてのローカル索引パーティションを再構築します。

```
ALTER TABLE sales MODIFY PARTITION jan97
REBUILD UNUSABLE LOCAL INDEXES;
```

次の文は、パーティション branch_ny の MAXEXTENTS およびロギング属性を変更します。

```
ALTER TABLE branch MODIFY PARTITION branch_ny
STORAGE (MAXEXTENTS 75) LOGGING;
```

表のパーティションの移動例： 次の文は、11-89 ページの「[表のパーティションの分割例](#)」で作成したパーティション p2b を表領域 omf_ts1: に移動します。

```
ALTER TABLE print_media_part
MOVE PARTITION p2b TABLESPACE omf_ts1;
```

表のパーティション名の変更例： 次の文は、表の名前を変更します。

```
ALTER TABLE employees RENAME TO employee;
```

次の文では、パーティション emp3 の名前が変更されます。

```
ALTER TABLE employee RENAME PARTITION emp3 TO employee3;
```

表のパーティションの切捨て例： 次の文は、パーティション sys_p017 内のすべてのデータを削除し、解放された領域の割当てを解除します。

```
ALTER TABLE deliveries
TRUNCATE PARTITION sys_p017 DROP STORAGE;
```

グローバル索引の更新例： 次の文は、サンプル表 sh.sales のパーティション sales_q1_2000 を分割し、定義されているグローバル索引を更新します。

```
ALTER TABLE sales SPLIT PARTITION sales_q1_2000
AT (TO_DATE('16-FEB-2000', 'DD-MON-YYYY'))
INTO (PARTITION q1a_2000, PARTITION q1b_2000)
UPDATE GLOBAL INDEXES;
```

オブジェクト識別子の指定例： 次の文は、オブジェクト型、主キーに基づくオブジェクト識別子に対応するオブジェクト表、およびユーザー定義 REF 列を持つ表を作成します。

```
CREATE TYPE emp_t AS OBJECT (empno NUMBER, address CHAR(30));
```

```
CREATE TABLE emp OF emp_t (
```

```
empno PRIMARY KEY)  
OBJECT IDENTIFIER IS PRIMARY KEY;
```

```
CREATE TABLE dept (dno NUMBER, mgr_ref REF emp_t SCOPE is emp);
```

次の文は、emp 表を参照する制約およびユーザー定義 REF 列を追加します。

```
ALTER TABLE dept ADD CONSTRAINT mgr_cons FOREIGN KEY (mgr_ref)  
REFERENCES emp;  
ALTER TABLE dept ADD sr_mgr REF emp_t REFERENCES emp;
```

表の列の追加例： 次の文は、NUMBER データ型の列 duty_pct、サイズが 3 の VARCHAR2 データ型の列 visa_needed (yes または no を格納) および CHECK 整合性制約を追加します。

```
ALTER TABLE countries  
ADD (duty_pct NUMBER(2,2) CHECK (duty_pct < 10.5),  
visa_needed VARCHAR2(3));
```

表の列の変更例： 次の文は、duty_pct 列のサイズを増やします。

```
ALTER TABLE countries  
MODIFY (duty_pct NUMBER(3,2));
```

MODIFY 句には列の定義が 1 つのため、定義を囲むカッコは任意指定です。

次の文は、employees 表の PCTFREE パラメータと PCTUSED パラメータの値を、それぞれ 30 と 60 に変更します。

```
ALTER TABLE employees  
PCTFREE 30  
PCTUSED 60;
```

LONG 列から LOB への変換例： 次の例は、サンプル表 pm.print_media の press_release 列のデータ型を LONG から CLOB に変更します。

```
ALTER TABLE print_media MODIFY (press_release CLOB);
```

エクステントの割当て例： 次の文は、employees 表に 5KB のエクステントを割り当て、そのエクステントをインスタンス 4 が使用できるようにします。

```
ALTER TABLE employees  
ALLOCATE EXTENT (SIZE 5K INSTANCE 4);
```

この文には、DATAFILE パラメータが指定されていないため、エクステントは employees 表が入っている表領域に属するデータ・ファイルの 1 つに割り当てられます。

デフォルト列値の指定例： 次の文は、product_information 表の min_price 列のデフォルト値を 10 に変更します。

```
ALTER TABLE product_information
  MODIFY (min_price DEFAULT 10);
```

続いて product_information 表に値を指定せずに新しい列 min_price を追加する場合、min_price 列の値は自動的に 0（ゼロ）になります。

```
INSERT INTO product_information (product_id, product_name,
  list_price)
  VALUES (300, 'left-handed mouse', 40.50);
```

```
SELECT product_id, product_name, list_price, min_price
  FROM product_information
 WHERE product_id = 300;
```

PRODUCT_ID	PRODUCT_NAME	LIST_PRICE	MIN_PRICE
300	left-handed mouse	40.5	10

以前に指定したデフォルト値を中止して、新しく追加する行にその値が自動的に挿入されないようにする場合、次の文に示すように、デフォルト値を NULL に置き換えます。

```
ALTER TABLE product_information
  MODIFY (min_price DEFAULT NULL);
```

MODIFY 句には、列の定義をすべて指定する必要はありません。列名および変更部分のみを指定してください。この文は、既存の行の既存の値には影響しません。

XMLType 表への制約の追加例： 次の例は、xwarehouses 表（15-67 ページの「[XMLType 表の例](#)」で作成）に主キー制約を追加します。

```
ALTER TABLE xwarehouses
  ADD (PRIMARY KEY(XMLDATA."WarehouseID"));
```

参照： この擬似例は、2-84 ページの「[XMLDATA](#)」を参照してください。

制約名の変更例： 次の文は、サンプル表 oe.customers の制約名を cust_fname_nn から cust_firstname_nn に変更します。

```
ALTER TABLE customers RENAME CONSTRAINT cust_fname_nn
  TO cust_firstname_nn;
```

制約の削除例： 次の文は、departments 表の主キーを削除します。

```
ALTER TABLE departments
  DROP PRIMARY KEY CASCADE;
```

主キー制約の名前が pk_dept であることがわかっている場合は、次のように指定しても削除できます。

```
ALTER TABLE departments
  DROP CONSTRAINT pk_dept CASCADE;
```

CASCADE 句によって、主キーを参照するすべての外部キーが削除されます。

次の文は、employees 表の email 列の一意キーを削除します。

```
ALTER TABLE employees
  DROP UNIQUE (email);
```

この文の DROP 句では CASCADE 句を省略します。CASCADE オプションを省略することによって、一意キーを参照する外部キーがある場合、その一意キーは削除されません。

LOB 列の例： 次の文は、CLOB 列の resume を employee 表に追加し、新しい列の LOB 記憶特性を指定します。

```
ALTER TABLE employees ADD (resume CLOB)
  LOB (resume) STORE AS resume_seg (TABLESPACE example);
```

次の文を入力して、キャッシュを使用できるように LOB 列の resume を変更します。

```
ALTER TABLE employees MODIFY LOB (resume) (CACHE);
```

ネストした表の例： 次の文は、ネストした表の列 skills を employees 表に追加します。

```
ALTER TABLE employees ADD (skills skill_table_type)
  NESTED TABLE skills STORE AS nested_skill_table;
```

また、ネストした表の記憶特性も変更できます。変更する場合、*nested_table_col_properties* に指定した記憶表の名前を使用してください。記憶表では、DML 文の問合せまたは実行はできません。記憶表は、ネストした表の列の記憶特性を変更するためにのみ使用します。

次の文は、ネストした表の列 client と記憶表 client_tab を使用して、表 vetservice を作成します。ネストした表 vetservice を変更して制約を指定します。

```
CREATE TYPE pet_table AS OBJECT
  (pet_name VARCHAR2(10), pet_dob DATE);

CREATE TABLE vetservice (vet_name VARCHAR2(30),
  client pet_table)
  NESTED TABLE client STORE AS client_tab;
```

```
ALTER TABLE client_tab ADD UNIQUE (ssn);
```

次の文は、ネストした表 `nested_skill_table` に、一意制約を追加します。

```
ALTER TABLE nested_skill_table ADD UNIQUE (a);
```

次の文は、REF 値のネストした表用の記憶表を変更して、REF の範囲が限定されることを指定します。

```
CREATE TYPE emp_t AS OBJECT (eno number, ename char(31));
CREATE TYPE emps_t AS TABLE OF REF emp_t;
CREATE TABLE emptab OF emp_t;
CREATE TABLE dept (dno NUMBER, employees emps_t)
  NESTED TABLE employees STORE AS deptemps;
ALTER TABLE deptemps ADD (SCOPE FOR (column_value) IS emptab);
```

同様に、次の文は、REF を ROWID とともに格納することを指定します。

```
ALTER TABLE deptemps ADD (REF(column_value) WITH ROWID);
```

これらの ALTER TABLE 文を正確に実行するためには、記憶表 `deptemps` が空である必要があります。また、ネストした表は、スカラー (REF) 表として定義されるため、Oracle は、暗黙的に列名 `COLUMN_VALUE` を記憶表に設定します。

参照：

- ネストした表の記憶域の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。
- ネストした表の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

REF 列の例： 次の文は、オブジェクト型 `dept_t` があらかじめ定義されています。次のように `staff` 表を作成します。

```
CREATE TABLE staff
  (name  VARCHAR(100),
   salary NUMBER,
   dept  REF dept_t);
```

オブジェクト表 `offices` を次のように作成します。

```
CREATE TABLE offices OF dept_t;
```

`dept` 列は、任意の表に格納された `dept_t` のオブジェクトに参照を格納できます。次のように `dept` 列に有効範囲制約を追加することによって、`departments` 表に格納されたオブジェクトのみが参照されるように制限できます。

```
ALTER TABLE staff  
  ADD (SCOPE FOR (dept) IS offices);
```

前述の ALTER TABLE 文は、staff 表が空である場合のみ正常に実行されます。

次の文は、staff の dept 列に REF 値を格納する際、ROWID も同時に格納します。

```
ALTER TABLE staff  
  ADD (REF(dept) WITH ROWID);
```

追加の例： ALTER TABLE 文を使用した整合性制約の定義例は、7-5 ページの「[constraints](#)」を参照してください。

表の記憶域パラメータ値の変更例は、7-51 ページの「[storage_clause](#)」を参照してください。

ALTER TABLESPACE

用途

ALTER TABLESPACE を使用すると、既存の表領域、1 つ以上のデータ・ファイルまたは一時ファイルを変更できます。

この句では、ディクショナリ管理表領域をローカル管理表領域に変換することはできません。変換するには、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』に記載されている DBMS_SPACE_ADMIN パッケージを使用してください。

参照： 表領域作成の詳細は、『Oracle9i データベース管理者ガイド』および 15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

前提条件

ALTER TABLESPACE システム権限を持っている場合、この文のすべての操作を実行できます。MANAGE TABLESPACE システム権限を持っている場合は、次の操作のみを実行できます。

- 表領域をオンラインまたはオフラインにする。
- バックアップを開始または終了する。
- 表領域を読み専用または読み書き両用にする。

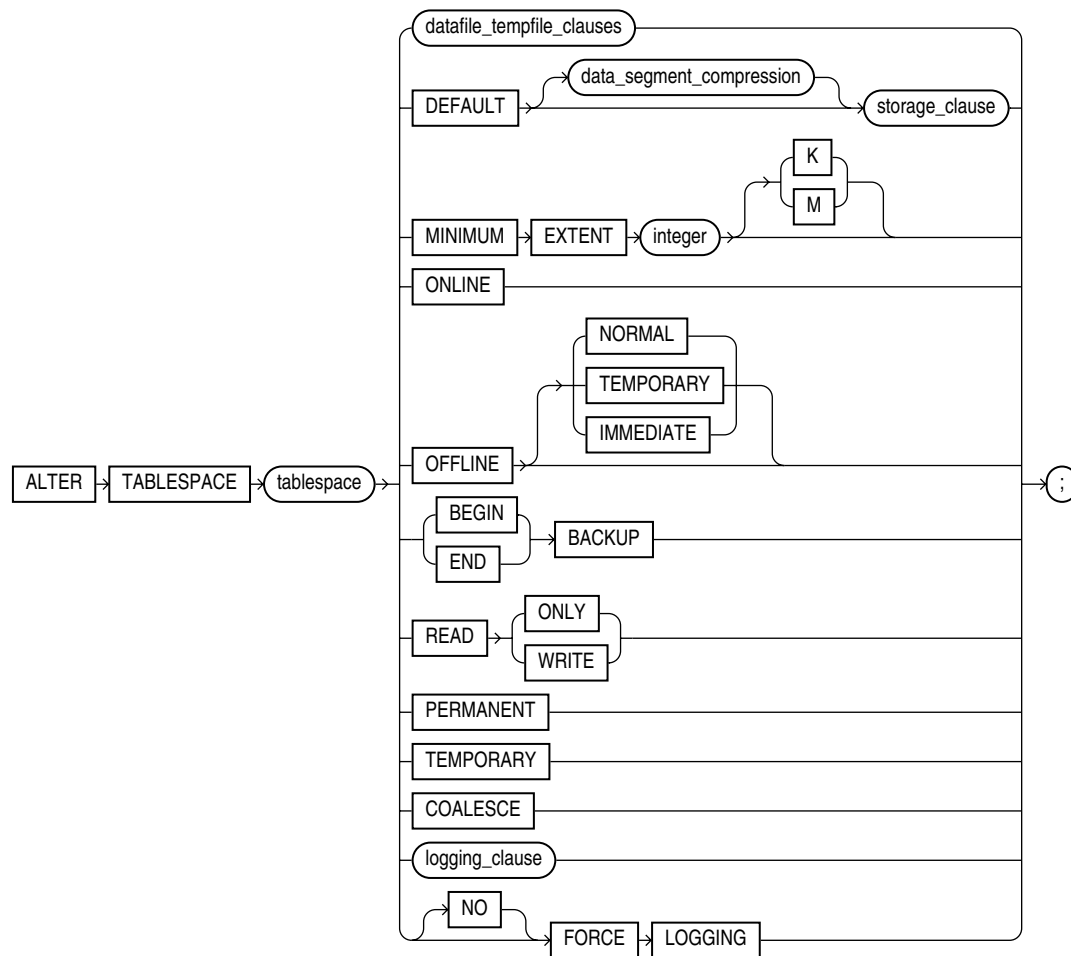
表領域を読み専用にする場合、次の条件が満たされている必要があります。

- 表領域がオンラインになっている。
- 表領域にアクティブなロールバック・セグメントがない。SYSTEM 表領域には SYSTEM ロールバック・セグメントがあるため、読み専用にはできません。また、読み専用表領域のロールバック・セグメントにはアクセスできないため、ロールバック・セグメントを削除してから、表領域を読み専用にすることをお勧めします。
- 表領域がオープン・バックアップに使用されていない。バックアップの終わりに表領域内のすべてのデータ・ファイルのヘッダー・ファイルが更新されるためです。

これらの条件を満たす場合、制限モードでこの機能を実行すると便利です。制限モードでは、RESTRICTED SESSION システム権限を持つユーザーのみがログインできます。

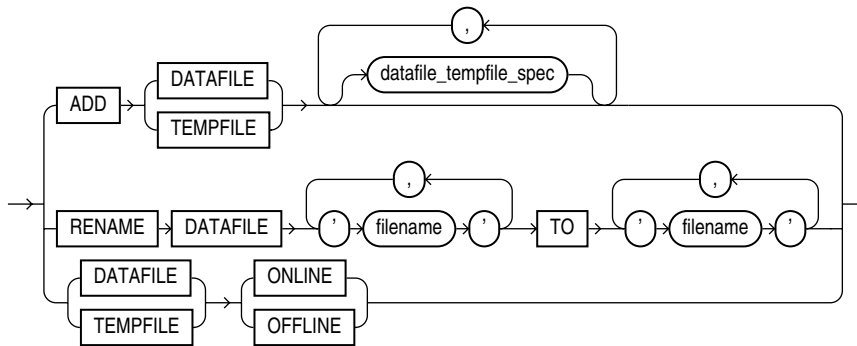
構文

alter_tablespace::=



(11-99 ページの「[datafile_tempfile_clauses::=](#)」、11-5 ページの「[data_segment_compression::=](#)」 (ALTER TABLE 構文の一部)、7-53 ページの「[storage_clause::=](#)」 および 7-42 ページの「[logging_clause::=](#)」を参照)

datafile_tempfile_clauses::=



(7-36 ページの「[datafile_tempfile_spec::=](#)」([file_specification](#)の一部)を参照)

セマンティクス

tablespace

変更する表領域の名前を指定します。

表領域の変更の制限事項：

- `tablespace` が UNDO 表領域の場合、この文では ADD DATAFILE、RENAME DATAFILE、DATAFILE ... ONLINE | OFFLINE および BEGIN | END BACKUP のみが指定可能です。
- ローカル管理の一時表領域に対してこの文で指定できるのは、ADD 句のみです。

参照： 自動 UNDO 管理および UNDO 表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

datafile_tempfile_clauses

`datafile_tempfile_clauses` を使用すると、データ・ファイルまたは一時ファイルを追加および変更できます。

ADD DATAFILE | TEMPFILE 句

ADD を指定すると、`datafile_tempfile_spec` によって指定されたデータ・ファイルまたは一時ファイルを追加できます。

この句は、ローカル管理の一時表領域に対して、どんな場合でも指定できる唯一の句です。

`datafile_tempfile_spec` を指定しないと、AUTOEXTEND が有効になった 100MB の Oracle 管理ファイルが作成されます。

データ・ファイルまたは一時ファイルを、オンラインのローカル管理表領域、またはオンラインまたはオフラインのディクショナリ管理表領域に追加できます。なお、そのデータ・ファイルが別のデータベースで使用中でないことを確認してください。

注意： オペレーティング・システムによっては、一時ファイルのブロックが実際にアクセスされるまで、一時ファイル用の領域が割り当てられない場合があります。領域の割当ての遅延のため、一時ファイルの作成およびサイズ変更が速くなります。ただし、後で一時ファイルが使用されるときに、十分なディスク領域を使用可能にする必要があります。発生する可能性がある問題を回避するには、一時ファイルを作成またはサイズ変更する前に、ディスク領域が、新しく作成する一時ファイルまたはサイズ変更後の一時ファイルのサイズより大きいことを確認してください。ディスク領域に余裕を持たせることによって、関連のない操作が原因で増加が予想されるディスク使用量に対応できます。十分な領域があることを確認した後で、作成またはサイズ変更操作を実行してください。

参照： 7-36 ページの「[file_specification](#)」、11-105 ページの「[データ・ファイルの追加例](#)」および 11-106 ページの「[Oracle 管理データ・ファイルの追加例](#)」を参照してください。

RENAME DATAFILE 句

1 つ以上の表領域のデータ・ファイルの名前を変更する場合は、RENAME DATAFILE で指定します。データベースをオープンしておくこと、および名前の変更前に表領域をオフラインにすることが必要です。それぞれの '`filename`' には、ご使用のオペレーティング・システムのファイル名の表記規則に従って、データ・ファイル名を完全に指定してください。

この句では、表領域を古いファイルではなく新しいファイルに対応付けます。オペレーティング・システムのファイル名は実際には変更されません。このため、オペレーティング・システム上でこのファイル名を変更する必要があります。

参照： 11-105 ページの「[表の移動および名前の変更例](#)」を参照してください。

DATAFILE | TEMPFILE ONLINE | OFFLINE

この句を使用すると、表領域のすべてのデータ・ファイルまたは一時ファイルを、オフラインまたはオンラインにできます。この句は、表領域の ONLINE/OFFLINE 状態には影響しません。

データベースは、マウントされている必要があります。表領域が SYSTEM、UNDO 表領域、またはデフォルトの一時表領域の場合、データベースをオープンしないでおく必要があります。

DEFAULT *storage_clause*

DEFAULT *storage_clause* には、表領域に作成される後続のオブジェクトに対する新しいデフォルトの記憶域パラメータを指定します。ディクショナリ管理の一時表の場合は、*storage_clause* の NEXT パラメータのみが考慮されます。

表領域のデフォルト記憶域の制限事項： この句は、ローカル管理表領域に対して指定できません。

参照： 7-51 ページの「[storage_clause](#)」を参照してください。

MINIMUM EXTENT

MINIMUM EXTENT 句を指定すると、表領域内のすべての使用済エクステンツまたは未使用エクステンツの大きさが、*integer* で指定したサイズ以上であること、およびその倍数であることが保証され、表領域における空き領域の断片化を制御できます。ディクショナリ管理の一時表領域の場合、この句は関係ありません。

MINIMUM EXTENT の制限事項： この句は、ローカル管理表領域に対して指定できません。

参照： MINIMUM EXTENT を使用した断片化の制御については、『Oracle9i データベース管理者ガイド』を参照してください。また、11-106 ページの「[表領域のロギング属性の変更例 :](#)」も参照してください。

ONLINE

ONLINE を指定すると、表領域をオンラインにできます。

OFFLINE

OFFLINE を指定する表領域をオフラインにし、そのセグメントにアクセスできないようにできます。表領域をオフラインにすると、そのすべてのデータ・ファイルもオフラインになります。

提案： 表領域を長期間オフラインにする前に、デフォルト表領域または一時表領域としてその表領域を割り当てられているユーザーの表領域割当てを変更した方がよい場合があります。表領域をオフラインにした場合、これらのユーザーは、その表領域内でオブジェクトに対して領域を割り当てたり、領域をソートすることはできません。ユーザーへの表領域の割当ての詳細は、12-21 ページの「[ALTER USER](#)」を参照してください。

表領域をオフラインにする場合の制限事項： 一時表領域はオフラインにできません。

NORMAL NORMAL を指定すると、SGA 以外にある表領域のすべてのデータ・ファイルにあるすべてのブロックがフラッシュされます。データ・ファイルをオンラインに戻す前に、表領域のメディア・リカバリを行う必要はありません。これはデフォルト値です。

TEMPORARY TEMPORARY を指定すると、Oracle は表領域内のすべてのオンライン・データ・ファイルに対してチェックポイントを実行しますが、すべてのファイルに書き込むことができることを保証しません。この表領域をオンラインに戻す前に、オフライン・ファイルのメディア・リカバリを行う必要があります。

IMMEDIATE IMMEDIATE を指定すると、Oracle は表領域のファイルが使用可能であることを保証しません。また、チェックポイントも実行しません。表領域をオンラインに戻す前に、メディア・リカバリを行う必要があります。

注意： ALTER TABLESPACE ... OFFLINE に対する FOR RECOVER 設定は、使用しないことをお勧めします。この構文は下位互換用にのみサポートされていますが、表領域のリカバリにはトランスポータブル表領域機能を使用することをお勧めします。

参照： メディア・リカバリを実行するトランスポータブル表領域の使用の詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

BEGIN BACKUP

BEGIN BACKUP プロシージャは、表領域を構成するデータ・ファイルのオープン・バックアップを実行することを示します。この句を指定することによって、ユーザーがこの表領域にアクセスできなくなることはありません。オープン・バックアップを開始する前に、この句を指定してください。

表領域のバックアップ開始の制限事項： この句は、読み込み専用の表領域またはローカル管理の一時表領域に対して指定できません。

注意： バックアップ中は、表領域の標準オフラインへの切替え、インスタンスの停止または表領域の別のバックアップ処理の開始は実行できません。

参照： 11-105 ページの「[表領域のバック・アップ例:](#)」を参照してください。

END BACKUP

END BACKUP は、表領域のオンライン・バックアップが完了したことを示します。オンライン・バックアップの完了後、できるだけ早くこの句を指定してください。インスタンスに障害または SHUTDOWN ABORT が発生した場合、次のインスタンス起動時にメディア・リカバリ（必要に応じて、アーカイブ REDO ログも）が必要であるとみなされます。

表領域のバックアップ終了の制限事項： この句は、読み専用表領域に対して使用できません。

参照：

- メディア・リカバリなしでデータベースを再起動する場合の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- 個々のデータ・ファイルまたは表領域のすべてのデータ・ファイルをオンライン（ホット）・バックアップ・モードから解除する場合の詳細は、9-34 ページの「ALTER DATABASE」の「[END BACKUP 句](#)」を参照してください。

READ ONLY | READ WRITE

READ ONLY によって、表領域を**読み専用推移モード**に設定します。この状態では、既存のトランザクションは完了（コミットまたはロールバック）できますが、以前の表領域内のブロックを変更した既存のトランザクションのロールバック以外は、その表領域に対してさらに書き込み操作（DML）を行うことはできません。

表領域を読み専用にした場合、そのファイルを読み専用メディアにコピーできます。その場合、SQL 文の ALTER DATABASE ... RENAME を使用して、新しいファイル位置を示すように制御ファイルのデータ・ファイルの名前を変更する必要があります。

参照：

- 読み専用の表領域の詳細は、『Oracle9i データベース概要』を参照してください。
- 9-12 ページの「[ALTER DATABASE](#)」を参照してください。

読み専用指定されている表領域に対して書き込み操作を可能にする場合は、READ WRITE を指定します。

PERMANENT | TEMPORARY

一時表領域を永続表領域に変換する場合は、PERMANENT を指定します。永続表領域とは、永続的なデータベース・オブジェクトを格納できる場所です。表領域を作成するときのデフォルトです。

永続表領域を一時表領域に変換する場合は、TEMPORARY を指定します。一時表領域とは、永続的なデータベース・オブジェクトを格納できない表領域です。一時表領域の中のオブジェクトはセッション中のみ保持されます。

一時表領域の制限事項：

- 表領域を標準的なブロック・サイズで作成しなかった場合、永続表領域を一時表領域に変換できません。
- FORCE LOGGING モードでは、表領域に対して TEMPORARY を指定できません。

COALESCE

この句は、表領域内の各データ・ファイルについて、連続する未使用エクステントをすべて結合して大きい連続エクステントにします。

logging_clause

表領域内のすべての表、索引およびパーティションのロギング属性を指定する場合、LOGGING を指定します。表レベル、索引レベルおよびパーティション・レベルでのロギング指定によって、表領域レベルのロギング属性を上書きできます。

既存の表領域のロギング属性を ALTER TABLESPACE 文によって変更した場合、この文の実行後に作成されたすべての表、索引およびパーティションに、新しいデフォルトのロギング属性（これは後で上書きもできます）が適用されます。既存のオブジェクトのロギング属性は変更されません。

FORCE LOGGING モードの表領域がある場合、この文で NOLOGGING を指定すると、表領域のデフォルト・ロギング・モードを NOLOGGING に設定できます。ただし、この設定は FORCE LOGGING モード以外の表領域には適用されません。

[NO] FORCE LOGGING

この句を使用すると、表領域で強制ロギング・モードを有効または無効にできます。データベースをオープンし、READ WRITE モードにしておく必要があります。この設定により、表領域のデフォルト LOGGING モードまたは NOLOGGING モードは変更されません。

強制ロギング・モードの制限事項： FORCE LOGGING は、UNDO 表領域および一時表領域に対して指定できません。

参照： FORCE LOGGING モードの使用の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。また、11-106 ページの「[表領域のロギング属性の変更例：](#)」を参照してください。

例

表領域のバック・アップ例： 次の文は、バックアップの開始をデータベースに通知します。

```
ALTER TABLESPACE tbs_01
  BEGIN BACKUP;
```

次の文は、バックアップが終了したことをデータベースに通知します。

```
ALTER TABLESPACE tbs_01
  END BACKUP;
```

表の移動および名前の変更例： 次に、tbs_01 表領域に割り当てられたデータ・ファイル名を、diskb:tbs_f5.dat から diska:tbs_f5.dat に変更する例を示します。

1. OFFLINE 句を指定した ALTER TABLESPACE 文を使用して、この表領域をオフラインにします。

```
ALTER TABLESPACE tbs_01 OFFLINE NORMAL;
```

2. オペレーティング・システムのコマンドを使用して、このファイルを diskb:tbs_f5.dat から diska:tbs_f5.dat にコピーします。
3. RENAME DATAFILE 句を指定した ALTER TABLESPACE 文を使用して、このデータ・ファイルの名前を変更します。

```
ALTER TABLESPACE tbs_01
  RENAME DATAFILE 'diskb:tbs_f5.dat'
  TO              'diska:tbs_f5.dat';
```

4. ONLINE 句を指定した ALTER TABLESPACE 文を使用して、この表領域をオンラインに戻します。

```
ALTER TABLESPACE tbs_01 ONLINE;
```

データ・ファイルの追加例： 次の文は、表領域にデータ・ファイルを追加します。さらに多くの領域が必要な場合、10KB の新しいエクステンツが最大 100KB まで追加されます。

```
ALTER TABLESPACE tbs_03
  ADD DATAFILE 'tbs_f04.dbf'
  SIZE 50K
  AUTOEXTEND ON
  NEXT 10K
  MAXSIZE 100K;
```

Oracle 管理データ・ファイルの追加例： 次の例は、Oracle 管理データ・ファイルを omf_ts1 表領域に追加します。(表領域の作成の詳細は、15-85 ページの「[Oracle 管理ファイルの作成例](#)」を参照してください。) 新しいデータ・ファイルは 100MB で、最大サイズが制限なしで自動拡張されます。

```
ALTER TABLESPACE omf_ts1 ADD DATAFILE;
```

表領域のログギング属性の変更例： 次の例は、表領域のデフォルトのログギング属性を NOLOGGING に変更します。

```
ALTER TABLESPACE tbs_03 NOLOGGING;
```

表領域のログギング属性を変更した場合でも、その表領域内の既存のスキーマ・オブジェクトのログギング属性には影響しません。表レベル、索引レベルおよびパーティション・レベルでのログギング指定によって、表領域レベルのログギング属性を上書きできます。

表領域のログギング属性の変更例： 次の文は、tbs_03 の各エクステンツの割当てを 128KB の倍数に変更します。

```
ALTER TABLESPACE tbs_03 MINIMUM EXTENT 128K;
```

SQL 文 : ALTER TRIGGER ~ COMMIT

この章では、次の SQL 文について説明します。

- ALTER TRIGGER
- ALTER TYPE
- ALTER USER
- ALTER VIEW
- ANALYZE
- ASSOCIATE STATISTICS
- AUDIT
- CALL
- COMMENT
- COMMIT

ALTER TRIGGER

用途

ALTER TRIGGER を使用すると、データベース・トリガーを使用可能、使用禁止またはコンパイルできます。

注意： この文を使用して既存のトリガーの宣言や定義は変更できません。トリガーを再宣言または再定義する場合は、OR REPLACE キーワードを指定した CREATE TRIGGER 文を使用します。

参照：

- トリガー作成の詳細は、15-89 ページの「[CREATE TRIGGER](#)」を参照してください。
- トリガー削除の詳細は、17-12 ページの「[DROP TRIGGER](#)」を参照してください。
- トリガーの概要は、『Oracle9i データベース概要』を参照してください。

前提条件

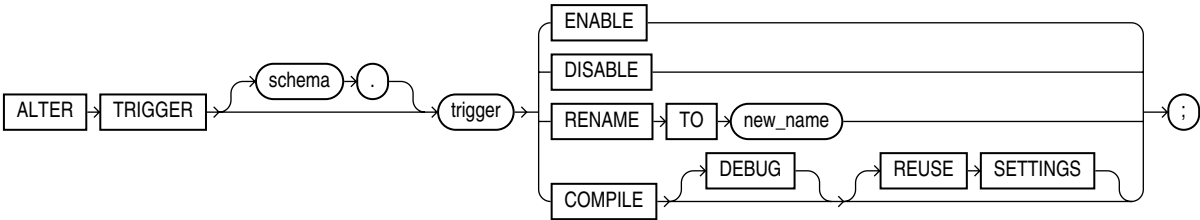
トリガーが自分のスキーマ内にある必要があります。自分のスキーマ内にはない場合は、ALTER ANY TRIGGER システム権限が必要です。

さらに、DATABASE 上のトリガーを変更する場合は、ADMINISTER DATABASE TRIGGER システム権限が必要です。

参照： DATABASE トリガーに基づいたトリガーの詳細は、15-89 ページの「[CREATE TRIGGER](#)」を参照してください。

構文

alter_trigger::=



セマンティクス

schema

削除するトリガーが含まれているスキーマを指定します。*schema* を指定しない場合、トリガーは自分のスキーマ内に定義されているとみなされます。

trigger

変更するトリガーの名前を指定します。

ENABLE 句

ENABLE を指定して、トリガーを使用可能にします。また、ALTER TABLE の ENABLE ALL TRIGGERS 句を使用することによって、表に対応付けられたすべてのトリガーを使用可能にできます。

参照： 11-2 ページの [ALTER TABLE](#) および 12-5 ページの「[トリガーを使用可能にする例](#)」を参照してください。

DISABLE 句

DISABLE を指定して、トリガーを使用禁止にします。また、ALTER TABLE の DISABLE ALL TRIGGERS 句を使用することによって、表に対応付けられたすべてのトリガーを使用禁止にできます。

参照： 11-2 ページの [ALTER TABLE](#) および 12-5 ページの「[トリガーを使用禁止にする例](#)」を参照してください。

RENAME 句

RENAME TO *new_name* を指定して、トリガーの名前を変更します。トリガーの名前は変更され、名前が変更される前と同じ状態になります。

注意： トリガーの名前を変更すると、USER_SOURCE、ALL_SOURCE および DBA_SOURCE データ・ディクショナリ・ビューに記憶されているトリガーのソースが再構築されます。その結果、トリガー・ソースが変更されていないなくても、これらのビューの TEXT 列のコメントおよび書式設定が変更される場合があります。

COMPILE 句

トリガーが使用可能または使用禁止であるかにかかわらず、トリガーを明示的にコンパイルするには、**COMPILE** を指定します。明示的に再コンパイルすることによって、実行時に暗黙的に再コンパイルする必要がなくなり、また、実行時のコンパイル・エラーとパフォーマンス上のオーバーヘッドもなくなります。

トリガーが依存するオブジェクトに無効なオブジェクトがある場合は、最初にそのオブジェクトが再コンパイルされます。トリガーの再コンパイルが正常に終了した場合、このトリガーは使用可能になります。

再コンパイル中、**Oracle** はすべての永続コンパイラのスイッチ設定を削除し、セッションからそれらを再び取得してコンパイルの終了時に格納します。この手順を回避するには、**REUSE SETTINGS** 句を指定します。

トリガーの再コンパイル時にエラーが発生した場合、エラーが戻り、そのトリガーは使用禁止のままです。**SQL*Plus** コマンド **SHOW ERRORS** を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

DEBUG **DEBUG** を指定すると、**PL/SQL** コンパイラに対して、**PL/SQL** デバッガ用のコードを生成および格納するように指示できます。

参照：

- これらのプロシージャについては、『**Oracle9i** アプリケーション開発者ガイド - 基礎編』を参照してください。
- リモート・オブジェクトを含むスキーマ・オブジェクト間の依存性を **Oracle** が管理する方法については、『**Oracle9i** データベース概要』を参照してください。

REUSE SETTINGS **REUSE SETTINGS** を指定すると、**Oracle** によるコンパイラ・スイッチ設定の削除および再取得を回避できます。この句では既存の設定が持続され、その設定で再コンパイルされます。

DEBUG と **REUSE SETTINGS** の両方を指定する場合は、**PLSQL_COMPILER_FLAGS** パラメータの永続格納値が **INTERPRETED**, **DEBUG** に設定されます。その他のコンパイラ・スイッチ値は変更されません。

参照： **PLSQL_COMPILER_FLAGS** パラメータと **COMPILE** 句の相互作用の詳細は、『**PL/SQL** ユーザーズ・ガイドおよびリファレンス』および『**Oracle9i** アプリケーション開発者ガイド - 基礎編』を参照してください。

例

トリガーを使用禁止にする例： サンプル・スキーマ hr には、employees 表で作成された update_job_history という名前のトリガーがあります。トリガーは、UPDATE 文によって従業員の job_id が変更されるたびに起動されます。トリガーは、job_history 表に従業員 ID、直前の職種の開始日と終了日、職種 ID および部門を含む行を挿入します。

このトリガーは、作成時に自動的に使用可能になります。その後、次の文を指定して使用禁止にできます。

```
ALTER TRIGGER update_job_history DISABLE;
```

使用禁止になると、UPDATE 文によって従業員の job_id が変更されてもトリガーは起動されません。

トリガーを使用可能にする例： トリガーを使用禁止にした後、次の文を使用してそのトリガーを再び使用可能にできます。

```
ALTER TRIGGER update_job_history ENABLE;
```

再びトリガーを使用可能にした場合、UPDATE 文によって従業員の job_id が変更されるたびにトリガーが起動されます。トリガーが使用禁止の場合、従業員の job_id が変更されても、他のトランザクションが再び job_id を変更するまで、トリガーは自動的に起動されません。

ALTER TYPE

用途

ALTER TYPE 文を使用すると、メンバー属性またはメソッドを追加または削除できます。オブジェクト型の既存のプロパティ (FINAL または INSTANTIABLE)、または型のスカラー属性も変更できます。

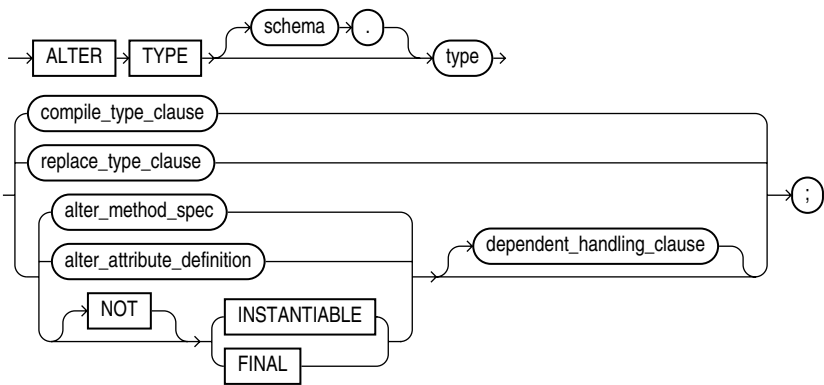
この文を使用すると、新しいオブジェクト・メンバーのサブプログラム仕様を追加することによって、型の仕様部または本体を再コンパイルしたり、オブジェクト型の仕様を変更できます。

前提条件

オブジェクト型が自分のスキーマ内にあり、CREATE TYPE または CREATE ANY TYPE システム権限が必要です。または、ALTER ANY TYPE システム権限が必要です。

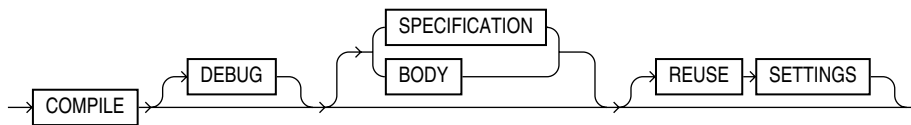
構文

alter_type::=

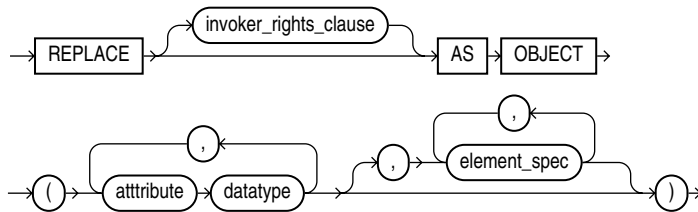


(12-7 ページの「[compile_type_clause::=](#)」、12-7 ページの「[replace_type_clause::=](#)」、12-9 ページの「[alter_method_spec::=](#)」、12-9 ページの「[alter_attribute_definition::=](#)」、12-10 ページの「[dependent_handling_clause::=](#)」を参照)

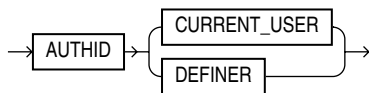
compile_type_clause::=



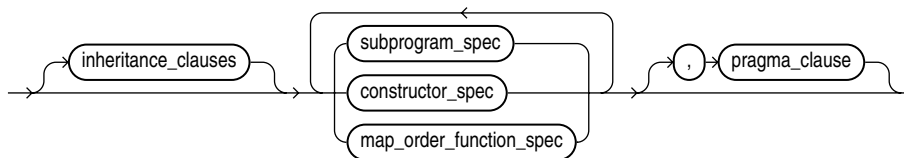
replace_type_clause::=



invoker_rights_clause::=

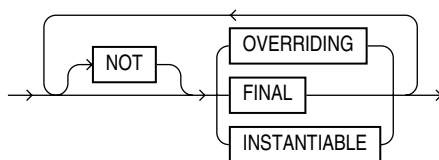


element_spec::=

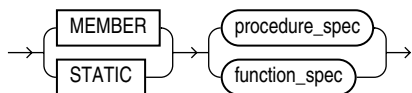


(12-7 ページの「[inheritance_clauses::=](#)」、12-8 ページの「[subprogram_spec::=](#)」、12-8 ページの「[constructor_spec::=](#)」、12-8 ページの「[map_order_function_spec::=](#)」、12-9 ページの「[pragma_clause::=](#)」を参照)

inheritance_clauses::=

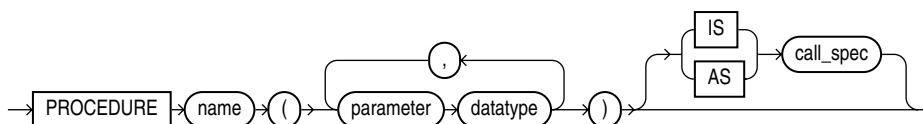


subprogram_spec::=

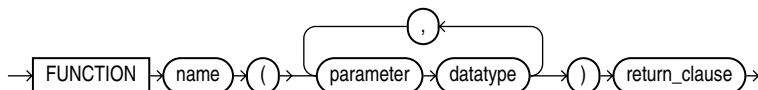


(12-8 ページの「[procedure_spec::=](#)」、12-8 ページの「[function_spec::=](#)」を参照)

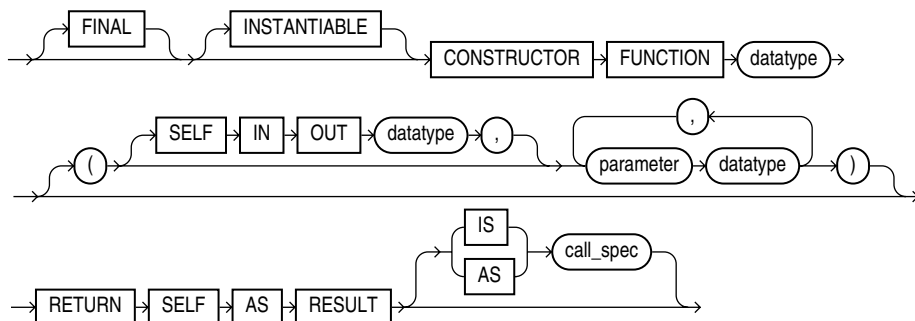
procedure_spec::=



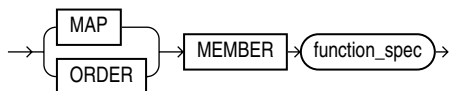
function_spec::=



constructor_spec::=

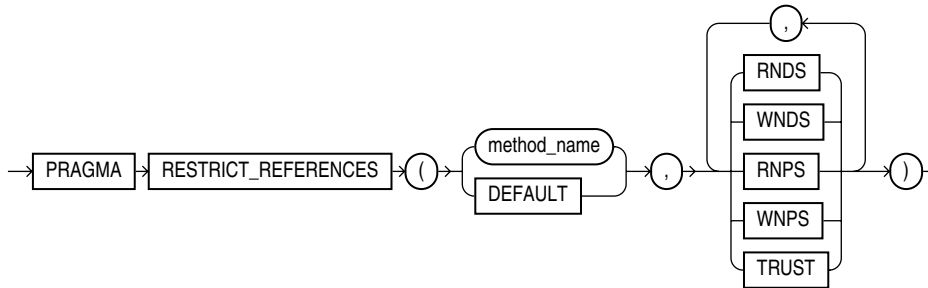


map_order_function_spec::=

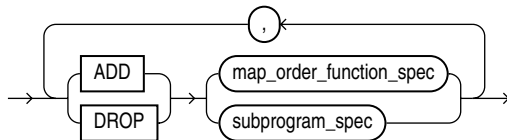


(12-8 ページの「[function_spec::=](#)」を参照)

pragma_clause::=

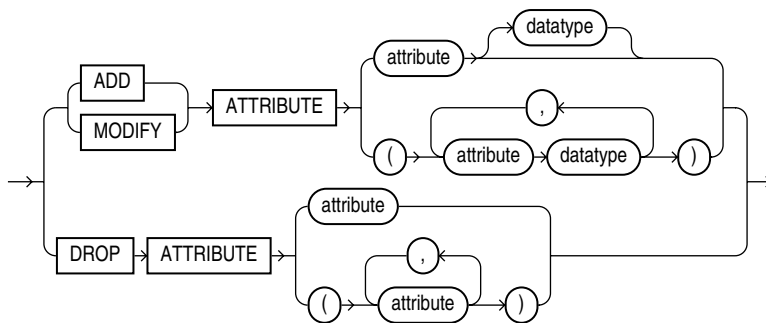


alter_method_spec::=

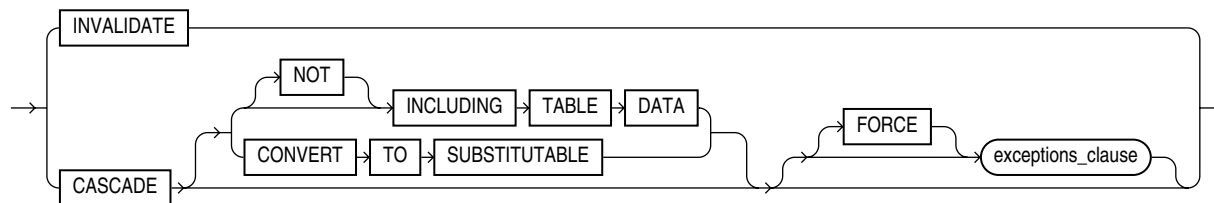


(12-8 ページの「`map_order_function_spec::=`」、12-8 ページの「`subprogram_spec::=`」を参照)

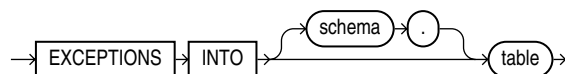
alter_attribute_definition::=



dependent_handling_clause::=



exceptions_clause::=



セマンティクス

schema

型を定義するスキーマを指定します。*schema* を指定しない場合、この型が現行のスキーマに存在するものとみなされます。

type

オブジェクト型、ネストした表型または ROWID 型の名前を指定します。

compile_type_clause

COMPILE を指定して、オブジェクト型の仕様部および本体をコンパイルします。SPECIFICATION または BODY のいずれも指定していない場合、これはデフォルトです。

再コンパイル中、Oracle はすべての永続コンパイラのスイッチ設定を削除し、セッションからそれらを再び取得してコンパイルの終了時に格納します。この手順を回避するには、REUSE SETTINGS 句を指定します。

型の再コンパイル時にエラーが発生した場合、エラーが戻り、その型は無効のままです。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

参照： 12-20 ページの「[型の再コンパイル例：](#)」および 12-20 ページの「[型仕様部の再コンパイル例：](#)」を参照してください。

DEBUG DEBUG を指定すると、PL/SQL コンパイラに対して、PL/SQL デバッガ用のコードを生成および格納するように指示できます。

SPECIFICATION SPECIFICATION を指定して、オブジェクト型の仕様部のみをコンパイルします。

BODY BODY を指定して、オブジェクト型の本体のみをコンパイルします。

REUSE SETTINGS REUSE SETTINGS を指定すると、Oracle によるコンパイラ・スイッチ設定の削除および再取得を回避できます。この句では既存の設定が持続され、その設定で再コンパイルされます。

DEBUG と REUSE SETTINGS の両方を指定する場合は、PLSQL_COMPILER_FLAGS パラメータの永続格納値が INTERPRETED, DEBUG に設定されます。その他のコンパイラ・スイッチ値は変更されません。

参照： PLSQL_COMPILER_FLAGS パラメータと COMPILE 句の相互作用の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

replace_type_clause

REPLACE 句によって、新しいメンバー・サブプログラム仕様を追加します。この句はオブジェクト型のみに有効で、ネストした表型または VARRAY 型には無効です。

attribute

オブジェクトの属性名を指定します。属性とは、名前と型指定子を持つオブジェクト構造を形成するデータ項目です。

element_spec

再定義するオブジェクトの要素を指定します。

inheritance_clauses *element_spec* の一部として *inheritance_clauses* を使用すると、スーパータイプとサブタイプの間の関係を指定できます。

OVERRIDING この句は、メンバー・メソッドでのみ使用可能です。OVERRIDING を指定すると、スーパータイプで定義されているメンバー・メソッドをこのメソッドで上書きします。このキーワードは、スーパータイプのメソッドを再定義する場合に必須です。NOT OVERRIDING がデフォルトです。

OVERRIDING の制限事項： OVERRIDING 句は、スタティック・メソッドまたは SQLJ オブジェクト型では無効です。

FINAL FINAL を指定すると、メソッドがこの型のサブタイプによって上書きされないようにできます。デフォルトは、NOT FINAL です。

NOT INSTANTIABLE 型がこのメソッドの実装を提供しない場合は、NOT INSTANTIABLE を指定します。デフォルトでは、すべてのメソッドは INSTANTIABLE です。

NOT INSTANTIABLE の制限事項： NOT INSTANTIABLE を指定する場合は、FINAL または STATIC を指定できません。

subprogram_spec MEMBER および STATIC を指定すると、属性として参照されるオブジェクト型に対応付けられたファンクション・サブプログラムまたはプロシージャ・サブプログラムを指定できます。

それぞれのプロシージャまたはファンクションの仕様部について、オブジェクト型本体に対応するメソッド本体を指定する必要があります。

参照：

- メンバー・メソッドとスタティック・メソッドの違い、およびこれらの例については、16-3 ページの「[CREATE TYPE](#)」を参照してください。
- パッケージ内のサブプログラム名のオーバーロードについては、『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。
- 16-25 ページの「[CREATE TYPE BODY](#)」を参照してください。

procedure_spec プロシージャ・サブプログラムの仕様部を指定します。

function_spec ファンクション・サブプログラムの仕様部を指定します。

pragma_clause pragma_clause は、データベースの表またはパッケージ変数（あるいはその両方）に対するメンバー・ファンクションの読み書きアクセスを拒否し、副作用の発生を防止するコンパイラ・ディレクティブです。

アプリケーションの下位互換を保つ必要がない場合は、この句を使用しないことをお勧めします。Oracle9i からは、純粋度チェックが実行時に行われるため、この句は使用されなくなります。ご使用のアプリケーションの下位互換用にこの句を使用する必要がある場合は、16-15 ページの「[pragma_clause](#)」（「CREATE TYPE」の下）を参照してください。

プラグマの制限事項： pragma_clause は、メソッドを削除すると無効になります。

参照：『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

map_order_function_spec 宣言する MEMBER または STATIC メソッドの数にかかわらず、MAP メソッドまたは ORDER メソッドのいずれかを宣言できます。ただし、スーパータイプが NOT FINAL MAP メソッドを定義する場合、サブタイプは MAP メソッドを上書きできません。いずれかのメソッドを宣言すると、SQL 内でオブジェクト・インスタンスを比較できます。

どちらのメソッドも宣言しない場合、比較できるのはオブジェクト・インスタンスの等価性と非等価性のみです。同じ型定義のインスタンスは、それぞれの対応する属性の各組が等しい場合にのみ等しくなります。2つのオブジェクト型の等価性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。

参照： オブジェクト値の比較の詳細は、2-46 ページの「[オブジェクト値](#)」を参照してください。

- MAP の場合、オブジェクトの順序付けられたすべてのインスタンスの中から、指定したインスタンスの相対的な位置を戻すメンバー・ファンクション (MAP メソッド) を指定します。MAP メソッドは暗黙的にコールされ、オブジェクト・インスタンスを事前定義済のスカラー型の値にマップすることによって、それらのオブジェクト・インスタンスに順序を設定します。比較条件および ORDER BY 句の順序が使用されます。

MAP メソッドの引数が NULL の場合、MAP メソッドは NULL を返し、メソッドは起動されません。

オブジェクトの仕様部には、1つの MAP メソッドのみを指定できます。この MAP メソッドは、ファンクションである必要があります。結果として生成される型は、事前定義済の SQL スカラー型である必要があります。また、MAP ファンクションに指定できる引数は、暗黙的な SELF 引数のみです。

注意： ソート (ORDER BY 句、GROUP BY 句、DISTINCT 句または UNION 句を使用) または結合を指定した問合せが *type* を参照し、これらの問合せをパラレル化する場合は、MAP メンバー・ファンクションを指定する必要があります。

サブタイプには、新しい MAP メソッドを定義できません。ただし、継承された MAP メソッドは上書きできます。

- ORDER の場合、オブジェクトのインスタンスを明示的な引数および暗黙的な SELF 引数とするメンバー・ファンクション (ORDER メソッド) を指定し、負の整数、0 (ゼロ) または正の整数のいずれかを返します。負の整数は、暗黙的な SELF 引数が明示的な引数より小さいことを示し、0 (ゼロ) は、両方が等しいことを示します。また、正の整数は、暗黙的な SELF 引数が明示的な引数より大きいことを示します。

ORDER メソッドのいずれかの引数が NULL の場合は、ORDER メソッドは NULL を返し、メソッドは起動されません。

同じオブジェクト型定義の各インスタンスを ORDER BY 句の中で比較した場合、ORDER メソッド・ファンクションがコールされます。

オブジェクトの仕様部には、1 つの ORDER メソッドのみ指定でき、その ORDER メソッドは戻り型が NUMBER のファンクションである必要があります。

サブタイプは、ORDER メソッドを定義できません。また、継承された ORDER メソッドを上書きすることもできません。

invoker_rights_clause

invoker_rights_clause を使用すると、オブジェクト型のメンバー・ファンクションおよびプロシージャが、そのオブジェクト型を所有するユーザーのスキーマで権限付きで実行されるか、または CURRENT_USER のスキーマで権限付きで実行されるかを指定できます。この仕様は、対応する型本体にも適用されます。

また、この句は、問合せ、DML 操作、およびその型のメンバー・ファンクションおよびプロシージャ内の動的 SQL 文の外部名の変換方法も定義します。

実行者権限の制限事項： この句はオブジェクト型のみに指定でき、ネストした表および VARRAY 型には指定できません。

AUTHID CURRENT_USER 句 CURRENT_USER を指定して、オブジェクト型のメンバー・ファンクションおよびプロシージャを CURRENT_USER の権限で実行します。この句によって **実行者権限型** が作成されます。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を CURRENT_USER のスキーマで変換することも指定します。その他すべての文の外部名は、型が存在するスキーマ内で変換します。

注意： 実行者権限を実行者権限状態で作成した場合、この句を指定し、この型への実行者権限を維持する必要があります。指定しないと、この状態は定義者権限に戻ります。

AUTHID DEFINER 句 DEFINER によって、ファンクションおよびプロシージャの存在するスキーマの所有者権限で、オブジェクト型のメンバー・ファンクションおよびプロシージャを実行し、メンバー・ファンクションおよびプロシージャが存在するスキーマで外部名を変換することを指定します。これはデフォルトです。

参照：

- `CURRENT_USER` の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 『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。

alter_method_spec

alter_method_spec を使用すると、*type* に対してメソッドを追加および削除できます。型に依存するすべてのファンクション索引が使用禁止になります。

1 つの ALTER TYPE 文で複数のメソッドを追加または削除できますが、各メソッドを参照できるのは 1 回のみです。

ADD メソッドを追加する場合は、型の階層内の既存の属性と競合しない名前にする必要があります。

参照： 12-19 ページの「[メンバー・ファンクションの追加例 :](#)」を参照してください。

DROP メソッドを削除すると、対象の型からメソッドが削除されます。

メソッドの削除の制限事項： スーパータイプから継承されたメソッドを、サブタイプで削除できません。かわりに、スーパータイプでメソッドを削除します。

subprogram_spec MEMBER および STATIC 句を使用すると、オブジェクト型に対してプロシージャ・サブプログラムを追加または削除できます。

サブプログラムの制限事項： スーパータイプのメンバー・メソッドを再定義するサブタイプのスタティック・メソッドは定義できません。またその逆も同様です。

参照： 16-12 ページの CREATE TYPE の「[subprogram_spec](#)」を参照してください。

map_order_function_spec MAP と ORDER のいずれかのメソッドを宣言すると、SQL 内でオブジェクト・インスタンスを比較できます。

MAP メソッドおよび ORDER メソッドの制限事項： サブタイプに ORDER メソッドを追加できません。

参照： 16-16 ページの CREATE TYPE の「[constructor_spec](#)」を参照してください。

alter_attribute_definition

`alter_attribute_definition` 句を使用すると、オブジェクト型の属性を追加、削除または変更できます。1つの ALTER TYPE 文で複数のメンバー属性またはメソッドを追加、削除または変更できますが、各属性または各メソッドを参照できるのは1回のみです。

ADD ATTRIBUTE 新しい属性は、型の階層内にすでに存在する属性またはメソッドと競合しない名前にする必要があります。新しい属性は、ローカル定義の属性リストの終わりに追加されます。

注意： スーパータイプに属性を追加すると、すべてのサブタイプに継承されます。サブタイプでは、継承された属性は、宣言した属性より常に優先されます。そのため、スーパータイプに属性を追加した後、暗黙的に変更されたサブタイプのマッピングの更新が必要な場合があります。

参照： 12-20 ページの「[コレクション属性の追加例:](#)」を参照してください。

DROP ATTRIBUTE 型から属性を削除すると、削除された属性に対応する列、索引、統計情報および削除する属性を参照する制約も削除されます。

削除する属性のデータ型を指定する必要はありません。

属性の削除の制限事項：

- スーパータイプから継承された属性を削除できません。かわりに、スーパータイプから属性を削除します。
- パーティション、サブパーティションまたはクラスタ・キーの一部になっている属性は、削除できません。
- オブジェクト表、主キーまたは索引構成表の主キー・ベースのオブジェクト識別子の属性は、削除できません。
- ルート型のすべての属性は削除できません。かわりに、型を削除します。ただし、サブタイプのローカルで宣言されたすべての属性は、削除できます。

MODIFY ATTRIBUTE 既存のスカラー属性のデータ型を変更できます。たとえば、VARCHAR2 または RAW の長さ、または数値属性の精度または位取りを増やすことができます。

属性の変更の制限事項： ファンクション索引、ドメイン索引またはクラスタ・キーで参照される属性のサイズを拡張することはできません。

[NOT] FINAL

[NOT] FINAL 句を使用すると、この型のサブタイプを以降で作成させるかどうかを指定できます。

- この型のサブタイプを以降で作成させない場合は、FINAL を指定します。
- この型のサブタイプを以降で作成させる場合は、NOT FINAL を指定します。

FINAL と NOT FINAL との間でプロパティを変更する場合は、依存列および表のデータを変換するために dependent_handling_clause の CASCADE 句を指定する必要があります。CASCADE NOT INCLUDING TABLE DATA が指定されたデータの変換は遅延できません。

- 型を NOT FINAL から FINAL へ変更した場合、CASCADE [INCLUDING TABLE DATA] を指定する必要があります。
- 型を FINAL から NOT FINAL へ変更した場合は、次のとおりです。
 - その型の代替可能な表および列を新しく作成する場合で、既存の依存表および列の代替性については考慮していない場合、CASCADE INCLUDING TABLE DATA を指定します。既存のすべての依存列および表は、NOT SUBSTITUTABLE AT ALL LEVELS と、マークされるため、既存の列および表に変更された型の新規のサブタイプ・インスタンスは挿入できません。
 - その型の代替可能な表および列を新しく作成し、既存の依存表および列に変更された型の新規のサブタイプ・インスタンスを格納する場合、CASCADE CONVERT TO SUBSTITUTABLE を指定します。明示的に NOT SUBSTITUTABLE AT ALL LEVELS とマークされたもの以外のすべての既存の依存列および表は、SUBSTITUTABLE AT ALL LEVELS とマークされます。

参照： オブジェクト型の発展の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

FINAL の制限事項： 型にサブタイプがある場合、ユーザー定義型を NOT FINAL から FINAL に変更できません。

[NOT] INSTANTIABLE

[NOT] INSTANTIABLE 句を使用すると、この型のオブジェクト・インスタンスを構成させるかどうかを指定します。

- この型のオブジェクト・インスタンスを構成させる場合は、INSTANTIABLE を指定します。
- このオブジェクト型のコンストラクタ（デフォルトまたはユーザー定義の）が存在しない場合は、NOT INSTANTIABLE を指定します。インスタンス化ができないメソッドを持つ型、および属性を持たない型には、これらのキーワードを指定する必要があります（継承、またはこの句で指定）。

NOT INSTANTIABLE の制限事項： 型に依存表がある場合、ユーザー定義型を INSTANTIABLE から NOT INSTANTIABLE に変更できません。

dependent_handling_clause

dependent_handling_clause を使用すると、変更された型に依存するオブジェクトの処理方法を Oracle に指示できます。この句を指定しないと、対象の型に依存する型または表がある場合に ALTER TYPE 文が異常終了します。

INVALIDATE 句

INVALIDATE を指定すると、メカニズムの検証なしで、依存するすべてのオブジェクトが無効になります。

注意： 型の変更が検証されないため、この句の使用には注意が必要です。たとえば、パーティション化キーまたはクラスタ・キーの属性を削除すると、表に書き込みができなくなります。

CASCADE 句

型の変更を依存する型および表まで広める場合は、CASCADE 句を指定します。依存する型または表でエラーがあったときに、FORCE を指定していない場合は、文は異常終了します。

INCLUDING TABLE DATA INCLUDING TABLE DATA を指定すると、すべてのユーザー定義列に格納されているデータを列の型の最新バージョンに変換します。これはデフォルト値です。

注意： 列データが Oracle8 リリース 8.0 のイメージ・フォーマットの場合、この句を指定する必要があります。この句は、FINAL と NOT FINAL の間で型のプロパティを変更する場合にも必須です。

- 列の型に各属性を追加すると、新しい属性はデータに追加され、NULL に初期化されます。
- 参照される型から各属性を削除すると、表の各行から対応する属性データが削除されます。

INCLUDING TABLE DATA を指定する場合、表のデータを含む表領域のすべてを読み込み / 書き込みモードにする必要があります。

NOT INCLUDING TABLE DATA を指定すると、列のメタデータはアップグレードされ、変更は型に反映されますが、依存する列はスキャンされず、この ALTER TYPE 文の一部としてはデータが更新されません。ただし、依存する列データはアクセスすることができ、後続のデータの間合せの結果では、型の変更が反映されます。

参照： 型属性の変更時に表データを含まない実装の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

FORCE 依存表および索引のエラーを無視し、すべてのエラーを指定した例外表のログに記録する場合は、FORCE を指定します。例外表は、DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE プロシージャの実行によって作成しておく必要があります。

例

メンバー・ファンクションの追加例： 次の例は、data_typ オブジェクト型 (16-19 ページの「オブジェクト型の例:」で作成) を使用します。メソッドは、data_typ に追加され、型の本体はそれに対応して変更されます。日付書式は、oe.orders サンプル表の order_date 列と一致しています。

```
ALTER TYPE data_typ
  ADD MEMBER FUNCTION qtr(der_qtr DATE)
  RETURN CHAR CASCADE;

CREATE OR REPLACE TYPE BODY data_typ IS
  MEMBER FUNCTION prod (invent NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN (year + invent);
  END;

  MEMBER FUNCTION qtr(der_qtr DATE) RETURN CHAR IS
  BEGIN
    IF (der_qtr < TO_DATE('01-APR', 'DD-MON')) THEN
      RETURN 'FIRST';
    ELSIF (der_qtr < TO_DATE('01-JUL', 'DD-MON')) THEN
      RETURN 'SECOND';
    ELSIF (der_qtr < TO_DATE('01-OCT', 'DD-MON')) THEN
      RETURN 'THIRD';
    ELSE
      RETURN 'FOURTH';
    END IF;
  END;
END;
```

コレクション属性の追加例： 次の例は、サンプル・スキーマ `oe` の `VARRAY` 型の `phone_list_typ` を、`customers` 表の `cust_address_typ` オブジェクト列に追加します。

```
ALTER TYPE cust_address_typ
  ADD ATTRIBUTE (phone phone_list_typ) CASCADE;
```

型の再コンパイル例： 次の例では、`customer_address_typ` 型を再コンパイルします。

```
ALTER TYPE customer_address_typ COMPILE;
```

型仕様部の再コンパイル例： 次の文は、`link2` の型本体をコンパイルします。

```
CREATE TYPE link1 AS OBJECT
  (a NUMBER);
/
CREATE TYPE link2 AS OBJECT
  (a NUMBER,
   b link1,
   MEMBER FUNCTION p(c1 NUMBER) RETURN NUMBER);
/

CREATE TYPE BODY link2 AS
  MEMBER FUNCTION p(c1 NUMBER) RETURN NUMBER IS
  BEGIN
    dbms_output.put_line(c1);
    RETURN c1;
  END;
END;
```

次の例は、`link2` の仕様部と本体の両方を無効にします。

```
ALTER TYPE link1 ADD ATTRIBUTE (b NUMBER) INVALIDATE;
```

別の文で仕様部と本体を再コンパイルすることによって、型を再コンパイルする必要があります。

```
ALTER TYPE link2 COMPILE SPECIFICATION;
```

```
ALTER TYPE link2 COMPILE BODY;
```

仕様部と本体の両方は同時にコンパイルできます。

```
ALTER TYPE link2 COMPILE;
```

ALTER USER

用途

ALTER USER を使用すると、次の処理ができます。

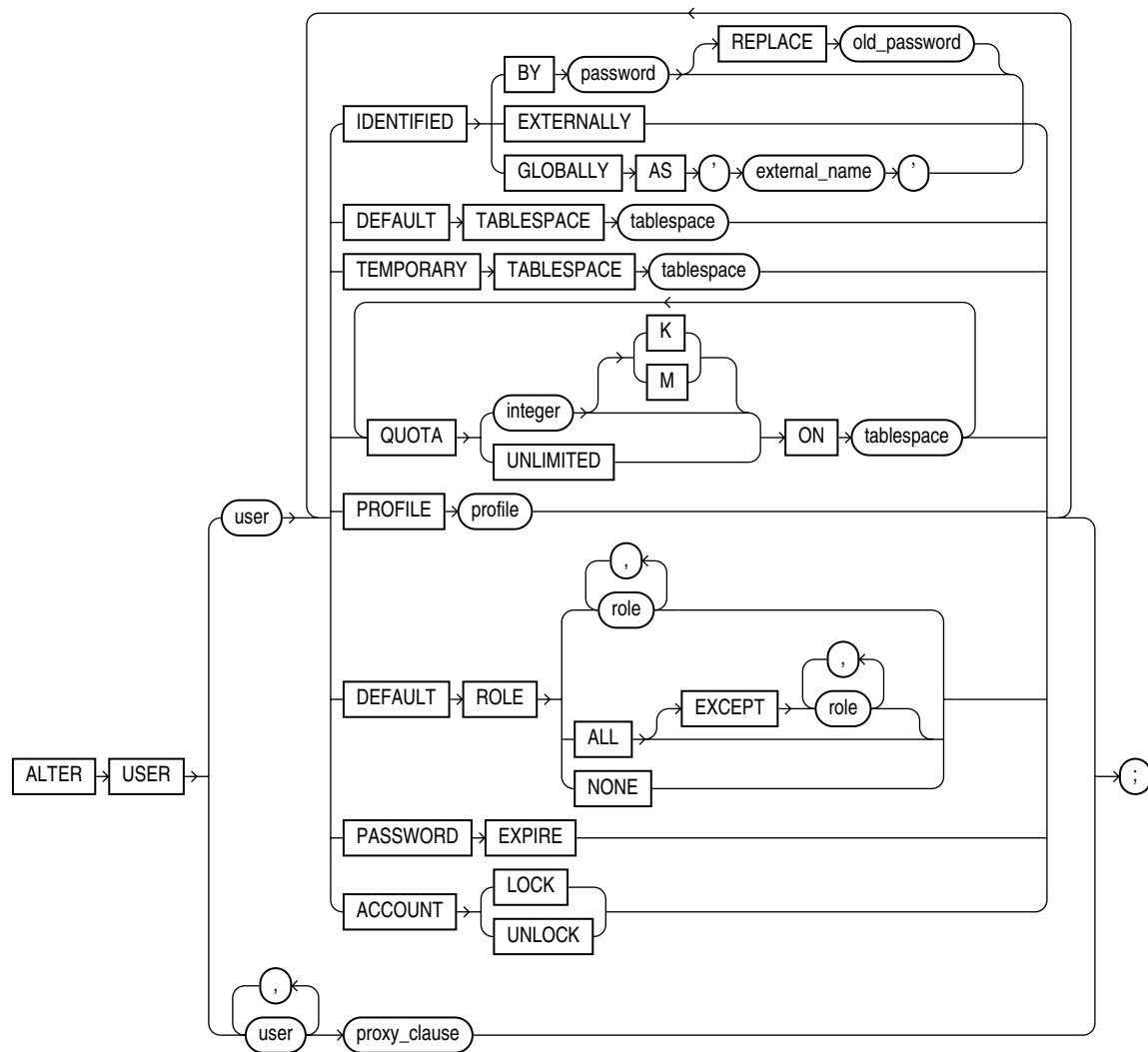
- データベース・ユーザーの認証またはデータベース・リソースの特性を変更します。
- プロキシ・サーバーが認証なしでクライアントとして接続することを許可します。

前提条件

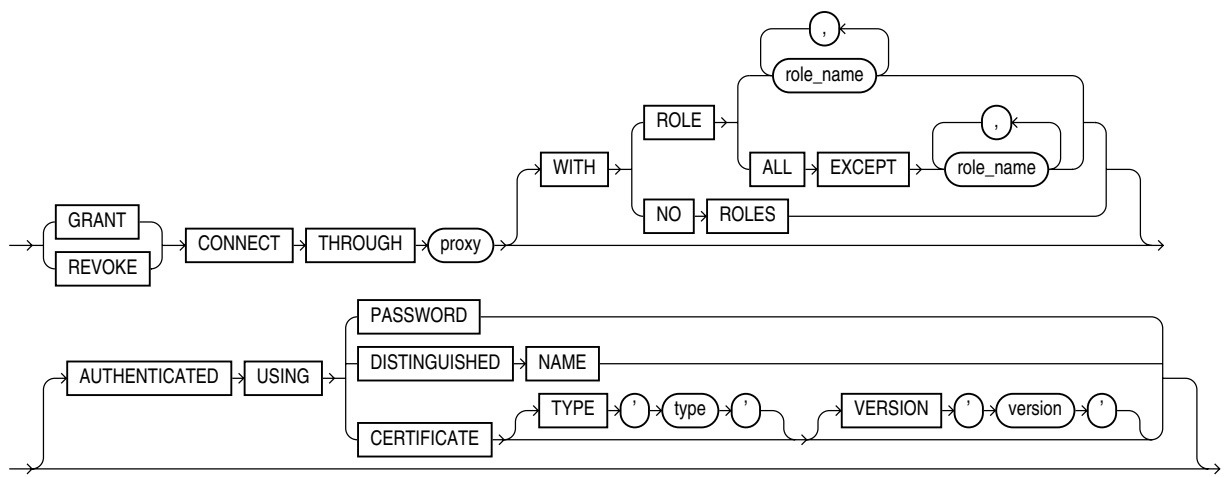
ALTER USER システム権限が必要です。ただし、ユーザー自身のパスワードはこの権限がない場合でも変更できます。

構文

alter_user::=



proxy_clause::=



セマンティクス

この項で説明するキーワード、パラメータおよび句は、ALTER USER 独自のものと、CREATE USER にもあるが、セマンティクスが異なるものがあります。その他のキーワード、パラメータおよび句には、CREATE USER 文と同じ意味があります。

注意： ユーザー名およびパスワードは、ご使用のプラットフォームに応じて、ASCII または EBCDIC 文字のみでエンコードすることをお薦めします。詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

参照：

- キーワードおよびパラメータについては、16-31 ページの「[CREATE USER](#)」を参照してください。
- ユーザーにデータベース・リソースへの制限を割り当てる方法については、14-66 ページの「[CREATE PROFILE](#)」を参照してください。

IDENTIFIED 句

- `BY password` を指定し、ユーザーの新しいパスワードを指定します。

注意： 異なるタイムスタンプで特定のパスワードを再設定する必要があります。1 秒以内に 1 つのパスワードを複数回再設定した場合（たとえば、スクリプトを使用して一連のパスワードの設定を繰り返した場合）、Oracle はパスワードが再利用できないというエラー・メッセージを返すことがあります。このため、パスワードの再設定には、スクリプトを使用しないことをお勧めします。

初めてパスワードを設定する場合、または ALTER USER システム権限があり、他のユーザーのパスワードを変更する場合は、REPLACE 句を省略できます。ただし、ALTER USER システム権限がないかぎり、複雑なパスワードの検証機能が使用可能な場合は、UTLPWDMG.SQL スクリプトを実行するか、ユーザーに割り当てられたプロファイルの PASSWORD_VERIFY_FUNCTION パラメータに検証機能を指定して、常に、REPLACE 句を指定する必要があります。

注意： 以前のパスワードを REPLACE 句に指定した場合でも、ユーザー自身が既存のパスワードを変更しないかぎり、以前のパスワードはチェックされません。そのようなチェックが必要な場合（たとえば、権限を持つユーザーが他のユーザーのパスワードを変更する場合）、複雑なパスワードの検証機能で、以前のパスワードが NULL のパスワードの変更を禁止するか、または ALTER USER のかわりに OCIPasswordChange () コールを使用します。詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

参照： 複雑なパスワードの検証機能については、『Oracle9i データベース管理者ガイド』を参照してください。

- GLOBALLY AS '*external_name*' を指定すると、LDAP V3 に準拠するディレクトリ・サービス（Oracle Internet Directory など）を使用して、ユーザーを認証する必要があることを示します。

ユーザーに直接付与されたすべての外部ロールが取り消された場合のみ、ユーザー・アクセス検証方法を IDENTIFIED GLOBALLY AS '*external_name*' に変更できます。

IDENTIFIED GLOBALLY AS '*external_name*' として作成されたユーザーを、IDENTIFIED BY *password* または IDENTIFIED EXTERNALLY に変更できます。

参照： 16-31 ページの [CREATE USER](#)、12-27 ページの「[ユーザー識別の変更例:](#)」および 12-27 ページの「[ユーザー認証の変更例:](#)」を参照してください。

TEMPORARY TABLESPACE 句

ユーザーの一時表領域として割り当てる表領域、または再度割り当てる表領域は、一時表領域なので、標準以外のブロック・サイズを使用した表領域を指定できません。指定した場合、Oracle からエラーは戻されません。ただし、ユーザーが行う以降の操作で一時セグメントが必要な場合は、エラーが戻されます。

DEFAULT ROLE 句

ログイン時にデフォルトによってユーザーに付与されるロールを指定します。この句では、GRANT 文を使用してユーザーに直接付与されているロールのみ指定できます。DEFAULT ROLE 句を使用して次のロールを使用可能にすることはできません。

- ユーザーに付与されていないロール
- 他のロールを介して付与されているロール
- 外部サービス（オペレーティング・システムなど）または Oracle Internet Directory によって管理されるロール

Oracle では、ユーザーがパスワードを指定しなかった場合でも、ログイン時にデフォルトのロールは使用可能になります。

参照： 14-73 ページの「[CREATE ROLE](#)」を参照してください。

proxy_clause

proxy_clause を使用すると、プロキシ（アプリケーションまたはアプリケーション・サーバー）の機能を制御でき、指定したデータベース・ユーザーまたはエンタープライズ・ユーザーとして接続し、すべてまたはいくつかのユーザー・ロールをアクティブにする、または非アクティブにできます。

注意： *proxy_clause* は、データベース・ユーザーおよびエンタープライズ・ユーザーに対していくつかのプロキシ認証を提供します。アプリケーション・ユーザーのプロキシ認証の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

参照： プロキシおよびそれらのデータベースについては、『Oracle9i データベース概要』および 12-28 ページの「[プロキシ・ユーザー例 :](#)」を参照してください。

GRANT | REVOKE

接続を許可するには、GRANT を指定します。接続を禁止するには、REVOKE を指定します。

CONNECT THROUGH 句

Oracle に接続するプロキシを識別します。AUTHENTICATED USING 句を指定しない場合は、Oracle は、ユーザーの認証にプロキシを想定します。

WITH ROLE WITH ROLE *role_name* を使用すると、プロキシは指定したユーザーとして接続でき、*role_name* で指定されたロールのみをアクティブにできます。

WITH ROLE ALL EXCEPT WITH ROLE ALL EXCEPT *role_name* を使用すると、プロキシは指定したユーザーとして接続でき、*role_name* で指定されたロール以外の、このユーザーに対応付けられたすべてのロールをアクティブにできます。

WITH NO ROLES WITH NO ROLES を使用すると、プロキシは指定したユーザーとして接続できますが、接続後にそのユーザーのロールを 1 つでもアクティブにすることは禁止されます。

WITH 句を指定しない場合、指定したユーザーに付与されているすべてのロールが自動的にアクティブになります。

AUTHENTICATED USING

プロキシ認証をプロキシ以外のソースで処理する場合は、AUTHENTICATED USING を指定します。この句は、GRANT CONNECT THROUGH *proxy* 句の一部のみに関連します。

PASSWORD プロキシに認証用のユーザーのデータベース・パスワードを設定する場合は、PASSWORD を指定します。プロキシはデータベースに依存し、パスワードに基づいてユーザーを認証します。

DISTINGUISHED NAME DISTINGUISHED NAME を指定すると、プロキシが識別名で識別され、グローバルに識別されたユーザーとして動作することが許可されます。

CERTIFICATE CERTIFICATE を指定すると、プロキシが証明書に含まれる識別名を持つ、グローバルに識別されたユーザーとして動作が許可されます。

DISTINGUISHED NAME と CERTIFICATE の両方の場合において、プロキシはすでに認証され、グローバル・データベース・ユーザーとして動作しています。

- *type* には、使用する証明書のタイプを指定します。*type* を指定しない場合、デフォルトは 'X.509' です。
- *version* には、使用する証明書のバージョンを指定します。*version* を指定しない場合、デフォルトは '3' です。

CERTIFICATE の制限事項： この句は、REVOKE CONNECT THROUGH *proxy* 句の一部として指定できません。

参照：

- データベース・セキュリティの概要は、『Oracle9i セキュリティ概要』を参照してください。
- 中間層システムおよびプロキシ認証の詳細は、『Oracle9i データベース管理者ガイド』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

例

ユーザー識別の変更例： 次の文は、ユーザー sidney (16-36 ページの「データベース・ユーザーの作成例：」で作成) のパスワードを second_2nd_pwd に、デフォルト表領域を表領域 example に変更します。

```
ALTER USER sidney
  IDENTIFIED BY second_2nd_pwd
  DEFAULT TABLESPACE example;
```

次の文は、new_profile プロファイル (14-71 ページの「プロファイルの作成例：」で作成) をサンプル・ユーザー sh: に割り当てます。

```
ALTER USER sh
  PROFILE new_profile;
```

後続のセッションでは、sh は new_profile プロファイルの制限に従います。

次の文は、sh に直接付与されているすべてのロール (dw_manager ロールを除く) をデフォルト・ロールに設定します。

```
ALTER USER sh
  DEFAULT ROLE ALL EXCEPT dw_manager;
```

sh の次のセッションの開始時には、dw_manager 以外で sh に直接付与されているすべてのロールが使用可能になります。

ユーザー認証の変更例： 次の文はユーザー app_user1 (16-36 ページの「データベース・ユーザーの作成例：」で作成) の検証メカニズムを変更します。

```
ALTER USER app_user1 IDENTIFIED GLOBALLY AS 'CN=tom,O=oracle,C=US';
```

次の文は、ユーザー sidney のパスワードを期限切れにします。

```
ALTER USER sidney PASSWORD EXPIRE;
```

PASSWORD EXPIRE を使用してデータベース・ユーザーのパスワードを期限切れにした場合、そのユーザー（または DBA）は、期限切れの後でデータベースにログインする際、パスワードを変更する必要があります。ただし、SQL*Plus などのツール製品を使用した場合、期限切れの後の最初のログイン時に、パスワードを変更できます。

プロキシ・ユーザー例： 次の文は、ユーザー `app_user` を変更します。例では、`app_user` がプロキシ・ユーザー `sh` を使用して接続できます。また、プロキシ `sh` を使用して接続したときに、`app_user` が `warehouse_user` ロール（14-75 ページの「[ロールの作成例 :](#)」で作成）を使用可能にできます。

```
ALTER USER app_user
GRANT CONNECT THROUGH sh
WITH ROLE warehouse_user;
```

注意： 基本的な構文を示すため、サンプル・データベース Sales History のユーザー（`sh`）をプロキシとして使用します。通常、プロキシ・ユーザーは、アプリケーション・サーバーまたは中間層のエンティティに存在します。アプリケーション・サーバーを経由して、アプリケーション・ユーザーとデータベースの間のインタフェースを作成する場合の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

参照：

- `app_user` ユーザーの作成方法については、16-36 ページの「[外部データベース・ユーザーの作成例 :](#)」を参照してください。
- `dw_user` ロールの作成方法については、14-75 ページの「[ロールの作成例 :](#)」を参照してください。

次の文は、ユーザー `app_user` がプロキシ・ユーザー `sh` を使用して接続する権限を取り消します。

```
ALTER USER app_user REVOKE CONNECT THROUGH sh;
```

次の仮想例は、プロキシ認証の他の方法を示します。

```
ALTER USER sully GRANT CONNECT THROUGH OAS1
AUTHENTICATED USING PASSWORD;
```

```
ALTER USER green GRANT CONNECT THROUGH WebDB
AUTHENTICATED USING DISTINGUISHED NAME;
```

```
ALTER USER brown GRANT CONNECT THROUGH WebDB
AUTHENTICATED USING CERTIFICATE TYPE 'X.509' VERSION '3';
```

ALTER VIEW

用途

ALTER VIEW は、無効なビューを明示的に再コンパイルする場合に使用します。明示的に再コンパイルすると、実行前にコンパイル・エラーを検査できます。再コンパイルは、ビューの実表を変更した後で、その変更がビューまたはそのビューに依存するオブジェクトに影響していないかどうかを確認するときに便利です。

ALTER VIEW を使用すると、制約のビューを定義、変更または削除できます。

ALTER VIEW 文を発行した場合、指定したビューは有効か無効にかかわらず再コンパイルされます。また、そのビューに依存するすべてのローカル・オブジェクトが無効になります。

注意：

- この文を使用して既存のビュー定義を変更することはできません。ビューを再定義する場合、OR REPLACE キーワードを指定した CREATE VIEW を使用する必要があります。
- 1 つ以上のマテリアライズド・ビューが参照しているビューを変更した場合、これらのマテリアライズド・ビューは無効になります。無効なマテリアライズド・ビューは、クエリー・リライトによって使用できません。また、リフレッシュすることもできません。

参照：

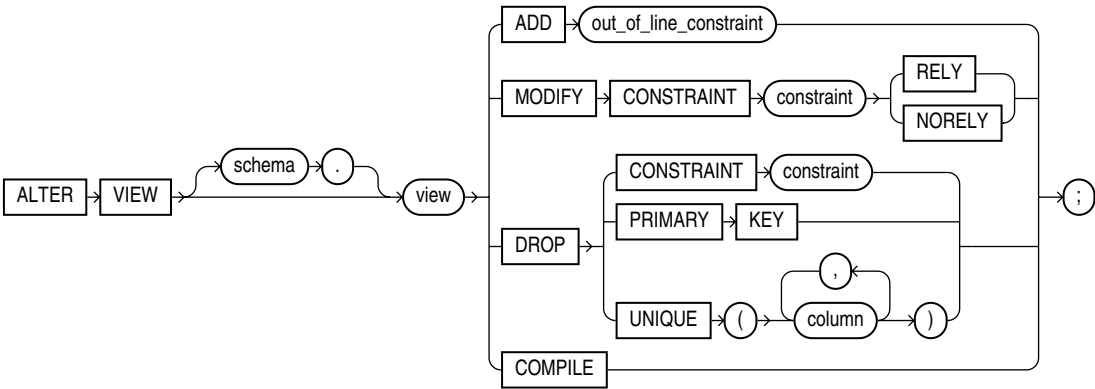
- ビューの再定義の詳細は、16-38 ページの「[CREATE VIEW](#)」を参照してください。
- 無効なマテリアライズド・ビュー・ログの再検証の詳細は、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- データ・ウェアハウスの概要は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- スキーマ・オブジェクト間の依存性については、『Oracle9i データベース概要』を参照してください。

前提条件

ビューが自分のスキーマ内にある必要があります。自分のスキーマ内にはない場合は、ALTER ANY TABLE システム権限が必要です。

構文

alter_view::=



(7-7 ページの *constraints* 構文の「*out_of_line_constraint::=*」を参照)

セマンティクス

schema

ビューを含むスキーマを指定します。*schema* を指定しない場合、ビューは自分のスキーマ内にあるとみなされます。

view

再コンパイルするビューの名前を指定します。

ADD 句

ADD 句を使用すると、*view* に制約を追加できます。

参照： 制約のビューおよび制限事項については、7-5 ページの「*constraints*」を参照してください。

MODIFY CONSTRAINT 句

MODIFY CONSTRAINT 句を使用すると、既存の制約のビューの RELY または NORELY 設定を変更できます。

制約の変更の制限事項： 一意制約または主キー制約が参照整合性制約の一部である場合、外部キーの削除または view の設定にあわせた変更をせずに、設定を変更することはできません。

参照： RELY および NORELY 設定の使用の詳細は、7-21 ページの「[RELY 句](#)」を参照してください。

DROP 句

DROP を使用すると、既存の制約のビューを削除できます。

制約の削除の制限事項： 一意制約または主キー制約がビューの参照整合性制約の一部である場合は削除できません。

COMPILE

COMPILE キーワードは必須です。指定したビューが再コンパイルされます。

例

ビューの変更例： 次の文は、16-51 ページの「[読込み専用ビューの作成例：](#)」で作成したビュー customer_ro を再コンパイルします。

```
ALTER VIEW customer_ro  
    COMPILE;
```

customer_ro の再コンパイル時にエラーが発生しなければ、customer_ro は有効になります。再コンパイル・エラーが発生した場合はエラーが戻り、customer_ro は無効のままとなります。

依存するオブジェクトもすべて無効になります。依存オブジェクトとは、customer_ro を参照する、プロシージャ、ファンクション、パッケージ本体、ビューなどです。その後、明示的に再コンパイルせずに、これらのオブジェクトを参照した場合、Oracle は、実行時にそれらを暗黙的に再コンパイルします。

ANALYZE

用途

ANALYZE 文を使用して、次のような非オブティマイザ統計情報を収集します。

- 索引または索引パーティション、表または表パーティション、索引構成表、クラスタまたはスカラー・オブジェクト属性の統計情報を収集または削除します。
- 索引または索引パーティション、表または表パーティション、索引構成表、クラスタまたはオブジェクト参照 (REF) の構造を検証します。
- 表またはクラスタの移行行と連鎖行を識別します。

注意： オブティマイザ統計情報の収集には、ANALYZE より DBMS_STATS パッケージを使用することをお勧めします。このパッケージを使用すると、パラレルでの統計収集、パーティション化オブジェクトに対するグローバル統計収集、および他の方法での統計収集の詳細なチューニングを行うことができます。また、統計情報に依存するコストベースのオブティマイザは、DBMS_STATS によって収集された統計情報のみを使用します。このパッケージの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

次のようなコスト・ベースのオブティマイザとは関係のない統計情報の収集には、(DBMS_STATS ではなく) ANALYZE 文を使用する必要があります。

- VALIDATE 句、LIST CHAINED ROWS 句の使用
 - 空きリストのブロックに関する情報の収集
-
-

前提条件

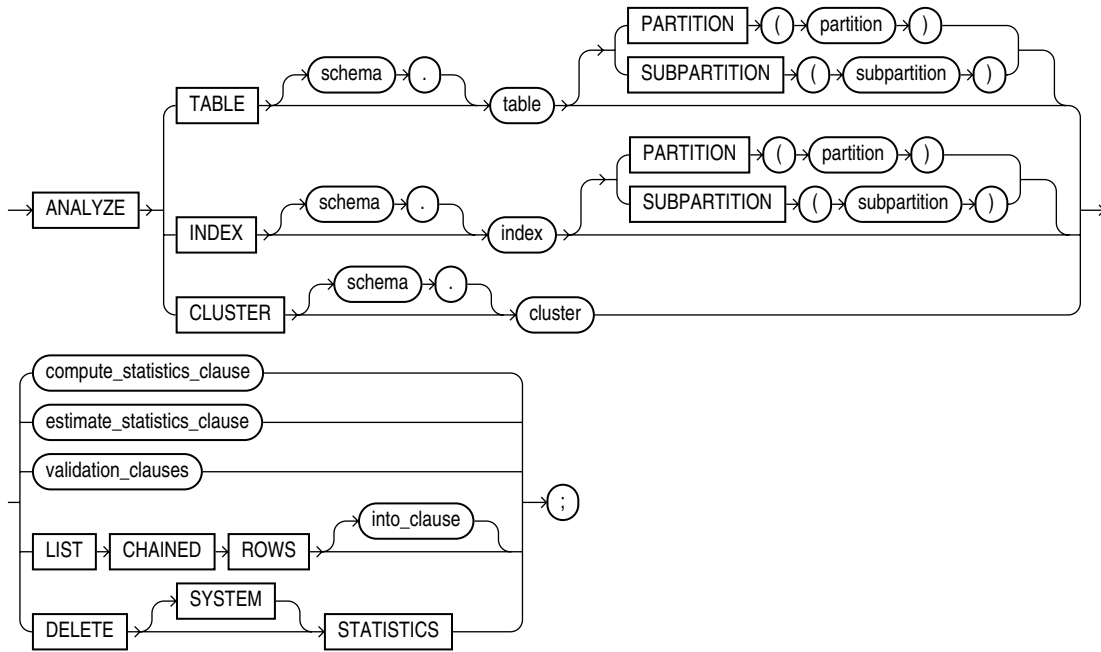
分析するスキーマ・オブジェクトがローカルである必要があります。自分のスキーマ内にならない場合は、ANALYZE ANY システム権限が必要です。

表またはクラスタの連鎖行をリスト表へ入れる場合、このリスト表が自分のスキーマ内にある必要があります。自分のスキーマ内にならない場合は、そのリスト表の INSERT 権限または INSERT ANY TABLE システム権限が必要です。

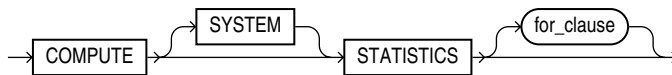
パーティション表の妥当性チェックを行う場合は、分析した ROWID を入れる表に対する INSERT 権限または INSERT ANY TABLE システム権限が必要です。

構文

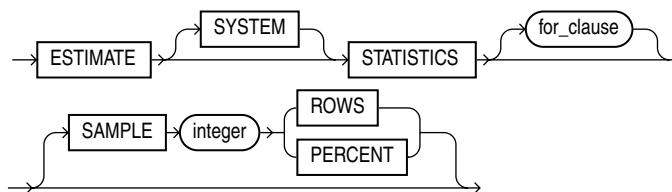
analyze::=



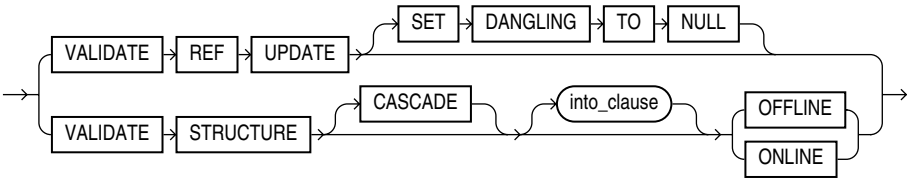
compute_statistics_clause::=



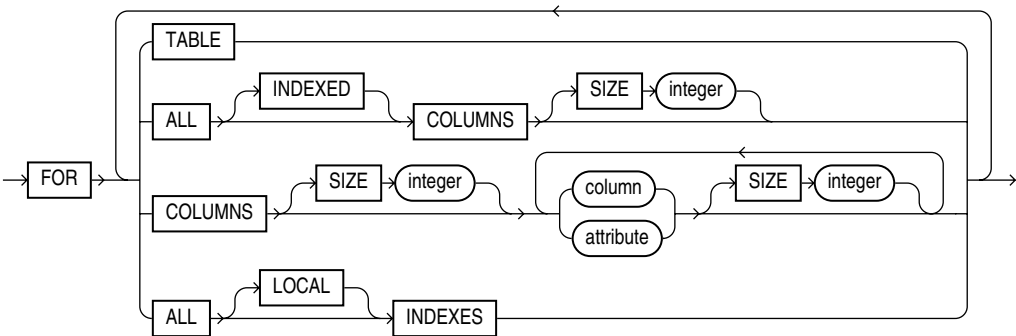
estimate_statistics_clause::=



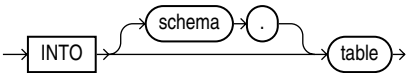
validation_clauses::=



for_clause::=



into_clause::=



セマンティクス

schema

索引、表またはクラスタが含まれているスキーマを指定します。*schema* を指定しない場合、索引、表またはクラスタは自分のスキーマ内にあるとみなされます。

INDEX index

分析する索引を指定します (*for_clause* を使用しない場合)。

索引については、次の統計情報が収集されます。アスタリスクが付いた統計は、常に厳密に計算されます。従来索引の場合、統計情報は、`USER_INDEXES`、`ALL_INDEXES` および `DBA_INDEXES` データ・ディクショナリ・ビューのカッコで示す列に表示されます。

- ルート・ブロックからリーフ・ブロック (BLEVEL) までの索引の深さ *
- リーフ・ブロックの数 (LEAF_BLOCKS)

- 個別索引値の数 (DISTINCT_KEYS)
- 索引の値ごとのリーフ・ブロックの平均数 (AVG_LEAF_BLOCKS_PER_KEY)
- (表に対する索引の) 索引の値ごとのデータ・ブロックの平均数 (AVG_DATA_BLOCKS_PER_KEY)
- クラスタ係数 (索引付きの値についての行が、どれだけ効率的に順序付けられているか) (CLUSTERING_FACTOR)

ドメイン索引の場合、索引に関連付けられた統計タイプで指定したユーザー定義統計収集ファンクションが、この文によって起動されます (12-46 ページの「[ASSOCIATE STATISTICS](#)」を参照)。ドメイン索引に関連付けられた統計タイプがない場合、その索引タイプに関連付けられた統計タイプが使用されます。索引またはその索引タイプの統計タイプがない場合、ユーザー定義統計情報は収集されません。ユーザー定義索引統計情報は、データ・ディクショナリ・ビュー USER_USTATS、ALL_USTATS および DBA_USTATS の STATISTICS 列に表示されます。

索引の分析の制限事項： IN_PROGRESS または FAILED のマークが付いたドメイン索引は分析できません。

参照：

- ドメイン索引の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 12-44 ページの「[索引の分析例](#)」を参照してください。

TABLE table

分析する表を指定します。表の統計情報を収集すると、*for_clauses* が使用されていないかぎり、各表の索引およびドメイン索引の統計情報も自動的に収集されます。

表を分析すると、すべてのファンクション索引に発生する式について統計情報が収集されます。したがって、表を分析する前に、必ずファンクション索引を作成してください。

参照： ファンクション索引の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。

表を分析すると、LOADING または FAILED のマークが付いたドメイン索引はすべてスキップされます。

索引構成表の場合、マッピング表が分析され、その PCT_ACCESSSS_DIRECT 統計情報も計算されます。これらの統計情報は、マッピング表のローカル ROWID の一部として格納されたと推測されるデータ・ブロック・アドレスの精度を評価します。

表については、次の統計情報が収集されます。アスタリスクが付いた統計は、常に厳密に計算されます。表の統計情報（ドメイン索引の状態を含む）は、データ・ディクショナリ・ビュー `USER_TABLES`、`ALL_TABLES` および `DBA_TABLES` のカッコで示す列に表示されます。

- 行数 (`NUM_ROWS`)
- 最高水位標を下回るデータ・ブロックの数（現在データを含むか含まないかにかかわらず、データを格納するようにフォーマットされているデータ・ブロックの数）× (`BLOCKS`)
- 未使用の表に対して割り当てられているデータ・ブロックの数× (`EMPTY_BLOCKS`)
- 各データ・ブロックにおける使用可能な空き領域サイズの平均値（バイト単位） (`AVG_SPACE`)
- 連鎖行の数 (`CHAIN_COUNT`)
- 行のオーバーヘッドを含む、バイト単位での行の平均の長さ (`AVG_ROW_LEN`)

表の分析の制限事項：

- `ANALYZE` を使用して、データ・ディクショナリ表の統計情報を収集しないでください。
- `ANALYZE` を使用して、外部表の統計情報を収集しないでください。ただし、この用途に `DBMS_STATS` パッケージは使用できます。
- `ANALYZE` を使用して、一時表のデフォルト統計情報を収集しないでください。ただし、一時表の 1 つ以上の列とユーザー定義統計タイプを対応付けている場合、`ANALYZE` を使用して一時表のユーザー定義統計情報を収集できます（`ANALYZE` 使用前に対応付けが行われている必要があります）。
- 統計情報の計算または推定ができない列型、`REF`、`VARRAY`、ネストした表、`LOB`（`LOB` は分析されずにスキップされる）、`LONG`、オブジェクト型などです。ただし、このような列に統計タイプが対応付けられている場合は、ユーザー定義統計情報が収集されます。

参照：

- 12-46 ページの「[ASSOCIATE STATISTICS](#)」を参照してください。
- データ・ディクショナリ・ビューの詳細は、『*Oracle9i データベース・リファレンス*』を参照してください。

PARTITION | SUBPARTITION

統計を収集するパーティションまたはサブパーティションを指定します。クラスタの分析時にこの句は使用できません。

`table` がコンポジット・パーティションのときに `PARTITION` を指定した場合、指定したパーティション内ですべてのサブパーティションが分析されます。

CLUSTER *cluster*

分析するクラスタを指定します。クラスタの統計情報を収集した場合、すべてのクラスタ表、およびクラスタ索引を含むすべての索引の統計も自動的に収集されます。

索引クラスタとハッシュ・クラスタには、単一クラスタ・キー (AVG_BLOCKS_PER_KEY) が使用するデータ・ブロックの平均数が収集されます。これらの統計情報は、データ・ディクショナリ・ビュー ALL_CLUSTERS、USER_CLUSTERS および DBA_CLUSTERS に表示されます。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』および 12-45 ページの「[クラスタの分析例 :](#)」を参照してください。

compute_statistics *clause*

COMPUTE STATISTICS は、分析対象のオブジェクトの正確な統計情報を計算してデータ・ディクショナリに格納します。表を分析した場合、表および列の統計情報が収集されます。

計算された統計情報および推定された統計情報は、分析したオブジェクトにアクセスする SQL 文の実行計画を選択するために、Oracle オプティマイザによって使用されます。また、これらの統計情報は SQL 文を記述するアプリケーション開発者にも役立ちます。

ユーザー定義統計情報ではなく、システム統計情報のみを計算する場合は、SYSTEM を指定します。SYSTEM を省略すると、システムが生成した統計情報と統計タイプで宣言した収集ファンクションによって生成された統計情報の両方が収集されます。

参照：

- 統計収集ファンクションの作成の詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- これらの統計情報の使用方法の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- 12-43 ページの「[統計情報の計算例 :](#)」を参照してください。

for_clause

for_clause は、表または索引全体を分析するか、特定の列のみを分析するかを指定します。次に示す句は、この文の ANALYZE TABLE にのみ使用できます。

FOR TABLE FOR TABLE を指定して、表および列の情報ではなく、表のみの統計情報が収集されるように制限します。

FOR COLUMNS FOR COLUMNS を指定して、すべての列または属性の列統計情報ではなく、指定した列およびスカラー・オブジェクト属性のみの列統計情報が収集されるように統計収集ファンクションを制限します。attribute には、オブジェクト内の項目の修飾列名を指定します。

FOR ALL COLUMNS FOR ALL COLUMNS を指定して、すべての列およびスカラー・オブジェクト属性の列統計情報を収集します。

FOR ALL INDEXED COLUMNS FOR ALL INDEXED COLUMNS を指定して、表にあるすべての索引付きの列の列統計情報を収集します。

列の統計情報を収集する場合、列全体に基づいた情報を収集するか、**SIZE** を指定してヒストグラムを使用します。

Oracle は、次の列統計情報を収集します。

- 列全体として、重複していない値の数
- 各区間の最大値と最小値

参照： ヒストグラムの詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』および 12-44 ページの「[ヒストグラムの作成例](#)」を参照してください。

列の統計情報は、データ・ディクショナリ・ビュー USER_TAB_COLUMNS、ALL_TAB_COLUMNS および DBA_TAB_COLUMNS に表示されます。ヒストグラムは、データ・ディクショナリ・ビュー USER_TAB_HISTOGRAMS、DBA_TAB_HISTOGRAMS、ALL_TAB_HISTOGRAMS;USER_PART_HISTOGRAMS、DBA_PART_HISTOGRAMS、ALL_PART_HISTOGRAMS;USER_SUBPART_HISTOGRAMS、DBA_SUBPART_HISTOGRAMS および ALL_SUBPART_HISTOGRAMS に表示されます。

注意： USER_TAB_COLUMNS、DBA_TAB_COLUMNS および ALL_TAB_COLUMNS の MAXVALUE および MINVALUE 列は、長さが 32 バイトです。32 バイトより長い列を分析する場合、および列に先行空白が埋め込まれている場合、Oracle は、先行空白のみを考慮し、予期しない統計情報を戻します。

ユーザー定義型が列に関連付けられている場合、*for_clause* は、その統計タイプを使用してユーザー定義統計情報を収集します。列に関連付けられた統計タイプがない場合、列の型に関連付けられた統計タイプがあるかがチェックされ、ある場合は、その統計タイプが使用されます。列またはそのユーザー定義型に関連付けられた統計タイプがない場合、ユーザー定義統計情報は収集されません。ユーザー定義列統計情報は、データ・ディクショナリ・ビュー USER_USTATS、ALL_USTATS および DBA_USTATS で STATISTICS 列に表示されます。

表全体および 1 つ以上の列の両方の統計情報を収集する場合、まず、表の統計情報を作成してから、次に、列の統計情報を作成してください。このようにしないと、表のみの ANALYZE が列 ANALYZE で作成されたヒストグラムを上書きしてしまいます。次の文はこの手順の例です。

```
ANALYZE TABLE emp ESTIMATE STATISTICS;
ANALYZE TABLE emp ESTIMATE STATISTICS
    FOR ALL COLUMNS;
```

FOR ALL INDEXES FOR ALL INDEXES を指定して、表に関連付けられたすべての索引を分析します。

FOR ALL LOCAL INDEXES FOR ALL LOCAL INDEXES を指定して、すべてのローカル索引パーティションを分析します。PARTITION 句および INDEX が指定されている場合、キーワード LOCAL を指定する必要があります。

SIZE ヒストグラムのバケットの最大数を指定します。デフォルト値は 75 で、最小値は 1、最大値は 254 です。

注意： サンプルの行数以上のバケットを含むヒストグラムは作成されません。また、サンプルに繰返しが非常に多い値が含まれる場合、指定された数のバケットは作成されますが、ALL_TAB_COLUMNS、DBA_TAB_COLUMNS および USER_TAB_COLUMNS ビューの NUM_BUCKETS 列で指定した値は、内部圧縮アルゴリズムのために小さくなる場合があります。

参照： 12-44 ページの「[ヒストグラムの作成例:](#)」を参照してください。

estimate_statistics_clause

ESTIMATE STATISTICS は、分析対象オブジェクトの統計情報を概算してデータ・ディクショナリに格納します。

計算された統計情報および推定された統計情報は、分析したオブジェクトにアクセスする SQL 文の実行計画を選択するために、Oracle オプティマイザによって使用されます。また、これらの統計情報は SQL 文を記述するアプリケーション開発者にも役立ちます。

ユーザー定義統計情報ではなく、システム統計情報のみを推定する場合は、SYSTEM を指定します。SYSTEM を省略すると、システムが生成した統計情報と統計タイプで宣言した収集ファンクションによって生成された統計情報の両方が推定されます。

参照： 統計収集ファンクションの作成の詳細は、『Oracle9i Data Cartridge Developer's Guide』および 12-44 ページの「[統計情報の推定例:](#)」を参照してください。

for_clause 12-37 ページの「[compute_statistics_clause](#)」を参照してください。

SAMPLE 統計情報を推定するために、分析対象のオブジェクトからサンプルとして抽出されるデータ量を指定します。このパラメータを指定しない場合、サンプルとして 1064 行が抽出されます。

サンプルのデフォルト値は、数千行までが表に対して適切な値です。表が大きい場合、より大きい値を **SAMPLE** に指定します。データの半分以上を指定した場合、すべてのデータが読み込まれ、統計が計算されます。

- **ROWS** は、表またはクラスタの *integer* 行、または索引の *integer* エントリ数をサンプルとして抽出します。**integer** は 1 以上である必要があります。
- **PERCENT** は、表またはクラスタの *integer* パーセント、または索引エントリの *integer* パーセントをサンプルとして抽出します。**integer** は 1 ～ 99 の範囲で指定します。

参照： これらの統計情報の使用方法の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

validation_clauses

validation_clauses を使用すると、REF および分析したオブジェクトの構造が検証されます。

VALIDATE REF UPDATE 句

VALIDATE REF UPDATE を指定して、指定された表の REF の妥当性チェックを行い、各 REF 内の ROWID 部分をチェックし、それを真の ROWID と比較します。誤りがある場合は修正します。この句は、表を分析する場合にのみ使用できます。

SET DANGLING TO NULL **SET DANGLING TO NULL** を指定すると、指定した表内の REF が（範囲が限定されるかどうかにかかわらず）無効なオブジェクトまたは存在しないオブジェクトを指している場合に、REF が NULL に設定されます。

注意： 表の所有者が参照先オブジェクトに対する **SELECT** オブジェクト権限を持っていない場合、このオブジェクトは無効とみなされ、NULL に設定されます。この結果、オブジェクトに対して適切な権限があるユーザーによって発行された問合せであっても、問合せでこれらの REF を使用することはできません。

VALIDATE STRUCTURE

VALIDATE STRUCTURE を指定すると、分析対象オブジェクトの構造が検証されます。COMPUTE STATISTICS 句および ESTIMATE STATISTICS 句で収集された統計情報は、Oracle オプティマイザで使用されますが、この句で収集された統計情報は、Oracle オプティマイザでは使用されません。

参照： 12-44 ページの「[表の検証例](#)」を参照してください。

- 表に対して、表のそれぞれのデータ・ブロックと行の整合性を検証します。索引構成表の場合、表の主キー索引に対する圧縮統計情報（最適なプレフィックス圧縮件数）も生成されます。
- クラスタに対して、自動的にクラスタ表の構造を検証します。
- パーティションに対して、行が適切なパーティションに属するかどうかを検証します。行が正しく照合されなかった場合は、ROWID が INVALID_ROWS 表に挿入されます。
- 一時表に対して、現行のセッション中に表の構造および索引を検証します。
- 索引に対して、索引のそれぞれのデータ・ブロックの整合性を検証し、ブロックの破損をチェックします。この句は、表のそれぞれの行が索引エントリを持っていること、またはそれぞれの索引エントリが表の行を指していることを確認するわけではありません。これらを確認する場合は、[CASCADE](#) 句を使用して表の構造を検証します。

通常のすべての索引用の圧縮統計情報（最適なプレフィックス圧縮件数）も計算されます。

データ・ディクショナリ・ビュー INDEX_STATS および INDEX_HISTOGRAM 内にある索引についての統計情報が格納されます。

参照： これらのビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

オブジェクトの構造に障害がある場合は、エラー・メッセージが戻ります。この場合、オブジェクトを削除して作成しなおす必要があります。

INTO VALIDATE STRUCTURE の INTO 句は、パーティション表のみに有効です。正しく照合されなかった行を持つパーティションの ROWID を格納するリスト表を指定します。schema を指定しない場合、リストは自分のスキーマ内にあるとみなされます。この句自体を指定しない場合、表の名前は INVALID_ROWS になります。この表を作成するために使用する SQL スクリプトは UTLVALID.SQL です。

CASCADE 表またはクラスタに関連付けられた索引の構造を検証する場合は、CASCADE を指定します。表を検証するときにこの句を指定すると、その表の索引も検証されます。クラスタを検証するときにこの句を指定した場合、クラスタ化表のすべての索引（クラスタ索引を含む）が検証されます。

この句を使用して使用可能な（以前は使用禁止であった）ファンクション索引を検証すると、検証エラーになる場合があります。この場合は、索引を再構築する必要があります。

ONLINE | OFFLINE ONLINE を指定すると、Oracle がオブジェクトの DML 操作中に検証を実行できるようになります。Oracle は、並行して操作が行える程度に、実行する検証の量を減らします。

注意： OFFLINE 指定でオブジェクトの構造を検証する場合と同様に、ONLINE 指定でオブジェクトの構造を検証する場合、Oracle は統計を収集しません。

OFFLINE を指定すると、実行する検証の量を最大化します。この設定は、検証中に INSERT、UPDATE および DELETE 文がオブジェクトに平行してアクセスすることを防ぎますが、問合せは許可されます。これはデフォルト値です。

ONLINE の制限事項： ONLINE は、クラスタ化されているオブジェクトを分析する場合には指定できません。

LIST CHAINED ROWS

LIST CHAINED ROWS によって、分析対象の表またはクラスタの移行行および連鎖行を識別できます。索引の分析時にこのオプションは使用できません。

INTO 句には、移行行および連鎖行をリストする表を指定します。*schema* を指定しない場合、この表が自分のスキーマにあるものとみなされます。この句自体を指定しない場合、表の名前は CHAINED_ROWS になります。このリスト表はローカル・データベース内にある必要があります。

次のいずれかのスクリプトを使用して、CHAINED_ROWS 表を作成できます。

- UTLCHAIN.SQL は、物理 ROWID を使用します。そのため、行は、索引構成表からではなく従来表から収集されます（次の注意を参照）。
- UTLCHN1.SQL は、ユニバーサル ROWID を使用します。そのため、行は、従来表および索引構成表の両方から収集されます。

独自の連鎖行表を作成する場合、この 2 つのスクリプトのいずれかで規定されるフォーマットに従う必要があります。

注意： ユニバーサル ROWID ではなく、主キーに基づく索引構成表を分析する場合、索引構成表ごとに別の連鎖行表を作成し、主キー記憶域を確保する必要があります。まず、SQL スクリプトの DBMSIOTC.SQL および PRVTIOTC.PLB を使用して、BUILD_CHAIN_ROWS_TABLE プロシージャを定義します。次に、このプロシージャを実行して、索引構成表の IOT_CHAINED_ROWS 表を作成します。

参照：

- これらのスクリプトを使用する場合の互換性については、『Oracle9i データベース移行ガイド』を参照してください。
- パッケージ化された SQL スクリプトの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_IOT パッケージを参照してください。
- 移行行および連鎖行の削除については、『Oracle9i データベース管理者ガイド』を参照してください。
- 12-45 ページの「[連鎖行のリスト例 :](#)」を参照してください。

DELETE STATISTICS

DELETE STATISTICS は、現在データ・ディクショナリに格納されている、分析対象のオブジェクトの統計情報を削除します。Oracle に統計情報を使用させないようにする場合、この文を使用します。

表を指定してこの句を使用した場合、指定した表のすべての索引の統計情報も自動的に削除されます。クラスタを指定してこの句を使用した場合、指定したクラスタのすべての表、およびこれらの表のすべての索引（クラスタ索引を含む）の統計情報が自動的に削除されます。

ユーザー定義統計情報ではなく、システム統計情報のみを削除する場合は、SYSTEM を指定します。SYSTEM を省略すると、オブジェクトのユーザー定義列または索引統計情報が収集された場合、Oracle は、統計情報を収集するために使用された情報タイプに指定されている統計削除ファンクションを起動して、ユーザー定義統計情報も削除します。

参照： 12-44 ページの「[統計情報の削除例 :](#)」を参照してください。

例

統計情報の計算例： 次の文は、サンプル表 oe.orders の統計情報を計算します。

```
ANALYZE TABLE orders COMPUTE STATISTICS;
```

次の文は、サンプル表 oe.orders のシステム統計情報のみを計算します。

```
ANALYZE TABLE orders COMPUTE SYSTEM STATISTICS;
```

次の文は、スカラー・オブジェクト属性の統計情報を計算します。

```
ANALYZE TABLE customers COMPUTE STATISTICS
FOR COLUMNS cust_address.postal_code;
```

統計情報の推定例： 次の文は、サンプル表 `oe.orders` とそのすべての索引の統計情報を推定します。

```
ANALYZE TABLE orders ESTIMATE STATISTICS;
```

統計情報の削除例： 次の文は、サンプル表 `oe.orders` とデータ・ディクショナリにあるそのすべての索引の統計情報を削除します。

```
ANALYZE TABLE orders DELETE STATISTICS;
```

ヒストグラムの作成例： 次の文は、サンプル表 `hr.locations` の `country_id` 列について、10 区間のヒストグラムを作成します。

```
ANALYZE TABLE locations
  COMPUTE STATISTICS FOR COLUMNS country_id SIZE 10;
```

`USER_TAB_COLUMNS` データ・ディクショナリ・ビューに問い合せて、統計情報を取り出すことができます。

```
SELECT NUM_DISTINCT, NUM_BUCKETS, SAMPLE_SIZE
  FROM USER_TAB_COLUMNS
 WHERE TABLE_NAME = 'LOCATIONS' AND COLUMN_NAME = 'COUNTRY_ID';

NUM_DISTINCT NUM_BUCKETS SAMPLE_SIZE
-----
          14             7           23
```

表のサイズによっては、`ANALYZE` 文で 10 バケットを指定しても、その `ANALYZE` 文で指定したバケットより少ない数が作成される場合があります。詳細は、12-39 ページの「[SIZE](#)」を参照してください。

また、表の単一のパーティションのヒストグラムも収集できます。次の文は、サンプル表 `sh.sales` のパーティション `sales_q2_2000` を分析します。

```
ANALYZE TABLE sales PARTITION (sales_q2_2000) COMPUTE STATISTICS;
```

索引の分析例： 次の文は、サンプル索引 `oe.inv_product_ix` の構造を検証します。

```
ANALYZE INDEX inv_product_ix VALIDATE STRUCTURE;
```

表の検証例： 次の文は、サンプル表 `hr.employees` およびそのすべての索引を分析します。

```
ANALYZE TABLE employees VALIDATE STRUCTURE CASCADE;
```

表に対する `VALIDATE REF UPDATE` 句は、指定した表の `REF` を検証します。また、それぞれの `REF` の `ROWID` 部分をチェックし、それを真の `ROWID` と比較します。その結果、`ROWID` が誤っていると判断されると、`ROWID` 部分が正しくなるように `REF` が更新されます。

次の文は、サンプル表 `oe.customers` の `REF` を検証します。

```
ANALYZE TABLE customers VALIDATE REF UPDATE;
```

次の文は、`DML` 操作を同時に実行することを許可して、サンプル表 `oe.customers` の構造を検証します。

```
ANALYZE TABLE customers VALIDATE STRUCTURE ONLINE;
```

クラスタの分析例： 次の文は、`personnel` クラスタ（13-9 ページの「[クラスタの作成例](#)」で作成）、そのすべての表、クラスタ索引を含むすべての索引を分析します。

```
ANALYZE CLUSTER personnel
      VALIDATE STRUCTURE CASCADE;
```

連鎖行のリスト例： 次の文は、`orders` 表のすべての連鎖行についての情報を収集します。

```
ANALYZE TABLE orders
      LIST CHAINED ROWS INTO chained_rows;
```

前述の文では、情報は表 `chained_rows` に格納されます。次の問合せでその行を検証できます（表に連鎖行が含まれない場合は、行は戻されません）。

```
SELECT owner_name, table_name, head_rowid, analyze_timestamp
      FROM chained_rows;
```

OWNER_NAME	TABLE_NAME	HEAD_ROWID	ANALYZE_TIMESTAMP
-----	-----	-----	-----
OE	ORDERS	AAAAZzAABAAABrXAAA	25-SEP-2000

ASSOCIATE STATISTICS

用途

ASSOCIATE STATISTICS 文を使用すると、統計収集、選択性またはコストに関するファンクションが含まれた統計タイプ（またはデフォルトの統計）を、1 つ以上の列、スタンドアロン・ファンクション、パッケージ、型、ドメイン索引または索引タイプに関連付けることができます。

現在のすべての統計タイプの関連付けの一覧については、USER_ASSOCIATIONS データ・ディクショナリ・ビューに問い合わせます。統計情報に関連付けられたオブジェクトを分析する場合、USER_USTATS ビューでその関連性を問い合わせることができます。

参照： ANALYZE が関連性で使用する優先順位については、12-32 ページの「ANALYZE」を参照してください。

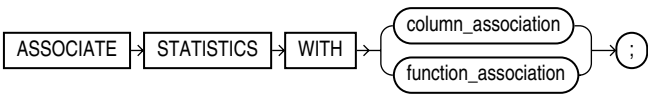
前提条件

この文を発行する場合は、ベース・オブジェクト（表、ファンクション、パッケージ、型、ドメイン索引または索引タイプ）を変更する適切な権限が必要です。さらに、デフォルト統計情報のみを関連付けていないかぎり、統計タイプに対する実行権限が必要です。統計タイプは、すでに定義されている必要があります。

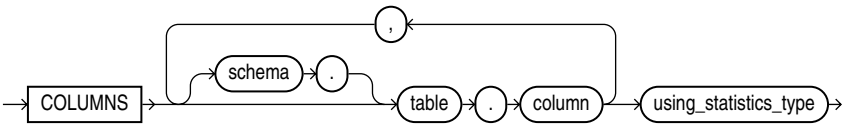
参照： 型の定義の詳細は、16-3 ページの「CREATE TYPE」を参照してください。

構文

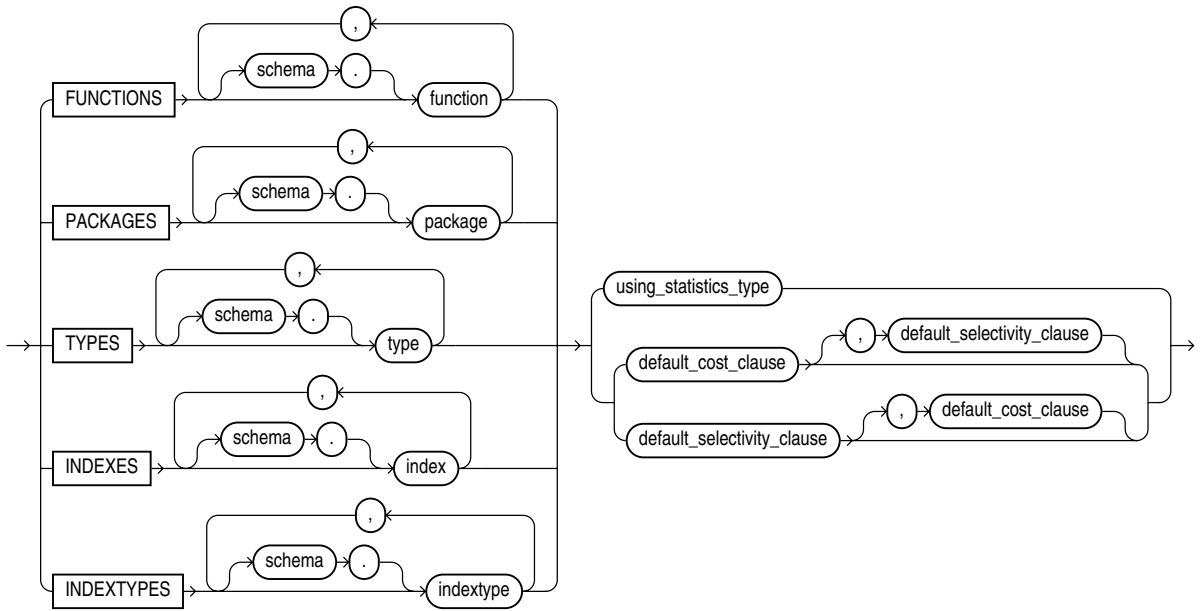
associate_statistics::=



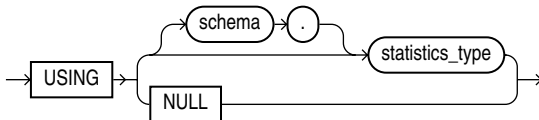
column_association::=



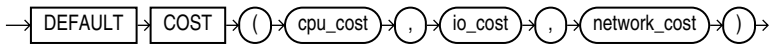
function_association::=



using_statistics_type::=



default_cost_clause::=



default_selectivity_clause::=



セマンティクス

column_association

1 つ以上の表の列を指定します。 *schema* を指定しない場合、表が自分のスキーマ内にあるとみなされます。

function_association

1 つ以上のスタンドアロン・ファンクション、パッケージ、ユーザー定義データ型、ドメイン索引または索引タイプを指定します。 *schema* を指定しない場合、オブジェクトは自分のスキーマ内にあるとみなされます。

- *FUNCTIONS* は、スタンドアロン・ファンクションのみを参照し、メソッド型または組込みファンクションは参照しません。
- *TYPES* はユーザー定義型のみを参照し、組込み SQL データ型は参照しません。

***function_association* の制限事項：** すでに関連性を定義してあるオブジェクトには指定できません。まず、このオブジェクトと統計情報の関連性を取り消す必要があります。

参照： 16-61 ページの「[DISASSOCIATE STATISTICS](#)」および 12-49 ページの「[統計情報の関連付け例](#)」を参照してください。

using_statistics_type

列、ファンクション、パッケージ、型、ドメイン索引または索引タイプと関連付けられている統計タイプ（または型のシノニム）を指定します。 *statistics_type* は作成済である必要があります。

NULL キーワードは、統計情報を列または索引に関連付ける場合のみに有効です。統計タイプをオブジェクト型に関連付ける場合は、そのオブジェクト型の列は、統計タイプを継承します。同様に、統計タイプを索引タイプに関連付ける場合は、索引タイプの索引インスタンスは統計タイプを継承します。列または索引に別の統計タイプを関連付けると、この継承を上書きできます。または、列または索引に統計タイプを関連付けない場合は、*using_statistics_type* 句で NULL を指定します。

統計タイプの指定の制限事項： ファンクション、パッケージ、型または索引タイプには NULL を指定できません。

参照： 統計収集ファンクションの作成の詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

default_cost_clause

スタンドアロン・ファンクション、パッケージ、型、ドメイン索引または索引タイプのデフォルトのコストを指定します。この句を指定する場合、CPU コスト、I/O コスト、ネットワーク・コストの順でそれぞれに対して 1 つの数を指定する必要があります。それぞれのコストは、ファンクションまたはメソッドを一度実行した場合、またはドメイン索引へ一度アクセスした場合の値です。指定できる値は 0（ゼロ）以上の整数です。

default_selectivity_clause

スタンドアロン・ファンクション、型、パッケージまたはユーザー定義演算子が指定された述語に対するデフォルトの選択性をパーセントで指定します。default_selectivity は、0 ～ 100 の整数値である必要があります。範囲外の場合は無視されます。

default_selectivity_clause 句の制限事項： DEFAULT SELECTIVITY は、ドメイン索引または索引タイプに指定できません。

参照： 12-49 ページの「[デフォルト・コストの指定例](#)」を参照してください。

例

統計情報の関連付け例： 次の文は、スタンドアロン・パッケージ emp_mgmt（14-51 ページの「[パッケージの作成例](#)」で作成）への関連性を作成します。

```
ASSOCIATE STATISTICS WITH PACKAGES emp_mgmt DEFAULT SELECTIVITY 10;
```

デフォルト・コストの指定例： 次の文は、ドメイン索引 t_a を使用して任意の述語を実装した場合、常に CPU コストは 100、I/O は 5、およびネットワーク・コストは 0 になるように指定します。

```
ASSOCIATE STATISTICS WITH INDEXES t_a DEFAULT COST (100,5,0);
```

オブティマイザは、コスト・ファンクションをコールするかわりに、これらのデフォルト・コストをそのまま使用します。

AUDIT

用途

AUDIT 文を使用すると、次の処理ができます。

- 後続のユーザー・セッションでの SQL 文の状態変化の監査。特定の SQL 文または特定のシステム権限によって許可されたすべての SQL 文の状態変化を監査します。SQL 文操作の監査は、後続セッションにのみ適用され、現行のセッションには適用されません。
- 特定のスキーマ・オブジェクトに対する操作の監査。スキーマ・オブジェクト操作の監査は、後続のセッションと同様に、現行のセッションにも適用されます。

参照：

- 値に基づいた監査方針を作成および管理する DBMS_FGA パッケージの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- SQL 文の監査を使用禁止にする場合の詳細は、17-79 ページの「[NOAUDIT](#)」を参照してください。

前提条件

SQL 文の状態変化を監査するには、AUDIT SYSTEM システム権限が必要です。

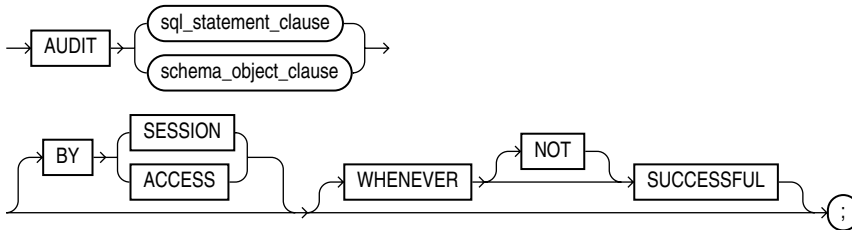
スキーマ・オブジェクト操作を監査するためには、監査対象のオブジェクトが自分のスキーマにあるか、AUDIT ANY システム権限が必要です。また、監査の対象とするオブジェクトがディレクトリ・オブジェクトの場合は、それが自分で作成したものであっても、AUDIT ANY システム権限が必要です。

監査結果を収集するには、初期化パラメータ AUDIT_TRAIL をデータベースに設定する必要があります。監査オプションは、監査が使用可能であるかどうかにかかわらず指定できます。ただし、監査を使用可能にしなければ、監査レコードは作成されません。

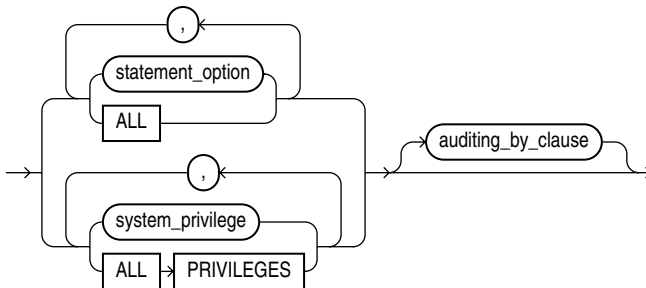
参照： AUDIT_TRAIL パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

構文

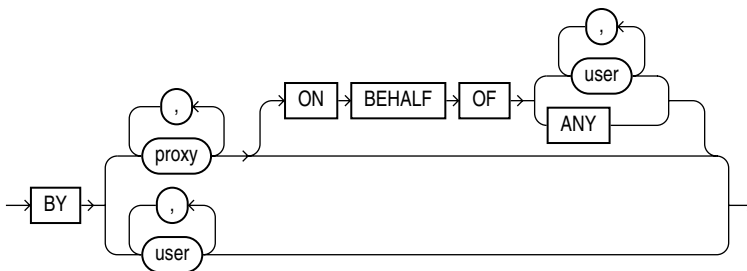
audit::=



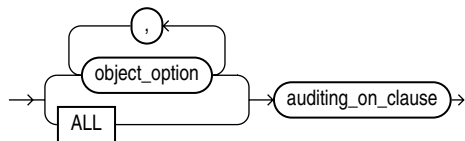
sql_statement_clause::=



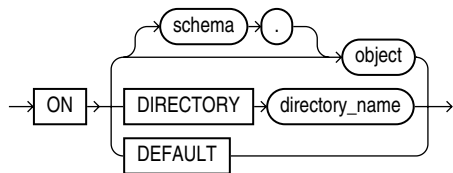
auditing_by_clause::=



schema_object_clause::=



auditing_on_clause::=



セマンティクス

sql_statement_clause

sql_statement_clause を使用して、SQL 文を監査します。

statement_option

特定の SQL 文を監査するには、文オプションを指定します。

監査されるたびに、次の情報を持つ監査レコードが生成されます。

- 操作を行ったユーザー
- 操作の種類
- 操作に関連するオブジェクト
- 操作の日付と時刻

監査レコードは、監査証跡に書き込まれます。監査証跡とは、監査レコードが入っているデータベースの表です。データ・ディクショナリ・ビューを問い合わせて監査証跡を調べることによって、データベース・アクティビティを再検討できます。

参照：

- 文オプションおよびそれによって監査される SQL 文のリストは、12-56 ページの表 12-1 および 12-59 ページの表 12-2 を参照してください。
- 監査証跡データ・ディクショナリ・ビューのリストの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 12-60 ページの「[ロールに関連する SQL 文の監査例 :](#)」を参照してください。

system_privilege

特定のシステム権限を与えられた SQL 文を監査する場合は、システム権限を指定します。

多くの個々のシステム権限を指定するのではなく、ロール CONNECT、RESOURCE および DBA を指定できます。これは、すべてのシステム権限がそのロールに付与されていることを監査することと同じです。

Oracle には、システム権限と文オプションをまとめて指定するための、次の 2 つのショートカットが用意されています。

ALL ALL は、表 12-1 のすべての文オプションを監査しますが、表 12-2 の追加文オプションを監査しません。

ALL PRIVILEGES システム権限を監査するためには、ALL PRIVILEGES を指定します。

注意： ロールまたはショートカットでなく、監査に個々のシステム権限および文オプションを指定することをお勧めします。ロールおよびショートカットに含まれるシステム権限および文オプションは、リリースごとに異なり、Oracle の今後のバージョンでサポートされない場合があります。

参照：

- すべてのシステム権限とそれによって許可される SQL 文のリストは、表 17-1 「[システム権限](#)」を参照してください。
- CONNECT、RESOURCE および DBA ロールの詳細は、17-27 ページの「[GRANT](#)」を参照してください。
- 12-61 ページの「[問合せおよび更新を行う SQL 文の監査例 :](#)」、12-61 ページの「[監査の削除例 :](#)」、12-61 ページの「[ディレクトリに関連する文の監査例 :](#)」を参照してください。

auditing_by_clause

auditing_by_clause は、特定のユーザーが発行する SQL 文のみを監査します。この句を指定しない場合、すべてのユーザー文が監査されます。

BY user この句は、特定のユーザーによって発行された SQL 文のみを監査するように制限します。

BY proxy この句は、特定のプロキシによって発行された SQL 文のみを監査するように制限します。

参照： プロキシおよびそれらのデータベースについては、『Oracle9i データベース概要』を参照してください。

ON BEHALF OF *user* は、特定のユーザーのプロキシとして実行された文を監査することを指定します。ANY は、すべてのユーザーのプロキシとして実行された文を監査することを指定します。

schema_object_clause

schema_object_clause は、スキーマ・オブジェクトの操作を監査します。

object_option

監査の対象とする操作を指定します。12-60 ページの表 12-3 に、各オブジェクト・オプションとそれが適用されるオプションのタイプを示します。それぞれのオブジェクト・オプションには、監査の対象となる SQL 文を指定します。たとえば、ALTER オプションを指定して表の監査を選択した場合、その表に対して発行される ALTER TABLE 文がすべて監査されます。また、SELECT オプションを指定して順序の監査を選択した場合、その順序の値を使用するすべての文が監査されます。

ALL

ALL をショートカットに指定することは、オブジェクト・タイプに適用できるオプションをすべて指定することと同じです。

auditing_on_clause

auditing_on_clause は、特定のスキーマ・オブジェクトを監査できます。

参照： 12-61 ページの「表の問合せの監査例：」、12-62 ページの「表の挿入および更新の監査例：」および 12-62 ページの「順序の操作の監査例：」を参照してください。

schema 監査の対象として選択されたオブジェクトが定義されているスキーマを指定します。*schema* を指定しない場合、オブジェクトは、自分のスキーマ内に定義されているものとみなされます。

object 監査するオブジェクトの名前を指定します。オブジェクトは、表、ビュー、順序、ストアド・プロシージャ、ストアド・ファンクション、ストアド・パッケージ、マテリアライズド・ビューまたはライブラリのいずれかである必要があります。

表、ビュー、順序、プロシージャ、ストアド・ファンクション、パッケージ、マテリアライズド・ビューまたはユーザー定義型については、それぞれシノニムも指定できます。

ON DEFAULT ON DEFAULT を指定して、オブジェクトを作成した後に特定のオブジェクト・オプションをデフォルト・オブジェクト・オプションとして構築します。デフォルト監査オプションを指定した場合、その後作成されるオブジェクトに対して、これらのオプションが自動的に適用され、監査が行われます。ビューに対するデフォルト監査オプションは、常に、そのビューの実表に対する監査オプションの論理和となります。

ALL_DEF_AUDIT_OPTS データ・ディクショナリ・ビューを問い合わせることによって、現在のデフォルト監査オプションを表示できます。

デフォルト監査オプションを変更した場合でも、以前作成したオブジェクトの監査オプションはそのまま残ります。AUDIT 文の ON 句にオブジェクトを指定した場合のみ、既存のオブジェクトの監査オプションを変更できます。

参照： 12-62 ページの「[デフォルト監査オプションの設定例 :](#)」を参照してください。

ON DIRECTORY directory_name ON DIRECTORY 句によって、監査対象のディレクトリ名を指定することができます。

BY SESSION

同一セッションの同一スキーマ・オブジェクトで発行された同じ種類の SQL 文すべて、および実行された同じ種類の操作について、1 つのレコードが書き込まれるようにする場合は、BY SESSION を指定します。

注意： Oracle は、オペレーティング・システム監査ファイルには書き込めますが、すでにエントリが特定の操作に書き込まれたかを検出するために読み込むことはできません。したがって、監査証跡にオペレーティング・システム・ファイルを使用している場合 (AUDIT_FILE_DEST 初期化パラメータを OS に設定してある場合)、BY SESSION を指定しても、Oracle は複数のレコードを監査証跡ファイルに書き込む場合があります。

BY ACCESS

監査された各文および操作について、1 つのレコードを書き込む場合は、BY ACCESS を指定します。

DDL 文を監査する文オプションまたはシステム権限を指定した場合、BY SESSION 句と BY ACCESS 句のどちらを指定しても、Oracle は、自動的に監査を行います。

DDL 文以外の SQL 文を監査する文オプションとシステム権限には、BY SESSION と BY ACCESS のどちらでも指定できます。デフォルトは BY SESSION です。

WHENEVER [NOT] SUCCESSFUL

SQL 文および操作が正常に実行された場合のみに監査する場合は、WHENEVER SUCCESSFUL を指定します。

SQL 文および操作が失敗またはエラーが発生した場合のみに監査する場合は、WHENEVER NOT SUCCESSFUL を指定します。

この句を省略すると、処理結果にかかわらず監査を実行します。

監査オプションの表

表 12-1 データベース・オブジェクトの文監査オプション

文オプション	SQL 文と操作
CLUSTER	CREATE CLUSTER
	AUDIT CLUSTER
	DROP CLUSTER
	TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT
	DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK
	DROP DATABASE LINK
DIMENSION	CREATE DIMENSION
	ALTER DIMENSION
	DROP DIMENSION
DIRECTORY	CREATE DIRECTORY
	DROP DIRECTORY
INDEX	CREATE INDEX
	ALTER INDEX
	DROP INDEX
NOT EXISTS	指定したオブジェクトが存在しない場合に失敗するすべての SQL 文

表 12-1 データベース・オブジェクトの文監査オプション（続き）

文オプション	SQL 文と操作
PROCEDURE ^a	CREATE FUNCTION
	CREATE LIBRARY
	CREATE PACKAGE
	CREATE PACKAGE BODY
	CREATE PROCEDURE
	DROP FUNCTION
	DROP LIBRARY
	DROP PACKAGE
	DROP PROCEDURE
PROFILE	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK
	DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM
	DROP PUBLIC SYNONYM
ROLE	CREATE ROLE
	ALTER ROLE
	DROP ROLE
	SET ROLE
ROLLBACK SEGMENT	CREATE ROLLBACK SEGMENT
	ALTER ROLLBACK SEGMENT
	DROP ROLLBACK SEGMENT
SEQUENCE	CREATE SEQUENCE
	DROP SEQUENCE
SESSION	Logons
SYNONYM	CREATE SYNONYM
	DROP SYNONYM
SYSTEM AUDIT	AUDIT <i>sql_statements</i>
	NOAUDIT <i>sql_statements</i>

表 12-1 データベース・オブジェクトの文監査オプション（続き）

文オプション	SQL 文と操作
SYSTEM GRANT	GRANT <i>system_privileges_and_roles</i> REVOKE <i>system_privileges_and_roles</i>
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE
TRIGGER	CREATE TRIGGER ALTER TRIGGER ENABLE および DISABLE 句付き DROP TRIGGER ALTER TABLE ENABLE ALL TRIGGERS 句付き DISABLE ALL TRIGGERS 句付き
TYPE	CREATE TYPE CREATE TYPE BODY ALTER TYPE DROP TYPE DROP TYPE BODY
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE VIEW DROP VIEW
^a Java スキーマ・オブジェクト（ソース、クラスおよびリソース）は、SQL 文の監査ではプロシージャと同じであるとみなされます。	

表 12-2 SQL 文のその他の文監査オプション

文オプション	SQL 文と操作
ALTER SEQUENCE	ALTER SEQUENCE
ALTER TABLE	ALTER TABLE
COMMENT TABLE	COMMENT ON TABLE <i>table</i> , <i>view</i> , <i>materialized view</i> COMMENT ON COLUMN <i>table.column</i> , <i>view.column</i> , <i>materialized view.column</i>
DELETE TABLE	DELETE FROM <i>table</i> , <i>view</i>
EXECUTE PROCEDURE	CALL プロシージャまたはファンクションの実行。または、変数、ライブラリ、パッケージ内のカーソルへのアクセス。
GRANT DIRECTORY	GRANT <i>privilege</i> ON <i>directory</i> REVOKE <i>privilege</i> ON <i>directory</i>
GRANT PROCEDURE	GRANT <i>privilege</i> ON <i>procedure</i> , <i>function</i> , <i>package</i> REVOKE <i>privilege</i> ON <i>procedure</i> , <i>function</i> , <i>package</i>
GRANT SEQUENCE	GRANT <i>privilege</i> ON <i>sequence</i> REVOKE <i>privilege</i> ON <i>sequence</i>
GRANT TABLE	GRANT <i>privilege</i> ON <i>table</i> , <i>view</i> , <i>materialized view</i> REVOKE <i>privilege</i> ON <i>table</i> , <i>view</i> , <i>materialized view</i>
GRANT TYPE	GRANT <i>privilege</i> ON TYPE REVOKE <i>privilege</i> ON TYPE
INSERT TABLE	INSERT INTO <i>table</i> , <i>view</i>
LOCK TABLE	LOCK TABLE <i>table</i> , <i>view</i>
SELECT SEQUENCE	<i>sequence.CURRVAL</i> または <i>sequence.NEXTVAL</i> を含む文
SELECT TABLE	SELECT FROM <i>table</i> , <i>view</i> , <i>materialized view</i>
UPDATE TABLE	UPDATE <i>table</i> , <i>view</i>

表 12-3 オブジェクト監査オプション

オブジェクト・ オプション	表	ビュー	順序	プロシージャ、 ファンクション、 パッケージ ^a	マテリア ライズド・ ビュー	ディレク トリ	ライ ブラリ	オブジェクト・ タイプ	コンテキ スト
ALTER	X	—	X	—	X	—	—	X	—
AUDIT	X	X	X	X	X	X	—	X	X
COMMENT	X	X	—	—	X	—	—	—	—
DELETE	X	X	—	—	X	—	—	—	—
EXECUTE	—	—	—	X	—	—	X	—	—
GRANT	X	X	X	X	—	X	X	X	X
INDEX	X	—	—	—	X	—	—	—	—
INSERT	X	X	—	—	X	—	—	—	—
LOCK	X	X	—	—	X	—	—	—	—
READ	—	—	—	—	—	X	—	—	—
RENAME	X	X	—	X	X	—	—	—	—
SELECT	X	X	X	—	X	—	—	—	—
UPDATE	X	X	—	—	X	—	—	—	—

^a Java スキーマ・オブジェクト（ソース、クラスおよびリソース）は、監査オプションではプロシージャ、ファンク
ションおよびパッケージと同じであるとみなされます。

例

ロールに関連する SQL 文の監査例： 次の文は、ロールの作成、変更、削除または設定を行
う各 SQL 文が正常に終了したかどうかにかかわらず、それらの文について監査を行います。

```
AUDIT ROLE;
```

次の文は、ロールの作成、変更、削除または設定を行う、正常に終了した各 SQL 文ごとに
監査を行います。

```
AUDIT ROLE  
  WHENEVER SUCCESSFUL;
```

次の文は、Oracle エラーが発生した CREATE ROLE 文、ALTER ROLE 文、DROP ROLE 文または SET ROLE 文について監査を行います。

```
AUDIT ROLE
    WHENEVER NOT SUCCESSFUL;
```

問合せおよび更新を行う SQL 文の監査例： 次の文は、表の問合せまたは更新を実行する文について監査を行います。

```
AUDIT SELECT TABLE, UPDATE TABLE;
```

次の文は、ユーザー hr および oe が発行する、表やビューの問合せまたは更新を実行する文について監査を行います。

```
AUDIT SELECT TABLE, UPDATE TABLE
    BY hr, oe;
```

監査の削除例： 次の文は、DELETE ANY TABLE システム権限で発行された文について監査を行います。

```
AUDIT DELETE ANY TABLE;
```

ディレクトリに関連する文の監査例： 次の文は、CREATE ANY DIRECTORY システム権限で発行された文について監査を行います。

```
AUDIT CREATE ANY DIRECTORY;
```

CREATE ANY DIRECTORY システム権限を使用しない CREATE DIRECTORY（および DROP DIRECTORY）文を監査する場合は、次の文を発行します。

```
AUDIT DIRECTORY;
```

次の文は、bfile_dir ディレクトリからファイルを読み込む各文について監査を行います。

```
AUDIT READ ON DIRECTORY bfile_dir;
```

表の問合せの監査例： 次の文は、スキーマ hr 内の employees 表を問い合わせる各 SQL 文について監査を行います。

```
AUDIT SELECT
    ON hr.employees;
```

次の文は、スキーマ hr 内の employees 表を問い合わせて正常に終了した各文について監査を行います。

```
AUDIT SELECT
    ON hr.employees
    WHENEVER SUCCESSFUL;
```

次の文は、スキーマ hr 内の employees 表を問い合わせたエラーが発生した SQL 文について監査を行います。

```
AUDIT SELECT
  ON hr.employees
  WHENEVER NOT SUCCESSFUL;
```

表の挿入および更新の監査例： 次の文は、スキーマ oe 内の customers 表に対して行を挿入または更新するそれぞれの文について監査を行います。

```
AUDIT INSERT, UPDATE
  ON oe.customers;
```

順序の操作の監査例： 次の文は、スキーマ hr 内の employees_seq 順序に対する操作を行うすべての文について監査を行います。

```
AUDIT ALL
  ON hr.employees_seq;
```

この文は、順序に対して操作を行う次の文について監査を行うため、ALL ショートカットを使用しています。

- ALTER SEQUENCE
- AUDIT
- GRANT
- 疑似列 CURRVAL または NEXTVAL を使用して、順序の値にアクセスするすべての文

デフォルト監査オプションの設定例： 次の文は、その後作成されるオブジェクトについて、デフォルト監査オプションを指定します。

```
AUDIT ALTER, GRANT, INSERT, UPDATE, DELETE
  ON DEFAULT;
```

その後作成されるオブジェクトについては、監査機能が使用可能な場合、指定したオプションによって自動的に次の監査が行われます。

- 表を作成した場合、その表に対して発行される ALTER 文、GRANT 文、INSERT 文、UPDATE 文または DELETE 文が自動的に監査されます。
- ビューを作成した場合、そのビューに対して発行される GRANT 文、INSERT 文、UPDATE 文または DELETE 文が自動的に監査されます。
- 順序を作成した場合、その順序に対して発行される ALTER 文または GRANT 文が自動的に監査されます。
- プロシージャ、パッケージまたはファンクションを作成した場合、それらに対して発行される ALTER 文または GRANT 文が自動的に監査されます。

CALL

用途

CALL 文を使用すると、SQL 内から**ルーチン**（スタンドアロン・プロシージャ、スタンドアロン・ファンクション、あるいは型またはパッケージ内で定義されたプロシージャまたはファンクション）を実行できます。

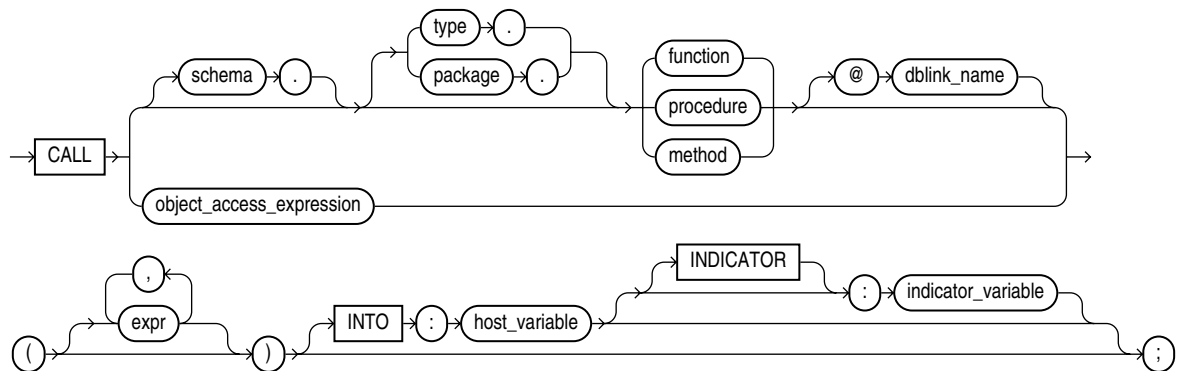
参照： これらのルーチンの作成の詳細は、『PL/SQL ユーザーズ・ガイド およびリファレンス』を参照してください。

前提条件

スタンドアロン・ルーチン、またはルーチンが定義されている型またはパッケージに対する EXECUTE 権限が必要です。

構文

call::=



セマンティクス

schema

スタンドアロン・ルーチン（または、ルーチンが含まれている型またはパッケージ）が存在するスキーマを指定します。**schema** を指定しない場合、ルーチンが自分のスキーマ内にあるとみなされます。

type または package

ルーチンが定義されている型またはパッケージを指定します。

function | procedure | method

コールするファンクション名またはプロシージャ名、またはファンクションまたはプロシージャに変換されるシノニムを指定します。

型のメンバーであるファンクションまたはプロシージャをコールする場合、最初の引数 (SELF) が NULL の IN OUT の場合、エラーが戻されます。SELF が NULL の IN の場合、NULL が戻されます。どちらの場合も、ファンクションまたはプロシージャはコールされません。

ファンクションの制限事項： ルーチンがファンクションの場合、INTO 句は必須です。

@dblink

分散データベース・システムで、スタンドアロン・ルーチンが含まれているデータベース (あるいは、ルーチンが含まれているパッケージまたはファンクション) の名前を指定します。dblink を指定しなかった場合、ローカル・データベースを指定したとみなされます。

参照： 直接ルーチンをコールする例については、12-65 ページの「[プロシージャのコール例:](#)」を参照してください。

object_access_expression

型コンストラクタ、バインド変数などのオブジェクト型の式を使用している場合は、*object_access_expression* 構文を使用して、型内で定義されたルーチンをコールできます。

このコンテキストで使用可能な構文は、オブジェクト・アクセス式のサブセットです (4-12 ページの「[オブジェクト・アクセス式](#)」を参照)。CALL 文で使用すると、メソッドのみコールできます (属性はコールできません)。また、*object_expr* はリテラルである必要があります (列名またはその他の非リテラルの式は使用できません)。したがって、使用可能な構文は次のとおりになります。

expression.method

expression はオブジェクト型のリテラル式で、*method* はオブジェクト型のメンバー・メソッドです。

参照： オブジェクト型の式を使用してルーチンをコールする例については、12-65 ページの「[オブジェクト型の式を使用したプロシージャのコール例:](#)」を参照してください。

argument

ルーチンに引数が必要な場合に、ルーチンの 1 つ以上の引数を指定します。

ルーチンへの引数の制限事項：

- 引数には、疑似列、オブジェクト参照ファンクション VALUE および REF は指定できません。
- ルーチンの IN OUT 引数または OUT 引数であるすべての引数は、ホスト変数の式に対応している必要があります。

INTO :host_variable

INTO 句は、ファンクションのコールにのみ適用されます。ファンクションの戻り値を格納するホスト変数を指定します。

:indicator_variable

ホスト変数の値または状態を指定します。

参照： ホスト変数およびインジケータ変数の詳細は、『Pro*C/C++ Precompiler プログラマーズ・ガイド』を参照してください。

例

プロシージャのコール例： 次の文は、remove_dept プロシージャ（14-55 ページの「[パッケージ本体の作成例:](#)」で作成）を使用して、娯楽部門（17-65 ページの「[順序値の挿入例:](#)」で作成）を削除します。

```
CALL remove_dept (162);
```

オブジェクト型の式を使用したプロシージャのコール例： 次の例では、CALL 文のオブジェクト型の式を使用したプロシージャのコール方法を示します。この例では、サンプルの注文入力スキーマ OE で warehouse_typ オブジェクト型を使用しています。

```
ALTER TYPE warehouse_typ
  ADD MEMBER FUNCTION ret_name
  RETURN VARCHAR2
  CASCADE;

CREATE OR REPLACE TYPE BODY warehouse_typ
  AS MEMBER FUNCTION ret_name
  RETURN VARCHAR2
  IS
  BEGIN
    RETURN self.warehouse_name;
  END;
```

```
        END;
/
VARIABLE x VARCHAR2(25);

CALL warehouse_typ(456, 'Warehouse 456', 2236).ret_name()
        INTO :x;

PRINT x;
X
-----
Warehouse 456
```

次の例では、外部関数を使用したプロシージャのコール方法を示します。

```
CREATE OR REPLACE FUNCTION ret_warehouse_typ(x warehouse_typ)
RETURN warehouse_typ
IS
    BEGIN
        RETURN x;
    END;
/
CALL ret_warehouse_typ(warehouse_typ(234, 'Warehouse 234',
        2235)).ret_name()
        INTO :x;

PRINT x;

X
-----
Warehouse 234
```

COMMENT

用途

COMMENT を使用すると、表、ビュー、マテリアライズド・ビューまたは列に関するコメントを、データ・ディクショナリに追加できます。

データ・ディクショナリ・ビューの USER_TAB_COMMENTS、DBA_TAB_COMMENTS、ALL_TAB_COMMENTS、USER_COL_COMMENTS、DBA_COL_COMMENTS または ALL_COL_COMMENTS を問い合わせることによって、特定の表または列に関するコメントを表示できます。

データベースからコメントを削除する場合、空の文字列 '' を設定します。

参照：

- SQL 文およびスキーマ・オブジェクトへのコメントの関連付けの詳細は、2-84 ページの「[コメント](#)」を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

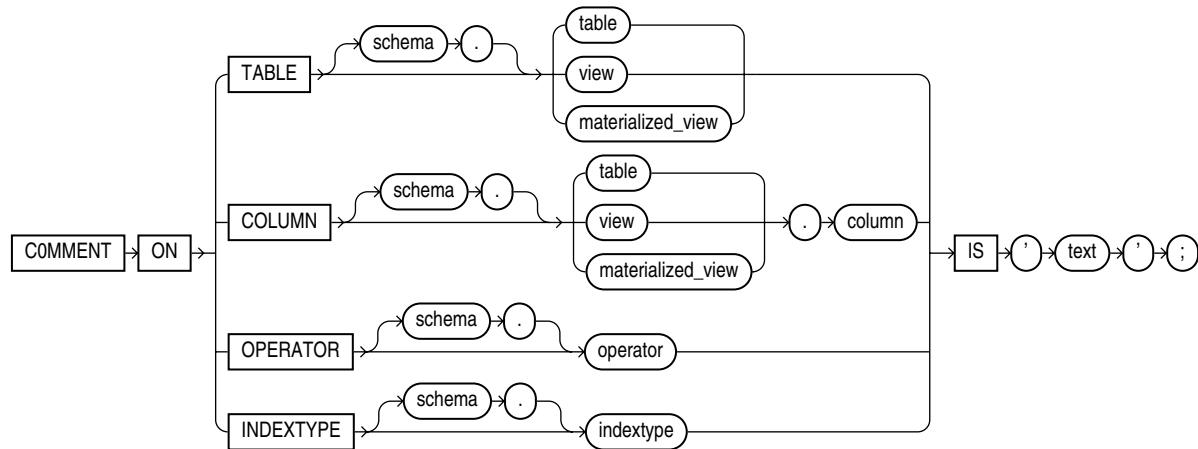
前提条件

コメントを追加するオブジェクトは自分のスキーマ内にあるか、次の条件を満たす必要があります。

- 表、ビューまたはマテリアライズド・ビューにコメントを追加する場合は、COMMENT ANY TABLE システム権限が必要です。
- 索引タイプにコメントを追加する場合は、COMMENT ANY INDEXTYPE システム権限が必要です。
- 演算子にコメントを追加する場合は、COMMENT ANY OPERATOR システム権限が必要です。

構文

comment::=



セマンティクス

TABLE 句

コメントする表、ビューまたはマテリアライズド・ビューの名前とスキーマです。 *schema* を指定しない場合、この表、ビューおよびマテリアライズド・ビューは自分のスキーマ内にあるとみなされます。

COLUMN 句

コメントする表、ビューまたはマテリアライズド・ビューの列の名前を指定します。 *schema* を指定しない場合、この表、ビューおよびマテリアライズド・ビューは自分のスキーマ内にあるとみなされます。

IS 'text'

コメントのテキストを指定します。

参照： 'text' の構文の詳細は、2-51 ページの「[テキスト・リテラル](#)」を参照してください。

OPERATOR 句

コメントする演算子の名前を指定します。 *schema* を指定しない場合、その演算子が自分のスキーマにあるとみなされます。

INDEXTYPE 句

コメントする索引タイプの名前を指定します。*schema* を指定しない場合、索引タイプが自分のスキーマ内に定義されているものとみなされます。

例

コメントの作成例： 次の文は、employees 表の job_id 列にコメントを挿入します。

```
COMMENT ON COLUMN employees.job_id  
    IS 'abbreviated job title';
```

次の文は、データベースからこのコメントを削除します。

```
COMMENT ON COLUMN employees.job_id IS ' ';
```

COMMIT

用途

COMMIT 文を使用すると、現行のトランザクションが終了し、トランザクションで実行したすべての変更が確定されます。**トランザクション**とは、Oracle が 1 つの単位として扱う一連の SQL 文です。また、この文によって、トランザクション内のセーブポイントがすべて消去され、トランザクションのロックが解除されます。

注意： Oracle では、データ定義言語（DDL）文の前後で暗黙的に COMMIT が発行されます。

この文を使用して、次のことができます。

- インダウト分散トランザクションを手動でコミットします。
- SET TRANSACTION 文で始まる読み込み専用トランザクションを終了します。

Oracle との接続を切断する前に、最新のトランザクションを含む、アプリケーション・プログラムのすべてのトランザクションを、COMMIT 文または ROLLBACK 文を使用して明示的に終了してください。トランザクションを明示的にコミットしなかった場合にプログラムが異常終了すると、コミットされていない最後のトランザクションは、自動的にロールバックされます。

Oracle ユーティリティおよび Oracle のツール製品が正常に終了すると、現行のトランザクションがコミットされます。Oracle プリコンパイラ・プログラムが正常に終了した場合は、トランザクションはコミットされず、現行のトランザクションは Oracle によってロールバックされます。

参照：

- トランザクションの詳細は、『Oracle9i データベース概要』を参照してください。
- トランザクションの特性の指定については、18-47 ページの「[SET TRANSACTION](#)」を参照してください。

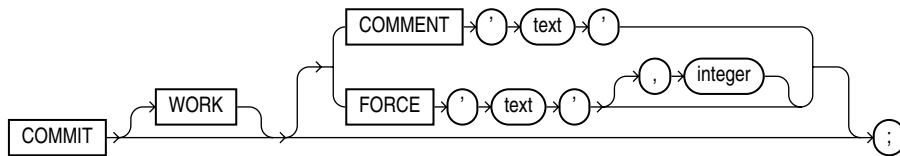
前提条件

現行のトランザクションをコミットするために、必要な権限は特にありません。

自分がコミットしたインダウト分散トランザクションを手動でコミットする場合は、FORCE TRANSACTION システム権限が必要です。別のユーザーがコミットしたインダウト分散トランザクションを手動でコミットする場合は、FORCE ANY TRANSACTION システム権限が必要です。

構文

commit::=



セマンティクス

WORK

標準 SQL に準拠するために、WORK キーワードがサポートされています。COMMIT 文と COMMIT WORK 文は同じです。

COMMENT 句

現行のトランザクションに関するコメントを指定します。'text' は引用符で囲まれた最大 255 文字のリテラルで、トランザクションの状態が不明（インダウト）になった場合に、そのトランザクション ID とともに、データ・ディクショナリ・ビュー DBA_2PC_PENDING に格納されます。

参照： SQL 文へのコメントの追加については、12-67 ページの「[COMMENT](#)」を参照してください。

FORCE 句

分散データベース・システムでは、FORCE 句によって手動でインダウト分散トランザクションをコミットできます。このトランザクションは、ローカル・トランザクション ID またはグローバル・トランザクション ID を含む 'text' で識別されます。このトランザクションの ID を確認する場合、データ・ディクショナリ・ビュー DBA_2PC_PENDING を問い合わせます。また、integer を指定することによって、このトランザクションに SCN を具体的に割り当てることができます。integer を指定しない場合、このトランザクションは現行の SCN を使用してコミットされます。

注意： FORCE 句を指定して COMMIT 文を発行した場合、指定したトランザクションのみがコミットされます。この文は、現行のトランザクションには影響しません。

FORCE の制限事項： PL/SQL では、FORCE 句を指定する COMMIT 文はサポートされていません。

例

挿入のコミット例： 次の文は、hr.regions 表に行を挿入してこの変更をコミットします。

```
INSERT INTO regions VALUES (5, 'Antarctica');
```

```
COMMIT WORK;
```

COMMIT のコメント例： 次の文は、現行のトランザクションをコミットして、コメントを対応付けます。

```
COMMIT  
  COMMENT 'In-doubt transaction Code 36, Call (415) 555-2637';
```

ネットワーク障害またはマシン障害によって分散トランザクションを適切にコミットできない場合、トランザクション ID とともにデータ・ディクショナリにコメントが格納されます。そのコメントには、障害が発生したアプリケーション部分が示されており、トランザクションがコミットされたデータベースの管理者に連絡する情報が提供されています。

強制インダウト・トランザクションの例： 次の文は、インダウト分散トランザクションを手動でコミットします。

```
COMMIT FORCE '22.57.53';
```

SQL 文 : CREATE CLUSTER ~ CREATE JAVA

この章では、次の SQL 文について説明します。

- CREATE CLUSTER
- CREATE CONTEXT
- CREATE CONTROLFILE
- CREATE DATABASE
- CREATE DATABASE LINK
- CREATE DIMENSION
- CREATE DIRECTORY
- CREATE FUNCTION
- CREATE INDEX
- CREATE INDEXTYPE
- CREATE JAVA

CREATE CLUSTER

用途

CREATE CLUSTER 文を使用すると、クラスタを作成できます。**クラスタ**とは、1 つ以上の列を共有する、1 つ以上の表のデータで構成されるスキーマ・オブジェクトです。同一のクラスタ・キーを共有する（すべての表の）すべての行がまとめて格納されます。

既存のクラスタの詳細は、データ・ディクショナリ USER_CLUSTERS、ALL_CLUSTERS および DBA_CLUSTERS に問い合わせます。

参照：

- クラスタの概要については、『Oracle9i データベース概要』を参照してください。
- クラスタのパフォーマンスについては、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- クラスタの使用方法については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

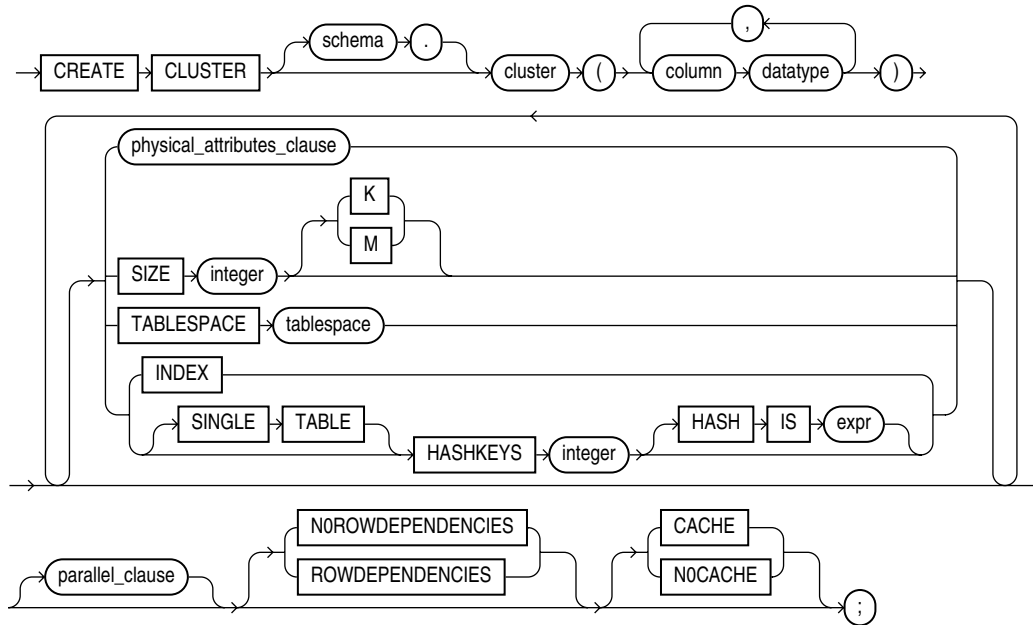
前提条件

自分のスキーマにクラスタを作成する場合は、CREATE CLUSTER システム権限が必要です。他のユーザーのスキーマ内にクラスタを作成する場合は、CREATE ANY CLUSTER システム権限が必要です。また、クラスタを設定するスキーマの所有者は、クラスタが定義されている表領域に対する領域の割当て制限、または UNLIMITED TABLESPACE システム権限のいずれかが必要です。

Oracle では、クラスタを最初に作成する場合は、クラスタに対する索引は自動的に作成されません。このため、クラスタ索引を作成するまでは、クラスタ化表に対して、データ操作言語（DML）文を発行できません。

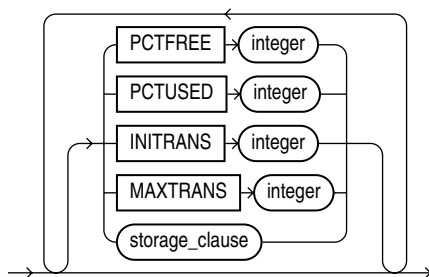
構文

create_cluster::=



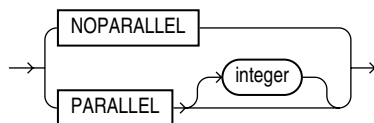
(13-3 ページの「[physical_attributes_clause::=](#)」を参照)

physical_attributes_clause::=



(7-51 ページの「[storage_clause](#)」を参照)

parallel_clause::=



セマンティクス

schema

作成するクラスタが定義されるスキーマを指定します。schema を指定しない場合、現行のスキーマにクラスタが作成されます。

cluster

作成するクラスタの名前を指定します。

クラスタを作成した後、そのクラスタに表を追加します。クラスタには、最大 32 個の表を指定できます。クラスタを作成し、そのクラスタに表を追加しても、クラスタを意識する必要はありません。クラスタ化されていない表と同様に、SQL 文を使用してクラスタ化表にアクセスできます。

参照： クラスタへの表の追加については、15-7 ページの「[CREATE TABLE](#)」、13-9 ページの「[クラスタの作成例 :](#)」および 13-9 ページの「[クラスタへの表の追加例 :](#)」を参照してください。

column

クラスタ・キーに 1 つ以上の列名を指定します。最大 16 個までクラスタ・キー列を指定できます。これらの列は、データ型およびサイズについて、クラスタ化表の列と対応している必要があります。名前は対応している必要はありません。

クラスタ・キー列の定義の一部として整合性制約は指定できません。そのかわり、クラスタに属している表に整合性制約を対応付けることができます。

参照： 13-9 ページの「[クラスタ・キーの例 :](#)」を参照してください。

datatype

各クラスタ・キー列のデータ型を指定します。

クラスタのデータ型の制限事項：

- データ型が LONG、LONG RAW、REF、ネストした表、VARRAY、BLOB、CLOB、BFILE またはユーザー定義オブジェクト型であるクラスタ・キー列は指定できません。

- 列のデータ型が、位取りが 0 である INTEGER 型や NUMBER 型でない場合、HASH IS 句は使用できません。
- ROWID 型の列を指定することはできますが、それらの列の値が有効な行 ID であることは保証されません。

参照： データ型の詳細は、2-2 ページの「[データ型](#)」を参照してください。

physical_attributes_clause

physical_attributes_clause によって、クラスタの記憶特性を指定できます。クラスタ内の各表もこれらの記憶特性を使用します。これらのパラメータの値を指定しない場合、次のデフォルトが使用されます。

- PCTFREE: 10
- PCTUSED: 40
- INITTRANS: 2 またはクラスタの表領域のデフォルト値が大きい方
- MAXTRANS: クラスタの表領域のデフォルト値

参照：

- *physical_attributes_clause* のパラメータの詳細は 7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- デフォルト値を含む *storage_clause* の詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

SIZE

同一クラスタ・キー値または同一ハッシュ値を持つすべての行を格納するために確保する領域を、バイト単位で指定します。K または M を使用すると、この領域を KB または MB 単位で指定できます。次に、この領域によって、データ・ブロックごとに格納されるクラスタやハッシュ値の最大値が決まります。SIZE の値がデータ・ブロック・サイズの約数でない場合、Oracle は、次に大きな約数を使用します。SIZE がデータ・ブロック・サイズより大きい場合、Oracle は、クラスタまたはハッシュ値ごとに、1 つ以上のデータ・ブロックを確保し、オペレーティング・システムのブロック・サイズを採用します。

Oracle は、クラスタ・キー値を持つ行に対して確保する必要がある領域を決定する際に、クラスタ・キーの長さを考慮します。クラスタ・キーが大きければ、それに必要なサイズも大きくなります。実際のサイズを参照するには、USER_CLUSTERS データ・ディクショナリ・ビューの KEY_SIZE 列を問い合わせます（なお、ハッシュ値は実際にはクラスタ内に格納されていないため、この方法はハッシュ・クラスタには適用されません）。

このパラメータを指定しない場合、各クラスタ・キー値またはハッシュ値ごとにデータ・ブロックが 1 つ確保されます。

TABLESPACE

クラスタを作成する表領域を指定します。

INDEX 句

INDEX を指定して、索引クラスタを作成します。索引クラスタには、同一のクラスタ・キー値が指定されている行がまとめて格納されます。それぞれのクラスタ・キー値は、そのキーを持つ表および行の数に関係なく、各データ・ブロックに 1 回のみ格納されます。

索引クラスタの作成後、クラスタ内の表に対してデータ操作言語（DML）文を発行する前に、そのクラスタ・キーに索引を付ける必要があります。この索引をクラスタ索引と呼びます。

注意： ハッシュ・クラスタに対してクラスタ索引は作成できないため、ハッシュ・クラスタ・キーで索引を作成する必要はありません。INDEX も HASHKEYS も指定しない場合は、デフォルトで索引クラスタが作成されます。

参照： クラスタ索引の作成方法の詳細は 13-61 ページの「[CREATE INDEX](#)」、索引クラスタの概要は『Oracle9i データベース概要』を参照してください。

HASHKEYS 句

HASHKEYS を指定して、ハッシュ・クラスタの作成およびハッシュ・クラスタのハッシュ値の数を指定します。ハッシュ・クラスタには、同一のハッシュ・キー値を持つ行がまとめて格納されます。それぞれの行のハッシュ値は、そのクラスタのハッシュ・ファンクションが戻す値です。

Oracle は、ハッシュ値の実際の数を決めるため、HASHKEYS 値を一番近い次の素数に切り上げます。このパラメータの最小値は 2 です。INDEX 句と HASHKEYS パラメータの両方を指定しない場合、デフォルトで索引クラスタが作成されます。

ハッシュ・クラスタの作成時に、Oracle は、SIZE パラメータおよび HASHKEYS パラメータの値に基づいて、クラスタに領域を割り当てます。

参照： Oracle がクラスタに領域を割り当てる方法の詳細は、『Oracle9i データベース概要』および 13-10 ページの「[ハッシュ・クラスタの例:](#)」を参照してください。

SINGLE TABLE SINGLE TABLE は、クラスタを、表を 1 つのみ持つタイプのハッシュ・クラスタに指定します。この句によって、表がクラスタの一部でない場合より行へのアクセスが高速になります。

単一表クラスタの制限事項： 同時にクラスタに存在できる表は1つのみです。ただし、表を削除して、同一のクラスタに別の表を作成することはできます。

参照： 13-10 ページの「[単一表ハッシュ・クラスタの例：](#)」を参照してください。

HASH IS *expr* ハッシュ・クラスタに対するハッシュ・ファンクションとして使用する式を指定します。式は、次の条件を満たす必要があります。

- 正の値に評価される必要があります。
- 式全体の値が位取り 0（ゼロ）の数になる場合は、任意のデータ型の列が参照される 1 つ以上の列を持ちます。たとえば、NUM_COLUMN * length(VARCHAR2_COLUMN) です。
- ユーザー定義の PL/SQL ファンクションを参照できません。
- 擬似列 LEVEL または ROWNUM を参照できません。
- ユーザー関連ファンクション（USERENV、UID および USER）または日時ファンクション（CURRENT_DATE、CURRENT_TIMESTAMP、DBTIMEZONE、EXTRACT（日時）、FROM_TZ、LOCALTIMESTAMP、NUMTODSINTERVAL、NUMTOYMINTERVAL、SESSIONTIMEZONE、SYSDATE、SYSTIMESTAMP、TO_DSINTERVAL、TO_TIMESTAMP、TO_DATE、TO_TIMESTAMP_TZ、TO_YMINTERVAL および TZ_OFFSET）を参照できません。
- 定数に評価されることはありません。
- スカラー副問合せ式には指定できません。
- （クラスタ名ではなく）スキーマ名またはオブジェクト名で修飾された列を持つことはできません。

HASH IS 句を指定しない場合、ハッシュ・クラスタに対して内部ハッシュ・ファンクションが使用されます。

既存のクラスタの詳細は、データ・ディクショナリ表

USER_CLUSTER_HASH_EXPRESSIONS、ALL_CLUSTER_HASH_EXPRESSIONS および DBA_CLUSTER_HASH_EXPRESSIONS に問い合わせます。

ハッシュ列のクラスタ・キーは、任意のデータ型で構成される 1 つ以上の列を持つことができます。複合クラスタ・キー、または整数以外の列で構成されるクラスタ・キーを持つハッシュ・クラスタに対しては、内部ハッシュ・ファンクションを使用する必要があります。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

parallel_clause

parallel_clause によって、クラスタ作成をパラレル化します。

注意： `parallel_clause` 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer integer には、パラレル操作で使用するパラレル・スレッド数である並列度を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、integer に値を指定する必要はありません。

クラスタ作成のパラレル化の制限事項： クラスタの表が LOB 型またはユーザー定義オブジェクト型の列を含む場合、クラスタでその後に行う INSERT、UPDATE または DELETE 操作と同様、この文は通知なしに逐次実行されます。

参照： 詳細は、15-51 ページの CREATE TABLE の「[parallel_clause の注意事項](#)」を参照してください。

NOROWDEPENDENCIES | ROWDEPENDENCIES

クラスタが**行レベル依存の追跡**を使用するかどうかを指定できます。この機能を使用すると、クラスタを構成する表の各行は、行を変更した最後のトランザクションのコミット時刻以降を示す SCN を持つことになります。この設定は、クラスタの作成後に変更できません。

ROWDEPENDENCIES 行レベル依存の追跡を使用可能にする場合は、ROWDEPENDENCIES を指定します。この設定は、主にレプリケーション環境でパラレル伝播を許可する場合に便利です。各行につきサイズが 6 バイト増えます。

NOROWDEPENDENCIES 行レベル依存の追跡機能を使用しない場合は、NOROWDEPENDENCIES を指定します。これはデフォルト値です。

参照： レプリケーション環境での行レベル依存の追跡の使用については、『Oracle9i アドバンスド・レプリケーション』を参照してください。

CACHE | NOCACHE

CACHE 全表スキンの実行時に、このクラスタに対して取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最高使用頻度側に入れる場合は、CACHE を指定します。この句は、小規模な参照表で有効です。

NOCACHE 全表スキンの実行時に、このクラスタに対して取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最低使用頻度側に入れる場合は、NOCACHE を指定します。これはデフォルトの動作です。

注意： NOCACHE は、*storage_clause* に KEEP を指定したクラスタには影響しません。

例

クラスタの作成例： 次の文は、クラスタ・キー列 department、クラスタ・サイズ 512 バイトおよび記憶域パラメータ値を指定した索引クラスタ personnel を作成します。

```
CREATE CLUSTER personnel
  (department NUMBER(4))
SIZE 512
STORAGE (initial 100K next 50K);
```

クラスタ・キーの例： 次の文は、personnel のクラスタ・キーにクラスタ索引を作成します。

```
CREATE INDEX idx_personnel ON CLUSTER personnel;
```

クラスタ索引の作成後、索引に表を追加し、その表に対して DML 操作を行うことができます。

クラスタへの表の追加例： 次の文は、サンプル表 hr.employees から部門表を作成し、前述の例で作成した personnel クラスタに追加します。

```
CREATE TABLE dept_10
  CLUSTER personnel (department_id)
  AS SELECT * FROM employees WHERE department_id = 10;

CREATE TABLE dept_20
  CLUSTER personnel (department_id)
  AS SELECT * FROM employees WHERE department_id = 20;
```

ハッシュ・クラスタの例： 次の文は、クラスタ・キー列 `cust_language`、最大ハッシュ・キー値 10（各サイズ 512 バイト）および記憶域パラメータ値を指定したハッシュ・クラスタ `language` を作成します。

```
CREATE CLUSTER language (cust_language VARCHAR2(3))
    SIZE 512 HASHKEYS 10
    STORAGE (INITIAL 100k next 50k);
```

この文では、`HASH IS` 句を指定していないため、Oracle は、そのクラスタに対して内部ハッシュ・ファンクションを採用します。

次の文は、`postal_code` と `country_id` 列で構成されるクラスタ・キーを持つ `address` という名前のハッシュ・クラスタを作成し、これらの列を含む SQL 式をハッシュ・ファンクションに使用します。

```
CREATE CLUSTER address
    (postal_code NUMBER, country_id CHAR(2))
    HASHKEYS 20
    HASH IS MOD(postal_code + country_id, 101);
```

単一表ハッシュ・クラスタの例： 次の文は、クラスタ・キー `customer_id` および最大ハッシュ・キー値 100（各サイズ 512 バイト）を指定した単一表ハッシュ・クラスタ `cust_orders` を作成します。

```
CREATE CLUSTER cust_orders (customer_id NUMBER(6))
    SIZE 512 SINGLE TABLE HASHKEYS 100;
```


CREATE CONTEXT

用途

次の場合に CREATE CONTEXT 文を使用します。

- コンテキスト（アプリケーションを検証および保護するアプリケーション定義の属性）のネームスペースを作成します。
- ネームスペースを、コンテキストを設定する外部作成パッケージと対応付けます。

指定されたパッケージの DBMS_SESSION.set_context プロシージャを使用して、コンテキスト属性を設定または再設定できます。

参照：

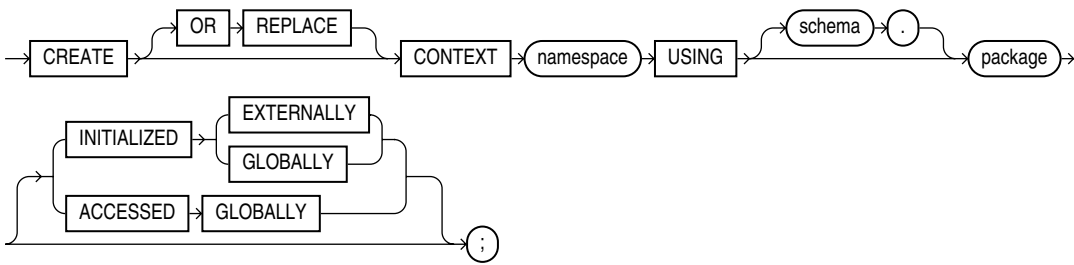
- コンテキストの定義および説明については、『Oracle9i データベース概要』を参照してください。
- DBMS_SESSION.set_context プロシージャについては、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

前提条件

コンテキスト・ネームスペースを作成する場合、CREATE ANY CONTEXT システム権限が必要です。

構文

create_context::=



セマンティクス

OR REPLACE

OR REPLACE を指定して、異なるパッケージを使用して既存コンテキストを再定義します。

namespace

作成または変更するためのコンテキスト・ネームスペース名を指定します。コンテキスト・ネームスペースは、スキーマ SYS に格納されます。

schema

package を所有するスキーマを指定します。schema を指定しない場合、Oracle は現在のスキーマを使用します。

package

ユーザー・セッションのネームスペースに基づくコンテキスト属性を設定または再設定する PL/SQL パッケージを指定します。

注意： 柔軟に設計できるように、Oracle は、コンテキスト作成時のスキーマの存在またはパッケージの妥当性を検証しません。

参照： パッケージの設定の詳細は、『Oracle9i PL/SQL パッケージ・プログラミングおよびタイプ・リファレンス』を参照してください。

INITIALIZED 句

INITIALIZED 句を使用すると、コンテキスト・ネームスペースを初期化する Oracle 以外のエンティティを指定することができます。

EXTERNALLY EXTERNALLY を使用すると、セッション確立時に OCI インタフェースを使用してネームスペースが初期化されます。

参照： セッション確立の OCI 使用の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

GLOBALLY GLOBALLY を指定すると、グローバル・ユーザーがデータベースへ接続するときに、LDAP ディレクトリによってネームスペースが初期化されます。

セッション確立後、設定した PL/SQL パッケージのみがネームスペースのすべての属性に書込みを行うコマンドを発行することができます。

参照：

- グローバルに初期化されるコンテキストの確立の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- LDAP ディレクトリを介したデータベースへの接続の詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

ACCESSED GLOBALLY

この句を使用すると、*namespace* に設定したすべてのアプリケーション・コンテキストはすべてのインスタンスからアクセスできます。この設定によって、複数のセッションがアプリケーション属性を共有できます。

例

アプリケーション・コンテキストの作成例： この例は、hr アプリケーション（14-51 ページの「[パッケージの作成例](#)」で作成）を検証および保護する PL/SQL パッケージ emp_mgmt を使用します。次の文は、コンテキスト・ネームスペース hr_context を作成し、emp_mgmt パッケージに関連付けます。

```
CREATE CONTEXT hr_context USING emp_mgmt;
```

SYS_CONTEXT ファンクションを使用するこのコンテキストに基づいて、データ・アクセスを制御できます。たとえば、emp_mgmt パッケージが、特定の従業員識別子として、属性 new_empno を定義するものとします。new_empno の値に基づき、アクセスを制限するビューを作成して employees 実表を次のように保護できます。

```
CREATE VIEW hr_org_secure_view AS
  SELECT * FROM employees
  WHERE employee_id = SYS_CONTEXT('hr_context', 'new_empno');
```

参照： 6-148 ページの「[SYS_CONTEXT](#)」を参照してください。

CREATE CONTROLFILE

注意： この文を使用する前に、データベース内のすべてのファイルの全体バックアップを実行することをお勧めします。詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

用途

CREATE CONTROLFILE 文は、次の場合に制御ファイルを再作成します。

- 既存の制御ファイルのすべてのコピーが、メディア障害により消失してしまった場合。
- データベース名を変更する場合。
- REDO ログ・ファイル・グループ、REDO ログ・ファイル・メンバー、アーカイブ REDO ログ・ファイル、データ・ファイル、またはデータベースを同時にマウントおよびオープンするインスタンスの最大数を変更する場合。

注意： CREATE CONTROLFILE 文を使用する必要がある場合は、DATAFILE 句に一時表領域または読み専用表領域のデータ・ファイルを含めないでください。これらのタイプのファイルは、後でデータベースに追加できます。

CREATE CONTROLFILE 文のかわりに、ALTER DATABASE BACKUP CONTROLFILE TO TRACE を使用できます。この文は、トレース・ファイルに SQL スクリプトを生成して、制御ファイルを再作成します。データベースに読み専用表領域または一時表領域が含まれている場合は、この SQL スクリプトに、データベースにこれらのファイルを追加するために必要なすべての SQL 文も含まれます。

参照： 既存のデータベース制御ファイルに基づくスクリプトの作成の詳細は、9-43 ページの ALTER DATABASE の「[BACKUP CONTROLFILE 句](#)」を参照してください。

CREATE CONTROLFILE 文を発行した場合、この文にユーザーが指定した情報に基づいて、新しい制御ファイルが作成されます。句を指定しない場合、Oracle は、以前の制御ファイルに対する値ではなく、デフォルト値を使用します。制御ファイルが正常に作成された後、初期化パラメータ CLUSTER_DATABASE で指定したモードでデータベースがマウントされます。データベースをオープンする前に、メディア・リカバリを行う必要があります。その後、インスタンスを停止し、データベースのすべてのファイルの全体バックアップを取ることをお勧めします。

参照：『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

前提条件

制御ファイルを作成する場合は、SYSDBA システム権限が必要です。

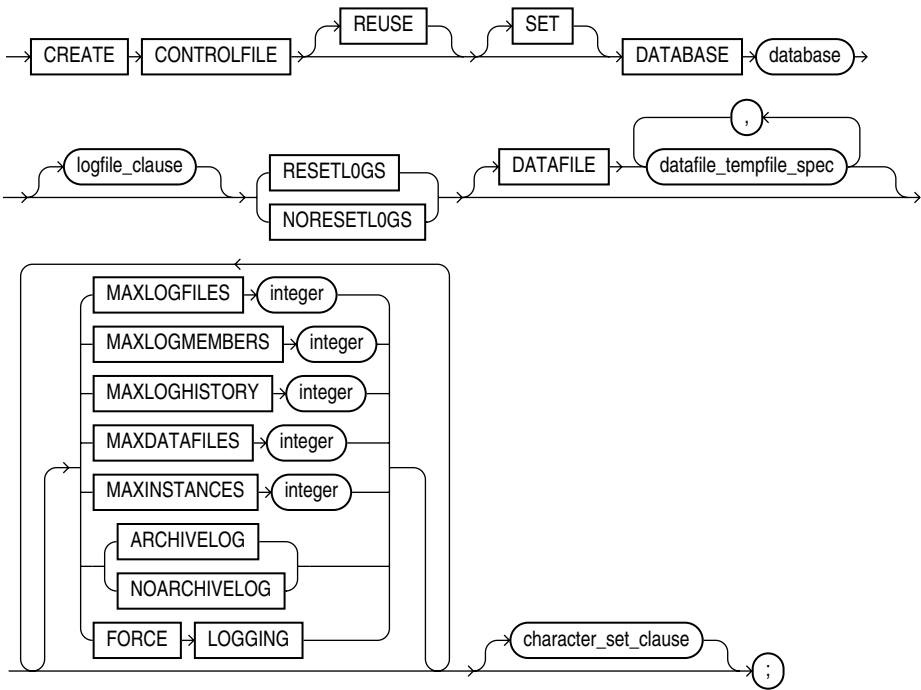
データベースをマウントしているインスタンスがあってはけません。制御ファイルが正常に作成された後、データベースは EXCLUSIVE 状態にマウントされたままで残されます。Oracle Real Application Clusters を使用している場合、DBA は次のインスタンスが起動する前に停止し、データベースを SHARED モード（CLUSTER_DATABASE 初期化パラメータの値が TRUE である場合のデフォルト）で再マウントする必要があります。

初期化パラメータ REMOTE_LOGIN_PASSWORDFILE が EXCLUSIVE に設定されている場合、Oracle は、制御ファイルを再作成しようとした際にエラーを戻します。このメッセージを回避するには、制御ファイルを再作成する前に、パラメータに SHARED を設定するか、またはパスワード・ファイルを再作成します。

参照： REMOTE_LOGIN_PASSWORDFILE パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

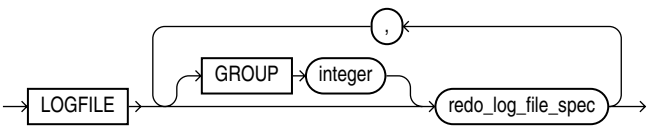
構文

create_controlfile::=



(7-36 ページの「`datafile_tempfile_spec::=`」(`file_specification` 構文の一部)を参照)

logfile_clause::=



(7-37 ページの「`redo_log_file_spec::=`」(`file_specification` 構文の一部)を参照)

character_set_clause::=



セマンティクス

REUSE

REUSE を指定すると、初期化パラメータ CONTROL_FILES によって特定される既存の制御ファイルを再利用することを指定できます。このとき、この初期化パラメータに現在格納されている情報は無視され、上書きされます。この句を指定しないと、既存の制御ファイルがある場合に、エラーが戻ります。

DATABASE 句

データベース名を指定します。このパラメータの値は、事前に CREATE DATABASE 文または CREATE CONTROLFILE 文で設定した既存のデータベース名である必要があります。

SET DATABASE 句

SET DATABASE を使用すると、データベース名を変更できます。データベース名は最大 8 バイトまで指定可能です。

logfile_clause

logfile_clause を使用すると、データベースの REDO ログ・ファイルを指定できます。すべての REDO ログ・ファイル・グループのすべてのメンバーを指定する必要があります。

GROUP integer ログ・ファイル・グループ番号を指定します。GROUP 値を指定した場合、データベースが前回オープンされたときの GROUP 値を基にして、この値が検証されます。

この句を省略した場合、システムのデフォルト値を使用してログ・ファイルが作成されます。また、DB_CREATE_ONLINE_LOG_DEST_n 初期化パラメータと DB_CREATE_FILE_DEST 初期化パラメータのいずれか（またはその両方）を設定した場合、および RESETLOGS を指定した場合は、DB_CREATE_ONLINE_LOG_DEST_n パラメータで指定したログ・ファイルのデフォルトの格納先に 2 つのログ・ファイルが作成されます。また、DB_CREATE_ONLINE_LOG_DEST_n が設定されていない場合は、DB_CREATE_FILE_DEST パラメータで指定した格納先に作成されます。

参照： この句の詳細は、7-36 ページの「[file_specification](#)」を参照してください。

RESETLOGS LOGFILE 句にリストされたファイルのコンテキストを無視する場合は、RESETLOGS を指定します。LOGFILE 句に指定したファイルは、存在していなくてもかまいません。LOGFILE 句の各 redo_log_file_spec で、SIZE パラメータを指定する必要があります。Oracle では、スレッド 1 にすべてのオンライン REDO ログ・ファイル・グループを割り当てることによって、このスレッドを任意のインスタンスで共通に使用できるようにします。この句を使用した後は、RESETLOGS 句を指定した ALTER DATABASE を使用してデータベースをオープンする必要があります。

NORESETLOGS LOGFILE 句に指定したすべてのファイルを、前回データベースをオープンしたときの状態で使用する場合は、NORESETLOGS を指定します。LOGFILE 句に指定したファイルは、存在する必要があります。また、バックアップからのリストアではなく、現行のオンライン REDO ログ・ファイルである必要があります。前回割り当てたスレッドに、REDO ログ・ファイル・グループが再度割り当てられ、そのスレッドが前回と同じく再度使用可能になります。

DATAFILE 句

データベースのデータ・ファイルを指定します。すべてのデータ・ファイルを指定する必要があります。これらのファイルは既存のファイルである必要がありますが、メディア・リカバリを必要とするリストア・バックアップでもかまいません。構文の詳細は、7-36 ページの「[file_specification](#)」を参照してください。

注意： この句では、一時データ・ファイル（一時ファイル）ではなく、データ・ファイルのみを指定します。一時ファイルの処理の詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

DATAFILE の制限事項： この DATAFILE 句では、`data_file_spec` の `autoextend_clause` は指定できません。

MAXLOGFILES 句

データベースに対して作成可能なオンライン REDO ログ・ファイル・グループの最大数を指定します。この値を基にして、制御ファイル内で REDO ログ・ファイル名に割り当てられる領域が決定されます。デフォルト値や最大値は、使用するオペレーティング・システムによって異なります。指定する値は、すべての REDO ログ・ファイル・グループの GROUP の最大値以上である必要があります。

MAXLOGMEMBERS 句

REDO ログ・ファイル・グループのメンバー（同一コピー）の最大数を指定します。この値を基にして、制御ファイル内で REDO ログ・ファイル名に割り当てられる領域が決定されます。最小値は 1 です。最大値およびデフォルト値は、使用するオペレーティング・システムによって異なります。

MAXLOGHISTORY 句

このパラメータは、Real Application Clusters 環境で ARCHIVELOG モードの Oracle を使用している場合に便利です。Real Application Clusters の自動メディア・リカバリに使用するアーカイブ REDO ログ・ファイル・グループの最大数を指定します。この値を基にして、制御ファイル内でアーカイブ REDO ログ・ファイル名に割り当てられる領域が決定されます。最小値は 0（ゼロ）です。デフォルト値は MAXINSTANCES 値の倍数で、使用するオペレーティング・システムによって異なります。最大値は、制御ファイルの最大サイズの制限のみを受けます。

MAXDATAFILES 句

CREATE DATABASE または CREATE CONTROLFILE 実行時の、制御ファイルのデータ・ファイル・セクションの初期サイズ設定を指定します。値が MAXDATAFILES より大きく、DB_FILES 以下のファイルを追加した場合、データ・ファイル・セクションにさらに多くのファイルを格納できるように、Oracle の制御ファイルが自動的に拡張されます。

インスタンスでアクセスできるデータ・ファイルの数は、初期化パラメータ DB_FILES の制限を受けます。

MAXINSTANCES 句

データベースを同時にマウントおよびオープンできるインスタンスの最大数を指定します。この値は、初期化パラメータ INSTANCES の値より優先されます。最小値は 1 です。最大値およびデフォルト値は、使用するオペレーティング・システムによって異なります。

ARCHIVELOG | NOARCHIVELOG

ARCHIVELOG を指定して、REDO ログ・ファイルを再利用する前に、ファイルの内容をアーカイブします。この句を指定した場合、インスタンスまたはシステム障害リカバリのみでなく、メディア・リカバリもできるようになります。

ARCHIVELOG 句および NOARCHIVELOG 句を省略した場合、デフォルトでは NOARCHIVELOG モードが選択されます。制御ファイルの作成後に、ALTER DATABASE 文を使用した場合、ARCHIVELOG モードと NOARCHIVELOG モードを切り替えることができます。

FORCE LOGGING

この句を使用すると、制御ファイルの作成後にデータベースは FORCE LOGGING モードになります。データベースがこのモードで実行されている場合は、一時表領域および一時セグメントへの変更以外のすべてのデータベース内の変更のログが記録されます。この設定は、各表領域で指定する NOLOGGING または FORCE LOGGING 設定、および各データベース・オブジェクトで指定する NOLOGGING 設定より優先され、これらの設定には影響されません。この句を省略すると、制御ファイルの作成後にデータベースは FORCE LOGGING モードになりません。

注意： FORCE LOGGING モードは、パフォーマンスに影響する場合があります。この設定の使用については、『Oracle9i データベース管理者ガイド』を参照してください。

character_set_clause

キャラクタ・セットを指定した場合、制御ファイルのキャラクタ・セット情報を再構成します。データベースのケース・メディア・リカバリが必要となる場合、データベースがオープンする前にこの情報が使用可能になり、リカバリ時に表領域名が正しく解析されます。この句は、デフォルトの US7ASCII 以外のキャラクタ・セットを使用している場合にのみ必要です。Oracle は、起動時に \$ORACLE_HOME/rdbms/log のアラート・ログに、現行のデータベース・キャラクタ・セットを出力します。

制御ファイルを再作成し、表領域のリカバリに Recovery Manager を使用している場合、およびデータ・ディクショナリに格納されている異なるキャラクタ・セットを指定する場合、表領域はリカバリされません（ただし、データベースのオープン時、制御ファイルのキャラクタ・セットは、データ・ディクショナリの正しいキャラクタ・セットに更新されます）。

注意： この句ではデータベース・キャラクタ・セットを変更できません。

参照： 表領域のリカバリの詳細は、『Oracle9i Recovery Manager ユーザーズ・ガイド』を参照してください。

例

制御ファイルの作成例： この文は、制御ファイルを再作成します。この文によって、データベース demo は WE8DEC キャラクタ・セットで作成されます。この例で使用する単語 *path* には、ご使用のシステムでの適切な Oracle ディレクトリへのパスを挿入してください。

STARTUP NOMOUNT

```
CREATE CONTROLFILE REUSE DATABASE "demo" NORESETLOGS NOARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 449
LOGFILE
    GROUP 1 '/path/oracle/dbs/t_log1.f' SIZE 500K,
    GROUP 2 '/path/oracle/dbs/t_log2.f' SIZE 500K
# STANDBY LOGFILE
```

```
DATAFILE
  '/path/oracle/dbs/t_db1.f',
  '/path/oracle/dbs/dbu19i.dbf',
  '/path/oracle/dbs/tbs_11.f',
  '/path/oracle/dbs/smundo.dbf',
  '/path/oracle/dbs/demo.dbf'
CHARACTER SET WE8DEC
;
```

CREATE DATABASE

注意： この文を実行すると、データベースの初期設定が行われ、指定ファイル内の現行のデータは消去されます。そのことを十分理解したうえで、この文を使用してください。

セキュリティ拡張の登録に関する注意： Oracle の今回のリリース以降では、デフォルトのデータベース・ユーザー・アカウントのセキュリティを保証する機能拡張があります。

安全な方法で Oracle9i を構成するために、ガイドが提供されています。このガイドに従ってデータベースを構成することをお勧めします。このガイドは、次の URL で参照できます。

<http://otn.oracle.co.jp/idba/security/oracle9ir2/>

用途

CREATE DATABASE 文は、汎用的なデータベースを作成する場合に使用します。

この文は、データベースの初期使用に備えて、指定した既存のデータ・ファイル上のデータがすべて消去されます。したがって、既存のデータベースに対してこの文を実行した場合、データ・ファイル上のすべてのデータが失われます。

この文を指定した場合、データベースの作成後、データベースは排他モードまたはパラレル・モード（初期化パラメータ CLUSTER_DATABASE の値に依存する）でマウントおよびオープンされ、通常の用途に使用可能になります。同時に、データベースの表領域およびロールバック・セグメントを作成できます。

参照：

- データベースの変更の詳細は、9-12 ページの「[ALTER DATABASE](#)」を参照してください。
- Oracle9i Java Virtual Machine (JVM) の作成の詳細は、『Oracle9i Java Developer's Guide』を参照してください。
- ロールバック・セグメントおよび表領域の作成の詳細は、14-76 ページの「[CREATE ROLLBACK SEGMENT](#)」および 15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

前提条件

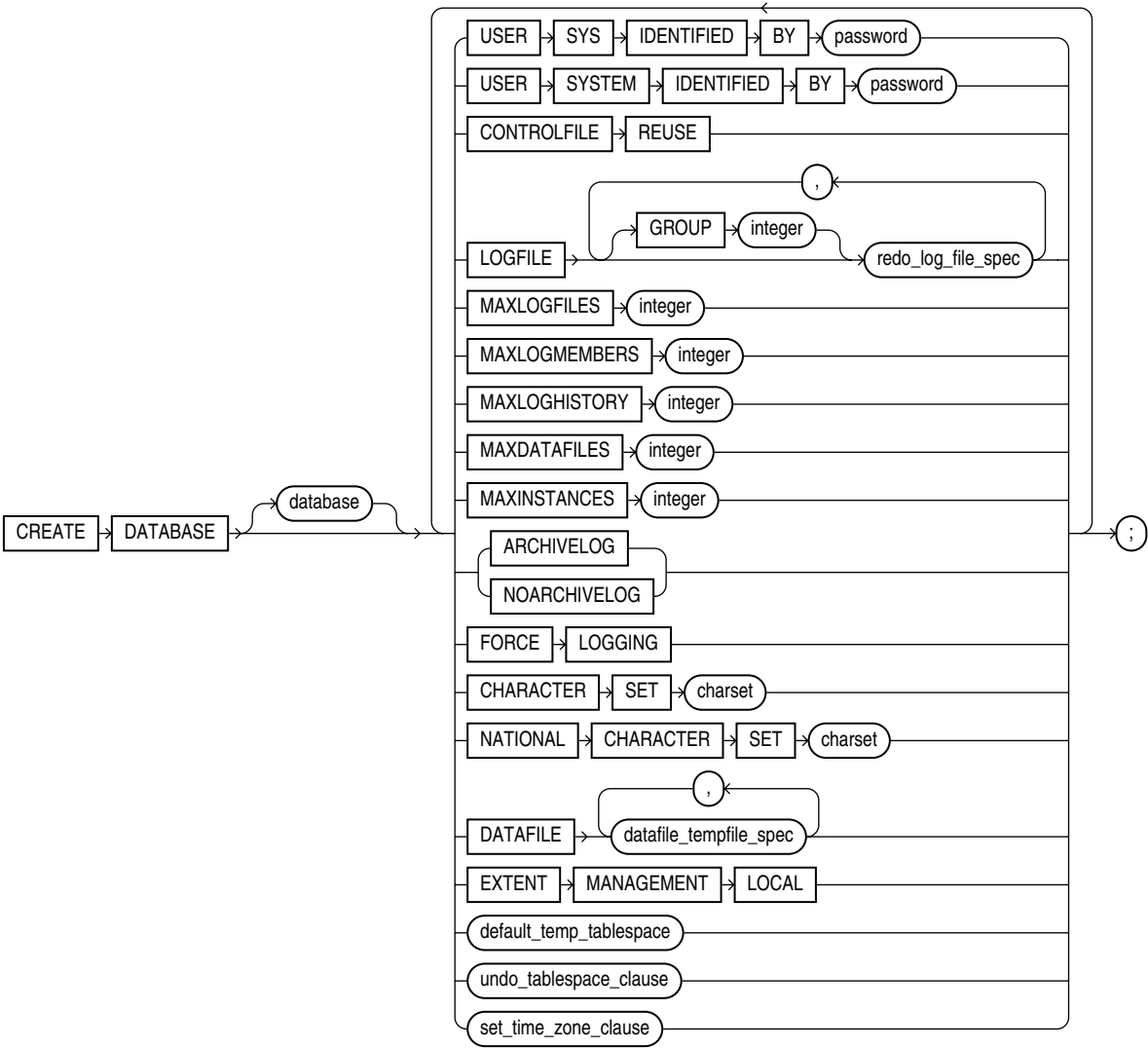
データベースを作成する場合は、SYSDBA システム権限が必要です。

初期化パラメータ `REMOTE_LOGIN_PASSWORDFILE` が `EXCLUSIVE` に設定されている場合、Oracle は、データベースを再作成しようとした際にエラーを戻します。エラーを発生させないためには、データベースを再作成する前にパラメータを `SHARED` に設定するか、またはパスワード・ファイルを再作成します。

参照： `REMOTE_LOGIN_PASSWORDFILE` パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

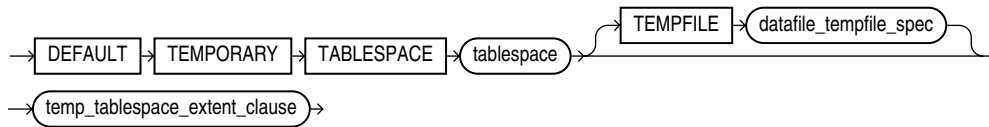
構文

create_database::=



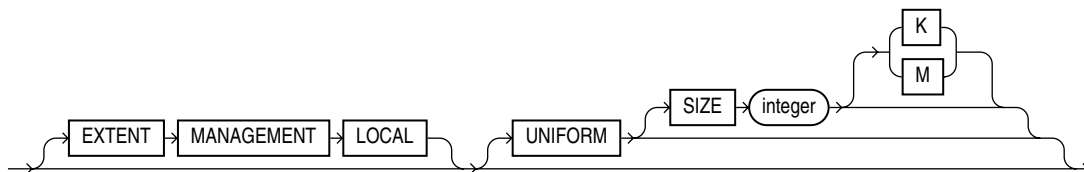
(7-37 ページの「redo_log_file_spec::=」、7-36 ページの「datafile_tempfile_spec::=」、13-25 ページの「default_temp_tablespace::=」、13-25 ページの「undo_tablespace_clause::=」、13-25 ページの「set_time_zone_clause::=」を参照)

default_temp_tablespace::=

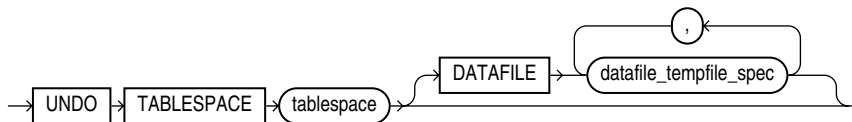


(7-36 ページの「`datafile_tempfile_spec::=`」(`file_specification`の一部)を参照)

temp_tablespace_extent::=

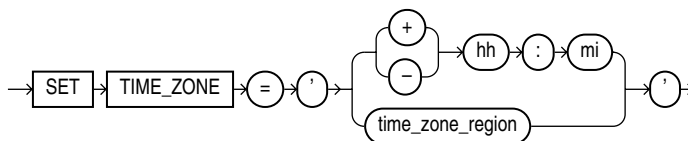


undo_tablespace_clause::=



(7-36 ページの「`datafile_tempfile_spec::=`」(`file_specification`の一部)を参照)

set_time_zone_clause::=



キーワードとパラメータ

database

作成するデータベースの名前を指定します。名前の長さは最大 8 バイトです。データベース名には、ASCII 文字のみ指定できます。指定したデータベース名は、Oracle によって制御ファイルに記録されます。後で、ALTER DATABASE 文を発行したときにデータベース名を明示的に指定すると、制御ファイル内の名前に基づいて、そのデータベース名が検証されます。

注意： データベース名には、ヨーロッパやアジアのキャラクタ・セットの中の特許文字は使用できません。たとえば、ウムラウト付きの文字は使用できません。

CREATE DATABASE 文でデータベース名を指定しない場合、初期化パラメータ DB_NAME で指定した名前が採用されます。初期化パラメータ DB_NAME が設定されており、そのパラメータの値とは異なる名前を指定した場合、Oracle はエラーを戻します。

参照： データベース名のその他の規則については、2-109 ページの「[スキーマ・オブジェクト名のネーミング計画](#)」を参照してください。

USER SYS ..., USER SYSTEM ...

これらの句を使用して SYS および SYSTEM ユーザーのパスワードを設定します。これらの句は Oracle9i の今回のリリースでは必須ではありません。ただし、いずれか一方の句を指定した場合は、もう一方の句も指定する必要があります。

これらの句を指定しない場合、Oracle は、デフォルト・パスワードとして、ユーザー SYS には「change_on_install」を、ユーザー SYSTEM には「manager」を作成します。これらのパスワードは、後で ALTER USER 文を使用して変更できます。また、データベース作成後に、ALTER USER を使用してパスワード管理属性を追加することもできます。

参照： 12-21 ページの「[ALTER USER](#)」を参照してください。

CONTROLFILE REUSE 句

CONTROLFILE REUSE を指定して、初期化パラメータ CONTROL_FILES で特定される既存の制御ファイルを再利用することを指定します。これによって、現在保存されている情報は無視され、上書きされます。通常、この句は、初めてデータベースを作成する際ではなく、データベースを再作成する際に使用します。制御ファイルを既存のファイルより大きくするためのパラメータ値もあわせて指定する場合、この句は使用できません。MAXLOGFILES、MAXLOGMEMBERS、MAXLOGHISTORY、MAXDATAFILES および MAXINSTANCES がこのようなパラメータです。

この句を指定しないと、CONTROL_FILES で指定した制御ファイルのいずれかがすでに存在する場合、Oracle はエラー・メッセージを戻します。

LOGFILE 句

REDO ログ・ファイルとして使用する 1 つ以上のファイルを指定します。各 redo_log_file_spec には、1 つ以上の REDO ログ・ファイルのメンバー（コピー）を含む REDO ログ・ファイル・グループを指定します。CREATE DATABASE 文に指定したすべての REDO ログ・ファイルは、REDO ログのスレッド番号 1 に追加されます。

参照： 詳細は、7-36 ページの「[file_specification](#)」を参照してください。

GROUP integer REDO ログ・ファイル・グループの識別番号を指定します。integer の値は 1 ～ MAXLOGFILES パラメータの値の範囲です。データベースには、2 つ以上の REDO ログ・ファイル・グループが必要です。同一の GROUP 値を持つ REDO ログ・ファイル・グループは複数指定できません。このパラメータを指定しない場合、値が自動的に生成されます。REDO ログ・ファイル・グループの GROUP 値は、動的パフォーマンス・ビュー V\$LOG で調べることができます。

LOGFILE を指定しない場合、次のようになります。

- 初期化パラメータ DB_CREATE_ONLINE_LOG_DEST_n と DB_CREATE_FILE_DEST のいずれか（または両方）を設定した場合、システムが生成する名前で、サイズが 100MB の 2 つの Oracle 管理ログ・ファイルが作成されます。ファイルは DB_CREATE_ONLINE_LOG_DEST_n が指定されていれば、そのディレクトリに、そうでなければ DB_CREATE_FILE_DEST パラメータで指定されるディレクトリに作成されます。
- どちらのパラメータも設定されていない場合、2 つの REDO ログ・ファイル・グループが作成されます。各デフォルト・ファイルの名前およびサイズは、使用するオペレーティング・システムによって異なります。

MAXLOGFILES 句

データベース用に作成可能な REDO ログ・ファイル・グループの最大数を指定します。この値を基にして、制御ファイル内で REDO ログ・ファイル名に割り当てられる領域が決定されます。デフォルト値、最小値および最大値は、使用するオペレーティング・システムによって異なります。

MAXLOGMEMBERS 句

REDO ログ・ファイル・グループのメンバー（コピー）の最大数を指定します。この値を基にして、制御ファイル内で REDO ログ・ファイル名に割り当てられる領域が決定されます。最小値は 1 です。最大値およびデフォルト値は、使用するオペレーティング・システムによって異なります。

MAXLOGHISTORY 句

このパラメータは、Real Application Clusters 環境で ARCHIVELOG モードの Oracle を使用している場合に便利です。Real Application Clusters の自動メディア・リカバリに使用するアーカイブ REDO ログ・ファイルの最大数を指定します。この値を基にして、制御ファイル内でアーカイブ REDO ログ・ファイル名に割り当てられる領域が決定されます。最小値は 0（ゼロ）です。デフォルト値は MAXINSTANCES 値の倍数で、使用するオペレーティング・システムによって異なります。最大値は、制御ファイルの最大サイズの制限のみを受けます。

MAXDATAFILES 句

CREATE DATABASE または CREATE CONTROLFILE 実行時の、制御ファイルのデータ・ファイル・セクションの初期サイズ設定を指定します。値が MAXDATAFILES より大きく、DB_FILES 以下のファイルを追加した場合、データ・ファイル・セクションにさらに多くのファイルを格納できるように、Oracle の制御ファイルが自動的に拡張されます。

インスタンスでアクセスできるデータ・ファイルの数は、初期化パラメータ DB_FILES の制限を受けます。

MAXINSTANCES 句

作成したデータベースを同時にマウントおよびオープンするインスタンスの最大数を指定します。この値は、初期化パラメータ INSTANCES の値より優先されます。最小値は 1 です。最大値およびデフォルト値は、使用するオペレーティング・システムによって異なります。

ARCHIVELOG | NOARCHIVELOG

ARCHIVELOG REDO ログ・ファイル・グループを再利用する前に、グループの内容をアーカイブする場合は、ARCHIVELOG を指定します。この句を指定した場合、メディア・リカバリができるようになります。

NOARCHIVELOG REDO ログ・ファイル・グループを再利用する前に、グループの内容をアーカイブする必要がない場合は、NOARCHIVELOG を指定します。この句を指定した場合、メディア・リカバリはできません。

デフォルトは NOARCHIVELOG モードです。データベースの作成後に、ALTER DATABASE 文を使用した場合、ARCHIVELOG モードと NOARCHIVELOG モードを切り替えることができます。

FORCE LOGGING

この句を使用すると、データベースは FORCE LOGGING モードになります。一時表領域および一時セグメントへの変更以外のデータベース内のすべての変更のログが記録されます。この設定は、各表領域で指定する NOLOGGING または FORCE LOGGING 設定、および各データベース・オブジェクトで指定する NOLOGGING 設定より優先され、これらの設定には影響されません。

FORCE LOGGING モードは、データベースのインスタンスで永続的です。つまり、データベースを停止し、再起動しても、データベースは FORCE LOGGING モードを維持します。ただし、制御ファイルを再作成した場合は、CREATE CONTROLFILE 文で FORCE LOGGING を指定しないかぎり、データベースは FORCE LOGGING モードではなくなります。

注意： FORCE LOGGING モードは、パフォーマンスに影響する場合があります。この設定の使用については、『Oracle9i データベース管理者ガイド』を参照してください。

参照： 13-14 ページの「[CREATE CONTROLFILE](#)」を参照してください。

CHARACTER SET 句

データベースにデータを格納するときのキャラクタ・セットを指定します。サポートされているキャラクタ・セットおよびこのパラメータのデフォルト値は、使用するオペレーティング・システムによって異なります。

CHARACTER SET の制限事項： AL16UTF16 キャラクタ・セットは、データベース・キャラクタ・セットとして指定できません。

参照： キャラクタ・セットの選択の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

NATIONAL CHARACTER SET 句

データ型が NCHAR、NCLOB または NVARCHAR2 で定義された列にデータを格納する際に使用する各国語キャラクタ・セット（AF16UTF16 と UTF8 のいずれか）を指定します。デフォルトは 'AL16UTF16' です。

参照： Unicode データ型のサポートの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

DATAFILE 句

データ・ファイルとして使用する 1 つ以上のファイルを指定します。ファイルは、すべて SYSTEM 表領域の一部となります。

注意： `undo_tablespace_clause` の DATAFILE 句と同様に、この句はオプションです。したがって、あいまいさを回避するために、この句で SYSTEM 表領域のデータ・ファイルを指定する場合は、オプションの DATAFILE 句を含まない `undo_tablespace_clause` の直後にこの句を指定しないでください。指定した場合、Oracle は DATAFILE 句を `undo_tablespace_clause` の一部と認識します。

自動 UNDO モードでデータベースを実行し、SYSTEM 表領域のデータ・ファイル名を指定した場合、すべての表領域に対してデータ・ファイルを生成するものとみなされます。Oracle 管理ファイルを使用している場合（DB_CREATE_FILE_DEST または DB_CREATE_ONLINE_LOG_DEST_n 初期化パラメータに値を設定した場合）、データ・ファイルは自動的に生成されます。Oracle 管理ファイルを使用していない場合にこの句を指定すると、`undo_tablespace_clause` および `default_temp_tablespace` 句も指定する必要があります。

この句を指定しない場合、次のようになります。

- 初期化パラメータ `DB_CREATE_FILE_DEST` が設定されている場合、パラメータで指定したデフォルトのファイル格納先に、サイズが **100MB** でシステムが生成する名前の、Oracle 管理データ・ファイルが作成されます。
- `DB_CREATE_FILE_DEST` 初期化パラメータが設定されていない場合、1 つのデータ・ファイルが作成されます。そのファイル名およびサイズは、使用するオペレーティング・システムによって異なります。

注意： `SYSTEM` 表領域に割り当てる初期領域は、全体で **5MB** 以上になるように指定してください。

参照： 構文の詳細は、7-36 ページの「[file_specification](#)」を参照してください。

EXTENT MANAGEMENT LOCAL

この句を使用すると、ローカル管理 `SYSTEM` 表領域を作成できます。この句を省略した場合、`SYSTEM` 表領域はディクショナリ管理となります。

注意： 一度ローカル管理 `SYSTEM` 表領域を作成すると、この表領域をディクショナリ管理に変更することはできません。このデータベース内に別のディクショナリ管理表領域を作成することもできません。

この句を指定した場合、ローカル管理 `SYSTEM` 表領域には、一時セグメントを格納できないため、データベースにデフォルト一時表領域が必要となります。

- `EXTENT MANAGEMENT LOCAL` を指定して、`DATAFILE` 句を指定しない場合は、`default_temp_tablespace` 句を省略できます。デフォルト一時表領域 `TEMP` がデータ・ファイル・サイズ **10MB** で、自動拡張を使用禁止にした状態で作成されます。
- `EXTENT MANAGEMENT LOCAL` および `DATAFILE` の両方の句を指定する場合は、`default_temp_tablespace` を指定し、その表領域のデータ・ファイルを明示的に指定する必要があります。

インスタンスを自動 `UNDO` 管理モードで開いた場合も、データベース `UNDO` 表領域には同様の要件があります。

- `EXTENT MANAGEMENT LOCAL` を指定して、`DATAFILE` 句を指定しない場合は、`undo_tablespace_clause` 句を省略できます。`UNDO` 表領域 `SYS_UNDOTBS` が作成されます。

- EXTENT MANAGEMENT LOCAL および DATAFILE の両方の句を指定する場合は、`undo_tablespace_clause` を指定し、その表領域のデータ・ファイルを明示的に指定する必要があります。

参照： ローカル管理およびディクショナリ管理表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

default_temp_tablespace

データベースのデフォルトの一時表領域を作成します。別の一時表領域を指定しないユーザーに、この一時表領域が割り当てられます。この句を指定しない場合、SYSTEM 表領域がデフォルトの一時表領域になります。

DB_CREATE_FILE_DEST 初期化パラメータを設定して Oracle 管理ファイルを使用可能にした場合、この句の TEMPFILE 句の部分はオプションです。このパラメータの値を指定していない場合は、TEMPFILE 句を指定する必要があります。

注意： オペレーティング・システムによっては、一時ファイルのブロックが実際にアクセスされるまで、一時ファイル用の領域が割り当てられない場合があります。領域の割当ての遅延のため、一時ファイルの作成およびサイズ変更が速くなります。ただし、後で一時ファイルが使用されるときに、十分なディスク領域を使用可能にする必要があります。発生する可能性がある問題を回避するには、一時ファイルを作成またはサイズ変更する前に、ディスク領域が、新しく作成する一時ファイルまたはサイズ変更後の一時ファイルのサイズより大きいことを確認してください。ディスク領域に余裕を持たせることによって、関連のない操作が原因で増加が予想されるディスク使用量に対応できます。十分な領域があることを確認した後で、作成またはサイズ変更操作を実行してください。

一時表領域の制限事項：

- この句では、SYSTEM 表領域を指定できません。
- デフォルトの一時表領域は、標準的なブロック・サイズである必要があります。

`temp_tablespace_extent` 句を使用すると、表領域の管理方法を指定できます。

EXTENT MANAGEMENT LOCAL この句は、表領域の一部がビットマップ用に確保されることを示します。すべての一時表領域にはローカル管理のエクステンツがあるため、この句はオプションです。

UNIFORM integer 一時表領域のエクステンツ・サイズをバイトで指定します。表領域のエクステンツはすべて同じサイズ（均一）です。この句を指定しない場合は、均一のエクステンツ 1MB を使用します。

SIZE integer 表領域のエクステントのサイズをバイト単位で指定します。K または M を使用して、KB または MB 単位で指定することもできます。

SIZE を指定しない場合は、デフォルトのエクステント・サイズ 1MB を使用します。

参照： ローカル管理表領域については、『Oracle9i データベース概要』を参照してください。

undo_tablespace_clause

インスタンスを自動 UNDO モードでオープンした場合（つまり、UNDO_MANAGEMENT 初期化パラメータを AUTO に設定した場合）、*undo_tablespace_clause* を指定して、UNDO データで使用する表領域を作成できます。UNDO 領域管理をロールバック・セグメントによって処理する場合、この句は省略します。UNDO_TABLESPACE 初期化パラメータの値を指定した場合も、この句を省略できます。パラメータが設定されており、この句を指定した場合、*tablespace* はそのパラメータ値と同じである必要があります。

DB_CREATE_FILE_DEST 初期化パラメータを設定して Oracle 管理ファイルを使用可能にした場合、この句の DATAFILE 句の部分はオプションです。このパラメータの値を指定していない場合は、DATAFILE 句を指定する必要があります。

- この句を指定すると、*tablespace* という名前の UNDO 表領域、および UNDO 表領域の一部として指定したデータ・ファイルが作成され、インスタンスの UNDO 表領域としてこの表領域が割り当てられます。Oracle は、この UNDO 表領域を使用して UNDO データを管理します。この句の DATAFILE 句は、13-29 ページの「[DATAFILE 句](#)」と同様に動作します。

注意： 初期化パラメータ・ファイル内の UNDO_TABLESPACE 初期化パラメータの値を指定する場合は、データベースをマウントする前に、この句で同じ名前を指定してください。この名前が異なると、データベースのオープン時にエラーが戻されます。

- この句を省略すると、SYS_UNDOTBS という名前のデフォルトの UNDO 表領域でデフォルトのデータベースが作成され、インスタンスの UNDO 表領域としてこのデフォルトの表領域が割り当てられます。この UNDO 表領域は、CREATE DATABASE 文で使用するデフォルトのファイルから、初期エクステントが 10MB のディスク領域を割り当てます。Oracle は、システムが生成するデータ・ファイル（13-29 ページの「[DATAFILE 句](#)」を参照）を処理します。Oracle が UNDO 表領域を作成できない場合、CREATE DATABASE 操作全体が失敗します。

参照：

- UNDO_MANAGEMENT パラメータを使用した自動 UNDO 管理モードのデータベース・インスタンスをオープンする場合の詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 自動 UNDO 管理および UNDO 表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- データベース作成後の UNDO 表領域を作成する場合の詳細は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

set_time_zone_clause

SET TIME_ZONE 句を使用すると、データベースのタイムゾーンを設定できます。次の 2 つの方法でタイムゾーンを設定します。

- UTC からの時差を指定。hh:mm の有効範囲は、-12:00 ～ +14:00 です。
- タイムゾーン地域を指定。有効な地域名を表示するには、V\$TIMEZONE_NAMES 動的パフォーマンス・ビューの TZNAME 列を問い合わせます。

注意： データベースのタイムゾーンを UTC ("0:00") に設定することをお薦めします。これによって、タイムゾーンの変換が不要になるため、特にデータベース間のパフォーマンスを向上できます。

参照： 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

すべての TIMESTAMP WITH LOCAL TIME ZONE データは、ディスクに格納されるときにデータベースのタイムゾーンに正規化されます。SET TIME_ZONE 句を指定しない場合、サーバーのオペレーティング・システムのタイムゾーンが使用されます。オペレーティング・システムのタイムゾーンが Oracle で有効でない場合、データベースのタイムゾーンは、デフォルトで UTC になります。

例

データベースの作成例： 次の文は、すべての各引数を完全に指定してデータベースを作成します。

```
CREATE DATABASE sample
  CONTROLFILE REUSE
  LOGFILE
    GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
    GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
  MAXLOGFILES 5
  MAXLOGHISTORY 100
  MAXDATAFILES 10
  MAXINSTANCES 2
  ARCHIVELOG
  CHARACTER SET AL32UTF8
  NATIONAL CHARACTER SET AL16UTF16
  DATAFILE
    'disk1:df1.dbf' AUTOEXTEND ON,
    'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
  DEFAULT TEMPORARY TABLESPACE temp_ts
  UNDO TABLESPACE undo_ts
  SET TIME_ZONE = '+02:00';
```

この例では、初期化パラメータ・ファイルの DB_CREATE_FILE_DEST パラメータに値を設定して、Oracle 管理ファイルを使用可能にしている状態を想定しています。したがって、DEFAULT TEMPORARY TABLESPACE および UNDO TABLESPACE 句にファイル指定は必要ありません。

CREATE DATABASE LINK

用途

CREATE DATABASE LINK 文は、データベース・リンクを作成する場合に使用します。データベース・リンクとは、他のデータベース上のオブジェクトにアクセスできる、データベース上のスキーマ・オブジェクトです。他のデータベースは、Oracle データベースである必要はありません。ただし、Oracle 以外のシステムにアクセスする場合は、Oracle 異機種間サービスを使用する必要があります。

データベース・リンクを作成した場合、そのリンクを利用して、他のデータベース上の表およびビューを参照できます。また、SQL 文では、表名またはビュー名に `@dblink` を付けることによって、他のデータベース上の表またはビューを参照できます。SELECT 文を使用すると、他のデータベース上の表またはビューの間合せができます。INSERT 文、UPDATE 文、DELETE 文または LOCK TABLE 文を使用してもリモート表およびビューにアクセスできます。

参照：

- PL/SQL ファンクション、プロシージャ、パッケージおよびデータ型を使用してリモート表またはビューへアクセスする方法については、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- ALL_DB_LINKS、DBA_DB_LINKS および USER_DB_LINKS データ・ディクショナリ・ビューの既存のデータベース・リンクの詳細、および V\$DBLINK 動的パフォーマンス・ビューを使用して既存のリンクのパフォーマンスを監視する方法については、『Oracle9i データベース・リファレンス』を参照してください。
- 既存のデータベース・リンクの削除については、16-67 ページの「[DROP DATABASE LINK](#)」を参照してください。
- DML 操作でのリンクの使用については、17-52 ページの「[INSERT](#)」、18-56 ページの「[UPDATE](#)」、16-53 ページの「[DELETE](#)」および 17-71 ページの「[LOCK TABLE](#)」を参照してください。

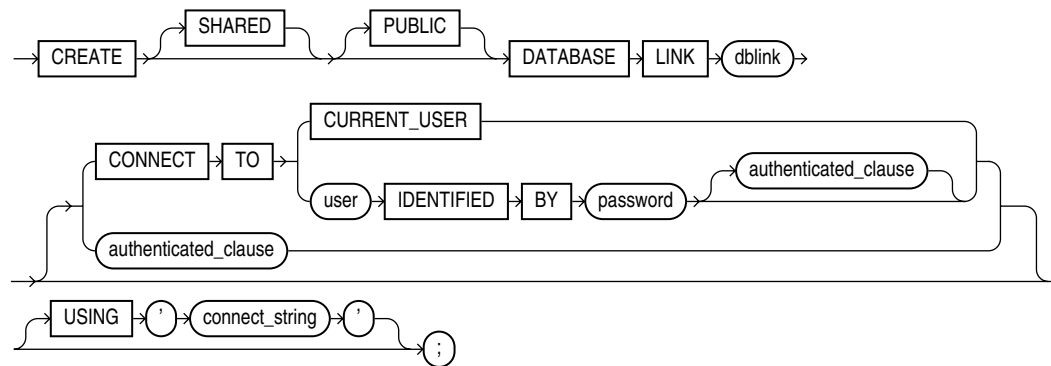
前提条件

プライベート・データベース・リンクを作成する場合、CREATE DATABASE LINK システム権限が必要です。パブリック・データベース・リンクを作成する場合は、CREATE PUBLIC DATABASE LINK システム権限が必要です。また、リモートの Oracle データベースに対する CREATE SESSION 権限が必要です。

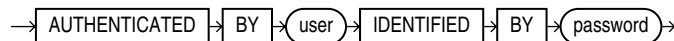
なお、ローカルとリモートの両方の Oracle データベースに、Oracle Net をインストールしておく必要があります。

構文

create_database_link::=



authenticated_clause::=



キーワードとパラメータ

SHARED

SHARED を指定して、1 つのネットワーク接続を使用することによって、複数ユーザーで共有できるパブリック・データベース・リンクを作成します。

PUBLIC

PUBLIC を指定して、すべてのユーザーが使用可能なパブリック・データベース・リンクを作成します。この句を指定しない場合、データベース・リンクはプライベートとなり、作成したユーザー専用になります。

参照： 13-39 ページの「パブリック・データベース・リンクの定義例:」を参照してください。

dblink

データベース・リンクの完全な名前または名前の一部を指定します。データベース名のみを指定した場合、ローカル・データベースのデータベース・ドメインが暗黙的に追加されます。

GLOBAL_NAMES 初期化パラメータの値が TRUE の場合、データベース・リンクは、接続先のデータベースと同じ名前を持つ必要があります。GLOBAL_NAMES の値が FALSE で、データベースのグローバル名を変更した場合、グローバル名を指定できます。

Real Application Clusters 構成の 1 つのセッションまたは 1 つのインスタンスでオープンできるデータベース・リンクの最大数は、OPEN_LINKS および OPEN_LINKS_PER_INSTANCE 初期化パラメータの値で決まります。

データベース・リンク作成の制限事項： 他のユーザーのスキーマ内にはデータベース・リンクを作成できません。また、*dblink* はスキーマ名を付けて指定できません（データベース・リンク名にはピリオドを指定できるため、*ralph.linktosales* のような名前を付けた場合、スキーマ *ralph* の *linktosales* という名前のデータベース・リンクと解析されるのではなく、名前全体が自分のスキーマにあるデータベース・リンク名と解析されます）。

参照：

- データベース・リンクのネーミングのガイドラインについては、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。
- GLOBAL_NAMES、OPEN_LINKS および OPEN_LINKS_PER_INSTANCE 初期化パラメータについては、『Oracle9i データベース・リファレンス』を参照してください。
- データベースのグローバル名の変更については、9-50 ページの「[RENAME GLOBAL_NAME 句](#)」（ALTER DATABASE 句）を参照してください。

CONNECT TO 句

CONNECT TO によって、リモート・データベースへの接続が可能になります。

CURRENT_USER 句

CURRENT_USER を指定して、**現行のユーザーのデータベース・リンク**を作成します。正常にリンクを作成する場合、現行のユーザーが、リモート・データベースに有効なアカウントを持つグローバル・ユーザーである必要があります。

データベース・リンクが直接使用される場合（ストアド・オブジェクト内から使用されていない場合）、現行のユーザーは接続ユーザーと同じです。

データベース・リンクを開始するストアド・オブジェクト（プロシージャ、ビューまたはトリガーなど）を実行する場合、CURRENT_USER とは、ストアド・オブジェクトを所有するユーザー名であり、オブジェクトをコールしたユーザー名ではありません。たとえば、データベース・リンクが（scott によって作成された）プロシージャ *scott.p* 内にあり、ユーザー *jane* がプロシージャ *scott.p* をコールした場合、現行のユーザーは *scott* になります。

ただし、ストアド・オブジェクトが実行者権限ファンクション、プロシージャまたはパッケージである場合、実行者認可 ID はリモート・ユーザーとしての接続に使用されます。たとえば、権限を持つデータベース・リンクがプロシージャ `scott.p` (`scott` によって作成された実行者権限プロシージャ) 内にあり、ユーザー `Jane` がプロシージャ `scott.p` をコールした場合、`CURRENT_USER` は `jane` であり、プロシージャは、`Jane` の権限で実行されます。

参照：

- 実行者権限ファンクションの詳細は、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- 13-40 ページの「[CURRENT_USER データベース・リンクの定義例:](#)」を参照してください。

user IDENTIFIED BY password

固定ユーザー・データベース・リンクを使用して、リモート・データベースに接続するためのユーザー名およびパスワードを指定します。この句を指定しない場合、データベース・リンクでは、データベースに接続している各ユーザーのユーザー名およびパスワードが使用されます。これを[接続ユーザー・データベース・リンク](#)といいます。

参照： 13-39 ページの「[固定ユーザー・データベース・リンクの定義例:](#)」を参照してください。

authenticated_clause

ターゲット・インスタンスのユーザー名およびパスワードを指定します。この句は、リモート・サーバーに対してユーザーを認証するもので、セキュリティ上必要です。指定するユーザー名およびパスワードは、リモート・インスタンスで有効なユーザー名およびパスワードである必要があります。ユーザー名およびパスワードは、認証用としてのみ使用されます。このユーザーを対象とした認証以外の操作はありません。

`SHARED` 句を使用している場合は、必ずこの句を指定してください。

USING 'connect_string'

リモート・データベースのサービス名を指定します。データベース名のみを指定した場合、接続文字列にデータベース・ドメインが暗黙的に追加され、完全なサービス名が作成されます。したがって、リモート・データベースのデータベース・ドメインが現行のデータベース・ドメインと異なる場合は、完全なサービス名を指定する必要があります。

参照： リモート・データベースの指定の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

例

次の例では、「local」という名前のデータベースと「remote」という名前の2つのデータベースを使用することを想定しています。この例では、Oracle ドメインを使用します。ご使用のドメインとは異なります。

パブリック・データベース・リンクの定義例： 次の例では、remote という共有パブリック・データベース・リンクを定義します。このデータベース・リンクは、サービス名「sales」で指定されたデータベースと対応しています。

```
CREATE PUBLIC DATABASE LINK remote
  USING 'remote';
```

このデータベース・リンクによって、local データベース上のユーザー hr がリモート・データベース上の表を更新できます (hr に適切な権限があることを想定しています)。

```
UPDATE employees@remote
  SET salary=salary*1.1
  WHERE last_name = 'Baer';
```

固定ユーザー・データベース・リンクの定義例： 次の例では、リモート・データベース上のユーザー hr が、local データベース上の hr スキーマに、local という名前の固定ユーザー・データベース・リンクを定義します。

```
CREATE DATABASE LINK local
  CONNECT TO hr IDENTIFIED BY hr
  USING 'local';
```

このデータベース・リンクが作成されると、hr は、この方法で local データベース上のスキーマ hr の表を問い合わせることができます。

```
SELECT * FROM employees@local;
```

ユーザー hr は、次のように DML 文を使用して、local データベース上のデータの変更もできます。

```
INSERT INTO employees@local
  (employee_id, last_name, email, hire_date, job_id)
  VALUES (999, 'Claus', 'sclaus@oracle.com', SYSDATE, 'SH_CLERK');

UPDATE jobs@local SET min_salary = 3000
  WHERE job_id = 'SH_CLERK';

DELETE FROM employees@local
  WHERE employee_id = 999;
```

この固定データベース・リンクを使用すると、remote データベース上のユーザー `hr` は、同じデータベース上の他のユーザーが所有する表にもアクセスできます。この文は、ユーザー `hr` が `oe.customers` 表に対する `SELECT` 権限を持っていることを想定しています。この文では、local データベースのユーザー `hr` に接続した後で、`oe` の `customers` 表への問合せが行われます。:

```
SELECT * FROM oe.customers@local;
```

CURRENT_USER データベース・リンクの定義例: 次の例では、リンク名としてサービス名全体を使用して、remote データベースを使用している現行のユーザーのデータベース・リンクを定義します。

```
CREATE DATABASE LINK remote.us.oracle.com
CONNECT TO CURRENT_USER
USING 'remote';
```

この文を発行するユーザーは、LDAP ディレクトリ・サービスに登録されたグローバル・ユーザーである必要があります。

特定の表が remote データベース上にあることを示さないように、シノニムを作成できます。次の文では、`emp_table` を参照すると、リモート・データベース上の `hr` が所有する `employees` 表に必ずアクセスします。

```
CREATE SYNONYM emp_table
FOR oe.employees@remote.us.oracle.com;
```

CREATE DIMENSION

用途

CREATE DIMENSION 文を使用すると、ディメンションを作成できます。ディメンションは、列集合の組の親子関係を定義します。列集合の列は、すべて同じ表から得られたものである必要があります。ただし、1つの列集合（またはレベル）の列は、別の集合の列とは異なる表から得ることができます。オブティマイザは、マテリアライズド・ビューとの関係を使用してクエリー・リライトを行います。サマリー・アドバイザは、特定のマテリアライズド・ビューの作成の推奨にこの関係を使用します。

注意： Oracle は、ディメンションの作成中に宣言する関係の妥当性チェックを自動的には行いません。 *hierarchy_clause* および CREATE DIMENSION の *join_clause* で指定する関係の妥当性チェックを行うには、DBMS_OLAP.validate_dimension プロシージャを実行する必要があります。

参照：

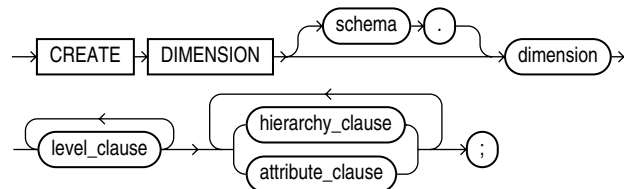
- マテリアライズド・ビューの詳細は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。
- クエリー・リライト、オブティマイザおよびサマリー・アドバイザの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- DBMS_OLAP.validate_dimension プロシージャの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

前提条件

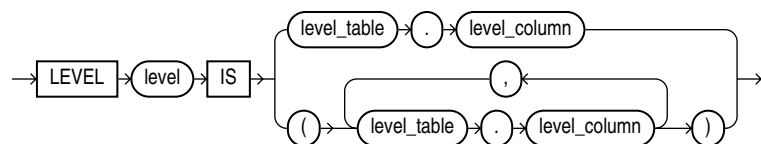
自分のスキーマ内にディメンションを作成する場合は、CREATE DIMENSION システム権限が必要です。他のユーザーのスキーマ内にディメンションを作成する場合は、CREATE ANY DIMENSION システム権限が必要です。どちらの場合も、ディメンションで参照されるオブジェクトに対して、SELECT オブジェクト権限が必要です。

構文

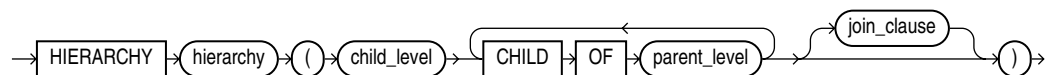
create_dimension::=



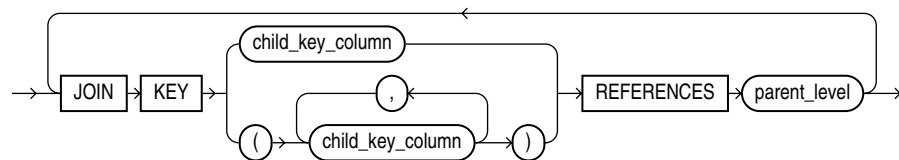
level_clause::=



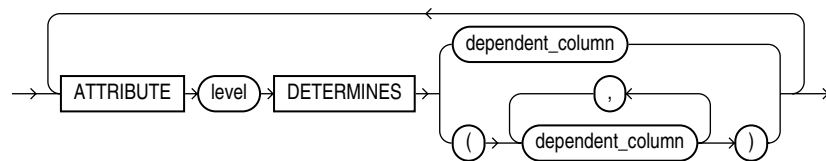
hierarchy_clause::=



join_clause::=



attribute_clause::=



セマンティクス

schema

ディメンションを作成するスキーマを指定します。**schema** を指定しない場合、自分のスキーマ内にディメンションが作成されます。

dimension

ディメンション名を指定します。名前は、スキーマ内で一意である必要があります。

level_clause

level_clause は、ディメンションのレベルを指定します。レベルは、ディメンション階層および属性を定義します。

level レベル名を指定します。

level_table . level_column レベルの列を指定します。最大 32 列を指定できます。この句で指定する表は、すでに存在している必要があります。

レベル列の制限事項：

- レベルの列は、すべて同じ表から得られたものである必要があります。
- 異なるレベルの列が異なる表から得られる場合、**join_clause** を指定する必要があります。
- 指定する列の集合は、このレベルに一意である必要があります。
- 指定する列は、他のディメンションでは指定できません。
- 各 **level_column** は、NULL 以外である必要があります（ただし、この列は NOT NULL 制約を受ける必要はありません）。

hierarchy_clause

ディメンションの線形階層レベルを定義します。各階層が、ディメンションのレベル間で親子関係の連鎖を形成します。ディメンションの階層は、互いに依存していません。階層は、共通の列を持つことができます。

ディメンションの各レベルは、句の中で最高 1 回指定され、**level_clause** で名前を付けておく必要があります。

hierarchy 階層名を指定します。この名前は、ディメンションで一意である必要があります。

child_level 親レベルと n:1 の関係を持つレベルの名前を指定します。child_level の level_columns は NULL 以外である必要があります。各 child_level 値は、parent_level という名前の次の値を一意に定義します。

子 level_table が親 level_table と異なる場合、join_clause でそれらの結合関係を指定する必要があります。

parent_level レベル名を指定します。

join_clause

複数の表に列が含まれるディメンションに内部等価結合関係を指定できます。この句は、階層で指定されたすべての列が同じ表にあるとはかぎらない場合にのみ指定する必要があり、このときのみ指定できます。

child_key_column

親レベルの列と結合互換性のある 1 つ以上の列を指定します。

スキーマおよび各 child_column の表を指定しない場合、hierarchy_clause の CHILD OF 関係からスキーマおよび表が判断されます。child_key_column のスキーマおよび列を指定する場合は、hierarchy_clause の parent_level の子のスキーマおよび表と一致している必要があります。

parent_level

レベル名を指定します。

ディメンションの結合の制限事項：

- 同じ階層の既存のレベルの組に対して、1 つの join_clause のみを指定できます。
- child_key_columns は NULL 以外であり、親キーが一意で NULL 以外である必要があります。条件を適用するために制約を定義する必要はありません。ただし、条件を満たさない場合、問合せが不適切な結果を戻すことがあります。
- 各子キーは、parent_level 表のキーと結合する必要があります。
- 自己結合はできません。つまり、child_key_columns を、parent_level として同じ表に置くことはできません。
- 子キー列は、すべて同じ表から得られたものである必要があります。
- 子キー列数は、parent_level の列数と一致し、列は結合可能である必要があります。
- 親レベルが複数の列で構成されている場合のみ、子キー列を指定します。

attribute_clause

*attribute_clause*によって、階層レベルによって一意に定義されている列を指定できます。*level*の列は、*dependent_columns*として同じ表からすべて得る必要があります。*dependent_columns*は、*level_clause*で指定されている必要はありません。

たとえば、階層レベルが市、都道府県名、および国の場合、市は市長、都道府県名は知事、国は首相を決定します。

例

ディメンションの作成例： 次の文は、サンプル・スキーマ *sh* に *customers_dim* ディメンションを作成します。

```
CREATE DIMENSION customers_dim
  LEVEL customer    IS (customers.cust_id)
  LEVEL city        IS (customers.cust_city)
  LEVEL state       IS (customers.cust_state_province)
  LEVEL country     IS (countries.country_id)
  LEVEL subregion   IS (countries.country_subregion)
  LEVEL region      IS (countries.country_region)
  HIERARCHY geog_rollup (
    customer        CHILD OF
    city            CHILD OF
    state           CHILD OF
    country         CHILD OF
    subregion       CHILD OF
    region
  )
  JOIN KEY (customers.country_id) REFERENCES country
)
ATTRIBUTE customer DETERMINES
(cust_first_name, cust_last_name, cust_gender,
 cust_marital_status, cust_year_of_birth,
 cust_income_level, cust_credit_limit)
ATTRIBUTE country DETERMINES (countries.country_name)
;
```

CREATE DIRECTORY

用途

CREATE DIRECTORY 文は、ディレクトリ・オブジェクトを作成する場合に使用します。ディレクトリ・オブジェクトは、外部バイナリ・ファイル LOB (BFILE) および外部表データが存在するサーバー・ファイル・システム上のディレクトリの別名を示します。PL/SQL コードおよび OCI コールで BFILE を参照する際、オペレーティング・システムのパス名をハード・エンコードせずにディレクトリ名を使用できます。このため、ファイル管理の汎用性が向上します。

すべてのディレクトリは 1 つのネームスペースに作成されるため、個々のユーザーのスキーマで所有されるわけではありません。ディレクトリに対するオブジェクト権限を特定ユーザーに付与することによって、そのディレクトリ構造内に格納されている BFILE へのアクセスを制限できます。

参照：

- BFILE オブジェクトの詳細は、2-26 ページの「[ラージ・オブジェクト \(LOB\) データ型](#)」を参照してください。
- オブジェクト権限の付与の詳細は、17-27 ページの「[GRANT](#)」を参照してください。
- 15-32 ページの CREATE TABLE の「[external_table_clause](#)」を参照してください。

前提条件

ディレクトリを作成する場合は、CREATE ANY DIRECTORY システム権限が必要です。

ディレクトリを作成する場合、そのディレクトリに対する READ オブジェクト権限および WRITE オブジェクト権限が自動的に付与され、他のユーザーおよびロールにこれらの権限を付与できます。DBA も、これらの権限を他のユーザーおよびロールに付与できます。

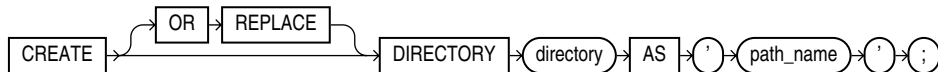
ディレクトリに対する WRITE 権限は、外部表との接続に便利です。これによって、権限受領者は、外部表のエージェントがディレクトリに書き込めるのがログ・ファイルなのか不良ファイルなのかを判断できます。

また、ファイル格納用として、対応するオペレーティング・システムのディレクトリも作成する必要があります。システム管理者およびデータベース管理者は、このオペレーティング・システムのディレクトリに、Oracle プロセスに対する読み込み権限および書き込み権限が正しく設定されていることを確認する必要があります。

ディレクトリに対して付与される権限は、オペレーティング・システムのディレクトリ用に定義されたアクセス権限とは無関係に作成されます。このため、これら2つは正確に対応しない場合があります。たとえば、サンプル・ユーザー `hr` に、ディレクトリ・スキーマ・オブジェクトに対する読み込み権限が付与されていても、それに対応するオペレーティング・システムのディレクトリに **Oracle** プロセスに対する読み込み権限が付与されていない場合は、エラーが発生します。

構文

create_directory::=



セマンティクス

OR REPLACE

既存のディレクトリ・データベース・オブジェクトを再作成する場合は、**OR REPLACE** を指定します。この句を指定した場合、既存のディレクトリに付与されているデータベース・オブジェクト権限を削除、再作成および再付与しなくても、そのディレクトリの定義を変更できます。

再定義したディレクトリに対する権限が付与されていたユーザーは、権限が再付与されなくてもそのディレクトリにアクセスできます。

参照： データベースからのディレクトリの削除については、16-71 ページの「[DROP DIRECTORY](#)」を参照してください。

directory

作成するディレクトリ・オブジェクトの名前を指定します。*directory* の最大長は 30 バイトです。ディレクトリ・オブジェクトは、スキーマ名で修飾できません。

注意： 指定したディレクトリが実際に存在するかどうかは検証されません。このため、オペレーティング・システムに存在するディレクトリを指定してください。また、オペレーティング・システムで使用するパス名が大文字と小文字を区別する場合は、必ず、正しい形式でディレクトリ名を指定してください（ただし、パス名の終わりにスラッシュを指定する必要はありません）。

'path_name'

ファイルが格納されているサーバー上のオペレーティング・システムのディレクトリのフルパス名を指定します。指定するフルパス名は、一重引用符で囲む必要があります。また、パス名の大文字と小文字は区別されます。

例

ディレクトリの作成例： 次の文は、サーバーのディレクトリを示す、ディレクトリのデータベース・オブジェクトを作成します。

```
CREATE DIRECTORY admin AS 'oracle/admin';
```

次の文は、オペレーティング・システムのディレクトリ `/private1/lob/files` に格納されている BFILE にアクセスできるように、ディレクトリのデータベース・オブジェクト `bfile_dir` を再定義します。

```
CREATE OR REPLACE DIRECTORY bfile_dir AS '/private1/LOB/files';
```

CREATE FUNCTION

用途

CREATE FUNCTION 文を使用すると、スタンドアロン・ストアド・ファンクションまたはコール仕様を作成できます。(また、CREATE PACKAGE 文を使用して、パッケージの一部としてファンクションを作成することもできます)。

ストアド・ファンクション (ユーザー・ファンクション) は、名前でコールできる PL/SQL 文の集合です。ストアド・ファンクションは、プロシージャとよく似ていますが、ファンクションは、コールした環境に値を戻す点で異なります。ストアド・ファンクションは、SQL 式の一部として使用できます。

コール仕様は、SQL および PL/SQL からコールできるように、Java メソッドまたは 3GL 世代言語 (3GL) ルーチンを宣言します。コール仕様は、コールされたときに起動する Java メソッドまたは共有ライブラリの名前付きファンクションを問い合わせます。引数および戻り値に対する型変換も問い合わせます。

参照：

- プロシージャおよびファンクションの概要は、14-60 ページの「[CREATE PROCEDURE](#)」を参照してください。
- ファンクションの作成例については、13-58 ページの「[例](#)」を参照してください。
- パッケージの作成については、14-48 ページの「[CREATE PACKAGE](#)」を参照してください。
- ファンクションの変更については、9-59 ページの「[ALTER FUNCTION](#)」を参照してください。
- 共有ライブラリについては、14-2 ページの「[CREATE LIBRARY](#)」を参照してください。
- スタンドアロン・ファンクションの削除については、16-72 ページの「[DROP FUNCTION](#)」を参照してください。
- 外部ファンクションの登録については、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

前提条件

ストアド・ファンクションを作成する前に、ユーザー SYS は、SQL スクリプト DBMSSTDY.SQL を実行する必要があります。このスクリプトの正確なファイル名および位置は、使用するオペレーティング・システムによって異なります。

自分のスキーマ内にディメンションを作成する場合は、CREATE PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にファンクションを作成する場合は、CREATE ANY PROCEDURE システム権限が必要です。他のユーザーのスキーマ内のファンクションを置換する場合は、ALTER ANY PROCEDURE システム権限が必要です。

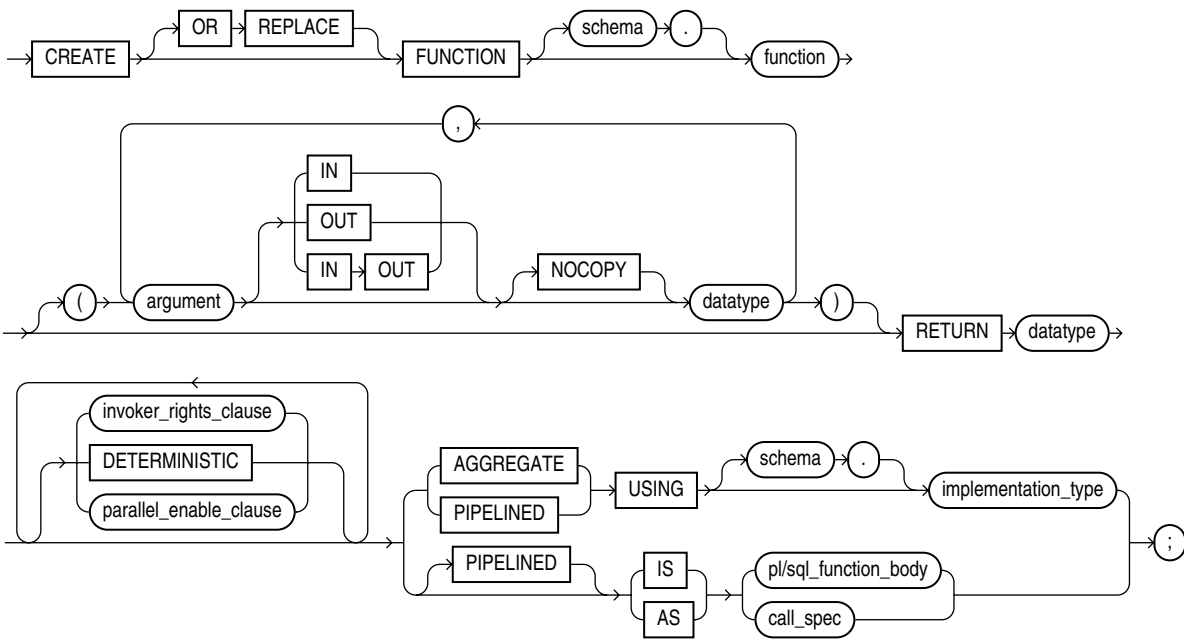
コール仕様を起動する場合、追加権限が必要となることがあります（たとえば、C コール仕様の C ライブラリに対する EXECUTE 権限）。

Oracle プリコンパイラ・プログラム内に CREATE FUNCTION 文を埋め込む場合、キーワード END-EXEC に続けて、各言語の埋込み SQL 文の終了記号を記述して文を終了する必要があります。

参照： このような前提条件の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』または『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。

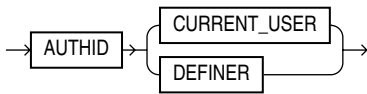
構文

create_function::=

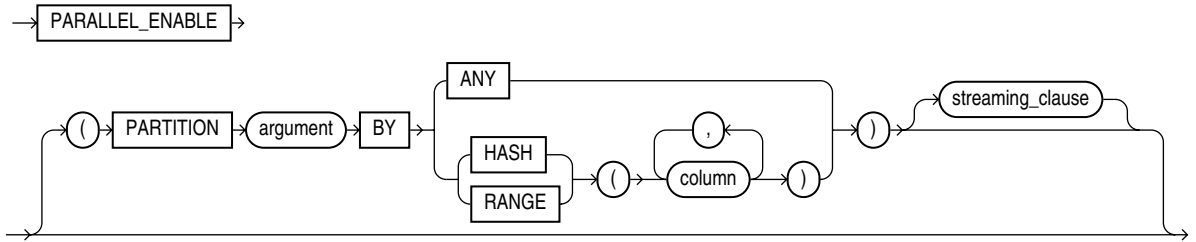


(13-51 ページの「[invoker_rights_clause::=](#)」、13-51 ページの「[parallel_enable_clause::=](#)」を参照)

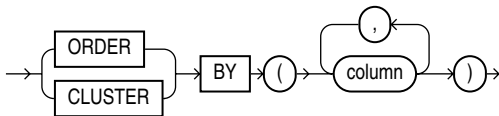
invoker_rights_clause::=



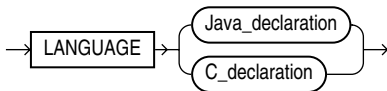
parallel_enable_clause::=



streaming_clause::=



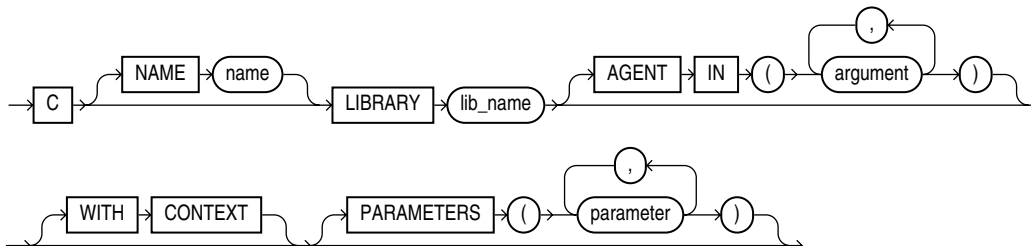
call_spec::=



Java_declaration::=



C_declaration::=



セマンティクス

OR REPLACE

既存のファンクションを再作成する場合は、OR REPLACE を指定します。この句を指定した場合、既存のファンクションに付与されているオブジェクト権限を削除、再作成および再付与しなくても、そのファンクションの定義を変更できます。ファンクションを再定義した場合、そのファンクションは再コンパイルされます。

再定義したファンクションに対して権限が付与されていたユーザーは、権限を再付与されなくても、そのファンクションにアクセスできます。

ファンクション索引がファンクションに依存している場合は、索引に DISABLED のマークを付けます。

参照： ファンクションの再コンパイルについては、9-59 ページの「[ALTER FUNCTION](#)」を参照してください。

schema

ファンクションを定義するスキーマを指定します。*schema* を指定しない場合、現行のスキーマにファンクションが作成されます。

function

作成するファンクションの名前を指定します。コンパイル・エラーでファンクション結果を作成した場合、Oracle はエラーを戻します。SHOW ERRORS コマンドを使用すると、関連するコンパイル・エラー・メッセージを表示できます。

ユーザー定義ファンクションの制限事項：

ユーザー定義ファンクションは、未変更定義を必要とする状況では使用できません。このため、ユーザー定義ファンクションは、次の場所では使用できません。

- CREATE TABLE 文または ALTER TABLE 文の CHECK 制約句
- CREATE TABLE 文または ALTER TABLE 文の DEFAULT 句

また、ファンクションが問合せまたは DML 文内からコールされる場合、そのファンクションには次の制限があります。

- OUT パラメータまたは IN OUT パラメータは指定できません。
- 現行のトランザクションのコミットやロールバック、セーブポイントの作成やロールバックまたはセッションやシステムの変更はできません。DDL 文が明示的に現行のトランザクションをコミットするため、ユーザー定義ファンクションは DDL 文を実行できません。

- ファンクションが **SELECT** 文からコールされる場合のデータベースへの書込みはできません。ただし、**DML** 文で副問合せからコールされるファンクションは、データベースへの書込みができます。
- ファンクションが **DML** 文からコールされる場合、ファンクションがコールされる文で修正される同じ表への書込みはできません。

OUT パラメータおよび **IN OUT** パラメータの制限を除き、**Oracle** は、**SQL** 文から直接コールされるファンクションのみでなく、そのファンクションがコールするすべてのファンクションや、そのファンクションまたはファンクションがコールすることによって実行される **SQL** 文からコールされるファンクションに対しても、これらの制限を適用します。

参照： 13-58 ページの「[ファンクションの作成例 :](#)」を参照してください。

argument

ファンクションへの引数の名前を指定します。ファンクションが引数を受け入れない場合は、ファンクション名の後のカッコを省略できます。

ファンクション引数の制限事項： 集計ファンクションを作成する場合、指定できる引数は 1 つのみです。

IN **IN** は、ファンクションをコールするときに、引数に値を指定する必要があることを示します。これはデフォルトです。

OUT **OUT** は、ファンクションによって引数の値が設定されることを示します。

IN OUT **IN OUT** は、引数の値を、ユーザーが指定することも、ファンクションで設定することも可能であることを示します。

NOCOPY **NOCOPY** は、できるだけ速く引数を渡すように指示します。この句は、**OUT** パラメータや **IN OUT** パラメータに対して、レコード、索引付き表、**VARRAY** などの大きい値を渡す際のパフォーマンスの向上に有効です。（**IN** パラメータ値には、常に **NOCOPY** が渡されます）。

- **NOCOPY** パラメータを指定すると、このパラメータに対応する実際の割当てとしてパッケージ変数が渡された場合に、パッケージ変数に対して行われた割当ては、すぐにこのパラメータに表示されます（または、このパラメータに対して行われた割当ては、すぐにパッケージ変数に表示されます）。
- このパラメータまたは別のパラメータに対して行われた変更は、同じ変数が両方に渡された場合、両方の名前を介してすぐに参照できます。
- プロシージャが未処理例外で終了した場合、このパラメータに対する割当ては、コール元の変数で参照できます。

このような効果がないコールもあります。この効果に問題がない場合にのみ NOCOPY を使用してください。

RETURN 句

ファンクションの戻り値のデータ型を指定します。すべてのファンクションが必ず値を戻すため、この句の指定は必須です。戻り値は、PL/SQL でサポートされているデータ型を持つことができます。

注意： Oracle SQL は、ブール値のパラメータまたは戻り値を持つファンクションのコールをサポートしていません。したがって、ユーザー定義ファンクションが SQL 文からコールされる場合は、数値 (0 または 1) または文字列 ('TRUE' または 'FALSE') を戻すように設計する必要があります。

データ型には、データ長、精度および位取りを指定できません。Oracle では、そのファンクションがコールされた環境から戻り値のデータ長、精度および位取りを導出します。

戻り型が SYS.AnyDataSet であり、問合せの FROM 句でファンクションを使用する場合、PIPELINED 句を指定し、ファンクションの実装タイプの一部として定義メソッド (ODCITableDescribe) を定義する必要があります。

参照：

- PL/SQL のデータ型については、『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。
- ODCITableDescribe ファンクションの定義については、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

invoker_rights_clause

invoker_rights_clause によって、ファンクションを、スキーマを所有するユーザーの権限でそのスキーマ内で実行するか、または CURRENT_USER の権限でそのスキーマ内で実行するかを指定できます。

この句は、問合せ、DML 操作およびファンクションにおける動的 SQL 文の外部名の変換方法も定義します。

AUTHID 句

- ファンクションを CURRENT_USER 権限で実行する場合は、CURRENT_USER を指定します。この句は**実行者権限ファンクション**を作成します。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を CURRENT_USER のスキーマで変換することも指定します。他のすべての文の外部名は、ファンクションを含むスキーマで変換します。

- ファンクションを含むスキーマの所有者権限でファンクションを実行する場合、およびファンクションを含むスキーマで外部名を変換する場合は、**DEFINER** を指定します。これはデフォルトであり、**定義者権限ファンクション**を作成します。

参照：

- **CURRENT_USER** の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 『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。

DETERMINISTIC 句

同じ値の引数でコールされた場合にファンクションが同じ結果値を戻すようにするには、**DETERMINISTIC** を指定します。

ファンクション索引の式で、または **REFRESH FAST** か **ENABLE QUERY REWRITE** というマークが付いているマテリアライズド・ビューの間合せからファンクションをコールするには、このキーワードを指定する必要があります。これらのコンテキストのいずれかに **DETERMINISTIC** ファンクションがあると、ファンクションを再実行せずに、前の計算結果が使用されます。

パッケージ変数を使用するファンクション、またはファンクションが戻す結果に影響する方法でデータベースにアクセスするファンクションを定義する場合は、この句を指定しないでください。Oracle でファンクションが再実行されない場合、この句を指定しなかった結果は取得されません。

次のセマンティック規則は、**DETERMINISTIC** 句の使用を制御します。

- トップレベル・サブプログラム **DETERMINISTIC** を宣言できる。
- パッケージ仕様部ではパッケージレベル・サブプログラム **DETERMINISTIC** を宣言できるが、パッケージ本体では宣言できない。
- (別のサブプログラム内またはパッケージ本体内で宣言される) プライベート・サブプログラムでは **DETERMINISTIC** を宣言できない。
- **DETERMINISTIC** サブプログラムは、コールされたプログラムが **DETERMINISTIC** を宣言しているかどうかにかかわらず、別のサブプログラムをコールできる。

参照：

- マテリアライズド・ビューについては、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- ファンクション索引の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。

parallel_enable_clause

PARALLEL_ENABLE は、パラレル問合せ操作のパラレル実行サーバーからファンクションを実行するための最適化ヒントを指定します。パッケージ変数などの変数は、パラレル実行サーバー内で共有されないことがあるため、ファンクションはそのようなセッション状態を使用してはいけません。

- オプションの PARTITION argument BY 句は、REF CURSOR 型を引数とするファンクションのみに使用できます。REF CURSOR 引数からファンクションへの入力のパパーティション化を定義できます。

ファンクションへの入力のパパーティション化は、ファンクションが表ファンクションとして使用される場合（問合せの FROM 句内）の問合せのパラレル化の方法に影響します。ANY を指定すると、データがパラレル実行サーバー間でランダムにパパーティション化されます。また、指定した列リストに RANGE または HASH パパーティション化を指定できます。

- オプションの streaming_clause を指定すると、指定した列リストでパラレル処理を順序付けまたはクラスタ化できます。
 - ORDER BY を指定すると、パラレル実行サーバーの行がローカルで順序付けされる必要があります。
 - CLUSTER BY を指定すると、パラレル実行サーバーの行が column_list で指定したキーと同じ値である必要があります。

これらのオプションのすべての句で指定した列は、ファンクションの REF CURSOR 引数によって戻された列を参照します。

参照： ユーザー定義の集計ファンクションの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』、『Oracle9i Data Cartridge Developer's Guide』および『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。

PIPELINED 句

PIPELINED を使用すると、**表ファンクション**の結果を繰り返し戻すよう、Oracle に指示できます。表ファンクションは、コレクション型（ネストした表または VARRAY）を戻します。問合せの FROM 句のファンクション名の前に TABLE キーワードを使用し、表ファンクションを問い合わせます。たとえば、次のようにします。

```
SELECT * FROM TABLE(function_name(...))
```

ファンクションによって生成されたものとして、行が戻されます。

- キーワード PIPELINED を単独で指定する場合（PIPELINED IS ...）、PL/SQL ファンクションの本体は PIPE キーワードを使用する必要があります。このキーワードによってコレクション全体を 1 つの値として戻すかわりに、ファンクション外のコレクションの 1 つの要素を戻すように Oracle に指示します。

- 操作の起動、フェッチおよび停止を含むインタフェースを事前に定義する場合、PIPELINED USING *implementation_type* 句を指定できます。実装タイプは、ODCI Table インタフェースを実装し、表ファンクションの作成時に存在する必要があります。この句は、C++ や Java などの外部言語で実装される表のファンクションに便利です。

ファンクションの戻り型が SYS.AnyDataSet である場合、ファンクションの実装タイプの一部として定義メソッド (ODCI Table Describe) も定義する必要があります。

参照：

- 表ファンクションの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- ODCI ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

AGGREGATE USING 句

AGGREGATE USING を指定すると、このファンクションは集計ファンクション、または行のグループを検証して単一の行を戻すファンクションとして識別されます。SELECT 構文のリスト、HAVING 句および ORDER BY 句で集計ファンクションを指定できます。

注意： 問合せでユーザー定義の集計ファンクションを指定する場合、**分析ファンクション**（問合せ結果セットを操作するファンクション）として使用できます。その場合は、組込み分析ファンクションを使用可能にする OVER *analytic_clause* 構文を使用します。構文およびセマンティクスの詳細は、6-9 ページの「**分析ファンクション**」を参照してください。

USING 句で、ファンクションの実装タイプの名前を指定します。実装タイプは、ODCI 集計ルーチンの実装を含むオブジェクト型である必要があります。*schema* を指定しない場合、実装タイプが自分のスキーマ内にあるとみなされます。

集計ファンクション作成の制限事項： この句を指定すると、ファンクションに対して入力引数を 1 つのみ指定できます。

参照： ODCI ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』および 13-59 ページの「**集計ファンクションの作成例：**」を参照してください。

IS | AS 句

この句の適切な部分を使用して、ファンクションの本体を宣言します。

pl/sql_subprogram_body *pl/sql_subprogram_body*を使用すると、PL/SQL サブプログラム本体でファンクションを宣言できます。

参照： PL/SQL サブプログラムの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』および 13-60 ページの「[ファンクションでのパッケージ・プロシージャの使用例](#)」を参照してください。

call_spec *call_spec*を使用して、Java メソッドまたは C 関数名、パラメータ・タイプおよび戻り型を SQL で相当するものにマップします。*Java_declaration*では、'string' が JAVA 実装メソッドを定義します。

参照：

- 『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- パラメータおよび *C_declaration* のセマンティクスの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

AS EXTERNAL AS EXTERNAL は、C 関数を宣言するもう 1 つの方法です。この句は以前のリリースのもので、下位互換用にのみサポートされています。AS LANGUAGE C 構文を使用することをお勧めします。

例

ファンクションの作成例： 次の文は、サンプル表 `oe.orders` にファンクション `get_bal` を作成します。

```
CREATE FUNCTION get_bal(acc_no IN NUMBER)
RETURN NUMBER
IS acc_bal NUMBER(11,2);
BEGIN
    SELECT order_total
    INTO acc_bal
    FROM orders
    WHERE customer_id = acc_no;
    RETURN(acc_bal);
END;
/
```

`get_bal` ファンクションを実行した場合、指定された預金口座の残高が戻ります。

このファンクションをコールする際は、残高を確認する口座番号 `acc_no` を引数として指定する必要があります。`acc_no` のデータ型は `NUMBER` です。

このファンクションでは、口座の残高が戻ります。`CREATE FUNCTION` 文の `RETURN` 句は、戻り値のデータ型が `NUMBER` であることを示しています。

このファンクションでは、`SELECT` 文によって、`order` 表の引数 `acc_no` で特定される行から `balance` 列が選択されます。また、`RETURN` 文によって、ファンクションがコールされる環境にこの値が戻ります。

この例で作成されたファンクションは、`SQL` 文の中で使用できます。たとえば、次のようにします。

```
SELECT get_bal(165) FROM DUAL;
```

```
GET_BAL(165)
-----
          2519
```

次の文では、`C` ルーチン `c_get_val` を外部ファンクションとして登録する `PL/SQL` スタンドアロン・ファンクション `get_val` が作成されます（この例では、パラメータは省略されています）。

```
CREATE FUNCTION get_val
( x_val IN NUMBER,
  y_val IN NUMBER,
  image IN LONG RAW )
RETURN BINARY_INTEGER AS LANGUAGE C
NAME "c_get_val"
LIBRARY c_utils
PARAMETERS (...);
```

集計ファンクションの作成例： 次の文は、数値を集計する `SecondMax` という名前の集計ファンクションを作成します。オブジェクト型 `SecondMaxImpl` ルーチンが `ODCIAggregate` ルーチンの実装を含んでいるものとします。

```
CREATE FUNCTION SecondMax (input NUMBER) RETURN NUMBER
PARALLEL_ENABLE AGGREGATE USING SecondMaxImpl;
```

参照： `SecondMaxImpl` の型および型本体の実装の詳細は、『*Oracle9i Data Cartridge Developer's Guide*』を参照してください。

サンプル表 `hr.employees` を問い合わせる次の文のように、問合せで集計ファンクションを使用します。

```
SELECT SecondMax(salary), department_id
FROM employees
GROUP BY department_id
HAVING SecondMax(salary) > 9000;
```

SECONDMAX (SALARY)	DEPARTMENT_ID
13500	80
17000	90

ファンクションでのパッケージ・プロシージャの使用例： 次の文は、DBMS_LOB プロシージャを使用して CLOB 列の長さを戻すファンクションを作成します。

```
CREATE OR REPLACE FUNCTION text_length(a CLOB)
  RETURN NUMBER DETERMINISTIC IS
BEGIN
  RETURN DBMS_LOB.GETLENGTH(a);
END;
```

参照： ファンクション索引を作成するファンクションの使用方法は、13-83 ページの「[LOB 列のファンクション索引の例:](#)」を参照してください。

CREATE INDEX

用途

CREATE INDEX 文は、次の索引を作成する場合に使用します。

- 表の 1 つ以上の列、パーティション表、索引構成表またはクラスタ
- 表またはクラスタの 1 つ以上のスカラー型オブジェクト属性
- ネストした表の列の索引を作成するためのネストした表の記憶表

索引は、スキーマ・オブジェクトの 1 つで、索引には、表またはクラスタの索引付き列の中に表示される各値のエントリが入ります。索引を使用した場合、行に直接、かつ高速にアクセスできます。Oracle は、次の索引をサポートしています。

- 通常の索引（デフォルトでは、B ツリー索引が作成されます。）
- **ビットマップ索引**。キー値に関連付けられた ROWID をビットマップとして格納します。
- **パーティション索引**。表の索引付き列に表示される各値のエントリが入るパーティションを構成します。
- **ファンクション索引**。式をベースとしています。式によって戻される値を評価する問合せを組み立てることができます。その式にはファンクション（組込みまたはユーザー定義）が含まれることがあります。
- **ドメイン索引**。アプリケーション固有の *indextype* 索引タイプのインスタンスです。

参照：

- 索引については、『Oracle9i データベース概要』を参照してください。
- 9-61 ページの「[ALTER INDEX](#)」を参照してください。
- 16-74 ページの「[DROP INDEX](#)」を参照してください。

前提条件

自分のスキーマ内に索引を作成する場合は、次のいずれかの条件が満たされている必要があります。

- 索引を作成する表またはクラスタが自分のスキーマ内に定義されている。
- 索引を作成する表に対する INDEX オブジェクト権限がある。
- CREATE ANY INDEX システム権限がある。

他のユーザーのスキーマ内に索引を作成する場合は、CREATE ANY INDEX システム権限が必要です。また、索引が定義されているスキーマの所有者には、UNLIMITED TABLESPACE システム権限、あるいはその索引または索引パーティションを格納するための表領域の割当て制限のいずれかが必要です。

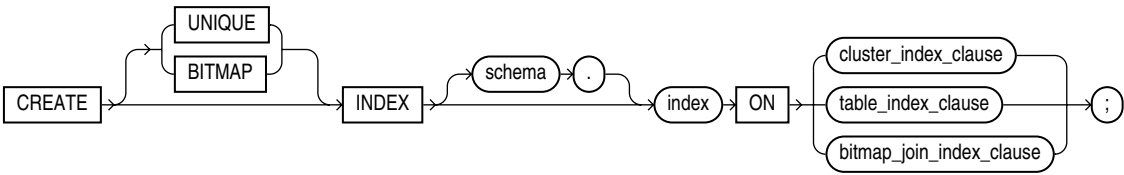
自分のスキーマにドメイン索引を作成する場合、従来索引の作成の前提条件の他に、索引タイプについての EXECUTE 権限が必要です。他のユーザーのスキーマにドメイン索引を作成する場合、索引の所有者にも索引タイプおよびその基礎となる実装タイプに EXECUTE 権限が必要です。ドメイン索引を作成する前に、索引タイプを定義する必要があります。

自表の自分のスキーマにファンクション索引を作成する場合、従来索引の作成の前提条件の他に、QUERY REWRITE システム権限が必要です。別のスキーマまたは別のスキーマの表に索引を作成する場合は、GLOBAL QUERY REWRITE 権限が必要です。どちらの場合も、表の所有者は、ファンクション索引で使用するファンクションについての EXECUTE オブジェクト権限が必要です。また、問合せでファンクション索引を使用する場合は、QUERY_REWRITE_ENABLED パラメータを TRUE に、QUERY_REWRITE_INTEGRITY パラメータを TRUSTED に設定する必要があります。

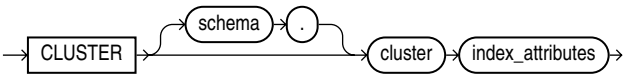
参照： 13-88 ページの「CREATE INDEXTYPE」を参照してください。

構文

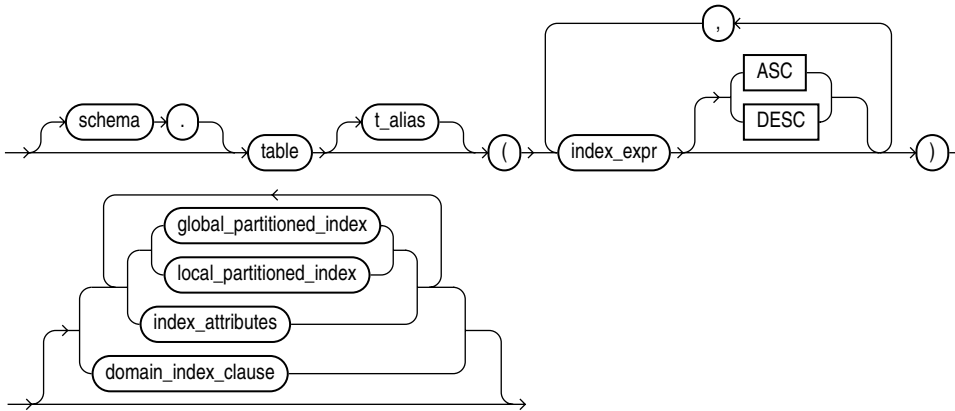
create_index::=



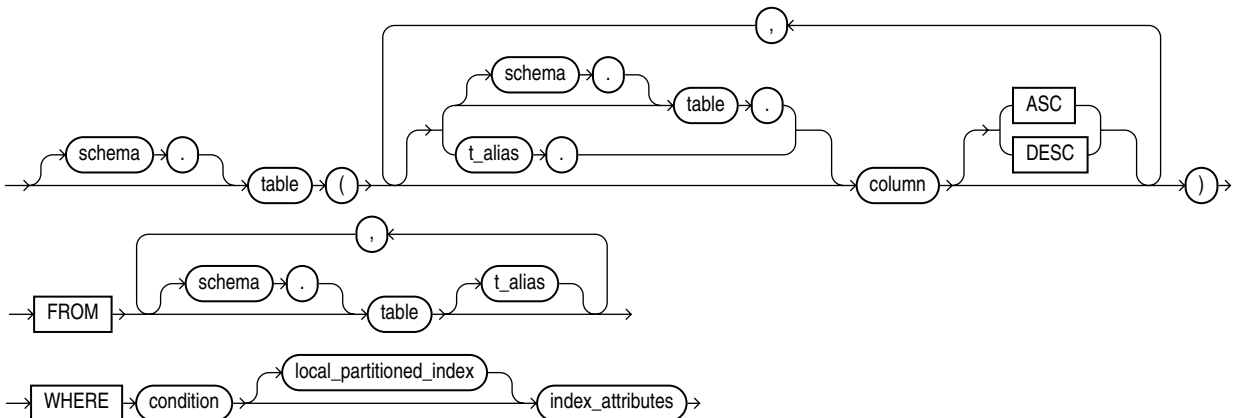
cluster_index_clause::=



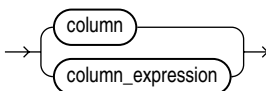
(13-64 ページの「index_attributes::=」を参照)

table_index_clause::=

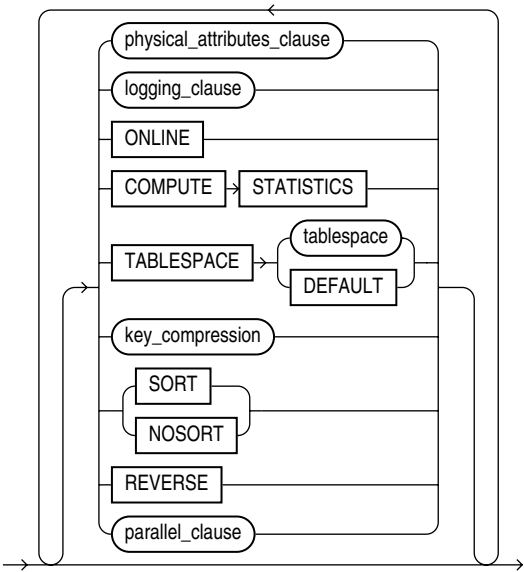
(13-65 ページの「[global_partitioned_index::=](#)」、13-65 ページの「[local_partitioned_index::=](#)」、13-64 ページの「[index_attributes::=](#)」、13-65 ページの「[domain_index_clause::=](#)」を参照)

bitmap_join_index_clause::=

(13-65 ページの「[local_partitioned_index::=](#)」、13-64 ページの「[index_attributes::=](#)」を参照)

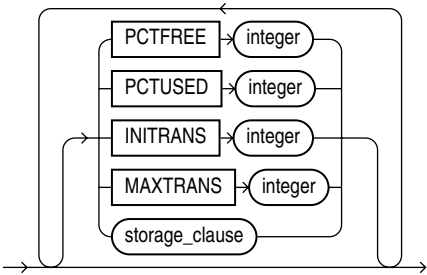
index_expr::=

index_attributes::=



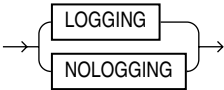
(13-64 ページの「[physical_attributes_clause::=](#)」、13-64 ページの「[logging_clause::=](#)」、13-65 ページの「[key_compression::=](#)」、13-67 ページの「[parallel_clause::=](#)」を参照)

physical_attributes_clause::=

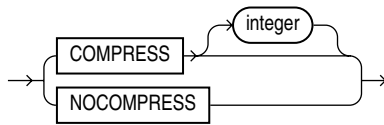


(7-53 ページの「[storage_clause::=](#)」を参照)

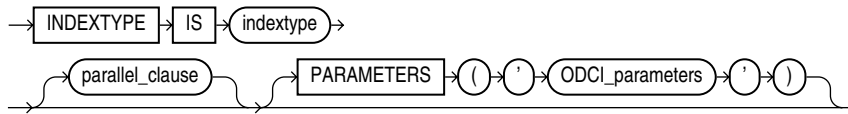
logging_clause::=



key_compression::=

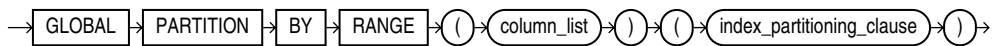


domain_index_clause::=



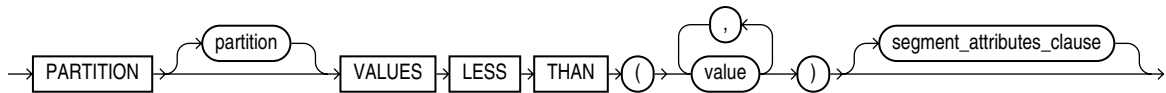
(13-67 ページの「[parallel_clause::=](#)」を参照)

global_partitioned_index::=



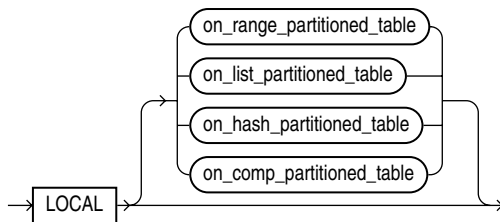
(13-65 ページの「[index_partitioning_clause::=](#)」を参照)

index_partitioning_clause::=

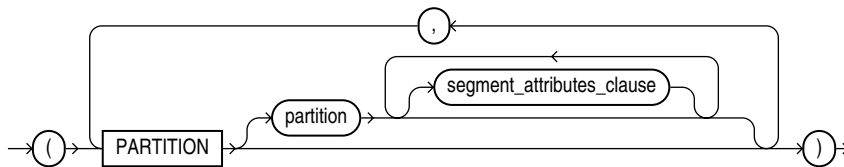


(13-66 ページの「[segment_attributes_clause::=](#)」を参照)

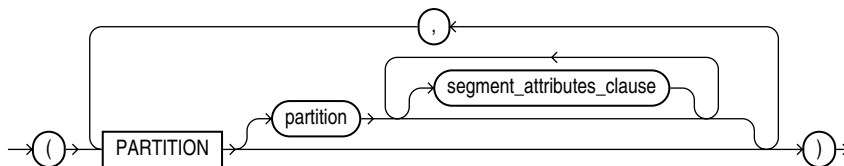
local_partitioned_index::=



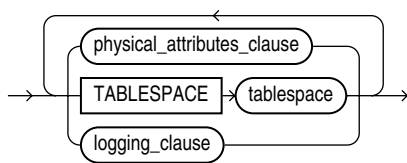
(13-66 ページの「[on_range_partitioned_table::=](#)」、13-66 ページの「[on_list_partitioned_table::=](#)」、13-66 ページの「[on_hash_partitioned_table::=](#)」、13-67 ページの「[on_comp_partitioned_table::=](#)」を参照)

on_range_partitioned_table::=

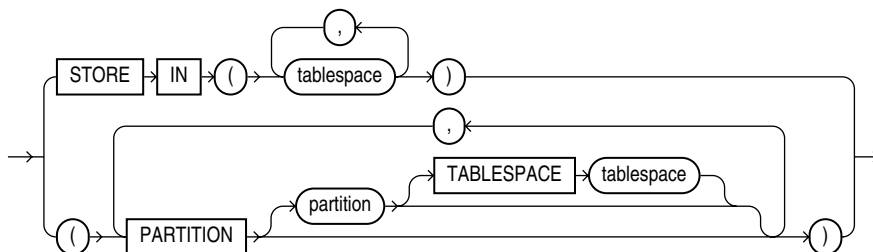
(13-66 ページの「[segment_attributes_clause::=](#)」を参照)

on_list_partitioned_table::=

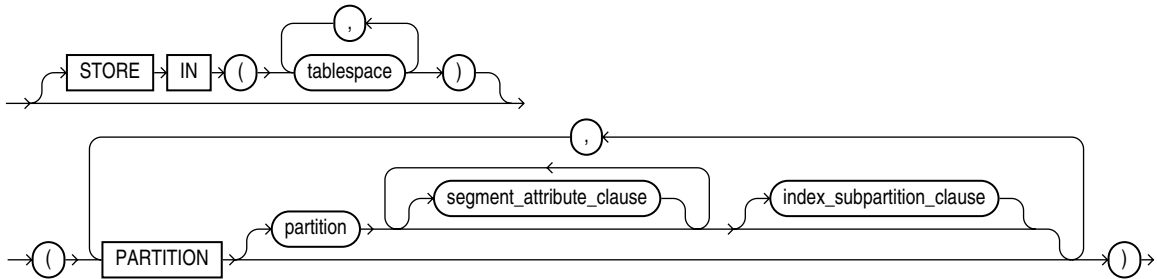
(13-66 ページの「[segment_attributes_clause::=](#)」を参照)

segment_attributes_clause::=

(13-64 ページの「[physical_attributes_clause::=](#)」、13-64 ページの「[logging_clause::=](#)」を参照)

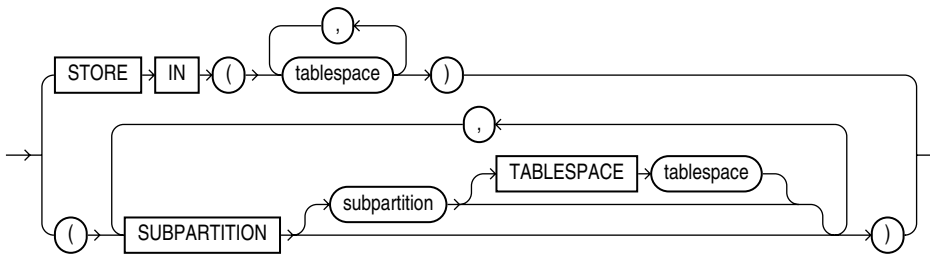
on_hash_partitioned_table::=

on_comp_partitioned_table::=

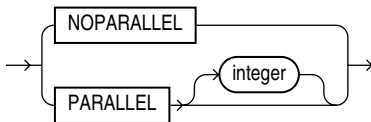


(13-66 ページの「[segment_attributes_clause::=](#)」、13-67 ページの「[index_subpartition_clause::=](#)」を参照)

index_subpartition_clause::=



parallel_clause::=



(7-51 ページの「[storage_clause](#)」を参照)

セマンティクス

UNIQUE

UNIQUE によって、索引のベースとなっている列の値が一意である必要があることを指定します。索引がローカル非同一キー索引 ([local partitioned_index](#) を参照) の場合、索引キーはパーティション・キーを含んでいる必要があります。

一意索引の制限事項：

- UNIQUE および BITMAP は同時に指定できません。
- ドメイン索引には UNIQUE を指定できません。

参照： 整合性制約の詳細は、7-5 ページの「[constraints](#)」を参照してください。

BITMAP

BITMAP によって、`index` が各行を分割した索引付けではなく、各個別キーのビットマップで作成されることを指定します。ビットマップ索引では、キー値にビットマップとして関連付けられた ROWID が格納されます。ビットマップ内の各ビットは使用可能な ROWID に対応しているため、ビットが設定されていれば、それに対応する ROWID を持つ行に、キー値が設定されていることになります。ビットマップの内部表現は、データ・ウェアハウスなど、低レベルの同時実行トランザクションが実行されるアプリケーションに最適です。

注意： Oracle では、ビットマップ索引の場合を除き、すべてのキー列が NULL である表の行には索引を作成しません。したがって、表のすべての行に索引を作成する場合、索引のキー列に NOT NULL 制約を指定するか、またはビットマップ索引を作成する必要があります。

ビットマップ索引の制限事項：

- グローバル・パーティション索引の作成には BITMAP を指定できません。
- 索引構成表がビットマップ化した 2 次索引に対応するマッピング表を持たない場合、索引構成表にビットマップ化した 2 次索引を作成できません。
- UNIQUE および BITMAP は同時に指定できません。
- ドメイン索引には BITMAP を指定できません。

参照：

- ビットマップ索引の使用の詳細は、『Oracle9i データベース概要』および『Oracle9i データベース・パフォーマンス・チューニング・ガイド およびリファレンス』を参照してください。
- マッピング表の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。
- 13-86 ページの「[ビットマップ索引の例：](#)」を参照してください。

schema

作成する索引が定義されるスキーマを指定します。*schema* を指定しないと、自分のスキーマ内に索引が作成されます。

index

作成する索引の名前を指定します。

参照： 13-81 ページの「[索引の作成例：](#)」および 13-82 ページの「[XMLType 表の索引の作成例：](#)」を参照してください。

cluster_index_clause

cluster_index_clause を使用して、クラスタ索引が作成されるクラスタを識別します。*cluster* を *schema* で修飾しない場合、そのクラスタが自分のスキーマ内に定義されているとみなされます。ハッシュ・クラスタにはクラスタ索引を作成できません。

参照： 13-2 ページの「[CREATE CLUSTER](#)」および 13-82 ページの「[クラスタ索引の作成例：](#)」を参照してください。

table_index_clause

索引を定義している表（およびその属性）を指定します。指定する *table* を *schema* で修飾しない場合は、その表が自分のスキーマに定義されているとみなされます。

ネストした表の記憶表に索引を作成することによって、ネストした表の列に索引を作成します。記憶表の *NESTED_TABLE_ID* 疑似列を組み込んだ一意索引を作成することは、ネストした表の値を持つ行がそれぞれ確実に異なるようにする有効な手段です。

参照： 13-86 ページの「[ネストした表の索引の例：](#)」を参照してください。

***table_index_clause* の制限事項：**

- ローカル・パーティション索引の場合は、*table* をパーティション化する必要があります。
- 索引構成表の場合、この文は 2 次索引を作成します。この 2 次索引には、*REVERSE* を指定できません。索引キーおよび論理 *ROWID* の結合サイズは、ブロック・サイズの半分未満にする必要があります。
- *table* が一時表の場合、索引も *table* と同様の有効範囲（セッションまたはトランザクション）を持つ一時的なものとなります。一時表の索引には、次の制限があります。
 - 索引は、パーティション索引またはドメイン索引であってははいけません。
 - *physical_attributes_clause* または *parallel_clause* は指定できません。
 - *LOGGING*、*NOLOGGING* または *TABLESPACE* は指定できません。

注意： セッションがバインドされていない場合にのみ、一時表で DDL 操作 (ALTER TABLE、DROP TABLE、CREATE INDEX など) を実行できます。セッションを一時表にバインドするには、一時表で INSERT 操作を実行します。セッションを一時表からアンバインドするには、TRUNCATE 文を発行するか、セッションを終了します。また、トランザクション固有の一時表からアンバインドするには、COMMIT または ABORT 文を発行します。

参照： 一時表の詳細は、15-7 ページの「[CREATE TABLE](#)」および『Oracle9i データベース概要』を参照してください。

t_alias

索引を作成する表に対して相関名 (別名) を指定します。

注意： *index_expr* がオブジェクト型属性またはオブジェクト型メソッドを参照する場合、この別名が必要になります。13-84 ページの「[型メソッドのファンクション索引の作成例 :](#)」および 13-86 ページの「[置換可能な列の索引の例 :](#)」を参照してください。

index_expr

index_expr は、索引に基づく列または列の式を指定します。

column 表内の列の名前を指定します。ビットマップ索引には最大 30 列まで指定できます。他の索引には最大 32 列まで指定できます。

スカラー・オブジェクト属性列またはネストした表の記憶表のシステム定義の NESTED_TABLE_ID 列には索引を作成できます。オブジェクト属性列を指定する場合、列名を表名で修飾する必要があります。ネストした表の列属性を指定する場合は、この属性は、一番外側の表の名前、ネストした表が定義されている列の名前、およびそのネストした表の列属性となるすべての中間属性の名前で修飾する必要があります。

索引列の制限事項： SCOPE 句で定義されている REF 型の列または属性の索引が Oracle でサポートされている場合を除き、ユーザー定義型、LONG、LONG RAW、LOB または REF 型の列または属性に索引を作成できません。

column_expression *table* の列、定数、SQL ファンクションおよびユーザー定義ファンクションの列から作成された式を指定します。*column_expression* を指定した場合、ファンクション索引が作成されます。

参照： 13-71 ページの「[ファンクション索引における注意事項](#)」、13-72 ページの「[ファンクション索引の制限事項](#)」 および 13-83 ページの「[ファンクション索引の例](#)」を参照してください。

ファンクションの名前解決は、索引作成者のスキーマに基づきます。*column_expression* で使用されるユーザー定義ファンクションは、CREATE INDEX 操作中に完全に名前解決されます。

ファンクション索引の作成後、ANALYZE 文を使用して索引とその基になる表の両方に関する統計項目を収集します。この統計項目が生成されるまで、ファンクション索引は使用できません。

参照： 12-32 ページの「[ANALYZE](#)」を参照してください。

ファンクション索引における注意事項：

- ファンクション索引を使用する表を問い合わせる場合、その問合せで *column_expression* が NULL でないことを確認する必要があります。ただし、WHERE 句に指定した列の順序が、ファンクション索引を定義した *column_expression* での順序と異なる場合でも、問合せにファンクション索引が使用されます。

参照： 13-83 ページの「[ファンクション索引の例](#)」を参照してください。

- 索引の基になるファンクションが無効または削除された場合は、索引に DISABLED がマークが付けられます。オプティマイザが索引の使用を選択した場合、DISABLED 索引の問合せは失敗します。索引が UNUSABLE にマークが付けられ、パラメータ SKIP_UNUSABLE_INDEXES が true に設定された場合を除き、DISABLED 索引の DML 操作は失敗します。

参照： このパラメータの詳細は、10-2 ページの「[ALTER SESSION](#)」を参照してください。

- ファンクション索引の使用も、QUERY_REWRITE_ENABLED セッション・パラメータの設定による影響を受けます。

参照： 10-2 ページの「[ALTER SESSION](#)」を参照してください。

- ファンクション、パッケージまたは型のパブリック・シノニムが `column_expression` で使用され、後で同じ名前の実際のオブジェクトが表の所有者のスキーマに作成された場合、ファンクション索引は使用禁止になります。その後、`ALTER INDEX ... ENABLE` または `ALTER INDEX ... REBUILD` を使用してファンクション索引を使用可能にした場合、`column_expression` で使用されているファンクション、パッケージまたは型は、パブリック・シノニムが最初に指定されたファンクション、パッケージまたは型への変換を続けます。新しいファンクション、パッケージまたは型への変換は行われません。
- ファンクション索引の定義によって文字データに内部変換が生成される場合に、NLS パラメータの設定を変更するときは注意が必要です。ファンクション索引は、NLS パラメータの現行のデータベース設定を使用します。セッション・レベルでパラメータを再設定した場合、ファンクション索引を使用して問合せを行うと、無効な結果が戻る場合があります。2つの照合パラメータ (NLS_SORT および NLS_COMP) は例外です。これらがセッション・レベルで再設定された場合でも、Oracle は正常に変換を処理します。

ファンクション索引の制限事項：

- `column_expression` で参照されるユーザー定義ファンクションは、DETERMINISTIC である必要があります。
- グローバル・パーティション・ファンクション索引では、`column_expression` がパーティション・キーであってははいけません。
- `column_expression` には、スカラー副問合せ式を除く式のすべての形式が可能です。
- パラメータがない場合も、すべてのファンクションをカッコで指定する必要があります。カッコで指定していない場合は、列名として解析されます。
- `column_expression` で指定するファンクションは、リピータブル値を戻す必要があります。たとえば、SYSDATE や USER ファンクションまたは ROWNUM 疑似列は指定できません。
- `column_expression` に集計ファンクションを含めることはできません。
- ネストした表にはファンクション索引を作成できません。

参照： 13-49 ページの「[CREATE FUNCTION](#)」および『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。

ASC | DESC

ASC または DESC を使用して、索引を昇順で作成するか降順で作成するかを指定します。文字データの索引は、データベース・キャラクタ・セットの文字値の昇順または降順で作成されます。

Oracle は、降順索引をファンクション索引として扱います。索引作成に QUERY REWRITE または GLOBAL QUERY REWRITE は必要ありません。ただし、他のファンクション索引のように、最初に索引および索引が定義されている表を分析するまで、降順索引は使用しません。この文の `column_expression` 句を参照してください。

昇順索引および降順索引の制限事項： これらの句は、ドメイン索引に対して指定できません。逆索引には DESC を指定できません。index をビットマップ化したり、COMPATIBLE 初期化パラメータに 8.1.0 未満の値を設定すると、DESC は無視されます。

index_attributes

index_attributes の句を使用して、索引の属性を指定します。

physical_attributes_clause physical_attributes_clause を使用して、索引における物理特性および記憶特性の値を設定します。

この句を省略すると、次のデフォルト値が使用されます。

- PCTFREE: 10
- INITTRANS: 2
- MAXTRANS: データ・ブロックのサイズに依存

索引の物理属性の制限事項： PCTUSED パラメータは、索引に対して指定できません。

参照：

- この句のパラメータの詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- デフォルト値を含む記憶域パラメータの詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

TABLESPACE tablespace には、索引、索引パーティションまたは索引サブパーティションを格納する表領域の名前を指定します。この句を指定しない場合、その索引を定義しているスキーマの所有者のデフォルトの表領域内に索引が作成されます。

ローカル索引の場合、tablespace のかわりにキーワード DEFAULT を指定できます。ローカル索引に追加される新規パーティションまたはサブパーティションは、基礎となる表の対応するパーティションまたはサブパーティションと同じ表領域内に作成されます。

COMPRESS COMPRESS を指定すると、キー圧縮が使用可能になります。これによって、キー列値の繰返しがなくなり、記憶域を削減できます。integer を使用して、接頭辞の長さ（圧縮する接頭辞列数）を指定します。

- 一意索引の場合、接頭辞の長さの有効範囲は、1 ～（キー列の数 - 1）です。デフォルトの接頭辞の長さは、（キー列の数 - 1）です。
- 一意でない索引の場合、接頭辞の長さの有効範囲は、1 ～キー列の数です。デフォルトの接頭辞の長さはキー列数です。

非パーティション索引（一意でない索引または 2 列以上の一意索引）のみを圧縮します。

キー圧縮の制限事項： COMPRESS は、ビットマップ索引に対して指定できません。

参照： 13-81 ページの「[索引の圧縮例：](#)」を参照してください。

NOCOMPRESS NOCOMPRESS を指定すると、キー圧縮が使用禁止になります。これはデフォルト値です。

SORT | NOSORT 索引の作成時に、Oracle はデフォルトで索引を昇順ソートします。NOSORT を指定すると、行がデータベース内に昇順で格納されるため、Oracle は索引の作成時に行のソートを行う必要がありません。索引列の行または列が昇順に格納されていない場合、Oracle はエラーを戻します。ソート時間および領域を削減するため、列を表へ初期ロードした直後にこの句を使用します。これらのキーワードのいずれも指定しない場合、デフォルトで SORT が使用されます。

NOSORT の制限事項：

- この句は、REVERSE と同時には指定できません。
- この句を使用して、クラスタ索引、パーティション索引またはビットマップ索引を作成することはできません。
- 索引構成表の 2 次索引には、この句を指定できません。

REVERSE REVERSE を指定すると、ROWID 以外の索引ブロックのバイトが逆順に格納されます。

逆索引の制限事項：

- この句は、NOSORT と同時には指定できません。
- ビットマップ索引または索引構成表は逆順には格納できません。

logging_clause 索引作成を、REDO ログ・ファイル内に記録する (LOGGING) か記録しない (NOLOGGING) かを指定します。この設定によって、索引に対する後続のダイレクト・ローダー (SQL*Loader) およびダイレクト・パス・インサート操作を記録するか記録しないかも決定します。デフォルトは LOGGING です。

非パーティション索引の場合、この句は索引のロギング属性を指定します。

パーティション索引の場合、この句は次のことを決定します。

- CREATE 文で指定されたすべてのパーティションのデフォルト値 (PARTITION 記述句で *logging_clause* を指定する場合を除く)
- 索引パーティションに関連付けられたセグメントに対するデフォルト値
- 後続の ALTER TABLE ... ADD PARTITION 操作中に明示的に追加されたローカル索引パーティションまたはサブパーティションに対するデフォルト値

索引のロギング属性は、その実表の属性に依存しません。

この句を指定しない場合、ロギング属性は表が存在する表領域の属性になります。

参照：

- この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。
- ロギングおよびパラレル DML の詳細は、『Oracle9i データベース概要』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- 13-82 ページの「[NOLOGGING モードの索引の作成例](#)」を参照してください。

ONLINE ONLINE によって、索引作成中に表での DML 操作ができることを指定します。

索引作成の制限事項：

- オンラインで索引を作成中は、パラレル DML はサポートされません。ONLINE を指定し、パラレル DML 文を発行すると、Oracle はエラーを戻します。
- ビットマップ索引またはクラスタ索引には、ONLINE を指定できません。
- UROWID 列の従来索引には、ONLINE を指定できません。
- 索引構成表の一意索引の場合、索引構成表内の索引キー列の数と主キー列の数の合計は、32 以下にする必要があります。

参照： オンラインでの索引作成および再作成については、『Oracle9i データベース概要』を参照してください。

COMPUTE STATISTICS COMPUTE STATISTICS によって、索引作成中の統計情報収集を指定します。これらの統計情報は、SQL 文の実行計画を選択する際に、オプティマイザによって使用中のデータ・ディクショナリに格納されます。

収集された統計情報のタイプは、作成する索引タイプによって異なります。

注意： 表のかわりに別の索引を使用して索引を作成する場合、元の索引は適切な統計情報を提供しない場合があります。このため、一般的に基となる表を使用して統計を計算します。その結果、統計は改善されますが、パフォーマンスが低下することがあります。

PL/SQL パッケージおよびプロシージャでは、その他の方法で統計を収集できます。

参照：『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』および 13-82 ページの「統計情報の計算例:」を参照してください。

parallel_clause

索引の作成をパラレル化する場合は、*parallel_clause* を指定します。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である**並列度**を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

Index Partitioning 句

global_partitioned_index 句および *local_partitioned_index* 句を使用すると、*index* をパーティション化できます。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

参照： 13-84 ページの「パーティション索引の例」を参照してください。

global_partitioned_index *global_partitioned_index* によって、索引のパーティション化がユーザー定義であり、基礎となる表と同一レベルでパーティション化されないことを指定できます。デフォルトでは、非パーティション索引はグローバル索引です。グローバル索引は、*column_list* に指定された表の列の値の範囲に基づいてパーティション化されます。ローカル索引にこの句は指定できません。

`column_list` では、索引の列リストの左の接頭辞を指定する必要があります。つまり、索引が列 `a`、`b` および `c` に定義されている場合は、`column_list` に `(a,b,c)`、`(a,b)` または `(a,c)` は指定できますが、`(b,c)`、`(c)` または `(b,a)` は指定できません。

グローバル・パーティション索引キーの制限事項：

- `column_list` に指定できるのは 32 列以下です。
- これらの列には、ROWID 類似列または ROWID 型の列は指定できません。

注意： 異なるキャラクタ・セットを使用してデータベースを使用しているか、使用する予定がある場合は、キャラクタ列を分割する際に注意してください。文字のソート順序は、すべてのキャラクタ・セットで同一ではありません。

参照： キャラクタ・セットのサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

index_partitioning_clause この句は、個々の索引パーティションを示すために使用します。この句が繰り返される数によってパーティションの数が決まります。`partition` を指定しない場合、名前は `SYS_Pn` の形式で生成されます。

`VALUES LESS THAN (value_list)` には、グローバル索引の現在のパーティションの上限（境界は含まない）を指定します。値のリストは、`global_partitioned_index` 句の列リストに対応するリテラル値を含む、カンマで区切られた順序リストです。最後のパーティションの値としては、必ず `MAXVALUE` を指定します。

注意： 索引が `DATE` 列でパーティション化されている場合、および日付書式で年の最初の 2 桁の数字が指定されていない場合、年の 4 文字書式マスクを指定して `TO_DATE` ファンクションを使用する必要があります。日付書式は、`NLS_TERRITORY` によって暗黙的に決定されるか、`NLS_DATE_FORMAT` によって明示的に決定されます。

参照：

- これらの初期化パラメータの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。
- 15-69 ページの「レンジ・パーティション化の例:」を参照してください。

local_partitioned_index

local_partitioned_index 句によって、*table* と同じパーティション数および同じパーティション境界を使用し、同じ列で索引をパーティション化できます。Oracle は、基礎となる表が再パーティション化された場合、ローカル索引のパーティションを自動的にメンテナンスします。

on_range_partitioned_table この句を使用すると、レンジ・パーティション表の索引パーティションの名前および属性を指定できます。この句を指定する場合、PARTITION 句は、表パーティションと同一の数と順序である必要があります。*partition* を指定しない場合、対応する表のパーティションと一貫した名前が生成されます。その名前が既存の索引パーティション名と競合する場合は、SYS_Pn の形式が使用されます。

on_list_partitioned_table *on_list_partitioned_table* 句は、13-78 ページの「*on_range_partitioned_table*」と同様です。

on_hash_partitioned_table この句を使用すると、ハッシュ・パーティション表の索引パーティションの名前および表領域の記憶域を指定できます。

PARTITION 句を指定する場合、これらの句の数は表パーティションの数と同一である必要があります。*partition* を指定しない場合、対応する表のパーティションと一貫した名前が生成されます。その名前が既存の索引パーティション名と競合する場合は、SYS_Pn の形式が使用されます。オプションで、1 つ以上の個々のパーティションに表領域の記憶域を指定できます。パーティションまたは STORE IN 句に表領域の記憶域を指定しない場合、同じ表領域の各索引パーティションを、対応する表のパーティションとして格納します。

STORE IN 句によって、すべての索引ハッシュ・パーティションを分散する複数の表領域を指定できます。表領域の数は、索引パーティションの数と等しくする必要はありません。索引パーティションの数が表領域の数より多い場合、表領域名を介して循環します。

on_comp_partitioned_table この句を使用すると、コンポジット・パーティション表の索引パーティションの名前および表領域の記憶域を指定できます。

STORE IN 句は、レンジ・ハッシュ・コンポジット・パーティション表に対してのみ有効です。この句によって、すべての索引ハッシュ・サブパーティションを分散する 1 つ以上のデフォルトの表領域を指定できます。*index_subpartition_clause* の第 2 STORE IN 句で個々のパーティションのサブパーティションに対して異なる表領域の記憶域を指定すると、この記憶域を上書きできます。

レンジ・リスト・コンポジット・パーティション表に対しては、デフォルトの表領域を PARTITION 句のリスト・サブパーティションに指定できます。

index_subpartition_clause の SUBPARTITION 句の個々のパーティションのリスト・サブパーティションに異なる表領域の記憶域を指定すると、この記憶域を上書きできます。

index_subpartition_clause この句を使用すると、コンポジット・パーティション表の索引サブパーティションに名前および表領域の記憶域を指定できます。

STORE IN 句は、レンジ・ハッシュ・コンポジット・パーティション表のハッシュ・サブパーティションに対してのみ有効です。この句によって、すべての索引ハッシュ・サブパーティションを分散する 1 つ以上の表領域を指定できます。SUBPARTITION 句は、レンジ・ハッシュおよびレンジ・リスト・コンポジット・パーティション表に対してのみ有効です。

SUBPARTITION 句を指定する場合、これらの句の数は表サブパーティションの数と同一である必要があります。subpartition を指定しない場合、対応する表のサブパーティションと一貫した名前が生成されます。その名前が既存の索引サブパーティション名と競合する場合は、SYS_SUBPn の形式が使用されます。

表領域の数は、索引サブパーティションの数と等しい必要はありません。索引サブパーティションの数が表領域の数より多い場合は、表領域名が繰り返し表示されます。

on_comp_partitioned_table 句または index_subpartition_clause のサブパーティションに表領域の記憶域を指定しない場合は、index に指定された表領域が使用されます。index に表領域の記憶域を指定しない場合は、同じ表領域のサブパーティションが、対応する表のサブパーティションとして格納されます。

domain_index_clause

domain_index_clause を使用して、index が、アプリケーション固有の indextype 索引タイプのインスタンスであるドメイン索引であることを指定します。

注意： ドメイン索引を作成する前に、複数の操作を行う必要があります。まず、索引タイプの実装タイプを作成します。また、ファンクション実装を作成し、そのファンクションを使用する演算子も作成します。次に演算子と実装タイプを関連付ける索引タイプを作成します。最後にこの句を使用してドメイン索引を作成します。

付録 D「例」にはこれらの操作のすべてを含んだ単純なドメイン索引の事例があります。これらの例は、リファレンス内の該当する SQL 文に分散した場合、参照が困難になるため、1 つの付録にまとめてあります。

index_expr (table_index_clause 内) で、索引が定義されている表の列またはオブジェクトの属性を指定します。基礎となる索引タイプが異なり、その索引タイプがユーザー定義操作の分割セットをサポートする場合、1 つの列に複数のドメイン索引を定義できます。

ドメイン索引の制限事項：

- index_expr (table_index_clause 内) は単一の列のみ指定でき、REF データ型、VARRAY、ネストした表、LONG または LONG RAW の列は指定できません。
- ビットマップ索引または一意ドメイン索引は指定できません。

indextype *indextype* には、索引タイプの名前を指定します。名前は、すでに定義された有効なスキーマ・オブジェクトです。

注意： Oracle Text がインストールされている場合、各種の組込み索引タイプを使用し、Oracle Text ドメイン索引を作成できます。Oracle Text および Oracle Text が使用する索引の詳細は、『Oracle Text リファレンス』を参照してください。

参照： 13-88 ページの「[CREATE INDEXTYPE](#)」を参照してください。

parallel_clause *parallel_clause* を使用して、ドメイン索引の作成をパラレル化します。非パーティション・ドメイン索引の場合、Oracle は明示的またはデフォルトの並列度を ODCIIndexCreate カートリッジ・ルーチンに渡し、ODCIIndexCreate カートリッジ・ルーチンが索引の並列性を確立します。

参照： ODCI ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

PARAMETERS PARAMETERS には、未解析のまま適切な ODCI 索引タイプ・ルーチンに渡されたパラメータ文字列を指定します。パラメータ文字列の最大長は 1,000 文字です。

構文の最上位でこの句を指定した場合、パラメータは索引パーティションのデフォルトのパラメータになります。LOCAL [PARTITION] 句の一部としてこの句を指定すると、個々のパーティションのデフォルトのパラメータを上書きできます。

ドメイン索引が作成されると、適切な ODCI ルーチンがコールされます。ルーチンが正常に戻らない場合、ドメイン索引は FAILED のマークが付けられます。失敗したドメイン索引でサポートされる操作は、DROP INDEX および REBUILD INDEX（非ローカル索引用）のみです。

参照： これらのルーチンについては、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

bitmap_join_index_clause

bitmap_join_index_clause を使用すると、ビットマップ・ジョイン・インデックスを定義できます。ビットマップ・ジョイン・インデックスは、単一の表に定義します。ディメンション表の列で構成する索引キーには、そのキーに対応するファクト表の ROWID が格納されます。データ・ウェアハウス環境では、一般的に、索引を定義する表を**ファクト表**といい、ファクト表と結合した表を**ディメンション表**といいます。ただし、結合索引の作成にはスター・スキーマは必須ではありません。

ON ON 句には、まずファクト表を指定し、次に索引を定義するディメンション表の列をカッコ内に指定します。

FROM FROM 句には、結合した表を指定します。

WHERE WHERE 句には、結合条件を指定します。

基礎となるファクト表がパーティション化されている場合、`local_partitioned_index` 句 (13-78 ページの「`local_partitioned_index`」を参照) のいずれかを指定する必要があります。

ビットマップ・ジョイン・インデックスの制限事項： 一般的なビットマップ索引の制限事項 (13-68 ページの「`BITMAP`」を参照してください。) に加え、ビットマップ・ジョイン・インデックスには次の制限事項があります。

- 索引構成表または一時表にビットマップ・ジョイン・インデックスを作成できません。
- FROM 句で表を 2 回指定できません。
- ファンクション結合索引は作成できません。
- ディメンション表の列は、主キー列であるか、または一意制約を含む必要があります。
- ディメンション表が複合主キーを含む場合、主キーの各列は結合の一部である必要があります。
- ファクト表がパーティション化されていない場合は、`local_index_clauses` を指定できません。

参照： ファクト表とディメンション表、およびデータ・ウェアハウス環境でのビットマップ索引の使用については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

例

一般的な索引の例

索引の作成例： 次の文は、サンプル表 `oe.orders` の `customer_id` 列にサンプル索引 `ord_customer_ix` を作成します。

```
CREATE INDEX ord_customer_ix
ON orders (customer_id);
```

索引の圧縮例： 次の文は、`COMPRESS` 句を使用して索引 `ord_customer_ix` を作成します。

```
CREATE INDEX ord_customer_ix
ON orders (customer_id, sales_rep_id)
COMPRESS 1;
```

索引は、`customer_id` 列値の繰返し項目を圧縮します。

統計情報の計算例： 次の文は、索引 `ord_customer_ix_demo` の作成中に、その統計情報を収集します。

```
CREATE INDEX ord_customer_ix_demo
  ON orders(customer_id, sales_rep_id)
  COMPUTE STATISTICS;
```

収集された統計のタイプは、作成する索引タイプによって異なります。

NOLOGGING モードの索引の作成例： サンプル表 `orders` が高速パラレル・ロードで作成された場合（すべての行がソート済である場合）、次の文は、迅速に索引を作成します。

```
/* Unless you first sort the table oe.orders, this example fails
   because you cannot specify NOSORT unless the base table is
   already sorted.
*/
CREATE INDEX ord_customer_ix_demo
  ON orders (order_mode)
  NOSORT
  NOLOGGING;
```

クラスタ索引の作成例： 次の文は、`personnel` クラスタ（13-9 ページの「[クラスタの作成例：](#)」で作成）に対して索引を作成します。

```
CREATE INDEX idx_personnel ON CLUSTER personnel;
```

クラスタ・キーのすべての列に索引が自動的に作成されるため、索引列は指定しません。クラスタ索引の場合は、すべての行に索引が付きます。

XMLType 表の索引の作成例： 次の文は、`xwarehouses` 表（15-67 ページの「[XMLType 表の例：](#)」で作成）の領域要素に索引を作成します。

```
CREATE INDEX area_index ON xwarehouses e
  (EXTRACTVALUE(VALUE(e), '/Warehouse/Area'));
```

この索引によって、たとえば、次の文にあるようなウェアハウスの平方フィートに基づいた表から選択する問合せのパフォーマンスが大幅に向上します。

```
SELECT e.getClobVal() AS warehouse
  FROM xwarehouses e
 WHERE EXISTSNODE(VALUE(e), '/Warehouse [Area>50000]') = 1;
```

参照： 6-57 ページの「[EXISTSNODE](#)」および 6-192 ページの「[VALUE](#)」を参照してください。

ファンクション索引の例

ファンクション索引の作成例： 次の文は、last_name 列の大文字評価に基づく employees 表にファンクション索引を作成します。

```
CREATE INDEX upper_ix ON employees (UPPER(last_name));
```

ファンクション索引の作成に必要な権限およびパラメータ設定の詳細は、13-61 ページの「[前提条件](#)」を参照してください。

全表スキャンの実行ではなく、索引を使用するように、ファンクション値を後続の間合せて NULL 以外にします。たとえば、次の文は、索引の使用を保証します。

```
SELECT first_name, last_name
   FROM employees WHERE UPPER(last_name) IS NOT NULL
   ORDER BY UPPER(last_name);
```

ただし、WHERE 句が指定されていない場合は、全表スキャンが実行されます。

索引の作成および後続の間合せを示す次の文では、間合せて列の順序が逆であっても、Oracle は income_ix を使用します。

```
CREATE INDEX income_ix
   ON employees(salary + (salary*commission_pct));

SELECT first_name||' '||last_name "Name"
   FROM employees
   WHERE (salary*commission_pct) + salary > 15000;
```

LOB 列のファンクション索引の例： 次の文は、ファンクション（13-60 ページの「[ファンクションでのパッケージ・プロシージャの使用例：](#)」で作成）を使用し、サンプル・スキーマ pm の LOB 列にファンクション索引を作成します。例では、ファンクション索引の統計情報を収集し、CLOB 列が 1000 字未満のサンプル表 print_media から行を検索します。

```
CREATE INDEX src_idx ON print_media(text_length(ad_sourcetext));

ANALYZE INDEX src_idx COMPUTE STATISTICS;

SELECT product_id FROM print_media
   WHERE text_length(ad_sourcetext) < 1000;

PRODUCT_ID
-----
      3060
      2056
      3106
      2268
```

型メソッドのファンクション索引の作成例：

この例には、2つの数値属性（length および width）を含むオブジェクト型 rectangle が必要です。area() メソッドは、四角形の領域を計算します。

```
CREATE TYPE rectangle AS OBJECT
( length  NUMBER,
  width   NUMBER,
  MEMBER FUNCTION area RETURN NUMBER DETERMINISTIC
);
```

```
CREATE OR REPLACE TYPE BODY rectangle AS
  MEMBER FUNCTION area RETURN NUMBER IS
  BEGIN
    RETURN (length*width);
  END;
END;
```

rectangle 型の表 rect_tab を作成する場合、次のように area() メソッドにファンクション索引を作成できます。

```
CREATE TABLE rect_tab OF rectangle;
CREATE INDEX area_idx ON rect_tab x (x.area());
```

この索引を使用して、効率的に次の形式の問合せを評価できます。

```
SELECT * FROM rect_tab x WHERE x.area() > 100;
```

パーティション索引の例

グローバル・パーティション索引の作成例： 次の文は、コストの範囲を3つのグループに分割した3つのパーティションを含むサンプル表 sh.sales にグローバル同一キー索引 cost_ix を作成します。

```
CREATE INDEX cost_ix ON sales (amount_sold)
  GLOBAL PARTITION BY RANGE (amount_sold)
    (PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (2500),
     PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

ハッシュ・パーティション表の索引の作成例： 次の文は、product_information_part パーティション表（15-71 ページの「ハッシュ・パーティション化の例：」で作成）の product_id 列にローカル索引を作成します。LOCAL の直後に記述された STORE IN 句は、product_information_part がハッシュ・パーティション化されていることを示します。Oracle は、tbs1 表領域と tbs2 表領域にハッシュ・パーティションを分割します。

```
CREATE INDEX prod_idx ON product_information_part (product_id) LOCAL
  STORE IN (tbs_1, tbs_2);
```

注意： 索引を作成するには、指定する表領域に対して割当て制限を持つ必要があります。これらの表領域の例は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

コンポジット・パーティション表の索引の作成例： 次の文は、composite_sales 表 (15-71 ページの「[コンポジット・パーティション表の例](#)」で作成) にローカル索引を作成します。STORAGE 句では、索引のデフォルトの記憶域属性を指定します。ただし、別の TABLESPACE 記憶域が指定されているため、このデフォルトは、パーティション q3_2000 および q4_2000 の 5 つのサブパーティションに上書きされます。

注意： 索引を作成するには、指定する表領域に対して割当て制限を持つ必要があります。表領域 tbs_1 および tbs_2 の作成については、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

```
CREATE INDEX sales_ix ON composite_sales(time_id, prod_id)
  STORAGE (INITIAL 1M MAXEXTENTS UNLIMITED)
  LOCAL
  (PARTITION q1_1998,
   PARTITION q2_1998,
   PARTITION q3_1998,
   PARTITION q4_1998,
   PARTITION q1_1999,
   PARTITION q2_1999,
   PARTITION q3_1999,
   PARTITION q4_1999,
   PARTITION q1_2000,
   PARTITION q2_2000
    (SUBPARTITION pq2001, SUBPARTITION pq2002,
     SUBPARTITION pq2003, SUBPARTITION pq2004,
     SUBPARTITION pq2005, SUBPARTITION pq2006,
     SUBPARTITION pq2007, SUBPARTITION pq2008),
   PARTITION q3_2000
    (SUBPARTITION c1 TABLESPACE tbs_1,
     SUBPARTITION c2 TABLESPACE tbs_1,
     SUBPARTITION c3 TABLESPACE tbs_1,
     SUBPARTITION c4 TABLESPACE tbs_1,
     SUBPARTITION c5 TABLESPACE tbs_1),
   PARTITION q4_2000
    (SUBPARTITION pq4001 TABLESPACE tbs_2,
     SUBPARTITION pq4002 TABLESPACE tbs_2,
     SUBPARTITION pq4003 TABLESPACE tbs_2,
     SUBPARTITION pq4004 TABLESPACE tbs_2)
  );
```

ビットマップ索引の例：

次の文は、表 `oe.product_information_part` (15-71 ページの「[ハッシュ・パーティション化の例：](#)」で作成) にビットマップ・ジョイン・インデックスを作成します。

```
CREATE BITMAP INDEX product_bm_ix
  ON product_information_part(list_price)
  TABLESPACE tbs_1
  LOCAL(PARTITION ix_p1 TABLESPACE tbs_2,
        PARTITION ix_p2,
        PARTITION ix_p3 TABLESPACE tbs_3,
        PARTITION ix_p4,
        PARTITION ix_p5 TABLESPACE tbs_4 );
```

`product_information_part` はパーティション表であるため、ビットマップ・ジョイン・インデックスはローカル・パーティションである必要があります。

注意： この例では、指定する表領域に対して割当て制限を持つ必要があります。表領域 `tbs_2`、`tbs_3` および `tbs_4` を作成する例は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

ネストした表の索引の例：

サンプル表 `pm.print_media` には、記憶表 `textdocs_nestedtab` に格納されたネストした表の列 `ad_textdocs_ntab` が含まれます。次の文は、記憶表 `textdocs_nestedtab` に一意索引を作成します。

```
CREATE UNIQUE INDEX nested_tab_ix
  ON textdocs_nestedtab(NESTED_TABLE_ID, document_typ);
```

疑似列 `NESTED_TABLE_ID` を組み込むことによって、ネストした表の列 `ad_textdocs_ntab` 内に固有の行が確保されます。

置換可能な列の索引の例：

置換可能な列を宣言した型の属性に、索引を作成できます。また、適切な `TREAT` ファンクションの使用によって、サブタイプの属性を参照できます。次の例は、表 `books` (15-63 ページの「[置換可能な表および列のサンプル：](#)」で作成) を使用します。この文は、`books` 表の、すべての `employee_t` 型の `author` の `salary` 属性に索引を作成します。

```
CREATE INDEX salary_i
  ON books (TREAT(author AS employee_t).salary);
```

`TREAT` ファンクションの引数のターゲットとなる型は、参照する属性を追加した型である必要があります。例では、`TREAT` のターゲットは、`employee_t` で、`salary` 属性を追加した型です。

この条件を満たさない場合、TREAT ファンクションがファンクションの定義式として解析され、ファンクション索引が作成されます。たとえば、次の文は型の階層内の他のすべての型のインスタンスに NULL を割り当て、パートタイム従業員の salary 属性にファンクション索引を作成します。

```
CREATE INDEX salary_func_i ON persons p
  (TREAT(VALUE(P) AS part_time_emp_t).salary);
```

SYS_TYPEID ファンクションの使用によって、置換可能な列を基礎とする型判別式の列に索引を作成できます。

注意： Oracle は型判別式の列を使用し、IS OF type 条件を含む問合せを評価します。現行のスキーマ型 ID 列のカーディナリティが正常な範囲で低い場合、ビットマップ索引を作成することをお勧めします。

次の文は、books 表の author 列の型 ID にビットマップ索引を作成します。

```
CREATE BITMAP INDEX typeid_i ON books (SYS_TYPEID(author));
```

参照：

- books 表を基礎とする型の階層の作成については、16-22 ページの「[型の階層例：](#)」を参照してください。
- 6-181 ページの「[TREAT](#)」を参照してください。
- 6-155 ページの「[SYS_TYPEID](#)」を参照してください。
- 5-19 ページの「[IS OF type 条件](#)」を参照してください。

CREATE INDEXTYPE

用途

CREATE INDEXTYPE 文を使用すると、(アプリケーション固有の) ドメイン索引を管理するルーチンを指定するオブジェクトである**索引タイプ**を作成できます。索引タイプは、表、ビューおよび他のスキーマ・オブジェクトと同じネームスペースにあります。この文は、索引タイプ名を実装タイプに結合し、順番に索引タイプを実装するユーザー定義索引ファンクションおよびプロシージャを指定し、参照します。

参照： 索引タイプの実装の詳細は、『Oracle9i Data Cartridge Developer’s Guide』および『Oracle9i データベース概要』を参照してください。

前提条件

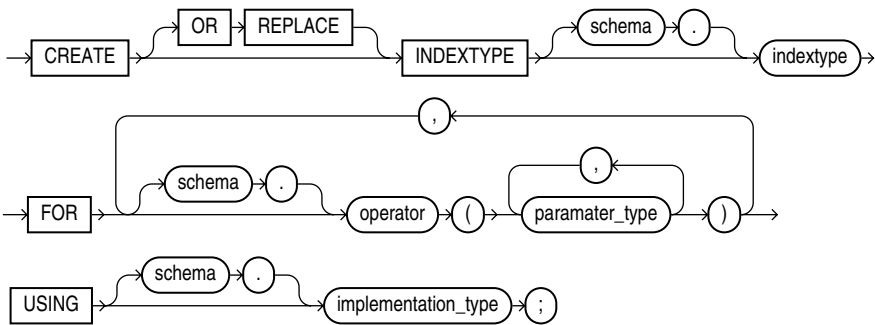
自分のスキーマに索引タイプを作成する場合は、CREATE INDEXTYPE システム権限が必要です。他のユーザーのスキーマ内に索引タイプを作成する場合は、CREATE ANY INDEXTYPE システム権限が必要です。どちらの場合も、実装タイプおよびサポートしている演算子に対する EXECUTE オブジェクト権限が必要です。

索引タイプは、1 つ以上の演算子をサポートしているため、索引タイプを作成する前に、その演算子またはサポートする演算子を設計し、これらの演算子に機能的な実装を指定します。

参照： 14-40 ページの「[CREATE OPERATOR](#)」を参照してください。

構文

create_indectype::=



セマンティクス

schema

索引タイプが存在するスキーマ名を指定します。*schema* を指定しない場合、自分のスキーマ内に索引タイプが作成されます。

indextype

作成する索引タイプの名前を指定します。

FOR 句

FOR 句によって、索引タイプにサポートされる演算子のリストを指定できます。

- *schema* には、演算子を含むスキーマを指定します。*schema* を指定しない場合、その演算子が自分のスキーマにあるとみなされます。
- *operator* には、索引タイプによってサポートされる演算子の名前を指定します。
この句に指定されるすべての演算子は有効な演算子である必要があります。
- *parameter_type* には、演算子へのパラメータ・タイプを指定します。

USING 句

USING 句によって、新しい索引タイプを実装するタイプを指定できます。

implementation_type には、ODCI を実装するタイプ名を指定します。

- ODCI でルーチンを実装する有効なタイプを指定する必要があります。
- 実装タイプは、索引タイプと同じスキーマに存在する必要があります。

参照： このインタフェースの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

例

索引タイプの作成例： 次の文は、*position_indextype* という名前の索引タイプを作成し、その索引タイプでサポートされている *position_between* 演算子、および索引インタフェースを実装する *position_im* タイプを指定します。この索引タイプを使用する拡張索引作成機能の使用例については、D-2 ページの「[拡張索引作成機能の使用法](#)」を参照してください。

```
CREATE INDEXTYPE position_indextype
  FOR position_between(NUMBER, NUMBER, NUMBER)
  USING position_im;
```

CREATE JAVA

用途

CREATE JAVA を使用すると、Java ソース、クラスまたはリソースを含むスキーマ・オブジェクトを作成できます。

参照：

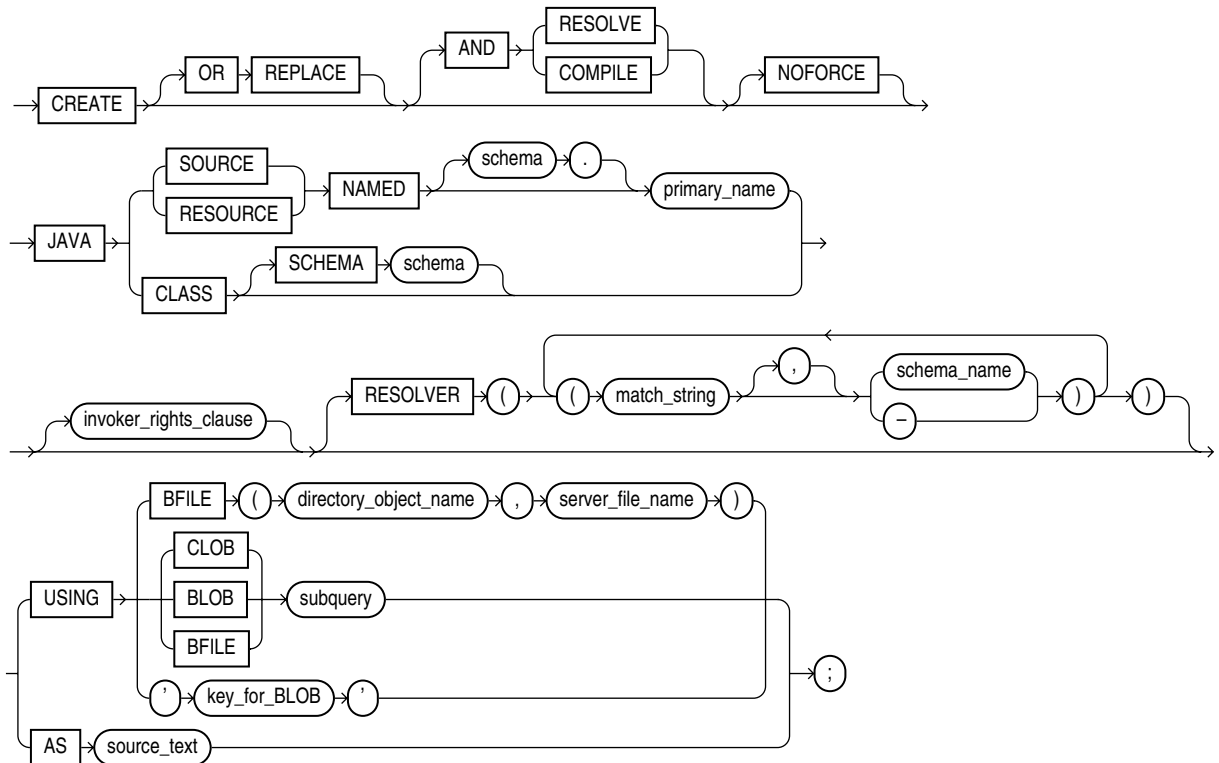
- Java の概要については、『Oracle9i Java Developer’s Guide』を参照してください。
- Java ストアド・プロシージャについては、『Oracle9i Java Stored Procedures Developer’s Guide』を参照してください。
- SQLJ については、『Oracle9i SQLJ 開発者ガイドおよびリファレンス』を参照してください。
- JDBC については、『Oracle9i JDBC 開発者ガイドおよびリファレンス』を参照してください。
- Oracle9iAS の Java のサポートについては、『Oracle9iAS Containers for J2EE ユーザーズ・ガイド』を参照してください。

前提条件

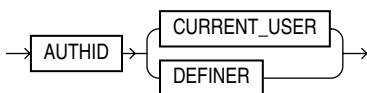
自分のスキーマに Java ソース、クラスまたはリソースを含むスキーマ・オブジェクトを作成または再作成する場合は、CREATE PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にスキーマ・オブジェクトを作成する場合は、CREATE ANY PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にスキーマ・オブジェクトを再作成する場合は、ALTER ANY PROCEDURE システム権限が必要です。

構文

create_java::=



invoker_rights_clause::=



セマンティクス

OR REPLACE

OR REPLACE を指定して、既存の Java クラス、ソースまたはリソースを含むスキーマ・オブジェクトを再作成します。この句を指定した場合、付与されているオブジェクト権限を削除、再作成および再付与しなくても、既存のオブジェクトの定義を変更できます。

Java スキーマ・オブジェクトを再定義し、RESOLVE または COMPILE を指定する場合、Oracle は、オブジェクトを再コンパイルまたは変換します。正常に変換またはコンパイルされたかどうかにかかわらず、Java スキーマ・オブジェクトを参照するクラスは有効になります。

再定義したファンクションに対して権限が付与されていたユーザーは、権限を再付与されなくても、そのファンクションにアクセスできます。

参照： その他の情報は、9-85 ページの「[ALTER JAVA](#)」を参照してください。

RESOLVE | COMPILE

RESOLVE および COMPILE は、同義のキーワードです。この文が正常に実行されると作成される Java スキーマ・オブジェクトを変換することを指定します。

- クラスに適用された場合、他のクラス・スキーマ・オブジェクトに対する参照名に変換されます。
- ソースに適用された場合、ソースがコンパイルされます。

RESOLVE および COMPILE の制限事項： Java リソースには、この句を指定できません。

NOFORCE

RESOLVE または COMPILE を指定しても、正常に変換またはコンパイルできない場合は、NOFORCE を指定して CREATE コマンドの結果をロールバックします。このオプションを省略した場合、正常に変換またはコンパイルできない場合でも処理は行われません（作成されたスキーマ・オブジェクトが残ります）。

JAVA SOURCE 句

JAVA SOURCE を指定して、Java ソース・ファイルをロードします。

JAVA CLASS 句

JAVA CLASS を指定して、Java クラス・ファイルをロードします。

JAVA RESOURCE 句

JAVA RESOURCE を指定して、Java リソース・ファイルをロードします。

NAMED 句

NAMED 句は、Java ソースまたはリソースの場合に指定します。*primary_name* は、二重引用符で囲む必要があります。

- Java ソースの場合、この句にはソース・コードが保持されているスキーマ・オブジェクト名を指定します。正常な CREATE JAVA SOURCE 文は、ソースによって定義された Java クラスをそれぞれ保持するために、追加スキーマ・オブジェクトも作成します。
- Java リソースの場合、この句にはスキーマ・オブジェクト名を指定し、Java リソースを保持します。

primary_name の小文字、または大文字と小文字の組合せを保持するには、二重引用符を使用します。

schema を省略した場合、自分のスキーマ内にオブジェクトが作成されます。

NAMED の制限事項：

- Java クラスには、NAMED を指定できません。
- *primary_name* は、データベース・リンクを含むことはできません。

SCHEMA 句

SCHEMA 句は、Java クラスにのみ適用されます。このオプションは、Java ファイルを含むオブジェクトが存在するスキーマを指定します。この句を省略した場合、自分のスキーマ内にオブジェクトが作成されます。

invoker_rights_clause

invoker_rights_clause を使用して、クラスのメソッドが、権限を持つそのクラスを所有するユーザーのスキーマ内で実行されるか、または権限を持つ CURRENT_USER のスキーマ内で実行されるかを指定します。

また、この句は、問合せ、DML 操作、およびその型のメンバー・ファンクションおよびプロシージャ内の動的 SQL 文の外部名の変換方法も定義します。

AUTHID CURRENT_USER

CURRENT_USER は、クラスのメソッドが CURRENT_USER 権限で実行されることを指定します。この句はデフォルトで、**実行者権限クラス**を作成します。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を CURRENT_USER のスキーマで変換することも指定します。他のすべての文における外部名は、メソッドを含むスキーマで変換します。

AUTHID DEFINER

DEFINER は、クラスが存在するスキーマの所有者権限でクラスのメソッドを実行すること、およびクラスが存在するスキーマで外部名を変換することを指定します。この句によって**定義者権限クラス**が作成されます。

参照：

- 『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- `CURRENT_USER` の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

RESOLVER 句

次の場合は、RESOLVER 句によって、Java スキーマ・オブジェクトに対する完全修飾 Java 名のマッピングを指定します。

- `match_string` は、完全な Java 名、Java 名と一致するワイルド・カードまたは任意の名前と一致するワイルド・カードを指定します。
- `schema_name` は、対応する Java スキーマ・オブジェクトを検索するスキーマを指定します。
- `schema_name` の代替としてのダッシュ (-) は、`match_string` が有効な Java 名と一致した場合、名前が変換されないことを示します。名前の変換は成功しますが、実行時にクラスがその名前を使用することはできません。

このマッピングは、後の変換で（暗黙的に、または ALTER ... RESOLVE 文で明示的に）使用されるコマンドで作成されるスキーマ・オブジェクトの定義とともに格納されます。

USING 句

USING 句は、Java クラスまたはリソースに対する文字 (CLOB または BFILE) またはバイナリ (BLOB または BFILE) データの順序を指定します。文字の順序を使用して、1 つのファイルが Java クラスまたはリソースに、または 1 つのソース・ファイルおよび 1 つ以上の導出クラスが Java ソースに定義されます。

BFILE 句

順序を含む、オペレーティング・システム (`directory_object_name`) およびサーバー・ファイル (`server_file_name`) であらかじめ作成されているファイルのディレクトリおよびファイル名を指定します。BFILE は、通常、CREATE JAVA SOURCE によって文字順序として、CREATE JAVA CLASS または CREATE JAVA RESOURCE によってバイナリ順序として解析されます。

CLOB | BLOB | BFILE subquery

指定した型 (CLOB、BLOB または BFILE) の行と列を選択する問合せを提供します。列の値は文字の順序を構成します

注意： 以前のリリースでは、`USING` 句は暗黙的にキーワード `SELECT` を提供しました。今回のリリースでは提供されません。ただし、キーワード `SELECT` なしの副問合せは、（下位互換性のために）サポートされています。

key_for_BLOB

`key_for_BLOB` 句は、次の暗黙的な問合せを提供します。

```
SELECT LOB FROM CREATE$JAVA$LOB$TABLE
WHERE NAME = 'key_for_BLOB';
```

key_for_BLOB 句の制限事項： このパラメータを使用する場合、表 `CREATE$JAVA$LOB$TABLE` が現行のスキーマ内にあり、`BLOB` 型の `LOB` 列および `VARCHAR2` 型の `NAME` 列が存在する必要があります。

AS source_text

Java または SQLJ ソースの文字列を指定します。

例

Java クラス・オブジェクトの作成例： 次の文は、Java バイナリ・ファイルにある名前を使用して、Java クラスを含むスキーマ・オブジェクトを作成します。

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

この例では、Java クラス `Agent.class` を含むオペレーティング・システム・ディレクトリに指定されるディレクトリ・オブジェクト `bfile_dir` がすでに存在していることが前提です。この例では、クラス名は Java クラス・スキーマ・オブジェクトの名前になります。

Java ソース・オブジェクトの作成例： 次の文は、Java ソース・スキーマ・オブジェクトを作成します。

```
CREATE JAVA SOURCE NAMED "Hello" AS
public class Hello {
    public static String hello() {
        return "Hello World";    } };

```

Java リソース・オブジェクトの作成例： 次の文は、`bfile` から `apptext` という名前の Java リソース・スキーマ・オブジェクトを作成します。

```
CREATE JAVA RESOURCE NAMED "appText"
USING BFILE (bfile_dir, 'textBundle.dat');
```

SQL 文 : CREATE LIBRARY ~ CREATE SPFILE

この章では、次の SQL 文について説明します。

- CREATE LIBRARY
- CREATE MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW LOG
- CREATE OPERATOR
- CREATE OUTLINE
- CREATE PACKAGE
- CREATE PACKAGE BODY
- CREATE PFILE
- CREATE PROCEDURE
- CREATE PROFILE
- CREATE ROLE
- CREATE ROLLBACK SEGMENT
- CREATE SCHEMA
- CREATE SEQUENCE
- CREATE SPFILE

CREATE LIBRARY

用途

CREATE LIBRARY 文を使用すると、オペレーティング・システム共有ライブラリに関連するスキーマ・オブジェクトを作成できます。このスキーマ・オブジェクトの名前は、CREATE FUNCTION または CREATE PROCEDURE 文の *call_spec* で使用できます。また、パッケージまたはタイプにおけるファンクションまたはプロシージャを宣言する際にも使用できます。これによって、SQL および PL/SQL は、3GL ファンクションおよびプロシージャに対してコールできます。

参照： ファンクションおよびプロシージャの詳細は、13-49 ページの「[CREATE FUNCTION](#)」および『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。

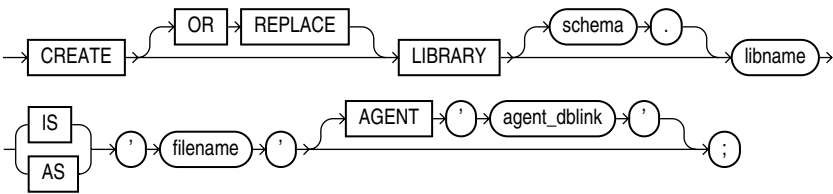
前提条件

自分のスキーマ内にライブラリを作成する場合は、CREATE LIBRARY システム権限が必要です。他のユーザーのスキーマ内にライブラリを作成する場合は、CREATE ANY LIBRARY システム権限が必要です。ライブラリに格納されているプロシージャおよびファンクションを使用する場合は、そのライブラリに対する EXECUTE オブジェクト権限が必要です。

CREATE LIBRARY 文は、共有ライブラリおよび動的リンクをサポートするプラットフォーム上でのみ有効です。

構文

create_library::=



セマンティクス

OR REPLACE

既存のライブラリを再作成する場合は、OR REPLACE を指定します。この句を指定した場合、既存のライブラリに付与されているスキーマ・オブジェクト権限を削除、再作成および再付与しなくても、ライブラリの定義を変更できます。

再定義したライブラリに対して権限を付与されていたユーザーは、権限を再付与されなくてもライブラリにアクセスできます。

libname

作成するライブラリの名前を指定します。 *call_spec* でファンクションまたはプロシージャを宣言するときは、この名前を指定します。

'filename'

引用符で囲まれた文字リテラルを指定します。文字列には、オペレーティング・システムの共有ライブラリの名前となるパスまたはファイル名を指定します。

'filespec' は、CREATE LIBRARY の実行中は解析されません。ライブラリ・ファイルの存在は、そのファイルからルーチンが実行されるまでチェックされません。

AGENT 句

サーバーではなく、データベース・リンクから外部プロシージャを実行する場合は、AGENT 句を指定します。外部プロシージャの実行には、*agent_dblink* に指定したデータベース・リンクが使用されます。この句を指定しない場合、サーバーのデフォルト・エージェント (extproc) が、外部プロシージャを実行します。

例

ライブラリの作成例： 次の文は、ライブラリ ext_lib を作成します。

```
CREATE LIBRARY ext_lib AS '/OR/lib/ext_lib.so';
/
```

次の文は、ライブラリ ext_lib を再作成します。

```
CREATE OR REPLACE LIBRARY ext_lib IS '/OR/newlib/ext_lib.so';
/
```

外部プロシージャ・エージェントの指定例： 次の例は、ライブラリ app_lib を作成し、パブリック・データベース sales.hq.acme.com から外部プロシージャを起動することを指定します。

```
CREATE LIBRARY app_lib as '${ORACLE_HOME}/lib/app_lib.so'
  AGENT 'sales.hq.acme.com';
/
```

参照： データベース・リンクの作成方法については、13-39 ページの「[パブリック・データベース・リンクの定義例:](#)」を参照してください。

CREATE MATERIALIZED VIEW

用途

CREATE MATERIALIZED VIEW 文を使用すると、**マテリアライズド・ビュー**を作成できます。マテリアライズド・ビューは、問合せ結果を含むデータベース・オブジェクトです。問合せの FROM 句には、表、ビューおよびその他のマテリアライズド・ビューを指定できます。これらをあわせて、**マスター表**（レプリケーション用語）または**ディテール表**（データ・ウェアハウス用語）といいます。このマニュアルでは、「マスター表」を使用します。マスター表が格納されているデータベースを**マスター・データベース**といいます。

注意： キーワード SNAPSHOT は、MATERIALIZED VIEW のかわりに下位互換用にサポートされています。

レプリケーションでは、マテリアライズド・ビューを使用すると、ローカル・ノード上にあるリモート・データのコピーのメンテナンスができます。コピーは、アドバンスト・レプリケーション機能によって更新可能となりますが、この機能がない場合は読み込み専用です。マテリアライズド・ビューのデータを、表またはビューと同じように選択することができます。レプリケーション環境では、通常作成されるマテリアライズド・ビューは、**主キー**、**ROWID**、**オブジェクト**および**副問合せ**のマテリアライズド・ビューです。

参照： レプリケーションをサポートするマテリアライズド・ビューの詳細は、『Oracle9i アドバンスト・レプリケーション』を参照してください。

データ・ウェアハウスでは、通常作成されるマテリアライズド・ビューは、**マテリアライズド集計ビュー**、**単一表マテリアライズド集計ビュー**および**マテリアライズド結合ビュー**です。3つのマテリアライズド・ビューは、クエリー・リライトで使用できます。クエリー・リライトとは、マスター表に関して記述したユーザー要求を、1つ以上のマテリアライズド・ビューを含む同等の要求に変換するための最適化手法です。

参照：

- 9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- データ・ウェアハウスをサポートするマテリアライズド・ビューの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

前提条件

マテリアライズド・ビューの作成に必要な権限は、ロールを介してではなく、直接付与する必要があります。

自分のスキーマ内にマテリアライズド・ビューを作成する場合は、次の条件に従う必要があります。

- CREATE MATERIALIZED VIEW システム権限と、CREATE TABLE または CREATE ANY TABLE のいずれかのシステム権限が必要です。
- 所有していないマテリアライズド・ビューのマスター表にアクセスする場合は、各表に対する SELECT オブジェクト権限または SELECT ANY TABLE システム権限が必要です。

他のユーザーのスキーマ内にマテリアライズド・ビューを作成する場合、次の条件に従う必要があります。

- CREATE ANY MATERIALIZED VIEW システム権限が必要です。
- マテリアライズド・ビューの所有者には、CREATE TABLE システム権限が必要です。スキーマ所有者が所有していないマテリアライズド・ビューの任意のマスター表、およびそのマスター表に定義された任意のマテリアライズド・ビュー・ログにアクセスするには、各表に対する SELECT オブジェクト権限、または SELECT ANY TABLE システム権限が必要です。

REFRESH ON COMMIT モードのマテリアライズド・ビューを作成する（ON COMMIT REFRESH 句を使用）場合は、前述の権限の他に、所有しないマスター表に対する ON COMMIT REFRESH オブジェクト権限、または ON COMMIT REFRESH システム権限が必要です。

前述の権限の他にも、**クエリー・リライトが使用可能な**マテリアライズド・ビューを作成する場合は、次の条件に従う必要があります。

- マスター表の所有者には、QUERY REWRITE システム権限が必要です。
- マスター表の所有者でない場合は、GLOBAL QUERY REWRITE システム権限、または自分のスキーマ以外の各表に対する QUERY REWRITE オブジェクト権限が必要です。
- スキーマ所有者がマスター表を所有していない場合は、そのスキーマ所有者には GLOBAL QUERY REWRITE 権限、または自分のスキーマ以外の各表に対する QUERY REWRITE オブジェクト権限が必要です。
- 事前作成コンテナにマテリアライズド・ビューを定義する（ON PREBUILT TABLE を使用）場合は、コンテナ表に対する With Grant Option 付きの SELECT 権限が必要です。

マテリアライズド・ビューを含むスキーマのユーザーは、マテリアライズド・ビューのマスター表および索引を格納するターゲット表領域に十分な割当て制限を持つか、または UNLIMITED TABLESPACE システム権限を持つ必要があります。

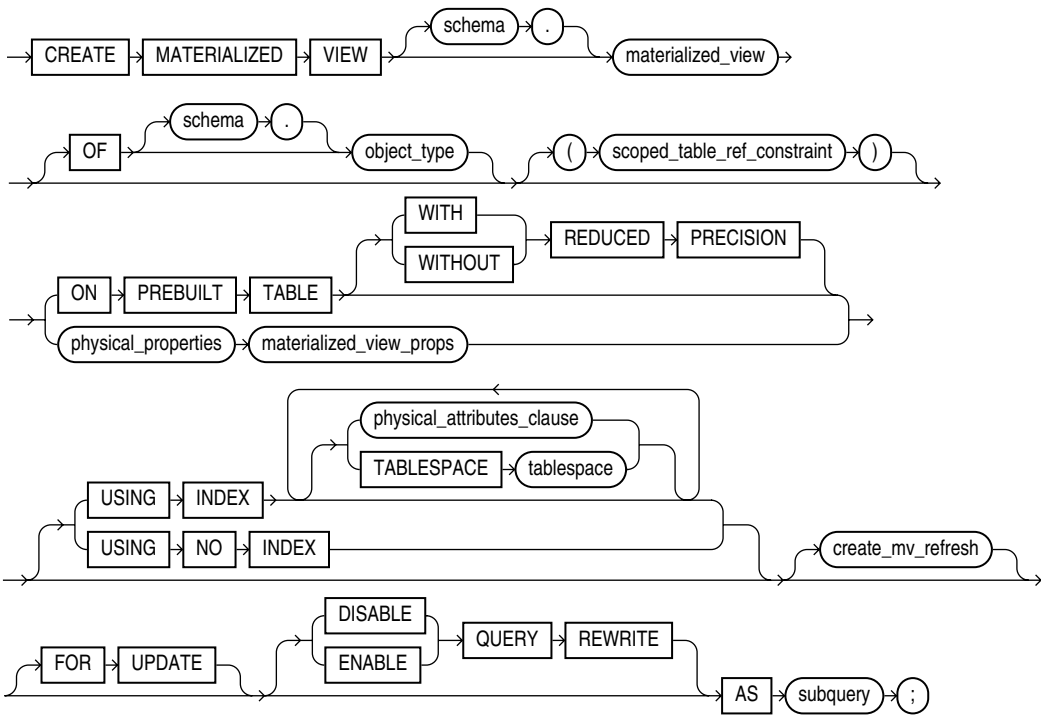
マテリアライズド・ビューを作成する場合、Oracle はマテリアライズド・ビューのスキーマ内すべてに、1 つの内部表および 1 つ以上の索引を作成します。また、1 つのビューを作成することもあります。これらのオブジェクトは、マテリアライズド・ビューのデータをメンテナンスするために使用します。ユーザーは、これらのオブジェクトを作成するための権限が必要です。

参照：

- これらの権限については、15-7 ページの「[CREATE TABLE](#)」、16-38 ページの「[CREATE VIEW](#)」、および 13-61 ページの「[CREATE INDEX](#)」を参照してください。
- レプリケーションのためのマテリアライズド・ビューの作成についての前提条件は、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウスのためのマテリアライズド・ビューの作成についての前提条件は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

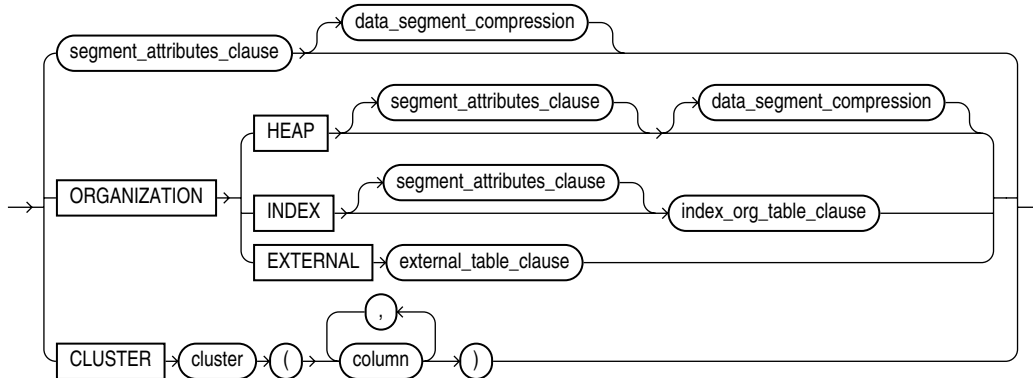
構文

create_materialized_view::=



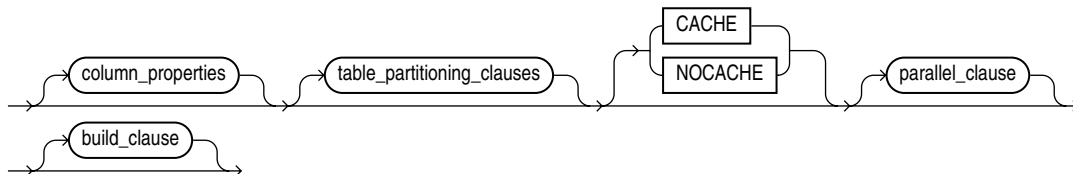
(14-8 ページの「[scoped_table_ref_constraint::=](#)」、14-8 ページの「[index_org_table_clause::=](#)」、14-7 ページの「[materialized_view_props::=](#)」、14-10 ページの「[physical_attributes_clause::=](#)」、14-9 ページの「[create_mv_refresh::=](#)」、18-5 ページの「[subquery::=](#)」を参照)

physical_properties::=

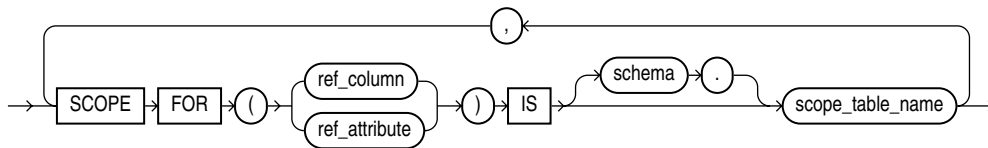
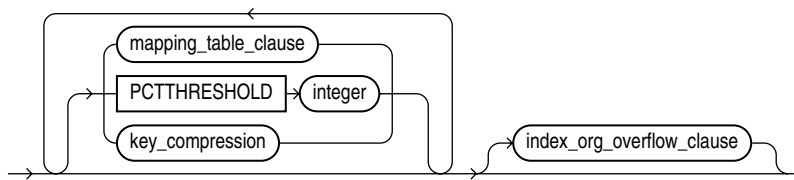


(14-9 ページの「[segment_attributes_clause::=](#)」、14-10 ページの「[data_segment_compression::=](#)」を参照)

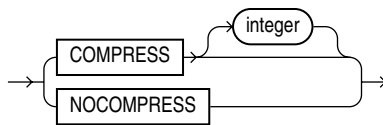
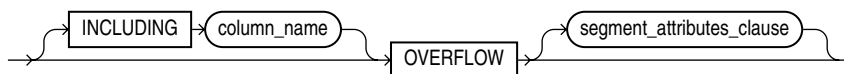
materialized_view_props::=



(14-10 ページの「[column_properties::=](#)」、15-42 ページの「[table_partitioning_clauses](#)」(CREATE TABLE 構文の一部)、14-13 ページの「[parallel_clause::=](#)」および 14-13 ページの「[build_clause::=](#)」を参照)

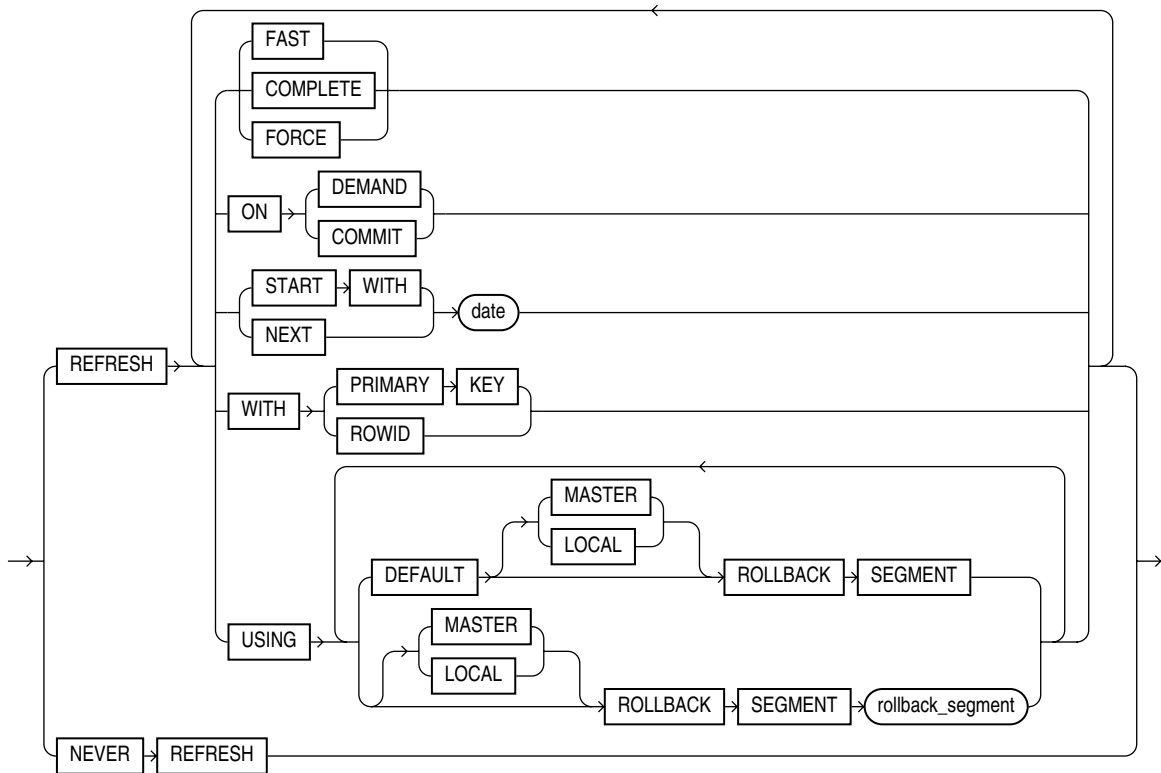
scoped_table_ref_constraint::=**index_org_table_clause::=**

(*mapping_table_clause*: マテリアライズド・ビューではサポートされません。
 14-8 ページの「[key_compression::=](#)」、14-8 ページの
 「[index_org_overflow_clause::=](#)」を参照)

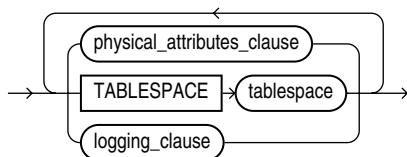
key_compression::=**index_org_overflow_clause::=**

(14-9 ページの「[segment_attributes_clause::=](#)」を参照)

create_mv_refresh::=

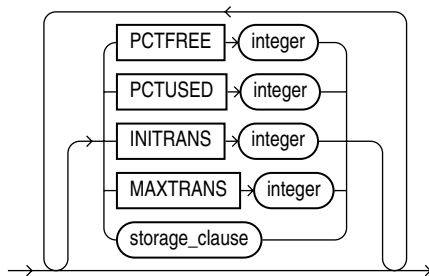


segment_attributes_clause::=



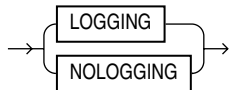
(14-10 ページの「[physical_attributes_clause::=](#)」、14-10 ページの「[logging_clause::=](#)」を参照)

physical_attributes_clause::=

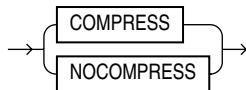


(7-42 ページの「[logging_clause::=](#)」を参照)

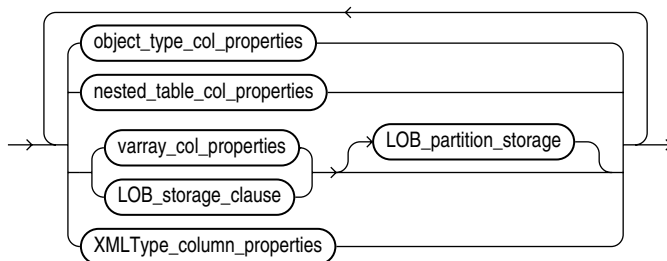
logging_clause::=



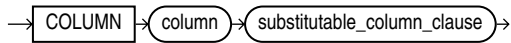
data_segment_compression::=



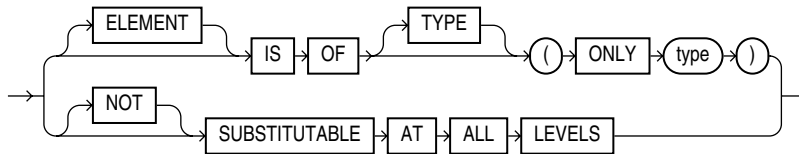
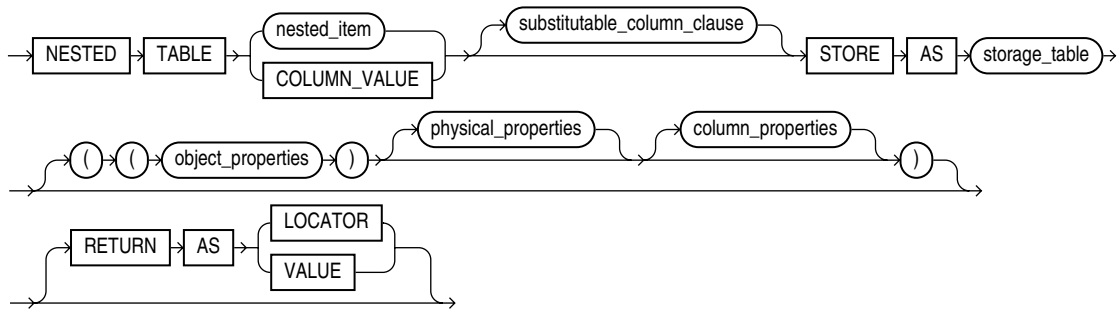
column_properties::=



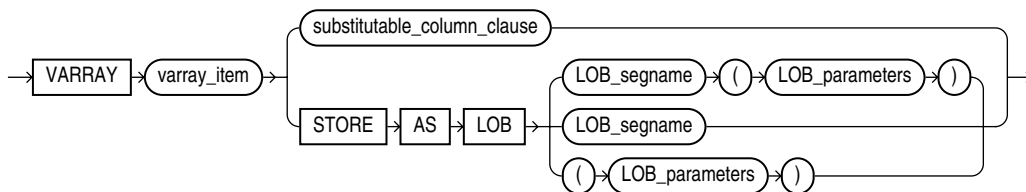
(14-11 ページの「[object_type_col_properties::=](#)」、14-11 ページの「[nested_table_col_properties::=](#)」、14-11 ページの「[varray_col_properties::=](#)」、14-13 ページの「[LOB_partition_storage::=](#)」および 14-12 ページの「[LOB_storage_clause::=](#)」を参照。
[XMLType_column_properties](#): マテリアライズド・ビューではサポートされていません。)

object_type_col_properties::=

(14-11 ページの「[substitutable_column_clause::=](#)」を参照)

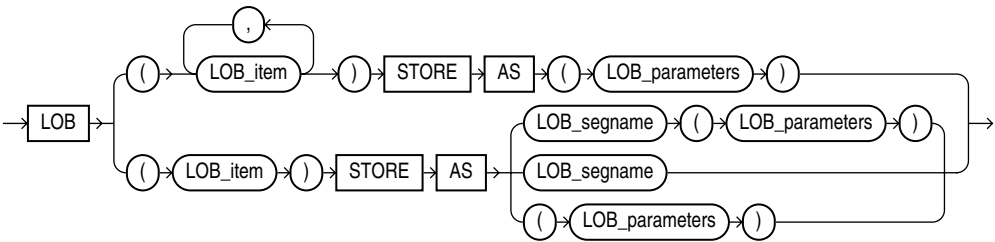
substitutable_column_clause::=**nested_table_col_properties::=**

(14-11 ページの「[substitutable_column_clause::=](#)」、15-10 ページの「[object_properties::=](#)」、15-11 ページの「[physical_properties::=](#)」(CREATE TABLE 構文の一部) および 14-10 ページの「[column_properties::=](#)」を参照)

varray_col_properties::=

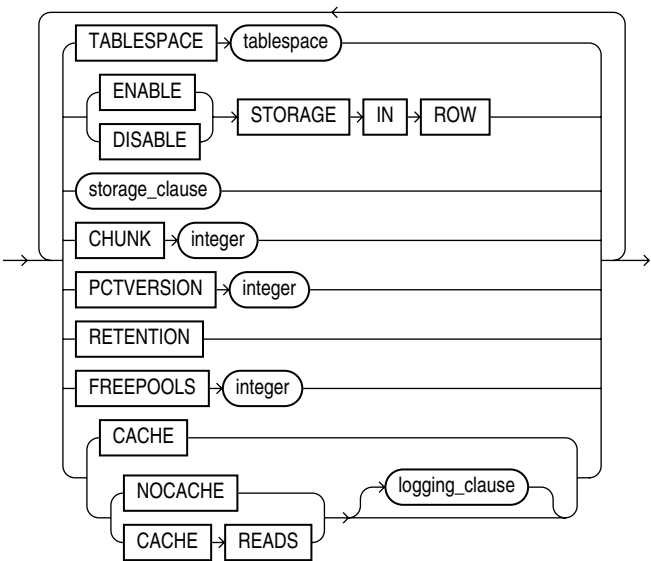
(14-11 ページの「[substitutable_column_clause::=](#)」、14-12 ページの「[LOB_parameters::=](#)」を参照)

LOB_storage_clause::=

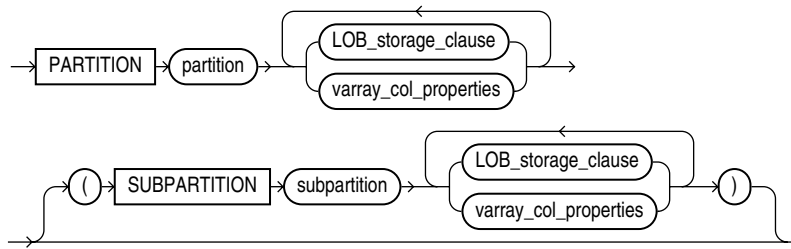


(14-12 ページの「[LOB_parameters::=](#)」を参照)

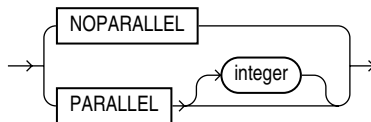
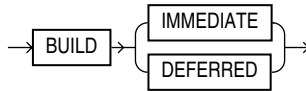
LOB_parameters::=



(7-53 ページの「[storage_clause::=](#)」、14-10 ページの「[logging_clause::=](#)」を参照)

LOB_partition_storage::=

(14-12 ページの「[LOB_storage_clause::=](#)」、14-11 ページの「[varray_col_properties::=](#)」を参照)

parallel_clause::=**build_clause::=****セマンティクス*****schema***

マテリアライズド・ビューを含むスキーマを指定します。*schema* を指定しないと、自分のスキーマ内にマテリアライズド・ビューが作成されます。

materialized_view

作成するマテリアライズド・ビューの名前を指定します。Oracle は、マテリアライズド・ビューを格納するための表、ビューおよび索引の名前を生成する際、マテリアライズド・ビュー名に接頭辞または接尾辞を追加します。

OF *object_type*

OF *type_name* 句を指定すると、*object_type* 型のオブジェクト・マテリアライズド・ビューを明示的に作成できます。

参照： `OF type_name` 句の詳細は、15-59 ページの「CREATE TABLE」の「[object_table](#) 句」を参照してください。

scoped_table_ref_constraint

SCOPE FOR 句を使用すると、参照の有効範囲を 1 つの表 *scope_table_name* に制限できます。REF 列または属性の値は、(REF 列と同じ型の) オブジェクト・インスタンスが格納されている *scope_table_name* 内のオブジェクトを指します。

参照： 詳細は、7-16 ページの「[REF 列の有効範囲制約](#)」を参照してください。

ON PREBUILT TABLE 句

ON PREBUILT TABLE 句を指定すると、既存の表を再初期化したマテリアライズド・ビューとして登録できます。データ・ウェアハウス環境において、大きいマテリアライズド・ビューを登録する場合に有効です。その表は、結果マテリアライズド・ビューと同じ名前で、同じスキーマにある必要があります。

マテリアライズド・ビューが削除されると、その既存の表は、1 つの表としての元の形に戻ります。

注意： この句は、表オブジェクトが副問合せの具体化を反映することを前提としています。マテリアライズド・ビューがそのマスター表のデータを正しく反映することを保証するために、この前提が満たされていることを確認することをお勧めします。

WITH REDUCED PRECISION 表またはマテリアライズド・ビュー列の精度が、副問合せで戻される精度と一致しない場合の精度の低下を認める場合は、WITH REDUCED PRECISION を指定します。

WITHOUT REDUCED PRECISION 表またはマテリアライズド・ビュー列の精度を副問合せによって戻された精度と一致させるか、または操作を中断するには、WITHOUT REDUCED PRECISION を指定します。これはデフォルト値です。

***prebuilt_table_clause* の制限事項：**

- *subquery* の各列の別名は、*table_name* の列に対応し、対応する列のデータ型が一致している必要があります。
- この句を指定する場合、非管理列 (*subquery* で参照されない列) にデフォルト値も指定しないかぎり、その列に NOT NULL 制約は指定できません。

参照： 14-27 ページの「[事前作成したマテリアライズド・ビューの作成例](#)」を参照してください。

physical_properties_clause

physical_properties_clause の構成要素は、マテリアライズド・ビューと表に対して同一のセマンティクスを持ちます。ただし、次の項で説明する例外および追加事項があります。

***physical_properties_clause* の制限事項：** マテリアライズド・ビューに対しては、`ORGANIZATION EXTERNAL` を指定できません。

segment_attributes_clause

segment_attributes_clause を使用すると、`PCTFREE`、`PCTUSED`、`INITRANS`、`MAXTRANS` パラメータの値（`USING INDEX` 句で使用する場合は `INITRANS` および `MAXTRANS` パラメータのみ）、およびマテリアライズド・ビューの記憶特性の設定、表領域の割当て、ロギングが発生するかどうかを指定できます。

TABLESPACE 句 マテリアライズド・ビューを作成する表領域を指定します。この句を指定しないと、マテリアライズド・ビューを含むスキーマのデフォルト表領域内にマテリアライズド・ビューが作成されます。

参照：

- デフォルト値を含む *physical_attributes_clause* のパラメータの詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- デフォルト値を含む *storage_clause* の詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

logging_clause `LOGGING` または `NOLOGGING` を指定して、マテリアライズド・ビューのロギング特性を設定します。デフォルトは、マテリアライズド・ビューが存在する表領域のロギング特性です。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

data_segment_compression

data_segment_compression 句を使用すると、ディスクおよびメモリーの使用量を削減するために、データ・セグメントを圧縮するかどうかを指定できます。`COMPRESS` キーワードを使用すると、データ・セグメントの圧縮が使用可能になります。`NOCOMPRESS` キーワードを使用すると、データ・セグメントの圧縮が使用禁止になります。

参照： データ・セグメントの圧縮の詳細は、15-29 ページの「`CREATE TABLE`」の「[data_segment_compression](#)」を参照してください。

index_org_table_clause

ORGANIZATION INDEX を指定すると、索引構成マテリアライズド・ビューを作成できます。このマテリアライズド・ビューでは、データ行は、マテリアライズド・ビューの主キーに定義した索引に格納されます。次のようなマテリアライズド・ビューの索引構成を指定できます。

- 読み専用および更新可能なオブジェクト・マテリアライズド・ビュー。マスター表に主キーが含まれている必要があります。
- 読み専用および更新可能な主キーに基づくマテリアライズド・ビュー。
- 読み専用 ROWID のマテリアライズド・ビュー。

index_org_table_clause のキーワードおよびパラメータは、CREATE TABLE と同じセマンティクスです。次の制限事項があります。

参照： 詳細は、15-30 ページの「CREATE TABLE」の「[*index_org_table_clause*](#)」を参照してください。

索引構成マテリアライズド・ビューの制限事項：

- CREATE MATERIALIZED VIEW の CACHE、NOCACHE、CLUSTER または ON PREBUILT TABLE 句は指定できません。
- *index_org_table_clause* には次の制限事項があります。
 - *mapping_table_clauses* を指定できません。
 - 複合主キーに基づくマテリアライズド・ビューのみに COMPRESS を指定できます。単一主キーと複合主キーのいずれかに基づくマテリアライズド・ビューに NOCOMPRESS を指定できます。

CLUSTER 句

ORGANIZATION CLUSTER 句を指定すると、指定したクラスタの一部としてマテリアライズド・ビューを作成できます。クラスタ化マテリアライズド・ビューは、クラスタの領域割当てを使用します。したがって、CLUSTER 句で物理属性または TABLESPACE 句を指定しないでください。

クラスタ化マテリアライズド・ビューの制限事項： ORGANIZATION CLUSTER を指定する場合、*table_partitioning_clauses* (*materialized_view_props* 内) は指定できません。

materialized_view_props

これらのプロパティ句を使用すると、既存の表に基づかないマテリアライズド・ビューを定義できます。既存の表に基づくマテリアライズド・ビューを作成するには、ON PREBUILT TABLE 句を使用します。

column_properties

column_properties 句を指定して、LOB、ネストした表、VARRAY または XMLType の列の記憶特性を指定します。

マテリアライズド・ビューの列の制限事項： *object_type_col_properties* は、マテリアライズド・ビューには関連しません。

参照： この句のパラメータの指定の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

table_partitioning_clauses

table_partitioning_clauses を使用すると、マテリアライズド・ビューが、指定された範囲の値またはハッシュ・ファンクションでパーティション化されます。マテリアライズド・ビューのパーティション化は、表のパーティション化と同じです

参照： 詳細は、15-42 ページの「[CREATE TABLE](#)」の「[table_partitioning_clauses](#)」を参照してください。

CACHE | NOCACHE

アクセス頻度の高いデータについて、CACHE を指定すると、全表スキャンの実行時にこの表用に取り出された各ブロックは、バッファ・キャッシュ内の最低使用頻度（LRU）リストの最高使用頻度側に置かれます。この属性は、小規模な参照表で有効です。NOCACHE を指定すると、ブロックは LRU リストの最低使用頻度側に置かれます。

注意： NOCACHE は、*storage_clause* に KEEP を指定したマテリアライズド・ビューには、影響しません。

参照： CACHE または NOCACHE の指定の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

parallel_clause

parallel_clause を使用すると、マテリアライズド・ビューへのパラレル操作をサポートするかどうかを指定できます。作成後にマテリアライズド・ビューに対する問合せおよび DML のデフォルトの並列度を設定します。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer integer には、パラレル操作で使用するパラレル・スレッド数である **並列度** を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、integer に値を指定する必要はありません。

参照： 詳細は、15-51 ページの CREATE TABLE の「[parallel_clause の注意事項](#)」を参照してください。

build_clause

build_clause を指定すると、マテリアライズド・ビューをいつ移入するかを指定できます。

IMMEDIATE IMMEDIATE を指定すると、マテリアライズド・ビューはすぐに移入されます。これはデフォルト値です。

DEFERRED DEFERRED を指定すると、マテリアライズド・ビューは次の REFRESH 操作で移入されます。最初の（遅延）リフレッシュは、常に、完全リフレッシュである必要があります。それ以前のマテリアライズド・ビューの値は UNUSABLE であるため、クエリー・リライトには使用できません。

USING INDEX 句

USING INDEX 句を使用すると、マテリアライズド・ビューのデータをメンテナンスするために使用されるデフォルトの索引の INITRANS パラメータ、MAXTRANS パラメータおよび STORAGE パラメータの値を変更できます。USING INDEX が設定されていない場合、この索引にはデフォルト値が使用されます。マテリアライズド・ビューの増分リフレッシュ（高速リフレッシュ）を高速に処理するには、デフォルトの索引が使用されます。

USING INDEX 句の制限事項： この句では、PCTUSED パラメータは指定できません。

USING NO INDEX 句

USING NO INDEX 句を指定すると、デフォルトの索引の作成を抑制できます。CREATE INDEX 文を使用することによって、代替する索引を明示的に作成できます。USING NO INDEX を指定し、増分リフレッシュ（REFRESH FAST）でマテリアライズド・ビューを作成する場合は、このような索引を作成する必要があります。

`create_mv_refresh`

`create_mv_refresh`を使用すると、マテリアライズド・ビューのデフォルトのリフレッシュ方法、リフレッシュ・モードおよびリフレッシュ時刻を指定できます。マテリアライズド・ビューのマスター表が変更された場合、マテリアライズド・ビューのデータを更新し、その時点でマスター表にあるデータを正確に反映させる必要があります。この句によって、自動的にマテリアライズド・ビューをリフレッシュする時刻をスケジューリングし、リフレッシュの方法およびモードを指定できます。

注意： この句では、デフォルトのリフレッシュ・オプションのみを設定します。リフレッシュを実際に実装する手順は、『Oracle9i アドバンスド・レプリケーション』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

参照： 14-28 ページの「マテリアライズド・ビューの定期的リフレッシュ例:」および 14-28 ページの「マテリアライズド・ビューの自動リフレッシュ時刻の例:」を参照してください。

FAST 句

FAST によって、増分リフレッシュ方法を指定します。これはマスター表に対して行った変更に従ってリフレッシュを行います。この変更は、マスター表に関連付けられたマテリアライズド・ビュー・ログ（従来型 DML 変更の場合）またはダイレクト・ローダー・ログ（ダイレクト・パス・インサート操作の場合）に格納されます。

REFRESH FAST を指定すると、マテリアライズド・ビューのマスター表のマテリアライズド・ビュー・ログが存在していない場合に、CREATE 文は正常に実行されません。（ダイレクト・パス・インサートが行われると、ダイレクト・ローダー・ログが自動的に作成されます。手動での操作は必要ありません。）

従来型 DML の変更の場合も、ダイレクト・パス・インサート操作の場合も、他の条件によって、高速リフレッシュへのマテリアライズド・ビューの適応性が制限されることがあります。

定義する副問合せに分析ファンクションが含まれている場合、マテリアライズド・ビューは高速リフレッシュに適応しません。

参照：

- レプリケーション環境における高速リフレッシュの制限については、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウス環境における高速リフレッシュの制限については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- 6-9 ページの「[分析ファンクション](#)」を参照してください。
- 14-29 ページの「[高速リフレッシュ可能なマテリアライズド・ビューの作成例](#)」を参照してください。

COMPLETE 句

COMPLETE によって、完全リフレッシュ方法を指定します。これは、マテリアライズド・ビューを定義する副問合せを実行することによって実装されます。完全リフレッシュを要求すると、高速リフレッシュが実行可能であっても、完全リフレッシュが実行されます。

FORCE 句

FORCE によって、リフレッシュ時、高速リフレッシュが可能な場合は高速リフレッシュを実行し、そうでない場合は完全リフレッシュを実行することを指定します。リフレッシュ方法（FAST、COMPLETE または FORCE）を指定しないと、デフォルトで FORCE が指定されます。

ON COMMIT 句

ON COMMIT によって、マテリアライズド・ビューのマスター表に対するトランザクションをコミットするときは、必ず高速リフレッシュが実行されるように指定します。この句を指定すると、リフレッシュ操作がコミット処理の一部として行われるため、コミットの完了に時間がかかります。

ON COMMIT のリフレッシュの制限事項： この句は、オブジェクト型を含むマテリアライズド・ビューについてはサポートしません。

参照：『Oracle9i アドバンスド・レプリケーション』および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

ON DEMAND 句

ON DEMAND によって、マテリアライズド・ビューが、3 つの DBMS_MVIEW プロシージャのいずれかのコールによる要求でリフレッシュされることを指定します。ON COMMIT および ON DEMAND のどちらも指定しなかった場合、ON DEMAND がデフォルトになります。

参照：

- これらのプロシージャの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- REFRESH ON DEMAND を指定することによって作成できるマテリアライズド・ビューのタイプについては、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

ON COMMIT または ON DEMAND を指定した場合、START WITH または NEXT を同時に指定できません。

START WITH 句

最初の自動リフレッシュ時刻を表す日付式を指定します。

NEXT 句

自動リフレッシュの間隔を計算するための日付式を指定します。

START WITH 値および NEXT 値は、将来の時刻に評価される値です。START WITH 値を省略した場合、Oracle はマテリアライズド・ビューの作成時に NEXT 式を評価することによって、最初の自動リフレッシュ時刻を判断します。START WITH 値を指定し、NEXT 値を指定しない場合、Oracle は 1 回のみマテリアライズド・ビューをリフレッシュします。START WITH 値および NEXT 値のどちらも指定しない場合、または `create_mv_refresh` を指定しない場合は、マテリアライズド・ビューは自動リフレッシュされません。

WITH PRIMARY KEY 句

WITH PRIMARY KEY によって、主キー・マテリアライズド・ビューを作成します。これはデフォルトであり、WITH ROWID に記述される値を除き、すべての場合に使用される必要があります。主キー・マテリアライズド・ビューによって、マテリアライズド・ビューのマスター表は、高速リフレッシュに対するマテリアライズド・ビューの適応性に影響されずに再編成されます。マスター表には、使用可能な主キー制約が定義されている必要があります。

主キー・マテリアライズド・ビューの制限事項： この句は、オブジェクト・マテリアライズド・ビューには指定できません。WITH OBJECT ID を指定してマテリアライズされたオブジェクトは暗黙的にリフレッシュされます。

参照： 主キー・マテリアライズド・ビューの詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。14-27 ページの「[主キー・マテリアライズド・ビューの作成例:](#)」も参照してください。

WITH ROWID 句

WITH ROWID によって、ROWID マテリアライズド・ビューを作成します。ROWID のマテリアライズド・ビューは、リリース 8.0 より前のマスター表と互換性があります。

マテリアライズド・ビューがマスター表の主キー列をすべて含まない場合、ROWID マテリアライズド・ビューを使用することができます。ROWID マテリアライズド・ビューは、単一表を基にしている必要があり、次のいずれも含むことができません。

- 固有ファンクションまたは集計ファンクション
- GROUP BY 句または CONNECT BY 句
- 副問合せ
- 結合
- 集合演算

完全リフレッシュが行われるまでは、マスター表の再編成後に、ROWID マテリアライズド・ビューは高速リフレッシュされません。

ROWID マテリアライズド・ビューの制限事項： この句は、オブジェクト・マテリアライズド・ビューには指定できません。WITH OBJECT ID を指定してマテリアライズドされたオブジェクトは暗黙的にリフレッシュされます。

参照： 14-26 ページの「マテリアライズド集計ビューの作成例：」および 14-27 ページの「ROWID マテリアライズド・ビューの作成例：」を参照してください。

USING ROLLBACK SEGMENT 句

マテリアライズド・ビューのリフレッシュ中に使用するリモート・ロールバック・セグメントを指定します。使用するロールバック・セグメント名を `rollback_segment` に指定します。

自動 UNDO モードでは、ロールバック・セグメントではなく UNDO 表領域が使用されるため、データベースが自動 UNDO モードの場合、この句は無効です。

DEFAULT DEFAULT は、使用するロールバック・セグメントが自動的に選択されることを指定します。DEFAULT を指定した場合、`rollback_segment` は指定できません。

DEFAULT は、マテリアライズド・ビューを変更（作成ではなく）する場合に有効です。

参照： 9-88 ページの「ALTER MATERIALIZED VIEW」を参照してください。

MASTER MASTER は、個々のマテリアライズド・ビュー用のリモート・マスター・サイトで使用されるリモート・ロールバック・セグメントを指定します。

LOCAL LOCAL は、マテリアライズド・ビューが含まれているローカル・リフレッシュ・グループで使用されるリモート・ロールバック・セグメントを指定します。

参照： DBMS_REFRESH パッケージを使用するローカル・マテリアライズド・ビューのロールバック・セグメントの指定については、『Oracle9i アドバンスド・レプリケーション』を参照してください。

MASTER または LOCAL を指定しない場合、デフォルトで LOCAL が使用されます。
`rollback_segment` を指定しない場合、使用するロールバック・セグメントが自動的に選択されます。

1 つのマスター・ロールバック・セグメントは、各マテリアライズド・ビューに格納され、マテリアライズド・ビューの作成およびリフレッシュ時にスキャンされます。複合マテリアライズド・ビューの場合、マスター・ロールバック・セグメントの指定は無視されます。

参照： 14-29 ページの「マテリアライズド・ビューにロールバック・セグメントを指定する例:」を参照してください。

NEVER REFRESH 句

NEVER REFRESH を指定して、Oracle のリフレッシュ・メカニズムまたはパッケージ・プロシージャによるマテリアライズド・ビューのリフレッシュを行わないようにします。このようなプロシージャから発行されるマテリアライズド・ビューの REFRESH 文は、無視されます。この句を戻すには、ALTER MATERIALIZED VIEW ... REFRESH 文を発行する必要があります。

FOR UPDATE 句

FOR UPDATE によって、副問合せ、主キー、オブジェクトまたは ROWID マテリアライズド・ビューを更新します。アドバンスド・レプリケーションと組み合わせて使用した場合、更新はマスターにも影響します。

参照： 『Oracle9i アドバンスド・レプリケーション』を参照してください。

QUERY REWRITE 句

QUERY REWRITE 句によって、マテリアライズド・ビューがクエリー・リライトで使用できるかどうかを指定します。

ENABLE 句

ENABLE を指定して、クエリー・リライトでマテリアライズド・ビューを使用可能にします。

クエリー・リライトを使用可能にする場合の制限事項：

- マテリアライズド・ビューのすべてのユーザー定義ファンクションが DETERMINISTIC である場合のみ、クエリー・リライトを使用可能にできます。

- 文内の式が反復可能な場合のみ、クエリー・リライトを使用可能にできます。たとえば、CURRENT_TIME、USER、順序値（たとえば、CURRVAL または NEXTVAL 疑似列）、または SAMPLE 句（マテリアライズド・ビューの変更内容と異なる行をサンプルとして抽出します）を含めることはできません。

注意：

- クエリー・リライトでのマテリアライズド・ビューの使用は、デフォルトでは無効です。この句を指定して、マテリアライズド・ビューをクエリー・リライトに適用できるようにする必要があります。
 - マテリアライズド・ビューを作成した後は、必ず分析してください。クエリー・リライトを最適化するために、ANALYZE 操作で生成した統計が必要です。
-

参照：

- クエリー・リライトの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- 13-49 ページの「[CREATE FUNCTION](#)」を参照してください。

DISABLE 句

DISABLE を指定して、マテリアライズド・ビューをクエリー・リライトで使用禁止にします。ただし、使用禁止にされたマテリアライズド・ビューをリフレッシュすることはできません。

AS subquery

定義するマテリアライズド・ビューの副問合せを指定します。マテリアライズド・ビューの作成時に、この副問合せが実行され、実行結果がマテリアライズド・ビューに格納されます。この副問合せは、有効な SQL 副問合せである必要があります。ただし、すべての問合せが高速リフレッシュされるわけではなく、すべての副問合せがクエリー・リライトに使用できるわけでもありません。

マテリアライズド・ビューの問合せの定義に関する注意事項：

- BUILD DEFERRED を指定する場合、すぐには副問合せの定義は実行されません。
- マテリアライズド・ビュー副問合せを定義する FROM 句に指定する各表およびビューを、それを定義しているスキーマで修飾することをお勧めします。

参照： その他の注意事項については、15-58 ページの「[CREATE TABLE](#)」の「[AS subquery](#)」を参照してください。

マテリアライズド・ビューの副問合せの定義に関する制限事項：

- ユーザー SYS が所有する表、ビューまたはマテリアライズド・ビューからマテリアライズド・ビューの副問合せの定義を選択できますが、このようなマテリアライズド・ビューに対して QUERY REWRITE は使用できません。
- GROUP BY 句を含むマテリアライズド結合ビューおよびマテリアライズド集計ビューは、索引構成表からは選択できません。
- マテリアライズド・ビューに LONG データ型の列を含めることはできません。
- 一時表に対しては、マテリアライズド・ビュー・ログを作成できません。そのため、副問合せの定義が一時表を参照する場合、マテリアライズド・ビューは高速リフレッシュに適応せず、同じ文で QUERY REWRITE を指定することもできません。
- 副問合せを定義する FROM 句が他のマテリアライズド・ビューを参照する場合、この文で作成するマテリアライズド・ビューのリフレッシュの前に、副問合せの定義が参照するマテリアライズド・ビューを常にリフレッシュしておく必要があります。

クエリー・リライトが可能なマテリアライズド・ビューを作成する場合、次の制限があります。

- 副問合せには、ROWNUM、USER、SYSDATE、リモート表、順序の他にも、データベースまたはパッケージ状態の書込みまたは読込みを行う PL/SQL ファンクションに対する（直接またはビューを介しての）参照を含めることはできません。
- マテリアライズド・ビューおよびマテリアライズド・ビューのマスター表はリモートにできません。

マテリアライズド・ビュー・ログを使用した高速リフレッシュを実行する場合は、いくつかの制限事項があります。

参照：

- データ・ウェアハウスの制限事項は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- レプリケーションの制限事項は、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- 14-26 ページの「マテリアライズド結合ビューの作成例：」、14-27 ページの「副問合せマテリアライズド・ビューの作成例：」および 14-30 ページの「ネステッド・マテリアライズド・ビューの作成例：」を参照してください。

例

次の例は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」の「例」で作成したマテリアライズド・ログを必須とします。

マテリアライズド集計ビューの作成例： 次の文は、サンプル表 `sh.sales` にマテリアライズド集計ビューを作成して移入し、デフォルトのリフレッシュ方法、モードおよび時刻を指定します。この文は、ここに示されている 2 つのログおよび 14-38 ページの「[マテリアライズド・ビュー・ログの作成例](#)」で作成されたマテリアライズド・ビュー・ログを使用します。

```
CREATE MATERIALIZED VIEW LOG ON times
  WITH ROWID, SEQUENCE (time_id, calendar_year)
  INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON products
  WITH ROWID, SEQUENCE (prod_id)
  INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW sales_mv
  BUILD IMMEDIATE
  REFRESH FAST ON COMMIT
  AS SELECT t.calendar_year, p.prod_id,
    SUM(s.amount_sold) AS sum_sales
  FROM times t, products p, sales s
  WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
  GROUP BY t.calendar_year, p.prod_id;
```

マテリアライズド結合ビューの作成例： 次の文は、サンプル・スキーマ `sh` の表を使用して、マテリアライズド集計ビュー `sales_by_month_by_state` を作成して移入します。マテリアライズド・ビューは、この文が正しく実行されるとすぐに移入されます。デフォルトで、次のマテリアライズド・ビューの問合せを再実行することによって、後続のリフレッシュが行われます。

```
CREATE MATERIALIZED VIEW sales_by_month_by_state
  TABLESPACE example
  PARALLEL 4
  BUILD IMMEDIATE
  REFRESH COMPLETE
  ENABLE QUERY REWRITE
  AS SELECT t.calendar_month_desc, c.cust_state_province,
    SUM(s.amount_sold) AS sum_sales
  FROM times t, sales s, customers c
  WHERE s.time_id = t.time_id AND s.cust_id = c.cust_id
  GROUP BY t.calendar_month_desc, c.cust_state_province;
```


事前作成したマテリアライズド・ビューの作成例： 次の文は、既存の集計表 sales_sum_table に対するマテリアライズド集計ビューを作成します。

```
CREATE TABLE sales_sum_table
    (month VARCHAR2(8), state VARCHAR2(40), sales NUMBER(10,2));

CREATE MATERIALIZED VIEW sales_sum_table
    ON PREBUILT TABLE WITH REDUCED PRECISION
    ENABLE QUERY REWRITE
    AS SELECT t.calendar_month_desc AS month,
        c.cust_state_province AS state,
        SUM(s.amount_sold) AS sales
    FROM times t, customers c, sales s
    WHERE s.time_id = t.time_id AND s.cust_id = c.cust_id
    GROUP BY t.calendar_month_desc, c.cust_state_province;
```

この例では、マテリアライズド・ビューは事前作成表と同じ名前を持ち、かつ事前作成表と同じデータ型で同じ数の列を持ちます。WITH REDUCED PRECISION 句は、マテリアライズド・ビュー列の精度と副問合せによって戻された値の精度の間に指定できます。

副問合せマテリアライズド・ビューの作成例： 次の文は、リモート・データベースの sh スキーマの countries および customers 表を基にした副問合せマテリアライズド・ビューを作成します。

```
CREATE MATERIALIZED VIEW foreign_customers FOR UPDATE
    AS SELECT * FROM sh.customers@remote cu
    WHERE EXISTS
        (SELECT * FROM sh.countries@remote co
         WHERE co.country_id = cu.country_id);
```

主キー・マテリアライズド・ビューの作成例： 次の文は、サンプル表 oe.product_information に主キー・マテリアライズド・ビュー catalog を作成します。

```
CREATE MATERIALIZED VIEW catalog
    REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 1/4096
    WITH PRIMARY KEY
    AS SELECT * FROM product_information;
```

ROWID マテリアライズド・ビューの作成例： 次の文は、サンプル表 oe.orders に ROWID マテリアライズド・ビューを作成します。

```
CREATE MATERIALIZED VIEW order_data REFRESH WITH ROWID
    AS SELECT * FROM orders;
```

マテリアライズド・ビューの定期的リフレッシュ例： 次の文は、主キー・マテリアライズド・ビュー emp_data を作成し、サンプル表 hr.employees のデータを移入します。

```
CREATE MATERIALIZED VIEW LOG ON employees
WITH PRIMARY KEY
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW emp_data
PCTFREE 5 PCTUSED 60
TABLESPACE example
STORAGE (INITIAL 50K NEXT 50K)
REFRESH FAST NEXT sysdate + 7
AS SELECT * FROM employees;
```

この文では、START WITH パラメータが指定されていないため、現行の SYSDATE を使用して NEXT 値が評価され、最初の自動リフレッシュ時刻が判断されます。従業員表のマテリアライズド・ビュー・ログが作成されたため、マテリアライズド・ビュー作成の 7 日後から、7 日ごとにマテリアライズド・ビューの高速リフレッシュが実行されます。

マテリアライズド・ビューが高速リフレッシュを行うための条件と一致するため、高速リフレッシュが行われます。また、前述の文では、Oracle がマテリアライズド・ビューをメンテナンスするために使用する表の記憶特性も設定しています。

マテリアライズド・ビューの自動リフレッシュ時刻の例： 次の文は、remote および local データベースの従業員表を問い合わせる複合マテリアライズド・ビュー all_customers を作成します。

```
CREATE MATERIALIZED VIEW all_customers
PCTFREE 5 PCTUSED 60
TABLESPACE example
STORAGE (INITIAL 50K NEXT 50K)
USING INDEX STORAGE (INITIAL 25K NEXT 25K)
REFRESH START WITH ROUND(SYSDATE + 1) + 11/24
NEXT NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24
AS SELECT * FROM sh.customers@remote
UNION
SELECT * FROM sh.customers@local;
```

このマテリアライズド・ビューは、翌日の午前 11:00 に自動的にリフレッシュされ、その後は、毎週月曜日の午後 3:00 にリフレッシュされます。デフォルトのリフレッシュ方法は、FORCE です。all_emps には UNION 演算子が含まれますが、UNION 演算子は高速リフレッシュでサポートされていないため、自動的に完全リフレッシュが実行されます。

前述の文では、マテリアライズド・ビューおよびこのマテリアライズド・ビューをメンテナンスするために使用される索引の記憶特性を次のように設定しています。

- 最初に、マテリアライズド・ビューの1番目と2番目のエクステンツのサイズをそれぞれ 50KB に設定します。
- 次の USING INDEX 句付きの指定では、索引の1番目と2番目のエクステンツのサイズをそれぞれ 25KB に変更します。

マテリアライズド・ビューにロールバック・セグメントを指定する例： 次の文では、リモート・マスターにあるロールバック・セグメント master_seg およびマテリアライズド・ビューを含むローカル・リフレッシュ・グループに対するロールバック・セグメント snap_seg を使用して、主キー・マテリアライズド・ビュー sales_emp を作成します。

注意： 自動 UNDO モードでは、ロールバック・セグメントではなく UNDO 表領域が使用されるため、データベースが自動 UNDO モードの場合、この例は適用されません。

```
CREATE MATERIALIZED VIEW sales_emp
  REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 7
  USING MASTER ROLLBACK SEGMENT master_seg
  LOCAL ROLLBACK SEGMENT snap_seg
  AS SELECT * FROM bar;
```

次の文には誤りがあり、DEFAULT ロールバック・セグメントを使用してセグメント名を指定しているため、エラーが生成されます。

```
/* The following statement is invalid. */
CREATE MATERIALIZED VIEW order_mv
  REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 7
  USING DEFAULT ROLLBACK SEGMENT mv_seg
  AS SELECT * FROM orders;
```

高速リフレッシュ可能なマテリアライズド・ビューの作成例： 次の文は、product_information および inventories 表から WHERE 条件で戻される行を制限する UNION 集合演算子を使用して、サンプル・スキーマ oe の表 order_items から列を選択する高速リフレッシュが可能なマテリアライズド・ビューを作成します。order_items および product_information のマテリアライズド・ビュー・ログは、14-38 ページの CREATE MATERIALIZED VIEW LOG の「例」で作成されたものです。次の例には、oe.inventories のマテリアライズド・ビュー・ログが必須です。

```
CREATE MATERIALIZED VIEW LOG ON inventories
  WITH (quantity_on_hand);
```

```
CREATE MATERIALIZED VIEW warranty_orders REFRESH FAST AS
  SELECT order_id, line_item_id, product_id FROM order_items o
     WHERE EXISTS
       (SELECT * FROM inventories i WHERE o.product_id = i.product_id
        AND i.quantity_on_hand IS NOT NULL)
  UNION
  SELECT order_id, line_item_id, product_id FROM order_items
     WHERE quantity > 5;
```

このマテリアライズド・ビューには、order_items (product_id を結合列とする) および inventories (quantity_on_hand をフィルタ列とする) で定義されたマテリアライズド・ビュー・ログが必須です。14-39 ページの「マテリアライズド・ビュー・ログにフィルタ列を指定する例:」および「マテリアライズド・ビュー・ログに結合列を指定する例:」を参照してください。

ネステッド・マテリアライズド・ビューの作成例: 次の例は、前述の例のマテリアライズド・ビューをマスター表として使用し、サンプル・スキーマ oe の特定の販売員で構成されるマテリアライズド・ビューを作成します。

```
CREATE MATERIALIZED VIEW my_warranty_orders
  AS SELECT w.order_id, w.line_item_id, o.order_date
  FROM warranty_orders w, orders o
  WHERE o.order_id = w.order_id
     AND o.sales_rep_id = 165;
```

CREATE MATERIALIZED VIEW LOG

用途

CREATE MATERIALIZED VIEW LOG 文を使用すると、マテリアライズド・ビューのマスター表に関連する表**マテリアライズド・ビュー・ログ**を作成できます。

注意： キーワード SNAPSHOT は、MATERIALIZED VIEW のかわりに下位互換用にサポートされています。

マスター表のデータで DML が変更された場合、Oracle は変更を記述する行をマテリアライズド・ビュー・ログに格納し、そのマテリアライズド・ビュー・ログを使用して、マスター表を基にしたマテリアライズド・ビューをリフレッシュします。このプロセスを、増分リフレッシュまたは**高速リフレッシュ**といいます。マテリアライズド・ビュー・ログがない場合、マテリアライズド・ビュー問合せが再実行され、マテリアライズド・ビューがリフレッシュされます。このプロセスが**完全リフレッシュ**です。通常、高速リフレッシュは、完全リフレッシュよりも高速に処理されます。

マテリアライズド・ビュー・ログは、マスター表と同一のスキーマ内のマスター・データベースにあります。マスター表に定義できるマテリアライズド・ビュー・ログは1つのみです。Oracle では、このマテリアライズド・ビュー・ログを使用して、マスター表に基づく高速リフレッシュが可能なすべてのマテリアライズド・ビューに対して、高速リフレッシュを実行します。

マテリアライズド結合ビュー（結合を含むマテリアライズド・ビュー）を高速リフレッシュする場合、マテリアライズド・ビューが参照するそれぞれのマスター表に対するマテリアライズド・ビュー・ログを作成する必要があります。

参照：

- マテリアライズド・ビューの概要は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」、『Oracle9i データベース概要』、『Oracle9i データ・ウェアハウス・ガイド』および『Oracle9i アドバンスド・レプリケーション』を参照してください。
- マテリアライズド・ビュー・ログの変更については、9-108 ページの「[ALTER MATERIALIZED VIEW LOG](#)」を参照してください。
- マテリアライズド・ビュー・ログの削除については、16-83 ページの「[DROP MATERIALIZED VIEW LOG](#)」を参照してください。
- ダイレクト・ローダー・ログの使用については、『Oracle9i データベース概要』および『Oracle9i データベース・ユーティリティ』を参照してください。

前提条件

マテリアライズド・ビュー・ログを作成するために必要な権限は、マテリアライズド・ビュー・ログに関連付けられた基礎となるオブジェクトの作成に必要な権限と直接関連しています。

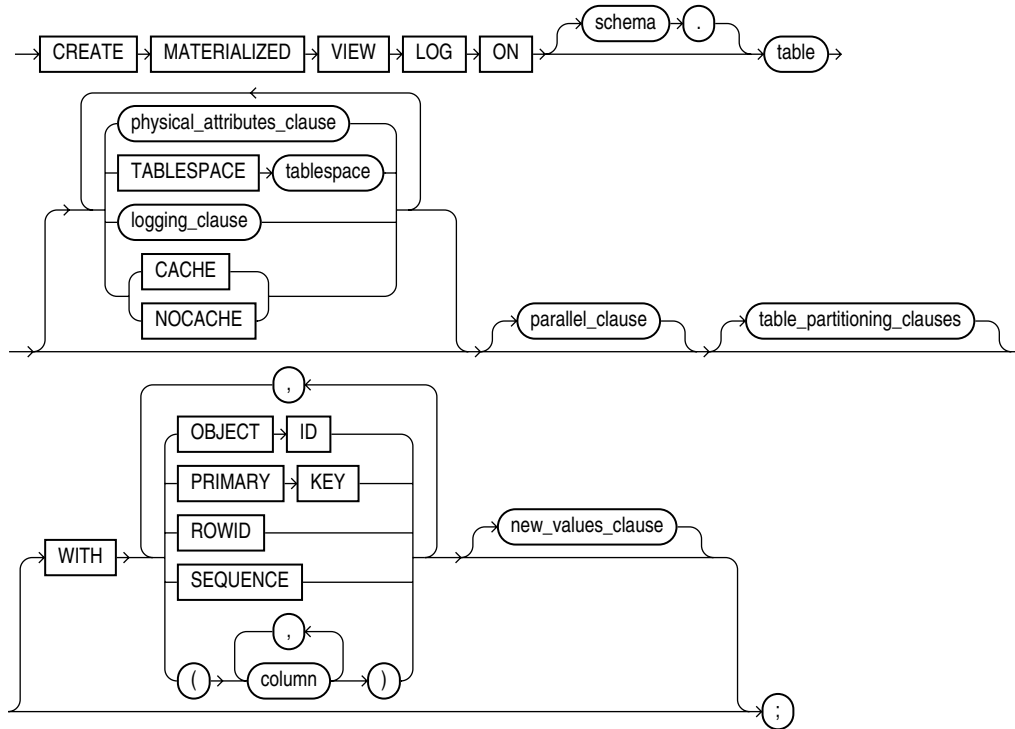
- マスター表を所有しているユーザーに CREATE TABLE 権限がある場合、関連するマテリアライズド・ビュー・ログを作成できます。
- 他のユーザーのスキーマ内に表のマテリアライズド・ビュー・ログを作成する場合は、CREATE ANY TABLE 権限、COMMENT ANY TABLE 権限、およびマスター表に対する SELECT 権限または SELECT ANY TABLE 権限が必要です。

どちらの場合も、マテリアライズド・ビュー・ログの所有者には、マテリアライズド・ビュー・ログを格納するための表領域に十分な割当て制限または UNLIMITED TABLESPACE システム権限が必要です。

参照： マテリアライズド・ビュー・ログを作成する場合の前提条件については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

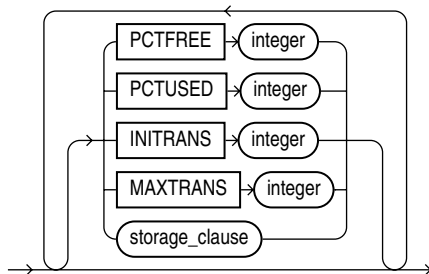
構文

create_materialized_vw_log::=



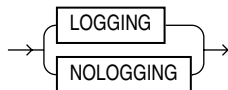
(14-10 ページの「[physical_attributes_clause::=](#)」、14-34 ページの「[logging_clause::=](#)」、14-34 ページの「[parallel_clause::=](#)」、15-42 ページの「[table_partitioning_clauses](#)」(CREATE TABLE の一部) および 14-34 ページの「[new_values_clause::=](#)」を参照)

physical_attributes_clause::=

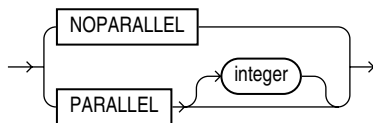


(7-51 ページの「[storage_clause](#)」を参照)

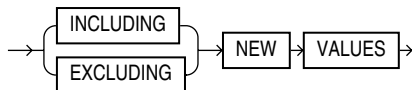
logging_clause::=



parallel_clause::=



new_values_clause::=



セマンティクス

schema

マテリアライズド・ビュー・ログのマスター表が定義されているスキーマを指定します。
schema を省略した場合、マスター表は自分のスキーマ内に定義されているものとみなされます。マテリアライズド・ビュー・ログは、マスター表のスキーマ内に作成されます。なお、ユーザー `sys` のスキーマ内の表に対しては、マテリアライズド・ビュー・ログを作成できません。

table

マテリアライズド・ビュー・ログを作成するマスター表の名前を指定します。

マテリアライズド・ビュー・ログのマスター表の制限事項： 一時表またはビューに対しては、マテリアライズド・ビュー・ログを作成できません。

参照： 14-38 ページの「[マテリアライズド・ビュー・ログの作成例:](#)」を参照してください。

physical_attributes_clause

physical_attributes_clause を使用して、マテリアライズド・ビュー・ログにおける物理特性および記憶特性の値の設定をします。

参照：

- *physical_attributes_clause* のパラメータの詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- デフォルト値を含む *storage_clause* の詳細は、7-51 ページの「[storage_clause](#)」を参照してください。

TABLESPACE 句

マテリアライズド・ビュー・ログを作成する表領域を指定します。この句を省略した場合、マテリアライズド・ビュー・ログのスキーマのデフォルト表領域内にマテリアライズド・ビュー・ログが作成されます。

logging_clause

LOGGING または NOLOGGING を指定して、マテリアライズド・ビュー・ログのロギング特性を設定します。デフォルトは、マテリアライズド・ビュー・ログが存在する表領域のロギング特性です。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

CACHE | NOCACHE

アクセス頻度の高いデータについて、CACHE を指定すると、全表スキャンの実行時にこのログ用に取り出された各ブロックは、バッファ・キャッシュ内の最低使用頻度（LRU）リストの最高使用頻度側に置かれます。この属性は、小規模な参照表で有効です。

NOCACHE を指定すると、ブロックは LRU リストの最低使用頻度側に置かれます。デフォルトは NOCACHE です。

注意： NOCACHE は、*storage_clause* に KEEP を指定したマテリアライズド・ビュー・ログには、影響しません。

参照： CACHE または NOCACHE の指定の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

parallel_clause

parallel_clause によって、パラレル操作でマテリアライズド・ビュー・ログをサポートするかどうかを指定できます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL NOPARALLEL を指定すると、シリアル実行が行われます。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である**並列度**を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

参照： 詳細は、15-51 ページの CREATE TABLE の「[parallel_clause の注意事項:](#)」を参照してください。

table_partitioning_clauses

table_partitioning_clauses を使用して、マテリアライズド・ビュー・ログが、指定された範囲の値またはハッシュ・ファンクションでパーティション化されることを指定できます。マテリアライズド・ビュー・ログのパーティション化は、表のパーティション化と同じです。

参照： 詳細は、15-42 ページの「CREATE TABLE」の「[table_partitioning_clauses](#)」を参照してください。

WITH 句

WITH 句を使用して、マスター内の行の更新時に、マテリアライズド・ビュー・ログに主キー、ROWID、オブジェクト ID またはこれらの行識別子の組合せを記録するかどうかを指定します。また、マテリアライズド・ビュー・ログに順序を追加し、レコードに対する追加の順序情報を提供することもできます。

この句を指定すると、マテリアライズド・ビュー・ログが**フィルタ列**（副問合せマテリアライズド・ビューが参照する主キー以外の列）または**結合列**（副問合せの WHERE 句で結合を定義する主キー以外の列）として参照する追加の列を記録するかどうかを指定できます。

この句を指定しない場合、または PRIMARY KEY、ROWID または OBJECT ID なしで句を指定する場合は、デフォルトで主キー値が格納されます。ただし、作成時に OBJECT ID または ROWID のみを指定した場合は、主キー値は暗黙的に格納されません。明示的に、またはデフォルトで作成された主キー・ログによって、主キー制約の追加の検証が実行されます。

OBJECT ID OBJECT ID によって、更新されるすべての行に対するシステム生成またはユーザー定義のオブジェクト識別子をマテリアライズド・ビュー・ログに記録することを指定します。

OBJECT ID の制限事項： OBJECT ID は、オブジェクト表のログの作成時のみに指定でき、記憶表用には指定できません。

PRIMARY KEY PRIMARY KEY によって、更新されるすべての行の主キーをマテリアライズド・ビュー・ログに記録することを指定します。

ROWID ROWID によって、更新されるすべての行の ROWID をマテリアライズド・ビュー・ログに記録することを指定します。

SEQUENCE SEQUENCE によって、追加の順序情報を提供する順序値をマテリアライズド・ビュー・ログに記録することを指定します。更新操作後の高速リフレッシュには、順序番号が必要です。

参照： マテリアライズド・ビュー・ログの順序番号の使用およびこの句の使用例については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

column 更新されるすべての行に対して、マテリアライズド・ビュー・ログに値を記録する列を指定します。通常、フィルタ列（マテリアライズド・ビューが参照する主キー以外の列）および結合列（副問合せの WHERE 句で結合を定義する主キー以外の列）を指定します。

WITH 句の制限事項：

- 各マテリアライズド・ビュー・ログに指定できるのは、PRIMARY KEY、ROWID、OBJECT ID をそれぞれ 1 回、および 1 つの列リストです。

- 主キー列は、マテリアライズド・ビュー・ログに暗黙的に記録されます。そのため、*column* が主キー列の 1 つを含む場合は、次の結合はいずれも指定できません。

```
WITH ... PRIMARY KEY ... (column)  
WITH ... (column) ... PRIMARY KEY  
WITH (column)
```

参照：

- マテリアライズド・ビュー・ログの値の、明示的および暗黙的な論理和については、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。
- フィルタ列および結合列については、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- 14-39 ページの「[マテリアライズド・ビュー・ログにフィルタ列を指定する例：](#)」および 14-39 ページの「[マテリアライズド・ビュー・ログに結合列を指定する例：](#)」を参照してください。

NEW VALUES 句

NEW VALUES 句によって、マテリアライズド・ビュー・ログの古い値と新しい値の両方を保存するかどうかを指定できます。

参照： 14-39 ページの「[マテリアライズド・ビュー・ログに新しい値を追加する例：](#)」を参照してください。

INCLUDING INCLUDING を指定すると、古い値と新しい値の両方を保存します。このログが単一表マテリアライズド集計ビューの表用で、マテリアライズド・ビューに高速リフレッシュを実行する場合、INCLUDING を指定してください。

EXCLUDING EXCLUDING を指定すると、ログに対する新しい値の記録を使用禁止にします。これはデフォルト値です。この句を使用すると、新しい値の記録によるオーバーヘッドを回避できます。ただし、この表に高速リフレッシュが可能な単一表マテリアライズド集計ビューが定義されている場合は、この句を使用しないでください。

例

マテリアライズド・ビュー・ログの作成例： 次の文は、物理特性および記憶特性を指定する oe.customers 表にマテリアライズド・ビュー・ログを作成します。

```
CREATE MATERIALIZED VIEW LOG ON customers  
  PCTFREE 5  
  TABLESPACE example  
  STORAGE (INITIAL 10K NEXT 10K);
```

このマテリアライズド・ビュー・ログは、主キー・マテリアライズド・ビューの高速リフレッシュのみをサポートします。次の文は、ROWID 句を指定してマテリアライズド・ビュー・ログの別のバージョンを作成します。これによって、高速リフレッシュが使用可能になるマテリアライズド・ビューのタイプが追加されます。

```
CREATE MATERIALIZED VIEW LOG ON customers WITH PRIMARY KEY, ROWID;
```

このマテリアライズド・ビュー・ログによって、ROWID マテリアライズド・ビューおよびマテリアライズド結合ビューに対する高速リフレッシュが実行可能になります。マテリアライズド集計ビューの高速リフレッシュを指定するには、次の文に示すとおり、SEQUENCE および INCLUDING NEW VALUES 句も指定する必要があります。

マテリアライズド・ビュー・ログにフィルタ列を指定する例： 次の文は、sh.sales 表にマテリアライズド・ビュー・ログを作成します。この文は、14-26 ページの「[マテリアライズド集計ビューの作成例:](#)」で使用します。この文は、このマテリアライズド・ビューで参照される表のすべての列を、フィルタ列として指定します。

```
CREATE MATERIALIZED VIEW LOG ON sales
WITH ROWID, SEQUENCE(amount_sold, time_id, prod_id)
INCLUDING NEW VALUES;
```

マテリアライズド・ビュー・ログに結合列を指定する例： 次の文は、サンプル・スキーマ oe の order_items 表にマテリアライズド・ビュー・ログを作成します。ログは、主キーおよび 14-29 ページの「[高速リフレッシュ可能なマテリアライズド・ビューの作成例:](#)」で結合列として使用した product_id を記録します。

```
CREATE MATERIALIZED VIEW LOG ON order_items WITH (product_id);
```

マテリアライズド・ビュー・ログに新しい値を追加する例： 次の例は、INCLUDING NEW VALUES を指定する oe.product_information 表にマテリアライズド・ビュー・ログを作成します。

```
CREATE MATERIALIZED VIEW LOG ON product_information
WITH ROWID, (list_price, min_price, category_id)
INCLUDING NEW VALUES;
```

次の文は、マテリアライズド集計ビューを作成し、product_information ログを使用します。

```
CREATE MATERIALIZED VIEW products_mv
REFRESH FAST ON COMMIT
AS SELECT SUM(list_price - min_price), category_id
FROM product_information
GROUP BY category_id;
```

使用するログに古い値と新しい値の両方が含まれるため、このマテリアライズド・ビューでは、高速リフレッシュを実行できます。

CREATE OPERATOR

用途

CREATE OPERATOR 文を使用すると、新しい演算子を作成し、バインディングを定義できます。

演算子は、索引タイプ、DML および問合せ SQL 文によって参照できます。演算子は、ファンクション、パッケージ、型および他のユーザー定義オブジェクトを順番に参照します。

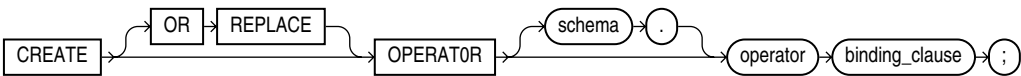
参照： これらの依存性および一般的な演算子については、『Oracle9i Data Cartridge Developer’s Guide』および『Oracle9i データベース概要』を参照してください。

前提条件

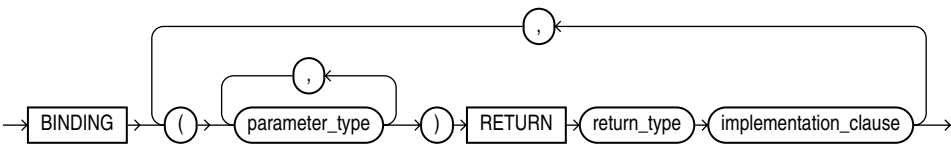
自分のスキーマ内に演算子を作成する場合は、CREATE OPERATOR システム権限が必要です。他のユーザーのスキーマ内に演算子を作成する場合は、CREATE ANY OPERATOR システム権限が必要です。どちらの場合も、参照するファンクションおよび演算子に対する EXECUTE 権限が必要です。

構文

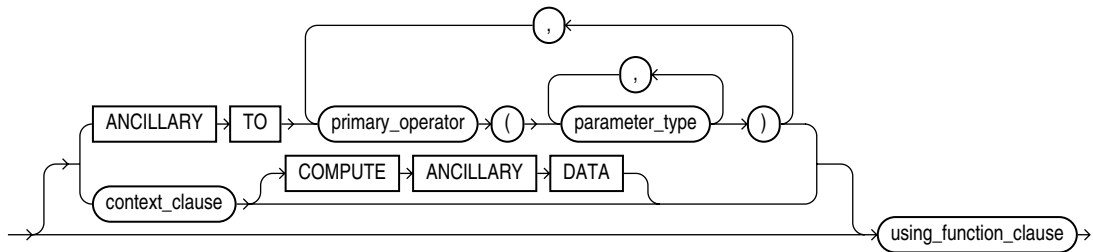
create_operator::=



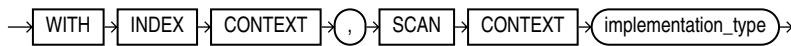
binding_clause::=



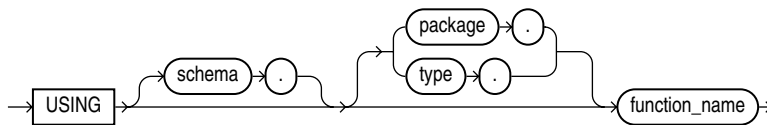
implementation_clause::=



context_clause::=



using_function_clause::=



セマンティクス

OR REPLACE

OR REPLACE を指定して、演算子スキーマ・オブジェクトの定義を置換します。

演算子の置換の制限事項： 演算子に依存オブジェクトがない場合のみ、定義を再作成できます（たとえば、演算子をサポートする索引タイプ）。

schema

演算子が含まれているスキーマを指定します。*schema* を省略した場合、自分のスキーマ内に演算子を作成されます。

operator

作成する演算子の名前を指定します。

binding_clause

binding_clause を使用して、演算子をファンクションにバインドするために、1 つ以上のパラメータ・データ型 (*parameter_type*) を指定します。各バインドの署名 (対応するファンクションの引数のデータ型順序) は、オーバーロードの規則に従って一意である必要があります。

parameter_type 自体をオブジェクト型にできます。この場合、任意にスキーマで修飾することもできます。

演算子のバインドの制限事項： REF、LONG または LONG RAW は *parameter_type* に指定できません。

参照： オーバーロードの詳細は、『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。

RETURN 句

バインドに戻りデータ型を指定します。

return_type 自体をオブジェクト型にできます。この場合、任意にスキーマで修飾することもできます。

戻りデータ型のバインドの制限事項： REF、LONG または LONG RAW は *return_type* に対して指定できません。

implementation_clause**ANCILLARY TO 句**

ANCILLARY TO 句を使用して、演算子バインドが、指定した主演算子バインド (*primary_operator*) を補助することを指定します。この句を指定する場合は、1 つのみの数値パラメータで前のバインドを指定しないでください。

context_clause

context_clause を使用して、演算子の機能的な実装により、スキャン・コンテキストとして使用される実装タイプ名を指定します。

COMPUTE ANCILLARY DATA COMPUTE ANCILLARY DATA によって、演算子バインディングが補助データを計算するように指定します。

using_function_clause

using_function_clause によって、バインディングを実装するファンクションを指定します。*function_name* には、スタンドアロン・ファンクション、パッケージ・ファンクション、型メソッドまたはこれらのシノニムを指定できます。

例

ユーザー定義演算子の作成例： 次の文は、非常に単純な等価性のファンクション実装を作成した後、このファンクションを使用する演算子を作成します。

```
CREATE FUNCTION eq_f(a VARCHAR2, b VARCHAR2) RETURN NUMBER AS
BEGIN
    IF a = b THEN RETURN 1;
    ELSE RETURN 0;
    END IF;
END;
/

CREATE OPERATOR eq_op
    BINDING (VARCHAR2, VARCHAR2)
    RETURN NUMBER
    USING eq_f;
```

CREATE OUTLINE

用途

CREATE OUTLINE 文を使用すると、ストアド・アウトラインを作成できます。ストアド・アウトラインは、実行計画を生成するためにオプティマイザが使用する属性集合です。最適化に影響する要因に変更があるにもかかわらず、特定の SQL 文が発行された場合、実行計画の生成に影響するアウトラインの集合を使用するように、オプティマイザに指示します。これらの要因における変更を考慮するように、アウトラインを変更することもできます。

注意： 後で発行する SQL 文には、アウトライン作成時に指定した文と一致する文字列を指定する必要があります。

参照：

- 実行計画の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- アウトラインの変更については、9-116 ページの「[ALTER OUTLINE](#)」を参照してください。
- USE_STORED_OUTLINES パラメータおよび USE_PRIVATE_OUTLINES パラメータの詳細は、10-2 ページの「[ALTER SESSION](#)」および 10-20 ページの「[ALTER SYSTEM](#)」を参照してください。

前提条件

パブリック・アウトラインまたはプライベート・アウトラインの作成には、CREATE ANY OUTLINE システム権限が必要です。

ソース・アウトラインからクローン・アウトラインを作成するには、SELECT_CATALOG_ROLE ロールが必要です。

プライベート・アウトラインを作成するには、DBMS_OUTLN_EDIT.CREATE_EDIT TABLES プロシージャの実行によって自分のスキーマのアウトライン・データを保持するアウトライン編集表が必要です。このプロシージャの実行には、DBMS_OUTLN_EDIT パッケージに対する EXECUTE 権限が必要です。

個々のセッションまたはシステムに対して、ストアド・アウトラインを使用可能または使用禁止にします。

- USE_STORED_OUTLINES パラメータを使用可能にすると、パブリック・アウトラインが使用可能になります。

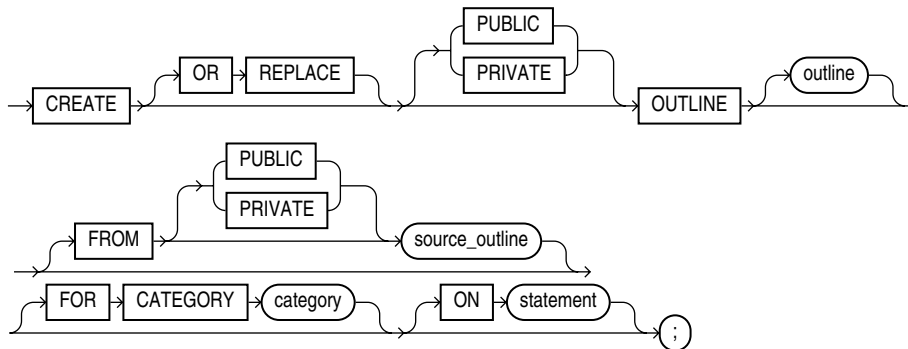
- `USE_PRIVATE_OUTLINES` パラメータを使用可能にすると、プライベート・ストアド・アウトラインが使用可能になります。

参照：

- アウトラインの使用によるパフォーマンス・チューニングの詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- `DBMS_OUTLN_EDIT` パッケージの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

構文

`create_outline::=`



セマンティクス

OR REPLACE

`OR REPLACE` を指定して、既存のアウトラインを同じ名前の新しいアウトラインと置き換えます。

PUBLIC | PRIVATE

`PUBLIC` が使用するアウトラインを作成する場合は、`PUBLIC` を指定します。これはデフォルト値です。

現行のセッションのみがプライベートに使用するアウトラインを作成する場合は、`PRIVATE` を指定します。このアウトラインのデータは、現行のスキーマに格納されます。

注意： 最初にプライベート・アウトラインを作成する前に、`OUTLN_PKG.CREATE_EDIT_TABLES` プロシージャを実行し、自分のスキーマ内に必要なアウトライン表および索引を作成する必要があります。

outline

ストアド・アウトラインに割り当てる一意の名前を指定します。`outline` を指定しない場合、システムがアウトライン名を生成します。

参照： 14-47 ページの「[アウトラインの作成例](#)」を参照してください。

FROM ... source_outline 句

FROM 句を使用すると、既存のアウトラインのコピーによって新しいアウトラインが作成されます。デフォルトでは、パブリック領域の `source_category` が検索されます。`PRIVATE` を指定すると、現行のスキーマのアウトラインが検索されます。

アウトラインのコピーの制限事項： FROM 句を指定する場合、ON 句は指定できません。

参照： 14-47 ページの「[プライベート・クローン・アウトラインの作成例 :](#)」および 14-47 ページの「[プライベート・アウトラインのパブリック領域への公開例 :](#)」を参照してください。

FOR CATEGORY 句

グループ・ストアド・アウトラインに使用される任意の名前を指定します。たとえば、週末に使用するアウトラインの 1 つのカテゴリおよび四半期末に使用する別のカテゴリを指定できます。`category` を指定しない場合、アウトラインは `DEFAULT` カテゴリに格納されます。

ON 句

文をコンパイルする際にアウトラインが作成される SQL 文を指定します。この句は、FROM 句を使用して既存のアウトラインのコピーを作成する場合のみに指定するオプションです。

次の文のいずれかを指定できます。

- SELECT
- DELETE
- UPDATE
- INSERT ... SELECT
- CREATE TABLE ... AS SELECT

アウトラインの SQL 文の制限事項：

- ON 句を指定する場合、FROM 句は指定できません。
- マルチ表の INSERT 文でアウトラインを作成できません。

注意： 単一文に対して複数アウトラインを指定できますが、同じ文に対するアウトラインは別のカテゴリにある必要があります。

例

アウトラインの作成例 次の文は、ON 文をコンパイルすることによってストアド・アウトラインを作成します。アウトラインは salaries という名前で、special カテゴリに格納されます。

```
CREATE OUTLINE salaries FOR CATEGORY special
  ON SELECT last_name, salary FROM employees;
```

USE_STORED_OUTLINES パラメータに special が設定されている場合、同じ SELECT 文が後でコンパイルされると、アウトライン salaries を作成する場合と同様に実行計画が生成されます。

プライベート・クローン・アウトラインの作成例： 次の文は、前述の例で作成したパブリック・アウトライン salaries に基づいて、ストアド・プライベート・アウトライン my_salaries を作成します。プライベート・アウトラインを作成する場合、プライベート・アウトラインを作成するユーザーには、DBMS_OUTLN_EDIT パッケージに対する EXECUTE 権限が必要です。また、このパッケージの CREATE_EDIT_TABLES プロシージャを実行する必要があります。

```
EXECUTE DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES;

CREATE OR REPLACE PRIVATE OUTLINE my_salaries
  FROM salaries;
```

プライベート・アウトラインのパブリック領域への公開例： 次の文は、プライベート編集の後でプライベート・アウトラインをパブリック領域にコピー（または公開）します。

```
CREATE OR REPLACE OUTLINE public_salaries
  FROM PRIVATE my_salaries;
```

CREATE PACKAGE

用途

CREATE PACKAGE 文を使用すると、**ストアド・パッケージ**を指定できます。ストアド・パッケージとは、関連するプロシージャ、ファンクション、およびデータベース上にまとめて格納されるその他のプログラム・オブジェクトの集合のことです。**仕様部**では、これらのオブジェクトを宣言します。

参照：

- パッケージ実装の指定については、14-53 ページの「[CREATE PACKAGE BODY](#)」を参照してください。
- スタンドアロン・ファンクションおよびプロシージャの作成については、13-49 ページの「[CREATE FUNCTION](#)」および 14-60 ページの「[CREATE PROCEDURE](#)」を参照してください。
- パッケージの変更については、9-118 ページの「[ALTER PACKAGE](#)」を参照してください。
- パッケージの削除については、16-88 ページの「[DROP PACKAGE](#)」を参照してください。
- パッケージの詳細および使用方法については、『Oracle9i アプリケーション開発者ガイド - 基礎編』および『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

前提条件

パッケージを作成する前に、ユーザー SYS は、DBMSSTD_X.SQL と呼ばれる SQL スクリプトを起動する必要があります。このスクリプトの正確なファイル名および位置は、使用するオペレーティング・システムによって異なります。

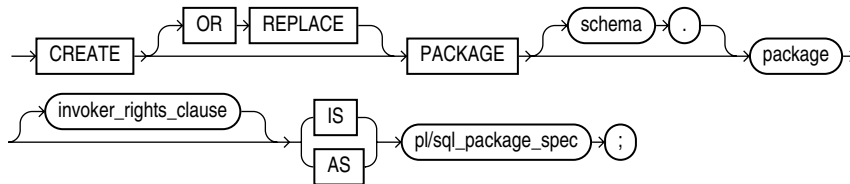
自分のスキーマ内にパッケージを作成する場合は、CREATE PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にパッケージを作成する場合は、CREATE ANY PROCEDURE システム権限が必要です。

Oracle プリコンパイラ・プログラム内に CREATE PACKAGE 文を埋め込む場合、キーワード END-EXEC に続けて、各言語の埋込み SQL 文の終了記号を記述して文を終了する必要があります。

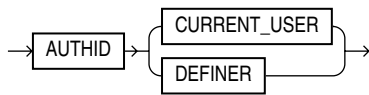
参照：『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

構文

create_package::=



invoker_rights_clause::=



セマンティクス

OR REPLACE

既存のパッケージ仕様部を再作成する場合は、**OR REPLACE** を指定します。この句を指定した場合、パッケージに対して付与されていたオブジェクト権限を削除、再作成および再付与しなくても、既存のパッケージの仕様部を変更できます。パッケージ仕様部を変更した場合、その仕様部は自動的に再コンパイルされます。

再定義したパッケージに対して権限が付与されていたユーザーは、権限が再付与されなくてもそのパッケージにアクセスできます。

ファンクション索引がパッケージに依存している場合、索引に **DISABLED** のマークが付きます。

参照： パッケージ仕様部の再コンパイルについては、9-118 ページの「**ALTER PACKAGE**」を参照してください。

schema

パッケージを含むスキーマを指定します。*schema* を指定しない場合、自分のスキーマ内にパッケージが作成されます。

package

作成するパッケージの名前を指定します。

パッケージの作成がコンパイル・エラーになった場合、Oracle はエラーを戻します。
SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

invoker_rights_clause

invoker_rights_clause によって、パッケージ内のファンクションおよびプロシージャが、権限を持つユーザーのスキーマ内で実行されるか、または、権限を持つ CURRENT_USER のスキーマ内で実行されるかを指定できます。この仕様部は対応するパッケージ本体にも適用されます。

この句は、問合せ、DML 操作およびパッケージにおける動的 SQL 文の外部名の変換方法も定義します。

AUTHID CURRENT_USER

パッケージを CURRENT_USER 権限で実行する場合は、CURRENT_USER を指定します。この句によって**実行者権限パッケージ**が作成されます。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を CURRENT_USER のスキーマで変換することも指定します。他のすべての文における外部名は、パッケージを含むスキーマで変換します。

AUTHID DEFINER

パッケージを含むスキーマの所有者権限でパッケージを実行する場合、およびパッケージを含むスキーマ内で外部名を変換する場合は、DEFINER を指定します。これはデフォルトで、**定義者権限パッケージ**を作成します。

参照：

- 『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。
- CURRENT_USER の判断方法については『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

pl/sql_package_spec

型定義、カーソル宣言、変数宣言、定数宣言、例外宣言、PL/SQL サブプログラム仕様およびコール仕様（PL/SQL 内で記述された C または Java ルーチンの宣言）を含むことができるパッケージ仕様部を指定します。

参照：

- PL/SQL パッケージ・プログラム・ユニットの詳細は、『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 が提供するパッケージについては、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- パッケージにおけるユーザー定義ファンクションの制限事項については、13-52 ページの「[ユーザー定義ファンクションの制限事項:](#)」を参照してください。

例

パッケージの作成例： 次の SQL 文は、emp_mgmt パッケージの仕様部を作成します。

```
CREATE OR REPLACE PACKAGE emp_mgmt AS
  FUNCTION hire (last_name VARCHAR2, job_id VARCHAR2,
    manager_id NUMBER, salary NUMBER,
    commission_pct NUMBER, department_id NUMBER)
    RETURN NUMBER;
  FUNCTION create_dept(department_id NUMBER, location_id NUMBER)
    RETURN NUMBER;
  PROCEDURE remove_emp(employee_id NUMBER);
  PROCEDURE remove_dept(department_id NUMBER);
  PROCEDURE increase_sal(employee_id NUMBER, salary_incr NUMBER);
  PROCEDURE increase_comm(employee_id NUMBER, comm_incr NUMBER);
  no_comm EXCEPTION;
  no_sal EXCEPTION;
END emp_mgmt;
/
```

emp_mgmt パッケージの仕様部では、次のパブリック・プログラム・オブジェクトが宣言されます。

- hire および create_dept の各ファンクション
- remove_emp、remove_dept、increase_sal および increase_comm の各プロシージャ
- no_comm および no_sal の各例外

これらのすべてのオブジェクトは、このパッケージに対するアクセス権限があるユーザーが使用できます。パッケージの作成後は、パッケージのパブリック・プロシージャまたはファンクションをコールしたり、パッケージのパブリック例外を発生させるアプリケーションを開発できます。

このパッケージのプロシージャおよびファンクションをコールする前に、パッケージ本体でこれらのプロシージャおよびファンクションを定義しておく必要があります。emp_mgmt パッケージの本体を作成する CREATE PACKAGE BODY 文の例は、14-53 ページの「[CREATE PACKAGE BODY](#)」を参照してください。

CREATE PACKAGE BODY

用途

CREATE PACKAGE BODY 文を使用すると、**ストアド・パッケージ**を作成できます。ストアド・パッケージとは、関連するプロシージャ、ストアド・ファンクション、およびデータベース上にまとめて格納されるその他のプログラム・オブジェクトの集合のことです。**本体**では、これらのオブジェクトを定義します。

一連のプロシージャやファンクションをスタンドアロンのスキーマ・オブジェクトとして作成するかわりの方法としてパッケージを使用する方法があります。

参照：

- スタンドアロン・ファンクションおよびプロシージャの作成については、13-49 ページの「[CREATE FUNCTION](#)」および 14-60 ページの「[CREATE PROCEDURE](#)」を参照してください。
- 作成方法を含むパッケージの詳細は、14-48 ページの「[CREATE PACKAGE](#)」を参照してください。
- 使用例については、14-55 ページの「[例](#)」を参照してください。
- パッケージの変更については、9-118 ページの「[ALTER PACKAGE](#)」を参照してください。
- データベースからのパッケージの削除については、16-88 ページの「[DROP PACKAGE](#)」を参照してください。

前提条件

パッケージを作成する前に、ユーザー SYS は、DBMSSTD_X.SQL と呼ばれる SQL スクリプトを実行する必要があります。このスクリプトの正確なファイル名および位置は、使用するオペレーティング・システムによって異なります。

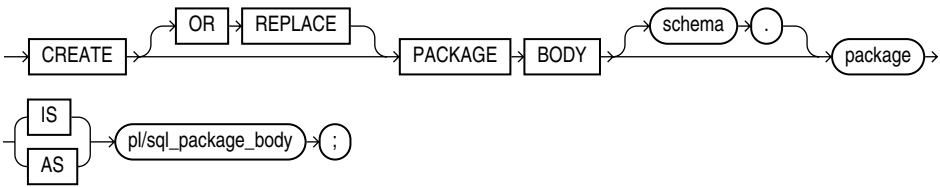
自分のスキーマ内にパッケージを作成する場合は、CREATE PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にパッケージを作成する場合は、CREATE ANY PROCEDURE システム権限が必要です。

Oracle プリコンパイラ・プログラム内に CREATE PACKAGE BODY 文を埋め込む場合、キーワード END-EXEC とその後にそれぞれの言語用の埋込み SQL 文終了記号を記述して文を終了する必要があります。

参照：『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。

構文

create_package_body::=



セマンティクス

OR REPLACE

既存のパッケージ本体を再作成する場合は、OR REPLACE を指定します。この句を指定した場合、パッケージに対して付与されていたオブジェクト権限を削除、再作成および再付与しなくても、既存のパッケージの本体を変更できます。パッケージ本体を変更した場合、その本体は自動的に再コンパイルされます。

再定義したパッケージに対して権限が付与されていたユーザーは、権限が再付与されなくてもそのパッケージにアクセスできます。

参照： パッケージ本体の再コンパイルについては、9-118 ページの「ALTER PACKAGE」を参照してください。

schema

パッケージを含むスキーマを指定します。schema を指定しない場合、現行のスキーマ内にパッケージが作成されます。

package

作成するパッケージの名前を指定します。

pl/sql_package_body

PL/SQL サブプログラム本体またはコール仕様（PL/SQL に記述された C または Java ルーチンの宣言）を含むことができるパッケージ本体を指定します。

参照：

- PL/SQL または C パッケージ・プログラム・ユニットへの書込みの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- Java パッケージ・プログラム・ユニットについては、『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- パッケージにおけるユーザー定義ファンクションの制限事項については、13-52 ページの「[ユーザー定義ファンクションの制限事項](#)」を参照してください。

例

パッケージ本体の作成例： 次の SQL 文は、emp_mgmt パッケージ（14-51 ページの「[パッケージの作成例](#)」で作成）の本体を作成します。

```
CREATE OR REPLACE PACKAGE BODY emp_mgmt AS
    tot_ems NUMBER;
    tot_depts NUMBER;
    FUNCTION hire
        (last_name VARCHAR2, job_id VARCHAR2,
         manager_id NUMBER, salary NUMBER,
         commission_pct NUMBER, department_id NUMBER)
        RETURN NUMBER IS new_empno NUMBER;
    BEGIN
        SELECT employees_seq.NEXTVAL
            INTO new_empno
            FROM DUAL;
        INSERT INTO employees
            VALUES (new_empno, 'First', 'Last', 'first.last@oracle.com',
                    '(123)123-1234', '18-JUN-02', 'IT_PROG', 90000000, 00,
                    100, 110);
        tot_ems := tot_ems + 1;
        RETURN(new_empno);
    END;
    FUNCTION create_dept(department_id NUMBER, location_id NUMBER)
        RETURN NUMBER IS
        new_deptno NUMBER;
    BEGIN
        SELECT departments_seq.NEXTVAL
            INTO new_deptno
            FROM dual;
        INSERT INTO departments
            VALUES (new_deptno, 'department name', 100, 1700);
        tot_depts := tot_depts + 1;
```

CREATE PACKAGE BODY

```
        RETURN(new_deptno);
    END;
    PROCEDURE remove_emp (employee_id NUMBER) IS
    BEGIN
        DELETE FROM employees
        WHERE employees.employee_id = remove_emp.employee_id;
        tot_emps := tot_emps - 1;
    END;
    PROCEDURE remove_dept(department_id NUMBER) IS
    BEGIN
        DELETE FROM departments
        WHERE departments.department_id = remove_dept.department_id;
        tot_depts := tot_depts - 1;
        SELECT COUNT(*) INTO tot_emps FROM employees;
    END;
    PROCEDURE increase_sal(employee_id NUMBER, salary_incr NUMBER) IS
    curr_sal NUMBER;
    BEGIN
        SELECT salary INTO curr_sal FROM employees
        WHERE employees.employee_id = increase_sal.employee_id;
        IF curr_sal IS NULL
            THEN RAISE no_sal;
        ELSE
            UPDATE employees
            SET salary = salary + salary_incr
            WHERE employee_id = employee_id;
        END IF;
    END;
    PROCEDURE increase_comm(employee_id NUMBER, comm_incr NUMBER) IS
    curr_comm NUMBER;
    BEGIN
        SELECT commission_pct
        INTO curr_comm
        FROM employees
        WHERE employees.employee_id = increase_comm.employee_id;
        IF curr_comm IS NULL
            THEN RAISE no_comm;
        ELSE
            UPDATE employees
            SET commission_pct = commission_pct + comm_incr;
        END IF;
    END;
END emp_mgmt;
/
```

パッケージ本体では、パッケージ仕様部で宣言した次のパブリック・プログラム・オブジェクトを定義しています。

- hire および create_dept の各ファンクション
- remove_emp、remove_dept、increase_sal および increase_comm の各プロシージャ

これらのオブジェクトは、パッケージ仕様部で宣言されているため、パッケージ外のアプリケーション・プログラム、プロシージャおよびファンクションからコールできます。たとえば、パッケージに対するアクセス権限がある場合は、increase_comm プロシージャをコールする emp_mgmt パッケージとは別に、increase_all_comms プロシージャを作成できます。

これらのオブジェクトはパッケージ本体で定義されているため、その定義を変更した場合でも、Oracle が依存スキーマ・オブジェクトを無効にすることはありません。たとえば、後で hire の定義を変更した場合に、increase_all_comms は実行するまで再コンパイルされません。

また、この例のパッケージ本体では、プライベート・プログラム・オブジェクトである変数 tot_emps および tot_depts が宣言されています。これらのオブジェクトは、パッケージ仕様部ではなくパッケージ本体で宣言されているため、パッケージ内の他のオブジェクトからアクセスできますが、パッケージ外からはアクセスできません。たとえば、変数 tot_depts の値を明示的に変更するアプリケーションは開発できません。ただし、ファンクション create_dept はパッケージの一部であるため、create_dept で tot_depts の値を変更できます。

CREATE PFILE

用途

CREATE PFILE 文を使用すると、バイナリのサーバー・パラメータ・ファイルをテキストの初期化パラメータ・ファイルにエクスポートできます。テキストのパラメータ・ファイルの作成は、データベースで使用している現行のパラメータ設定リストの取得に便利です。また、テキスト・エディタで簡単に編集でき、CREATE SPFILE 文を使用してサーバー・パラメータ・ファイルに変換して戻すこともできます。

この文が正常に実行されると、サーバーにテキストのパラメータ・ファイルが作成されます。Real Application Clusters 環境では、すべてのインスタンスのすべてのパラメータ設定が含まれます。サーバー・パラメータ・ファイルのパラメータ設定と同じ行のコメントも含まれます。

参照：

- サーバー・パラメータ・ファイルの詳細は、14-87 ページの「[CREATE SPFILE](#)」を参照してください。
- Oracle9i より前のテキストの初期化パラメータ・ファイル、および Oracle9i のバイナリのサーバー・パラメータ・ファイルの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- Real Application Clusters 環境でのサーバー・パラメータ・ファイルの使用については、『Oracle9i Real Application Clusters 管理』を参照してください。

前提条件

この文を実行するには、SYSDBA ロールまたは SYSOPER ロールが必要です。この文は、インスタンスの起動前と起動後のいずれかで実行できます。

構文

create_pfile::=



セマンティクス

pfile_name

作成するテキストのパラメータ・ファイル名を指定します。*pfile_name* を指定しないと、プラットフォーム固有のデフォルトの初期化パラメータ・ファイル名が使用されます。

spfile_name

テキストのファイルを作成する元となるバイナリのサーバー・パラメータ・ファイル名を指定します。

- *spfile_name* を指定する場合、そのファイルがサーバーに存在する必要があります。ファイルがオペレーティング・システムのサーバー・パラメータ・ファイルのデフォルト・ディレクトリに存在しない場合、フルパス名を指定する必要があります。
- *spfile_name* を指定しない場合、オペレーティング・システムのサーバー・パラメータ・ファイルのデフォルト・ディレクトリからプラットフォーム固有のデフォルトのサーバー・パラメータ・ファイル名が検索され、使用されます。そのディレクトリにファイルが存在しない場合、エラーが戻されます。

参照： デフォルトのパラメータ・ファイル名は、『Oracle9i データベース管理者ガイド』（またはオペレーティング・システム固有のドキュメント）を参照してください。

例

パラメータ・ファイルの作成例： 次の例は、バイナリのサーバー・パラメータ・ファイル production.ora からテキストのパラメータ・ファイル my_init.ora を作成します。

```
CREATE PFILE = 'my_init.ora' FROM SPFILE = 's_params.ora';
```

注意： 通常、オペレーティング・システムのパラメータ・ファイルには、フルパスのファイル名を指定する必要があります。パスについては、ご使用のオペレーティング・システムの Oracle マニュアルを参照してください。

CREATE PROCEDURE

用途

CREATE PROCEDURE 文を使用すると、スタンドアロンのストアド・プロシージャまたはコール仕様を作成できます。

プロシージャとは、名前によってコールできる PL/SQL 文の集合です。コール仕様 (call spec) は、SQL および PL/SQL からコールできるように、Java メソッドまたは 3GL を宣言します。コール仕様は、コールされたときに起動する Java メソッドを問い合わせます。引数および戻り値に対する型変換も問い合わせます。

ストアド・プロシージャには、開発、整合性、セキュリティ、パフォーマンスおよびメモリー割当ての面でいくつかのメリットがあります。

参照：

- コール方法などのストアド・プロシージャの詳細、および外部プロシージャの登録については、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- プロシージャと類似点が多いファンクションの詳細は、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- パッケージの作成の詳細は、14-48 ページの「[CREATE PACKAGE](#)」を参照してください。(CREATE PROCEDURE 文では、スタンドアロンのスキーマ・オブジェクトとしてプロシージャが作成されます。また、プロシージャをパッケージの一部としても作成できます。)
- スタンドアロン・プロシージャの変更および削除については、9-122 ページの「[ALTER PROCEDURE](#)」および 16-90 ページの「[DROP PROCEDURE](#)」を参照してください。
- 共有ライブラリについては、14-2 ページの「[CREATE LIBRARY](#)」を参照してください。

前提条件

プロシージャを作成する前に、ユーザー SYS は、DBMSSTD_X.SQL と呼ばれる SQL スクリプトを実行する必要があります。このスクリプトの正確なファイル名および位置は、使用するオペレーティング・システムによって異なります。

自分のスキーマ内にプロシージャを作成する場合は、CREATE PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にプロシージャを作成する場合は、CREATE ANY PROCEDURE システム権限が必要です。他のユーザーのスキーマ内にプロシージャを再配置する場合は、ALTER ANY PROCEDURE システム権限が必要です。

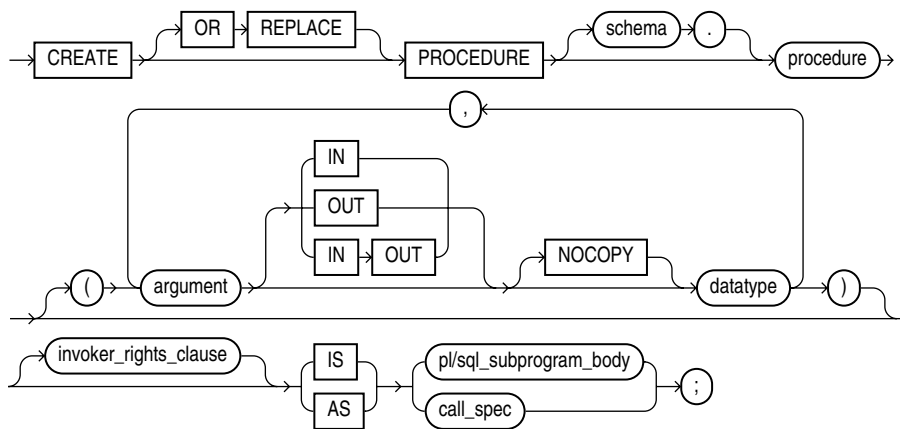
コール仕様を起動する場合、その他の権限が必要になることがあります（たとえば、C コール仕様の C ライブラリには EXECUTE 権限が必要です）。

Oracle プリコンパイラ・プログラム内に CREATE PROCEDURE 文を埋め込む場合、キーワード END-EXEC とその後にそれぞれの言語用の埋込み SQL 文終了記号を記述して文を終了する必要があります。

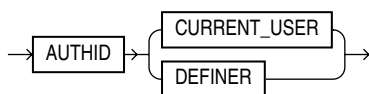
参照： 詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』または『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。

構文

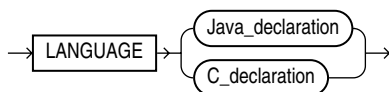
create_procedure::=



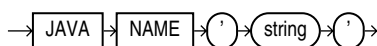
invoker_rights_clause::=

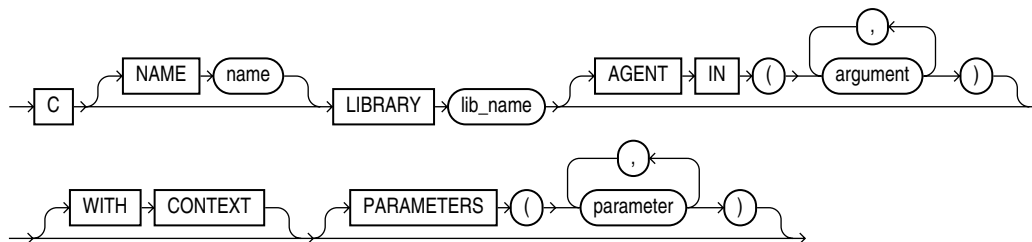


call_spec::=



Java_declaration::=



C_declaration::=

セマンティクス

OR REPLACE

既存のプロシージャを再作成する場合は、OR REPLACE を指定します。この句を指定した場合、プロシージャに付与されているオブジェクト権限を削除、再作成および再付与しなくても、既存のプロシージャの定義を変更できます。プロシージャを再定義した場合、そのプロシージャは自動的に再コンパイルされます。

再定義したプロシージャに対して権限が付与されていたユーザーは、権限が再付与されなくてもそのプロシージャにアクセスできます。

ファンクション索引がパッケージに依存している場合、索引に DISABLED のマークが付きます。

参照： プロシージャの再コンパイルについては、9-122 ページの「[ALTER PROCEDURE](#)」を参照してください。

schema

プロシージャを定義するスキーマを指定します。schema を省略した場合、現行のスキーマ内にプロシージャが作成されます。

procedure

作成するプロシージャの名前を指定します。

プロシージャの作成でコンパイル・エラーがある場合、Oracle はエラーを戻します。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

argument

プロシージャへの引数の名前を指定します。プロシージャが引数を受け入れない場合は、プロシージャ名の後のカッコを省略できます。

IN IN は、プロシージャをコールするときに、引数に値を指定する必要があることを示します。

OUT OUT は、プロシージャが、実行後にコールの環境に対して、指定した引数の値を戻すことを示します。

IN OUT IN OUT は、プロシージャのコール時に、引数に値を指定し、実行後にコール先の環境に値を戻すことを示します。

IN、OUT および IN OUT のいずれの引数も指定しない場合、デフォルトでは IN が設定されます。

NOCOPY NOCOPY は、この引数をできるだけ速く戻すことを指定します。この句によって、OUT パラメータや IN OUT パラメータに対して、レコード、索引付き表、VARRAY などの大きい値を戻す際のパフォーマンスが大幅に向上します。(IN パラメータ値には、常に NOCOPY が渡されます)。

- NOCOPY パラメータを指定すると、このパラメータに対応する実際の割当てとしてパッケージ変数が渡された場合に、パッケージ変数に対して行われた割当ては、すぐにこのパラメータに表示されます (または、このパラメータに対して行われた割当ては、すぐにパッケージ変数に表示されます)。
- このパラメータまたは別のパラメータに対して行われた変更は、同じ変数が両方に渡された場合、両方の名前を介してすぐに参照できます。
- プロシージャが未処理例外で終了した場合、このパラメータに対する割当ては、コール元の変数で参照できます。

このような効果がないコールもあります。この効果に問題がない場合にのみ NOCOPY を使用してください。

datatype 引数のデータ型を指定します。引数には、PL/SQL でサポートされるデータ型を指定できます。

データ型には、データ長、精度または位取りは指定できません。たとえば、VARCHAR2 (10) は無効ですが、VARCHAR2 は有効です。Oracle では、そのプロシージャがコールされた環境から引数のデータ長、精度および位取りを導出します。

invoker_rights_clause

invoker_rights_clause によって、プロシージャを、スキーマを所有するユーザーの権限でそのスキーマ内で実行するか、または CURRENT_USER の権限でそのスキーマ内で実行するかを指定できます。

この句は、問合せ、DML 操作およびプロシージャにおける動的 SQL 文の外部名の変換方法も指定します。

AUTHID CURRENT_USER

プロシージャを `CURRENT_USER` 権限で実行する場合は、`CURRENT_USER` を指定します。この句によって**実行者権限プロシージャ**が作成されます。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を `CURRENT_USER` のスキーマで変換することも指定します。他のすべての文における外部名は、プロシージャを含むスキーマで変換します。

AUTHID DEFINER

プロシージャを含むスキーマの所有者権限でプロシージャを実行する場合、およびプロシージャを含むスキーマ内で外部名を変換する場合は、`DEFINER` を指定します。これはデフォルトで、**定義者権限プロシージャ**を作成します。

参照：

- 『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。
- `CURRENT_USER` の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

IS | AS 句***pl/sql_subprogram_body***

PL/SQL サブプログラム本体のプロシージャを宣言します。

参照： PL/SQL サブプログラムの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

call_spec

call_spec を使用して、Java メソッドまたは C 関数名、パラメータ・タイプおよび戻り型を SQL で相当するものにマップします。

Java_declaration では、`'string'` が JAVA 実装メソッドを定義します。

参照：

- パラメータおよび *Java_declaration* のセマンティクスの詳細は、『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- パラメータおよび *C_declaration* のセマンティクスの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

AS EXTERNAL AS EXTERNAL 句は、C 関数を宣言するもう 1 つの方法です。この句は以前のリリースのもので、下位互換用にのみサポートされています。AS LANGUAGE C 構文を使用することをお勧めします。

例

プロシージャの作成例： 次の文は、スキーマ hr にプロシージャ remove_emp を作成します。

```
CREATE PROCEDURE remove_emp (employee_id NUMBER) AS
    tot_emps NUMBER;
BEGIN
    DELETE FROM employees
        WHERE employees.employee_id = remove_emp.employee_id;
    tot_emps := tot_emps - 1;
END;
/
```

remove_emp プロシージャは、指定された従業員を削除します。このプロシージャをコールする場合は、削除する従業員の employee_id を指定する必要があります。この引数のデータ型は NUMBER です。

このプロシージャでは、DELETE 文を使用して、employee 表から employee_id の行を削除します。

参照： このプロシージャをパッケージに取り込む方法は、14-55 ページの「[パッケージ本体の作成例：](#)」を参照してください。

次の例では、外部プロシージャ c_find_root によって、ポインタがパラメータとみなされます。プロシージャ find_root は、BY REFERENCE 句を使用した参照によって、そのパラメータを渡します。

```
CREATE PROCEDURE find_root
    ( x IN REAL )
IS LANGUAGE C
    NAME c_find_root
    LIBRARY c_utils
    PARAMETERS ( x BY REFERENCE );
```

CREATE PROFILE

用途

CREATE PROFILE 文を使用すると、プロファイルを作成できます。プロファイルとは、データベース・リソースの制限の設定です。あるユーザーに対してプロファイル割り当てた場合、そのユーザーは、その割当て制限を超えることはできません。

参照： パスワード管理およびパスワード保護の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

前提条件

プロファイルを作成する場合、CREATE PROFILE システム権限が必要です。
次の方法でユーザーに対するリソース制限を指定します。

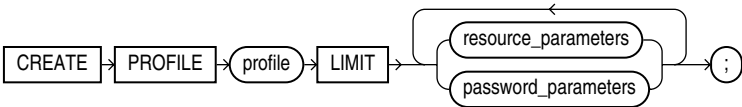
- ALTER SYSTEM 文または初期化パラメータ RESOURCE_LIMIT で動的にリソース制限を使用可能にします（このパラメータは、パスワード・リソースには適用されません。パスワード・リソースは、常に使用可能です）。
- CREATE PROFILE 文を使用して、制限を定義するプロファイルを作成します。
- CREATE USER または ALTER USER 文を使用して、プロファイルを割り当てます。

参照：

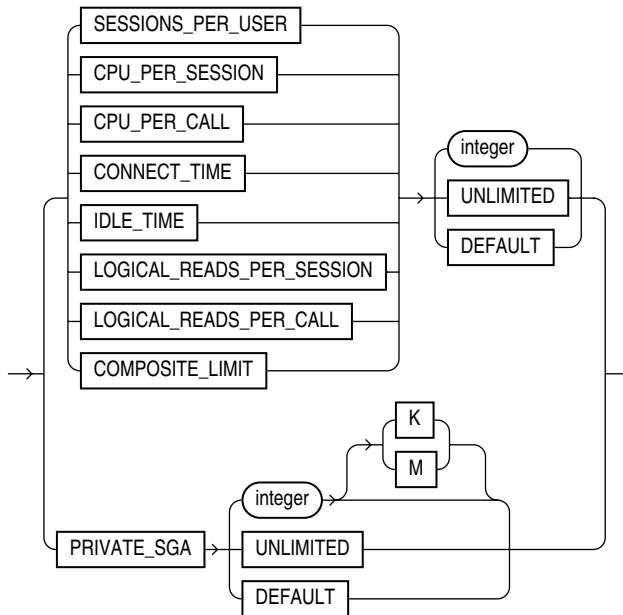
- 動的にリソース制限を使用可能にする場合の詳細は、10-20 ページの「ALTER SYSTEM」を参照してください。
- RESOURCE_LIMIT パラメータについては、『Oracle9i データベース・リファレンス』を参照してください。
- プロファイルについては、16-31 ページの「CREATE USER」および 12-21 ページの「ALTER USER」を参照してください。

構文

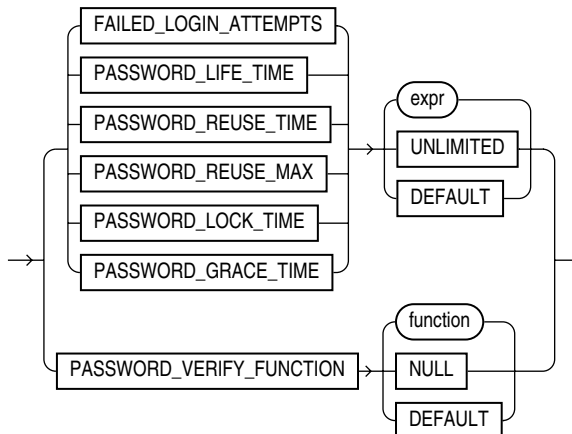
create_profile::=



resource_parameters::=



password_parameters::=



セマンティクス

profile

作成するプロファイルの名前を指定します。プロファイルを使用した場合、ユーザーが使用可能なデータベース・リソースを1つのコールまたは1つのセッションごとに制限できます。

Oracle では、次の方法でリソース制限を適用します。

- `CONNECT_TIME` または `IDLE_TIME` で指定したセッション・リソース制限を超えた場合、現行のトランザクションが自動的にロールバックされ、セッションが終了します。ユーザー・プロセスで次にコールを実行すると、エラーが戻ります。
- 他のセッション・リソースに対する制限を超える処理を実行しようとした場合、その処理が自動的に判断され、現行の文がロールバックされ、エラーが戻されます。この後、ユーザーは、現行のトランザクションをコミットまたはロールバックできます。そして、セッションを終了する必要があります。
- 1つのコールに対する制限を超える処理を実行しようとした場合、その処理が自動的に中断され、現行の文がロールバックされ、エラーが戻されます。この場合、現行のトランザクションは有効です。

注意：

- 日を単位として、時間を制限するすべてのパラメータに対して、分数で日数を指定できます。たとえば、1時間は1/24、1分は1/1440になります。
 - リソース制限を使用可能にしているかどうかにかかわらず、ユーザーに対してリソース制限を指定できます。ただし、Oracle では、実際にその制限が使用可能になってからリソース制限が適用されます。
-

参照： 14-71 ページの「[プロファイルの作成例：](#)」を参照してください。

UNLIMITED

リソース・パラメータで指定した場合、UNLIMITED は、このプロファイルを割り当てられたユーザーが無制限にリソースを使用できることを指定します。パスワード・パラメータでUNLIMITED を指定した場合は、パラメータに制限が設定されていないことを示します。

DEFAULT

このプロファイルにあるリソースの制限を指定しない場合は、DEFAULT を指定します。このプロファイルを割り当てられたユーザーは、DEFAULT プロファイルで指定した対象リソースに対する制限を受けます。DEFAULT プロファイルは、最初に無制限のリソースを定義します。ALTER PROFILE 文でこの制限を変更できます。

明示的にプロファイルが割り当てられていないユーザーは、DEFAULT プロファイルに定義されている制限を受けます。ユーザーに明示的に割り当てられているプロファイルでリソースに対する制限が省略されている場合、または制限に対して DEFAULT が指定されている場合、ユーザーは DEFAULT プロファイルで定義されているリソースに関する制限を受けます。

resource_parameters

SESSIONS_PER_USER ユーザーを制限する同時セッションの数を指定します。

CPU_PER_SESSION 1セッション当たりの CPU 時間制限を指定します。この値は 100 分の 1 秒単位で指定します。

CPU_PER_CALL 1 コール（解析、実行またはフェッチ）当たりの CPU 時間制限を指定します。この値は 100 分の 1 秒単位で指定します。

CONNECT_TIME 1セッション当たりの合計経過時間制限を指定します。この値は分単位で指定します。

IDLE_TIME セッション中の連続的な非活動時間の同期を制限します。この値は分単位で指定します。長時間実行の間合せなどの処理は、この制限を受けません。

LOGICAL_READS_PER_SESSION メモリーおよびディスクから読み込まれるブロックなど、1セッション中に読み込まれるデータ・ブロックの数の制限を指定します。

LOGICAL_READS_PER_CALL SQL 文を処理するコール（解析、実行またはフェッチ）で読み込まれるデータ・ブロックの数の制限を指定します。

PRIVATE_SGA 1つのセッションでシステム・グローバル領域（SGA）の共有プール内に割り当てることができるプライベート領域をバイト単位で指定します。K または M を使用すると、この制限を KB または MB 単位で指定できます。

注意： この制限は、共有サーバー・アーキテクチャを使用している場合のみに適用されます。SGA 内のセッション用のプライベート領域には、プライベート SQL および PL/SQL 領域が含まれますが、共有 SQL および PL/SQL 領域は含まれません。

COMPOSITE_LIMIT 1セッション当たりのリソースの総コストを**サービス単位**で指定します。サービス単位の合計は、CPU_PER_SESSION、CONNECT_TIME、LOGICAL_READS_PER_SESSION および PRIVATE_SGA の重み付き合計として計算されます。

参照： セッション・リソースの重み付けについては、9-129 ページの「[ALTER RESOURCE COST](#)」を参照してください。

これらのパラメータに *expr* を指定すると、式はスカラー副問合せ式を除くすべての形式が可能です。

参照： 14-71 ページの「[プロファイルのリソース制限の設定の例 :](#)」を参照してください。

password_parameters

FAILED_LOGIN_ATTEMPTS ユーザー・アカウントがロックされる前に、そのアカウントへのログインに失敗できる回数を指定します。

PASSWORD_LIFE_TIME 同じパスワードを認証に使用できる日数を制限します。この期間内にパスワードを変更しないと、そのパスワードは使用できなくなり、それ以降の接続は拒否されます。

PASSWORD_REUSE_TIME パスワードが再利用できなくなるまでの日数を指定します。PASSWORD_REUSE_TIME を整数値に設定する場合は、PASSWORD_REUSE_MAX を UNLIMITED に設定する必要があります。

PASSWORD_REUSE_MAX 現行のパスワードを再利用する前に必要な、パスワードの変更回数を指定します。PASSWORD_REUSE_MAX を整数値に設定する場合は、PASSWORD_REUSE_TIME を UNLIMITED に設定する必要があります。

PASSWORD_LOCK_TIME ログインが指定された回数連続して失敗した場合、アカウントがロックされる日数を指定します。

PASSWORD_GRACE_TIME 警告が出され、ログインが許可される猶予期間の日数を指定します。猶予期間中にパスワードが変更されない場合、そのパスワードは使用できなくなります。

PASSWORD_VERIFY_FUNCTION PASSWORD_VERIFY_FUNCTION 句によって、PL/SQL の複雑なパスワード検証スクリプトを CREATE PROFILE 文の引数として渡すことを可能にします。Oracle にはデフォルトのスクリプトがありますが、ユーザー固有のルーチンを作成することも、サード・パーティのソフトウェアを使用することもできます。

- *function* には、複雑なパスワード検証ルーチンの名前を指定します。
- NULL を指定して、パスワードの検証が行われないことを示します。

パスワード・パラメータの制限事項：

- PASSWORD_REUSE_TIME を整数値に設定する場合、PASSWORD_REUSE_MAX を UNLIMITED に設定する必要があります。PASSWORD_REUSE_MAX を整数値に設定する場合、PASSWORD_REUSE_TIME を UNLIMITED に設定する必要があります。
- PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX の両方を UNLIMITED に設定した場合、どちらのパスワードのリソースも使用されません。
- PASSWORD_REUSE_MAX を DEFAULT に設定し、PASSWORD_REUSE_TIME を UNLIMITED に設定した場合、DEFAULT プロファイルに定義された PASSWORD_REUSE_MAX 値が使用されます。
- PASSWORD_REUSE_TIME を DEFAULT に設定し、PASSWORD_REUSE_MAX を UNLIMITED に設定した場合、DEFAULT プロファイルに定義された PASSWORD_REUSE_TIME 値が使用されます。
- PASSWORD_REUSE_TIME および PASSWORD_REUSE_MAX の両方を DEFAULT に設定した場合、DEFAULT プロファイルに定義された値はどちらでも使用されます。

参照： 14-72 ページの「[プロファイルのパスワード制限の設定例:](#)」を参照してください。

例

プロファイルの作成例： 次の文は、プロファイル new_profile を作成します。

```
CREATE PROFILE new_profile
  LIMIT PASSWORD_REUSE_MAX DEFAULT
        PASSWORD_REUSE_TIME UNLIMITED;
```

プロファイルのリソース制限の設定の例： 次の文は、プロファイル app_user を作成します。

```
CREATE PROFILE app_user LIMIT
  SESSIONS_PER_USER          UNLIMITED
  CPU_PER_SESSION            UNLIMITED
  CPU_PER_CALL                3000
  CONNECT_TIME               45
  LOGICAL_READS_PER_SESSION  DEFAULT
  LOGICAL_READS_PER_CALL     1000
  PRIVATE_SGA                15K
  COMPOSITE_LIMIT            5000000;
```

ユーザーに `app_user` プロファイルを割り当てた場合、そのユーザーは、後続のセッションで次の制限を受けます。

- ユーザーは、無制限に同時セッションを使用できます。
- 1つのセッションにおいて、ユーザーは CPU 時間を無制限に消費できます。
- ユーザーが作成した 1つのコールで消費できる CPU 時間は 30 秒以下です。
- 1つのセッションで継続できる時間は 45 分以下です。
- 1つのセッションのメモリーおよびディスクのデータ・ブロック数は、DEFAULT プロファイルで指定した制限を受けます。
- ユーザーが作成した 1つのコールで、メモリーまたはディスクから読み込むことができるデータ・ブロック数の合計は 1000 以下です。
- 1つのセッションで割り当てることができる SGA 内のメモリーは、15KB 以下です。
- 1つのセッションのリソースの総コストは、サービス単位で 500 万を超えることはできません。リソースの総コストの計算式は、ALTER RESOURCE COST 文で指定します。
- `system_manager` プロファイルでは、IDLE_TIME に対する制限が指定されていないため、ユーザーは DEFAULT プロファイルに指定されている対象リソースの制限を受けます。

プロファイルのパスワード制限の設定例： 次の文は、`app_user2` プロファイルに、パスワード制限値を設定して作成します。

```
CREATE PROFILE app_user2 LIMIT
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LIFE_TIME 60
  PASSWORD_REUSE_TIME 60
  PASSWORD_REUSE_MAX UNLIMITED
  PASSWORD_VERIFY_FUNCTION verify_function
  PASSWORD_LOCK_TIME 1/24
  PASSWORD_GRACE_TIME 10;
```

この例では、Oracle のパスワード検証ファンクション `verify_function` を使用します。この検証ファンクションの使用方法または独自の検証ファンクションの設計方法については、『Oracle9i データベース管理者ガイド』を参照してください。

CREATE ROLE

用途

CREATE ROLE 文を使用すると、**ロール**を作成できます。ロールは、ユーザーまたは他のロールに付与できる権限の集合です。ロールを使用してデータベース権限を管理できます。ロールに権限を追加したうえで、ユーザーにそのロールを付与できます。その結果、ユーザーはロールを使用可能にし、そのロールによって付与された権限を使用できるようになります。

ロールには、そのロールに付与されたすべての権限、およびそのロールに付与された他のロールのすべての権限が含まれています。新しく作成されたロールには、ロールや権限は付与されていません。GRANT 文を使用して、ロールに様々な権限を追加します。

NOT IDENTIFIED、IDENTIFIED EXTERNALLY または BY *password* ロールを作成した場合、そのロールは Admin Option 付きで付与されます。ただし、ロール IDENTIFIED GLOBALLY を作成した場合、ロールは付与されません。

参照：

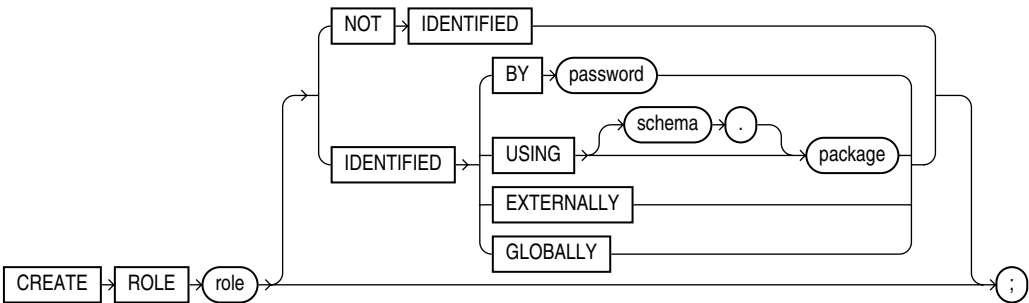
- ロールの付与については、17-27 ページの「[GRANT](#)」を参照してください。
- ロールを使用可能にする場合は、12-21 ページの「[ALTER USER](#)」を参照してください。
- ロールの変更については、9-132 ページの「[ALTER ROLE](#)」を参照してください。
- データベースからのロールの削除については、16-94 ページの「[DROP ROLE](#)」を参照してください。
- 現行のセッションに対するロールの使用可能および使用禁止については、18-44 ページの「[SET ROLE](#)」を参照してください。

前提条件

CREATE ROLE システム権限が必要です。

構文

create_role::=



セマンティクス

role

作成するロールの名前を指定します。データベース・キャラクタ・セットにマルチバイト文字がサポートされている場合でも、ロールにはシングルバイト文字を1つ以上使用することをお薦めします。

配布メディアで提供されている SQL スクリプトには、いくつかのロールが定義されています。

参照： 事前定義済のロールのリストは、17-27 ページの「[GRANT](#)」を参照してください。

NOT IDENTIFIED 句

NOT IDENTIFIED を指定すると、指定するロールがデータベースによって認可され、パスワードを入力しなくてもこのロールを使用可能にできます。

IDENTIFIED 句

IDENTIFIED 句を使用すると、SET ROLE 文によってロールを使用可能にする前に、指定したメソッドによってユーザーが認可される必要があります。

BY password BY password 句によって、ローカル・ロールを作成します。また、ロールを使用可能にするときに、パスワードを指定する必要があることを示します。データベース・キャラクタ・セットにマルチバイト文字が含まれている場合でも、データベース・キャラクタ・セットのシングルバイト文字のみで指定することもできます。

USING package USING package によって、**アプリケーション・ロール**を作成します。アプリケーション・ロールとは、アプリケーションが認可されたパッケージを使用することによってのみ使用可能になるロールです。schema を指定しない場合、パッケージが自分のスキーマ内にあるとみなされます。

EXTERNALLY EXTERNALLY によって、**外部ロール**を作成します。ロールを使用可能にする前に、ユーザーがオペレーティング・システムやサード・パーティ・サービスなどの外部サービスによって認可されている必要があることを示します。

オペレーティング・システムによっては、ユーザーがオペレーティング・システムに対してパスワードを指定しないと、ロールが使用可能にできない場合もあります。

GLOBALLY GLOBALLY によって、**グローバル・ロール**を作成します。また、SET ROLE 文を使用して、またはログイン時にロールを使用可能にする前に、ユーザーがエンタープライズ・ディレクトリ・サービスによってロールの使用を認可されている必要があることを示します。

NOT IDENTIFIED 句および IDENTIFIED 句を両方とも省略した場合、ロールには NOT IDENTIFIED がデフォルト値として使用されます。

例

ロールの作成例： 次の文は、ロール dw_manager を作成します。

```
CREATE ROLE dw_manager;
```

この後に dw_manager を付与されるユーザーは、このロールに付与されているすべての権限を継承します。

次の例のように、パスワードの指定によってロールにセキュリティのレイヤーを追加できます。

```
CREATE ROLE dw_manager
    IDENTIFIED BY warehouse;
```

この後、dw_manager ロールを付与されたユーザーは、パスワード warehouse を指定して、SET ROLE 文でロールを使用可能にする必要があります。

次の文は、グローバル・ロール warehouse_user を作成します。

```
CREATE ROLE warehouse_user IDENTIFIED GLOBALLY;
```

次の文は、同じロールを外部ロールとして作成します。

```
CREATE ROLE warehouse_user IDENTIFIED EXTERNALLY;
```

CREATE ROLLBACK SEGMENT

用途

CREATE ROLLBACK SEGMENT 文を使用すると、ロールバック・セグメントを作成できます。ロールバック・セグメントとは、トランザクションによる変更を元に戻す（取り消す）ために必要なデータを格納する際に Oracle が使用するオブジェクトです。

ここでは、データベースがロールバック UNDO モードで実行されている（初期化パラメータ UNDO_MANAGEMENT を MANUAL に設定、またはすべて設定しない）ことを前提としています。

データベースが自動 UNDO 管理モードで実行されている場合（初期化パラメータ UNDO_MANAGEMENT を AUTO に設定）、ユーザー作成ロールバック・セグメントを変更することはできません。この場合、CREATE ROLLBACK SEGMENT または ALTER ROLLBACK SEGMENT 文の応答でエラーが戻されます。このエラーを回避するには、UNDO_SUPPRESS_ERRORS パラメータを TRUE に設定します。

また、データベースにローカル管理 SYSTEM 表領域がある場合、ディクショナリ管理表領域にロールバック・セグメントは作成できません。その場合は、次のいずれかの処理を行います。

- ロールバック・セグメントのかわりに UNDO 表領域を使用して UNDO データを保持する自動 UNDO 管理機能を使用します。
- ローカル管理表領域を作成して、ロールバック・セグメントを保持します。

自動 UNDO 管理を使用することをお勧めします。

注意：

- 1 つの表領域に複数のロールバック・セグメントを作成できます。一般に、複数のロールバック・セグメントがあると、パフォーマンスが向上します。
 - 表領域にロールバック・セグメントを追加する場合、表領域は必ずオンラインである必要があります。
 - ロールバック・セグメントを作成した場合、最初はオフライン状態になります。そのロールバック・セグメントを Oracle インスタンスによってトランザクション可能にする場合、ALTER ROLLBACK SEGMENT 文を使用してオンラインの状態にしてください。データベース起動時に自動的にオンライン状態にする場合、初期化パラメータ ROLLBACK_SEGMENTS の値にそのセグメントの名前を追加してください。
-
-

SYSTEM 以外の表領域のオブジェクトを使用する場合は、次の注意事項があります。

- ロールバック UNDO モードのデータベースを実行するには、1 つ以上のロールバック・セグメント (SYSTEM ロールバック・セグメント以外の) がオンラインである必要があります。
- 自動 UNDO 管理モードでデータベースを実行するには、1 つ以上の UNDO 表領域がオンラインである必要があります。

参照：

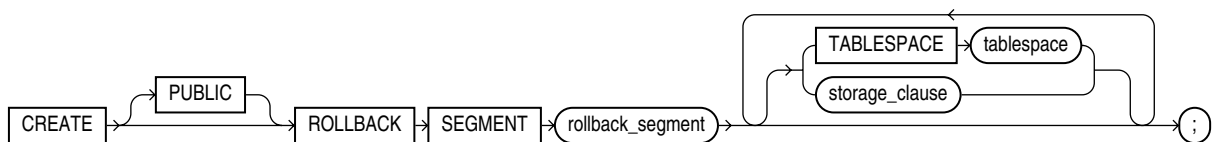
- ロールバック・セグメントの変更については、9-134 ページの「[ALTER ROLLBACK SEGMENT](#)」を参照してください。
- ロールバック・セグメントの削除については、16-95 ページの「[DROP ROLLBACK SEGMENT](#)」を参照してください。
- UNDO_MANAGEMENT パラメータおよび UNDO_SUPPRESS_ERRORS パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 自動 UNDO 管理の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

前提条件

ロールバック・セグメントを作成するには、CREATE ROLLBACK SEGMENT システム権限が必要です。

構文

create_rollback_segment::=



(7-53 ページの「[storage_clause::=](#)」を参照)

キーワードとパラメータ

PUBLIC

PUBLIC を指定すると、ロールバック・セグメントがパブリックとなり、すべてのインスタンスに対して使用可能になります。この句を省略した場合、ロールバック・セグメントはプライベートになり、インスタンスの初期化パラメータ ROLLBACK_SEGMENTS で指定したインスタンスに対してのみ使用可能になります。

rollback_segment

作成するロールバック・セグメントの名前を指定します。

TABLESPACE

TABLESPACE 句を使用して、ロールバック・セグメントが作成される表領域を識別します。この句を省略した場合、ロールバック・セグメントは SYSTEM 表領域に作成されます。

注意： Oracle は、頻繁にロールバック・セグメントにアクセスする必要があります。そのため、明示的または黙示的に（この句を省略して）ロールバック・セグメントを SYSTEM 表領域に作成しないことをお勧めします。さらに、ロールバック・セグメントが含まれている表領域の競合を回避するために、この表領域には表、索引などの他のオブジェクトを含めないでください。また、エクステンツの割当ておよび割当て解除は、最小限に抑える必要があります。

そのためには、自動割当てを使用禁止にしてローカル管理表領域にロールバック・セグメントを作成します。ローカル管理表領域とは、EXTENT MANAGEMENT LOCAL 句に UNIFORM を指定して作成された表領域です。（AUTOALLOCATE 設定はサポートされません。）

参照：

- 15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。
- ロールバック・セグメントを作成および使用可能にする方法の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

storage_clause

storage_clause によって、ロールバック・セグメントの記憶特性を指定できます。

注意：

- *storage_clause* の OPTIMAL パラメータは、ロールバック・セグメントにのみ適用されるので、特に重要です。
 - *storage_clause* の PCTINCREASE パラメータは、CREATE ROLLBACK SEGMENT では指定できません。
-
-

参照： 7-51 ページの「[*storage_clause*](#)」を参照してください。

例

ロールバック・セグメントの作成例： 次の文は、適切に構成された表領域にデフォルトの記憶域値でロールバック・セグメントを作成します。

```
CREATE TABLESPACE rbs_ts
  DATAFILE 'rbs01.dbf' SIZE 10M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 100K;

/* This example and the next will fail if your database is in
   Automatic Undo Mode.
*/
CREATE ROLLBACK SEGMENT rbs_one
  TABLESPACE rbs_ts;
```

前述の文は、次の文と同じ結果になります。

```
CREATE ROLLBACK SEGMENT rbs_one
  TABLESPACE rbs_ts
  STORAGE
  ( INITIAL 10K
    NEXT 10K
    MAXEXTENTS UNLIMITED);
```

CREATE SCHEMA

用途

CREATE SCHEMA を使用すると、複数表およびビューを作成し、1 つのトランザクションで複数の権限の付与を行うことができます。

CREATE SCHEMA 文を実行すると、挿入されている個々の文が実行されます。すべての文が正常に実行された場合、そのトランザクションがコミットされます。文の結果が 1 つでもエラーになった場合は、すべての文がロールバックされます。

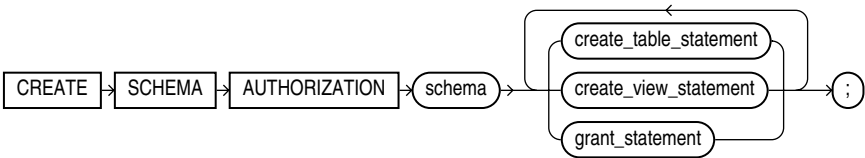
注意： この文では、実際にスキーマが作成されるわけではありません。ユーザーを作成すると、自動的にスキーマが作成されます (16-31 ページの「[CREATE USER](#)」を参照してください)。この文により、表およびビューとともにスキーマが作成され、複数のトランザクションで複数の SQL 文を発行しなくても、これらのオブジェクトに権限が付与されます。

前提条件

CREATE SCHEMA 文には、CREATE TABLE 文、CREATE VIEW 文および GRANT 文を含めることができます。CREATE SCHEMA 文を発行する場合は、挿入した文を発行するための権限が必要です。

構文

create_schema::=



キーワードとパラメータ

schema

スキーマの名前を指定します。スキーマ名は、Oracle ユーザー名と一致している必要があります。

create_table_statement

この CREATE SCHEMA 文の一部として発行する CREATE TABLE 文を指定します。この文の終わりには、セミコロン（またはその他の終了文字）を付けないでください。

参照： 15-7 ページの「[CREATE TABLE](#)」を参照してください。

create_view_statement

この CREATE SCHEMA 文の一部として発行する CREATE VIEW 文を指定します。この文の終わりには、セミコロン（またはその他の終了文字）を付けないでください。

参照： 16-38 ページの「[CREATE VIEW](#)」を参照してください。

grant_statement

この CREATE SCHEMA 文の一部として発行する GRANT *object_privileges* 文を指定します。この文の終わりには、セミコロン（またはその他の終了文字）を付けないでください。

参照： 17-27 ページの「[GRANT](#)」を参照してください。

CREATE SCHEMA 文は、Oracle でサポートされている完全な構文ではなく、標準 SQL で定義されている構文のみをサポートします。

CREATE TABLE、CREATE VIEW および GRANT の各文を指定する順序は重要ではありません。CREATE SCHEMA 文の中の文では、既存のオブジェクトまたは同じ CREATE SCHEMA 文の他の文で作成したオブジェクトを参照できます。

スキーマに対する権限付与の制限事項： *parallel_clause* 構文は、CREATE SCHEMA の CREATE TABLE 文で使用できますが、オブジェクトの作成時に並列度は使用されません。

参照： 15-50 ページの「[CREATE TABLE](#)」の「[parallel_clause](#)」を参照してください。

例

スキーマの作成例： 次の文は、サンプルの注文入力ユーザー oe 用の oe という名前のスキーマ、表 new_product、ビュー new_product_view を作成し、サンプルの人事情報のユーザー hr に new_product_view に対する SELECT 権限を付与します。

```
CREATE SCHEMA AUTHORIZATION oe
  CREATE TABLE new_product
    (color VARCHAR2(10) PRIMARY KEY, quantity NUMBER)
  CREATE VIEW new_product_view
    AS SELECT color, quantity FROM new_product WHERE color = 'RED'
  GRANT select ON new_product_view TO hr;
```

CREATE SEQUENCE

用途

CREATE SEQUENCE 文を使用すると、**順序**を作成できます。順序とは、データベース・オブジェクトの 1 つで、これを使用して複数のユーザーが一意的な整数を生成することができます。順序を使用した場合、主キー値が自動的に生成されます。

順序番号が生成されると、順序はトランザクションのコミットやロールバックとは無関係に増加していきます。2 人のユーザーが、同時に同一の順序を増加させると、ユーザーがそれぞれ順序番号を生成しているため、取得する順序番号間に違いが発生することもあります。他のユーザーが生成した順序番号は取得できません。あるユーザーが一度順序値を生成すると、他のユーザーがその順序を増加させたかどうかに関係なく、順序を生成したユーザーは引き続きその値にアクセスできます。

順序番号は表から独立して生成されるため、1 つ以上の表に対して同一の順序を使用することができます。生成された順序番号が、最終的にロールバックされるトランザクションで使用されたため、個々の順序番号が連続していないように見える場合があります。また、他のユーザーが同一順序を使用していることを個々のユーザーが認識しない場合もあります。

順序が作成されると、SQL 文の中で CURRVAL 疑似列を使用してその値にアクセスできます（この場合、その順序の現在の値が戻ります）。また、NEXTVAL 疑似列を使用してもアクセスできます（この場合は、順序が増加され、新しい値が戻ります）。

参照：

- CURRVAL および NEXTVAL の使用方法については、2-77 ページの「[疑似列](#)」を参照してください。
- 順序の使用については、2-79 ページの「[順序値の使用](#)」を参照してください。
- 順序の変更または削除の詳細は、9-138 ページの「[ALTER SEQUENCE](#)」または 17-2 ページの「[DROP SEQUENCE](#)」を参照してください。

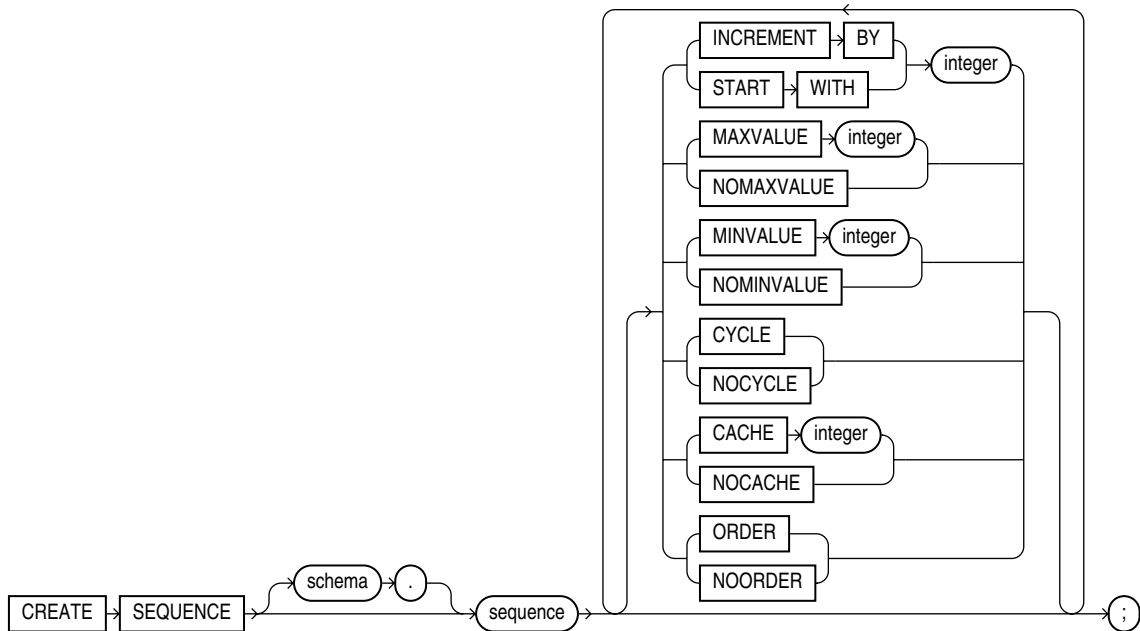
前提条件

自分のスキーマ内に順序を作成する場合は、CREATE SEQUENCE 権限が必要です。

他のユーザーのスキーマ内に順序を作成する場合は、CREATE ANY SEQUENCE 権限が必要です。

構文

`create_sequence::=`



セマンティクス

schema

順序を含むスキーマを指定します。*schema* を省略した場合、自分のスキーマ内に順序が作成されます。

sequence

作成する順序の名前を指定します。

次の句のうちどれも指定しない場合は、1 から始まる昇順の順序が作成され、上限なしで1 ずつ増加していきます。INCREMENT BY に -1 のみを指定した場合は、初期値を -1 として、下限なしで1 ずつ減少していきます。

- **昇順で、無制限に増加する順序を作成する場合は**、MAXVALUE パラメータを省略するか、または NOMAXVALUE を指定します。降順の場合は、MINVALUE パラメータを省略するか、または NOMINVALUE を指定します。

- **昇順で、事前に定義した制限で停止する順序を作成する場合は**、MAXVALUE パラメータに値を指定します。降順の場合は、MINVALUE パラメータの値を指定します。NOCYCLE も指定します。順序が制限に達したときに順序番号をさらに生成した場合、エラーが発生します。
- **事前に定義した制限に達した後で初期値に戻る順序を作成する場合は**、MAXVALUE パラメータと MINVALUE パラメータの両方に値を指定します。CYCLE も指定します。MINVALUE を指定しない場合、デフォルトで NOMINVALUE (値 1) が設定されます。

順序パラメータ

INCREMENT BY 順序の番号間の増分間隔を指定します。この値は、0 (ゼロ) 以外の正の整数または負の整数になります。この値には、28 桁以内の値を指定できます。この値の絶対値は、MAXVALUE と MINVALUE の差未満である必要があります。この値が負の場合、順序は降順になります。この増分値が正の場合、順序は昇順になります。この句を省略した場合、デフォルトで増分間隔は 1 に設定されます。

START WITH 生成する順序番号の初期値を指定します。この句を指定した場合、順序の最小値より大きい値を初期値として昇順を開始することも、最大値よりも小さい値を初期値として降順を開始することもできます。昇順の場合、デフォルト値は順序の最小値になります。降順の場合、デフォルト値は順序の最大値になります。28 桁以内の整数値を指定できます。

注意： この値は、必ずしも、順序の最大値または最小値に達した後に、昇順で循環する順序が戻るときの値ではありません。

MAXVALUE 順序の最大値を指定します。28 桁以内の整数値を指定できます。MAXVALUE 値は、START WITH 以上で、かつ MINVALUE を超える値である必要があります。

NOMAXVALUE NOMAXVALUE は、順序の最大値を、昇順の場合は 10^{27} 、降順の場合は -1 に指定します。これはデフォルト値です。

MINVALUE 順序の最小値を指定します。28 桁以内の整数値を指定できます。MINVALUE 値は、START WITH 以下で、かつ MAXVALUE 未満である必要があります。

NOMINVALUE NOMINVALUE は、順序の最小値を、昇順の場合は 1、降順の場合は -10^{26} に指定します。これはデフォルト値です。

CYCLE CYCLE は、順序が最大値または最小値に達しても、引き続き値を生成することを示します。つまり、昇順の場合は、最大値に達すると最小値が生成されます。降順の場合は、最小値に達すると最大値が生成されます。

NOCYCLE NOCYCLE は、順序が最大値または最小値に達した場合は、それ以上の値を生成できないことを示します。これはデフォルト値です。

CACHE より高速にアクセスできるように、メモリー上に事前に割り当て、保持しておく順序番号値を指定します。28 桁以内の整数値を指定できます。このパラメータの最小値は 2 です。循環する順序の場合、この値は、そのサイクル内で生成される値の数未満である必要があります。指定したサイクル内で生成される順序番号の数を超える値はキャッシュできません。したがって、CACHE に指定できる値の最大値は、次の式で求められる値未満である必要があります。

$$(\text{CEIL} (\text{MAXVALUE} - \text{MINVALUE})) / \text{ABS} (\text{INCREMENT})$$

システム障害が発生すると、キャッシュされた順序の値のうち、コミットされた DML 文で使用されていないものはすべて失われます。したがって、失われる可能性がある値の数は、CACHE パラメータの値と等しくなります。

注意： Real Application Clusters 環境で順序を使用する場合は、パフォーマンスを向上するために、CACHE の設定を使用することをお勧めします。

NOCACHE NOCACHE は、順序の値が事前に割り当てられていないことを示します。

CACHE および NOCACHE の両方を省略した場合、デフォルトで 20 の順序番号がキャッシュされます。

ORDER ORDER は、要求どおりの順で順序番号を生成することを保証する場合に指定します。順序番号をタイムスタンプとして使用する場合に、この句を使用できます。通常、主キー生成用の順序については、順序どおりに生成するかどうかの保証は重要ではありません。

Real Application Clusters 環境で Oracle を使用する場合、ORDER は順序どおりの生成を保証する場合のみに必要です。排他モードの場合、順序番号は必ず順序どおりに生成されます。

NOORDER NOORDER は、要求どおりの順で順序番号を生成することを保証しない場合に指定します。これはデフォルト値です。

例

順序の作成例： 次の文は、サンプル・スキーマ `oe` に順序 `customers_seq` を作成します。この順序は、`customers` 表に行が追加されるときに、顧客 ID 番号として使用できます。

```
CREATE SEQUENCE customers_seq
  START WITH      1000
  INCREMENT BY    1
  NOCACHE
  NOCYCLE;
```

最初に `customers_seq.nextval` を参照したときは、1000 が戻されます。次に参照したときは、1001 が戻されます。同様に、この後の各参照で、前回参照された値より 1 大きい値が戻されます。

CREATE SPFILE

用途

CREATE SPFILE を使用すると、クライアントの初期化パラメータ・ファイルから**サーバー・パラメータ・ファイル**を作成できます。サーバー・パラメータ・ファイルは、サーバーのみに存在し、データベースを起動するためにクライアントから呼ばれるバイナリのファイルです。

サーバー・パラメータ・ファイルを指定すると、個々のパラメータを永続的に変更できます。サーバー・パラメータ・ファイルを使用する場合、新しいパラメータ値を永続化する ALTER SYSTEM SET *parameter* 文を指定できます。新しい値は、現行のインスタンスのみでなく、その後で起動するすべてのインスタンスにおいて適用されます。以前のクライアントのパラメータ・ファイルでは、パラメータ値を永続的に変更できませんでした。これらのファイルは、サーバー上に存在するため、Oracle による自動データベースのチューニングおよび Recovery Manager によるバックアップが可能です。

データベースの起動時にサーバー・パラメータ・ファイルを使用するには、CREATE SPFILE 文を使用して以前のテキストの初期化パラメータ・ファイルからサーバー・パラメータ・ファイルを作成しておく必要があります。

Real Application Clusters 環境のすべてのインスタンスは、同一のサーバー・パラメータ・ファイルを使用する必要があります。ただし、個々のインスタンスで 1 つのファイル内の同じパラメータで異なる設定が可能な場合もあります。インスタンス固有のパラメータ定義は、*SID.parameter = value* で指定します。*SID* にはインスタンス識別子を指定します。

サーバー・パラメータ・ファイルを使用したデータベースの起動方法は、作成したサーバー・パラメータ・ファイルをデフォルトで作成したか、または非デフォルトで作成したかによって異なります。サーバー・パラメータ・ファイルの使用方法については、14-89 ページの「[サーバー・パラメータ・ファイルの作成例](#)」を参照してください。

参照：

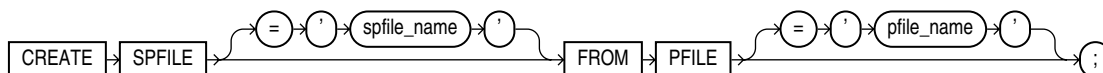
- バイナリのサーバー・パラメータ・ファイルから正規のテキストのパラメータ・ファイルを作成する場合の詳細は、14-58 ページの「[CREATE PFILE](#)」を参照してください。
- Oracle9i より前の初期化パラメータ・ファイル、および Oracle9i のサーバー・パラメータ・ファイルの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- Real Application Clusters 環境でのサーバー・パラメータ・ファイルの使用については、『Oracle9i Real Application Clusters 管理』を参照してください。

前提条件

この文を実行するには、SYSDBA システム権限または SYSOPER システム権限が必要です。この文は、インスタンスの起動前と起動後のいずれかで実行できます。ただし、`spfile_name` を使用して、インスタンスがすでに起動済の場合は、この文で同じ `spfile_name` を指定できません。

構文

create_spfile::=



セマンティクス

spfile_name

この句を指定すると、作成するサーバー・パラメータ・ファイルの名前を指定できます。

- `spfile_name` を指定しない場合、プラットフォーム固有のデフォルトのサーバー・パラメータ・ファイル名が使用されます。`spfile_name` がサーバーにすでに存在する場合、そのファイルは上書きされます。デフォルトのサーバー・パラメータ・ファイルを使用する場合、ファイル名を参照せずにデータベースが起動されます。
- `spfile_name` を指定する場合、デフォルト以外のサーバー・パラメータ・ファイルを作成します。この場合、データベースを起動するときに、サーバー・パラメータ・ファイルを指す単一行の以前のパラメータ・ファイルを作成し、`STARTUP` コマンドで単一行のファイルを指定する必要があります。

参照：

- デフォルトおよびデフォルト以外のサーバー・パラメータ・ファイルを使用したデータベースの起動の詳細は、14-89 ページの「サーバー・パラメータ・ファイルの作成例：」を参照してください。
- デフォルトのパラメータ・ファイル名は、『Oracle9i データベース管理者ガイド』（またはオペレーティング・システム固有のドキュメント）を参照してください。

pfile_name

サーバー・パラメータ・ファイルを作成する元になる以前の初期化パラメータ・ファイル名を指定します。

- `pfile_name` を指定する場合、パラメータ・ファイルはサーバーに存在する必要があります。ファイルがオペレーティング・システムのパラメータ・ファイルのデフォルト・ディレクトリに存在しない場合、フルパス名を指定する必要があります。
- `pfile_name` を指定しない場合、オペレーティング・システムのパラメータ・ファイルのデフォルト・ディレクトリからデフォルトのパラメータ・ファイル名が検索され、使用されます。そのディレクトリにファイルが存在しない場合、エラーが戻されます。

注意： Real Application Clusters 環境では、この文でサーバー・パラメータ・ファイルを指定して作成する前に、まず、すべてのインスタンスのパラメータ・ファイルを 1 つのファイルに結合する必要があります。この手順の実行の詳細は、『Oracle9i Real Application Clusters セットアップおよび構成』を参照してください。

例

サーバー・パラメータ・ファイルの作成例： 次の例は、クライアントの初期化パラメータ・ファイル `t_init1.ora` からデフォルトのサーバー・パラメータ・ファイルを作成します。

```
CREATE SPFILE
FROM PFILE = '$ORACLE_HOME/work/t_init1.ora';
```

注意： 通常、オペレーティング・システムのパラメータ・ファイルには、フルパスのファイル名を指定する必要があります。パスについては、ご使用のオペレーティング・システムの Oracle マニュアルを参照してください。

デフォルトのサーバー・パラメータ・ファイルを作成する場合、その後のデータベースの起動は、次のように `PFILE` パラメータなしで `SQL*Plus` のコマンド `STARTUP` を実行し、サーバー・パラメータ・ファイルを使用します。

```
STARTUP
```

次の例は、クライアントの初期化パラメータ・ファイル `t_init1.ora` からデフォルト以外のサーバー・パラメータ・ファイル `s_params.ora` を作成します。

```
CREATE SPFILE = 's_params.ora'
FROM PFILE = '$ORACLE_HOME/work/t_init1.ora';
```

デフォルト以外のサーバー・パラメータ・ファイルを作成する場合、次の単一行を含む以前のパラメータ・ファイルを最初に作成し、その後でデータベースを起動します。

```
spfile = 's_params.ora'
```

このパラメータ・ファイル名は、ご使用のオペレーティング・システムのネーミング規則に従う必要があります。その後、**STARTUP** コマンドで単一行のパラメータ・ファイルを使用します。次の例では、単一行のパラメータ・ファイルの名前が `new_param.ora` であるとした場合のデータベースの起動方法を示します。

```
STARTUP PFILE=new_param.ora
```

SQL 文 : CREATE SYNONYM ~ CREATE TRIGGER

この章では、次の SQL 文について説明します。

- CREATE SYNONYM
- CREATE TABLE
- CREATE TABLESPACE
- CREATE TEMPORARY TABLESPACE
- CREATE TRIGGER

CREATE SYNONYM

用途

CREATE SYNONYM 文を使用すると、シノニムを作成できます。シノニムとは、表、ビュー、順序、プロシージャ、ストアド・ファンクション、パッケージ、マテリアライズド・ビュー、Java クラス・スキーマ・オブジェクト、ユーザー定義オブジェクト型および別のシノニムに付ける別名です。

シノニムによって、データの独立性および位置の透過性を実現できます。シノニムを使用した場合、どのユーザーが表やビューを所有しているか、どのデータベースに表やビューが格納されているかに関係なく、アプリケーションは機能します。ただし、シノニムはデータベース・オブジェクトに対する権限にかわるものではありません。シノニムを使用するユーザーに対して、そのような権限をあらかじめ付与しておく必要があります。

シノニムを参照できる DML 文は、SELECT、INSERT、UPDATE、DELETE、EXPLAIN PLAN および LOCK TABLE です。

シノニムを参照できる DDL 文は、AUDIT、NOAUDIT、GRANT、REVOKE および COMMENT です。

参照： シノニムの概要については、『Oracle9i データベース概要』を参照してください。

前提条件

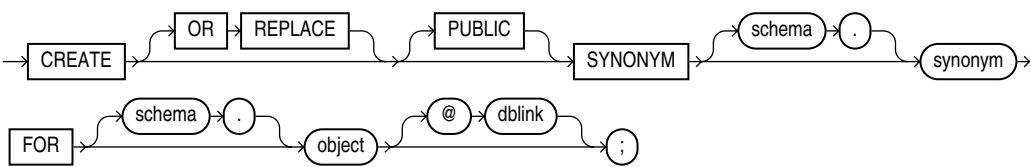
自分のスキーマ内にプライベート・シノニムを作成する場合は、CREATE SYNONYM システム権限が必要です。

他のユーザーのスキーマ内にプライベート・シノニムを作成する場合は、CREATE ANY SYNONYM システム権限が必要です。

パブリック・シノニムを作成する場合は、CREATE PUBLIC SYNONYM システム権限が必要です。

構文

create_synonym::=



セマンティクス

OR REPLACE

既存のシノニムを再作成する場合は、OR REPLACE を指定します。この句を使用すると、既存のシノニムの定義をはじめに削除しなくても、その定義を変更できます。

シノニムの置換の制限事項： OR REPLACE 句は、依存表または有効なユーザー定義オブジェクト型を持つ型シノニムに対して使用できません。

PUBLIC

PUBLIC を指定して、パブリック・シノニムを作成します。パブリック・シノニムには、すべてのユーザーがアクセスできます。ただし、シノニムを使用するには、基礎となるオブジェクトに対する適切な権限が必要です。

オブジェクトの先頭にスキーマ名が指定されていない場合、およびオブジェクトの後にデータベース・リンクが指定されていない場合は、オブジェクトの参照を変換するときのみ、パブリック・シノニムが使用されます。

この句を省略した場合、シノニムはプライベートとなり、定義しているスキーマ内にかぎりアクセスできます。プライベート・シノニム名は、スキーマ内で一意である必要があります。

パブリック・シノニムの作成の注意事項：

- パブリック・シノニムを作成した後、依存表または依存する有効なユーザー定義オブジェクト型が存在する場合、依存オブジェクトと同じスキーマ内には、そのシノニムと同じ名前での別のデータベース・オブジェクトは作成できません。
- 既存のスキーマと同じ名前のパブリック・シノニムを作成しないでください。同じ名前のパブリック・シノニムを作成すると、その名前が使用されるすべての PL/SQL ユニットが無効になります。

schema

シノニムを含むスキーマを指定します。*schema* を省略した場合、自分のスキーマ内にシノニムが作成されます。PUBLIC を指定した場合、スキーマは指定できません。

synonym

作成するシノニムの名前を指定します。

注意： シノニム名の最大長は 32 バイトです。30 バイトを超える名前は、Java の機能にのみ使用できます。30 バイトを超える名前を指定した場合、名前は暗号化され、暗号化された表現でデータ・ディクショナリに格納されます。実際の暗号にはアクセスできません。また、元の指定とデータ・ディクショナリの表現のどちらも、シノニム名としては使用できません。

参照： 15-5 ページの「[CREATE SYNONYM の例 :](#)」および 15-6 ページの「[シノニムの変換例 :](#)」を参照してください。

FOR 句

シノニムを作成するオブジェクトを指定します。シノニムを作成するスキーマ・オブジェクトには、次のものがあります。

- 表またはオブジェクト表
- ビューまたはオブジェクト・ビュー
- 順序
- ストアド・プロシージャ、ファンクションまたはパッケージ
- マテリアライズド・ビュー
- Java クラス・スキーマ・オブジェクト
- ユーザー定義オブジェクト型
- シノニム

スキーマ・オブジェクトは、現在存在している必要はなく、スキーマ・オブジェクトへのアクセス権限も必要ありません。

FOR 句の制限事項： スキーマ・オブジェクトは、パッケージに含めることはできません。

schema オブジェクトが存在するスキーマを指定します。オブジェクトに *schema* を指定しなかった場合、そのスキーマ・オブジェクトは自分のスキーマ内にあるとみなされます。

注意： リモート・データベース上のプロシージャやファンクションに対するシノニムを作成する場合、この CREATE 文で *schema* を指定する必要があります。または、オブジェクトが存在するデータベースにローカル・パブリック・シノニムを作成することもできます。ただし、その後は、プロシージャやファンクションの後続のコールすべてにデータベース・リンクを組み込む必要があります。

dblink データベース・リンクを完全に指定するか、またはデータベース・リンクの一部を指定する場合、スキーマ・オブジェクトが格納されているリモート・データベース上でそのスキーマ・オブジェクトのシノニムを作成することができます。dblink を指定して、*schema* を省略した場合、シノニムは、データベース・リンクで指定されたスキーマ内のオブジェクトを参照します。リモート・データベースでは、オブジェクトを定義しているスキーマを指定することをお勧めします。

dblink を省略した場合、オブジェクトがローカル・データベース上にあるものとみなされます。

データベース・リンクの制限事項： dblink は、Java クラス・シノニムに対して指定できません。

参照：

- データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。
- データベース・リンクの作成方法の詳細は、13-35 ページの「[CREATE DATABASE LINK](#)」を参照してください。

例

CREATE SYNONYM の例： 次の文は、スキーマ hr 内の表 locations に対してシノニム offices を定義します。

```
CREATE SYNONYM offices
FOR hr.locations;
```

次の文は、remote データベース上のスキーマ oe 内の employees 表に対してパブリック・シノニムを作成します。

```
CREATE PUBLIC SYNONYM emp_table
FOR oe.employees@remote.us.oracle.com;
```

別のスキーマ内に実表が定義されている場合は、実表と同じ名前をシノニムに指定することもできます。

シノニムの変換例： Oracle は、オブジェクトの参照を、パブリック・シノニム・レベルで変換する前に、スキーマ・レベルで変換しようとします。たとえば、スキーマ `oe` と `sh` の両方に `customers` という名前の表が存在するとします。次の例では、ユーザー `SYSTEM` が、`oe.customers` に対して `customers` という名前のパブリック・シノニムを作成します。

```
CREATE PUBLIC SYNONYM customers FOR oe.customers;
```

ユーザー `sh` が次の文を発行すると、`sh.customers` から行数が戻されます。

```
SELECT COUNT(*) FROM customers;
```

`oe.customers` から行数を取得するには、ユーザー `sh` は、`customers` の前にスキーマ名を指定する必要があります（ユーザー `sh` は、`oe.customers` に対する `SELECT` 権限も必要です）。

```
SELECT COUNT(*) FROM oe.customers;
```

ユーザー `hr` のスキーマに `customers` という名前のオブジェクトは存在しないが、`hr` が `oe.customers` に対する `SELECT` 権限を持つ場合、`hr` は、パブリック・シノニム `customers` を使用して、`oe` のスキーマ内の `customers` 表にアクセスできます。

```
SELECT COUNT(*) FROM customers;
```

CREATE TABLE

用途

CREATE TABLE 文を使用すると、次の型の表を作成できます。

- **リレーショナル表**。ユーザー・データを格納する基本構造です。
- **オブジェクト表**。列の定義にオブジェクト型を使用する表です。特定の型のオブジェクト・インスタンスを格納するように明示的に定義されます。

オブジェクト型を作成しておき、リレーショナル表の作成時に列の中でそのオブジェクト型を使用することもできます。

問合せを指定しない場合、データを含まない表が作成されます。INSERT 文を使用した場合、表に行を追加できます。表を作成した後、ALTER TABLE 文で ADD 句を指定すると、追加する列、パーティションおよび整合性制約を定義できます。ALTER TABLE 文で MODIFY 句を指定すると、既存の列またはパーティションの定義を変更できます。

参照：

- オブジェクト作成の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』、『Oracle9i データベース管理者ガイド』および 16-3 ページの「[CREATE TYPE](#)」を参照してください。
- 11-2 ページの「[ALTER TABLE](#)」を参照してください。

前提条件

自分のスキーマ内に**リレーショナル表**を作成する場合は、CREATE TABLE システム権限が必要です。他のユーザーのスキーマ内に表を作成する場合は、CREATE ANY TABLE システム権限が必要です。また、表を定義しているスキーマの所有者は、表を格納するため表領域への割当て制限または UNLIMITED TABLESPACE システム権限が必要です。

これらの表権限に加え、**オブジェクト表**またはオブジェクト型の列が存在するリレーショナル表を作成する場合は、表の所有者に、表が参照するすべての型にアクセスするための EXECUTE オブジェクト権限が付与されているかまたは EXECUTE ANY TYPE システム権限が付与されている必要があります。これらの権限は、ロールを介して取得するのではなく、明示的に付与される必要があります。

さらに、表の所有者が表へのアクセス権限を他のユーザーに付与する場合、所有者には、参照する型に対する Grant Option 付きの EXECUTE 権限または Admin Option 付きの EXECUTE ANY TYPE システム権限が必要です。これらの権限を持っていない場合、表の所有者は、表へのアクセス権限を他のユーザーに付与できません。

一意制約または主キー制約を有効にする場合は、表に索引を作成するための権限が必要です。Oracle は、その表を含むスキーマにある一意または主キー列に索引を作成するので、この権限が必要になります。

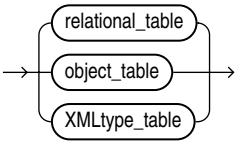
外部表を作成する場合は、外部データが存在するディレクトリに対する READ オブジェクト権限が必要です。

参照：

- 13-61 ページの「[CREATE INDEX](#)」を参照してください。
- 型を使用する表の作成に必要な権限については、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

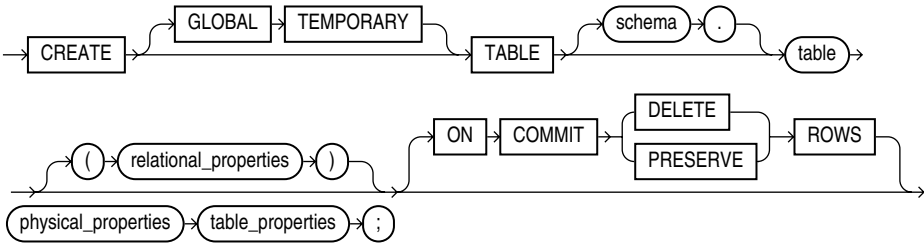
構文

create_table::=

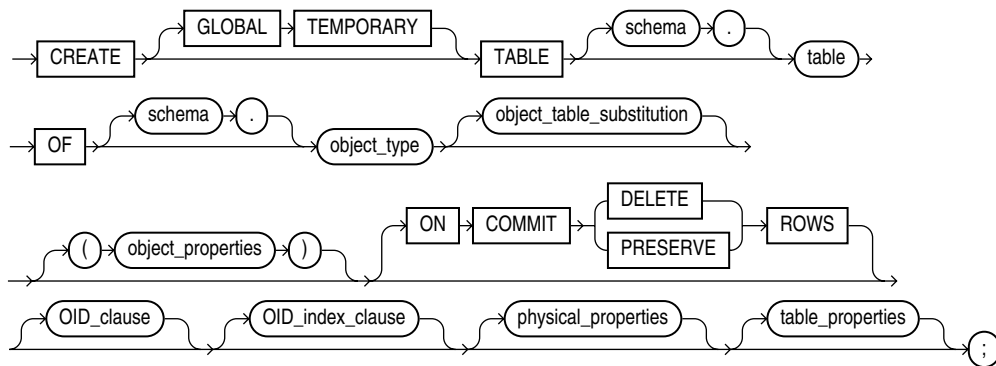


(15-8 ページの「[relational_table::=](#)」、15-9 ページの「[object_table::=](#)」、15-9 ページの「[XMLType_table::=](#)」を参照)

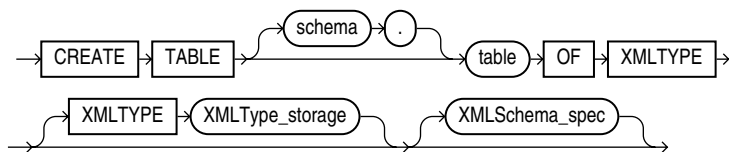
relational_table::=



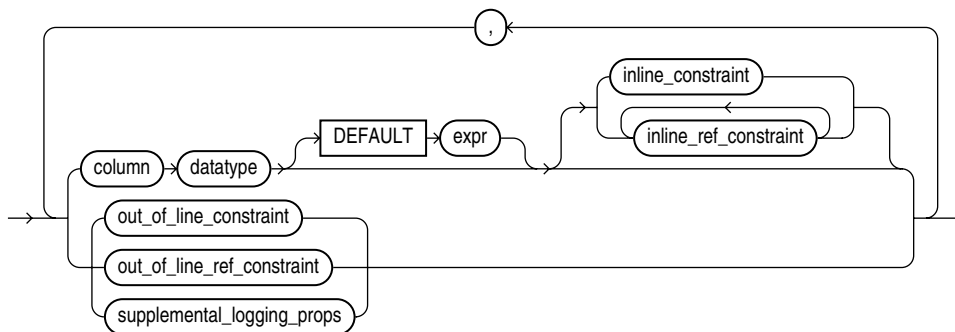
(15-9 ページの「[relational_properties::=](#)」、15-11 ページの「[physical_properties::=](#)」、15-12 ページの「[table_properties::=](#)」を参照)

object_table::=

(15-10 ページの「[object_table_substitution::=](#)」、15-10 ページの「[object_properties::=](#)」、15-10 ページの「[OID_clause::=](#)」、15-10 ページの「[OID_index_clause::=](#)」、15-11 ページの「[physical_properties::=](#)」、15-12 ページの「[table_properties::=](#)」を参照)

XMLType_table::=

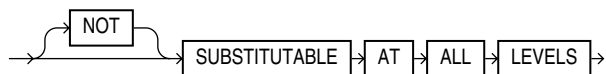
(15-15 ページの「[XMLType_storage::=](#)」、15-15 ページの「[XMLSchema_spec::=](#)」を参照)

relational_properties::=

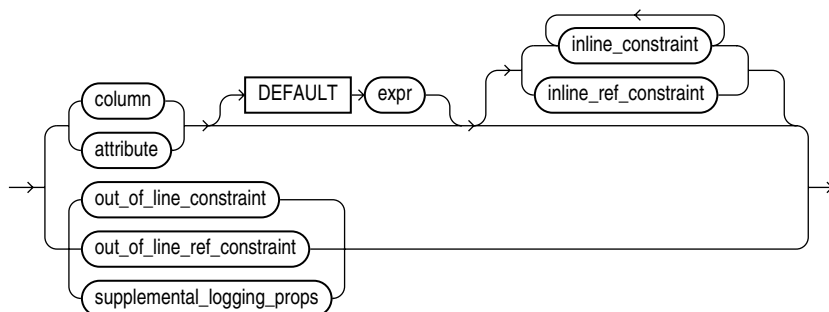
(7-6 ページの「[constraints::=](#)」を参照)

CREATE TABLE

object_table_substitution::=

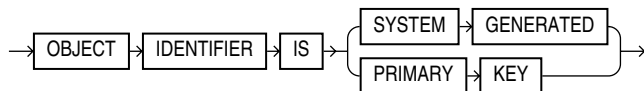


object_properties::=

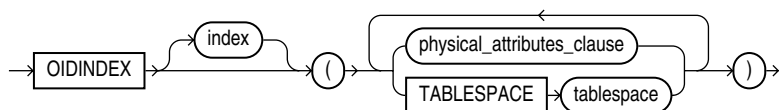


(7-6 ページの「[constraints::=](#)」、15-16 ページの「[supplemental_logging_props::=](#)」を参照)

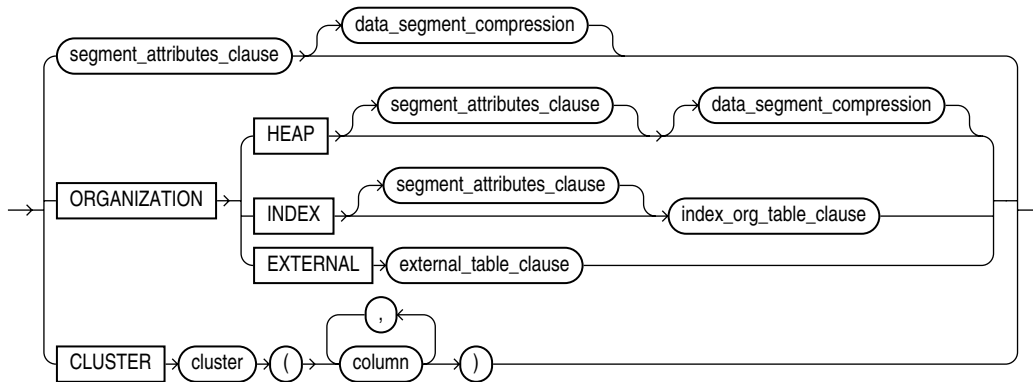
OID_clause::=



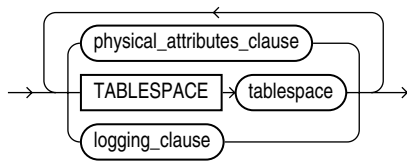
OID_index_clause::=



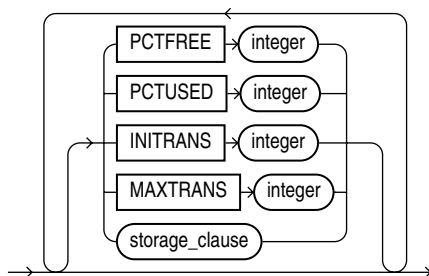
(15-11 ページの「[physical_attributes_clause::=](#)」を参照)

physical_properties::=

(15-11 ページの「[segment_attributes_clause::=](#)」、15-12 ページの「[data_segment_compression::=](#)」、15-16 ページの「[index_org_table_clause::=](#)」、15-17 ページの「[external_table_clause::=](#)」を参照)

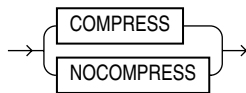
segment_attributes_clause::=

(15-11 ページの「[physical_attributes_clause::=](#)」、15-14 ページの「[logging_clause::=](#)」を参照)

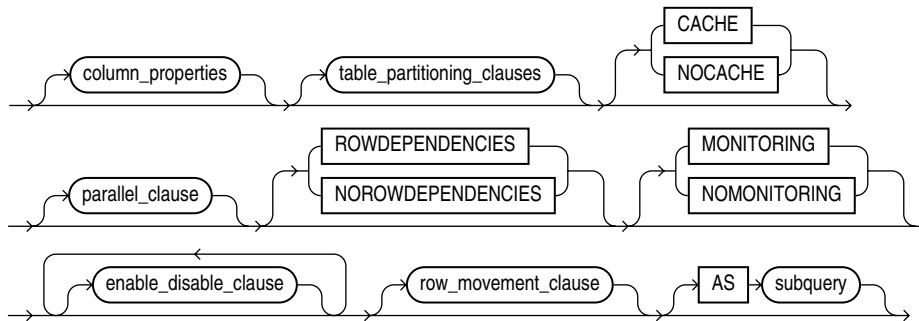
physical_attributes_clause::=

(7-51 ページの「[storage_clause](#)」を参照)

data_segment_compression::=

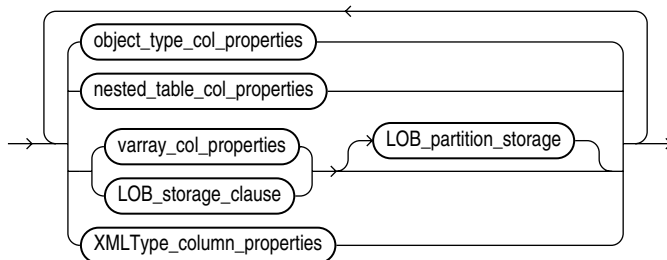


table_properties::=

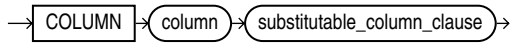
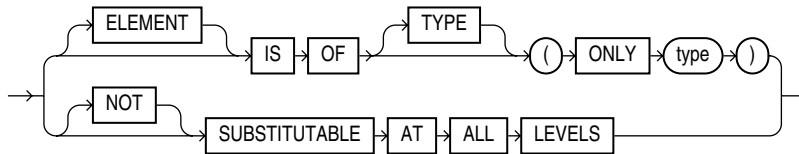
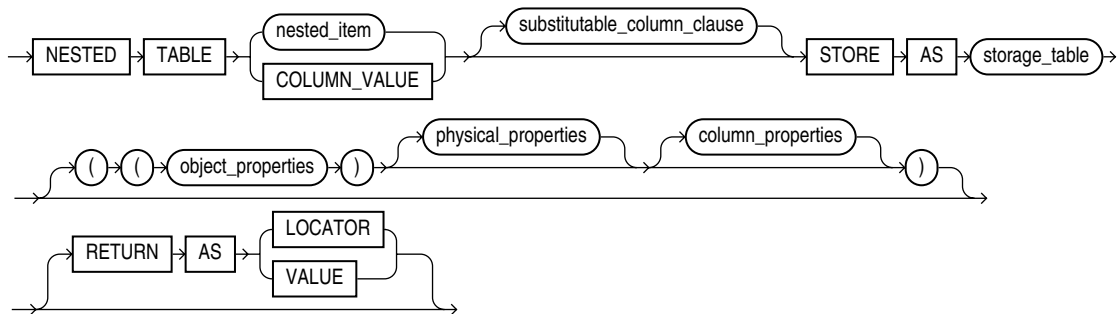


(15-17 ページの「[table_partitioning_clauses::=](#)」、15-22 ページの「[parallel_clause::=](#)」、15-22 ページの「[enable_disable_clause::=](#)」、18-5 ページの「[subquery::=](#)」を参照)

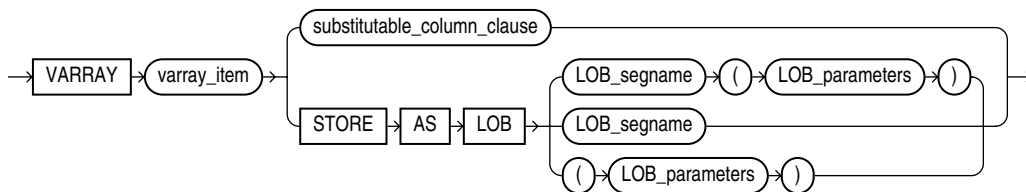
column_properties::=



(15-13 ページの「[object_type_col_properties::=](#)」、15-13 ページの「[nested_table_col_properties::=](#)」、15-13 ページの「[varray_col_properties::=](#)」、15-14 ページの「[LOB_storage_clause::=](#)」、15-15 ページの「[LOB_partition_storage::=](#)」、15-15 ページの「[XMLType_column_properties::=](#)」を参照)

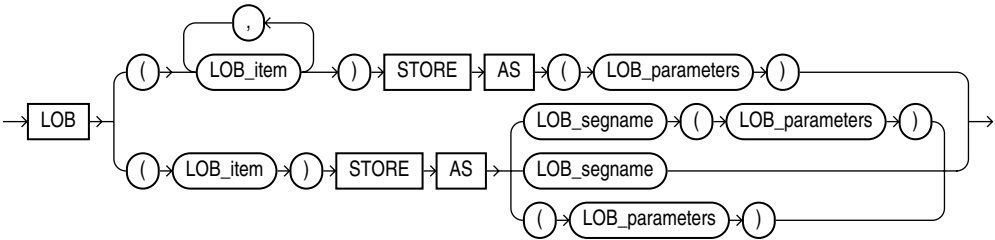
object_type_col_properties::=**substitutable_column_clause::=****nested_table_col_properties::=**

(15-13 ページの「[substitutable_column_clause::=](#)」、15-10 ページの「[object_properties::=](#)」、15-11 ページの「[physical_properties::=](#)」、15-12 ページの「[column_properties::=](#)」を参照)

varray_col_properties::=

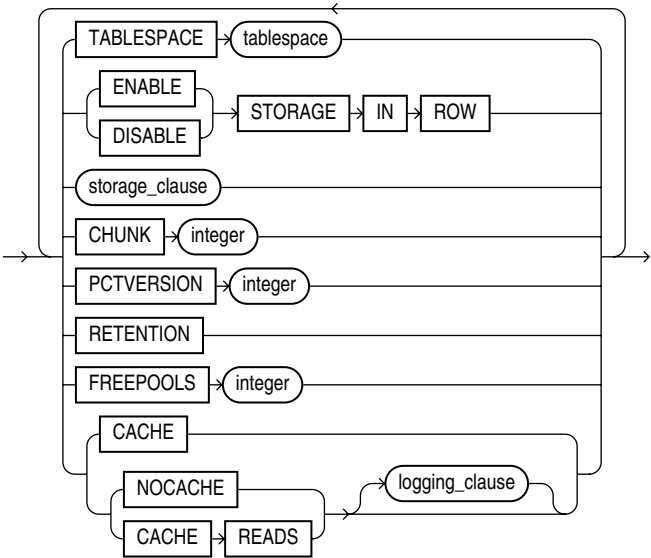
(15-13 ページの「[substitutable_column_clause::=](#)」、15-14 ページの「[LOB_parameters::=](#)」を参照)

LOB_storage_clause::=



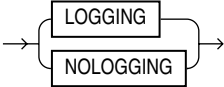
(15-14 ページの「LOB_parameters::=」を参照)

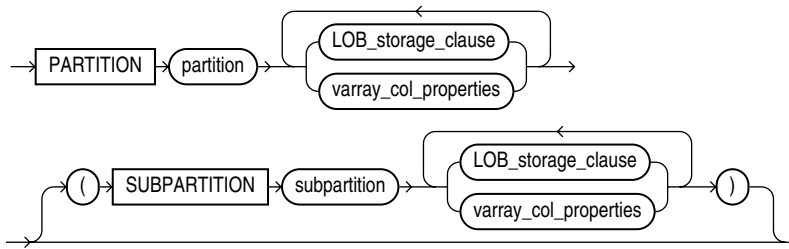
LOB_parameters::=



(7-53 ページの「storage_clause::=」を参照)

logging_clause::=

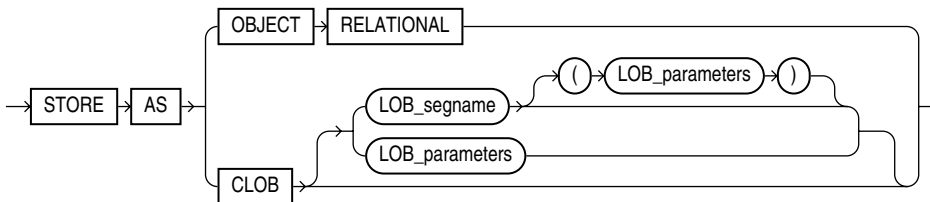


LOB_partition_storage::=

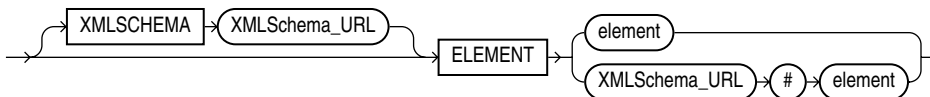
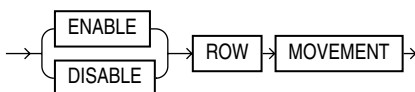
(15-14 ページの「[LOB_storage_clause::=](#)」、15-13 ページの「[varray_col_properties::=](#)」を参照)

XMLType_column_properties::=

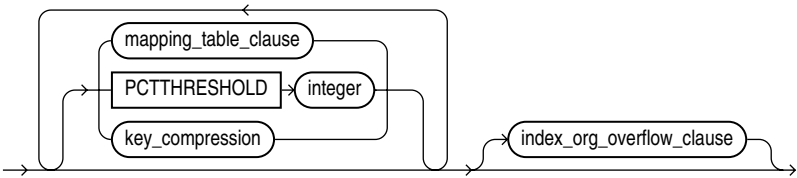
(15-15 ページの「[XMLType_storage::=](#)」、15-15 ページの「[XMLSchema_spec::=](#)」を参照)

XMLType_storage::=

(15-14 ページの「[LOB_parameters::=](#)」を参照)

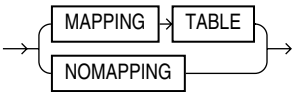
XMLSchema_spec::=**row_movement_clause::=**

index_org_table_clause::=

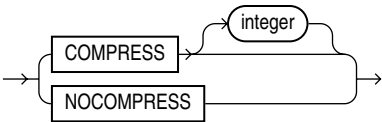


(15-16 ページの「`mapping_table_clause::=`」、15-16 ページの「`key_compression::=`」、15-16 ページの「`index_org_overflow_clause::=`」を参照)

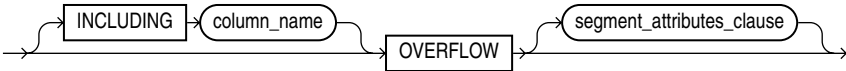
mapping_table_clause::=



key_compression::=

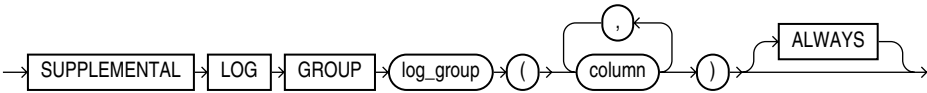


index_org_overflow_clause::=



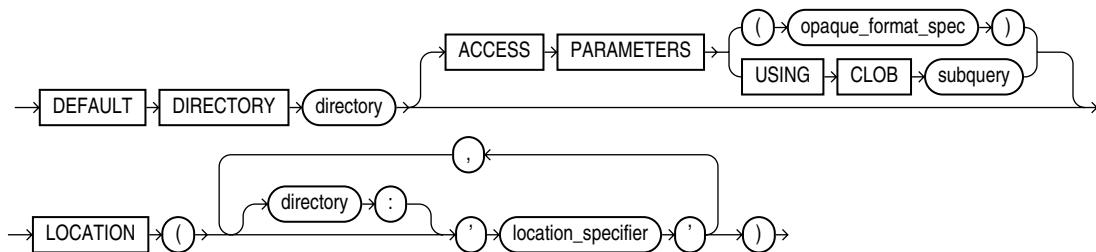
(15-11 ページの「`segment_attributes_clause::=`」を参照)

supplemental_logging_props::=

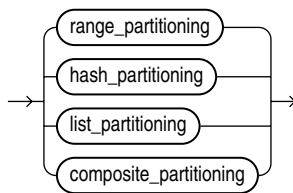


external_table_clause::=

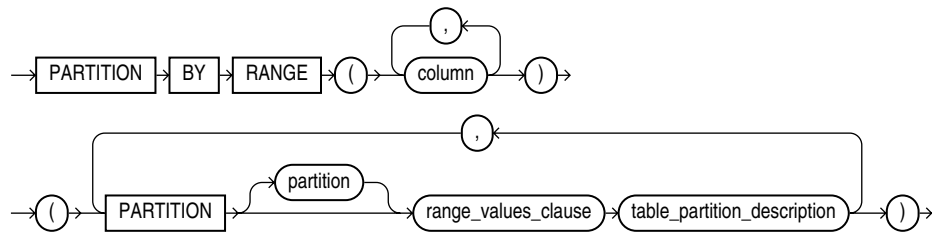
(15-17 ページの「[external_data_properties::=](#)」を参照)

external_data_properties::=

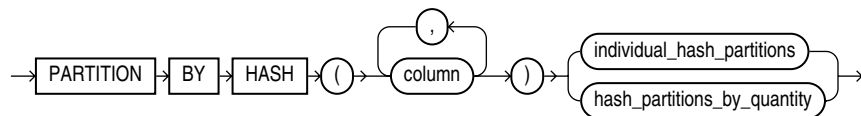
(*opaque_format_spec*: *opaque_format_spec* の値の指定方法は、『Oracle9i データベース・ユーティリティ』を参照してください。)

table_partitioning_clauses::=

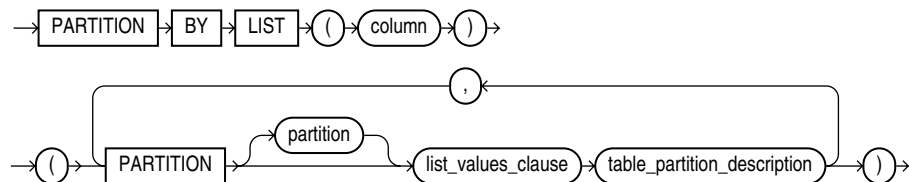
(15-18 ページの「[range_partitioning::=](#)」、15-18 ページの「[hash_partitioning::=](#)」、15-18 ページの「[list_partitioning::=](#)」、15-19 ページの「[composite_partitioning::=](#)」を参照)

range_partitioning::=

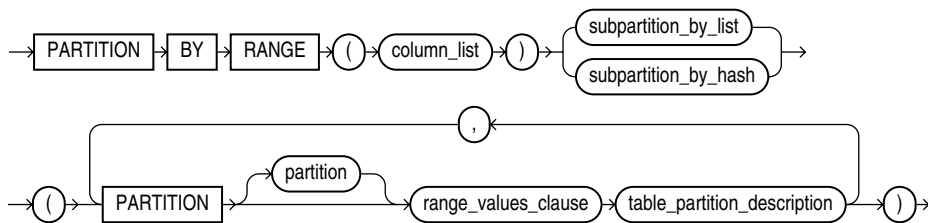
(15-20 ページの「[range_values_clause::=](#)」、15-21 ページの「[table_partition_description::=](#)」を参照)

hash_partitioning::=

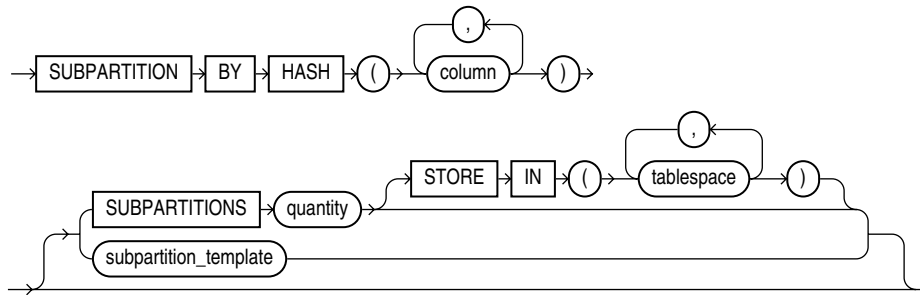
(15-19 ページの「[individual_hash_partitions::=](#)」、15-20 ページの「[hash_partitions_by_quantity::=](#)」を参照)

list_partitioning::=

(15-21 ページの「[list_values_clause::=](#)」、15-21 ページの「[table_partition_description::=](#)」を参照)

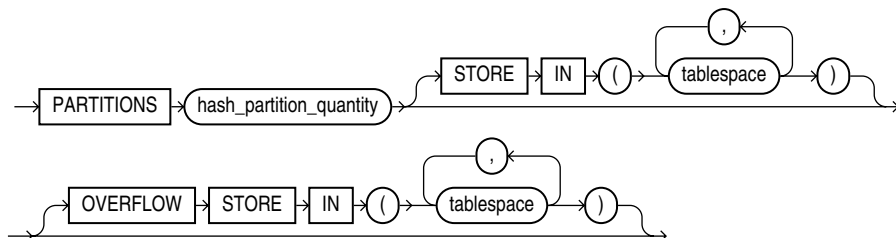
composite_partitioning::=

(15-20 ページの「[subpartition_by_list::=](#)」、15-19 ページの「[subpartition_by_hash::=](#)」、15-20 ページの「[range_values_clause::=](#)」、15-21 ページの「[table_partition_description::=](#)」を参照)

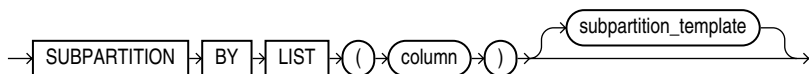
subpartition_by_hash::=

CREATE TABLE

hash_partitions_by_quantity::=

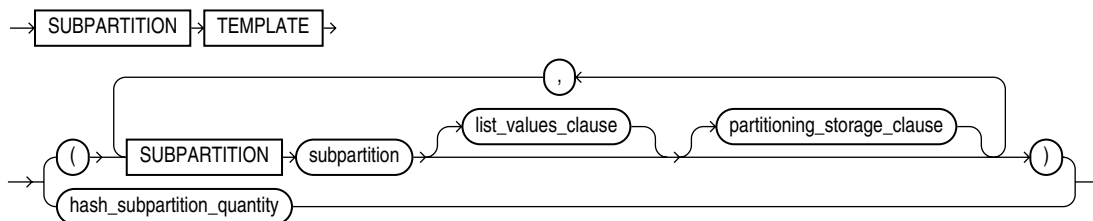


subpartition_by_list::=



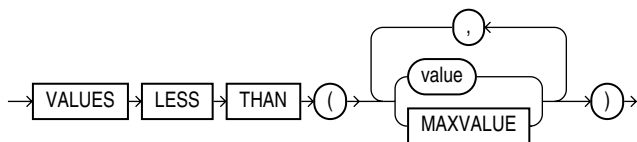
(15-20 ページの「[subpartition_template::=](#)」を参照)

subpartition_template::=

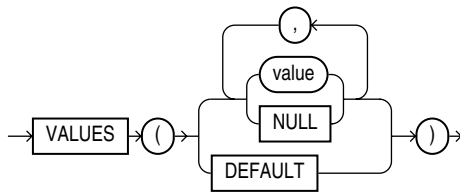


(15-21 ページの「[list_values_clause::=](#)」、15-22 ページの「[partitioning_storage_clause::=](#)」を参照)

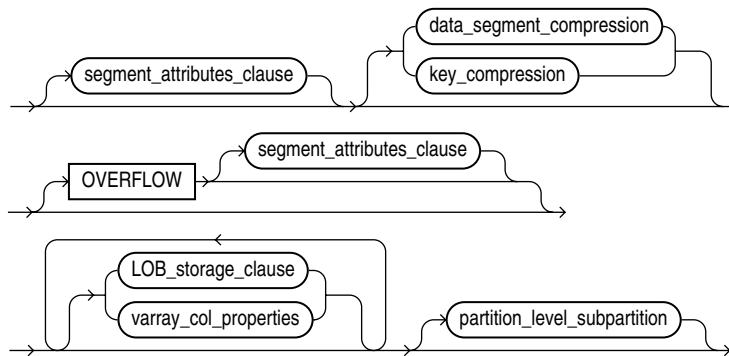
range_values_clause::=



list_values_clause::=

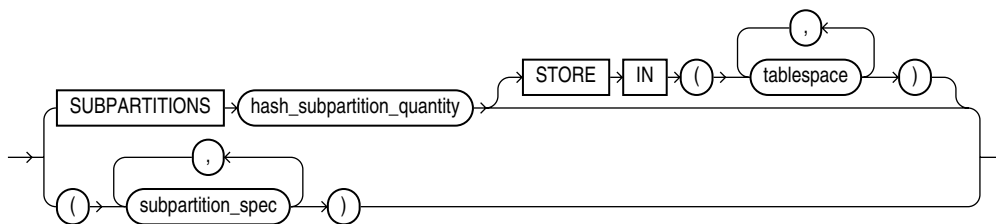


table_partition_description::=



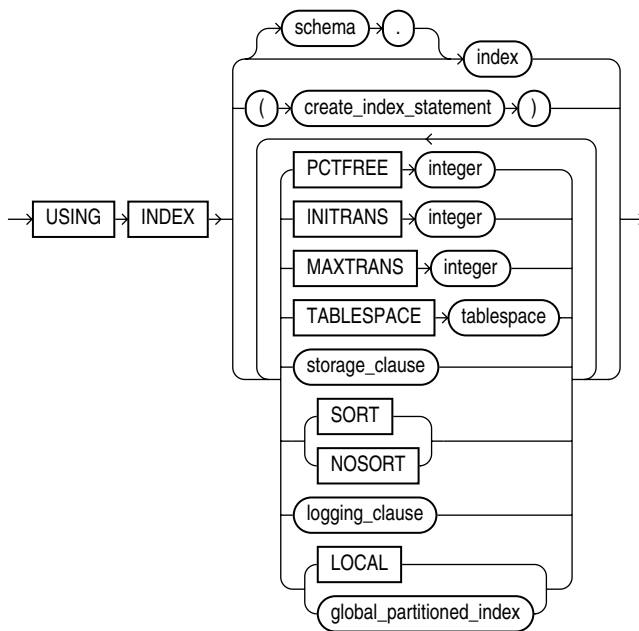
(15-11 ページの「[segment_attributes_clause::=](#)」、15-12 ページの「[data_segment_compression::=](#)」、15-14 ページの「[LOB_storage_clause::=](#)」、15-13 ページの「[varray_col_properties::=](#)」、15-21 ページの「[partition_level_subpartition::=](#)」を参照)

partition_level_subpartition::=



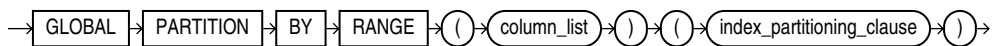
(15-22 ページの「[subpartition_spec::=](#)」を参照)

using_index_clause::=



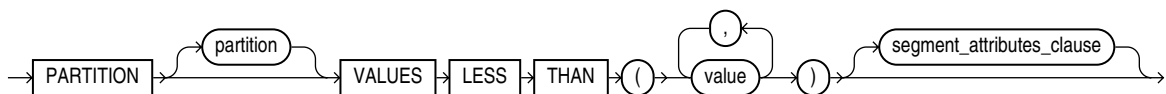
(13-62 ページの「[create_index::=](#)」、7-42 ページの「[logging_clause::=](#)」、15-23 ページの「[global_partitioned_index::=](#)」を参照)

global_partitioned_index::=



(15-23 ページの「[index_partitioning_clause::=](#)」を参照)

index_partitioning_clause::=



(15-11 ページの「[segment_attributes_clause::=](#)」を参照)

セマンティクス

relational_table

GLOBAL TEMPORARY

GLOBAL TEMPORARY を指定して、表が一時的で、その定義がすべてのセッションで参照できることを示します。一時表のデータは、データを表に挿入するセッションでのみ参照できます。

一時表は、標準表の定義に準拠する定義を持ちますが、**セッション固有**または**トランザクション固有**データのいずれかを含みます。データがセッション固有であるか、トランザクション固有であるかは、**ON COMMIT** キーワードで指定します。

注意： セッションがバインドされていない場合にのみ、一時表で DDL 操作 (ALTER TABLE、DROP TABLE、CREATE INDEX など) を実行できます。セッションを一時表にバインドするには、一時表で INSERT 操作を実行します。セッションを一時表からアンバインドするには、TRUNCATE 文を発行するか、セッションを終了します。また、トランザクション固有の一時表からアンバインドするには、COMMIT または ABORT 文を発行します。

参照： 一時表の詳細は、『Oracle9i データベース概要』を参照してください。また、15-63 ページの「[一時表の例 :](#)」も参照してください。

一時表の制限事項

- 一時表は、パーティション化、索引構成化またはクラスタ化できません。
- 一時表には、外部キー制約を指定できません。
- 一時表は、ネストした表または VARRAY 型の列を含むことはできません。
- *LOB_storage_clause* の TABLESPACE、*storage_clause*、*logging_clause*、MONITORING、NOMONITORING または *LOB_index_clause* は指定できません。
- 一時表にパラレル DML およびパラレル問合せはサポートされていません (パラレル・ヒントは無視されます。*parallel_clause* を指定すると、エラーが戻されます)。
- *segment_attributes_clause*、*nested_table_col_properties* または *parallel_clause* は指定できません。
- 一時表での分散トランザクションはサポートされていません。

schema

表を含むスキーマを指定します。*schema* を省略した場合、自分のスキーマ内に表が作成されます。

table

作成する表（またはオブジェクト表）の名前を指定します。

参照： 15-62 ページの「[一般的な例](#)」を参照してください。

relational_properties

relational_properties 句を使用すると、リレーショナル表のコンポーネントを指定できます。

column

表の列の名前を指定します。

AS *subquery* を指定する場合、索引構成表を作成しないかぎり、*column* および *datatype* を省略できます。索引構成表の作成時に AS *subquery* を指定する場合は、*column* を指定し、*datatype* を省略する必要があります。

表の列の絶対最大数は 1000 です。ただし、オブジェクト表（または、オブジェクトの列、ネストした表、VARRAY または REF 型のリレーショナル表）を作成する場合、制限の 1000 列までをカウントする有効な非表示列を作成して、ユーザー定義型の列をリレーショナル列にマップします。

datatype

列のデータ型を指定します。

注意： 次の場合は、*datatype* は省略できます。

- AS *subquery* を指定する場合（索引構成表を作成して AS *subquery* を指定する場合は、データ型を省略する必要があります。）
 - 文が、参照整合性制約で外部キーとして列を指定する場合（Oracle では、参照整合性制約の参照キーに対応する列のデータ型が列に自動的に割り当てられます。）
-
-

表の列のデータ型の制限事項： ROWID 型の列を指定することはできますが、それらの列の値が有効な行 ID であることは保証されません。

参照： Oracle が提供するデータ型については、2-2 ページの「[データ型](#)」を参照してください。

DEFAULT

DEFAULT 句によって、後続の INSERT 文が列の値を省略した場合に値が列に割り当てられるように指定できます。式のデータ型は、列のデータ型と一致する必要があります。列には、この式が入る長さが必要です。

DEFAULT 式には、リテラル引数、列の参照またはネストしたファンクションの起動を戻さない、任意の SQL ファンクションを含めることができます。

デフォルトの列値の制限事項： DEFAULT 式に、PL/SQL ファンクション、他の列（LEVEL、PRIOR および ROWNUM 疑似列）への参照または完全に指定されていない日付定数は指定できません。

参照： `expr` の構文については、4-2 ページの「SQL 式」を参照してください。

inline_ref_constraint | *out_of_line_ref_constraint*

これらの句を使用すると、REF 型の列を指定できます。これらの句の唯一の違いは、表レベルで *out_of_line_ref_constraint* を指定することです。このため、定義する REF 型の列または属性を識別する必要があります。REF 型の列または属性を識別した後、*inline_ref_constraint* を指定します。

参照： これらの制約の構文、詳細および使用例は、7-5 ページの「[constraints](#)」を参照してください。

inline_constraint

inline_constraint を使用すると、整合性制約を列定義の一部として定義できます。

オブジェクト型の列のスカラー属性に、一意制約、主キー制約および参照制約を作成できます。また、オブジェクト型の列の NOT NULL 制約、オブジェクト型の列またはオブジェクト型の列の属性を参照する CHECK 制約も作成できます。

参照： これらの制約の構文、詳細および使用例は、7-5 ページの「[constraints](#)」を参照してください。

out_of_line_constraint

out_of_line_constraint 構文を使用すると、整合性制約を表定義の一部として定義できます。

注意： DEFERRABLE 以外の主キー制約を索引構成表に指定してください。

参照： `out_of_line_constraint` 構文の詳細は、7-5 ページの「[constraints](#)」を参照してください。

supplemental_logging_props

`supplemental_logging_props` によって、追加のデータがログ・ストリームに入れられ、ログに基づくツール製品がサポートされます。

ON COMMIT

ON COMMIT 句は、一時表を作成する場合のみに適用されます。この句は、一時表のデータがトランザクションまたはセッションの存続期間中保持されるかどうかを指定します。

DELETE ROWS トランザクション固有の一時表に DELETE ROWS を指定します（デフォルト）。各コミット後に表を切り捨てます（すべての行を削除します）。

PRESERVE ROWS セッション固有の一時表に対して、PRESERVE ROWS を指定します。セッション終了時に表を切り捨てます（すべての行を削除します）。

physical_properties

物理プロパティは、エクステンツおよびセグメントの処理、および表の記憶特性に関係します。

segment_attributes_clause

physical_attributes_clause

`physical_attributes_clause` は、PCTFREE、PCTUSED、INITRANS および MAXTRANS パラメータの値、および表の記憶特性を指定します。

- 非パーティション表の場合、指定した各パラメータおよび記憶特性は、表に関連付けられたセグメントの実際の物理属性となります。
- パーティション表の場合、パーティション作成文の PARTITION 句で明示的に値を上書きしないかぎり、指定したパラメータおよび記憶特性の値は、CREATE 文（および後続の ALTER TABLE ... ADD PARTITION 文）で指定するすべてのパーティションに関連付けられたセグメントのデフォルト物理属性になります。

この句を省略すると、次のデフォルト値が使用されます。

- PCTFREE: 10
- PCTUSED: 40
- INITRANS: 1
- MAXTRANS: データ・ブロックのサイズに依存

参照：

- 物理属性パラメータの詳細は、7-47 ページの「[physical_attributes_clause](#)」を参照してください。
- 記憶域パラメータの詳細は、7-51 ページの「[storage_clause](#)」を参照してください。
- 15-62 ページの「[記憶域の例 :](#)」を参照してください。

TABLESPACE

Oracle が、表、オブジェクト表 OID 索引、パーティション、LOB のデータ・セグメント、LOB の索引セグメントまたは索引構成表のオーバフロー・データ・セグメントを作成する表領域を指定します。TABLESPACE を省略した場合、その表を含むスキーマの所有者のデフォルトの表領域内に作成されます。

1 つ以上の LOB 列を持つヒープ構成表の場合、LOB 記憶域に対する TABLESPACE を省略すると、表を作成する表領域に LOB データおよび索引セグメントが作成されます。

ただし、1 つ以上の LOB 列を持つ索引構成表の場合、TABLESPACE を省略すると、索引構成表の主キー索引セグメントが作成された表領域に、LOB データおよび索引セグメントが作成されます。

非パーティション表の場合、TABLESPACE に指定する値は、表に関連付けられたセグメントの実際の物理属性となります。パーティション表の場合、TABLESPACE に指定する値は、PARTITION 記述で TABLESPACE に指定しないかぎり、この CREATE 文（および後続の ALTER TABLE ... ADDPARTITION 文）で指定されたすべてのパーティションに関連付けられたセグメントのデフォルト物理属性となります。

参照： 表領域の詳細は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

logging_clause

表（および制約のために必要な索引）、パーティションまたは LOB の記憶特性の作成を REDO ログ・ファイルに記録する（LOGGING）かしないか（NOLOGGING）を指定します。表のロギング属性は、その索引の属性に依存しません。

表、パーティションまたは LOB の記憶域に対して、後で実行されるダイレクト・ローダー（SQL*Loader）操作およびダイレクト・パス・インサート操作のログをとる（LOGGING）かとらない（NOLOGGING）かも指定します。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

data_segment_compression

data_segment_compression 句は、ヒープ構成表にのみ有効です。この句を使用すると、ディスク使用量を削減するためにデータ・セグメントを圧縮するかどうかを指定できます。COMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用可能になります。NOCOMPRESS キーワードを指定すると、データ・セグメントの圧縮が使用禁止になります。デフォルトは NOCOMPRESS です。

データ・セグメントの圧縮を使用可能にすると、Oracle でデータの圧縮が効果的であると判断された場合にデータが圧縮されます。LOB データ・セグメントは圧縮されません。この句は、挿入操作や更新操作が少ないデータ・ウェアハウスなどの環境で特に有効です。データ・セグメントの圧縮は、ヒープ構成表の次の部分に対して指定できます。

- ヒープ構成表全体の場合は、*relational_table* または *object_table* の *physical_properties* 句で指定します。
- レンジ・パーティションの場合は、*range_partitioning* 句の *table_partition_description* で指定します。
- リスト・パーティションの場合は、*list_partitioning* 句の *table_partition_description* で指定します。
- ネストした表の記憶表の場合は、*nested_table_col_properties* 句で指定します。

参照： 圧縮率の計算の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。また、データ圧縮の使用例は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

データ・セグメントの圧縮の制限事項：

- 索引構成表、オーバーフロー・セグメント、オーバーフロー・セグメントのパーティションまたは索引構成表のマッピング表セグメントには、データ・セグメントの圧縮を指定できません。
- ハッシュ・パーティション、ハッシュ・サブパーティションまたはリスト・サブパーティションには、データ・セグメントの圧縮を指定できません。
- 外部表にはデータ・セグメントの圧縮を指定できません。

RECOVERABLE | UNRECOVERABLE

これらのキーワードは以前のリリースのもので、それぞれ LOGGING および NOLOGGING に置き換えられています。RECOVERABLE および UNRECOVERABLE は、下位互換用のためにサポートされていますが、LOGGING および NOLOGGING キーワードを使用することをお勧めします。

[UN]RECOVERABLE の制限事項：

- パーティション表または LOB 記憶特性に RECOVERABLE を指定できません。
- パーティション表または索引構成表に UNRECOVERABLE を指定できません。
- AS *subquery* でのみ UNRECOVERABLE を指定できます。

ORGANIZATION

ORGANIZATION 句によって、表のデータ行が格納される順序を指定できます。

HEAP HEAP は、構成する *table* のデータ行の格納順序を特定しないことを示します。これはデフォルト値です。

INDEX INDEX は、*table* を索引構成表として作成することを示します。索引構成表では、表の主キーが定義された索引内にデータ行が格納されます。

EXTERNAL EXTERNAL は、表がデータベースの外部にある読み専用表であることを示します。

参照： 15-67 ページの「[外部表の例 :](#)」を参照してください。

index_org_table_clause

index_org_table_clause を使用すると、索引構成表を作成できます。主キーに基づく索引にある表の行（主キー列の値と非キー列の値の両方）がメンテナンスされます。このため、索引構成表は主キーベースのアクセスおよび操作に最適です。索引構成表は、次のいずれかの表です。

- CREATE INDEX 文を使用して主キーベースで索引付けされるクラスタ化されていない表。
- 索引クラスタに格納されるクラスタ化表。索引クラスタは、表に対する主キーをクラスタ・キーにマップする CREATE CLUSTER 文を使用して作成されます。

主キーは行を一意に識別するため、索引構成表には主キーを指定してください。主キーには DEFERRABLE を指定できません。索引構成表の行に直接アクセスする場合は、ROWID のかわりに主キーを使用してください。

索引構成表がパーティション化され、LOB 列を含む場合、最初に *index_org_table_clause*、次に *LOB_storage_clause*、その後に適切な *table_partitioning_clauses* を指定する必要があります。

参照： 15-66 ページの「[索引構成表の例 :](#)」を参照してください。

注意： 索引構成表を作成する場合は、CREATE TABLE ...AS SELECT 文の副問合せで、TO_LOB ファンクションを使用して LONG 列を LOB 列に変換することはできません。LONG 列を含まない索引構成表を作成し、INSERT ... AS SELECT 文で TO_LOB ファンクションを使用してください。

索引構成表の制限事項：

- ROWID 型の列は、索引構成表に指定できません。
- *composite_partitioning_clause* は、索引構成表に指定できません。

PCTTHRESHOLD integer 索引ブロック内で、索引構成表の行を格納するために確保されている領域の割合（パーセント）を指定します。PCTTHRESHOLD は、主キーを保持するために十分な大きさである必要があります。指定したしきい値を超える列から始まる行の後続列はすべて、オーバーフロー・セグメントに格納されます。PCTTHRESHOLD は 1 ～ 50 の値を取る必要があります。PCTTHRESHOLD を指定しない場合のデフォルト値は 50 です。

PCTTHRESHOLD の制限事項： PCTTHRESHOLD は、索引構成表の個別パーティションに対して指定できません。

mapping_table_clause MAPPING TABLE によって、ローカルのマッピングが物理 ROWID に作成され、ヒープ構成表に格納されます。このマッピングは、索引構成表のビットマップ索引の作成に必要です。

マッピング表は、親である索引構成表と同じ表領域に作成されます。問合せ、DML 操作またはマッピング表の記憶特性の変更は実行できません。

表のマッピングの制限事項： パーティション化された索引構成表には、*mapping_table_clause* を指定できません。

key_compression *key_compression* によって、索引構成表のキー圧縮を使用可能または使用禁止にできます。

- COMPRESS を指定して、**キー圧縮**を使用可能にします。これによって、索引構成表の主キー列の値が重複しなくなります。*integer*を使用して、接頭辞の長さ（圧縮する接頭辞列数）を指定します。

接頭辞の長さの有効範囲は、1 ～（主キー列数 -1）までです。デフォルトでは（主キー列数 -1）になります。

- NOCOMPRESS を指定して、索引構成表でのキー圧縮を使用禁止にします。これはデフォルト値です。

索引構成表のキー圧縮の制限事項： パーティション・レベルでは、COMPRESS を指定できませんが、*integer* で接頭辞の長さを指定できません。

index_org_overflow_clause *index_org_overflow_clause* によって、指定されたしきい値を超える索引構成表のデータ行を、この句で指定したデータ・セグメントに格納できます。

- 索引構成表を作成した場合、各列の最大サイズを評価し、最大限の列を概算します。オーバーフロー・セグメントが必要であるにもかかわらず、OVERFLOW を指定していない場合は、エラーが発生し CREATE TABLE 文は実行されません。このチェック機能によって、索引構成表に対する後続の DML 操作が、オーバーフロー・セグメントがないために失敗することを防げます。
- OVERFLOW キーワードの後の句に指定するすべての物理属性および記憶特性は、表のオーバーフロー・セグメントにのみ適用されます。索引構成表自体の物理属性と記憶特性、すべてのパーティションに対するデフォルト値、および各パーティションに対する値は、このキーワードの前に指定する必要があります。
- 索引構成表に 1 つ以上の LOB 列が含まれる場合は、LOB がインラインに格納できるほど小さい場合でも、OVERFLOW を指定しないと、アウトラインに格納されます。
- 表がパーティション化されている場合、オーバーフロー・データ・セグメントが主キー索引セグメントと同一レベルでパーティション化されます。

INCLUDING *column_name* 索引構成表の行を索引部分とオーバーフロー部分に分割する列を指定します。主キー列は常に索引に格納されます。*column_name* は、最後の主キー列でもその他の主キー以外の列でもかまいません。*column_name* に続くすべての主キー以外の列は、オーバーフロー・データ・セグメントに格納されます。

INCLUDING 句の制限事項： 索引構成表の個々のパーティションにこの句は指定できません。

注意： *column_name* で行を分割しようとした場合に、行の索引部分のサイズが、PCTTHRESHOLD の値（指定またはデフォルト）を超えると、Oracle は PCTTHRESHOLD の値に基づいて、行を切り離します。

external_table_clause *external_table_clause* によって、外部表を作成できます。外部表は読み込み専用表で、そのメタデータはデータベースに格納されますが、データはデータベースの外部に格納されます。外部表では、データを最初はデータベースにロードせずに、データベースの外部でデータを問い合わせることができます。

参照： 外部表の使用方法的詳細は、『Oracle9i データ・ウェアハウス・ガイド』、『Oracle9i データベース管理者ガイド』および『Oracle9i データベース・ユーティリティ』を参照してください。

外部表の場合、データベースにデータが存在しないため、表の作成時に通常は使用可能な句の小規模のサブセットを定義します。

- `relational_properties` 句内では、`column`、`datatype` および `inline_constraint` のみを指定できます。
- `physical_properties_clause` 内では、表の構成 (`ORGANIZATION EXTERNAL external_table_clause`) のみを指定できます。
- `table_properties` 句内では、`parallel_clause` のみを指定できます。
`parallel_clause` を指定すると、外部データに対する後続の問合せをパラレル化できます。

参照： 15-67 ページの「[外部表の例:](#)」を参照してください。

外部表の制限事項：

- `external_table_clause` を指定する場合、同じ CREATE TABLE 文で他の句を指定することはできません。
- 外部表を一時表にすることはできません。
- 外部表には制約を指定できません。
- 外部表にオブジェクト型列、LOB 列または LONG 列を含めることはできません。

TYPE

TYPE `access_driver_type` を指定すると、外部表の **アクセス・ドライバ** を指定できます。アクセス・ドライバは、データベースに対する外部表を解析する API です。TYPE を指定しない場合、デフォルトのアクセス・ドライバ `ORACLE_LOADER` が使用されます。

参照： `ORACLE_LOADER` アクセス・ドライバの詳細は、『Oracle9i データベース・ユーティリティ』を参照してください。

DEFAULT DIRECTORY

DEFAULT DIRECTORY を指定すると、外部データ・ソースが存在するファイル・システムのディレクトリに対応するデフォルト・ディレクトリ・オブジェクトを 1 つ指定できます。デフォルト・ディレクトリは、アクセス・ドライバから使用でき、エラー・ログなどの補助ファイルを格納できます。

ACCESS PARAMETERS

オプションの **ACCESS PARAMETERS** 句を指定すると、その外部表用のアクセス・ドライバ仕様のパラメータに値を割り当てることができます。

- *opaque_format_spec* を指定すると、パラメータおよびその値をリストできます。*opaque_format_spec* の値の指定方法は、『Oracle9i データベース・ユーティリティ』を参照してください。

opaque_format_spec で指定するファイル名は、表定義の列と一致している必要があります。表定義の列と一致していない *opaque_format_spec* のフィールドは無視されます。

- **USING CLOB subquery** を指定すると、副問合せを使用して、パラメータおよびその値を導出できます。副問合せには、集合演算子または **ORDER BY** 句を含めません。1 つの CLOB データ型を含む単一行を戻します。

opaque_format_spec でパラメータを指定する場合、または副問合せを使用してそれらを導出する場合は、この句は解析されません。外部データのコンテキスト情報は、アクセス・ドライバが解析します。

LOCATION

LOCATION 句を指定すると、1 つ以上の外部データ・ソースを指定できます。通常、*location_specifier* はファイルで、必要ではありません。**Oracle** はこの句を解析しません。外部データのコンテキスト情報は、アクセス・ドライバが解析します。

REJECT LIMIT

REJECT LIMIT 句を指定すると、**Oracle** エラーが戻され、問合せが異常終了する前の、外部データの問合せで許容される変換エラーの数を指定できます。デフォルト値は 0 です。

CLUSTER 句

CLUSTER 句は、表がクラスタの一部であることを示します。この句で指定する各列は、クラスタの各列に対応する表の列となります。一般に、表のクラスタ列は、主キーまたは主キーの一部を構成する 1 つ以上の列です。

参照： 13-2 ページの「[CREATE CLUSTER](#)」を参照してください。

クラスタ・キー内の列ごとに表から 1 つの列を指定します。列は、名前ではなく位置で一致させます。

クラスタ化表はクラスタの領域割当てを使用します。このため、**PCTFREE**、**PCTUSED**、**INITRANS** または **MAXTRANS** パラメータ、**TABLESPACE** 句または *storage_clause* を **CLUSTER** 句とともに使用しないでください。

クラスタ化表の制限事項：

- オブジェクト表および LOB 列を含む表はクラスタの一部にはできません。
- クラスタが同じ ROWDEPENDENCIES または NOROWDEPENDENCIES 設定で作成されていないかぎり、CLUSTER を ROWDEPENDENCIES または NOROWDEPENDENCIES とともに指定することはできません。

table_properties***column_properties***

column_properties 句を使用すると、列の記憶域属性を指定できます。

object_type_col_properties

object_type_col_properties を使用すると、オブジェクト列、属性、あるいは列または属性の集合要素の記憶特性を指定できます。

column *column* には、オブジェクト列または属性を指定します。

substitutable_column_clause *substitutable_column_clause* を使用すると、同じ階層のオブジェクト列または属性が互いに置換可能かどうかを指定できます。列が特定の型であるか、サブタイプのインスタンスを含むものであるか、またはその両方を指定できます。

- ELEMENT を指定すると、コレクションの要素型が宣言した型のサブタイプに制約されます。
- IS OF [TYPE] (ONLY type) 句を指定すると、オブジェクト列の型が宣言した型のサブタイプに制約されます。
- NOT SUBSTITUTABLE AT ALL LEVELS を指定すると、オブジェクト列がサブタイプに対応するインスタンスを持つことはできません。また、置換は、埋込みオブジェクト属性、埋込みのネストした表および VARRAY の要素には使用できません。デフォルトは、SUBSTITUTABLE AT ALL LEVELS です。

***substitutable_column_clause* の制限事項：**

- この句は、オブジェクト列の属性には指定できません。ただし、オブジェクト表自体の代替性が設定されていない場合、リレーショナル表におけるオブジェクト型の列およびオブジェクト表のオブジェクト列に対してこの句を指定できます。
- コレクション型の列の場合、この句で指定できる部分は [NOT] SUBSTITUTABLE AT ALL LEVELS のみです。

LOB_storage_clause

*LOB_storage_clause*によって、LOB データ・セグメントの記憶域属性を指定できます。

非パーティション表の場合（パーティション句なしで *physical_properties* 句に指定した場合）、この句は、LOB データ・セグメントの表の記憶域属性を指定します。

パーティション表の場合、この句は指定した位置に応じて実装されます。

- 表レベルで指定されたパーティション表の場合（パーティション句とともに *physical_properties* 句で指定した場合）、この句は、各パーティションまたはサブパーティションに対応付けられた LOB データに対するデフォルト記憶域属性を指定します。この記憶域属性は、パーティションまたはサブパーティション・レベルで *LOB_storage_clause* によって上書きされないかぎり、すべてのパーティションまたはサブパーティションに適用されます。
- パーティション表の各パーティションの場合（*table_partition_description* の一部として指定した場合）、この句は、そのパーティションのデータ・セグメントの記憶域属性、またはこのパーティションのサブパーティションのデフォルト記憶域属性を指定します。パーティション・レベルの *LOB_storage_clause* は、表レベルの *LOB_storage_clause* を上書きします。
- パーティション表の各サブパーティションの場合（*subpartition_by_hash* または *subpartition_by_list* の一部として指定した場合）、この句は、サブパーティションのデータ・セグメントの記憶域属性を指定します。サブパーティション・レベルの *LOB_storage_clause* は、パーティション・レベルおよび表レベルの *LOB_storage_clauses* を上書きします。

パーティション表の LOB 記憶域の制限事項： *table* がパーティション化されている場合、*LOB_index_clause* を指定できません。

LOB_item

表の表領域および記憶特性とは異なる表領域および記憶特性を明示的に定義する場合に、その LOB 列名または LOB オブジェクト属性を指定します。各 *LOB_item* にシステム管理された索引が自動的に作成されます。

LOB_segname

LOB データ・セグメントの名前を指定します。*LOB_item* が複数指定されている場合は、*LOB_segname* を使用できません。

LOB_parameters

LOB_parameters 句によって、様々な LOB 記憶域の要素を指定できます。

ENABLE STORAGE IN ROW: 行の記憶域を使用可能にした場合、LOB 値の長さが、約 4000 バイトからシステム制御情報分を引いた長さより小さければ、LOB 値がインラインに格納されます。これはデフォルト値です。

行の記憶域を使用可能にする場合の制限事項： `index_org_table_clause` で OVERFLOW セグメントを指定しないかぎり、索引構成表に対して、このパラメータを指定できません。

DISABLE STORAGE IN ROW: 行の記憶域を使用禁止にした場合、LOB 値の長さに関係なく、LOB 値はアウトライン（行の外側）に格納されます。

注意： LOB 値が格納されている場所にかかわらず、LOB ロケータは常にインライン（行の内側）に格納されます。STORAGE IN ROW の値は、一度設定すると、表を移動しないかぎり、変更できません。11-82 ページの「ALTER TABLE」の「[move_table_clause](#)」を参照してください。

CHUNK integer LOB の操作用に割り当てるバイト数を指定します。`integer` にデータベースのブロック・サイズの倍数を指定しなかった場合、自動的に次に大きい倍数（バイト単位）に切り上げられます。たとえば、データベースのブロック・サイズが 2048 バイトのときに `integer` に 2050 を指定すると、4096 バイト（2 ブロック）が割り当てられます。最大値は 32768（32KB）で、これが Oracle のブロック・サイズとして使用できる最も大きな値です。デフォルトの CHUNK サイズは、Oracle での 1 データベース・ブロックです。

CHUNK の値は、一度設定すると変更できません。

注意： CHUNK の値は、NEXT の値（デフォルト値または `storage_clause` で指定された値）以下である必要があります。CHUNK の値が NEXT の値を超えると、エラーが戻ります。

PCTVERSION integer LOB の記憶域全体のうち、旧バージョンの LOB を保持するために使用される最大記憶域の割合（パーセント）を指定します。デフォルト値は 10 です。これは、LOB の記憶域全体の 10% が使用されるまで以前のバージョンの LOB データが上書きされないことを意味します。

PCTVERSION パラメータを使用して、データベースを手動 UNDO モードと自動 UNDO モードのどちらで実行するかを指定できます。PCTVERSION は、手動 UNDO モードのデフォルト値です。RETENTION は、自動 UNDO モードのデフォルト値です。

PCTVERSION の制限事項： PCTVERSION と RETENTION の両方は指定できません。

RETENTION この句を使用すると、旧バージョンの LOB 列を保持するかどうかを指定できます。UNDO_RETENTION 初期化パラメータの値を使用して、時間内にデータベースに保持する UNDO データ量を判断できます。

データベースが自動 UNDO モードで稼働している場合にのみ、RETENTION パラメータを使用できます。自動 UNDO モードでは、PCTVERSION を指定しないかぎり、RETENTION がデフォルト値となります。

RETENTION の制限事項： PCTVERSION と RETENTION の両方は指定できません。

FREEPOOLS integer LOB セグメントに対する空きリストのグループ数を指定します。通常、*integer* は、Real Application Clusters 環境のインスタンス数です。シングル・インスタンス・データベースの場合、この値は 1 になります。

データベースが自動 UNDO モードで稼働している場合にのみ、このパラメータを指定できます。自動 UNDO モードでは、*storage_clause* で FREELIST GROUPS パラメータを指定しないかぎり、FREEPOOLS がデフォルト値になります。FREEPOOLS と FREELIST GROUPS のどちらも指定しない場合、データベースが自動 UNDO 管理モードで稼働している場合は FREEPOOLS 1 のデフォルト値が使用され、手動 UNDO 管理モードで稼働している場合は FREELIST GROUPS 1 のデフォルト値が使用されます。

FREEPOOLS の制限事項： *storage_clause* では、FREEPOOLS および FREELIST GROUPS パラメータを指定できません。

LOB_index_clause

この句は現在使用されていません。この句を指定しても無視されます。各 LOB の列に対する索引が自動的に生成され、LOB 索引が内部的に指定され、管理されます。

参照：

- サイズが GB になる LOB を作成する場合のガイドラインを含む LOB の詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』参照してください。
- 15-66 ページの「[LOB 列の例:](#)」を参照してください。
- 以前のリリースから移行された表での LOB 索引の管理方法の詳細は、『Oracle9i データベース移行ガイド』を参照してください。

varray_col_properties

VARRAY 型のデータが格納される LOB に対して、別々の記憶特性を指定する場合は、*varray_col_properties* を使用します。*varray_item* がマルチレベル・コレクションの場合、*varray_item* 内にネストされたすべてのコレクション項目は、常に *varray_item* と同じ LOB に格納されます。

- 非パーティション表の場合（パーティション句なしで *physical_properties* 句に指定した場合）、この句は、VARRAY の LOB データ・セグメントの記憶域属性を指定します。
- 表レベルで指定されたパーティション表の場合（パーティション句とともに *physical_properties* 句で指定した場合）、この句は、各パーティション（またはサブパーティション）に対応付けられた VARRAY の LOB データ・セグメントに対するデフォルト記憶域属性を指定します。

- パーティション表の各パーティションの場合 (*table_partition_description* の一部として指定した場合)、この句は、そのパーティションの VARRAY の LOB データ・セグメントの記憶域属性、またはこのパーティションのサブパーティションにある VARRAY の LOB データ・セグメントのデフォルト記憶域属性を指定します。パーティション・レベルの *varray_col_properties* は、表レベルの *varray_col_properties* を上書きします。
- パーティション表の各サブパーティションの場合 (*subpartition_by_hash* または *subpartition_by_list* の一部として指定した場合)、この句は、このサブパーティションの VARRAY のデータ・セグメントの記憶域属性を指定します。サブパーティション・レベルの *varray_col_properties* は、パーティション・レベルおよび表レベルの *varray_col_properties* を上書きします。

STORE AS LOB 句 STORE AS LOB を指定したときに実行される処理は、次のとおりです。

- VARRAY の最大サイズが 4000 バイト未満で、行の記憶域を使用禁止にしていない場合、VARRAY はインラインに格納されます。
- VARRAY の最大サイズが 4000 バイトを超える場合または行の記憶域を使用禁止にしている場合、VARRAY はアウトラインに格納されます。

STORE AS LOB を指定しなかった場合、VARRAY 記憶域と LOB は別々に処理されます。記憶域は、VARRAY 列の実際のサイズではなく、VARRAY の最大サイズ (要素数×要素サイズ + システム制御情報分の容量) に基づいて決定されます。

- この句を指定せず、VARRAY の最大サイズが 4000 バイト未満の場合、VARRAY は LOB としてではなくインライン RAW データとして格納されます。
- この句を指定せず、VARRAY の最大サイズが 4000 バイトを超える場合、VARRAY は常に LOB として格納されます。実際のサイズが 4000 バイト未満の場合、VARRAY はインライン LOB として格納されます。また、実際のサイズが 4000 バイトを超える場合、VARRAY はアウトライン LOB として格納されます。これはその他の LOB 列でも同じです。

VARRAY を LOB として格納する場合の制限事項： *LOB_parameters* の TABLESPACE パラメータは、この句の一部として指定できません。VARRAY に対する LOB 表領域は、デフォルトではそれを格納する表の表領域になります。

substitutable_column_clause *substitutable_column_clause* の動作は、15-35 ページの「*object_type_col_properties*」の場合と同じです。

参照： 15-63 ページの「置換可能な表および列のサンプル:」を参照してください。

nested_table_col_properties

ネストした表に対して別々の記憶特性を指定し、そのネストした表を索引構成表として定義できるようにする場合は、*nested_table_col_properties* を使用します。この記憶表は、(デフォルトの記憶特性によって) 親表と同じ表領域内に作成されます。この記憶表には、この表の作成の基になった列のネストした表の値が格納されます。

ネストした表の型を持つ列または列属性付きで表を作成する場合は、この句を挿入する必要があります。*nested_table_col_properties* 句内で、親であるオブジェクト表に対する場合と同じ働きをする句は、ここでは繰り返されません。

nested_item 型がネストした表である列 (または、その表のオブジェクト型の最上位の属性) の名前を指定します。

COLUMN_VALUE ネストした表がマルチレベル・コレクションの場合、内部のネストした表または VARRAY には名前が割り当てられていない場合があります。この場合、*nested_item* 名のかわりに COLUMN_VALUE を指定します。

参照： *nested_item* および COLUMN_VALUE の使用例は、15-65 ページの「[マルチレベル・コレクションの例](#)」を参照してください。

storage_table *nested_item* の行を含む表の名前を指定します。非パーティション表の場合、記憶表は親表と同じスキーマおよび同じ表領域内に作成されます。パーティション表の場合、記憶表はスキーマのデフォルトの表領域内に作成されます。

記憶表の制限事項：

- ネストした表の記憶表はパーティション化できません。
- *storage_table* で DML 文を直接問い合わせたり、実行することはできませんが、その記憶特性は、ALTER TABLE 文で名前を指定することによって変更できます。

参照： ネストした表の列に対する記憶特性の変更方法については、11-2 ページの「[ALTER TABLE](#)」を参照してください。

RETURN AS 問合せの結果として何を戻り値とするかを指定します。

- VALUE は、ネストした表自体のコピーを戻します。
- LOCATOR は、コレクション・ロケータをネストした表のコピーに戻します。

注意： ロケータのセッションには有効範囲が決められており、分散したセッションに使用できません。LOB ロケータとは異なり、コレクション・ロケータはコレクション・インスタンスの変更に使用できません。

segment_attributes_clause または *LOB_storage_clause* を指定しない場合、ネストした表はヒープ構成され、デフォルトの記憶特性で作成されます。

ネストした表の列のプロパティの制限事項：

- この句は、一時表には指定できません。
- *OID_clause* は指定できません。
- (*segment_attributes_clause* の一部として) *TABLESPACE* をネストした表に指定できません。表領域は常に親表の表領域です。
- 作成時、*out_of_line_ref_constraint*、*inline_ref_constraint* またはネストした表の属性に対する外部キー制約を (*object_properties* の一部として) 指定することはできません。ただし、ネストした表を修正して *ALTER TABLE* を使用する制約を追加できます。
- 記憶表に対して問合せまたは DML 文を直接実行できませんが、*ALTER TABLE* 文に記憶表の名前を指定することで、ネストした表の列の記憶特性を変更できます。

参照：

- ネストした表の列に対する記憶特性の変更方法については、11-2 ページの「[ALTER TABLE](#)」を参照してください。
- 15-65 ページの「[ネストした表の例：](#)」および 15-65 ページの「[マルチレベル・コレクションの例：](#)」を参照してください。

XMLType_column_properties

XMLType_column_properties を指定すると、*XMLType* 列に対する記憶域属性を指定できます。

XMLType_storage *XMLType* 列は、LOB 列またはオブジェクト・リレーショナル列に格納できます。

- オブジェクト・リレーショナル列に *XMLType* データを格納するには、*STORE AS OBJECT RELATIONAL* を指定します。データ・オブジェクトをリレーショナルに格納すると、リレーショナル列に索引を定義できるため、問合せのパフォーマンスが向上します。

オブジェクト・リレーショナル形式での格納を指定する場合、*XMLSchema_spec* 句も指定する必要があります。
- CLOB 列に *XMLType* データを格納する場合、*STORE AS CLOB* を指定します。CLOB 列にデータを格納すると、元の内容が保持されるため、検索時間が短縮されます。

LOB 記憶域を定義する場合、LOB パラメータと *XMLSchema_spec* 句のいずれかを指定できますが、両方は指定できません。特定のスキーマ・ベースの XML インスタンスに表や列を制限する場合は、*XMLSchema_spec* 句を指定します。

XMLSchema_spec この句を使用すると、登録した XMLSchema (XMLSCHEMA 句、または ELEMENT 句の一部として登録) の URL および XML 要素名を指定できます。XMLSchema の URL は省略可能ですが、要素は必ず指定します。XMLSchema の URL を指定する場合は、DBMS_XMLSCHEMA パッケージを使用して XMLSchema をあらかじめ登録しておく必要があります。

参照：

- *LOB_segname* および *LOB_parameters* 句の詳細は、11-45 ページの「*LOB_storage_clause*」を参照してください。
- オブジェクト・リレーショナル表における XMLType 列の例は、15-68 ページの「XMLType 列の例 :」を参照してください。XMLSchema の作成例は、D-11 ページの「SQL 文での XML の使用方法」を参照してください。
- XMLType 列と表および XMLSchema の作成の詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。

table_partitioning_clauses

table_partitioning_clauses を使用すると、パーティション表を作成できます。

一般的なパーティション化の制限事項：

- クラスタの一部である表は、パーティション化できません。
- LONG または LONG RAW 列を含む表は、パーティション化できません。

注意： ブロック・サイズが異なる表領域のパーティション化されたデータベース・エンティティの記憶域には、制限事項があります。これらの制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

参照： 15-69 ページの「パーティション化の例」を参照してください。

range_partitioning

range_partitioning 句を使用すると、列リストの値の範囲で表をパーティション化できます。索引構成表の場合、列リストは表の主キー列のサブセットである必要があります。

column

行がどのパーティションに属するかを判断するために使用される、列の順序リストを指定します（パーティション化キー）。

パーティション化キー列の制限事項： 列リスト内の列には、ROWID、LONG、LOB または TIMESTAMP WITH TIME ZONE 以外の組み込みデータ型を指定できます。

PARTITION *partition*

名前の *partition* は、2-105 ページの「スキーマ・オブジェクトのネーミング規則」で説明するとおり、スキーマ・オブジェクトとその部分のネーミング規則に従って指定する必要があります。 *partition* を指定しない場合、名前は SYS_Pn の形式で生成されます。

注意：

- 64K - 1 以内のパーティションと 64K - 1 以内のサブパーティションを指定できます。この数字未満の制限を適用する要因については、『Oracle9i データベース管理者ガイド』を参照してください。
 - パーティションが 1 つのみのパーティション表も作成できます。ただし、これは非パーティション表とは異なります。たとえば、非パーティション表にはパーティションを追加できません。
-

range_values_clause

現行パーティションの上限（境界は含まない）を指定します。値リストは、*range_partitioning* 句の列リストに対応するリテラル値を含む順序リストです。値リスト内のリテラルのかわりに、キーワード MAXVALUE を使用できます。MAXVALUE には、常に他の値（NULL を含む）より高位にソートされる最大値を指定します。

パーティション境界の上限に MAXVALUE 以外の値を指定した場合、表に暗黙の整合性制約が課せられます。

注意： 表が DATE 列でパーティション化されている場合および日付書式で年の最初の 2 桁の数字が指定されていない場合、年の「YYYY」4 文字書式マスクで TO_DATE ファンクションを使用する必要があります。（「RRRR」書式マスクは、サポートしていません。）日付書式は、NLS_TERRITORY によって暗黙的に決定され、NLS_DATE_FORMAT によって明示的に決定されます。

参照：

- パーティション・バウンドの詳細は、『Oracle9i データベース概要』を参照してください。
- これらの初期化パラメータの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。
- 15-69 ページの「レンジ・パーティション化の例 :」を参照してください。

table_partition_description

table_partition_description を使用すると、表の物理特性および記憶特性を定義できます。

segment_attributes_clause 句および *data_segment_compression* 句の機能は、表全体の *table_properties* と同じです。

key_compression 句および OVERFLOW 句の機能は、*index_org_table_clause* と同じです。

LOB_storage_clause *LOB_storage_clause* を使用すると、このパーティションの 1 つ以上の LOB 項目に対して LOB 記憶特性を指定できます。または、このパーティションに対して任意のリスト・サブパーティションを指定できます。LOB 項目に *LOB_storage_clause* を指定しない場合、各 LOB データ・パーティションに対する名前が生成されます。LOB データおよび LOB 索引パーティションに対するシステム生成名は、それぞれ SYS_LOB_Pn および SYS_IL_Pn の形式をとります。ここで、P はパーティション、n はシステム生成番号です。LOB サブパーティションに対するシステム生成名は、SYS_LOB_SUBPn および SYS_IL_SUBPn です。

varray_col_properties *varray_col_properties* を使用すると、このパーティションまたはこのパーティションの任意のリスト・サブパーティション内にある 1 つ以上の VARRAY 項目に対して記憶特性を指定できます。

***table_partition_description* の制限事項：** *partition_level_subpartition* 句は、コンポジット・パーティション表に対してのみ使用できます。15-49 ページの「*partition_level_subpartition*」を参照してください。

hash_partitioning

hash_partitioning を使用して、表がハッシュ方式でパーティション化されるように指定します。

column 行がどのパーティションに属するかを判断するために使用される、列の順序リストを指定します（パーティション化キー）。

列の値にパーティション化キーとして指定されたハッシュ・ファンクションを使用して、列をパーティションに割り当てます。次のいずれかの方法で、ハッシュ・パーティション化できます。

- **individual_hash_partitions:** この句を使用すると、個々のパーティションを名前で指定できます。パーティション名を省略すると、SYS_Pn の形式でパーティション名が割り当てられます。

個々のハッシュ・パーティションを指定する場合の制限事項：

partitioning_storage_clause でサブパーティションに対して指定できる句は、TABLESPACE 句のみです。

注意： 異なるキャラクタ・セットを使用してデータベースを使用しているか、使用する予定がある場合は、キャラクタ列を分割する際に注意してください。文字のソート順序は、すべてのキャラクタ・セットで同一ではありません。

- **hash_partitions_by_quantity:** パーティション数を指定できます。この場合、SYS_Pn の形式でパーティション名が割り当てられます。STORE IN 句は、ハッシュ・パーティションが格納されている 1 つ以上の表領域を指定します。表領域数はパーティション数と同じである必要はありません。パーティション数が表領域数より多い場合は、表領域名が繰り返し使用されます。

ハッシュ・パーティション化の両方の方法でロード・バランシングを最適化するには、偶数のパーティション数を指定します。また、ハッシュ・パーティション化の両方の方法に対して指定できる属性は、TABLESPACE のみです。ハッシュ・パーティションは、その他のすべての属性を表レベルのデフォルトから継承します。

表レベルで指定された表領域の記憶域は、パーティション・レベルで指定された表領域の記憶域で上書きされ、パーティション・レベルで指定された表領域の記憶域は、サブパーティション・レベルで指定された表領域の記憶域で上書きされます。

hash_partitions_by_quantity 句の STORE IN 句と *partitioning_storage_clause* の TABLESPACE 句の両方で表領域の記憶域を指定する場合、パーティションの位置は、表の作成時に STORE IN 句によって指定されます。TABLESPACE 句は、以降の操作のために表レベルのデフォルト表領域を指定します。

参照： ハッシュ・パーティション化については、『Oracle9i データベース概要』を参照してください。

ハッシュ・パーティション化の制限事項：

- *column_list* に指定できるのは 16 列以下です。
- 列リストには、ROWID または UROWID 疑似列を含めることはできません。

- 列リストには、ROWID、LONG または LOB 以外の組込みデータ型を指定できます。

参照： キャラクタ・セットのサポートの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

list_partitioning

list_partitioning 句を使用すると、*column* のリテラル値のリストで表をパーティション化します。リスト・パーティション化は、個々の行が固有のパーティションにマップする方法に関する制御に便利です。

パーティション名を省略すると、SYS_Pn の形式でパーティション名が割り当てられます。

list_values_clause 各パーティションの *list_values_clause* では、1 つ以上の値を割り当てる必要があります。複数のパーティションに同じ値（NULL を含む）を割り当てることはできません。リスト・パーティションは、順序付けされていません。

注意： VALUES 句のパーティション値にリテラル NULL を指定した場合、後続の間合せで、そのパーティション内のデータにアクセスするには、WHERE 句で、比較条件ではなく IS NULL 条件を使用する必要があります。

DEFAULT キーワードを指定すると、行の挿入先となるパーティションが作成されます。この行は、別のパーティションにはマップされません。このため、DEFAULT を指定できるのは 1 つのパーティションのみです。そのパーティションに対して、その他の値を指定することはできません。また、デフォルト・パーティションは、パーティションの中で最後に定義する必要があります（レンジ・パーティションで MAXVALUE を使用する場合と同様）。

各パーティションの値のリストを導出する文字列は、最大 4KB です。すべてのパーティションの総数を、64K-1 以下に指定します。

リスト・パーティション化の制限事項：

- リスト・パーティションはサブパーティション化できません。
- *column_list* でパーティション化キーを 1 つ指定する必要があります。LOB 列は、指定できません。
- パーティション化キーがオブジェクト型列の場合は、列型の 1 つの属性でもパーティション化できません。
- *list_values_clause* 内の各値は、表のパーティション全体を通して一意にする必要があります。
- パーティションおよび索引構成表をリストできません。

composite_partitioning

`composite_partitioning` 句を使用すると、まず、表がレンジ・パーティション化され、次にそれらのパーティションがハッシュ・サブパーティション化またはリスト・サブパーティション化されます。レンジ・パーティション化とハッシュ・サブパーティション化またはリスト・サブパーティション化の組合せを、**コンボジット・パーティション化**といいます。

`subpartition_by_hash` または `subpartition_by_list` 句を使用して、各コンボジット・パーティションのサブパーティション化のタイプを設定した後、各レンジ・パーティションを定義する必要があります。

- 非コンボジット・レンジ・パーティションと同じ要件を持つ `range_values_clause` を指定する必要があります。
- `table_partition_description` を使用すると、各パーティションの物理特性および記憶特性を定義できます。`table_partition_description` の中では、`partition_level_subpartition` 句を使用して個々のサブパーティションのプロパティを定義できます。
- パーティション名を省略すると、SYS_Pn の形式で名前が生成されます。
- ハッシュ・サブパーティション、リスト・サブパーティションまたは LOB サブパーティションに指定できる特性は、TABLESPACE のみです。

コンボジット・パーティション化の制限事項： コンボジット・パーティション化は、索引構成表に対して指定できません。そのため、コンボジット・パーティション表に対して、`table_partition_description` の OVERFLOW 句は無効です。

subpartition_template `subpartition_template` は、レンジ・ハッシュ・コンボジットパーティション化とレンジ・リスト・コンボジット・パーティション化の両方に共通するオプション要素です。このテンプレートを使用することで、各表のパーティションにデフォルトのサブパーティションを定義できます。明示的にサブパーティションを定義しなかったパーティションには、このデフォルト・サブパーティションが作成されます。この句は、対称型パーティションの作成時に有効です。パーティション・レベルでサブパーティションを定義すると (`partition_level_subpartition` 句で指定)、この句を上書きできます。

テンプレートを使用してサブパーティションを定義する場合、各サブパーティションの名前を指定する必要があります。

サブパーティションのテンプレートの制限事項：

- `partitioning_storage_clause` で指定できる句は、TABLESPACE のみです。
- 1 つの LOB サブパーティションに対して TABLESPACE を指定した場合、次にその LOB 列のすべての LOB サブパーティションに対して TABLESPACE を指定する必要があります。複数のサブパーティションに対して同じ表領域を指定できます。

- `subpartition_template` 内で、または個々のサブパーティションを定義するときに、`partitioning_storage_clause` を使用してリスト・サブパーティションに対して異なる LOB 記憶域を指定した場合、(LOB と VARRAY 列の両方に) `LOB_segname` を指定する必要があります。

`subpartition_by_hash`

`subpartition_by_hash` 句を使用すると、表の各パーティションをハッシュ・サブパーティション化できます。列リストのサブパーティション化はパーティション化キーには関連しませんが、同じ制限事項が適用されます (15-43 ページの「`column`」を参照)。

`subpartition_template` 句または `SUBPARTITIONS quantity` 句を使用すると、サブパーティションを定義できます。15-47 ページの `subpartition_template` を参照してください。どちらの場合も、ロード・バランシングを最適化するには、偶数のパーティション数を指定する必要があります。

SUBPARTITIONS quantity 表の各パーティションにおけるデフォルトのサブパーティション数と、格納されている 1 つ以上の表領域を任意で指定します。

デフォルト値は 1 です。この 2 つの句および `subpartition_template` を指定しない場合、後で `partition_level_subpartition` 句を指定しないかぎり、1 つのハッシュ・サブパーティションを持つパーティションが作成されます。

ハッシュ・パーティション化の制限事項： コンポジット・パーティション化に関する一般的な制限事項 (15-47 ページの「`composite_partitioning`」を参照) に加え、`subpartition_template` のハッシュ・サブパーティションには `list_values_clause` を指定できません。

`subpartition_by_list`

`subpartition_by_list` 句を使用すると、表の各パーティションを列のリテラル値に基づいてサブパーティション化できます。

`subpartition_template` を指定しない場合、`table_partition_description` の `partition_level_subpartition` 句を使用して、各パーティションごとにリスト・サブパーティションを定義できます。`subpartition_template` と `partition_level_subpartition` の両方を指定しない場合、1 つのデフォルト・サブパーティションが作成されます。

リスト・サブパーティション化の制限事項： コンポジット・パーティション化に関する一般的な制限事項 (15-47 ページの「`composite_partitioning`」を参照) に加え、次の制限事項があります。

- サブパーティション化キー列を 1 つのみ指定できます。
- 表レベルと同じ要件をもつ `list_values_clause` を指定する必要があります。

- `subpartition_template` では、`hash_subpartition_quantity` 句を指定できません。

`partition_level_subpartition`

`table_partition_description` は、コンポジット・パーティション表に対してのみ有効です。この句を使用すると、パーティションのハッシュ・サブパーティションまたはリスト・サブパーティションを指定できます。この句を使用すると、`subpartition_by_hash` 句（レンジ・ハッシュ・コンポジット・パーティションの場合）または `subpartition_template`（レンジ・ハッシュ・コンポジット・パーティションまたはレンジ・リスト・コンポジット・パーティションの場合）で設定したデフォルト値が上書きされます。

コンポジット・パーティション：

- サブパーティションの数を指定できます（または、サブパーティションが格納される 1 つ以上の表領域を指定することもできます）。この場合、`SYS_SUBPn` という形式でサブパーティション名が割り当てられます。表領域の数は、サブパーティションの数と等しくなくてもかまいません。パーティション数が表領域数より多い場合、表領域名を循環させます。
- または、`subpartition_spec` を使用して、個々のパーティション名を指定できます。また、オプションで各パーティションが格納される表領域を指定することもできます。
- `partition_level_subpartition` を省略すると、サブパーティションのテンプレートが作成されている場合は、そのテンプレートに基づいてサブパーティションが作成されます。サブパーティションのテンプレートが作成されていない場合、1 つのハッシュ・サブパーティションまたは 1 つのデフォルト・リスト・サブパーティションが作成されます。
- `partition_level_subpartition` 句全体を指定しない場合、次のようにサブパーティション名が割り当てられます。
 - サブパーティションのテンプレートを定義し、サブパーティション名を指定した場合、「`partition_name` アンダースコア (`_`) `subpartition_name`」（たとえば、`P1_SUB1`）という形式でサブパーティション名が生成されます。
 - サブパーティションのテンプレートを定義していない場合、またはサブパーティションのテンプレートは定義したが、パーティション名を指定していない場合、`SYS_SUBP n` という形式でサブパーティション名が生成されます。
- `partition_spec` で指定できる `partitioning_storage_clause` の句は、`TABSPACE` のみです。

レンジ・ハッシュ・コンポジット・パーティションの場合、`subpartition_spec` の `list_values_clause` は無効です。

レンジ・リスト・コンポジット・パーティションは、次の方法で指定できます。

- *hash_subpartition_quantity*は適切でないため、*partition_level_subpartition*の下位ブランチを使用する必要があります。
- *subpartition_spec*の中では、各サブパーティションに対して*list_values_clause*を指定する必要があります。このとき、同じパーティションのその他のサブパーティション内に存在しない値を指定する必要があります。

CACHE | NOCACHE | CACHE READS

CACHE 句を使用して、バッファ・キャッシュ内でのブロックの格納方法を指定します。次に、CACHE または NOCACHE も指定しない場合の例を示します。

- CREATE TABLE 文では、NOCACHE がデフォルトです。
- ALTER TABLE 文では、既存の値は変更されません。

CACHE 句 この句は、アクセス頻度の高いデータについて、全表スキャンの実行時に、この表のために取り出されたブロックを、バッファ・キャッシュ内の最低使用頻度（LRU）リストの最高使用頻度側に置く場合に指定します。この属性は、小規模な参照表で有効です。

LOB_storage_clause のパラメータとして CACHE を使用すると、より高速なアクセスのために、LOB データ値がバッファ・キャッシュに配置されます。

CACHE の制限事項： CACHE は、索引構成表に対して指定できません。ただし、索引構成表は暗黙的に CACHE 動作を提供します。

NOCACHE 句 この句は、アクセス頻度の低いデータについて、全表スキャンの実行時に、この表のために取り出されたブロックを、バッファ・キャッシュ内の LRU リストの最低使用頻度側に置く場合に指定します。

LOB_storage_clause のパラメータとして NOCACHE を指定すると、LOB 値がバッファ・キャッシュに入れられないか、またはバッファ・キャッシュに入れられ、LRU リストの最低使用頻度側に置かれるかのいずれかを指定できます。（デフォルトでは後者です）。

NOCACHE の制限事項： NOCACHE は、索引構成表に対して指定できません。

CACHE READS CACHE READS は LOB 記憶域にのみ適用されます。LOB 値が書込み操作中ではなく読み込み操作中にバッファ・キャッシュに入れられることを指定します。

参照： *logging_clause* を *LOB_parameters* の一部として指定した場合については、7-42 ページの「[logging_clause](#)」を参照してください。

parallel_clause

parallel_clause を使用すると、表を平行作成した後、その表に関する問合せおよび DML 操作（INSERT、UPDATE、DELETE および MERGE）に対するデフォルトの並列度を設定できます。

注意： *parallel_clause* 構文は、以前のリリースの構文に代わるものです。以前のリリースの構文は下位互換用にサポートされていますが、動作がわずかに異なることがあります。

NOPARALLEL シリアル実行を行う場合は、NOPARALLEL を指定します。これはデフォルト値です。

PARALLEL 並列度を選択する場合は、PARALLEL を指定します。並列度とは、すべての関係するインスタンスで使用可能な CPU の数に、初期化パラメータ PARALLEL_THREADS_PER_CPU の値を掛けたものです。

PARALLEL integer *integer* には、パラレル操作で使用するパラレル・スレッド数である並列度を指定します。各パラレル・スレッドは、1、2 個のパラレル実行サーバーを使用します。通常、最適な並列度が計算されるため、*integer* に値を指定する必要はありません。

***parallel_clause* の注意事項：**

- トリガーまたは参照整合性制約を定義した表に対しては、パラレル化を使用できません。
- 表にビットマップ索引を定義した場合：
 - 表がパーティション化されていない場合、後続の DML 操作はシリアル実行されます。
 - 表がパーティション化されている場合、並列度は、DML 操作でアクセスされるパーティション数に制限されます。
- 表に LOB 型またはユーザー定義オブジェクト型の列が含まれている場合、この LOB 型またはオブジェクト型の列を変更する後続の INSERT、UPDATE および DELETE 操作は、通知なしにシリアル実行されます。ただし、後続の間合せはパラレルで実行されます。
- *parallel_clause* の結果はパラレル・ヒントによって上書きされます。
- DML 文および CREATE TABLE ... AS SELECT 文はパラレルで実行されるリモート・オブジェクトを参照します。ただし、リモート・オブジェクトはリモート・データベースにある必要があります。参照は、ローカル・データベースにあるオブジェクトにループバックできません（たとえば、ローカル・データベースのオブジェクトを指定するリモート・データベースのシノニムから参照することはできません）。
- LOB 列を含む表での DML 操作はパラレル化できます。ただし、パーティション内並列性はサポートされていません。

参照： パラレル化操作の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』、『Oracle9i データベース概要』、および『Oracle9i データ・ウェアハウス・ガイド』を参照してください。また、15-64 ページの「[PARALLEL の例:](#)」を参照してください。

NOROWDEPENDENCIES | ROWDEPENDENCIES

表が**行レベル依存の追跡**を使用するかどうかを指定できます。この機能を使用すると、表の各行は、行を変更した最後のトランザクションのコミット時刻以降を示すシステム変更番号 (SCN) を持つことになります。この設定は、表の作成後に変更できません。

ROWDEPENDENCIES 行レベル依存の追跡を使用可能にする場合は、**ROWDEPENDENCIES** を指定します。この設定は、主にレプリケーション環境でパラレル伝播を許可する場合に便利です。各行につきサイズが 6 バイト増えます。

NOROWDEPENDENCIES 表を行レベル依存の追跡機能に使用しない場合は、**NOROWDEPENDENCIES** を指定します。これはデフォルト値です。

参照： レプリケーション環境での行レベル依存の追跡の使用については、『Oracle9i アドバンスド・レプリケーション』を参照してください。

MONITORING | NOMONITORING

MONITORING 表の変更統計を収集する場合は、**MONITORING** を指定します。これらの統計表は、一定の期間に DML 文の影響を受ける行数の推定値です。これらは、オブティマイザが使用またはユーザーが分析する場合に使用できます。

MONITORING の制限事項： **MONITORING** は、一時表に対して指定できません。

NOMONITORING 表の変更統計を収集しない場合は、**NOMONITORING** を指定します。これはデフォルト値です。

NOMONITORING の制限事項： **NOMONITORING** は、一時表に対して指定できません。

enable_disable_clause

enable_disable_clause によって、制約が適用されるかどうかを指定できます。デフォルトでは、制約は **ENABLE VALIDATE** 状態で作成されます。

制約を使用可能および使用禁止にする場合の制限事項：

- 整合性制約を使用可能または使用禁止にする場合、この文または前の文に制約を定義する必要があります。
- 参照された一意キー制約または主キー制約が使用可能でない場合は、外部キー制約を使用可能にできません。

参照： 制約の詳細は、7-5 ページの「[constraints](#)」を参照してください。また、15-64 ページの「[ENABLE VALIDATE の例:](#)」および 15-64 ページの「[DISABLE の例:](#)」も参照してください。

ENABLE 句 制約を表のデータに適用する場合は、ENABLE を指定します。

注意： 一意制約または主キー制約を使用可能にした場合、キーに索引が存在しないと、一意索引が作成されます。その後で制約が使用禁止になった場合、この索引は削除されます。そのため、制約が使用可能になるたびに索引が再作成されます。

索引の再作成を避け、余分な索引を削除するために、最初に使用禁止にした主キー制約および一意制約を新しく作成します。その後、一意でない索引を作成して（または、既存の一意でない索引を使用して）制約を適用してください。制約が使用禁止の場合、一意でない索引は削除されず、後続の ENABLE 操作が容易になります。

- ENABLE VALIDATE を使用すると、すべての旧データと新データが制約に従うことを指定できます。制約が使用可能で、妥当性チェック済の場合は、すべてのデータが現在有効で、今後も有効であることが保証されます。

整合性制約に違反する行が表にある場合、制約は使用禁止のままエラーを戻します。すべての行が制約に従っている場合、Oracle は制約を使用可能にします。新規データが整合性制約に違反する場合、その後の文は実行されず、整合性制約の違反を示すエラーが戻されます。

注意： 主キー制約を ENABLE VALIDATE モードに設定すると、妥当性チェック・プロセスによって主キー列に NULL が含まれないことが検証されます。これによるオーバーヘッドを回避するには、データを列に入力する前、およびこの表の主キー制約を使用可能にする前に、主キーの各列に NOT NULL というマークを付けます。

- ENABLE NOVALIDATE を使用すると、制約データに対して新しく行うすべての DML 操作が制約に従うことが保証されます。この句は、表の既存データが制約に従っていることを保証しないため、表ロックは必要ありません。

VALIDATE も NOVALIDATE も指定しない場合、VALIDATE がデフォルトになります。

ENABLE NOVALIDATE から ENABLE VALIDATE に単一制約状態を変更すると、パラレルで操作が実行できるため、読み込み、書き込みまたはその他の DDL 操作が中断されません。

ENABLE 句の制限事項： 使用禁止になっている一意キーまたは主キーを参照する外部キーを使用可能にすることはできません。

DISABLE 句 整合性制約を使用禁止にする場合は、DISABLE を指定します。データ・ディクショナリでは、使用禁止になっている整合性制約は、使用可能な制約とともに表示されず。この句を指定せずに制約を作成した場合は、その制約は自動的に使用可能になります。

- DISABLE VALIDATE は、制約を使用禁止にして制約の索引を削除しますが、制約は有効のままです。この機能を使用すると、大量のデータをロードでき、また索引用の領域が削減されるため、データ・ウェアハウスで非常に有効です。この設定を使用することで、ALTER TABLE 文の *exchange_partition_clause* または SQL*Loader を使用して、データを非パーティション表からパーティション表にロードすることもできます。他の SQL 文を使用した表に対するすべての変更（挿入、更新および削除）は禁止されます。

参照： この設定の使用方法的詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

- DISABLE NOVALIDATE は、Oracle によって制約がメンテナンスされないこと（使用禁止になっているため）、および制約が真であると保証されないこと（妥当性チェックが行われていないため）を示します。

外部キー制約が DISABLE NOVALIDATE 状態であっても、外部キーが参照している主キーを持つ表を削除できません。また、オプティマイザは、DISABLE NOVALIDATE 状態でも制約を使用できます。

参照： この設定の使用方法的詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

VALIDATE も NOVALIDATE も指定しない場合、NOVALIDATE がデフォルトになります。

一意索引を使用している一意制約または主キー制約を使用禁止にすると、一意索引は削除されます。

UNIQUE UNIQUE 句によって、指定された列または列の組合せに定義された一意制約を使用可能または使用禁止にできます。

PRIMARY KEY PRIMARY KEY 句によって、表の主キー制約を使用可能または使用禁止にできます。

CONSTRAINT CONSTRAINT 句によって、*constraint* に指定する整合性制約を使用可能または使用禁止にできます。

KEEP | DROP INDEX この句を使用すると、一意キー制約または主キー制約を適用するために使用される索引を保持するかまたは削除するかを指定できます。

索引の保持と削除の制限事項： 一意キー制約または主キー制約が使用禁止の場合にのみ、この句を指定できます。

using_index_clause *using_index_clause* を使用すると、一意キー制約または主キー制約の適用で使用する索引を指定または作成することができます。

using_index_clause は、一意制約または主キー制約を使用可能にしている場合にのみ指定できます。*using_index_clause* の句はどのような順序でも指定できますが、各句を指定できるのは1回のみです。

- *schema.index* を指定すると、指定した索引を使用して制約が適用されます。索引がない、または制約の適用に索引が使用できない場合、エラーが戻されます。
- *create_index_statement* を指定すると、索引が作成され、制約の適用に使用されます。索引が作成できない、または制約の適用に索引が使用できない場合、エラーが戻されます。
- 既存の索引の指定も、新しい索引の作成も行わない場合は、索引が自動で作成されます。この場合、次のようになります。
 - 索引には制約と同じ名前が割り当てられます。
 - 索引には、INITRANS、MAXTRANS、TABLESPACE、PCTFREE および STORAGE の各パラメータ値を選択できます。PCTUSED または *logging_clause* を索引に指定することはできません。
 - 表がパーティション化されている場合、一意制約または主キー制約にローカル・パーティション索引またはグローバル・パーティション索引を指定できます。

using_index_clause の制限事項：

- この句は、ビュー制約には指定できません。
- この句は、NOT NULL、外部キーまたは CHECK 制約には指定できません。
- 索引構成表の主キーが使用可能な場合は、索引の指定 (*schema.index*) および作成 (*create_index_statement*) はできません。

参照：

- この句への索引属性の指定については、13-73 ページの「[index_attributes](#)」を参照してください。
- [using_index_clause](#)、主キー制約および一意制約の詳細は、7-5 ページの「[constraints](#)」を参照してください。
- LOCAL と [global_index_clause](#) の詳細および索引に関する NOSORT と [logging_clause](#) の詳細は、13-61 ページの「[CREATE INDEX](#)」を参照してください。
- INITRANS、MAXTRANS、TABLESPACE、STORAGE および PCTFREE パラメータの詳細は、15-27 ページの「[segment_attributes_clause](#)」を参照してください。
- 制約の適用で使用する索引の作成方法の例は、7-32 ページの「[明示的な索引の制御例](#)」を参照してください。

global_partitioned_index *global_partitioned_index* によって、索引のパーティション化がユーザー定義であり、基礎となる表と同一レベルでパーティション化されないことを指定できます。デフォルトでは、非パーティション索引はグローバル索引です。グローバル索引は、*column_list* に指定された表の列の値の範囲に基づいてパーティション化されます。ローカル索引にこの句は指定できません。

column_list では、索引の列リストの左の接頭辞を指定する必要があります。つまり、索引が列 a、b および c に定義されている場合は、*column_list* に (a,b,c)、(a,b) または (a,c) は指定できますが、(b,c)、(c) または (b,a) は指定できません。

グローバル・パーティション索引キーの制限事項：

- *column_list* に指定できるのは 32 列以下です。
- これらの列には、ROWID 類似列または ROWID 型の列は指定できません。

注意： 異なるキャラクタ・セットを使用してデータベースを使用しているか、使用する予定がある場合は、キャラクタ列を分割する際に注意してください。文字のソート順序は、すべてのキャラクタ・セットで同一ではありません。

参照： キャラクタ・セットのサポートについては、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

index_partitioning_clause この句は、個々の索引パーティションを示すために使用します。この句が繰り返される数によってパーティションの数が決まります。*partition* を指定しない場合、名前は SYS_Pn の形式で生成されます。

VALUES LESS THAN (*value_list*) には、グローバル索引の現在のパーティションの上限（境界は含まない）を指定します。値のリストは、*global_partitioned_index* 句の列リストに対応するリテラル値を含む、カンマで区切られた順序リストです。最後のパーティションの値としては、必ず MAXVALUE を指定します。

注意： 索引が DATE 列でパーティション化されている場合、および日付書式で年の最初の 2 桁の数字が指定されていない場合、年の 4 文字書式マスクを指定して TO_DATE ファンクションを使用する必要があります。日付書式は、NLS_TERRITORY によって暗黙的に決定されるか、NLS_DATE_FORMAT によって明示的に決定されます。

CASCADE CASCADE を指定して、指定した整合性制約に依存する整合性制約を使用禁止にします。参照整合性制約を構成する主キーまたは一意キーを使用禁止にする場合、この句を指定します。

CASCADE の制限事項： DISABLE を指定した場合のみ、CASCADE を指定できます。

row_movement_clause

row_movement_clause を使用して、表の列を移動できます。データ・セグメントの圧縮時や、パーティション・データの更新操作時などに、列を移動できます。

注意： データ・アクセスで静的な ROWID が必要な場合は、行移動を有効にしないでください。通常の表（ヒープ構成表）の場合は、行を移動すると、その行の ROWID が変更されます。索引構成表での行の移動の場合は、論理 ROWID の物理推測コンポーネントは不正確になりますが、論理 ROWID は有効のままです。

- 行を移動するには、ENABLE を指定します。このとき、ROWID は変更されます。
- 行を移動しない場合は、DISABLE を指定します。このとき、ROWID は変更されません。

行の移動の制限事項： この句は、パーティション化された索引構成表に対して指定できません。

この句を省略すると、列移動は使用禁止になります。

AS subquery

副問合せを指定して表の内容を定義します。表の作成時に、副問合せの結果として戻された行を表の中に挿入します。

オブジェクト表の場合、*subquery* には表の型に対応する 1 つの式、または表の型の最上位属性の数のどちらかを設定できます。

参照： 18-4 ページの「[SELECT](#)」を参照してください。

subquery が、既存のマテリアライズド・ビューと（部分的または完全に）同じビューを戻す場合、Oracle は、*subquery* に指定された 1 つ以上の表のかわりにマテリアライズド・ビュー（クエリー・リライト用）を使用することがあります。

参照： マテリアライズド・ビューおよびクエリー・リライトの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

データ型およびデータ長は、副問合せの結果から導出されます。整合性制約、その他の列および表の属性には次の規則が適用されます。

- 副問合せで、列を含む式ではなく列を選択した場合、選択した表の列に NOT NULL 制約が明示的に作成されていると、新しい表の対応する列にも NOT NULL 制約が自動的に定義されます。ただし、選択した表の列（たとえば、主キー用）に Oracle によって暗黙的に作成された NOT NULL 制約は、新しい表には引き継がれません。制約に違反する行がある場合、表は作成されずエラーが戻されます。
- 選択した表の列に暗黙的に作成された NOT NULL 制約（たとえば、主キー）は、新しい表には引き継がれません。
- 主キー、一意キー、外部キー、CHECK 制約、パーティション化条件、索引および列のデフォルト値は、新しい表に引き継がれません。

subquery 内のすべての式が、式ではなく列の場合、表定義から列を完全に省略できます。この場合、表の列名は *subquery* の列の名前と同じになります。

TO_LOB ファンクションと組み合わせて *subquery* を使用すると、別の表の LONG 列の値を、作成する表の列の LOB 値に変換できます。

参照：

- LONG を LOB にコピーする理由およびタイミングについては、『Oracle9i データベース移行ガイド』を参照してください。
- TO_LOB ファンクションの使用方法については、6-6 ページの「[変換ファンクション](#)」を参照してください。
- *order_by_clause* の詳細は、18-4 ページの「[SELECT](#)」を参照してください。

parallel_clause この文で *parallel_clause* を指定した場合、INITIAL 記憶域パラメータに対して指定する値は無視され、かわりに NEXT パラメータの値が使用されます。

参照： これらのパラメータの詳細は、7-51 ページの「*storage_clause*」を参照してください。

ORDER BY ORDER BY 句は、副問合せによって戻される行を順序付けます。

注意： この句を CREATE TABLE で指定した場合、この句が表全体にわたるデータを順序付けるとはかぎりません（たとえば、パーティション間での順序付けは行われません）。同じキーの索引を ORDER BY キー列として作成する場合に、この句を指定します。Oracle は、ORDER BY キーのデータをクラスタ化し、索引キーに対応させます。

表の副問合せの定義の制限事項：

- 表中の列数は、副問合せ中の式の数と同じである必要があります。
- 列定義では、列名、デフォルト値および整合性制約のみを指定できます。データ型は指定できません。
- AS *subquery* を含む CREATE TABLE 文には、外部キー制約を定義できません。そのかわりに、制約を指定せずに表を作成し、後で ALTER TABLE 文を使用してその制約を追加できます。

object_table

OF 句によって、明示的に *object_type* 型の**オブジェクト表**を作成できます。オブジェクト表の各列は、*object_type* 型の最上位の属性に対応します。各行には、オブジェクト・インスタンスが入り、また各インスタンスには、行の挿入時に一意のシステム生成オブジェクト識別子（OID）が割り当てられます。*schema* を省略した場合、オブジェクト表が自分のスキーマ内に作成されます。

オブジェクト表、XMLType 表、オブジェクト・ビューおよび XMLType ビューには、列名が付けられません。そのため、システム生成列 SYS_NC_ROWINFO\$ が定義されます。問合せでこの列名を使用し、WITH OBJECT IDENTIFIER 句を指定して、オブジェクト・ビューを作成できます。

参照： 15-73 ページの「**オブジェクト列と表の例**」を参照してください。

object_table_substitution

object_table_substitution 句を使用すると、サブタイプに対応する行オブジェクトの、このオブジェクト表への挿入を許可するかどうかを指定できます。

NOT SUBSTITUTABLE AT ALL LEVELS NOT SUBSTITUTABLE AT ALL LEVELS を指定すると、作成するオブジェクト表は置換できなくなります。また、置換は、すべての埋込みオブジェクト属性および埋込みのネストした表と配列の要素には使用禁止です。デフォルトは、SUBSTITUTABLE AT ALL LEVELS です。

object_properties

オブジェクト表のプロパティは、基本的にリレーショナル表と同じです。ただし、列を指定するかわりに、オブジェクトの属性を指定します。

attribute には、オブジェクト内の項目の修飾した列名を指定します。

OID_clause

OID_clause によって、オブジェクト表のオブジェクト識別子（OID）がシステム生成されるか、表の主キーを基に作成されるかを指定できます。デフォルトは SYSTEM GENERATED です。

***OID_clause* の制限事項：**

- 主キー制約を表に指定していないと、OBJECT IDENTIFIER IS PRIMARY KEY は指定できません。
- この句は、ネストした表には指定できません。

注意： 主キー OID はローカルで（グローバルである必要はありません）一意です。グローバルで一意の識別子が必要な場合は、主キーがグローバルで一意であることを確認してください。

OID_index_clause

この句は、*OID_clause* を SYSTEM GENERATED として指定している場合のみに適用されます。非表示のオブジェクト識別子列に索引を指定します。また、任意に記憶特性を指定します。

index には、非表示のシステム生成オブジェクト識別子列の索引の名前を指定します。*index* を省略すると、名前が生成されます。

physical_properties* および *table_properties

これらの句のセマンティクスは、この章の前述の項に示されています。15-27 ページの「[physical_properties](#)」および 15-35 ページの「[table_properties](#)」を参照してください。

XMLType_table

XMLType_table 構文を使用すると、XMLType データ型の表を作成できます。

オブジェクト表、XMLType 表、オブジェクト・ビューおよび XMLType ビューには、列名が付けられません。そのため、システム生成列 SYS_NC_ROWINFO\$ が定義されます。問合せでこの列名を使用し、WITH OBJECT IDENTIFIER 句を指定して、オブジェクト・ビューを作成できます。

XMLType_storage

この句を使用すると、基礎となる列の記憶域の管理方法を指定できます。

OBJECT RELATIONAL オブジェクト・リレーショナル列に XMLType データを格納するには、OBJECT RELATIONAL を指定します。OBJECT RELATIONAL を指定する場合、XMLSchema_storage_clause でも XMLSchema を指定する必要があります。また、(DBMS_XMLSCHEMA パッケージを使用して) あらかじめこのスキーマを登録しておく必要があります。登録したスキーマに基づいて表が作成されます。

CLOB CLOB 列に XMLType データを格納するには、CLOB を指定します。CLOB を指定する場合、LOB セグメント名または LOB_parameters 句 (あるいはその両方) を指定する必要があります。

XMLSchema_spec

この句を使用すると、登録した XMLSchema (XMLSCHEMA 句、または ELEMENT 句の一部として登録) の URL および XML 要素名を指定できます。XMLSchema の URL は省略可能ですが、要素は必ず指定します。XMLSchema の URL を指定する場合は、DBMS_XMLSCHEMA パッケージを使用して XMLSchema をあらかじめ登録しておく必要があります。

参照：

- DBMS_XMLSCHEMA パッケージの詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- XML の作成方法および使用方法の詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。
- 15-67 ページの「XMLType 表の例:」を参照してください。

例

一般的な例

次の文は、人事情報のサンプル・スキーマ (hr) が所有する employees 表を定義します。

```
CREATE TABLE employees
( employee_id    NUMBER(6)
, first_name     VARCHAR2(20)
, last_name      VARCHAR2(25)
    CONSTRAINT emp_last_name_nn NOT NULL
, email          VARCHAR2(25)
    CONSTRAINT emp_email_nn     NOT NULL
, phone_number   VARCHAR2(20)
, hire_date      DATE DEFAULT SYSDATE
    CONSTRAINT emp_hire_date_nn NOT NULL
, job_id         VARCHAR2(10)
    CONSTRAINT emp_job_nn      NOT NULL
, salary         NUMBER(8,2)
    CONSTRAINT emp_salary_nn  NOT NULL
, commission_pct NUMBER(2,2)
, manager_id     NUMBER(6)
, department_id  NUMBER(4)
, dn            VARCHAR2(300)
, CONSTRAINT emp_salary_min
    CHECK (salary > 0)
, CONSTRAINT emp_email_uk
    UNIQUE (email)
) ;
```

この表は 12 列で構成されます。employee_id 列は、NUMBER データ型です。hire_date 列は、データ型が DATE、デフォルト値が SYSDATE です。last_name 列は、VARCHAR2 型で、NOT NULL 制約などがあります。

記憶域の例： 次の文は、記憶域の容量が小さく、割当てに制限のある example 表領域の中にサンプル表 employees_demo を定義します。

```
CREATE TABLE employees_demo
( employee_id    NUMBER(6)
, first_name     VARCHAR2(20)
, last_name      VARCHAR2(25)
    CONSTRAINT emp_last_name_nn NOT NULL
, email          VARCHAR2(25)
    CONSTRAINT emp_email_nn     NOT NULL
, phone_number   VARCHAR2(20)
```

```

, hire_date      DATE  DEFAULT SYSDATE
  CONSTRAINT emp_hire_date_nn  NOT NULL
, job_id         VARCHAR2(10)
  CONSTRAINT     emp_job_nn   NOT NULL
, salary         NUMBER(8,2)
  CONSTRAINT     emp_salary_nn NOT NULL
, commission_pct NUMBER(2,2)
, manager_id     NUMBER(6)
, department_id  NUMBER(4)
, dn             VARCHAR2(300)
, CONSTRAINT     emp_salary_min
                CHECK (salary > 0)
, CONSTRAINT     emp_email_uk
                UNIQUE (email)
)
TABLESPACE example
STORAGE (INITIAL      6144
        NEXT         6144
        MINEXTENTS    1
        MAXEXTENTS    5 );

```

一時表の例： 次の文は、販売員が使用するサンプル・データベースの一時表 `today_sales` を作成します。各販売員のセッションは、自身のその日の販売データを表に格納します。一時的なデータは、セッションの終わりに削除されます。

```

CREATE GLOBAL TEMPORARY TABLE today_sales
ON COMMIT PRESERVE ROWS
AS SELECT * FROM orders WHERE order_date = SYSDATE;

```

置換可能な表および列のサンプル： 次の文は、`person_t` 型（16-22 ページの「[型の階層例](#)」で作成）から置換可能な表を作成します。

```

CREATE TABLE persons OF person_t;

```

次の文は、`person_t` 型の置換可能な列で表を作成します。

```

CREATE TABLE books (title VARCHAR2(100), author person_t);

```

`persons` または `books` に挿入するときに、`person_t` またはそのサブタイプの属性に対する値を指定できます。例では、17-65 ページの「[置換可能な表および列への挿入例](#)」で示した文を挿入します。

組込みファンクションおよび条件を使用して、このような表からデータを抽出することができます。その例は、6-181 ページの「[TREAT](#)」、6-155 ページの「[SYS_TYPEID](#)」および 5-19 ページの「[IS OF type 条件](#)」を参照してください。

PARALLEL の例： 次の文は、最適な数のパラレル実行サーバーを使用する表を作成して、employees をスキャンし、dept_80 に移入します。

```
CREATE TABLE dept_80
  PARALLEL
  AS SELECT * FROM employees
  WHERE department_id = 80;
```

パラレル化を使用することによって、表を作成するためにパラレル実行サーバーが使用されるため、表作成が高速化されます。表が作成された後、表へのアクセスに表作成と同じ並列度が使用されるため、表の間合せも高速化します。

NOPARALLEL の例： 次の文は、同じ表をシリアルで作成します。後続の DML および表の間合せもシリアルで実行されます。

```
CREATE TABLE dept_80
  AS SELECT * FROM employees
  WHERE department_id = 80;
```

ENABLE VALIDATE の例： 次の文は、サンプル表 departments_demo を作成します。この例では、NOT NULL 制約が定義され、ENABLE VALIDATE 状態に置かれます。

```
CREATE TABLE departments_demo
  ( department_id    NUMBER(4)
  , department_name  VARCHAR2(30)
    CONSTRAINT dept_name_nn NOT NULL
  , manager_id      NUMBER(6)
  , location_id     NUMBER(4)
  , dn              VARCHAR2(300)
  ) ;
```

DISABLE の例： 次の文は、同じ departments_demo 表を作成しますが、使用禁止の主キー制約も定義します。

```
CREATE TABLE departments_demo
  ( department_id    NUMBER(4) PRIMARY KEY DISABLE
  , department_name  VARCHAR2(30)
    CONSTRAINT dept_name_nn NOT NULL
  , manager_id      NUMBER(6)
  , location_id     NUMBER(4)
  , dn              VARCHAR2(300)
  ) ;
```


ネストした表の例： 次の文は、ネストした表の列 `ad_textdocs_ntab` を使用してサンプル表 `pm.print_media` を作成します。

```
CREATE TABLE print_media
( product_id      NUMBER(6)
, ad_id           NUMBER(6)
, ad_composite    BLOB
, ad_sourcetext   CLOB
, ad_finaltext    CLOB
, ad_fltextn      NCLOB
, ad_textdocs_ntab textdoc_tab
, ad_photo        BLOB
, ad_graphic      BFILE
, ad_header       adheader_typ
, press_release   LONG
) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;
```

マルチレベル・コレクションの例： 次の例では、アカウント・マネージャが2つのレベルのネストした表を使用して顧客の表を作成します。

```
CREATE TYPE phone AS OBJECT (telephone NUMBER);
/
CREATE TYPE phone_list AS TABLE OF phone;
/
CREATE TYPE my_customer AS OBJECT (
    cust_name VARCHAR2(25),
    phones phone_list);
/
CREATE TYPE customer_list AS TABLE OF my_customer;
/
CREATE TABLE business_contacts (
    company_name VARCHAR2(25),
    company_reps customer_list)
NESTED TABLE company_reps STORE AS outer_ntab
(NESTED TABLE phones STORE AS inner_ntab);
```

前述の例を以下のように使用した場合、内部のネストした表に列または属性名がない場合に `COLUMN_VALUE` キーワードが使用されます。

```
CREATE TYPE phone AS TABLE OF NUMBER;
/
CREATE TYPE phone_list AS TABLE OF phone;
/
CREATE TABLE my_customers (
    name VARCHAR2(25),
    phone_numbers phone_list)
NESTED TABLE phone_numbers STORE AS outer_ntab
(NESTED TABLE COLUMN_VALUE STORE AS inner_ntab);
```

LOB 列の例： 次の文は、LOB 記憶特性を追加して pm.print_media を作成します。

```
CREATE TABLE print_media
( product_id      NUMBER(6)
, ad_id           NUMBER(6)
, ad_composite    BLOB
, ad_sourcetext   CLOB
, ad_finaltext    CLOB
, ad_fltextn      NCLOB
, ad_textdocs_ntab textdoc_tab
, ad_photo        BLOB
, ad_graphic      BFILE
, ad_header       adheader_typ
, press_release   LONG
) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab_new
LOB (ad_sourcetext, ad_finaltext) STORE AS
(TABLESPACE example
  STORAGE (INITIAL 6144 NEXT 6144)
  CHUNK 4000
  NOCACHE LOGGING);
```

この例では、CHUNK の値を 4096（2048 のブロック・サイズの近似倍数）まで切り上げます。

索引構成表の例： 次の文は、サンプル索引構成表 hr.countries を作成します。

```
CREATE TABLE countries
( country_id      CHAR(2)
  CONSTRAINT country_id_nn NOT NULL
, country_name    VARCHAR2(40)
, currency_name   VARCHAR2(25)
, currency_symbol VARCHAR2(3)
, region          VARCHAR2(15)
, CONSTRAINT      country_c_id_pk
                  PRIMARY KEY (country_id )
)
ORGANIZATION INDEX
INCLUDING country_name
PCTTHRESHOLD 2
STORAGE
( INITIAL 4K
  NEXT 2K
  PCTINCREASE 0
  MINEXTENTS 1
  MAXEXTENTS 1 )
OVERFLOW
STORAGE
( INITIAL 4K
```

```

NEXT 2K
PCTINCREASE 0
MINEXTENTS 1
MAXEXTENTS 1 );

```

外部表の例： 次の文は、サンプル表 `hr.departments` のサブセットを示す外部表を作成します。ORACLE_LOADER アクセス・ドライバの詳細および *opaque_format_spec* への値の指定方法については、『Oracle9i データベース・ユーティリティ』を参照してください。

```

CREATE TABLE dept_external (
  deptno      NUMBER(6),
  dname       VARCHAR2(20),
  loc         VARCHAR2(25)
)
ORGANIZATION EXTERNAL
(TYPE oracle_loader
DEFAULT DIRECTORY admin
ACCESS PARAMETERS
(
  RECORDS DELIMITED BY newline
  BADFILE 'ulcase1.bad'
  DISCARDFILE 'ulcase1.dis'
  LOGFILE 'ulcase1.log'
  SKIP 20
  FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
  (
    deptno      INTEGER EXTERNAL(6),
    dname       CHAR(20),
    loc         CHAR(25)
  )
)
LOCATION ('ulcase1ctl')
)
REJECT LIMIT UNLIMITED;

```

参照： admin ディレクトリの作成方法は、13-48 ページの「[ディレクトリの作成例](#)」を参照してください。

XMLType の例

この項では、XMLType 表または XMLType 列の作成例について説明します。この例の詳細は、D-11 ページの「[SQL 文での XML の使用方法](#)」を参照してください。

XMLType 表の例： 次の文は、暗黙的に CLOB 列を持つ非常に単純な XMLType 表を作成します。

```
CREATE TABLE xwarehouses OF XMLTYPE;
```

データは暗黙的に CLOB 列に格納されるため、LOB 列に対するすべての制限事項が適用されます。これらの制限事項を回避するには、次の例で示すとおり XMLSchema ベースの表を作成します。XMLSchema は事前に作成しておく必要があります（詳細は、D-11 ページの「SQL 文での XML の使用方法」を参照してください）。

```
CREATE TABLE xwarehouses OF XMLTYPE
XMLSCHEMA "http://www.oracle.com/xwarehouses.xsd"
ELEMENT "Warehouse";
```

XMLSchema ベースの表に制約を定義したり、索引を作成できるため、後続の間合せのパフォーマンスが大幅に向上します。XMLType 表にオブジェクト・リレーショナル・ビューを作成し、オブジェクト・リレーショナル表に XMLType ビューを作成できます。

参照：

- 制約の追加例は、D-11 ページの「SQL 文での XML の使用方法」を参照してください。
- 索引の作成例は、13-82 ページの「XMLType 表の索引の作成例：」を参照してください。
- XMLType ビューの作成例は、16-52 ページの「XMLType ビューの作成例：」を参照してください。

XMLType 列の例： 次の文は、XMLType 列を持つ表を作成し、CLOB として格納します。この表では XMLSchema が必要ないため、コンテンツ構造は事前に定義しません。

```
CREATE TABLE xwarehouses (
  warehouse_id      NUMBER,
  warehouse_spec     XMLTYPE)
XMLTYPE warehouse_spec STORE AS CLOB
(TABLESPACE example
 STORAGE (INITIAL 6144 NEXT 6144)
 CHUNK 4000
 NOCACHE LOGGING);
```

次の例は、同様の表を作成しますが、オブジェクト・リレーショナル XMLType 列に XMLType データを格納します。この列の構造は、指定したスキーマによって判断されます。

```
CREATE TABLE xwarehouses (
  warehouse_id      NUMBER,
  warehouse_spec     XMLTYPE)
XMLTYPE warehouse_spec STORE AS OBJECT RELATIONAL
XMLSCHEMA "http://www.oracle.com/xwarehouses.xsd"
ELEMENT "Warehouse";
```

パーティション化の例

レンジ・パーティション化の例： サンプル・スキーマ sh の sales 表は、レンジでパーティション化されています。次の例は、range_sales 表を簡略化して作成します（この例では、制約および記憶域要素は省略されています）。

```
CREATE TABLE range_sales
( prod_id          NUMBER(6)
, cust_id          NUMBER
, time_id          DATE
, channel_id       CHAR(1)
, promo_id         NUMBER(6)
, quantity_sold    NUMBER(3)
, amount_sold      NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
(PARTITION SALES_Q1_1998 VALUES LESS THAN (TO_DATE('01-APR-1998','DD-MON-YYYY')),
PARTITION SALES_Q2_1998 VALUES LESS THAN (TO_DATE('01-JUL-1998','DD-MON-YYYY')),
PARTITION SALES_Q3_1998 VALUES LESS THAN (TO_DATE('01-OCT-1998','DD-MON-YYYY')),
PARTITION SALES_Q4_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999','DD-MON-YYYY')),
PARTITION SALES_Q1_1999 VALUES LESS THAN (TO_DATE('01-APR-1999','DD-MON-YYYY')),
PARTITION SALES_Q2_1999 VALUES LESS THAN (TO_DATE('01-JUL-1999','DD-MON-YYYY')),
PARTITION SALES_Q3_1999 VALUES LESS THAN (TO_DATE('01-OCT-1999','DD-MON-YYYY')),
PARTITION SALES_Q4_1999 VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY')),
PARTITION SALES_Q1_2000 VALUES LESS THAN (TO_DATE('01-APR-2000','DD-MON-YYYY')),
PARTITION SALES_Q2_2000 VALUES LESS THAN (TO_DATE('01-JUL-2000','DD-MON-YYYY')),
PARTITION SALES_Q3_2000 VALUES LESS THAN (TO_DATE('01-OCT-2000','DD-MON-YYYY')),
PARTITION SALES_Q4_2000 VALUES LESS THAN (MAXVALUE))
;
```

パーティション表のメンテナンス操作については、『Oracle9i データベース管理者ガイド』を参照してください。

リスト・パーティション化の例： 次の文は、サンプル表 oe.customers をリスト・パーティション表として作成します（この例では、サンプル表の一部の列およびすべての制約は省略されています）。

```
CREATE TABLE list_customers
( customer_id      NUMBER(6)
, cust_first_name  VARCHAR2(20)
, cust_last_name   VARCHAR2(20)
, cust_address     CUST_ADDRESS_TYP
, nls_territory    VARCHAR2(30)
, cust_email       VARCHAR2(30))
PARTITION BY LIST (nls_territory) (
PARTITION asia VALUES ('CHINA', 'THAILAND'),
PARTITION europe VALUES ('GERMANY', 'ITALY', 'SWITZERLAND'),
```

```
PARTITION west VALUES ('AMERICA'),
PARTITION east VALUES ('INDIA'),
PARTITION rest VALUES (DEFAULT);
```

LOB 列のあるパーティション表の例： 次の文は、2 つのパーティション p1 と p2、および 3 つの LOB 列 b、c、d を持つパーティション表 part_tab を作成します。この文では、サンプル表 pm.print_media を使用しますが、LONG 列はパーティション化でサポートされないため、LONG 列の press_release は省略します。

```
CREATE TABLE print_media_demo
( product_id NUMBER(6)
, ad_id NUMBER(6)
, ad_composite BLOB
, ad_sourcetext CLOB
, ad_finaltext CLOB
, ad_fltextn NCLOB
, ad_textdocs_ntab textdoc_tab
, ad_photo BLOB
, ad_graphic BFILE
, ad_header adheader_typ
) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab_demo
LOB (ad_composite, ad_photo, ad_finaltext)
STORE AS (STORAGE (NEXT 20M))
PARTITION BY RANGE (product_id)
(PARTITION p1 VALUES LESS THAN (3000) TABLESPACE tbs_1
LOB (ad_composite, ad_photo)
STORE AS (TABLESPACE tbs_2 STORAGE (INITIAL 10M)),
PARTITION P2 VALUES LESS THAN (MAXVALUE)
LOB (ad_composite, ad_finaltext)
STORE AS (TABLESPACE tbs_3)
)
TABLESPACE tbs_4;
```

パーティション p1 は、表領域 tbs_1 にあります。ad_composite および ad_finaltext に対する LOB データのパーティションは、表領域 tbs_2 にあります。ad_photo に対する LOB データのパーティションは、表領域 tbs_1 にあります。LOB 列 ad_composite および ad_finaltext に、記憶域属性 INITIAL を指定します。他の属性は、デフォルトの表レベルの仕様から継承されます。表レベルで指定されていないデフォルトの LOB 記憶域属性は、ad_composite 列および ad_finaltext 列の場合は表領域 tbs_2 から継承されます。ad_photo 列の場合は、表領域 tbs_1 から継承されます。LOB 索引パーティションは、対応する LOB データ・パーティションと同じ表領域に存在します。他の記憶域属性は、LOB データ・パーティションの対応する属性値および索引パーティションがある表領域のデフォルト属性に基づきます。

パーティション P2 は、デフォルトの表領域 tbs_4 内にあります。ad_composite および ad_photo に対する LOB データは、表領域 tbs_3 内にあります。ad_finaltext に対する LOB データは、表領域 tbs_4 内にあります。ad_composite 列および ad_photo 列に対する LOB 索引は、表領域 tbs_3 内にあります。ad_finaltext 列に対する LOB 索引は、表領域 tbs_4 内にあります。

ハッシュ・パーティション化の例： サンプル表 oe.hash_product は、パーティション化されていません。次の例は、パフォーマンス上の理由で、このような大規模な表をハッシュでパーティション化します。（この例では、表領域は仮想の名前です。）

```
CREATE TABLE hash_products
( product_id          NUMBER(6)
, product_name        VARCHAR2(50)
, product_description VARCHAR2(2000)
, category_id         NUMBER(2)
, weight_class        NUMBER(1)
, warranty_period     INTERVAL YEAR TO MONTH
, supplier_id         NUMBER(6)
, product_status      VARCHAR2(20)
, list_price          NUMBER(8,2)
, min_price           NUMBER(8,2)
, catalog_url         VARCHAR2(50)
, CONSTRAINT         product_status_lov
                     CHECK (product_status in ('orderable'
                                                , 'planned'
                                                , 'under development'
                                                , 'obsolete'))
)
PARTITION BY HASH (product_id)
PARTITIONS 5
STORE IN (tbs_1, tbs_2, tbs_3, tbs_4);
```

コンポジット・パーティション表の例： 次の例は、表（15-69 ページの「レンジ・パーティション化の例：」で作成）のデータを販売時刻で分割します。時刻と販売チャネルに従って最近のデータにアクセスする場合、コンポジット・パーティション化は、より適切です。次の例では、同じ range_sales 表のコピーを作成しますが、レンジ・ハッシュ・コンポジット・パーティション化します。最新のデータがあるパーティションは、Oracle 定義とユーザー定義の両方のサブパーティション名でサブパーティション化されます。（この例では、制約および記憶域属性は省略してあります。）

```
CREATE TABLE composite_sales
( prod_id          NUMBER(6)
, cust_id          NUMBER
, time_id          DATE
, channel_id       CHAR(1)
, promo_id         NUMBER(6)
```

```
, quantity_sold NUMBER(3)
, amount_sold     NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
SUBPARTITION BY HASH (channel_id)
(PARTITION SALES_Q1_1998 VALUES LESS THAN (TO_DATE('01-APR-1998','DD-MON-YYYY')),
PARTITION SALES_Q2_1998 VALUES LESS THAN (TO_DATE('01-JUL-1998','DD-MON-YYYY')),
PARTITION SALES_Q3_1998 VALUES LESS THAN (TO_DATE('01-OCT-1998','DD-MON-YYYY')),
PARTITION SALES_Q4_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999','DD-MON-YYYY')),
PARTITION SALES_Q1_1999 VALUES LESS THAN (TO_DATE('01-APR-1999','DD-MON-YYYY')),
PARTITION SALES_Q2_1999 VALUES LESS THAN (TO_DATE('01-JUL-1999','DD-MON-YYYY')),
PARTITION SALES_Q3_1999 VALUES LESS THAN (TO_DATE('01-OCT-1999','DD-MON-YYYY')),
PARTITION SALES_Q4_1999 VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY')),
PARTITION SALES_Q1_2000 VALUES LESS THAN (TO_DATE('01-APR-2000','DD-MON-YYYY')),
PARTITION SALES_Q2_2000 VALUES LESS THAN (TO_DATE('01-JUL-2000','DD-MON-YYYY'))
SUBPARTITIONS 8,
PARTITION SALES_Q3_2000 VALUES LESS THAN (TO_DATE('01-OCT-2000','DD-MON-YYYY'))
(SUBPARTITION ch_c,
SUBPARTITION ch_i,
SUBPARTITION ch_p,
SUBPARTITION ch_s,
SUBPARTITION ch_t),
PARTITION SALES_Q4_2000 VALUES LESS THAN (MAXVALUE)
SUBPARTITIONS 4)
;
```

次の例は、サンプル表 `oe.customers` に基づいて、顧客のパーティション表を作成します。この例では、表は `credit_limit` 列でパーティション化され、`nls_territory` 列でリスト・サブパーティション化されます。個々のサブパーティションを定義してテンプレートを上書きしないかぎり、サブパーティション・テンプレートによって、後から追加されたパーティションのサブパーティション化が決定されます。このコンポジット・パーティション化によって、指定した地域内の掛貸限度範囲に基づいて、表を問い合わせることができます。

```
CREATE TABLE customers_part (
  customer_id      NUMBER(6),
  cust_first_name  VARCHAR2(20),
  cust_last_name   VARCHAR2(20),
  nls_territory    VARCHAR2(30),
  credit_limit     NUMBER(9,2))
PARTITION BY RANGE (credit_limit)
SUBPARTITION BY LIST (nls_territory)
SUBPARTITION TEMPLATE
(SUBPARTITION east VALUES
('CHINA', 'JAPAN', 'INDIA', 'THAILAND'),
SUBPARTITION west VALUES
('AMERICA', 'GERMANY', 'ITALY', 'SWITZERLAND'),
```



```

        SUBPARTITION other VALUES (DEFAULT))
(PARTITION p1 VALUES LESS THAN (1000),
 PARTITION p2 VALUES LESS THAN (2500),
 PARTITION p3 VALUES LESS THAN (MAXVALUE));

```

オブジェクト列と表の例

オブジェクト表の作成例： 次の文は、オブジェクト型 department_typ を指定します。

```

CREATE TYPE department_typ AS OBJECT
( d_name   VARCHAR2(100),
  d_address VARCHAR2(200) );
/

```

オブジェクト表 departments_obj_t は、department_typ 型の部門オブジェクトを保持します。

```

CREATE TABLE departments_obj_t OF department_typ;

```

次の文は、ユーザー定義オブジェクト型 salesrep_typ を持つオブジェクト表 salesreps を作成します。

```

CREATE OR REPLACE TYPE salesrep_typ AS OBJECT
( repId NUMBER,
  repName VARCHAR2(64));

CREATE TABLE salesreps OF salesrep_typ;

```

有効範囲付 REF を持つ表の作成例： 次の例では、department_typ 型および departments_obj_t 表（15-73 ページの「[オブジェクト表の作成例](#)」で作成）を使用します。この文を実行すると、有効範囲付き REF を持つ表が作成されます。

```

CREATE TABLE employees_obj
( e_name   VARCHAR2(100),
  e_number NUMBER,
  e_dept   REF department_typ SCOPE IS departments_obj_t );

```

次の文は、参照整合性制約が定義されている REF 列を含む表を作成します。

```

CREATE TABLE employees_obj
( e_name   VARCHAR2(100),
  e_number NUMBER,
  e_dept   REF department_typ REFERENCES departments_obj_t);

```

ユーザー定義 OID を含む表の作成例： 次の文は、オブジェクト型および OID が主キー・ベースの対応するオブジェクト表を作成します。

```
CREATE TYPE employees_typ AS OBJECT
    (e_no NUMBER, e_address CHAR(30));

CREATE TABLE employees_obj_t OF employees_typ (e_no PRIMARY KEY)
    OBJECT IDENTIFIER IS PRIMARY KEY;
```

この後は、次の 2 つのいずれかの方法で emp オブジェクト表を参照できます。

```
CREATE TABLE departments_t
    (d_no    NUMBER,
     mgr_ref REF employees_typ SCOPE IS employees_obj_t);

CREATE TABLE departments_t (
    d_no NUMBER,
    mgr_ref REF employees_typ
        CONSTRAINT mgr_in_emp REFERENCES employees_obj_t);
```

タイプ列での制約の指定例

```
CREATE TYPE address_t AS OBJECT
    ( hno    NUMBER,
      street VARCHAR2(40),
      city   VARCHAR2(20),
      zip    VARCHAR2(5),
      phone  VARCHAR2(10) );

CREATE TYPE person AS OBJECT
    ( name      VARCHAR2(40),
      dateofbirth DATE,
      homeaddress address,
      manager   REF person );

CREATE TABLE persons OF person
    ( homeaddress NOT NULL,
      UNIQUE (homeaddress.phone),
      CHECK (homeaddress.zip IS NOT NULL),
      CHECK (homeaddress.city <> 'San Francisco') );
```

CREATE TABLESPACE

用途

CREATE TABLESPACE 文を使用すると、**表領域**を作成できます。表領域とは、永続スキーマ・オブジェクトを格納する、データベース内に割り当てられた領域です。

表領域は、最初は読み書き両用として作成されます。その後、ALTER TABLESPACE 文でその表領域をオフラインまたはオンラインに設定したり、表領域にデータ・ファイルを追加したり、読み専用に設定することができます。

DROP TABLESPACE 文を使用して、データベースから表領域を削除することもできます。

CREATE TEMPORARY TABLESPACE 文を使用して、セッションの存続期間中のみスキーマ・オブジェクトを格納する表領域を作成できます。

参照：

- 表領域については、『Oracle9i データベース概要』を参照してください。
- 表領域の変更については、11-97 ページの「[ALTER TABLESPACE](#)」を参照してください。
- 表領域の削除については、17-9 ページの「[DROP TABLESPACE](#)」を参照してください。
- 15-86 ページの「[CREATE TEMPORARY TABLESPACE](#)」を参照してください。

前提条件

CREATE TABLESPACE システム権限が必要です。

表領域を作成する場合、表領域を格納するデータベースを作成する必要があります。また、そのデータベースをオープンしておく必要もあります。

参照： 13-22 ページの「[CREATE DATABASE](#)」を参照してください。

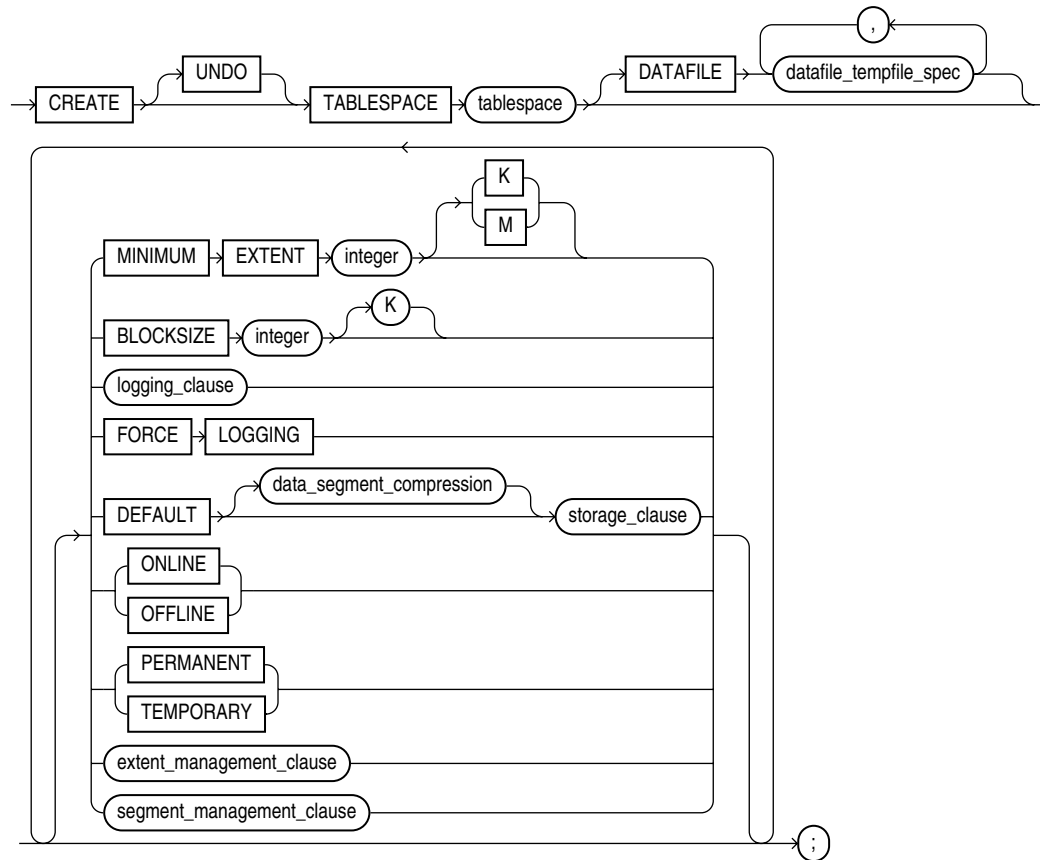
SYSTEM 以外の表領域のオブジェクトを使用する場合は、次の注意事項があります。

- ロールバック UNDO モードのデータベースを実行するには、1 つ以上のロールバック・セグメント（SYSTEM ロールバック・セグメント以外の）がオンラインである必要があります。
- 自動 UNDO 管理モードでデータベースを実行するには、1 つ以上の UNDO 表領域がオンラインである必要があります。

参照： 14-76 ページの「CREATE ROLLBACK SEGMENT」を参照してください。

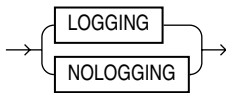
構文

create_tablespace::=

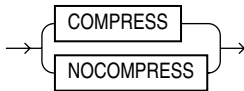


(7-36 ページの「`datafile_tempfile_spec::=`」(`file_specification` の一部)、15-77 ページの「`logging_clause::=`」、15-77 ページの「`data_segment_compression::=`」、7-53 ページの「`storage_clause::=`」、15-77 ページの「`extent_management_clause::=`」 および 15-77 ページの「`segment_management_clause::=`」を参照)

logging_clause::=

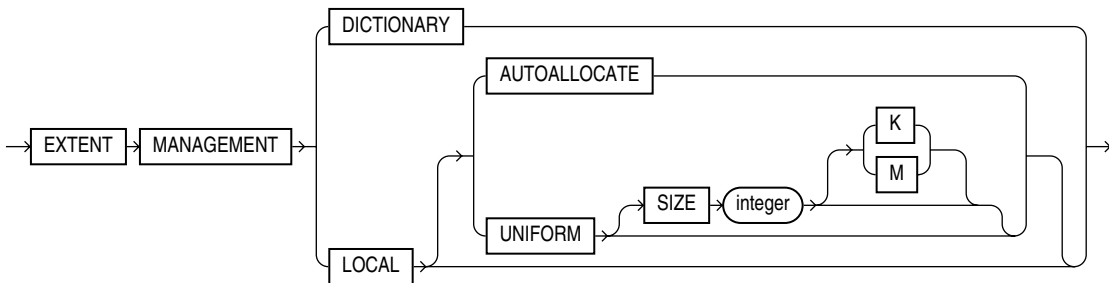


data_segment_compression::=

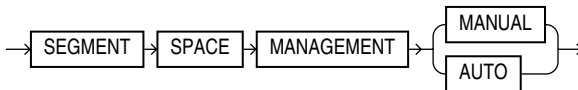


(7-51 ページの「[storage_clause](#)」を参照)

extent_management_clause::=



segment_management_clause::=



セマンティクス

UNDO

UNDO を指定すると、UNDO 表領域を作成できます。自動 UNDO 管理モードでデータベースを実行する場合、Oracle は、ロールバック・セグメントのかわりに UNDO 表領域を使用して UNDO 領域を管理します。自動 UNDO 管理モードで実行している場合にこの句は便利ですが、データベースは作成されません。

自動 UNDO 管理モードでデータベースを起動すると、UNDO 表領域が割り当てられます。UNDO 表領域がインスタンスに割り当てられない場合、SYSTEM ロールバック・セグメントが使用されます。UNDO 表領域の作成によってこれを回避することができ、他の UNDO 表領域がその時点で割り当てられていない場合、インスタンスに暗黙的に割り当てられます。

UNDO 表領域の制限事項：

- この表領域にはデータベース・オブジェクトを作成できません。システム管理の UNDO データ用に確保されています。
- ローカル・エクステント管理を指定するために UNDO 表領域に対して指定できるのは、DATAFILE 句および *extent_management_clause* 句のみです。
(*extent_management_clause* を使用して、ディクショナリ・エクステント管理を指定できません。) すべての UNDO 表領域は、永続的、読み込み / 書き込み可能およびロギング・モードで作成されます。MINIMUM EXTENT および DEFAULT STORAGE に対する値は、システムで生成されます。

参照：

- 自動 UNDO 管理および UNDO 表領域の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- データベース作成中の暗黙的または明示的な UNDO 表領域の作成については、13-22 ページの「[CREATE DATABASE](#)」を参照してください。
- UNDO 表領域の変更の詳細は、「[ALTER TABLESPACE](#)」を参照してください。
- UNDO 表領域の削除の詳細は、「[DROP TABLESPACE](#)」を参照してください。
- UNDO_MANAGEMENT パラメータを使用した自動 UNDO 管理モードのデータベース・インスタンスをオープンする場合の詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 15-84 ページの「[UNDO 表領域の作成例 :](#)」を参照してください。

tablespace

作成する表領域の名前を指定します。

DATAFILE *datafile_tempfile_spec*

表領域を構成するデータ・ファイルまたはファイルを指定します。

注意： RAW デバイスをサポートするオペレーティング・システムで RAW デバイスをデータ・ファイルとして指定する場合、*datafile_tempfile_spec* に REUSE キーワードを指定しても意味がありません。このような CREATE TABLESPACE 文は、REUSE の指定の有無にかかわらず実行されます。

DATAFILE 句は、初期化パラメータ DB_CREATE_FILE_DEST を設定する場合のみに使用するオプションです。この場合、パラメータで指定したデフォルトのファイル出力先に、100MB のファイルがシステム名で作成されます。このファイルでは AUTOEXTEND が有効になっているため、最大サイズの制限はありません。

参照：

- AUTOEXTEND パラメータなどの詳細は、7-36 ページの「[file_specification](#)」を参照してください。
- 15-84 ページの「[表領域の自動拡張を使用可能にする場合の例：](#)」および 15-85 ページの「[Oracle 管理ファイルの作成例：](#)」を参照してください。

MINIMUM EXTENT 句

表領域で使用するエクステントの最小サイズを指定します。この句によって、表領域内のすべての使用済エクステントまたは未使用エクステントの大きさが、integer で指定したサイズ以上であり、その倍数になるように指定することによって、表領域における空き領域の断片化を制御します。

注意： ディクショナリ管理の一時表領域の場合、この句は関係ありません。

参照： MINIMUM EXTENT を使用した断片化の制御の詳細は、『Oracle9i データベース概要』を参照してください。また、15-85 ページの「[最小エクステント・サイズの指定例：](#)」も参照してください。

BLOCKSIZE 句

BLOCKSIZE 句を使用すると、表領域に対して標準以外のブロック・サイズを指定できます。この句を指定するには、DB_CACHE_SIZE および 1 つ以上の DB_nK_CACHE_SIZE パラメータが設定されている必要があります。また、この句で指定する整数は、DB_nK_CACHE_SIZE パラメータの 1 つの設定と対応している必要があります。

BLOCKSIZE の制限事項： 一時表領域の場合（TEMPORARY を指定する場合）、または表領域を一時表領域として任意のユーザーに割り当てる場合は、標準以外のブロック・サイズを指定できません。

参照： バッファ・キャッシュでの複数のブロック・サイズの混在、およびパーティション・オブジェクトでの複数のブロック・サイズの使用に関する制限事項については、『Oracle9i データベース管理者ガイド』を参照してください。

logging_clause

表領域内のすべての表、索引、マテリアライズド・ビュー、マテリアライズド・ビュー・ログおよびパーティションのデフォルトのロギング属性を指定します。デフォルトは LOGGING です。

表レベル、索引レベル、マテリアライズド・ビュー・レベル、マテリアライズド・ビュー・ログ・レベル、パーティション・レベルでのロギングを指定することで、表領域レベルのロギング属性を上書きできます。

参照： この句の詳細は、7-42 ページの「[logging_clause](#)」を参照してください。

FORCE LOGGING

この句を使用すると、表領域は FORCE LOGGING モードになります。一時セグメントへの変更を除き、表領域内のオブジェクトに対するすべての変更のログが取られ、個々のオブジェクトの NOLOGGING 設定が上書きされます。データベースをオープンし、READ WRITE モードにしておく必要があります。

この設定によって、NOLOGGING 属性が削除されることはありません。つまり、FORCE LOGGING と NOLOGGING の両方を指定できます。この場合、後から表領域に作成されるオブジェクトのデフォルト・ロギング・モードは NOLOGGING になりますが、表領域（またはデータベース）が FORCE LOGGING モードの場合、このデフォルト値は無視されます。この表領域で FORCE LOGGING モードを無効にすると、デフォルト値である NOLOGGING が再度適用されます。

注意： FORCE LOGGING モードは、パフォーマンスに影響する場合があります。この設定の使用については、『Oracle9i データベース管理者ガイド』を参照してください。

強制ロギングの制限事項： FORCE LOGGING は、UNDO 表領域および一時表領域に対して指定できません。

DEFAULT *storage_clause*

表領域内に作成されるすべてのオブジェクトに対するデフォルトの記憶域パラメータを指定します。

ディクショナリ管理の一時表領域の場合、*storage_clause* の NEXT パラメータのみを考慮します。

参照： 記憶域パラメータの詳細は、7-51 ページの「[storage_clause](#)」を参照してください。また、15-84 ページの「[デフォルト記憶域を持つ表領域の作成例](#)」も参照してください。

ONLINE | OFFLINE 句

ONLINE ONLINE を指定して、表領域に対するアクセス権限を付与されているユーザーに対して、作成直後の表領域を使用可能にします。これはデフォルト値です。

OFFLINE OFFLINE を指定して、作成直後の表領域を使用禁止にします。

データ・ディクショナリ・ビュー DBA_TABLESPACES は、各表領域がオンラインまたはオフラインのいずれであるかを示します。

PERMANENT | TEMPORARY 句

PERMANENT 表領域を永続オブジェクトの格納に使用する場合は、PERMANENT を指定します。これはデフォルト値です。

TEMPORARY 表領域を一時オブジェクトの格納にのみ使用する場合は、TEMPORARY を指定します。たとえば、ORDER BY 句の処理での暗黙的なソートに使用されるセグメントの格納などです。

この句を使用して作成される一時表領域は常にディクショナリ管理されるため、EXTENT MANAGEMENT LOCAL 句は指定できません。ローカル管理一時表領域を作成するには、CREATE TEMPORARY TABLESPACE 文を使用します。

注意： CREATE TEMPORARY TABLESPACE 文を使用して、一時ファイルを含むローカル管理一時表領域を作成してください。ディクショナリ管理表領域の新規作成は、サポートされなくなる予定です。

一時表領域の制限事項： TEMPORARY を指定する場合、BLOCKSIZE 句は指定できません。

extent_management_clause

extent_management_clause によって、表領域のエクステンツの管理方法を指定できます。

注意： この句でエクステンツの管理を指定した場合は、表領域を移行しないかぎり、エクステンツの管理を変更できません。

- 表領域をローカルで管理する場合は、LOCAL を指定します。ローカル管理表領域では、ビットマップ用に表領域の一部を確保します。これはデフォルト値です。

- AUTOALLOCATE は、表領域がシステム管理されることを指定します。ユーザーはエクステント・サイズを指定できません。初期化パラメータ COMPATIBLE が 9.0.0 以上に設定されている場合は、これがデフォルトです。
- UNIFORM は、表領域が SIZE バイトの均一のエクステントで管理されることを指定します。K または M を使用して、KB または MB 単位でエクステント・サイズを指定することもできます。SIZE のデフォルト値は 1MB です。
- ディクショナリ表を使用して表領域を管理する場合は、DICTIONARY を指定します。初期化パラメータ COMPATIBLE が 9.0.0 未満に設定されている場合は、これがデフォルトです。

ディクショナリ管理表領域の制限事項： データベースの SYSTEM 表領域がローカル管理される場合、DICTIONARY は指定できません。

注意： ローカル管理表領域のみを作成してください。ローカル管理表領域は、ディクショナリ管理表領域よりも効率的に管理できます。ディクショナリ管理表領域の新規作成は、サポートされなくなる予定です。

extent_management_clause を指定しない場合、COMPATIBLE の設定、MINIMUM EXTENT 句および DEFAULT storage_clause が解析され、エクステント管理が判断されます。初期化パラメータ COMPATIBLE が 9.0.0 未満の場合は、ディクショナリ管理表領域が作成されます。初期化パラメータ COMPATIBLE が 9.0.0 以上の場合は、表領域はローカル管理となります。

- DEFAULT storage_clause を指定しない場合、ローカル管理の自動割当て表領域が作成されます。
- DEFAULT storage_clause を指定する場合は、次のようになります。
 - MINIMUM EXTENT 句を指定した場合、MINIMUM EXTENT、INITIAL および NEXT の値が等しく、PCTINCREASE の値が 0（ゼロ）であるかどうかが評価されます。その場合、エクステントのサイズが INITIAL で、ローカル管理される均一な表領域が作成されます。MINIMUM EXTENT、INITIAL および NEXT パラメータが等しくない場合または PCTINCREASE が 0（ゼロ）でない場合は、指定したエクステントの記憶域パラメータが無視され、ローカル管理の自動割当て表領域が作成されます。
 - MINIMUM EXTENT 句を指定しなかった場合、INITIAL および NEXT の記憶域の値が等しく、PCTINCREASE が 0（ゼロ）であるかどうかが評価されます。その場合、表領域はローカル管理され、エクステント・サイズは均一です。そうでない場合、表領域はローカル管理され、エクステント・サイズは自動割当てされます。

参照： ローカル管理表領域については、『Oracle9i データベース概要』を参照してください。

エクステンツ管理の制限事項：

- 永続的なローカル管理表領域には、永続オブジェクトを格納できます。ローカル管理表領域に一時オブジェクトを格納する必要がある場合（たとえば、ユーザーの一時表領域に割り当てる場合）は、CREATE TEMPORARY TABLESPACE 文を使用します。
- LOCAL を指定する場合、DEFAULT *storage_clause*、MINIMUM EXTENT または TEMPORARY は指定できません。

参照： 表領域の移行によってエクステンツの管理を変更する場合は、『Oracle9i データベース移行ガイド』を参照してください。また、15-85 ページの「[ローカル管理表領域の作成例：](#)」を参照してください。

segment_management_clause

segment_management_clause は、永続的で、ローカル管理表領域のみに関連します。Oracle が空きリストまたはビットマップを使用する表領域のセグメントにある使用済領域および空き領域を追跡するかどうかを指定できます。

MANUAL Oracle が空きリストを使用して表領域のセグメントにある空き領域を管理する場合は、MANUAL を指定します。

AUTO Oracle がビットマップを使用して表領域のセグメントにある空き領域を管理する場合は、AUTO を指定します。AUTO を指定すると、この表領域のオブジェクトに対して後で指定する記憶域仕様 PCTUSED、FREELIST および FREELIST GROUPS は無視されます。この設定を、**自動セグメント領域管理**といいます。

既存の表領域のセグメント管理を確認するには、DBA_TABLESPACES または USER_TABLESPACES データ・ディクショナリ・ビューの SEGMENT_SPACE_MANAGEMENT 列を問い合わせます。

注意： AUTO を指定する場合、次のことに注意します。

- エクステンツ管理を LOCAL UNIFORM に設定する場合は、各エクステンツに 5 以上のデータベース・ブロックがあり、データベースのブロック・サイズが指定されていることを確認する必要があります。
 - エクステンツ管理を LOCAL AUTOALLOCATE に設定する場合、およびデータベースのブロック・サイズが 16KB 以上の場合は、最小で 1MB のエクステンツが作成され、セグメント領域管理が管理されます。
-
-

自動セグメント領域管理の制限事項：

- この句は、永続的なローカル管理表領域に対してのみ指定できます。
- この句は、SYSTEM 表領域に対して指定できません。

参照：

- 自動セグメント領域管理およびその使用については、『Oracle9i データベース管理者ガイド』を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
- 15-85 ページの「[表領域に対してセグメント領域管理を指定する場合の例：](#)」を参照してください。

例

UNDO 表領域の作成例： 次の例は、データ・ファイル undotbs_1a.f を持つ、10MB の UNDO 表領域 undots1 を作成します。

```
CREATE UNDO TABLESPACE undots1
  DATAFILE 'undotbs_1a.f'
  SIZE 10M AUTOEXTEND ON;
```

デフォルト記憶域を持つ表領域の作成例： 次の文は、1 つのデータ・ファイルを持つ表領域 tbs_01 を作成します。

```
CREATE TABLESPACE tbs_01
  DATAFILE 'tbs_f2.dat' SIZE 40M
  DEFAULT STORAGE (INITIAL 128K NEXT 128K
    MINEXTENTS 1 MAXEXTENTS 999)
  ONLINE;
```

表領域の自動拡張を使用可能にする場合の例： 次の文は、1 つのデータ・ファイルを持つ表領域 tbs_02 を作成します。さらに領域が必要な場合、500KB のエクステンツが最大サイズ 10MB まで追加されます。

```
CREATE TABLESPACE tbs_02
  DATAFILE 'diskb:tbs_f5.dat' SIZE 500K REUSE
  AUTOEXTEND ON NEXT 500K MAXSIZE 100M;
```

最小エクステント・サイズの指定例： 次の文は、1 つのデータ・ファイルを持つ表領域 tbs_03 を作成し、すべてのエクステントを 500KB の倍数として割り当てます。

```
CREATE TABLESPACE tbs_03
  DATAFILE 'tbs_f03.dbf' SIZE 20M
  MINIMUM EXTENT 500K
  DEFAULT STORAGE (INITIAL 128K NEXT 128K)
  LOGGING;
```

ローカル管理表領域の作成例： 次の文は、データベース・ブロック・サイズが 2KB であると仮定します。

```
CREATE TABLESPACE tbs_04 DATAFILE 'file_1.f' SIZE 10M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

この文で、すべてのエクステントが 128KB で、ビットマップの各ビットが 64 ブロックを示す、ローカル管理表領域を作成します。

表領域に対してセグメント領域管理を指定する場合の例： 次の例では、自動セグメント領域管理の表領域を作成します。

```
CREATE TABLESPACE auto_seg_ts DATAFILE 'file_2.f' SIZE 1M
  EXTENT MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO;
```

Oracle 管理ファイルの作成例： 次の例は、データ・ファイルを作成するデフォルトの位置を設定し、デフォルトの位置にあるデータ・ファイルを持つ表領域を作成します。データ・ファイルは 100MB で自動拡張可能であり、最大サイズの制限がありません。

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST = '$ORACLE_HOME/rdbms/log';
```

```
CREATE TABLESPACE omf_ts1;
```

次の例は、Oracle 管理の、自動拡張されない 100MB のデータ・ファイルを持つ表領域を作成します。

```
CREATE TABLESPACE omf_ts2 DATAFILE AUTOEXTEND OFF;
```

CREATE TEMPORARY TABLESPACE

用途

CREATE TEMPORARY TABLESPACE 文を使用すると、ローカル管理一時表領域を作成できます。一時表領域は、セッションの存続期間中に、スキーマ・オブジェクトを含むことができるデータベース内の領域の割当てです。後でこの一時表領域を特定のユーザーに割り当てる場合、そのユーザーが開始するトランザクションのソート操作用にこの表領域が使用されます。

CREATE TABLESPACE 文を使用して、永続スキーマ・オブジェクトを含む表領域を作成します。

ディクショナリ管理される一時表領域を作成するには、TEMPORARY 句を指定した CREATE TABLESPACE 文を使用します。

注意： メディア・リカバリは一時ファイルを認識しません。

参照：

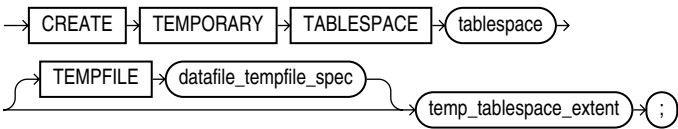
- 永続スキーマ・オブジェクトを格納する表領域およびディクショナリ管理される一時表領域の作成方法は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。
- ユーザーに対する一時表領域の割当てについては、16-31 ページの「[CREATE USER](#)」を参照してください。

前提条件

CREATE TABLESPACE システム権限が必要です。

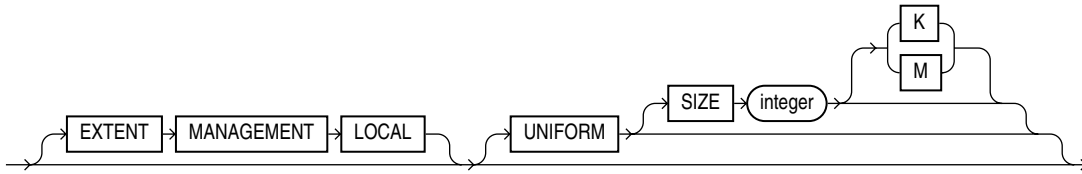
構文

create_temporary_tablespace::=



(7-36 ページの「[datafile_tempfile_spec::=](#)」、15-87 ページの「[temp_tablespace_extent::=](#)」を参照)

`temp_tablespace_extent::=`



セマンティクス

tablespace

一時表領域の名前を指定します。

TEMPFILE *datafile* *tempfile_spec*

表領域を構成する一時ファイルを指定します。

初期化パラメータ `DB_CREATE_FILE_DEST` が設定されている場合のみ、`TEMPFILE` 句を省略できます。この場合、パラメータで指定するデフォルトのファイル格納先に、Oracle 管理の 100MB の一時ファイルが作成されます。このファイルでは `AUTOEXTEND` が有効になっているため、最大サイズの制限はありません。`DB_CREATE_FILE_DEST` パラメータが設定されていない場合、`TEMPFILE` 句を指定する必要があります。

注意： オペレーティング・システムによっては、一時ファイルのブロックが実際にアクセスされるまで、一時ファイル用の領域が割り当てられない場合があります。領域の割当ての遅延のため、一時ファイルの作成およびサイズ変更が速くなります。ただし、後で一時ファイルが使用されるときに、十分なディスク領域を使用可能にする必要があります。発生する可能性がある問題を回避するには、一時ファイルを作成またはサイズ変更する前に、ディスク領域が、新しく作成する一時ファイルまたはサイズ変更後の一時ファイルのサイズより大きいことを確認してください。ディスク領域に余裕を持たせることによって、関連のない操作が原因で増加が予想されるディスク使用量に対応できます。十分な領域があることを確認した後で、作成またはサイズ変更操作を実行してください。

参照： `AUTOEXTEND` パラメータなどの詳細は、7-36 ページの「[file_specification](#)」を参照してください。

temp_tablespace_extent

temp_tablespace_extent 句を使用すると、表領域の管理方法を指定できます。

EXTENT MANAGEMENT LOCAL この句は、表領域の一部がビットマップ用に確保されることを示します。CREATE TEMPORARY TABLESPACE を使用して作成したすべての一時表領域にはローカル管理されるエクステントがあるため、この句はオプションです。ディクショナリ管理される一時表領域を作成するには、TEMPORARY 句を指定した CREATE TABLESPACE 文を使用します。

UNIFORM 一時表領域のエクステントはすべて同じサイズ（均一）で、キーワードはオプションです。ただし、SIZE を指定する場合は、UNIFORM を指定する必要があります。

SIZE integer 表領域のエクステントのサイズをバイト単位で指定します。K または M を使用して、KB または MB 単位で指定することもできます。

SIZE を指定しない場合は、デフォルトのエクステント・サイズ 1MB を使用します。

参照： ローカル管理表領域については、『Oracle9i データベース概要』を参照してください。

例

一時表領域の作成例： 次の文は、サンプル・データベースのデータベース・ユーザーに対するデフォルトの一時表領域を作成します。

```
CREATE TEMPORARY TABLESPACE temp_demo
  TEMPFILE 'temp01.dbf' SIZE 5M AUTOEXTEND ON;
```

デフォルトのデータベース・ブロック・サイズを 2KB とした場合、マップの各ビットは 1 つのエクステントを表し、各ビットは 8000 ブロックをマップします。

次の例は、データ・ファイルを作成するデフォルトの位置を設定し、デフォルトの位置にある Oracle 管理一時ファイルを持つ表領域を作成します。一時ファイルは 100MB で自動拡張可能であり、最大サイズの制限がありません（Oracle 管理ファイルのデフォルト値）。

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST = '$ORACLE_HOME/rdbms/log';

CREATE TEMPORARY TABLESPACE tbs_05;
```

CREATE TRIGGER

用途

CREATE TRIGGER 文を使用すると、次のデータベース・トリガーを作成および使用可能にすることができます。

- 表、スキーマまたはデータベースに対応したストアド PL/SQL ブロック
- 無名 PL/SQL ブロック、あるいは PL/SQL または JAVA で実装されているプロシージャへのコール

指定された条件が発生した場合、トリガーは自動的に実行されます。

トリガーを作成した場合、そのトリガーは自動的に使用可能になります。この後、DISABLE 句および ENABLE 句を指定した ALTER TRIGGER 文または ALTER TABLE 文を使用して、トリガーを使用禁止または使用可能にすることができます。

参照：

- トリガーの様々な型については、『Oracle9i データベース概要』を参照してください。
- トリガーの設計方法の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- トリガーの使用可能化、使用禁止化またはコンパイルの詳細は、12-2 ページの「ALTER TRIGGER」および 11-2 ページの「ALTER TABLE」を参照してください。
- トリガーの削除については、17-12 ページの「DROP TRIGGER」を参照してください。

前提条件

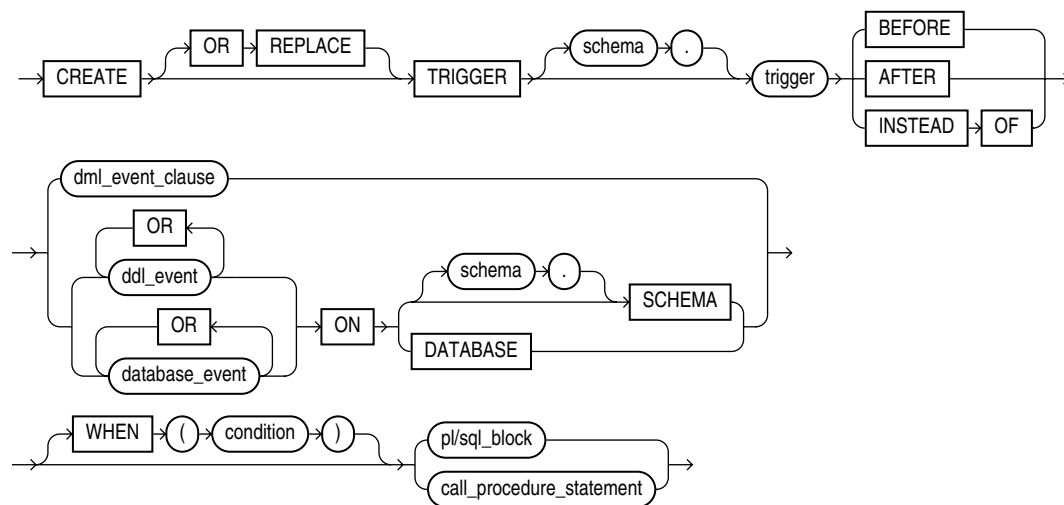
トリガーを作成する前に、ユーザー SYS は、DBMSSTD_X.SQL と呼ばれる SQL スクリプトを実行する必要があります。このスクリプトの正確なファイル名および位置は、使用するオペレーティング・システムによって異なります。

- 自分のスキーマ内の表または自分のスキーマ (SCHEMA) に対するトリガーを自分のスキーマ内に作成する場合は、CREATE TRIGGER 権限が必要です。
- 任意のスキーマ内の表または別のユーザーのスキーマ (schema.SCHEMA) に対するトリガーを任意のスキーマ内に作成する場合は、CREATE ANY TRIGGER 権限が必要です。
- 前述の権限の他にも、データベースに対するトリガーを作成する場合は、ADMINISTER DATABASE TRIGGER システム権限が必要です。

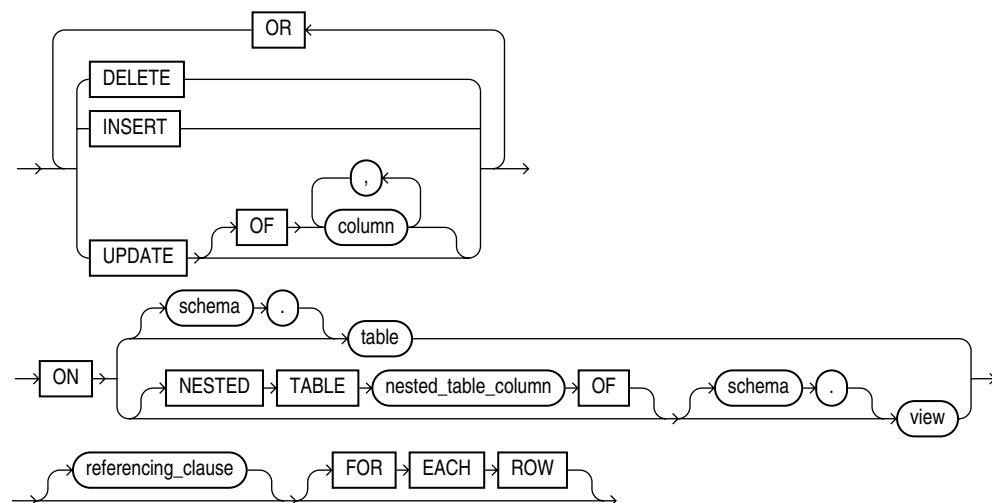
トリガーが SQL 文を発行、またはプロシージャやファンクションをコールする場合、そのトリガーの所有者には、これらの操作を行うための権限が必要です。これらの権限はロールを介して付与するのではなく、所有者に直接付与する必要があります。

構文

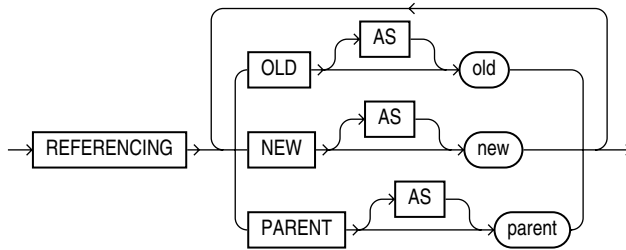
create_trigger::=



dml_event_clause::=



referencing_clause::=



セマンティクス

OR REPLACE

既存のトリガーを再作成する場合は、OR REPLACE を指定します。この句を指定すると、既存のトリガーの定義を削除しなくても変更できます。

schema

作成するトリガーが含まれるスキーマを指定します。schema を指定しない場合、自分のスキーマにトリガーが作成されます。

trigger

作成するトリガーの名前を指定します。

トリガーのコンパイル・エラーが発生した場合、このトリガーは作成されますが実行時に正常に処理されません。この場合、コンパイル・エラーになったトリガーが使用禁止にされるか、コンパイル・エラーのないバージョンに置き換えられるか、または削除されるまでは、トリガーを起動するすべての DML 文は実行できません。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

注意： マテリアライズド・ビューの実表にトリガーを作成する場合は、トリガーがマテリアライズド・ビューのリフレッシュ中に起動しないようにする必要があります（リフレッシュ中、DBMS_MVIEW プロシージャ I_AM_A_REFRESH は TRUE を戻します）。

BEFORE

BEFORE を指定すると、トリガーを起動するイベントが実行される前に、トリガーが起動されます。行レベル・トリガーの場合、対象となる各行が変更される前に起動されます。

BEFORE トリガーの制限事項：

- ビューおよびオブジェクト・ビューに対しては BEFORE トリガーを指定できません。
- :NEW 値の書込みはできますが、:OLD 値の書込みはできません。

AFTER

AFTER を指定すると、トリガーを起動するイベント実行後に、トリガーが起動されます。行レベル・トリガーの場合、対象となる各行が変更された後に起動されます。

AFTER トリガーの制限事項：

- ビューおよびオブジェクト・ビューに対しては AFTER トリガーを指定できません。
- :OLD 値も :NEW 値も書込みはできません。

注意： 表に対するマテリアライズド・ビュー・ログを作成した場合、その表に AFTER ROW トリガーが暗黙的に作成されます。このトリガーは、INSERT 文、UPDATE 文または DELETE 文で表のデータが変更された場合、マテリアライズド・ビュー・ログに 1 行を挿入します。複数の行レベル・トリガーを起動する順序は制御できません。マテリアライズド・ビューの内容に影響するトリガーは記述しないでください。

参照： マテリアライズド・ビュー・ログの詳細は、14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」を参照してください。

INSTEAD OF

INSTEAD OF を指定すると、トリガーを起動するイベントのかわりに、トリガーが起動されます。INSTEAD OF トリガーは、ビューの DML イベントに有効です。DDL またはデータベース・イベントには有効ではありません。

ビューが更新可能であり、そのビューに INSTEAD OF トリガーがある場合、トリガーが優先されます。つまり、ビューで DML を実行するかわりに、Oracle はトリガーを起動します。

ビューが階層に属する場合、サブビューはトリガーを継承しません。

注意： Oracle ファイングレイイン・アクセス・コントロールを使用すると、ビューで行レベルのセキュリティ・ポリシーを定義できます。これらのポリシーは、DML 操作に応じて定義されたルールを施行します。INSTEAD OF トリガーもビューで定義した場合、Oracle は、ビューで DML を実行するかわりに INSTEAD OF トリガーを起動するため、行レベルのセキュリティ・ポリシーを施行しません。

INSTEAD OF トリガーの制限事項：

- INSTEAD OF トリガーは、ビューのみに有効です。INSTEAD OF トリガーは表には指定できません。
- :OLD 値と :NEW 値の両方の読み込みはできますが、:OLD 値と :NEW 値の書き込みはできません。

注意： 同一の表に対する同一の文について、同じ型（BEFORE、AFTER または INSTEAD OF）のトリガーを複数作成できます。これらのトリガーが起動される順番は不確定です。アプリケーションが同じ文を対象とした同じ種類のトリガーを続けて起動する必要がある場合、これらのトリガーを1つのトリガーに結合し、適切な順序で元のトリガーのトリガー・アクションを実行するようにしてください。

参照： 15-101 ページの「[INSTEAD OF トリガーの作成例：](#)」を参照してください。

dml_event_clause

dml_event_clause によって、トリガーを起動する3つのDML文のうちの1つを指定できます。Oracle は、既存のユーザー・トランザクションでトリガーを起動します。

参照： 15-100 ページの「[DML トリガーの作成例：](#)」を参照してください。

DELETE

DELETE 文が表から行を削除する、またはネストした表から要素を削除するときにトリガーを起動する場合は、DELETE を指定します。

INSERT

INSERT 文が表に行を挿入する、またはネストした表に要素を挿入するときにトリガーを起動する場合は、INSERT を指定します。

UPDATE

OF 句の後に指定した列のいずれかの値を UPDATE 文で変更したときに、トリガーを起動する場合は、UPDATE を指定します。OF 句を省略した場合、UPDATE 文で表またはネストした表の任意の列の値を変更するたびにトリガーが起動されます。

UPDATE トリガーでは、オブジェクト型、VARRAY 型および REF 型の列を OF 句の後に指定した場合、UPDATE 文によってこれらの列のいずれかの値が変更される場合にトリガーが起動されます。ただし、トリガー自体の列の値は変更できません。

注意： Oracle Call Interface (OCI) 関数または DBMS_LOB パッケージを使用してオブジェクト列の LOB 値または LOB 属性を更新した場合、その列または属性を含む表に定義されているトリガーは起動されません。

UPDATE 操作でのトリガーの制限事項：

- UPDATE OF は、INSTEAD OF トリガーに指定できません。UPDATE 文でビューの任意の列の値を変更した場合、INSTEAD OF トリガーが起動されます。
- UPDATE OF 句には、ネストした表または LOB 列を指定できません。

参照： ビューに対する挿入、更新または削除を禁止する構造体については、16-38 ページの「[CREATE VIEW](#)」の「AS subquery」を参照してください。

ネストした表の列に DML 操作を直接実行した場合、そのネストした表の列が入っている表に定義されているトリガーは起動されません。

ddl_event

トリガーを起動する 1 つ以上の DDL 文を指定します。特に指定がないかぎり、DATABASE または SCHEMA のイベント用のトリガーを作成できます。これらのイベント用の BEFORE トリガーおよび AFTER トリガーを作成できます。Oracle は、既存のユーザー・トランザクションでトリガーを起動します。

DDL イベントのトリガーの制限事項： PL/SQL プロシージャを介して実行された DDL 操作は、トリガー・イベントとして指定できません。

参照： 15-101 ページの「[DDL トリガーの作成例:](#)」を参照してください。

次の `ddl_event` 値が有効です。

ALTER ALTER を指定すると、ALTER 文でデータ・ディクショナリのデータベース・オブジェクトを変更した場合、トリガーが起動されます。

ALTER 操作でのトリガーの制限事項： トリガーは、ALTER DATABASE 文では起動されません。

ANALYZE ANALYZE を指定すると、統計情報が収集または削除された場合、またはデータベース・オブジェクトの構造が検証された場合、トリガーが起動されます。

ASSOCIATE STATISTICS ASSOCIATE STATISTICS を指定すると、統計タイプがデータベース・オブジェクトと関連付けられた場合、トリガーが起動されます。

AUDIT AUDIT を指定すると、SQL 文の状態変化またはスキーマ・オブジェクトに対する操作が追跡された場合、トリガーが起動されます。

COMMENT COMMENT を指定すると、データベース・オブジェクトに対するコメントがデータ・ディクショナリに追加された場合、トリガーが起動されます。

CREATE CREATE を指定すると、CREATE 文でデータ・ディクショナリに新しいデータベース・オブジェクトを追加した場合、トリガーが起動されます。

CREATE 操作でのトリガーの制限事項： トリガーは、CREATE DATABASE 文または CREATE CONTROLFILE 文では起動されません。

DISASSOCIATE STATISTICS DISASSOCIATE STATISTICS を指定すると、統計タイプがデータベース・オブジェクトとの関連付けを解除した場合、トリガーが起動されます。

DROP DROP を指定すると、DROP 文でデータ・ディクショナリのデータベース・オブジェクトを削除した場合、トリガーが起動されます。

GRANT GRANT を指定すると、ユーザーが、別のユーザーまたはロールにシステム権限、ロールまたはオブジェクト権限を付与した場合、トリガーが起動されます。

NOAUDIT NOAUDIT を指定すると、NOAUDIT 文で SQL 文またはスキーマ・オブジェクトの操作の追跡を停止させた場合、トリガーが起動されます。

RENAME RENAME を指定すると、RENAME 文でデータベース・オブジェクトの名前を変更した場合、トリガーが起動されます。

REVOKE REVOKE を指定すると、REVOKE 文でユーザーまたはロールからシステム権限、ロールまたはオブジェクト権限を取り消した場合、トリガーが起動されます。

TRUNCATE TRUNCATE を指定すると、TRUNCATE 文で表またはクラスタから行を削除し、記憶特性を再設定した場合、トリガーが起動されます。

DDL DDL を指定すると、前述の DDL 文のいずれかを発行した場合、トリガーが起動されます。

database_event

トリガーを起動する 1 つ以上のデータベースの状態を指定します。特に指定がないかぎり、DATABASE または SCHEMA のイベント用のトリガーを作成できます。これらのトリガー・イベントについて、Oracle が自律型トランザクションの有効範囲をオープンし、トリガーを起動して、(既存のユーザー・トランザクションには関係なく) 別のトランザクションをコミットします。

参照： 15-101 ページの「データベース・イベント・トリガーの作成例」を参照してください。

SERVERERROR SERVERERROR を指定すると、サーバー・エラー・メッセージがログされた場合、トリガーが起動されます。

次のエラーが発生しても、SERVERERROR トリガーは起動されません（string には文字列が入ります）。

- ORA-01403: データが見つかりません。
- ORA-01422: 完全フェッチが要求よりも多くの行を戻しました
- ORA-01423: 完全フェッチで余分な行をチェック中にエラーが発生しました
- ORA-01034: Oracle は使用できません。
- ORA-04030: string バイト (string,string) を割り当てようとしてプロセス・メモリーが不足しました。

LOGON LOGON を指定すると、クライアント・アプリケーションがデータベースにログオンした場合、トリガーが起動されます。

LOGOFF LOGOFF を指定すると、クライアント・アプリケーションがデータベースからログオフした場合、トリガーが起動されます。

STARTUP STARTUP を指定すると、データベースがオープンされた場合、トリガーが起動されます。

SHUTDOWN SHUTDOWN を指定すると、データベースのインスタンスが停止された場合、トリガーが起動されます。

SUSPEND SUSPEND を指定すると、サーバー・エラーによってトランザクションが停止した場合、トリガーが起動されます。

注意：

- LOGON、STARTUP、SERVERERROR および SUSPEND に適用されるのは AFTER トリガーのみです。
 - LOGOFF および SHUTDOWN に適用されるのは BEFORE トリガーのみです。
 - DATABASE のみに適用されるのは AFTER STARTUP トリガーおよび BEFORE SHUTDOWN トリガーです。
-
-

参照： 自律型トランザクションの有効範囲の詳細は、『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。

ON *table* | *view*

ON 句によって、トリガーが作成されるデータベース・オブジェクトを決定できます。

table | *view*

トリガーを作成する、次のいずれかのスキーマ、表またはビューの名前を指定します。

- 表またはビュー
- オブジェクト表またはオブジェクト・ビュー
- ネストした表型の列

schema を指定しない場合、この表が自分のスキーマに存在するとみなされます。トリガーは、索引構成表に作成できます。

スキーマの制限事項： トリガーは、SYS スキーマ内の表には作成できません。

NESTED TABLE 句

ビューの列 *nested_table_column* に定義するトリガーを指定します。そのようなトリガーは、DML の操作対象がネストした表の要素である場合にのみ起動されます。

ネストされた表の制限事項： NESTED TABLE は、INSTEAD OF トリガーにのみ指定できます。

DATABASE

DATABASE を指定すると、データベース全体にトリガーを定義します。データベース・ユーザーがトリガー・イベントを開始すると、トリガーが起動されます。

SCHEMA

SCHEMA を指定すると、現行のスキーマにトリガーを定義します。*schema* として接続しているユーザーがトリガー・イベントを開始すると、トリガーが起動されます。

参照： 15-102 ページの「[SCHEMA トリガーの作成例 :](#)」を参照してください。

referencing_clause

referencing_clause によって、関連名を指定できます。関連名は、特に現行の行における新旧の値を参照する場合に、PL/SQL ブロックおよび行レベル・トリガーの **WHEN** 条件で使用します。デフォルトの関連名は **OLD** および **NEW** です。行レベル・トリガーを **OLD** または **NEW** という表に対応付ける場合は、この句を使用して異なる関連名を指定します。これにより、表名と関連名との混乱を避けることができます。

- トリガーがネストした表に定義される場合、**OLD** および **NEW** はネストした表の行を参照し、**PARENT** は親である表の現行の行を参照します。
- オブジェクト表またはビューにトリガーが定義されている場合、**OLD** および **NEW** はオブジェクト・インスタンスを参照します。

***referencing_clause* の制限事項：** *referencing_clause* は、DDL イベント **CREATE** の **INSTEAD OF** トリガーには無効です。

FOR EACH ROW

FOR EACH ROW を指定すると、トリガーを行トリガーとして設定します。行レベル・トリガーは、トリガーを起動する文の対象になり、かつ **WHEN** 条件で定義したオプションのトリガー制約を満たす行ごとに 1 回ずつ起動されます。

INSTEAD OF トリガー以外のトリガーを指定する際に、この句を省略した場合、そのトリガーは文レベル・トリガーになります。文レベル・トリガーは、トリガーを起動する文が発行されたときにオプションのトリガー制約が満たされていると、1 回のみ起動されます。

INSTEAD OF トリガー文は、各行について暗黙的にアクティブになります。

行トリガーの制限事項： この句は、(DDL およびデータベース・イベント・トリガーではなく) DML イベント・トリガーにのみ有効です。

WHEN 句

トリガー制約 (Oracle がトリガーを起動するために必要な **SQL** 条件) を指定します。*condition* の構文は、[第 5 章「条件」](#) を参照してください。この条件には関連名を指定する必要があります。問合せは指定できません。

NEW および **OLD** キーワードを **WHEN** 句で指定する場合は、バインド変数とみなされないため、コロンの (:) を前に付けません。ただし、**WHEN** 句以外のすべての参照では、コロンを付けて **NEW** および **OLD** を指定する必要があります。

参照： 15-101 ページの「[トリガー本体のプロシージャのコール例:](#)」を参照してください。

トリガー条件の制限事項：

- DML イベント・トリガーに対してこの句を指定する場合、FOR EACH ROW も指定する必要があります。トリガーを起動する文の対象となる行ごとに、この条件が評価されます。
- INSTEAD OF トリガー文にトリガー制約は指定できません。
- オブジェクト列、オブジェクト列の属性、VARRAY、ネストした表、LOB 列を参照できます。トリガー制約内では PL/SQL ファンクションおよびメソッドは起動できません。

pl/sql_block

Oracle がトリガーを起動するために実行する PL/SQL ブロックを指定します。

データベース・トリガーの PL/SQL ブロックは、システム・イベント属性の抽出用にのみ設計された SYS スキーマ内の一連の組込みファンクションの 1 つを指定できます。これらのファンクションは、データベース・トリガーの PL/SQL ブロック内でのみ使用できます。

トリガーの実装の制限事項：

- トリガーの PL/SQL ブロックは、ブロックが同一のトランザクションで実行される場合、トランザクション制御 SQL 文 (COMMIT、ROLLBACK、SAVEPOINT および SET CONSTRAINT) を指定できません。
- PL/SQL ブロック内のトリガー・アクションの LOB 列を参照および使用できます。トリガー・アクション内にある LOB 列の :NEW 値は変更できますが、:OLD 値は変更できません。

参照：

- PL/SQL ブロックの書き込み方法など、PL/SQL の詳細は、『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。
- これらのファンクションの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 15-101 ページの「[トリガー本体のプロシージャのコール例:](#)」を参照してください。

call_procedure_statement

call_procedure_statement 句によって、トリガー・コード・インラインを PL/SQL ブロックとして指定するかわりに、ストアード・プロシージャをコールできるようになります。この文の構文は、次の例外を除いて、12-63 ページの「[CALL](#)」の文と同じです。

- ファンクションにのみ適用されるため、CALL の INTO 句は指定できません。
- *expr* 内のバインド変数は指定できません。

- 定義されているトリガーの表の列を参照する場合、:NEW および :OLD を指定する必要があります。

参照： 15-101 ページの「[トリガー本体のプロシージャのコール例:](#)」を参照してください。

例

DML トリガーの作成例： 次の文では、名前をつけられる BEFORE トリガーの基本構文を示します。そのトリガーを書き込み、この表に発行された DML 文に制限を配置します（その文が発行できる場合）。

```
CREATE TRIGGER schema.trigger_name
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON schema.table_name
  pl/sql_block
```

DELETE、INSERT または UPDATE の各文が表に対して実行されると、トリガーが起動されます。トリガーは BEFORE 文トリガーであるため、このトリガーを起動する文が実行される前に 1 回起動されます。

次の例では、パーシャル BEFORE 行トリガーを示します。たとえば、PL/SQL ブロックによって、従業員の給与が従業員の職種に対して設定された給与の範囲内にあるよう指定されたとします。

```
CREATE TRIGGER hr.salary_check
  BEFORE INSERT OR UPDATE OF salary, job_id ON hr.employees
  FOR EACH ROW
  WHEN (new.job_id <> 'AD_VP')
  pl/sql_block
```

このトリガーは、次の文のいずれかが発行されるたびに起動されます。

- employees 表に対して行を追加する INSERT 文
- employees 表の salary 列値または job_id 列値を変更する UPDATE 文

salary_check は BEFORE 行レベル・トリガーであるため、UPDATE 文で更新される各行を変更または INSERT 文で挿入される各行を追加する前に、このトリガーが起動されます。

salary_check には、管理部門の副社長（AD_VP）の給与を確認できないトリガー制約が設定されています。

DDL トリガーの作成例： 次の文は、任意の DDL 文 CREATE の AFTER 文トリガーを作成します。このトリガーを使用して、自分のスキーマにある新規データ・ディクショナリ・オブジェクトの作成を監査します。

```
CREATE TRIGGER audit_db_object AFTER CREATE
ON SCHEMA
pl/sql_block
```

トリガー本体のプロシージャのコール例： PL/SQL ブロック内のトリガー本体を設定するかわりに、コール側プロシージャによって前述の例で記述された salary_check トリガーを作成することができます。従業員の給与が適切な範囲内にあることを確認する hr.salary_check プロシージャを定義したと仮定します。次のようにトリガー salary_check を作成できます。

```
CREATE TRIGGER hr.salary_check
BEFORE INSERT OR UPDATE OF salary, job_id ON hr.employees
FOR EACH ROW
WHEN (new.job_id <> 'AD_VP')
CALL check_sal(:new.job_id, :new.salary, :new.last_name);
```

check_sal プロシージャは、PL/SQL、C または Java で実装されます。また、:NEW 値のかわりに CALL 句の :OLD 値を指定できます。

データベース・イベント・トリガーの作成例 次の文では、すべてのエラーをログする基本構文を示します。PL/SQL ブロックが特定のエラー（無効なログイン: エラー番号は 1017）に対して特別な処理を行うと仮定します。このトリガーは、AFTER 文トリガーで、文の実行に失敗（ログインの失敗など）した後で起動されます。

```
CREATE TRIGGER log_errors AFTER SERVERERROR ON DATABASE
BEGIN
IF (IS_SERVERERROR (1017)) THEN
    <special processing of logon error>
ELSE
    <log error number>
END IF;
END;
```

INSTEAD OF トリガーの作成例： この例では、顧客情報および注文情報を表示する oe.order_info ビューを作成します。

```
CREATE VIEW order_info AS
SELECT c.customer_id, c.cust_last_name, c.cust_first_name,
       o.order_id, o.order_date, o.order_status
FROM customers c, orders o
WHERE c.customer_id = o.customer_id;
```

orders 表の主キー (order_id) が結合ビューの結果セットで一意でないため、通常は、このビューは更新できません。このビューを更新可能にするには、ビューに INSTEAD OF トリガーを作成し、ビューに対して INSERT 文を処理します。

```
CREATE OR REPLACE TRIGGER order_info_insert
  INSTEAD OF INSERT ON order_info
  DECLARE
    duplicate_info EXCEPTION;
    PRAGMA EXCEPTION_INIT (duplicate_info, -00001);
  BEGIN
    INSERT INTO customers
      (customer_id, cust_last_name, cust_first_name)
    VALUES (
      :new.customer_id,
      :new.cust_last_name,
      :new.cust_first_name);
    INSERT INTO orders (order_id, order_date, customer_id)
    VALUES (
      :new.order_id,
      :new.order_date,
      :new.customer_id);
  EXCEPTION
    WHEN duplicate_info THEN
      RAISE_APPLICATION_ERROR (
        num=> -20107,
        msg=> 'Duplicate customer or order ID');
  END order_info_insert;
/
```

すべての NOT NULL 列に値があれば、ビューを介してどちらの実表にも追加できるようになります。

```
INSERT INTO order_info VALUES
  (999, 'Smith', 'John', 2500, '13-MAR-2001', 0);
```

SCHEMA トリガーの作成例： 次の例は、サンプル・スキーマ hr に BEFORE 文トリガーを作成します。hr というアカウントで接続したユーザーがデータベース・オブジェクトを削除しようとする、オブジェクトの削除前に、このトリガーが起動されます。

```
CREATE OR REPLACE TRIGGER drop_trigger
  BEFORE DROP ON hr.SCHEMA
  BEGIN
    RAISE_APPLICATION_ERROR (
      num => -20000,
      msg => 'Cannot drop object');
  END;
/
```

16

SQL 文 : CREATE TYPE ~ DROP ROLLBACK SEGMENT

この章では、次の SQL 文について説明します。

- CREATE TYPE
- CREATE TYPE BODY
- CREATE USER
- CREATE VIEW
- DELETE
- DISASSOCIATE STATISTICS
- DROP CLUSTER
- DROP CONTEXT
- DROP DATABASE LINK
- DROP DIMENSION
- DROP DIRECTORY
- DROP FUNCTION
- DROP INDEX
- DROP INDEXTYPE
- DROP JAVA
- DROP LIBRARY
- DROP MATERIALIZED VIEW

-
- DROP MATERIALIZED VIEW LOG
 - DROP OPERATOR
 - DROP OUTLINE
 - DROP PACKAGE
 - DROP PROCEDURE
 - DROP PROFILE
 - DROP ROLE
 - DROP ROLLBACK SEGMENT

CREATE TYPE

用途

CREATE TYPE 文を使用すると、**オブジェクト型**、**SQLJ オブジェクト型**（オブジェクト型の一種）、名前付きの可変配列（**VARRAY**）、**ネストした表型**または**不完全なオブジェクト型**の仕様部を作成できます。CREATE TYPE 文および CREATE TYPE BODY 文を使用してオブジェクト型を作成します。CREATE TYPE 文では、オブジェクト型の名前、オブジェクトの属性、メソッドおよびその他のプロパティを指定します。CREATE TYPE BODY 文には、その型でのメソッドに対するコードが含まれます。

注意：

- 型仕様で属性のみ宣言し、メソッドは宣言しないオブジェクト型を作成する場合は、型本体を指定する必要はありません。
 - SQLJ オブジェクト型を作成する場合は、型本体を指定できません。型の実装は **Java** クラスとして指定されます。
-

ユーザー定義型を作成した場合、Oracle は、暗黙的に、そのコンストラクタ・メソッドを定義します。**コンストラクタ**は、システム提供のプロシージャで、SQL 文または PL/SQL コード内で、その型の値のインスタンスを構成するために使用します。コンストラクタ・メソッドの名前は、ユーザー定義型の名前と同じです。また、*constructor_spec* 構文を使用して、ユーザー定義コンストラクタを作成することもできます。

オブジェクト型のコンストラクタ・メソッドのパラメータは、オブジェクト型のデータ属性です。また、そのオブジェクト型用の属性定義と同じ順序で定義されます。ネストした表または VARRAY コンストラクタのパラメータは、ネストした表または VARRAY の要素です。

不完全型とは、フォワード型定義によって作成される型です。このオブジェクト型には名前がありますが、属性およびメソッドがないため、不完全といわれます。他の型からの参照が可能なため、互いに参照する型の定義に使用できます。ただし、不完全オブジェクト型を使用して表やオブジェクト列またはネストした表型の列を作成する場合は、型を完全に指定しておく必要があります。

参照：

- 型のメンバー・メソッドの作成については、16-25 ページの「**CREATE TYPE BODY**」を参照してください。
- オブジェクト、不完全型、VARRAY およびネストした表の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』および『Oracle9i データベース概要』を参照してください。

前提条件

自分のスキーマ内に型を作成する場合は、CREATE TYPE システム権限が必要です。他のユーザーのスキーマ内に型を作成する場合は、CREATE ANY TYPE システム権限が必要です。これらの権限は、明示的に取得することもロールを介して取得することもできます。

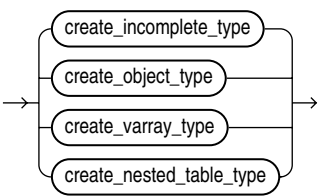
サブタイプを作成する場合は、UNDER ANY TYPE システム権限またはスーパータイプに対する UNDER オブジェクト権限が必要です。

型の所有者には、表定義内で参照する他のすべての型にアクセスするための EXECUTE オブジェクト権限が必要です。または、EXECUTE ANY TYPE システム権限が必要です。所有者は、これらの権限をロールを介して取得することはできません。

型の所有者が、型にアクセスする権限を他のユーザーに付与する場合、所有者には、参照型に対する Grant Option 付きの EXECUTE オブジェクト権限、または Admin Option 付きの EXECUTE ANY TYPE システム権限が必要です。これらの権限がないと、型の所有者は、型にアクセスする権限を他のユーザーに付与できません。

構文

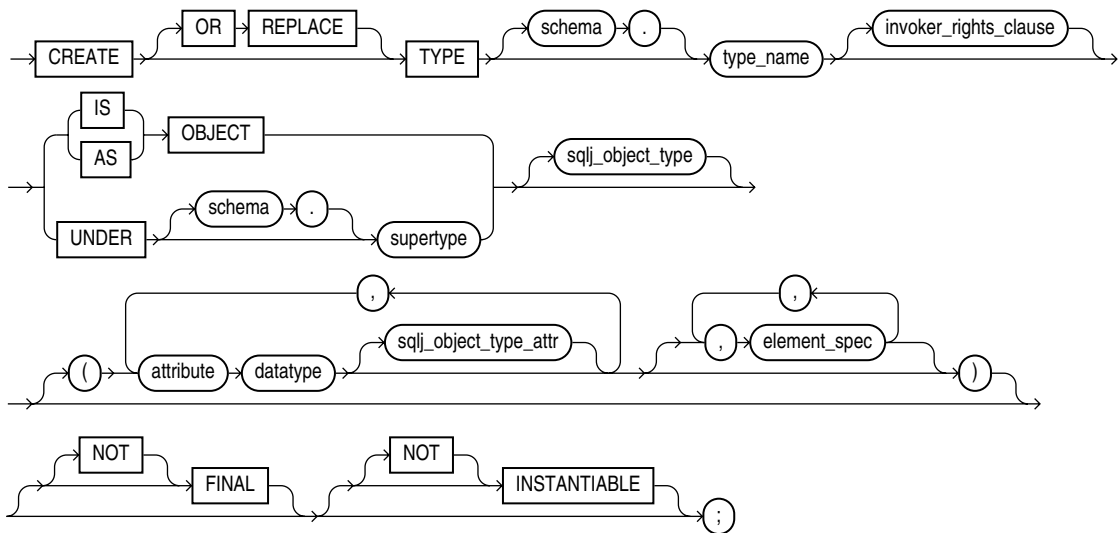
create_type::=



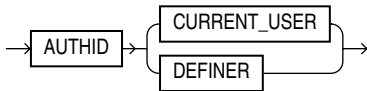
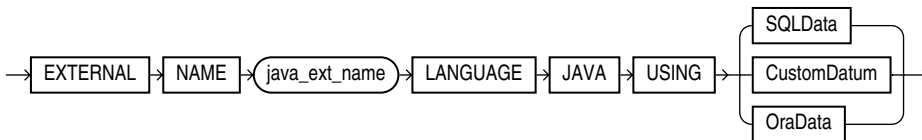
(16-4 ページの「[create_incomplete_type::=](#)」、16-5 ページの「[create_object_type::=](#)」、16-8 ページの「[create_varray_type::=](#)」、16-9 ページの「[create_nested_table_type::=](#)」を参照)

create_incomplete_type::=

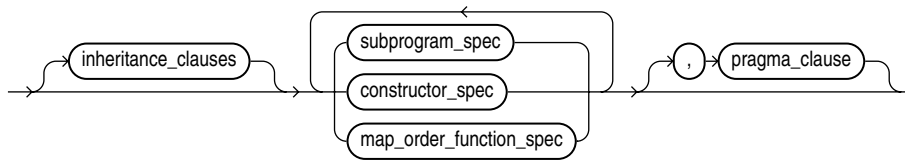


create_object_type::=

(16-5 ページの「[invoker_rights_clause::=](#)」、16-6 ページの「[element_spec::=](#)」を参照)

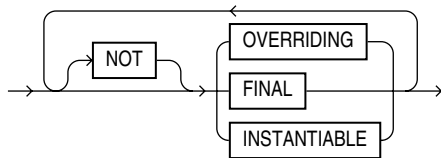
invoker_rights_clause::=**sqlj_object_type::=****sqlj_object_type_attr::=**

element_spec::=

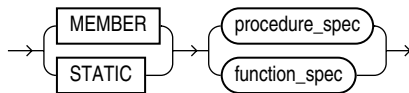


(16-6 ページの「[inheritance_clauses::=](#)」、16-6 ページの「[subprogram_spec::=](#)」、
16-7 ページの「[constructor_spec::=](#)」、16-7 ページの
「[map_order_function_spec::=](#)」、16-8 ページの「[pragma_clause::=](#)」を参照)

inheritance_clauses::=

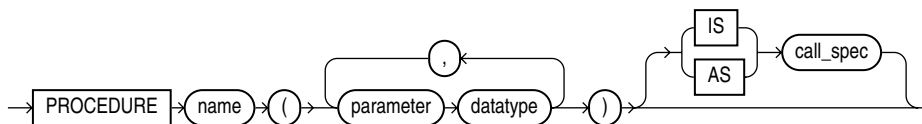


subprogram_spec::=



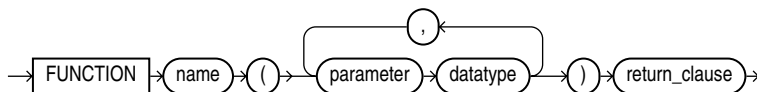
(16-6 ページの「[procedure_spec::=](#)」、16-6 ページの「[function_spec::=](#)」を参照)

procedure_spec::=

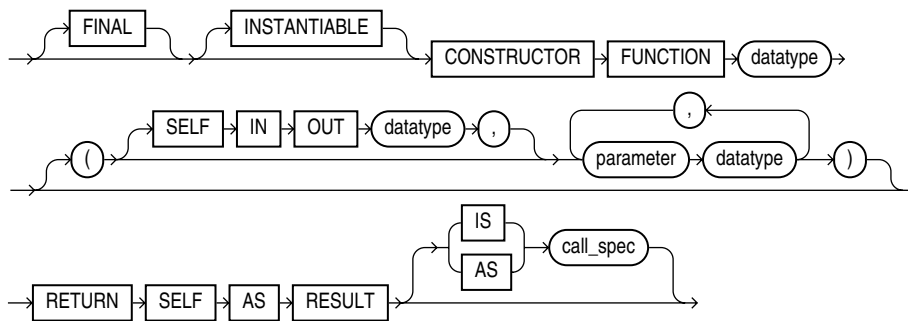


(16-8 ページの「[call_spec::=](#)」を参照)

function_spec::=



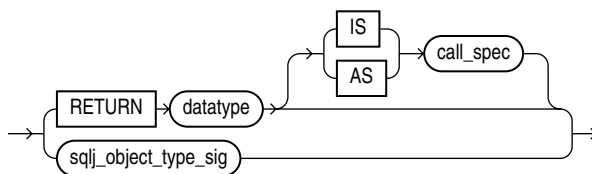
(16-7 ページの「[return_clause::=](#)」を参照)

constructor_spec::=

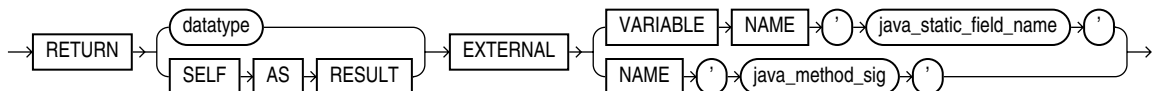
(16-8 ページの「[call_spec::=](#)」を参照)

map_order_function_spec::=

(16-6 ページの「[function_spec::=](#)」を参照)

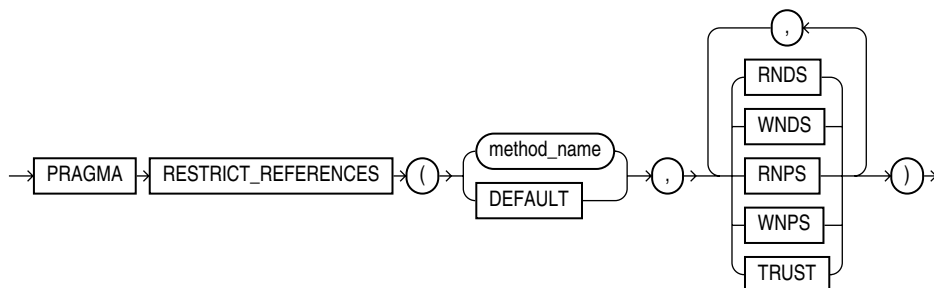
return_clause::=

(16-8 ページの「[call_spec::=](#)」、16-7 ページの「[sqlj_object_type_sig::=](#)」を参照)

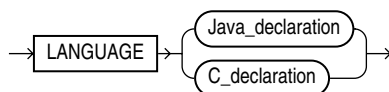
sqlj_object_type_sig::=

CREATE TYPE

pragma_clause::=



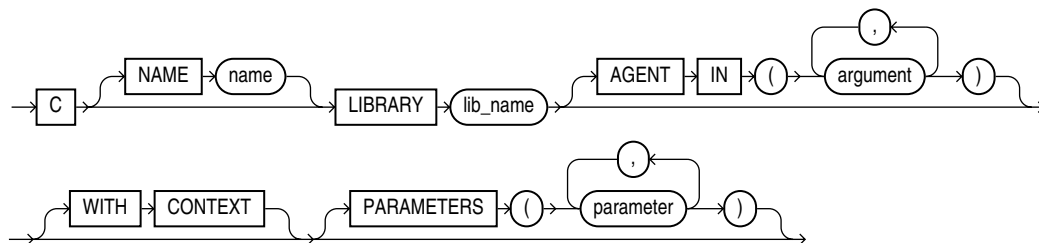
call_spec::=



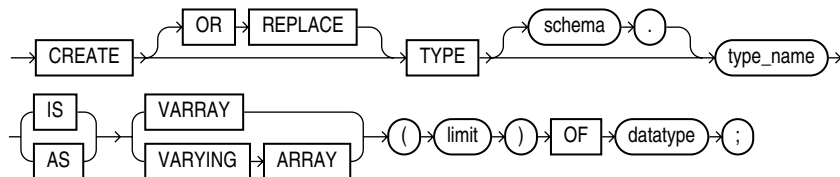
Java_declaration::=



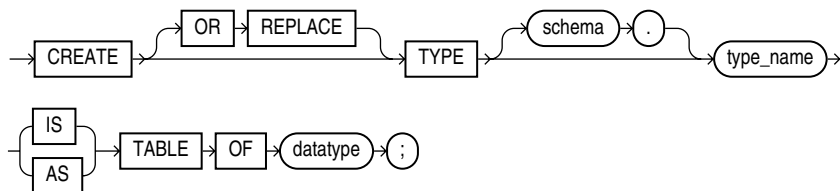
C_declaration::=



create_varray_type::=



create_nested_table_type::=



セマンティクス

OR REPLACE

既存の型を再作成する場合は、OR REPLACE を指定します。この句を指定した場合、既存のトリガーの定義をはじめに削除しなくても、その定義を変更できます。

再作成するオブジェクト型に対する権限があらかじめ付与されている場合は、再作成後にあらためて権限を付与されなくてもそのオブジェクト型を使用および参照できます。

ファンクション索引が型によって異なる場合、その索引には DISABLED のマークが付きます。

schema

作成する型が定義されるスキーマを指定します。schema を指定しない場合、現行のスキーマにその型が作成されます。

type_name

オブジェクト型、ネストした表型または VARRAY 型の名前を指定します。

型の作成時にコンパイル・エラーが発生した場合、エラーを戻します。SQL*Plus コマンド SHOW ERRORS を使用して、関連するコンパイラ・エラー・メッセージを表示できます。

create_object_type

create_object_type 句を使用すると、(不完全型ではなく) ユーザー定義型オブジェクト型を作成できます。データ構造を形成する変数を**属性**といいます。オブジェクトの動作を定義するメンバー・サブプログラムを**メソッド**といいます。キーワード AS OBJECT は、オブジェクト型の作成時に必要です。

参照： 16-19 ページの「[オブジェクト型の例:](#)」を参照してください。

invoker_rights_clause

invoker_rights_clause を使用すると、オブジェクト型のメンバー・ファンクションおよびプロシージャが、そのオブジェクト型を所有するユーザーのスキーマで権限付きで実行されるか、または `CURRENT_USER` のスキーマで権限付きで実行されるかを指定できます。この指定は、対応する型本体にも適用されます。

また、この句は、問合せ、DML 操作、およびその型のメンバー・ファンクションおよびプロシージャ内の動的 SQL 文の外部名の変換方法も定義します。

- `AUTHID CURRENT_USER` を指定して、オブジェクト型のメンバー・ファンクションおよびプロシージャを `CURRENT_USER` の権限で実行します。この句によって **実行者権限型** が作成されます。

また、この句は、問合せ、DML 操作、および動的 SQL 文の外部名を `CURRENT_USER` のスキーマで変換することも指定します。その他すべての文の外部名は、型が存在するスキーマ内で変換します。

- ファンクションおよびプロシージャが存在するスキーマの所有者権限で、オブジェクト型のメンバー・ファンクションおよびプロシージャを実行し、メンバー・ファンクションおよびプロシージャが存在するスキーマで外部名を変換する場合は、`AUTHID DEFINER` を指定します。これはデフォルトであり、**定義者権限型** を作成します。

実行者権限の制限事項：

- この句はオブジェクト型だけに指定でき、ネストした表および `VARRAY` 型には指定できません。
- サブタイプを作成する場合は、明確にするためにこの句を指定できます。ただし、サブタイプはスーパータイプの権限モデルを継承するため、スーパータイプに指定した値と異なる値を指定できません。
- スーパータイプが定義者権限で作成された場合、スーパータイプと同じスキーマでサブタイプを作成する必要があります。

参照：

- `CURRENT_USER` の判断方法については、『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 『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。

AS OBJECT 句

`AS OBJECT` を指定すると、トップレベル（ルート）のオブジェクト型を作成できます。

UNDER 句

UNDER *supertype* を指定すると、既存の型のサブタイプを作成できます。既存のスーパータイプは、オブジェクト型である必要があります。この文で作成するサブタイプは、そのスーパータイプのプロパティを継承します。また、プロパティの一部を上書きしたり、スーパータイプから識別する新しいプロパティを追加します。

参照： 16-20 ページの「[サブタイプの例 :](#)」および 16-22 ページの「[型の階層例 :](#)」を参照してください。

sqlj_object_type

この句を指定すると、**SQLJ オブジェクト型**を作成することができます。SQLJ オブジェクト型の場合、Java クラスを SQL のユーザー定義型へマップします。他のユーザー定義型と同様に、SQLJ オブジェクト型に表または列を定義できます。

1 つの Java クラスを複数の SQLJ オブジェクト型にマップすることもできます。SQLJ オブジェクト型のサブタイプまたはスーパータイプが存在する場合、それも SQLJ オブジェクト型である必要があります。つまり、階層のすべての型は SQLJ オブジェクト型である必要があります。

java_ext_name Java クラスの名前を指定します。クラスが存在する場合、パブリックである必要があります。スキーマを含む Java の外部名は検証されます。

複数の SQLJ オブジェクト型を同一のクラスにマップすることもできます。ただし、次のことに注意してください。

- スーパータイプをマップするクラス直属のサブクラスに、サブタイプをマップする必要があります。
- スーパータイプが共通の 2 つのサブタイプを、同一のクラスにマップすることはできません。

SQLData | CustomDatum | OraData 型の Java インスタンスを作成するメカニズムを選択します。SQLData、CustomDatum および OraData は、使用するメカニズムのインタフェースです。

参照： これらのインタフェースの詳細は、『Oracle9i JDBC 開発者ガイド およびリファレンス』を参照してください。16-20 ページの「[SQLJ オブジェクト型の例 :](#)」も参照してください。

element_spec

element_spec を指定すると、オブジェクト型の各属性を指定できます。

attribute

attribute には、オブジェクト属性の名前を指定します。属性とは、名前と型指定子を持つオブジェクト構造を形成するデータ項目です。各オブジェクト型には、1 つ以上の属性を指定する必要があります。

サブタイプを作成する場合、連鎖するスーパータイプで宣言されている属性名またはメソッド名と同じ属性名を指定することはできません。

datatype

datatype には、属性の Oracle 組み込みデータ型またはユーザー定義型を指定します。

データ型の属性の制限事項：

- ROWID 型、LONG 型または LONG ROW 型の属性は指定できません。
- ユーザー定義オブジェクト型に対して UROWID のデータ型は指定できません。
- REF 型のオブジェクトを指定する場合、ターゲット・オブジェクトにはオブジェクト識別子が必要です。
- ネストした表または表の VARRAY 列として使用するためのコレクション型を作成する場合、AnyType 型、AnyData 型または AnyDataSet 型の属性は指定できません。

参照： 指定可能なデータ型については、2-2 ページの「[データ型](#)」を参照してください。

sqlj_object_type_attr

この句は、*sqlj_object_type* 句を指定する場合（Java クラスを SQLJ オブジェクト型にマップする場合）のみに有効です。SQLJ オブジェクト型の属性に対応する Java フィールドの外部名を指定します。Java の *field_name* は、クラスに存在する必要があります。Java の *field_name* を、同じ型の階層にある 1 つ以上の SQLJ オブジェクト型の属性にマップすることはできません。

この句は、SQLJ オブジェクト型を作成する場合のオプションです。

subprogram_spec

subprogram_spec を指定すると、プロシージャ・サブプログラムをオブジェクト型に対応付けることができます。

MEMBER 句

属性として参照されるオブジェクト型に関連付けられたファンクションまたはプロシージャ・サブプログラムを指定します。通常、*object_expression.method()* のように、「自己参照的」スタイルでメンバー・メソッドを起動します。このクラスのメソッドには、メソッド本体で SELF として参照される暗黙的な最初の引数があります。この引数は、メソッドが起動されるオブジェクトを表します。

メンバー・メソッドの制限事項： Java クラスを SQLJ オブジェクト型へマップする場合は、メンバー・メソッドを指定できません。

参照： 16-23 ページの「[メンバー・メソッドの作成例:](#)」を参照してください。

STATIC 句

オブジェクト型に関連付けられたファンクションまたはプロシージャ・サブプログラムを指定します。メンバー・メソッドとは異なり、スタティック・メソッドには暗黙的なパラメータがありません（本体で SELF を参照できません）。通常、`type_name.method()` として起動されます。

スタティック・メソッドの制限事項：

- Java クラスのメンバー・メソッドを SQLJ オブジェクト型のスタティック・メソッドにマップできません。
- メンバー・メソッドの場合も、スタティック・メソッドの場合も、プロシージャの仕様部またはファンクションの仕様部に対応するメソッド本体をオブジェクト型本体に指定する必要があります。

参照： 16-23 ページの「[スタティック・メソッドの作成例:](#)」を参照してください。

[NOT] FINAL, [NOT] INSTANTIABLE

これらの句を構文の最上位で指定すると、型の継承する属性が指定されます。

[NOT] FINAL 句を使用すると、この型のサブタイプを以降で作成させるかどうかを指定できます。

- この型のサブタイプを以降で作成させない場合は、FINAL を指定します。これはデフォルト値です。
- この型のサブタイプを以降で作成させる場合は、NOT FINAL を指定します。

[NOT] INSTANTIABLE 句を使用すると、この型のオブジェクト・インスタンスを構成させるかどうかを指定します。

- この型のオブジェクト・インスタンスを構成させる場合は、INSTANTIABLE を指定します。これはデフォルト値です。
- このオブジェクト型のコンストラクタ（デフォルトまたはユーザー定義の）が存在しない場合は、NOT INSTANTIABLE を指定します。インスタンス化ができないメソッドを持つ型、および属性を持たない型には、これらのキーワードを指定する必要があります（継承、またはこの句で指定）。

inheritance_clauses

element_spec の一部として *inheritance_clauses* を使用すると、スーパータイプとサブタイプの間の関係を指定できます。

OVERRIDING この句は、メンバー・メソッドでのみ使用可能です。OVERRIDING を指定すると、スーパータイプで定義されているメンバー・メソッドをこのメソッドで上書きします。このキーワードは、スーパータイプのメソッドを再定義する場合に必須です。NOT OVERRIDING がデフォルトです。

OVERRIDING の制限事項： OVERRIDING 句は、スタティック・メソッドまたは SQLJ オブジェクト型では無効です。

FINAL FINAL を指定すると、メソッドがこの型のサブタイプによって上書きされないようにできます。デフォルトは、NOT FINAL です。

NOT INSTANTIABLE 型がこのメソッドの実装を提供しない場合は、NOT INSTANTIABLE を指定します。デフォルトでは、すべてのメソッドは INSTANTIABLE です。

NOT INSTANTIABLE の制限事項： NOT INSTANTIABLE を指定する場合は、FINAL または STATIC を指定できません。

参照： 16-16 ページの「[*constructor_spec*](#)」を参照してください。

procedure_spec* または *function_spec

この句を使用すると、プロシージャまたはファンクションのパラメータおよびデータ型を指定できます。このサブプログラムは、プロシージャまたはファンクションの宣言を含みませんが、対応する CREATE TYPE BODY 文は発行する必要があります。

プロシージャの仕様部およびファンクションの仕様部の制限事項： サブタイプを作成する場合は、継承しているかどうかにかかわらず、プロシージャまたはファンクションに、連鎖するスーパータイプで宣言されている属性の名前と同じ名前を指定できません。

return_clause *return_clause* の最初の形式はファンクションのみに有効です。構文は省略形です。

参照：

- メソッド起動およびメソッドの詳細は、『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。
- 使用可能なすべての句を含む完全な構文については、14-60 ページの「[CREATE PROCEDURE](#)」および 13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- 16-25 ページの「[CREATE TYPE BODY](#)」を参照してください。
- ユーザー定義ファンクションの制限事項については、13-52 ページの「[ユーザー定義ファンクションの制限事項](#)」を参照してください。

sqlj_object_type_sig SQLJ オブジェクト型のファンクションまたはプロシージャを作成する場合は、*return_clause* のこの形式を使用します。

- Java クラスを SQLJ オブジェクト型にマップし、**EXTERNAL NAME** を指定する場合は、Java メソッドが戻す値は SQL が戻す値と互換性があり、また Java メソッドがパブリックである必要があります。また、メソッド・シグネチャ（メソッド名とパラメータ・タイプ）は、型の階層で一意である必要があります。
- **EXTERNAL VARIABLE NAME** を指定する場合、Java のスタティック・フィールドの型は戻り型と互換性がある必要があります。

call_spec

JAVA メソッドまたは C 関数名、パラメータ型および戻り型を SQL にマップするコール仕様を指定します。型のメンバー・メソッドすべてがこの句で定義された場合、対応する **CREATE TYPE BODY** 文を発行する必要はありません。

Java_declaration では、'*string*' が JAVA 実装メソッドを定義します。

参照：

- 『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- パラメータおよび *C_declaration* のセマンティクスについては、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

pragma_clause

pragma_clause によって、コンパイラ・ディレクティブを指定できます。PRAGMA RESTRICT_REFERENCES コンパイラ・ディレクティブは、データベースの表またはパッケージ変数（あるいはその両方）に対するメンバー・ファンクションの読み書きアクセスを拒否し、副作用の発生を防止します。

注意： アプリケーションの下位互換を保つ必要がない場合は、この句を使用しないことをお勧めします。Oracle9i からは、純粋度チェックが実行時に行われるため、この句は使用されなくなります。

method プラグマが適用されている MEMBER ファンクションまたはプロシージャの名前を指定します。

DEFAULT DEFAULT を指定すると、プラグマが明示的に指定されていない型のすべてのメソッドにプラグマが適用されます。

WNDS WNDS を指定すると、データベースに書き込み禁止状態制約（データベース表を変更しない）が適用されます。

WNPS WNPS を指定すると、パッケージに書き込み禁止状態制約（パッケージ変数を変更しない）が適用されます。

RNDS RNDS を指定すると、データベースに読み込み禁止状態制約（データベース表を問い合わせない）が適用されます。

RNPS RNPS を指定すると、パッケージに読み込み禁止状態制約（パッケージ変数を参照しない）が適用されます。

TRUST TRUST を指定すると、プラグマに指定されている制限が、実際に適用されているのではなく、単に真であることを指定します。

参照：『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

constructor_spec

この句を使用すると、ユーザー定義オブジェクト型の初期化インスタンスを戻すファンクションであるユーザー定義コンストラクタが作成できます。各コンストラクタのパラメータの番号、順序またはデータ型が異なる場合は、単一オブジェクト型に対して複数のコンストラクタを宣言できます。

- ユーザー定義コンストラクタ・ファンクションは、常に FINAL および INSTANTIABLE であるため、これらのキーワードはオプションです。
- ユーザー定義コンストラクタのパラメータの受渡しモードは、常に SELF IN OUT です。そのため、明示的に指定する必要がある場合を除き、この句を指定する必要はありません。
- RETURN SELF AS RESULT は、コンストラクタによって戻される値の実行時型が、SELF 引数の実行時型と同一であることを指定します。

参照： ユーザー定義コンストラクタの詳細および例は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。16-23 ページの「[コンストラクタの例:](#)」も参照してください。

map_order_function_spec

定義する MEMBER または STATIC メソッドの数にかかわらず、型仕様部には、MAP または ORDER メソッドのいずれかを定義できます。また、サブタイプには MAP および ORDER メソッドのどちらも定義できません。ただし、スーパータイプが非最終の MAP メソッドを定義する場合、サブタイプは MAP メソッドを上書きできます。(サブタイプは ORDER をオーバーライドできません。) いずれかのメソッドを宣言すると、SQL 内でオブジェクト・インスタンスを比較できます。

Java クラスから SQL 型へのマップ時に、MAP メソッドまたは ORDER メソッドのいずれかを定義できます。ただし、MAP メソッドおよび ORDER メソッドは、Java クラスのメンバー・ファンクションにマップする必要があります。

MAP メソッドも ORDER メソッドも定義されない場合、比較できるのは等価性または非等価性のみです。したがって、オブジェクト・インスタンスに順序を付けることはできません。同じ型定義のインスタンスは、それぞれの対応する属性の各組が等しい場合にのみ等しくなります。2 つのオブジェクト型の等価性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。

オブジェクト・インスタンスで大規模のソートまたはハッシュ結合処理を実行する場合に、MAP を使用してください。MAP を使用してオブジェクトをスカラー値にマップすると、そのスカラーは、ソート中およびマージ中に使用されます。MAP メソッドは、各オブジェクトを比較するメソッドを起動する必要がある ORDER メソッドより効率的です。ハッシュ結合には MAP を使用する必要があります。ハッシュ・メカニズムはオブジェクト値でハッシュするため、ORDER メソッドを使用することはできません。

参照： オブジェクト値の比較の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

MAP MEMBER この句は、オブジェクトの順序付けられたすべてのインスタンスの中から、指定したインスタンスの相対的な位置を戻すメンバー・ファンクション (MAP メソッド) を指定します。MAP メソッドは暗黙的にコールされ、オブジェクト・インスタンスを事前定義済みのスカラー型の値にマップすることによって、それらのオブジェクト・インスタンスに順序を設定します。PL/SQL は、この順序を使用してブール式の計算と比較を行います。

MAP メソッドの引数が NULL の場合、MAP メソッドは NULL を戻し、メソッドはコールされません。

オブジェクトの仕様部には、1つの MAP メソッドのみを指定できます。この MAP メソッドは、ファンクションである必要があります。結果として生成される型は、事前定義済の SQL スカラー型である必要があります。また、MAP メソッドに指定できる引数は、暗黙的な SELF 引数のみです。

注意：（ORDER BY 句、GROUP BY 句、DISTINCT 句または UNION 句を使用する）ソートまたは結合を指定した問合せが `type_name` を参照し、これらの問合せをパラレル化する場合は、MAP メンバー・ファンクションを指定する必要があります。

サブタイプには、新しい MAP メソッドを定義できません。ただし、継承された MAP メソッドは上書きできます。

ORDER MEMBER この句によって、オブジェクトのインスタンスを明示的な引数および暗黙的な SELF 引数として取るメンバー・ファンクション（ORDER メソッド）を指定し、負の整数、0（ゼロ）または正の整数のいずれかを戻します。負、正または 0（ゼロ）は、それぞれ、暗黙的な SELF 引数が明示的な引数より小さい、等しいまたは大きいことを示します。

ORDER メソッドのどちらかの引数が NULL の場合は、ORDER メソッドは NULL を返し、メソッドはコールされません。

同じオブジェクト型定義の各インスタンスを ORDER BY 句の中で比較した場合、ORDER メソッド `map_order_function_spec` がコールされます。

オブジェクトの仕様部には、1つの ORDER メソッドのみ指定でき、その ORDER メソッドは戻り型が NUMBER のファンクションである必要があります。

サブタイプは ORDER メソッドを定義および上書きできません。

create_varray_type

`create_varray_type` は、それぞれが同じデータ型を持つ要素の順序付け集合としての型を作成します。名前および 0（ゼロ）以上の最大限度を指定する必要があります。配列限度は、整数リテラルである必要があります。Oracle では、匿名の VARRAY はサポートされません。

VARRAY 内に含まれるオブジェクトの型名は、次のいずれかである必要があります。

- 組込みデータ型
- REF
- オブジェクト型

VARRAY 型の制限事項： プロシージャで使用するために（たとえば、PL/SQL、ビュー問合せなどで）、XMLType または LOB 型の VARRAY 型を作成できます。ただし、前述の VARRAY 型のデータベース格納がサポートされていないため、前述の VARRAY 型のオブジェクト表またはオブジェクト型列は作成できません。

参照： 16-22 ページの「[VARRAY 型の例 :](#)」を参照してください。

create_nested_table_type

create_nested_table_type によって、データ型で名前付きのネストした表を作成します。

- *datatype* がオブジェクト型である場合、ネストした表型は、オブジェクト型の名前および属性と一致する列を持つ表を記述します。
- *datatype* がスカラー型である場合、ネストした表の型は、「column_value」という 1 つのスカラー型列を持つ表を記述します。

ネストした表型の制限事項： *datatype* に NCLOB は指定できません。ただし、CLOB または BLOB は指定できます。

参照： 16-22 ページの「[名前付き表型の例 :](#)」および 16-22 ページの「[VARRAY を含むネストした表型の例 :](#)」を参照してください。

例

オブジェクト型の例： 次の例は、サンプルの注文入力スキーマ (oe) 用にサンプルの *customer_typ* 型を作成します。

```
CREATE TYPE customer_typ AS OBJECT
(
  customer_id      NUMBER(6)
, cust_first_name  VARCHAR2(20)
, cust_last_name   VARCHAR2(20)
, cust_address     CUST_ADDRESS_TYP
, phone_numbers    PHONE_LIST_TYP
, nls_language     VARCHAR2(3)
, nls_territory    VARCHAR2(30)
, credit_limit     NUMBER(9,2)
, cust_email       VARCHAR2(30)
, cust_orders      ORDER_LIST_TYP
);
```

次の例は、CREATE TYPE BODY 文で実装されるメンバー・ファンクション prod を持つ data_type オブジェクト型を作成します。

```
CREATE TYPE data_type AS OBJECT
(
  year NUMBER,
  MEMBER FUNCTION prod(invent NUMBER) RETURN NUMBER
);

CREATE TYPE BODY data_type IS
  MEMBER FUNCTION prod (invent NUMBER) RETURN NUMBER IS
    BEGIN
      RETURN (year + invent);
    END;
END;
```

サブタイプの例： 次の文は、サンプル・スキーマ oe にサブタイプ corporate_customer_type を作成します。これは前述の例で作成したスーパータイプ customer_type に基づき、account_mgr_id 属性を追加しています。

```
CREATE TYPE corporate_customer_type UNDER customer_type
(
  account_mgr_id NUMBER(6)
);
```

SQLJ オブジェクト型の例： 次の例は、SQLJ オブジェクト型およびサブタイプを作成します。address_t 型は、Java クラス Examples.Address にマップします。サブタイプ long_address_t は、Java クラス Examples.LongAddress にマップします。この例は、これらの型に対する Java インスタンスの作成に使用するメカニズムとして SQLData を指定します。これらの型仕様部の各ファンクションは、Java クラスの実装に対応しています。

参照： これらの型仕様部のファンクションを Java で実装する場合の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーションアル機能』を参照してください。

```
CREATE TYPE address_t AS OBJECT
EXTERNAL NAME 'Examples.Address' LANGUAGE JAVA
USING SQLData(
  street_attr varchar(250) EXTERNAL NAME 'street',
  city_attr varchar(50) EXTERNAL NAME 'city',
  state varchar(50) EXTERNAL NAME 'state',
  zip_code_attr number EXTERNAL NAME 'zipCode',
  STATIC FUNCTION recom_width RETURN NUMBER
    EXTERNAL VARIABLE NAME 'recommendedWidth',
  STATIC FUNCTION create_address RETURN address_t
    EXTERNAL NAME 'create()' return Examples.Address',
  STATIC FUNCTION construct RETURN address_t
    EXTERNAL NAME 'create()' return Examples.Address',
  STATIC FUNCTION create_address (street VARCHAR, city VARCHAR,
```

```

        state VARCHAR, zip NUMBER) RETURN address_t
    EXTERNAL NAME 'create (java.lang.String, java.lang.String, java.lang.String,
int) return Examples.Address',
    STATIC FUNCTION construct (street VARCHAR, city VARCHAR,
        state VARCHAR, zip NUMBER) RETURN address_t
    EXTERNAL NAME
        'create (java.lang.String, java.lang.String, java.lang.String, int) return
Examples.Address',
    MEMBER FUNCTION to_string RETURN VARCHAR
    EXTERNAL NAME 'tojava.lang.String() return java.lang.String',
    MEMBER FUNCTION strip RETURN SELF AS RESULT
    EXTERNAL NAME 'removeLeadingBlanks () return Examples.Address'
) NOT FINAL;

CREATE OR REPLACE TYPE long_address_t
UNDER address_t
EXTERNAL NAME 'Examples.LongAddress' LANGUAGE JAVA
USING SQLData(
    street2_attr VARCHAR(250) EXTERNAL NAME 'street2',
    country_attr VARCHAR (200) EXTERNAL NAME 'country',
    address_code_attr VARCHAR (50) EXTERNAL NAME 'addrCode',
    STATIC FUNCTION create_address RETURN long_address_t
    EXTERNAL NAME 'create() return Examples.LongAddress',
    STATIC FUNCTION construct (street VARCHAR, city VARCHAR,
        state VARCHAR, country VARCHAR, addr_cd VARCHAR)
    RETURN long_address_t
    EXTERNAL NAME
        'create(java.lang.String, java.lang.String,
        java.lang.String, java.lang.String, java.lang.String)
        return Examples.LongAddress',
    STATIC FUNCTION construct RETURN long_address_t
    EXTERNAL NAME 'Examples.LongAddress()'
        return Examples.LongAddress',
    STATIC FUNCTION create_longaddress (
        street VARCHAR, city VARCHAR, state VARCHAR, country VARCHAR,
        addr_cd VARCHAR) return long_address_t
    EXTERNAL NAME
        'Examples.LongAddress (java.lang.String, java.lang.String,
        java.lang.String, java.lang.String, java.lang.String)
        return Examples.LongAddress',
    MEMBER FUNCTION get_country RETURN VARCHAR
    EXTERNAL NAME 'country_with_code () return java.lang.String'
);

```

型の階層例： 次の文は、型の階層を作成します。employee_t 型は、person_t 型から name および ssn 属性を継承し、department_id および salary 属性を追加しています。part_time_emp_t 型は、employee_t および employee_t を介した person_t からすべての属性を継承し、num_hrs 属性を追加しています。part_time_emp_t はデフォルトで最終型であるため、そのサブタイプを作成できません。

```
CREATE TYPE person_t AS OBJECT (name VARCHAR2(100), ssn NUMBER)
    NOT FINAL;
```

```
CREATE TYPE employee_t UNDER person_t
    (department_id NUMBER, salary NUMBER) NOT FINAL;
```

```
CREATE TYPE part_time_emp_t UNDER employee_t (num_hrs NUMBER);
```

この型の階層を使用して、置換可能な表および置換可能な列を含む表を作成できます。詳細は、15-63 ページの「[置換可能な表および列のサンプル:](#)」を参照してください。

VARRAY 型の例： 次の文は、サンプル・スキーマ oe に、5 つの要素を含む VARRAY 型 phone_list_typ を作成します。

```
CREATE TYPE phone_list_typ AS VARRAY(5) OF VARCHAR2(25);
```

名前付き表型の例： 次の例は、サンプル・スキーマ pm でオブジェクト型 textdoc_typ の名前付き表型 textdoc_tab を作成します。

```
CREATE TYPE textdoc_typ AS OBJECT
    ( document_typ      VARCHAR2(32)
      , formatted_doc    BLOB
    ) ;
```

```
CREATE TYPE textdoc_tab AS TABLE OF textdoc_typ;
```

VARRAY を含むネストした表型の例： 次のマルチレベル・コレクションの例は、サンプル表 oe.customers の一例です。この例では、cust_address オブジェクト列は、VARRAY 列 phone_list_typ を埋め込んだネストした表の列になります。

```
CREATE TYPE phone_list_typ AS VARRAY(5) OF VARCHAR2(25);
```

```
CREATE TYPE cust_address_typ2 AS OBJECT
    ( street_address    VARCHAR2(40)
      , postal_code      VARCHAR2(10)
      , city             VARCHAR2(30)
      , state_province   VARCHAR2(10)
      , country_id       CHAR(2)
      , phone            phone_list_typ
    ) ;
```

```
CREATE TYPE cust_nt_address_typ
  AS TABLE OF cust_address_typ;
```

コンストラクタの例： 次の文は、システム定義のコンストラクタを起動して、demo_typ オブジェクトを構成し、構成したオブジェクトを demo_tab 表に入れます。

```
CREATE TYPE demo_typ AS OBJECT (a1 NUMBER, a2 NUMBER);

CREATE TABLE demo_tab (b1 NUMBER, b2 demo_typ);

INSERT INTO demo_tab VALUES (1, demo_typ(2,3));
```

参照： コンストラクタの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』および『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。

メンバー・メソッドの作成例： 次の文は、メソッド・コンストラクタ col.getbar() を起動します。(例は getbar メソッドがすでに存在するものとします。)

```
CREATE TYPE demo_typ2 AS OBJECT (a1 NUMBER,
  MEMBER FUNCTION getbar RETURN NUMBER);

CREATE TABLE demo_tab2(col demo_typ2);

SELECT col.getbar() FROM demo_tab2;
```

ファンクションとは異なり、メソッドを起動する場合は、メソッドが他に引数を取らない場合でもカッコが必要です。

スタティック・メソッドの作成例： 次の文は、employee_t 型の定義を変更し、変更した定義を construct_emp ファンクションと関連付けます。例は、最初にオブジェクト型 department_t を作成し、次に department_t 型の属性を含むオブジェクト型 employee_t を作成します。

```
CREATE OR REPLACE TYPE department_t AS OBJECT (
  deptno number(10),
  dname CHAR(30));

CREATE OR REPLACE TYPE employee_t AS OBJECT(
  empid RAW(16),
  ename CHAR(31),
  dept REF department_t,
  STATIC function construct_emp
    (name VARCHAR2, dept REF department_t)
  RETURN employee_t
);
```

この文は、次の型本体の文が必要です。

```
CREATE OR REPLACE TYPE BODY employee_t IS
    STATIC FUNCTION construct_emp
        (name varchar2, dept REF department_t)
    RETURN employee_t IS
    BEGIN
        return employee_t(SYS_GUID(),name,dept);
    END;
END;
```

次にオブジェクト表を作成し、表に挿入します。

```
CREATE TABLE emptab OF employee_t;
INSERT INTO emptab
    VALUES (employee_t.construct_emp('John Smith', NULL));
```

CREATE TYPE BODY

用途

CREATE TYPE BODY を使用すると、オブジェクト型仕様部で定義されたメンバー・メソッドを定義または実装することができます。CREATE TYPE 文および CREATE TYPE BODY 文を使用してオブジェクト型を作成します。CREATE TYPE 文では、オブジェクト型の名前、オブジェクトの属性、メソッドおよびその他のプロパティを指定します。CREATE TYPE BODY 文には、その型でのメソッドに対するコードが含まれます。

call_spec を定義していないオブジェクト型仕様部に指定された各メソッドには、オブジェクト型本体の対応するメソッド本体を指定する必要があります。

注意： SQLJ オブジェクト型を作成する場合は、型本体を指定できません。型の実装は Java クラスとして指定されます。

参照： 型仕様部の作成および変更については、16-3 ページの「[CREATE TYPE](#)」および 12-6 ページの「[ALTER TYPE](#)」を参照してください。

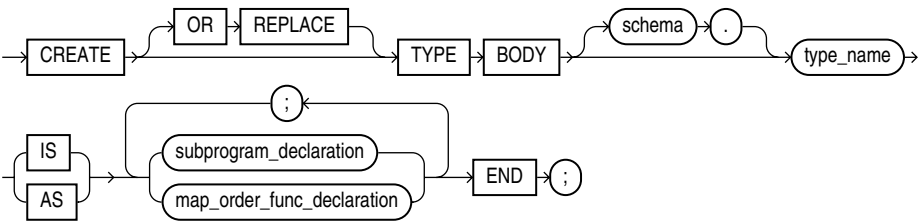
前提条件

オブジェクト型用の CREATE TYPE 仕様部で行われるすべてのメンバー宣言には、それに対応する構造が CREATE TYPE 文または CREATE TYPE BODY 文内に存在する必要があります。

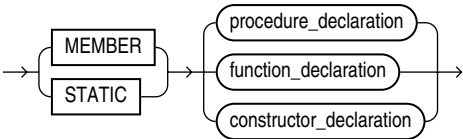
自分のスキーマ内で型本体を作成または再作成する場合は、CREATE TYPE システム権限または CREATE ANY TYPE システム権限が必要です。他のユーザーのスキーマ内でオブジェクト型を作成する場合は、CREATE ANY TYPE システム権限が必要です。他のユーザーのスキーマ内でオブジェクト型を置換する場合は、DROP ANY TYPE システム権限が必要です。

構文

create_type_body::=

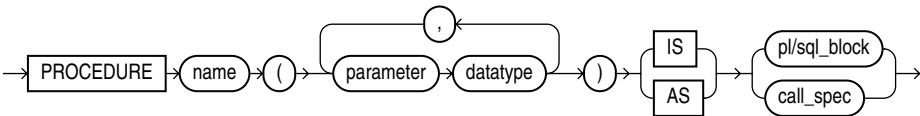


subprogram_declaration::=



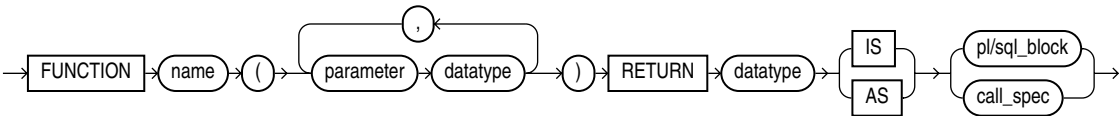
(16-26 ページの「[procedure_declaration::=](#)」、16-26 ページの「[function_declaration::=](#)」、16-26 ページの「[constructor_declaration::=](#)」を参照)

procedure_declaration::=



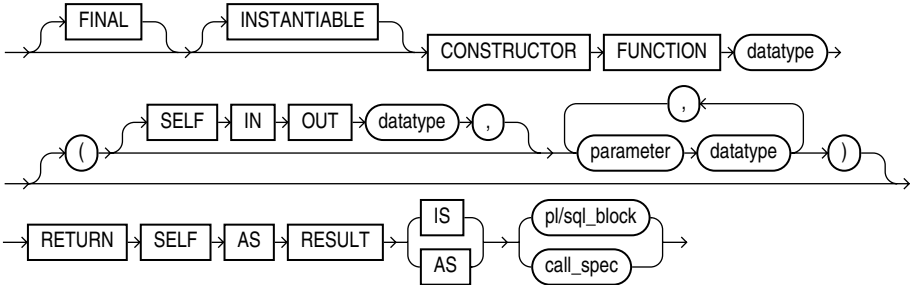
(16-8 ページの「[call_spec::=](#)」を参照)

function_declaration::=

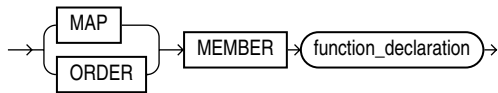


(16-8 ページの「[call_spec::=](#)」を参照)

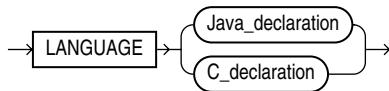
constructor_declaration::=



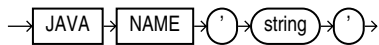
map_order_func_declaration::=



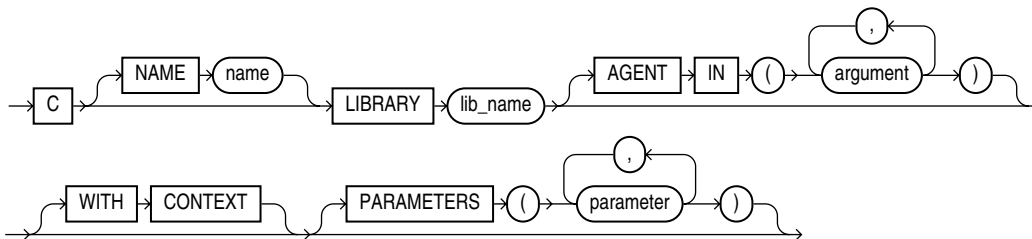
call_spec::=



Java_declaration::=



C_declaration::=



セマンティクス

OR REPLACE

既存の型本体を再作成する場合は、OR REPLACE を指定します。この句を指定した場合、既存の型本体の定義をはじめに削除しなくても、その定義を変更できます。

再作成されたオブジェクト型本体に対する権限を付与されているユーザーは、権限を再付与されなくても、そのオブジェクト型本体を使用および参照できます。

この句を使用した場合、ALTER TYPE ... REPLACE 文によって追加された仕様部に、新規メソッド・サブプログラム定義を追加できます。

schema

作成する型本体が定義されるスキーマを指定します。*schema* を省略した場合、Oracle では現行の自分のスキーマ内に型本体が作成されます。

type_name

オブジェクト型の名前を指定します。

IS | AS**MEMBER | STATIC**

オブジェクト型仕様部に関連付けられたファンクションまたはプロシージャ・サブプログラムの型を指定します。

各プロシージャまたはファンクション宣言には、対応するメソッド名およびオプション・パラメータ・リスト、また、ファンクションの場合は、オブジェクト型仕様部の戻り型を定義する必要があります。

procedure_declaration* および *function_declaration プロシージャまたはファンクション・サブプログラムを宣言します。

constructor_declaration ユーザー定義コンストラクタ・サブプログラムを宣言します。コンストラクタ・ファンクションの RETURN 句は、RETURN SELF AS RESULT である必要があります。この設定は、コンストラクタ・ファンクションによって戻される値で最も指定される型が、コンストラクタ・ファンクションに渡された SELF 引数で最も指定される型と同じであることを示します。

参照：

- ユーザー定義ファンクションの制限事項については、16-3 ページの「[CREATE TYPE](#)」を参照してください。
- パッケージ内のサブプログラム名のオーバーロードについては、『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。
- 型本体の構成要素の詳細は、14-60 ページの「[CREATE PROCEDURE](#)」、13-49 ページの「[CREATE FUNCTION](#)」および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- ユーザー定義コンストラクタの詳細および例は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

pl/sql_block プロシージャまたはファンクションを宣言します。

参照：『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。

call_spec JAVA メソッドまたは C 関数名、パラメータ型および戻り型を SQL にマップするコール仕様を指定します。

Java_declaration では、'string' が JAVA 実装メソッドを定義します。

参照：

- 『Oracle9i Java Stored Procedures Developer's Guide』を参照してください。
- パラメータおよび *C_declaration* のセマンティクスの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

map_order_func_declaration

宣言する MEMBER または STATIC メソッドの数にかかわらず、MAP メソッドまたは ORDER メソッドのいずれかを宣言できます。MAP または ORDER のいずれかのメソッドを宣言すると、SQL 内でオブジェクト・インスタンスを比較できます。

どちらのメソッドも宣言しない場合、比較できるのはオブジェクト・インスタンスの等価性と非等価のみです。同じ型定義のインスタンスは、それぞれの対応する属性の各組が等しい場合にのみ等しくなります。

MAP MEMBER 句

MAP MEMBER を指定すると、メンバー・ファンクション (MAP メソッド) が定義または実装されます。メンバー・ファンクションは、オブジェクトのすべてのインスタンスの順序付けにおいて、指定したインスタンスの相対的な位置を戻します。MAP メソッドは暗黙的にコールされ、オブジェクト・インスタンスを事前定義済のスカラー型の値にマップすることによって、それらのオブジェクト・インスタンスに順序を指定します。PL/SQL は、この順序を使用してブール式の計算と比較を行います。

MAP メソッドの引数が NULL の場合、MAP メソッドは NULL を戻し、メソッドはコールされません。

オブジェクト型本体には、1 つの MAP メソッドのみを指定できます。この MAP メソッドは、ファンクションである必要があります。この MAP ファンクションに指定できる引数は、暗黙的な SELF 引数のみです。

ORDER MEMBER 句

ORDER MEMBER を指定すると、オブジェクトのインスタンスを明示的な引数および暗黙的な SELF 引数として取るメンバー・ファンクション (ORDER メソッド) を指定し、負の整数、0 (ゼロ) または正の整数のいずれかを戻します。負、正または 0 (ゼロ) は、それぞれ、暗黙的な SELF 引数が明示的な引数より小さい、等しいまたは大きいことを示します。

ORDER メソッドのどちらかの引数が NULL の場合は、ORDER メソッドは NULL を戻し、メソッドはコールされません。

同じオブジェクト型定義の各インスタンスを ORDER BY 句の中で比較した場合、ORDER MEMBER *function_declaration* がコールされます。

オブジェクトの仕様部には、1 つの ORDER メソッドのみ指定でき、その ORDER メソッドは戻り型が NUMBER のファンクションである必要があります。

function_declaration ファンクション・サブプログラムを宣言します。

参照： 使用可能なすべての句を含む完全な構文については、14-60 ページの「[CREATE PROCEDURE](#)」および 13-49 ページの「[CREATE FUNCTION](#)」を参照してください。

AS EXTERNAL AS EXTERNAL は、C 関数を宣言するもう 1 つの方法です。この句は以前のリリースのもので、下位互換用にのみサポートされています。 *call_spec* 構文は、*c_declaration* とともに使用することをお勧めします。

例

型本体の作成例は 16-19 ページの「[CREATE TYPE](#)」の「[例](#)」にあります。

型本体の更新例： 次の例は、型に属性が追加された場合の *data_typ* オブジェクト型 (16-19 ページの「[オブジェクト型の例 :](#)」を参照) の型本体の更新方法を示します。

```
ALTER TYPE data_typ
  ADD MEMBER FUNCTION qtr(der_qtr DATE)
  RETURN CHAR CASCADE;

CREATE OR REPLACE TYPE BODY data_typ IS
  MEMBER FUNCTION prod (invent NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN (year + invent);
  END;

  MEMBER FUNCTION qtr(der_qtr DATE) RETURN CHAR IS
  BEGIN
    IF (der_qtr < TO_DATE('01-APR', 'DD-MON')) THEN
      RETURN 'FIRST';
    ELSIF (der_qtr < TO_DATE('01-JUL', 'DD-MON')) THEN
      RETURN 'SECOND';
    ELSIF (der_qtr < TO_DATE('01-OCT', 'DD-MON')) THEN
      RETURN 'THIRD';
    ELSE
      RETURN 'FOURTH';
    END IF;
  END;
END;
/
END;
```

CREATE USER

用途

CREATE USER 文を使用すると、データベース・**ユーザー**（データベースにログイン可能なアカウント）を作成でき、Oracle がそのユーザーによるアクセスを許可する方法が確立されます。

注意： プロキシ（アプリケーションまたはアプリケーション・サーバー）によって、Oracle とユーザーを接続することができます。構文および説明は、12-21 ページの「[ALTER USER](#)」を参照してください。

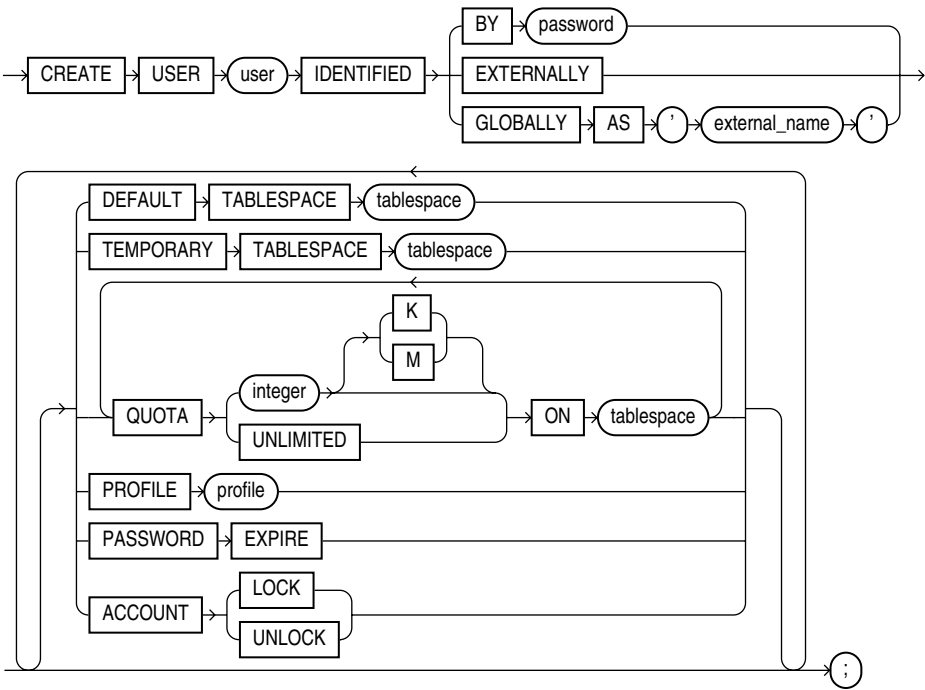
前提条件

CREATE USER システム権限が必要です。CREATE USER 文を使用して作成したユーザーの権限ドメインは空（権限を付与されていない状態）になります。Oracle にログインするユーザーには、CREATE SESSION システム権限が必要です。そのため、ユーザーを作成した場合、必ず CREATE SESSION 権限をそのユーザーに付与してください。

参照： 17-27 ページの「[GRANT](#)」を参照してください。

構文

create_user::=



セマンティクス

user

作成するユーザー名を指定します。この名前には、使用しているデータベース・キャラクタ・セットの文字のみを指定できます。また、2-105 ページの「スキーマ・オブジェクトのネーミング規則」で説明した規則に従う必要があります。データベース・キャラクタ・セットがマルチバイト文字を含んでいても、ユーザー名にはシングルバイト文字を 1 つ以上使用することをお勧めします。

注意： ユーザー名およびパスワードは、ご使用のプラットフォームに応じて、ASCII または EBCDIC 文字のみでエンコードすることをお勧めします。詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

参照： 16-36 ページの「データベース・ユーザーの作成例 :」を参照してください。

IDENTIFIED 句

IDENTIFIED 句によって、ユーザー認証をどのように行うか示します。

BY password

BY *password* を指定すると、**ローカル・ユーザー**を作成し、ログインするときに、パスワードを指定する必要があることを示します。データベース・キャラクタ・セットにマルチバイト文字が含まれている場合でも、パスワードを指定できるのは、データベース・キャラクタ・セットのシングルバイト文字のみです。

Oracle の複雑なパスワード検証ルーチンを使用していない場合、パスワードは、2-105 ページの「スキーマ・オブジェクトのネーミング規則」にある規則に従う必要があります。そのルーチンでは、通常のネーミング規則より複雑な文字の組合せが必要です。このルーチンを UTLPWDMG.SQL スクリプトで実装します。

注意： ユーザー名およびパスワードは、ご使用のプラットフォームに応じて、ASCII または EBCDIC 文字のみでエンコードすることをお勧めします。詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

参照： パスワード管理およびパスワード保護の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

EXTERNALLY 句

EXTERNALLY を指定すると、**外部ユーザー**を作成できます。このユーザーは、外部サービス（オペレーティング・システムまたはサード・パーティのサービス）で認証する必要があります。この場合、オペレーティング・システムのログイン認証によって、特定のオペレーティング・システム・ユーザーから特定のデータベース・ユーザーへのアクセスが可能になります。

注意： ご使用のオペレーティング・システムにログインする際のセキュリティが十分でない場合は、IDENTIFIED EXTERNALLY を使用しないことをお勧めします。詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

参照： 16-36 ページの「外部データベース・ユーザーの作成例 :」を参照してください。

GLOBALLY 句

GLOBALLY 句を使用すると、**グローバル・ユーザー**を作成することができます。このユーザーは、エンタープライズ・ディレクトリ・サービスによって認証される必要があります。`'external_name'` 文字列は、次のいずれかの形式になります。

- このユーザーを識別するエンタープライズ・ディレクトリ・サービスの X.509 の名前。文字列は、`'CN=username,other_attributes'` という形式である必要があります。`other_attributes` は、ディレクトリ内のユーザーの識別名 (DN) 以外の部分です。
- NULL 文字列 ('') は、エンタープライズ・ディレクトリ・サービスが、認証されたグローバル・ユーザーを、適切なデータベース・スキーマに適切なロール付きでマップすることを示します。

注意： 特定のユーザーとして接続し、ALTER USER 文を使用してユーザーのロールをアクティブにするために、アプリケーション・サーバーの機能を制御できます。

参照：

- グローバル・ユーザーの詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。
- 12-21 ページの「[ALTER USER](#)」を参照してください。
- 詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』およびご使用のオペレーティング・システム固有のドキュメントを参照してください。
- 16-37 ページの「[グローバル・データベース・ユーザーの作成例：](#)」を参照してください。

DEFAULT TABLESPACE 句

ユーザーが作成するオブジェクトを格納するデフォルトの表領域を指定します。この句を省略した場合、オブジェクトはデフォルトで SYSTEM 表領域に格納されます。

デフォルトの一時表領域の制限事項： ローカル管理表領域 (UNDO 表領域を含む) またはディクショナリ管理の一時表領域は、ユーザーのデフォルトの表領域として指定できません。

参照： 表領域の概要および UNDO 表領域の詳細は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

TEMPORARY TABLESPACE 句

ユーザーの一時セグメントが確保される表領域を指定します。この句を省略した場合、一時セグメントはデフォルトの `SYSTEM` 表領域に確保されます。

ユーザーの一時表領域の制限事項：

- 表領域は一時表領域で、標準ブロック・サイズである必要があります。
- 表領域は、UNDO 表領域または自動セグメント領域管理の表領域にできません。

参照： UNDO 表領域およびセグメント管理の詳細は、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

QUOTA 句

QUOTA 句を使用すると、表領域の最大 *integer* バイトの領域をユーザーに対して割り当てることができます。K または M を使用して、KB または MB 単位で指定することもできます。この割当て制限は、ユーザーが割当て可能な表領域の最大領域です。

CREATE USER 文では、複数の表領域に対して複数の QUOTA 句を指定できます。

UNLIMITED を使用すると、表領域への領域を無制限に割り当てることができます。

PROFILE 句

ユーザーに割り当てるプロファイルを指定します。このプロファイルによって、ユーザーが使用できるデータベース・リソース容量が制限されます。この句を省略した場合、DEFAULT プロファイルがユーザーに割り当てられます。

参照： 17-27 ページの「[GRANT](#)」および 14-66 ページの「[CREATE PROFILE](#)」を参照してください。

PASSWORD EXPIRE 句

ユーザーのパスワードを期限切れにする場合は、PASSWORD EXPIRE を指定します。この設定によって、ユーザーがデータベースにログインする前に、ユーザー（または DBA）にパスワードを変更させます。

ACCOUNT 句

ACCOUNT LOCK を指定すると、ユーザー・アカウントをロックし、アクセスを禁止にします。ACCOUNT UNLOCK を指定すると、ユーザー・アカウントのロックを解除し、アクセスを可能にします。

例

注意： example 表領域は、シード・データベースに存在し、サンプル・スキーマにアクセス可能であるため、次のすべての例は、この表領域を使用します。

データベース・ユーザーの作成例： PASSWORD EXPIRE 指定して新規ユーザーを作成する場合、そのデータベースにログインする前に、そのユーザーのパスワードを変更する必要があります。次の文は、sidney ユーザーを作成します。

```
CREATE USER sidney
  IDENTIFIED BY welcome
  DEFAULT TABLESPACE example
  QUOTA 10M ON example
  TEMPORARY TABLESPACE temp
  QUOTA 5M ON system
  PROFILE app_user
  PASSWORD EXPIRE;
```

前述のユーザー sidney には次の特性があります。

- パスワード welcome
- 10MB の割当て制限のあるデフォルト表領域 example
- 一時表領域 temp
- 5MB の割当て制限のある表領域 SYSTEM へのアクセス
- プロファイル app_user (14-71 ページの「[プロファイルの作成例](#)」で作成) によって定義されているデータベース・リソースの制限
- sidney がデータベースにログインする前に変更する必要がある期限切れのパスワード

外部データベース・ユーザーの作成例： 次の例は、データベースにアクセスする前に外部ソースによって識別される必要のある、外部ユーザーを作成します。

```
CREATE USER app_user1
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE example
  QUOTA 5M ON example
  PROFILE app_user;
```

ユーザー `app_user1` には次の追加の特性があります。

- デフォルトの表領域 `example`
- デフォルトの一時表領域 `example`
- 表領域 `example` に 5MB の領域、およびデータベースの一時表領域に無制限の割当て
- プロファイル `app_user` によって定義されているデータベース・リソースの制限

オペレーティング・システム・アカウント `app_user2` によってのみアクセス可能な別のユーザーを作成する場合、`app_user2` に初期化パラメータ `OS_AUTHENT_PREFIX` 値の接頭辞を付けます。たとえば、この値が「`ops$`」の場合、次の文でユーザー `ops$app_user2` を作成できます。

```
CREATE USER ops$app_user2
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE example
  QUOTA 5M ON example
  PROFILE app_user;
```

グローバル・データベース・ユーザーの作成例： 次の例は、グローバル・ユーザーを作成します。グローバル・ユーザーを作成する場合、エンタープライズ・ディレクトリ・サーバーでユーザーを識別する X.509 の名前を指定することができます。

```
CREATE USER global_user
  IDENTIFIED GLOBALLY AS 'CN=analyst, OU=division1, O=oracle, C=US'
  DEFAULT TABLESPACE example
  QUOTA 5M ON example;
```

CREATE VIEW

用途

CREATE VIEW 文を使用すると、**ビュー**を定義できます。ビューは論理表で、1 つ以上の表またはビューがベースとなります。ビューにデータそのものが格納されているわけではありません。ビューの基礎になる表を**実表**といいます。

LOB およびオブジェクト・データ型（オブジェクト型、REF、ネストした表または VARRAY 型）をサポートする**オブジェクト・ビュー**またはリレーショナル・ビューを既存のビュー・メカニズムで作成します。オブジェクト・ビューとは、ユーザー定義型のビューのことで、ビューの各行に、それぞれが一意的オブジェクト識別子を持つオブジェクトが含まれます。

XMLType ビューも定義できます。このビューは、オブジェクト・ビューと似ていますが、XMLType の XML Schema ベースの表のデータを表示します。

参照：

- ビューの様々な型およびその使用方法については、『Oracle9i データベース概要』、『Oracle9i アプリケーション開発者ガイド - 基礎編』および『Oracle9i データベース管理者ガイド』を参照してください。
- XMLType ビューの詳細は、『Oracle9i XML データベース開発者ガイド - Oracle XML DB』を参照してください。
- ビューの変更については、12-29 ページの「[ALTER VIEW](#)」を参照してください。
- データベースからのビューの削除については、17-21 ページの「[DROP VIEW](#)」を参照してください。

前提条件

自分のスキーマ内にビューを作成する場合は、CREATE VIEW システム権限が必要です。他のユーザーのスキーマ内にビューを作成する場合は、CREATE ANY VIEW システム権限が必要です。

サブビューを作成する場合は、UNDER ANY TYPE システム権限またはスーパービューに対する UNDER オブジェクト権限が必要です。

ビューを含むスキーマの所有者は、そのビューの基礎となっているすべての表またはビューに対する行の選択、挿入、更新または削除の権限が必要です。また、所有者には、これらの権限がロールを介して付与されるのではなく、直接付与されている必要があります。

オブジェクト・ビューの作成時にオブジェクト型の基本コンストラクタ・メソッドを使用する場合、次のいずれかが真である必要があります。

- オブジェクト型が作成対象のビューと同じスキーマに属している。
- EXECUTE ANY TYPE システム権限がある。
- そのオブジェクト型に対する EXECUTE オブジェクト権限を持っている。

参照： 作成するビューの基礎となる表またはビューで、ビューの所有者に必要な権限の詳細は、18-4 ページの「[SELECT](#)」、17-52 ページの「[INSERT](#)」、18-56 ページの「[UPDATE](#)」および 16-53 ページの「[DELETE](#)」を参照してください。

パーティション・ビュー

パーティション・ビューは、パーティション化機能を必要とするアプリケーション向けに、リリース 7.3 で導入されたものです。Oracle9i では、パーティション・ビューがサポートされているため、何も変更しなくてもリリース 7.3 からのアプリケーションのアップグレードができます。通常、Oracle9i にアップグレードした後で、パーティション・ビューをパーティションに移行します。

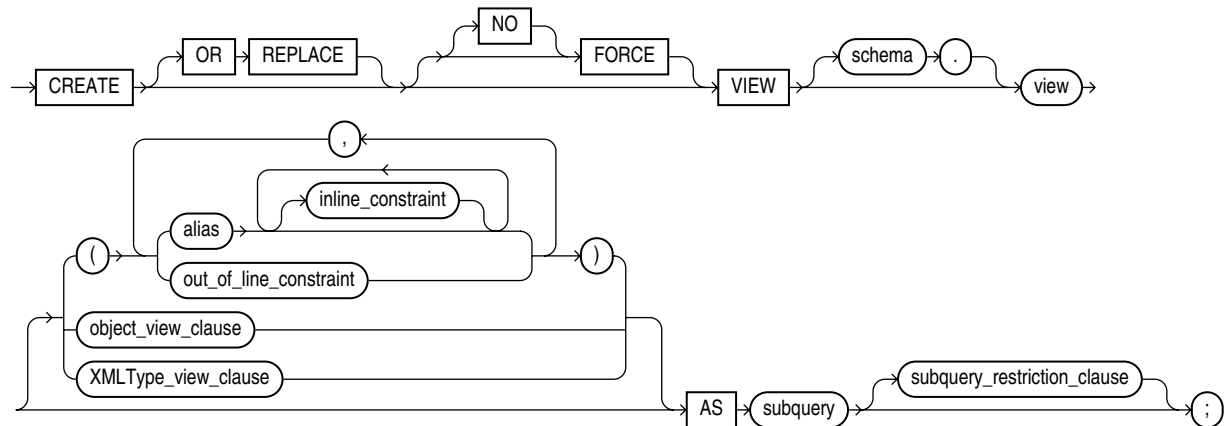
Oracle9i では、CREATE TABLE 文を使用して、パーティション表を簡単に作成できます。パーティション表には、パーティション・ビューと同じメリットがあるのみでなく、パーティション・ビューのデメリットも補います。通常の動作環境では、パーティション・ビューではなく、パーティション表を使用することをお勧めします。

参照：

- パーティション・ビューのデメリットについては、『Oracle9i データベース概要』を参照してください。
- パーティション・ビューのパーティションへの移行については、『Oracle9i データベース管理者ガイド』を参照してください。
- パーティション表の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

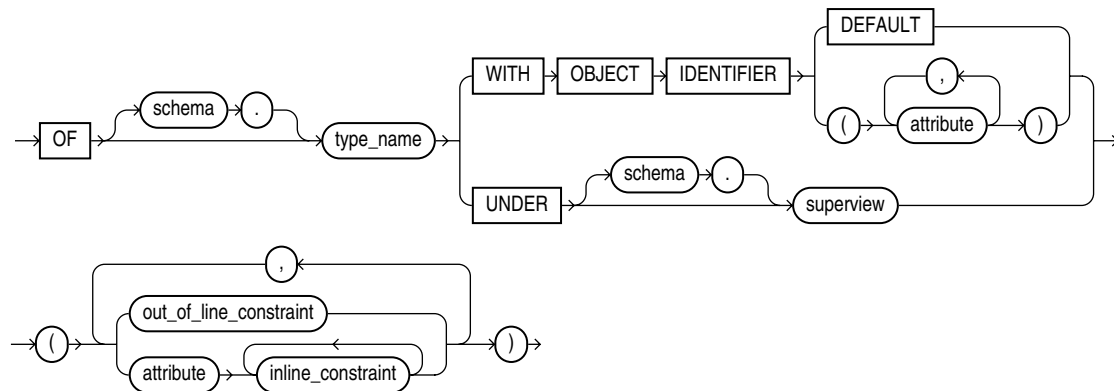
構文

create_view::=

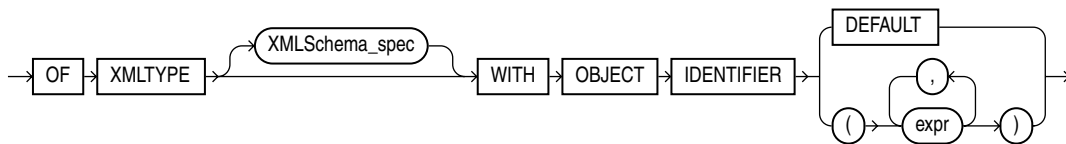
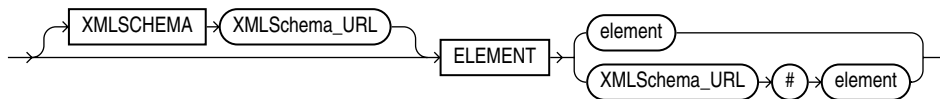
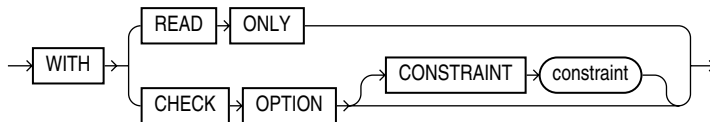


(7-6 ページの「[inline_constraint::=](#)」、7-7 ページの「[out_of_line_constraint::=](#)」([constraints](#) 構文の一部)、16-40 ページの「[object_view_clause::=](#)」、16-41 ページの「[XMLType_view_clause::=](#)」、18-5 ページの「[subquery::=](#)」([SELECT](#) 構文の一部) および 16-41 ページの「[subquery_restriction_clause::=](#)」を参照)

object_view_clause::=



(7-6 ページの「[inline_constraint::=](#)」 および 7-7 ページの「[out_of_line_constraint::=](#)」([constraints](#) 構文の一部) を参照)

XMLType_view_clause::=**XMLSchema_spec::=****subquery_restriction_clause::=****セマンティクス****OR REPLACE**

既存のビューを再作成する場合は、OR REPLACE を指定します。この句を使用した場合、以前に付与されたオブジェクト権限を削除、再作成、再付与しなくても、既存のビュー定義を変更できます。

ビューを再作成した場合、ビュー内で定義された INSTEAD OF トリガーが削除されます。

すべてのマテリアライズド・ビューが view に依存している場合、そのマテリアライズド・ビューには UNUSABLE というマークが付けられ、それらが再利用可能になるようにリストアする場合は、完全なリフレッシュが必要になります。無効なマテリアライズド・ビューはクエリー・リライトで使用されることはなく、また、再コンパイルされるまでリフレッシュされません。

参照：

- 無効なマテリアライズド・ビュー・ログのリフレッシュについては、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- マテリアライズド・ビューの概要は、『Oracle9i データベース概要』を参照してください。
- `INSTEAD OF` 句については、15-89 ページの「[CREATE TRIGGER](#)」を参照してください。

FORCE

ビューの実表または参照先オブジェクト型が存在しているか、またはそのビューを含んでいるスキーマの所有者が、それらの表やオブジェクトに対する権限を持っているかにかかわらず、強制的にビューを作成する場合は、`FORCE` を指定します。`SELECT`、`INSERT`、`UPDATE` または `DELETE` 文をビューに対して発行する場合、前述の条件が真である必要があります。

ビュー定義が制約を含む場合、実表または参照するオブジェクトが存在しないときに `CREATE VIEW ... FORCE` は正常に実行されません。また、ビュー定義が存在しない制約を参照する場合も、`CREATE VIEW ... FORCE` は正常に実行されません。

NO FORCE

実表が存在し、ビューを含むスキーマの所有者がその権限を持っている場合のみビューを作成する場合は、`NOFORCE` を指定します。これはデフォルト値です。

schema

ビューが定義されるスキーマを指定します。*schema* を省略した場合、自分のスキーマにビューが作成されます。

view

ビューまたはオブジェクト・ビューの名前を指定します。

ビューの制限事項： あるビューに `INSTEAD OF` トリガーがある場合、そのビューで作成されるビューが更新可能であっても、そのビューには `INSTEAD OF` トリガーがある必要があります。

参照： 16-49 ページの「[ビューの作成例](#)」を参照してください。

alias

ビューの問合せによって選択された式に対して名前を指定します。別名の数は、ビューによって選択された式の数と一致している必要があります。別名は、次の Oracle スキーマ・オブジェクトのネーミングの規則に従う必要があります。別名は、ビュー内で一意である必要があります。

別名を省略した場合、Oracle はビューの問合せの列または列の別名から別名を導出します。このため、ビューの問合せが列の名前のみでなく式を含んでいる場合は、別名を使用する必要があります。また、ビュー定義が制約を含む場合は、別名を指定する必要があります。

ビューの別名の制限事項： オブジェクト・ビュー作成時に別名は指定できません。

参照： 2-110 ページの「スキーマ・オブジェクトの構文および SQL 文の構成要素」を参照してください。

inline_constraint および out_of_line_constraint

ビューおよびオブジェクト・ビューに制約を指定できます。out_of_line_constraint 句を使用すると、ビュー・レベルで制約を定義できます。適切な別名の後で inline_constraint 句を使用すると、列定義または属性定義の一部として制約を定義できます。

Oracle では、ビュー制約を適用していません。ただし、ビューに対する操作には、基となる実表に定義されている整合性制約が適用されます。つまり、実表に対する制約によって、ビューに制約を適用できます。

ビュー制約の制限事項： ビュー制約は表制約のサブセットで、次の制限事項があります。

- ビューには、一意制約、主キー制約および外部キー制約のみ指定できます。ただし、WITH CHECK OPTION 句を使用してビューを定義することは可能です。これは、ビューに CHECK 制約を指定することと同じです。
- ビュー制約は直接適用されないため、INITIALLY DEFERRED または DEFERRABLE は指定できません。
- ビュー制約は、DISABLE NOVALIDATE モードのみでサポートされています。ビュー制約を宣言する場合は、キーワード DISABLE NOVALIDATE を指定する必要があります、他のモードは指定できません。
- using_index_clause、exceptions_clause 句、または references_clause の ON DELETE 句は指定できません。
- オブジェクト列の属性にビュー制約を定義することはできません。

参照： 制約の概要およびビュー制約の制限については、7-5 ページの「constraints」を参照してください。また、16-49 ページの「制約付きのビューの作成例：」も参照してください。

object_view_clause

`object_view_clause` を指定すると、オブジェクト型にビューを定義できます。

参照： 16-51 ページの「オブジェクト・ビューの作成例：」を参照してください。

OF type_name 句

`type_name` 型のオブジェクト・ビューを明示的に作成します。オブジェクト・ビューの列が、`type_name` 型の最上位属性に対応しています。各行にはオブジェクト・インスタンスが含まれ、また、各インスタンスは、`WITH OBJECT IDENTIFIER` 句で指定したオブジェクト識別子 (OID) に関連付けられます。`schema` を省略した場合、自分のスキーマ内にオブジェクト・ビューが作成されます。

オブジェクト表、XMLType 表、オブジェクト・ビューおよび XMLType ビューには、列名が付けられません。そのため、システム生成列 `SYS_NC_ROWINFO$` が定義されます。問合せでこの列名を使用し、`WITH OBJECT IDENTIFIER` 句を指定して、オブジェクト・ビューを作成できます。

WITH OBJECT IDENTIFIER 句

`WITH OBJECT IDENTIFIER` 句を使用すると、最上位 (ルート) のオブジェクト・ビューを指定できます。オブジェクト・ビュー内の各行を識別するためのキーとして使用されるオブジェクト型の属性を指定します。ほとんどの場合、各属性は実表の主キー列と対応します。属性リストが一意で、ビューの 1 行ずつを識別することを確認する必要があります。

オブジェクト・ビューの制限事項：

- オブジェクト・ビュー内の複数のインスタンスに変換される主キー REF を参照解除または確保しようとした場合、Oracle はエラーを戻します。
- サブビューはスーパービューのオブジェクト識別子を継承するため、サブビューを作成する場合は、この句を指定できません。

注意： Oracle8i リリース 8.0 の構文 `WITH OBJECT OID` は、この構文と置き換えられます。キーワード `WITH OBJECT OID` は下位互換用にサポートされていますが、新しい構文 `WITH OBJECT IDENTIFIER` を使用することをお勧めします。

オブジェクト・ビューがオブジェクト表またはオブジェクト・ビュー上で定義されている場合は、この句を省略するか、または `DEFAULT` を指定します。

DEFAULT 各行を一意に識別するために、基になるオブジェクト表またはオブジェクト・ビュー固有のオブジェクト識別子を使用する場合は、`DEFAULT` を指定します。

attribute *attribute* には、オブジェクト・ビューに対して作成されるオブジェクト識別子の基になるオブジェクト型の属性を指定します。

UNDER 句

UNDER 句を使用すると、オブジェクト・スーパービューに基づくサブビューを指定できます。

ビューがスーパービューまたはサブビューのどちらであるかを確認するには、USER_VIEWS、ALL_VIEWS または DBA_VIEWS データ・ディクショナリ・ビューの SUPERVIEW_NAME を問い合わせます。

サブビューの制限事項：

- サブビューは、スーパービューと同じスキーマに作成する必要があります。
- オブジェクト型 *type_name* は、*superview* 直属のサブタイプである必要があります。
- 同一のスーパービューには、特定の型のサブビューを 1 つのみ作成できます。

参照：

- オブジェクト作成の詳細は、16-3 ページの「[CREATE TYPE](#)」を参照してください。
- データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

AS subquery

ビューの基になっている表（実表）の列と行を識別する副問合せを指定します。副問合せの SELECT 構文のリストには 1000 以内の式を指定できます。

リモート表およびビューを参照するビューを作成する場合、指定するデータベース・リンクを CREATE DATABASE LINK 文の CONNECT TO 句を使用して作成する必要があります。また、ビュー副問合せのスキーマ名で指定する必要があります。

副問合せで *flashback_clause* を使用してビューを作成する場合、AS OF 式は作成時には解析されず、ユーザーがビューの問合せを実行するたびに解析されます。

参照： 16-50 ページの「[結合ビューの作成例：](#)」を参照してください。

参照： フラッシュバック問合せの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

ビュー副問合せの定義の制限事項：

- ビュー副問合せは、CURRVAL および NEXTVAL 疑似列を選択できません。
- ビュー副問合せが ROWID、ROWNUM または LEVEL の各疑似列を選択する場合、そのビュー副問合せには列の別名が必要です。
- ビュー副問合せがアスタリスク (*) を使用して表のすべての列を選択し、後でその表に新しい列を追加する場合、CREATE OR REPLACE VIEW 文を発行してビューを再作成するまで、そのビューにそれらの列は含まれません。
- オブジェクト・ビューの場合、ビュー副問合せの SELECT 構文のリスト内の要素数は、そのオブジェクト型の最上位属性数と同じである必要があります。それぞれの選択要素のデータ型は、対応する最上位属性と同じである必要があります。
- SAMPLE 句は指定できません。

前述の制限は、マテリアライズド・ビューにも適用されます。

更新可能なビューの作成の注意事項：

更新可能なビューとは、実表の行を挿入、更新または削除できるビューです。更新可能なビューを作成するか、またはビューに INSTEAD OF トリガーを作成して更新可能にすることができます。

更新可能なビューの列を更新する方法を確認するには、USER_UPDATABLE_COLUMNS データ・ディクショナリ・ビューを問い合わせます。（このビューで表示される情報は、更新可能なビューのみに意味があります。）

- ビューを固有の特性として更新可能にする場合、次の構造体を指定しないでください。
 - 集合演算子
 - DISTINCT 演算子
 - 集計ファンクションまたは分析ファンクション
 - GROUP BY、ORDER BY、CONNECT BY または START WITH 句
 - SELECT 構文のリストにあるコレクション式
 - SELECT 構文のリストにある副問合せ
 - 結合（ただし、後述のように例外がいくつかあります）
- 更新可能なビューが疑似列または式を含む場合は、これらの疑似列または式を参照する UPDATE 文を使用して、実表の行を更新できません。

- 結合ビューを変更する場合、次の条件が真である必要があります。
 - DML 文は、結合の基になる 1 つの表のみに影響する。
 - INSERT 文の場合、WITH CHECK OPTION 付きでビューを作成できない。また、値を挿入するすべての列は**キー保存表**の列である。実表の各主キーまたは一意キーの値に対するキー保存表は、結合ビューで一意である。
 - UPDATE 文の場合、更新されるすべての列がキー保存表から抽出される。ビューが WITH CHECK OPTION 付きで作成された場合、結合列、およびビュー内で 2 回以上参照される表の列は更新できない。
- DELETE 文の場合、結合により 1 つ以上のキー保存表が作成されると、ビューが WITH CHECK OPTION 付きで作成されたかどうかにかかわらず、FROM 句に指定された最初の表から削除されます。

参照：

- 更新可能なビューの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- オブジェクト型をサポートするオブジェクト・ビューまたはリレーショナル・ビューの更新の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 16-49 ページの「[更新可能なビューの作成例：](#)」を参照してください。
- 更新可能な結合ビューおよびキー保存表の詳細は、16-50 ページの「[結合ビューの作成例：](#)」を参照してください。
- 更新できないビューに対する INSTEAD OF トリガーの例については、15-101 ページの「[INSTEAD OF トリガーの作成例：](#)」を参照してください。

XMLType_view_clause

この句を使用すると、XMLType ビューを作成できます。このビューには、XMLType の XML Schema ベースの表のデータが表示されます。XMLSchema_spec により、XML データを対応するオブジェクト・リレーショナル・データにマップするために使用する XML Schema が指定されます。XML Schema は、XMLType ビューを作成する前に作成しておく必要があります。

オブジェクト表、XMLType 表、オブジェクト・ビューおよび XMLType ビューには、列名が付けられません。そのため、システム生成列 SYS_NC_ROWINFO\$ が定義されます。問合せでこの列名を使用し、WITH OBJECT IDENTIFIER 句を指定して、オブジェクト・ビューを作成できます。

参照：

- XMLType ビューおよび XML Schema の詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。
- 16-52 ページの「XMLType ビューの作成例 :」および 16-51 ページの「XMLType 表のビューの作成例 :」を参照してください。

subquery_restriction_clause

subquery_restriction_clause を使用すると、次のいずれかの方法で、定義する副問合せを制限できます。

WITH READ ONLY 表またはビューが更新禁止であることを示す場合は、WITH READ ONLY を指定します。

WITH CHECK OPTION 副問合せに存在しない行を生成する表またはビューの変更が禁止であることを示す場合は、WITH CHECK OPTION を指定します。

CONSTRAINT *constraint* CHECK OPTION 制約の名前を指定します。この識別子を省略した場合、その制約に SYS_Cn という形式の名前が自動的に割り当てられます。この場合の *n* は、その制約名をデータベース内で一意の名前にする整数です。

WITH CHECK OPTION の注意： 表に WITH CHECK OPTION を指定すると、挿入および更新処理の結果、表を定義する副問合せによって選択可能な表が戻されることが保証されます。次の場合、ビューには WITH CHECK OPTION を指定できません。

- 対象のビュー、または対象のビューの基礎となるビューを定義する副問合せ内に、副問合せが含まれている場合。
 - INSERT、UPDATE または DELETE 操作が INSTEAD OF トリガーを使用して実行された場合。
-
-

参照： 16-51 ページの「読み込み専用ビューの作成例 :」を参照してください。

例

ビューの作成例： 次の文は、サンプル表 `employees` の `emp_view` という名前のビューを作成します。このビューには、部門 20 の従業員とその年間給与が表示されます。

```
CREATE VIEW emp_view AS
  SELECT last_name, salary*12 annual_salary
  FROM employees
  WHERE department_id = 20;
```

この場合、副問合せで式 `salary*12` に列の別名 (`annual_salary`) を使用しているため、この式に基づく列の名前をビュー宣言で定義する必要はありません。

制約付きのビューの作成例： 次の文は、サンプル表 `hr.employees` の制限付きのビューを作成し、ビューの列 `email` に一意制約を定義し、ビューの列 `emp_id` にビューに対する主キー制約を定義します。

```
CREATE VIEW emp_sal (emp_id, last_name,
  email UNIQUE RELY DISABLE NOVALIDATE,
  CONSTRAINT id_pk PRIMARY KEY (emp_id) RELY DISABLE NOVALIDATE)
AS SELECT employee_id, last_name, email FROM employees;
```

更新可能なビューの作成例： 次の文は、`employees` 表内の販売事務員および購買係全員の更新可能なビュー `clerk` を作成します。従業員の ID、姓、部門番号および職種はビューで参照でき、事務長の行のこれらの列のみを更新できます。

```
CREATE VIEW clerk AS
  SELECT employee_id, last_name, department_id, job_id
  FROM employees
  WHERE job_id = 'PU_CLERK'
     or job_id = 'SH_CLERK'
     or job_id = 'ST_CLERK';
```

このビューによって、購買係 `job_id` を購買マネージャ (`PU_MAN`) に変更します。

```
UPDATE clerk SET job_id = 'PU_MAN' WHERE employee_id = 118;
```

次の例は、同じビューを `WITH CHECK OPTION` 付きで作成します。新しい従業員が事務員ではない場合は、`clerk` に新しい行を挿入できません。従業員の `job_id` を他の事務タイプに更新できますが、事務員以外の `job_id` の場合はビューが `employees` にアクセスできないため、正常に更新できません。

```
CREATE VIEW clerk AS
  SELECT employee_id, last_name, department_id, job_id
  FROM employees
  WHERE job_id = 'PU_CLERK'
     or job_id = 'SH_CLERK'
```

```
or job_id = 'ST_CLERK'
WITH CHECK OPTION;
```

結合ビューの作成例： 結合ビューは、結合を含むビューの副問合せです。副問合せの結合において、一意索引を持つ列が1列以上ある場合、結合ビューで1つの実表を変更できます。結合ビューの中の列が更新可能であるかどうかは、USER_UPDATABLE_COLUMNSを問い合わせることでわかります。たとえば、次のようにします。

```
CREATE VIEW locations_view AS
  SELECT d.department_id, d.department_name, l.location_id, l.city
  FROM departments d, locations l
  WHERE d.location_id = l.location_id;

SELECT column_name, updatable
  FROM user_updatable_columns
  WHERE table_name = 'LOCATIONS_VIEW';
```

COLUMN_NAME	UPD
-----	---
DEPARTMENT_ID	YES
DEPARTMENT_NAME	YES
LOCATION_ID	NO
CITY	NO

前述の例では、locations 表の location_id 列に対する主キー索引は、locations_view ビュー内で一意ではありません。そのため、locations はキー保存表ではなく、その実表の列は更新できません。

```
INSERT INTO locations_view VALUES
  (999, 'Entertainment', 87, 'Roma');
INSERT INTO locations_view VALUES
*
ERROR at line 1:
ORA-01776: 結合ビューを介して複数の実表を変更できません。
```

departments 表にマップするビュー内のすべての列が更新可能とマークされていて、departments の主キーがビューに含まれているため、departments 実表に対して、行の挿入、更新または削除ができます。

```
INSERT INTO locations_view (department_id, department_name)
  VALUES (999, 'Entertainment');

1 row created.
```

注意： NOT NULL 列に DEFAULT 値を指定していない場合は、結合内のすべての表のすべての NOT NULL 列が含まれないかぎり、ビューを使用して表に挿入することはできません。

参照： 結合ビューの更新の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

読込み専用ビューの作成例： 次の文は、oe.customers 表の customer_ro という名前の読込み専用ビューを作成します。顧客の姓、言語、掛貸限度額のみがこのビューで参照できます。

```
CREATE VIEW customer_ro (name, language, credit)
AS SELECT cust_last_name, nls_language, credit_limit
FROM customers
WITH READ ONLY;
```

オブジェクト・ビューの作成例： 次の例では、oc スキーマ内での inventory_typ 型の作成方法、およびその型を基にした oc_inventories ビューの作成方法を示します。

```
CREATE TYPE inventory_typ AS OBJECT
( product_id          number(6)
, warehouse           warehouse_typ
, quantity_on_hand    number(8)
) ;

CREATE OR REPLACE VIEW oc_inventories OF inventory_typ
WITH OBJECT IDENTIFIER (product_id)
AS SELECT i.product_id,
       warehouse_typ(w.warehouse_id, w.warehouse_name, w.location_id),
       i.quantity_on_hand
FROM inventories i, warehouses w
WHERE i.warehouse_id=w.warehouse_id;
```

XMLType 表のビューの作成例： 次の例は、XMLType 表 xwarehouses (15-67 ページの「XMLType 表の例：」で作成) の通常のビューを作成します。

```
CREATE VIEW warehouse_view AS
SELECT VALUE(p) AS warehouse_xml
FROM xwarehouses p;
```

このビューからは、次のように選択します。

```
SELECT e.warehouse_xml.getclobval()
FROM warehouse_view e
WHERE EXISTSNODE(warehouse_xml, '//*[Docks]') =1;
```

XMLType ビューの作成例： オブジェクト・リレーショナル表に、XMLType ビューを作成する場合があります。次の例は、オブジェクト・リレーショナル表（サンプル表 `oe.warehouses` 内の XMLType 列 `warehouse_spec` と同様の表）を作成し、その表の XMLType ビューを作成します。

```
CREATE TABLE warehouse_table
(
    WarehouseID      NUMBER,
    Area              NUMBER,
    Docks              NUMBER,
    DockType          VARCHAR2(100),
    WaterAccess        VARCHAR2(10),
    RailAccess         VARCHAR2(10),
    Parking            VARCHAR2(20),
    VClearance         NUMBER
);

INSERT INTO warehouse_table
VALUES(5, 103000,3,'Side Load','false','true','Lot',15);

CREATE VIEW warehouse_view OF XMLTYPE
XMLSCHEMA "http://www.oracle.com/xwarehouses.xsd"
ELEMENT "Warehouse"
WITH OBJECT ID
(extract(sys_nc_rowinfo$, '/Warehouse/Area/text()').getnumberval())
AS SELECT XMLELEMENT("Warehouse",
    XMLFOREST(WarehouseID as "Building",
               area as "Area",
               docks as "Docks",
               docktype as "DockType",
               wateraccess as "WaterAccess",
               railaccess as "RailAccess",
               parking as "Parking",
               VClearance as "VClearance"))
FROM warehouse_table;
```

このビューは、次のように問い合わせます。

```
SELECT VALUE(e) FROM warehouse_view e;
```

DELETE

用途

DELETE を使用すると、表、パーティション表、ビューの実表、またはビューの実表パーティションから行を削除できます。

前提条件

表から行を削除する場合、その表が自分のスキーマ内に存在している必要があります。存在していない場合は、その表の DELETE 権限が必要です。

ビューの実表から行を削除する場合、そのビューを含むスキーマの所有者には、その実表の DELETE 権限が必要です。また、他のスキーマ内にビューが存在している場合は、そのビューの DELETE 権限が必要です。

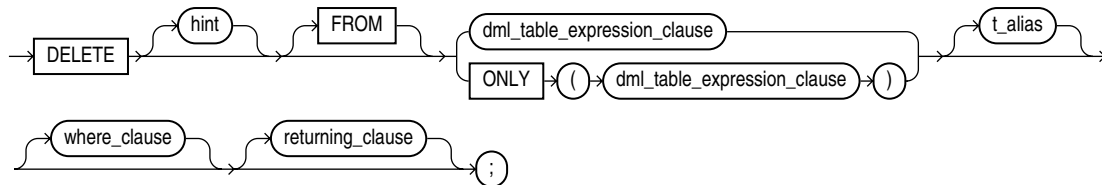
DELETE ANY TABLE システム権限があれば、任意の表、表パーティションまたはビューの実表から行を削除できます。

次の場合は、削除するオブジェクトの SELECT 権限も必要です。

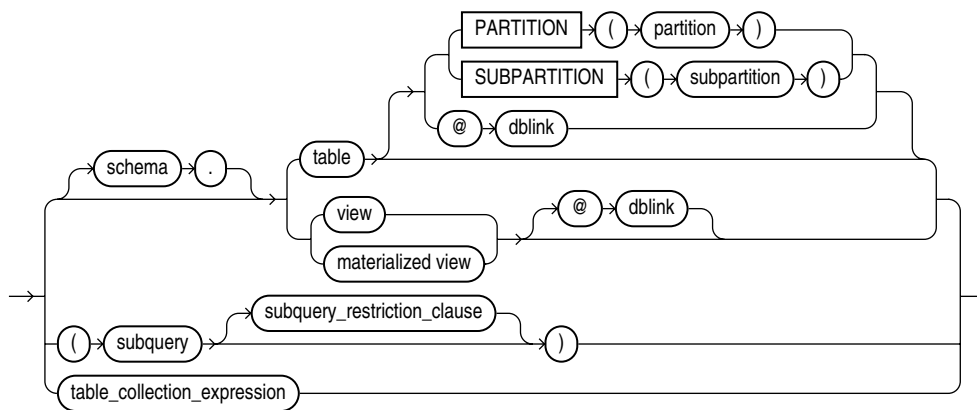
- オブジェクトがリモート・データベースに存在する。
- 初期化パラメータ SQL92_SECURITY が TRUE に設定され、DELETE 操作が表列 (`where_clause` 内の列など) を参照する。

構文

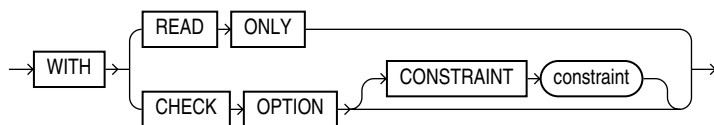
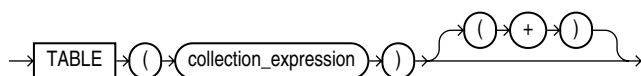
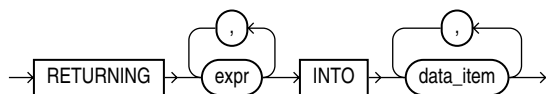
`delete::=`



(16-54 ページの「[DML_table_expression_clause::=](#)」、16-54 ページの「[where_clause::=](#)」、16-54 ページの「[returning_clause::=](#)」を参照)

DML_table_expression::=

(18-5 ページの「[subquery::=](#)」、16-54 ページの「[subquery_restriction_clause::=](#)」、16-54 ページの「[table_collection_expression::=](#)」を参照)

subquery_restriction_clause::=**table_collection_expression::=****where_clause::=****returning_clause::=**

セマンティクス

hint

文の実行計画を選択する場合に、オブティマイザに指示を与えるためのコメントを指定します。

参照： ヒントの構文および説明については、2-86 ページの「[ヒント](#)」および『Oracle9i データベース・パフォーマンス・チューニング・ガイド』およびリファレンス』を参照してください。

from_clause

FROM 句を使用すると、行を削除するデータベース・オブジェクトを指定できます。

ONLY 構文は、ビューのみに関連します。FROM 句のビューがビューの階層に属し、サブビューから行を削除しない場合は、ONLY 句を使用します。

DML_table_expression_clause

schema

表またはビューが含まれているスキーマを指定します。*schema* を指定しない場合、表またはビューが自分のスキーマにあるとみなされます。

table | view | materialized view | subquery

行が削除される表、ビュー、列または副問合せから生成される列の名前を指定します。*view* を指定した場合、ビューの実表から行が削除されます。

表（またはビューの実表）が 1 つ以上のドメイン索引列を含む場合、この文は、適切な索引タイプ削除ルーチンを実行します。

参照： これらのルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

表に対して DELETE 文を実行すると、その表に定義されている DELETE トリガーが起動されます。

行の削除によって解放される表や索引のすべての領域は、表および索引によって引き続き保持されます。

PARTITION (*partition_name*) および SUBPARTITION (*subpartition_name*)

更新対象の表内にあるパーティションまたはサブパーティションの名前を指定します。

パーティション表から値を削除する場合、そのパーティション名を指定する必要はありません。ただし、パーティション名を指定した方が、複雑な *where_clause* より効率が上がる場合があります。

参照： 16-60 ページの「[パーティションの行の削除例:](#)」を参照してください。

dblink

表またはビューが存在しているリモート・データベースとのデータベース・リンクの完全名または部分名を指定します。Oracle の分散機能を使用している場合にのみ、リモート表またはリモート・ビューから行を削除できます。

参照： データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。また、16-60 ページの「[リモート・データベースの行の削除例:](#)」も参照してください。

dblink を省略した場合、その表またはビューはローカル・データベースにあるとみなされます。

subquery_restriction_clause

subquery_restriction_clause を使用すると、次のいずれかの方法で副問合せを制限できます。

WITH READ ONLY 表またはビューが更新禁止であることを示す場合は、WITH READ ONLY を指定します。

WITH CHECK OPTION 副問合せに存在しない行を生成する表またはビューの変更が禁止であることを示す場合は、WITH CHECK OPTION を指定します。

CONSTRAINT *constraint* CHECK OPTION 制約の名前を指定します。この識別子を省略した場合、その制約に SYS_Cn という形式の名前が自動的に割り当てられます。この場合の *n* は、その制約名をデータベース内で一意の名前にする整数です。

参照： 18-32 ページの「[WITH CHECK OPTION 句の使用例:](#)」を参照してください。

table_collection_expression

問合せおよび DML 操作で、`collection_expression` 値を表として扱う場合に、`table_collection_expression` を指定します。`collection_expression` は、副問合せ、列、ファンクションまたはコレクション・コンストラクタのいずれかです。その形式にかかわらず、集合値（ネストした表型または `VARRAY` 型の値）を戻す必要があります。このようなコレクションの要素抽出プロセスを **コレクション・ネスト解除** といいます。

注意： 以前のリリースの Oracle では、`collection_expression` が副問合せの場合、`table_collection_expression` を「`THE subquery`」と表現していました。現在、このような表現方法はされていません。

相関副問合せで `table_collection_expression` を使用して、他の表に存在する値で行を削除します。

参照： 18-36 ページの「[表のコレクション例:](#)」を参照してください。

collection_expression ネストした表の列を `table` または `view` から選択する副問合せを指定します。

注意： 以前のリリースでは、`table_collection_expression` は「`THE subquery`」と表現されていました。現在、このような表現方法はされていません。

dml_table_expression_clause の制限事項：

- `table` (または `view` の実表) に、`IN_PROGRESS` または `FAILED` とマークされたドメイン索引がある場合は、この文は実行できません。
- 関係する索引パーティションが `UNUSABLE` とマークされている場合は、パーティションに挿入できません。
- `dml_table_expression_clause` の副問合せでは、`ORDER BY` 句を指定できません。
- ビューを定義する問合せに次のいずれかの構造体が含まれている場合は、`INSTEAD OF` トリガーを使用する場合を除いて、ビューからの削除はできません。
 - 集合演算子
 - `DISTINCT` 演算子
 - 集計ファンクションまたは分析ファンクション
 - `GROUP BY`、`ORDER BY`、`CONNECT BY` または `START WITH` 句
 - `SELECT` 構文のリストにあるコレクション式

- SELECT 構文のリストにある副問合せ
- 結合（一部の例外を除く）

詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

UNUSABLE のマークが付いている索引、索引パーティションまたは索引サブパーティションを指定する場合、SKIP_UNUSABLE_INDEXES パラメータが `true` に設定されていないかぎり、DELETE 文は正常に実行されません。

参照： 10-2 ページの「[ALTER SESSION](#)」を参照してください。

table_collection_expression

where_clause

where_clause を使用すると、条件を満たす行のみが削除されます。この条件は、表を参照したり、副問合せを含むことができます。Oracle の分散機能を使用している場合にのみ、リモート表またはリモート・ビューから行を削除できます。

参照： 条件の構文については、[第 5 章「条件」](#)を参照してください。

注意： この句がリモート・オブジェクトを参照する *subquery* を含む場合、参照がローカル・データベース上でオブジェクトにループバックしないかぎり、DELETE はパラレルで実行されます。ただし、*dml_table_expression_clause* の *subquery* がリモート・オブジェクトを参照する場合は、UPDATE はシリアルで実行されます。詳細は、15-50 ページの CREATE TABLE の「[parallel_clause](#)」を参照してください。

dblink を省略した場合、その表またはビューはローカル・データベースにあるとみなされます。

where_clause を省略した場合、表またはビューのすべての行が削除されます。

t_alias 文中で参照する表、ビュー、副問合せまたはコレクション値の**相関名**です。通常、別名は相関問合せを持つ DELETE 文で使用されます。

注意： *dml_table_expression_clause* がいずれかのオブジェクト型属性またはオブジェクト型メソッドを参照する場合、この別名が必要です。

returning_clause

DML (INSERT、UPDATE または DELETE) 文に影響される行を取り出します。この句は、表、マテリアライズド・ビュー、および単一の実表を持つビューに指定できます。

単一行で処理する場合、*returning_clause* 付きの DML 文で、処理された行および ROWID を使用している列式、処理された行への REF を取り出し、ホスト変数または PL/SQL 変数に格納します。

複数行で処理する場合、*returning_clause* 付きの DML 文で、式の値、ROWID および処理された行に関連する REF をバインド配列に格納します。

expr *expr* リストの各項目は、適切な構文で表す必要があります。スカラー副問合せ式を除くすべての形式が有効です。

INTO 変更された行の値を、*data_item* リストに指定する変数に格納することを示します。

data_item 取り出された *expr* 値を格納するホスト変数または PL/SQL 変数です。

RETURNING リストの各式については、INTO リストに、対応する型に互換がある PL/SQL 変数またはホスト変数を指定する必要があります。

RETURNING 句の制限事項： 次の操作は実行できません。

- マルチテーブル・インサートへの *returning_clause* の指定。
- パラレル DML またはリモート・オブジェクトでのこの句の使用。
- この句での LONG 型の取出し。
- INSTEAD OF トリガーが定義されたビューに対しての、この句の指定。

参照： BULK COLLECT 句を使用してコレクション変数に複数の値を戻す場合は、『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。

注意： この句を使用すると、削除された列から値を戻すことができるため、DELETE 文の後で SELECT を実行する必要がなくなります。

参照： 16-60 ページの「[RETURNING 句の使用例:](#)」を参照してください。

例

行の削除例： 次の文は、サンプル表 `oe.product_descriptions` からすべての行を削除します。

```
DELETE FROM product_descriptions;
```

次の文は、サンプル表 `hr.employees` から手数料率が 10% 未満の購買係を削除します。

```
DELETE FROM employees
  WHERE job_id = 'PU_CLERK'
  AND commission_pct < .1;
```

次の文は前述の例と同じ効果がありますが、副問合せを使用します。

```
DELETE FROM (SELECT * FROM employees)
  WHERE job_id = 'PU_CLERK'
  AND commission_pct < .1;
```

リモート・データベースの行の削除例： 次の文は、データベース・リンク `remote` によってアクセス可能なデータベースの、ユーザー `hr` が所有する `locations` 表から指定した行を削除します。

```
DELETE FROM hr.locations@remote
  WHERE location_id > 3000;
```

ネストした表の行の削除例： 次の文は、ネストした表 `projs` の行のうち、部門番号が 123 か 456 のいずれかであるもの、または部門の予算が 456.78 を超えるものを削除します。

```
DELETE TABLE(SELECT projs FROM dept d WHERE d.dno = 123) p
  WHERE p.pno IN (123, 456) OR p.budgets > 456.78;
```

パーティションの行の削除例： 次の例は、`sh.sales` 表のパーティション `sales_q1_1998` から行を削除します。

```
DELETE FROM sales PARTITION (sales_q1_1998)
  WHERE amount_sold > 10000;
```

RETURNING 句の使用例： 次の例は、削除された行から `salary` 列を戻し、その結果をバインド配列 `:bnd1` に格納します。(バインド配列は事前に宣言しておく必要があります。)

```
DELETE FROM employees
  WHERE job_id = 'SA_REP'
  AND hire_date + TO_YMINTERVAL('01-00') < SYSDATE;
  RETURNING salary INTO :bnd1;
```

DISASSOCIATE STATISTICS

用途

DISASSOCIATE STATISTICS 文を使用すると、統計タイプ（またはデフォルト統計）と、列、スタンドアロン・ファンクション、パッケージ、型、ドメイン索引または索引タイプとの関連付けを解除できます。

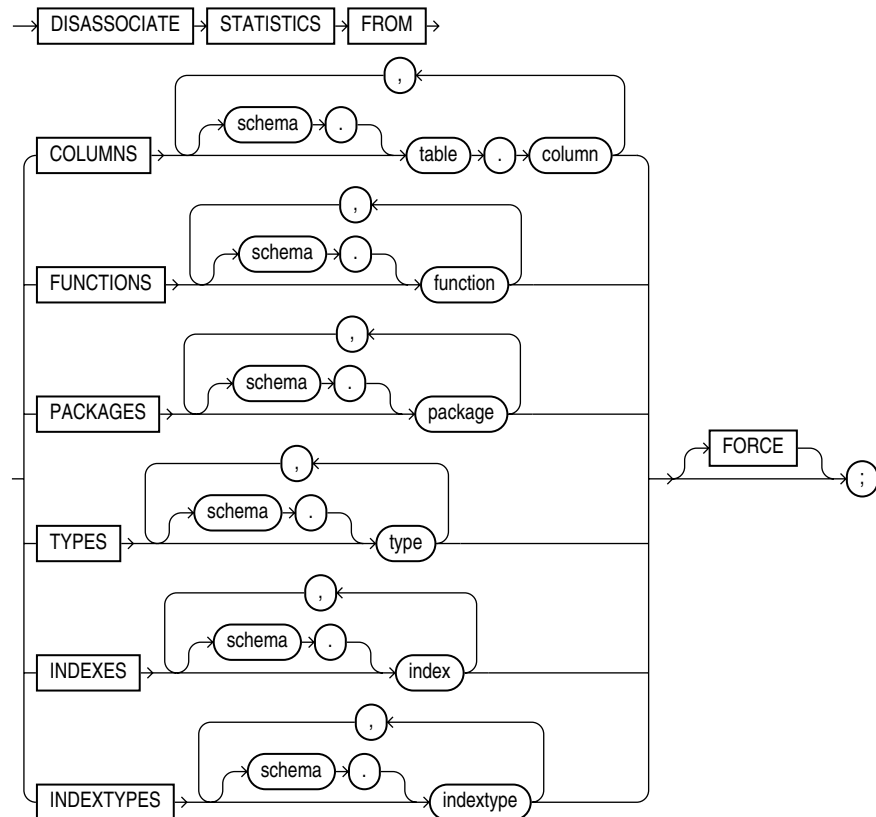
参照： 統計タイプの関連付けの詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」を参照してください。

前提条件

この文を発行する場合は、ベース・オブジェクト（表、ファンクション、パッケージ、型、ドメイン索引または索引タイプ）を変更する適切な権限が必要です。

構文

disassociate_statistics::=



セマンティクス

FROM COLUMNS | FUNCTIONS | PACKAGES | TYPES | INDEXES | INDEXTYPES

統計との関連付けを解除する 1 つ以上の列、スタンドアロン・ファンクション、パッケージ、型、ドメイン索引または索引タイプを指定します。

schema を指定しない場合、オブジェクトは自分のスキーマ内にあるとみなされます。

オブジェクトのユーザー定義の統計を収集する場合、FORCE を指定しないかぎり文は実行されません。

FORCE

FORCE を指定すると、統計タイプを使用するオブジェクトでの統計の有無にかかわらず、関連付けを解除します。統計が存在する場合、関連付けが解除される前に統計は削除されます。

注意： 統計タイプが関連付けられているオブジェクトを削除する場合、FORCE オプションを持つ統計タイプの関連付けは自動的に解除され、統計タイプで収集したすべての統計は削除されます。

例

統計の関連付けの解除例： 次の文は、hr スキーマの emp_mgmt パッケージと統計との関連付けを解除します。

```
DISASSOCIATE STATISTICS FROM PACKAGES hr.emp_mgmt;
```

DROP CLUSTER

用途

- DROP CLUSTER 文を使用すると、データベースからクラスタを削除できます。
- 個々の表は非クラスタ化できません。そのかわり、次の手順を実行します。
1. 既存の表と同じ構成および内容で、新しい表を CLUSTER 句なしで作成します。
 2. 既存の表を削除します。
 3. RENAME 文を使用して、新しい表に既存の表の名前を指定します。
 4. 既存のクラスタ化表に対して付与された権限は適用されないため、新しく作成した非クラスタ化表に権限を付与します。

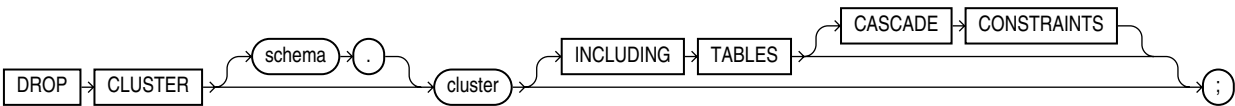
参照： これらの手順の詳細は、15-7 ページの「[CREATE TABLE](#)」、17-6 ページの「[DROP TABLE](#)」、17-84 ページの「[RENAME](#)」および 17-27 ページの「[GRANT](#)」を参照してください。

前提条件

削除するクラスタが自分のスキーマ内にあるか、または DROP ANY CLUSTER システム権限が必要です。

構文

drop_cluster::=



セマンティクス

schema

クラスタが含まれているスキーマを指定します。schema を指定しない場合、そのクラスタが自分のスキーマにあるとみなされます。

cluster

削除するクラスタの名前を指定します。クラスタを削除するとクラスタの索引も削除され、その索引のデータ・ブロックを含むクラスタ領域が表領域に戻されます。

INCLUDING TABLES

INCLUDING TABLES を指定すると、クラスタに属するすべての表が削除されます。

CASCADE CONSTRAINTS

CASCADE CONSTRAINTS を指定すると、クラスタに含まれる表の主キーまたは一意キーを参照するクラスタ外の表にあるすべての参照整合性制約が削除されます。このような参照整合性制約がある場合にこの句の指定を省略した場合、エラー・メッセージが表示され、クラスタは削除されません。

例

クラスタの削除例： 次の文は、クラスタ（13-9 ページの「CREATE CLUSTER」の「例」で作成）を変更します。

次の文は、language クラスタを削除します。

```
DROP CLUSTER language;
```

次の文は、dept_10、dept_20 およびこれらの表の主キーまたは一意キーを参照する参照整合性制約と同様に personnel クラスタを削除します。

```
DROP CLUSTER personnel
    INCLUDING TABLES
    CASCADE CONSTRAINTS;
```

DROP CONTEXT

用途

DROP CONTEXT 文を使用すると、データベースからコンテキスト・ネームスペースを削除できます。

注意： コンテキスト・ネームスペースを削除した場合でも、ユーザー・セッションに設定されたネームスペースのコンテキストは無効になりません。ただし、ユーザーがそのコンテキストを設定しようとした場合、そのコンテキストは無効になります。

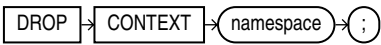
参照： コンテキストの詳細は、13-11 ページの「[CREATE CONTEXT](#)」および『Oracle9i データベース概要』を参照してください。

前提条件

DROP ANY CONTEXT システム権限が必要です。

構文

drop_context::=



セマンティクス

namespace

削除するコンテキスト・ネームスペースを指定します。組込みネームスペース USERENV は削除できません。

参照： USERENV の詳細は、6-148 ページの「[SYS_CONTEXT](#)」を参照してください。

例

アプリケーション・コンテキストの削除例： 次の文は、コンテキスト（13-11 ページの「[CREATE CONTEXT](#)」で作成）を削除します。

```
DROP CONTEXT hr_context;
```


DROP DATABASE LINK

用途

DROP DATABASE LINK 文を使用すると、データベースからデータベース・リンクを削除できます。

参照： データベース・リンクの作成方法については、13-35 ページの「[CREATE DATABASE LINK](#)」を参照してください。

前提条件

プライベート・データベース・リンクは自分のスキーマにある必要があります。パブリック・データベース・リンクを削除する場合は、DROP PUBLIC DATABASE LINK システム権限が必要です。

構文

drop_database_link::=



セマンティクス

PUBLIC

パブリック・データベース・リンクを削除する場合は、PUBLIC を指定する必要があります。

dblink

削除するデータベース・リンクの名前を指定します。

データベース・リンクの削除の制限事項： 他のユーザーのスキーマ内のデータベース・リンクは削除できません。また、データベース・リンク名ではピリオドが許可されるため、スキーマ名で *dblink* を修飾することはできません。そのため、たとえば、*ralph.linktosales* のような名前を付けた場合、スキーマ *ralph* の中の *linktosales* という名前のデータベース・リンクであるとは見なされず、名前全体が自分のスキーマにあるデータベース・リンク名であると解析されます。

例

データベース・リンクの削除例： 次の文は、パブリック・データベース・リンク remote (13-39 ページの「[パブリック・データベース・リンクの定義例:](#)」で作成) を削除します。

```
DROP PUBLIC DATABASE LINK remote;
```

DROP DIMENSION

用途

DROP DIMENSION を使用すると、名前付けしたディメンションを削除できます。

注意： この文によって、ディメンションに指定された関連を使用するマテリアライズド・ビューが無効になるわけではありません。ただし、クエリー・リライトによって再書き込みされた要求が無効になり、そのようなビューに対する後続の操作の処理速度が遅くなることがあります。

参照：

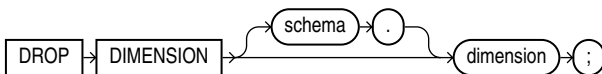
- ディメンションの作成については、13-41 ページの「[CREATE DIMENSION](#)」を参照してください。
- ディメンションの変更については、9-56 ページの「[ALTER DIMENSION](#)」を参照してください。
- 『Oracle9i データベース概要』を参照してください。

前提条件

この文を使用する場合、ディメンションが自分のスキーマ内に設定されているか、または DROP ANY DIMENSION システム権限が必要です。

構文

drop_dimension::=



セマンティクス

schema

ディメンションが格納されているスキーマの名前を指定します。*schema* を指定しない場合、そのディメンションは自分のスキーマにあるとみなされます。

dimension

削除するディメンションの名前を指定します。そのディメンションはすでに存在している必要があります。

例

ディメンションの削除例： この例は、sh.customers_dim を削除します。

```
DROP DIMENSION customers_dim;
```

参照： ディメンションの作成および変更の例については、13-45 ページの「[ディメンションの作成例 :](#)」および 9-58 ページの「[ディメンションの変更例 :](#)」を参照してください。

DROP DIRECTORY

用途

DROP DIRECTORY 文を使用すると、データベースからディレクトリ・オブジェクトを削除できます。

参照： ディレクトリの作成については、13-46 ページの「[CREATE DIRECTORY](#)」を参照してください。

前提条件

ディレクトリを削除する場合は、DROP ANY DIRECTORY システム権限が必要です。

注意： PL/SQL または OCI プログラムによる関連ファイル・システム内のファイルへのアクセス中は、DROP によるディレクトリの削除はできません。

構文

drop_directory::=

DROP → DIRECTORY → (directory_name) → ;

セマンティクス

directory_name

削除するディレクトリ・データベース・オブジェクトの名前を指定します。

Oracle はディレクトリ・オブジェクトを削除しますが、サーバーのファイル・システム上の関連するオペレーティング・システム・ディレクトリは削除しません。

例

ディレクトリの削除例： 次の文は、ディレクトリ・オブジェクト bfile_dir を削除します。

```
DROP DIRECTORY bfile_dir;
```

参照： 13-48 ページの「[ディレクトリの作成例:](#)」を参照してください。

DROP FUNCTION

用途

DROP FUNCTION 文を使用すると、データベースからスタンドアロン・ストアド・ファンクションを削除できます。

注意： この文を使用してパッケージの一部のファンクションを削除しないでください。DROP PACKAGE 文を使用して完全なパッケージを削除するか、OR REPLACE 句付きの CREATE PACKAGE 文を使用するファンクションを含まないパッケージを再定義します。

参照：

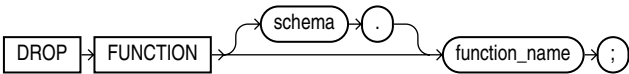
- ファンクションの作成については、13-49 ページの「[CREATE FUNCTION](#)」を参照してください。
- ファンクションの変更については、9-59 ページの「[ALTER FUNCTION](#)」を参照してください。

前提条件

削除するファンクションが自分のスキーマ内にあるか、または DROP ANY PROCEDURE システム権限が必要です。

構文

drop_function::=



セマンティクス

schema

ファンクションが含まれているスキーマを指定します。schema を指定しない場合、ファンクションが自分のスキーマ内に定義されているものとみなされます。

function_name

削除するファンクションの名前を指定します。

Oracle は、削除したファンクションに依存するローカル・オブジェクト、または削除したファンクションをコールするローカル・オブジェクトを無効にします。後でこのようなオブジェクトを参照した場合、Oracle は、そのオブジェクトを再コンパイルしようとします。削除したファンクションを再作成しないかぎり、エラーが戻されます。

任意の統計タイプがファンクションに関連付けられている場合、FORCE 句を使用して統計タイプの関連付けが解除され、統計タイプを使用して収集されたユーザー定義のすべての統計が削除されます。

参照：

- リモート・オブジェクトなどのスキーマ・オブジェクト間の依存関係を Oracle が管理する方法の詳細は、『Oracle9i データベース概要』を参照してください。
- 統計タイプの関連の詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

例

ファンクションの削除例： 次の文は、サンプル・スキーマ oe 内のファンクション SecondMax を削除します。この場合、SecondMax に依存するすべてのオブジェクトは無効になります。

```
DROP FUNCTION oe.SecondMax;
```

参照： SecondMax ファンクションの作成の詳細は、13-58 ページの「[ファンクションの作成例：](#)」を参照してください。

DROP INDEX

用途

DROP INDEX 文を使用すると、データベースから索引またはドメイン索引を削除できます。

索引を削除すると、ビュー、パッケージ、パッケージ本体、ファンクション、プロシージャなど、基になる表に依存するすべてのオブジェクトが削除されます。

グローバル・パーティション索引、レンジ・パーティション索引またはハッシュ・パーティション索引を削除した場合、すべての索引パーティションが同様に削除されます。コンポジット・パーティション索引を削除した場合、すべての索引パーティションおよびサブパーティションが同様に削除されます。

また、ドメイン索引を削除すると、次のようになります。

- 適切なルーチンが起動されます。これらのルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- 統計タイプがドメイン索引に関連付けられている場合、FORCE 句を使用して統計タイプの関連付けが解除され、統計タイプを使用して収集されたユーザー定義の統計が削除されます。

参照：

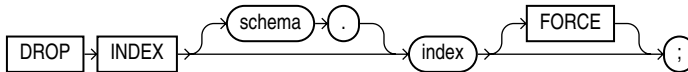
- 索引の作成については、13-61 ページの「[CREATE INDEX](#)」を参照してください。
- 索引の変更については、9-61 ページの「[ALTER INDEX](#)」を参照してください。
- ドメイン索引の詳細は、13-61 ページの「[CREATE INDEX](#)」の「`domain_index_clause`」を参照してください。
- 統計タイプの関連の詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

前提条件

索引が自分のスキーマ内にあるか、または DROP ANY INDEX システム権限が必要です。

構文

drop_index::=



セマンティクス

schema

削除する索引が含まれているスキーマを指定します。*schema* を指定しない場合、索引が自分のスキーマ内に定義されているものとみなされます。

index

削除する索引の名前を指定します。索引を削除した場合、その索引に割り当てられていたすべてのデータ・ブロックが索引の表領域に戻されます。

索引の削除の制限事項： 索引または索引パーティションのいずれかに `IN_PROGRESS` のマークが付けられている場合は、ドメイン索引を削除できません。

FORCE

`FORCE` は、ドメイン索引のみに適用されます。この句を指定すると、索引タイプ・ルーチンの起動がエラーを戻した場合、または索引に `IN_PROGRESS` のマークが付けられている場合でも、ドメイン索引を削除できます。`FORCE` がないと、その索引タイプ・ルーチンの起動がエラーを戻す場合、または索引に `IN_PROGRESS` マークが付けられている場合にドメイン索引を削除できません。

例

索引の削除例： この文は、索引 `ord_customer_ix_demo` (13-81 ページの「[一般的な索引の例](#)」で作成) を削除します。

```
DROP INDEX ord_customer_ix_demo;
```

DROP INDEXTYPE

用途

DROP INDEXTYPE 文を使用すると、索引タイプを削除できます。同時に、統計タイプとの関連付けも解除されます。

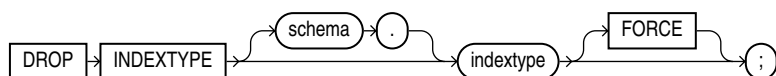
参照： 索引タイプの詳細は、13-88 ページの「[CREATE INDEXTYPE](#)」を参照してください。

前提条件

索引タイプが自分のスキーマ内にあるか、または DROP ANY INDEXTYPE システム権限が必要です。

構文

drop_indextype::=



セマンティクス

schema

索引タイプが含まれているスキーマを指定します。*schema* を指定しない場合、索引タイプが自分のスキーマ内に定義されているものとみなされます。

indextype

削除する索引タイプの名前を指定します。

任意の統計タイプが索引タイプに関連付けられている場合、統計タイプと索引タイプの関連付けが解除され、統計タイプを使用して収集されたすべての統計が削除されます。

参照： 統計情報の関連付けの詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

FORCE

FORCE を指定すると、索引タイプが 1 つ以上のドメイン索引から現在参照されている状態でも、索引タイプを削除できます。これらのドメイン索引に `INVALID` のマークが付けられます。FORCE を指定しない場合、任意のドメイン索引が索引タイプを参照していると索引タイプを削除できません。

例

索引タイプの削除例： 次の文は、索引タイプ `textindextype` を削除し、この索引タイプで定義されているすべてのドメイン索引に `INVALID` のマークを付けます。

```
DROP INDEXTYPE textindextype FORCE;
```

DROP JAVA

用途

DROP JAVA 文を使用すると、Java ソース、クラスまたはリソース・スキーマ・オブジェクトを削除できます。

参照：

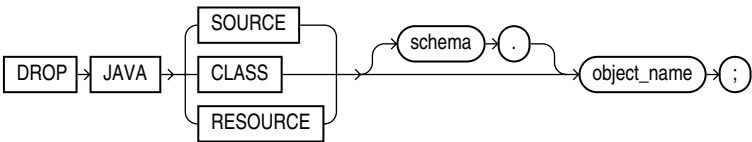
- Java オブジェクトの作成については、13-90 ページの「[CREATE JAVA](#)」を参照してください。
- Java ソース、クラスおよびリソースの変換の詳細は、『Oracle9i Java Stored Procedures Developer’s Guide』を参照してください。

前提条件

Java ソース、クラスまたはリソースが自分のスキーマ内にあるか、または DROP ANY PROCEDURE システム権限が必要です。また、このコマンドを使用する場合は、Java クラスに対する EXECUTE オブジェクト権限が必要です。

構文

drop java::=



セマンティクス

JAVA SOURCE

SOURCE を指定すると、Java ソース・スキーマ・オブジェクトおよびそのオブジェクトから導出されたすべての Java クラス・スキーマ・オブジェクトが削除されます。

JAVA CLASS

CLASS を指定すると、Java クラス・スキーマ・オブジェクトが削除されます。

JAVA RESOURCE

RESOURCE を指定すると、Java リソース・スキーマ・オブジェクトが削除されます。

object_name

既存の Java クラス、ソースまたはリソース・スキーマ・オブジェクトの名前を指定します。
object_name を二重引用符で囲むと、小文字の名前、または大文字と小文字を組み合わせた名前を指定できます。

例

Java クラス・オブジェクトの削除例： 次の文は、Java クラス MyClass を削除します。

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

DROP LIBRARY

用途

DROP LIBRARY を使用すると、データベースから外部プロシージャ・ライブラリを削除できます。

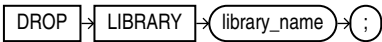
参照： ライブラリ作成については、14-2 ページの「[CREATE LIBRARY](#)」を参照してください。

前提条件

DROP ANY LIBRARY システム権限が必要です。

構文

drop_library::=



セマンティクス

library_name

削除対象の外部プロシージャ・ライブラリの名前を指定します。

例

ライブラリの削除例： 次の文は、ext_lib ライブラリ（14-3 ページの「[ライブラリの作成例](#)」で作成）を削除します。

```
DROP LIBRARY ext_lib;
```

DROP MATERIALIZED VIEW

用途

DROP MATERIALIZED VIEW 文を使用すると、データベースから既存のマテリアライズド・ビューを削除できます。

注意： キーワード **SNAPSHOT** は、**MATERIALIZED VIEW** のかわりに下位互換用にサポートされています。

参照：

- 種類の説明を含むマテリアライズド・ビューの詳細は、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。
- マテリアライズド・ビューの変更については、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- レプリケーション環境でのマテリアライズド・ビューについては、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウス環境でのマテリアライズド・ビューの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

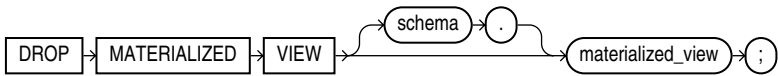
前提条件

マテリアライズド・ビューが自分のスキーマ内にあるか、または DROP ANY MATERIALIZED VIEW システム権限が必要です。また、Oracle がマテリアライズド・ビューのデータを管理するために使用する内部表、ビュー、索引を削除する権限も必要です。

参照： マテリアライズド・ビューを管理するために使用するオブジェクトの削除に必要な権限については、17-6 ページの「[DROP TABLE](#)」、17-21 ページの「[DROP VIEW](#)」および 16-74 ページの「[DROP INDEX](#)」を参照してください。

構文

drop_materialized_view::=



セマンティクス

schema

マテリアライズド・ビューが含まれているスキーマを指定します。*schema* を指定しない場合、マテリアライズド・ビューは自分のスキーマ内にあるとみなされます。

materialized_view

削除する既存のマテリアライズド・ビューの名前を指定します。

- マスター表から、最後にリフレッシュされた単純マテリアライズド・ビューを削除した場合、マスター表のマテリアライズド・ビュー・ログのうち、削除されたマテリアライズド・ビューのリフレッシュに必要な行のみが自動的に削除されます。
- マスター表を削除した場合、Oracle は、その表に基づくマテリアライズド・ビューを自動的に削除しません。ただし、削除したマスター表に基づくマテリアライズド・ビューをリフレッシュしようとした場合、エラー・メッセージが戻されます。
- マテリアライズド・ビューを削除した場合、マテリアライズド・ビューを使用するために再書き込みされたコンパイル済の要求が無効になり、自動的に再コンパイルされます。事前にマテリアライズド・ビューが作成されている場合、その表は削除されませんが、マテリアライズド・ビューのリフレッシュ・メカニズムによる管理ができなくなります。

例

マテリアライズド・ビューの削除例： 次の文は、サンプル・スキーマ hr のマテリアライズド・ビュー emp_data を削除します。

```
DROP MATERIALIZED VIEW emp_data;
```

次の文は、sales_by_month_by_state マテリアライズド・ビューおよびマテリアライズド・ビューの基礎となる表を削除します（基礎となる表が ON PREBUILT TABLE 句付きで CREATE MATERIALIZED VIEW 文に登録されていない場合）。

```
DROP MATERIALIZED VIEW sales_by_month_by_state;
```


DROP MATERIALIZED VIEW LOG

用途

DROP MATERIALIZED VIEW LOG 文を使用すると、データベースからマテリアライズド・ビュー・ログを削除できます。

注意： キーワード **SNAPSHOT** は、**MATERIALIZED VIEW** のかわりに下位互換用にサポートされています。

参照：

- マテリアライズド・ビューについては、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」および 9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。
- マテリアライズド・ビュー・ログについては、14-31 ページの「[CREATE MATERIALIZED VIEW LOG](#)」を参照してください。
- レプリケーション環境でのマテリアライズド・ビューについては、『Oracle9i アドバンスド・レプリケーション』を参照してください。
- データ・ウェアハウス環境でのマテリアライズド・ビューについては、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

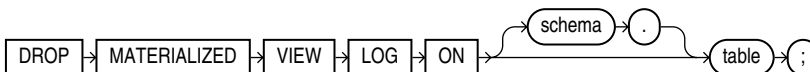
前提条件

マテリアライズド・ビュー・ログを削除する場合は、表を削除する権限が必要です。

参照： 17-6 ページの「[DROP TABLE](#)」を参照してください。

構文

drop_materialized_view_log::=



セマンティクス

schema

削除するマテリアライズド・ビュー・ログおよびそのマスター表が含まれているスキーマを指定します。*schema* を指定しない場合、マテリアライズド・ビュー・ログおよびマスター表は自分のスキーマ内にあるとみなされます。

table

削除するマテリアライズド・ビュー・ログに関連付けられたマスター表の名前を指定します。

マテリアライズド・ビュー・ログを削除した場合、マテリアライズド・ビュー・ログのマスター表に基づく一部のマテリアライズド・ビューが高速リフレッシュできなくなります。これらのマテリアライズド・ビューは、ROWID マテリアライズド・ビュー、主キー・マテリアライズド・ビューおよび副問合せビューを含みます。

参照： マテリアライズド・ビューの種類については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

例

マテリアライズド・ビュー・ログの削除例： 次の文は、oe.customers マスター表のマテリアライズド・ビュー・ログを削除します。

```
DROP MATERIALIZED VIEW LOG ON customers;
```

DROP OPERATOR

用途

DROP OPERATOR 文を使用すると、ユーザー定義演算子を削除できます。

参照：

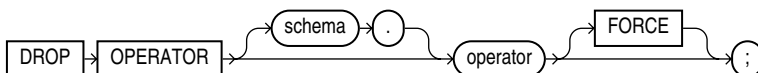
- 演算子の作成については、14-40 ページの「[CREATE OPERATOR](#)」を参照してください。
- 演算子の概要については、3-6 ページの「[ユーザー定義演算子](#)」および『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- ユーザー定義索引タイプの演算子の削除については、9-83 ページの「[ALTER INDEX TYPE](#)」を参照してください。

前提条件

演算子が自分のスキーマ内にあるか、または DROP ANY OPERATOR システム権限が必要です。

構文

drop_operator::=



セマンティクス

schema

演算子が含まれているスキーマを指定します。*schema* を指定しない場合、その演算子が自分のスキーマにあるとみなされます。

operator

削除する演算子の名前を指定します。

FORCE

FORCE を指定すると、演算子が現在 1 つ以上のスキーマ・オブジェクト（索引タイプ、パッケージ、ファンクション、プロシージャなど）によって参照されている場合でも、その演算子を削除し、演算子に依存するオブジェクトに `INVALID` のマークが付きます。FORCE を使用しないと、任意のスキーマ・オブジェクトが演算子を参照しているときに演算子を削除できません。

例

ユーザー定義演算子の削除例： 次の文は、演算子 `eq_op` を削除します。

```
DROP OPERATOR eq_op;
```

FORCE 句が指定されていないため、この演算子のバインディングのいずれかが索引タイプによって参照されている場合、この操作は実行されません。

DROP OUTLINE

用途

DROP OUTLINE 文を使用すると、ストアド・アウトラインを削除できます。

参照：

- アウトラインの作成については、14-44 ページの「[CREATE OUTLINE](#)」を参照してください。
- アウトラインについては、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

前提条件

アウトラインを削除する場合は、DROP ANY OUTLINE システム権限が必要です。

構文

drop_outline::=



セマンティクス

outline

削除するアウトラインの名前を指定します。

そのアウトラインが削除された後、ストアド・アウトラインが作成された SQL 文がコンパイルされると、オブティマイザはアウトラインの影響なしに新しい実行計画を生成します。

例

アウトラインの削除例： 次の文は、ストアド・アウトライン salaries を削除します。

```
DROP OUTLINE salaries;
```

DROP PACKAGE

用途

DROP PACKAGE 文を使用すると、データベースからストアド・パッケージを削除できます。この文は、パッケージの本体および仕様部を削除します。

注意： この文を使用してパッケージから単一のオブジェクトを削除しないでください。かわりに、CREATE PACKAGE 文および CREATE PACKAGE BODY 文に OR REPLACE 句を指定して、そのオブジェクトがないパッケージを再作成してください。

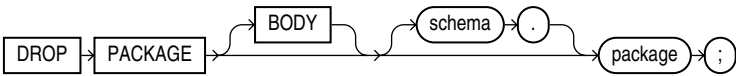
参照： 14-48 ページの「[CREATE PACKAGE](#)」を参照してください。

前提条件

削除するパッケージが自分のスキーマ内にあるか、または DROP ANY PROCEDURE システム権限が必要です。

構文

drop_package::=



セマンティクス

BODY

BODY を指定すると、パッケージ本体のみを削除できます。この句を省略した場合、パッケージ本体と仕様部の両方が削除されます。

パッケージ本体のみを削除して仕様部は削除しない場合、依存するオブジェクトは無効になりません。ただし、パッケージ本体を再作成しないかぎり、パッケージ仕様部で宣言したプロシージャやストアド・ファンクションはコールできません。

schema

パッケージが含まれているスキーマを指定します。schema を指定しない場合、パッケージが自分のスキーマ内に定義されているとみなされます。

package

削除するパッケージの名前を指定します。

そのパッケージ仕様部に依存するすべてのローカル・オブジェクトが無効になります。後でこのようなオブジェクトのいずれかを参照した場合、Oracle がそのオブジェクトを再コンパイルしようとします。削除したパッケージを再作成しないかぎり、エラーが戻されます。

統計タイプがパッケージに関連付けられている場合、FORCE 句を使用して統計タイプの関連付けが解除され、統計タイプを使用して収集されたユーザー定義のすべての統計が削除されます。

参照：

- リモート・オブジェクトを含むスキーマ・オブジェクト間の依存関係を Oracle が管理する方法については、『Oracle9i データベース概要』を参照してください。
- 12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

例

パッケージの削除例： 次の文は、emp_mgmt パッケージの仕様部および本体（14-51 ページの「[パッケージの作成例：](#)」で作成）を削除し、その仕様部に依存するすべてのオブジェクトを無効にします。

```
DROP PACKAGE emp_mgmt;
```

DROP PROCEDURE

用途

DROP PROCEDURE 文を使用すると、データベースからスタンドアロンのストアド・プロシージャを削除できます。この文を使用してパッケージの一部であるプロシージャを削除しないでください。DROP PACKAGE 文を使用して完全にパッケージを削除するか、OR REPLACE 句付きの CREATE PACKAGE 文を使用して、プロシージャを含まないパッケージを再定義します。

参照：

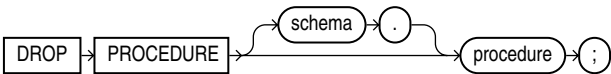
- プロシージャの作成については、14-60 ページの「[CREATE PROCEDURE](#)」を参照してください。
- プロシージャの変更については、9-122 ページの「[ALTER PROCEDURE](#)」を参照してください。

前提条件

削除するプロシージャが自分のスキーマ内にあるか、または DROP ANY PROCEDURE システム権限が必要です。

構文

drop_procedure::=



セマンティクス

schema

プロシージャが含まれているスキーマを指定します。*schema* を指定しない場合、プロシージャが自分のスキーマ内に定義されているとみなされます。

procedure

削除するプロシージャの名前を指定します。

プロシージャを削除すると、そのプロシージャに依存するすべてのローカル・オブジェクトは無効になります。後でこのようなオブジェクトのいずれかを参照した場合、Oracle がそのオブジェクトを再コンパイルしようとします。削除したプロシージャを再作成しない限り、エラー・メッセージが戻されます。

参照： リモート・オブジェクトを含むスキーマ・オブジェクト間の依存関係を Oracle が管理する方法については、『Oracle9i データベース概要』を参照してください。

例

プロシージャの削除例： 次の文は、ユーザー hr が所有するプロシージャ remove_emp を削除し、remove_emp に依存するすべてのオブジェクトを無効にします。

```
DROP PROCEDURE hr.remove_emp;
```

DROP PROFILE

用途

DROP PROFILE 文を使用すると、データベースからプロファイルを削除できます。

参照：

- プロファイルの作成については、14-66 ページの「[CREATE PROFILE](#)」を参照してください。
- プロファイルの変更については、9-125 ページの「[ALTER PROFILE](#)」を参照してください。

前提条件

DROP PROFILE システム権限が必要です。

構文

drop_profile::=



セマンティクス

profile

削除するプロファイルの名前を指定します。

プロファイルの削除の制限事項： DEFAULT プロファイルは削除できません。

CASCADE

CASCADE を指定すると、削除するプロファイルが割り当てられているすべてのユーザーから、そのプロファイルの割当てが解除されます。このようなユーザーに対しては、DEFAULT プロファイルが自動的に割り当てられます。現在、複数のユーザーに割り当てられているプロファイル削除の場合は、必ずこの句を指定します。

例

プロファイルの削除例： 次の文は、プロファイル `app_user`（14-71 ページの「[プロファイルの作成例 :](#)」で作成）を削除します。

```
DROP PROFILE app_user CASCADE;
```

`app_user` プロファイルが削除され、現在、`app_user` プロファイルが割り当てられているユーザーについては、`DEFAULT` プロファイルが割り当てられます。

DROP ROLE

用途

DROP ROLE 文を使用すると、データベースからロールを削除できます。ロールを削除した場合、そのロールが付与されていたすべてのユーザーからそのロールが取り消され、データベースからも削除されます。すでに使用可能なロールのユーザー・セッションには影響しません。ただし、削除後は新しいユーザー・セッションはロールを使用可能にできません。

参照：

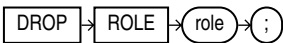
- ロールの作成の詳細は、14-73 ページの「[CREATE ROLE](#)」を参照してください。
- ロールを使用可能にするときに必要な認証の変更については、9-132 ページの「[ALTER ROLE](#)」を参照してください。
- 現行のセッションに対するロールの使用禁止については、18-44 ページの「[SET ROLE](#)」を参照してください。

前提条件

Admin Option 付きのロールが付与されているか、または DROP ANY ROLE システム権限が必要です。

構文

drop_role::=



セマンティクス

role

削除するロールの名前を指定します。

例

ロールの削除例： 次の文は、ロール dw_manager（14-75 ページの「[ロールの作成例：](#)」で作成）を削除します。

```
DROP ROLE dw_manager;
```

DROP ROLLBACK SEGMENT

用途

DROP ROLLBACK SEGMENT 文を使用すると、データベースからロールバック・セグメントを削除できます。ロールバック・セグメントを削除した場合、ロールバック・セグメントに割り当てられていたすべての領域が表領域に戻されます。

注意： データベースが自動 UNDO 管理モードで起動されている場合、ロールバック・セグメントに有効な操作はこの文のみです。このモードの場合、ロールバック・セグメントを作成または変更できません。

参照：

- ロールバック・セグメントの作成については、14-76 ページの「[CREATE ROLLBACK SEGMENT](#)」を参照してください。
- ロールバック・セグメントの変更については、9-134 ページの「[ALTER ROLLBACK SEGMENT](#)」を参照してください。
- 15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。

前提条件

DROP ROLLBACK SEGMENT システム権限が必要です。

構文

drop_rollback_segment::=

DROP → ROLLBACK → SEGMENT → rollback_segment → ;

セマンティクス

rollback_segment

削除するロールバック・セグメントの名前を指定します。

ロールバック・セグメントの削除の制限事項：

- ロールバック・セグメントがオフラインになっている場合にのみ削除できます。ロールバック・セグメントがオフラインであるかどうかを判断するには、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS に問い合わせます。オフラインのロールバック・セグメントは、STATUS 列の値が AVAILABLE になっています。また、[ALTER ROLLBACK SEGMENT](#) 文に OFFLINE 句を指定して、ロールバック・セグメントをオフラインにすることもできます。
- SYSTEM ロールバック・セグメントは削除できません。

例

ロールバック・セグメントの削除例： 次の文は、ロールバック・セグメント rbs_ts を削除します。

```
DROP ROLLBACK SEGMENT rbs_ts;
```

SQL 文 : DROP SEQUENCE ~ ROLLBACK

この章では、次の SQL 文について説明します。

- DROP SEQUENCE
- DROP SYNONYM
- DROP TABLE
- DROP TABLESPACE
- DROP TRIGGER
- DROP TYPE
- DROP TYPE BODY
- DROP USER
- DROP VIEW
- EXPLAIN PLAN
- GRANT
- INSERT
- LOCK TABLE
- MERGE
- NOAUDIT
- RENAME
- REVOKE
- ROLLBACK

DROP SEQUENCE

用途

DROP SEQUENCE を使用すると、データベースから順序を削除できます。

この文を使用した場合、順序の削除および再作成を行って、順序を再開できます。たとえば、現在、値が 150 の順序を初期値 27 で再開する場合、順序を削除して同じ名前で再作成し、START WITH 値を 27 にします。

参照：

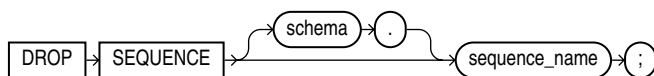
- 順序の作成の詳細は、14-82 ページの「[CREATE SEQUENCE](#)」を参照してください。
- 順序の変更の詳細は、9-138 ページの「[ALTER SEQUENCE](#)」を参照してください。

前提条件

削除する順序が自分のスキーマ内にあるか、または DROP ANY SEQUENCE システム権限が必要です。

構文

drop_sequence::=



セマンティクス

schema

順序が含まれているスキーマを指定します。*schema* の指定を省略した場合、順序が自分のスキーマ内に定義されているとみなされます。

sequence_name

削除する順序の名前を指定します。

例

順序の削除例： 次の文は、ユーザー oe（14-86 ページの「[順序の作成例 :](#)」で作成）が所有する順序 customers_seq を削除します。この文を発行する場合は、ユーザー oe として接続するか、または DROP ANY SEQUENCE システム権限が必要です。

```
DROP SEQUENCE oe.customers_seq;
```

DROP SYNONYM

用途

DROP SYNONYM 文を使用すると、データベースからシノニムを削除できます。また、シノニムの削除および再作成を行って、シノニムの定義を変更することもできます。

参照： シノニムの詳細は、15-2 ページの「[CREATE SYNONYM](#)」を参照してください。

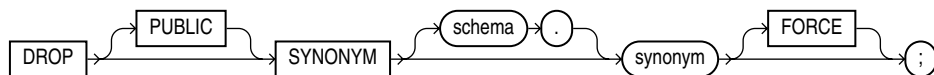
前提条件

プライベート・シノニムを削除する場合は、そのシノニムが自分のスキーマ内にあるか、または DROP ANY SYNONYM システム権限が必要です。

パブリック・シノニムを削除する場合は、DROP PUBLIC SYNONYM システム権限が必要です。

構文

drop_synonym::=



セマンティクス

PUBLIC

パブリック・シノニムを削除する場合は、PUBLIC を指定する必要があります。PUBLIC を指定した場合、*schema* は指定できません。

schema

シノニムが含まれているスキーマを指定します。*schema* の指定を省略した場合、シノニムが自分のスキーマ内に定義されているとみなされます。

synonym

削除するシノニムの名前を指定します。

マテリアライズド・ビューのシノニム、表またはマテリアライズド・ビューを含むマテリアライズド・ビュー、あるいはマテリアライズド・ビューが依存する表が削除された場合、マテリアライズド・ビューは無効になります。

シノニムの削除の制限事項： FORCE も指定しない場合、依存する表またはユーザー定義型を持つ型のシノニムは削除できません。

FORCE

依存表またはユーザー定義型を持つシノニムでも強制的に削除します。

注意： FORCE を使用して、依存性のあるオブジェクトのシノニムを削除することはお薦めしません。この操作によって、他のユーザー定義型が無効になるか、またはこのシノニムに依存する表列に UNUSED のマークが付く場合があります。型の依存性の詳細は、『Oracle9i アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

例

シノニムの削除例： 次の文は、パブリック・シノニム customers (15-6 ページの「[シノニムの変換例](#)」で作成) を削除します。

```
DROP PUBLIC SYNONYM customers;
```

DROP TABLE

用途

DROP TABLE 文を使用すると、表またはオブジェクト表、およびそれに含まれるすべてのデータをデータベースから削除できます。

注意： DROP TABLE 文はロールバックできません。

注意： 外部表の場合、この文はデータベースにある表のメタデータのみを削除します。データベースの外部に存在する実際のデータには影響しません。

注意： セッションがバインドされていない場合にのみ、一時表で DDL 操作 (ALTER TABLE、DROP TABLE、CREATE INDEX など) を実行できます。セッションを一時表にバインドするには、一時表で INSERT 操作を実行します。セッションを一時表からアンバインドするには、TRUNCATE 文を発行するか、セッションを終了します。また、トランザクション固有の一時表からアンバインドするには、COMMIT または ABORT 文を発行します。

表を削除すると、表の依存オブジェクトが無効になり、表のオブジェクト権限が削除されます。表を再作成する場合、表のオブジェクト権限を再度付与し、表の索引、整合性制約およびトリガーを作成し、記憶域パラメータを再指定する必要があります。表の切捨ての場合、このような影響はありません。そのため、表を削除して再作成するより、TRUNCATE 文で行を削除する方が効果的です。

参照：

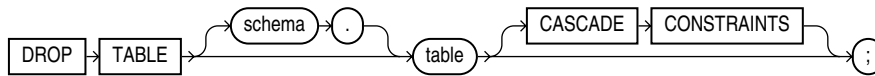
- 表の作成については、15-7 ページの「[CREATE TABLE](#)」を参照してください。
- 表の変更については、11-2 ページの「[ALTER TABLE](#)」を参照してください。
- 表を削除せずに表からデータを削除する場合の詳細は、18-51 ページの「[TRUNCATE](#)」および 16-53 ページの「[DELETE](#)」を参照してください。

前提条件

削除する表が自分のスキーマ内にあるか、または DROP ANY TABLE システム権限が必要です。

構文

drop_table::=



セマンティクス

schema

変更する表が定義されているスキーマを指定します。*schema* の指定を省略した場合、表が自分のスキーマ内に定義されているとみなされます。

table

削除する表、オブジェクト表または索引構成表の名前を指定します。Oracle によって次の操作が自動的に実行されます。

- 表からすべての行が削除されます。
- 表のすべての索引およびドメイン索引のみでなく、その表に定義されているすべてのトリガーが、作成者やスキーマの所有者とは関係なく削除されます。*table* がパーティション化されている場合、対応するローカル索引パーティションも同様に削除されます。
- *table* のネストした表および LOB のすべての記憶表が削除されます。
- レンジ・パーティション表またはハッシュ・パーティション表を削除すると、すべての表パーティションも削除されます。コンポジット・パーティション表を削除した場合、すべてのパーティションおよびサブパーティションが同様に削除されます。
- 索引構成表の場合、索引構成表に定義したマッピング表も削除されます。
- ドメイン索引の場合は、この文によって適切な削除ルーチンが起動されます。

参照： これらのルーチンの詳細は、『Oracle9i Data Cartridge Developer’s Guide』を参照してください。

- 任意の統計タイプが表に関連付けられている場合、FORCE 句を使用して統計タイプの関連付けが解除され、統計タイプを使用して収集されたユーザー定義のすべての統計が削除されます。

参照： 統計タイプの関連の詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

- 表がクラスタの一部でない場合は、表とその索引に割り当てられていたすべてのデータ・ブロックを、表とその索引が格納されていた表領域に戻します。

注意： DROP CLUSTER 文に INCLUDING TABLES 句を指定した場合、クラスタおよびそのすべての表を削除できます。これによって、表を 1 つずつ削除する必要がなくなります。16-64 ページの「[DROP CLUSTER](#)」を参照してください。

- 表がビュー、マテリアライズド・ビューのコンテンツ表またはマスター表の実表である場合、あるいは表がストアド・プロシージャ、ファンクションまたはパッケージで参照されている場合、依存するオブジェクトは無効になりますが、削除はされません。表を再作成するか、これらのオブジェクトを一度削除してその表に依存しない形で再作成しないかぎり、これらのオブジェクトは使用できません。
- 表を再作成する場合、ストアド・プロシージャ、ファンクションまたはパッケージで参照する列、およびマテリアライズド・ビューの定義で使用されていた副問合せで選択されるすべての列が、その表に含まれている必要があります。ビュー、ストアド・プロシージャ、ファンクション、パッケージに対するオブジェクト権限を付与されていたユーザーに、これらの権限を再付与する必要はありません。
- 表がマテリアライズド・ビューのマスター表の場合、マテリアライズド・ビューの問合せはできます。ただし、そのマテリアライズド・ビューの副問合せによって選択されるすべての列が含まれる表を再作成しないかぎり、リフレッシュはできません。
- 表がマテリアライズド・ビュー・ログを含む場合、このログおよびその表に関連付けられているダイレクト・パス・インサートのその他のリフレッシュ情報は削除されます。

表の削除の制限事項： ネストした表の記憶表は直接削除できません。かわりに、ALTER TABLE ... DROP COLUMN 句を使用して、ネストした表の列を削除する必要があります。

CASCADE CONSTRAINTS

CASCADE CONSTRAINTS は、削除した表の主キーまたは一意キーを参照するすべての参照整合性制約を削除します。このような参照整合性制約があるときにこの句の指定を省略した場合、エラーが戻され、表は削除されません。

例

表の削除例： 次の文は、oe.list_customers 表（15-69 ページの「[リスト・パーティション化の例：](#)」で作成）を削除します。

```
DROP TABLE list_customers;
```

DROP TABLESPACE

用途

DROP TABLESPACE を使用すると、データベースから表領域を削除できます。

参照：

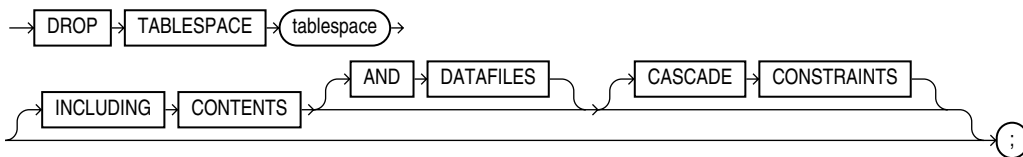
- 表領域の作成については、15-75 ページの「[CREATE TABLESPACE](#)」を参照してください。
- 表領域の変更については、11-97 ページの「[ALTER TABLESPACE](#)」を参照してください。

前提条件

DROP TABLESPACE システム権限が必要です。アクティブ・トランザクションを保持するロールバック・セグメントを含む場合は、表領域を削除できません。

構文

drop_tablespace::=



セマンティクス

tablespace

削除する表領域の名前を指定します。

表領域の状態がオンラインまたはオフラインのどちらであっても、その表領域を削除できます。実行中のトランザクション内の SQL 文で、表領域内のいずれかのオブジェクトにアクセスすることがないように、表領域はオフラインにしてから削除することをお勧めします。

表領域がデフォルト表領域または一時表領域として割り当てられていたユーザーを変更することもできます。表領域が削除された後では、このようなユーザーはオブジェクトに領域を割り当てたり、表領域内で領域をソートすることはできません。ALTER USER 文を使用すると、ユーザーに新しいデフォルト表領域および一時表領域を割り当てることができます。

表領域にあるすべてのデータ・ファイルおよび一時ファイルに関するすべてのメタデータが、データ・ディクショナリから削除されます。表領域にある Oracle 管理データ・ファイルおよび一時ファイルが、オペレーティング・システムから自動的に削除されます。その他のデータ・ファイルおよび一時ファイルは、INCLUDING CONTENTS AND DATAFILES を指定しないかぎり、オペレーティング・システムから削除されません。

表領域の削除の制限事項：

- SYSTEM 表領域は削除できません。
- ドメイン索引またはドメイン索引によって作成されたオブジェクトを格納している表領域は削除できません。
- いずれかのインスタンスによって使用されている場合、またはコミットされていないトランザクションのロールバックに必要な UNDO データを含む場合は、UNDO 表領域を削除できません。

参照： ドメイン索引の詳細は、『Oracle9i Data Cartridge Developer's Guide』および『Oracle9i データベース概要』を参照してください。

INCLUDING CONTENTS

INCLUDING CONTENTS は、表領域の中のすべてのデータベース・オブジェクトを削除します。データベース・オブジェクトを格納している表領域を削除する場合は、必ずこの句を指定します。表領域が空でない場合にこの句を省略した場合、エラーが戻され、表領域は削除されません。

パーティション表の場合、次のパーティションが表領域に（すべてではなく）一部含まれていると、INCLUDING CONTENTS を指定しても DROP TABLESPACE コマンドが正常に実行されません。

- レンジ・パーティション表またはハッシュ・パーティション表のパーティション
- コンポジット・パーティション表のサブパーティション

注意： パーティション表のすべてのパーティションが *tablespace* に存在する場合、DROP TABLESPACE ... INCLUDING CONTENTS は、*tablespace* を削除し、関連付けられた索引セグメント、LOB データ・セグメントおよび他の表領域にある LOB 索引セグメントも削除します。

パーティション化された索引構成表の場合、すべての主キー索引セグメントがこの表領域に存在する場合、この句は、他の表領域にあるオーバーフロー・セグメントも、他の表領域にある関連するマッピング表と同様に削除します。キー索引セグメントのいくつかが存在しない場合、その文は実行されません。その場合、その表領域を削除する前に、ALTER TABLE ... MOVE PARTITION を使用して、それらの主キー索引セグメントをこの表領域に移動し、この表領域にオーバーフロー・データ・セグメントの存在しないパーティションを削除します。また、パーティション化された索引構成表も削除します。

表領域がマテリアライズド・ビューのマスター表を含む場合、マテリアライズド・ビューは無効になります。

表領域がマテリアライズド・ビュー・ログを含む場合、このログおよびその表に関連付けられているダイレクト・パス・インサートのその他のリフレッシュ情報は削除されます。

AND DATAFILES

INCLUDING CONTENTS を指定すると、AND DATAFILES 句によって、関連するオペレーティング・システム・ファイルも同様に削除されます。アラート・ログに各オペレーティング・システム・ファイルに対するメッセージが書き込まれます。この句は Oracle 管理ファイルには必要ではありません。

CASCADE CONSTRAINTS

CASCADE CONSTRAINTS は、*tablespace* に含まれる表の主キーまたは一意キーを参照する、*tablespace* の外の表にあるすべての参照整合性制約を削除します。このような参照整合性制約があるときにこの句の指定を省略した場合、エラーが戻され、表領域は削除されません。

例

表領域の削除例： 次の文は、tbs_01 表領域を削除し、tbs_01 に含まれる主キーおよび一意キーを参照するすべての参照整合性制約を削除します。

```
DROP TABLESPACE tbs_01
    INCLUDING CONTENTS
    CASCADE CONSTRAINTS;
```

オペレーティング・システム・ファイルの削除例： 次の例は、tbs_02 表領域およびそれに関連するすべてのオペレーティング・システムのデータ・ファイルを削除します。

```
DROP TABLESPACE tbs_02
    INCLUDING CONTENTS AND DATAFILES;
```

DROP TRIGGER

用途

DROP TRIGGER を使用すると、データベースからデータベース・トリガーを削除できます。

参照：

- トリガーの作成については、15-89 ページの「[CREATE TRIGGER](#)」を参照してください。
- トリガーの使用可能化、使用禁止化またはコンパイルの詳細は、12-2 ページの「[ALTER TRIGGER](#)」を参照してください。

前提条件

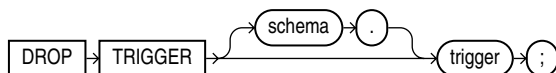
削除するトリガーが自分のスキーマ内にあるか、または DROP ANY TRIGGER システム権限が必要です。

他のユーザーのスキーマ内のデータベースにあるトリガーを削除する場合は、ADMINISTER DATABASE TRIGGER システム権限が必要です。

参照： これらの権限の詳細は、15-89 ページの「[CREATE TRIGGER](#)」を参照してください。

構文

drop_trigger::=



セマンティクス

schema

削除するトリガーが含まれているスキーマを指定します。*schema* の指定を省略した場合、トリガーが自分のスキーマ内に定義されているとみなされます。

trigger

削除するトリガーの名前を指定します。データベースからトリガーが削除され、再度、起動されることはありません。

例

トリガーの削除例： 次の文は、スキーマ `oe` 内のトリガー `order` を削除します。

```
DROP TRIGGER oe.order;
```

DROP TYPE

用途

DROP TYPE 文を使用すると、オブジェクト型、VARRAY 型またはネストした表型の仕様部および本体を削除できます。

参照：

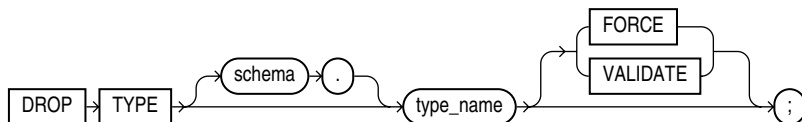
- オブジェクト型の本体のみの削除については、17-17 ページの「[DROP TYPE BODY](#)」を参照してください。
- 型の作成については、16-3 ページの「[CREATE TYPE](#)」を参照してください。

前提条件

削除するオブジェクト型、VARRAY 型またはネストした表型が自分のスキーマ内にあるか、または DROP ANY TYPE システム権限が必要です。

構文

drop_type::=



セマンティクス

schema

型が含まれているスキーマを指定します。*schema* の指定を省略した場合、型が自分のスキーマ内に定義されているとみなされます。

type_name

削除するオブジェクト型、VARRAY 型またはネストした表型の名前を指定します。型依存性または表依存性のない型のみ削除できます。

type_name がスーパータイプの場合、FORCE も指定しないと、この文は正常に実行されません。FORCE を指定すると、そのスーパータイプに依存するすべてのサブタイプが無効になります。

`type_name` が統計タイプの場合、`FORCE` も指定しないと、この文は正常に実行されません。`FORCE` を指定した場合、最初に `type_name` に関連付けられたすべてのオブジェクトの関連付けが解除され、`type_name` が削除されます。

参照： 統計タイプの詳細は、12-46 ページの「[ASSOCIATE STATISTICS](#)」および 16-61 ページの「[DISASSOCIATE STATISTICS](#)」を参照してください。

`type_name` が統計タイプに関連付けられたオブジェクト型の場合、最初にこの統計タイプから `type_name` の関連付けが解除され、`type_name` が削除されます。ただし、統計タイプを使用して統計が収集された場合、この統計タイプから `type_name` の関連付けを解除することはできません。このため、この文は正常に実行されません。

`type_name` が索引タイプの実装タイプの場合、索引タイプに `INVALID` のマークが付けられます。

`type_name` にパブリック・シノニムが定義されている場合、このシノニムも削除されます。

`FORCE` オプションを指定していない場合に削除できるのは、依存性のないスタンドアロンのスキーマ・オブジェクトとして定義されているオブジェクト型またはネストした表型または `VARRAY` 型のみです。これはデフォルトの動作です。

参照： 13-88 ページの「[CREATE INDEXTYPE](#)」を参照してください。

FORCE

依存するデータベース・オブジェクトを持つ型でも強制的に削除します。削除する型に依存するすべての列に `UNUSED` のマークが付けられ、それらの列にアクセスできなくなります。

注意： `FORCE` を使用して、依存性のある型を削除することはお勧めしません。この操作はリカバリ不能であり、依存表または列のデータにアクセスできなくなる場合があります。型の依存性の詳細は、『[Oracle9i アプリケーション開発者ガイド - 基礎編](#)』を参照してください。

VALIDATE

型を削除するときに `VALIDATE` を指定すると、格納されているこの型のインスタンスがスーパータイプのいずれかの置換可能な列の範囲であることが検証されます。このようなインスタンスがない場合、`Oracle` は削除操作を完了します。

この句はサブタイプのみに意味があります。明示的な型または表依存性のないサブタイプを安全に削除するために、このオプションの使用をお勧めします。

例

オブジェクト型の削除例： 次の文は、オブジェクト型 `person_t`（16-22 ページの「[型の階層例](#)」で作成）を削除します。

```
DROP TYPE person_t;
```

DROP TYPE BODY

用途

DROP TYPE BODY 文を使用すると、オブジェクト型、VARRAY 型またはネストした表型の本体を削除できます。型本体を削除しても、オブジェクト型の仕様部は残ります。また、削除した型本体は再作成できます。その本体を再作成する場合、型本体を削除したオブジェクト型は引き続き使用できますが、メンバー・ファンクションはコールできません。

参照：

- オブジェクトの本体および仕様部を削除する場合は、17-14 ページの「[DROP TYPE](#)」を参照してください。
- 型本体の詳細は、16-25 ページの「[CREATE TYPE BODY](#)」を参照してください。

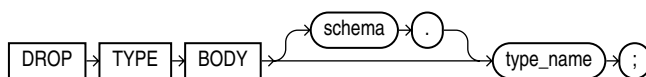
前提条件

オブジェクト型の本体が自分のスキーマ内にある必要があります。また、次の権限が必要です。

- CREATE TYPE または CREATE ANY TYPE システム権限
- DROP ANY TYPE システム権限

構文

drop_type_body::=



セマンティクス

schema

オブジェクト型が含まれているスキーマを指定します。*schema* の指定を省略した場合、オブジェクト型が自分のスキーマ内に定義されているとみなされます。

type_name

削除するオブジェクト型の本体の名前を指定します。

型本体の削除の制限事項： 型依存性および表依存性がない場合にのみ型の本体を削除できます。

例

オブジェクト型の本体の削除例： 次の文は、オブジェクト型の本体 data_typ（16-30 ページの「[型本体の更新例 :](#)」で作成）を削除します。

```
DROP TYPE BODY data_typ;
```


DROP USER

用途

DROP USER 文を使用すると、データベース・ユーザーを削除できます。また、オプションでユーザーのオブジェクトを削除することもできます。

参照：

- ユーザーの作成については、16-31 ページの「[CREATE USER](#)」を参照してください。
- ユーザー定義の変更については、12-21 ページの「[ALTER USER](#)」を参照してください。

前提条件

DROP USER システム権限が必要です。

構文

drop_user::=



セマンティクス

user

削除するユーザーを指定します。CASCADE を指定しない場合、またはユーザーのオブジェクトを最初に明示的に削除しない場合、所有するスキーマにオブジェクトが含まれているユーザーは削除されません。

CASCADE

CASCADE は、ユーザーを削除する前に、そのユーザーのスキーマ内にあるすべてのオブジェクトを削除します。所有するスキーマにオブジェクトが含まれているユーザーを削除する場合は、必ずこの句を指定します。

- ユーザーのスキーマに表がある場合、表が削除され、その表の主キーまたは一意キーを参照している他のユーザーのスキーマ内にある表の参照整合性制約も自動的に削除されます。
- この句で表が削除される場合、その表の列で作成されたすべてのドメイン索引も削除され、適切な削除ルーチンが起動されます。

参照： これらのルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

- Oracle は、他のスキーマにある、削除された自分のスキーマ内のオブジェクトのビューまたはシノニム、および削除された自分のスキーマ内のオブジェクトを問い合わせるストアド・プロシージャ、ファンクション、パッケージのオブジェクトを削除せず、無効にします。
- 削除されたユーザーのスキーマ内にある表に基づく他のスキーマ内のマテリアライズド・ビューは削除されません。ただし、実表は存在しないため、他のスキーマ内のマテリアライズド・ビューはリフレッシュできなくなります。
- ユーザーのスキーマにあるすべてのトリガーは削除されます。
- ユーザーが作成したロールは削除されません。

注意： Oracle では、ユーザーが所有するすべての型も **FORCE** で削除します。詳細は、17-15 ページの「**DROP TYPE**」にある **FORCE** キーワードを参照してください。

例

データベース・ユーザーの削除例： ユーザー **sidney** のスキーマ内にオブジェクトがない場合、次の文を発行すると、**sidney** を削除できます。

```
DROP USER sidney;
```

sidney のスキーマ内にオブジェクトがある場合は、次の文のように **CASCADE** 句を指定して、**sidney** とそのオブジェクトを削除する必要があります。

```
DROP USER sidney CASCADE;
```

DROP VIEW

用途

DROP VIEW 文を使用すると、データベースからビューまたはオブジェクト・ビューを削除できます。ビューを削除して再作成すると、ビューの定義を変更できます。

参照：

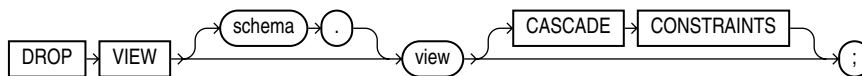
- ビューの作成については、16-38 ページの「[CREATE VIEW](#)」を参照してください。
- ビューの変更については、12-29 ページの「[ALTER VIEW](#)」を参照してください。

前提条件

削除するビューが自分のスキーマ内にあるか、または DROP ANY VIEW システム権限が必要です。

構文

drop_view::=



セマンティクス

schema

ビューを含むスキーマを指定します。*schema* の指定を省略した場合、ビューが自分のスキーマ内に定義されているとみなされます。

view

削除するビューの名前を指定します。

ビューを参照するビュー、マテリアライズド・ビューおよびシノニムは削除されませんが、INVALID のマークが付けられます。このようなビューおよびシノニムは削除または再定義するか、無効なビューやシノニムをもう一度有効にするビューを別に定義します。

ビューにサブビューが定義してある場合、サブビューも同様に無効になります。ビューがサブビューを持つかどうかを確認するには、USER_VIEWS、ALL_VIEWS または DBA_VIEWS データ・ディクショナリ・ビューの SUPERVIEW_NAME 列を問い合わせます。

参照：

- 15-7 ページの「[CREATE TABLE](#)」および 15-2 ページの「[CREATE SYNONYM](#)」を参照してください。
- 無効なマテリアライズド・ビューの再検証の詳細は、9-88 ページの「[ALTER MATERIALIZED VIEW](#)」を参照してください。

CASCADE CONSTRAINTS

CASCADE CONSTRAINTS は、削除するビューの主キーまたは一意キーを参照するすべての参照整合性制約を削除します。このような制約が存在する状態でこの句を省略すると、DROP 文は正常に実行されません。

例

ビューの削除例： 次の文は、emp_view ビュー（16-49 ページの「[ビューの作成例：](#)」で作成）を削除します。

```
DROP VIEW emp_view;
```

EXPLAIN PLAN

用途

EXPLAIN PLAN 文を使用すると、指定した SQL 文を実行するために Oracle が使用する実行計画を決定できます。この文によって、実行計画の各手順を記述している行が、指定した表に挿入されます。SQL トレース機能の一部として EXPLAIN PLAN 文を発行することもできます。

コストベースの最適化を使用している場合、この文によって文を実行するコストも決まります。表にドメイン索引が定義されている場合、ユーザー定義の CPU および I/O コストが挿入されます。

配布メディアの SQL スクリプトの中に、サンプル出力表 PLAN_TABLE の定義があります。使用する出力表は、列の名前およびデータ型がこのサンプル表と同じである必要があります。このスクリプトの一般的な名前は、UTLXPLAN.SQL です。正確な名前および位置は、使用するオペレーティング・システムによって異なります。

注意： キャッシュされたカーソルに関する情報は、次の動的パフォーマンス・ビューから得られます。

- SQL カーソルが使用する作業領域の詳細は、V\$SQL_WORKAREA を問い合わせます。
 - キャッシュされたカーソルの実行計画の詳細は、V\$SQL_PLAN を問い合わせます。
 - キャッシュされたカーソルの各ステップでの実行統計または実行計画の操作（生成された行数、読み込まれたブロック数など）は、V\$SQL_PLAN_STATISTICS ビューを問い合わせます。
 - 前述の 3 つのビューの処理結果を 1 つにまとめた情報を取得する場合は、V\$SQL_PLAN_STATISTICS_ALL を問い合わせます。
-

参照：

- 実行計画の出力、SQL トレース機能の使用方法、および実行計画の生成と解析方法については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- 動的パフォーマンス・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

前提条件

EXPLAIN PLAN 文を実行する場合、実行計画を格納する既存の出力表に行を挿入するための権限が必要です。

出力する実行計画の適用対象である SQL 文を実行するための権限も必要です。この SQL 文でビューにアクセスする場合は、このビューの基礎になっているすべての表およびビューへのアクセス権限が必要です。このビューが別のビューに基づき、さらにこの別のビューが、ある表に基づいている場合、別のビューとそのビューの基礎になっている表へのアクセス権限が必要です。

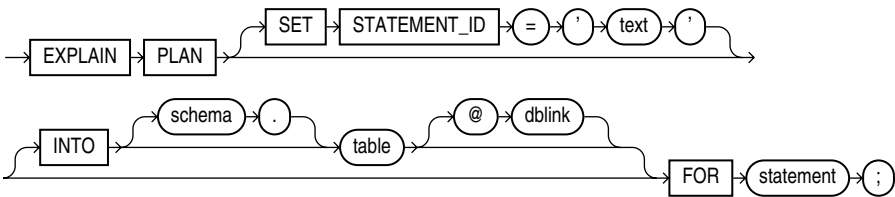
EXPLAIN PLAN で作成された実行計画を検証する場合は、出力表へ問い合わせる権限が必要です。

EXPLAIN PLAN 文はデータ操作言語（DML）文であり、データ定義言語（DDL）文ではありません。そのため、EXPLAIN PLAN 文で加えられた変更内容は暗黙的にコミットされません。出力表の EXPLAIN PLAN 文で生成された行を保存する場合は、この文を指定したトランザクションをコミットする必要があります。

参照： 計画表の移入および問合せに必要な権限の詳細は、17-52 ページの「INSERT」および 18-4 ページの「SELECT」を参照してください。

構文

explain_plan::=



セマンティクス

SET STATEMENT_ID 句

実行計画が出力される表の中で、この実行計画に該当する行にある STATEMENT_ID 列の値を指定します。この値によって、実行計画の行を出力表の中の他の行と区別できます。ユーザーの出力表に多数の実行計画の行が含まれている場合は、必ず、STATEMENT_ID の値を指定します。この句を省略した場合、デフォルトで STATEMENT_ID 値が NULL に設定されます。

INTO table 句

出力表の名前を指定し、オプションとしてそのスキーマおよびデータベースの名前も指定します。この表は、EXPLAIN PLAN 文を使用する前に作成しておく必要があります。

schema の指定を省略した場合、表が自分のスキーマ内に定義されているとみなされます。

dblink には、出力表が格納されているリモートの Oracle データベースに対するデータベース・リンクの完全な名前または名前の一部を指定します。Oracle の分散機能を使用している場合にのみ、リモート出力表を指定できます。*dblink* の指定を省略した場合、表がローカル・データベース上にあるとみなされます。

参照： データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。

INTO 句を省略した場合、出力表は、ローカル・データベース上の自分のスキーマ内にある PLAN_TABLE であるとみなされます。

FOR statement 句

実行計画生成の対象となる SELECT、INSERT、UPDATE、DELETE、CREATE TABLE、CREATE INDEX または ALTER INDEX ... REBUILD 文を指定します。

EXPLAIN PLAN の注意事項：

- *statement* が *parallel_clause* を持つ場合、結果として生成される実行計画はパラレルで実行されます。ただし、EXPLAIN PLAN コマンドによって計画表に実際に文が挿入されるため、ユーザーが発行したパラレル DML 文は、トランザクションの最初の DML 文ではなくなります。これは、トランザクション 1 つにつきパラレル DML 文は 1 つという Oracle の制限に違反するため、この文はシリアルで実行されます。文をパラレルで実行するには、EXPLAIN PLAN 文をコミットまたはロールバックしてから、パラレル DML 文を発行する必要があります。
- 一時表上で操作するための実行計画を判断する場合、EXPLAIN PLAN は、同一セッションから実行する必要があります。これは、一時表内のデータがセッション固有であるためです。

例

実行計画の例： 次の文は、UPDATE 文の実行計画およびコストを決定し、実行計画を記述した行を STATEMENT_ID 値 'Raise in Tokyo' とともに、指定した plan_table 表に挿入します。

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'Raise in Tokyo'
  INTO plan_table
  FOR UPDATE employees
    SET salary = salary * 1.10
    WHERE department_id =
      (SELECT department_id FROM departments
       WHERE location_id = 1200);
```

次の SELECT 文は、plan_table 表への問合せを実行し、実行計画およびコストを戻します。

```
SELECT LPAD(' ',2*(LEVEL-1))||operation operation, options,
object_name, position
FROM plan_table
START WITH id = 0 AND statement_id = 'Raise in Tokyo'
CONNECT BY PRIOR id = parent_id AND
statement_id = 'Raise in Tokyo';
```

問合せによって、次の実行計画が戻されます。

OPERATION	OPTIONS	OBJECT_NAME	POSITION

UPDATE STATEMENT			2
UPDATE		EMPLOYEES	1
TABLE ACCESS	FULL	EMPLOYEES	1
VIEW		index\$_join\$_00	1
		2	
HASH JOIN			1
INDEX	RANGE SCAN	DEPT_LOCATION_I	1
		X	
INDEX	FAST FULL SCAN	DEPT_ID_PK	2

POSITION 列の 1 行目の値は、文のコストが 1 であることを示しています。

実行計画のパーティション化例： サンプル表 sh.sales が、time_id 列でパーティション化されます。パーティション sales_q3_2000 には、2000 年 10 月 1 日より小さい時刻の値があり、time_id 列にローカル索引 sales_time_bix があります。

次の問合せを考えてみます。

```
EXPLAIN PLAN FOR
SELECT * FROM sales
WHERE time_id BETWEEN :h AND '01-OCT-2000';
```

ここで、:h はすでに宣言されているバインド変数を指します。EXPLAIN PLAN 文は、PLAN_TABLE を出力表とする次の問合せを実行します。次の問合せによって、パーティション情報などの基本的な実行計画が得られます。

```
SELECT operation, options, partition_start, partition_stop,
partition_id FROM plan_table;
```

GRANT

用途

GRANT 文を使用すると、次の権限を付与できます。

- ユーザーおよびロールに対するシステム権限
- ユーザーおよびロールに対するロール。権限とロールは、ともにローカル、グローバルまたは外部になります。表 17-1 に、システム権限のリストを、操作対象のデータベース・オブジェクト別に示します。表 17-2 に、Oracle で事前定義されているロールを示します。
- ユーザー、ロールおよび PUBLIC に対する、特定のオブジェクトに対するオブジェクト権限。表 17-3 に、各オブジェクトに付与できるオブジェクト権限を示します。表 17-4 に、オブジェクト権限とその権限によって許可される操作を示します。

注意： データベース・ユーザーにロールの使用を許可する方法は、データベースや GRANT 文を介して行う以外にもあります。たとえば、オペレーティング・システムによっては、Oracle ユーザーに対して、初期化パラメータ OS_ROLES でロールを付与する機能を備えている場合もあります。オペレーティング・システム機能を使用してユーザーにロールを付与する場合、GRANT 文を使用してユーザーにシステム権限を付与したり、その他のロールにシステム権限およびロールを付与することはできますが、そのユーザーにロールを付与することはできません。

参照：

- ローカル、グローバルおよび外部権限の定義については、16-31 ページの「[CREATE USER](#)」および 14-73 ページの「[CREATE ROLE](#)」を参照してください。
- その他の認証方法の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- 権限付与の取消しについては、17-86 ページの「[REVOKE](#)」を参照してください。

前提条件

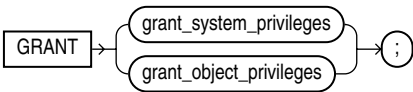
システム権限を付与する場合は、Admin Option 付きのシステム権限または GRANT ANY PRIVILEGE システム権限が付与されている必要があります。

ロールを付与する場合は、Admin Option 付きのロールまたは GRANT ANY ROLE システム権限が付与されているか、あるいは付与するロールが自分で作成したロールである必要があります。

オブジェクト権限を付与する場合は、オブジェクトの所有者である必要があります。そうでない場合、オブジェクトの所有者から Grant Option 付きのオブジェクト権限または GRANT ANY OBJECT PRIVILEGE システム権限が付与されている必要があります。

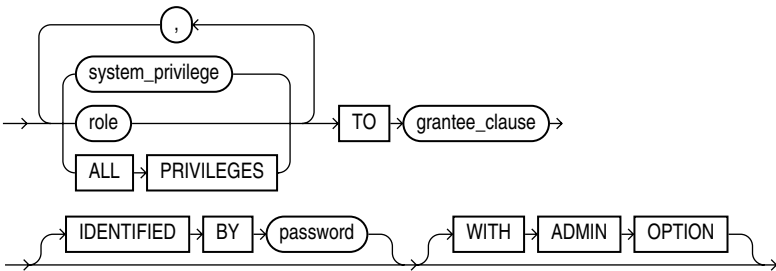
構文

grant::=



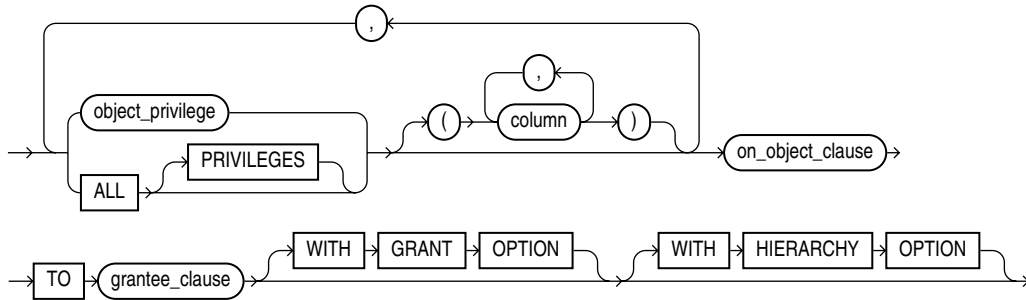
(17-28 ページの「[grant_system_privileges::=](#)」、17-29 ページの「[grant_object_privileges::=](#)」を参照)

[grant_system_privileges::=](#)



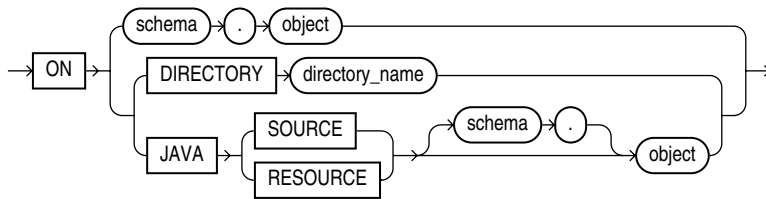
(17-29 ページの「[grantee_clause::=](#)」を参照)

grant_object_privileges::=

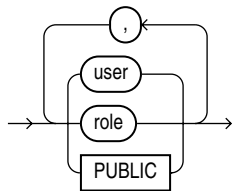


(17-29 ページの「[on_object_clause::=](#)」、17-29 ページの「[grantee_clause::=](#)」を参照)

on_object_clause::=



grantee_clause::=



セマンティクス

grant_system_privileges

system_privilege

付与するシステム権限を指定します。表 17-1 に、システム権限のリストを、操作対象のデータベース・オブジェクト別に示します。

- **ユーザー**に権限を付与する場合、ユーザーの権限ドメインに権限が登録されます。このユーザーはその権限をすぐに使用できます。

参照： 17-49 ページの「[ユーザーに対してシステム権限を付与する例：](#)」を参照してください。

- **ロール**に権限を付与する場合、ロールの権限ドメインに権限が登録されます。ロールが付与されているまたは使用可能となっているユーザーは、その権限をすぐに使用できます。なお、ロールが付与されている他のユーザーも、そのロールを使用可能にしてその権限を使用できます。

参照： 17-49 ページの「[ロールに対してシステム権限を付与する例：](#)」を参照してください。

- **PUBLIC**に権限を付与する場合、各ユーザーの権限ドメインに権限が登録されます。すべてのユーザーは、その権限によって許可される操作をすぐに実行できます。

すべてのシステム権限を一度に指定できるショートカットがあります。

- **ALL PRIVILEGES**。ALL PRIVILEGES を指定すると、SELECT ANY DICTIONARY 権限を除き、17-35 ページの表 17-1「[システム権限](#)」に示すすべてのシステム権限を付与できます。

role

付与するロールを指定します。事前に定義されたロールまたはユーザーが定義したロールを付与できます。表 17-2 は、事前に定義されたロールのリストです。

- **ユーザー**にロールを付与する場合、ユーザーに付与されたロールを使用できます。ユーザーはそのロールをすぐに使用可能にし、ロールの権限ドメインに登録されている権限を使用できます。
- 他の**ロール**にロールを付与する場合、権限受領者のロールの権限ドメインに、権限付与者のロールの権限ドメインが登録されます。権限受領者のロールを付与されているユーザーは、それを使用可能にした場合、付与されたロールの権限ドメイン内の権限を使用できます。

参照： 17-50 ページの「[ロールに対してロールを付与する例:](#)」を参照してください。

- **PUBLIC** にロールを付与する場合、すべてのユーザーがロールを使用できます。ユーザーはそのロールを使用可能にし、ロールの権限ドメインに登録されている権限をすぐに使用できます。

参照： ユーザー定義ロールの作成については、14-73 ページの「[CREATE ROLE](#)」を参照してください。

IDENTIFIED BY 句

IDENTIFIED BY 句を使用すると、パスワードによって既存ユーザーが明確に識別できるか、または存在しないユーザーを作成できます。この句は、権限受領者がロールまたは PUBLIC の場合は無効です。grantee_clause に指定するユーザーが存在しない場合、パスワードおよびこの句で指定する権限とロール付きでユーザーが作成されます。

参照： ユーザー名およびパスワードの制限事項については、16-31 ページの「[CREATE USER](#)」を参照してください。

With Admin Option

With Admin Option を指定すると、権限受領者は次のことができます。

- ロールが GLOBAL ロールでない場合、そのロールを他のユーザーまたはロールに付与できます。
- ロールを他のユーザーまたは他のロールから取り消すことができます。
- ロールへのアクセスに必要な認証を変更するため、ロールを変更できます。
- ロールを削除できます。

たとえば、With Admin Option を指定せずにユーザーにロールまたはシステム権限を付与し、後で With Admin Option を指定してその権限およびロールを付与した場合、そのユーザーはその権限またはロールに関して Admin Option を持つことになります。

システム権限またはロールの Admin Option をユーザーから取り消す場合、そのユーザーの権限またはロールを完全に取り消し、その次に Admin Option を指定せずに権限またはロールをユーザーに付与します。

参照： 17-49 ページの「[Admin Option 付きロールを付与する例:](#)」を参照してください。

grantee_clause

TO grantee_clause は、システム権限、ロールまたはオブジェクト権限が付与されているユーザーまたはロールを識別します。

権限受領者の制限事項： TO *grantee_clause* にユーザー、ロールおよび PUBLIC を指定できるのは 1 回のみです。

PUBLIC すべてのユーザーに権限を付与する場合は、PUBLIC を指定します。

システム権限およびロールの付与の制限事項：

- 付与する権限およびロールのリストに、権限またはロールを指定できるのは 1 回のみです。
- ロールに対してそのロールと同じロールは付与できません。
- ロール IDENTIFIED GLOBALLY は、どのようなユーザーまたはロールに対しても付与できません。
- ロール IDENTIFIED EXTERNALLY は、グローバル・ユーザーまたはグローバル・ロールに対して付与できません。
- ロールは交互に付与できません。たとえば、ロール *banker* をロール *teller* に付与した場合、逆に *teller* を *banker* に付与することはできません。

grant_object_privileges

object_privilege

付与するオブジェクト権限を指定します。表 17-3 に示すいずれかの値を指定します。表 17-4 も参照してください。

オブジェクト権限の制限事項： 付与する権限のリストに権限を指定できるのは 1 回のみです。

ALL [PRIVILEGES]

Grant Option 付きで付与されているオブジェクト権限に対するすべての権限を付与する場合に、ALL を指定します。オブジェクトが定義されているスキーマを所有しているユーザーは、自動的に Grant Option 付きのオブジェクトに関するすべての権限を持っています（キーワード PRIVILEGES はセマンティクスを明確にするためのものであり、指定は任意です。）

column

権限を付与する表またはビューの列を指定します。INSERT、REFERENCES または UPDATE の各権限を付与する場合にのみ、列を指定できます。列を指定しない場合、権限受領者には表またはビューのすべての列に対して指定した権限が付与されます。

既存の列オブジェクトに関する権限の付与については、データ・ディクショナリ・ビュー USER_COL_PRIVS、ALL_COL_PRIVS および DBA_COL_PRIVS に問い合わせます。

参照： データ・ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』および 17-51 ページの「[別々の列に複数のオブジェクト権限を付与する例:](#)」を参照してください。

on_object_clause

on_object_clause は、権限が付与されているオブジェクトを識別します。ディレクトリ・スキーマ・オブジェクト、Java ソースおよびリソース・スキーマ・オブジェクトは、異なるネームスペースに格納されるため、別々に識別されます。

GRANT ANY OBJECT PRIVILEGE システム権限が付与されている場合のみ (*object* の所有者ではなく、*object With Grant Option* で *object_privilege* が付与されていない)、この権限付与を行うと、オブジェクト所有者の役割を果たすことになります。

*_TAB_PRIVS データ・ディクショナリ・ビューには、この権限付与が *object* の所有者によって行われたことが反映されます。

参照：

- 17-49 ページの「[ロールに対してオブジェクト権限を付与する例:](#)」を参照してください。
- GRANT ANY OBJECT PRIVILEGE システム権限による権限の取消し操作の詳細は、17-95 ページの「[GRANT ANY OBJECT PRIVILEGE を使用する操作を取り消す例:](#)」を参照してください。

With Grant Option

With Grant Option は、権限受領者が、他のユーザーまたはロールに対してオブジェクト権限を付与する場合に指定します。

With Grant Option の付与の制限事項： With Grant Option は、ユーザーまたは PUBLIC に権限を付与する場合にのみ指定できます。ロールに付与する場合は指定できません。

With Hierarchy Option

With Hierarchy Option を指定すると、この文の後に作成されるサブオブジェクトも含め、*object* のすべてのサブオブジェクト（ビューを基に作成したサブビューなど）に対する指定したオブジェクト権限を付与できます。

注意： この句は、SELECT オブジェクト権限とあわせて指定する場合のみ意味があります。

object 権限を付与するスキーマ・オブジェクトを指定します。object に *schema* を指定しなかった場合、そのオブジェクトは自分のスキーマ内にあるとみなされます。オブジェクトには次のタイプがあります。

- 表、ビューまたはマテリアライズド・ビュー
- 順序
- プロシージャ、ファンクションまたはパッケージ
- ユーザー定義型
- これらの項目のシノニム
- ディレクトリ、ライブラリ、演算子または索引タイプ
- Java ソース、クラスまたはリソース

注意： パーティション表の単一パーティションに直接権限を付与することはできません。

参照： 17-50 ページの「ユーザーに対して表へのオブジェクト権限を付与する例」、17-50 ページの「ビューへのオブジェクト権限を付与する例：」および 17-50 ページの「別のスキーマの順序に対してオブジェクト権限を付与する例：」を参照してください。

DIRECTORY *directory_name* 権限を付与するディレクトリ・スキーマ・オブジェクトを指定します。*directory_name* にはスキーマ名を指定できません。

参照： 13-46 ページの「[CREATE DIRECTORY](#)」および 17-50 ページの「[ディレクトリへのオブジェクト権限を付与する例：](#)」を参照してください。

JAVA SOURCE | RESOURCE JAVA 句によって、権限が付与された Java ソースまたはリソース・スキーマ・オブジェクトを指定します。

参照： 13-90 ページの「[CREATE JAVA](#)」を参照してください。

システム権限およびオブジェクト権限

表 17-1 システム権限

システム権限名	許可される操作
クラスタ	
CREATE CLUSTER	権限を付与したスキーマ内でのクラスタの作成
CREATE ANY CLUSTER	任意のスキーマ内でのクラスタの作成 CREATE ANY TABLE と同様に動作します。
ALTER ANY CLUSTER	任意のスキーマ内でのクラスタの変更
DROP ANY CLUSTER	任意のスキーマ内でのクラスタの削除
コンテキスト	
CREATE ANY CONTEXT	任意のコンテキスト・ネームスペースの作成
DROP ANY CONTEXT	任意のコンテキスト・ネームスペースの削除
データベース	
ALTER DATABASE	データベースの変更
ALTER SYSTEM	ALTER SYSTEM 文の発行
AUDIT SYSTEM	AUDIT <i>sql_statements</i> 文の発行
データベース・リンク	
CREATE DATABASE LINK	権限を付与したスキーマ内でのプライベート・データベース・リンクの作成
CREATE PUBLIC DATABASE LINK	パブリック・データベース・リンクの作成
DROP PUBLIC DATABASE LINK	パブリック・データベース・リンクの削除
デバッグ	
DEBUG CONNECT SESSION	Java Debug Wire Protocol (JDWP) を使用するデバッガへの現行のセッションの接続

注意: ANY オブジェクトの権限 (CREATE ANY CLUSTER など) を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ `o7_dictionary_accessibility` を `FALSE` に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。

表 17-1 システム権限（続き）

システム権限名	許可される操作
DEBUG ANY PROCEDURE	任意のデータベース・オブジェクトのすべての PL/SQL コードおよび Java コードのデバッグ、アプリケーションによって実行されたすべての SQL 文に関する情報の表示 注意： この権限を付与することは、データベース内の適用可能なすべてのオブジェクトについて DEBUG オブジェクト権限を付与することと同じです。
ディメンション	
CREATE DIMENSION	権限を付与したスキーマ内でのディメンションの作成
CREATE ANY DIMENSION	任意のスキーマ内でのディメンションの作成
ALTER ANY DIMENSION	任意のスキーマ内でのディメンションの変更
DROP ANY DIMENSION	任意のスキーマ内でのディメンションの削除
ディレクトリ	
CREATE ANY DIRECTORY	ディレクトリ・データベース・オブジェクトの作成
DROP ANY DIRECTORY	ディレクトリ・データベース・オブジェクトの削除
索引タイプ	
CREATE INDEXTYPE	権限を付与したスキーマ内での索引タイプの作成
CREATE ANY INDEXTYPE	任意のスキーマ内での索引タイプの作成
ALTER ANY INDEXTYPE	任意のスキーマ内での索引タイプの変更
DROP ANY INDEXTYPE	任意のスキーマ内での索引タイプの削除
EXECUTE ANY INDEXTYPE	任意のスキーマ内での索引タイプの参照
索引	
CREATE ANY INDEX	任意のスキーマでの任意の表へのドメイン索引または索引の作成
ALTER ANY INDEX	任意のスキーマでの索引の変更
DROP ANY INDEX	任意のスキーマでの索引の削除
QUERY REWRITE	マテリアライズド・ビューまたは索引が権限を付与したスキーマ内の表およびビューを参照している場合のそのマテリアライズド・ビューを使用したリライト、ファンクション索引の作成
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
GLOBAL QUERY REWRITE	マテリアライズド・ビューまたは索引が任意のスキーマ内の表またはビューを参照している場合のそのマテリアライズド・ビューを使用したリライト、ファンクション索引の作成
ライブラリ	
CREATE LIBRARY	権限を付与したスキーマ内での外部プロシージャまたはファンクション・ライブラリの作成
CREATE ANY LIBRARY	任意のスキーマ内での外部プロシージャまたはファンクション・ライブラリの作成
DROP ANY LIBRARY	任意のスキーマ内での外部プロシージャまたはファンクション・ライブラリの削除
マテリアライズド・ビュー	
CREATE MATERIALIZED VIEW	権限を付与したスキーマ内でのマテリアライズド・ビューの作成
CREATE ANY MATERIALIZED VIEW	任意のスキーマでのマテリアライズド・ビューの作成
ALTER ANY MATERIALIZED VIEW	任意のスキーマでのマテリアライズド・ビューの変更
DROP ANY MATERIALIZED VIEW	任意のスキーマでのマテリアライズド・ビューの削除
QUERY REWRITE	マテリアライズド・ビューまたは索引が権限を付与したスキーマ内の表およびビューを参照している場合のそのマテリアライズド・ビューを使用したリライト、ファンクション索引の作成
GLOBAL QUERY REWRITE	マテリアライズド・ビューまたは索引が任意のスキーマ内の表またはビューを参照している場合のそのマテリアライズド・ビューを使用したリライト、ファンクション索引の作成
ON COMMIT REFRESH	データベースの任意の表に対する REFRESH ON COMMIT モードのマテリアライズド・ビューの作成 データベースの任意の表に対する REFRESH ON DEMAND モードのマテリアライズド・ビューの、REFRESH ON COMMIT モードのマテリアライズド・ビューへの変更
FLASHBACK ANY TABLE	任意のスキーマ内の任意の表、ビュー、マテリアライズド・ビューでの SQL フラッシュバック問合せの発行（この権限は、DBMS_FLASHBACK プロシージャの実行には不要）
演算子	
CREATE OPERATOR	権限を付与したスキーマ内での演算子およびバインディングの作成
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
CREATE ANY OPERATOR	任意のスキーマ内での演算子およびバインディングの作成
DROP ANY OPERATOR	任意のスキーマ内での演算子の削除
EXECUTE ANY OPERATOR	任意のスキーマ内での演算子の参照
アウトライン	
CREATE ANY OUTLINE	アウトラインを使用する任意のスキーマ内で使用するパブリック・アウトラインの作成
ALTER ANY OUTLINE	アウトラインの変更
DROP ANY OUTLINE	アウトラインの削除
プロシージャ	
CREATE PROCEDURE	権限を付与したスキーマ内でのストアド・プロシージャ、ストアド・ファンクション、ストアド・パッケージの作成
CREATE ANY PROCEDURE	任意のスキーマ内でのストアド・プロシージャ、ストアド・ファンクション、ストアド・パッケージの作成
ALTER ANY PROCEDURE	任意のスキーマ内でのストアド・プロシージャ、ストアド・ファンクション、ストアド・パッケージの変更
DROP ANY PROCEDURE	任意のスキーマ内でのストアド・プロシージャ、ストアド・ファンクション、ストアド・パッケージの削除
EXECUTE ANY PROCEDURE	プロシージャまたはファンクションの実行（スタンドアロンまたはパッケージ） 任意のスキーマ内でのパブリック・パッケージ変数の参照
プロファイル	
CREATE PROFILE	プロファイルの作成
ALTER PROFILE	プロファイルの変更
DROP PROFILE	プロファイルの削除
ロール	
CREATE ROLE	ロールの作成
ALTER ANY ROLE	データベース内の任意のロールの変更
DROP ANY ROLE	ロールの削除
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
GRANT ANY ROLE	データベース内の任意のロールの付与
ロールバック・セグメント	
CREATE ROLLBACK SEGMENT	ロールバック・セグメントの作成
ALTER ROLLBACK SEGMENT	ロールバック・セグメントの変更
DROP ROLLBACK SEGMENT	ロールバック・セグメントの削除
順序	
CREATE SEQUENCE	権限を付与したスキーマ内での順序の作成
CREATE ANY SEQUENCE	任意のスキーマ内での順序の作成
ALTER ANY SEQUENCE	データベース内の任意の順序の変更
DROP ANY SEQUENCE	任意のスキーマ内での順序の削除
SELECT ANY SEQUENCE	任意のスキーマ内での順序の参照
セッション	
CREATE SESSION	データベースへの接続
ALTER RESOURCE COST	セッション・リソースに対するコストの設定
ALTER SESSION	ALTER SESSION 文の発行
RESTRICTED SESSION	SQL*Plus の STARTUP RESTRICT 文を使用した、インスタンス起動後のログイン
スナップショット「マテリアライズド・ビュー」を参照。	
シノニム	
CREATE SYNONYM	権限を付与したスキーマ内でのシノニムの作成
CREATE ANY SYNONYM	任意のスキーマ内でのプライベート・シノニムの作成
CREATE PUBLIC SYNONYM	パブリック・シノニムの作成
DROP ANY SYNONYM	任意のスキーマ内でのプライベート・シノニムの削除
DROP PUBLIC SYNONYM	パブリック・シノニムの削除
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
表	
注意： 外部表の場合、有効なシステム権限は、CREATE ANY TABLE、ALTER ANY TABLE、DROP ANY TABLE および SELECT ANY TABLE です。	
CREATE TABLE	権限を付与したスキーマ内での表の作成
CREATE ANY TABLE	任意のスキーマ内での表の作成なお、表が設定されるスキーマの所有者は、表領域内にその表を定義するための割当て制限が必要です。
ALTER ANY TABLE	スキーマ内の任意の表またはビューの変更
BACKUP ANY TABLE	エクスポート・ユーティリティを使用した他のユーザーのスキーマからのオブジェクトの増分エクスポート
DELETE ANY TABLE	任意のスキーマ内での表、表パーティション、またはビューからの行の削除
DROP ANY TABLE	任意のスキーマ内での表または表パーティションの削除または切捨て
INSERT ANY TABLE	任意のスキーマ内の表またはビューへの行の挿入
LOCK ANY TABLE	任意のスキーマ内の表またはビューのロック
SELECT ANY TABLE	任意のスキーマ内の表、ビューまたはマテリアライズド・ビューの問合せ
FLASHBACK ANY TABLE	任意のスキーマ内の任意の表、ビュー、マテリアライズド・ビューでの SQL フラッシュバック問合せの発行（この権限は、DBMS_FLASHBACK プロシージャの実行には不要）
UPDATE ANY TABLE	任意のスキーマ内の表またはビューの行の更新
表領域	
CREATE TABLESPACE	表領域の作成
ALTER TABLESPACE	表領域の変更
DROP TABLESPACE	表領域の削除
MANAGE TABLESPACE	表領域のオンラインとオフラインの切替え、表領域のバックアップの開始および終了の制御
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
UNLIMITED TABLESPACE	任意の表領域の無制限な使用この権限は、設定されている任意の割当て制限を上書きします。ユーザーからこの権限を取り消した場合、ユーザーのスキーマ・オブジェクトはそのまま残りますが、表領域の割当て制限が許可されないかぎり、それ以上表領域を割り当てることはできません。このシステム権限をロールに付与することはできません。
トリガー	
CREATE TRIGGER	権限を付与したスキーマ内でのデータベース・トリガーの作成
CREATE ANY TRIGGER	任意のスキーマ内でのデータベース・トリガーの作成
ALTER ANY TRIGGER	任意のスキーマ内でのデータベース・トリガーの使用可能化、使用禁止化またはコンパイル
DROP ANY TRIGGER	任意のスキーマ内でのデータベース・トリガーの削除
ADMINISTER DATABASE TRIGGER	データベース内でのトリガーの作成（CREATE TRIGGER または CREATE ANY TRIGGER 権限が必要）
型	
CREATE TYPE	権限を付与したスキーマ内でのオブジェクト型およびオブジェクト型本体の作成
CREATE ANY TYPE	任意のスキーマ内でのオブジェクト型およびオブジェクト型本体の作成
ALTER ANY TYPE	任意のスキーマ内でのオブジェクト型の変更
DROP ANY TYPE	任意のスキーマ内でのオブジェクト型およびオブジェクト型本体の削除
EXECUTE ANY TYPE	特定のユーザーに付与した場合、任意のスキーマ内でのオブジェクト型およびコレクション型を使用および参照した、任意のスキーマ内のオブジェクト型メソッドの起動 EXECUTE ANY TYPE をロールに付与した場合、使用可能なロールを保持したユーザーは、任意のスキーマ内のオブジェクト型メソッドを起動できません。
UNDER ANY TYPE	非最終オブジェクト型のサブタイプの作成
注意： ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
ユーザー	
CREATE USER	ユーザーの作成この権限により、次の操作を実行できます。 ■ 任意の表領域に対する割当て制限の設定 ■ デフォルトの表領域および一時表領域の設定 ■ CREATE USER 文の一部としてのプロファイルの設定
ALTER USER	任意のユーザーの変更この権限により、次の操作を実行できます。 ■ 他のユーザーのパスワードまたは認証方法の変更 ■ 任意の表領域に対する割当て制限の設定 ■ デフォルトの表領域および一時表領域の設定 ■ プロファイルおよびデフォルト・ロールの設定
BECOME USER	別のユーザーになること（全データベース・インポートを実行するユーザーに必要）
DROP USER	ユーザーの削除
ビュー	
CREATE VIEW	権限を付与したスキーマ内でのビューの作成
CREATE ANY VIEW	任意のスキーマ内でのビューの作成
DROP ANY VIEW	任意のスキーマ内でのビューの削除
UNDER ANY VIEW	オブジェクト・ビューのサブビューの作成
FLASHBACK ANY TABLE	任意のスキーマ内の任意の表、ビュー、マテリアライズド・ビューでの SQL フラッシュバック問合せの発行（この権限は、DBMS_FLASHBACK プロシージャの実行には不要）
その他	
ANALYZE ANY	任意のスキーマ内の任意の表、クラスタ、索引の分析
AUDIT ANY	AUDIT <i>schema_objects</i> 文を使用した、任意のスキーマ内の任意のオブジェクトの監査
COMMENT ANY TABLE	任意のスキーマ内の任意の表、ビュー、列についてのコメントの記述
注意: ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。	

表 17-1 システム権限（続き）

システム権限名	許可される操作
EXEMPT ACCESS POLICY	ファイングレイン・アクセス・コントロールの回避 注意： 強力なシステム権限で、権限受領者がアプリケーション駆動のセキュリティ・ポリシーを回避できます。データベース管理者がこの権限を付与する場合は、注意が必要です。
FORCE ANY TRANSACTION	ローカル・データベース内の、インダウト分散トランザクションのコミットまたはロールバック 分散トランザクション・エラーを意図的に発生させます。
FORCE TRANSACTION	ローカル・データベース内の、インダウト分散トランザクションのコミットまたはロールバック
GRANT ANY OBJECT PRIVILEGE	任意のオブジェクト権限の付与 GRANT ANY OBJECT PRIVILEGE 権限を持つオブジェクト所有者または他のユーザーにより付与されたオブジェクト権限を取り消します。
GRANT ANY PRIVILEGE	任意のシステム権限の付与
RESUMABLE	再開可能な領域割当ての使用可能化
SELECT ANY DICTIONARY	SYS スキーマ内のデータ・ディクショナリ・オブジェクトへの問合せ 初期化パラメータ O7_DICTIONARY_ACCESSIBILITY のデフォルトの FALSE 設定を選択的に上書きします。
SYSDBA	STARTUP および SHUTDOWN 操作の実行 ALTER DATABASE: オープン、マウント、バックアップまたはキャラクタ・セットの変更 CREATE DATABASE ARCHIVELOG および RECOVERY CREATE SPFILE RESTRICTED SESSION 権限を含みます。
SYSOPER	STARTUP および SHUTDOWN 操作の実行 ALTER DATABASE OPEN MOUNT BACKUP ARCHIVELOG および RECOVERY CREATE SPFILE RESTRICTED SESSION 権限を含みます。

注意：ANY オブジェクトの権限（CREATE ANY CLUSTER など）を付与した場合、SYS スキーマを含めたすべてのスキーマ内で、そのタイプのオブジェクトにアクセスできるようになります。SYS スキーマ内のオブジェクトへのアクセスを禁止するには、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定します。ANY オブジェクトに付与された権限によって、SYS 以外のスキーマにアクセスできるようになります。

表 17-2 Oracle によって定義されたロール

定義されたロール	用途
CONNECT、RESOURCE および DBA	以前のリリースとの互換性を確保します。DBA_SYS_PRIVS データ・ディクショナリ・ビューを問い合わせることによって、これらのロールにまとめられた権限を確認できます。 注意：データベースのセキュリティを維持するために、独自のロールを設計することをお勧めします。これらのロールは、今後の Oracle のリリースでは自動的に作成されない可能性があります。 参照：このビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。
DELETE_CATALOG_ROLE EXECUTE_CATALOG_ROLE SELECT_CATALOG_ROLE	データ・ディクショナリ・ビューおよびパッケージにアクセスします。 参照：これらのロールの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
EXP_FULL_DATABASE IMP_FULL_DATABASE	インポート / エクスポート・ユーティリティに有効です。 参照：これらのロールの詳細は、『Oracle9i データベース・ユーティリティ』を参照してください。
AQ_USER_ROLE AQ_ADMINISTRATOR_ROLE	Oracle のアドバンスト・キューイング機能を使用する場合、これらのロールが必要です。 参照：これらのロールの詳細は、『Oracle9i アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
SNMPAGENT	このロールは、Enterprise Manager/Intelligent Agent で使用します。 参照：『Oracle Enterprise Manager 管理者ガイド』を参照してください。
RECOVERY_CATALOG_OWNER	リカバリ・カタログを所有するユーザーを作成する場合、このロールが必要です。 参照：リカバリ・カタログの詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。
HS_ADMIN_ROLE	Oracle の異種機間サービス機能を使用する DBA がデータ・ディクショナリにある適切な表にアクセスする場合、このロールが必要です。 参照：詳細は、『Oracle9i Heterogeneous Connectivity Administrator’s Guide』および『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表 17-3 特定のオブジェクトで使用可能なオブジェクト権限

オブジェクト 権限	表	ビュー	順序	プロシージャ、 ファンクション、 パッケージ ^a	マテリア ライズド・ ビュー	ディレ クトリ	ライ ブラリ	ユーザ 定義型	演算子	索引 タイプ
ALTER	X	—	X	—	—	—	—	—	—	—
DELETE	X	X	—	—	X ^b	—	—	—	—	—
EXECUTE	—	—	—	X	—	—	X	X	X	X
DEBUG	X	X	—	X	—	—	—	X	—	—
FLASHBACK	X	X	—	—	X	—	—	—	—	—
INDEX	X	—	—	—	—	—	—	—	—	—
INSERT	X	X	—	—	X ^b	—	—	—	—	—
ON COMMIT REFRESH	X	—	—	—	—	—	—	—	—	—
QUERY REWRITE	X	—	—	—	—	—	—	—	—	—
READ	—	—	—	—	—	X	—	—	—	—
REFERENCES	X	X	—	—	—	—	—	—	—	—
SELECT	X	X	X	—	X	—	—	—	—	—
UNDER	—	X	—	—	—	—	—	X	—	—
UPDATE	X	X	—	—	X ^b	—	—	—	—	—
WRITE	—	—	—	—	—	X	—	—	—	—

^a Oracle では、Java クラス、ソースまたはリソースを、オブジェクト権限の付与のためのプロシージャとして扱います。

^b DELETE、INSERT および UPDATE 権限は、更新可能なマテリアライズド・ビューにのみ付与できます。

表 17-4 オブジェクト権限とその権限によって許可される操作

オブジェクト権限	許可される操作
次の 表権限 は、表の操作を許可します。次のいずれかのオブジェクト権限がある場合は、LOCK TABLE 文を使用して任意のロック・モードで表をロックできます。	
注意： 外部表に有効な権限は、ALTER および SELECT のみです。	
ALTER	ALTER TABLE 文での表定義の変更
DELETE	DELETE 文での表の行の削除 注意： 表がリモート・データベースにある場合は、DELETE 権限とともに表に対する SELECT 権限を付与する必要があります。
DEBUG	デバッガを介したアクセス ■ 表に定義されているトリガーの本体の PL/SQL コード ■ 表を直接参照する SQL 文に関する情報
INDEX	CREATE INDEX 文での表の索引の作成
INSERT	INSERT 文での表への新しい行の追加
REFERENCES	表参照制約の作成この権限はロールには付与できません。
SELECT	SELECT 文での表の間合せ
UPDATE	UPDATE 文での表のデータの変更 注意： 表がリモート・データベースにある場合は、UPDATE 権限とともに表に対する SELECT 権限を付与する必要があります。
次の ビュー権限 は、ビューの操作を許可します。次のいずれかのオブジェクト権限がある場合は、LOCK TABLE 文を使用して任意のロック・モードでビューをロックできます。	
ビューの権限を付与する場合、そのビューのすべての実表に関して Grant Option 付きの権限が必要です。	
DEBUG	デバッガを介したアクセス ■ ビューに定義されているトリガーの本体の PL/SQL コード ■ ビューを直接参照する SQL 文に関する情報
DELETE	DELETE 文でのビューの行の削除
INSERT	INSERT 文でのビューへの新しい行の追加
REFERENCES	ビューへの外部キー制約の定義
SELECT	SELECT 文でのビューの間合せ
UNDER	ビューのサブビューの作成このビューの直属のスーパービューに With Grant Option 付きの UNDER ANY VIEW 権限を持つ場合のみ、このオブジェクト権限を付与できます。

表 17-4 オブジェクト権限とその権限によって許可される操作（続き）

オブジェクト権限	許可される操作
UPDATE	UPDATE 文でのビューのデータの変更
次の 順序権限 は、順序の操作を許可します。	
ALTER	ALTER SEQUENCE 文での順序定義の変更
SELECT	CURRVAL 疑似列および NEXTVAL 疑似列を使用した順序の値の検査および増分
プロシージャ、ファンクションおよびパッケージ権限 は、プロシージャ、ファンクションおよびパッケージの操作を許可します。この権限は、 Java ソース、クラスおよびリソース にも適用されます。Oracle では、これらはオブジェクト権限の付与のために生成されたプロシージャとして扱われます。	
DEBUG	プロシージャ、ファンクションまたはパッケージに定義されたすべてのパブリック変数、非パブリック変数、メソッドおよび型へのデバッグを介したアクセス プロシージャ、ファンクションまたはパッケージ内の行または指示境界へのブレークポイントの設定、またはこれらの場所のいずれかでの停止この権限は、メソッドまたはパッケージの仕様部および本体の宣言にアクセスする権限を付与します。 注意：JDWP インタフェースを使用した PL/SQL および Java ルーチンのデバッグは、現在セッション・ユーザーが DEBUG ANY PROCEDURE システム権限を介して付与されたすべてのオブジェクトへのデバッグ権限を持っている場合のみ使用できます。 DEBUG ANY PROCEDURE システム権限は、現在データベース・セッションと JDWP ベースのデバッグ間に確立された接続許可を判断する場合に、DEBUG CONNECT SESSION システム権限とともにチェックされます。ユーザーが両方のシステム権限を持っていない場合、接続は許可されません。
EXECUTE	プロシージャまたはファンクションのコンパイルまたは直接実行、あるいはパッケージの仕様部に宣言されたプログラム・オブジェクトへのアクセス プロシージャ、ファンクションまたはパッケージに定義されたパブリック変数、型およびメソッドへのデバッグを介したアクセスこの権限は、メソッドまたはパッケージの仕様部のみの宣言にアクセスする権限を付与します。 注意：プロシージャ、ファンクションまたはパッケージを間接的に実行する場合、ユーザーはこの権限を持つ必要はありません。 参照：『Oracle9i データベース概要』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
次の マテリアライズド・ビュー権限 は、マテリアライズド・ビューについての操作を許可します。	
ON COMMIT REFRESH	指定した表に対する REFRESH ON COMMIT モードのマテリアライズド・ビューの作成
QUERY REWRITE	指定した表で使用するクエリー・リライトに対するマテリアライズド・ビューの作成

表 17-4 オブジェクト権限とその権限によって許可される操作（続き）

オブジェクト権限	許可される操作
SELECT	SELECT 文でのマテリアライズド・ビューの間合せ
シノニム権限は、基本オブジェクトに対して付与される権限と同じです。シノニムに権限を付与することは、基本オブジェクトに権限を付与することと同じです。同様に、基本オブジェクトに対して権限を付与することは、オブジェクトのすべてのシノニムに権限を付与することと同じです。あるユーザーにシノニムの権限を付与した場合、そのユーザーは、シノニム名または基本オブジェクト名を SQL 文に指定して、その権限を使用できます。	
次のディレクトリ権限では、ディレクトリ・オブジェクトをポインタとして使用することにより、オペレーティング・システムのディレクトリに格納されている各ファイルにデータベースから安全にアクセスできるようになります。このディレクトリ・オブジェクトには、ファイルが格納されているオペレーティング・システムのディレクトリへのフルパス名が定義されています。これらのファイルは実際にはデータベース外に格納されているため、Oracle サーバーの各プロセスは、ファイル・システム・サーバーに対して適切なファイル・アクセス権限も持っている必要があります。オペレーティング・システムに対するオブジェクト権限ではなく、ディレクトリ・データベース・オブジェクトに対するオブジェクト権限を個々のデータベース・ユーザーに付与することによって、Oracle はファイル運用時のセキュリティを実現できます。	
READ	ディレクトリ内のファイルの読み込み
WRITE	ディレクトリ内のファイルの書き込み外部表に接続する場合のみに役立ちます。これによって、権限受領者は、外部表のエージェントがディレクトリに書き込めるのがログ・ファイルなのか不良ファイルなのかを判断できます。 制限事項：この権限によって、権限受領者が BFILE に書き込むことはできません。
次のライブラリ権限は、ライブラリの操作を許可します。	
EXECUTE	特定のオブジェクトの使用および参照、また、そのメソッドの呼出し
次のオブジェクト型権限は、データベース・オブジェクト型の操作を許可します。	
DEBUG	オブジェクト型に定義されたすべてのパブリック変数、非パブリック変数、メソッドおよび型へのデバッグを介したアクセス 型本体内の行または指示境界へのブレークポイントの設定、またはこれらの場所のいずれかでの停止
EXECUTE	特定のオブジェクトの使用と参照、およびそのメソッドの呼出し オブジェクト型に定義されたパブリック変数、型およびメソッドへのデバッグを介したアクセス
UNDER	型のサブタイプの作成この型の直属のスーパータイプに With Grant Option 付きの UNDER ANY TYPE 権限を持つ場合のみ、このオブジェクト権限を付与できます。
次の索引タイプ権限は、索引タイプの操作を許可します。	
EXECUTE	索引タイプの参照
次の演算子権限は、ユーザー定義演算子の操作を許可します。	
EXECUTE	演算子の参照

例

ユーザーに対してシステム権限を付与する例： サンプル・ユーザー hr に CREATE SESSION システム権限を付与し、hr が Oracle にログインできるようにするには、次の文を発行します。

```
GRANT CREATE SESSION
  TO hr;
```

ロールに対してシステム権限を付与する例： 次の文は、データ・ウェアハウス管理者ロール（14-75 ページの「[ロールの作成例](#)」で作成）に、適切なシステム権限を付与します。

```
GRANT
  CREATE ANY MATERIALIZED VIEW
  , ALTER ANY MATERIALIZED VIEW
  , DROP ANY MATERIALIZED VIEW
  , QUERY REWRITE
  , GLOBAL QUERY REWRITE
  TO dw_manager
  WITH ADMIN OPTION;
```

dw_manager の権限ドメインには、マテリアライズド・ビューに関連するシステム権限が含まれます。

Admin Option 付きロールを付与する例： サンプル・ユーザー sh に Admin Option 付きの dw_manager ロールを付与するには、次の文を発行します。

```
GRANT dw_manager
  TO sh
  WITH ADMIN OPTION;
```

dw_manager ロールによって、sh は次の操作を実行できます。

- ロールを使用可能にして、CREATE MATERIALIZED VIEW システム権限を含むそのロールの権限ドメインに登録されている権限の使用
- 他のユーザーに対するそのロールの付与および取消し
- ロールの削除

ロールに対してオブジェクト権限を付与する例： 次の文は、データ・ウェアハウス・ユーザー・ロール（14-75 ページの「[ロールの作成例](#)」で作成）に、SELECT オブジェクト権限を付与します。

```
GRANT SELECT ON sh.sales TO warehouse_user;
```

ロールに対してロールを付与する例： 次の文は、dw_manager ロールに warehouse_user ロール（両方とも、14-75 ページの「[ロールの作成例:](#)」で作成）を付与します。

```
GRANT warehouse_user TO dw_manager;
```

dw_manager ロールには、warehouse_user ロールのドメインにあるすべての権限が含まれます。

ディレクトリへのオブジェクト権限を付与する例： 次の文は、ユーザー hr にディレクトリ bfile_dir に対する READ 権限を Grant Option 付きで付与します。

```
GRANT READ ON DIRECTORY bfile_dir TO hr
WITH GRANT OPTION;
```

ユーザーに対して表へのオブジェクト権限を付与する 例 次の文は、ユーザー hr に対して、oe.bonuses 表（17-77 ページの「[表へのマージ例:](#)」で作成）についてのすべての権限を Grant Option 付きで付与します。

```
GRANT ALL ON bonuses TO hr
WITH GRANT OPTION;
```

この結果、hr は次の操作が実行できます。

- bonuses 表に対するすべての権限の使用
- 他のユーザーまたはロールに対する、bonuses 表についての権限の付与

ビューへのオブジェクト権限を付与する例： 次の文は、ビュー emp_view（16-49 ページの「[ビューの作成例:](#)」で作成）についての SELECT 権限および UPDATE 権限をすべてのユーザーに付与します。

```
GRANT SELECT, UPDATE
ON emp_view TO PUBLIC;
```

この結果、すべてのユーザーが、従業員の詳細についてのビューを問合せおよび更新できるようになります。

別のスキーマの順序に対してオブジェクト権限を付与する例： 次の文は、ユーザー hr に対して、スキーマ oe 内の customers_seq 順序の SELECT 権限を付与します。

```
GRANT SELECT
ON oe.customers_seq TO hr;
```

ユーザー hr は、次の文を指定して、順序の次の値を作成できるようになります。

```
SELECT oe.customers_seq.NEXTVAL
FROM DUAL;
```


別々の列に複数のオブジェクト権限を付与する例： 次の文は、ユーザー oe に、スキーマ hr にある employees 表の employee_id 列に対する REFERENCES 権限、および employee_id、salary、commission_pct 列に対する UPDATE 権限を付与します。

```
GRANT REFERENCES (employee_id),
      UPDATE (employee_id, salary, commission_pct)
  ON hr.employees
  TO oe;
```

この結果、ユーザー oe は、employee_id 列、salary 列および commission_pct 列の値を更新できるようになります。oe は、employee_id 列を参照する参照整合性制約を定義することもできます。ただし、GRANT 文にはこれらの列のみが指定されているため、ユーザー oe は employees 表の他の列を操作できません。

たとえば、oe は制約付きの表を作成できます。

```
CREATE TABLE dependent
  (dependno    NUMBER,
   dependname  VARCHAR2(10),
   employee    NUMBER
   CONSTRAINT in_emp REFERENCES hr.employees(employee_id) );
```

スキーマ hr 内の employees 表の従業員に対応する dependent 表の依存性が、制約 in_emp によって保証されます。

INSERT

用途

INSERT 文を使用すると、表、ビューの実表、パーティション表のパーティション、コンポジット・パーティション表のサブパーティション、オブジェクト表またはオブジェクト・ビューの実表に、行を追加できます。

前提条件

表に行を挿入する場合は、その表が自分のスキーマ内にある必要があります。自分のスキーマ内にない場合は、その表に対する INSERT 権限が必要です。

ビューの実表に行を挿入する場合、ビューが定義されているスキーマの所有者には、その実表に対する INSERT 権限が必要です。また、他のユーザーのスキーマ内のビューに行を挿入する場合は、そのビューに対する INSERT 権限が必要です。

INSERT ANY TABLE システム権限があれば、任意の表または任意のビューの実表に行を挿入できます。

従来型 INSERT およびダイレクト・パス・インサート

表、パーティションまたはビューにデータを挿入するために使用する INSERT 文には、従来型 INSERT およびダイレクト・パス・インサートの 2 種類があります。従来型 INSERT 文を発行すると、表の空き領域を再利用して挿入され、参照整合性制約が維持されます。ダイレクト・パス・インサートの場合、表の既存データの後に挿入するデータが追加されます。データは、バッファ・キャッシュを回避してデータ・ファイルに直接書き込まれます。既存データの空き領域は再利用されません。これによって挿入操作中のパフォーマンスが向上します。また、これは Oracle のダイレクト・パスのローダー・ユーティリティ、SQL*Loader の機能に似ています。

ダイレクト・パス・インサートには、次の制限があります。これらの制限に違反する場合、他にエラーがないかぎりメッセージが戻されず、従来型 INSERT がシリアルで実行されます。

- DML 文の有無にかかわらず、1 つのトランザクションで複数のダイレクト・パス・インサート文を使用できます。ただし、ある DML 文が特定の表、パーティションまたは索引を変更した後は、トランザクションの他の DML 文は表、パーティションまたは索引にアクセスできません。
- 同じ表、パーティションまたは索引にアクセスする問合せは、ダイレクト・パス・インサート文の前であれば許可されますが、後は許可されません。
- シリアルまたはパラレル文が、同じトランザクションからダイレクト・パス・インサートによって変更された表にアクセスしようとする場合、エラーが戻され、文が拒否されます。

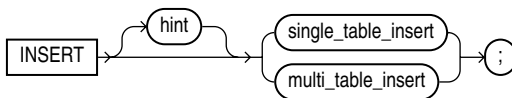
- 初期化パラメータ ROW_LOCKING は、INTENT に設定されません。
- 対象となる表は、索引構成化またはクラスタ化できません。
- 対象となる表は、オブジェクト型または LOB 列を含むことはできません。
- 対象となる表には、トリガーまたは参照整合性制約を定義できません。
- 対象となる表は、レプリケートできません。
- ダイレクト・パス・インサート文を含むトランザクションは、分散できません。

参照：

- ダイレクト・パス・インサートの詳細は、『Oracle9i データベース概要』を参照してください。
- SQL*Loader の詳細は、『Oracle9i データベース・ユーティリティ』を参照してください。
- パラレル・ダイレクト・パス・インサートをチューニングする方法の詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

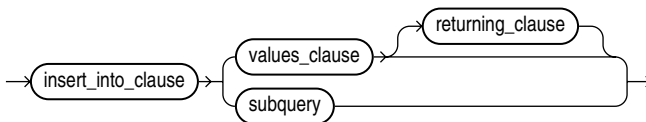
構文

insert::=



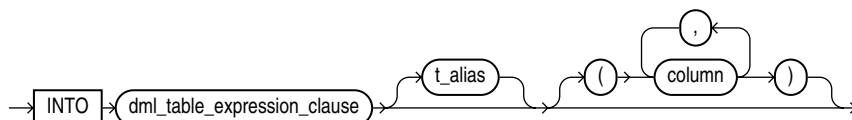
(17-53 ページの「[single_table_insert::=](#)」、17-54 ページの「[multi_table_insert::=](#)」を参照)

single_table_insert::=



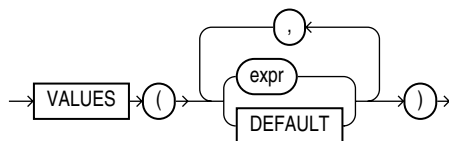
(17-54 ページの「[insert_into_clause::=](#)」、17-54 ページの「[values_clause::=](#)」、17-54 ページの「[returning_clause::=](#)」、18-5 ページの「[subquery::=](#)」を参照)

insert_into_clause::=

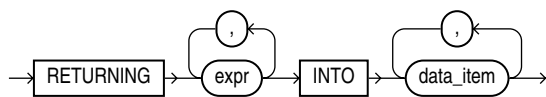


(17-55 ページの「[DML_table_expression_clause::=](#)」を参照)

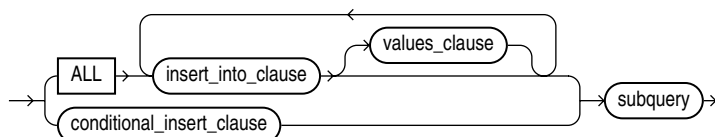
values_clause::=



returning_clause::=

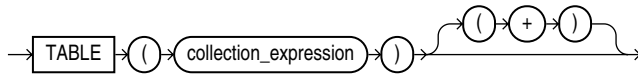


multi_table_insert::=



(17-54 ページの「[insert_into_clause::=](#)」、17-54 ページの「[values_clause::=](#)」、17-55 ページの「[conditional_insert_clause::=](#)」、18-5 ページの「[subquery::=](#)」を参照)

table_collection_expression::=



セマンティクス

hint

文の実行計画を選択する場合に、オプティマイザに指示を与えるためのコメントを指定します。

マルチテーブル・インサートの場合、対象となる表に対して **PARALLEL** ヒントを指定すると、表が **PARALLEL** の指定付きで作成または変更されていない場合、マルチテーブル・インサート文全体がパラレル化されます。**PARALLEL** ヒントを指定しない場合、対象となるすべての表が **PARALLEL** の指定付きで作成または変更されていないかぎり、挿入操作はパラレル化されません。

参照：

- ヒントの構文および説明については、2-86 ページの「**ヒント**」および『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- 17-63 ページの「**マルチテーブル・インサートの制限事項：**」を参照してください。

single_table_insert

シングルテーブル・インサートの場合、明示的に値を指定する、または副問合せで値を検索することによって、表、ビューまたはマテリアライズド・ビューの 1 行に値を挿入します。

副問合せで *flashback_clause* を使用すると、過去のデータを *table* に挿入できます。

参照： この句の詳細は、18-14 ページの「**SELECT**」の「*flashback_clause*」を参照してください。

シングルテーブル・インサートの制限事項： 副問合せで値を検索する場合、副問合せの **SELECT** 構文のリストには **INSERT** 文の列リストと同じ数の列が含まれている必要があります。列リストを指定しない場合は、副問合せで表の各列の値を指定する必要があります。

参照： 17-64 ページの「**表への値の挿入例：**」を参照してください。

insert_into_clause

INSERT INTO 句を使用すると、Oracle がデータを挿入する対象となる表またはオブジェクトを指定できます。

DML_table_expression_clause

INSERT INTO *dml_table_expression_clause* を使用すると、データを挿入するオブジェクトを指定できます。

***dml_table_expression_clause* の制限事項：**

- *table* (または *view* の実表) に、IN_PROGRESS または FAILED とマークされたドメイン索引がある場合は、この文は実行できません。
- 関係する索引パーティションが UNUSABLE とマークされている場合は、パーティションに挿入できません。
- *dml_table_expression_clause* の *subquery* の ORDER BY 句に関して、順序付けは、挿入された行または表の各エクステンツ内のみに保証されています。既存の行に関連する新しい行の順序付けは保証されていません。
- WITH CHECK OPTION を指定してビューを作成した場合、ビューに定義されている問合せを満たす行のみビューに挿入されます。
- 単一の実表を使用してビューを作成した場合、行をビューに挿入し、*returning_clause* を使用してその行の値を取り出せます。
- ビューを定義する問合せに、INSTEAD OF トリガー以外の次のいずれかの構造体が含まれる場合は、そのビューは挿入できません。
 - 集合演算子
 - DISTINCT 演算子
 - 集計ファンクションまたは分析ファンクション
 - GROUP BY、ORDER BY、CONNECT BY または START WITH 句
 - SELECT 構文のリストにあるコレクション式
 - SELECT 構文のリストにある副問合せ
 - 結合（一部の例外を除く）

詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- UNUSABLE のマークが付いている索引、索引パーティションまたは索引サブパーティションを指定する場合、SKIP_UNUSABLE_INDEXES セッション・パラメータが TRUE に設定されていないかぎり、INSERT 文は正常に実行されません。

参照： SKIP_UNUSABLE_INDEXES セッション・パラメータの詳細は、10-2 ページの「ALTER SESSION」を参照してください。

schema 表、ビューまたはマテリアライズド・ビューが含まれるスキーマを指定します。*schema* の指定を省略した場合、オブジェクトが自分のスキーマ内に定義されているとみなされます。

table | view | subquery 行を挿入する表またはオブジェクト表の名前、ビューまたはオブジェクト・ビューの名前、マテリアライズド・ビューの名前、あるいは副問合せから戻された列の名前を指定します。ビューまたはオブジェクト・ビューを指定した場合、Oracle はそのビューの実表に行を挿入します。

挿入される値がオブジェクト表に対する REF の場合、およびそのオブジェクト表に主キー・オブジェクト識別子がある場合、REF を挿入する列は、オブジェクト表に対する参照整合性制約または有効範囲制約を持つ REF 列である必要があります。

table (または *view* の実表) に、1 列以上のドメイン索引がある場合は、この文が適切な索引タイプの挿入ルーチンを実行します。

表に対して INSERT 文を発行した場合、その表に対して定義されている INSERT トリガーが起動します。

参照： これらのルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。

PARTITION (*partition_name*) | SUBPARTITION (*subpartition_name*) 挿入先の *table* (または *view* の実表) 内にあるパーティションまたはサブパーティションの名前を指定します。

挿入する行が特定のパーティションまたはサブパーティションにマップされない場合、Oracle はエラーを戻します。

ターゲットのパーティションおよびサブパーティションの制限事項： この句は、オブジェクト表またはオブジェクト・ビューでは無効です。

dblink 表またはビューが格納されているリモート・データベースへのデータベース・リンクの完全名または部分名を指定します。Oracle の分散機能を使用している場合にのみ、リモート表またはリモート・ビューに行を挿入できます。

dblink を指定しない場合、Oracle は、その表またはビューがローカル・データベース内にあるとみなします。

参照： データベース・リンクの参照方法の詳細は、2-110 ページの「[スキーマ・オブジェクトの構文および SQL 文の構成要素](#)」を参照してください。また、17-65 ページの「[リモート・データベースへの挿入例:](#)」も参照してください。

subquery_restriction_clause *subquery_restriction_clause* を使用すると、次のいずれかの方法で副問合せを制限できます。

WITH READ ONLY 表またはビューが更新禁止であることを示す場合は、WITH READ ONLY を指定します。

WITH CHECK OPTION 副問合せに存在しない行を生成する表またはビューの変更が禁止であることを示す場合は、WITH CHECK OPTION を指定します。

CONSTRAINT *constraint* CHECK OPTION 制約の名前を指定します。この識別子を省略した場合、その制約に SYS_Cn という形式の名前が自動的に割り当てられます。この場合の *n* は、その制約名をデータベース内で一意の名前にする整数です。

参照： 18-32 ページの「[WITH CHECK OPTION 句の使用例:](#)」を参照してください。

table_collection_expression 問合せおよび DML 操作で、*collection_expression* 値を表として扱う場合に、*table_collection_expression* を指定します。*collection_expression* は、副問合せ、列、ファンクションまたはコレクション・コンストラクタのいずれかです。その形式にかかわらず、集合値（ネストした表型または VARRAY 型の値）を戻す必要があります。このようなコレクションの要素抽出プロセスを **コレクション・ネスト解除** といいます。

注意： 以前のリリースの Oracle では、*collection_expression* が副問合せの場合、*table_collection_expression* を「THE subquery」と表現していました。現在、このような表現方法はされていません。

参照： 18-36 ページの「[表のコレクション例:](#)」を参照してください。

t_alias

文内で参照する表、ビューまたは副問合せの**相関名**（別名）を指定します。

表の別名の制限事項： *t_alias* は、マルチテーブル・インサートに指定できません。

column

表またはビューの列を指定します。挿入された行では、このリストにある各列に *values_clause* または副問合せの値が代入されます。

このリストに 1 つ以上の列を指定しない場合、挿入された行の列の値には、表の作成時または最終更新時に指定した列のデフォルト値が使用されます。省略された列のいずれかに NOT NULL 制約があり、デフォルト値がない場合、制約違反のエラーが発生して INSERT 文がロールバックされます。

列リストを指定しない場合は、`values_clause` または問合せに、表の列をすべて指定する必要があります。

参照： 列のデフォルト値の詳細は、15-7 ページの「[CREATE TABLE](#)」を参照してください。

`values_clause`

シングルテーブル・インサート操作の場合、表またはビューに挿入する値の行を指定します。なお、値は、列リスト内の各列について `values_clause` に指定する必要があります。列リストを指定しない場合、`values_clause` または副問合せで、表の各列の値を指定する必要があります。

マルチテーブル・インサート操作の場合、`values_clause` の各式は、副問合せの SELECT 構文のリストによって戻される列を参照する必要があります。`values_clause` を省略すると、副問合せの SELECT 構文のリストによって挿入する値が決定されるため、`insert_into_clause` に対応する列リストと同じ列数を含んでいる必要があります。`insert_into_clause` で列リストを指定しない場合、対象となる表のすべての列に対する値を計算した行を指定する必要があります。

どちらの挿入操作の場合も、`insert_into_clause` で列リストを指定すると、リストの各列に値の句または副問合せからの対応する値が割り当てられます。`values_clause` の任意の値に対して `DEFAULT` を指定できます。表またはビューの対応する列に対してデフォルト値を指定すると、その値が挿入されます。対応する列に対してデフォルト値を指定しない場合、`NULL` が挿入されます。

注意： パラレル・ダイレクト・パス・インサートは、`values_clause` はサポートせず、INSERT 文の副問合せ構文のみサポートします。シリアル・ダイレクト・パスおよびパラレル・ダイレクト・パス・インサートの詳細は、『Oracle9i データベース概要』を参照してください。

挿入値の制限事項：

- オブジェクトにある内部 LOB 属性を空または `NULL` 以外の値で初期化することはできません。つまり、リテラルを使用することはできません。
- BFILE ロケータを `NULL` に、またはディレクトリ別名およびファイル名に初期化するまで、BFILE 値を挿入できません。
- リスト・パーティション表に挿入する場合、1 つのパーティションの `partition_value` リストに存在していないパーティション化キー列に値を挿入できません。
- ビューに挿入する場合は、`DEFAULT` を指定できません。

注意： RAW 列に文字リテラルを挿入する場合、後続の問合せ中に RAW 列にある索引は使用されずに、全表スキャンが行われます。

参照：

- BFILE ロケータの初期化の詳細は、『Oracle Call Interface プログラマーズ・ガイド』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 有効な式の構文の詳細は、4-2 ページの「SQL 式」および 18-4 ページの「SELECT」を参照してください。
- XMLType 表への値の挿入の詳細は、D-11 ページの「SQL 文での XML の使用方法」を参照してください。
- 17-66 ページの「BFILE への挿入例：」、17-65 ページの「置換可能な表および列への挿入例：」、17-66 ページの「TO_LOB ファンクションを使用した挿入例：」、17-65 ページの「順序値の挿入例：」および 17-65 ページの「バインド変数を使用した挿入例：」を参照してください。

returning_clause

DML (INSERT、UPDATE または DELETE) 文に影響される行を取り出します。この句は、表、マテリアライズド・ビュー、および単一の実表を持つビューに指定できます。

単一行で処理する場合、*returning_clause* 付きの DML 文で、処理された行および ROWID を使用している列式、処理された行への REF を取り出し、ホスト変数または PL/SQL 変数に格納します。

複数行で処理する場合、*returning_clause* 付きの DML 文で、式の値、ROWID および処理された行に関連する REF をバインド配列に格納します。

expr *expr* リストの各項目は、適切な構文で表す必要があります。スカラー副問合せ式を除くすべての形式が有効です。

INTO 変更された行の値を、*data_item* リストに指定する変数に格納することを示します。

data_item 取り出された *expr* 値を格納するホスト変数または PL/SQL 変数です。

RETURNING リストの各式については、INTO リストに、対応する型に互換がある PL/SQL 変数またはホスト変数を指定する必要があります。

RETURNING 句の制限事項： 次の操作は実行できません。

- マルチテーブル・インサートへの *returning_clause* の指定。
- パラレル DML またはリモート・オブジェクトでのこの句の使用。
- この句での LONG 型の取出し。
- INSTEAD OF トリガーが定義されたビューに対しての、この句の指定。

参照： BULK COLLECT 句を使用してコレクション変数に複数の値を戻す場合は、『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。

multi_table_insert

マルチテーブル・インサートでは、副問合せの評価によって戻された行から導出され、計算された行が 1 つ以上の表に挿入されます。

注意： 副問合せの SELECT 構文のリストでは、表の別名は定義されていません。そのため、SELECT 構文のリストに依存する句では、表の別名は参照できません。たとえば、式内のオブジェクト列を参照しようとしても、表の別名は参照できません。表の別名とともに式を使用する場合は、列の別名を使用して式を SELECT 構文のリストに挿入してから、マルチテーブル・インサートの VALUES 句または WHEN 条件で列の別名を参照する必要があります。

ALL into_clause

複数の *insert_into_clause* を後ろに伴った ALL を指定すると、**無条件のマルチテーブル・インサート**が実行されます。副問合せによって戻される各行に対して、各 *insert_into_clause* が 1 回実行されます。

conditional_insert_clause

conditional_insert_clause を指定すると、**条件付きのマルチテーブル・インサート**が実行されます。実行する *insert_into_clause* を判断する WHEN 条件を使用して、各 *insert_into_clause* が抽出されます。WHEN 条件の各式は、副問合せの SELECT 構文のリストによって戻される列を参照する必要があります。1 つのマルチテーブル・インサートには、最大 127 個まで WHEN 句の指定が可能です。

ALL ALL を指定すると、他の WHEN 句の評価結果にかかわらず、各 WHEN 句が評価されます。条件が真と評価された各 WHEN 句に対して、対応する INTO 句リストが実行されます。

FIRST FIRST を指定すると、文で指定されている順序で各 WHEN 句が評価されます。真と評価された最初の WHEN 句に対して、対応する INTO 句が実行され、指定された行に対する後続の WHEN 句はスキップされます。

ELSE 句 指定された行に対して、WHEN 句が真と評価されない場合、次のようになります。

- ELSE 句を指定する場合、ELSE 句に関連する INTO 句リストが実行されます。
- 他の句を指定しない場合、行に対して何も実行されません。

参照： 17-66 ページの「マルチテーブル・インサートの例 :」を参照してください。

マルチテーブル・インサートの制限事項：

- 表のみにマルチテーブル・インサートが実行でき、ビューまたはマテリアライズド・ビューには実行できません。
- リモート表には、マルチテーブル・インサートを実行できません。
- マルチテーブル・インサートの実行時、表のコレクション式は指定できません。
- マルチテーブル・インサートでは、すべての *insert_into_clause* に最大 999 列を組み合わせて指定できます。
- マルチテーブル・インサートは、Real Application Clusters 環境の場合、対象となる表が索引構成されている場合、または対象となる表にビットマップ索引が定義されている場合は、パラレル化されません。
- プラン・スタビリティは、マルチ表の INSERT 文にはサポートされません。
- マルチ表の INSERT 文の副問合せに、順序は使用できません。

subquery

表に挿入される行を戻す副問合せを指定します。副問合せによって、INSERT 文の対象となる表を含む任意の表、ビューおよびマテリアライズド・ビューを参照できます。副問合せで選択された行が 1 行もない場合、表に行は挿入されません。

subquery を TO_LOB ファンクションと組み合わせて、LONG 列にある値を、同じ表の異なる列または別の表にある LOB 値に変換できます。ビュー内で LONG を LOB に移行する場合、実表内で移行を実行してからビューに LOB を追加する必要があります。

副問合せを使用する挿入の注意事項：

- *subquery* が、既存のマテリアライズド・ビューと（部分的または完全に）同じビューを戻す場合、Oracle は、*subquery* に指定された 1 つ以上の表のかわりにマテリアライズド・ビュー（クエリー・リライト用）を使用することがあります。

参照： マテリアライズド・ビューおよびクエリー・リライトの詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

- *subquery* がリモート・オブジェクトを参照する場合、参照がローカル・データベース上でオブジェクトにループバックしないかぎり、INSERT はパラレルで実行されますただし、*dml_table_expression_clause* の *subquery* がリモート・オブジェクトを参照する場合は、INSERT はシリアルで実行されます。詳細は、15-50 ページの「[parallel_clause](#)」(「[CREATE TABLE](#)」)を参照してください。

参照：

- 17-65 ページの「[副問合せを持つ値の挿入例：](#)」を参照してください。
- 17-66 ページの「[BFILE への挿入例：](#)」を参照してください。
- BFILE の初期化の詳細は、『Oracle Call Interface プログラマーズ・ガイド』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 有効な式の構文の詳細は、4-2 ページの「[SQL 式](#)」および 18-4 ページの「[SELECT](#)」を参照してください。

例

表への値の挿入例： 次の文は、サンプル表 `departments` に行を挿入します。

```
INSERT INTO departments
VALUES (280, 'Recreation', 121, 1700);
```

`manager_id` 列のデフォルト値が 121 として `departments` 表が作成された場合、次の文を発行できます。

```
INSERT INTO departments
VALUES (280, 'Recreation', DEFAULT, 1700);
```

次の文は、`employees` 表に 6 つの列で構成される行を挿入します。NULL または科学表記の数値を設定されている列がそれぞれ 1 つ含まれています。

```
INSERT INTO employees (employee_id, last_name, email,
hire_date, job_id, salary, commission_pct)
VALUES (207, 'Gregory', 'pgregory@oracle.com',
sysdate, 'PU_CLERK', 1.2E3, NULL);
```

次の文は、前述の例と同じ結果を表しますが、*dml_table_expression_clause* にある副問合せを使用します。

```
INSERT INTO
(SELECT employee_id, last_name, email, hire_date, job_id,
salary, commission_pct FROM employees)
VALUES (207, 'Gregory', 'pgregory@oracle.com',
sysdate, 'PU_CLERK', 1.2E3, NULL);
```

副問合せを持つ値の挿入例： 次の文は、歩合給が給与の 25% 以上の従業員を bonuses 表 (17-77 ページの「[表へのマージ例:](#)」で作成) にコピーします。

```
INSERT INTO bonuses
  SELECT employee_id, salary*1.1
  FROM employees
  WHERE commission_pct > 0.25 * salary;
```

リモート・データベースへの挿入例： 次の文は、データベース・リンク remote がアクセスできるデータベース上の、ユーザー hr が所有する employees 表に行を挿入します。

```
INSERT INTO employees@remote
  VALUES (8002, 'Juan', 'Fernandez', 'juanf@hr.com', NULL,
  TO_DATE('04-OCT-1992', 'DD-MON-YYYY'), 'SH_CLERK', 3000,
  NULL, 121, 20);
```

順序値の挿入例： 次の文は、従業員順序の次の値を持つ行を、departments 表に挿入します。

```
INSERT INTO departments
  VALUES (departments_seq.nextval, 'Entertainment', 162, 1400);
```

バインド変数を使用した挿入例： 次の文は、出力バインド変数 bnd1 および bnd2 に挿入された行の値を戻します。(バインド変数は最初に宣言しておく必要があります。)

```
INSERT INTO employees
  (employee_id, last_name, email, hire_date, job_id, salary)
  VALUES
  (employees_seq.nextval, 'Doe', 'john.doe@oracle.com',
  SYSDATE, 'SH_CLERK', 2400)
  RETURNING salary*12, job_id INTO :bnd1, :bnd2;
```

置換可能な表および列への挿入例： 次の例は、persons 表 (15-63 ページの「[置換可能な表および列のサンプル:](#)」で作成) に挿入します。最初の文は、ルート型 person_t を使用します。2 番目の挿入は、person_t のサブタイプ employee_t を使用し、3 番目の挿入は employee_t のサブタイプ part_time_emp_t を使用します。

```
INSERT INTO persons VALUES (person_t('Bob', 1234));
INSERT INTO persons VALUES (employee_t('Joe', 32456, 12, 100000));
INSERT INTO persons VALUES (
  part_time_emp_t('Tim', 5678, 13, 1000, 20));
```

次の例は、books 表 (15-63 ページの「[置換可能な表および列のサンプル:](#)」で作成) に挿入します。属性値の指定が、置換可能な表の例と同一であることに注意してください。

```
INSERT INTO books VALUES (
  'An Autobiography', person_t('Bob', 1234));
INSERT INTO books VALUES (
```

```
'Business Rules', employee_t('Joe', 3456, 12, 10000));
INSERT INTO books VALUES (
  'Mixing School and Work',
  part_time_emp_t('Tim', 5678, 13, 1000, 20));
```

組込みファンクションおよび条件を使用して、置換可能な表および列からデータを抽出することができます。その例は、6-181 ページの「[TREAT](#)」、6-155 ページの「[SYS_TYPEID](#)」および 5-19 ページの「[IS OF type 条件](#)」を参照してください。

TO_LOB ファンクションを使用した挿入例： 次の例は、LONG データを次の long_tab 表にある LOB 列にコピーします。

```
CREATE TABLE long_tab (pic_id NUMBER, long_pics LONG RAW);
```

まず、LOB を持つ表を作成します。

```
CREATE TABLE lob_tab (pic_id NUMBER, lob_pics BLOB);
```

次に、INSERT ... SELECT を使用して、LONG 列のすべての行にあるデータを、新しく作成した LOB 列にコピーします。

```
INSERT INTO lob_tab
  SELECT pic_id, TO_LOB(long_pics) FROM long_tab;
```

移行が問題なく終了したことを確認した後、long_pics 表を削除できます。別の方法として、表が他の列を含む場合、次のように入力して表から LONG 列を削除できます。

```
ALTER TABLE long_tab DROP COLUMN long_pics;
```

BFILE への挿入例： 次の例では、サンプル表 pm.print_media に行を挿入します。BFILENAME を使用して、サーバーのファイル・システムのバイナリ・ファイルを識別します。

```
CREATE DIRECTORY media_dir AS '/demo/schema/product_media';

INSERT INTO print_media (product_id, ad_id, ad_graphic)
  VALUES (3000, 31001,
    bfilename('MEDIA_DIR', 'modem_comp_ad.gif'));
```

マルチテーブル・インサートの例： 次の例はマルチテーブル・インサートの構文を使用して、サンプル表 sh.sales に、異なる構造の入力表からデータを挿入します。

注意： 例を簡潔に示すために表の列が無視されているため、sales 表の制約は使用禁止になっています。

入力表は次のように構成されています。

```
SELECT * FROM sales_input_table;
```

PRODUCT_ID	CUSTOMER_ID	WEEKLY_ST	SALES_SUN	SALES_MON	SALES_TUE	SALES_WED	SALES_THU	SALES_FRI	SALES_SAT
111	222	01-OCT-00	100	200	300	400	500	600	700
222	333	08-OCT-00	200	300	400	500	600	700	800
333	444	15-OCT-00	300	400	500	600	700	800	900

マルチ表の INSERT 文を次に示します。

```
INSERT ALL
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date, sales_sun)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+1, sales_mon)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+2, sales_tue)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+3, sales_wed)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+4, sales_thu)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+5, sales_fri)
  INTO sales (prod_id, cust_id, time_id, amount)
  VALUES (product_id, customer_id, weekly_start_date+6, sales_sat)
SELECT product_id, customer_id, weekly_start_date, sales_sun,
       sales_mon, sales_tue, sales_wed, sales_thu, sales_fri, sales_sat
FROM sales_input_table;
```

sales 表には次の行のみが存在するものとします。内容は次のとおりです。

```
SELECT * FROM sales;
```

PROD_ID	CUST_ID	TIME_ID	C	PROMO_ID	QUANTITY_SOLD	AMOUNT	COST
111	222	01-OCT-00				100	
111	222	02-OCT-00				200	
111	222	03-OCT-00				300	
111	222	04-OCT-00				400	
111	222	05-OCT-00				500	
111	222	06-OCT-00				600	
111	222	07-OCT-00				700	
222	333	08-OCT-00				200	
222	333	09-OCT-00				300	
222	333	10-OCT-00				400	
222	333	11-OCT-00				500	

222	333 12-OCT-00	600
222	333 13-OCT-00	700
222	333 14-OCT-00	800
333	444 15-OCT-00	300
333	444 16-OCT-00	400
333	444 17-OCT-00	500
333	444 18-OCT-00	600
333	444 19-OCT-00	700
333	444 20-OCT-00	800
333	444 21-OCT-00	900

次の例は、複数の表に挿入します。販売員に様々なサイズの注文に関する情報を提供するとします。小口、中口、大口および特別（超大口）注文についての表を作成し、これらの表にサンプル表 `oe.orders` のデータを移入します。

```
CREATE TABLE small_orders
  (order_id      NUMBER(12)    NOT NULL,
   customer_id   NUMBER(6)     NOT NULL,
   order_total    NUMBER(8,2),
   sales_rep_id  NUMBER(6)
  );

CREATE TABLE medium_orders AS SELECT * FROM small_orders;

CREATE TABLE large_orders AS SELECT * FROM small_orders;

CREATE TABLE special_orders
  (order_id      NUMBER(12)    NOT NULL,
   customer_id   NUMBER(6)     NOT NULL,
   order_total    NUMBER(8,2),
   sales_rep_id  NUMBER(6),
   credit_limit   NUMBER(9,2),
   cust_email     VARCHAR2(30)
  );
```

最初のマルチテーブル・インサートでは、小口、中口および大口注文の表に対してのみ移入を行います。

```
INSERT ALL
  WHEN order_total < 1000000 THEN
    INTO small_orders
  WHEN order_total > 1000000 AND order_total < 2000000 THEN
    INTO medium_orders
  WHEN order_total > 2000000 THEN
    INTO large_orders
  SELECT order_id, order_total, sales_rep_id, customer_id
  FROM orders;
```

large_orders 表への挿入のかわりに、ELSE 句を使用しても、同じ結果が得られます。

```
INSERT ALL
  WHEN order_total < 100000 THEN
    INTO small_orders
  WHEN order_total > 100000 AND order_total < 200000 THEN
    INTO medium_orders
  ELSE
    INTO large_orders
  SELECT order_id, order_total, sales_rep_id, customer_id
  FROM orders;
```

次の例は、前述の例と同様、小口、中口および大口注文の表に挿入し、件数が 2,900,000 を超える注文を special_orders 表に挿入します。この表は、文を単純にするために列の別名を使用する方法も示しています。

```
INSERT ALL
  WHEN ottl < 100000 THEN
    INTO small_orders
      VALUES(oid, ottl, sid, cid)
  WHEN ottl > 100000 and ottl < 200000 THEN
    INTO medium_orders
      VALUES(oid, ottl, sid, cid)
  WHEN ottl > 200000 THEN
    into large_orders
      VALUES(oid, ottl, sid, cid)
  WHEN ottl > 290000 THEN
    INTO special_orders
  SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
    o.sales_rep_id sid, c.credit_limit cl, c.cust_email cem
  FROM orders o, customers c
  WHERE o.customer_id = c.customer_id;
```

最後の例は、FIRST 句を使用して、件数が 2,900,000 を超える注文を special_orders 表に挿入し、これらの注文を large_orders 表から削除します。

```
INSERT FIRST
  WHEN ottl < 100000 THEN
    INTO small_orders
      VALUES(oid, ottl, sid, cid)
  WHEN ottl > 100000 and ottl < 200000 THEN
    INTO medium_orders
      VALUES(oid, ottl, sid, cid)
  WHEN ottl > 290000 THEN
    INTO special_orders
  WHEN ottl > 200000 THEN
    INTO large_orders
      VALUES(oid, ottl, sid, cid)
```

INSERT

```
SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,  
       o.sales_rep_id sid, c.credit_limit cl, c.cust_email cem  
FROM orders o, customers c  
WHERE o.customer_id = c.customer_id;
```

LOCK TABLE

用途

LOCK TABLE 文を使用すると、1 つ以上の表、表パーティションまたはサブパーティションを特定のモードでロックできます。操作中の表またはビューに対する他のユーザーによるアクセスを許可または制限するため、自動ロックを手動で無効にします。

ロックによっては、同じ表に同時に設定できる場合、または表ごとに 1 つのみ設定できる場合があります。

ロックされた表は、トランザクションをコミットするか、全体をロールバックするか、または表をロックする前のセーブポイントにロールバックするまでロックされています。

ロックした場合でも他のユーザーが表を問い合わせることができます。問合せによって表がロックされることはありません。読み込みプログラムは書き込みプログラムをロックすることではなく、書き込みプログラムが読み込みプログラムをロックすることもあります。

参照：

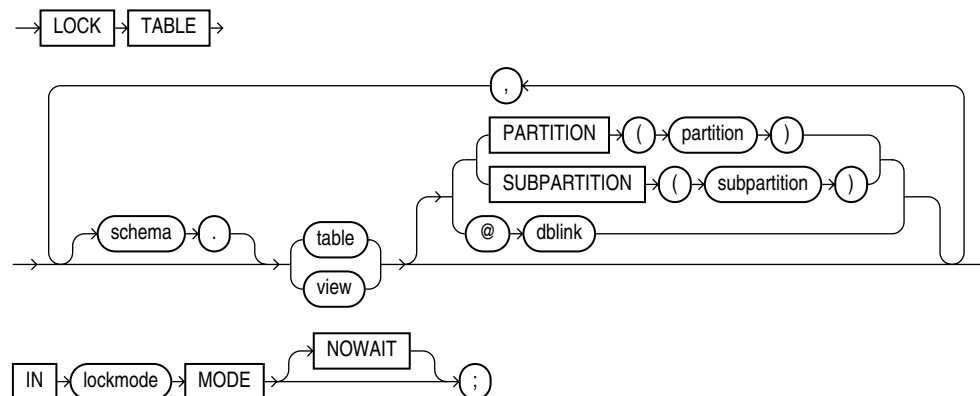
- ロック・モードの相互作用については、『Oracle9i データベース概要』を参照してください。
- 12-70 ページの「[COMMIT](#)」を参照してください。
- 17-97 ページの「[ROLLBACK](#)」を参照してください。
- 18-2 ページの「[SAVEPOINT](#)」を参照してください。

前提条件

表またはビューが自分のスキーマ内にある必要があります。自分のスキーマ内でない場合は、LOCK ANY TABLE システム権限が付与されているか、表またはビューに対するオブジェクト権限が必要です。

構文

lock_table::=



セマンティクス

schema

表またはビューが含まれているスキーマを指定します。*schema* を指定しない場合、表またはビューが自分のスキーマにあるとみなされます。

table / view

ロックする表の名前を指定します。

view を指定した場合、ビューの実表がロックされます。

PARTITION (*partition*) または SUBPARTITION (*subpartition*) を指定する場合、最初にその表を暗黙的にロックします。表ロックは、*partition* または *subpartition* に指定したロックと同じです。ただし、次の 2 つの例外があります。

- SHARE ロックをサブパーティションに指定する場合、Oracle は、表を暗黙的に ROW SHARE ロックします。
- EXCLUSIVE ロックをサブパーティションに指定する場合、Oracle は、表を暗黙的に ROW EXCLUSIVE ロックします。

PARTITION を指定し、*table* がコンポジット・パーティション化されている場合、Oracle は、パーティションのすべてのサブパーティションをロックします。

表のロックの制限事項： *table* が階層の一部である場合、階層のルートである必要があります。

dblink

表またはビューが格納されている、Oracle のリモート・データベースに対するデータベース・リンクを指定します。Oracle 分散機能を使用している場合のみ、リモート・データベースで表およびビューをロックできます。LOCK TABLE 文を使用してロックする表は、すべて同じデータベース上にある必要があります。

dblink を指定しない場合、Oracle は、その表またはビューがローカル・データベース内にあるとみなします。

参照： データベース・リンクの指定方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。

***lockmode* 句**

次のいずれかのモードを指定します。

ROW SHARE ROW SHARE は、ロックされた表への同時アクセスを可能にしますが、排他アクセスのために表全体をロックすることはできなくなります。ROW SHARE は、SHARE UPDATE と同じ意味で、以前のリリースの Oracle との互換性を保つために用意されています。

ROW EXCLUSIVE ROW EXCLUSIVE は、ROW SHARE と同じですが、SHARE モードでロックはできません。ROW EXCLUSIVE ロックは、更新、挿入、削除の実行時に自動的に適用されます。

SHARE UPDATE 「ROW SHARE」を参照してください。

SHARE SHARE は、同時問合せは実行できますが、ロックされた表は更新できません。

SHARE ROW EXCLUSIVE SHARE ROW EXCLUSIVE は、表全体を見る場合に使用します。これを使用すると他のユーザーがその表内の行を見ることはできますが、SHARE モードで表のロックまたは行の更新を行うことはできません。

EXCLUSIVE EXCLUSIVE は、ロックされた表上で問合せは実行できますが、他のアクティビティは実行できません。

NOWAIT

指定した表（または指定したパーティションまたはサブパーティション）が他のユーザーによってすでにロックされている場合、制御をすぐに戻すには NOWAIT を指定します。この場合、表、パーティションまたはサブパーティションが他のユーザーによってロックされていることを示すエラー・メッセージが戻ります。

この句を指定しない場合、Oracle は、表が使用可能になるまで待ち状態になり、表をロックした後に、発行元に制御を戻します。

例

表のロック例： 次の文は、employees 表を排他モードでロックします。他のユーザーがすでに表をロックしている場合でも、待ち状態にはなりません。

```
LOCK TABLE employees
  IN EXCLUSIVE MODE
  NOWAIT;
```

次の文は、データベース・リンク remote を介してアクセスできるリモート表 employees をロックします。

```
LOCK TABLE employees@remote
  IN SHARE MODE;
```


MERGE

用途

MERGE 文を使用すると、1 つの表から行を選択し、その他の表に対して更新および挿入できます。対象の表に更新と挿入のどちらを実行するかは、ON 句の条件に基づきます。

この文は、2 つ以上の操作を組み合わせるときに便利です。DML 文 INSERT および UPDATE を複数指定する必要がなくなります。

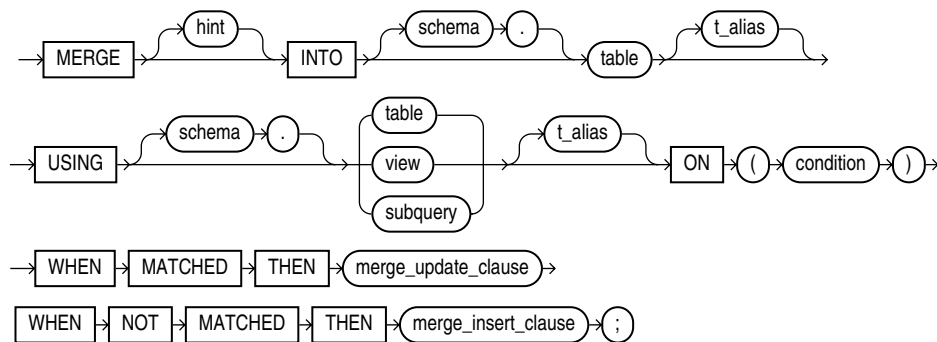
MERGE は、決定的な文です。つまり、同一の MERGE 文で対象となる表の同じ行を複数回更新できません。

前提条件

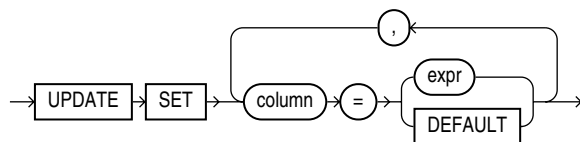
対象となる表に対する INSERT オブジェクト権限と UPDATE オブジェクト権限、およびソース表に対する SELECT 権限が必要です。

構文

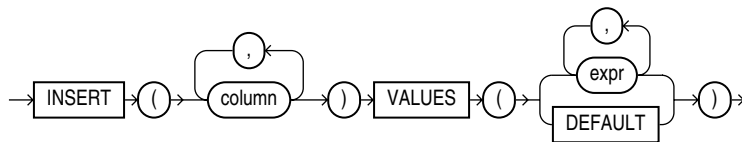
merge::=



merge_update_clause::=



merge_insert_clause::=



セマンティクス

INTO 句

INTO 句を使用すると、更新または挿入の対象となる表を指定できます。

USING 句

USING 句を使用すると、更新または挿入の対象となるデータのソースを指定できます。ソースには、表、ビューまたは副問合せの結果を指定できます。

ON 句

ON 句を使用すると、MERGE の更新操作または挿入操作の条件を指定できます。対象となる表の中で検索条件が真となる各行は、ソース表の対応するデータに基づいて行が更新されます。どの行も条件が真とならない場合、ソース表の対応する行に基づいて対象となる表に行が挿入されます。

WHEN MATCHED | NOT MATCHED

この句を使用すると、ON 句内にある結合条件の結果の応答方法を指定できます。これら 2 つの句は、いずれかの順序で指定できます。

merge_update_clause

merge_update_clause を指定すると、対象となる表の新しい列値を指定できます。ON 句の条件が真となる場合、更新が実行されます。更新が実行されると、対象となる表に定義されているすべての更新トリガーがアクティブになります。

ビューの更新の制限事項：

- ビューを更新する場合は、DEFAULT を指定できません。
- 参照する列は、ON *condition* 句で更新できません。

merge_insert_clause

`merge_insert_clause` を指定すると、ON 句の条件が偽となる場合に対象となる表の列に挿入する値を指定できます。挿入が実行されると、対象となる表に定義されているすべての挿入トリガーがアクティブになります。

ビューへのマージの制限事項： ビューを更新する場合は、DEFAULT を指定できません。

例

表へのマージ例： 次の例は、サンプル・スキーマ `oe` 内に `bonus` 列のデフォルト値を 100 として `bonuses` 表を作成します。次に、`bonuses` 表に販売実績があった (`oe.orders` 表の `sales_rep_id` 列に基づく) すべての従業員を挿入します。最終的に、人事情報マネージャがすべての従業員にボーナスを支給することを決定します。販売実績がなかった従業員には、給与の 1% がボーナスとして支給されます。販売実績があった従業員には、給与の 1% がボーナスに加算されて支給されます。MERGE 文は、これらの変更を 1 行で実装します。

```
CREATE TABLE bonuses (employee_id NUMBER, bonus NUMBER DEFAULT 100);
```

```
INSERT INTO bonuses(employee_id)
  (SELECT e.employee_id FROM employees e, orders o
   WHERE e.employee_id = o.sales_rep_id
   GROUP BY e.employee_id);
```

```
SELECT * FROM bonuses;
```

EMPLOYEE_ID	BONUS
153	100
154	100
155	100
156	100
158	100
159	100
160	100
161	100
163	100

```
MERGE INTO bonuses D
  USING (SELECT employee_id, salary, department_id FROM employees
   WHERE department_id = 80) S
  ON (D.employee_id = S.employee_id)
  WHEN MATCHED THEN UPDATE SET D.bonus = D.bonus + S.salary*.01
  WHEN NOT MATCHED THEN INSERT (D.employee_id, D.bonus)
  VALUES (S.employee_id, S.salary*0.1);
```

MERGE

EMPLOYEE_ID	BONUS
-----	-----
153	180
154	175
155	170
156	200
158	190
159	180
160	175
161	170
163	195
157	950
145	1400
170	960
179	620
152	900
169	1000
.	
.	
.	

NOAUDIT

用途

NOAUDIT を使用すると、AUDIT 文によって有効になった監査を停止できます。

NOAUDIT 文は先に発行した AUDIT 文と同じ構文である必要があります。また、NOAUDIT 文は、その特定の AUDIT 文のみを無効にします。たとえば、1 番目の AUDIT 文 (A) は特定のユーザーに対する監査を有効にするものとします。2 番目の文 (B) が、すべてのユーザーに対して監査を有効にします。すべてのユーザーに対して監査を無効にする NOAUDIT 文 (C) は、文 B を無効にします。ただし、文 A は無効にされず、文 A が指定したユーザーの監査は継続されます。

参照： 監査の詳細は、12-50 ページの「[AUDIT](#)」を参照してください。

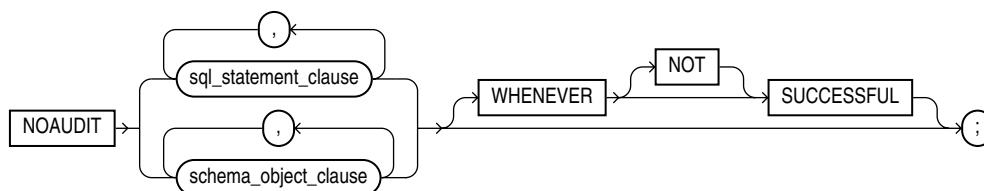
前提条件

SQL 文の監査を停止するには、AUDIT SYSTEM システム権限が必要です。

スキーマ・オブジェクトの監査を停止するには、監査を停止するオブジェクトが自分のスキーマ内にある必要があります。自分のスキーマ内でない場合は、AUDIT ANY システム権限が必要です。監査の対象として選択するオブジェクトがディレクトリの場合、自分が作成したディレクトリであっても、AUDIT ANY システム権限が必要です。

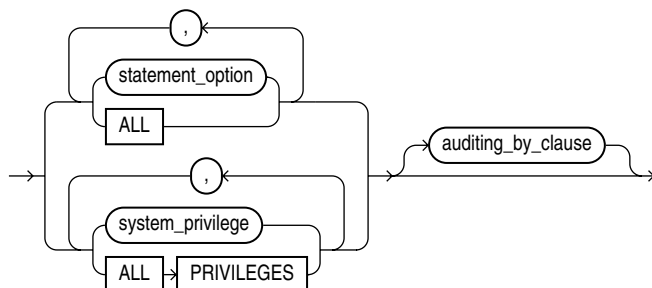
構文

noaudit::=

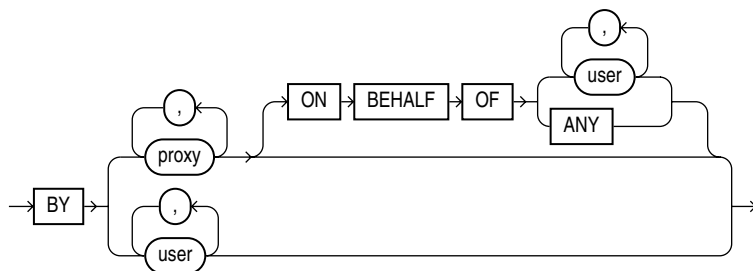


(17-80 ページの「[sql_statement_clause::=](#)」、17-80 ページの「[schema_object_clause::=](#)」を参照)

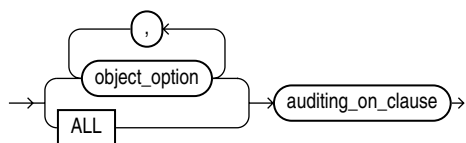
sql_statement_clause::=



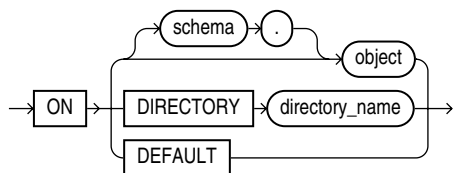
auditing_by_clause::=



schema_object_clause::=



auditing_on_clause::=



セマンティクス

sql_statement_clause

sql_statement_clause を使用し、特定の SQL 文の監査を停止します。

statement_option *statement_option* には、監査を停止する文のオプションを指定します。

参照： 文のオプションおよびそれによって監査される SQL 文のリストは、12-56 ページの表 12-1 および 12-59 ページの表 12-2 を参照してください。

ALL 現在監査されているすべての文オプションの監査を停止する場合に、ALL を指定します。

system_privilege *system_privilege* には、監査を停止するシステム権限を指定します。

参照： システム権限および各システム権限によって許可される文については、17-35 ページの表 17-1 を参照してください。

ALL PRIVILEGES 現在監査されているすべてのシステム権限の監査を停止する場合に、ALL PRIVILEGES を指定します。

auditing_by_clause *auditing_by_clause* は、特定のユーザーが発行する SQL 文の監査のみを停止します。この句を指定しない場合、すべてのユーザー文の監査が停止されます。

- 指定したユーザーのそれ以降のセッションで発行される SQL 文の監査のみを取り消す場合に、BY *user* を指定します。この句を指定しない場合、すべてのユーザー文の監査を停止します。ただし、WHENEVER SUCCESSFUL で説明する状況は除きます。
- 特定のユーザーまたは任意のユーザーのかわりに指定されたプロキシによって発行された SQL 文の監査のみを停止する場合に、BY *proxy* を指定します。

schema_object_clause

schema_object_clause を使用し、特定のデータベース・オブジェクトの監査を停止します。

object_option *object_option* には、ON 句で指定したオブジェクトへの監査を停止する操作の種類を指定します。

参照： これらのオプションのリストは、12-60 ページの表 12-3 を参照してください。

ALL ALL をショートカットに指定することは、オブジェクト・タイプに適用できるオプションをすべて指定することと同じです。

auditing_on_clause *auditing_on_clause* では、監査を停止する特定のスキーマ・オブジェクトを指定できます。

- オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、ファンクション、パッケージ、マテリアライズド・ビューまたはライブラリのオブジェクト名を指定します。object に *schema* を指定しなかった場合、そのオブジェクトは自分のスキーマ内にあるとみなされます。

参照： 特定のスキーマ・オブジェクトの監査については、12-50 ページの「AUDIT」を参照してください。

- DIRECTORY では、監査を停止するディレクトリ名を指定できます。
- DEFAULT を指定して、オブジェクトを作成した後に、特定のオブジェクト・オプションをデフォルト・オブジェクト・オプションとして削除します。

WHENEVER [NOT] SUCCESSFUL 正常に実行されたスキーマ・オブジェクトに対する SQL 文および操作の監査のみを停止する場合は、WHENEVER SUCCESSFUL を指定します。

WHENEVER NOT SUCCESSFUL は、Oracle エラーとなった文および操作の監査のみを停止します。

この句を指定しない場合、正常に実行されたかどうかにかかわらず、すべての文および操作の監査が停止されます。

例

ロールに関連する SQL 文の監査の停止例： 次の文は、ロールを作成または削除するすべての SQL 文の監査を停止します。

```
NOAUDIT ROLE;
```

特定ユーザーが所有するオブジェクトに対する更新または問合せの監視の停止例： 次の文は、ユーザー hr および oe によって発行された、表の問合せまたは更新を実行する文を監査している場合、hr の問合せの監査のみを停止します。

```
NOAUDIT SELECT TABLE BY hr;
```

この結果、ユーザー hr の問合せの監査のみが停止されます。oe の問合せと更新、および hr の更新の監査は継続されます。

特定のオブジェクト権限で許可された文の監査の停止例： 次の文は、DELETE ANY TABLE システム権限に許可されたすべての文の監査を停止します。

```
NOAUDIT DELETE ANY TABLE;
```

特定のオブジェクトに対する問合せの監査の停止例： スキーマ hr 内の employees 表に問い合わせるすべての SQL 文の監査を選択していた場合、次の文を発行すると、この監査が停止されます。

```
NOAUDIT SELECT
  ON hr.employees;
```

正常に実行される問合せの監査の停止例： 次の文は、正常に終了した問合せの監査を停止します。

```
NOAUDIT SELECT
  ON hr.employees
  WHENEVER SUCCESSFUL;
```

この文は、正常に終了した問合せの監査のみ停止します。Oracle は、Oracle エラーの結果発生した問合せの監査を継続します。

RENAME

用途

RENAME 文を使用すると、表、ビュー、順序またはプライベート・シノニムの名前を変更できます。

- 古いオブジェクトの整合性制約、索引および権限付与は、新しいオブジェクトに自動的に移行されます。
- 名前を変更した表を参照するビュー、シノニム、ストアド・プロシージャ、ストアド・ファンクションなど、名前を変更したオブジェクトに依存するオブジェクトはすべて無効になります。

注意： RENAME 文はロールバックできません。

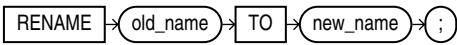
参照： 15-2 ページの「[CREATE SYNONYM](#)」および 17-4 ページの「[DROP SYNONYM](#)」を参照してください。

前提条件

オブジェクトが自分のスキーマ内にある必要があります。

構文

rename::=



セマンティクス

old_name

既存の表、ビュー、順序またはプライベート・シノニムの名前を指定します。

new_name

既存のオブジェクトに指定する新しい名前を指定します。新しい名前は、同じネームスペース内の他のスキーマ・オブジェクトに使用されている名前以外を指定する必要があります。また、スキーマ・オブジェクトのネーミング規則に従って指定する必要があります。

オブジェクトの名前変更の制限事項：

- パブリック・シノニムの名前は変更できません。そのかわり、該当するパブリック・シノニムを削除し、新しい名前でのパブリック・シノニムを作成してください。
- 依存表または依存する有効なユーザー定義オブジェクト型を持つ型のシノニムの名前は変更できません。

参照： 2-105 ページの「スキーマ・オブジェクトのネーミング規則」を参照してください。

例

データベース・オブジェクトの名前の変更例： 次の例は、サンプル表 `hr.departments` のコピーを使用します。表の名前を `departments_new` から `emp_departments` に変更する場合、次の文を発行します。

```
RENAME departments_new TO emp_departments;
```

この文では列名を直接変更できません。ただし、`ALTER TABLE ... rename_column_clause` を使用すると、列の名前を変更できます。

参照： 11-54 ページの「`rename_column_clause`」を参照してください。

列名を変更するもう 1 つの方法は、AS 副問合せを指定した `CREATE TABLE` 文とともに、`RENAME` 文を使用する方法です。この方法は、列の名前を変更する場合のみでなく、表の構造を変更する場合に有効です。次の文は、サンプル表 `hr.job_history` を再作成し、列の名前を `department_id` から `dept_id` に変更します。

```
CREATE TABLE temporary
    (employee_id, start_date, end_date, job_id, dept_id)
AS SELECT
    employee_id, start_date, end_date, job_id, department_id
FROM job_history;

DROP TABLE job_history;

RENAME temporary TO job_history;
```

注意： 前述の例の場合、`job_history` 表に定義されている整合性制約は失われます。これらの整合性制約は、`ALTER TABLE` 文を使用して、新しい `job_history` 表に再定義する必要があります。

REVOKE

用途

REVOKE 文は、次の処理を行う場合に使用します。

- ユーザーおよびロールからのシステム権限の取消し
- ユーザーおよびロールからのロールの取消し
- ユーザーまたはロールからの特定のオブジェクトに対するオブジェクト権限の取消し

参照：

- システム権限およびロールの付与については、17-27 ページの「[GRANT](#)」を参照してください。
- それぞれのオブジェクト型に対するオブジェクト権限の概要については、17-45 ページの表 [17-3](#) を参照してください

前提条件

システム権限を取り消すには、Admin Option 付きの権限が必要です。

ロールを取り消すには、Admin Option 付きのロールが必要です。なお、GRANT ANY ROLE システム権限がある場合は、ロールを自由に取消することができます。

オブジェクト権限を取り消すには、以前にユーザーとロールにオブジェクト権限が付与されているか、または GRANT ANY OBJECT PRIVILEGE システム権限が付与されている必要があります。後者の場合は、オブジェクト所有者により付与されたかまたは所有者の役割を持つユーザー（GRANT ANY OBJECT PRIVILEGE を持つユーザー）によって付与されたオブジェクト権限を取り消すことができます。ただし、With Grant Option によって付与されたオブジェクト権限を取り消すことはできません。

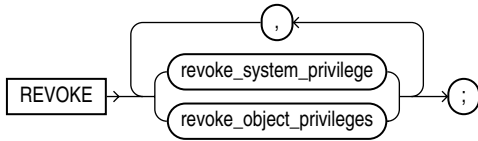
参照： 17-95 ページの「[GRANT ANY OBJECT PRIVILEGE を使用する操作を取り消す例](#)」を参照してください。

REVOKE 文によって取り消すことができる権限およびロールは、GRANT 文によって直接付与されているものにかぎられます。この句では、次の権限を取り消すことはできません。

- 取消し側に付与されていない権限またはロール
- オペレーティング・システムを介して付与されているロールまたはオブジェクト権限
- ロールを介して取消し側に付与されている権限またはロール

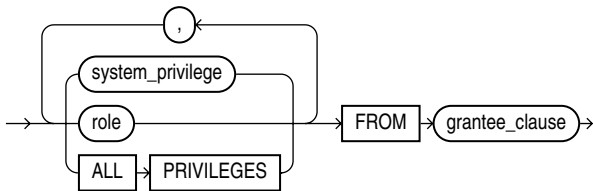
構文

revoke::=



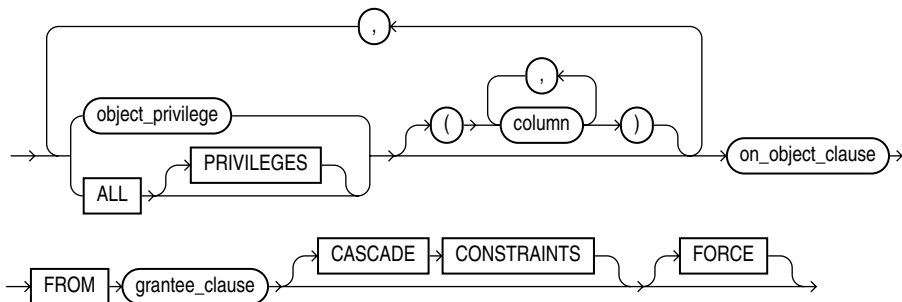
(17-87 ページの「[revoke_system_privileges::=](#)」、17-87 ページの「[revoke_object_privileges::=](#)」を参照)

[revoke_system_privileges::=](#)

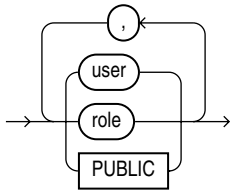
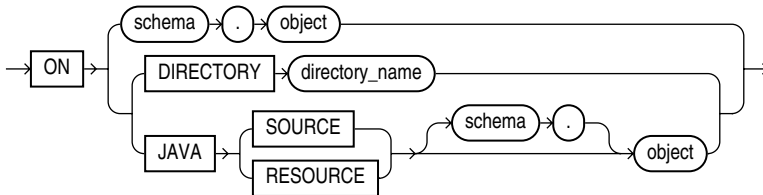


(17-88 ページの「[grantee_clause::=](#)」を参照)

[revoke_object_privileges::=](#)



(17-88 ページの「[on_object_clause::=](#)」、17-88 ページの「[grantee_clause::=](#)」を参照)

grantee_clause::=**on_object_clause::=**

セマンティクス

revoke_system_privileges

system_privilege

取り消すシステム権限を指定します。

参照： システム権限のリストは、17-35 ページの表 17-1 を参照してください。

- **ユーザーの権限**を取り消す場合は、ユーザーの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、このユーザーはその権限を使用できなくなります。

参照： 17-93 ページの「ユーザーからシステム権限を取り消す例:」を参照してください。

- **ロールの権限**を取り消す場合は、ロールの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、そのロールが使用可能となっている場合でも、この権限は使用できません。また、そのロールが権限付与されている他のユーザーは、ロールを使用可能にしても、その権限を使用できなくなります。

参照： 17-93 ページの「ロールからシステム権限を取り消す例:」を参照してください。

- **PUBLIC の権限**を取り消す場合は、PUBLIC を介して権限を付与されている各ユーザーの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、ユーザーは権限を使用できなくなります。ただし、直接またはロールを介して権限が付与されているユーザーからは、権限を取り消すことはできません。

システム権限の取消しの制限事項： 取り消す権限のリストに権限を指定できるのは1回のみです。

すべてのシステム権限を一度に指定できるショートカットがあります。

- ALL PRIVILEGES: ALL PRIVILEGES を指定すると、17-35 ページの表 17-1 に示すすべてのシステム権限を取り消すことができます。

role

取り消すロールを指定します。

- **ユーザー**からロールを取り消す場合は、ユーザーによるロールの使用を禁止します。そのロールが使用可能となっている場合に、ロールの権限ドメインの権限が使用可能な場合はその権限を使用できます。ただし、ユーザーが後からロールを使用可能にできません。

参照： 17-93 ページの「[ユーザーからロールを取り消す例：](#)」を参照してください。

- 他の**ロール**からロールを取り消す場合、Oracle は取消し側ロールの権限ドメインから、取り消されたロールの権限ドメインを削除します。取り消されたロールの権限を付与され、そのロールの権限が使用可能になっているユーザーは、そのロールの権限が使用可能な間は、取り消されたロールの権限ドメインの権限を引き続き使用できます。ただし、取り消された権限を付与されていても、ロールの取消し操作の後でそれを使用可能にしたユーザーは、取り消されたロールの権限ドメインの権限を使用できません。

参照： 17-93 ページの「[ロールからロールを取り消す例：](#)」を参照してください。

- **PUBLIC** のロールを取り消す場合、PUBLIC を介してロールが付与されているすべてのユーザーに対して、そのロールを使用禁止にします。そのロールを使用可能としているユーザーは、権限ドメインの権限が使用可能であるかぎり、権限ドメインでその権限を引き続き使用できます。ただし、ユーザーが後からロールを使用可能にすることはできません。直接またはロールを介して権限が付与されているユーザーからは、ロールを取り消すことはできません。

システム・ロールの取消しの制限事項： 取り消すロールのリストにロールを指定できるのは1回のみです。

参照： 事前定義されたロールのリストは、17-44 ページの表 17-2 を参照してください。

grantee_clause

FROM *grantee_clause* は、システム権限、ロールまたはオブジェクト権限が取り消されるユーザーまたはロールを識別します。

PUBLIC すべてのユーザーから権限を取り消す場合は、PUBLIC を指定します。

revoke_object_privileges

object_privilege

取り消すオブジェクト権限を指定します。ALTER、DELETE、EXECUTE、INDEX、INSERT、READ、REFERENCES、SELECT または UPDATE のいずれかの値に置換することができます。

注意： 操作は権限によって許可されます。権限を取り消すことによって、取り消されたユーザーは、その操作ができなくなります。ただし、複数のユーザーが、同じ権限を同一ユーザー、ロールまたは PUBLIC に対して付与できます。権限受領者の権限ドメインの権限を取り消す場合、すべての権限付与者が権限を取り消す必要があります。権限を取り消さない権限付与者が 1 人でもいれば、権限受領者は引き続きその権限を使用できます。

ユーザーの権限を取り消す場合は、ユーザーの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、このユーザーはその権限を使用できなくなります。

- ユーザーがその権限を他のユーザーまたはロールに付与している場合、他のユーザーまたはロールの権限も取り消されます。
- 権限を使用する SQL 文を記述したプロシージャ、ファンクションまたはパッケージが定義されているスキーマを持つユーザーのオブジェクト権限を取り消すと、それらのプロシージャ、ファンクションまたはパッケージは実行できなくなります。
- そのユーザーのスキーマにオブジェクトのビューが含まれる場合、ビューは無効になります。
- 参照整合性制約の定義権限を使用したユーザーの REFERENCES 権限を取り消す場合、CASCADE CONSTRAINTS 句も指定する必要があります。

参照： 17-93 ページの「ユーザーからオブジェクト権限を取り消す例:」を参照してください。

ロールの権限を取り消す場合は、ロールの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、そのロールが使用可能となっている場合でも、この権限は使用できません。ロールが権限付与されている他のユーザーは、ロールを使用可能にした場合でも、権限を使用できなくなります。

PUBLIC の権限を取り消す場合は、PUBLIC を介して権限を付与されている各ユーザーの権限ドメインの権限を取り消します。この取消しはすぐに有効になるため、それらのすべてのユーザーは、その権限を使用できなくなります。ただし、直接またはロールを介して権限が付与されているユーザーからは、権限を取り消すことはできません。

参照： 17-94 ページの「[PUBLIC からオブジェクト権限を取り消す例:](#)」を参照してください。

オブジェクト権限の取消しの制限事項 取り消す権限のリストに権限を指定できるのは 1 回のみです。FROM 句にユーザー、ロールまたは PUBLIC を指定できるのは 1 回のみです。

ALL [PRIVILEGES]

ユーザーまたはロールに付与されているすべてのオブジェクト権限を取り消す場合に、ALL を指定します（キーワード PRIVILEGES はセマンティクスを明確にするためのものであり、指定は任意です。）

注意： オブジェクトに権限が付与されていない場合、処理は行われず、エラーも戻りません。

参照： 17-94 ページの「[ユーザーからすべてのオブジェクト権限を取り消す例:](#)」を参照してください。

CASCADE CONSTRAINTS

この句は、REFERENCES 権限または ALL [PRIVILEGES] を取り消すときにのみ適用されます。取消し側で REFERENCES 権限（ALL [PRIVILEGES] を付与して明示的または暗黙的に付与された権限）を使用して定義した参照整合性制約を削除します。

参照： 17-94 ページの「[CASCADE CONSTRAINTS でオブジェクト権限を取り消す例:](#)」を参照してください。

FORCE

表または型に依存するユーザー定義型オブジェクトで、EXECUTE オブジェクト権限を取り消す場合に、FORCE を指定します。表に依存するユーザー定義型オブジェクトでは、FORCE を使用して EXECUTE オブジェクト権限を取り消します。

FORCE を指定した場合、すべての権限が取り消されますが、すべての依存するオブジェクトには INVALID のマークが付けられ、依存表のデータにはアクセスできなくなります。また、すべての依存するファンクション索引には、UNUSABLE のマークが付けられます（必要な型の権限を再付与した場合、表に対して再度妥当性チェックが行われます）。

参照： 型依存性およびユーザー定義オブジェクト権限の詳細は、『Oracle9i データベース概要』を参照してください。

on_object_clause

on_object_clause は、権限を取り消すオブジェクトを識別します。

object オブジェクト権限を取り消すオブジェクトを指定します。取り消すことができるオブジェクトは次のとおりです。

- 表、ビュー、順序、プロシージャ、ストアド・ファンクション、パッケージまたはマテリアライズド・ビュー
- 表、ビュー、順序、プロシージャ、ストアド・ファンクション、パッケージ、マテリアライズド・ビューまたはユーザー定義型に対するシノニム
- ライブラリ、索引タイプまたはユーザー定義演算子

オブジェクトに *schema* を指定しなかった場合、そのオブジェクトは自分のスキーマ内にあるとみなされます。

参照： 17-94 ページの「[ユーザーから順序のオブジェクト権限を取り消す例 :](#)」を参照してください。

(Grant Option の有無にかかわらず) SELECT オブジェクト権限をマテリアライズド・ビューまたは表を含むマテリアライズド・ビューから取り消すと、マテリアライズド・ビューは無効になります。

(Grant Option の有無にかかわらず) マテリアライズド・ビューのマスター表のいずれかにおける SELECT オブジェクト権限を取り消すと、マテリアライズド・ビューとそれに含まれる表またはマテリアライズド・ビューの両方が無効になります。

DIRECTORY *directory_name* 権限を取り消すディレクトリ・オブジェクトを指定します。*directory_name* にはスキーマを指定できません。このオブジェクトはディレクトリである必要があります。

参照： 13-46 ページの「[CREATE DIRECTORY](#)」および 17-95 ページの「[ユーザーからディレクトリへのオブジェクト権限を取り消す例 :](#)」を参照してください。

JAVA SOURCE | RESOURCE JAVA 句によって、権限が取り消された Java ソースまたはリソース・スキーマ・オブジェクトを指定します。

例

ユーザーからシステム権限を取り消す例： 次の文は、ユーザー hr および oe の DROP ANY TABLE システム権限を取り消します。

```
REVOKE DROP ANY TABLE
FROM hr, oe;
```

この結果、hr および oe は他のユーザーのスキーマに定義されている表を削除できなくなります。

ユーザーからロールを取り消す例： 次の文は、ユーザー sh のロール dw_manager を取り消します。

```
REVOKE dw_manager
FROM sh;
```

この結果、sh はロール dw_manager を使用できなくなります。

ロールからシステム権限を取り消す例： 次の文は、ロール dw_manager の CREATE TABLESPACE システム権限を取り消します。

```
REVOKE CREATE TABLESPACE
FROM dw_manager;
```

ロール dw_manager を使用可能にしても、ユーザーは表領域を作成できません。

ロールからロールを取り消す例： 次の文は、ロール dw_manager からロール dw_user を取り消します。

```
REVOKE dw_user
FROM dw_manager;
```

この結果、dw_user の権限はロール dw_manager に付与されなくなります。

ユーザーからオブジェクト権限を取り消す例： 次の文は、orders 表に対する DELETE、INSERT、SELECT および UPDATE 権限をユーザー hr に付与します。

```
GRANT ALL
ON orders TO hr;
```

次の文は、ユーザー hr から表 orders に対する DELETE 権限を取り消します。

```
REVOKE DELETE
ON orders FROM hr;
```

ユーザーからすべてのオブジェクト権限を取り消す例： 次の文は、ユーザー hr の表 orders に対する残りのすべての権限を取り消します。

```
REVOKE ALL
  ON orders FROM hr;
```

PUBLIC からオブジェクト権限を取り消す例： 次の文は、ロール public に権限を付与することによって、すべてのユーザーにビュー emp_details_view に対する SELECT 権限および UPDATE 権限を付与します。

```
GRANT SELECT, UPDATE
  ON emp_details_view TO public;
```

次の文は、すべてのユーザーから emp_details_view に対する UPDATE 権限を取り消します。

```
REVOKE UPDATE
  ON emp_details_view FROM public;
```

ユーザーは、emp_details_view ビューへの問合せはできますが、更新はできなくなります。ただし、emp_details_view に対する UPDATE 権限も任意のユーザーに直接またはロールを介して付与している場合は、これらのユーザーはその権限を保持します。

ユーザーから順序のオブジェクト権限を取り消す例： 次の文は、ユーザー oe にスキーマ hr 内の departments_seq 順序に対する SELECT 権限を付与します。

```
GRANT SELECT
  ON hr.departments_seq TO oe;
```

oe から departments_seq に対する SELECT 権限を取り消す場合、次の文を発行します。

```
REVOKE SELECT
  ON hr.departments_seq FROM oe;
```

ただし、ユーザー hr が departments に対する SELECT 権限を sh に付与している場合、sh は、hr からの権限付与によって departments を使用できます。

CASCADE CONSTRAINTS でオブジェクト権限を取り消す例： 次の文は、oe に、スキーマ hr 内の employees 表に対する REFERENCES 権限および UPDATE 権限を付与します。

```
GRANT REFERENCES, UPDATE
  ON hr.employees TO oe;
```

oe は、REFERENCES 権限を使用して、スキーマ hr 内の employees 表を参照する dependent 表の制約を定義できます。

```
CREATE TABLE dependent
(dependno    NUMBER,
dependname  VARCHAR2(10),
employee    NUMBER
CONSTRAINT in_emp REFERENCES hr.employees(employee_id) );
```

CASCADE CONSTRAINTS 句を指定した次の文を発行することによって、oe の hr.employees に対する REFERENCES 権限を取り消すことができます。

```
REVOKE REFERENCES
ON hr.employees
FROM oe
CASCADE CONSTRAINTS;
```

oe の hr.employees に対する REFERENCES 権限を取り消した場合、oe は制約を定義する権限が必要になるため、in_emp 制約が自動的に削除されます。

ただし、oe が他のユーザーから hr.employees に対する REFERENCES 権限を付与されている場合は、Oracle はその制約を削除しません。他のユーザーから権限付与されたため、oe は制約に対して必要な権限を保持しています。

ユーザーからディレクトリへのオブジェクト権限を取り消す例： 次の文は、hr のディレクトリ bfile_dir に対する READ 権限を取り消します。

```
REVOKE READ ON DIRECTORY bfile_dir FROM hr;
```

GRANT ANY OBJECT PRIVILEGE を使用する操作を取り消す例： データベース管理者によってユーザー sh に GRANT ANY OBJECT PRIVILEGE が付与されているとします。ユーザー hr が、employees 表に対する UPDATE 権限をユーザー oe に付与します。

```
CONNECT hr/hr
GRANT UPDATE ON employees TO oe WITH GRANT OPTION;
```

これによって、オブジェクト権限を別のユーザーに付与する権限がユーザー oe に付与されます。

```
CONNECT oe/oe
GRANT UPDATE ON hr.employees TO pm;
```

oe には hr によって権限が与えられたため、GRANT ANY OBJECT PRIVILEGE が付与されているユーザー sh は、ユーザー hr にかわってユーザー oe の UPDATE 権限を取り消すことができます。

```
CONNECT sh/sh
REVOKE UPDATE ON hr.employees FROM oe;
```

pm に権限を付与したのがオブジェクトの所有者 (hr)、ユーザー sh または GRANT ANY OBJECT PRIVILEGE を持つ他のユーザーではなく、ユーザー oe であったため、ユーザー sh は、ユーザー pm の UPDATE 権限を明示的に取り消すことはできません。ただし、前述の文は、連鎖的な取消しを行い、取り消された権限に依存するすべての権限を取り消します。そのため、pm のオブジェクト権限も暗黙的に取り消されます。

ROLLBACK

用途

ROLLBACK 文を使用すると、現行のトランザクションで実行された処理を取り消すことができます。また、インダウト分散トランザクションで実行された処理を手動で取り消すこともできます。

注意： アプリケーション・プログラムでは、COMMIT または ROLLBACK 文を使用してトランザクションを明示的に終了することをお勧めします。トランザクションを明示的にコミットせずにプログラムが異常終了した場合、コミットされていない最後のトランザクションがロールバックされます。

参照：

- トランザクションの詳細は、『Oracle9i データベース概要』を参照してください。
- 現行トランザクションの特性の設定については、18-47 ページの「[SET TRANSACTION](#)」を参照してください。
- 12-70 ページの「[COMMIT](#)」を参照してください。
- 18-2 ページの「[SAVEPOINT](#)」を参照してください。

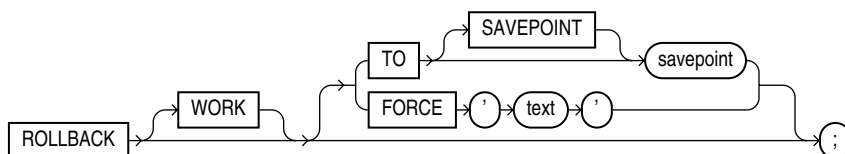
前提条件

現行のトランザクションをロールバックする場合、権限は不要です。

コミットしたインダウト分散トランザクションを手動でロールバックする場合は、FORCE TRANSACTION システム権限が必要です。他のユーザーがコミットしたインダウト分散トランザクションを手動でロールバックする場合は、FORCE ANY TRANSACTION システム権限が必要です。

構文

rollback::=



セマンティクス

WORK

キーワード WORK の指定は任意です。ANSI 互換性のために提供されています。

TO SAVEPOINT 句

現行のトランザクションをロールバックするセーブポイントを指定します。この句を指定しない場合、ROLLBACK 文によってトランザクション全体がロールバックされます。

TO SAVEPOINT 句を指定しない ROLLBACK コマンドによって、次の処理が行われます。

- トランザクションの終了
- 現行のトランザクションに対するすべての変更の取消し
- トランザクション中のセーブポイントの消去
- トランザクションのロックの解除

参照： 18-2 ページの「[SAVEPOINT](#)」を参照してください。

TO SAVEPOINT 句を指定する ROLLBACK コマンドによって、次の処理が行われます。

- 指定したセーブポイント後のトランザクションにロールバックします。
- 指定したセーブポイントより後に作成されたセーブポイントはすべて消去されます。指定したセーブポイントはそのまま残るため、そのセーブポイントまで繰り返しロールバックできます。指定したセーブポイントより前に作成されたセーブポイントも残ります。
- 指定したセーブポイント以降に設定された表と行のロックを解除します。セーブポイント後にロックされた行へのアクセスを要求した他のトランザクションは、コミットまたはロールバックされるまで待つ必要があります。行を要求していない他のトランザクションは、すぐに行の要求およびアクセスができます。

インダウト・トランザクションの制限事項： インダウト・トランザクションは、セーブポイントまで手動でロールバックできません。

FORCE 句

インダウト分散トランザクションを手動でロールバックする場合に、FORCE を指定します。このトランザクションは、ローカル・トランザクション ID またはグローバル・トランザクション ID を含む 'text' で識別されます。このトランザクションの ID を確認する場合、データ・ディクショナリ・ビュー DBA_2PC_PENDING を問い合わせます。

FORCE 句を指定した ROLLBACK 文では、指定したトランザクションのみをロールバックするため注意してください。この文は、現行のトランザクションには影響しません。

ロールバックの強制的制限事項： PL/SQL では、FORCE 句を指定する ROLLBACK 文はサポートされていません。

参照： 分散トランザクションおよびインダウト・トランザクションのロールバックについては、『Oracle9i Heterogeneous Connectivity Administrator's Guide』を参照してください。

例

ロールバック・トランザクションの例： 次の文は、現行のトランザクション全体をロールバックします。

```
ROLLBACK;
```

次の文は、現行のトランザクションをセーブポイント `banda_sal` にロールバックします。

```
ROLLBACK TO SAVEPOINT banda_sal;
```

参照： この例の詳細は、18-3 ページの「[セーブポイントの作成例:](#)」を参照してください。

次の文は、インダウト分散トランザクションを手動でロールバックします。

```
ROLLBACK WORK  
FORCE '25.32.87';
```

SQL 文 : SAVEPOINT ~ UPDATE

この章では、次の SQL 文について説明します。

- `SAVEPOINT`
- `SELECT`
- `SET CONSTRAINT [S]`
- `SET ROLE`
- `SET TRANSACTION`
- `TRUNCATE`
- `UPDATE`

SAVEPOINT

用途

SAVEPOINT 文を使用すると、トランザクション内でロールバックされる位置を指定できます。

参照：

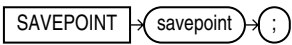
- セーブポイントの詳細は、『Oracle9i データベース概要』を参照してください。
- トランザクションのロールバックの詳細は、17-97 ページの「[ROLLBACK](#)」を参照してください。
- 現行トランザクションの特性の設定については、18-47 ページの「[SET TRANSACTION](#)」を参照してください。

前提条件

ありません。

構文

savepoint::=



セマンティクス

savepoint

作成するセーブポイントの名前を指定します。

同一トランザクション内のセーブポイント名は、区別する必要があります。同じ識別子のセーブポイントを指定した場合、最初のセーブポイントは消去されます。セーブポイントを作成した後は、処理の継続、作業のコミット、トランザクション全体のロールバックまたはセーブポイントまでのロールバックができます。

例

セーブポイントの作成例： 次の文は、サンプル表 `hr.employees` の Banda と Greene の給与を更新するために、部門の給与合計が 314,000 ドルを超えていないことを確認してから、Greene の給与を再入力します。

```
UPDATE employees
  SET salary = 7000
  WHERE last_name = 'Banda';
SAVEPOINT banda_sal;

UPDATE employees
  SET salary = 12000
  WHERE last_name = 'Greene';
SAVEPOINT greene_sal;

SELECT SUM(salary) FROM employees;

ROLLBACK TO SAVEPOINT banda_sal;

UPDATE employees
  SET salary = 11000
  WHERE last_name = 'Greene';

COMMIT;
```

SELECT

用途

SELECT 文または副問合せを使用すると、1 つ以上の表、オブジェクト表、ビュー、オブジェクト・ビューまたはマテリアライズド・ビューからデータを取り出すことができます。

注意： SELECT 文の結果（またはその一部）が既存のマテリアライズド・ビューと同じ場合、そのマテリアライズド・ビューを SELECT 文で指定した 1 つ以上の表のかわりに使用できます。このような置換を**クエリー・リライト**といいます。これは、コストの最適化が使用可能で、`QUERY_REWRITE_ENABLED` パラメータが `TRUE` に設定されている場合にのみ行われます。クエリー・リライトが行われるかどうかを確認する場合は、`EXPLAIN PLAN` 文を使用してください。

参照：

- 問合せおよび副問合せの詳細は、[第 8 章「SQL 問合せおよび副問合せ」](#)を参照してください。
- マテリアライズド・ビューおよびクエリー・リライトの詳細は、『[Oracle9i データ・ウェアハウス・ガイド](#)』を参照してください。
- 17-23 ページの「[EXPLAIN PLAN](#)」を参照してください。

前提条件

表またはマテリアライズド・ビューからデータを選択する場合、表またはマテリアライズド・ビューが自分のスキーマ内にあるか、またはその表またはマテリアライズド・ビューに対する SELECT 権限が必要です。

ビューの実表から行を選択する場合、次の条件を 2 つとも満たしている必要があります。

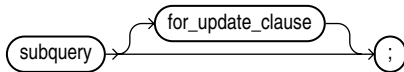
- ビューに対する SELECT 権限を持っている。
- ビューを含むスキーマの所有者が、実表に対する SELECT 権限を持っている。

SELECT ANY TABLE システム権限を持っている場合、任意の表、マテリアライズド・ビューまたはビューの実表からデータを選択できます。

フラッシュバック問合せを発行（`flashback_clause` を使用）するには、SELECT 構文のリスト内のオブジェクトに対する FLASHBACK オブジェクト権限または FLASHBACK ANY TABLE システム権限が必要です。

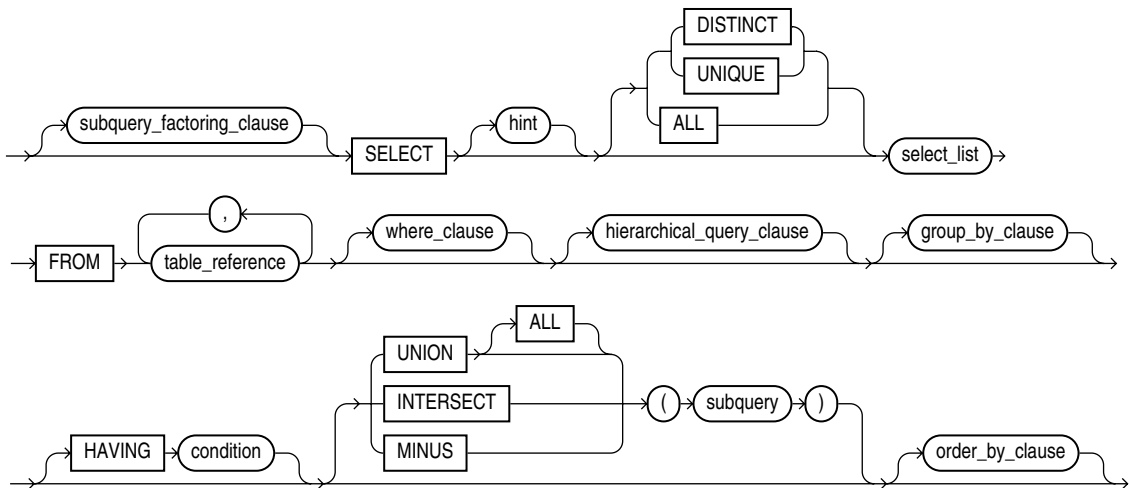
構文

select::=



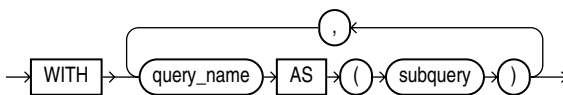
(18-10 ページの「[for_update_clause::=](#)」を参照)

subquery::=

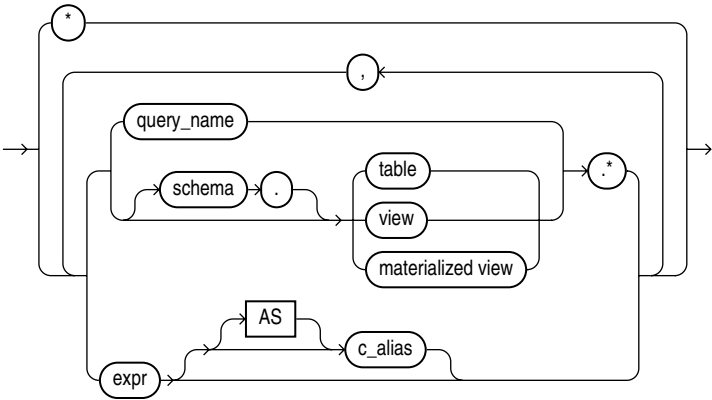


(18-5 ページの「[subquery_factoring_clause::=](#)」、18-6 ページの「[select_list::=](#)」、18-6 ページの「[table_reference::=](#)」、18-8 ページの「[hierarchical_query_clause::=](#)」、18-9 ページの「[group_by_clause::=](#)」、18-10 ページの「[order_by_clause::=](#)」を参照)

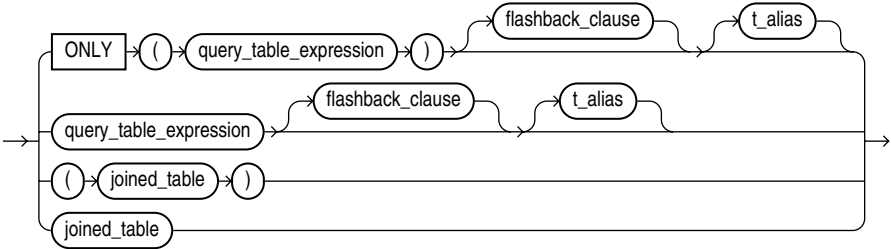
subquery_factoring_clause::=



select_list::=

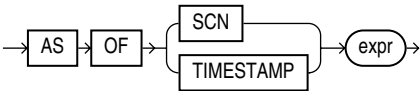


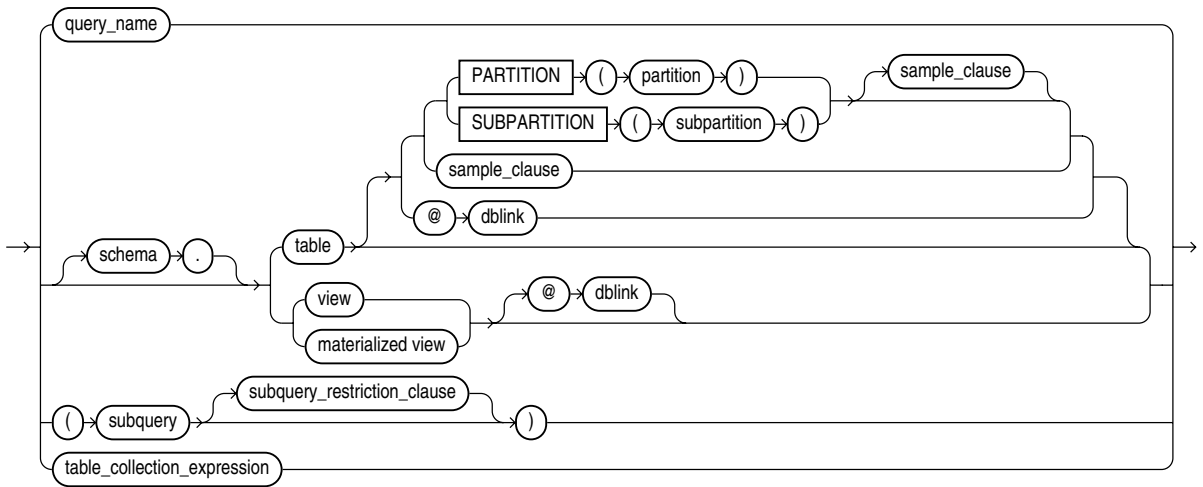
table_reference::=



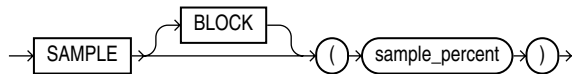
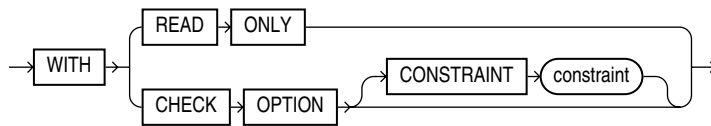
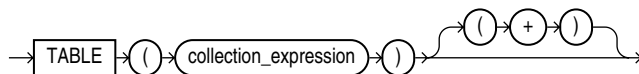
(18-7 ページの「[query_table_expression::=](#)」、18-6 ページの「[flashback_clause::=](#)」を参照)

flashback_clause::=

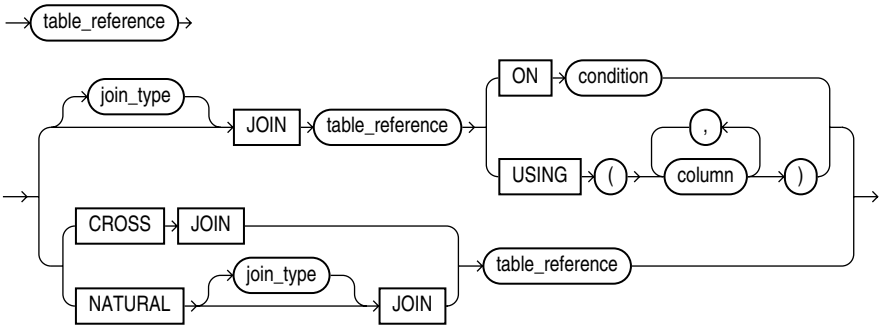


query_table_expression::=

(18-7 ページの「**subquery_restriction_clause::=**」、18-7 ページの「**table_collection_expression::=**」を参照)

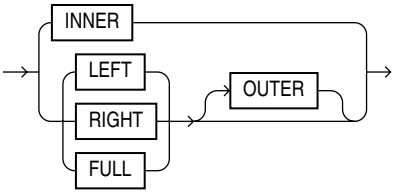
sample_clause::=**subquery_restriction_clause::=****table_collection_expression::=**

joined_table::=

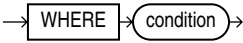


(18-6 ページの「table_reference::=」を参照)

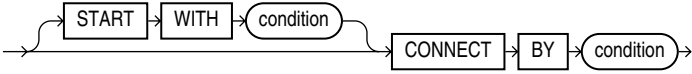
join_type::=



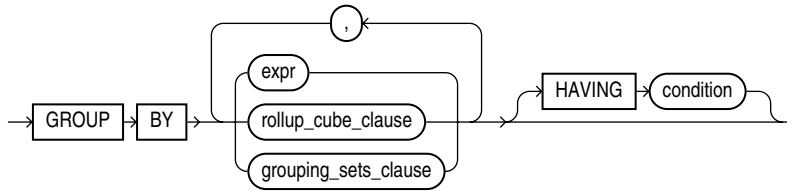
where_clause::=



hierarchical_query_clause::=

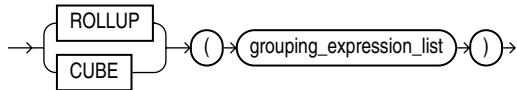


group_by_clause::=



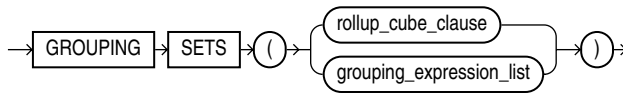
(18-9 ページの「[rollup_cube_clause::=](#)」、18-9 ページの「[grouping_sets_clause::=](#)」を参照)

rollup_cube_clause::=



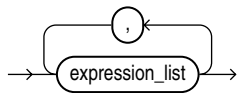
(18-9 ページの「[grouping_expression_list::=](#)」を参照)

grouping_sets_clause::=

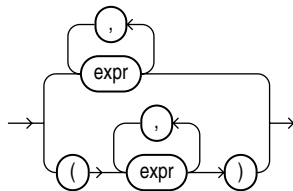


(18-9 ページの「[rollup_cube_clause::=](#)」、18-9 ページの「[grouping_expression_list::=](#)」を参照)

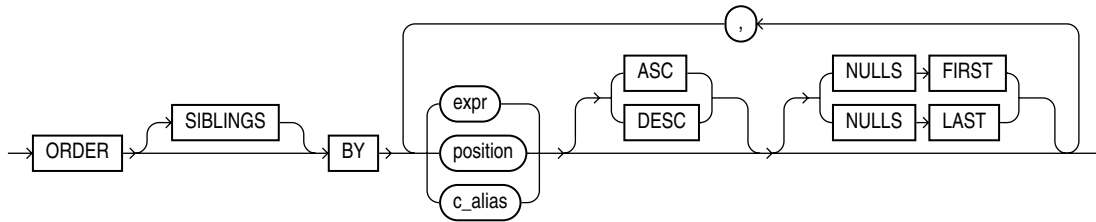
grouping_expression_list::=



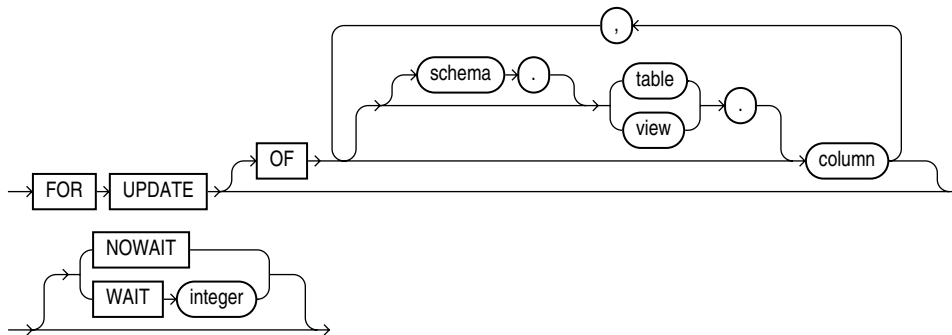
expression_list::=



order_by_clause::=



for_update_clause::=



セマンティクス

subquery_factoring_clause

subquery_factoring_clause (WITH *query_name*) を指定すると、副問合せのブロックに名前を割り当てることができます。問合せの名前を指定することによって、問合せに複数存在する副問合せブロックを参照することができます。問合せの名前をインライン・ビューまたは一時表として扱うことによって、問合せが最適化されます。

最上位の SELECT 文およびほとんどの副問合せでこの句を指定できます。問合せの名前は、後続のすべての副問合せ（自身の問合せ名を定義する副問合せを除く）および主問合せから参照できます。

副問合せのファクタリングの制限事項：

- この句はネストできません。つまり、他の *subquery_factoring_clause* の副問合せとして *subquery_factoring_clause* を指定できません。
- 集合演算子を指定した問合せでは、集合演算子の副問合せでは *subquery_factoring_clause* を指定できませんが、FROM 副問合せでは *subquery_factoring_clause* を指定することができます。

参照：

- インライン・ビューの詳細は、『Oracle9i データベース概要』を参照してください。
- 副問合せのファクタリング機能の使用については、『Oracle9i データ・ウェアハウス・ガイド』および『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 18-26 ページの「[副問合せのファクタリング例：](#)」を参照してください。

hint

文の実行計画を選択する場合に、オプティマイザに指示を与えるためのコメントを指定します。

参照： ヒントの構文および説明については、2-86 ページの「[ヒント](#)」および『Oracle9i データベース・パフォーマンス・チューニング・ガイド』およびリファレンス』を参照してください。

DISTINCT | UNIQUE

選択された行に重複行の 1 行のみを戻す場合に、DISTINCT または UNIQUE（これらの 2 つのキーワードは同義）を指定します。重複行とは、SELECT 構文のリスト中のそれぞれの式で一致する値を持つ行のことです。

DISTINCT および UNIQUE 問合せの制限事項：

- DISTINCT または UNIQUE を指定する場合、SELECT 構文のリスト中の式すべての総バイト数は、データ・ブロックのサイズからオーバーヘッド分を引いたサイズに制限されます。このサイズは、初期化パラメータ DB_BLOCK_SIZE によって指定されます。
- `select_list` に LOB 列が含まれている場合、DISTINCT は指定できません。

ALL

重複行を含め、選択されたすべての行を戻す場合に、ALL を指定します。デフォルトは ALL です。

*** (アスタリスク)**

FROM 句に指定されているすべての表、ビューまたはマテリアライズド・ビューのすべての列を選択する場合に、アスタリスクを指定します。

注意： 表から選択する（FROM 句に、ビューやマテリアライズド・ビューではなく表を指定する）場合、ALTER TABLE SET UNUSED 文によって UNUSED のマークが付けられた列は選択されません。

参照： 11-2 ページの [ALTER TABLE](#)、18-26 ページの「[単純問合せの例 :](#)」および 18-41 ページの「[DUAL 表からの選択例 :](#)」を参照してください。

select_list

select_list を指定すると、データベースから取り出す列を指定できます。

query_name

query_name には、*subquery_factoring_clause* ですでに指定されている名前を指定します。*select_list* で *query_name* を指定するには、*subquery_factoring_clause* を指定する必要があります。*select_list* で *query_name* を指定するには、*query_table_expression* (FROM 句) で *query_name* を指定する必要があります。

table.* | view.* | materialized view.*

指定した表、ビューまたはマテリアライズド・ビューのすべての列を選択する場合に、ピリオドおよびアスタリスクの後にオブジェクト名を指定します。結合とは、2 つ以上の表、ビューまたはマテリアライズド・ビューの行を選択する問合せです。

他のユーザーのスキーマの表、ビューまたはマテリアライズド・ビューから選択する場合に、スキーマ修飾子を使用します。*schema* を指定しない場合、この表、ビューおよびマテリアライズド・ビューは自分のスキーマ内にあるとみなされます。

参照： 8-9 ページの「[結合](#)」を参照してください。

expr

選択する情報を表す式を指定します。リスト中の列が含まれている表、ビューまたはマテリアライズド・ビューが FROM 句で *schema* 名で指定されている場合のみ、その列名を *schema* 名で指定できます。オブジェクト型のメンバー・メソッドを指定するときは、メソッドが引数をとらない場合でも、カッコを使用するメソッド名に従う必要があります。

参照： 18-41 ページの「[順序値の選択例 :](#)」を参照してください。

c_alias 列式に他の名前（別名）を指定します。この別名は、列のヘッダーで使います。AS キーワードはオプションです。別名によって、問合せ中に SELECT 構文のリストの項目名を効果的に変更できます。問合せにおいて、別名は *order_by_clause* で使用できますが、他の句では使用できません。

参照：

- 複数のマテリアライズド・ビューの間合せで、UNION ALL 演算子とともに `expr AS c_alias` 構文を使用する場合の詳細は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- `expr` の構文については、4-2 ページの「SQL 式」を参照してください。

SELECT 構文のリストの制限事項：

- この文に `group_by_clause` も指定されている場合、この SELECT 構文のリストには次の式のみ指定できます。
 - 定数
 - 集計ファンクション、USER ファンクション、UID ファンクションおよび SYSDATE ファンクション
 - `group_by_clause` に指定されているものと同じ式
 - グループ内のすべての行が同じ値に評価される前述の式を伴っている式
- 結合内のキー保存表が 1 つのみの場合、結合ビューから ROWID を選択することができます。表の ROWID がビューの ROWID になります。

参照： キー保存表の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

- 複数の表に同じ名前の列がある場合、表の名前でその列名を修飾する必要があります。

FROM 句

FROM 句を指定すると、どのオブジェクトからデータを選択するかを指定できます。

query_table_expression

`query_table_expression` 句を使用すると、表、ビュー、マテリアライズド・ビューまたはパーティションの識別、あるいはオブジェクトを識別する副問合せの指定を行うことができます。

参照： 18-33 ページの「副問合せの使用例 :」を参照してください。

ONLY ONLY 句は、ビューのみに適用されます。FROM 句のビューがビューの階層に属し、サブビューに行を含まない場合は、ONLY 句を使用します。

flashback_clause

flashback_clause を使用すると、表、ビューまたはマテリアライズド・ビューの過去のデータを問い合わせることができます。SCN を指定する場合、*expr* は数値に評価される必要があります。TIMESTAMP を指定する場合、*expr* はタイムスタンプ値に評価される必要があります。指定されたシステム変更番号または時刻に存在した行が戻されます。

注意： この句は、SQL によるフラッシュバックを実装します。この句を使用すると、SELECT 構文のリスト内の各オブジェクトについて異なるシステム変数番号またはタイムスタンプを指定できます。

DBMS_FLASHBACK パッケージを使用して、セッション・レベルのフラッシュバックも実装できます。セッション・レベルのフラッシュバックの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』および『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

フラッシュバック問合せの制限事項：

- *flashback_clause* は、リモート・データベース・オブジェクトには適用できません。ただし、*flashback_clause* が適用されるローカル・オブジェクトを使用して、リモート・オブジェクトを結合に含めることができます。
- *query_table_expression* で *query_name* を指定した場合、この句は指定できません。

参照：

- フラッシュバック問合せの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- 18-28 ページの「[フラッシュバック問合せの使用例：](#)」を参照してください。

PARTITION | SUBPARTITION PARTITION または SUBPARTITION には、データを取り出す表のパーティションの名前またはサブパーティションを指定します。

レンジ・パーティション表およびリスト・パーティション表では、この句のかわりに、データの取出しを *table* の 1 つ以上のパーティションに制限する条件を WHERE 句に指定できます。Oracle はこの条件を認識し、これらのパーティションのみからデータをフェッチします。（ハッシュ・パーティション・データには、この WHERE 条件を指定することはできません。）

参照： 18-27 ページの「[パーティションからの行の選択例：](#)」を参照してください。

dblink *dblink* には、表、ビューまたはマテリアライズド・ビューが存在するリモート・データベースのデータベース・リンクの完全名または部分名を指定します。このデータベースは、Oracle のデータベースである必要はありません。

参照：

- データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。
- 分散問合せの詳細は、8-15 ページの「[分散問合せ](#)」および 18-40 ページの「[分散問合せの使用例：](#)」を参照してください。

dblink を指定しない場合、その表、ビューまたはマテリアライズド・ビューは、ローカル・データベースに存在するものとみなされます。

データベース・リンクの制限事項：

- リモート表のユーザー定義型またはオブジェクト REF を問い合わせることはできません。
- リモート表の AnyType 型、AnyData 型または AnyDataSet 型の列を問い合わせることはできません。

table | view | materialized view *table*、*view* または *materialized view* には、データを選択する表、ビューまたはマテリアライズド・ビューの名前を指定します。

sample_clause

sample_clause では、表全体の行ではなく、表からランダムなサンプル行を選択します。

参照： 18-27 ページの「[サンプルの選択例：](#)」を参照してください。

BLOCK *BLOCK* を指定した場合、ランダムな行サンプリングのかわりに、ランダムなブロック・サンプリングが行われます。

参照： 相違点については、『Oracle9i データベース概要』を参照してください。

sample_percent *sample_percent* は、サンプルに含まれているとみなされる行またはブロックの割合（パーセント）を指定する数字です。値は .000001 ～ 99 の範囲内である必要があります。

問合せのサンプリングの制限事項：

- 1つの表から選択する問合せでのみ **SAMPLE** を指定できます。結合はサポートされません。ただし、**CREATE TABLE ...AS SELECT** を使用して、基礎となる表のサンプルをマテリアライズし、新しく作成されたサンプルを参照するように元の問合せを書きなおしても、同じ結果になります。他の表に対してサンプルをマテリアライズするために、追加問合せを書き込むことができます。

参照： 18-27 ページの「[サンプルの選択例：](#)」を参照してください。

- **SAMPLE** を指定した場合、自動的にコストベースのオプティマイザが使用されます。ルールベース・オプティマイザは、この句ではサポートされません。

注意： 統計的に適切でない想定値でこの機能を使用した場合、不正確な、または望ましくない結果になります。

subquery_restriction_clause *subquery_restriction_clause* を使用すると、次のいずれかの方法で副問合せを制限できます。

WITH READ ONLY 表またはビューが更新禁止であることを示す場合は、**WITH READ ONLY** を指定します。

WITH CHECK OPTION 副問合せに存在しない行を生成する表またはビューの変更が禁止であることを示す場合は、**WITH CHECK OPTION** を指定します。

CONSTRAINT constraint **CHECK OPTION** 制約の名前を指定します。この識別子を省略した場合、その制約に **SYS_cn** という形式の名前が自動的に割り当てられます。この場合の *n* は、その制約名をデータベース内で一意の名前にする整数です。

参照： 18-32 ページの「[WITH CHECK OPTION 句の使用例：](#)」を参照してください。

table_collection_expression

問合せおよび DML 操作で、*collection_expression* 値を表として扱う場合に、*table_collection_expression* を指定します。*collection_expression* は、副問合せ、列、ファンクションまたはコレクション・コンストラクタのいずれかです。その形式にかかわらず、集合値（ネストした表型または **VARRAY** 型の値）を戻す必要があります。このようなコレクションの要素抽出プロセスを **コレクション・ネスト解除** といいます。

注意： 以前のリリースの Oracle では、*collection_expression* が副問合せの場合、*table_collection_expression* を「**THE subquery**」と表現していました。現在、このような表現方法はされていません。

`collection_expression` は、FROM 句で左側に定義された表の列を参照できます。これを**左相関**といいます。左相関は `table_collection_expression` のみで行われます。その他の副問合せは、その副問合せ以外で定義された列を参照することはできません。

オプションの (+) を使用した場合、コレクションが NULL または空である場合、すべてのフィールドに NULL が設定された行を `table_collection_expression` が戻すように指定できます。この (+) は `collection_expression` が左相関を使用する場合にのみ有効です。結果は、外部結合の結果と似ています。

注意： UPDATE または DELETE 操作で副問合せの WHERE 句に「+」構文を使用する場合は、副問合せの FROM 句に 2 つの表を指定する必要があります。副問合せに結合が存在しないかぎり、外部結合構文は無視されます。

参照：

- 8-11 ページの「[外部結合](#)」を参照してください。
- 18-36 ページの「[表のコレクション例 :](#)」および 18-37 ページの「[コレクションのネスト解除例 :](#)」を参照してください。

t_alias

表、ビュー、マテリアライズド・ビューまたは問合せを評価するための副問合せの**相関名** (別名) を指定します。相関名は、相関問合せ内で頻繁に使用されます。表、ビューまたはマテリアライズド・ビューを参照する問合せでは、この別名を参照する必要があります。

注意： `query_table_expr_clause` がオブジェクト型の属性またはオブジェクト型のメソッドを参照する場合、この別名が必要です。

参照： 18-40 ページの「[相関副問合せの使用例 :](#)」を参照してください。

joined_table

`joined_table` 構文を使用すると、データが選択され、結合の一部となる表を識別できます。

参照： 結合の詳細は、8-9 ページの「[結合](#)」、18-33 ページの「[結合問合せの使用例 :](#)」、18-34 ページの「[自己結合の使用例 :](#)」および 18-34 ページの「[外部結合の使用例 :](#)」を参照してください。

join_type *join_type* には、実行する結合の種類を指定します。

- **INNER** を指定すると、内部結合が明示的に実行されます。これはデフォルト値です。
- **RIGHT** を指定すると、右側外部結合が実行されます。
- **LEFT** を指定すると、左側外部結合が実行されます。
- **FULL** を指定すると、完全な外部結合または両側外部結合が実行されます。内部結合に加え、内部結合の結果が戻されない両方の表からの行は、保持され、**NULL** で拡張されます。
- **RIGHT**、**LEFT** または **FULL** の後にオプションの **OUTER** キーワードを指定し、外部結合の実行を明示的に示すことができます。

JOIN **JOIN** キーワードは、結合の実行を明示的に示します。この構文を使用すると、**WHERE** 句の結合で使用されている、カンマで区切られた表の式を、**FROM** 句の結合構文に置き換えることができます。

ON condition **ON** 句を使用すると、結合条件を指定できます。これによって、**WHERE** 句の検索またはフィルタ条件とは分離して結合条件を指定できます。

USING column 両方の表で同じ名前の列同士を等価結合する場合、**USING column** 句に使用する列を指定します。両方の表で同じ名前の列同士を結合する場合のみ、この句を使用できます。列名を表の名前および別名で修飾しないでください。

USING 句を含む外部結合の場合、問合せによって単一列が戻されます。この単一列は、結合内の一致する 2 つの列が結合したものです。この結合は、次のように機能します。

COALESCE (a, b) = a if a NOT NULL, else b.

したがって、次のことがいえます。

- 左側外部結合では、**FROM** 句内の左側の表から共通するすべての列値が戻されます。
- 右側外部結合では、**FROM** 句内の右側の表から共通するすべての列値が戻されます。
- 完全な外部結合では、結合された両方の表から共通するすべての列値が戻されます。

結合で列を使用する場合の制限事項： **LOB** 列またはコレクション列は、**USING column** 句で指定できません。

参照： 18-34 ページの「[外部結合の使用例 :](#)」を参照してください。

CROSS JOIN **CROSS** キーワードは、クロス結合の実行を示します。クロス結合は、2 つの関係のクロス積を生成します。カンマで区切られた **Oracle** の表記法と基本的に同じです。

NATURAL JOIN NATURAL キーワードは、自然結合の実行を示します。自然結合は、2つの表の間で同じ名前のすべての列に基づきます。2つの表から関連する列の値が等しい行を選択します。自然結合で使用する列を指定する場合は、表の名前または別名で列名を修飾しないでください。

注意： 自然結合またはクロス結合の表の組合せが不明瞭な場合があります。次に例を示します。

```
a NATURAL LEFT JOIN b LEFT JOIN c ON b.c1 = c.c1
```

次のいずれかの方法に解析されます。

```
a NATURAL LEFT JOIN (b LEFT JOIN c ON b.c1 = c.c1)
(a NATURAL LEFT JOIN b) LEFT JOIN c ON b.c1 = c.c1
```

このような不明瞭さを回避するには、カッコを使用して結合する表の組合せを指定してください。このようなカッコがないと、左から右へ表が組み合せられ、左の結合が使用されます。

自然結合の制限事項： LOB 列、AnyType 列、AnyData 列、AnyDataSet 列またはコレクション列は、自然結合の一部として指定できません。

where_clause

WHERE 条件を指定すると、選択する行を 1 つ以上の条件を満たす行のみに制限できます。*condition* には、有効な SQL 条件を指定します。

この句を省略した場合、FROM 句に指定されている表、ビューまたはマテリアライズド・ビューのすべての行が戻されます。

注意： この句がパーティション表または索引の DATE 列を参照する場合、Oracle は次の 2 つの条件を満たす場合にのみパーティション・プルーニングを行います。

(1) 4 桁書式マスクの TO_DATE ファンクションを使用して年を完全に指定し、表または索引パーティションを作成する。

(2) 2 または 4 桁書式マスクの TO_DATE ファンクションを使用して、問合せの *where_clause* に日付を指定する。

参照：

- 条件の構文については、[第 5 章「条件」](#)を参照してください。
- 18-27 ページの「[パーティションからの行の選択例：](#)」を参照してください。

hierarchical_query_clause

hierarchical_query_clause では、階層順序で行を選択できます。

階層問合せを含む SELECT 文では、SELECT 構文のリスト内の LEVEL 疑似列を使用できます。LEVEL は、ルート・ノードには 1 を、ルート・ノードの子であるノードには 2 を、孫であるノードには 3 を戻します（以下同様）。階層問合せによって戻されるレベルの数値は、使用可能なユーザー・メモリーによって制限されます。

この句を指定する場合、ORDER BY および GROUP BY を指定すると、CONNECT BY 結果の階層順序が破棄されるため、これらの句のどちらも指定しないでください。親が同一の兄弟関係にある行を順序付ける場合は、ORDER SIBLINGS BY 句を使用します。

参照： 階層問合せの詳細は、8-3 ページの「[階層問合せ](#)」および 18-38 ページの「[LEVEL 疑似列の使用例：](#)」を参照してください。

START WITH 句

階層問合せのルートとして使用される行を識別する場合に条件を指定します。Oracle では、この条件を満たすすべての行がルートで使用されます。この句を省略した場合、表内のすべての行がルート行として使用されます。START WITH 条件には、副問合せを含めることができますが、スカラー副問合せ式を含めることはできません。

CONNECT BY 句

階層の親 / 子の行の関連を識別する条件を指定します。*connect_by_condition* には、[第 5 章「条件」](#)のすべての条件を含めることができます。ただし、親である行を参照するための PRIOR 演算子を使用する必要があります。

CONNECT BY 句の制限事項： *connect_by_condition* には、正規の副問合せおよびスカラー副問合せ式を含めることはできません。

参照：

- LEVEL の詳細は、2-77 ページの「[疑似列](#)」を参照してください。
- 階層問合せの概要については、8-3 ページの「[階層問合せ](#)」を参照してください。
- 18-31 ページの「[階層問合せの例：](#)」を参照してください。

階層問合せに関する注意事項：

ORDER BY 句とともに階層問合せを指定する場合、ORDER BY 句に SIBLINGS キーワードを指定しないかぎり、ORDER BY 句は、階層問合せによって指定される順序より優先されます。

階層問合せでの WHERE 句の Oracle の処理は、WHERE 句が結合を含むかどうかによって異なります。

- WHERE 句が結合を含む場合、Oracle は、CONNECT BY 処理の前に述語結合を適用します。
- WHERE 句に結合が含まれない場合、Oracle は、CONNECT BY 処理の後に、CONNECT BY 述語以外のすべての述語を適用します。この場合、階層の他の行に影響はありません。

group_by_clause

それぞれの行の *expr* の値に基づいて選択した行をグループ化し、各グループのサマリー情報を 1 行戻す場合に、GROUP BY 句を使用します。この句に CUBE または ROLLUP 拡張要素が指定された場合、標準グループ化の他に超集合グループ化が生成されます。

GROUP BY 句の式には、SELECT 構文のリストに指定されている列であるかどうかにかかわらず、FROM 句の表、ビューおよびマテリアライズド・ビューの列を指定できます。

GROUP BY 句は行をグループ化しますが、結果セットの順序は保証しません。グループ化の順序付けを行うには、ORDER BY 句を使用します。

参照：

- データを集計する SQL グループ化構文の詳細および例については、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。
- この句の例については、6-67 ページの「[GROUP_ID](#)」、「[GROUPING](#)」および「[GROUPING_ID](#)」ファンクションを参照してください。
- 18-29 ページの「[GROUP BY 句の使用例：](#)」を参照してください。

ROLLUP *simple_grouping_clause* の ROLLUP 操作は、GROUP BY で指定した式 *n*、*n*-1、*n*-2、... 0 の最初の値に基づいて選択した行をグループ化します。また、それぞれのグループのサマリー情報を 1 行戻します。ROLLUP 操作を SUM ファンクションとともに使用して、**小計値**を出力できます。ROLLUP を SUM とともに使用すると、最も詳細なレベルの小計から総計までが生成されます。COUNT などの集計ファンクションは、他の種類の超集合の出力に使用できます。

たとえば、*simple_grouping_clause* の ROLLUP 句に式を 3 つ指定した場合 (*n*=3)、操作の結果は *n*+1=3+1=4 グループになります。

最初の '*n*' 式の値でグループ化した行を**標準行**、その他を**超集合行**といいます。

参照： マテリアライズド・ビューで ROLLUP を使用する場合は、『Oracle9i データ・ウェアハウス・ガイド』を参照してください。

CUBE *simple_grouping_clause* の CUBE 操作は、指定した式のあらゆる組合せの値に基づいて選択した行をグループ化します。また、それぞれのグループのサマリー情報を 1 行戻します。CUBE 操作を使用して、**クロス集計値**を出力できます。

たとえば、*simple_grouping_clause* の CUBE 句に式を 3 つ指定した場合 ($n=3$)、操作の結果は $2^n = 2^3 = 8$ グループになります。'n' 式の値でグループ化した行を**標準行**、その他を**超集合行**といいます。

参照：

- マテリアライズド・ビューで CUBE を使用する場合は、[『Oracle9i データ・ウェアハウス・ガイド』](#)を参照してください。
- 18-29 ページの「[GROUP BY CUBE 句の使用例 :](#)」を参照してください。

GROUPING SETS GROUPING SETS は、データを複数にグループ化する GROUP BY 句をさらに拡張したものです。これによって、不要な集計が排除され、効率的に集計できるようになります。必要なグループを指定すると、Oracle が CUBE または ROLLUP によって生成された集計のすべてを実行する必要がなくなります。GROUPING SETS 句で指定したすべてのグループ化が計算され、UNION ALL 操作で個々のグループ化の結果が組み合わせられます。UNION ALL は、結果セットが重複行を含むことを許可します。

GROUP BY 句では、様々な方法で式を組み合わせることができます。

- **複合列**を指定するには、カッコで列をグループ化します。Oracle は、ROLLUP 操作または CUBE 操作の計算でこれらを一単位として処理します。
- **グルーピング・セットの連結**を指定するには、カンマで複数のグルーピング・セット、ROLLUP 操作および CUBE 操作を分割します。Oracle は、これらを 1 つの GROUP BY 句に結合します。結果は、各グルーピング・セットからのグループ化のクロス積です。

参照： 18-29 ページの「[GROUPING SETS 句の使用例 :](#)」を参照してください。

HAVING 句

戻す行のグループを、指定した条件が真である行のグループのみに制限する場合に、HAVING 句を使用します。この句を省略した場合、すべてのグループのサマリー行が戻されます。

where_clause および *hierarchical_query_clause* の後に、GROUP BY および HAVING を指定します。GROUP BY および HAVING 句を両方指定する場合は、どちらを先に指定してもかまいません。

HAVING 句の制限事項： HAVING 条件に、スカラー副問合せ式を含めることはできません。

参照： 18-31 ページの「[HAVING 条件の使用例 :](#)」を参照してください。

GROUP BY 句の制限事項： 式には、スカラー副問合せ式を除くすべての形式が可能です。

- LOB 列、ネストした表または VARRAY を *expr* の一部として指定できません。
- *group_by_clause* がオブジェクト型列を参照する場合、問合せはパラレル化されません。

参照： *expr* の構文については、4-2 ページの「SQL 式」を、条件の構文については、第 5 章「条件」を参照してください。

集合演算子：UNION、UNION ALL、INTERSECT および MINUS

これらの集合演算子は、2 つの SELECT 文によって戻された行を 1 つの結果に結合します。それぞれのコンポーネント問合せで選択される列の数とデータ型は同じである必要がありますが、列の長さは異なってもかまいません。結果セット内の列の名前は、集合演算子の前にある SELECT 構文のリスト内の式の名前です。

集合演算子で 2 つ以上の問合せを結合する場合、隣接する問合せを左から右へと評価します。この評価順序を変更する場合、カッコを使用します。

参照： これらの演算子の詳細は、8-7 ページの「UNION [ALL]、INTERSECT および MINUS 演算子」を参照してください。

集合演算子の制限事項：

- 集合演算子は、データ型が BLOB、CLOB、BFILE、VARRAY またはネストした表である列に対しては無効になります。
- UNION、INTERSECT および MINUS 演算子は、LONG 列に対しては無効になります。
- 集合演算子の前の SELECT 構文のリストに式が含まれている場合、*order_by_clause* でその式を参照するには、式に列の別名を指定する必要があります。
- *for_update_clause* は、これらの集合演算子とともに指定できません。
- これらの演算子の副問合せには、*order_by_clause* を指定できません。
- TABLE コレクション式を含む SELECT 文では、これらの演算子を使用できません。

注意： SQL 規格に準拠するために、Oracle の今後のリリースでは、他の集合演算子より優先順位の高い INTERSECT 演算子が提供されます。したがって、INTERSECT 演算子と他の集合演算子を使用する問合せでは、カッコを使用して評価順序を指定してください。

order_by_clause ORDER BY 句を使用して、文によって戻された行を順序付けます。
order_by_clause を指定しない場合、同じ問合せで取り出される行の順序が異なることがあります。

SIBLINGS SIBLINGS キーワードは、*hierarchical_query_clause* (CONNECT BY) を指定する場合のみに有効です。ORDER SIBLINGS BY は階層問合せ句で指定した任意の順序を保持し、兄弟関係にある階層に *order_by_clause* を適用します。

expr *expr* は、*expr* の値を基準にして行に順番を付けます。式は、SELECT 構文のリストの列、あるいは FROM 句の表、ビューまたはマテリアライズド・ビューの列に基づきます。

position *position* は、SELECT 構文のリストのその位置にある式の値に基づいて行を順序付けます。*position* には整数を指定する必要があります。

参照： 問合せ結果の順位付けの詳細は、8-9 ページの「[問合せ結果のソート](#)」を参照してください。

order_by_clause には複数の式を指定できます。この場合、まず、最初の式の値に基づいて行がソートされ、次に、最初の式と同じ値を持つ行が 2 番目の式の値に基づいてソートされる、というように処理が行われます。NULL 値は昇順では最後に、降順では先頭にソートされます。

ASC | DESC 昇順か降順かを指定します。デフォルトは ASC です。

NULLS FIRST | NULLS LAST NULL 値を含む戻された行が順序の最初にくるか、最後にくるかを指定します。

NULLS LAST は昇順のデフォルトで、NULLS FIRST は降順のデフォルトです。

ORDER BY 句の制限事項：

- この文中で DISTINCT 演算子を指定した場合、SELECT 構文のリストに指定された列でないかぎり、この句は列を参照することはできません。
- *order_by_clause* には最大 255 個の式を指定できます。
- LOB 列、ネストした表または VARRAY を使用しての順位付けはできません。
- 同じ文中で *group_by_clause* を指定する場合、この *order_by_clause* は次の式に制限されます。
 - 定数
 - 集計ファンクション
 - 分析ファンクション
 - USER ファンクション、UID ファンクションおよび SYSDATE ファンクション
 - *group_by_clause* に指定されているものと同じ式
 - グループ内のすべての行が同じ値に評価されるこれらの式を導出する式

参照： 18-31 ページの「[ORDER BY 句の使用例 :](#)」を参照してください。

for_update_clause

FOR UPDATE 句によって、トランザクションが終了する前に、選択した行が別のユーザーによってロックまたは更新されることがないように、選択した行をロックします。最上位の SELECT 文でのみ、この句を指定できます。副問合せでは指定できません。

注意： LOB 値を更新する場合、その LOB を含む行をロックしておく必要があります。行をロックする方法の 1 つに、埋込み SELECT ... FOR UPDATE 文があります。この場合、プログラム言語の 1 つまたは DBMS_LOB パッケージを使用します。LOB の書込み前に行う行のロックの詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

親表の行がロックされても、ネストした表の行はロックされません。ネストした表の行をロックする場合、ネストした表を明示的にロックする必要があります。

FOR UPDATE 句の制限事項：

- この句を DISTINCT 演算子または CURSOR 式、集合演算子、*group_by_clause*、または集計関数ファクションの構造体とともに指定することはできません。
- この句がロックした表は、同じ文で参照された LONG 列および順序と同じデータベース内にある必要があります。

参照： 18-32 ページの「[FOR UPDATE 句の使用例 :](#)」を参照してください。

OF ... column

結合内の特定の表の選択された行のみをロックする場合に、OF ... column 句を使用します。OF 句の列は、どの表またはビューの行をロックするかを識別する場合にのみ使用します。指定する列は重要ではありません。ただし、列の別名ではなく、実際の列名を指定する必要があります。この句を省略した場合、問合せ内のすべての表の選択された行がロックされます。

NOWAIT | WAIT

NOWAIT および WAIT 句は、他のユーザによってロックされている行を SELECT 文がロックしようとする場合に処理する方法を Oracle に指示します。

NOWAIT NOWAIT を指定すると、ロックされている場合に制御がすぐに戻ります。

WAIT WAIT を指定すると、行が使用可能になるまで *integer* 秒待機した後で制御が戻されます。

WAIT および NOWAIT のどちらも指定しない場合、行が使用可能になるまで待機した後で SELECT 文の結果が戻されます。

例

副問合せのファクタリング例： 次の文は、結合を含む初期間合せブロックに対する問合せの名前 dept_costs および avg_cost を作成し、主問合せの本体でその問合せの名前を使用します。

```
WITH
  dept_costs AS (
    SELECT department_name, SUM(salary) dept_total
      FROM employees e, departments d
     WHERE e.department_id = d.department_id
    GROUP BY department_name),
  avg_cost AS (
    SELECT SUM(dept_total)/COUNT(*) avg
      FROM dept_costs)
SELECT * FROM dept_costs
  WHERE dept_total >
    (SELECT avg FROM avg_cost)
 ORDER BY department_name;
```

DEPARTMENT_NAME	DEPT_TOTAL

Sales	313800
Shipping	156400

単純問合せの例： 次の文は、部門番号 30 の従業員表 employees の行を選択します。

```
SELECT *
  FROM employees
 WHERE department_id = 30;
```

次の文は、部門番号 30 の購買係を除くすべての従業員の名前、職種、給与および部門番号を選択します。

```
SELECT last_name, job_id, salary, department_id
  FROM employees
 WHERE NOT (job_id = 'PU_CLERK' AND department_id = 30);
```

次の文は、FROM 句の副問合せから、部門のすべての従業員数と給与合計がすべての部門に占める割合を算出します。

```
SELECT a.department_id "Department",
       a.num_emp/b.total_count "%_Employees",
       a.sal_sum/b.total_sal "%_Salary"
FROM
  (SELECT department_id, COUNT(*) num_emp, SUM(salary) sal_sum
   FROM employees
   GROUP BY department_id) a,
  (SELECT COUNT(*) total_count, SUM(salary) total_sal
   FROM employees) b;
```

パーティションからの行の選択例： FROM 句にキーワード PARTITION を指定することによって、パーティション表の 1 つのパーティションから行を選択できます。次の SQL 文は、サンプル表 sh.sales の sales_q2_2000 パーティションへ別名を割り当て、行を取り出します。

```
SELECT * FROM sales PARTITION (sales_q2_2000) s
WHERE s.amount_sold > 10000;
```

次の文は、oe.orders 表から指定した日付より前の注文の行を選択します。

```
SELECT * FROM orders
WHERE order_date < TO_DATE('2000-06-15', 'YYYY-MM-DD');
```

サンプルの選択例： 次の問合せは、oe.orders 表の注文数を推定します。

```
SELECT COUNT(*) * 100 FROM orders SAMPLE (1);
```

次の文は、サンプル表 hr.employees のサンプル・サブセットを作成し、結果のサンプル表と departments を結合します。これによって、結合問合せで *sample_clause* を指定できないという制限を回避できます。

```
CREATE TABLE sample_emp AS
  SELECT employee_id, department_id FROM employees SAMPLE(10);

SELECT e.employee_id FROM sample_emp e, departments d
WHERE e.department_id = d.department_id
AND d.department_name = 'Sales';
```

フラッシュバック問合せの使用例： 次の文は、サンプル表 `hr.employees` の現在の値を表示し、値を変更します。

```
SELECT salary FROM employees
WHERE last_name = 'Chung';
```

```
      SALARY
-----
      3800
```

```
UPDATE employees SET salary = 4000
WHERE last_name = 'Chung';
1 row updated.
```

```
SELECT salary FROM employees
WHERE last_name = 'Chung';
```

```
      SALARY
-----
      4000
```

更新前の値を確認するには、次のフラッシュバック問合せを使用します。

```
SELECT salary FROM employees
AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY)
WHERE last_name = 'Chung';
```

```
      SALARY
-----
      3800
```

元の値に戻すには、フラッシュバック問合せを別の UPDATE 文の副問合せとして使用します。

```
UPDATE employees SET salary =
  (SELECT salary FROM employees
   AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY)
   WHERE last_name = 'Chung')
WHERE last_name = 'Chung';
1 row updated.
```

```
SELECT salary FROM employees
WHERE last_name = 'Chung';
```

```
      SALARY
-----
      3800
```

GROUP BY 句の使用例： 次の文は、employees 表の各部門について最低給与と最高給与を戻します。

```
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
GROUP BY department_id;
```

次の文は、各部門の事務員について最高給与と最低給与を戻します。

```
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
WHERE job_id = 'PU_CLERK'
GROUP BY department_id;
```

GROUP BY CUBE 句の使用例： 部門および職種のすべての組合せについて、従業員数と平均年収を戻すには、サンプル表 hr.employees および hr.departments に次の問合せを発行します。

```
SELECT DECODE(GROUPING(department_name), 1, 'All Departments',
              department_name) AS department_name,
       DECODE(GROUPING(job_id), 1, 'All Jobs', job_id) AS job_id,
       COUNT(*) "Total Empl", AVG(salary) * 12 "Average Sal"
FROM employees e, departments d
WHERE d.department_id = e.department_id
GROUP BY CUBE (department_name, job_id);
```

DEPARTMENT_NAME	JOB_ID	Total Empl	Average Sal
Accounting	AC_ACCOUNT	1	99600
Accounting	AC_MGR	1	144000
Accounting	All Jobs	2	121800
Administration	AD_ASST	1	52800
.			
.			
.			
All Departments	ST_MAN	5	87360
All Departments	All Jobs	107	77798.1308

GROUPING SETS 句の使用例： 次の例は、指定した 3 つのグループで集計した販売の合計を示します。

- (channel_desc, calendar_month_desc, country_id)
- (channel_desc, country_id)
- (calendar_month_desc, country_id)

GROUPING SETS 構文を指定しない場合、SQL はより複雑になり、問合せの効果は低くなります。たとえば、3 つの分散した問合せを実行し、重複を含まないすべての行を戻す、または CUBE(channel_desc, calendar_month_desc, country_id) 操作を指定した問合せを実行し、生成される 8 つのグループから 5 つを除去します。

```
SELECT channel_desc, calendar_month_desc, co.country_id,
       TO_CHAR(sum(amount_sold) , '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries co
WHERE sales.time_id=times.time_id
      AND sales.cust_id=customers.cust_id
      AND sales.channel_id= channels.channel_id
      AND customers.country_id = co.country_id
      AND channels.channel_desc IN ('Direct Sales', 'Internet')
      AND times.calendar_month_desc IN ('2000-09', '2000-10')
      AND co.country_id IN ('UK', 'US')
GROUP BY GROUPING SETS(
    (channel_desc, calendar_month_desc, co.country_id),
    (channel_desc, co.country_id),
    ( calendar_month_desc, co.country_id ) );
```

CHANNEL_DESC	CALENDAR	CO	SALES\$
-----	-----	----	-----
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-10	US	2,908,706
Internet	2000-09	UK	911,739
Internet	2000-10	UK	876,571
Internet	2000-09	US	1,732,240
Internet	2000-10	US	1,893,753
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Internet		UK	1,788,310
Internet		US	3,625,993
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459

参照： これらのファンクションの詳細は、6-67 ページの「[GROUP_ID](#)」、「[GROUPING](#)」および「[GROUPING_ID](#)」ファンクションを参照してください。

階層問合せの例： CONNECT BY 句を指定した次の問合せは、親である行の employee_id 値が子である行の manager_id 値と等しいという階層関係を定義します。

```
SELECT last_name, employee_id, manager_id FROM employees
       CONNECT BY employee_id = manager_id;
```

次の CONNECT BY 句では、PRIOR 演算子が employee_id 値のみに適用されます。この条件を評価するために、Oracle は親である行に対しては employee_id の値を評価し、子である行に対しては manager_id、salary および commission_pct のそれぞれの値を評価します。

```
SELECT last_name, employee_id, manager_id FROM employees
       CONNECT BY PRIOR employee_id = manager_id
       AND salary > commission_pct;
```

子である行を限定する場合、manager_id の値と親である行の employee_id の値が等しく、salary の値が commission_pct の値より大きい必要があります。

HAVING 条件の使用例： 次の文は、従業員の最低給与が 5,000 ドル未満の部門についての最高給与と最低給与を戻します。

```
SELECT department_id, MIN(salary), MAX (salary)
       FROM employees
       GROUP BY department_id
       HAVING MIN(salary) < 5000;
```

DEPARTMENT_ID	MIN(SALARY)	MAX(SALARY)
10	4400	4400
30	2500	11000
50	2100	8200
60	4200	9000

ORDER BY 句の使用例： 次の文は、employees 表から営業担当員のすべてのレコードを選択し、その歩合によって降順にソートします。

```
SELECT *
       FROM employees
       WHERE job_id = 'PU_CLERK'
       ORDER BY commission_pct DESC;
```

次の文は、employees 表から情報を選択し、最初に部門番号で昇順にソートした後、給与で降順にソートします。

```
SELECT last_name, department_id, salary
       FROM employees
       ORDER BY department_id ASC, salary DESC;
```

次の文は、先の SELECT 文と同じ情報を選択し、位置に基づく ORDER BY 句の指定を使用します。

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY 2 ASC, 3 DESC;
```

FOR UPDATE 句の使用例： 次の文は、employees 表中のオックスフォード勤務 (location_id は 2500) の購買係の行をロックし、departments 表中の購買係が存在するオックスフォードの部門の行をロックします。

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e, departments d
WHERE job_id = 'SA_REP'
AND e.department_id = d.department_id
AND location_id = 2500
FOR UPDATE;
```

次の文は、employees 表中のオックスフォード勤務 (location_id は 2500) の購買係の行のみをロックします。departments 表では、行のロックはできません。

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e, departments d
WHERE job_id = 'SA_REP'
AND e.department_id = d.department_id
AND location_id = 2500
FOR UPDATE OF e.salary;
```

WITH CHECK OPTION 句の使用例： 次の文は、3 番目の値が副問合せ *where_clause* の条件に違反していても有効です。

```
INSERT INTO (SELECT department_id, department_name, location_id
FROM departments WHERE location_id < 2000)
VALUES (9999, 'Entertainment', 2500);
```

ただし、次の文は WITH CHECK OPTION 句を含むため、無効になります。

```
INSERT INTO (SELECT department_id, department_name, location_id
FROM departments WHERE location_id < 2000 WITH CHECK OPTION)
VALUES (9999, 'Entertainment', 2500);
*
```

ERROR at line 2:

ORA-01402: ビューの WITH CHECK OPTION WHERE 句でエラーが発生しました。

結合問合せの使用例： 次の例は、問合せにおける様々な表の結合方法を示します。最初の例では、等価結合は、それぞれの従業員の名前と職種、およびその従業員が属する部門の番号と名前を戻します。

```
SELECT last_name, job_id, departments.department_id, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
...			
Sciarra	FI_ACCOUNT	100	Finance
Urman	FI_ACCOUNT	100	Finance
Popp	FI_ACCOUNT	100	Finance
...			

従業員の名前および職種は部門名とは別の表に格納されているため、このデータを戻す場合は結合を使用する必要があります。次の結合条件に従って、2つの表の行が結合されます。

```
employees.department_id = departments.department_id
```

次の等価結合は、すべての販売マネージャの名前、職種、部門番号および部門名を戻します。

```
SELECT last_name, job_id, departments.department_id, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND job_id = 'SA_MAN';
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
...			
Russell	SA_MAN	80	Sales
Partners	SA_MAN	80	Sales
Errazuriz	SA_MAN	80	Sales
Cambrault	SA_MAN	80	Sales
Zlotkey	SA_MAN	80	Sales
...			

この問合せは、次の *where_clause* を使用して 'SA_MAN' という job 値を持つ行のみを戻すこと以外は、前述の例と同じです。

副問合せの使用例： 次の文は、従業員 'Lorentz' と同じ部門で働く従業員を判断します。

```
SELECT last_name, department_id FROM employees
WHERE department_id =
(SELECT department_id FROM employees
WHERE last_name = 'Lorentz');
```

employees 表の職種を変更した (job_history 表に示される) すべての従業員の給与を 10% アップする場合、次の文を発行します。

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE employee_id IN (SELECT employee_id FROM job_history);
```

departments 表から 3 つの列のみを伴って新しい表 new_departments を作成するには、次の文を発行します。

```
CREATE TABLE new_departments
  (department_id, department_name, location_id)
  AS SELECT department_id, department_name, location_id
  FROM departments;
```

自己結合の使用例： 次の問合せは、自己結合を使用して、それぞれの従業員の名前と一緒にその従業員の上司の名前を戻します (出力を短くするために WHERE 句は追加されています)。

```
SELECT e1.last_name||' works for '||e2.last_name
       "Employees and Their Managers"
FROM   employees e1, employees e2
WHERE  e1.manager_id = e2.employee_id
       AND e1.last_name LIKE 'R%';
```

Employees and Their Managers

```
Rajs works for Mourgos
Raphaely works for King
Rogers works for Kaufling
Russell works for King
```

この問合せの結合条件では、サンプル表 employees に対する別名 e1 および e2 を使用します。

```
e1.manager_id = e2.employee_id
```

外部結合の使用例： 次の例は、左側外部結合を使用し、従業員がいない部門を含むサンプル・スキーマ hr 内のすべての部門名を戻します。

```
SELECT d.department_id, e.last_name
FROM   departments d LEFT OUTER JOIN employees e
      ON d.department_id = e.department_id
ORDER BY d.department_id;
```

DEPARTMENT_ID LAST_NAME

```
10 Whalen
```

```

        20 Hartstein
        20 Fay
        30 Raphaely
...
        250
        260
        270

```

これは、次に示す以前の Oracle の外部結合の構文と同じ問合せです。

```

SELECT d.department_id, e.last_name
FROM departments d, employees e
WHERE d.department_id = e.department_id(+)
ORDER BY d.department_id;

```

この構文ではなく、前述の例で示した、より柔軟性が高い Oracle9i の FROM 句の結合構文を使用することをお勧めします。

左側外部結合では、従業員のいない部門を含むすべての部門が戻されます。右側外側結合が指定された同一文では、どの部門にも割り当てられていない従業員を含むすべての従業員が戻されます。

注意： 前述の例の従業員表には、従業員 Zeuss が追加されていますが、この従業員はサンプル・データには含まれません。

```

SELECT d.department_id, e.last_name
FROM departments d RIGHT OUTER JOIN employees e
ON d.department_id = e.department_id
ORDER BY d.department_id;

```

```

DEPARTMENT_ID LAST_NAME
-----
...

```

```

        110 Higgins
        110 Gietz
            Grant
            Zeuss

```

この結果からは、Grant と Zeuss という従業員の department_id が NULL かどうか、またはその department_id が departments 表に存在するかどうかは不明です。これを確認するには、完全な外部結合が必要です。

```

SELECT d.department_id as d_dept_id, e.department_id as e_dept_id,
       e.last_name
FROM departments d FULL OUTER JOIN employees e
ON d.department_id = e.department_id

```

```
ORDER BY d.department_id;

D_DEPT_ID  E_DEPT_ID LAST_NAME
-----
...
      110          110 Gietz
      110          110 Higgins
...
      260
      270
                999 Zeuss
                Grant
```

この例の列名は、結合状態にある両方の表で同じであるため、結合構文の共通列機能 (USING 句) も使用できます。この機能によって、一致する 2 つの department_id 列が結合されます。出力は、前述の例と同じです。

```
SELECT department_id AS d_e_dept_id, e.last_name
FROM departments d FULL OUTER JOIN employees e
USING (department_id)
ORDER BY department_id;

D_E_DEPT_ID LAST_NAME
-----
...
      110 Higgins
      110 Gietz
...
      260
      270
                Grant
                Zeuss
```

表のコレクション例： DML 操作は、表の列として定義された場合にのみ、ネストした表で実行できます。したがって、INSERT、DELETE または UPDATE 文の *query_table_expr_clause* が *table_collection_expression* の場合、コレクション式は、TABLE ファンクションを使用して表のネストした表の列を選択する副問合せである必要があります。次の例は、次の使用例に基づいています。

データベースに、department_id 列、location 列、manager_id 列を持つ hr_info 表と、マネージャごとのすべての従業員の last_name 列、department_id 列および salary 列を持つネストした表型 people の列が含まれていると仮定します。

```
CREATE TYPE people_typ AS OBJECT (
    last_name      VARCHAR2(25),
    department_id  NUMBER(4),
    salary         NUMBER(8,2));
/
```

```

CREATE TYPE people_tab_typ AS TABLE OF people_typ;
/
CREATE TABLE hr_info (
    department_id    NUMBER(4),
    location_id      NUMBER(4),
    manager_id       NUMBER(6),
    people            people_tab_typ)
    NESTED TABLE people STORE AS people_stor_tab;

```

```
INSERT INTO hr_info VALUES (280, 1800, 999, people_tab_typ());
```

次の例では、hr_info 表の部門番号 280 のネストした表 people の列に値を挿入します。

```

INSERT INTO TABLE(SELECT h.people FROM hr_info h
    WHERE h.department_id = 280)
VALUES ('Smith', 280, 1750);

```

次の例では、部門番号 280 のネストした表 people を更新します。

```

UPDATE TABLE(SELECT h.people FROM hr_info h
    WHERE h.department_id = 280) p
SET p.salary = p.salary + 100;

```

次の例では、部門番号 280 のネストした表 people を削除します。

```

DELETE TABLE(SELECT h.people FROM hr_info h
    WHERE h.department_id = 280) p
WHERE p.salary > 1700;

```

コレクションのネスト解除例： ネストした表の列からデータを選択するには、TABLE ファンクションを再度使用して、ネストした表を表の列として処理します。このプロセスをコレクション・ネスト解除と呼びます。

次の文を使用して、hr_info（前述の例で作成）および hr_info にネストした表 people の列から、すべての行を取得できます。

```

SELECT t1.department_id, t2.* FROM hr_info t1, TABLE(t1.people) t2
    WHERE t2.department_id = t1.department_id;

```

people は、hr_info のネストした表の列ではなく、last_name、department_id、address、hiredate および salary 列と別の表であると仮定します。次の文を使用して、前述の例と同じ行を抽出できます。

```

SELECT t1.department_id, t2.*
    FROM hr_info t1, TABLE(CAST(MULTISET(
        SELECT t3.last_name, t3.department_id, t3.salary
        FROM people t3
        WHERE t3.department_id = t1.department_id)
    AS people_tab_typ)) t2;

```

最後に、people は hr_info 表のネストした表の列でも、表そのものでもないと仮定します。かわりに、すべての従業員の名前、部門および給与を様々な情報から抽出する people_func ファンクションを作成しておきます。次の問合せを使用して、前述の例と同様の情報を得ることができます。

```
SELECT t1.department_id, t2.* FROM hr_info t1, TABLE(CAST
    (people_func( ... ) AS people_tab_typ)) t2;
```

参照： コレクション・ネスト解除の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

LEVEL 疑似列の使用例： 次の文を実行すると、すべての従業員が階層順序で戻されます。職種が 'AD_VP' である従業員がルート行となるように定義されています。また、親である行の従業員番号が上司の従業員番号となるように、親である行の子である行が定義されています。

```
SELECT LPAD(' ',2*(LEVEL-1)) || last_name org_chart,
       employee_id, manager_id, job_id
FROM employees
START WITH job_id = 'AD_VP'
CONNECT BY PRIOR employee_id = manager_id;
```

ORG_CHART	EMPLOYEE_ID	MANAGER_ID	JOB_ID
-----	-----	-----	-----
Kochhar	101	100	AD_VP
Greenberg	108	101	FI_MGR
Faviet	109	108	FI_ACCOUNT
Chen	110	108	FI_ACCOUNT
Sciarra	111	108	FI_ACCOUNT
Urman	112	108	FI_ACCOUNT
Popp	113	108	FI_ACCOUNT
Whalen	200	101	AD_ASST
Mavris	203	101	HR_REP
Baer	204	101	PR_REP
Higgins	205	101	AC_MGR
Gietz	206	205	AC_ACCOUNT
De Haan	102	100	AD_VP
Hunold	103	102	IT_PROG
Ernst	104	103	IT_PROG
Austin	105	103	IT_PROG
Pataballa	106	103	IT_PROG
Lorentz	107	103	IT_PROG

次の文は、前述の例とほぼ同じですが、職種が 'FI_MGR' である従業員は選択されません。

```
SELECT LPAD(' ',2*(LEVEL-1)) || last_name org_chart,
       employee_id, manager_id, job_id
FROM employees
WHERE job_id != 'FI_MGR'
START WITH job_id = 'AD_VP'
CONNECT BY PRIOR employee_id = manager_id;
```

ORG_CHART	EMPLOYEE_ID	MANAGER_ID	JOB_ID
Kochhar	101	100	AD_VP
Faviet	109	108	FI_ACCOUNT
Chen	110	108	FI_ACCOUNT
Sciarra	111	108	FI_ACCOUNT
Urman	112	108	FI_ACCOUNT
Popp	113	108	FI_ACCOUNT
Whalen	200	101	AD_ASST
Mavris	203	101	HR_REP
Baer	204	101	PR_REP
Higgins	205	101	AC_MGR
Gietz	206	205	AC_ACCOUNT
De Haan	102	100	AD_VP
Hunold	103	102	IT_PROG
Ernst	104	103	IT_PROG
Austin	105	103	IT_PROG
Pataballa	106	103	IT_PROG
Lorentz	107	103	IT_PROG

Greenberg が管理する従業員は戻されますが、マネージャ Greenberg は戻されません。

次の文も、前述の例と同じですが、LEVEL 疑似列を使用して管理階層の最初の 2 つのレベルのみが選択されます。

```
SELECT LPAD(' ',2*(LEVEL-1)) || last_name org_chart,
       employee_id, manager_id, job_id
FROM employees
START WITH job_id = 'AD_PRES'
CONNECT BY PRIOR employee_id = manager_id AND LEVEL <= 2;
```

ORG_CHART	EMPLOYEE_ID	MANAGER_ID	JOB_ID
King	100		AD_PRES
Kochhar	101	100	AD_VP
De Haan	102	100	AD_VP
Raphaely	114	100	PU_MAN
Weiss	120	100	ST_MAN

Frapp	121	100 ST_MAN
Kaufling	122	100 ST_MAN
Vollman	123	100 ST_MAN
Mourgos	124	100 ST_MAN
Russell	145	100 SA_MAN
Partners	146	100 SA_MAN
Errazuriz	147	100 SA_MAN
Cambrault	148	100 SA_MAN
Zlotkey	149	100 SA_MAN
Hartstein	201	100 MK_MAN

分散問合せの使用例： 次の文は、ローカル・データベース上の `departments` 表と、remote データベース上の `employees` 表を結合します。

```
SELECT last_name, department_name
FROM employees@remote, departments
WHERE employees.department_id = departments.department_id;
```

相関副問合せの使用例： 次に、相関副問合せの構文の一般的な例を示します。

```
SELECT select_list
FROM table1 t_alias1
WHERE expr operator
      (SELECT column_list
       FROM table2 t_alias2
       WHERE t_alias1.column
            operator t_alias2.column);

UPDATE table1 t_alias1
SET column =
      (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column);

DELETE FROM table1 t_alias1
WHERE column operator
      (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column);
```

次の文は、部門内の平均給与を超える給与を支給されている従業員の情報を戻します。給与情報が格納されている `employees` 表に別名を割り当て、相関副問合せではその別名を使用します。

```
SELECT department_id, last_name, salary
FROM employees x
WHERE salary > (SELECT AVG(salary)
                FROM employees)
```

```
WHERE x.department_id = department_id)
ORDER BY department_id;
```

親問合せでは、相関副問合せを使用して同一部門の従業員の平均給与を、employees 表の行ごとに計算します。相関副問合せは、employees 表の各行について次の手順を実行します。

1. 行の department_id を判断します。
2. department_id に基づいて親問合せが評価されます。
3. 行の部門の平均給与より高い給与の行がある場合は、その行を戻します。

副問合せは、employees 表の各行につき 1 回ずつ評価されます。

DUAL 表からの選択例： 次の文は、現在の日付を戻します。

```
SELECT SYSDATE FROM DUAL;
```

employees 表から簡単に SYSDATE を選択できますが、このとき、employees 表のすべての行に対して 1 件ずつ 14 行の同じ SYSDATE が戻ります。このため、DUAL から選択する方が便利です。

順序値の選択例： 次の文は、employees_seq 順序を増分し、新しい値を戻します。

```
SELECT employees_seq.nextval
FROM dual;
```

次の文は、employees_seq の現在値を選択します。

```
SELECT employees_seq.currval
FROM dual;
```

SET CONSTRAINT[S]

用途

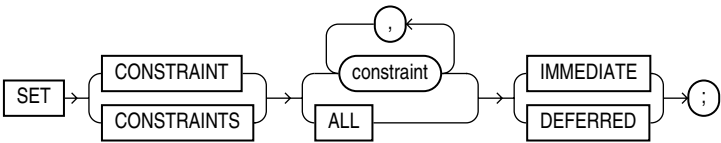
SET CONSTRAINTS 文を使用すると、遅延可能な制約の検証を、各 DML 文の実行後に行うか、トランザクションのコミット時に行うかをトランザクションごとに指定できます。

前提条件

遅延可能な制約を検証する時期を指定する場合は、制約が適用される表に対する SELECT 権限を持っているか、またはその表が自分のスキーマ内にある必要があります。

構文

set_constraints::=



セマンティクス

constraint

1 つ以上の整合性制約の名前を指定します。

ALL

このトランザクション内のすべての遅延可能な制約を設定する場合に、ALL を指定します。

IMMEDIATE

遅延可能な制約によって指定された条件が、各 DML 文の直後に検証されるようにする場合に、IMMEDIATE を指定します。

DEFERRED

遅延可能な制約によって指定された条件が、トランザクションのコミット時に検証されるようにする場合に、DEFERRED を指定します。

注意： SET CONSTRAINTS ALL IMMEDIATE 文を発行することによって、遅延可能な制約をコミットする前に、それらの制約が完全に適用されたかどうかを検証できます。

例

制約の設定例： 次の文は、このトランザクション内のすべての遅延可能な制約が、各 DML 文の直後に検証されるように設定します。

```
SET CONSTRAINTS ALL IMMEDIATE;
```

次の文は、トランザクションのコミット時に 3 つの遅延制約を検証します。この例では、制約を NOT DEFERRABLE に設定すると失敗します。

```
SET CONSTRAINTS emp_job_nn, emp_salary_min ,  
                hr.jhist_dept_fk@remote DEFERRED;
```

SET ROLE

用途

SET ROLE 文を使用すると、現行のセッションのロールを使用可能化、使用禁止化またはコンパイルすることができます。

ユーザー・ログイン時に、Oracle は、ユーザーに明示的に付与されたすべての権限およびユーザーのすべてのデフォルトのロールを使用可能にします。セッション中、ユーザーまたはアプリケーションはそのセッションに対して使用可能になっているロールを変更するために何度でも SET ROLE 文を使用できます。ただし、同時に使用可能にできるロールの数は、初期化パラメータ MAX_ENABLED_ROLES の値によって制限されます。

SESSION_ROLES データ・ディクショナリ・ビューを検索することにより、現在使用可能なロールを参照できます。

参照：

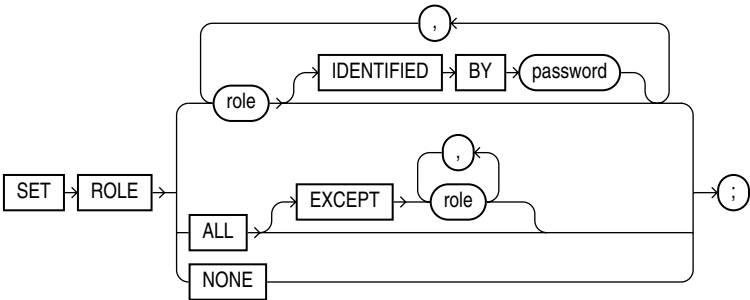
- ロールの作成の詳細は、14-73 ページの「[CREATE ROLE](#)」を参照してください。
- ユーザーのデフォルト・ロールの変更については、12-21 ページの「[ALTER USER](#)」を参照してください。
- SESSION_ROLES セッション・パラメータの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

前提条件

SET ROLE 文に指定するロールが付与されている必要があります。

構文

set_role::=



セマンティクス

role

現行のセッションで使用可能にするロールを指定します。リストされないロールおよび使用不可のロールは、現行のセッションで使用禁止になります。

IDENTIFIED BY *password* 句では、ロールに対するパスワードを指定します。ロールにパスワードが設定されている場合は、指定する必要があります。

ロールの設定の制限事項： ロールは、直接的または他のロールを介して付与されていないかぎり、指定できません。

ALL 句

現行のセッションに対して付与されているすべてのロールを使用可能にする場合に、ALL を指定します。ただし、EXCEPT 句に任意に指定されているロールは除きます。

EXCEPT 句に指定するロールは、ユーザーに直接付与されている必要があります。他のロールによってユーザーに付与されたものは無効です。

直接付与されているロール、および他のロールを介してユーザーに付与されているロールを EXCEPT 句に指定した場合、そのロールの付与先のロールにより、そのロールは使用可能のままになります。

ALL 句の制限事項： また、このオプションを使用して、ユーザーに直接付与されているパスワード付きのロールを使用可能にすることはできません。

NONE

現行のセッションで、DEFAULT ロールを含むすべてのロールを使用禁止にする場合に、NONE を指定します。

例

ロールの設定例： 次の文は、現行のセッションのパスワード `warehouse` によって識別されるロール `dw_manager` を使用可能にします。

```
SET ROLE dw_manager IDENTIFIED BY warehouse;
```

次の文は、現行のセッションで付与されているロールをすべて使用可能にします。

```
SET ROLE ALL;
```

次の文は、dw_manager を除く ロールをすべて使用可能にします。

```
SET ROLE ALL EXCEPT dw_manager;
```

次の文は、現行のセッションで付与されているすべてのロールを使用禁止にします。

```
SET ROLE NONE;
```


SET TRANSACTION

用途

SET TRANSACTION 文は、現行のトランザクションを読み専用または読み書き両用として設定する場合、分離レベルとして設定する場合または指定したロールバック・セグメントにトランザクションを割り当てる場合に使用します。

SET TRANSACTION 文によって実行される処理は、現行のトランザクションのみに影響します。他のユーザーまたは他のトランザクションには影響しません。COMMIT 文または ROLLBACK 文を発行すると、トランザクションは終了します。なお、現行のトランザクションは、データ定義文が実行される前後に暗黙的にコミットされます。

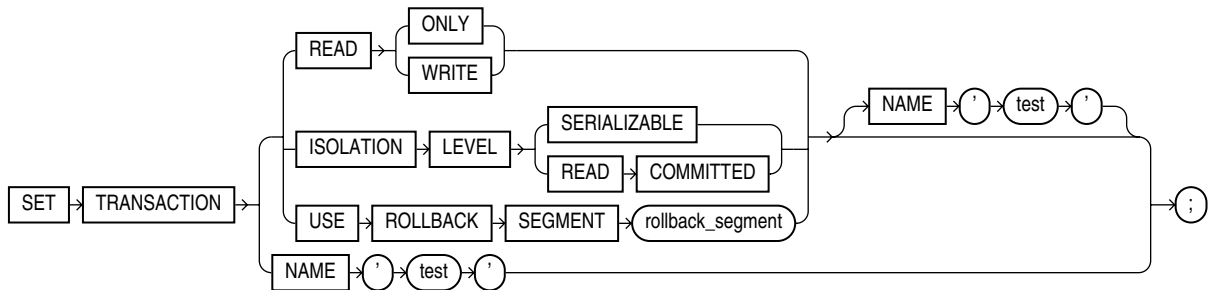
参照： 12-70 ページの「[COMMIT](#)」および 17-97 ページの「[ROLLBACK](#)」を参照してください。

前提条件

SET TRANSACTION 文を使用する場合、トランザクションの先頭に記述する必要があります。ただし、SET TRANSACTION 文が必要ないトランザクションもあります。

構文

set_transaction::=



セマンティクス

READ ONLY

READ ONLY 句は、現行のトランザクションを読み専用を設定します。この句では、トランザクション・レベルの読み一貫性を設定します。

そのトランザクションの後続のすべての問合せでは、そのトランザクション開始前にコミットされた変更のみが参照されます。読み専用トランザクションは、他のユーザーが更新中の 1 つ以上の表に対して、複数の問合せを実行するレポートに便利です。

注意： ユーザー SYS は、この句を使用できません。SYS による問合せでは、SYS がトランザクションを READ ONLY に設定した場合でも、トランザクション中の変更が戻されます。

読み専用トランザクションの制限事項： 読み専用トランザクションは、次の文のみを使用できます。

- 副問合せ (*for_update_clause* を指定しない SELECT 文)
- LOCK TABLE
- SET ROLE
- ALTER SESSION
- ALTER SYSTEM

参照：『Oracle9i データベース概要』を参照してください。

READ WRITE

現行のトランザクションを読み書き両用に設定する場合に、READ WRITE を指定します。この句では、**文レベルの読み一貫性**を設定します。これはデフォルトです。

読み書き両用トランザクションの制限事項： 同一トランザクション内では、読み一貫性のレベル（トランザクション・レベルおよび文レベル）を切り替えることができません。

ISOLATION LEVEL 句

データベースを変更するトランザクションがどのように処理されるかを指定する場合に、ISOLATION LEVEL 句を指定します。

- SQL92 規格に定義されているシリアル化可能トランザクション分離モードを設定する場合に、SERIALIZABLE を指定します。シリアル化可能トランザクションに、データ操作言語（DML）が含まれている場合、その DML が、シリアル化可能トランザクションの開始時にコミットされていないトランザクションですでに更新されているリソースを更新すると、その DML 文は正常に実行されません。

注意： SERIALIZABLE モードで運用する場合は、初期化パラメータ COMPATIBLE を 7.3.0 以上に設定してください。

- Oracle トランザクションでは、デフォルトで READ COMMITTED が設定されています。別のトランザクションで行ロックを保持しておく必要がある DML がトランザクションに指定されていると、DML 文は行ロックが解除されるまで待ち状態になります。

USE ROLLBACK SEGMENT 句

現行のトランザクションを、指定したロールバック・セグメントに割り当てる場合に、USE ROLLBACK SEGMENT を指定します。この句によって、現行のトランザクションは暗黙的に読み書き両用トランザクションに設定されます。

この句では、トランザクションのタイプごとに、異なるサイズのロールバック・セグメントを割り当てることができます。次に例を示します。

- 実行時間が長い複数の問合せが、同時に同じ表を読み込んでいない場合は、小さいトランザクションを、小さいロールバック・セグメントに割り当てることができます。ロールバック・セグメントが小さいと、メモリー内に保持される可能性が高くなります。
- 実行時間が長い複数の問合せによって同時に読み込まれる表を変更するトランザクションには、大きいロールバック・セグメントを割り当てることができます。これは、読みみ貫性問合せのために必要なロールバック情報が、上書きされないようにするためです。
- 大量のデータを挿入、更新または削除するトランザクションは、そのトランザクションのロールバック情報を保持するための十分な大きさを持つロールバック・セグメントに割り当てることができます。

1 つの SET TRANSACTION 文または同じトランザクション内の異なる文に、READ ONLY 句および USE ROLLBACK SEGMENT 句は指定できません。読み専用トランザクションはロールバック情報を生成しないため、ロールバック・セグメントは割り当てられません。

NAME 句

NAME 句を使用すると、現行のトランザクションに名前を割り当てることができます。この句は、分散データベース環境でインダウト・トランザクションを識別および変換する場合に便利です。text 文字列の最大長は 255 バイトです。

分散トランザクションに対して名前を指定する場合、トランザクションのコミット時に、名前はコミットのコメントとなり、COMMIT 文で明示的に指定した任意のコメントを上書きします。

例

トランザクションの設定例： 次の文は、サンプル・スキーマ Order Entry (oe) のトロントのウェアハウスにある在庫の製品と量を計算するもので、各月の最後の日の真夜中に起動されます。このレポートは、異なるウェアハウスに対して在庫を追加および削除する他のユーザーの影響は受けません。

```
COMMIT;
```

```
SET TRANSACTION READ ONLY NAME 'Toronto';
```

```
SELECT product_id, quantity_on_hand FROM inventories  
       WHERE warehouse_id = 5;
```

```
COMMIT;
```

最初の COMMIT によって、SET TRANSACTION がトランザクションの最初の文であることが保証されます。最後の COMMIT 文は、データベースに対する変更を保存するためではありません。単に、読み込み専用トランザクションを終了するためのものです。

次の文は、ユーザーの現行のトランザクションを、ロールバック・セグメント rs_one に割り当てます。

```
SET TRANSACTION USE ROLLBACK SEGMENT rs_one;
```

TRUNCATE

用途

TRUNCATE を使用すると、表またはクラスタからすべての行を削除できます。デフォルトでは、削除された行が使用していたすべての領域（ただし、記憶域パラメータ MINEXTENTS で指定された行は除きます）の割当てが解除され、記憶域パラメータ NEXT が、切捨て処理によってセグメントから最後に削除されたエクステンツのサイズに設定されます。

表を削除して再作成するより、TRUNCATE 文で行を削除する方が効果的です。表を削除して再作成した場合、その表に依存するオブジェクトが無効になり、表に対するオブジェクト権限を再度付与する必要があります。また、表の索引、整合性制約およびトリガーを再作成し、その記憶域パラメータを再指定する必要があります。表の切捨ての場合、このような影響はありません。

注意： TRUNCATE 文はロールバックできません。

参照：

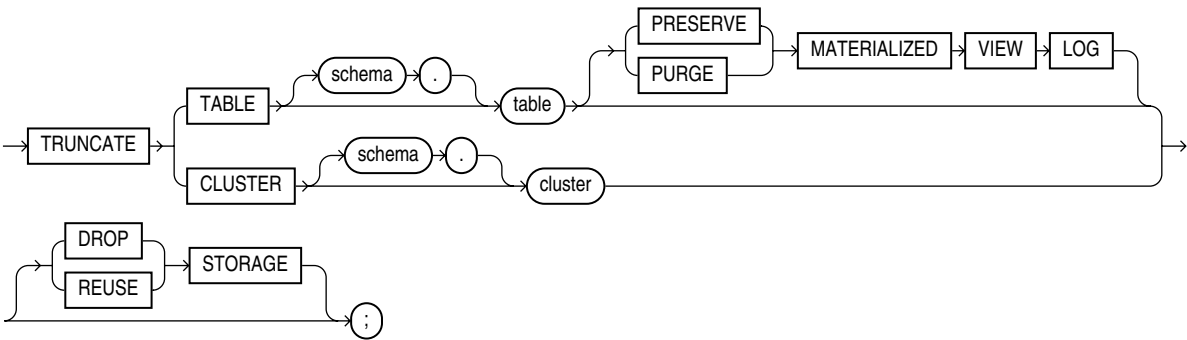
- データベースから表のデータを削除する他の方法については、16-53 ページの「[DELETE](#)」および 17-6 ページの「[DROP TABLE](#)」を参照してください。
- クラスタ化表の削除については、16-64 ページの「[DROP CLUSTER](#)」を参照してください。

前提条件

表またはクラスタを切り捨てるには、表またはクラスタが自分のスキーマ内にあるか、または DROP ANY TABLE システム権限が必要です。

構文

truncate::=



セマンティクス

TABLE 句

切り捨てる表が設定されているスキーマおよびその表の名前を指定します。クラスタを構成する表は、切り捨てることができません。 *schema* を指定しない場合、表が自分のクラスタに定義されているとみなされます。

- 索引構成表や一時表も切り捨てることができます。一時表を切り捨てた場合、現行のセッションで作成された行のみが削除されます。
- 表を切り捨てると、記憶域パラメータ NEXT が、切捨て処理中にセグメントから最後に削除されたエクステンツのサイズに変更されます。
- *table* に対する索引（ローカル索引のレンジ・パーティションとハッシュ・パーティション、およびローカル索引のサブパーティション）の UNUSABLE のインジケータも、自動的に切捨ておよびリセットされます。
- *table* が空でない場合は、表中の非パーティション索引およびグローバル・パーティション索引のすべてのパーティションに UNUSABLE のマークが付けられます。
- ドメイン索引の場合は、この文が、適切な TRUNCATE ルーチンを起動し、ドメイン索引のデータを切り捨てます。

参照： ドメイン索引の詳細は、『Oracle9i Data Cartridge Developer’s Guide』を参照してください。

- *table*（索引構成表および標準表）が LOB 列を含む場合、すべての LOB データおよび LOB 索引セグメントは切り捨てられます。

- `table` がパーティション化されている場合、各パーティションまたはサブパーティションの LOB データ・セグメントおよび LOB 索引セグメントと同様に、パーティションおよびサブパーティションも切り捨てられます。

注意： 表を切り捨てた場合、表中の索引データおよび表に対応付けされたマテリアライズド・ビュー・ダイレクト・パス・インサート情報もすべて、自動的に削除されます（この情報は、マテリアライズド・ビュー・ログのいずれにも依存していません）。このダイレクト・パス・インサート情報を削除した場合、マテリアライズド・ビューの増分リフレッシュのデータが失われる場合があります。

表の切捨での制限事項：

- クラスタを構成する表は、個別に切り捨てることはできません。これを行うには、クラスタを切り捨てるか、表のすべての行を削除するか、または表を削除して再作成する必要があります。
- 使用可能になっている参照整合性制約の親である表は、切り捨てることはできません。その表を切り捨てる場合、制約を無効にしておく必要があります（整合性制約が自己参照型の場合は、例外として表を切り捨てることができます）。
- `table` が階層に属する場合、階層のルートである必要があります。
- ドメイン索引が `table` で定義されている場合、IN_PROGRESS のマークが付いている索引および索引パーティションは切り捨てることはできません。

MATERIALIZED VIEW LOG 句

MATERIALIZED VIEW LOG 句では、表が切り捨てられた場合に、この表に定義されているマテリアライズド・ビュー・ログを保存するか、または削除するかを指定できます。この句を使用した場合、マテリアライズド・ビュー・マスター表を、エクスポート / インポートにより再編成できます。この場合、マスター表で定義された主キー・マテリアライズド・ビューを高速リフレッシュする機能は影響を受けません。主キー・マテリアライズド・ビューの連続高速リフレッシュをサポートする場合、マテリアライズド・ビュー・ログに主キー情報を記録する必要があります。

注意： キーワード SNAPSHOT は、MATERIALIZED VIEW のかわりに下位互換用にサポートされています。

PRESERVE マスター表を切り捨てたときにマテリアライズド・ビュー・ログを保存する場合は、PRESERVE を指定します。これはデフォルト値です。

PURGE マスター表を切り捨てたときにマテリアライズド・ビュー・ログを削除する場合は、PURGE を指定します。

参照： マテリアライズド・ビュー・ログおよび TRUNCATE 文の詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。

CLUSTER 句

切り捨てるクラスタが設定されているスキーマと、そのクラスタの名前を指定します。なお、索引クラスタは切り捨てられますが、ハッシュ・クラスタは切り捨てられません。*schema* を指定しない場合、クラスタが自分のスキーマ内に定義されているとみなされます。

クラスタを切り捨てた場合、そのクラスタにある表のすべての索引データも自動的に削除されます。

STORAGE 句

STORAGE 句を指定すると、行の切捨てによって空いた領域をどのようにするかを指定できます。DROP STORAGE 句および REUSE STORAGE 句は、対応する索引から削除されたデータの空き領域にも適用されます。

DROP STORAGE 表またはクラスタの MINEXTENTS パラメータで割り当てられた領域を除き、表またはクラスタから削除された行から、すべての領域の割当てを解除する場合に、DROP STORAGE を指定します。解放された領域は、表領域の他のオブジェクトに使用されます。記憶域パラメータ NEXT は、切捨て処理中にセグメントから最後に削除されたエクステントのサイズに設定されます。これはデフォルト値です。

REUSE STORAGE 表またはクラスタに割り当てられた削除行から領域を確保する場合に、REUSE STORAGE を指定します。STORAGE の値は、表またはクラスタを作成したときの値にリセットされません。この領域は、挿入操作または更新操作によってその表またはクラスタ内に作成される新規データによってのみ使用されます。記憶域パラメータは現行の設定のまま残ります。

注意： 切り捨てるオブジェクトに対して、2 つ以上の空きリストを指定している場合は、REUSE STORAGE 句によって、インスタンスへの空きリストのマッピングも削除され、最高水位標は第 1 エクステントの始まりにリセットされます。

例

表の切捨て例： 次の文は、サンプル表 `hr.employees` のコピーのすべての行を削除して、解放された領域を `employees` 表が定義されている表領域に戻します。

```
TRUNCATE TABLE employees;
```

ここでは、`employees` 表の索引データもすべて削除され、解放された領域は、それらの索引が定義されていた表領域に戻されます。

切捨て後の空き領域の保持例： 次の文は、`personnel` クラスタ内の表のすべての行を削除しますが、表に割り当てられている領域はそのままにしておきます。

```
TRUNCATE CLUSTER personnel REUSE STORAGE;
```

この文では、`personnel` クラスタにある表のすべての索引データも削除されます。

切捨て後のマテリアライズド・ビュー・ログの保存例： 次の文は、マテリアライズド・ビュー・ログを保存する `TRUNCATE` 文の使用例です。

```
TRUNCATE TABLE sales_demo PRESERVE MATERIALIZED VIEW LOG;
```

```
TRUNCATE TABLE orders_demo;
```

UPDATE

用途

UPDATE 文を使用すると、表またはビューの実表の既存の値を変更できます。

前提条件

表の値を更新する場合は、表が自分のスキーマ内にあるか、またはその表に対する UPDATE 権限が必要です。

ビューの実表の値を更新する場合は、次のことが必要です。

- そのビューに対する UPDATE 権限を持っている。
- そのビューが設定されているスキーマの所有者が、実表に対する UPDATE 権限を持っている。

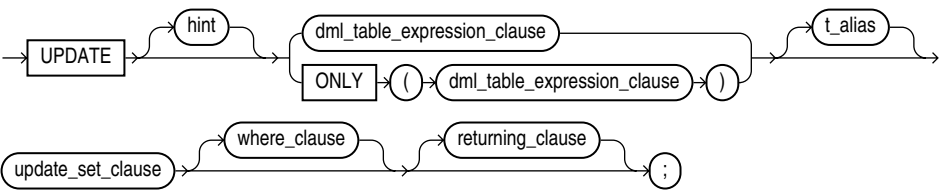
UPDATE ANY TABLE システム権限がある場合は、任意の表または任意の実表の値を更新できます。

次の場合は、更新するオブジェクトの SELECT 権限も必要です。

- オブジェクトがリモート・データベースに存在する。
- 初期化パラメータ SQL92_SECURITY が TRUE に設定され、UPDATE 操作が表列 (where_clause 内の列など) を参照する。

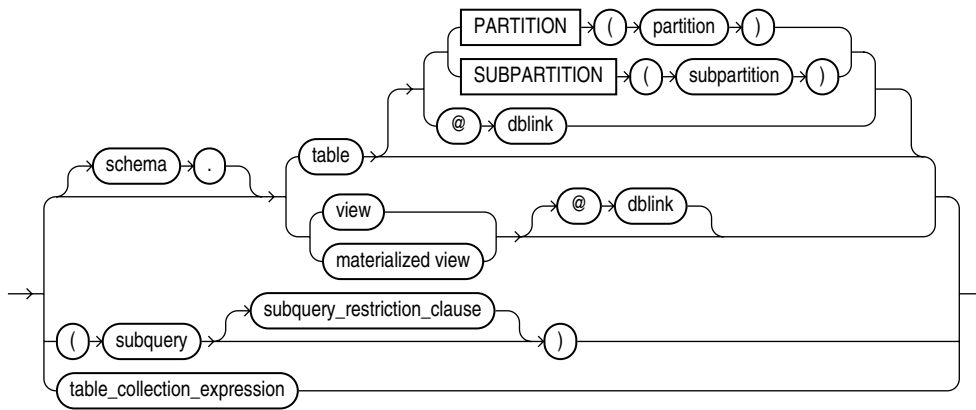
構文

update::=



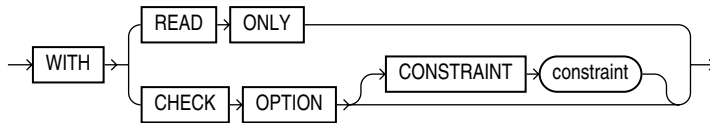
(18-57 ページの「[DML_table_expression_clause::=](#)」、18-58 ページの「[update_set_clause::=](#)」、18-58 ページの「[where_clause::=](#)」、18-58 ページの「[returning_clause::=](#)」を参照)

DML_table_expression::=

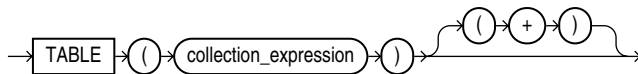


(18-5 ページの「[subquery::=](#)」(SELECT 構文の一部)、18-57 ページの「[subquery_restriction_clause::=](#)」および 18-57 ページの「[table_collection_expression::=](#)」を参照)

subquery_restriction_clause::=



table_collection_expression::=



DML_table_expression_clause

ONLY 句は、ビューのみに適用されます。UPDATE 句のビューが階層に属し、任意のサブビューから行を変更しない場合は、ONLY 構文を指定します。

参照： 18-61 ページの「[dml_table_expression_clause の制限事項](#)」および 18-65 ページの「[表の更新例](#)」を参照してください。

schema

表またはビューが含まれているスキーマを指定します。*schema* を指定しない場合、表またはビューが自分のスキーマにあるとみなされます。

table | view | subquery

表、ビュー、マテリアライズド・ビュー、または副問合せから戻された列の名前を指定します。指定した値は、更新する必要があります。表に対して UPDATE 文を実行した場合、その表に対応付けられた UPDATE トリガーが起動します。

view を指定した場合、ビューの実表が更新されます。ビューを定義する問合せに次のいずれかの要素が含まれる場合は、INSTEAD OF トリガーを除き、そのビューを更新することはできません。

- 集合演算子
- DISTINCT 演算子
- 集計ファンクションまたは分析ファンクション
- GROUP BY、ORDER BY、CONNECT BY または START WITH 句
- SELECT 構文のリストにあるコレクション式
- SELECT 構文のリストにある副問合せ
- 結合（一部の例外を除く）

詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

また、WITH CHECK OPTION を指定してビューを作成した場合、実行結果がビューを定義する問合せの条件を満たす場合にのみ、ビューを更新できます。

table（または *view* の実表）に、1 列以上のドメイン索引列がある場合は、この文が適切な索引タイプの更新ルーチンを実行します。

マテリアライズド・ビューを指定し、そのマテリアライズド・ビューで FOR UPDATE 句を使用している場合、Oracle は、マテリアライズド・ビューのデータを更新します。

参照：

- 索引タイプの更新ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- 更新可能なマテリアライズド・ビューについては、14-4 ページの「[CREATE MATERIALIZED VIEW](#)」を参照してください。

PARTITION (*partition*) | SUBPARTITION (*subpartition*)

更新対象の表内にあるパーティションまたはサブパーティションの名前を指定します。パーティション表内の値を更新する場合は、パーティション名を指定する必要はありません。ただし、パーティション名を指定した方が、複雑な *where_clause* を使用するよりも効果的な場合もあります。

参照： 18-66 ページの「[パーティションの更新例 :](#)」を参照してください。

dblink

表またはビューが格納されているリモート・データベースへのデータベース・リンクの完全名または部分名を指定します。Oracle の分散機能を使用している場合にかぎり、データベース・リンクを使用して、リモート表またはリモート・ビューを更新できます。

dblink を指定しない場合、Oracle は、その表またはビューがローカル・データベース内にあるとみなします。

参照： データベース・リンクの参照方法の詳細は、2-112 ページの「[リモート・データベース内のオブジェクトの参照](#)」を参照してください。

subquery_restriction_clause

subquery_restriction_clause を使用すると、次のいずれかの方法で副問合せを制限できます。

WITH READ ONLY 表またはビューが更新禁止であることを示す場合は、WITH READ ONLY を指定します。

WITH CHECK OPTION 副問合せに存在しない行を生成する表またはビューの変更が禁止であることを示す場合は、WITH CHECK OPTION を指定します。

CONSTRAINT *constraint* CHECK OPTION 制約の名前を指定します。この識別子を省略した場合、その制約に SYS_Cn という形式の名前が自動的に割り当てられます。この場合の *n* は、その制約名をデータベース内で一意の名前にする整数です。

参照： 18-32 ページの「[WITH CHECK OPTION 句の使用例 :](#)」を参照してください。

table_collection_expression

問合せおよび DML 操作で、`collection_expression` 値を表として扱う場合に、`table_collection_expression` を指定します。`collection_expression` は、副問合せ、列、ファンクションまたはコレクション・コンストラクタのいずれかです。その形式にかかわらず、集合値（ネストした表型または `VARRAY` 型の値）を戻す必要があります。このようなコレクションの要素抽出プロセスを **コレクション・ネスト解除** といいます。

注意： 以前のリリースの Oracle では、`collection_expression` が副問合せの場合、`table_collection_expression` を「`THE subquery`」と表現していました。現在、このような表現方法はされていません。

`table_collection_expression` を使用して、ある表の行を別の表の行を基にして更新できます。たとえば、四半期ごとの売上表を、年度ごとの売上表にまとめることができます。

t_alias

文内で参照する表、ビューまたは副問合せの**関連名**（別名）を指定します。

注意： `dml_table_expression_clause` がオブジェクト型の属性またはオブジェクト型のメソッドを参照する場合、この別名が必要です。

参照： 18-66 ページの「[関連更新の例:](#)」を参照してください。

dml_table_expression_clause の制限事項：

- `table`（または `view` の実表）に、`IN_PROGRESS` または `FAILED` とマークされたドメイン索引がある場合は、この文は実行できません。
- 関係する索引パーティションが `UNUSABLE` とマークされている場合は、パーティションに挿入できません。
- `dml_table_expression_clause` の副問合せには `order_by_clause` を指定できません。
- `UNUSABLE` のマークが付いている索引、索引パーティションまたは索引サブパーティションを指定する場合、`SKIP_UNUSABLE_INDEXES` セッション・パラメータが `TRUE` に設定されていないかぎり、`UPDATE` 文は正常に実行されません。

参照： `SKIP_UNUSABLE_INDEXES` セッション・パラメータの詳細は、10-2 ページの「[ALTER SESSION](#)」を参照してください。

update_set_clause

update_set_clause によって、列の値を設定します。

column

更新する表またはビューの列の名前を指定します。*update_set_clause* に表の列を指定しない場合、その列の値は変更されません。

column が LOB オブジェクト属性を参照している場合、まず空または NULL の値で初期化する必要があります。リテラルで更新はできません。また、UPDATE 以外の SQL 文を使用して LOB 値を更新する場合は、LOB を含む最初の行をロックしておく必要があります。

参照： 18-25 ページの「[for_update_clause](#)」を参照してください。

column がパーティション表のパーティション化キーに含まれる場合、別のパーティションまたはサブパーティションに行を移動する列の値を変更すると、行の移動を可能にしないかぎり、UPDATE は正常に実行されません。

参照： 詳細は、15-7 ページの「[CREATE TABLE](#)」の「[row_movement_clause](#)」または 11-2 ページの「[ALTER TABLE](#)」を参照してください。

また、*column* がリスト・パーティション表のパーティション化キーの一部である場合、パーティションの *partition_value* リストに存在していない列の値を指定すると、UPDATE は正常に実行されません。

subquery

更新される行ごとに 1 行ずつ戻す副問合せを指定します。

- *update_set_clause* で 1 列のみを指定した場合、副問合せは 1 つの値のみを戻します。
- *update_set_clause* で複数の列を指定した場合、副問合せは指定した列の数の値を戻します。

副問合せ内で *flashback_clause* を使用すると、過去のデータで *table* を更新できます。

参照： この句の詳細は、18-14 ページの「[SELECT](#)」の「[flashback_clause](#)」を参照してください。

副問合せが行を戻さなかった場合は、列には NULL が割り当てられます。

注意： *subquery* がリモート・オブジェクトを参照する場合、参照がローカル・データベースのオブジェクトにループバックしないかぎり、UPDATE はパラレルで実行されます。ただし、*dml_table_expression_clause* の *subquery* がリモート・オブジェクトを参照する場合は、UPDATE はシリアルで実行されます。

参照：

- 18-4 ページの [SELECT](#) および 8-12 ページの「[副問合せの使用法](#)」を参照してください。
- 15-50 ページの「[CREATE TABLE](#)」の「[parallel_clause](#)」を参照してください。

expr

対応する列に割り当てられた値を変換する式を指定します。

参照： *expr* の構文については、[第 4 章「式」](#)を参照してください。また、18-66 ページの「[オブジェクト表の更新例:](#)」も参照してください。

DEFAULT DEFAULT を指定すると、以前に列のデフォルト値として指定した値を列に設定できます。対応する列に対してデフォルト値を指定していない場合、列に NULL が設定されます。

デフォルト値の設定の制限事項： ビューを更新する場合は、DEFAULT を指定できません。

VALUE 句

VALUE 句によって、オブジェクト表の行全体を指定できます。

VALUE 句の制限事項： この句は、オブジェクト表に対してのみ指定できます。

注意： RAW 列に文字リテラルを挿入する場合、後続の問合せ中に RAW 列にある索引は使用されずに、全表スキャンが行われます。

参照： 18-66 ページの「[オブジェクト表の更新例:](#)」を参照してください。

where_clause

where_clause によって、指定した条件が真の行のみが更新されるように制限できます。この句を指定しない場合、表またはビューのすべての行が更新されます。

where_clause は、値を更新する行を決定します。*where_clause* を指定しない場合、すべての行が更新されます。*where_clause* の条件を満たす各行ごとに、*update_set_clause* の等号 (=) の左側にある列に、対応する右側の式の値が設定されます。式は行が更新される場合に評価されます。

参照： 条件の構文については、第 5 章「条件」を参照してください。

returning_clause

DML (INSERT、UPDATE または DELETE) 文に影響される行を取り出します。この句は、表、マテリアライズド・ビュー、および単一の実表を持つビューに指定できます。

単一行で処理する場合、*returning_clause* 付きの DML 文で、処理された行および ROWID を使用している列式、処理された行への REF を取り出し、ホスト変数または PL/SQL 変数に格納します。

複数行で処理する場合、*returning_clause* 付きの DML 文で、式の値、ROWID および処理された行に関連する REF をバインド配列に格納します。

expr *expr* リストの各項目は、適切な構文で表す必要があります。スカラー副問合せ式を除くすべての形式が有効です。

INTO 変更された行の値を、*data_item* リストに指定する変数に格納することを示します。

data_item 取り出された *expr* 値を格納するホスト変数または PL/SQL 変数です。

RETURNING リストの各式については、INTO リストに、対応する型に互換がある PL/SQL 変数またはホスト変数を指定する必要があります。

RETURNING 句の制限事項： 次の操作は実行できません。

- マルチテーブル・インサートへの *returning_clause* の指定。
- パラレル DML またはリモート・オブジェクトでのこの句の使用。
- この句での LONG 型の取出し。
- INSTEAD OF トリガーが定義されたビューに対しての、この句の指定。

参照： BULK COLLECT 句を使用してコレクション変数に複数の値を戻す場合は、『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。

例

表の更新例： 次の文は、職種が SH_CLERK のすべての従業員の歩合に NULL 値を指定します。

```
UPDATE employees
  SET commission_pct = NULL
  WHERE job_id = 'SH_CLERK';
```

次の文は、Douglas Grant を部門 20 の管理者に昇格させ、給与を 1,000 ドル引き上げます。

```
UPDATE employees SET
  job_id = 'SA_MAN', salary = salary + 1000, department_id = 20
  WHERE first_name||' '||last_name = 'Douglas Grant';
```

次の文は、remote データベースの employees 表の従業員の給与を増加します。

```
UPDATE employees@remote
  SET salary = salary*1.1
  WHERE last_name = 'Baer';
```

次の例は、UPDATE 文の次の構文要素を示します。

- 単一文にまとめた *update_set_clause* の 2 つの形式
- 関連副問合せ
- 更新された行を制限する *where_clause*

```
UPDATE employees a
  SET department_id =
    (SELECT department_id
     FROM departments
      WHERE location_id = '2100'),
    (salary, commission_pct) =
    (SELECT 1.1*AVG(salary), 1.5*AVG(commission_pct)
     FROM employees b
      WHERE a.department_id = b.department_id)
  WHERE department_id IN
    (SELECT department_id
     FROM departments
      WHERE location_id = 2900
       OR location_id = 2700);
```

この UPDATE 文によって、次の処理が実行されます。

- ジュネーブまたはミュンヘン (location_id は 2900 または 2700) で働く従業員のみを更新します。
- これらの従業員の department_id をボンベイ (location_id は 2100) の対応する department_id に設定します。
- 各従業員の給与を、その部門の平均給与の 10% 引き上げます。
- 各従業員の歩合を、その部門の平均歩合の 50% 引き上げます。

パーティションの更新例： 次の文は、sales 表の 1 つのパーティションの値を更新します。

```
UPDATE sales PARTITION (sales_q1_1999) s
  SET s.promo_id = 494
  WHERE amount_sold > 9000;
```

オブジェクト表の更新例： 次の文は、異なるオブジェクト表 table2 の行を選択して、オブジェクト表 table1 の行を更新します。

```
UPDATE table1 p SET VALUE(p) =
  (SELECT VALUE(q) FROM table2 q WHERE p.id = q.id)
  WHERE p.id = 10;
```

この例では、SET 句と副問合せの両方で、VALUE オブジェクト参照関数を使用します。

関連更新の例： 次の文は、部門番号 123 の部門に関連するネストした表 projs の特定の行を更新します。

```
UPDATE TABLE(SELECT projs
  FROM dept d WHERE d.dno = 123) p
  SET p.budgets = p.budgets + 1
  WHERE p.pno IN (123, 456);
```

UPDATE 操作中に RETURNING 句を使用する例： 次の文は、更新された行の値を戻し、PL/SQL 変数 bnd1、bnd2、bnd3 に結果を格納します。

```
UPDATE employees
  SET job_id = 'SA_MAN', salary = salary + 1000, department_id = 140
  WHERE last_name = 'Jones'
  RETURNING salary*0.25, last_name, department_id
  INTO :bnd1, :bnd2, :bnd3;
```

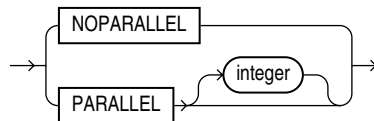
構文図の読み方

構文図とは、SQL の有効な構文を図で示したものです。構文図は、矢印が示す方向に左から右へ読んでください。

コマンドおよびキーワードは、四角形の中に大文字で書かれています。コマンドおよびキーワードは、四角形の中に書かれているとおりに指定してください。パラメータは、楕円形の中に小文字で書かれています。パラメータには変数を指定します。句読点、デリミタ、終了記号は円の中に書かれています。

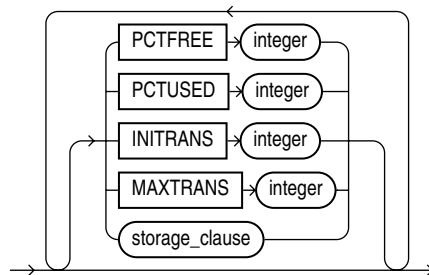
構文図の中にパスが複数ある場合は、ユーザーがとるパスを選択できます。次の例では、NOPARALLEL または PARALLEL のいずれかを指定できます。

parallel_clause::=



キーワード、演算子、パラメータに複数の選択肢がある場合は、選択できるオプションが縦に並べて書かれています。次の構文図では、スタックにある 5 つのパラメータから 1 つ以上を指定することができます。

physical_attributes_clause::=



構文図で使用するパラメータと、各文でそのパラメータに代入する値の例を次の表に示します。

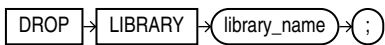
パラメータ	説明	例
<i>table</i>	パラメータによって指定された型のオブジェクト名で置き換える必要があります。オブジェクト型の一覧は、2-101 ページの「 スキーマ・オブジェクト 」を参照してください。	employees
<i>c</i>	ご使用のデータベース・キャラクタ・セットの単一文字で置き換える必要があります。	T s
' <i>text</i> '	一重引用符で囲んだテキスト文字列で置き換える必要があります。' <i>text</i> ' の構文は、2-51 ページの「 テキスト・リテラル 」を参照してください。	'Employee records'
<i>char</i>	CHAR または VARCHAR2 データ型の式か、一重引用符で囲んだ文字リテラルで置き換える必要があります。	last_name 'Smith'
<i>condition</i>	TRUE または FALSE に評価される条件で置き換える必要があります。 <i>condition</i> の構文は、 第 5 章「条件」 を参照してください。	last_name > 'A'
<i>date</i> <i>d</i>	日付定数または DATE データ型の式で置き換える必要があります。	TO_DATE('01-Jan-2002', 'DD-MON-YYYY')
<i>expr</i>	4-2 ページの「 SQL 式 」にある <i>expr</i> 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。	salary + 1000
<i>integer</i>	2-52 ページの「 整数リテラル 」にある <i>integer</i> の構文の説明で定義されている整数で置き換える必要があります。	72
<i>number</i> <i>m</i> <i>n</i>	NUMBER データ型の式、または 2-53 ページの「 数値リテラル 」にある <i>number</i> の構文の説明で定義されている数値定数で置き換える必要があります。	AVG(salary) 15 * 7
<i>raw</i>	RAW データ型の式で置き換える必要があります。	HEXTORAW('7D')
<i>subquery</i>	SELECT 文で置き換える必要があります。この SELECT 文は、別の SQL 文中で使用されます。18-4 ページの「 SELECT 」を参照してください。	SELECT last_name FROM employees

パラメータ	説明	例
<code>db_name</code>	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名で置き換える必要があります。	<code>sales_db</code>
<code>db_string</code>	Oracle Net 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る必要があります。詳細は、ご使用の Oracle Net プロトコルのユーザーズ・ガイドを参照してください。	—

必須キーワードとパラメータ

必須キーワードおよびパラメータは、単独で示されているか、または選択肢のリストとして縦に並べて書かれています。必須キーワードおよびパラメータが 1 つの場合は、メイン・パス、つまり現在選択しているパスの線の上に書かれています。次の例では、`library_name` が必須パラメータです。

drop_library::=

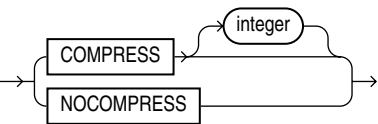


HQ_LIB という名前のライブラリがあるとすると、この図は、次の文を示しています。

```
DROP LIBRARY hq_lib;
```

メイン・パスに交わる縦に並んだリストの中に、複数のキーワードまたはパラメータがある場合、そのうちの 1 つが必須です。つまり、複数のキーワードまたはパラメータから、いずれか 1 つを選択する必要があります。ただし、メイン・パスに存在しなくてもかまいません。次の例では、いずれかの設定値を選択する必要があります。

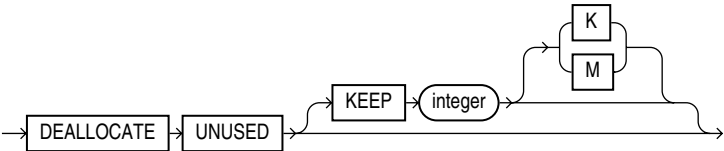
key_compression::=



オプションのキーワードとパラメータ

キーワードおよびパラメータが、メイン・パスより上に縦に並べてリストされている場合は、それらのキーワードおよびパラメータがオプションであることを示しています。次の例では、上のパスへ進んでも、続けてメイン・パスに沿って進んでもかまいません。

deallocate_unused_clause::=



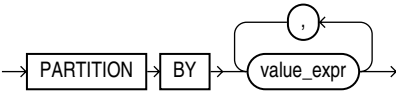
この図に従うと、次の文はすべて有効です。

```
DEALLOCATE UNUSED;  
DEALLOCATE UNUSED KEEP 1000;  
DEALLOCATE UNUSED KEEP 10M;
```

構文のループ

ループがある場合、ループ内の構文を何度でも繰り返して実行できます。次の例では、値の式を 1 つ選択した後、繰り返し別の式を選択できます。式と式の間はカンマで区切ります。

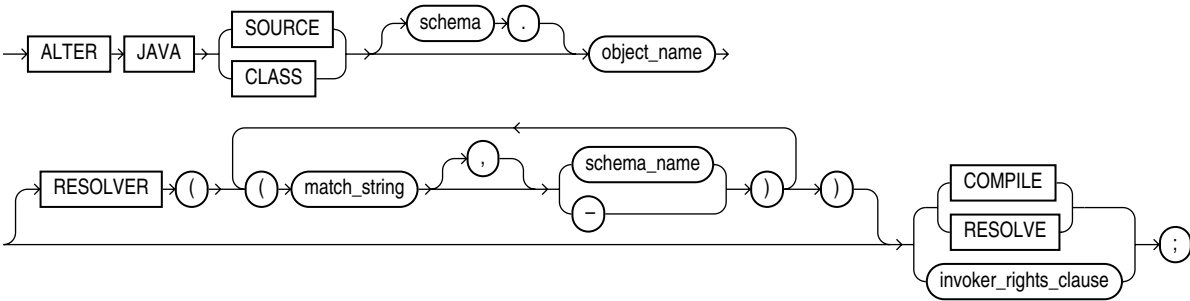
query_partition_clause::=



複数の部分に分割された構文図

複数の部分に分割された構文図は、すべてのメイン・パスの端と端が結合されているものとしてみてください。次の例は、2 つに分割された構文図です。

alter_java::=



この図に従うと、次の文は有効です。

```
ALTER JAVA SOURCE jsource_1 COMPILE;
```


データベース・オブジェクト

表や列などの Oracle 識別子名は、30 文字以内に制限されています。先頭文字は英字である必要がありますが、それ以外は、英字、数字、ドル記号 (\$)、シャープ記号 (#) およびアンダースコア () を任意に組み合わせて指定できます。

ただし、Oracle の識別子を二重引用符 (") で囲むと、すべての有効な文字 (空白を含む) を組み合わせて指定できます。

Oracle の識別子は、二重引用符で囲んだとき以外は、大文字、小文字を区別しません。

詳細は、2-105 ページの「[スキーマ・オブジェクトのネーミング規則](#)」を参照してください。

Oracle と標準 SQL

この付録では、SQL:1999 規格への Oracle の規格準拠について説明します。SQL:1999 規格の必須部分は、Core SQL:1999 として知られ、SQL:1999 の Part 2「Foundation」および Part 5「Bindings」に記載されています。基礎的な機能は、Part 2 の Annex F の「SQL/Foundation feature taxonomy and definition for Core SQL」表で分析されています。バインディング機能は、Part 5 の Annex F の「SQL/Bindings feature taxonomy and definition for Core SQL」表で分析されています。

この付録では、ANSI（米国規格協会）および ISO（国際標準化機構）によって確立された SQL 規格への Oracle の規格準拠について説明します（ANSI および ISO の SQL 規格は同一です）。この付録では、次の内容を説明します。

- ANSI 規格
- ISO 規格
- Oracle の規格準拠
- FIPS の規格準拠
- 標準 SQL に対する Oracle 拡張機能
- キャラクタ・セットのサポート

ANSI 規格

SQL に関連する ANSI のドキュメントは、次のとおりです。

- ANSI/ISO/IEC 9075-1:1999, Information technology—Database languages—SQL—Part 1: Framework (SQL/Framework)
 - ANSI/ISO/IEC 9075-1:1999/Amd.1:2000
- ANSI/ISO/IEC 9075-2:1999, Information technology—Database languages—SQL—Part 2: Foundation (SQL/Foundation)
- ANSI/ISO/IEC 9075-5:1999, Information technology—Database languages—SQL—Part 5: Host Language Bindings (SQL/Bindings)

ANSI 規格のコピーについては、次の宛先に申し込んでください。

American National Standards Institute
11 West 42nd Street
New York, NY 10036 USA
電話番号 : +1.212.642.4900
FAX 番号 : +1.212.398.0023

Web サイトからも入手できます。URL は、次のとおりです。

<http://webstore.ansi.org/ansidocstore/default.asp>

SQL 規格を含む ANSI 規格のサブセットは、X3 または NCITS 規格です。これらは、NCITS (National Committee for Information Technology Standards) から入手できます。URL は、次のとおりです。

<http://www.cssinfo.com/ncitsgate.html>

ISO 規格

SQL に関連する ISO のドキュメントは、次のとおりです。

- ISO/IEC 9075-1:1999, Information technology—Database languages—SQL—Part 1: Framework (SQL/Framework)
 - ISO/IEC 9075-1:1999/Amd.1:2000
- ISO/IEC 9075-2:1999, Information technology—Database languages—SQL—Part 2: Foundation (SQL/Foundation)
- ISO/IEC 9075-5:1999, Information technology—Database languages—SQL—Part 5: Host Language Bindings (SQL/Bindings)

ISO 規格のコピーについては、次の宛先に申し込んでください。

International Organization for Standardization
1 Rue de Varembo
Case postale 56
CH-1211, Geneva 20, Switzerland
電話番号 : +41.22.749.0111
FAX 番号 : +41.22.733.3430
URL: <http://www.iso.ch/>

Web ストアからも入手できます。URL は、次のとおりです。

<http://www.iso.ch/cate/cat.html>

Oracle の規格準拠

ANSI および ISO の SQL 規格では、規格準拠性について明示する箇所に、準拠の種類および実装されている機能について記載することを義務付けています。次の製品は、表 B-1 に示す ANSI 規格および ISO 規格に完全（または部分的）に準拠しています。

- Oracle9i データベース・サーバー
- Pro*C/C++ リリース 9.2.0
- Pro*COBOL リリース 9.2.0
- Pro*FORTRAN リリース 1.8.77
- SQL Module for Ada (Mod*Ada) リリース 9.2.0
- Pro*COBOL 1.8 リリース 1.8.77
- Pro*PL/I リリース 1.6.28
- OTT リリース 9.2.0
- OTT8 リリース 8.1.8

表 B-1 に、Oracle が完全にサポートする Core SQL:1999 機能を示します。

表 B-1 完全にサポートする Core SQL:1999 機能

機能識別番号	機能
E011	数値データ型
E031	識別子
E061	基本的な述語および検索条件

表 B-1 完全にサポートする Core SQL:1999 機能（続き）

機能識別番号	機能
E081	基本的な権限
E091	集合ファンクション
E101	基本的なデータ操作
E111	単一行の SELECT 文
E131	NULL 値のサポート（値のかわりの NULL）
E141	基本的な整合性制約
E151	トランザクションのサポート
E152	基本的な SET TRANSACTION 文
E153	副問合せを持つ更新可能な問合せ
E161	先頭に 2 つの負の符号 (-) を付けた SQL 文のコメント
E171	SQLSTATE のサポート
F041	基本的な結合表
F051	基本的な日付および時刻
F081	ビューの UNION および EXCEPT
F131	グループ操作
F181	複数モジュールのサポート
F201	CAST ファンクション
F221	明示的なデフォルト
F261	CASE 式
F311	スキーマ定義文
F471	スカラー副問合せの値
F481	拡張 NULL 述語
B011	埋込み ADA
B012	埋込み C
B013	埋込み COBOL
B014	埋込み Fortran

表 B-1 完全にサポートする Core SQL:1999 機能 (続き)

機能識別番号	機能
T431	拡張グループ化機能
T611	基本的な OLAP 演算子
T621	拡張数値ファンクション

表 B-2 に、Oracle が部分的にサポートする Core SQL:1999 機能を示します。

表 B-2 部分的にサポートする Core SQL:1999 機能

機能識別子、機能	部分的なサポート
E021、文字データ型	次の副機能を完全にサポートします。 ■ E021-01、CHARACTER データ型 ■ E021-07、文字の連結 ■ E021-08、UPPER および LOWER ファンクション ■ E021-09、TRIM ファンクション ■ E021-10、文字データ型間の暗黙的な加算 ■ E021-11、文字の比較 次の副機能を部分的にサポートします。 ■ E021-02、CHARACTER VARYING データ型 (Oracle は、長さ 0 の VARCHAR 列と NULL を区別しません) ■ E021-03、文字リテラル (Oracle は、長さ 0 のリテラル '' を NULL とみなします) Oracle は、次の副機能と同等の機能を持ちます。 ■ E021-04、CHARACTER_LENGTH ファンクションのかわりに、LENGTH ファンクションを使用します。 ■ E021-05、OCTET_LENGTH ファンクションのかわりに、LENGTHB ファンクションを使用します。 ■ E021-06、SUBSTRING ファンクションのかわりに、SUBSTR ファンクションを使用します。 ■ E021-11、POSITION ファンクションのかわりに、INSTR ファンクションを使用します。

表 B-2 部分的にサポートする Core SQL:1999 機能（続き）

機能識別子、機能	部分的なサポート
F031、基本的なスキーマ操作	次の副機能を完全にサポートします。 ■ F031-01、実表を作成するための CREATE TABLE 文 ■ F031-02、CREATE VIEW 文 ■ F031-03、GRANT 文 次の副機能を部分的にサポートします。 ■ F031-04、ALTER TABLE 文の ADD COLUMN 句（Oracle は、この構文のオプションのキーワード COLUMN をサポートしません） 次の副機能はサポートしません（Oracle は、キーワード RESTRICT をサポートしません）。 ■ F031-13、DROP TABLE 文の RESTRICT 句 ■ F031-16、DROP VIEW 文の RESTRICT 句 ■ F031-19、REVOKE 文の RESTRICT 句
E051、基本的な問合せ指定	次の副機能を完全にサポートします。 ■ E051-01、SELECT DISTINCT ■ E051-02、GROUP BY 句 ■ E051-04、<SELECT 構文のリスト> にない列を GROUP BY に含めることができます。 ■ E051-05、SELECT 構文のリスト項目の名前の変更 ■ E051-06、HAVING 句 ■ E051-07、SELECT 構文のリストでの * の修飾 次の副機能を部分的にサポートします。 ■ E051-08、FROM 句の相関名（Oracle は、相関名をサポートしますが、オプションの AS キーワードはサポートしません） 次の副機能はサポートされません。 ■ E051-09、FROM 句での列の名前の変更

表 B-2 部分的にサポートする Core SQL:1999 機能（続き）

機能識別子、機能	部分的なサポート
E071、基本的な問合せ式	次の副機能を完全にサポートします。 ■ E071-01、UNION DISTINCT 表演算子 ■ E071-02、UNION ALL 表演算子 ■ E071-05、表演算子によって結合された列の型は同じでなくてもかまいません。 ■ E071-06、副問合せの表演算子 Oracle は、次の副機能と同等の機能を持ちます。 ■ E071-03、EXCEPT DISTINCT 表演算子 EXCEPT DISTINCT のかわりに MINUS を使用します。
E121、基本的なカーソルのサポート	次の副機能を完全にサポートします。 ■ E121-01、DECLARE CURSOR ■ E121-02、ORDER BY 列は、SELECT 構文のリストになくてもかまいません。 ■ E121-03、ORDER BY 句の値の式 ■ E121-04、OPEN 文 ■ E121-06、位置付けされた UPDATE 文 ■ E121-07、位置付けされた DELETE 文 ■ E121-08、CLOSE 文 ■ E121-10、FETCH 文および暗黙的な NEXT 次の副機能はサポートされません。 ■ E121-17、WITH HOLD カーソル（規格では、カーソルは ROLLBACK 中は保持されませんが、Oracle では ROLLBACK 中も保持されます。）
F812、基本的なフラグ付け	Oracle は、フラガーを持っていますが、SQL:1999 よりも SQL-92 に準拠しています。

表 B-2 部分的にサポートする Core SQL:1999 機能（続き）

機能識別子、機能	部分的なサポート
T321、基本的な SQL 起動ルーチン	次の副機能を完全にサポートします。 ■ T321-03、ファンクションの起動 ■ T321-04、CALL 文 Oracle は、次の副機能を異なる構文でサポートします。 ■ T321-01、オーバーロードしないユーザー定義ファンクション ■ T321-02、オーバーロードしないユーザー定義プロシージャ Oracle の構文 CREATE FUNCTION および CREATE PROCEDURE と規格の構文の違いは、次のとおりです。 ■ 規格の構文では、パラメータのモード（IN、OUT または INOUT）はパラメータ名の前ですが、Oracle 構文では、パラメータ名の後です。 ■ 規格の構文では、INOUT を使用しますが、Oracle の構文では、IN OUT を使用します。 ■ Oracle の構文では、復帰型の後およびルーチン本体の定義の前に IS または AS が必要ですが、規格の構文では、必要ありません。 ■ ルーチンの本体が C である場合、そのルーチンに名前を付けるとき、規格の構文では、キーワード LANGUAGE C EXTERNAL NAME を使用しますが、Oracle の構文では、LANGUAGE C NAME を使用します。 ■ ルーチンの本体が SQL である場合、Oracle では、プロシージャを独自に拡張した PL/SQL を使用します。 Oracle は、次の副機能を PL/SQL ではサポートしますが、Oracle SQL ではサポートしません。 ■ T321-05、RETURN 文

Oracle には、表 B-3 で示す機能と同等の機能があります。

表 B-3 Core SQL:1999 機能と同等の機能

機能識別子、機能	同等の機能
F021、基本的な情報スキーマ	Oracle は、この機能のビューを持ちませんが、同じ情報を別のメタデータ・ビューで使用可能にします。 ■ TABLES のかわりに、ALL_TABLES を使用します。 ■ COLUMNS のかわりに、ALL_TAB_COLUMNS を使用します。 ■ VIEWS のかわりに、ALL_VIEWS を使用します。 ただし、ユーザーのビューに WITH CHECK OPTION が定義されていたり、更新可能な場合は、Oracle の ALL_VIEWS は表示されません。ビューに WITH CHECK OPTION が定義されているかどうかを確認する場合は、ALL_CONSTRAINTS を使用し、TABLE_NAME をそのビュー名にして、CONSTRAINT_TYPE が 'V' であるビューを検索します。 ■ TABLE_CONSTRAINTS、REFERENTIAL_CONSTRAINTS および CHECK_CONSTRAINTS のかわりに、ALL_CONSTRAINTS を使用します。 ただし、Oracle の ALL_CONSTRAINTS は、制約が遅延可能か初期遅延かは表示しません。

表 B-4 に、Oracle がサポートしない Core SQL:1999 機能を示します。

表 B-4 サポートしない Core SQL:1999 機能

機能識別番号	機能
F501	機能および準拠するビュー
S011	DISTINCT データ型

注意： Oracle は、「E182、モジュール言語」をサポートしません。この機能は、規格の表 31 に示されていますが、Core にはモジュール言語と埋込み言語の選択肢があることが示されているのみです。モジュール言語および埋込み言語は、機能は同じで、SQL 文がホスト・プログラミング言語と関連付けられている点のみが異なります。Oracle は、埋込み言語をサポートします。

FIPS の規格準拠

Oracle は、最新の FIPS（Federal Information Processing Standard）である FIPS PUB 127-2 に完全に準拠しています。現在、この規格は、公開されていません。ただし、FIPS PUB 127-2 で定義されたデータベース要素のサイズに関する情報に依存するアプリケーションを使用するユーザーのために、準拠性についての詳細を[表 B-5](#) に示します。

表 B-5 データベース要素のサイズ設定

データベース要素	FIPS	Oracle9i
識別子の長さ（バイト単位）	18	30
CHARACTER データ型の長さ（バイト単位）	240	2000
NUMERIC データ型の 10 進精度	15	38
DECIMAL データ型の 10 進精度	15	38
INTEGER データ型の 10 進精度	9	38
SMALLINT データ型の 10 進精度	4	38
FLOAT データ型の 2 進精度	20	126
REAL データ型の 2 進精度	20	63
DOUBLE PRECISION データ型の 2 進精度	30	126
表の中の列	100	1000
INSERT 文の中の値	100	1000
UPDATE 文内の SET 句 ^(a)	20	1000
行の長さ ^(b,c)	2,000	2,000,000
一意制約の中の列	6	32
一意制約の長さ ^(b)	120	(d)
外部キー列リストの長さ ^(b)	120	(d)
GROUP BY 句の中の列	6	255 ^(e)
GROUP BY 列のリストの長さ	120	(e)
ORDER BY 句の中のソート指定	6	255 ^(e)
ORDER BY 列のリストの長さ	120	(e)
参照整合性制約内の列	6	32

表 B-5 データベース要素のサイズ設定 (続き)

データベース要素	FIPS	Oracle9i
SQL 文で参照される表	15	無制限
同時にオープンできるカーソル	10	(f)
SELECT 構文のリストの項目	100	1000

表 B-5 の注意事項:

(a) UPDATE 文の SET 句の数とは、SET キーワードの後に続くカンマで区切られる項目の数のことです。

(b) FIPS PUB では、列セットの長さを次の値の合計として規定しています。つまり、列の数を 2 倍した値、各文字列の長さ (バイト単位)、各真数値列の 10 進精度に 1 を加えた値、各概数値列の 2 進精度を 4 で割って 1 を加えた値の合計です。

(c) 行の最大長に対する Oracle の制限は、長さ 2GB の LONG 値とそれぞれの長さが 4000 バイトである 999 の VARCHAR2 値を含む行の最大長に基づいています。

$$2(254) + 231 + (999(4000))$$

(d) 一意キー制約に対する Oracle の制限は、Oracle データ・ブロックのサイズ (初期化パラメータ DB_BLOCK_SIZE によって指定される) の半分からオーバーヘッドを引いたものになります。

(e) Oracle は、GROUP BY 句の列数や ORDER BY 句のソート指定の数に対して制限を設定しません。ただし、GROUP BY 句や ORDER BY 句のすべての式のサイズの合計は、Oracle データ・ブロックのサイズ (初期化パラメータ DB_BLOCK_SIZE によって指定される) からオーバーヘッドを引いたサイズに制限されています。

(f) 同時にオープンできるカーソルの数に対する Oracle の制限は、初期化パラメータ OPEN_CURSORS によって指定されます。このパラメータの最大値は、使用しているオペレーティング・システムで使用可能なメモリーによって異なりますが、どんな場合でも 100 を超えません。

標準 SQL に対する Oracle 拡張機能

Oracle では、標準 SQL 以外にも様々な機能をサポートしています。Oracle アプリケーションでは、Core SQL:1999 の使用と同様に、これらの拡張機能を使用できます。

他の SQL 処理系へのアプリケーションの移植性を考慮する場合、Oracle の FIPS フラガーを使用して、埋込み SQL プログラムでの Entry SQL92 に Oracle の拡張機能を位置付けてください。FIPS フラガーは、Oracle プリコンパイラと SQL*Module コンパイラの一部です。

参照： FIPS フラガーの使用の詳細は、『Pro*COBOL Precompiler プログラマーズ・ガイド』および『Pro*C/C++Precompiler プログラマーズ・ガイド』を参照してください。

キャラクタ・セットのサポート

Oracle は、ほとんどの各国語およびベンダー固有のエンコードされたキャラクタ・セットの規格をサポートしています。Oracle がサポートするキャラクタ・セットの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』の付録 A「ロケール・データ」を参照してください。

Unicode は、エンコードされたユニバーサル・キャラクタ・セットで、単一キャラクタ・セットを使用したすべての言語の情報を格納できます。Unicode は、XML、Java、JavaScript、LDAP などの最新の規格が必要です。Unicode は、ISO/IEC 規格 10646 に準拠しています。ISO/IEC 規格 10646 のコピーについては、次の宛先に申し込んでください。

International Organization for Standardization
1 Rue de Varemb_
Case postale 56
CH-1211, Geneva 20, Switzerland
電話番号 : +41.22.749.0111
FAX 番号 : +41.22.733.3430
URL: <http://www.iso.ch/>

Oracle9i は、Unicode 規格の最新バージョンである Unicode 3.0 に完全に準拠しています。この規格の最新情報については、次の Unicode Consortium の Web サイトを参照してください。

<http://www.unicode.org>

Oracle は、3 つのデータベース・キャラクタ・セット（ASCII ベースのプラットフォーム用の UTF8 と AL32UTF8、および EBCDIC プラットフォーム用の UTFE）によってエンコードされた UTF-8（8 ビット）を使用します。Unicode のサポートを段階的に実装する場合は、SQL の NCHAR データ型（NCHAR、NVARCHAR2 および NCLOB）に対して、Unicode データを、各国語キャラクタ・セットで、UTF-16 または UTF-8 エンコード形式によって格納できます。

参照： Oracle のキャラクタ・セットのサポートの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

Oracle の予約語

この付録では、Oracle の予約語を示します。アスタリスク (*) が付いた語は、ANSI の予約語でもあります。

注意： Oracle では、次の表に示す予約語の他に、暗黙的に生成されるスキーマ・オブジェクトとサブオブジェクトには「SYS_」で始まるシステム生成名を使用します。名前解決での競合を避けるため、この接頭辞は明示的に指定するスキーマ・オブジェクト名やサブオブジェクト名では使用しないでください。

ACCESS
ADD *
ALL *
ALTER *
AND *
ANY *
AS *
ASC *
AUDIT
BETWEEN *
BY *
CHAR *
CHECK *
CLUSTER
COLUMN
COMMENT
COMPRESS
CONNECT *
CREATE *
CURRENT *
DATE *
DECIMAL *

DEFAULT *
DELETE *
DESC *
DISTINCT *
DROP *
ELSE *
EXCLUSIVE
EXISTS
FILE
FLOAT *
FOR *
FROM *
GRANT *
GROUP *
HAVING *
IDENTIFIED
IMMEDIATE *
IN *
INCREMENT
INDEX
INITIAL
INSERT *
INTEGER *
INTERSECT *
INTO *
IS *
LEVEL *
LIKE *
LOCK
LONG
MAXEXTENTS
MINUS
MLSLABEL
MODE
MODIFY
NOAUDIT
NOCOMPRESS
NOT *
NOWAIT
NULL *
NUMBER
OF *
OFFLINE
ON *
ONLINE

OPTION *
OR *
ORDER *
PCTFREE
PRIOR *
PRIVILEGES *
PUBLIC *
RAW
RENAME
RESOURCE
REVOKE *
ROW
ROWID
ROWNUM
ROWS *
SELECT *
SESSION *
SET *
SHARE
SIZE *
SMALLINT *
START
SUCCESSFUL
SYNONYM
SYSDATE
TABLE *
THEN *
TO *
TRIGGER
UID
UNION *
UNIQUE *
UPDATE *
USER *
VALIDATE
VALUES *
VARCHAR *
VARCHAR2
VIEW *
WHENEVER *
WHERE
WITH *

このマニュアルには、ほぼすべての SQL 文の例が記載されています。この付録では、単一の SQL 文のコンテキストに適した詳細な例を示します。これらの例は、特定の Oracle 機能を活用するために使用する一連の手順を示すためのものです。他の章にある個々の SQL 文の構文図およびセマンティクスにかわるものではありません。記載されている参照項目を使用して、必要な権限、制限事項、構文などの補足情報を参照してください。

この付録では、次の内容を説明します。

- [拡張索引作成機能の使用方法](#)
- [SQL 文での XML の使用方法](#)

拡張索引作成機能の使用方法

この項では、拡張索引作成機能の単純で現実的な使用例で必要となる手順の例を示します。

HR.employees 表の給与をランク付けし、10 ～ 20 にランク付けされる給与を検索するとします。この場合、次のように DENSE_RANK ファンクションを使用できます。

```
SELECT last_name, salary FROM
  (SELECT last_name, DENSE_RANK() OVER
    (ORDER BY salary DESC) rank_val, salary FROM employees)
WHERE rank_val BETWEEN 10 AND 20;
```

参照： 6-51 ページの「[DENSE_RANK](#)」を参照してください。

このネストした問合せは多少複雑で、employees 表のソートのみではなく全体スキャンも必要です。同じ結果が得られるもう 1 つの方法には、拡張索引作成機能を使用する方法があります。この方法による問合せは単純になります。この問合せには、ROWID による索引スキャンおよび表アクセスのみ必要です。そのため、問合せが効率的に実行されます。

最初の手順では、position_im 実装タイプ（索引の定義、メンテナンスおよび作成を行うためのメソッド・ヘッダーを含む）を作成します。（型本体のほとんどが PL/SQL を使用します。）

参照：

- 16-3 ページの「[CREATE TYPE](#)」および 16-25 ページの「[CREATE TYPE BODY](#)」を参照してください。
- この文の ODCI ルーチンの詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- 『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。

```
CREATE OR REPLACE TYPE position_im AS OBJECT
(
  curnum NUMBER,
  howmany NUMBER,
  lower_bound NUMBER, --lower_bound and upper_bound are used for the
  upper_bound NUMBER, --index-based functional implementation.
  STATIC FUNCTION ODCIGETINTERFACES(ifclist OUT SYS.ODCIOBJECTLIST) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXCREATE
    (ia SYS.ODCIINDEXINFO, parms VARCHAR2, env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXTRUNCATE (ia SYS.ODCIINDEXINFO,
    env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXDROP(ia SYS.ODCIINDEXINFO,
    env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXINSERT(ia SYS.ODCIINDEXINFO, rid ROWID,
```

```

        newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
    STATIC FUNCTION ODCIINDEXDELETE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
        env SYS.ODCIEnv) RETURN NUMBER,
    STATIC FUNCTION ODCIINDEXUPDATE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
        newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
    STATIC FUNCTION ODCIINDEXSTART(SCTX IN OUT position_im, ia SYS.ODCIINDEXINFO,
        op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
        strt NUMBER, stop NUMBER, lower_pos NUMBER,
        upper_pos NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
    MEMBER FUNCTION ODCIINDEXFETCH(SELF IN OUT position_im, nrows NUMBER,
        rids OUT SYS.ODCIRIDLIST, env SYS.ODCIEnv)
        RETURN NUMBER,
    MEMBER FUNCTION ODCIINDEXCLOSE(env SYS.ODCIEnv) RETURN NUMBER
);
/

CREATE OR REPLACE TYPE BODY position_im
IS
    STATIC FUNCTION ODCIGETINTERFACES(ifclist OUT SYS.ODCIOBJECTLIST)
        RETURN NUMBER IS
    BEGIN
        ifclist := SYS.ODCIOBJECTLIST(SYS.ODCIOBJECT('SYS','ODCIINDEX2'));
        RETURN ODCICONST.SUCCESS;
    END ODCIGETINTERFACES;

    STATIC FUNCTION ODCIINDEXCREATE (ia SYS.ODCIINDEXINFO, parms VARCHAR2, env SYS.ODCIEnv) RETURN NUMBER
    IS
        stmt    VARCHAR2(2000);
    BEGIN
        -- construct the sql statement
        stmt := 'Create Table ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB' || '(col_val, base_rowid, constraint pk PRIMARY KEY ' ||
            '(col_val, base_rowid)) ORGANIZATION INDEX AS SELECT ' ||
            ia.INDEXCOLS(1).COLNAME || ', ROWID FROM ' ||
            ia.INDEXCOLS(1).TABLESCHEMA || '.' || ia.INDEXCOLS(1).TABLERNAME;

        EXECUTE IMMEDIATE stmt;
        RETURN ODCICONST.SUCCESS;
    END;

    STATIC FUNCTION ODCIINDEXDROP(ia SYS.ODCIINDEXINFO, env SYS.ODCIEnv) RETURN NUMBER IS
        stmt VARCHAR2(2000);
    BEGIN
        -- construct the sql statement
        stmt := 'DROP TABLE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB';

```

```
EXECUTE IMMEDIATE stmt;
RETURN ODCICONST.SUCCESS;
END;

STATIC FUNCTION ODCIINDEXTRUNCATE(ia SYS.ODCIINDEXINFO, env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
BEGIN
    -- construct the sql statement
    stmt := 'TRUNCATE TABLE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME || '_STORAGE_TAB';

    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
END;

STATIC FUNCTION ODCIINDEXINSERT(ia SYS.ODCIINDEXINFO, rid ROWID,
                                newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
BEGIN
    -- construct the sql statement
    stmt := 'INSERT INTO ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
        '_STORAGE_TAB VALUES (' || newval || ', ' || rid || ')';

    -- execute the statement
    EXECUTE IMMEDIATE stmt;

    RETURN ODCICONST.SUCCESS;
END;

STATIC FUNCTION ODCIINDEXDELETE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
                                env SYS.ODCIEnv)
    RETURN NUMBER IS
    stmt VARCHAR2(2000);
BEGIN
    -- construct the sql statement
    stmt := 'DELETE FROM ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
        '_STORAGE_TAB WHERE col_val = ' || oldval || ' AND base_rowid = ' || rid || '';

    -- execute the statement
    EXECUTE IMMEDIATE stmt;

    RETURN ODCICONST.SUCCESS;
END;

STATIC FUNCTION ODCIINDEXUPDATE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
                                newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
BEGIN
```

```

-- construct the sql statement
stmt := 'UPDATE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
        '_STORAGE_TAB SET col_val = ''' || newval || ''' WHERE f2 = ''' || rid || '''';

-- execute the statement
EXECUTE IMMEDIATE stmt;

RETURN ODCICONST.SUCCESS;
END;

STATIC FUNCTION ODCIINDEXSTART(SCTX IN OUT position_im, ia SYS.ODCIINDEXINFO,
                               op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
                               strt NUMBER, stop NUMBER, lower_pos NUMBER,
                               upper_pos NUMBER, env SYS.ODCIEnv) RETURN NUMBER IS

    rid          VARCHAR2(5072);
    storage_tab_name VARCHAR2(65);
    lower_bound_stmt VARCHAR2(2000);
    upper_bound_stmt VARCHAR2(2000);
    range_query_stmt VARCHAR2(2000);
    lower_bound    NUMBER;
    upper_bound    NUMBER;
    cnum           INTEGER;
    nrows          INTEGER;

BEGIN
    -- Take care of some error cases.
    -- The only predicates in which position operator can appear are
    --     op() = 1      OR
    --     op() = 0      OR
    --     op() between 0 and 1
    IF (((strt != 1) AND (strt != 0)) OR
        ((stop != 1) AND (stop != 0)) OR
        ((strt = 1) AND (stop = 0))) THEN
        RAISE_APPLICATION_ERROR(-20101,
                                'incorrect predicate for position_between operator');
    END IF;

    IF (lower_pos > upper_pos) THEN
        RAISE_APPLICATION_ERROR(-20101, 'Upper Position must be greater than or
        equal to Lower Position');
    END IF;

    IF (lower_pos <= 0) THEN
        RAISE_APPLICATION_ERROR(-20101, 'Both Positions must be greater than zero');
    END IF;

```

```

storage_tab_name := ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
'_STORAGE_TAB';
upper_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
storage_tab_name || ') */ DISTINCT ' ||
'col_val FROM ' || storage_tab_name || ' ORDER BY ' ||
'col_val DESC) WHERE rownum <= ' || lower_pos;
EXECUTE IMMEDIATE upper_bound_stmt INTO upper_bound;

IF (lower_pos != upper_pos) THEN
    lower_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
storage_tab_name || ') */ DISTINCT ' ||
'col_val FROM ' || storage_tab_name ||
' WHERE col_val < ' || upper_bound || ' ORDER BY ' ||
'col_val DESC) WHERE rownum <= ' ||
(upper_pos - lower_pos);
    EXECUTE IMMEDIATE lower_bound_stmt INTO lower_bound;
ELSE
    lower_bound := upper_bound;
END IF;

IF (lower_bound IS NULL) THEN
    lower_bound := upper_bound;
END IF;

range_query_stmt := 'Select base_rowid FROM ' || storage_tab_name ||
' WHERE col_val BETWEEN ' || lower_bound || ' AND ' ||
upper_bound;

cnum := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(cnum, range_query_stmt, DBMS_SQL.NATIVE);

-- set context as the cursor number
SCTX := position_im(cnum, 0, 0, 0);

-- return success
RETURN ODCICONST.SUCCESS;
END;

MEMBER FUNCTION ODCIINDEXFETCH(SELF IN OUT position_im, nrows NUMBER,
rids OUT SYS.ODCIRIDLIST, env SYS.ODCIEnv)
RETURN NUMBER IS
    cnum    INTEGER;
    rid_tab DBMS_SQL.Varchar2_table;
    rlist    SYS.ODCIRIDLIST := SYS.ODCIRIDLIST();
    i        INTEGER;
    d        INTEGER;
BEGIN

```



```

cnum := SELF.curnum;

IF self.howmany = 0 THEN
    dbms_sql.define_array(cnum, 1, rid_tab, nrows, 1);
    d := DBMS_SQL.EXECUTE(cnum);
END IF;

d := DBMS_SQL.FETCH_ROWS(cnum);

IF d = nrows THEN
    rlist.extend(d);
ELSE
    rlist.extend(d+1);
END IF;

DBMS_SQL.COLUMN_VALUE(cnum, 1, rid_tab);

for i in 1..d loop
    rlist(i) := rid_tab(i+SELF.howmany);
end loop;

SELF.howmany := SELF.howmany + d;
rids := rlist;

RETURN ODCICONST.SUCCESS;
END;

MEMBER FUNCTION ODCIINDEXCLOSE(env SYS.ODCIEnv) RETURN NUMBER IS
    cnum INTEGER;
BEGIN
    cnum := SELF.curnum;
    DBMS_SQL.CLOSE_CURSOR(cnum);
    RETURN ODCICONST.SUCCESS;
END;

END;
/

```

次の手順では、索引タイプに関連付けられる演算子に必要な
function_for_position_between ファンクション実装を作成します。

このファンクションは、索引ベースのファンクション評価で使用するためのものです。そのため、索引コンテキストおよびスキャン・コンテキストをパラメータとして指定します。

参照：

- 索引ベースのファンクション実装を作成する場合の詳細は、『Oracle9i Data Cartridge Developer's Guide』を参照してください。
- 13-49 ページの「[CREATE FUNCTION](#)」および『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。

```
CREATE OR REPLACE FUNCTION function_for_position_between
    (col NUMBER, lower_pos NUMBER, upper_pos NUMBER,
     indexctx IN SYS.ODCIIndexCtx,
     scanctx IN OUT position_im,
     scanflg IN NUMBER)

RETURN NUMBER AS
    rid          ROWID;
    storage_tab_name VARCHAR2(65);
    lower_bound_stmt VARCHAR2(2000);
    upper_bound_stmt VARCHAR2(2000);
    col_val_stmt   VARCHAR2(2000);
    lower_bound    NUMBER;
    upper_bound    NUMBER;
    column_value   NUMBER;
BEGIN
    IF (indexctx.IndexInfo IS NOT NULL) THEN
        storage_tab_name := indexctx.IndexInfo.INDEXSCHEMA || '.' ||
            indexctx.IndexInfo.INDEXNAME || '_STORAGE_TAB';
    IF (scanctx IS NULL) THEN
        --This is the first call.  Open a cursor for future calls.
        --First, do some error checking
        IF (lower_pos > upper_pos) THEN
            RAISE_APPLICATION_ERROR(-20101,
                'Upper Position must be greater than or equal to Lower Position');
        END IF;
        IF (lower_pos <= 0) THEN
            RAISE_APPLICATION_ERROR(-20101,
                'Both Positions must be greater than zero');
        END IF;
        --Obtain the upper and lower value bounds for the range we're interested
        --in.
        upper_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
            storage_tab_name || ') */ DISTINCT ' ||
            'col_val FROM ' || storage_tab_name || ' ORDER BY ' ||
            'col_val DESC) WHERE rownum <= ' || lower_pos;
        EXECUTE IMMEDIATE upper_bound_stmt INTO upper_bound;
        IF (lower_pos != upper_pos) THEN
            lower_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
                storage_tab_name || ') */ DISTINCT ' ||
```

```

        'col_val FROM ' || storage_tab_name ||
        ' WHERE col_val < ' || upper_bound || ' ORDER BY ' ||
        'col_val DESC) WHERE rownum <= ' ||
        (upper_pos - lower_pos);
EXECUTE IMMEDIATE lower_bound_stmt INTO lower_bound;
ELSE
    lower_bound := upper_bound;
END IF;
IF (lower_bound IS NULL) THEN
    lower_bound := upper_bound;
END IF;
--Store the lower and upper bounds for future function invocations for
--the positions.
scanctx := position_im(0, 0, lower_bound, upper_bound);
END IF;
--Fetch the column value corresponding to the rowid, and see if it falls
--within the determined range.
col_val_stmt := 'Select col_val FROM ' || storage_tab_name ||
    ' WHERE base_rowid = ''' || indexctx.Rid || '''';
EXECUTE IMMEDIATE col_val_stmt INTO column_value;
IF (column_value <= scanctx.upper_bound AND
    column_value >= scanctx.lower_bound AND
    scanflg = ODCICONST.RegularCall) THEN
    RETURN 1;
ELSE
    RETURN 0;
END IF;
ELSE
    RAISE_APPLICATION_ERROR(-20101, 'A column that has a domain index of' ||
        'Position indextype must be the first argument');
END IF;
END;
/

```

次の手順は、`function_for_position_between` ファンクションを使用する `position_between` 演算子を作成します。この演算子には、索引列 `NUMBER` を最初の引数として指定し、その後に `NUMBER` の下限および上限を 2 番目および 3 番目の引数として指定します。

参照： 14-40 ページの「[CREATE OPERATOR](#)」を参照してください。

```

CREATE OR REPLACE OPERATOR position_between
    BINDING (NUMBER, NUMBER, NUMBER) RETURN NUMBER
    WITH INDEX CONTEXT, SCAN CONTEXT position_im
    USING function_for_position_between;

```

索引コンテキストおよびスキャン・コンテキストが、索引ベースのファンクション評価に渡されるように、この CREATE OPERATOR 文には WITH INDEX CONTEXT, SCAN CONTEXT position_im 句が含まれています。

次の手順は、position_operator に必要な索引タイプ position_indextype を作成します。

参照： 13-88 ページの「[CREATE INDEXTYPE](#)」を参照してください。

```
CREATE INDEXTYPE position_indextype
  FOR position_between(NUMBER, NUMBER, NUMBER)
  USING position_im;
```

演算子 position_between は、索引ベースのファンクション実装を使用します。そのため、索引情報がファンクション評価に渡されるように、参照列にドメイン索引を定義する必要があります。そのため、最後の手順では、索引タイプ position_indextype を使用してドメイン索引 salary_index を作成します。

参照： 13-61 ページの「[CREATE INDEX](#)」を参照してください。

```
CREATE INDEX salary_index ON employees(salary)
  INDEXTYPE IS position_indextype;
```

これで、演算子 position_between を使用して、元の間合せを次のように書き換えることができます。

```
SELECT last_name, salary FROM employees
  WHERE position_between(salary, 10, 20)=1
  ORDER BY salary DESC;
```

LAST_NAME	SALARY
-----	-----
Tucker	10000
King	10000
Baer	10000
Bloom	10000
Fox	9600
Bernstein	9500
Sully	9500
Greene	9500
Hunold	9000
Faviet	9000
McEwen	9000
Hall	9000
Hutton	8800
Taylor	8600
Livingston	8400

Gietz	8300
Chen	8200
Fripp	8200
Weiss	8000
Olsen	8000
Smith	8000
Kaufling	7900

SQL 文での XML の使用方法

この項では、XMLType データをデータベースで使用方法を説明します。

XMLType 表

サンプル・スキーマ oe には表 warehouses が含まれ、この表には XMLType 列 warehouse_spec が含まれます。warehouse_spec 情報を持つ別の表を作成するとします。次の例は、暗黙的に CLOB 列を 1 列のみ持つ非常に単純な XMLType 表を作成します。

```
CREATE TABLE xwarehouses OF XMLTYPE;
```

このような表には、次の文に示すように、XMLType 構文を使用してデータを挿入することができます。（この例で挿入されるデータは、サンプル表 oe.warehouses の warehouse_spec 列にあるデータに対応します。warehouse_id は 1 です。）

```
INSERT INTO xwarehouses VALUES
(xmltype('<?xml version="1.0"?>
<Warehouse>
  <WarehouseId>1</WarehouseId>
  <WarehouseName>Southlake, Texas</WarehouseName>
  <Building>Owned</Building>
  <Area>25000</Area>
  <Docks>2</Docks>
  <DockType>Rear load</DockType>
  <WaterAccess>true</WaterAccess>
  <RailAccess>N</RailAccess>
  <Parking>Street</Parking>
  <VClearance>10</VClearance>
</Warehouse>'));
```

参照： XMLType およびそのメンバー・メソッドの詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。

この表の間合せには、次の文を使用します。

```
SELECT e.getClobVal() FROM xwarehouses e;
```

データは暗黙的に CLOB 列に格納されるため、LOB 列に対するすべての制限事項が適用されます。これらの制限事項を回避するには、XMLSchema ベースの表を作成します。

XMLSchema は、XML 要素を対応するオブジェクト・リレーショナル・データにマップします。次の例は、XMLSchema をローカルに登録します。XMLSchema (xwarehouses.xsd) には、xwarehouses 表と同じ構造が反映されます。

参照： XMLSchema の作成の詳細は、『Oracle9i XML データベース開発者ガイド -Oracle XML DB』を参照してください。

```
begin
dbms_xmlschema.registerSchema(
  'http://www.oracle.com/xwarehouses.xsd',
  '<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oracle.com/xwarehouses.xsd"
    xmlns:who="http://www.oracle.com/xwarehouses.xsd"
    version="1.0">

<simpleType name="RentalType">
  <restriction base="string">
    <enumeration value="Rented"/>
    <enumeration value="Owned"/>
  </restriction>
</simpleType>

<simpleType name="ParkingType">
  <restriction base="string">
    <enumeration value="Street"/>
    <enumeration value="Lot"/>
  </restriction>
</simpleType>

<element name = "Warehouse">
  <complexType>
    <sequence>
      <element name = "WarehouseId" type = "positiveInteger"/>
      <element name = "WarehouseName" type = "string"/>
      <element name = "Building" type = "who:RentalType"/>
      <element name = "Area" type = "positiveInteger"/>
      <element name = "Docks" type = "positiveInteger"/>
      <element name = "DockType" type = "string"/>
      <element name = "WaterAccess" type = "boolean"/>
      <element name = "RailAccess" type = "boolean"/>
      <element name = "Parking" type = "who:ParkingType"/>
      <element name = "VClearance" type = "positiveInteger"/>
    </sequence>
  </complexType>
```

```

    </element>
</schema>',
    TRUE, TRUE, FALSE, FALSE);
end;
/

```

これで、次の例に示すように、XMLSchema ベースの表を作成できます。

```

CREATE TABLE xwarehouses OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/xwarehouses.xsd"
    ELEMENT "Warehouse";

```

デフォルトでは、この表はオブジェクト・リレーショナル表として格納されます。そのため、次の例に示すように、この表にデータを挿入できます。（この例で挿入されるデータは、サンプル表 `oe.warehouses` の `warehouse_spec` 列にあるデータに対応します。`warehouse_id` は 1 です。）

```

INSERT INTO xwarehouses VALUES (
    xmltype.createxml('<?xml version="1.0"?>
    <who:Warehouse xmlns:who="http://www.oracle.com/xwarehouse.xsd"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:schemaLocation="http://www.oracle.com/xwarehouse.xsd
    http://www.oracle.com/xwarehouse.xsd">
        <WarehouseId>1</WarehouseId>
        <WarehouseName>Southlake, Texas</WarehouseName>
        <Building>Owned</Building>
        <Area>25000</Area>
        <Docks>2</Docks>
        <DockType>Rear load</DockType>
        <WaterAccess>true</WaterAccess>
        <RailAccess>false</RailAccess>
        <Parking>Street</Parking>
        <VClearance>10</VClearance>
    </who:Warehouse>'))';
...

```

XMLSchema ベースの表には制約を定義できます。その場合は、XML 要素 `Warehouse` 内の適切な属性を参照するように、XMLDATA 疑似列を使用します。

```

ALTER TABLE xwarehouses ADD (PRIMARY KEY (XMLDATA."WarehouseId"));

```

`xwarehouses` のデータはオブジェクト・リレーショナルに格納されるため、可能な場合に基礎となる記憶域を参照できるように、この XMLType 表への問合せが書き換えられます。そのため、次の問合せでは、前述の例の主キー制約によって作成された索引を使用します。

```
SELECT * FROM xwarehouses x
WHERE EXISTSNODE(VALUE(x), '/Warehouse[WarehouseId="1"]') = 1;
```

```
SELECT * FROM xwarehouses x
WHERE EXTRACTVALUE(VALUE(x), '/Warehouse/WarehouseId') = 1;
```

XMLSchema ベースの表に索引を明示的に作成すると、後続の間合せのパフォーマンスが大幅に向上します。XMLType 表にオブジェクト・リレーショナル・ビューを作成することも、オブジェクト・リレーショナル表に XMLType ビューを作成することもできます。

参照：

- XMLDATA 疑似列の詳細は、2-84 ページの「XMLDATA」を参照してください。
- 16-52 ページの「XMLType ビューの作成例：」を参照してください。
- 13-82 ページの「XMLType 表の索引の作成例：」を参照してください。

XMLType 列

サンプル表 `oe.warehouses` は、XMLType 型の `warehouse_spec` 列を使用して作成されました。記憶域は指定していないため、XMLType 列は暗黙的に CLOB として格納されます。この項の例は、2 つのタイプの記憶域を使用して、簡略化した `oe.warehouses` 表を作成します。

最初の例は、CLOB として格納された XMLType 表を持つ表を作成します。この表に XMLSchema は不要です。そのため、事前に決められたコンテンツの構造はありません。

```
CREATE TABLE xwarehouses (
  warehouse_id      NUMBER,
  warehouse_spec     XMLTYPE)
XMLTYPE warehouse_spec STORE AS CLOB
(TABLESPACE example
 STORAGE (INITIAL 6144 NEXT 6144)
 CHUNK 4000
 NOCACHE LOGGING);
```

次の例でも前述の例とほぼ同じ表を作成しますが、指定された XMLSchema によって構造が決められているオブジェクト・リレーショナル XMLType 列に、XMLType データが格納されます。

```
CREATE TABLE xwarehouses (
  warehouse_id      NUMBER,
  warehouse_spec     XMLTYPE)
XMLTYPE warehouse_spec STORE AS OBJECT RELATIONAL
  XMLSCHEMA "http://www.oracle.com/xwarehouses.xsd"
  ELEMENT "Warehouse";
```


記号

- \$ (ドル記号)
 - 数値書式要素, 2-61
- % (パーセント) LIKE 演算子で使用, 5-16
- , (カンマ)
 - 数値書式要素, 2-61
 - 日時書式要素, 2-65
- (ダッシュ)
 - 日時書式要素, 2-65
- . (ピリオド)
 - 数値書式要素, 2-61
 - 日時書式要素, 2-65
- / (スラッシュ)
 - 日時書式要素, 2-65
- : (コロン)
 - 日時書式要素, 2-65
- ; (セミコロン)
 - 日時書式要素, 2-65

数字

- 0 (ゼロ)
 - 数値書式要素, 2-61
- 16 進数値
 - 戻す, 2-63
- 20 世紀, 2-69
- 21 世紀, 2-69
- 9 (九)
 - 数値書式要素, 2-61

A

- A.D. 日時書式要素, 2-65, 2-69
- A.M. 日時書式要素, 2-65, 2-69

- ABORT LOGICAL STANDBY 句
 - ALTER DATABASE, 9-47
- ABS ファンクション, 6-15
- ACCESSED GLOBALLY 句
 - CREATE CONTEXT, 13-12
- ACCOUNT LOCK 句
 - ALTER USER 「CREATE USER」を参照, 12-23
 - CREATE USER, 16-35
- ACCOUNT UNLOCK 句
 - ALTER USER 「CREATE USER」を参照
 - CREATE USER, 16-35
- ACOS ファンクション, 6-16
- ACTIVATE STANDBY DATABASE 句
 - ALTER DATABASE, 9-44
- ACTIVE_INSTANCE_COUNT 初期化パラメータ
 - ALTER SYSTEM で設定, 10-34
- ADD DATAFILE 句
 - ALTER TABLESPACE, 11-99
- ADD LOG GROUP 句
 - ALTER TABLE, 11-35
- ADD LOGFILE GROUP 句
 - ALTER DATABASE, 9-39
- ADD LOGFILE MEMBER 句
 - ALTER DATABASE, 9-20, 9-40
- ADD LOGFILE THREAD 句
 - ALTER DATABASE, 9-39
- ADD LOGFILE 句
 - ALTER DATABASE, 9-20
- ADD OVERFLOW 句
 - ALTER TABLE, 11-40
- ADD PARTITION 句
 - ALTER TABLE, 11-66, 11-68, 11-69
- ADD PRIMARY KEY 句
 - ALTER MATERIALIZED VIEW LOG, 9-112

- ADD ROWID 句
 - ALTER MATERIALIZED VIEW, 9-112
 - ALTER MATERIALIZED VIEW LOG, 9-112
- ADD SUPPLEMENTAL LOG DATA 句
 - ALTER DATABASE, 9-41
- ADD TEMPFILE 句
 - ALTER TABLESPACE, 11-99
- ADD VALUES 句
 - ALTER TABLEMODIFY PARTITION, 11-62
- ADD_MONTHS ファンクション, 6-16
- ADD 句
 - ALTER DIMENSION, 9-58
 - ALTER INDEXTYPE, 9-84
 - ALTER TABLE, 11-42
 - ALTER VIEW, 12-30
- ADMINISTER DATABASE TRIGGER システム権限, 17-41
- ADVISE 句
 - ALTER SESSION, 10-3
- AD 日時書式要素, 2-65, 2-69
- AFTER 句
 - CREATE TRIGGER, 15-92
- AFTER トリガー, 15-92
- AGENT 句
 - CREATE LIBRARY, 14-3
- ALL EXCEPT 句
 - SET ROLE, 18-45
- ALL PRIVILEGES 句
 - GRANT, 17-32
 - REVOKE, 17-91
- ALL PRIVILEGES ショートカット
 - AUDIT, 12-53
- ALL_COL_COMMENTS データ・ディクショナリ・ビュー, 12-67
- ALL_ROWS ヒント, 2-88
- ALL_TAB_COMMENTS データ・ディクショナリ・ビュー, 12-67
- ALLOCATE EXTENT 句
 - ALTER CLUSTER, 9-8, 9-9
 - ALTER INDEX, 9-63, 9-68
 - ALTER MATERIALIZED VIEW, 9-94
 - ALTER TABLE, 11-35
- ALLOW CORRUPTION 句
 - ALTER DATABASE RECOVER, 9-29
- ALL 演算子, 5-5
- ALL 句
 - SELECT, 18-11
 - SET CONSTRAINTS, 18-42
 - SET ROLE, 18-45
- ALL ショートカット
 - AUDIT, 12-53
 - ALTER ANY CLUSTER システム権限, 17-35
 - ALTER ANY DIMENSION システム権限, 17-36
 - ALTER ANY INDEXTYPE システム権限, 17-36
 - ALTER ANY INDEX システム権限, 17-36
 - ALTER ANY MATERIALIZED VIEW システム権限, 17-37
 - ALTER ANY OUTLINE システム権限, 17-38
 - ALTER ANY PROCEDURE システム権限, 17-38
 - ALTER ANY ROLE システム権限, 17-38
 - ALTER ANY SEQUENCE システム権限, 17-39
 - ALTER ANY TABLE システム権限, 17-40
 - ALTER ANY TRIGGER システム権限, 17-41
 - ALTER ANY TYPE システム権限, 17-41
 - ALTER CLUSTER 文, 9-7
 - ALTER DATABASE システム権限, 17-35
 - ALTER DATABASE 文, 9-12
 - ALTER DIMENSION 文, 9-56
 - ALTER FUNCTION 文, 9-59
 - ALTER INDEXTYPE 文, 9-83
 - ALTER INDEX 文, 9-61
 - ALTER JAVA CLASS 文, 9-85
 - ALTER JAVA SOURCE 文, 9-85
 - ALTER MATERIALIZED VIEW LOG 文, 9-108
 - ALTER MATERIALIZED VIEW 文, 9-88
 - ALTER OPERATOR 文, 9-115
 - ALTER OUTLINE 文, 9-116
 - ALTER PACKAGE 文, 9-118
 - ALTER PROCEDURE 文, 9-122
 - ALTER PROFILE システム権限, 17-38
 - ALTER PROFILE 文, 9-125
 - ALTER RESOURCE COST システム権限, 17-39
 - ALTER RESOURCE COST 文, 9-129
 - ALTER ROLE 文, 9-132
 - ALTER ROLLBACK SEGMENT システム権限, 17-39
 - ALTER ROLLBACK SEGMENT 文, 9-134
 - ALTER SEQUENCE 文, 9-138
 - ALTER SESSION システム権限, 17-39
 - ALTER SESSION 文, 10-2
 - ALTER SNAPSHOT LOG 「ALTER MATERIALIZED VIEW LOG」を参照
 - ALTER SNAPSHOT 「ALTER MATERIALIZED VIEW」を参照
 - ALTER SYSTEM システム権限, 17-35

- ALTER SYSTEM 文, 10-20
- ALTER TABLESPACE システム権限, 17-40
- ALTER TABLESPACE 文, 11-97
- ALTER TABLE 文, 11-2
- ALTER TRIGGER 文, 12-2
- ALTER TYPE 文, 12-6
- ALTER USER システム権限, 17-42
- ALTER USER 文, 12-21
- ALTER VIEW 文, 12-29
- alter_external_table_clause
 - ALTER TABLE, 11-17
- ALTER オブジェクト権限, 17-45
 - 順序, 17-47
 - 表, 17-46
- ALTER 文
 - トリガー, 15-94
- AM 日時書式要素, 2-65, 2-69
- ANALYZE ANY システム権限, 17-42
- ANALYZE CLUSTER 文, 12-32
- ANALYZE INDEX 文, 12-32
- ANALYZE TABLE 文, 12-32
- ANCILLARY TO 句
 - CREATE OPERATOR, 14-42
- AND DATAFILES 句
 - DROP TABLESPACE, 17-11
- AND_EQUAL ヒント, 2-88
- AND 条件, 5-8, 5-9
- ANSI「ANSI (米国規格協会)」を参照
- ANSI (米国規格協会), B-1
 - 暗黙的なデータ型変換, 2-34
 - 規格, xv, 1-2, B-2
 - サポートしているデータ型, 2-5
 - データ型, 2-34
 - Oracle データ型への変換, 2-34
- ANY 演算子, 5-5
- APPEND ヒント, 2-89
- AQ_ADMINISTRATOR_ROLE ロール, 17-44
- AQ_TM_PROCESSES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-34
- AQ_USER_ROLE ロール, 17-44
- ARCHIVE LOG 句
 - ALTER SYSTEM, 10-23
- ARCHIVE_LAG_TARGET 初期化パラメータ
 - ALTER SYSTEM で設定, 10-35
- ARCHIVELOG 句
 - ALTER DATABASE, 9-20, 9-38
 - CREATE CONTROLFILE, 13-19
 - CREATE DATABASE, 13-28
- AS EXTERNAL 句
 - CREATE FUNCTION, 14-65
 - CREATE TYPE BODY, 16-30
- AS OBJECT 句
 - CREATE TYPE, 16-9
- AS TABLE 句
 - CREATE TYPE, 16-19
- AS VARRAY 句
 - CREATE TYPE, 16-18
- ASCII
 - キャラクタ・セット, 2-44
- ASCIISTR ファンクション, 6-18
- ASCII ファンクション, 6-17
- ASC 句
 - CREATE INDEX, 13-72
- ASIN ファンクション, 6-19
- ASSOCIATE STATISTICS 文, 12-46
- AS 句
 - CREATE JAVA, 13-95
- AS 副問合せ句
 - CREATE MATERIALIZED VIEW, 14-24
 - CREATE TABLE, 15-58
 - CREATE VIEW, 16-45
- ATAN2 ファンクション, 6-20
- ATAN ファンクション, 6-19
- ATTRIBUTE 句
 - ALTER DIMENSION, 9-57
 - CREATE DIMENSION, 13-42, 13-45
- AUDIT ANY システム権限, 17-42
- AUDIT SYSTEM システム権限, 17-35
- AUDIT_FILE_DEST 初期化パラメータ
 - ALTER SYSTEM で設定, 10-35
- AUDIT_SYS_OPERATIONS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-35
- AUDIT_TRAIL 初期化パラメータ
 - ALTER SYSTEM で設定, 10-36
- AUTHENTICATED BY 句
 - CREATE DATABASE LINK, 13-38
- AUTHENTICATED 句
 - ALTER USER, 12-26
- AUTHID CURRENT_USER 句
 - ALTER JAVA, 9-86
 - CREATE FUNCTION, 13-54
 - CREATE JAVA, 13-91, 13-93
 - CREATE PACKAGE, 14-50
 - CREATE PROCEDURE, 14-63

- CREATE TYPE, 12-14, 16-10
- AUTHID DEFINER 句
 - ALTER JAVA, 9-86
 - CREATE FUNCTION, 13-54
 - CREATE JAVA, 13-91, 13-93
 - CREATE PACKAGE, 14-50
 - CREATE PROCEDURE, 14-63
 - CREATE TYPE, 12-14, 16-10
- AUTOALLOCATE 句
 - CREATE TABLESPACE, 15-82
- AUTOEXTEND 句
 - ALTER DATABASE, 9-19
 - CREATE DATABASE, 13-25
 - CREATE TEMPORARY TABLESPACE, 15-87
- AVG ファンクション, 6-20

B

B

- 数値書式要素, 2-61
- B.C. 日時書式要素, 2-65, 2-69
- BACKGROUND_CORE_DUMP 初期化パラメータ
 - ALTER SYSTEM で設定, 10-36
- BACKGROUND_DUMP_DEST 初期化パラメータ
 - ALTER SYSTEM で設定, 10-36
- BACKUP ANY TABLE システム権限, 17-40
- BACKUP CONTROLFILE 句
 - ALTER DATABASE, 9-21, 9-43
- BACKUP_TAPE_IO_SLAVES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-37
- BC 日時書式要素, 2-65, 2-69
- BECOME USER システム権限, 17-42
- BEFORE 句
 - CREATE TRIGGER, 15-91
- BEFORE トリガー, 15-91
- BEGIN BACKUP 句
 - ALTER TABLESPACE, 11-102
- BFILE
 - データ型, 2-30
 - ロケータ, 2-30
- BFILENAME ファンクション, 6-22
- BIN_TO_NUM ファンクション, 6-23
- BINDING 句
 - CREATE OPERATOR, 14-40, 14-42
- BITAND ファンクション, 6-24
- BITMAP_MERGE_AREA_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-37

- BITMAP 句
 - CREATE INDEX, 13-68
- BLANK_TRIMMING 初期化パラメータ
 - ALTER SYSTEM で設定, 10-38
- BLOB データ型, 2-30
 - トランザクションのサポート, 2-30
- BLOCKSIZE 句
 - CREATE TABLESPACE, 15-79
- BODY 句
 - ALTER PACKAGE, 9-119
- BUFFER_POOL_KEEP 初期化パラメータ
 - ALTER SYSTEM で設定, 10-38
- BUFFER_POOL_RECYCLE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-39
- BUFFER_POOL パラメータ
 - STORAGE 句, 7-58
- BUILD DEFERRED 句
 - CREATE MATERIALIZED VIEW, 14-18
- BUILD IMMEDIATE 句
 - CREATE MATERIALIZED VIEW, 14-18
- BY ACCESS 句
 - AUDIT, 12-55
- BY proxy 句
 - AUDIT, 12-54
- BY SESSION 句
 - AUDIT, 12-55
- BY user 句
 - AUDIT, 12-54
- BYTE キャラクタ・セマンティクス, 2-9, 2-11
- BYTE 長さセマンティクス, 11-50

C

C

- 数値書式要素, 2-61
- CACHE READS 句
 - ALTER TABLE, 11-45
 - CREATE TABLE, 15-50
- CACHE 句
 - ALTER MATERIALIZED VIEW, 9-99
 - ALTER MATERIALIZED VIEW LOG, 9-112
 - ALTER TABLE, 11-36, 15-50
 - CREATE CLUSTER, 13-9
 - CREATE MATERIALIZED VIEW, 14-17
 - CREATE MATERIALIZED VIEW LOG, 14-35

- CACHE パラメータ
 - ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138
 - CREATE SEQUENCE, 14-85
- CACHE ヒント, 2-89
- CALL 句
 - CREATE TRIGGER, 15-99
- CALL プロシージャ文
 - CREATE TRIGGER, 15-99
- CALL 文, 12-63
- CASCADE CONSTRAINTS 句
 - DROP CLUSTER, 16-65
 - DROP TABLE, 17-8
 - DROP TABLESPACE, 17-11
 - DROP VIEW, 17-22
 - REVOKE, 17-91
- CASCADE 句
 - CREATE TABLE, 15-57
 - DROP PROFILE, 16-92
 - DROP USER, 17-19
- CASE 式, 4-6
 - 検索, 4-6
 - 単純, 4-6
- CAST ファンクション, 6-25
 - MULTISET パラメータ, 6-25
- CATSEARCH 条件, 5-2
- CC 日時書式要素, 2-65
- CEIL ファンクション, 6-28
- CHANGE CATEGORY 句
 - ALTER OUTLINE, 9-117
- CHAR VARYING データ型、ANSI, 2-34
- CHARACTER SET パラメータ
 - ALTER DATABASE, 9-47
 - CREATE CONTROLFILE, 13-20
 - CREATE DATABASE, 13-29
- CHARACTER VARYING データ型
 - ANSI, 2-34
- CHARACTER データ型
 - ANSI, 2-34
 - DB2, 2-35
 - SQL/DS, 2-35
- CHARTOROWID ファンクション, 6-28
- CHAR キャラクタ・セマンティクス, 2-9, 2-11
- CHAR データ型, 2-9
 - ANSI, 2-34
 - VARCHAR2 への変換, 2-60
- CHAR の長さセマンティクス, 11-50
- CHECK DATAFILES 句
 - ALTER SYSTEM, 10-26
- CHECKPOINT 句
 - ALTER SYSTEM, 10-26
- CHECK 句
 - CREATE TABLE, 15-26
 - 制約, 7-15
- CHECK 制約, 7-15
- CHOOSE ヒント, 2-89
- CHR ファンクション, 6-29
- CHUNK 句
 - ALTER TABLE, 11-46
 - CREATE TABLE, 15-37
- CIRCUITS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-39
- CLEAR LOGFILE 句
 - ALTER DATABASE, 9-20, 9-42
- CLOB データ型, 2-31
 - トランザクションのサポート, 2-31
- CLOSE DATABASE LINK 句
 - ALTER SESSION, 10-3
- CLUSTER_DATABASE_INSTANCES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-40
- CLUSTER_DATABASE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-40
- CLUSTER_INTERCONNECTS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-40
- CLUSTER 句
 - ANALYZE, 12-37
 - CREATE INDEX, 13-69
 - CREATE TABLE, 15-34
 - TRUNCATE, 18-54
- CLUSTER ヒント, 2-89
- COALESCE SUBPARTITION 句
 - ALTER TABLE, 11-62
- COALESCE 句
 - ALTER INDEX, 9-76
 - ALTER TABLE, 11-42, 11-62
 - ALTER TABLESPACE, 11-104
 - パーティション, 11-69
- COALESCE ファンクション, 6-31
 - 様々な CASE 式, 6-31
- COLUMNS 句
 - ASSOCIATE STATISTICS, 12-46, 12-48
- COMMENT ANY TABLE システム権限, 17-42

COMMENT 句
 COMMIT, 12-71
 COMMENT 文, 12-67
 COMMIT IN PROCEDURE 句
 ALTER SESSION, 10-4
 COMMIT TO SWITCHOVER 句
 ALTER DATABASE, 9-45
 COMMIT_POINT_STRENGTH 初期化パラメータ
 ALTER SYSTEM で設定, 10-41
 COMMIT 文, 12-70
 COMPATIBLE 初期化パラメータ
 ALTER SYSTEM で設定, 10-41
 COMPILE 句
 ALTER DIMENSION, 9-58
 ALTER FUNCTION, 9-60
 ALTER JAVA SOURCE, 9-86
 ALTER MATERIALIZED VIEW, 9-104
 ALTER PACKAGE, 9-119
 ALTER PROCEDURE, 9-123
 ALTER TRIGGER, 12-4
 ALTER TYPE, 12-10
 ALTER VIEW, 12-31
 CREATE JAVA, 13-92
 COMPOSE ファンクション, 6-32
 COMPOSITE_LIMIT パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-69
 COMPRESS 句
 ALTER INDEX ... REBUILD, 9-73
 CREATE TABLE, 15-31
 COMPUTE STATISTICS 句
 ALTER INDEX...REBUILD, 9-74
 ANALYZE, 12-37
 CREATE INDEX, 13-75
 CONCAT ファンクション, 6-33
 CONNECT BY 句
 SELECT, 8-5, 18-20
 問合せおよび副問合せ, 18-20
 CONNECT THROUGH 句
 ALTER USER, 12-26
 CONNECT TO 句
 CREATE DATABASE LINK, 13-37
 CONNECT_TIME パラメータ
 ALTER PROFILE, 9-126
 ALTER RESOURCE COST, 9-129
 CONNECT 句
 SELECT および副問合せ, 18-8
 CONNECT ロール, 17-44
 CONSIDER FRESH 句
 ALTER MATERIALIZED VIEW, 9-104
 CONSTRAINT(S) セッション・パラメータ, 10-11
 CONTAINS 条件, 5-2
 CONTROL_FILE_RECORD_KEEP_TIME 初期化パラメータ
 ALTER SYSTEM で設定, 10-41
 CONTROL_FILES 初期化パラメータ
 ALTER SYSTEM で設定, 10-42
 CONTROLFILE REUSE 句
 CREATE DATABASE, 13-26
 controlfile 句
 ALTER DATABASE, 9-21
 CONVERT 句
 ALTER DATABASE, 9-49
 CONVERT ファンクション, 6-34
 CORE_DUMP_DEST 初期化パラメータ
 ALTER SYSTEM で設定, 10-42
 CORR ファンクション, 6-35
 COSH ファンクション, 6-37
 COS ファンクション, 6-37
 COUNT ファンクション, 6-38
 COVAR_POP ファンクション, 6-40
 COVAR_SAMP ファンクション, 6-42
 CPU_COUNT 初期化パラメータ
 ALTER SYSTEM で設定, 10-42
 CPU_PER_CALL パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-69
 CPU_PER_SESSION パラメータ
 ALTER PROFILE, 9-126
 ALTER RESOURCE COST, 9-129
 CREATE PROFILE, 14-69
 CREATE ANY CLUSTER システム権限, 17-35
 CREATE ANY CONTEXT システム権限, 17-35
 CREATE ANY DIMENSION システム権限, 17-36
 CREATE ANY DIRECTORY システム権限, 17-36
 CREATE ANY INDEXTYPE システム権限, 17-36
 CREATE ANY INDEX システム権限, 17-36
 CREATE ANY LIBRARY システム権限, 17-37
 CREATE ANY MATERIALIZED VIEW システム権限, 17-37
 CREATE ANY OPERATOR システム権限, 17-38
 CREATE ANY OUTLINE システム権限, 17-38
 CREATE ANY PROCEDURE システム権限, 17-38
 CREATE ANY SEQUENCE システム権限, 17-39

CREATE ANY SYNONYM システム権限, 17-39
 CREATE ANY TABLE システム権限, 17-40
 CREATE ANY TRIGGER システム権限, 17-41
 CREATE ANY TYPE システム権限, 17-41
 CREATE ANY VIEW システム権限, 17-42
 CREATE CLUSTER システム権限, 17-35
 CREATE CLUSTER 文, 13-2
 CREATE CONTEXT 文, 13-11
 CREATE CONTROLFILE 文, 13-14
 CREATE DATABASE LINK システム権限, 17-35
 CREATE DATABASE LINK 文, 13-35
 CREATE DATABASE 文, 13-22
 CREATE DATAFILE 句
 ALTER DATABASE, 9-18, 9-35
 CREATE DIMENSION
 システム権限, 17-36
 CREATE DIMENSION 文, 13-41
 CREATE DIRECTORY 文, 13-46
 CREATE FUNCTION 文, 13-49
 CREATE INDEX
 文, 13-61
 CREATE INDEXTYPE
 文, 13-88
 CREATE INDEXTYPE システム権限, 17-36
 CREATE JAVA 文, 13-90
 CREATE LIBRARY システム権限, 17-37
 CREATE LIBRARY 文, 14-2
 CREATE MATERIALIZED VIEW LOG 文, 14-31
 CREATE MATERIALIZED VIEW システム権限, 17-37
 CREATE MATERIALIZED VIEW 文, 14-4
 CREATE OPERATOR システム権限, 17-37
 CREATE OPERATOR 文, 14-40
 CREATE OUTLINE 文, 14-44
 CREATE PACKAGE BODY 文, 14-53
 CREATE PACKAGE 文, 14-48
 CREATE PFILE 文, 14-58
 CREATE PROCEDURE システム権限, 17-38
 CREATE PROCEDURE 文, 14-60
 CREATE PROFILE システム権限, 17-38
 CREATE PROFILE 文, 14-66
 CREATE PUBLIC DATABASE LINK システム権限,
 17-35
 CREATE PUBLIC SYNONYM システム権限, 17-39
 CREATE ROLE システム権限, 17-38
 CREATE ROLE 文, 14-73
 CREATE ROLLBACK SEGMENT システム権限, 17-39
 CREATE ROLLBACK SEGMENT 文, 14-76
 CREATE SCHEMA 文, 14-80
 CREATE SESSION システム権限, 17-39
 CREATE SEQUENCE 文, 14-82
 CREATE SESSION システム権限, 17-39
 CREATE SPFILE 文, 14-87
 CREATE STANDBY CONTROLFILE 句
 ALTER DATABASE, 9-21, 9-42
 CREATE SYNONYM システム権限, 17-39
 CREATE SYNONYM 文, 15-2
 CREATE TABLESPACE システム権限, 17-40
 CREATE TABLESPACE 文, 15-75
 CREATE TABLE システム権限, 17-40
 CREATE TABLE 文, 15-7
 CREATE TEMPORARY TABLESPACE 文, 15-86
 CREATE TRIGGER システム権限, 17-41
 CREATE TRIGGER 文, 15-89
 CREATE TYPE BODY 文, 16-25
 CREATE TYPE システム権限, 17-41
 CREATE TYPE 文, 16-3
 CREATE USER システム権限, 17-42
 CREATE USER 文, 16-31
 CREATE VIEW システム権限, 17-42
 CREATE VIEW 文, 16-38
 CREATE_BITMAP_AREA_SIZE 初期化パラメータ
 ALTER SYSTEM で設定, 10-43
 CREATE_STORED_OUTLINES 初期化パラメータ
 ALTER SYSTEM で設定, 10-43
 CREATE_STORED_OUTLINES セッション・パラメータ,
 10-11
 CREATE 文
 トリガー, 15-94
 CUBE 句
 SELECT 文, 18-22
 CUME_DIST ファンクション, 6-44
 CURRENT_DATE ファンクション, 6-46
 CURRENT_SCHEMA セッション・パラメータ, 10-12
 CURRENT_TIMESTAMP ファンクション, 6-46
 CURRENT_USER 句
 CREATE DATABASE LINK, 13-37
 CURRVAL 疑似列, 2-78, 14-82
 CURSOR_SHARING 初期化パラメータ
 ALTER SESSION で設定, 10-7, 10-44
 CURSOR_SPACE_FOR_TIME 初期化パラメータ
 ALTER SYSTEM で設定, 10-44
 CURSOR 式, 4-8
 CustomDatum Java 記憶形式, 16-11

CYCLE パラメータ

ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138

CREATE SEQUENCE, 14-84

NOCYCLE パラメータ

ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138

CREATE SEQUENCE, 14-85

C 関数

オブジェクト型へのマップ, 16-15

C 句

CREATE TYPE, 16-15

CREATE TYPE BODY, 16-29

D

D

数値書式要素, 2-61

DATAFILE END BACKUP 句

ALTER DATABASE, 9-34

DATAFILE OFFLINE 句

ALTER DATABASE, 9-36

DATAFILE ONLINE 句

ALTER DATABASE, 9-36

DATAFILE RESIZE 句

ALTER DATABASE, 9-36

DATAFILE 句

ALTER DATABASE, 9-18, 9-36

CREATE DATABASE, 13-29

DATE データ型, 2-17

ユリウス, 2-19

DATE 列

日時列への変換, 11-49

DAY 日時書式要素, 2-69

DB_CACHE_SIZE パラメータ

ALTER SYSTEM, 10-73, 10-87

DB_BLOCK_BUFFERS 初期化パラメータ

ALTER SYSTEM で設定, 10-45

DB_BLOCK_CHECKING 初期化パラメータ

ALTER SESSION で設定, 10-7

ALTER SYSTEM で設定, 10-46

DB_BLOCK_CHECKSUM 初期化パラメータ

ALTER SYSTEM で設定, 10-46

DB_BLOCK_SIZE 初期化パラメータ

ALTER SYSTEM で設定, 10-46

DB_CACHE_ADVICE 初期化パラメータ

ALTER SYSTEM で設定, 10-47

DB_CACHE_SIZE 初期化パラメータ

ALTER SYSTEM で設定, 10-47, 10-73, 10-87

DB_CREATE_FILE_DEST 初期化パラメータ

ALTER SESSION で設定, 10-7

ALTER SYSTEM で設定, 10-48

DB_CREATE_ONLINE_LOG_DEST_n 初期化パラメータ

ALTER SESSION で設定, 10-7

ALTER SYSTEM で設定, 10-48

DB_DOMAIN 初期化パラメータ

ALTER SYSTEM で設定, 10-48

DB_FILE_MULTIBLOCK_READ_COUNT 初期化パラメータ

ALTER SESSION で設定, 10-7

ALTER SYSTEM で設定, 10-49

DB_FILE_NAME_CONVERT 初期化パラメータ

ALTER SYSTEM で設定, 10-49

DB_FILES 初期化パラメータ

ALTER SYSTEM で設定, 10-50

DB_KEEP_CACHE_SIZE 初期化パラメータ

ALTER SYSTEM で設定, 10-50

DB_NAME 初期化パラメータ

ALTER SYSTEM で設定, 10-51

DB_nK_CACHE_SIZE 初期化パラメータ

ALTER SYSTEM で設定, 10-45

DB_RECYCLE_CACHE_SIZE 初期化パラメータ

ALTER SYSTEM で設定, 10-51

DB_WRITER_PROCESSES 初期化パラメータ

ALTER SYSTEM で設定, 10-51

DB2 データ型, 2-34

Oracle データ型への変換, 2-35

暗黙的な変換, 2-35

制限, 2-35

DBA_2PC_PENDING データ・ディクショナリ・ビュー, 10-3

DBA_COL_COMMENTS データ・ディクショナリ・ビュー, 12-67

DBA_ROLLBACK_SEGS データ・ディクショナリ・ビュー, 16-95

DBA_TAB_COMMENTS データ・ディクショナリ・ビュー, 12-67

DBA ロール, 17-44

DBLINK_ENCRYPT_LOGIN 初期化パラメータ

ALTER SYSTEM で設定, 10-52

DBMS_OUTPUT パッケージ, 12-4

DBMS_ROWID パッケージ

拡張 ROWID, 2-32

DBMSSTD.SQL スクリプト, 13-49, 14-48, 14-53,
 14-60
 トリガー, 15-89
 DBTIMEZONE ファンクション, 6-48
 DBWR_IO_SLAVES 初期化パラメータ
 ALTER SYSTEM で設定, 10-52
 DDAY 日時書式要素, 2-65
 DDD 日時書式要素, 2-65
 DDL データ定義言語 (DDL) を参照
 DD 日時書式要素, 2-65
 DEALLOCATE UNUSED 句
 ALTER CLUSTER, 9-8, 9-10
 ALTER INDEX, 9-63
 ALTER TABLE, 11-36
 DEBUG ANY PROCEDURE システム権限, 17-36
 DEBUG オブジェクト権限, 17-45
 オブジェクト型, 17-48
 ビュー, 17-46
 表, 17-46
 ファンクション、プロシージャまたはパッケージ,
 17-47
 DEBUG 句
 ALTER FUNCTION, 9-60
 ALTER PACKAGE, 9-120
 ALTER PROCEDURE, 9-123
 ALTER TRIGGER, 12-4
 ALTER TYPE, 12-10
 DECIMAL データ型
 ANSI, 2-34
 DB2, 2-35
 SQL/DS, 2-35
 DECODE ファンクション, 6-48
 DECOMPOSE ファンクション, 6-50
 DEFAULT COST 句
 ASSOCIATE STATISTICS, 12-47, 12-49
 DEFAULT ROLE 句
 ALTER USER, 12-25
 DEFAULT SELECTIVITY 句
 ASSOCIATE STATISTICS, 12-47, 12-49
 DEFAULT TABLESPACE 句
 ALTER USER 「CREATE USER」を参照
 CREATE USER, 16-34
 DEFAULT TEMPORARY TABLESPACE 句
 ALTER DATABASE, 9-49
 CREATE DATABASE, 13-25
 DEFAULT 記憶域句
 ALTER TABLESPACE, 11-101
 CREATE TABLESPACE, 15-80
 DEFAULT 句
 ALTER TABLE, 11-43
 CREATE TABLE, 15-26
 DEFAULT プロファイル
 ユーザーへの割当て, 16-92
 DEFERRABLE 句
 制約, 7-18
 DEFERRED 句
 SET CONSTRAINTS, 18-42
 definer-rights functions, 13-54
 DELETE ANY TABLE システム権限, 17-40
 DELETE STATISTICS 句
 ANALYZE, 12-43
 DELETE_CATALOG_ROLE ロール, 17-44
 DELETE オブジェクト権限, 17-45
 ビュー, 17-46
 表, 17-46
 DELETE 文, 16-53
 トリガー, 15-93
 DENSE_RANK ファンクション, 6-51
 Deref ファンクション, 6-53
 DESC 句
 CREATE INDEX, 13-72
 DETERMINISTIC 句
 CREATE FUNCTION, 13-55
 DG_BROKER_CONFIG_FILEn 初期化パラメータ
 ALTER SYSTEM で設定, 10-52
 DG_BROKER_START 初期化パラメータ
 ALTER SYSTEM で設定, 10-53
 DISABLE ALL TRIGGERS 句
 ALTER TABLE, 11-85
 DISABLE DISTRIBUTED RECOVERY 句
 ALTER SYSTEM, 10-28
 DISABLE NOVALIDATE 制約状態, 7-20, 15-54
 DISABLE PARALLEL DML 句
 ALTER SESSION, 10-4
 DISABLE QUERY REWRITE 句
 ALTER MATERIALIZED VIEW, 9-103
 CREATE MATERIALIZED VIEW, 14-23
 DISABLE RESTRICTED SESSION 句
 ALTER SYSTEM, 10-28
 DISABLE RESUMABLE 句
 ALTER SESSION, 10-6
 DISABLE ROW MOVEMENT 句
 ALTER TABLE, 11-6, 11-39
 CREATE TABLE, 15-15, 15-57

DISABLE STORAGE IN ROW 句
 ALTER TABLE, 11-45
 CREATE TABLE, 15-37
 DISABLE TABLE LOCK 句
 ALTER TABLE, 11-85
 DISABLE THREAD 句
 ALTER DATABASE, 9-50
 DISABLE VALIDATE 制約状態, 7-20, 15-54
 DISABLE 句
 ALTER INDEX, 9-75
 ALTER TRIGGER, 12-3
 CREATE TABLE, 15-52
 DISASSOCIATE STATISTICS 文, 16-61
 DISCONNECT SESSION 句
 ALTER SYSTEM, 10-26
 DISK_ASYNC_IO 初期化パラメータ
 ALTER SYSTEM で設定, 10-53
 DISPATCHERS 初期化パラメータ
 ALTER SYSTEM で設定, 10-54
 DISTINCT 句
 SELECT, 18-11
 DML_LOCKS 初期化パラメータ
 ALTER SYSTEM で設定, 10-55
 DML データ操作言語を参照
 domain_index_clause
 CREATE INDEX, 13-65
 DOUBLE PRECISION データ型 (ANSI), 2-34
 DRIVING_SITE ヒント, 2-90
 DROP ANY CLUSTER システム権限, 17-35
 DROP ANY CONTEXT システム権限, 17-35
 DROP ANY DIMENSION システム権限, 17-36
 DROP ANY DIRECTORY システム権限, 17-36
 DROP ANY INDEXTYPE システム権限, 17-36
 DROP ANY INDEX システム権限, 17-36
 DROP ANY LIBRARY システム権限, 17-37
 DROP ANY MATERIALIZED VIEW システム権限,
 17-37
 DROP ANY OPERATOR システム権限, 17-38
 DROP ANY OUTLINE システム権限, 17-38
 DROP ANY PROCEDURE システム権限, 17-38
 DROP ANY ROLE システム権限, 17-38
 DROP ANY SEQUENCE システム権限, 17-39
 DROP ANY SYNONYM システム権限, 17-39
 DROP ANY TABLE システム権限, 17-40
 DROP ANY TRIGGER システム権限, 17-41
 DROP ANY TYPE システム権限, 17-41
 DROP ANY VIEW システム権限, 17-42
 DROP CLUSTER 文, 16-64
 DROP COLUMN 句
 ALTER TABLE, 11-51
 DROP CONSTRAINT 句
 ALTER TABLE, 11-57
 DROP CONTEXT 文, 16-66
 DROP DATABASE LINK 文, 16-67
 DROP DIMENSION 文, 16-69
 DROP DIRECTORY 文, 16-71
 DROP FUNCTION 文, 16-72
 DROP INDEXTYPE 文, 16-76
 DROP INDEX 文, 16-74
 DROP JAVA 文, 16-78
 DROP LIBRARY 文, 16-80
 DROP LOG GROUP 句
 ALTER TABLE, 11-35
 DROP LOGFILE MEMBER 句
 ALTER DATABASE, 9-20, 9-40
 DROP LOGFILE 句
 ALTER DATABASE, 9-20, 9-39
 DROP MATERIALIZED VIEW LOG 文, 16-83
 DROP MATERIALIZED VIEW 文, 16-81
 DROP OPERATOR 文, 16-85
 DROP OUTLINE 文, 16-87
 DROP PACKAGE BODY 文, 16-88
 DROP PACKAGE 文, 16-88
 DROP PARTITION 句
 ALTER INDEX, 9-66, 9-79
 ALTER TABLE, 11-70
 DROP PRIMARY 制約句
 ALTER TABLE, 11-57
 DROP PROCEDURE 文, 16-90
 DROP PROFILE システム権限, 17-38
 CREATE PROFILE 文, 16-92
 DROP PUBLIC DATABASE LINK システム権限,
 17-35
 DROP PUBLIC SYNONYM システム権限, 17-39
 DROP ROLE 文, 16-94
 DROP ROLLBACK SEGMENT システム権限, 17-39
 DROP ROLLBACK SEGMENT 文, 16-95
 DROP SEQUENCE 文, 17-2
 DROP SUPPLEMENTAL LOG DATA 句
 ALTER DATABASE, 9-41
 DROP SYNONYM 文, 17-4
 DROP TABLESPACE システム権限, 17-40
 DROP TABLESPACE 文, 17-9
 DROP TABLE 文, 17-6

- DROP TRIGGER 文, 17-12
- DROP TYPE BODY 文, 17-17
- DROP TYPE 文, 17-14
- DROP UNIQUE 制約句
 - ALTER TABLE, 11-57
- DROP USER システム権限, 17-42
- DROP USER 文, 17-19
- DROP VALUES 句
 - ALTER TABLEMODIFY PARTITION, 11-62
- DROP VIEW 文, 17-21
- DROP 句
 - ALTER DIMENSION, 9-58
 - ALTER INDEXTYPE, 9-84
- DROP 制約句
 - ALTER VIEW, 12-31
- DROP 文
 - トリガー, 15-94
- DRS_START 初期化パラメータ
 - ALTER SYSTEM で設定, 10-56
- DUAL ダミー表, 2-106, 8-14
- DUMP ファンクション, 6-55
- DYNAMIC_SAMPLING ヒント, 2-90
- DY 日時書式要素, 2-65, 2-69

E

E

- 数値書式要素, 2-61
- EBCDIC キャラクタ・セット, 2-44
- EE 日時書式要素, 2-65
- EMPTY_BLOB ファンクション, 6-56
- EMPTY_CLOB ファンクション, 6-56
- ENABLE ALL TRIGGERS 句
 - ALTER TABLE, 11-85
- ENABLE DISTRIBUTED RECOVERY 句
 - ALTER SYSTEM, 10-28
- ENABLE NOVALIDATE 制約状態, 7-20, 15-53
- ENABLE PARALLEL DML 句
 - ALTER SESSION, 10-4
- ENABLE QUERY REWRITE 句
 - ALTER MATERIALIZED VIEW, 9-103
 - CREATE MATERIALIZED VIEW, 14-23
- ENABLE RESTRICTED SESSION 句
 - ALTER SYSTEM, 10-28
- ENABLE RESUMABLE 句
 - ALTER SESSION, 10-6

- ENABLE ROW MOVEMENT 句
 - ALTER TABLE, 11-6, 11-39
 - CREATE TABLE, 15-15, 15-57
- ENABLE STORAGE IN ROW 句
 - ALTER TABLE, 11-45
 - CREATE TABLE, 15-36
- ENABLE TABLE LOCK 句
 - ALTER TABLE, 11-85
- ENABLE THREAD 句
 - ALTER DATABASE, 9-50
- ENABLE VALIDATE 制約状態, 7-19, 15-53
- ENABLE 句
 - ALTER INDEX, 9-75
 - ALTER TRIGGER, 12-3
 - CREATE TABLE, 15-52
- END BACKUP 句
 - ALTER DATABASE ... DATAFILE, 9-36
 - ALTER TABLESPACE, 11-102, 11-103
- ENQUEUE_RESOURCES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-56
- ERROR_ON_OVERLAP_TIME セッション・パラメータ, 10-12
- ESTIMATE STATISTICS 句
 - ANALYZE, 12-39
- EVENT 初期化パラメータ
 - ALTER SYSTEM で設定, 10-57
- EXCEPTIONS INTO 句
 - ALTER TABLE, 11-79
 - 制限事項, 11-79
- EXCHANGE PARTITION 句
 - ALTER TABLE, 11-25, 11-77
- EXCHANGE SUBPARTITION 句
 - ALTER TABLE, 11-25, 11-77
- EXCLUDING NEW VALUES 句
 - ALTER MATERIALIZED VIEW LOG, 9-113
 - CREATE MATERIALIZED VIEW LOG, 14-38
- EXCLUSIVE ロック・モード, 17-73
- EXECUTE ANY INDEXTYPE システム権限, 17-36
- EXECUTE ANY OPERATOR システム権限, 17-38
- EXECUTE ANY PROCEDURE システム権限, 17-38
- EXECUTE ANY TYPE システム権限, 17-41
- EXECUTE_CATALOG_ROLE ロール, 17-44
- EXECUTE オブジェクト権限, 17-45
 - 演算子, 17-48
 - オブジェクト型, 17-48
 - 索引タイプ, 17-48

- ファンクション、プロシージャまたはパッケージ,
17-47
- ライブラリ, 17-48
- EXEMPT ACCESS POLICY システム権限, 17-43
- EXISTSNode ファンクション, 6-57
- EXISTS 条件, 5-13, 5-14
- EXP_FULL_DATABASE ロール, 17-44
- EXPLAIN PLAN 文, 17-23
- EXP ファンクション, 6-58
- EXTENT MANAGEMENT DICTIONARY 句
 - CREATE TABLESPACE, 15-82
- EXTENT MANAGEMENT LOCAL 句
 - CREATE DATABASE, 13-30
 - CREATE TABLESPACE, 15-81
 - CREATE TEMPORARY TABLESPACE, 15-88
- EXTENT MANAGEMENT 句
 - CREATE DATABASE, 13-25
 - CREATE TABLESPACE, 15-77, 15-81
 - CREATE TEMPORARY TABLESPACE, 15-87
 - 一時表領域, 15-88
- EXTRACTVALUE ファンクション, 6-61
- EXTRACT (XML) ファンクション, 6-60
- EXTRACT (日時) ファンクション, 6-58
- E 日時書式要素, 2-65

F

- FAILED_LOGIN_ATTEMPTS パラメータ
 - ALTER PROFILE, 9-126
 - CREATE PROFILE, 14-70
- FAL_CLIENT 初期化パラメータ
 - ALTER SYSTEM で設定, 10-57
- FAL_SERVER 初期化パラメータ
 - ALTER SYSTEM で設定, 10-57
- FAST_START_IO_TARGET 初期化パラメータ
 - ALTER SESSION で設定, 10-58
- FAST_START_MTTR_TARGET 初期化パラメータ
 - ALTER SYSTEM で設定, 10-58
- FAST_START_PARALLEL_ROLLBACK 初期化パラメータ
 - ALTER SYSTEM で設定, 10-59
- FF 日時書式要素, 2-65
- FILE_MAPPING 初期化パラメータ
 - ALTER SYSTEM で設定, 10-59
- FILESYSTEMIO_OPTIONS 初期化パラメータ
 - ALTER SYSTEM で使用, 10-59

- FINAL 句
 - CREATE TYPE, 16-13, 16-14
- FIPS
 - 準拠, B-10
 - フラグ付け, 10-12
- FIRST_ROWS(n) ヒント, 2-91
- FIRST_VALUE ファンクション, 6-64
- FIRST ファンクション, 6-62
- FIXED_DATE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-60
- FLAGGER セッション・パラメータ, 10-12
- FLASHBACK ANY TABLE システム権限, 17-37, 17-40, 17-42
- FLASHBACK オブジェクト権限, 17-45
- FLOAT データ型, 2-13
 - DB2, 2-35
 - SQL/DS, 2-35
- FLOAT データ型 (ANSI), 2-34
- FLOOR ファンクション, 6-66
- FLUSH SHARED POOL 句
 - ALTER SYSTEM, 10-29
- FM 書式モデル修飾子, 2-71
- FM 数値書式要素, 2-61
- FOR EACH ROW 句
 - CREATE TRIGGER, 15-98
- FOR UPDATE 句
 - CREATE MATERIALIZED VIEW, 14-23
 - SELECT, 18-10, 18-25
- FORCE ANY TRANSACTION システム権限, 17-43
- FORCE LOGGING 句
 - ALTER DATABASE, 9-38
 - ALTER TABLESPACE, 11-104
 - CREATE CONTROLFILE, 13-19
 - CREATE DATABASE, 13-28
 - CREATE TABLESPACE, 15-80
- FORCE PARALLEL DML 句
 - ALTER SESSION, 10-4
- FORCE TRANSACTION システム権限, 17-43
- FORCE_UNION_REWRITE ヒント, 2-90
- FORCE 句
 - COMMIT, 12-71
 - CREATE VIEW, 16-42
 - DISASSOCIATE STATISTICS, 16-63
 - DROP INDEX, 16-75
 - DROP INDEXTYPE, 16-77
 - DROP OPERATOR, 16-86
 - DROP TYPE, 17-15

- REVOKE, 17-91
- ROLLBACK, 17-98
- FOR 句
 - ANALYZE COMPUTE STATISTICS, 12-37
 - ANALYZE ESTIMATE STATISTICS, 12-37
 - CREATE INDEX TYPE, 13-89
 - EXPLAIN PLAN, 17-25
- FREELIST GROUPS パラメータ
 - STORAGE 句, 7-56
- FREELISTS パラメータ
 - STORAGE 句, 7-57
- FREEPOOLS パラメータ
 - LOB 記憶域, 15-38
- FROM COLUMNS 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM FUNCTIONS 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM INDEXES 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM INDEX TYPES 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM PACKAGES 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM TYPES 句
 - DISASSOCIATE STATISTICS, 16-62
- FROM_TZ ファンクション, 6-66
- FROM 句
 - 問合せ, 8-10
- FULL ヒント, 2-91
- FX 書式モデル修飾子, 2-72

G

- GC_FILES_TO_LOCKS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-60
- GLOBAL PARTITION BY RANGE 句
 - CREATE INDEX, 7-22, 13-65, 13-76, 15-56
- GLOBAL QUERY REWRITE システム権限, 17-37
- GLOBAL TEMPORARY 句
 - CREATE TABLE, 15-24
- GLOBAL_CONTEXT_POOL_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-61
- GLOBAL_NAMES 初期化パラメータ
 - ALTER SESSION で設定, 10-8
 - ALTER SYSTEM で設定, 10-61
- GRANT ANY OBJECT PRIVILEGE システム権限, 17-43

- GRANT ANY PRIVILEGE システム権限, 17-43
- GRANT ANY ROLE システム権限, 17-39
- GRANT CONNECT THROUGH 句
 - ALTER USER, 12-23, 12-25
- GRANT 句
 - ALTER USER, 12-25
- GRAPHIC データ型
 - DB2, 2-35
 - SQL/DS, 2-35
- GREATEST ファンクション, 6-67
- GROUP BY 句
 - CUBE 拡張, 18-22
 - ROLLUP 拡張, 18-21
 - SELECT および副問合せ, 18-9, 18-21
 - 重複するグループ化の識別, 6-67
- GROUP_ID ファンクション, 6-67
- GROUPING SETS 句
 - SELECT および副問合せ, 18-22
- GROUPING_ID ファンクション, 6-70
- GROUPING ファンクション, 6-69
- GUARD ALL 句
 - ALTER DATABASE, 9-51
- GUARD NONE 句
 - ALTER DATABASE, 9-51
- GUARD STANDBY 句
 - ALTER DATABASE, 9-51
- GUARD 句
 - ALTER DATABASE, 9-51
- G 数値書式要素, 2-61

H

- HASH IS 句
 - CREATE CLUSTER, 13-7
- HASH_AJ ヒント, 2-91
- HASH_AREA_SIZE 初期化パラメータ
 - ALTER SESSION で設定, 10-8
 - ALTER SYSTEM で設定, 10-61
- HASH_JOIN_ENABLED 初期化パラメータ
 - ALTER SESSION で設定, 10-8
 - ALTER SYSTEM で設定, 10-62
- HASHKEYS 句
 - CREATE CLUSTER, 13-6
- HASH ヒント, 2-91
- HAVING 条件
 - GROUP BY 句, 18-22
- HEXTORAW ファンクション, 6-71

HH12 日時書式要素, 2-65
HH24 日時書式要素, 2-65
HH 日時書式要素, 2-65
HI_SHARED_MEMORY_ADDRESS 初期化パラメータ
ALTER SYSTEM で設定, 10-62
HIERARCHY 句
CREATE DIMENSION, 13-42, 13-43
HS_ADMIN_ROLE ロール, 17-44
HS_AUTOREGISTER 初期化パラメータ
ALTER SYSTEM で設定, 10-62

I

IDENTIFIED BY 句
ALTER ROLE 「CREATE ROLE」を参照, 9-133
CREATE DATABASE LINK, 13-38
SET ROLE, 18-45
IDENTIFIED EXTERNALLY 句
ALTER ROLE 「CREATE ROLE」を参照
ALTER USER 「CREATE USER」を参照
CREATE ROLE, 14-74
CREATE USER, 16-33
IDENTIFIED GLOBALLY 句
ALTER ROLE 「CREATE ROLE」を参照, 9-133
ALTER USER, 12-24
CREATE ROLE, 14-74
CREATE USER, 16-34
IDLE_TIME パラメータ
ALTER PROFILE, 9-126
IFILE 初期化パラメータ
ALTER SYSTEM で設定, 10-63
IMMEDIATE 句
SET CONSTRAINTS, 18-42
IMP_FULL_DATABASE ロール, 17-44
IN OUT パラメータ
CREATE FUNCTION, 13-53
CREATE PROCEDURE, 14-63
INCLUDING CONTENTS 句
DROP TABLESPACE, 17-10
INCLUDING DATAFILES 句
ALTER DATABASE TEMPFILE DROP 句, 9-37
INCLUDING NEW VALUES 句
ALTER MATERIALIZED VIEW LOG, 9-113
CREATE MATERIALIZED VIEW LOG, 14-38
INCLUDING TABLES 句
DROP CLUSTER, 16-65

INCREMENT BY 句
ALTER SEQUENCE 「CREATE SEQUENCE」を参照
INCREMENT BY パラメータ
CREATE SEQUENCE, 14-84
INDEX_ASC ヒント, 2-92
INDEX_DESC ヒント, 2-92
INDEXTYPE 句
CREATE INDEX, 13-65, 13-79
INDEX オブジェクト権限, 17-45
表, 17-46
INDEX 句
ANALYZE, 12-34
CREATE CLUSTER, 13-6
INDEX ヒント, 2-92
INITCAP ファンクション, 6-72
INITIALIZED EXTERNALLY 句
CREATE CONTEXT, 13-12
INITIALIZED GLOBALLY 句
CREATE CONTEXT, 13-12
INITIALLY DEFERRED 句
制約, 7-19
INITIALLY IMMEDIATE 句
制約, 7-19
INITIAL パラメータ
STORAGE 句, 7-54
INITTRANS パラメータ
ALTER CLUSTER, 9-9
ALTER INDEX, 9-63, 9-69
ALTER MATERIALIZED VIEW LOG, 9-110
ALTER TABLE, 11-33
CREATE INDEX 「CREATE TABLE」を参照, 13-73
CREATE MATERIALIZED VIEW LOG 「CREATE TABLE」を参照
CREATE MATERIALIZED VIEW 「CREATE TABLE」を参照
CREATE TABLE, 7-49
INSERT ANY TABLE システム権限, 17-40
INSERT オブジェクト権限, 17-45
ビュー, 17-46
表, 17-46
INSERT 句
MERGE, 17-76
INSERT 文, 17-52
append, 2-89
トリガー, 15-93

INSTANCE_GROUPS 初期化パラメータ
 ALTER SYSTEM で設定, 10-63
 INSTANCE_NAME 初期化パラメータ
 ALTER SYSTEM で設定, 10-64
 INSTANCE_NUMBER 初期化パラメータ
 ALTER SYSTEM で設定, 10-64
 INSTANCE セッション・パラメータ, 10-13
 INSTANTIABLE 句
 CREATE TYPE, 16-13
 INSTEAD OF 句
 CREATE TRIGGER, 15-92
 INSTEAD OF トリガー, 15-92
 INSTR2 ファンクション, 6-73
 INSTR4 ファンクション, 6-73
 INSTRB ファンクション, 6-73
 INSTRC ファンクション, 6-73
 INSTR ファンクション, 6-73
 INTEGER データ型
 ANSI, 2-34
 DB2, 2-35
 SQL/DS, 2-35
 INTERSECT 集合演算子, 3-6, 18-23
 INTERVAL DAY TO SECOND データ型, 2-23
 INTERVAL YEAR TO MONTH データ型, 2-22
 INTO 句
 EXPLAIN PLAN, 17-24
 INSERT, 17-57
 INT データ型 (ANSI), 2-34
 INVALIDATE GLOBAL INDEXES 句
 ALTER TABLE, 11-81
 IN パラメータ
 CREATE PROCEDURE, 14-63
 CREATE FUNCTION, 13-53
 IN リスト, 2-100
 IS NOT NULL 演算子, 5-13
 IS NULL 演算子, 5-13
 IS OF type 条件, 5-19
 ISO 「ISO (国際標準化機構)」を参照, 1-2
 ISOLATION_LEVEL セッション・パラメータ, 10-13
 ISO (国際標準化機構), B-1
 規格, xv, 1-2, B-2
 IW 日時書式要素, 2-65
 IYYY 日時書式要素, 2-65
 IYY 日時書式要素, 2-65
 IY 日時書式要素, 2-65
 I 日時書式要素, 2-65

J

Java
 Java ソース・スキーマ・オブジェクト
 作成, 13-92
 記憶形式
 CustomDatum, 16-11
 SQLData, 16-11
 クラス
 削除, 16-78
 作成, 13-90, 13-92
 変換, 9-85, 13-92
 スキーマ・オブジェクト
 名前解決, 13-94
 ソース
 コンパイル, 9-85, 13-92
 削除, 16-78
 作成, 13-90
 メソッド
 戻り型, 16-14
 リソース
 削除, 16-78
 作成, 13-90, 13-92
 JAVA_MAX_SESSIONSPACE_LIMIT 初期化パラメータ
 ALTER SYSTEM で設定, 10-65
 JAVA_MAX_SESSIONSPACE_SIZE 初期化パラメータ
 ALTER SYSTEM で設定, 10-64
 JAVA_POOL_SIZE 初期化パラメータ
 ALTER SYSTEM で設定, 10-65
 JAVA 句
 CREATE TYPE, 16-15
 CREATE TYPE BODY, 16-29
 Java メソッド
 オブジェクト型へのマップ, 16-15
 JOB_QUEUE_PROCESSES 初期化パラメータ
 ALTER SYSTEM で設定, 10-66
 JOIN KEY 句
 ALTER DIMENSION, 9-57
 CREATE DIMENSION, 13-44
 JOIN 句
 CREATE DIMENSION, 13-42
 J 日時書式要素, 2-65

K

KILL SESSION 句
ALTER SYSTEM, 10-27

L

LAG ファンクション, 6-75
LANGUAGE 句
CREATE PROCEDURE, 14-64
CREATE TYPE, 16-15
CREATE TYPE BODY, 16-29
LARGE_POOL_SIZE 初期化パラメータ
ALTER SYSTEM で設定, 10-66
LAST_DAY ファンクション, 6-78
LAST_VALUE ファンクション, 6-79
LAST ファンクション, 6-76
LEAD ファンクション, 6-81
LEAST ファンクション, 6-82
LENGTH2 ファンクション, 6-82
LENGTH4 ファンクション, 6-82
LENGTHB ファンクション, 6-82
LENGTHC ファンクション, 6-82
LENGTH ファンクション, 6-82
LEVEL 疑似列, 2-81, 18-20
階層問合せ, 2-81
LEVEL 句
ALTER DIMENSION, 9-57
CREATE DIMENSION, 13-42, 13-43
LICENSE_MAX_SESSIONS 初期化パラメータ
ALTER SYSTEM で設定, 10-67
LICENSE_SESSIONS_WARNING 初期化パラメータ
ALTER SYSTEM で設定, 10-68
LIKE 条件, 5-14
LIST CHAINED ROWS 句
ANALYZE, 12-42
LN ファンクション, 6-83
LOB
CACHE READS 設定, 2-28
外部, 2-26
記憶域
インライン, 15-36
属性, 15-36
特性, 7-50
キャッシュへの値の保存, 11-45, 15-50
旧バージョンの保存, 15-37
索引, 15-38

処理されるバイト数, 15-37
属性の初期化, 2-27
ディレクトリの指定, 13-46
内部, 2-26
表領域
定義, 15-28
物理属性の変更, 11-55
列
LONG と LONG RAW の違い, 2-26
移入, 2-27
ロギング属性, 15-28
ロケータ, 2-26
LOB_storage_clause
ALTER MATERIALIZED VIEW, 9-91
LOB 記憶域句
ALTER MATERIALIZED VIEW, 9-97
ALTER TABLE, 11-14, 11-45
CREATE MATERIALIZED VIEW, 14-10, 14-12, 14-17
CREATE TABLE, 15-14, 15-36
パーティション, 11-47
LOB 索引句
ALTER TABLE, 11-47
CREATE TABLE, 15-38
LOB データ型, 2-26
LOB 列
LONG 列からの作成, 2-14, 11-49
記憶域の変更, 11-45
結合の制限, 8-10
制限, 2-27
追加, 11-42
プロパティの定義
マテリアライズド・ビュー, 14-10
変更, 11-48
マテリアライズド・ビューの記憶特性, 9-97
LOCAL_LISTENER 初期化パラメータ
ALTER SYSTEM で設定, 10-68
LOCALTIMESTAMP ファンクション, 6-84
LOCAL 句
CREATE INDEX, 13-65, 13-78
LOCK ANY TABLE システム権限, 17-40
LOCK TABLE 文, 17-71
LOCK_NAME_SPACE 初期化パラメータ
ALTER SYSTEM で設定, 10-68
LOCK_SGA 初期化パラメータ
ALTER SYSTEM で設定, 10-69

LOG_ARCHIVE_START パラメータ
 ALTER SYSTEM, 10-73
 LOG_ARCHIVE_DEST_n 初期化パラメータ
 ALTER SESSION で設定, 10-8, 10-70
 DELAY の設定の上書き, 9-31
 LOG_ARCHIVE_DEST_STATE_n 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-71
 LOG_ARCHIVE_DEST 初期化パラメータ
 ALTER SYSTEM で設定, 10-69
 LOG_ARCHIVE_DUPLEX_DEST 初期化パラメータ
 ALTER SYSTEM で設定, 10-71
 LOG_ARCHIVE_FORMAT 初期化パラメータ
 ALTER SYSTEM で設定, 10-72
 LOG_ARCHIVE_MAX_PROCESSES 初期化パラメータ
 ALTER SYSTEM で設定, 10-72
 LOG_ARCHIVE_MIN_SUCCEED_DEST 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-72
 LOG_ARCHIVE_TRACE 初期化パラメータ
 ALTER SYSTEM で設定, 10-73
 LOG_BUFFER 初期化パラメータ
 ALTER SYSTEM で設定, 10-73
 LOG_CHECKPOINT_INTERVAL 初期化パラメータ
 ALTER SYSTEM で設定, 10-74
 LOG_CHECKPOINT_TIMEOUT 初期化パラメータ
 ALTER SYSTEM で設定, 10-74
 LOG_CHECKPOINTS_TO_ALERT 初期化パラメータ
 ALTER SYSTEM で設定, 10-74
 log_file_clauses
 ALTER DATABASE, 9-20
 LOG_FILE_NAME_CONVERT 初期化パラメータ
 ALTER SYSTEM で設定, 10-75
 LOG_PARALLELISM 初期化パラメータ
 ALTER SYSTEM で設定, 10-76
 LOGFILE GROUP 句
 CREATE CONTROLFILE, 13-17
 LOGFILE 句
 CREATE DATABASE, 13-26
 LOGGING 句
 ALTER INDEX, 9-70
 ALTER INDEX ... REBUILD, 9-74
 ALTER MATERIALIZED VIEW, 9-98
 ALTER MATERIALIZED VIEW LOG, 9-112
 ALTER TABLE, 11-35
 ALTER TABLESPACE, 11-104
 CREATE MATERIALIZED VIEW, 14-15
 CREATE MATERIALIZED VIEW LOG, 14-35
 CREATE TABLE, 15-28
 CREATE TABLESPACE, 15-80
 LOGICAL_READS_PER_CALL パラメータ
 ALTER PROFILE, 9-126
 LOGICAL_READS_PER_SESSION パラメータ
 ALTER PROFILE, 9-126
 ALTER RESOURCE COST, 9-130
 LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータ
 ALTER SYSTEM で設定, 10-76
 LOGOFF データベース・イベント
 トリガー, 15-96
 LOGON データベース・イベント
 トリガー, 15-96
 LOG ファンクション, 6-85
 LONG RAW データ型, 2-25
 CHAR データからの変換, 2-26
 LONG VARCHAR データ型
 DB2, 2-35
 SQL/DS, 2-35
 LONG VARGRAPHIC データ型
 DB2, 2-35
 SQL/DS, 2-35
 LONG データ型, 2-14
 トリガー内, 2-15
 LONG 列
 LOB への変換, 2-14, 11-49
 参照先, 2-14
 制限, 2-14
 テキスト列の格納, 2-14
 ドメイン索引, 11-49
 ビュー定義の格納, 2-14
 LOWER ファンクション, 6-86
 LPAD ファンクション, 6-86
 LTRIM ファンクション, 6-87
 L 数値書式要素, 2-61

M

MAKE_REF ファンクション, 6-88
 MANAGE TABLESPACE システム権限, 17-40
 MANAGED STANDBY RECOVERY 句
 ALTER DATABASE, 9-30
 MAP MEMBER 句
 ALTER TYPE, 12-13

- CREATE TYPE, 16-29
- MAPPING TABLE 句
 - ALTER TABLE, 11-65, 11-83
- MAP メソッド
 - 型の定義, 16-17
 - 指定, 12-13
- MATCHES 条件, 5-2
- MAX_COMMIT_PROPAGATION_DELAY 初期化パラメータ
 - ALTER SYSTEM で設定, 10-76
- MAX_DISPATCHERS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-77
- MAX_DUMP_FILE_SIZE 初期化パラメータ
 - ALTER SESSION で設定, 10-8
 - ALTER SYSTEM で設定, 10-77
- MAX_ENABLED_ROLES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-77
- MAX_ROLLBACK_SEGMENTS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-78
- MAX_SHARED_SERVERS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-78
- MAXDATAFILES パラメータ
 - CREATE CONTROLFILE, 13-19
 - CREATE DATABASE, 13-28
- MAXEXTENTS パラメータ
 - STORAGE 句, 7-56
- MAXINSTANCES パラメータ
 - CREATE CONTROLFILE, 13-19
 - CREATE DATABASE, 13-28
- MAXLOGFILES パラメータ
 - CREATE CONTROLFILE, 13-18
 - CREATE DATABASE, 13-27
- MAXLOGHISTORY パラメータ
 - CREATE CONTROLFILE, 13-19
 - CREATE DATABASE, 13-27
- MAXLOGMEMBERS パラメータ
 - CREATE CONTROLFILE, 13-18
 - CREATE DATABASE, 13-27
- MAXSIZE 句
 - ALTER DATABASE, 9-19
- MAXTRANS パラメータ
 - ALTER CLUSTER, 9-9
 - ALTER INDEX, 9-63, 9-69
 - ALTER MATERIALIZED VIEW LOG, 9-110
 - ALTER TABLE, 11-33
 - CREATE INDEX「CREATE TABLE」を参照, 13-73

- CREATE MATERIALIZED VIEW LOG「CREATE TABLE」を参照, 14-35
- CREATE MATERIALIZED VIEW「CREATE TABLE」を参照, 14-15
- CREATE TABLE, 7-50
- MAXVALUE パラメータ
 - ALTER SEQUENCE「CREATE SEQUENCE」を参照, 9-138
 - CREATE SEQUENCE, 14-84
- MAX ファンクション, 6-89
- MEMBER 句
 - ALTER TYPE, 12-12
 - CREATE TYPE, 16-12
 - CREATE TYPE BODY, 16-28
- MERGE PARTITIONS 句
 - ALTER TABLE, 11-75
- MERGE_AJ ヒント, 2-91
- merge_insert_clause
 - MERGE, 17-77
- MERGE ヒント, 2-93
- MERGE 文, 17-75
- MIGRATE 句
 - ALTER DATABASE, 9-25
- MINEXTENTS パラメータ
 - STORAGE 句, 7-55
- MINIMIZE RECORDS PER BLOCK 句
 - ALTER TABLE, 11-38
- MINIMUM EXTENT 句
 - ALTER TABLESPACE, 11-101
 - CREATE TABLESPACE, 15-79
- MINUS 集合演算子, 3-6, 18-23
- MINVALUE パラメータ
 - ALTER SEQUENCE「CREATE SEQUENCE」を参照, 9-138
 - CREATE SEQUENCE, 14-84
- MIN ファンクション, 6-91
- MI 数値書式要素, 2-61
- MI 日時書式要素, 2-65
- MM 日時書式要素, 2-65
- MODE 句
 - LOCK TABLE, 17-73
- MODIFY CONSTRAINT 句
 - ALTER TABLE, 11-11, 11-56
 - ALTER VIEW, 12-31
- MODIFY DEFAULT ATTRIBUTES 句
 - ALTER INDEX, 9-65, 9-78
 - ALTER TABLE, 11-59

MODIFY LOB 記憶域句
 ALTER MATERIALIZED VIEW, 9-92, 9-97
 ALTER TABLE, 11-55
 MODIFY LOB 句
 ALTER TABLE, 11-55
 MODIFY NESTED TABLE 句
 ALTER TABLE, 11-11, 11-55
 MODIFY PARTITION 句
 ALTER INDEX, 9-66, 9-78
 ALTER MATERIALIZED VIEW, 9-98
 ALTER TABLE, 11-60
 MODIFY scoped_table_ref_constraint 句
 ALTER MATERIALIZED VIEW, 9-99
 MODIFY SUBPARTITION 句
 ALTER INDEX, 9-67, 9-80
 ALTER TABLE, 11-63
 MODIFY VARRAY 句
 ALTER TABLE, 11-16, 11-56
 MODIFY 句
 ALTER TABLE, 11-48
 MOD ファンクション, 6-92
 MONITORING USAGE 句
 ALTER INDEX, 9-76
 MONITORING 句
 ALTER TABLE, 11-37
 CREATE TABLE, 15-52
 MONTHS_BETWEEN ファンクション, 6-93
 MONTH 日時書式要素, 2-65, 2-69
 MON 日時書式要素, 2-65, 2-69
 MOUNT 句
 ALTER DATABASE, 9-24
 MOVE ONLINE 句
 ALTER TABLE, 11-83
 MOVE PARTITION 句
 ALTER TABLE, 11-64
 MOVE SUBPARTITION 句
 ALTER TABLE, 11-65
 MOVE 句
 ALTER TABLE, 11-30, 11-82
 MTS. 「共有サーバー」を参照
 MULTISSET パラメータ
 CAST ファンクション, 6-25

N

NAMED 句
 CREATE JAVA, 13-92
 NAME 句
 SET TRANSACTION, 18-49
 NATIONAL CHAR VARYING データ型 (ANSI), 2-34
 NATIONAL CHARACTER SET パラメータ
 ALTER DATABASE, 9-47
 CREATE DATABASE, 13-29
 NATIONAL CHARACTER VARYING データ型 (ANSI), 2-34
 NATIONAL CHARACTER データ型 (ANSI), 2-34
 NATIONAL CHAR データ型 (ANSI), 2-34
 NCHAR VARYING データ型 (ANSI), 2-34
 NCHAR データ型, 2-10
 ANSI, 2-34
 NCHR ファンクション, 6-94
 NCLOB データ型, 2-31
 トランザクションのサポート, 2-31
 NESTED TABLE 句
 ALTER TABLE, 11-12, 11-44
 CREATE TABLE, 15-13, 15-40
 CREATE TRIGGER, 15-97
 NEW_TIME ファンクション, 6-94
 NEXT_DAY ファンクション, 6-96
 NEXTVAL 疑似列, 2-78, 14-82
 NEXT 句
 ALTER MATERIALIZED VIEW...REFRESH, 9-102
 NEXT パラメータ
 STORAGE 句, 7-54
 NL_SJ ヒント, 2-91
 NLS_CALENDAR 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-78
 NLS_CHARSET_DECL_LEN ファンクション, 6-96
 NLS_CHARSET_ID ファンクション, 6-97
 NLS_CHARSET_NAME ファンクション, 6-98
 NLS_COMP 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-79
 NLS_CURRENCY 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-79
 NLS_DATE_FORMAT 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-80
 NLS_DATE_LANGUAGE 初期化パラメータ, 2-69
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-80

NLS_DUAL_CURRENCY 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-80
 NLS_INITCAP ファンクション, 6-98
 NLS_ISO_CURRENCY 初期化パラメータ
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-81
 NLS_LANGUAGE 初期化パラメータ, 2-69, 8-9
 ALTER SESSION で設定, 10-8
 ALTER SYSTEM で設定, 10-81
 NLS_LENGTH_SEMANTICS 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-81
 変更, 2-10
 NLS_LOWER ファンクション, 6-100
 NLS_NCHAR_CONV_EXCP 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-82
 NLS_NUMERIC_CHARACTERS 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-82
 NLS_SORT 初期化パラメータ, 8-9
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-82
 NLS_TERRITORY 初期化パラメータ, 2-69
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-83
 NLS_TIMESTAMP_FORMAT 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-83
 NLS_TIMESTAMP_TZ_FORMAT 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-83
 NLS_UPPER ファンクション, 6-102
 NLSSORT ファンクション, 6-100
 NO FORCE LOGGING 句
 ALTER DATABASE, 9-38
 ALTER TABLESPACE, 11-104
 NO_EXPAND ヒント, 2-94
 NO_INDEX ヒント, 2-95
 NO_MERGE ヒント, 2-95
 NO_PUSH_PRED ヒント, 2-95
 NOAPPEND ヒント, 2-94
 NOARCHIVELOG 句
 ALTER DATABASE, 9-20, 9-38
 CREATE CONTROLFILE, 13-19
 CREATE DATABASE, 9-26, 13-28
 NOAUDIT 文, 17-79
 NOCACHE 句
 ALTER CLUSTER, 9-10
 ALTER MATERIALIZED VIEW, 9-99
 ALTER MATERIALIZED VIEW LOG, 9-112
 ALTER SEQUENCE 「CREATE SEQUENCE」を参照
 ALTER TABLE, 11-36, 15-50
 CREATE CLUSTER, 13-9
 CREATE MATERIALIZED VIEW, 14-17
 CREATE MATERIALIZED VIEW LOG, 14-35
 CREATE SEQUENCE, 14-85
 NOCACHE ヒント, 2-94
 NOCOMPRESS 句
 ALTER INDEX ... REBUILD, 9-73
 CREATE INDEX, 13-74
 CREATE TABLE, 15-31
 NOCOPY 句
 CREATE FUNCTION, 13-53
 CREATE PROCEDURE, 14-63
 NOFORCE 句
 CREATE JAVA, 13-92
 CREATE VIEW, 16-42
 NOLOGGING モード
 強制ロギング・モード, 7-43
 パーティション・オブジェクト, 7-43
 非パーティション・オブジェクト, 7-43
 NOMAXVALUE パラメータ
 ALTER SEQUENCE 「CREATE SEQUENCE」を参照
 CREATE SEQUENCE, 14-84
 NOMINIMIZE RECORDS PER BLOCK 句
 ALTER TABLE, 11-38
 NOMINVALUE パラメータ
 ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138
 CREATE SEQUENCE, 14-84
 NOMONITORING USAGE 句
 ALTER INDEX, 9-76
 NOMONITORING 句
 ALTER TABLE, 11-37
 CREATE TABLE, 15-52
 NONE 句
 SET ROLE, 18-45
 NOORDER パラメータ
 ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138

- CREATE SEQUENCE, 14-85
- NOPARALLEL_INDEX ヒント, 2-95
- NOPARALLEL 句
 - CREATE INDEX, 7-46, 9-11, 9-29, 9-69, 9-98, 9-111, 11-81, 13-8, 13-76, 14-18, 14-36, 15-51
- NOPARALLEL ヒント, 2-95
- NORELY 句
 - 制約, 7-21
- NORESETLOGS 句
 - CREATE CONTROLFILE, 13-18
- NOREVERSE パラメータ
 - ALTER INDEX ... REBUILD, 9-72
- NOREWRITE ヒント, 2-96
- NOROWDEPENDENCIES 句
 - CREATE CLUSTER, 13-8
 - CREATE TABLE, 15-52
- NOSORT 句
 - ALTER INDEX, 13-74
- NOT DEFERRABLE 句
 - 制約, 7-18
- NOT FINAL 句
 - CREATE TYPE, 16-13
- NOT IDENTIFIED 句
 - ALTER ROLE 「CREATE ROLE」を参照
 - CREATE ROLE, 14-74
- NOT INSTANTIABLE 句
 - CREATE TYPE, 16-13, 16-14
- NOT NULL 句
 - CREATE TABLE, 15-26
- NOT 条件, 5-8, 5-9
- NOWAIT 句
 - LOCK TABLE, 17-73
- NTILE ファンクション, 6-103
- NULL, 2-76
 - 0 (ゼロ) との違い, 2-76
 - 条件, 2-77
 - 表, 2-77
 - 比較条件, 2-76
 - ファンクション, 2-76
- NULLIF ファンクション, 6-104
 - CASE 式の書式, 6-104
- NULL 条件, 5-13
- NUMBER データ型, 2-12
 - VARCHAR2 への変換, 2-60
 - 位取り, 2-12
 - 精度, 2-12

- NUMERIC データ型 (ANSI), 2-34
- NUMTODSINTERVAL ファンクション, 6-105
- NUMTOYMINTERVAL ファンクション, 6-106
- NVARCHAR2 データ型, 2-11
- NVL2 ファンクション, 6-108
- NVL ファンクション, 6-107

O

- O7_DICTIONARY_ACCESSIBILITY 初期化パラメータ
 - ALTER SYSTEM で設定, 10-84
- OBJECT IDENTIFIER 句
 - CREATE TABLE, 15-60
- OBJECT_CACHE_MAX_SIZE_PERCENT 初期化パラメータ
 - ALTER SESSION で設定, 10-9
 - ALTER SYSTEM で設定, 10-84
- OBJECT_CACHE_OPTIMAL_SIZE 初期化パラメータ
 - ALTER SESSION で設定, 10-9
 - ALTER SYSTEM で設定, 10-84
- OFFLINE 句
 - ALTER ROLLBACK SEGMENT, 9-135
 - ALTER TABLESPACE, 11-101
 - CREATE TABLESPACE, 15-81
- OF 句
 - CREATE VIEW, 16-44
- OIDINDEX 句
 - CREATE TABLE, 15-60
- OID「オブジェクト識別子」を参照, 2-36
- OLAP_PAGE_POOL_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-85
- ON COMMIT REFRESH オブジェクト権限, 17-45
 - マテリアライズド・ビュー, 17-47
- ON COMMIT REFRESH システム権限, 17-37
- ON COMMIT 句
 - CREATE TABLE, 15-27
- ON DATABASE 句
 - CREATE TRIGGER, 15-97
- ON DEFAULT 句
 - AUDIT, 12-55
 - NOAUDIT, 17-82
- ON DELETE CASCADE 句
 - 制約, 7-14
- ON DELETE SET NULL 句
 - 制約, 7-14
- ON DIRECTORY 句
 - AUDIT, 12-55

- NOAUDIT, 17-82
- ON NESTED TABLE 句
 - CREATE TRIGGER, 15-97
- ON object 句
 - NOAUDIT, 17-82
 - REVOKE, 17-92
- ON PREBUILT TABLE
 - CREATE MATERIALIZED VIEW, 14-14
- ON SCHEMA 句
 - CREATE TRIGGER, 15-97
- ONLINE 句
 - ALTER ROLLBACK SEGMENT, 9-135
 - ALTER TABLESPACE, 11-101
 - CREATE INDEX, 13-75
 - CREATE TABLESPACE, 15-81
- ONLINE パラメータ
 - ALTER INDEX ... REBUILD, 9-73
- ON 句
 - CREATE OUTLINE, 14-46
- OPEN NORESETLOGS 句
 - ALTER DATABASE, 9-25
- OPEN READ ONLY 句
 - ALTER DATABASE, 9-25
- OPEN READ WRITE 句
 - ALTER DATABASE, 9-24
- OPEN RESETLOGS 句
 - ALTER DATABASE, 9-25
- OPEN_CURSORS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-85
- OPEN_LINKS_PER_INSTANCE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-86
- OPEN_LINKS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-85
- OPEN 句
 - ALTER DATABASE, 9-24
- OPTIMAL パラメータ
 - STORAGE 句, 7-57
- OPTIMIZER_DYNAMIC_SAMPLING 初期化パラメータ
 - ALTER SYSTEM で設定, 10-86
- OPTIMIZER_FEATURES_ENABLE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-86
- OPTIMIZER_INDEX_CACHING 初期化パラメータ
 - ALTER SESSION で設定, 10-9
 - ALTER SYSTEM で設定, 10-87
- OPTIMIZER_INDEX_COST_ADJ 初期化パラメータ
 - ALTER SESSION で設定, 10-9
- ALTER SYSTEM で設定, 10-87
- OPTIMIZER_MAX_PERMUTATIONS 初期化パラメータ
 - ALTER SESSION で設定, 10-9
 - ALTER SYSTEM で設定, 10-87
- OPTIMIZER_MODE 初期化パラメータ
 - ALTER SESSION で設定, 10-9
 - ALTER SYSTEM で設定, 10-88
- OR REPLACE 句
 - CREATE CONTEXT, 13-12
 - CREATE DIRECTORY, 13-47
 - CREATE FUNCTION, 13-52, 13-91
 - CREATE LIBRARY, 14-2
 - CREATE OUTLINE, 14-45
 - CREATE PACKAGE, 14-49
 - CREATE PACKAGE BODY, 14-54
 - CREATE PROCEDURE, 14-62
 - CREATE TRIGGER, 15-91
 - CREATE TYPE, 16-9
 - CREATE TYPE BODY, 16-27
 - CREATE VIEW, 16-41
- ORACLE_TRACE_COLLECTION_NAME 初期化パラメータ
 - ALTER SYSTEM で設定, 10-88
- ORACLE_TRACE_COLLECTION_PATH 初期化パラメータ
 - ALTER SYSTEM で設定, 10-88
- ORACLE_TRACE_COLLECTION_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-89
- ORACLE_TRACE_ENABLE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-89
- ORACLE_TRACE_FACILITY_NAME 初期化パラメータ
 - ALTER SYSTEM で設定, 10-89
- ORACLE_TRACE_FACILITY_PATH 初期化パラメータ
 - ALTER SYSTEM で設定, 10-90
- Oracle9i Text
 - CATSEARCH, 5-2
 - CONTAINS, 5-2
 - MATCHES, 5-2
 - SCORE 演算子, 3-2
 - 組込み条件, 5-2
 - ドメイン索引の作成, 13-80
- Oracle のツール製品
 - SQL のサポート, 1-4

Oracle の予約語, C-1

ORDER BY 句
 SELECT, 8-9, 18-10, 18-23
 ROWNUM, 2-83
 問合せ, 8-9

ORDER MEMBER 句
 ALTER TYPE, 12-13
 CREATE TYPE BODY, 16-29

ORDER SIBLINGS BY 句
 SELECT, 18-24

ORDERED_PREDICATES ヒント, 2-96

ORDERED ヒント, 2-96

ORDER 句
 ALTER SEQUENCE 「CREATE SEQUENCE」を参照, 9-138

ORDER パラメータ
 CREATE SEQUENCE, 14-85

ORDER メソッド
 型の定義, 16-17
 指定, 12-13

ORDSYS.ORDAudio メディア型, 2-41

ORDSYS.ORDDoc メディア型, 2-41

ORDSYS.ORDImage メディア型, 2-41

ORDSYS.ORDVideo メディア型, 2-41

ORGANIZATION EXTERNAL 句
 CREATE TABLE, 15-30, 15-32

ORGANIZATION HEAP 句
 CREATE TABLE, 15-30

ORGANIZATION INDEX 句
 CREATE TABLE, 15-30

OR 条件, 5-8, 5-9

OS_AUTHENT_PREFIX 初期化パラメータ
 ALTER SYSTEM で設定, 10-90

OS_ROLES 初期化パラメータ
 ALTER SYSTEM で設定, 10-91

out-of-line 制約
 CREATE TABLE, 15-26

OUT パラメータ
 CREATE FUNCTION, 13-53
 CREATE PROCEDURE, 14-63

OVERFLOW 句
 ALTER INDEX, 9-67
 ALTER TABLE, 11-40
 CREATE TABLE, 15-32

OVERRIDING 句
 ALTER TYPE, 12-11
 CREATE TYPE, 16-14

OVER 句
 分析ファンクション, 6-9, 6-11

P

P.M. 日時書式要素, 2-65, 2-69

PARALLEL_ADAPTIVE_MULTI_USER 初期化パラメータ
 ALTER SYSTEM で設定, 10-91

PARALLEL_AUTOMATIC_TUNING 初期化パラメータ
 ALTER SYSTEM で設定, 10-91

PARALLEL_ENABLE 句
 CREATE FUNCTION, 13-56

PARALLEL_EXECUTION_MESSAGE_SIZE 初期化パラメータ
 ALTER SYSTEM で設定, 10-92

PARALLEL_INSTANCE_GROUP 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-92

PARALLEL_MAX_SERVERS 初期化パラメータ
 ALTER SYSTEM で設定, 10-92, 10-93

PARALLEL_MIN_PERCENT 初期化パラメータ
 ALTER SESSION で設定, 10-9
 ALTER SYSTEM で設定, 10-93

PARALLEL_THREADS_PER_CPU 初期化パラメータ
 ALTER SYSTEM で設定, 10-94

PARALLEL 句
 ALTER CLUSTER, 9-8, 9-10
 ALTER DATABASE, 9-29
 ALTER INDEX, 9-63, 9-69
 ALTER MATERIALIZED VIEW, 9-93, 9-98
 ALTER MATERIALIZED VIEW LOG, 9-110, 9-111
 ALTER TABLE, 11-81
 CREATE CLUSTER, 13-7
 CREATE INDEX, 13-76
 CREATE MATERIALIZED VIEW, 14-13, 14-17
 CREATE MATERIALIZED VIEW LOG, 14-34, 14-36
 CREATE TABLE, 15-22, 15-50

PARALLEL ヒント, 2-97

PARAMETERS 句
 ALTER INDEX ... REBUILD, 9-74
 CREATE INDEX, 13-80

PARTITION ... LOB 記憶域句
 ALTER TABLE, 11-47

PARTITION BY HASH 句
 CREATE TABLE, 15-44

PARTITION BY LIST 句
 CREATE TABLE, 15-46

PARTITION BY RANGE 句
 CREATE TABLE, 15-18, 15-42

partition_storage_clause
 ALTER TABLE, 11-15

PARTITION_VIEW_ENABLED 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-94

PARTITION 句
 ANALYZE, 12-36
 CREATE INDEX, 7-23, 13-77, 15-57
 CREATE TABLE, 15-43
 DELETE, 16-56
 INSERT, 17-58
 LOCK TABLE, 17-72
 UPDATE, 18-60

PASSWORD EXPIRE 句
 ALTER USER 「CREATE USER」を参照
 CREATE USER, 16-35

PASSWORD_GRACE_TIME パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PASSWORD_LIFE_TIME パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PASSWORD_LOCK_TIME パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PASSWORD_REUSE_MAX パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PASSWORD_REUSE_TIME パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PASSWORD_VERIFY_FUNCTION パラメータ
 ALTER PROFILE, 9-126
 CREATE PROFILE, 14-70

PATH_VIEW, 5-13, 5-20

PCT_ACCESS_DIRECT 統計情報
 索引構成表, 12-35

PCTFREE パラメータ
 ALTER CLUSTER, 9-9
 ALTER INDEX, 9-63, 9-69
 ALTER MATERIALIZED VIEW LOG, 9-110
 ALTER TABLE, 11-33
 CREATE MATERIALIZED VIEW LOG 「CREATE TABLE」を参照
 CREATE MATERIALIZED VIEW 「CREATE TABLE」を参照
 CREATE TABLE, 7-48

PCTINCREASE パラメータ
 STORAGE 句, 7-55

PCTTHRESHOLD パラメータ
 CREATE TABLE, 11-40, 15-31

PCTUSED パラメータ
 ALTER CLUSTER, 9-9
 ALTER INDEX, 9-63, 9-69
 ALTER MATERIALIZED VIEW LOG, 9-110
 ALTER TABLE, 11-33
 CREATE INDEX 「CREATE TABLE」を参照
 CREATE MATERIALIZED VIEW LOG 「CREATE TABLE」を参照, 14-35
 CREATE MATERIALIZED VIEW 「CREATE TABLE」を参照, 14-15
 CREATE TABLE, 7-49

PCTVERSION パラメータ
 LOB 記憶域, 15-37
 LOB 記憶域句, 11-46

PERCENT_RANK ファンクション, 6-110

PERCENTILE_CONT ファンクション, 6-112

PERCENTILE_DISC ファンクション, 6-114

PERMANENT 句
 ALTER TABLESPACE, 11-103
 CREATE TABLESPACE, 15-81

PGA_AGGREGATE_TARGET 初期化パラメータ
 ALTER SYSTEM で設定, 10-95

PIPELINED 句
 CREATE FUNCTION, 13-56

PL/SQL
 前回のリリースとの互換性, 10-97
 プログラム本体
 CREATE FUNCTION, 13-58

PLAN_TABLE サンプル表, 17-23

PLSQL_COMPILER_FLAGS 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-95

PLSQL_DEBUG セッション・パラメータ, 10-13

PLSQL_NATIVE_C_COMPILER 初期化パラメータ
 ALTER SYSTEM で設定, 10-95

PLSQL_NATIVE_LIBRARY_DIR 初期化パラメータ
 ALTER SYSTEM で設定, 10-96

PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT 初期化
パラメータ
ALTER SYSTEM で設定, 10-96
PLSQL_NATIVE_LINKER 初期化パラメータ
ALTER SYSTEM で設定, 10-96
PLSQL_NATIVE_MAKE_FILE_NAME 初期化パラメータ
ALTER SYSTEM で設定, 10-97
PLSQL_NATIVE_MAKE_UTILITY 初期化パラメータ
ALTER SYSTEM で設定, 10-97
PLSQL_V2_COMPATIBILITY 初期化パラメータ
ALTER SYSTEM で設定, 10-97
PM 日時書式要素, 2-65, 2-69
POWER ファンクション, 6-115
PQ_DISTRIBUTE ヒント, 2-98
PRAGMA RESTRICT_REFERENCES, 12-12
PRAGMA 句
ALTER TYPE, 12-12
CREATE TYPE, 16-8, 16-15
PRE_PAGE_SGA 初期化パラメータ
ALTER SYSTEM で設定, 10-98
PRIMARY KEY 句
CREATE TABLE, 15-26
制約, 7-12
PRIOR 句
階層問合せ, 8-3
PRIVATE_SGA パラメータ
ALTER PROFILE, 9-126
ALTER RESOURCE COST, 9-130
PRIVATE 句
CREATE OUTLINE, 14-45
PROCESSES 初期化パラメータ
ALTER SYSTEM で設定, 10-98
PROFILE 句
ALTER USER「CREATE USER」を参照, 12-23
CREATE USER, 16-35
PR 数値書式要素, 2-61
PUBLIC 句
CREATE OUTLINE, 14-45
CREATE ROLLBACK SEGMENT, 14-78
CREATE SYNONYM, 15-3
DROP DATABASE LINK, 16-67
PUSH_PRED ヒント, 2-98

Q

QUERY REWRITE オブジェクト権限, 17-45

マテリアライズド・ビュー, 17-47
QUERY REWRITE システム権限, 17-36, 17-37
QUERY_REWRITE_ENABLED 初期化パラメータ
ALTER SESSION で設定, 10-10
ALTER SYSTEM で設定, 10-98
QUERY_REWRITE_INTEGRITY 初期化パラメータ
ALTER SESSION で設定, 10-10
ALTER SYSTEM で設定, 10-99
QUIESCE RESTRICTED 句
ALTER SYSTEM, 10-30
QUOTA 句
ALTER USER「CREATE USER」を参照
CREATE USER, 16-35
Q 日時書式要素, 2-65

R

RANK ファンクション, 6-116
RATIO_TO_REPORT ファンクション, 6-118
RAWTOHEX ファンクション, 6-119
RAWTONHEX ファンクション, 6-119
RAW データ型, 2-25
CHAR データからの変換, 2-26
RDBMS_SERVER_DN 初期化パラメータ
ALTER SYSTEM で設定, 10-99
READ ONLY 句
ALTER TABLESPACE, 11-103
READ WRITE 句
ALTER TABLESPACE, 11-103
READ_ONLY_OPEN_DELAYED 初期化パラメータ
ALTER SYSTEM で設定, 10-99
READ オブジェクト権限, 17-45
マテリアライズド・ディレクトリ, 17-48
REAL データ型 (ANSI), 2-34
REBUILD PARTITION 句
ALTER INDEX, 9-72
REBUILD SUBPARTITION 句
ALTER INDEX, 9-72
REBUILD UNUSABLE LOCAL INDEXES 句
ALTER TABLE, 11-80
REBUILD 句
ALTER INDEX, 9-64, 9-71
ALTER MATERIALIZED VIEW, 9-100
ALTER OUTLINE, 9-117
RECOVER AUTOMATIC 句
ALTER DATABASE, 9-27

- RECOVER CANCEL 句
 - ALTER DATABASE, 9-14, 9-30
- RECOVER CONTINUE 句
 - ALTER DATABASE, 9-14, 9-30
- RECOVER DATABASE 句
 - ALTER DATABASE, 9-14, 9-27
- RECOVER DATAFILE 句
 - ALTER DATABASE, 9-14, 9-28
- RECOVER LOGFILE 句
 - ALTER DATABASE, 9-14, 9-29
- RECOVER MANAGED STANDBY DATABASE 句
 - ALTER DATABASE, 9-16
- RECOVER STANDBY DATAFILE 句
 - ALTER DATABASE, 9-28
- RECOVER STANDBY TABLESPACE 句
 - ALTER DATABASE, 9-28
- RECOVER TABLESPACE 句
 - ALTER DATABASE, 9-14, 9-28
- RECOVERABLE, 9-71, 15-29
 - 「LOGGING 句」を参照
- RECOVERY_CATALOG_OWNER ロール, 17-44
- RECOVERY_PARALLELISM 初期化パラメータ
 - ALTER SYSTEM で設定, 10-100
- RECOVER 句
 - ALTER DATABASE, 9-26
- REDO ログ, 9-24
 - アーカイブ位置, 10-25
 - アーカイブ・モードの指定, 13-28
 - クラスタ・データベースの指定スレッドの使用可能化, 9-50
 - クラスタ・データベースの指定スレッドの使用禁止化, 9-50
 - グループの切替え, 10-29
 - サイズ, 7-38
 - 再利用, 7-38
 - 削除, 9-38, 9-39
 - 指定, 7-36, 13-26
 - メディア・リカバリ用, 9-29
 - 自動アーカイブ, 10-23
 - 起動, 10-25
 - 停止, 10-25
 - 自動名前生成, 9-26, 9-27
 - 手動アーカイブ, 10-23
 - SCN, 10-23
 - グループ番号, 10-24
 - 現行, 10-24
 - 順序番号, 10-23
 - すべて, 10-25
 - 次, 10-25
 - 消去, 9-38
 - スレッド, 10-23
 - スレッドの使用可能化および使用禁止化, 9-38
 - 追加, 9-38, 9-39
 - 変更を削除する, 9-24
 - メンバー
 - 既存のグループへの追加, 9-40
 - 削除, 9-40
 - 名前の変更, 9-37
 - 論理スタンバイ・データベースへの適用, 9-46
- REDO ログ・ファイル
 - 指定, 7-36
 - 制御ファイルの指定, 13-16
- REDO 割当てラッチ
 - 高い競合の回避, 10-76
- REF, 2-36, 7-16
 - DANGLING, 12-40
 - OID のコンテナ, 2-36
 - 検証, 12-40
 - 更新, 12-40
- REFERENCES オブジェクト権限, 17-45
 - ビュー, 17-46
 - 表, 17-46
- REFERENCES 句
 - CREATE TABLE, 15-26
- REFERENCING 句
 - CREATE TRIGGER, 15-91, 15-98
- REFRESH COMPLETE 句
 - ALTER MATERIALIZED VIEW, 9-101
 - CREATE MATERIALIZED VIEW, 14-19
- REFRESH FAST 句
 - ALTER MATERIALIZED VIEW, 9-100
 - CREATE MATERIALIZED VIEW, 14-19
- REFRESH FORCE 句
 - ALTER MATERIALIZED VIEW, 9-101
 - CREATE MATERIALIZED VIEW, 14-19
- REFRESH ON COMMIT 句
 - ALTER MATERIALIZED VIEW, 9-101
 - CREATE MATERIALIZED VIEW, 14-19
- REFRESH ON DEMAND 句
 - ALTER MATERIALIZED VIEW, 9-101
 - CREATE MATERIALIZED VIEW, 14-19
- REFRESH 句
 - ALTER MATERIALIZED VIEW, 9-96, 9-100
 - CREATE MATERIALIZED VIEW, 14-9

REFTOHEX ファンクション, 6-121
 REF 制約, 7-16
 ALTER TABLE, 11-43
 マテリアライズド・ビューの有効範囲の定義, 9-95
 REF 表制約, 7-16
 CREATE TABLE, 15-26
 REF ファンクション, 6-120
 REF 列
 指定, 15-26
 範囲の最限定, 9-99
 表や列レベルからの指定, 15-26
 REGISTER LOGFILE 句
 ALTER DATABASE, 9-45
 REGISTER 句
 ALTER SYSTEM, 10-31
 REGR_AVGX ファンクション, 6-122
 REGR_AVGY ファンクション, 6-122
 REGR_COUNT ファンクション, 6-122
 REGR_INTERCEPT ファンクション, 6-122
 REGR_R2 ファンクション, 6-122
 REGR_SLOPE ファンクション, 6-122
 REGR_SXX ファンクション, 6-122
 REGR_SXY ファンクション, 6-122
 REGR_SYY ファンクション, 6-122
 RELY 句
 制約, 7-21
 REMOTE_ARCHIVE_ENABLE 初期化パラメータ
 ALTER SYSTEM で設定, 10-100
 REMOTE_DEPENDENCIES_MODE 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-100
 REMOTE_LISTENER 初期化パラメータ
 ALTER SYSTEM で設定, 10-101
 REMOTE_LOGIN_PASSWORDFILE 初期化パラメータ
 ALTER SYSTEM で設定, 10-101
 制御ファイル, 13-15
 データベース, 13-23
 REMOTE_OS_AUTHENT 初期化パラメータ
 ALTER SYSTEM で設定, 10-101
 REMOTE_OS_ROLES 初期化パラメータ
 ALTER SYSTEM で設定, 10-102
 RENAME CONSTRAINT 句
 ALTER TABLE, 11-57
 RENAME DATAFILE 句
 ALTER TABLESPACE, 11-99
 RENAME FILE 句
 ALTER DATABASE, 9-13, 9-37
 RENAME GLOBAL_NAME 句
 ALTER DATABASE, 9-50
 RENAME PARTITION 句
 ALTER INDEX, 9-66, 9-79
 ALTER TABLE, 11-71
 RENAME SUBPARTITION 句
 ALTER INDEX, 9-66, 9-79
 ALTER TABLE, 11-71
 RENAME 句
 ALTER INDEX, 9-76
 ALTER OUTLINE, 9-117
 ALTER TABLE, 11-38
 ALTER TRIGGER, 12-3
 RENAME 文, 17-84
 REPLACE AS OBJECT 句
 ALTER TYPE, 12-11
 REPLACE ファンクション, 6-129
 REPLICATION_DEPENDENCY_TRACKING 初期化パラメータ
 ALTER SYSTEM で設定, 10-102
 RESET COMPATIBILITY 句
 ALTER DATABASE, 9-49
 RESETLOGS パラメータ
 CREATE CONTROLFILE, 13-17
 RESOLVER 句
 ALTER JAVA CLASS, 9-86
 ALTER JAVA SOURCE, 9-86
 CREATE JAVA, 13-94
 RESOLVE 句
 ALTER JAVA CLASS, 9-86
 CREATE JAVA, 13-92
 RESOURCE_LIMIT 初期化パラメータ
 ALTER SYSTEM で設定, 10-102
 RESOURCE_MANAGER_PLAN 初期化パラメータ
 ALTER SYSTEM で設定, 10-103
 RESOURCE_VIEW, 5-13, 5-20
 RESOURCE ロール, 17-44
 RESTRICT_REFERENCES プラグラマ
 ALTER TYPE, 12-12
 RESTRICTED SESSION システム権限, 17-35, 17-39
 RESUMABLE システム権限, 17-43
 RESUME 句
 ALTER SYSTEM, 10-29
 RETENTION パラメータ
 LOB 記憶域, 15-37
 RETURNING 句
 DELETE, 16-59

- INSERT, 17-54, 17-61
- UPDATE, 18-58, 18-64
- RETURN 句
 - CREATE FUNCTION, 13-54
 - CREATE OPERATOR, 14-42
 - CREATE TYPE, 16-14
 - CREATE TYPE BODY, 16-30
- REUSE SETTINGS 句
 - ALTER FUNCTION, 9-60
 - ALTER PACKAGE, 9-120
 - ALTER PROCEDURE, 9-123
 - ALTER TRIGGER, 12-4
 - ALTER TYPE, 12-11
- REUSE 句
 - CREATE CONTROLFILE, 13-17
 - ファイル指定, 7-38
- REVERSE 句
 - CREATE INDEX, 13-74
- REVERSE パラメータ
 - ALTER INDEX ... REBUILD, 9-72
- REVOKE CONNECT THROUGH 句
 - ALTER USER, 12-23, 12-25
- REVOKE 句
 - ALTER USER, 12-25
- REVOKE 文, 17-86
- REWRITE ヒント, 2-99
- RM 日時書式要素, 2-65
- RNDS 属性
 - PRAGMA RESTRICT_REFERENCES, 16-16
- RNPS 属性
 - PRAGMA RESTRICT_REFERENCES, 16-16
- RN 数値書式要素, 2-61
- ROLLBACK_SEGMENTS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-103
- ROLLBACK 文, 17-97
- ROLLUP 句
 - SELECT 文, 18-21
- ROUND ファンクション
 - 書式モデル, 6-210
 - 数値ファンクション, 6-130
 - 日付ファンクション, 6-131
- ROW EXCLUSIVE ロック・モード, 17-73
- ROW SHARE ロック・モード, 17-73
- ROW_LOCKING 初期化パラメータ
 - ALTER SYSTEM で設定, 10-104
- ROW_NUMBER ファンクション, 6-131

- ROWDEPENDENCIES 句
 - CREATE CLUSTER, 13-8
 - CREATE TABLE, 15-52
- ROWID
 - ROW 部分, 2-32
 - 外部キー表, 2-33
 - 拡張, 2-32
 - 基本, 2-32
 - 直接利用できない, 2-32
 - 索引構成表, 2-33
 - 使用方法, 2-82
 - 制限, 2-32
 - 互換性と移行, 2-33
 - 説明, 2-31
 - 非物理, 2-33
 - ファイル部分, 2-32
 - ブロック部分, 2-32
- ROWIDTOCHAR ファンクション, 6-133
- ROWIDTONCHAR ファンクション, 6-133
- ROWID 疑似列, 2-31, 2-33, 2-82
- ROWID データ型, 2-31
- ROWID ヒント, 2-99
- ROWNUM 疑似列, 2-83
 - 使用方法, 2-84
- RPAD ファンクション, 6-134
- RRRR 日時書式要素, 2-65
- RR 日時書式要素, 2-65, 2-69
- RTRIM ファンクション, 6-135
- RULE ヒント, 2-99

S

- SAMPLE 句
 - SELECT, 18-15
 - SELECT および副問合せ, 18-7
- SAVEPOINT 文, 18-2
- SCC 日時書式要素, 2-65
- SCHEMA 句
 - CREATE JAVA, 13-93
- SCOPE FOR 句
 - ALTER MATERIALIZED VIEW, 9-95
 - CREATE MATERIALIZED VIEW, 14-14
- SCORE 演算子, 3-2
- SEGMENT MANAGEMENT FREELISTS 句
 - CREATE TABLESPACE, 15-83
- SEGMENT MANAGEMENT PAGETABLE 句
 - CREATE TABLESPACE, 15-83

- SELECT ANY DICTIONARY システム権限, 17-43
- SELECT ANY SEQUENCE システム権限, 17-39
- SELECT ANY TABLE システム権限, 17-40
- SELECT_CATALOG_ROLE ロール, 17-44
- SELECT オブジェクト権限, 17-45
 - 順序, 17-47
 - ビュー, 17-46
 - 表, 17-46
 - マテリアライズド・ビュー, 17-48
- SELECT 構文のリスト, 8-2
 - 順序付け, 8-9
- SELECT 文, 8-2, 18-4
- SERIAL_REUSE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-104
- SERVERERROR イベント
 - トリガー, 15-95, 15-96
- SERVICE_NAMES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-104
- SESSION_CACHED_CURSORS 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-105
- SESSION_MAX_OPEN_FILES 初期化パラメータ
 - ALTER SYSTEM で設定, 10-105
- SESSION_ROLES ビュー, 18-44
- SESSIONS_PER_USER パラメータ
 - ALTER PROFILE, 9-126
- SESSIONS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-105
- SESSIONTIMEZONE ファンクション, 6-135
- SET CONSTRAINT(S) 文, 18-42
- SET DANGLING TO NULL 句
 - ANALYZE, 12-40
- SET DATABASE 句
 - CREATE CONTROLFILE, 13-17
- SET ROLE 文, 18-44
- SET STANDBY DATABASE 句
 - ALTER DATABASE, 9-44
- SET STATEMENT_ID 句
 - EXPLAIN PLAN, 17-24
- SET TIME_ZONE 句
 - ALTER DATABASE, 9-23, 9-48
 - ALTER SESSION, 10-14
 - CREATE DATABASE, 13-25
- SET TRANSACTION 文, 18-47
- SET UNUSED 句
 - ALTER TABLE, 11-51

- SET 句
 - ALTER SESSION, 10-6
 - ALTER SYSTEM, 10-31
 - UPDATE, 18-62
- SGA_MAX_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-106
- SGA「システム・グローバル領域 (SGA)」を参照
- SHADOW_CORE_DUMP 初期化パラメータ
 - ALTER SYSTEM で設定, 10-106
- SHARE ROW EXCLUSIVE ロック・モード, 17-73
- SHARE UPDATE ロック・モード, 17-73
- SHARED_MEMORY_ADDRESS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-106
- SHARED_POOL_RESERVED_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-107
- SHARED_POOL_SIZE 初期化パラメータ
 - ALTER SYSTEM で設定, 10-107
- SHARED_SERVER_SESSIONS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-108
- SHARED_SERVERS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-108
- SHARED 句
 - CREATE DATABASE LINK, 13-36
- SHRINK 句
 - ALTER ROLLBACK SEGMENT, 9-136
- SHUTDOWN イベント
 - トリガー, 15-95
- SHUTDOWN 句
 - ALTER SYSTEM, 10-31
- SIGN ファンクション, 6-136
- SINGLE TABLE 句
 - CREATE CLUSTER, 13-6
- SINH ファンクション, 6-137
- SIN ファンクション, 6-137
- SIZE 句
 - ALTER CLUSTER, 9-9
 - CREATE CLUSTER, 13-5
 - ファイル指定, 7-38
- SKIP_UNUSABLE_INDEXES セッション・パラメータ, 10-14
- SMALLINT データ型
 - ANSI, 2-34
 - DB2, 2-35
 - SQL/DS, 2-35
- SNMPAGENT ロール, 17-44
- SOME 演算子, 5-5

SORT_AREA_RETAINED_SIZE 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-109
 SORT_AREA_SIZE 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-109
 SOUNDEX ファンクション, 6-138
 SPECIFICATION 句
 ALTER PACKAGE, 9-119
 SPFILE 初期化パラメータ
 ALTER SYSTEM で設定, 10-110
 SPLIT PARTITION 句
 ALTER INDEX, 9-66, 9-79
 ALTER TABLE, 11-72
 SPTH 日時書式要素の接尾辞, 2-71
 SP 日時書式要素の接尾辞, 2-71
 SQL:99 規格, 1-2
 SQL_TRACE 初期化パラメータ
 ALTER SYSTEM で設定, 10-110
 SQL_TRACE セッション・パラメータ, 10-14
 SQL92_SECURITY 初期化パラメータ
 ALTER SYSTEM で設定, 10-110
 SQLData Java 記憶形式, 16-11
 SQL/DS データ型, 2-34
 Oracle データ型への変換, 2-35
 暗黙的な変換, 2-35
 制限, 2-35
 SQLJ オブジェクト型
 Java クラスのマッピング, 16-12
 作成, 16-11
 SQL*Loader の挿入のログの記録, 9-69
 SQL ファンクション
 ABS, 6-15
 ACOS, 6-16
 ADD_MONTHS, 6-16
 ASCII, 6-17
 ASCIISTR, 6-18
 ASIN, 6-19
 ATAN, 6-19
 ATAN2, 6-20
 AVG, 6-20
 BFILENAME, 6-22
 BIN_TO_NUM, 6-23
 BITAND, 6-24
 CAST, 6-25
 CEIL, 6-28
 CHARTOROWID, 6-28
 CHR, 6-29
 COALESCE, 6-31
 COMPOSE, 6-32
 CONCAT, 6-33
 CONVERT, 6-34
 CORR, 6-35
 COS, 6-37
 COSH, 6-37
 COUNT, 6-38
 COVAR_POP, 6-40
 COVAR_SAMP, 6-42
 CUME_DIST, 6-44
 CURRENT_DATE, 6-46
 CURRENT_TIMESTAMP, 6-46
 DBTIMEZONE, 6-48
 DECOMPOSE, 6-50
 DENSE_RANK, 6-51
 DEREF, 6-53
 DUMP, 6-55
 EMPTY_BLOB, 6-56
 EMPTY_CLOB, 6-56
 EXISTSNode, 6-57
 EXP, 6-58
 EXTRACT (XML), 6-60
 EXTRACT (日時), 6-58
 FIRST, 6-62
 FIRST_VALUE, 6-64
 FLOOR, 6-66
 FROM_TZ, 6-66
 GREATEST, 6-67
 GROUP_ID, 6-67
 GROUPING, 6-69
 GROUPING_ID, 6-70
 HEXTORAW, 6-71
 EXTRACTXML, 6-61
 INITCAP, 6-72
 INSTR, 6-73
 INSTR2, 6-73
 INSTR4, 6-73
 INSTRB, 6-73
 INSTRC, 6-73
 LAG, 6-75
 LAST, 6-76
 LAST_DAY, 6-78
 LAST_VALUE, 6-79
 LEAD, 6-81
 LEAST, 6-82

LENGTH, 6-82
 LENGTH2, 6-82
 LENGTH4, 6-82
 LENGTHB, 6-82
 LENGTHC, 6-82
 LN, 6-83
 LOB 列への適用, 6-2
 LOCALTIMESTAMP, 6-84
 LOG, 6-85
 LOWER, 6-86
 LPAD, 6-86
 LTRIM, 6-87
 MAKE_REF, 6-88
 MAX, 6-89
 MIN, 6-91
 MOD, 6-92
 MONTHS_BETWEEN, 6-93
 NCHR, 6-94
 NEW_TIME, 6-94
 NEXT_DAY, 6-96
 NLS_CHARSET_DECL_LEN, 6-96
 NLS_CHARSET_ID, 6-97
 NLS_CHARSET_NAME, 6-98
 NLS_INITCAP, 6-98
 NLS_LOWER, 6-100
 NLS_UPPER, 6-102
 NLSSORT, 6-100
 NLV2, 6-108
 NTILE, 6-103
 NULLIF, 6-104
 NUMTODSINTERVAL, 6-105
 NUMTOYMINTERVAL, 6-106
 NVL, 6-107
 PERCENT_RANK, 6-110
 PERCENTILE_CONT, 6-112
 PERCENTILE_DISC, 6-114
 POWER, 6-115
 RANK, 6-116
 RATIO_TO_REPORT, 6-118
 RAWTOHEX, 6-119
 RAWTONHEX, 6-119
 REF, 6-120
 REFTOHEX, 6-121
 REGR_AVGX, 6-122
 REGR_AVGY, 6-122
 REGR_COUNT, 6-122
 REGR_INTERCEPT, 6-122
 REGR_R2, 6-122
 REGR_SLOPE, 6-122
 REGR_SXX, 6-122
 REGR_SXY, 6-122
 REGR_SYY, 6-122
 REPLACE, 6-129
 ROUND (数値), 6-130
 ROUND (日付), 6-131
 ROW_NUMBER, 6-131
 ROWIDTOCHAR, 6-133
 ROWIDTONCHAR, 6-133
 RPAD, 6-134
 RTRIM, 6-135
 SESSIONTIMEZONE, 6-135
 SIGN, 6-136
 SIN, 6-137
 SINH, 6-137
 SOUNDEX, 6-138
 SQRT, 6-139
 STDDEV, 6-140
 STDDEV_POP, 6-141
 STDDEV_SAMP, 6-142
 SUBSTR, 6-144
 SUBSTR2, 6-144
 SUBSTR4, 6-144
 SUBSTRB, 6-144
 SUBSTRC, 6-144
 SUM, 6-145
 SYS_CONNECT_BY_PATH, 6-147
 SYS_CONTEXT, 6-148
 SYS_DBURIGEN, 6-153
 SYS_EXTRACT_UTC, 6-154
 SYS_GUID, 6-154
 SYS_TYPEID, 6-155
 SYS_XMLAGG, 6-156
 SYS_XMLGEN, 6-157
 SYSDATE, 6-159
 SYSTIMESTAMP, 6-159
 TAN, 6-160
 TANH, 6-160
 TO_CHAR (数値), 6-164
 TO_CHAR (日時), 6-162
 TO_CHAR (文字), 6-161
 TO_CLOB, 6-166
 TO_DATE, 6-166
 TO_DSINTERVAL, 6-168
 TO_LOB, 6-169

- TO_MULTI_BYTE, 6-170
- TO_NCHAR (数値), 6-172
- TO_NCHAR (日時), 6-172
- TO_NCHAR (文字), 6-171
- TO_NCLOB, 6-173
- TO_NUMBER, 6-174
- TO_SINGLE_BYTE, 6-175
- TO_TIMESTAMP, 6-175
- TO_YMINTERVAL, 6-178
- TRANSLATE, 6-178
- TRANSLATE...USING, 6-180
- TREAT, 6-181
- TRIM, 6-183
- TRUNC (数値), 6-184
- TRUNC (日付), 6-185
- TZ_OFFSET, 6-185
- UID, 6-186
- UNISTR, 6-187
- UPDATEXML, 6-188
- UPPER, 6-189
- USER, 6-190
- USERENV, 6-190
- VALUE, 6-192
- VAR_POP, 6-193
- VAR_SAMP, 6-194
- VARIANCE, 6-196
- VSIZ, 6-197
- WIDTH_BUCKET, 6-198
- オブジェクト参照, 6-15
- 集計, 6-7
- 数値, 6-3
- 線形リグレッション, 6-122
- 単一行, 6-3
 - その他, 6-6
- 日付, 6-5
- 分析, 6-9
- 変換, 6-6
- 文字
 - 数値, 6-5
 - 文字値, 6-4
- SQL 文
 - DDL, 9-3
 - DML, 9-4
 - 監査
 - アクセス, 12-55
 - 正常終了, 12-56
 - セッション, 12-55
 - 停止, 17-79
 - プロキシ, 12-54
 - ユーザー, 12-54
 - 構成, 9-5
 - 再開可能な領域割当て, 10-6
 - システム制御, 9-5
 - 実行計画の判断, 17-23
 - 種類, 9-3
 - セッション制御, 9-5
 - セッションでの発生, 12-50
 - 停止およびコンパイル, 10-6
 - トランザクション制御, 9-4
 - 取消し, 17-97
 - ロールバック, 17-97
- SQRT ファンクション, 6-139
- SSSS 日時書式要素, 2-65
- SS 日時書式要素, 2-65
- STANDBY_ARCHIVE_DEST 初期化パラメータ
 - ALTER SYSTEM で設定, 10-111
- STANDBY_FILE_MANAGEMENT 初期化パラメータ
 - ALTER SYSTEM で設定, 10-111
- STAR_TRANSFORMATION_ENABLED 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-111
- STAR_TRANSFORMATION ヒント, 2-100
- START LOGICAL STANDBY APPLY 句
 - ALTER DATABASE, 9-46
- START WITH 句
 - ALTER MATERIALIZED VIEW...REFRESH, 9-102
 - SELECT および副問合せ, 18-8
 - 問合せおよび副問合せ, 18-20
- START WITH パラメータ
 - CREATE SEQUENCE, 14-84
- startup_clauses
 - ALTER DATABASE, 9-14
- STARTUP イベント
 - トリガー, 15-95
- STATIC 句
 - ALTER TYPE, 12-12
 - CREATE TYPE, 16-13
 - CREATE TYPE BODY, 16-28
- STATISTICS_LEVEL 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-112
- STDDEV_POP ファンクション, 6-141
- STDDEV_SAMP ファンクション, 6-142

STDDEV ファンクション, 6-140
 STOP LOGICAL STANDBY 句
 ALTER DATABASE, 9-47
 STORAGE IN ROW 句
 ALTER TABLE, 11-45
 STORAGE 句
 ALTER CLUSTER, 9-9
 ALTER INDEX, 9-63, 9-69
 ALTER MATERIALIZED VIEW LOG, 9-110
 ALTER ROLLBACK SEGMENT, 9-134, 9-136
 CREATE MATERIALIZED VIEW LOG, 14-34
 CREATE MATERIALIZED VIEW LOG 「CREATE
 TABLE」を参照
 CREATE MATERIALIZED VIEW 「CREATE
 TABLE」を参照
 CREATE ROLLBACK SEGMENT, 14-79
 CREATE TABLE, 7-50, 15-11
 CREATE TABLESPACE, 15-77
 STORE IN 句
 ALTER TABLE, 11-41, 15-45
 Structured Query Language (SQL)
 ファンクション, 6-2
 Oracle のツール製品のサポート, 1-4
 埋込み, 1-3
 キーワード, A-3
 規格, 1-2, B-1
 構文, 9-5, A-1
 説明, 1-2
 パラメータ, A-3
 文
 監査, 12-56
 コストの判断, 17-23
 SUBPARTITION BY HASH 句
 CREATE TABLE, 15-19, 15-48
 SUBPARTITION BY LIST 句
 CREATE TABLE, 15-48
 SUBPARTITION 句
 ANALYZE, 12-36
 DELETE, 16-56
 INSERT, 17-58
 LOCK TABLE, 17-72
 UPDATE, 18-60
 SUBSTR2 ファンクション, 6-144
 SUBSTR4 ファンクション, 6-144
 SUBSTRB ファンクション, 6-144
 SUBSTRC ファンクション, 6-144
 SUBSTR ファンクション, 6-144

SUM ファンクション, 6-145
 SUSPEND 句
 ALTER SYSTEM, 10-29
 SWITCH LOGFILE 句
 ALTER SYSTEM, 10-29
 SYEAR 日時書式要素, 2-65
 SYS_CONNECT_BY_PATH ファンクション, 6-147
 SYS_CONTEXT ファンクション, 6-148
 SYS_DBURIGEN ファンクション, 6-153
 SYS_EXTRACT_UTC ファンクション, 6-154
 SYS_GUID ファンクション, 6-154
 SYS_NC_ROWINFO\$ 疑似列, 15-59, 15-61, 16-44,
 16-47
 SYS_NC_ROWINFO\$ 列, 15-59, 15-61, 16-44, 16-47
 SYS_TYPEID ファンクション, 6-155
 SYS_XMLAGG ファンクション, 6-156
 SYS_XMLGEN ファンクション, 6-157
 SYSDATE ファンクション, 6-159
 SYSDBA システム権限, 17-43
 SYSOPER システム権限, 17-43
 SYSTEM 表領域
 ローカル管理, 13-30
 SYSTEM ユーザー
 パスワードの割当て, 13-26
 SYSTIMESTAMP ファンクション, 6-159
 SYS スキーマ
 格納された関数, 15-99
 格納されたデータベース・トリガー, 15-99
 監査, 10-35
 SYS ユーザー
 パスワードの割当て, 13-26
 SYYYY 日時書式要素, 2-65
 S 数値書式要素, 2-61

T

TABLESPACE 句
 ALTER INDEX ... REBUILD, 9-73
 CREATE CLUSTER, 13-6
 CREATE INDEX, 13-73
 CREATE MATERIALIZED VIEW, 14-15
 CREATE MATERIALIZED VIEW LOG, 14-35
 CREATE ROLLBACK SEGMENT, 14-78
 CREATE TABLE, 15-28
 TABLE 句
 ANALYZE, 12-35
 DELETE, 16-58

- INSERT, 17-59
- SELECT, 18-16
- TRUNCATE, 18-52
- UPDATE, 18-59, 18-61, 18-62
- TANH ファンクション, 6-160
- TAN ファンクション, 6-160
- TAPE_ASYNCH_IO 初期化パラメータ
 - ALTER SYSTEM で設定, 10-112
- TEMPFILE 句
 - ALTER DATABASE, 9-18, 9-36
 - CREATE TEMPORARY TABLESPACE, 15-87
- TEMPORARY TABLESPACE 句
 - ALTER USER 「CREATE USER」を参照, 12-23
 - CREATE USER, 16-35
- TEMPORARY 句
 - ALTER TABLESPACE, 11-103
 - CREATE TABLESPACE, 15-81
- TEST 句
 - ALTER DATABASE RECOVER, 9-29
- THREAD 初期化パラメータ
 - ALTER SYSTEM で設定, 10-112
- THSP 日時書式要素の接尾辞, 2-71
- TH 日時書式要素の接尾辞, 2-71
- TIME_ZONE セッション・パラメータ, 10-14
- TIMED_OS_STATISTICS 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-113
- TIMED_STATISTICS 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-113
- TIMESTAMP WITH LOCAL TIME ZONE データ型, 2-21
- TIMESTAMP WITH TIME ZONE データ型, 2-20
- TIMESTAMP データ型, 2-20
 - DB2, 2-35
 - SQL/DS, 2-35
- TIME データ型
 - DB2, 2-35
 - SQL/DS, 2-35
- TM 数値書式要素, 2-61
- TO SAVEPOINT 句
 - ROLLBACK, 17-98
- TO_CHAR
 - 数値変換ファンクション, 6-164
 - 日時変換ファンクション, 6-162
- TO_CHAR ファンクション, 2-60, 2-64, 2-71
- TO_CHAR (文字変換) ファンクション, 6-161
- TO_CLOB ファンクション, 6-166
- TO_DATE ファンクション, 2-64, 2-69, 2-72, 6-166
- TO_DSINTERVAL ファンクション, 6-168
- TO_LOB ファンクション, 6-169
- TO_MULTI_BYTE ファンクション, 6-170
- TO_NCHAR (数値) ファンクション, 6-172
- TO_NCHAR (日時変換) ファンクション, 6-172
- TO_NCHAR (文字) ファンクション, 6-171
- TO_NCLOB ファンクション, 6-173
- TO_NUMBER ファンクション, 2-60, 6-174
- TO_SINGLE_BYTE ファンクション, 6-175
- TO_TIMESTAMP_TZ ファンクション
 - SQL ファンクション
 - TO_TIMESTAMP_TZ, 6-176
- TO_TIMESTAMP ファンクション, 6-175
- TO_YMINTERVAL ファンクション, 6-178
- TRACE_ENABLED 初期化パラメータ
 - ALTER SYSTEM で設定, 10-113
- TRACEFILE_IDENTIFIER 初期化パラメータ
 - ALTER SESSION で設定, 10-10
 - ALTER SYSTEM で設定, 10-114
- TRANSACTION_AUDITING 初期化パラメータ
 - ALTER SYSTEM で設定, 10-114
- TRANSACTIONS_PER_ROLLBACK_SEGMENT 初期化パラメータ
 - ALTER SYSTEM で設定, 10-115
- TRANSACTIONS 初期化パラメータ
 - ALTER SYSTEM で設定, 10-114
- TRANSLATE ...USING ファンクション, 6-180
- TRANSLATE ファンクション, 6-178
- TREAT ファンクション, 6-181
- TRIM ファンクション, 6-183
- TRUNCATE PARTITION 句
 - ALTER TABLE, 11-71
- TRUNCATE SUBPARTITION 句
 - ALTER TABLE, 11-71
- TRUNCATE 文, 18-51
- TRUNC ファンクション
 - 書式モデル, 6-210
 - 数値ファンクション, 6-184
 - 日付ファンクション, 6-185
- TRUST 属性
 - PRAGMA RESTRICT_REFERENCES, 16-16
- TZ_OFFSET ファンクション, 6-185
- TZD 日時書式要素, 2-65
- TZH 日時書式要素, 2-65
- TZM 日時書式要素, 2-65

TZR 日時書式要素, 2-65

U

UID ファンクション, 6-186

UNDER ANY TABLE システム権限, 17-41

UNDER ANY VIEW システム権限, 17-42

UNDER_PATH 条件, 5-20

UNDER オブジェクト権限, 17-45
型, 17-48

ビュー, 17-46

UNDER 句

CREATE VIEW, 16-45

UNDO

システム管理, 9-134, 13-32

ロールバック, 9-134, 13-32

UNDO_MANAGEMENT 初期化パラメータ

ALTER SYSTEM で設定, 10-115

UNDO_RETENTION 初期化パラメータ

ALTER SYSTEM で設定, 10-116

UNDO_SUPPRESS_ERRORS 初期化パラメータ

ALTER SESSION で設定, 10-10

ALTER SYSTEM で設定, 10-116

UNDO_TABLESPACE 初期化パラメータ

ALTER SYSTEM で設定, 10-116

UNDO のロールバック, 9-134, 13-32

UNDO 表領域

削除, 17-10

作成, 13-32, 15-77

変更, 11-99

UNDO 表領域句

CREATE DATABASE, 13-32

CREATE TABLESPACE, 15-77

UNIFORM 句

CREATE TABLESPACE, 15-82

UNION ALL 集合演算子, 3-6, 18-23

UNION 集合演算子, 3-6, 18-23

UNIQUE 句

CREATE INDEX, 13-67

CREATE TABLE, 15-26

SELECT, 18-11

UNISTR ファンクション, 6-187

UNLIMITED TABLESPACE システム権限, 17-41

UNQUIESCE 句

ALTER SYSTEM, 10-30

UNRECOVERABLE, 9-71, 15-29

「NOLOGGING 句」を参照

UNUSABLE LOCAL INDEXES 句

ALTER MATERIALIZED VIEW, 9-98

ALTER TABLE, 11-80

UNUSABLE 句

ALTER INDEX, 9-76

UPDATE ANY TABLE システム権限, 17-40

UPDATE BLOCK REFERENCES 句

ALTER INDEX, 9-77, 9-78

ALTER TABLE, 11-41

UPDATE GLOBAL INDEXES 句

ALTER TABLE, 11-81

UPDATE SET 句

MERGE, 17-75

UPDATEXML ファンクション, 6-188

UPDATE オブジェクト権限, 17-45

ビュー, 17-47

表, 17-46

UPDATE 文, 18-56

トリガー, 15-93

UPGRADE 句

ALTER TABLE, 11-37

UPPER ファンクション, 6-189

URL

生成, 6-153

UROWID

外部キー表, 2-33

索引構成表, 2-33

説明, 2-33

UROWID データ型, 2-33

USE_STORED_OUTLINES 初期化パラメータ

ALTER SESSION で設定, 10-117

USE_CONCAT ヒント, 2-100

USE_INDIRECT_DATA_BUFFERS 初期化パラメータ

ALTER SYSTEM で設定, 10-117

USE_MERGE ヒント, 2-101

USE_NL ヒント, 2-101

USE_PRIVATE_OUTLINES セッション・パラメータ,
10-15

USE_STORED_OUTLINES セッション・パラメータ,
10-16, 10-117

USER SYSTEM 句

CREATE DATABASE, 13-26

USER SYS 句

CREATE DATABASE, 13-26

USER_COL_COMMENTS データ・ディクショナリ・
ビュー, 12-67

USER_DUMP_DEST 初期化パラメータ
 ALTER SYSTEM で設定, 10-117
USER_TAB_COMMENTS データ・ディクショナリ・
 ビュー, 12-67
USERENV ファンクション, 6-190
USER ファンクション, 6-190
USING BFILE 句
 CREATE JAVA, 13-94
USING BLOB 句
 CREATE JAVA, 13-94
USING CLOB 句
 CREATE JAVA, 13-94
USING INDEX 句
 ALTER MATERIALIZED VIEW, 9-99
 ALTER TABLE, 11-31
 CREATE MATERIALIZED VIEW, 14-18
 CREATE TABLE, 15-55
 制約, 7-21
USING NO INDEX 句
 CREATE MATERIALIZED VIEW, 14-18
USING ROLLBACK SEGMENT 句
 ALTER MATERIALIZED VIEW...REFRESH, 9-103
 CREATE MATERIALIZED VIEW, 14-22
USING 句
 ALTER INDEXTYPE, 9-84
 ASSOCIATE STATISTICS, 12-47, 12-48
 CREATE DATABASE LINK, 13-38
 CREATE INDEXTYPE, 13-89
 CREATE OPERATOR, 14-42
UTC
 日時値から抽出, 6-154
UTC オフセット
 タイムゾーン地域に置換, 2-21
UTL_FILE_DIR 初期化パラメータ
 ALTER SYSTEM で設定, 10-118
UTLCHN.SQL スクリプト, 12-42
UTLEXPT1.SQL スクリプト, 11-79
UTLXPLAN.SQL スクリプト, 17-23
U 数値書式要素, 2-61

V

VALIDATE REF UPDATE 句
 ANALYZE, 12-40
VALIDATE STRUCTURE 句
 ANALYZE, 12-41

VALIDATE 句
 DROP TYPE, 17-15
VALUES LESS THAN 句
 CREATE TABLE, 15-43
VALUES 句
 CREATE INDEX, 7-23, 13-77, 15-57
 INSERT, 17-60
VALUE ファンクション, 6-192
VAR_POP ファンクション, 6-193
VAR_SAMP ファンクション, 6-194
VARCHAR2 データ型, 2-11
 NUMBER への変換, 2-60
VARCHAR データ型, 2-11
 DB2, 2-35
 SQL/DS, 2-35
VARGRAPHIC データ型
 DB2, 2-35
 SQL/DS, 2-35
VARIANCE ファンクション, 6-196
VARRAY, 2-37
 記憶特性, 11-45, 11-56, 15-38
 作成, 16-3, 16-8, 16-18
 仕様部の削除, 17-14
 ネストした表との比較, 2-46
 比較規則, 2-46
 本体の削除, 17-17
 戻り値の変更, 11-55
 ライン外での格納, 2-37
 列プロパティの変更, 11-16
VARRAY 句
 ALTER TABLE, 11-14
VARRAY 列プロパティ
 ALTER TABLE, 11-14, 11-45
 CREATE MATERIALIZED VIEW, 14-11
 CREATE TABLE, 15-13, 15-38
VARYING 配列「VARRAY」を参照
VSIZE ファンクション, 6-197
V 数値書式要素, 2-61

W

WHEN MATCHED 句
 MERGE, 17-76
WHEN NOT MATCHED 句
 MERGE, 17-76
WHENEVER NOT SUCCESSFUL 句
 NOAUDIT, 17-82

WHENEVER SUCCESSFUL 句
 AUDIT sql_statements, 12-56
 NOAUDIT, 17-82
WHEN 句
 CREATE TRIGGER, 15-98
WHERE 句
 DELETE, 16-58
 SELECT, 8-4
 UPDATE, 18-64
 問合せおよび副問合せ, 18-19
WIDTH_BUCKET ファンクション, 6-198
With Admin Option 句
 Grant, 17-31
WITH CHECK OPTION 句
 CREATE VIEW, 16-41, 16-48
 DELETE, 16-56
 INSERT, 17-59
 SELECT, 18-7
 UPDATE, 18-60
With Grant Option 句
 Grant, 17-33
With Hierarchy Option
 GRANT, 17-33
WITH INDEX CONTEXT 句
 CREATE OPERATOR, 14-42
WITH OBJECT IDENTIFIER 句
 CREATE VIEW, 16-44
WITH OBJECT ID 句
 CREATE MATERIALIZED VIEW LOG, 14-37
WITH OBJECT OID 「WITH OBJECT IDENTIFIER」を
 参照
WITH PRIMARY KEY 句
 ALTER MATERIALIZED VIEW, 9-102
 CREATE MATERIALIZED VIEW LOG, 14-37
 CREATE MATERIALIZED VIEW...REFRESH,
 14-19
WITH query_name 句
 SELECT, 18-10
WITH READ ONLY 句
 CREATE VIEW, 16-41, 16-48
 DELETE, 16-56
 INSERT, 17-59
 SELECT, 18-7
 UPDATE, 18-60
WITH ROWID 句
 CREATE MATERIALIZED VIEW LOG, 14-37

 CREATE MATERIALIZED VIEW...REFRESH,
 14-19
 REF 制約, 7-17
WITH SEQUENCE 句
 CREATE MATERIALIZED VIEW LOG, 14-37
WNDS 属性
 PRAGMA RESTRICT_REFERENCES, 16-16
WNPS 属性
 PRAGMA RESTRICT_REFERENCES, 16-16
WORKAREA_SIZE_POLICY 初期化パラメータ
 ALTER SESSION で設定, 10-10
 ALTER SYSTEM で設定, 10-118
WRITE オブジェクト権限
 ディレクトリ, 17-48
WW 日時書式要素, 2-65
W 日時書式要素, 2-65

X

XMLDATA 疑似列, 2-84
XML データベース・リポジトリ
 SQL アクセス, 5-13
XMLGenFormatType オブジェクト, 2-74
XMLType 記憶域句
 CREATE TABLE, 15-41
XMLType ビュー, 16-47
 問合せ, 16-47
XMLType 表
 索引の作成, 13-82
 作成, 15-61, 15-67
XMLType 列
 記憶域, 11-47, 15-41
 プロパティ, 11-47, 15-41
XML 書式モデル, 2-74
XML データ
 記憶域, 15-41
XML データベース・リポジトリ
 SQL アクセス, 5-20
XML 文書
 XML のフラグメントから生成, 6-156
 データベースからの検索, 6-153
XML のフラグメント, 6-60
X 数値書式要素, 2-61
X 日時書式要素, 2-65

Y

Y,YYY 日時書式要素, 2-65

YEAR 日時書式要素, 2-65

YYYY 日時書式要素, 2-65

YYY 日時書式要素, 2-65

YY 日時書式要素, 2-65

Y 日時書式要素, 2-65

あ

アーカイブ REDO ログ

位置, 9-27

格納場所, 10-70

アーカイブ・モード

指定, 13-28

アーカイブ・ログ

スタンバイ・データベースへの適用, 9-31

アウトライン

PUBLIC による使用, 14-45

置換え, 14-45

オブティマイザによる使用, 10-117

オブティマイザによるプライベート・アウトライン
の使用, 10-15

グループの格納, 14-46

現行のセッションによる使用, 14-45

異なるカテゴリへの割当て, 9-116, 9-117, 9-118

コピー, 14-46

再構築, 9-116, 9-118

再コンパイル, 9-117

作成, 14-44

実行計画の生成での使用, 14-44, 10-16

自動作成および自動格納, 10-43

セッション中のリストア, 10-11

データベースからの削除, 16-87

動的な使用可能化または使用禁止化, 14-44

名前の変更, 9-116, 9-117, 9-118

付与

システム権限, 17-38

文に作成, 14-46

空きリスト

LOB に対する指定, 15-38

表、パーティション、クラスタまたは索引の指定,
7-56

圧縮

索引キー, 9-64

アップグレード

リリース 7.3.4 から Oracle9i リリース 2, 9-25

アプリケーション

検証, 13-11

保護, 13-11

ユーザーとしての接続の許可, 12-25

アプリケーション・サーバー

ユーザーとしての接続の許可, 12-25

暗黙的なデータ変換, 2-46, 2-47, 2-49

い

移行行

クラスタ, 12-37

リスティング, 12-42

以上 / 以下のテスト, 5-5

一意索引, 13-67

一意制約

索引, 15-55

使用可能, 15-54

一意の問合せ, 18-11

一時表

作成, 15-7, 15-24

セッション固有, 15-24

トランザクション固有, 15-24

一時表領域

SQL 例, 15-88

エクステンツ管理の個別指定, 15-87

作成, 15-86

データベース作成時のエクステンツ管理の指定,
13-25

デフォルト, 9-49

ユーザーへの指定, 16-35

一時ファイル

一時表領域の定義, 15-86

オフライン化, 9-36

オンライン化, 9-36

サイズ, 7-38

サイズ変更, 9-36

再利用, 7-38

削除, 9-36

指定, 7-36, 15-87

自動拡張, 7-39

自動拡張の使用可能化, 7-39, 9-36

自動拡張の使用禁止化, 9-36

データベースの定義, 13-25

名前の変更, 9-37

- 表領域の定義, 15-76
- 位置の透過性, 15-2
- 一般リカバリ句
 - ALTER DATABASE, 9-14, 9-26
- インスタンス
 - グローバル・ネーム解決, 10-61
 - 索引エクステンツの使用可能化, 9-68
 - パラメータの設定, 10-31
 - メモリー要件, 10-45
- インスタンス・リカバリ
 - 中断後に継続する, 9-26
- インダウト・トランザクション
 - 強制コミット, 12-71
 - 強制実行, 12-71
 - ロールバック, 17-97
 - ロールバックの強制, 17-98
- インライン制約
 - ALTER TABLE, 11-43
 - CREATE TABLE, 15-26
- インライン・ビュー, 8-12

う

- 埋込み SQL, 1-3, 9-5
 - プリコンパイラのサポート, 9-5

え

- エクステンツ
 - インスタンスを使用したアクセス制限, 9-68
 - オブジェクト作成時に割り当てられるエクステンツ数の指定, 7-55
 - オブジェクトの最大エクステンツ数の指定, 7-56
 - オブジェクトの第1エクステンツの指定, 7-54
 - オブジェクトの第2エクステンツの指定, 7-54
 - サイズ増加の割合の指定, 7-55
 - サブパーティションへの割当て, 11-35
 - パーティションへの割当て, 11-35
 - 表への割当て, 11-35
- エラー・メッセージ
 - 言語の設定, 10-8
- 演算子, 3-1
 - コメント, 12-68
 - 索引タイプからの削除, 9-84
 - 索引タイプへの追加, 9-84
 - 算術, 3-3
 - 実装の指定, 14-41

- 集合, 3-6, 18-23
- 単項, 3-2
- バイナリ, 3-2
- 付与
 - システム権限, 17-37
- 変更, 9-115
- ユーザー定義, 3-6
 - 削除, 16-85
 - 作成, 14-40
 - 実装タイプ, 14-42
 - 実装を設定するファンクション, 14-42
 - バインド実装方法, 14-42
 - バインドの戻り型, 14-42
 - ファンクションへのバインド, 14-42
- ユーザー定義型, 9-115
- 優先順位, 3-2
- 連結, 3-4

お

- 応答時間
 - 最適化, 2-91
- オブジェクト・アクセス式, 4-12
- オブジェクト・インスタンス
 - タイプ, 5-19
- オブジェクト「オブジェクト型」または「データベース・オブジェクト」を参照, 16-3
- オブジェクト型, 2-36
 - MAP メソッド, 16-17
 - ORDER メソッド, 16-17
 - SQL 例, 16-19
 - VARRAY, 16-8
- 値
 - 比較, 16-29
- 新しいメンバーのサブプログラムの追加, 12-11
- 依存する型の処理, 12-18
- 依存する型の無効化, 12-18
- 継承, 16-14
 - コンポーネント, 2-36
- 作成, 16-3, 16-5
- サブタイプ, 12-11
 - サブタイプの権限, 17-33
 - サブタイプの指定, 16-11
- 参照 REF を参照
- 参照の再構築の展開, 9-100
- システム権限の付与, 17-41
- 使用可能なオブジェクト・インスタンス, 16-13

- 使用可能なサブタイプ, 16-13
- 仕様部および本体のコンパイル, 12-10
- 仕様部の削除, 17-14
- スーパータイプ, 12-11
- スタティック・メソッド, 16-13
- 属性, 2-115
 - 階層のメンバーシップ, 11-43
 - 型の階層, 15-35
 - 代替性, 11-43
- 統計タイプ, 12-46
- 統計タイプの削除, 17-14
- トップレベル, 16-10
- ネストした表, 16-9
- 比較規則, 2-46
 - MAP ファンクション, 2-46
 - ORDER ファンクション, 2-46
- ファンクション・サブプログラム, 12-12, 16-12, 16-28
 - 宣言, 16-30
- 不完全, 16-3, 16-4
- プロシージャ・サブプログラム, 12-12, 16-12, 16-28
 - 宣言, 16-30
- 本体
 - SQL 例, 16-30
 - 再作成, 16-27
 - 作成, 16-25
- 本体の削除, 17-17
- メソッド, 2-115
- メソッドの削除, 12-15
- メソッドの追加, 12-15
- メンバー・メソッドの定義, 16-25
- ユーザー定義
 - 作成, 16-9
- ルートの指定, 16-10
- オブジェクト型マテリアライズド・ビュー
 - 作成, 14-13
- オブジェクト型列
 - 階層のメンバーシップ, 11-43
 - 型の階層, 15-35
 - 代替性, 11-43
 - プロパティの定義
 - マテリアライズド・ビュー, 14-10, 14-11
 - プロパティの変更
 - 表, 11-12, 11-43
- オブジェクト・キャッシュ, 10-9, 10-84
- オブジェクト権限
 - DEBUG, 17-45
 - FLASHBACK, 17-45
 - ON COMMIT REFRESH, 17-45
 - QUERY REWRITE, 17-45
 - UNDER, 17-45
 - データベース・オブジェクト
 - 取消し, 17-92
 - 取消し, 17-88
 - PUBLIC, 17-91
 - ユーザー, 17-86, 17-90
 - ロール, 17-86, 17-90
 - 付与, 14-73
 - 特定の列, 17-32
 - 複数, 14-80
- オブジェクト参照ファンクション, 6-15
- オブジェクト識別子
 - REF に含まれる, 2-36
 - オブジェクト・ビュー, 16-44
 - 索引の指定, 15-60
 - システム生成, 15-60
 - 指定, 15-60
 - 主キー, 15-60
- オブジェクト・ビュー, 16-44
 - 作成, 16-44
 - サブビューの作成, 16-45
- 実表
 - 行の追加, 17-52
- 定義, 16-38
- 問合せ, 16-44
- オブジェクト表
 - アップグレード, 11-37
 - 階層の一部, 15-59
 - 行の追加, 17-52
 - 最新バージョンに更新, 11-37
 - 作成, 15-9, 15-59
 - システム生成の列名, 15-59, 15-61, 16-44, 16-47
 - 問合せ, 15-59
- オペランド, 3-1
- オペレーティング・システム・ファイル
 - 削除, 9-37, 17-11
- オンライン REDO ログ
 - 初期化しなおし, 9-42
- オンライン索引, 13-75
 - 再構築, 11-83
- オンライン・バックアップ
 - 表領域のオンライン・バックアップの終了, 11-103

か

カーソル

- キャッシュ, 17-23

階層

- ディメンションからの削除, 9-58

- ディメンションの追加, 9-58

- ディメンションの定義, 13-43

階層問合せ, 2-81, 8-3, 18-20

- 親である行, 2-81, 8-4

- 子である行, 2-81, 8-4

- 順序付け, 18-24

- 説明, 2-81

- リーフ行, 2-81

- ルートおよびノードの値の検出, 6-147

階層問合せ句

- SELECT および副問合せ, 18-8

外部 LOB, 2-26

外部キー制約, 7-13

外部キー表

- ROWID, 2-33

外部結合, 8-11

- 制限事項, 8-11

外部表, 15-30

- 作成, 15-32

- 制限事項, 15-33

- 変更, 11-58

外部ファンクション, 13-49, 14-60

外部プロシージャ, 14-58, 14-60

- リモート・データベースからの実行, 14-3

外部ユーザー, 14-74, 16-33

科学表記法, 2-61

拡張 ROWID, 2-32

- 基本, 2-32

- 直接利用できない, 2-32

拡張索引作成機能

- 例, D-2

拡張サブパーティション表名, 2-103

- DML 文, 2-104

- 構文, 2-104

- 制限, 2-104

拡張パーティション表名, 2-103

- DML 文, 2-104

- 構文, 2-104

- 制限, 2-104

数

- SQL 構文内, 2-53

- 構文, 2-53

- 精度, 2-53

- 比較規則, 2-42

- 浮動小数点, 2-12, 2-13

- フルスベルでの表記, 2-71

- 丸め, 2-13

- 型「オブジェクト型」または「データ型」を参照

- 型コンストラクタ式, 4-14

型メソッド

- 戻り型, 16-14

各国語キャラクタ・セット

- 可変長文字列, 2-11

- 固定幅と可変幅, 2-11

- 変更, 9-47

- マルチバイト・キャラクタ・セット, 2-10, 2-11

- マルチバイト・キャラクタ・データ, 2-31

監査

- SQL 文, 12-51, 12-56

- プロキシ, 12-51

- ユーザー, 12-51

- SQL 文の停止, 17-79

- SYS スキーマに接続されたユーザー, 10-35

オプション

- SQL 文, 12-59

- データベース・オブジェクト, 12-56

- システム権限, 12-51

- スキーマの SQL 文, 12-52

- ディレクトリの SQL 文, 12-52

方針

- 値に基づいた方針, 12-50

関数

- 逆分散, 6-112, 6-114

完全な外部結合, 18-18

管理スタンバイ・リカバリ

- 既存の終了, 9-33

- 自動終了, 9-32

- 制御の復帰, 9-33

- 遅延の上書き, 9-31

- バックグラウンド・プロセスの, 9-31

管理リカバリ

- データベース, 9-16

- 待ち時間, 9-31

き

- キー圧縮, 15-31

- 索引
 - 使用禁止, 9-73
- 索引構成表, 15-31
- 索引再構築, 11-83
- 使用可能, 9-71
- 使用禁止, 9-73, 13-74
- 定義, 9-73
- キー保存表, 16-47
- キーワード, 2-106
 - オブジェクト名, 2-106
 - オプション, A-3
 - 必須, A-3
- 記憶域パラメータ
 - デフォルトの変更, 11-101
 - リセット, 18-51
- 期間式, 4-12
- 期間データ型, 2-16
- 疑似列, 2-77
 - CURRVAL, 2-78
 - LEVEL, 2-81
 - NEXTVAL, 2-78
 - ROWID, 2-82
 - ROWNUM, 2-83
 - 使用方法, 2-84
 - SYS_NC_ROWINFO\$, 15-59, 15-61, 16-44, 16-47
 - XMLDATA, 2-84
- 逆索引, 13-74
- 逆分散関数, 6-112, 6-114
- キャッシュされたカーソル
 - 実行計画, 17-23
- キャラクタ・セット
 - 共通, 2-44
 - データベースへの指定, 13-29
 - 変更, 9-47
 - マルチバイト・キャラクタ, 2-106
- キャラクタ・ラージ・オブジェクト「CLOB」を参照
- 行
 - 違反制約の格納, 11-79
 - 階層順序の選択, 8-3
 - 削除
 - クラスタ, 18-51
 - パーティションおよびサブパーティション, 16-56
 - 表, 18-51
 - 表およびビュー, 16-53
 - 制約の指定, 7-15

- 挿入
 - サブパーティション, 17-58
 - パーティション, 17-58
 - リモート・データベース, 17-58
 - パーティション間の移動, 15-15, 15-57
 - 表への追加, 17-52
- 兄弟関係
 - 階層問合せでの順序付け, 18-24
- 共有サーバー
 - システム・パラメータ, 10-107
 - パラメータ
 - DISPATCHERS, 10-54
 - プロセス
 - 終了, 10-107
 - 追加プロセスの作成, 10-107
- 共有ブール
 - フラッシュ, 10-29
- 行レベル依存の追跡, 13-8, 15-52

<

- 空白埋め
 - 書式モデルへの指定, 2-71
 - 抑制, 2-71
- 空白埋め比較, 2-44
- クエリー・リライト
 - 使用可能, 10-98
 - 使用可能および使用禁止, 10-10
 - 使用禁止, 10-98
 - 定義済, 18-4
 - ディメンション, 13-41
 - ファンクション索引, 10-10
 - ルールベースの最適化, 10-10
- グローバリゼーション・サポート
 - セッション設定の変更, 10-8
- 位取り
 - NUMBER データ型, 2-12
 - 精度より大きい, 2-13
 - 負, 2-13
- クラスタ
 - SQL 例, 16-65
 - 移行行および連鎖行, 12-37, 12-42
 - エクステンツの割当て, 9-8, 9-9
- キー値
 - 領域の変更, 9-9
 - 領域の割当て, 13-5

- 記憶域属性
 - 変更, 9-8
- 記憶特性の変更, 9-9
- クラスタ索引, 13-69
- 構造の検証, 12-41
- 索引, 13-6
- 作成, 13-2
- 作成表領域, 13-6
- システム権限の付与, 17-35
- データベースからの削除, 16-64
- 統計情報の収集, 12-37
- 取り出されたブロックのキャッシュ, 13-9
- ハッシュ, 13-6
 - 単一表, 13-6
- 表の削除, 16-65
- 表への割当て, 15-34
- 物理属性
 - 指定, 13-5
 - 変更, 9-8
- 並列度
 - 作成時, 13-7
 - 変更, 9-8, 9-10
- 変更, 9-7
- 未使用エクステンツの割当て解除, 9-8
- 未使用領域の開放, 9-10
- グルーピング・セット, 18-22
- グループ化
 - 重複の排除, 6-67
- グループ比較条件, 5-7
- グローバル・インデックス「グローバル・パーティション索引」を参照
- グローバル・データベース名
 - 解決, 10-61
- グローバル・パーティション索引, 7-22, 13-76, 13-78, 15-56
- グローバル・ユーザー, 14-74, 16-34
- クローン・データベース
 - マウント, 9-24
- クロス結合, 18-18

け

- 結合, 8-9
 - 外部, 8-11
 - 制限事項, 8-11
 - 完全な外部結合, 18-18
 - クロス, 18-18

- 結合条件のない, 8-11
- 自己, 8-10
- 自然, 18-19
- 条件
 - 定義, 8-10
- 等価結合, 8-10
- 内部, 8-11, 18-18
- ネステッド・ループの最適化, 10-9
- パラレルおよび PQ_DISTRIBUTE ヒント, 2-98
- 左側外部結合, 18-18
- 右側外部結合, 18-18
- 結合ビュー
 - 更新可能化, 16-47
 - 変更, 16-57, 17-57, 18-59
 - 例, 16-50
- 権限
 - 「システム権限」または「オブジェクト権限」を参照
 - オブジェクト型のサブタイプ, 17-33
 - 権限受領者からの取消し, 17-88
- 検証
 - オンラインのデータベース・オブジェクト, 12-42
 - クラスタ, 12-41
 - 索引, 12-41
 - データベース・オブジェクト
 - オフライン, 12-42
 - 表, 12-41

こ

- 更新
 - MERGE の使用, 17-75, 17-76
 - 同時の挿入, 17-75
- 更新操作
 - サブリメンタル・ログ・データの収集, 9-41
- 構文図, A-1
 - 複数の部分に分割された図, A-4
 - ループ, A-4
- コール
 - CPU 時間の制限, 14-69
 - 読み込まれたデータ・ブロックの制限, 14-69
- コール仕様
 - CREATE PROCEDURE, 14-64
 - CREATE TYPE, 16-15
 - CREATE TYPE BODY, 16-29
 - プロシージャ, 14-60

- コミット
 - 自動, 12-70
- コメント, 2-84
 - SQL 文, 2-84
 - 演算子, 12-68
 - オブジェクトからの削除, 12-67
 - オブジェクトへの追加, 12-67
 - 索引タイプ, 12-69
 - 指定, 2-85
 - スキーマ・オブジェクト, 2-86
 - データ・ディクショナリからの削除, 12-67
 - トランザクションとの関連, 12-71
 - 表, 12-68
 - 表示, 12-67
 - 表の列, 12-68
- 固有の問合せ, 18-11
- コレクション
 - VARRAY, 2-37
 - 行の挿入, 17-59
 - 検索メソッドの変更, 11-11
 - ネスト解除, 18-16
 - 例, 18-37
 - ネストした表, 2-37
 - 表としての扱い, 16-58, 17-59, 18-16, 18-59, 18-61
 - 変更, 11-55
- コレクション型
 - マルチレベル, 15-40
- コレクション型値
 - データ型への変換, 6-25
- コレクション・ネスト解除, 18-16
 - 例, 18-37
- コンストラクタ
 - オブジェクト型の定義, 16-16
 - ユーザー定義, 16-16
- コンストラクタ・メソッド
 - オブジェクト型, 16-3
- コンテキスト
 - システム権限の付与, 17-35
 - ネームスペースの作成, 13-11
- コンテキスト・ネームスペース
 - LDAP ディレクトリを使用する初期化, 13-12
 - OCI を使用する初期化, 13-12
 - インスタンスへのアクセス性, 13-13
 - データベースからの削除, 16-66
 - パッケージへの関連付け, 13-11

- コンパイラ・スイッチ
 - 削除および保持, 9-60, 9-120, 9-123, 12-4, 12-11
- コンボジット・パーティション化
 - 表の作成時, 15-19, 15-47
 - レンジ-リスト, 11-61, 15-48

さ

- サーバー・パラメータ・ファイル
 - 作成, 14-87
- サービス名
 - リモート・データベース, 13-38
- 再開可能な領域割当て, 10-6
- 最高水位標
 - クラスタ, 9-10
 - 索引, 9-68
 - 表, 11-36, 12-36
- 再構築, 9-100
- 索引, 9-69
 - UNUSABLE のマーク付け, 9-76
 - XML Type 表, 13-82
 - アクセス・パスの最適化, 10-9
 - アプリケーション固有, 13-88
 - 一意, 13-67
 - 移動, 9-71
 - オンライン, 13-75
 - オンラインの場合の再構築, 9-73
 - キー圧縮, 9-73
 - キー圧縮の使用可能化, 9-71
 - キーの反復の排除, 9-71
 - 記憶域属性, 13-73
 - 逆, 9-71, 9-72, 13-74
 - クラスタ, 13-69
 - クラスタへの作成, 13-62
 - グローバル・パーティション, 7-22, 13-76, 13-78, 15-56
 - 更新, 11-81
 - グローバル・パーティションの作成, 13-65
- 降順, 13-72
 - クエリー・リライト, 13-72
 - ファンクション索引, 13-72
- 構造の検証, 12-41
- コンボジット・パーティション表, 13-78
- コンボジット・パーティション表の作成, 13-67
- 再構築, 9-71
- 再構築操作のログの記録, 9-71, 9-74
- 再構築の統計, 9-74

- 再作成, 9-71
- 索引構成表, 13-69
- 索引に基づく, 13-79
- 索引パーティションの削除, 16-75
- 索引ブロックの内容のマージ, 9-76
- 作成, 13-61
- 作成の平行化, 13-76
- サブパーティション
 - UNUSABLE のマーク付け, 9-80
 - 移動, 9-71
 - エクステンツの割当て, 9-80
 - 記憶特性の変更, 9-77
 - 再構築, 9-71
 - 再作成, 9-71
 - 使用禁止, 9-76
 - デフォルト属性の変更, 9-77
 - 名前の変更, 9-79
 - 表領域の指定, 9-71, 9-73
 - 物理属性の変更, 9-69
 - 変更, 9-71
 - 未使用領域の解放, 9-68, 9-80
- システム権限の付与, 17-36
- 使用禁止, 9-76
- 昇順, 13-72
- 使用の統計, 9-76
- 新規エクステンツの割当て, 9-68
- スカラー型オブジェクト属性上, 13-69
- 制約の適用, 11-57, 15-55
- 属性の変更, 9-69, 9-71
- ダイレクト・パス・インサートのログの記録, 9-69
- データベースからの削除, 16-74
- 統計情報, 13-75
- 統計情報の収集, 12-34
- 統計タイプの削除, 16-75
- ドメイン, 13-61, 13-79, 13-88
- ドメイン例, D-2
- 名前の変更, 9-71, 9-76
- ネストした表の記憶表上, 13-69
- パーティション, 2-103, 13-61, 13-76
 - UNUSABLE のマーク付け, 9-79, 11-80
 - 記憶特性の変更, 9-77
 - 再構築, 9-71
 - 再作成, 9-71
 - 削除, 9-77, 9-79
 - 実特性の変更, 9-78
 - 使用禁止, 9-76
 - 使用禁止の再構築, 11-80
 - 新規の追加, 9-79
 - デフォルト属性の変更, 9-77
 - 名前の変更, 9-79
 - 表領域の指定, 9-71, 9-73
 - 物理属性の変更, 9-69
 - 分割, 9-77, 9-79
 - 未使用領域の解放, 9-68
 - ユーザー定義, 7-22, 13-76, 15-56
- パーティション化, 13-76
- パーティション表, 13-69
- ハッシュ・パーティション表, 13-78
 - 作成, 13-66
- B ツリー, 13-61
- ビットマップ, 13-68
- ビットマップ・ジョイン, 13-80
- 表, 13-63
 - 表の列, 13-69
 - 表領域, 13-73
 - 表領域の指定, 9-71, 9-73
 - ファンクション, 13-61
 - 作成, 13-70
 - 物理属性, 13-73
 - ブロック内容のマージ, 9-71
 - 並列性の変更, 9-69
 - 未使用領域の解放, 9-68
 - 未ソート, 13-74
 - リスト・パーティション表
 - 作成, 13-66
 - 例, 13-81
 - レンジ・パーティション表, 13-78
 - レンジ・パーティション表の作成, 13-66
 - ローカル・パーティション, 13-78
- 索引キー
 - 圧縮, 9-64
- 索引クラスタ
 - 作成, 13-6
- 索引構成表
 - 2 次索引の更新, 9-77
 - PCT_ACCESS_DIRECT 統計情報, 12-35
 - ROWID, 2-33
 - 移動, 11-82
 - オーバーフロー・セグメント
 - 記憶域の指定, 11-41, 15-45
 - 再構築, 11-82
 - 作成, 15-7
 - 主キー索引
 - 合せる, 11-42

- 更新, 11-41
- パーティション化された 2 次索引の更新, 9-78
- ビットマップ索引の作成, 15-31
- 変更, 11-39
- マッピング表, 11-83
 - 移動, 11-65
- マッピング表の作成, 15-31
- 索引サブパーティション, 13-67
- 索引タイプ
 - インスタンス, 13-61
 - コメント, 12-69
 - 索引タイプに基づく索引, 13-79
 - 削除ルーチンの起動, 16-75
 - 作成, 13-88
 - システム権限の付与, 17-36
 - 実装タイプの変更, 9-83
 - 操作の追加, 9-83
 - データベースからの削除, 16-76
 - 統計情報の関連付け, 12-47, 12-48
 - 統計情報のタイプの関連性の削除, 16-76
 - 変更, 9-83
- 索引パーティション
 - 削除, 9-66
 - サブパーティションの作成, 13-67
- サブタイプ, 12-11
 - 安全な削除, 17-15
- サブパーティション
 - 値の変更, 18-60
 - 合せる, 11-62
 - エクステンツの割当て, 11-35, 11-63
 - 行の削除, 11-71, 16-56
 - 行の挿入, 17-58
 - 行の追加, 17-52
 - 作成, 15-19
 - 指定, 15-47
 - 挿入操作のロギング, 11-35
 - 追加, 11-61
 - テンプレートの置換え, 11-59
 - テンプレートの削除, 11-59
 - テンプレートの作成, 11-59, 15-47
 - 名前の変更, 11-71
 - ハッシュ, 15-48
 - 非パーティション表への変換, 11-77
 - 表との交換, 11-25
 - 物理属性
 - 変更, 11-33
 - 別のセグメントへの移動, 11-65

- 未使用領域の解放, 11-36, 11-63
- リスト, 15-48
- リスト・パーティションの追加, 11-61
- ロック, 17-71
- サブパーティション・テンプレート
 - 置換え, 11-59
 - 作成, 11-59
- サブリメンタル・ロギング
 - 最小, 9-41
 - 識別キー (フル), 9-41
- 算術
 - DATE 値, 2-19
 - 演算子, 3-3
- 参照整合性制約, 7-13

し

式

- CASE, 4-6
- CURSOR, 4-8
- DUAL 表の計算, 8-14
- SQL 構文内, 4-2
- オブジェクト・アクセス, 4-12
- 型コンストラクタ, 4-14
- 期間, 4-12
- スカラー副問合せ, 4-13
- 宣言型の変更, 6-181
- 単純, 4-4
- 日時, 4-10
- 比較, 6-48
- 複合, 4-5
- 変数, 4-15
 - リスト, 4-16
- 自己結合, 8-10
- システム・イベント
 - 属性, 15-99
 - トリガー, 15-95
- システム・グローバル領域
 - 更新, 10-26
 - フラッシュ, 10-29
- システム権限
 - ADMINISTER DATABASE TRIGGER, 17-41
 - ALTER ANY CLUSTER, 17-35
 - ALTER ANY DIMENSION, 17-36
 - ALTER ANY INDEX, 17-36
 - ALTER ANY INDEXTYPE, 17-36
 - ALTER ANY MATERIALIZED VIEW, 17-37

- ALTER ANY OUTLINE, 17-38
- ALTER ANY PROCEDURE, 17-38
- ALTER ANY ROLE, 17-38
- ALTER ANY SEQUENCE, 17-39
- ALTER ANY TABLE, 17-40
- ALTER ANY TRIGGER, 17-41
- ALTER ANY TYPE, 17-41
- ALTER DATABASE, 17-35
- ALTER PROFILE, 17-38
- ALTER RESOURCE COST, 17-39
- ALTER ROLLBACK SEGMENT, 17-39
- ALTER SESSION, 17-39
- ALTER SYSTEM, 17-35
- ALTER TABLESPACE, 17-40
- ALTER USER, 17-42
- ANALYZE ANY, 17-42
- AUDIT ANY, 17-42
- AUDIT SYSTEM, 17-35
- BACKUP ANY TABLE, 17-40
- BECOME USER, 17-42
- COMMENT ANY TABLE, 17-42
- CREATE ANY CLUSTER, 17-35
- CREATE ANY CONTEXT, 17-35
- CREATE ANY DIMENSION, 17-36
- CREATE ANY DIRECTORY, 17-36
- CREATE ANY INDEX, 17-36
- CREATE ANY INDEXTYPE, 17-36
- CREATE ANY LIBRARY, 17-37
- CREATE ANY MATERIALIZED VIEW, 17-37
- CREATE ANY OPERATOR, 17-38
- CREATE ANY OUTLINE, 17-38
- CREATE ANY PROCEDURE, 17-38
- CREATE ANY SEQUENCE, 17-39
- CREATE ANY SYNONYM, 17-39
- CREATE ANY TABLE, 17-40
- CREATE ANY TRIGGER, 17-41
- CREATE ANY TYPE, 17-41
- CREATE ANY VIEW, 17-42
- CREATE CLUSTER, 17-35
- CREATE DATABASE LINK, 17-35
- CREATE DIMENSION, 17-36
- CREATE INDEXTYPE, 17-36
- CREATE LIBRARY, 17-37
- CREATE MATERIALIZED VIEW, 17-37
- CREATE OPERATOR, 17-37
- CREATE PROCEDURE, 17-38
- CREATE PROFILE, 17-38

- CREATE PUBLIC DATABASE LINK, 17-35
- CREATE PUBLIC SYNONYM, 17-39
- CREATE ROLE, 17-38
- CREATE ROLLBACK SEGMENT, 17-39
- CREATE SEQUENCE, 17-39
- CREATE SESSION, 17-39
- CREATE SYNONYM, 17-39
- CREATE TABLE, 17-40
- CREATE TABLESPACE, 17-40
- CREATE TRIGGER, 17-41
- CREATE TYPE, 17-41
- CREATE USER, 17-42
- CREATE VIEW, 17-42
- DEBUG ANY PROCEDURE, 17-36
- DELETE ANY TABLE, 17-40
- DROP ANY CLUSTER, 17-35
- DROP ANY CONTEXT, 17-35
- DROP ANY DIMENSION, 17-36
- DROP ANY DIRECTORY, 17-36
- DROP ANY INDEX, 17-36
- DROP ANY INDEXTYPE, 17-36
- DROP ANY LIBRARY, 17-37
- DROP ANY MATERIALIZED VIEW, 17-37
- DROP ANY OPERATOR, 17-38
- DROP ANY OUTLINE, 17-38
- DROP ANY PROCEDURE, 17-38
- DROP ANY ROLE, 17-38
- DROP ANY SEQUENCE, 17-39
- DROP ANY SYNONYM, 17-39
- DROP ANY TABLE, 17-40
- DROP ANY TRIGGER, 17-41
- DROP ANY TYPE, 17-41
- DROP ANY VIEW, 17-42
- DROP PROFILE, 17-38
- DROP PUBLIC DATABASE LINK, 17-35
- DROP PUBLIC SYNONYM, 17-39
- DROP ROLLBACK SEGMENT, 17-39
- DROP TABLESPACE, 17-40
- DROP USER, 17-42
- EXECUTE ANY INDEXTYPE, 17-36
- EXECUTE ANY OPERATOR, 17-38
- EXECUTE ANY PROCEDURE, 17-38
- EXECUTE ANY TYPE, 17-41
- EXEMPT ACCESS POLICY, 17-43
- FLASHBACK ANY TABLE, 17-37, 17-40, 17-42
- FORCE ANY TRANSACTION, 17-43
- FORCE TRANSACTION, 17-43

- GLOBAL QUERY REWRITE, 17-37
- GRANT ANY OBJECT PRIVILEGE, 17-43
- GRANT ANY PRIVILEGE, 17-43
- GRANT ANY ROLE, 17-39
- INSERT ANY TABLE, 17-40
- LOCK ANY TABLE, 17-40
- MANAGE TABLESPACE, 17-40
- ON COMMIT REFRESH, 17-37
- QUERY REWRITE, 17-36, 17-37
- RESTRICTED SESSION, 17-35, 17-39
- RESUMABLE, 17-43
- SELECT ANY DICTIONARY, 17-43
- SELECT ANY SEQUENCE, 17-39
- SELECT ANY TABLE, 17-40
- SYSDBA, 17-43
- SYSOPER, 17-43
- UNDER ANY TYPE, 17-41
- UNDER ANY VIEW, 17-42
- UNLIMITED TABLESPACE, 17-41
- UPDATE ANY TABLE, 17-40
- 一覧, 17-35
- 取消し, 17-86
 - PUBLIC, 17-89
 - ユーザー, 17-88
 - ロール, 17-88
- 付与, 14-73, 17-27
 - PUBLIC, 17-30
 - ユーザー, 17-30
 - ロール, 17-30
- システム制御文, 9-5
 - PL/SQL サポート, 9-5
- システムの日付
 - 変更, 10-60
- システム・リソース
 - 使用可能および使用禁止, 10-102
- 自然結合, 18-19
- 実行計画
 - 削除するアウトライン, 16-87
 - 判断, 17-23
 - 保存, 14-44
- 実行時の再コンパイル
 - 回避, 9-122, 12-29
- 実行者権限
 - Java クラスについての変更, 9-86
 - Java クラスの定義, 13-91, 13-93
 - オブジェクト型の定義, 16-10
 - オブジェクト型の変更, 12-14
 - パッケージの定義, 14-49
 - ファンクションの定義, 13-54
 - プロシージャの定義, 14-61
- 実行者権限ファンクション
 - 定義, 13-54
- 自動 UNDO 管理モード, 9-134, 13-32
- 自動セグメント領域管理, 2-15, 15-83
- シノニム
 - 作成, 15-2
 - シノニム, 15-2
 - 定義の変更, 17-4
 - データベースからの削除, 17-4
 - 名前の変更, 17-84, 17-85
 - パブリック, 15-3
 - 削除, 17-4
 - 付与
 - システム権限, 17-39
 - プライベート・シノニムの削除, 17-4
 - リモート, 15-5
 - ローカル, 15-5
- 集計関数
 - ユーザー定義型, 13-57
- 集計ファンクション, 6-7
- 集合演算子, 3-6, 18-23
 - INTERSECT, 3-6
 - MINUS, 3-6
 - UNION, 3-6
 - UNION ALL, 3-6
- 主キー
 - 値の生成, 14-82
- 主キー制約, 7-12
 - 索引, 15-55
 - 使用可能, 15-55
- 順序, 2-78, 14-82
 - 値の事前割当て, 14-85
 - 値の順序付け, 9-138
 - 値のリサイクル, 9-138
 - 値へのアクセス, 14-82
 - キャッシュされた値の数の変更, 9-138
 - 再起動, 17-2
 - 値, 14-84
 - 異なる番号, 9-139
 - 事前に定義した制限, 14-83
- 最小値
 - 設定, 14-84
 - 設定または変更, 9-138
 - 排除, 9-138

- 最大値
 - 設定, 14-84
 - 設定または変更, 9-138
 - 排除, 9-138
- 再利用, 14-82
- 作成, 14-82
- 事前に定義した制限での停止, 14-83
- シノニム, 15-2
- 使用場所, 2-78
- 使用方法, 2-79
- 初期値の設定, 14-84
- 増分, 14-82
- 増分値の設定, 14-84
- データベースからの削除, 17-2
- 名前の変更, 17-84
- 付与
 - システム権限, 17-39
- 変更
 - 増分値, 9-138
 - 無制限での作成, 14-83
 - 連続する値の保証, 14-85
- 順序のリセット, 9-24
- 上位 N 番までの問合せ, 2-83
- 小計値
 - 導出, 18-21
- 条件
 - EXISTS, 5-13, 5-14
 - IS OF type, 5-19
 - LIKE, 5-14
 - NULL, 5-13
 - SQL 構文内, 5-1
 - UNDER_PATH, 5-20
 - グループ比較, 5-7
 - 単純比較, 5-6
 - 比較, 5-4
 - 複合, 5-21
 - メンバーシップ, 5-10
 - レンジ, 5-12
 - 論理, 5-8
- 小数点文字, 2-53
 - 指定, 2-61
 - セッション用のリセットする, 10-9
- 小のテスト, 5-5
- 初期化パラメータ
 - CIRCUITS, 10-39
 - セッション設定の変更, 10-6

- 書式
 - データベースからの戻り値, 2-58
 - データベースに格納された値, 2-58
 - 日付および数値「書式モデル」を参照
- 書式モデル, 2-58
 - XML, 2-74
 - 指定, 2-60
 - 修飾子, 2-71
 - 数値, 2-60
 - 数値、要素, 2-61
 - 日付, 2-64
 - 最大長, 2-64
 - 書式要素, 2-65
 - デフォルト書式, 2-64
 - 変更, 2-64
 - 戻り値の書式の変更, 2-59
- 序数
 - 指定, 2-71
 - フルスペルでの表記, 2-71

す

- 数値書式モデル, 2-60
- 数値ファンクション, 6-3
- スーパータイプ, 12-11
- スカラー副問合せ, 4-13
- スカラー副問合せ式, 4-13
- スキーマ
 - 作成, 14-80
 - セッションの変更, 10-12
 - 定義, 2-101
- スキーマ・オブジェクト, 2-101
 - 位置の保護, 15-2
 - 一覧, 2-101
 - オブジェクト型, 2-36
- 監査
 - オプション, 12-60
- 構造の検証, 12-41
- 再コンパイル, 9-3
- 再認可, 9-3
- 削除, 17-19
- 参照, 2-110, 10-12
- 所有者の保護, 15-2
- 代替名の提供, 15-2
- デフォルトのバッファ・プールの定義, 7-58
- 名前解決, 2-111

- ネーミング
 - 規則, 2-105
 - 計画, 2-109
 - 例, 2-109
- ネームスペース, 2-107
- パーティション索引, 2-103
- パーティション表, 2-103
- 部分, 2-103
- 他のスキーマ内, 2-112
- リモート・データベース, 2-112
- リモート・ビューへのアクセス, 13-35
- スター型変換, 2-100
- スタンドアロン・プロシージャ
 - 削除, 16-90
- スタンバイ・データベース
 - アーカイブ・ログの適用, 9-31
 - アクティブ, 9-44
 - 使用の制御, 9-51
 - プライマリ状態へのコミット, 9-45
 - マウント, 9-24
 - メディア・リカバリの設計, 9-26
 - リカバリ, 9-27, 9-28
- ストアド・ファンクション, 13-49
- スループット
 - 最適化, 2-88

せ

- 制御ファイル
 - 強制ロギング・モード, 13-19
 - 再作成, 13-14
 - 再利用可能, 13-17, 13-26
 - スタンバイの作成, 9-42
 - バックアップ, 9-43
- 制限 ROWID, 2-32
 - 互換性と移行, 2-33
- 整合性制約「制約」を参照
- 整数
 - SQL 構文内, 2-52
 - 一意の値の生成, 14-82
 - 構文, 2-52
 - 指定, 2-12
 - 精度, 2-52
- 精度
 - NUMBER データ型, 2-12
 - 桁数, 2-53

- 制約
 - CHECK, 7-15
 - REF 表, 7-16
 - REF 列, 7-16
 - 一意
 - 索引の属性, 7-21
 - 使用可能, 15-54
 - 違反の行の格納, 11-79
 - 外部キー, 7-13
 - 既存の制約の変更, 11-56
 - 削除, 11-11, 11-57, 17-11
 - 参照整合性, 7-13
 - 主キー, 7-12
 - 索引の属性, 7-21
 - 使用可能, 15-55
 - 使用可能, 15-52, 15-55
 - 使用禁止, 15-52
 - CASCADE, 15-57
 - 制限事項, 7-11
 - チェック
 - 各 DML 文の終わり, 7-18
 - トランザクションの終わり, 7-18
 - トランザクションの始まり, 7-19
 - 遅延可能, 7-18, 18-42
 - 規定, 10-11
 - 定義, 7-5, 15-7
 - 表, 15-26
 - 列, 15-26
 - トランザクション内の状態設定, 18-42
 - 名前の変更, 11-57
 - ビュー
 - 削除, 12-31, 17-22
 - 変更, 12-31
 - 表の作成後の無効化, 11-84
 - 表の作成後の有効化, 11-84
 - 表の作成時の無効化, 15-22
 - 表の作成時の有効化, 15-22
 - 表への追加, 11-56
 - 変更, 11-11
 - 制約の追加, 11-56
- セーブポイント
 - 指定, 18-2
 - 消去, 12-70
 - ロールバック, 17-98
- セキュリティ
 - 規定, 15-89

セグメント

領域管理

空きリストの使用, 15-83

自動, 15-83

手動, 15-83

ビットマップの使用, 15-83

セグメント属性句

CREATE TABLE, 15-15

セッション

CPU 時間の制限, 9-129

オブジェクト・キャッシュ, 10-9

グローバル・ネーム解決, 10-8

権限を持つユーザーに制限, 10-28

合計経過時間の制限, 9-129

終了, 10-27

制限, 10-30

切断, 10-26

タイムゾーンの設定, 10-14

同時セッションの数, 10-67

特性の変更, 10-6

非アクティブ期間の制限, 9-125

プライベート SGA 領域の制限, 9-130

別のインスタンスへの切替え, 10-13

読み込まれたデータ・ブロックの制限, 9-130

リソース・コスト制限の計算, 9-129

リソース・コスト制限の変更, 9-129

リソース・コストの制限, 9-129

リソースの合計の制限, 9-125

ロールの影響, 9-133

セッション制御文, 9-5

PL/SQL サポート, 9-5

セッション・パラメータ

INSTANCE, 10-13

PLSQL_DEBUG, 10-13

設定の変更, 10-11

セッション・ロック

解放, 10-27

線形リグレッション・ファンクション, 6-122

そ

相関副問合せ, 8-13

相関名

DELETE, 16-58

SELECT, 18-17

索引のベース表, 13-70

挿入

MERGE の使用, 17-76

従来型, 17-52

条件, 17-62

ダイレクト・パス, 17-52

単一表, 17-56

同時の更新, 17-75

マルチテーブルの例, 17-66

マルチ表, 17-62

ソート操作

言語処理手順の変更, 10-9

属性

オブジェクト型の最大数, 15-25

ディメンションからの削除, 9-58

ディメンションの追加, 9-58

ディメンションの定義, 13-45

ユーザー定義型

Java フィールドへのマップ, 16-12

た

大のテスト, 5-5

タイムスタンプ

ローカル・タイムゾーンへの変換, 4-10

タイムゾーン

書式化, 2-67

セッションの確認, 6-135

データの変換, 4-10

データベースへの指定, 13-33

ダイレクト・パス・インサート, 17-52

単一行ファンクション, 6-3

その他, 6-6

単一表, 17-56

単項演算子, 3-2

単純式, 4-4

単純比較条件, 5-6

ダンプ・ファイル

サイズの制限, 10-8

ち

チェックポイント

強制実行, 10-26

遅延可能制約, 18-42

中央値の値, 6-114

長期スタンバイ・リカバリ・モード, 9-30

つ

通貨記号

- ISO, 2-61
- セッションの設定, 10-8
- 連合, 2-63
- ローカル, 2-62

て

定数値「リテラル」を参照

ディスパッチャ・プロセス

- 終了, 10-107
- 追加プロセスの作成, 10-107

ディメンション

階層

- 削除, 9-58
- 追加, 9-58
- 定義, 13-43
- 変更, 9-56

作成, 13-41

システム権限の付与, 17-36

属性

- 削除, 9-58
- 追加, 9-58
- 定義, 13-45
- 変更, 9-56

データベースからの削除, 16-69

無効のコンパイル, 9-58

例, 13-45

レベル

- 削除, 9-58
- 追加, 9-58
- 定義, 13-43

レベルの定義, 13-42

ディレクトリ・オブジェクト

オペレーティング・システム・ディレクトリの別 名, 13-46

監査, 12-55

再定義, 13-47

作成, 13-46

システム権限の付与, 17-36

データベースからの削除, 16-71

ディレクトリ「ディレクトリ・オブジェクト」を参照 データ

UNDO

- 保存, 14-76

一時ファイルとしての指定, 15-24

検索, 8-2

集計

GROUP BY のグルーピング・セットの連結, 18-22

GROUP BY の複合列, 18-22

グルーピング・セット, 18-22

入力に対する整合性チェック, 2-12

非依存, 15-2

頻繁に使用されるキャッシュ, 11-36, 15-50

データ・オブジェクト番号

拡張 ROWID, 2-32

データ型, 2-2

ANSI のサポート, 2-5

BFILE, 2-9, 2-30

BLOB, 2-9, 2-30

CHAR, 2-8, 2-9

CLOB, 2-8, 2-31

DATE, 2-7, 2-17

INTERVAL DAY TO SECOND, 2-23

INTERVAL YEAR TO MONTH, 2-22

LONG, 2-7, 2-14

LONG RAW, 2-8, 2-25

NCHAR, 2-8, 2-10

NCLOB, 2-9, 2-31

NUMBER, 2-7, 2-12

NVARCHAR2, 2-7, 2-11

Oracle が提供する型, 2-38

RAW, 2-8, 2-25

ROWID, 2-8, 2-31

TIMESTAMP, 2-20

TIMESTAMP WITH LOCAL TIME ZONE, 2-21

TIMESTAMP WITH TIME ZONE, 2-20

UROWID, 2-8, 2-33

VARCHAR, 2-11

VARCHAR2, 2-7, 2-11

XML 型, 2-39

期間, 2-16

空間型, 2-41

組込み, 2-7

コレクション型値への変換, 6-25

統計情報の関連付け, 12-47, 12-48

長さセマンティクス, 2-9, 2-11

日時, 2-16

任意型, 2-38

比較規則, 2-42

別のデータ型への変換, 6-25

- メディア型, 2-41
- 文字, 2-9
- ユーザー定義, 2-35
- データ・セグメントの圧縮, 9-97, 11-34, 14-15, 15-29
- データ操作言語 (DML)
 - 影響される行の取出し, 16-59, 17-61, 18-64
 - 行の取出し, 16-59, 17-61, 18-64
 - 索引付け中の許可, 9-71
 - 操作
 - 索引再構築中, 11-83
 - 索引作成中, 13-75
 - トリガー, 15-93
 - 操作の制限, 10-30
 - トリガー
 - LOB 列および属性, 2-28
 - パラレル化, 15-50
 - 文, 9-4
 - PL/SQL サポート, 9-4
- データ定義言語 (DDL)
 - イベントおよびトリガー, 15-94
 - 排他的アクセスを必要とする文, 9-3
 - 文, 9-3
 - PL/SQL サポート, 9-3
 - 暗黙的コミット, 9-3
 - 再コンパイルの原因, 9-3
- データ・ディクショナリ
 - コメントの追加, 12-67
- データ・ファイル
 - 一時表領域の定義, 15-86
 - オフライン化, 9-36
 - オンライン化, 9-36
 - オンラインの情報更新, 10-26
 - オンライン・バックアップ, 11-102
 - オンライン・バックアップの終了, 9-36, 11-102
 - サイズ, 7-38
 - サイズの変更, 9-36
 - サイズ変更, 9-36
 - 再利用, 7-38
 - 削除, 17-11
 - システム生成, 9-35
 - 指定, 7-36
 - 表領域, 15-78
 - 自動拡張, 7-39
 - 自動拡張の有効化, 7-39
 - 新規作成, 9-35
 - 損失または破損した場合の再作成, 9-35
- データベースの定義, 13-25
- データベースへの指定, 13-29
- 名前の変更, 9-37
- 破損のリカバリ, 9-26
- 表領域の定義, 15-76
- メディア・リカバリの設計, 9-26
- リカバリ, 9-28
- 論理ボリュームおよび物理デバイスへのマッピング, 10-59
- データベース
 - FORCE LOGGING モード, 9-38
 - Oracle7 のデータ・ディクショナリからの変換, 9-49
 - REDO ログ生成を可能にする, 9-24
 - アーカイブ・モード
 - 指定, 13-28
 - アカウント
 - 作成, 16-31
 - アクティビティの再開, 10-29
 - アクティビティの停止, 10-29
 - アップグレード, 9-49
 - 一時ファイル
 - 変更, 9-35
 - イベント
 - 監査, 15-95
 - 透過的なログイン, 15-95
 - トリガー, 15-95
 - インスタンス, 13-28
 - オープン, 9-24, 13-22
 - メディア・リカバリ後, 9-25
 - オンライン
 - ログ・ファイルの追加, 9-39
 - 各国語キャラクタ・セット
 - 指定, 13-29
 - 管理リカバリ, 9-16
 - キャッシュ
 - バッファ, 10-45
 - キャラクタ・セット
 - 指定, 13-29
 - 変更, 9-47
 - キャラクタ・セットの指定, 13-29
 - キャラクタ・セットの変更, 9-47
 - 休止状態, 10-30
 - 強制ロギング・モード, 13-19, 13-28
 - グローバル名の変更, 9-50
 - 再作成の準備, 9-42
 - 作成, 13-22

- 時間ベースのリカバリ, 9-28
- システム権限の付与, 17-35
- システム・ユーザー・パスワード, 13-26
- 使用の制御, 9-51
- スクリプトを作成する, 9-42
- スタンバイ
 - ログ・ファイルの追加, 9-39
- スタンバイ状態へのコミット, 9-45
- すべてのデータの消去, 13-22
- 制御, 9-51
- 制御ファイルの再作成, 13-14
- 制御ファイルの再利用, 13-26
- 接続文字列, 2-113
- タイムゾーン
 - 判断, 6-48
 - 有効な値の設定, 9-48, 13-33
- データ非消失モード, 9-44
- データ・ファイル
 - 指定, 13-29
 - 変更, 9-35
- 特性の変更, 13-14
- 取消しベースのリカバリ, 9-28
 - 終了, 9-30
- 名前の変更, 13-14, 13-17
- ネーミング, 9-24
- 破損の再構成, 9-26
- バックアップの終了, 9-34
- ブロック
 - サイズの指定, 15-79
- 別のデータベースへのサブセットの移動, 11-77
- 変更, 9-12
- 変更禁止, 9-51
- 変更ベースのリカバリ, 9-28
- 保護モード, 9-44
- マウント, 9-24, 13-22
- メディア・リカバリの設計, 9-26
- ユーザーに対する無制限のリソース, 14-68
- ユーザーに対するリソース制限, 14-66
- ユーザーへの読み込み専用トランザクションの制限, 9-25
- 読み書き両用, 9-24
- 読み込み専用, 9-24
- リカバリ, 9-26, 9-27
 - テスト, 9-29
 - 破損ブロックの許容, 9-29
 - バックアップ制御ファイルの使用, 9-28

- リセット
 - 現行のログ順序, 9-25
 - 前回のバージョン, 9-49
- リモート
 - アクセス, 8-15
 - サービス名, 13-38
 - 接続, 13-37
 - 挿入, 17-58
 - 表ロック, 17-73
 - ユーザーの認証, 13-38
- ログ・ファイル
 - 指定, 13-26
 - 変更, 9-38
- データベース・オブジェクト
 - 削除, 17-19
 - スキーマ, 2-101
 - 非スキーマ, 2-102
- データベース・トリガー「トリガー」を参照
- データベース・リンク, 8-15
 - 共有, 13-36
 - クローズ, 10-3
 - 現行のユーザー, 13-37
 - 構文, 2-113
 - 作成, 2-112, 13-35
 - 参照, 2-114
 - システム権限の付与, 17-35
 - シノニムの作成, 15-5
 - データベースからの削除, 16-67
 - ネーミング, 2-113
 - パブリック, 13-36
 - 削除, 16-67
 - ユーザー名およびパスワード, 2-113
- データ変換, 2-46
 - 暗黙的
 - デメリット, 2-46
 - 暗黙的と明示的, 2-46
 - 暗黙的に行う場合, 2-47, 2-49
 - キャラクタ・データ型, 2-48
 - 明示的に行う場合, 2-49
- デカルト積, 8-11
- デバッグ
 - システム権限の付与, 17-35
- デフォルトの索引の抑制, 14-18

と

- 問合せ, 8-2, 18-4

- SELECT 構文のリスト, 8-2
- 値で戻された行のグループ化, 18-21
- 階層「階層問合せ」を参照
- 階層の順序付け, 18-24
- 外部結合, 18-17
- 過去のデータ, 18-14
- 行ロック, 18-25
- 結果のソート, 8-9
- 結合, 8-9, 18-17
- 構文, 8-2
- コメント, 8-3
- 上位 N 番, 2-83
- 相関
 - 左相関, 18-16
- 定義済, 8-2
- トップレベル, 8-2
- 任意の行からの選択, 18-15
- ヒント, 8-3
- 複合, 8-9
- 複数表の参照, 8-9
- 分散, 8-15
- 戻された行の順序, 18-23
- 等価結合, 8-10
 - ディメンションの定義, 13-44
- 等価性のテスト, 5-5, 5-11
- 統計情報
 - 関連付けの削除, 16-63
 - 厳密な計算, 12-37
 - 索引, 13-75
 - 索引再構築中の収集, 9-71
 - 索引の使用, 9-76
 - 推定, 12-39
 - スカラー・オブジェクト属性
 - 収集, 12-32
 - スキーマ・オブジェクト
 - 削除, 12-32
 - 収集, 12-32
 - データ・ディクショナリからの削除, 12-43
 - ユーザー定義
 - 削除, 16-75, 16-76, 16-89, 17-7, 17-14
- 統計タイプ
 - 関数との関連付け, 12-47, 12-48
 - 関連付け
 - 列, 12-48
 - 関連付けの解除
 - 型, 16-61
 - 索引タイプ, 16-61
 - ドメイン索引, 16-61
 - パッケージ, 16-61
 - ファンクション, 16-61
 - 列, 16-61
 - 索引タイプとの関連付け, 12-47, 12-48
 - データ型との関連付け, 12-47, 12-48
 - ドメイン索引との関連付け, 12-47, 12-48
 - パッケージへの関連付け, 12-47, 12-48
 - 特殊文字
 - パスワード, 14-70
 - ドメイン索引, 13-61, 13-79, 13-88
 - LONG 列, 11-49
 - 再構築, 9-71
 - 削除ルーチンの起動, 17-7
 - 作成のパラレル化, 13-80
 - 前提条件の作成, 13-79
 - データベースからの削除, 16-74
 - 統計情報の関連付け, 12-47, 12-48
 - 変更, 9-74
 - 文字列変更の指定, 9-74
 - ユーザー定義の CPU および I/O コストの判断, 17-23
 - 例, D-2
 - トランザクション
 - 暗黙的コミット, 9-3, 9-4, 9-5
 - インダウト
 - 強制実行, 12-71
 - コミット, 12-70
 - 変換, 18-49
 - 完了, 10-26
 - コメント, 12-71
 - 自動コミット, 12-70
 - 終了, 12-70
 - セーブポイント, 18-2
 - ネーミング, 18-49
 - 分散トランザクションの強制実行, 10-3
 - 分離レベル, 18-47
 - 変更の確定, 12-70
 - 読み書き両用, 18-47
 - 読込み専用, 18-47
 - ロールバック, 10-27, 14-76, 17-97
 - セーブポイント, 17-98
 - ロックの解除, 12-70
 - 割付け
 - ロールバック・セグメント, 18-47
 - トランザクション制御文, 9-4
 - PL/SQL サポート, 9-5

トリガー

- AFTER, 15-92
- BEFORE, 15-91
- DDL イベント, 15-94
- DML 操作, 15-90, 15-93
- INSTEAD OF, 15-92
 - 削除, 16-41
- SQL 例, 15-100
- 起動の順序, 15-93
- 行値
 - 新旧, 15-98
- 行の指定, 15-98
- コンパイル, 12-2, 12-4
- 再作成, 15-91
- 作成, 15-89
 - 複数, 15-93
- 実行
 - PL/SQL ブロック, 15-99
 - 外部プロシージャでの実行, 15-99
- 使用可能, 11-85, 12-2, 12-3, 15-89
- 使用禁止, 11-85, 12-2, 12-3
- 制限事項, 15-98
- データベース
 - 削除, 17-12, 17-19
 - 変更, 12-3
- データベース・イベント, 15-95
- データベースからの削除, 17-12
- 名前を変更, 12-3
- ビュー, 15-92
- 付与
 - システム権限, 17-41
- 文, 15-98

な

- 内部 LOB, 2-26
- 内部結合, 8-11, 18-18
- 夏時間, 2-24
 - 開始または終了, 2-24
 - 境界ケース, 2-24

に

- 日時式, 4-10
- 日時書式要素, 2-65
 - ISO 規格, 2-69
 - RR, 2-69

- 大文字化, 2-65
- グローバル化・サポート, 2-69
- 接尾辞, 2-71
- 日時データ型, 2-16
 - 夏時間, 2-24
- 日時の演算
 - 夏時間の演算, 2-24
- 日時ファンクション, 6-5
- 日時フィールド
 - 日時または期間値からの抽出, 6-58
- 日時列
 - DATE 列からの作成, 11-49

ね

- ネームスペース
 - オブジェクトのネーミング規則, 2-107
 - スキーマ・オブジェクト, 2-107
 - 非スキーマ・オブジェクト, 2-108
- ネステッド・ループ結合
 - 最適化, 10-9
- ネストした表, 2-37
 - VARRAY との比較, 2-46
 - 記憶特性, 11-44, 15-40
 - 索引構成表としての定義, 11-44
 - 索引列, 13-70
 - 作成, 16-3, 16-9
 - 仕様部の削除, 17-14
 - 比較規則, 2-46
 - ビュー内での更新, 15-92
 - 変更, 11-55
 - 本体の削除, 17-17
 - マテリアライズド・ビュー, 14-10, 14-11
 - マルチレベル, 15-40
 - 戻り値の変更, 11-55
 - 列プロパティの変更, 11-12
- ネストした副問合せ, 8-12

は

- パーティション
 - LOB 記憶特性, 11-47
 - 値の変更, 18-60
- エクステンツ
 - 索引への割当て, 9-68
 - エクステンツの割当て, 11-35
 - 記憶特性, 7-50

- 行の削除, 11-71, 16-56
- 行の挿入, 17-58
- 行の追加, 17-52
- コンボジット, 2-103
 - 指定, 15-47
- 索引, 13-76
- 削除, 11-70
- 挿入操作のロギング, 11-35
- 追加, 11-58
- 名前の変更, 11-71
- ハッシュ, 2-103
 - 合せる, 11-69
 - 指定, 15-44
 - 追加, 11-68
- 非パーティション表への変換, 11-77
- 表との交換, 11-25
- 表領域
 - 定義, 15-28
- 物理属性
 - 変更, 11-33
- 分割, 11-72
- 別のセグメントへの移動, 11-64
- 変更, 11-58, 11-60
- マージ, 11-75
- 未使用領域の解放, 11-36
- リスト・パーティションの追加, 11-69
- リテラル値に基づくパーティション, 15-46
- レンジ, 2-103
 - 指定, 15-42
 - 追加, 11-66
- ロギング属性, 15-28
- ロック, 17-71
- パーティション化
 - 句
 - ALTER INDEX, 9-65
 - ALTER TABLE, 11-58
 - マテリアライズド・ビュー, 9-97, 14-7, 14-17
 - マテリアライズド・ビュー・ログ, 9-111, 14-36
 - レンジ, 15-18
 - パーティション化された索引構成表
 - 2次索引の更新, 9-78
 - パーティション交換
 - 制限事項, 11-80
 - パーティション索引, 2-103, 13-61, 13-78
 - ユーザー定義, 7-22, 13-76, 15-56
 - ローカルの作成, 13-65
 - パーティション表, 2-103
 - バイナリ演算子, 3-2
 - バイナリ・ラージ・オブジェクト「BLOB」を参照
 - 配布
 - ヒント, 2-98
 - パスワード
 - 期限切れ, 16-35
 - 使用および再利用の制限, 14-70
 - 使用禁止化, 14-70
 - 特殊文字, 14-70
 - パラメータ
 - ALTER PROFILE, 14-71
 - CREATE PROFILE, 14-67
 - 複雑さに対する保証, 14-70
 - 猶予期間, 14-70
 - ロック, 14-70
 - パッケージ
 - 再定義, 14-49
 - 作成, 14-48
 - 実行時のコンパイルの防止, 9-119
 - 実行者権限, 14-50
 - シノニム, 15-2
 - スキーマと権限の指定, 14-50
 - データベースからの削除, 16-88
 - 統計情報の関連付け, 12-47, 12-48
 - 統計タイプの削除, 16-89
 - 明示的な再コンパイル, 9-119
 - パッケージ・プロシージャ
 - 削除, 16-90
 - パッケージ本体
 - 再作成, 14-54
 - 作成, 14-53
 - データベースからの削除, 16-88
 - ハッシュ・クラスタ
 - 1つの表、作成, 13-6
 - 作成, 13-6
 - ハッシュ・ファンクションの指定, 13-7
 - ハッシュ結合
 - 使用可能および使用禁止, 10-8
 - メモリーの割当て, 10-8
 - ハッシュ・パーティション
 - 合せる, 11-62
 - 追加, 11-68
 - ハッシュ・パーティション化句
 - CREATE TABLE, 15-22, 15-44
 - パフォーマンス
 - 索引アクセス・パスに対する最適化, 10-9
 - セッション・オブティマイザ・アプローチ, 10-9

- ネステッド・ループ結合に対して最適化する, 10-9
- パブリック・シノニム, 15-3
 - 削除, 17-4
- パブリック・データベース・リンク
 - 削除, 16-67
- パブリック・ロールバック・セグメント, 14-78
- パラメータ
 - 構文
 - オプション, A-3
 - 必須, A-3
- パラメータ・ファイル
 - 作成, 14-58
- パラレル結合
 - PQ_DISTRIBUTE ヒント, 2-98
- パラレル実行
 - DDL 文, 10-4
 - DML 文, 10-4
 - ヒント, 2-97
- 範囲条件, 5-12

ひ

- ヒープ構成表
 - 作成, 15-7
- 比較関数
 - MAP, 16-29
 - ORDER, 16-29
- 比較条件, 5-4
- 比較セマンティクス
 - 空白埋め, 2-44
 - 非空白埋め, 2-43
 - 文字列, 2-43
- 引数
 - 演算子, 3-1
- 非空白埋め比較セマンティクス, 2-43
- 非スキーマ・オブジェクト
 - 一覧, 2-102
 - ネームスペース, 2-108
- ヒストグラム
 - 等幅の作成, 6-198
- 左側外部結合, 18-18
- 日付
 - 算術, 2-19
 - 比較規則, 2-42
- 日時の演算
 - 境界ケース, 10-12
- 日付書式モデル, 2-64

- 句読点, 2-65
- テキスト, 2-65
- 日付ファンクション, 6-5
- ビット・ベクトル
 - 数値への変換, 6-23
- ビットマップ索引, 13-68
 - 結合索引の作成, 13-63
- ビュー
 - XMLType, 16-47
 - XMLType の作成, 16-52
 - XMLType の間合せ, 16-47
 - オブジェクト・サブビューの作成, 16-45
 - オブジェクトの作成, 16-44
 - 結合
 - キー保存表, 16-47
 - 結合を使用した更新可能化, 16-47
 - 更新可能, 16-46
 - 再コンパイル, 12-29
 - 再作成, 16-41
 - 削除
 - 実表の行, 16-53
 - データベース, 17-21
 - 作成
 - コメント, 12-67
 - 実表の前, 16-42
 - 複数, 14-80
 - 実表
 - 行の追加, 17-52
 - シノニム, 15-2
 - 制約の削除, 12-31
 - 制約の変更, 12-31
 - 定義, 16-38
 - データ検索, 18-4
 - 名前の変更, 17-84
 - 副間合せ, 16-45
 - 制限, 16-48
 - 付与
 - システム権限, 17-42
 - 変更
 - 実表の値, 18-56
 - 定義, 17-21
 - リモート・ビューへのアクセス, 13-35
- ビュー制約
 - 削除, 17-22
- 表
 - LOB 記憶特性, 7-50
 - SQL 例, 15-62

- XMLType の作成, 15-61
- XMLType の問合せ, 15-61
- 新しいセグメントへの移動, 11-82
- 移行行および連鎖行, 12-42
- 一時
 - セッション固有, 15-24
 - データの存続期間, 15-27
 - トランザクション固有, 15-24
- 移動, 11-30
- エクステンツの割当て, 11-35
- オブジェクト
 - 作成, 15-9
- オブジェクトの問合せ, 15-59
- 外部, 15-30
 - 作成, 15-32
 - 制限事項, 15-33
- 外部構成, 15-30
- 記憶域属性
 - 定義, 15-7
- 記憶域プロパティ, 15-27, 15-35
- 記憶特性
 - 定義, 7-50
- 既存値の変更, 18-56
- キャッシュへのブロックの保存, 11-36, 15-50
- 行の削除, 16-53
- 行の追加, 17-52
- クラスタへの割当て, 15-34
- 構成の定義, 15-30
- 構造の検証, 12-41
- コメント, 12-68
- コメントの作成, 12-67
- 索引構成, 15-30
 - オーバーフロー・セグメント, 15-32
 - 索引ブロックの領域, 11-40, 15-31
- 索引構成表の移動, 11-82
- 削除
 - クラスタ, 16-65
 - 索引, 17-7
 - 所有者, 17-19
 - パーティション, 17-7
- 作成, 15-7
 - 複数, 14-80
- サブパーティション属性, 11-59
- シノニム, 15-2
- 使用禁止の索引, 10-14
- 制限
 - ブロックのレコード, 11-38
 - ダイレクト・パスを使用した挿入, 17-52
- データ検索, 18-4
- データベースからの削除, 17-6
- データベースの外部へのデータの格納, 15-32
- デフォルト物理属性
 - 変更, 11-33
- 問合せ内で結合, 18-17
- 統計情報の収集, 11-37, 12-35
- 統計タイプの削除, 17-7
- 名前の変更, 11-38, 17-84
- ネスト
 - 記憶特性, 15-40
 - 作成, 16-19
- パーティション化, 2-103, 15-7, 15-42
 - 行をパーティション間で移動可能にする, 11-39
 - デフォルト属性, 11-59
- パーティション属性, 11-59
- パラレル化
 - デフォルト値の設定, 15-50
- パラレル作成, 15-50
- ヒープ構成, 15-30
- 非クラスタ化, 16-64
- ビューを介した更新, 16-46
- 表領域
 - 定義, 15-7, 15-28
- 副問合せへの行の挿入, 15-58
- 物理属性
 - 変更, 11-33
- 付与
 - システム権限, 17-40
- 並列度
 - 指定, 15-7
- 並列度の変更, 11-81
- 別名, 2-115
 - CREATE INDEX, 13-70
 - DELETE, 16-58
- 未使用領域の解放, 11-36
- リモート・ビューへのアクセス, 13-35
- リレーショナル
 - 作成, 15-8
- ロギング
 - 挿入操作, 11-35
 - 表作成, 15-28
- ロック, 17-71
- 標準 SQL, B-1
- Oracle 拡張機能, B-11

- 表セグメント
 - データ圧縮, 11-34, 15-29
- 表のファンクション
 - 作成, 13-56
- 表パーティション・セグメント
 - 圧縮, 11-34, 15-29
- 表領域, 11-101
 - FORCE LOGGING モード, 11-104, 15-80
 - UNDO
 - 削除, 17-10
 - 作成, 13-32, 15-77
 - 変更, 11-99
 - 一時
 - 作成, 15-86
 - ユーザーへの指定, 16-35
 - 一時オブジェクト, 15-81
 - 一時ファイル
 - 追加, 11-99
 - 永続オブジェクト, 15-81
 - エクステンツ管理, 15-81, 15-88
 - エクステンツ・サイズ, 15-79
 - オフライン化, 11-101, 15-81
 - オンライン化, 11-101, 15-81
 - オンライン・バックアップの終了, 11-103
 - 書き込み操作を実行可能にする, 11-103
 - コンテンツの削除, 17-10
 - 作成, 15-75
 - システム権限の付与, 17-40
 - 指定
 - 索引再構築, 11-83
 - データ・ファイル, 15-78
 - 表, 15-27
 - ユーザー, 16-34
 - 自動セグメント領域管理, 2-15, 15-83
 - セッション存続期間, 15-86
 - 損失または破損した場合の再構成, 9-26, 9-35
 - データ・ファイル
 - 追加, 11-99
 - renaming, 11-99
 - データ・ファイルのバックアップ, 11-102
 - データベースからの削除, 17-9
 - データベースの一時定義, 13-25
 - デフォルトの一時, 9-49
 - 名前の認識, 9-49
 - 変換
 - 一時的から永続的, 11-103
 - 永続的から一時的, 11-103

- 未使用エクステンツの結合, 11-104
- 未使用のエクステンツ・サイズ, 11-101
- メディア・リカバリの設計, 9-26
- ユーザーの領域の割当て, 16-35
- 読み込み専用, 11-103
- リカバリ, 9-26, 9-28
- ローカル管理, 7-54
 - 一時, 15-88
 - 変更, 11-99
- ロギング属性, 11-104, 15-80
- 表ロック
 - EXCLUSIVE, 17-72, 17-73
 - ROW EXCLUSIVE, 17-72, 17-73
 - ROW SHARE, 17-72, 17-73
 - SHARE, 17-72
 - SHARE ROW EXCLUSIVE, 17-73
 - SHARE UPDATE, 17-73
 - 継続期間, 17-71
 - サブパーティション, 17-72
 - 使用可能, 11-85
 - 使用禁止, 11-85
 - 問合せ, 17-71
 - パーティション, 17-72
 - モード, 17-73
 - リモート・データベース, 17-73
- ヒント, 8-3
 - ALL_ROWS ヒント, 2-88
 - AND_EQUAL ヒント, 2-88
 - CACHE ヒント, 2-89
 - CLUSTER ヒント, 2-89
 - FIRST_ROWS ヒント, 2-91
 - FULL ヒント, 2-91
 - HASH ヒント, 2-91
 - INDEX_ASC ヒント, 2-92
 - INDEX_DESC ヒント, 2-92
 - INDEX ヒント, 2-92
 - NO_EXPAND ヒント, 2-94
 - NO_MERGE ヒント, 2-95
 - NO_PUSH_PRED ヒント, 2-95
 - NO_PUSH_SUBQ ヒント, 2-95
 - NOCACHE ヒント, 2-94
 - NOPARALLEL ヒント, 2-95
 - NOREWRITE ヒント, 2-96
 - ORDERED ヒント, 2-96
 - PARALLEL ヒント, 2-97
 - PQ_DISTRIBUTE ヒント, 2-98
 - PUSH_PRED ヒント, 2-98

PUSH_SUBQ ヒント, 2-98
REWRITE ヒント, 2-99
ROWID ヒント, 2-99
RULE ヒント, 2-99
SQL 文, 2-86
USE_CONCAT ヒント, 2-100
USE_MERGE ヒント, 2-101
USE_NL ヒント, 2-101
オブティマイザの受渡し, 18-56
構文, 2-86

ふ

ファイル

REDO ログ・ファイル・グループとしての指定,
7-36
一時ファイルとしての指定, 7-36
データ・ファイルとしての指定, 7-36

ファンクション

権限, 12-14
スキーマ, 12-14
3GL、コール, 14-2
「SQL ファンクション」を参照
built_in
式, 4-11
COMMIT または ROLLBACK 文の発行, 10-4
DECODE, 6-48
user_defined
式, 4-11
外部, 13-49, 14-60
格納戻り値, 12-65
結果の戻りの繰返し, 13-56
権限, 16-10
コール, 12-63
コレクションの戻り, 13-56
再作成, 13-52, 13-91
索引の定義, 13-70
実行, 12-63
パラレル問合せプロセス, 13-56
実行時のコンパイルの防止, 9-59
シノニム, 15-2
スキーマ, 16-10
スキーマとユーザー権限の指定, 13-54
ストアド, 13-49
線形リグレッション, 6-122
宣言の変更, 13-52
定義の変更, 13-52

データベースからの削除, 16-72
統計情報の関連付け, 12-47, 12-48
統計情報、デフォルト・コストの割当て, 12-47
統計情報、デフォルト選択性の定義, 12-47
日時, 6-5
ネーミング規則, 2-109
パーティション化
パラレル問合せプロセス, 13-56
表, 13-56
分析
ユーザー定義, 13-57
保存されたコピーの使用, 13-55
無効の再コンパイル, 9-59
明示的な再コンパイル, 9-60
戻り値のデータ型, 13-54
ユーザー定義, 6-211
集計, 13-57
例, 13-58
ファンクション索引, 13-61
クエリー・リライト, 10-10
作成, 13-70
使用可能, 9-71, 9-75, 10-98
使用可能および使用禁止, 9-71
使用禁止, 10-98
リフレッシュ, 9-47
ファンクション式
組込み, 4-11
ユーザー定義, 4-11
不完全なオブジェクト型, 16-3
作成, 16-3, 16-4
複合外部キー, 7-13
複合式, 4-5
複合主キー, 7-12
複合条件, 5-21
副問合せ, 8-2, 8-12, 18-4
インライン・ビュー, 8-12
拡張された副問合せのネスト解除, 8-14
過去のデータ, 18-14
式のかわりに使用, 4-13
スカラー, 4-13
式, 4-13
相関, 8-13
定義済, 8-2
名前の割当て, 18-10
ネスト, 8-12
ネスト解除, 8-14
表データの挿入, 15-58

- ファクタリング, 18-10
- 副問合せを含む, 8-12
- 副問合せのネスト解除, 8-14
- 物理スタンバイ・データベース
 - アクティブ, 9-44
- 物理属性句
 - ALTER CLUSTER, 9-8
 - ALTER INDEX, 9-63, 9-69
 - ALTER MATERIALIZED VIEW LOG, 9-110
 - ALTER TABLE, 11-33
 - CREATE CLUSTER, 13-3
 - CREATE MATERIALIZED VIEW, 14-9
 - CREATE TABLE, 15-16, 15-27
- 不等価性のテスト, 5-11
- 浮動小数点数, 2-12, 2-13
- 不等性のテスト, 5-5
- 負の位取り, 2-13
- プライベート・アウトライン
 - オプティマイザによる使用, 10-15
- フラッシュバック問合せ, 18-14
 - 挿入を指定した使用, 17-56, 18-62
- プラン・スタビリティ, 14-44
- プリコンパイル
 - Oracle, 1-3
- フルスペルで表した数
 - 指定, 2-71
- プロキシ句
 - ALTER USER, 12-23, 12-25
- プロシージャ
 - 3GL、コール, 14-2
 - COMMIT または ROLLBACK 文の発行, 10-4
 - 依存するローカル・オブジェクトの無効化, 16-90
 - 外部, 14-58, 14-60
 - リモート・データベースからの実行, 14-3
 - 権限, 12-14, 16-10
 - コール, 12-63
 - 再コンパイル, 9-122
 - 再作成, 14-62
 - 作成, 14-58, 14-60
 - 実行, 12-63
 - 実行時のコンパイルを回避する, 9-123
 - シノニム, 15-2
 - スキーマ, 12-14, 16-10
 - スキーマと権限の指定, 14-63
- 宣言
 - C 関数, 14-64
 - Java メソッド, 14-64

- データベースからの削除, 16-90
- ネーミング規則, 2-109
- 付与
 - システム権限, 17-38
 - 明示的にコンパイルする, 9-123
- プロファイル
 - 作成, 14-66
 - 例, 14-71
 - データベースからの削除, 16-92
 - 付与
 - システム権限, 17-38
 - 変更の例, 9-127
 - ユーザーへの割当て, 16-35
 - ユーザーへの割当ての削除, 16-92
 - リソース制限の削除, 9-125
 - リソース制限の追加, 9-125
 - リソース制限の変更, 9-125
- 分散問合せ, 8-15
 - 制限事項, 8-15
- 分析関数
 - ユーザー定義, 13-57
- 分析ファンクション, 6-9
 - AVG, 6-20
 - CORR, 6-35
 - COUNT, 6-38
 - COVAR_POP, 6-40
 - COVAR_SAMP, 6-42
 - CUME_DIST, 6-44
 - DENSE_RANK, 6-51
 - FIRST, 6-62
 - FIRST_VALUE, 6-64
 - LAG, 6-75
 - LAST, 6-76
 - LAST_VALUE, 6-79
 - LEAD, 6-81
 - MAX, 6-89
 - MIN, 6-91
 - NTILE, 6-103
 - OVER 句, 6-9, 6-11
 - PERCENT_CONT, 6-112
 - PERCENT_DISC, 6-114
 - PERCENT_RANK, 6-110
 - RANK, 6-116
 - RATIO_TO_REPORT, 6-118
 - ROW_NUMBER, 6-131
 - STDDEV, 6-140
 - STDDEV_POP, 6-141

STDDEV_SAMP, 6-142
SUM, 6-145
VAR_POP, 6-193
VAR_SAMP, 6-194
VARIANCE, 6-196
逆分散, 6-112, 6-114
構文, 6-9
線形リグレーション, 6-122
ユーザー定義, 6-11

へ

別名

問合せおよび副問合せでの指定, 18-17
ビュー問合せの式, 16-43
列, 8-3

変換

ファンクション, 6-6
文字列から日付への規則, 2-74

変数式, 4-15

ほ

本文

CHAR および VARCHAR2 データ型のプロパティ,
2-52
SQL 構文内, 2-51
構文, 2-51
日付および数値書式, 2-58

ま

マスター・データベース, 14-4

マスター表, 14-4

マッピング表

索引構成表, 11-83, 15-31
変更, 11-41

マテリアライズド結合ビュー, 14-31

マテリアライズド・ビュー, 9-100, 14-18

LOB 記憶域属性, 9-97

ROWID, 14-21

ROWID の値

マスター表への記録, 9-112

ROWID ベースから主キー・ベースへの変更, 9-102

移入, 14-18

オブジェクト型マテリアライズド・ビューの作成,
14-13

完全リフレッシュ, 9-101, 14-20

記憶域属性, 14-15

変更, 9-96

キャッシュへのブロックの保存, 9-99

強制リフレッシュ, 9-101

クエリー・リライト

資格, 7-21

使用可能および使用禁止, 9-103

クエリー・リライトの使用可能 / 使用禁止, 14-23

結合, 14-31

更新可能, 14-23

高速リフレッシュ, 9-100, 14-19, 14-20

コメントの作成, 12-67

再検証, 9-104

索引特性

変更, 9-98

作成, 14-4

シノニム, 15-2

主キー, 14-21

マスター表の値の記録, 9-112

主キー・ベースへの変更, 9-113

制約, 7-21

次の COMMIT でのリフレッシュ, 9-101, 14-20

データ・ウェアハウス, 14-4

データ検索, 18-4

データベースからの削除, 16-81

デフォルトの索引の作成の抑制, 14-18

パーティション, 9-97

副問合せ, 14-24

物理属性, 14-15

変更, 9-96

並列度, 9-98, 9-111

作成, 14-17

マスター表の DML 後でのリフレッシュ, 9-101,
14-20

マスター表の削除, 16-82

メンテナンスする索引, 14-18

有効範囲の制限, 14-14

リフレッシュ, 9-47

リフレッシュ時刻

変更, 9-100

リフレッシュ中に再作成, 9-101

リフレッシュ・モード

変更, 9-100

例, 14-26, 14-38

レプリケーション, 14-4

ロギングの変更, 9-98

- マテリアライズド・ビュー・セグメント
 - データ圧縮, 9-97, 14-15
- マテリアライズド・ビュー・パーティション・セグメント
 - 圧縮, 9-97, 14-15
- マテリアライズド・ビュー・ログ, 14-31
 - ROWID に基づく, 9-113
 - 新しい値の除外, 9-113
 - 新しい値の保存, 9-113
 - オブジェクト ID に基づく, 9-113
 - 記憶域属性
 - 指定, 14-35
 - 高速リフレッシュに必要な, 14-31
 - 作成, 14-31
 - 作成の平行化, 14-36
 - データベースからの削除, 16-83
 - パーティション, 14-36
 - 物理属性
 - 指定, 14-35
 - 物理属性の変更, 9-111
 - 古い値の保存, 14-38
 - 列の追加, 9-112
 - ロギングの変更, 9-112
- マルチスレッド・サーバー「共有サーバー」を参照
- マルチテーブル・インサート, 17-62
 - 条件, 17-62
 - 無条件, 17-62
 - 例, 17-66
- マルチレベル・コレクション, 15-40

み

- 右側外部結合, 18-18
- 未ソート索引, 13-74

め

- 明示的なデータ変換, 2-46, 2-49
- メソッド
 - サブタイプでの上書きの回避, 16-14
 - 実装なし, 16-14
 - スーパータイプのメソッドの上書き, 16-14
 - スタティック, 16-13
- メディア型
 - ORDSYS.ORDAudio, 2-41
 - ORDSYS.ORDDoc, 2-41
 - ORDSYS.ORDImage, 2-41

- ORDSYS.ORDVideo, 2-41
- メディア・リカバリ
 - 起動時のメディア・リカバ리를回避する, 9-36
 - 指定の REDO ログ, 9-26
 - 準備, 9-38
 - 使用禁止, 9-34
 - 使用中の実行, 9-30
 - スタンバイ・データベース, 9-26
 - 制限事項, 9-26
 - 設計, 9-26
 - 長期スタンバイ・リカバリ, 9-30
 - データ・ファイル, 9-26
 - データベース, 9-26
 - 平行化, 9-29
 - 表領域, 9-26
- メンバーシップ条件, 5-10

も

- 文字
 - 単一文字の比較規則, 2-44
- 文字長セマンティクス, 11-50
- 文字ファンクション, 6-4, 6-5
- 文字リテラル「Text」を参照
- 文字列
 - ASCII 値への変換, 6-18
 - Unicode への変換, 6-32
 - 各国語キャラクタ・セット, 2-10
 - 可変長, 2-11, 2-14
 - 完全一致, 2-72
 - 固定長, 2-9
 - 長さが 0 (ゼロ), 2-10
 - 比較規則, 2-43

ゆ

- ユーザー
 - SQL 例, 16-36
 - アカウントのロック, 16-35
 - 一時表領域, 16-35
 - 外部, 14-74, 16-33
 - グローバル, 14-74, 16-34
 - グローバル認証の変更, 12-24
 - 最大同時ユーザー数, 10-67
 - 作成, 16-31
 - データベースからの削除, 17-19
 - データベース・リンク, 13-37

デフォルトの表領域, 16-34
認証の変更, 12-26
パスワードの期限切れ, 16-35
表およびビューへのアクセスの制限, 17-71
付与

システム権限, 17-42
リモート・サーバーの認証, 13-38
領域の割当て, 16-35
ローカル, 14-74, 16-33
割付け

デフォルト・ロール, 12-25
プロファイル, 16-35

ユーザー定義演算子, 3-6

ユーザー定義型, 2-35

Java クラスへのマップ, 16-11
定義, 16-9

ユーザー定義の集計ファンクション, 13-57

ユーザー定義の統計情報

削除, 16-75, 16-76, 16-89, 17-7, 17-14

ユーザー定義ファンクション, 6-211

制限, 13-52

名前の優先順位, 6-213

ネーミング規則, 6-214

優先順位

演算子, 3-2

条件, 5-4

ユニバーサル ROWID「UROWID」を参照

ユリウス日, 2-19

よ

予約語, 2-106, C-1

ら

ラージ・オブジェクト「LOB データ型」を参照

ライセンス

制限の変更, 10-67, 10-68

ライブラリ

再作成, 14-2

作成, 14-2

システム権限の付与, 17-37

データベースからの削除, 16-80

ライブラリ・ユニット「Java スキーマ・オブジェクト」
を参照

り

リカバリ

使用中のメディアの実行, 9-30

中断後のインスタンスの継続, 9-26

データの廃棄, 9-24

データベース, 9-14

パラレル化, 9-29

分散、可能, 10-28

メディアの設計, 9-26

リカバリ句

ALTER DATABASE, 9-14

リスト・パーティション

追加, 11-61

リスト・パーティション化

値の削除, 11-62

値の追加, 11-62

デフォルト・パーティション

削除, 11-63

追加, 11-63

デフォルト・パーティションとデフォルト以外の
パーティションのマージ, 11-75

デフォルト・パーティションの作成, 15-46

デフォルト・パーティションの追加, 11-69

デフォルト・パーティションの分割, 11-72

パーティションの作成, 15-46

パーティションの追加, 11-63, 11-69

リスナー

登録, 10-31

リソース・パラメータ

CREATE PROFILE, 14-67

リソース・マネージャ, 10-30

リテラル

SQL 構文内, 2-51

SQL 文およびファンクション, 2-51

リリース 7.3.4

Oracle9i リリース 2 へのアップグレード, 9-25

レシーショナル表

作成, 15-8, 15-24

る

累積分布, 6-44

ルーチン

コール, 12-63

実行, 12-63

れ

列

LOB

記憶域属性, 11-45

REF

記述, 7-16

値の制限, 7-5

親子関係, 13-41

記憶域の変更, 11-43

記憶域プロパティ, 15-35

既存の制約の変更, 11-48

コメント, 12-68

コメントの作成, 12-67

最大数, 15-25

索引に基づく, 13-70

修飾名, 8-2

主キーとしての指定, 7-12

制約の指定, 15-26

置換可能な識別型, 6-155

追加, 11-42

定義, 15-7

デフォルト値の指定, 15-26

統計情報の関連付け, 12-48

統計情報の収集, 12-37

名前の変更, 11-54

表からの削除, 11-51

プロパティの変更, 11-12, 11-43

別名, 8-3

列 REF 制約

CREATE TABLE, 15-26

列制約

制限事項, 11-50

レプリケーション

行レベル依存の追跡, 13-8, 15-52

レベル

ディメンションからの削除, 9-58

ディメンションの追加, 9-58

ディメンションの定義, 13-43

連結演算子, 3-4

連鎖行

クラスタ, 12-37

リスティング, 12-42

レンジ・パーティション

値, 15-43

作成, 15-42

追加, 11-66

ろ

ローカル管理表領域

記憶域属性, 7-54

変更, 11-99

ローカル・パーティション索引, 13-78

ローカル・ユーザー, 14-74, 16-33

ロール

AQ_ADMINISTRATOR_ROLE, 17-44

AQ_USER_ROLE, 17-44

CONNECT, 17-44

DBA, 17-44

DELETE_CATALOG_ROLE, 17-44

EXECUTE_CATALOG_ROLE, 17-44

EXP_FULL_DATABASE, 17-44

HS_ADMIN_ROLE, 17-44

IMP_FULL_DATABASE, 17-44

RECOVERY_CATALOG_OWNER, 17-44

DELETE_CATALOG_ROLE, 17-44

SNMPAGENT, 17-44

アプリケーション, 9-133

エンタープライズ・ディレクトリ・サービスを使用した識別, 14-74

外部からの識別, 14-74

作成, 14-73

使用可能

現行セッション, 18-44, 18-45

使用禁止

現行セッション, 18-44, 18-45

データベースからの削除, 16-94

取消し, 17-86

PUBLIC, 17-89

別のロール, 16-94, 17-89

ユーザー, 16-94, 17-89

認証

エンタープライズ・ディレクトリ・サービス, 14-74

外部サービス, 14-74

データベース, 14-74

パスワード, 14-74

変更, 9-132

パスワードによる識別, 14-74

パッケージを使用した識別, 14-74

付与, 17-27

PUBLIC, 17-31

システム権限, 17-38

別のロール, 17-30

- ユーザー, 17-30
- ユーザー・セッションへの効果, 9-133
- RESOURCE, 17-44
- ロールバック・セグメント
 - SQL 例, 14-79
 - オフライン化, 9-134, 9-135
 - オンライン化, 9-134, 9-135
 - 記憶特性, 14-79
 - 記憶特性の変更, 9-134, 9-136
 - サイズの縮小, 9-134, 9-136
 - 最適なサイズの指定, 7-57
 - 作成, 14-76
 - システム生成, 14-76
 - データベースからの削除, 16-95
 - パブリック, 14-78
 - 表領域の指定, 14-78
 - 付与
 - システム権限, 17-39
- ロギング
 - REDO ログ・サイズ, 7-43
 - 最小限度の指定, 7-43
 - サブリメンタル
 - 削除, 9-41
 - 補助ログ・グループの削除, 11-35
 - 補助ログ・グループの追加, 11-35
- ログ・グループ
 - 削除, 11-35
- ログ・データ
 - 更新操作中の収集, 9-41
- ログ・ファイル
 - 削除, 9-38
 - セッション・パスの設定, 10-8
 - 追加, 9-38
 - データベースへの指定, 13-26
 - 登録, 9-45
 - 名前の変更, 9-37
 - 変更, 9-38
- ロック
 - 自動
 - 変更, 17-71
- ロック「表ロック」を参照
- 論理条件, 5-8
- 論理スタンバイ・データベース
 - アクティブ, 9-44
 - 異常終了, 9-47
 - 停止, 9-47

