

Oracle8*i*

概要 Vol.1

リリース 8.1

ORACLE[®]

Oracle8i 概要 Vol.1 リリース 8.1

部品番号 : A62756-1

第 1 版 1999 年 5 月

原本名 : Oracle8i Concepts, Volume 1, Release 8.1.5

原本部品番号 : A67782-01

原本著者 : Lefty Leverenz, Diana Rehfield

原本協力者 : Steve Bobrowski, Cynthia Chin-Lee, Cindy Closkey, Bill Creekbaum, Jason Durbin, John Frazzini, Richard Mateosian, Denis Raphaely, Danny Sokolsky, Richard Allen, David Anderson, Andre Bakker, Bill Bridge, Atif Chaudry, Jeff Cohen, Benoit Dageville, Sandy Dreskin, Ahmed Ezzat, Jim Finnerty, Diana Foch-Lorentz, Anurag Gupta, Gary Hallmark, Michael Hartstein, Terry Hayes, Alex Ho, Chin Hong, Ken Jacobs, Sandeep Jain, Amit Jasuja, Hakan Jakobsson, Robert Jenkins, Jr., Ashok Joshi, Mohan Kamath, Jonathan Klein, R. Kleinro, Robert Kooi, Vishu Krishnamurthy, Muralidhar Krishnaprasad, Andre Kruglikov, Tirthankar Lahiri, Juan Loaiza, Brom Mahbod, William Maimone, Andrew Mendelsohn, Reza Monajemi, Mark Moore, Rita Moran, Denise Oertel, Mark Porter, Maria Pratt, Tuomas Pystynen, Patrick Ritto, Hasan Rizvi, Sriram Samu, Hari Sankar, Gordon Smith, Leng Leng Tan, Lynne Thieme, Alvin To, Alex Tsukerman, William Waddington, Joyo Wijaya, Linda Willis, Andrew Witkowski, Mohamed Zait

Copyright © 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラムの使用、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。

当ソフトウェア（プログラム）のリバース・エンジニアリング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 Legend が適用されます。

Restricted Rights Legend

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs 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 52.227-14, Rights in Data -- General, including Alternate III (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

Vol.1

はじめに	XXV
------------	-----

第 I 部 Oracle の概要

1 Oracle Server の基礎知識

データベースと情報管理	1-2
Oracle Server	1-4
Oracle データベース	1-8
データベースの構造と領域管理	1-8
論理データベース構造	1-8
物理データベース構造	1-11
メモリー構造とプロセス	1-13
メモリー構造	1-13
プロセスのアーキテクチャ	1-16
プログラム・インタフェース	1-19
Oracle の動作例	1-20
データベース管理のオブジェクト・リレーショナル・モデル	1-21
リレーショナル・モデル	1-21
オブジェクト・リレーショナル・モデル	1-21
スキーマとスキーマ・オブジェクト	1-22
データ・ディクショナリ	1-28
データの同時実行性と一貫性	1-29
同時実行性	1-29
読み込み一貫性	1-30
ロックのメカニズム	1-31
分散処理と分散データベース	1-32
クライアント / サーバー・アーキテクチャ : 分散処理	1-32
複数層アーキテクチャ : アプリケーション・サーバー	1-33

分散データベース.....	1-33
表のレプリケーション.....	1-35
Oracle と Net8.....	1-36
起動操作と停止操作.....	1-36
データベース・セキュリティ.....	1-37
セキュリティのメカニズム.....	1-38
権限.....	1-40
データベースのバックアップとリカバリ.....	1-44
回復が重要な理由.....	1-44
障害のタイプ.....	1-44
回復に使用される構造.....	1-46
基本的な回復手順.....	1-49
Recovery Manager	1-50
データ・アクセス.....	1-50
SQL— 構造化問合せ言語.....	1-51
トランザクション.....	1-52
PL/SQL.....	1-54
データの整合性.....	1-56

第 II 部 データベースの構造

2 データ・ディクショナリ

データ・ディクショナリの基礎知識.....	2-2
データ・ディクショナリの構造.....	2-2
SYS、データ・ディクショナリの所有者.....	2-3
データ・ディクショナリの使用法.....	2-3
Oracle によるデータ・ディクショナリの使用法.....	2-3
ユーザーと DBA によるデータ・ディクショナリの使用法.....	2-5
動的パフォーマンス表.....	2-7

3 表領域とデータ・ファイル

データベース、表領域およびデータ・ファイル.....	3-2
データベースへの多くの領域の割当て.....	3-4
表領域.....	3-7
SYSTEM 表領域.....	3-7

複数の表領域の使用方法.....	3-8
表領域内の領域管理.....	3-8
オンライン表領域とオフライン表領域.....	3-9
読み込み専用表領域.....	3-11
一時表領域.....	3-12
データベース間での表領域のトランスポート.....	3-14
データ・ファイル	3-16
データ・ファイルの内容.....	3-16
データ・ファイルのサイズ.....	3-17
オフライン・データ・ファイル.....	3-17
一時データ・ファイル.....	3-17

4 データ・ブロック、エクステントおよびセグメント

データ・ブロック、エクステントおよびセグメントの関係.....	4-2
データ・ブロック	4-3
データ・ブロックの形式.....	4-3
PCTFREE、PCTUSED および行連鎖の基礎知識.....	4-5
エクステント	4-10
エクステントが割り当てられる時期.....	4-10
エクステントの数とサイズの決定.....	4-11
エクステントの割当て方法.....	4-12
エクステントが割当て解除される時期.....	4-13
セグメント	4-16
データ・セグメント.....	4-16
索引セグメント.....	4-17
一時セグメント.....	4-17
ロールバック・セグメント.....	4-19

第 III 部 Oracle インスタンス

5 データベースとインスタンスの起動と停止

Oracle インスタンスの概要.....	5-2
インスタンスとデータベース.....	5-2
管理者権限での接続.....	5-3

パラメータ・ファイル.....	5-3
インスタンスとデータベースの起動	5-5
インスタンスを起動する.....	5-5
データベースをマウントする.....	5-6
データベースのオープン.....	5-7
データベースとインスタンスの停止	5-9
データベースのクローズ.....	5-9
データベースのディスマウント.....	5-10
インスタンスの停止.....	5-10

6 分散処理

Oracle クライアント / サーバー・アーキテクチャ	6-2
分散処理	6-2
Net8	6-4
Net8 の機能.....	6-5
ネットワーク・リスナー.....	6-6
複数層アーキテクチャ	6-6
クライアント.....	6-7
アプリケーション・サーバー.....	6-7
データベース・サーバー.....	6-8

7 メモリー・アーキテクチャ

Oracle メモリー構造の概要	7-2
システム・グローバル領域 (SGA)	7-2
データベース・バッファ・キャッシュ.....	7-3
REDO ログ・バッファ.....	7-5
共有プール.....	7-6
大規模プール.....	7-11
SGA のサイズ.....	7-12
SGA のメモリーの使用方法の制御.....	7-13
プログラム・グローバル領域 (PGA)	7-13
PGA の内容.....	7-14
PGA のサイズ.....	7-15
ソート領域	7-16
仮想メモリー	7-17

ソフトウェア・コード領域.....	7-17
8 プロセスのアーキテクチャ	
プロセスの概要	8-2
マルチ・プロセス Oracle システム	8-2
プロセスのタイプ	8-2
ユーザー・プロセス	8-4
接続とセッション	8-4
Oracle プロセス	8-5
サーバー・プロセス.....	8-5
バックグラウンド・プロセス.....	8-6
トレース・ファイルと ALERT ファイル.....	8-15
マルチスレッド・サーバー構成	8-16
ディスパッチャの要求キューと応答キュー.....	8-17
共有サーバー・プロセス.....	8-19
人工デッドロック.....	8-19
マルチスレッド・サーバーの限定的運用.....	8-20
マルチスレッド・サーバーを使用する Oracle の例	8-20
専用サーバー構成	8-22
専用サーバー・プロセスを使用する Oracle の例	8-23
プログラム・インタフェース	8-24
プログラム・インタフェースの構造.....	8-25
プログラム・インタフェース・ドライバ.....	8-25
オペレーティング・システムの通信ソフトウェア	8-26
9 データベース・リソースの管理	
データベース・リソース・マネージャの概要	9-2
資源消費グループとリソース・プラン	9-3
資源消費グループ.....	9-3
リソース・プラン.....	9-4
資源割当方法	9-5
CPU 資源割当方法 : 強調	9-6
最大並列度による資源割当方法 : 絶対.....	9-6
リソース・プラン・ダイレクティブ	9-7
例	9-7

資源消費グループとリソース・プランの使用方法.....	9-7
サブプランの使用方法.....	9-8
複数レベルのリソース・プランの使用方法.....	9-9
並列度制限によるリソース・ダイレクティブの使用方法.....	9-9
まとめ.....	9-10
データベース・リソース・マネージャの使用方法.....	9-10

第 IV 部 オブジェクト・リレーショナル DBMS

10 スキーマ・オブジェクト

スキーマ・オブジェクトの概要.....	10-2
表.....	10-3
表データの格納方法.....	10-4
NULL	10-7
列のデフォルト値.....	10-8
ネストした表.....	10-9
一時表.....	10-10
ビュー.....	10-11
ビューの記憶領域.....	10-12
ビューの使用方法.....	10-13
ビューのメカニズム.....	10-14
依存性とビュー.....	10-15
更新可能な結合ビュー	10-15
オブジェクト・ビュー.....	10-16
インライン・ビュー.....	10-16
マテリアライズド・ビュー.....	10-17
マテリアライズド・ビューのリフレッシュ.....	10-18
マテリアライズド・ビュー・ログ.....	10-18
ディメンション.....	10-18
シーケンス・ジェネレーター.....	10-19
シノニム.....	10-20
索引.....	10-21
一意索引と非一意索引.....	10-22
複合索引.....	10-22
索引とキー.....	10-23

索引と NULL	10-23
ファンクションベース索引.....	10-24
索引の格納方法.....	10-25
キー圧縮.....	10-28
逆キー索引.....	10-29
ビットマップ索引.....	10-30
索引構成表	10-35
索引構成表の利点.....	10-36
行オーバーフロー領域付きの索引構成表.....	10-37
索引構成表の2次索引.....	10-37
索引構成表のその他の機能.....	10-38
索引構成表に適したアプリケーション.....	10-38
アプリケーション・ドメイン索引	10-40
索引タイプ.....	10-41
ドメイン索引.....	10-41
ユーザー定義オペレータ.....	10-42
クラスタ	10-44
パフォーマンスの考慮事項.....	10-46
クラスタ化されたデータ・ブロックの形式.....	10-46
クラスタ・キー.....	10-47
クラスタ索引.....	10-47
ハッシュ・クラスタ	10-48
ハッシュ・クラスタへのデータの格納方法.....	10-49
ハッシュ・キー値.....	10-51
ハッシュ関数.....	10-52
ハッシュ・クラスタに対する領域の割当て.....	10-53
単一表ハッシュ・クラスタ.....	10-55

11 パーティション表とパーティション索引

パーティション化の基礎知識	11-2
パーティション化とは何か.....	11-2
パーティション化の利点.....	11-5
パーティション・ビューを使用した手動のパーティション化.....	11-11
パーティション化の基本的なモデル	11-13
レンジ・パーティション化.....	11-15

ハッシュ・パーティション化.....	11-17
コンポジット・パーティション化.....	11-17
パーティション名とサブパーティション名.....	11-19
パーティション化およびサブパーティション化された列およびキー.....	11-19
レンジ・パーティション化のパーティション・バウンド.....	11-20
同一レベル・パーティション化.....	11-23
表および索引をパーティション化するときのルール.....	11-26
表のパーティション化.....	11-26
索引のパーティション化.....	11-28
LOB 列を持つ表のパーティション化.....	11-38
索引構成表と 2 次索引のパーティション化.....	11-41
DML パーティション・ロックとサブパーティション・ロック.....	11-45
DML パーティション・ロック.....	11-45
DML サブパーティション・ロック.....	11-46
Oracle Parallel Server のパフォーマンスの考慮事項.....	11-47
メンテナンス操作.....	11-47
パーティションのメンテナンス操作.....	11-48
索引の管理.....	11-58
パーティション表およびパーティション索引についての権限.....	11-61
パーティション表およびパーティション索引についての監査.....	11-61
拡張パーティション表名と拡張サブパーティション表名.....	11-62
PARTITION と SUBPARTITION の仕様.....	11-62
表としてのパーティションまたはサブパーティションの表示.....	11-62
拡張パーティション表名と拡張サブパーティション表名の使用.....	11-63

12 ビルトイン・データ型

Oracle データ型の概要.....	12-2
文字データ型.....	12-4
CHAR データ型.....	12-5
VARCHAR2 および VARCHAR データ型.....	12-5
文字データ型と NLS キャラクタ・セットの列の長さ.....	12-6
NCHAR および NVARCHAR2 データ型.....	12-6
LOB 文字データ型.....	12-6
LONG データ型.....	12-6
NUMBER データ型.....	12-7

内部数値形式.....	12-8
DATE データ型	12-9
ユリウス暦の使用方式.....	12-10
日付算術.....	12-10
世紀と西暦 2000 年.....	12-11
LOB データ型	12-11
BLOB データ型.....	12-12
CLOB および NCLOB データ型.....	12-12
BFILE データ型.....	12-13
RAW および LONG RAW データ型	12-13
ROWID および UROWID データ型	12-14
ROWID 疑似列.....	12-14
物理 ROWID.....	12-14
論理 ROWID.....	12-18
Oracle 以外のデータベースの ROWID.....	12-20
ANSI データ型、DB2 データ型および SQL/DS データ型	12-20
データ変換	12-21

13 ユーザー定義データ型

基礎知識	13-2
複合データ・モデル.....	13-2
マルチメディア・データ型.....	13-3
ユーザー定義データ型	13-3
オブジェクト型.....	13-3
コレクション型.....	13-10
アプリケーション・インタフェース	13-12
SQL.....	13-12
PL/SQL.....	13-13
Pro*C/C++.....	13-13
OCI.....	13-14
OTT.....	13-14
JPublisher.....	13-15
JDBC.....	13-15
SQLJ.....	13-15

14 ユーザー定義データ型の使用方法

基礎知識.....	14-2
オブジェクト型と参照.....	14-3
オブジェクト属性のプロパティ.....	14-3
オブジェクト参照.....	14-7
名前変換.....	14-7
コレクション.....	14-9
コレクションの問合せ.....	14-9
コレクションのネスト解除.....	14-10
ネストした表のロケータ.....	14-10
コレクションの DML.....	14-11
ユーザー定義型の権限とそのメソッド.....	14-12
システム権限.....	14-12
スキーマ・オブジェクト権限.....	14-12
新しい型または表での型の使用方法.....	14-12
例.....	14-13
型アクセスとオブジェクト・アクセスについての権限.....	14-13
依存性と不完全な型.....	14-15
不完全な型を完全にする方法.....	14-16
表の型の依存性.....	14-16
ユーザー定義型の記憶領域.....	14-17
リーフ・レベル属性.....	14-17
行オブジェクト.....	14-17
列オブジェクト.....	14-17
REF.....	14-18
ネストした表.....	14-18
VARRAY.....	14-19
ユーティリティ.....	14-19
ユーザー定義型の Import/Export.....	14-19
ユーザー定義型のロード.....	14-19

15 オブジェクト・ビュー

基礎知識.....	15-2
オブジェクト・ビューの利点.....	15-2
オブジェクト・ビューの定義.....	15-3

オブジェクト・ビューの使用方法.....	15-4
オブジェクト・ビューの更新.....	15-5
ビュー内でネストした表の列の更新.....	15-5

索引

Vol. 2

はじめに.....	xxv
-----------	-----

第 V 部 データ・アクセス

16 SQL と PL/SQL

構造化問合せ言語 (SQL).....	16-2
SQL 文.....	16-3
非標準 SQL の識別.....	16-6
再帰 SQL.....	16-6
カーソル.....	16-6
共有 SQL.....	16-7
解析.....	16-7
SQL の処理.....	16-8
SQL 文の実行の概要.....	16-8
DML 文の処理.....	16-10
DDL 文の処理.....	16-14
トランザクションの制御.....	16-14
PL/SQL.....	16-14
PL/SQL が実行される方法.....	16-15
PL/SQL の言語要素.....	16-17
ストアド・プロシージャ.....	16-18
外部プロシージャ.....	16-19

17 トランザクションの管理

トランザクションの基礎知識.....	17-2
文の実行とトランザクションの制御.....	17-3
文レベルのロールバック.....	17-4

Oracle とトランザクションの管理	17-4
トランザクションのコミット.....	17-5
トランザクションのロールバック.....	17-6
セーブポイント.....	17-7
2 フェーズ・コミット・メカニズム.....	17-7
離散トランザクションの管理	17-8
自律型トランザクション	17-8
自律型 PL/SQL ブロック.....	17-9
自律型ブロック内のトランザクション制御文.....	17-10

18 プロシージャとパッケージ

ストアド・プロシージャとパッケージの基礎知識	18-2
ストアド・プロシージャとファンクション.....	18-2
パッケージ.....	18-4
プロシージャとファンクション	18-6
プロシージャのガイドライン.....	18-7
プロシージャの利点.....	18-7
無名 PL/SQL ブロックとストアド・プロシージャ.....	18-9
スタンドアロン・プロシージャ.....	18-9
定義者権限と起動者権限.....	18-9
ストアド・プロシージャの依存性の追跡.....	18-10
外部プロシージャ.....	18-11
パッケージ	18-11
パッケージの利点.....	18-14
パッケージの依存性の追跡.....	18-16
Oracle が提供するパッケージ.....	18-16
Oracle がプロシージャとパッケージを格納する方法	18-16
プロシージャとパッケージのコンパイル.....	18-16
コンパイル済みコードのメモリーへの格納.....	18-16
プロシージャとパッケージのデータベースへの格納.....	18-17
Oracle がプロシージャとパッケージを実行する方法	18-17
ユーザー・アクセスの検査.....	18-18
プロシージャの有効性の検査.....	18-18
プロシージャの実行.....	18-18

19 アドバンスト・キューイング

メッセージ・キューイングの基礎知識.....	19-2
Oracle Advanced Queuing	19-3
キューイング・エンティティ	19-4
アドバンスト・キューイングの機能.....	19-6

20 トリガー

トリガーの基礎知識.....	20-2
トリガーの使用法.....	20-3
トリガーの各部分.....	20-5
トリガー・イベントまたはトリガー文.....	20-6
トリガー条件.....	20-7
トリガー・アクション.....	20-7
トリガーのタイプ.....	20-8
行トリガーと文トリガー.....	20-8
BEFORE トリガーと AFTER トリガー	20-9
トリガー・タイプの組合せ.....	20-9
INSTEAD OF トリガー.....	20-11
システム・イベントとユーザー・イベントのトリガー.....	20-17
トリガーの実行.....	20-20
トリガーの実行モデルと整合性制約のチェック.....	20-20
トリガーのデータ・アクセス.....	20-22
PL/SQL トリガーの記憶領域.....	20-23
トリガーの実行.....	20-23
トリガーの依存性のメンテナンス.....	20-24

21 Oracle の依存性の管理

依存性の問題の基礎知識.....	21-2
スキーマ・オブジェクトの依存性の解決.....	21-4
ビューと PL/SQL プログラム・ユニットのコンパイル.....	21-5
ファンクションベース索引の依存性.....	21-7
依存性の管理と存在しないスキーマ・オブジェクト.....	21-8
共有 SQL の依存性管理.....	21-9
ローカルおよびリモートの依存性の管理.....	21-10
ローカル依存性の管理.....	21-10

リモート依存性の管理.....	21-10
-----------------	-------

第 VI 部 SQL 文の最適化

22 オプティマイザ

最適化とは何か.....	22-2
実行計画.....	22-2
実行の順序.....	22-6
オプティマイザのプラン・スタビリティ.....	22-7
コストベース最適化.....	22-8
コストベース・アプローチの目標.....	22-8
コストベース最適化の統計.....	22-9
コストベースのアプローチを使用する場合.....	22-15
拡張可能な最適化.....	22-16
ユーザー定義統計.....	22-17
ユーザー定義選択性.....	22-17
ユーザー定義コスト.....	22-17
ルールベース最適化.....	22-18

23 オプティマイザの操作

オプティマイザの操作の概要.....	23-2
オプティマイザの操作.....	23-2
SQL 文のタイプ.....	23-3
式と条件の評価.....	23-4
定数.....	23-4
LIKE 演算子.....	23-5
IN 演算子.....	23-5
ANY または SOME 演算子.....	23-5
ALL 演算子.....	23-6
BETWEEN 演算子.....	23-7
NOT 演算子.....	23-7
推移律.....	23-7
DETERMINISTIC 関数.....	23-9
文の変換と最適化.....	23-9
OR から複合問合せへの変換.....	23-10

複合文から結合文への変換.....	23-12
ビューにアクセスする文の最適化.....	23-14
複合問合せの最適化.....	23-26
分散型の文の最適化.....	23-29
最適化アプローチと目標の選択.....	23-30
OPTIMIZER_MODE 初期化パラメータ	23-30
データ・ディクショナリ内の統計データ	23-31
ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータ	23-31
FIRST_ROWS、ALL_ROWS、CHOOSE および RULE の各ヒント	23-32
PL/SQL とオブティマイザの目標.....	23-32
アクセス・パスの選択.....	23-32
アクセス方法.....	23-32
アクセス・パス.....	23-35
アクセス・パスの選択.....	23-48

24 結合の最適化

結合文の最適化.....	24-2
結合操作.....	24-2
結合文の実行計画の選択.....	24-8
外部結合のビュー	24-11
アンチ・ジョインとセミ・ジョインの最適化.....	24-13
「スター」問合せの最適化.....	24-14
スター問合せの例.....	24-14
スター問合せのチューニング.....	24-15
スター型変換.....	24-16

第 VII 部 パラレル SQL とダイレクト・ロード・インサート

25 ダイレクト・ロード・インサート

ダイレクト・ロード・インサートの基礎知識.....	25-2
ダイレクト・ロード・インサートの利点.....	25-2
INSERT ... SELECT 文.....	25-3
ダイレクト・ロード・インサート文の種類.....	25-3
シリアル INSERT とパラレル INSERT	25-3
ロギング・モード.....	25-5

ダイレクト・ロード・インサートについてのその他の考慮事項	25-8
索引のメンテナンス.....	25-8
領域についての考慮事項.....	25-8
ロックについての考慮事項.....	25-11
ダイレクト・ロード・インサートの制限事項	25-11

26 パラレル実行

パラレル実行の概要	26-2
パラレル化できる操作.....	26-2
Oracle が操作をパラレル化する方法.....	26-3
パラレル実行のプロセス・アーキテクチャ	26-5
パラレル・サーバー・プール.....	26-7
パラレル実行サーバーの通信方法.....	26-9
SQL 文のパラレル化.....	26-10
並行度の設定	26-14
Oracle による操作の並行度の決定方法.....	26-15
作業負荷のバランス調整.....	26-17
SQL 文のパラレル化ルール.....	26-18
パラレル問合せ	26-25
索引構成表のパラレル問合せ.....	26-26
オブジェクト型のパラレル問合せ.....	26-27
パラレル DDL	26-27
パラレル化できる DDL 文.....	26-27
パラレルの CREATE TABLE ... AS SELECT.....	26-28
回復可能性とパラレル DDL.....	26-29
パラレル DDL の領域管理.....	26-30
パラレル DML	26-32
手動によるパラレル化と比較したときのパラレル DML の利点.....	26-33
パラレル DML を使用する場合.....	26-34
パラレル DML の使用可能化.....	26-35
パラレル DML のためのトランザクション・モデル.....	26-36
パラレル DML の回復.....	26-37
パラレル DML の領域に関する考慮事項.....	26-38
パラレル DML のためのリソースのロックとエンキュー.....	26-38
パラレル DML の制限事項.....	26-40

関数のパラレル実行.....	26-43
親和性.....	26-44
親和性とパラレル問合せ.....	26-44
親和性とパラレル DML	26-45
その他の並行性.....	26-46

第 VIII 部 データの保護

27 データの同時実行性と一貫性

マルチユーザー環境におけるデータの同時実行性と一貫性.....	27-2
回避可能な現象とトランザクション分離レベル.....	27-2
ロックのメカニズム.....	27-3
データの同時実行性と一貫性の管理方法.....	27-4
マルチバージョン一貫性制御.....	27-4
文レベルの読み取り一貫性.....	27-6
トランザクション・レベルの読み取り一貫性.....	27-6
Oracle Parallel Server における読み取り一貫性	27-7
Oracle の分離レベル	27-7
分離レベルの設定.....	27-8
コミット読み取りと直列可能分離の比較.....	27-10
分離レベルの選択.....	27-13
データをロックする方法.....	27-15
トランザクションとデータ同時実行性.....	27-16
デッドロック	27-17
ロックの種類.....	27-19
DML (データ) ロック	27-20
DDL ロック (ディクショナリ・ロック).....	27-27
ラッチと内部ロック.....	27-29
明示的 (手動) データ・ロック	27-30
Oracle のロック管理サービス	27-37

28 データの整合性

データ整合性の定義.....	28-2
データ整合性のタイプ.....	28-3
Oracle がデータの整合性を施行する方法	28-4

整合性制約の基礎知識	28-5
整合性制約の利点.....	28-5
整合性制約のパフォーマンス・コスト.....	28-7
整合性制約のタイプ	28-7
NOT NULL 整合性制約.....	28-7
UNIQUE キー整合性制約	28-8
PRIMARY KEY 整合性制約	28-11
FOREIGN KEY (参照) 整合性制約.....	28-12
CHECK 整合性制約	28-17
制約チェックのメカニズム	28-18
デフォルト列値と整合性制約チェック	28-19
遅延制約チェック	28-20
制約の属性.....	28-20
SET CONSTRAINTS モード.....	28-20
一意制約と一意索引.....	28-21
制約の状態	28-21
制約の状態変更.....	28-23

29 データベース・アクセスの制御

データベース・セキュリティ	29-2
スキーマ、データベース・ユーザーおよびセキュリティ・ドメイン	29-2
ユーザーの認証	29-3
オペレーティング・システムによる認証.....	29-4
ネットワークによる認証.....	29-4
Oracle データベースによる認証	29-7
複数層の認証と許可.....	29-8
データベース管理者の認証.....	29-11
ユーザー表領域の設定と割当て制限	29-13
デフォルト表領域.....	29-13
一時表領域.....	29-13
表領域のアクセスと割当て制限.....	29-13
ユーザー・グループ PUBLIC	29-14
ユーザー・リソースの制限とプロファイル	29-15
システム・リソースのタイプと制限.....	29-15
プロファイル.....	29-18

ライセンス.....	29-18
同時使用ライセンス.....	29-19
名前付きユーザー・ライセンス.....	29-20

30 権限、ルールおよびセキュリティ・ポリシー

権限.....	30-2
システム権限.....	30-2
スキーマ・オブジェクト権限.....	30-3
表のセキュリティに関するトピック.....	30-4
ビューのセキュリティに関するトピック.....	30-5
プロシージャのセキュリティに関するトピック.....	30-7
型のセキュリティに関するトピック.....	30-10
ルール.....	30-15
ルールの一般的な使用方法.....	30-16
ルールのメカニズム.....	30-17
ルールの付与と取消し.....	30-17
ルールの付与と取消しを実行できるユーザー.....	30-18
ルールの命名.....	30-18
ルールとユーザーのセキュリティ・ドメイン.....	30-18
PL/SQL ブロックとルール.....	30-18
データ定義言語の文とルール.....	30-19
事前定義済みのルール.....	30-20
オペレーティング・システムとルール.....	30-20
分散環境におけるルール.....	30-20
ファイン・グレイン・アクセス・コントロール.....	30-21
動的述語.....	30-21
セキュリティ・ポリシーの例.....	30-21
アプリケーション・コンテキスト.....	30-23

31 監査

監査の基礎知識.....	31-2
監査機能.....	31-2
監査のメカニズム.....	31-4
文監査.....	31-6
権限監査.....	31-7

スキーマ・オブジェクト監査	31-7
ビューとプロシージャのスキーマ・オブジェクト監査オプション.....	31-8
文、権限およびスキーマ・オブジェクトの監査対象の限定	31-9
成功した文の実行と失敗した文の実行の監査.....	31-9
BY SESSION 監査と BY ACCESS 監査.....	31-9
ユーザー別の監査.....	31-11

32 データベースの回復

データベース回復の基礎知識	32-2
エラーと障害.....	32-2
データベースの回復で使用される構造	32-6
データベースのバックアップ.....	32-6
REDO ログ.....	32-7
ロールバック・セグメント.....	32-8
制御ファイル.....	32-8
ロールフォワードとロールバック	32-8
REDO ログとロールフォワード.....	32-9
ロールバック・セグメントとロールバック.....	32-9
回復パフォーマンスの改善	32-10
回復の平行実行.....	32-10
ファースト・スタート・リカバリ.....	32-13
透過的アプリケーション・フェイルオーバーによって障害を隠す方法.....	32-14
Recovery Manager	32-14
リカバリ・カタログ.....	32-14
平行化.....	32-16
レポートの生成.....	32-16
データベースのアーカイブ・モード	32-17
NOARCHIVELOG モード (メディア回復が使用禁止).....	32-17
ARCHIVELOG モード (メディア回復が使用可能).....	32-17
制御ファイル	32-20
制御ファイルの内容.....	32-20
制御ファイルの多重化.....	32-21
データベースのバックアップ	32-22
全体データベース・バックアップ.....	32-22
部分データベース・バックアップ.....	32-23

Export および Import ユーティリティ	32-25
読み込み専用表領域とバックアップ	32-25
耐障害性	32-25
障害回復の計画	32-25
自動化されたスタンバイ・データベース	32-26

第 IX 部 分散データベースとレプリケーション

33 分散データベース

Oracle の分散データベース・アーキテクチャ	33-2
クライアントとサーバー	33-2
ネットワーク	33-4
データベースとデータベース・リンク	33-4
データベース・リンク	33-5
スキーマ・オブジェクトのネーム変換	33-6
複数のバージョンの Oracle Server の間の接続	33-6
分散データベースと分散処理	33-7
分散データベースとデータベース・レプリケーション	33-7
異機種間分散データベース	33-8
異機種間サービス	33-8
異機種間サービス・エージェント	33-8
機能	33-9
分散データベース・アプリケーションの開発	33-10
分散問合せの最適化	33-10
リモート SQL 文と分散 SQL 文	33-10
リモート・プロシージャ・コール (RPC)	33-11
リモート・トランザクションと分散トランザクション	33-12
分散データベース・システムにおける透過性	33-13
Oracle 分散データベース・システムの管理	33-15
サイト自律性	33-15
分散データベースのセキュリティ	33-16
Oracle 分散データベースの管理ツール	33-18
Enterprise Manager	33-18
サードパーティの管理ツール	33-19
SNMP のサポート	33-19

各国語サポート.....	33-19
--------------	-------

34 データベースのレプリケーション

レプリケーションとは.....	34-2
レプリケーションのオブジェクト、グループおよびサイト.....	34-2
マルチマスター・レプリケーション.....	34-6
マルチマスター・レプリケーションの使用法.....	34-6
スナップショット・レプリケーション.....	34-8
読み込み専用スナップショット.....	34-8
更新可能スナップショット.....	34-9
スナップショット・レプリケーションの使用法.....	34-11
マルチマスターおよびスナップショットのハイブリッド構成.....	34-13
レプリケート環境の管理.....	34-14
レプリケーション・カタログ.....	34-14
レプリケーション管理 API と管理要求.....	34-14
Oracle Replication Manager	34-15
レプリケーションの競合.....	34-15
特殊レプリケーション・オプション.....	34-15

第 X 部 付録

A オペレーティング・システム固有の情報

索引

はじめに

この 2 巻構成のマニュアルでは、オブジェクト・リレーショナル・データベース管理システムである Oracle Server の全機能について説明します。このマニュアルでは、Oracle Server がどのように機能するかを説明します。その情報は、Oracle Server の他のマニュアルに記載されている実用上の情報の多くについて、その概念上の基礎になるものです。このマニュアルの情報は、すべてのオペレーティング・システム上で稼働する Oracle Server を対象にしています。

Oracle8i および Oracle8i Enterprise Edition

『Oracle8i 概要』では、Oracle8i および Oracle8i Enterprise Edition 製品の機能を説明しています。Oracle8i および Oracle8i Enterprise Edition は、同じ基本機能を持っています。ただし、Enterprise Edition のみで使用可能な拡張機能もいくつかあり、これらの一部はオプションです。たとえば、表をパーティション化するには、Enterprise Edition と Partitioning Option が必要です。

対象読者

このマニュアルは、データベース管理者、システム管理者およびデータベース・アプリケーションの開発者を対象にしています。

前提条件

読者には、リレーショナル・データベースの概念と、Oracle を実行しているオペレーティング・システム的环境についての知識が必要です。最初に、[第 1 章「Oracle Server の基礎知識」](#)を必ず読んでください。第 1 章では、このマニュアル全体で使用されている概念と用語について包括的に説明されています。

インストールおよび移行について

このマニュアルは、インストールや移行の手引き書ではありません。したがって、主としてインストールに関心のある方は、使用するオペレーティング・システム固有の Oracle のマニュアルを参照してください。また、主としてデータベースおよびアプリケーションの移行に関心のある方は、『Oracle8i 移行ガイド』を参照してください。

データベース管理について

このマニュアルは、Oracle Server のアーキテクチャ、プロセス、構造およびその他の概念について説明しています。Oracle Server を管理する方法については説明していません。詳細は、『Oracle8i 管理者ガイド』を参照してください。

アプリケーション設計について

このマニュアルには、[管理者](#)だけでなく、Oracle に精通したユーザーや、高度なデータベース・アプリケーションの設計者にも役立つ情報が記載されています。ただし、データベース・アプリケーションの開発者は、『Oracle8i アプリケーション開発者ガイド 基礎編』と、Oracle データベース・アプリケーションの開発に使用するツールまたは言語製品のマニュアルも参照する必要があります。

このマニュアルの構成

このマニュアルは 2 巻構成で、次のような部に分かれています。

- Vol.1
 - [第 I 部 : Oracle の概要](#)
 - [第 II 部 : データベースの構造](#)
 - [第 III 部 : Oracle インスタンス](#)
 - [第 IV 部 : オブジェクト・リレーショナル DBMS](#)
- Vol.2
 - [第 V 部 : データ・アクセス](#)

- 第 VI 部 : SQL 文の最適化
- 第 VII 部 : パラレル SQL とダイレクト・ロード・インサート
- 第 VIII 部 : データの保護
- 第 IX 部 : 分散データベースとレプリケーション
- 第 X 部 : 付録

Vol.1

第 I 部 : Oracle の概要

第 1 章「Oracle Server の基礎知識」

Oracle データ・サーバーを理解する上で必要になる概念と用語について概説します。このマニュアルで説明されている詳細な情報を読む前に、この章を読んでおいてください。

第 II 部 : データベースの構造

第 2 章「データ・ディクショナリ」

データ・ディクショナリについて説明します。データ・ディクショナリは、Oracle データベースについての読み込み専用の情報を格納した表とビューで構成されています。

第 3 章「表領域とデータ・ファイル」

Oracle データベース内で、物理的な記憶領域が、表領域と呼ばれる論理的な区画にどのように分割されているかについて説明します。さらに、表領域に対応付けられる物理的なオペレーティング・システム・ファイル（データ・ファイル）についても説明します。

第 4 章「データ・ブロック、エクステントおよびセグメント」

Oracle データベース内の各種オブジェクトにデータがどのように格納され、記憶領域がどのように割り当てられるかについて説明します。この章で取り上げる領域管理に関する背景知識は、次の章と第 10 章「スキーマ・オブジェクト」の内容を補足するものです。

第 III 部 : Oracle インスタンス

第 5 章「データベースとインスタンスの起動と停止」

Oracle インスタンスについて説明し、データベース管理者が Oracle データベース・システムへのアクセスを制御する方法について説明します。さらに、データベースの実行状態を制御するパラメータについても説明します。

第 6 章「分散処理」

Oracle データ・サーバーを実行できる分散処理環境について説明します。

第7章「メモリー・アーキテクチャ」

Oracle データベース・システムで使用するメモリー構造について説明します。

第8章「プロセスのアーキテクチャ」

Oracle インスタンスのプロセス・アーキテクチャと、Oracle で利用できる各種のプロセス構成について説明します。

第9章「データベース・リソースの管理」

データ・リソース・マネージャを使用してリソース使用を制御する方法について説明します。

第IV部：オブジェクト・リレーショナルDBMS

第10章「スキーマ・オブジェクト」

表、ビュー、順序およびシノニムなど、特定のユーザーのドメイン（スキーマ）内に作成できるデータベース・オブジェクトについて説明します。また、索引、マテリアライズド・ビュー、ディメンションおよびクラスタなど、データ検索を効率化するオプションの構造についても説明します。

第11章「パーティション表とパーティション索引」

パーティション化を利用して、大規模な表や索引を管理しやすい区画に分割する方法を説明します。

第12章「ビルトイン・データ型」

Oracle データベースの表に格納できるリレーショナル・データのデータ型について説明します。たとえば、固定長文字列、可変長文字列、数値、日付およびバイナリ・ラージ・オブジェクト（BLOB）などがあります。

第13章「ユーザー定義データ型」

Oracle が提供するオブジェクト拡張機能の概要について説明します。

第14章「ユーザー定義データ型の使用方法」

Oracle データ・サーバーで利用可能なユーザー定義のオブジェクト型について説明します。

第15章「オブジェクト・ビュー」

Oracle データ・サーバーによってビューに提供される拡張機能について説明します。

Vol.2

第Ⅴ部：データ・アクセス

第16章「SQLとPL/SQL」

Oracle と対話するために使用する SQL（構造化問合せ言語）と、SQL への Oracle のプロシージャ型言語機能拡張である PL/SQL について説明します。

第17章「トランザクションの管理」

トランザクションの概念を定義し、トランザクションを制御するために使用する SQL 文について説明します。トランザクションとは、1 単位としてまとめて実行される論理作業単位です。

第18章「プロシージャとパッケージ」

データベース内に格納される PL/SQL プログラム・ユニットである、プロシージャ、ファンクションおよびパッケージと呼ばれるプロシージャ型言語の構成体について説明します。

第19章「アドバンスト・キューイング」

Oracle のアドバンスト・キューイング機能について説明します。この機能を使用すると、メッセージをキューに格納して、Oracle Server に遅延取出しと遅延処理を実行させることができます。

第20章「トリガー」

トリガーと呼ばれるプロシージャ型言語の構成体について説明します。トリガーは、データベース表に行が挿入、更新または削除されたときに暗黙に実行されるプロシージャです。

第21章「Oracle の依存性の管理」

プロシージャ、パッケージ、トリガー、ビューなどのオブジェクトの依存性を Oracle がどのように管理するかについて説明します。

第Ⅵ部：SQL 文の最適化

第22章「オブティマイザ」

オブティマイザについて説明します。オブティマイザは、各 SQL 文を最も効率的に実行する方法を選択する Oracle の機能です。

第23章「オブティマイザの操作」

SQL 文の最適な実行方法がオブティマイザでどのように選択されるかについて説明します。

第24章「結合の最適化」

結合、アンチ・ジョインおよびセミ・ジョインを含む SQL 文がオブティマイザでどのように実行されるか、また、オブティマイザでビットマップ索引を使用してスター問合せを実行し、事実表を複数のディメンション表に結合する方法についても説明します。

第 VII 部：パラレル SQL とダイレクト・ロード・インサート

第 25 章「ダイレクト・ロード・インサート」

シリアルまたはパラレルに実行できるダイレクト・ロード・インサート・パスおよび NOLOGGING オプションについて説明します。

第 26 章「パラレル実行」

SQL 文（問合せ、DML および DDL 文）のパラレル実行と、SQL 文のパラレル化の規則について説明します。

第 VIII 部：データの保護

第 27 章「データの同時実行性と一貫性」

マルチ・ユーザー環境において、Oracle が共有情報への同時アクセスを提供し、その情報の正確さを維持する仕組みについて説明します。そして、複数のユーザーが同時に操作を実行しても相互に干渉し合わないよう、Oracle が自動的に使用するメカニズムについて説明します。

第 28 章「データの整合性」

データの整合性と、整合性を施行するために使用できる整合性制約の宣言について説明します。

第 29 章「データベース・アクセスの制御」

データとデータベース・リソースへのユーザー・アクセスを制御する方法について説明します。

第 30 章「権限、ロールおよびセキュリティ・ポリシー」

システム・レベルとスキーマ・オブジェクト・レベルでのセキュリティについて説明します。

第 31 章「監査」

Oracle の監査機能がデータベース・アクティビティを追跡する仕組みについて説明します。

第 32 章「データベースの回復」

データベースの回復に使用されるファイルと構造と、起こり得る障害から Oracle データベースを保護する方法について説明します。

第 IX 部：分散データベースとレプリケーション

第 33 章「分散データベース」

分散データベース・アーキテクチャ、リモート・データ・アクセスおよび表のレプリケーションについて説明します。

第 34 章「データベースのレプリケーション」

分散データベース・システムにおける Oracle データベースのレプリケーションについて説明します。

第 X 部：付録

付録 A「オペレーティング・システム固有の情報」

このマニュアル内に記載されている、オペレーティング・システムに固有の情報のリストです。

このマニュアルの使用法

すべての読者は、必ず第 1 章「Oracle Server の基礎知識」を読んでください。この章では、Oracle に関連する概念と用語について解説しており、それ以降の章に記載されている詳細な説明を読むための基礎になります。

このマニュアルの各部は、前述の対象読者の中でも、さらに特定の読者を対象にしています。たとえば、主としてセキュリティ管理関連の情報を必要としている管理者は、第 1 章を読んだ後に第 VIII 部「データの保護」、特に第 29 章「データベース・アクセスの制御」、第 30 章「権限、ロールおよびセキュリティ・ポリシー」および第 31 章「監査」を重点的に読む必要があります。

このマニュアルで使用する表記規則

このマニュアルでは、情報の種類に応じていくつかの表記を使用します。

マニュアルの本文

このマニュアルの本文では、次のような表記規則が使用されています。

英大文字

英大文字のテキストは、コマンド・キーワード、データベース・オブジェクト名、パラメータ、ファイル名などを明示するために使用されます。

たとえば、「デフォルト値を挿入した後で、Oracle は DEPTNO 列に定義されている FOREIGN KEY 整合性制約をチェックします。」または「プライベート・ロールバック・セグメントを作成する場合は、そのセグメントの名前を ROLLBACK_SEGMENTS 初期化パラメータに指定する必要があります。」のように表記します。

コード例

SQL、Oracle Enterprise Manager の行モード (Server Manager) および SQL*Plus のコマンドや文は、モノスペース・フォントで記載します。

たとえば、次のとおりです。

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');  
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

例文には、コンマや引用符などの句読点が含まれている場合があります。例文に示されている句読点はすべて必須です。すべての例文はセミコロン (;) で終わります。アプリケーションによっては、1 つの文を終了させるためにセミコロンまたはその他の終了文字が必要な場合と、必要ない場合があります。

コード例の中の英大文字

例文の中の英大文字の語は、Oracle SQL のキーワードを示します。ただし、実際に SQL 文を発行するときには、キーワードの大 / 小文字は区別されません。

コード例の中の英小文字

例文の中の英小文字の語は、単なる例として使用されている語を示します。たとえば、英小文字の語は、表、列またはファイルの名前を示します。

第 I 部

Oracle の概要

第 I 部では、Oracle Server の概念と用語について概説します。次の 1 つの章が含まれています。

- [第 1 章の「Oracle Server の基礎知識」](#)

このマニュアルの残りの部分では、第 1 章に要約されている概念をさらに詳しく説明します。

Oracle Server の基礎知識

I am Sir Oracle,

And when I ope my lips, let no dog bark!

Shakespeare: *The Merchant of Venice*

この章では、Oracle Server の概要について説明します。内容は次のとおりです。

- データベースと情報管理
- データベースの構造と領域管理
- メモリー構造とプロセス
- データベース管理のオブジェクト・リレーショナル・モデル
- データの同時実行性と一貫性
- 分散処理と分散データベース
- 起動操作と停止操作
- データベース・セキュリティ
- データベースのバックアップとリカバリ
- データ・アクセス

注意： この章には、Oracle8i と Oracle8i Enterprise Edition の両方に関連する情報が記載されています。この章に記載されている機能とオプションの中には、Oracle8i Enterprise Edition を購入した場合にのみ使用可能なものも含まれています。

データベースと情報管理

データベース・サーバーは、情報管理の問題を解決する鍵となります。一般にサーバーでは、多数のユーザーが同時に同じデータをアクセスできるように、マルチユーザー環境において大量のデータを確実に管理することが必要です。このような管理を行いながら、一方で高いパフォーマンスを実現し、また、権限のないアクセスに対して保護機能を備えながら、障害の回復についても効率のよい解決方法を提供する必要があります。

Oracle Server は、次に示す機能について、効率のよい効果的な解決方法を提供します。

クライアント / サーバー環境（分散処理）	コンピュータ・システムまたはネットワークを最大限に活用するために、Oracle では処理をデータベース・サーバーとクライアント・アプリケーション・プログラムに分割して実行します。データベース管理システムが稼働しているコンピュータはデータベース・サーバーのすべての役割を果たし、データベース・アプリケーションが稼働しているワークステーションは主にデータの解釈と表示を行います。
大規模データベースと領域管理	Oracle は、数 TB のデータを格納できる最大規模のデータベースをサポートしています。また、高価なハードウェア装置を効率よく使用するために、領域の使用方法を完全に管理します。
多数の同時実行データベース・ユーザー	Oracle は、同じデータを操作する複数のデータベース・アプリケーションを多数のユーザーが同時に実行することをサポートします。Oracle はデータの競合を最小限に抑え、データの同時実行性を保証します。
接続性	Oracle ソフトウェアでは、いろいろなタイプのコンピュータとオペレーティング・システムを接続して、ネットワーク全体で情報を共有できます。
高いトランザクション処理パフォーマンス	Oracle は、システム全体の高いパフォーマンスを保ちつつ、前述の各機能を実行します。データベース・ユーザーにとって、処理パフォーマンスの低さが問題になることはありません。
高い可用性	サイトによっては、Oracle が、データベースの処理能力を低下させる時間帯を設けることなく、1 日 24 時間稼働することもあります。データベースのバックアップやコンピュータ・システムの部分的な障害など、日常的なシステムの運用によってデータベースの使用を中断させることはありません。
制御可能な可用性	Oracle は、データベース・レベルと下位データベース・レベルで、データベースの可用性を選択的に制御できます。たとえば、他のアプリケーションに影響を及ぼすことなく、管理者が、特定のアプリケーションを使用禁止にして、そのアプリケーションのデータを再ロードすることができます。

オープンかつ業界標準	<p>Oracle は、データ・アクセス言語、オペレーティング・システム、ユーザー・インタフェースおよびネットワーク通信プロトコルについて、業界で受け入れられている規格に準拠しています。Oracle は、顧客の投資を守るオープン・システムです。</p> <p>Oracle は、システム管理の規格としてシンプル・ネットワーク管理プロトコル（SNMP）規格もサポートしています。このプロトコルを使用すると、管理者は 1 つの管理インタフェースで異機種システムを管理できます。</p>
扱いやすいセキュリティ機能	<p>許可されていないデータベース・アクセスと許可されていないデータベース使用からデータベースを保護するために、Oracle では、データ・アクセスの制限と監視を可能にするフェイルセーフ・セキュリティ機能が提供されます。データ・アクセスの設計がかなり複雑な場合でも、これらの機能によって管理が容易になります。</p>
データベースに施行される整合性	<p>Oracle では、データの整合性、つまり許容可能なデータについての規格を示す「ビジネス・ルール」が施行されます。これにより、多数のデータベース・アプリケーションでチェック機能をコーディングして管理するコストが削減されます。</p>
移植性	<p>Oracle ソフトウェアは、さまざまなオペレーティング・システム上で機能します。Oracle 向けに開発されたアプリケーションは、わずかな修正を加えるだけで、あるいはまったく修正することなく、任意のオペレーティング・システムに移植できます。</p>
互換性	<p>Oracle ソフトウェアには、ほとんどの業界標準オペレーティング・システムを含む業界規格との互換性があります。したがって、Oracle 向けに開発されたアプリケーションは、わずかな修正を加えるだけで、あるいはまったく修正することなく、事実上どのシステムでも利用できます。</p>
分散システム	<p>ネットワーク分散環境では、物理的には複数のコンピュータ上にあるデータが Oracle によって結合されて 1 つの論理データベースとなり、すべてのネットワーク・ユーザーがその論理データベースにアクセスできます。分散システムは、非分散型のシステムと同水準のユーザーの透過性とデータの一貫性を維持しているだけでなく、ローカル・データベース管理システムの持つ利点も備えています。</p> <p>Oracle には、Oracle 以外のデータベースにあるデータへの透過的なアクセスを実現する異機種データベース・オブションも用意されています。</p>

レプリケート環境

Oracle ソフトウェアでは、表のグループおよびそれらの表をサポートするオブジェクトを、複数のサイトにレプリケートできます。Oracle では、これらのサイトへのデータ変更のレプリケーションについて、データ・レベルとスキーマ・レベルの両方をサポートします。Oracle の柔軟なレプリケーション・テクノロジーにより、基本的なプライマリ・サイトのレプリケーションだけでなく、高度な動的モデルと共用所有権モデルもサポートされます。

この後の項では、Oracle アーキテクチャの概要について幅広く説明します。アーキテクチャ全体の構成を各部分ごとに説明していきます。

Oracle Server

Oracle Server は、情報管理におけるオープンかつ広範囲で統合的なアプローチを実現する、オブジェクト・リレーショナル・データベース管理システムです。Oracle Server は、Oracle データベースと Oracle Server インスタンスで構成されます。この後の各項では、データベースとインスタンスの関係について説明します。

構造化問合せ言語（SQL）

SQL は、データベースを定義して操作するためのプログラミング言語です。SQL データベースはリレーショナル・データベースです。これは、データが一連の単純なリレーションに基づいて格納されるということです。1つのデータベースは、1つ以上の表を持ちます。さらに、それぞれの表には列と行があります。たとえば、従業員データベースの表には、従業員番号という列があり、各行のその列には従業員番号が格納されます。

表の中のデータは、SQL コマンドを使用して定義したり、操作できます。データをセットアップするには、データ定義言語（DDL）コマンドを使用します。DDL コマンドには、データベースや表を作成および変更するコマンドが含まれます。

また、表の中のデータの更新、削除または取出しには、データ操作コマンド（DML）を使用します。DML コマンドには、データの変更やフェッチを行うコマンドが含まれます。最も使用頻度の高い SQL コマンドは、SELECT コマンドです。このコマンドは、データベースからデータを取り出すときに使用します。

SQL コマンドの他に、Oracle Server には PL/SQL と呼ばれるプロシージャ型言語も用意されています。PL/SQL によって、プログラマは SQL 文のプログラムを書くことができます。PL/SQL を使用すると、SQL プログラムの流れを制御したり、変数を使用するだけでなく、エラー処理プロシージャを書くこともできます。

データベース構造

Oracle データベースには、論理構造と物理構造の両方が存在します。物理的なサーバー構造と論理的なサーバー構造が分離されているため、論理的な記憶構造へのアクセスに影響を与えずに、データの物理的な記憶領域を管理できます。

物理データベース構造 Oracle データベースの物理構造は、データベースを構成するオペレーティング・システム・ファイルによって決定されます。各 Oracle データベースは、3 つのタイプのファイル、つまり、1 つ以上のデータ・ファイル、複数の REDO ログ・ファイル、および 1 つ以上の制御ファイルから構成されます。Oracle データベースのファイルが、データベース情報のための実際の物理的な記憶領域を提供します。

論理データベース構造 Oracle データベースの論理構造は、次の 2 つによって決定されます。

- 1 つ以上の表領域

「表領域」とは、この章で後述する論理記憶領域のことです。

- データベースのスキーマ・オブジェクト

「スキーマ」とは、オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタ、データベース・リンクなどの構造体が含まれます。

表領域、セグメントおよびエクステンツなどの論理構造によって、データベースの物理領域がどのように使用されるかが決まります。スキーマ・オブジェクトと、それらのオブジェクト間の関連が、データベースのリレーショナル設計を形成します。

データベース構造の詳細は、1-8 ページの「[データベースの構造と領域管理](#)」を参照してください。

データ・ユーティリティ

Oracle データベースのサブセットをデータベース間で移動できるように、Export、Import および SQL*Loader という 3 つのユーティリティが用意されています。

Export Export ユーティリティを使用すると、ハードウェア構成やソフトウェア構成の異なるプラットフォーム上にあっても、Oracle データベース間でデータ・オブジェクトを容易に転送できます。Export では、Oracle データベースからオブジェクト定義と表データが抽出され、通常はディスクまたはテープにある Oracle の 2 進形式の Export ダンプ・ファイルに格納されます。

これらのファイルは、ftp を介して、または物理的に（テープの場合）他のサイトに送って使用できます。また、Import ユーティリティを使用すると、ネットワークを介して接続されていないマシン上のデータベース間でデータを転送したり、通常のバックアップ手順を補足する意味でバックアップとして使用できます。

Oracle データベースに対して Export を実行すると、最初にオブジェクト（表など）が、次に関連オブジェクト（索引、コメントおよび権限付与など）が抽出され、Export ファイルに書き込まれます。

Import Import ユーティリティは、Export ユーティリティによって Oracle データベースから抽出された（および Export ダンプ・ファイルに格納された）データ・オブジェクトを、別の Oracle データベースに挿入します。Export ダンプ・ファイルを読み取れるのは、Import のみです。

Import では、Export ユーティリティによって Oracle データベースから抽出され、通常はディスクまたはテープにある Oracle の 2 進形式の Export ダンプ・ファイルに格納された、オブジェクト定義と表データが読み込まれます。

また、Export および Import ユーティリティを使用すると、オフライン・インスタンスेशनなど、Oracle アドバンスト・レプリケーション機能の特定部分の処理が容易になります。

追加情報： Oracle アドバンスト・レプリケーションの詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

SQL*Loader Export ダンプ・ファイルを読み取れるのは、Oracle Import ユーティリティのみです。固定形式や区切りファイルからロード・データを読む必要がある場合は、SQL*Loader ユーティリティを使用できます。SQL*Loader では、データが外部ファイルから Oracle データベース内の表にロードされます。SQL*Loader はさまざまな形式の入力データを受け入れ、フィルタ処理を実行（レコードをデータ値に基づいて選択的にロード）し、同じロード・セッション中に、そのデータを Oracle データベースの表にロードできます。

追加情報： Export、Import および SQL*Loader の詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

Oracle インスタンス

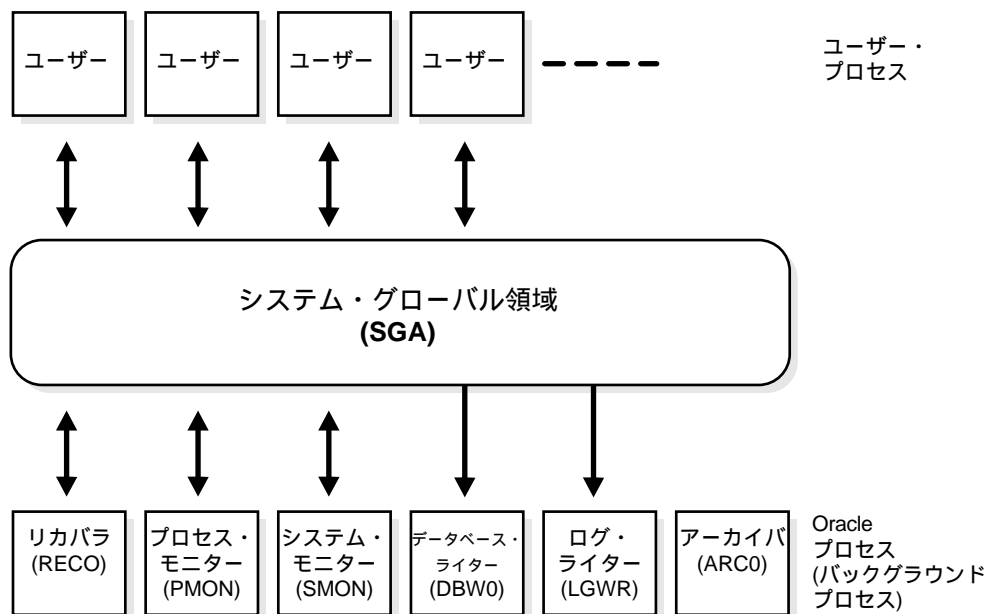
データベースを起動するたびに、システム・グローバル領域（SGA）が割り当てられ、Oracle バックグラウンド・プロセスが起動されます。システム・グローバル領域とは、各データベース・ユーザーが共有するデータベース情報のために使用されるメモリー領域のことです。バックグラウンド・プロセスとメモリー・バッファの組合せのことを、「Oracle インスタンス」と呼びます。

Oracle インスタンスには、ユーザー・プロセスおよび Oracle プロセスという、2 つのタイプのプロセスがあります。

- 「ユーザー・プロセス」は、アプリケーション・プログラム（Oracle Forms アプリケーションなど）や Oracle Tool（Oracle Enterprise Manager など）のコードを実行する。
- 「Oracle プロセス」には、ユーザー・プロセスのための処理を実行するサーバー・プロセスと、Oracle Server のためのメンテナンス処理を実行するバックグラウンド・プロセスがある。

図 1-1 に、マルチ・プロセスの Oracle インスタンスを示します。

図 1-1 Oracle インスタンス



通信ソフトウェアと Net8

ユーザー・プロセスとサーバー・プロセスがネットワーク内の異なるコンピュータ上にある場合や、ユーザー・プロセスがディスパッチャ・プロセスを介して共有サーバー・プロセスに接続している場合、ユーザー・プロセスとサーバー・プロセスは Net8 を使用して通信します。「ディスパッチャ」はオプションのバックグラウンド・プロセスで、これはマルチスレッド・サーバー構成にのみ存在します。「Net8」は、コンピュータ間でデータを正常に転送するための、標準通信プロトコルへの Oracle のインターフェースです。

詳細は、1-36 ページの「[Oracle と Net8](#)」を参照してください。

Oracle Parallel Server : 複数インスタンス・システム

注意： Oracle Parallel Server オプションは、Oracle8i Enterprise Edition を購入した場合にのみ使用可能です。

ハードウェア・アーキテクチャによっては（共有ディスク・システムなど）、複数のコンピュータがデータ、ソフトウェアまたは周辺装置へのアクセスを共有できます。Parallel Server オプション付きの Oracle は、1 つの物理データベースを「共有」する複数のデータベース・インスタンスを実行することによって、このようなアーキテクチャを活用できます。適切なアプリケーションでは、Oracle Parallel Server によって、複数のマシン上のユーザーが、優れたパフォーマンスで 1 つのデータベースにアクセスできます。

追加情報： Oracle Parallel Server の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

Oracle データベース

「Oracle データベース」とは、1 単位として扱われるデータの集合です。データベースの一般的な目的は、関連する情報を格納したり、取り出したりすることです。

データベースには、「論理構造」と「物理構造」があります。Oracle データベースの論理構造および物理構造の概要は、1-8 ページの「[論理データベース構造](#)」および 1-11 ページの「[物理データベース構造](#)」を参照してください。

データベースのオープンとクローズ

Oracle データベースは、「オープン」（アクセス可能）または「クローズ」（アクセス不能）のどちらかの状態です。通常、データベースはオープンで、使用可能な状態になっています。ただし、データベースのデータをユーザーから切り離す必要があるような特定の管理機能を実行するために、データベースをクローズする場合があります。

データベースの構造と領域管理

この項では、Oracle データベースを構成する構造について説明します。制御可能なデータの可用性、論理データ構造と物理データ構造の分離およびディスク領域管理のきめ細かい制御についての Oracle の解決方法を理解できます。

「Oracle データベース」とは、1 単位として扱われるデータの集合です。データベースの一般的な目的は、関連する情報の格納や取出しです。データベースには、「論理構造」と「物理構造」があります。

論理データベース構造

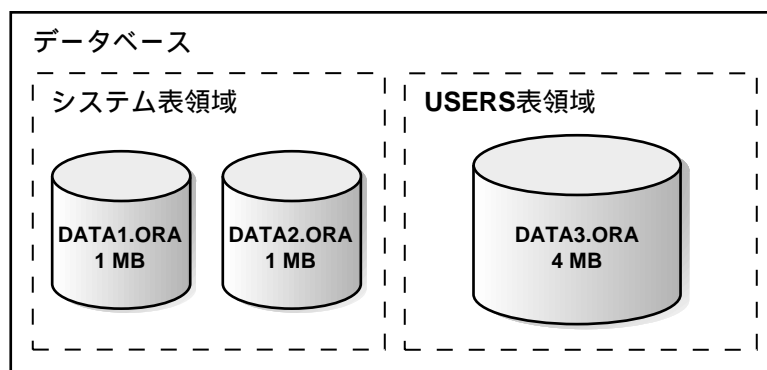
Oracle データベースの論理構造には、表領域、スキーマ・オブジェクト、データ・ブロック、エクステンツおよびセグメントが含まれます。

表領域

データベースは、関連する論理構造がグループ化された「表領域」と呼ばれる論理記憶単位に分けられています。たとえば、通常、表領域は1つのアプリケーションのすべてのオブジェクトをグループにまとめ、管理的な操作を容易にします。

データベース、表領域およびデータ・ファイル データベース、表領域およびデータ・ファイルの間の関連を図 1-2 に示します（データ・ファイルについては次の項で説明します）。

図 1-2 データベース、表領域およびデータ・ファイル



この図から次のようなことがわかります。

- 各データベースは、論理的に1つ以上の表領域に分けられる。
- 表領域内のすべての論理構造のデータを物理的に格納するために、1つ以上のデータ・ファイルが各表領域に対して明示的に作成される。
- 表領域のデータ・ファイルのサイズを合わせると、表領域全体の記憶容量になる（SYSTEM 表領域は 2MB、USERS 表領域は 4MB の記憶容量を持つ）。
- データベースの表領域の記憶容量を合わせると、データベース全体の記憶容量になる（6MB）。

オンライン表領域とオフライン表領域 表領域は、「オンライン」（アクセス可能）または「オフライン」（アクセス不能）にできます。ユーザーが表領域内の情報にアクセスできるように、通常、表領域はオンラインになっています。ただし、場合によっては、1つの表領域をオフラインにしてデータベースの一部を使用できないようにすると同時に、データベースの残りの部分には通常どおりアクセスできるようにすることもできます。これによって、多くの管理業務を容易に実行できます。

スキーマとスキーマ・オブジェクト

「スキーマ」とは、データベース・オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造体です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタおよびデータベース・リンクなどの構造体が含まれます。(表領域とスキーマには何の関連もありません。同じスキーマ内のオブジェクトが別々の表領域に存在したり、異なるスキーマにあるオブジェクトが1つの表領域に保持されることがあります。) スキーマ・オブジェクトの詳細は、1-22ページの「[スキーマとスキーマ・オブジェクト](#)」を参照してください。

データ・ブロック、エクステントおよびセグメント

Oracle では、データ・ブロック、エクステントおよびセグメントなどの論理記憶構造体によってディスク領域の使用をきめ細かく制御できます。これらの詳細は、[第4章「データ・ブロック、エクステントおよびセグメント」](#)を参照してください。

Oracle **データ・ブロック** 最もきめ細かいレベルとして、Oracle データベースのデータは「データ・ブロック」に格納されます。1つのデータ・ブロックは、ディスク上の特定のバイト数の物理データベース領域に対応します。データ・ブロックのサイズは、データベースの作成時に Oracle データベースに対して指定します。データベースは、Oracle データ・ブロック内の空きデータベース領域を使用して領域を割り当てます。

エクステント 論理的なデータベース領域の次のレベルは、エクステントと呼ばれます。「エクステント」は、特定数の連続したデータ・ブロックであり、1回の割当てで取得され、特定のタイプの情報を格納するために使用されます。

セグメント エクステントの上位に位置する論理的なデータベース記憶領域のレベルは、セグメントと呼ばれます。「セグメント」は、特定の論理構造に割り当てられるエクステントの集合です。たとえば、次のような異なるタイプのセグメントがあります。

データ・セグメント クラスタ化されていない表は、それぞれ1つのデータ・セグメントを持っています。表のデータはすべて、そのデータ・セグメントのエクステントに格納されます。パーティション表の場合は、各パーティションに1つずつデータ・セグメントがあります。

各クラスタは、1つのデータ・セグメントを持っています。クラスタ内のあらゆる表のデータが、そのクラスタのデータ・セグメントに格納されます。

索引セグメント 各索引は、そのデータをすべて格納する索引セグメントを1つ持っています。パーティション索引の場合は、各パーティションに1つずつ索引セグメントがあります。

ロールバック・セグメント データベース管理者は、「UNDO (取消し)」情報を一時的に格納するために、データベースごとに1つ以上のロールバック・セグメントを作成します。

ロールバック・セグメント内の情報の用途は、次のとおりです。

- 読取り一貫性を備えたデータベース情報を生成するため（1-30 ページの「[読込み一貫性](#)」を参照）
- データベースの回復時（1-44 ページの「[データベースのバックアップとリカバリ](#)」を参照）
- ユーザーの必要に応じて、コミットされていないトランザクションをロールバックするため

一時セグメント

一時セグメントは、SQL 文の実行を完了するために一時的な作業領域が必要なときに、Oracle によって作成されます。その文の実行が終了すると、一時セグメントのエクステントは後で使用できるようにシステムに戻されます。

Oracle は、1 つのセグメントの既存のエクステントが満杯になった時点で、領域を動的に割り当てます。したがって、セグメントの既存のエクステントが満杯になっている場合は、必要に応じてそのセグメントに別のエクステントを割り当てます。エクステントは必要に応じて割り当てられるため、1 つのセグメントの各エクステントはディスク上で連続している場合と連続していない場合があります。

物理データベース構造

この後の項では、データ・ファイル、REDO ログ・ファイルおよび制御ファイルなど、Oracle データベースの物理構造について説明します。

データ・ファイル

すべての Oracle データベースは、1 つ以上の物理「データ・ファイル」を持っています。データベースのデータ・ファイルには、すべてのデータベース・データが格納されます。表や索引などの論理データベース構造のデータは、物理的にはデータベースに割り当てられたデータ・ファイルに格納されます。

データ・ファイルの特性は次のとおりです。

- 1 つのデータ・ファイルは、1 つのデータベースのみに対応付けられる。
- データ・ファイルには、データベースの領域をすべて使用してしまった場合にファイルが自動的に拡張されるように、特定の特性を設定できる。
- 前述のように、1 つ以上のデータ・ファイルによって、表領域と呼ばれるデータベース記憶の論理単位が形成される。

データ・ファイルの使用方法 通常のデータベース操作中に、データ・ファイル内のデータは必要に応じて読み込まれ、Oracle のメモリー・キャッシュに格納されます。たとえば、あるユーザーがデータベースの表に入っている一部のデータにアクセスする場合を考えます。

要求された情報がデータベースのメモリー・キャッシュに存在しない場合には、その情報は適切なデータ・ファイルから読み込まれて、メモリーに格納されます。

修正されたデータや新規のデータは、必ずしも即時にデータ・ファイルに書き込まれるわけではありません。ディスクへのアクセス回数を減らし、パフォーマンスを向上させるために、データはメモリー内に蓄積されてから、適切なデータ・ファイルに一度に書き込まれます。これらの操作は、Oracle の DBW_n バックグラウンド・プロセスによって決定されます。(Oracle のメモリーおよびプロセスの構造、データベースのデータをデータ・ファイルに書き込むときのアルゴリズムの詳細は、1-13 ページの「[メモリー構造とプロセス](#)」を参照してください。)

REDO ログ・ファイル

すべての Oracle データベースは、2 つ以上の「REDO ログ・ファイル」の集合を持っています。データベースの REDO ログ・ファイルの集合は、データベースの「REDO ログ」と呼ばれます。REDO ログは REDO エントリ（「REDO レコード」とも呼ばれます）からなり、各 REDO エントリは、データベースに対する単一のアトミック変更を記述する変更ベクトルのグループです。

REDO ログの主な目的は、データに加えられた変更をすべて記録することです。万一障害が発生して、修正済みのデータがデータ・ファイルに書き込まれなかった場合でも、その変更は REDO ログから取得できるため、実行した作業が失われることはありません。

REDO ログ・ファイルは、障害からデータベースを保護する上で重要です。REDO ログ自体にかかわる障害から保護するため、Oracle では 2 つ以上の REDO ログのコピーを異なるディスク上に維持できるように、「多重化された REDO ログ」を使用できます。

REDO ログ・ファイルの使用方法 REDO ログ・ファイル内の情報は、データベース・データをデータ・ファイルに書き込む妨げとなるシステム障害やメディア障害が起きた場合に、データベースを回復するためにのみ使用されます。

たとえば、予期しない停電によってデータベース操作が突然停止したような場合、メモリー内のデータはデータ・ファイルに書き込まれずに失われます。ただし、電気が復旧した後、データベースがオープンされると、失われたデータがすべて回復されます。Oracle は、最新の REDO ログ・ファイル内にある情報をデータベースのデータ・ファイルに反映させることにより、電源障害が発生した時点までデータベースを復元します。

回復操作中に REDO ログ・ファイルの情報を反映させる処理のことを、「ロールフォワード」と呼びます。1-44 ページの「[データベースのバックアップとリカバリ](#)」を参照してください。

制御ファイル

すべての Oracle データベースには、「制御ファイル」があります。この制御ファイルには、データベースの物理構造を指定するエントリが含まれています。たとえば、制御ファイルには次のようなタイプの情報が含まれます。

- データベース名

- データベースのデータ・ファイルと REDO ログ・ファイルの名前および位置
- データベース作成のタイムスタンプ

Oracle では、REDO ログと同じように、制御ファイルを保護するために制御ファイルを多重化できます。

制御ファイルの使用法 Oracle データベースのインスタンスが起動するたびに、データベース操作のためにオープンする必要があるデータベース・ファイルと REDO ログ・ファイルが、制御ファイルによって識別されます。データベースの物理的な構造が変更されると（データベース・ファイルや REDO ログ・ファイルの新規作成など）、データベースの制御ファイルがその変更を反映するように、Oracle によって自動的に修正されます。

データベースの制御ファイルは、データベースの回復が必要な場合にも使用されます。詳細は、1-44 ページの「[データベースのバックアップとリカバリ](#)」および第 32 章「[データベースの回復](#)」を参照してください。

メモリ構造とプロセス

この項では、データベースを管理するために Oracle Server が使用するメモリとプロセスの構造について説明します。この項で説明するアーキテクチャの特徴によって、特に次のような処理をサポートする Oracle Server の機能を理解することができます。

- 多数のユーザーによる単一のデータベースへの同時アクセス
- 同時実行のマルチユーザー、マルチアプリケーションのデータベース・システムで要求される高パフォーマンス

Oracle Server は、メモリ構造とプロセスを使用してデータベースの管理やアクセスを行います。すべてのメモリ構造は、データベース・システムを構成するコンピュータのメイン・メモリ内に存在します。

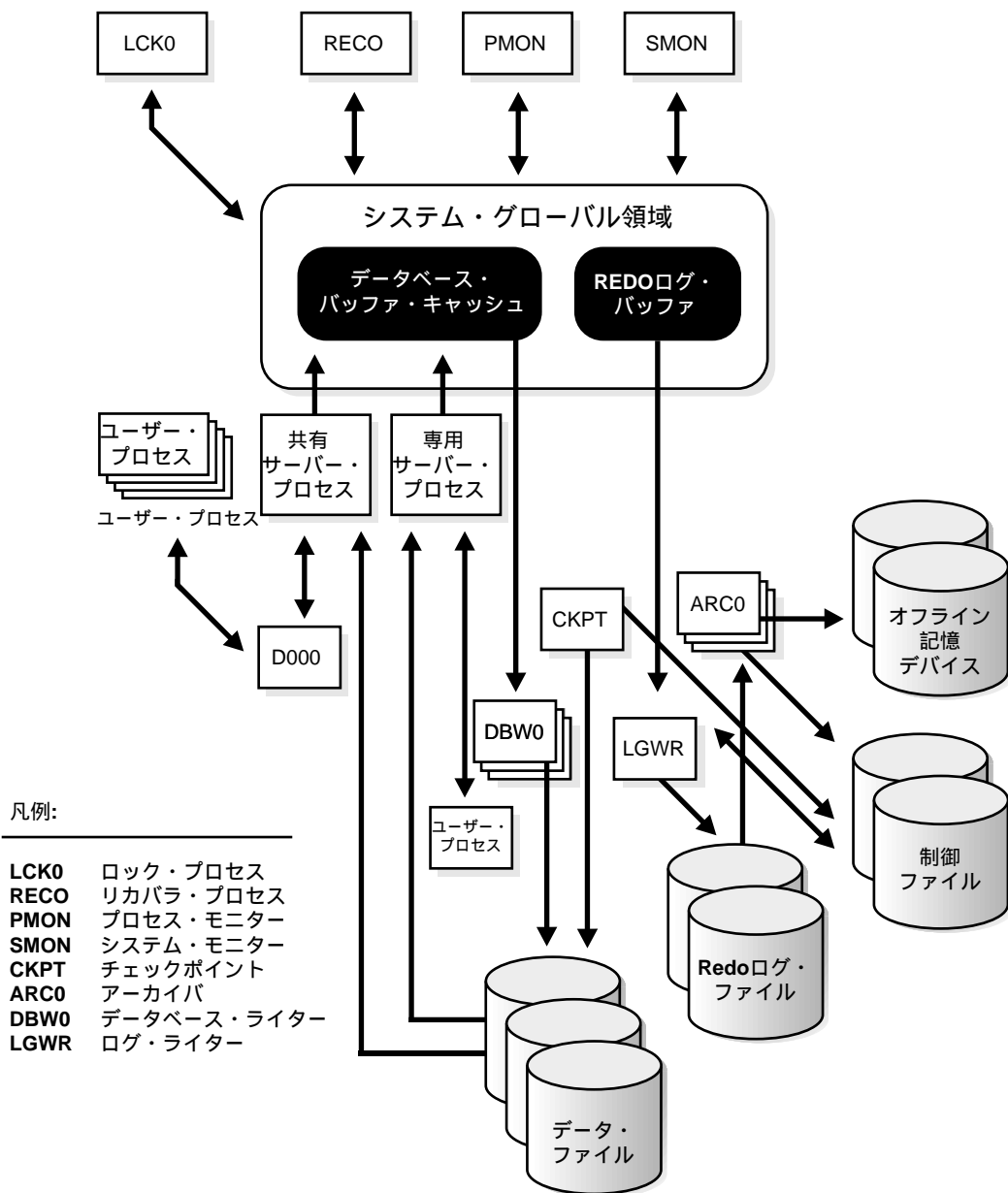
「プロセス」とは、これらのコンピュータのメモリ内で作業を行うジョブまたはタスクです。

1-14 ページの [図 1-3](#) に、Oracle Server のメモリとプロセスの構造の代表的な例を示します。

メモリ構造

Oracle はメモリ構造を作成して使用することによって、いくつかのジョブを完了させます。たとえば、実行中のプログラム・コードやユーザー間で共有されるデータを格納するときに、メモリが使用されます。基本的なメモリ構造、つまりシステム・グローバル領域（データベース・バッファ、REDO ログ・バッファおよび共有プールなど）とプログラム・グローバル領域が、Oracle に対応付けられます。この後、それぞれのメモリ領域について詳しく説明します。

図 1-3 Oracle のメモリー構造とプロセス



システム・グローバル領域 (SGA)

「システム・グローバル領域 (SGA)」は、1 つの Oracle インスタンスのためのデータと制御情報を含む共有メモリ領域です。SGA と Oracle バックグラウンド・プロセスによって、1 つの Oracle インスタンスが構成されます。(詳細は、1-6 ページ「[Oracle インスタンス](#)」および 1-17 ページ「[バックグラウンド・プロセス](#)」を参照してください)。

システム・グローバル領域はインスタンスの起動時に割り当てられ、そのインスタンスの停止時に割当てが解除されます。各インスタンスは、それぞれ専用のシステム・グローバル領域を持っています。

システム・グローバル領域内のデータは、Oracle Server に現在接続しているユーザー間で共有されます。最適なパフォーマンスを得るには、システム・グローバル領域全体をできるだけ (ただし、実メモリーの範囲内で) 大きくして、メモリーにできるだけ多くのデータを格納し、ディスク I/O を最小限にとどめるようにします。

システム・グローバル領域内に格納される情報は、データベース・バッファ、REDO ログ・バッファおよび共有プールなど、いくつかのタイプのメモリー構造に分けられます。これらの領域は固定サイズであり、インスタンスの起動時に作成されます。

データベース・バッファ・キャッシュ システム・グローバル領域の「データベース・バッファ」は、最後に使用されたデータベース・データのブロックを格納します。1 つのインスタンス内のデータベース・バッファの集合が、「データベース・バッファ・キャッシュ」です。このバッファ・キャッシュには、修正済みのブロックおよび未修正のブロックが含まれています。最後に使用されたデータ (多くの場合、最も使用頻度の高いデータ) をメモリー内に保持することによって、必要なディスク I/O が少なくなり、パフォーマンスが向上します。

REDO ログ・バッファ システム・グローバル領域の「REDO ログ・バッファ」は、「REDO エントリ」を格納します。REDO エントリは、データベースに対して加えられた変更のログです。REDO ログ・バッファ内に格納された REDO ログ・エントリは、オンライン REDO ログ・ファイルに書き込まれます。REDO ログ・ファイルは、データベースの回復が必要になったときに使用されます。バッファのサイズは固定されています。

共有プール 共有プールは、共有 SQL 領域などの共有メモリーが含まれている、システム・グローバル領域の一部です。データベースに送られる一意の SQL 文それぞれを処理するために、共有 SQL 領域が必要になります (1-51 ページの「[SQL 文](#)」を参照してください)。共有 SQL 領域には、解析ツリーや、対応する文の実行計画などの情報が格納されます。同じ文を発行する複数のアプリケーションが 1 つの共有 SQL 領域を共用するため、他に使用可能な共有メモリーがより多く残ります。

大規模プール 大規模プールは、Oracle のバックアップおよび復元操作、I/O サーバー・プロセスおよびマルチスレッド・サーバーと Oracle XA のセッション・メモリー用に、大量のメモリー割当てを提供する SGA 内のオプションの領域です。

文ハンドルまたはカーソル「カーソル」は、特定の文に対応付けられたメモリーのハンドル（名前またはポインタ）です。（Oracle コール・インタフェース（OCI）では、これらを「文ハンドル」と呼びます。）多くの Oracle ユーザーは Oracle ユーティリティの自動カーソル処理を使用しますが、アプリケーション設計者はプログラム・インタフェースによってカーソルを自由に制御できます。

たとえば、プリコンパイラのアプリケーション開発では、カーソルはプログラムで使用可能な名前付きのリソースであり、特にアプリケーション内に埋め込まれた SQL 文の解析に使用できます。アプリケーション開発者は、SQL 文の実行フェーズを制御し、そのパフォーマンスが改善されるように、アプリケーションをコーディングできます。

プログラム・グローバル領域（PGA）

「プログラム・グローバル領域（PGA）」とは、1 つのサーバー・プロセスのためのデータと制御情報を含むメモリー・バッファです。サーバー・プロセスの起動時に、Oracle によって作成されます。PGA 内の情報は Oracle の構成によって異なります。

プロセスのアーキテクチャ

「プロセス」は「制御のスレッド」、つまり一連のステップを実行できるオペレーティング・システムのメカニズムです。オペレーティング・システムによっては、「ジョブ」または「タスク」という用語を使用します。プロセスには、通常、プロセスそのものを実行するための専用のプライベート・メモリー領域があります。

Oracle Server には、ユーザー・プロセスおよび Oracle プロセスという 2 つの一般的なタイプのプロセスがあります。

ユーザー（クライアント）プロセス

「ユーザー・プロセス」は、Pro*C/C++ プログラムなどのアプリケーション・プログラムや、Oracle Enterprise Manager などの Oracle Tools のソフトウェア・コードを実行するために作成され、メンテナンスされます。また、ユーザー・プロセスは、サーバー・プロセスとの通信を管理します。

後述のように、ユーザー・プロセスは、プログラム・インタフェースを介してサーバー・プロセスと通信します。

Oracle プロセス・アーキテクチャ

「Oracle プロセス」は、他のプロセスによってコールされ、コールしたプロセスにかわって機能を実行します。各 Oracle プロセスとその特有の機能について、次に説明します。それらのプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。

サーバー・プロセス

Oracle は、接続されたユーザー・プロセスの要求を処理するために、「サーバー・プロセス」を作成します。サーバー・プロセスは、対応付けられたユーザー・プロセスの要求を実行するために、ユーザー・プロセスと通信し、Oracle と対話します。たとえば、ユーザーがシステム・グローバル領域のデータベース・バッファ内に存在しないデータを問い合わせた場合、対応するサーバー・プロセスが、データ・ファイルからシステム・グローバル領域に適切なデータ・ブロックを読み込みます。

Oracle では、サーバー・プロセス当りのユーザー・プロセス数が変動できるように構成できます。「専用サーバー構成」では、1 つのサーバー・プロセスが 1 つのユーザー・プロセスの要求を処理します。「マルチスレッド・サーバー構成」では、多数のユーザー・プロセスが少数のサーバー・プロセスを共有できます。これにより、サーバー・プロセスの数を最低限に抑え、使用可能なシステム・リソースの利用効率を最大化できます。

ユーザー・プロセスとサーバー・プロセスが別個のプロセスになっているシステムと、1 つのプロセスにまとめられているシステムがあります。システムがマルチスレッド・サーバーを使用している場合、またはユーザー・プロセスとサーバー・プロセスが別々のマシン上で実行されている場合、ユーザー・プロセスとサーバー・プロセスは別のプロセスでなければなりません。クライアント / サーバー・システムでは、ユーザー・プロセスとサーバー・プロセスが分離され、それぞれ別々のマシンで実行されます。

バックグラウンド・プロセス

Oracle は、インスタンスごとに 1 組の「バックグラウンド・プロセス」を作成します。バックグラウンド・プロセスは、各ユーザー・プロセスごとに実行されている複数の Oracle プログラムが処理する機能を 1 つにまとめます。バックグラウンド・プロセスは、I/O を非同期的に実行し、他の Oracle プロセスを監視することによって、並列性を強化してパフォーマンスおよび信頼性を向上させます。

SGA と 1 組の Oracle バックグラウンド・プロセスによって、1 つの Oracle インスタンスが構成されます。(SGA の詳細は、1-6 ページの「[Oracle インスタンス](#)」および 1-15 ページの「[システム・グローバル領域 \(SGA\)](#)」を参照してください。) Oracle インスタンスは、それぞれ複数のバックグラウンド・プロセスを使用します。これらのプロセスの名前は、DBWn、LGWR、CKPT、SMON、PMON、ARCn、RECO、Dnnn、LCK0、SNPn および QMNn です。

データベース・ライター (DBWn)「データベース・ライター」は、変更されたブロックをデータベース・バッファのキャッシュからデータ・ファイルに書き込みます。ほとんどのシステムでは 1 つのデータベース・ライター・プロセス (DBW0) があれば十分ですが、データを頻繁に変更するシステムでは、書き込みのパフォーマンスを改善するために追加プロセス (DBW1 ~ DBW9) を構成できます。DBWn プロセスの個数は、初期化パラメータ DB_WRITER_PROCESSES で指定します。

Oracle は事前ロギングを行うので、DBWn がトランザクションのコミット時にブロックを書き込む必要はありません（1-52 ページの「[トランザクション](#)」を参照）。そのかわりに、DBWn はバッチ書き込みを効率よく実行するように設計されています。通常、DBWn が書き込みを行うのは、システム・グローバル領域に読み込む必要のあるデータが増加し、空き状態のデータベース・バッファがほとんどなくなった場合のみです。最低使用頻度のデータから先に、データ・ファイルに書き込まれます。DBWn は、チェックポイント機能などの他の機能のためにも書き込みを行います。

ログ・ライター (LGWR)「ログ・ライター」は、REDO ログ・エントリをディスクに書き込みます。システム・グローバル領域 (SGA) の REDO ログ・バッファ内で REDO ログ・エントリが生成され、LGWR はこの REDO ログ・エントリをオンライン REDO ログ・ファイルに順次書き込みます。多重化された REDO ログがデータベースに存在する場合、LGWR は REDO ログ・エントリをオンライン REDO ログ・ファイルのグループに書き込みます。

チェックポイント (CKPT) 特定のタイミングで、システム・グローバル領域内のすべての修正されたデータベース・バッファが、DBWn によってデータ・ファイルに書き込まれます。このイベントはチェックポイントと呼ばれます。「チェックポイント」プロセスは、チェックポイント時に DBWn に信号を送り、データベースのすべてのデータ・ファイルと制御ファイルを更新する最新のチェックポイントが反映されるように更新します。

システム・モニター (SMON)「システム・モニター」は、障害インスタンスの再起動時にクラッシュ回復を実行します。複数インスタンス・システム (Oracle Parallel Server を使用するシステム) では、あるインスタンスの SMON プロセスが、障害を起こした他のインスタンスのインスタンス回復を実行できます。また、SMON は、使用されなくなった一時セグメントをクリーン・アップし、クラッシュ時とインスタンス回復時にファイル読みエラーまたはオフライン・エラーのためにスキップされたトランザクションを回復します。これらのトランザクションは最終的に、表領域またはファイルがオンライン状態に戻った時点で、SMON によって回復されます。また、SMON はデータベースのディクショナリが管理される表領域内で、使用可能エクステンツを合わせて連続した空き領域を作成し、割当てを容易にします。

プロセス・モニター (PMON) ユーザー・プロセスが障害を起こすと、「プロセス・モニター」がプロセスの回復を実行します。PMON は、キャッシュをクリーン・アップしたり、プロセスが使用していたリソースを解放します。また、ディスパッチャ・プロセス（後述）とサーバー・プロセスをチェックし、障害が発生している場合はそれらを再起動します。

アーカイバ (ARCn)「アーカイバ」は、オンライン REDO ログ・ファイルが満杯になるか、ログ・スイッチが発生すると、それをアーカイブ記憶域にコピーします。ほとんどのシステムの場合は、1 個の ARCn プロセス (ARC0) で十分ですが、LOG_ARCHIVE_MAX_PROCESSES 動的初期化パラメータを使用すると最高 10 個の ARCn プロセスを指定できます。作業負荷が現行の ARCn プロセス数には大きくなりすぎると、LGWR は最高 10 個まで別の ARCn プロセスを自動的に起動します。ARCn がアクティブになるのは、データベースが ARCHIVELOG モードで、自動アーカイブが使用可能になっているときのみです。（1-46 ページの「[REDO ログ](#)」を参照。）

リカバラ (RECO)「リカバラ」は、分散データベースのネットワーク障害またはシステム障害が原因で保留中になっている分散トランザクションを解決するために使用されます。ローカル RECO は、特定の間隔でリモート・データベースに接続し、保留中の分散トランザクションがあれば、そのローカル部分を自動的にコミットまたはロールバックします。

ディスパッチャ (Dnnn)「ディスパッチャ」は、オプションのバックグラウンド・プロセスです。これが存在するのは、マルチスレッド・サーバー構成を使用している場合のみです。使用中の通信プロトコルごとに、少なくとも 1 つのディスパッチャ・プロセス (D000、..、Dnnn) が作成されます。各ディスパッチャ・プロセスは、接続されたユーザー・プロセスから利用できる共有サーバー・プロセスへの要求の経路を指示し、適切なユーザー・プロセスにその応答を戻します。

ロック (LCK0)「ロック」プロセス (LCK0) は、Oracle Parallel Server でインスタンス間ロックに使用されます。詳細は、1-7 ページの「[Oracle Parallel Server：複数インスタンス・システム](#)」を参照してください。

ジョブ・キュー (SNPn)

分散データベース構成では、最大 36 個の「ジョブ・キュー・プロセス」(SNP0、...、SNP9、SNPA、...、SNPZ) が自動的に表スナップショットをリフレッシュできます。これらのプロセスは定期的に起動され、自動リフレッシュがスケジュールされているスナップショットをリフレッシュします。複数のジョブ・キュー・プロセスを使用すると、プロセス間でスナップショットのリフレッシュ・タスクが共有されます。これらのプロセスは、DBMS_JOB パッケージによって作成されるジョブ要求も実行し、キューイングされたメッセージを他のデータベースのキューに波及させます。

キュー・モニター (QMNn)「キュー・モニター」は、Oracle Advanced Queuing (Oracle AQ) のメッセージ・キューをモニターするオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。

プログラム・インタフェース

「プログラム・インタフェース」は、ユーザー・プロセスがサーバー・プロセスとの通信に使用するメカニズムです。プログラム・インタフェースは、クライアント側のツールまたはアプリケーション (Oracle Forms など) と Oracle ソフトウェアの間の標準的な通信手段として機能します。その機能は次のとおりです。

- 通信メカニズムとしての機能。データ要求の形式設定、データの引渡しおよびエラーの検出と返送を行います。
- データ変換の機能。特に異機種コンピュータ間での変換、または外部ユーザー・プログラムのデータ型への変換を行います。

Oracle の動作例

次に、ユーザー・プロセスとそれに対応するサーバー・プロセスが別々のマシン（ネットワークを介して接続）上に存在する Oracle の構成について、具体的に説明します。

1. 現在、インスタンスは、Oracle が稼働しているコンピュータ（「ホスト」または「データベース・サーバー」と呼ばれることが多い）上で実行されています。
2. アプリケーションを実行しているコンピュータ（「ローカル・マシン」または「クライアント・ワークステーション」）は、ユーザー・プロセスでアプリケーションを実行します。クライアント・アプリケーションは、適切な Net8 ドライバを使用して、サーバーへの接続を確立しようとします。
3. サーバーは、適切な Net8 ドライバを実行しています。サーバーはアプリケーションからの接続要求を検出すると、ユーザー・プロセスのために（専用の）サーバー・プロセスを作成します。
4. ユーザーは SQL 文を実行して、トランザクションをコミットします。たとえば、ユーザーは表の行に入っている名前を変更します。
5. サーバー・プロセスは SQL 文を受け取り、同一の SQL 文を含む共有 SQL 領域がないかどうか共有プールをチェックします。共有 SQL 領域が見つかった場合、サーバー・プロセスは要求されたデータに対するユーザーのアクセス権限をチェックし、既存の共有 SQL 領域を使用して SQL 文を処理します。共有 SQL 領域が見つからなかった場合には、解析と処理を実行するための新しい共有 SQL 領域をその文に割り当てます。
6. サーバー・プロセスは、実際のデータ・ファイル（表）から必要なデータ値を取得するか、またはシステム・グローバル領域内に格納されている必要なデータ値を取得します。
7. サーバー・プロセスは、システム・グローバル領域内のデータを修正します。DBW n プロセスは、書込みが効率よく行われる時期を判断して、修正したブロックをディスクに書き込みます。トランザクションがコミットされると、LGWR プロセスがそのトランザクションをオンライン REDO ログ・ファイルに即時に記録します。
8. トランザクションが成功すると、サーバー・プロセスは、ネットワークを介してアプリケーションにメッセージを送信します。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体を通じて、他のバックグラウンド・プロセスも実行されていて、介入が必要かどうかを監視しています。さらに、データベース・サーバーは、他のユーザーのトランザクションを管理し、同一のデータを要求する異なるトランザクション間の競合を防止します。

これらのステップは、Oracle が実行する操作の最も基本的なレベルにすぎません。（第 8 章「プロセスのアーキテクチャ」を参照。）

データベース管理のオブジェクト・リレーショナル・モデル

データベース管理システムは、階層型からネットワークへ、そしてネットワークからリレーショナル・モデルへと発展してきました。主流になっているデータベース・モデルは、「リレーショナル・モデル」です。Oracle は、リレーショナル・モデルをさらに「オブジェクト・リレーショナル・モデル」へと拡張しています。オブジェクト・リレーショナル・モデルでは、複雑な業務モデルをリレーショナル・データベースに格納できます。

リレーショナル・モデル

リレーショナル・モデルには、次のような 3 つの要素があります。

構造体	構造体とは、データベースのデータを格納またはアクセスするための、正しく定義されたオブジェクト（表、ビューおよび索引など）のことです。構造体とそこに含まれるデータは、「操作」によって処理されます。
操作	操作とは、ユーザーがデータベースのデータや構造を処理するための、明確に定義されたアクションのことです。データベースの操作は、事前に定義された整合性ルールに従う必要があります。
整合性ルール	整合性ルールとは、データベースのデータや構造に対してどの操作を実行できるかを定めるルールです。このルールにより、データベースのデータや構造が保護されます。

リレーショナル・データベース管理システムには、次のような利点があります。

- 物理データ記憶域と論理データベース構造が独立している
- すべてのデータに柔軟に、また簡単にアクセスできる
- データベース設計に柔軟性を持たせることができる
- データの記憶域と冗長性を軽減できる

オブジェクト・リレーショナル・モデル

オブジェクト・リレーショナル・モデルでは、ユーザーは「オブジェクト型」を定義できます。オブジェクト型には、データの構造と、データを操作するメソッドの両方を指定し、そのデータ型をリレーショナル・モデルで使用できます。

注意： ユーザー定義のオブジェクト型を使用できるのは、Oracle8i Enterprise Edition を購入した場合のみです。

オブジェクト型は、アプリケーション・プログラムが取り扱う実社会のエンティティ（注文など）を抽象化したものです。オブジェクト型は、次の3種類の要素から構成されます。

- 「名前」。これは、オブジェクト型を一意に識別するものです。
- 「属性」。これは、ビルトイン・データ型、またはその他のユーザー定義型です。属性は、実社会のエンティティ（実体）の構造をモデル化します。
- 「メソッド」。これは、PL/SQL で作成されてデータベースに格納されたファンクションやプロシージャ、またはCなどの言語で記述されて外部に格納されたファンクション（関数）や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データに対して実行できる特定の操作をインプリメントします。すべてのオブジェクト型には、データ型の仕様に基づいて新しいオブジェクトを作成するための「コンストラクタ・メソッド」があります。

スキーマとスキーマ・オブジェクト

「スキーマ」は、ユーザーが使用可能なデータベース・オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラストおよびデータベース・リンクなどの構造体が含まれます。（表領域とスキーマには何の関連もありません。同じスキーマ内のオブジェクトが別々の表領域に存在したり、異なるスキーマにあるオブジェクトが1つの表領域に保持されたりすることがあります。）

表

「表」は、Oracle データベースにおけるデータ格納の基本単位です。データベースの表には、ユーザーがアクセスできるすべてのデータが保持されています。

表データは「行」と「列」に格納されます。すべての表は、「表名」と列の集合によって定義します。各列には、「列名」、「データ型」（CHAR、DATE または NUMBER など）および「幅」（DATE のようにデータ型によって事前定義されていることもある）や「位取り」と「精度」（NUMBER データ型の場合のみ）を指定します。表の作成後は、そこに有効なデータ行を挿入していくことができます。表の行は問合せ、削除または更新できます。

Oracle8i Enterprise Edition の Partitioning Option を使用すると、表を「パーティション化」できます。詳細は、[第 11 章「パーティション表とパーティション索引」](#)を参照してください。

表のデータに対して定義された業務規則を施行するために、整合性制約とトリガーを表に対して定義できます。詳細は、1-56 ページの「[データの整合性](#)」を参照してください。

ビュー

「ビュー」とは、1 つ以上の表のデータをユーザーの必要に合せて調整したデータ表現です。ビューは、「格納された問合せ」とみなすこともできます。

ビューに実際にデータが格納されているわけではありません。データは、ビューの基礎となる表から導出されます。これらの表をビューの「実表」といいます。実表は表の場合もあれば、それ自体がビューの場合もあります。

表の場合と同じように、ビューに対しても、いくつかの制限付きで問合せ、更新、挿入および削除を実行できます。ビューに対して実行する操作は、すべて実表にも影響します。

通常、ビューは次のような目的で使用されます。

- 事前に定義された行と列の集合のみにアクセスを限定することにより、表のセキュリティ・レベルを強化する。たとえば、表のビューを作成するときに、機密性の高いデータ（給与情報など）が格納されている列はビューに含めないように定義できます。
- データの複雑さを隠す。たとえば、12 個の月別売上表を組み合わせた 1 つのビューを作成して、年間データを分析し報告できます。また、複数の表から関連する列や行を「結合」して表示するビューを作成することもできます。しかも、データが実際にはいくつかの表から取られていることを感じさせずに表示できます。
- ユーザーのコマンドを単純化する。たとえば、ユーザーが相関副問合せについて知らなくても、複数の表から情報を選択できるようにします。
- 実表とは異なる視点からデータを提示する。たとえば、実表の列名を変更せずに、ビューで列名を改名できます。
- 複雑な問合せを保存する。たとえば、ある問合せで表の情報に対して膨大な計算を実行したとします。この問合せをビューとして保存しておけば、そのビューに問合せを行うときにのみ同じ計算が実行されます。

2 つ以上の表の結合（複数の表からデータを選択する SELECT 文）を実行するビューは、特定の条件下でしか更新できません。詳細は、10-15 ページの「更新可能な結合ビュー」を参照してください。

マテリアライズド・ビュー

「マテリアライズド・ビュー」は、問合せ結果を別々のスキーマ・オブジェクトに格納することによって、表データへの間接アクセスを提供します。通常のビューは記憶領域を占めず、データも含まれていませんが、マテリアライズド・ビューには、1 つ以上の実表やビューに対する問合せで得られた行が含まれます。マテリアライズド・ビューの格納場所は、実表と同じデータベースでも、異なるデータベースでもかまいません。

マテリアライズド・ビューをそのマスター表と同じデータベースに格納すると、「クエリー・リライト」を通じて問合せのパフォーマンスを改善できます。集合データや結合を伴う問合せの場合、オプティマイザは、事前に計算され、マテリアライズド・ビューに格納されている結果にアクセスするように、問合せをリライトできます。クエリー・リライトは、データ・ウェアハウス環境で特に役立ちます。

マテリアライズド・ビューは、「スナップショット」とも呼ばれます。通常、この用語は、リモート・データベース内のデータをレプリケートするために使用される、マテリアライズド・ビューを指します。（1-35 ページの「表のレプリケーション」を参照。）SQL 文では、キーワード SNAPSHOT および MATERIALIZED VIEW は同義です。

順序

「順序」は、データベース表の数値列について、連続する一意の数値のリストを生成します。順序を使用すると、1つの表または複数の表の行に対して一意の数値が自動的に生成されるので、アプリケーションのプログラミングが容易になります。

たとえば、2人のユーザーがEMP表に新しい従業員の行を同時に挿入しているとします。順序を使用してEMPNO列に一意の従業員番号を生成すると、一方のユーザーは、もう一方のユーザーが従業員番号を入力するのを待つ必要はありません。どちらのユーザーにも、順序により、正しい値が自動的に生成されるためです。

順序番号は表とは独立しているので、同じ順序を1つ以上の表に対して指定できます。作成された順序は、さまざまなユーザーがアクセスしたとき、実際の順序番号を生成します。

プログラム・ユニット

このマニュアルで使用している「プログラム・ユニット」という用語は、ストアド・プロシージャ、ファンクション、パッケージ、トリガーおよび無名ブロックを指します。

「プロシージャ」または「ファンクション」は、SQLおよびPL/SQL（SQLに対するOracleのプロシージャ型言語拡張機能）文を、特定の作業を行う実行単位としてグループ化したものです。SQLとPL/SQLの詳細は、1-50ページの「[データ・アクセス](#)」を参照してください。

プロシージャとファンクションを使用すると、SQLが持つ平易さや柔軟性と、構造型プログラミング言語が持つプロシージャ的な機能性を合体できます。PL/SQLを使用して、このようなプロシージャやファンクションを定義し、継続して使用できるようにデータベースに格納します。プロシージャとファンクションはほとんど同じですが、唯一異なる点として、プロシージャはコール元に値を戻さないのに対して、ファンクションは必ず1つの値を戻します。

「パッケージ」は、関連したプロシージャ、ファンクションおよびその他のパッケージ構成体をカプセル化し、データベースの単位としてまとめて格納するための手段です。パッケージにより、データベース管理者やアプリケーション開発者の操作が効率的になるだけでなく、データベースの機能性やパフォーマンスも向上します。

シノニム

「シノニム」とは、表、ビュー、順序またはプログラム・ユニットの別名のことです。シノニムはスキーマ・オブジェクトそのものではなく、スキーマ・オブジェクトへの直接参照です。次のような目的で使われます。

- スキーマ・オブジェクトの実名と所有者を隠す
- スキーマ・オブジェクトへのパブリックなアクセスを実現する
- リモート・データベースの表、ビューまたはプログラム・ユニットについて位置の透過性を実現する
- データベース・ユーザー側のSQL文を単純にする

シノニムは、パブリックおよびプライベートのどちらにもできます。個々のユーザーは、当人のみが使用できる「プライベート・シノニム」を作成できます。「パブリック・シノニム」は、多くの場合、データベース管理者によって作成されます。パブリック・シノニムにより、基礎となるスキーマ・オブジェクトをすべてのデータベース・ユーザーが一般的な方法で、またシステム全体で使用できるようになります。

索引

「索引」は、表と対応付けられるオプションの構造体です。データ検索のパフォーマンスを向上させるために作成します。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。それと同じように、Oracle の索引を使用すると表データに高速にアクセスできます。

1 つの要求を処理するとき、Oracle は、要求された行を効率よく見つけるために使用可能な索引をいくつか（またはすべて）使用します。索引が有効なのは、アプリケーションが表内のある範囲の行（給与が 1000 ドルを超えるすべての従業員など）または特定の行を頻繁に問い合わせる場合です。

索引は、表の 1 つ以上の列に対して作成されます。作成された索引は、Oracle により自動的に維持され、使用されます。表データに対する変更（新しい行の追加、行の更新または削除など）は、関連するすべての索引にも自動的に取り込まれます。その操作は、ユーザーに対して透過的です。

索引は、論理的にも物理的にもデータから独立しています。表や他の索引に影響を及ぼすことなく、いつでも削除したり作成できます。索引を削除してもアプリケーションはすべて引き続き機能しますが、それまで索引が設定されていたデータに対するアクセスは低速になります。

Oracle8i Enterprise Edition の Partitioning Option を使用すると、索引を「パーティション化」できます。詳細は、[第 11 章「パーティション表とパーティション索引」](#)を参照してください。

クラスタとハッシュ・クラスタ

クラスタとハッシュ・クラスタは、表データを格納するためのオプションの構造体です。これらのクラスタは、データ検索のパフォーマンスを高めるために作成します。

クラスタ化表「クラスタ」とは、物理的にまとめて格納される 1 つ以上の表のグループです。それらの表は、共通の列を共有しており、しばしば一緒に使用されるため、まとめて格納されます。関連する行が物理的にまとめて格納されているため、ディスク・アクセス時間が短縮されます。

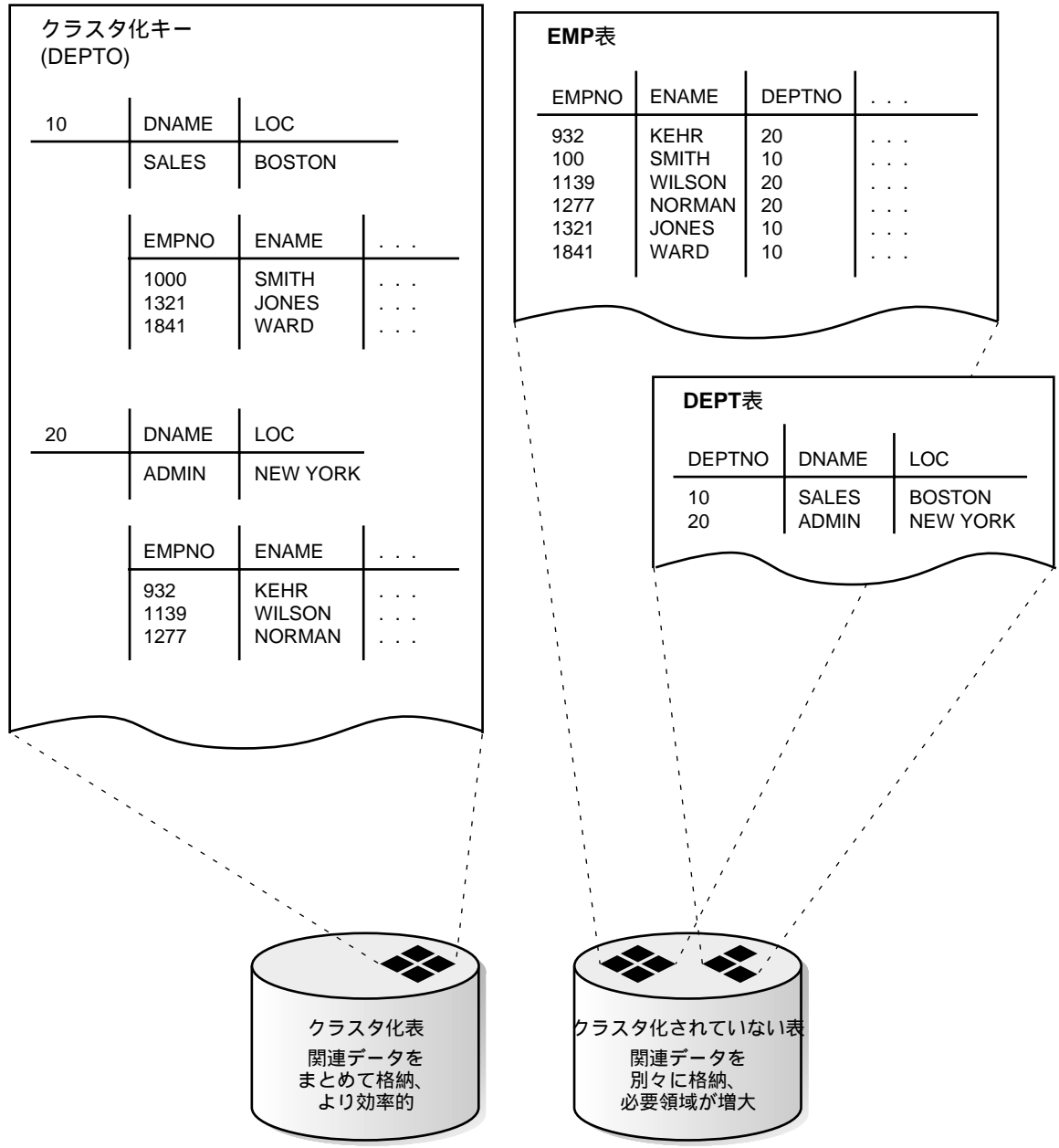
クラスタ内の表にあって相互に関連する列のことを「クラスタ・キー」といいます。クラスタの行が最小限の I/O で検索できるように、クラスタ・キーには索引が設定されています。索引クラスタ（非ハッシュ・クラスタ）のクラスタ・キーに含まれるデータは、複数の表について 1 回だけ格納すればよいので、表が個別に格納されている（クラスタ化されていない）場合と比べると、クラスタは一連の表をより効率的に格納できます。

図 1-4 に、クラスタ化されたデータとクラスタ化されていないデータがどのように物理的に格納されるかを示します。

データの分散状況や、データに対して最も頻繁に実行される SQL 操作にもよりますが、クラスタを使用するとデータ検索のパフォーマンスも向上します。特に、クラスタ化された表を結合して問い合わせる場合、結合された表に共通する行は同じ I/O 操作で検索されるので、クラスタを使用することに利点があります。

索引と同じように、クラスタがアプリケーションの設計に影響することはありません。表がクラスタの一部になっているかどうかは、ユーザーとアプリケーションに対して透過的です。クラスタ化表に格納されているデータは、クラスタ化されていない表のデータと同じように SQL を使用してアクセスします。

図 1-4 クラスタ化表とクラスタ化されていない表



ハッシュ・クラスタ「ハッシュ・クラスタ」は、通常の索引クラスタ（ハッシュ関数ではなく索引がキーになっているクラスタ）の場合と同様の方法で表データをクラスタ化します。ただし、行は、行のクラスタ・キー値に「ハッシュ関数」を適用した結果に基づいてハッシュ・クラスタに格納されます。キーの値が同じすべての行は、ディスク上にまとめて格納されます。

等式を条件にした問合せを使用して表を問い合わせる（部門が 10 のすべての行を戻すなど）ことが多い場合、索引表または索引クラスタよりも、ハッシュ・クラスタの方が適切です。ハッシュ・クラスタを使用する問合せでは、指定されたクラスタ・キー値がハッシュ化されます。結果として生成されるハッシュ・キー値は、指定された行が格納されているディスク領域を直接指します。

ディメンション

「ディメンション」は、1 対の列または列セット間の階層（親子）関係を定義します。子レベルの値は、それぞれが親レベルの 1 つの値にのみ対応付けられます。

ディメンション・スキーマ・オブジェクトは、表相互の論理関係のコンテナであり、データ記憶域は対応付けられていません。CREATE DIMENSION 文では、次の事項を指定します。

- 複数の LEVEL 句。それぞれがディメンション内の 1 列または列セットを識別します。
- 1 つ以上の HIERARCHY 句。隣接する LEVEL 間の親子関係を指定します。
- オプションの ATTRIBUTE 句。それぞれが、個々の LEVEL に対応付けられている他の列または列セットを識別します。

ディメンション内の列は、同じ表からのもの（「非正規化」）でも、または複数の表からのもの（「完全または一部正規化」）でもかまいません。複数の表からの列によるディメンションを定義するには、HIERARCHY 句の JOIN KEY 句を使用して表を内側の等価結合で接続します。

データベース・リンク

「データベース・リンク」とは、あるデータベースから別のデータベースへの「パス」を記述する名前付きスキーマ・オブジェクトです。分散データベースで「グローバル・オブジェクト名」が参照されると、データベース・リンクが暗黙に使用されます。詳細は、1-33 ページの「[分散データベース](#)」および第 33 章「[分散データベース](#)」を参照してください。

データ・ディクショナリ

どの Oracle データベースにも、「データ・ディクショナリ」が 1 つずつあります。Oracle データ・ディクショナリは、データベースの「読み専用」の参照として使用される、表およびビューの集合です。たとえば、データ・ディクショナリには、データベースの論理構造と物理構造の両方に関する情報が格納されます。このような重要な情報の他にも、データ・ディクショナリには次のような情報が格納されます。

- Oracle データベースの有効なユーザー

- データベース内の表に定義された整合性制約に関する情報
- スキーマ・オブジェクトに割り当てられている領域の量とその使用率

データ・ディクショナリは、データベースの作成時に作成されます。常にデータベースの正確な状態が反映されるよう、データ・ディクショナリは、特定のアクション（データベース構造の変更など）が実行されるたびに、Oracle により自動的に更新されます。データベースを運用するうえで、データ・ディクショナリは重要な役割を果たします。進行する作業の記録、検証および指示を行うからです。たとえば、データベースの操作中、Oracle は、スキーマ・オブジェクトが存在しているかどうか、およびユーザーが適切なアクセス権を持っているかどうかを検証するためにデータ・ディクショナリを読み込みます。

データの同時実行性と一貫性

この項では、情報管理システムの次のような重要な要件を満たすために Oracle が使用するソフトウェアのメカニズムについて説明します。

- データを一貫した方法で読み込み、修正する。
- マルチユーザー・システムのデータの同時実行性を最大限に高める。
- データベース・システムの数多くのユーザーの生産性を最大にするための高パフォーマンスを実現する。

同時実行性

マルチユーザー・データベース管理システムの最大の関心は、「同時実行性」をどのように制御するか、つまり多数のユーザーによる同一データへの同時アクセスをどのように制御するかということです。十分な同時実行制御機能がなければ、データが不当に更新または変更され、データの整合性が損なわれる可能性があります。

多数のユーザーが同一データにアクセスしている場合、データの同時実行性を管理する 1 つの方法は、各ユーザーに順番待ちをさせることです。データベース管理システムの目標は、その待ち時間を、各ユーザーにとって存在しないか、または無視できる程度まで短縮することです。すべてのデータ操作（DML）文をできるだけ干渉がなくなるように処理し、同時実行のトランザクション間の破壊的な相互作用を防止する必要があります。破壊的な相互作用とは、不正確なデータの更新または基礎となるデータ構造の不正な変更を引き起こす相互作用です。パフォーマンスとデータの整合性は、どちらも無視できません。

Oracle では、さまざまなタイプのロックとマルチバージョン一貫性モデルを使用することで、このような問題を解決します。この 2 つの機能については、この項で後述します。これらの機能は、トランザクションの概念に基づいています。1-54 ページ「[トランザクションを使用したデータの一貫性](#)」で後述するように、同時実行性と一貫性に関するこれらの機能がトランザクションで十分に活用されるようにするのは、アプリケーション設計者の責任です。

読み込み一貫性

Oracle がサポートする読み込み一貫性は、次のようなことを保証します。

- 文が参照するデータの集合が、ある特定の時点で一貫しており、文の実行中に変化しない（文レベルの読み込み一貫性）。
- データベース・データを読み込むユーザーは、その同じデータを書き込むユーザーまたはそれを読み込む他のユーザーのために待機しない。
- データベース・データを書き込むユーザーは、その同じデータを読み込むユーザーのために待機しない。
- 同時実行のトランザクションで同一行を更新しようとする場合にのみ、データを書き込むユーザーは他の書き込みユーザーを待機する。

Oracle の読み込み一貫性の実現について考える最も簡単な方法は、ユーザーがそれぞれ自分専用のデータベースのコピーを操作している状況を想定することです（つまりマルチバージョンの一貫性モデル）。

読み込み一貫性、ロールバック・セグメントおよびトランザクション

マルチバージョンの一貫性モデルを管理するために、表に対する問合せ（読み込み）や同時更新（書き込み）の実行時に Oracle は読み込み一貫性を備えたデータの集合を作成する必要があります。更新が発生すると、それによって変更されたデータの元の値が、データベースのロールバック・セグメントに記録されます。この更新がコミットされていないトランザクションの一部である限り、修正されたデータを後から問い合わせたユーザーはデータの元の値を参照することになります。つまり、Oracle はシステム・グローバル領域内の現在の情報とロールバック・セグメント内の情報を使用して、問合せのために表データの「読み込み一貫性」を備えたビューを作成します。

トランザクションがコミットされた時点で初めて、トランザクションの変更が確定します。ユーザーのトランザクションがコミットされた「後に」開始された文だけが、そのコミットされたトランザクションによって加えられた変更を参照することになります。

トランザクションは、読み込み一貫性を実現するための重要な機能であることに注意してください。コミット済みの（またはコミットされていない）SQL 文から構成されているトランザクションは、次のように機能します。

- 読み込みユーザーのために生成される、読み込み一貫性を備えたビューの開始点を示す。
- データベースの他のトランザクションが、修正されたデータを読み込みまたは更新のために参照できる時期を制御する。

読み専用トランザクション

デフォルトでは、文レベルの読み一貫性が保証されます。1つの問合せによって戻された一連のデータは、ある特定の時点での一貫性を保っています。ただし、場合によっては、この他にトランザクション・レベルの読み一貫性が必要になることもあります。これは、1つのトランザクション内で複数の問合せを実行するときに、そのトランザクション内の問合せが、途中でコミットされた他のトランザクションの結果を参照しないように、1つのトランザクション内のすべての問合せに同一時点を基準とした読み一貫性を持たせる機能です。

複数の表に対して多数の問合せを実行し、更新をしない場合は、「読み専用トランザクション」を使用します。トランザクションが読み専用であることを指定した後は、それぞれの問合せの結果が同一時点について読み一貫性を持つことがわかっているため、任意の表に対して必要な数の問合せを実行できます。

ロックのメカニズム

Oracle は、さらに「ロック」も使用してデータへの同時アクセスを制御します。ロックは、Oracle データにアクセスするユーザー間で破壊的な相互作用が起きるのを防止するためのメカニズムです。

ロックは、次の2つの重要なデータベースの目標を達成するために使用されます。

一貫性	ユーザーがデータを使用し終わるまで、参照中または変更中のデータが（他のユーザーによって）変更されないことを保証する。
整合性	データベースのデータと構造が、すべての変更を正しい順序で反映していることを保証する。

ロックは、データの整合性を保証すると同時に、無制限の数のユーザーがデータに同時実行アクセスできるようにします。

自動ロック

Oracle のロックは自動的に実行されるため、ユーザーのアクションは必要ありません。要求されるアクションによっては、必要に応じて暗黙のロックが SQL 文に対して発生します。

Oracle の洗練されたロック・マネージャ（ロック管理機能）は、行レベルで表データを自動的にロックします。行レベルで表データをロックすることによって、同一データへの競合が最小限に抑えられます。

Oracle のロック・マネージャは、ロックを設定する操作のタイプに応じて、異なるタイプの行ロックを保持します。ロックには一般に、「排他ロック」と「共有ロック」の 2 つのタイプがあります。排他ロックは、1 つのリソース（行や表など）に対して 1 つだけ取得できます。一方、共有ロックは、1 つのリソースに対して数多く取得できます。排他ロックと共有ロックでは、ロックされたリソースに対する問合せはいつでも許可されますが、そのリソースに対する他のアクティビティ（更新や削除など）は禁止されます。

手動ロック

状況によっては、ユーザーは、デフォルト・ロックを変更できます。Oracle では、自動ロック機能を行レベル（後続の文で更新する行を最初に問い合わせしておく）と表レベルの両方で手動変更できます。

分散処理と分散データベース

現代のコンピュータ環境においてコンピュータ・ネットワーキングが普及するにつれて、データベース管理システムには、処理機能と格納機能を分散する機能が必要とされています。この項では、これらの要件に応える Oracle のアーキテクチャ上の特徴について説明します。

クライアント / サーバー・アーキテクチャ : 分散処理

「分散処理」では、関連するジョブの集合に対する処理を分割するために、複数のプロセッサを使用します。分散処理では、異なるプロセッサが関連するタスクのサブセットを集中処理することにより、1 つのプロセッサの処理負荷が低減され、システム全体のパフォーマンスと能力が向上します。

Oracle データベース・システムでは、「クライアント / サーバー・アーキテクチャ」を使用することにより、分散処理を容易に活用できます。このアーキテクチャでは、データベース・システムが、フロントエンドの「クライアント」部およびバックエンドの「サーバー」部という 2 つに部分に分割されます。

クライアント

クライアント部は、フロントエンドのデータベース・アプリケーションであり、キーボード、ディスプレイおよびマウスなどのポインティング・デバイスを使用してユーザーと対話します。クライアント部は、データ・アクセスは受け持たず、サーバー部によって管理されるデータの要求、処理および表示を集中的に実行します。クライアント・ワークステーションは、このようなジョブに合わせて最適化できます。たとえば、必要なディスク容量の縮小や、グラフィック機能の活用ができます。

サーバー

サーバー部は、Oracle ソフトウェアを実行し、同時実行の共有データ・アクセスに必要な機能を処理します。サーバー部は、クライアント・アプリケーションから発行された SQL 文や PL/SQL 文を受け取って処理します。サーバー部を管理するコンピュータは、このような処理内容に応じて最適化できます。たとえば、大容量のディスクや高速プロセッサを搭載できます。

複数層アーキテクチャ：アプリケーション・サーバー

「複数層アーキテクチャ」の構成要素は、次のとおりです。

- 操作を開始するクライアントまたは起動側プロセス。
- 操作の各部分を実行する 1 つ以上のアプリケーション・サーバー。「アプリケーション・サーバー」は、クライアント用のデータにアクセスし、なんらかの問合せ処理を実行することによって、データベース・サーバーの負荷をある程度軽減するプロセスです。アプリケーション・サーバーは、クライアントと複数のデータベース・サーバー間のインタフェースとして働き、セキュリティ・レベルを高める役割を果たします。
- 操作に使用されるほとんどのデータのリポジトリとして機能するエンド・サーバーまたはデータベース・サーバー。

このアーキテクチャでは、アプリケーション・サーバーを使用して次のことを実行できます。

- Web ブラウザなど、クライアントの資格証明を検証する。
- Oracle データベース・サーバーに接続する。
- クライアントにかわって要求された操作を実行する。

クライアントの認証は、接続のすべての層で保たれます。Oracle データベース・サーバーは、アプリケーション・サーバーが独自に実行する操作（データベース・サーバーへの接続要求など）とは別に、クライアントにかわって実行する操作を監査します。アプリケーション・サーバーの権限は、クライアント操作中に不要な操作や望ましくない操作を実行できないように制限されます。

分散データベース

「分散データベース」は、ユーザーからは単一の論理データベースのように見えますが、実際には複数のデータベース・サーバーによって管理される、データベースのネットワークです。分散データベース内のすべてのデータベースのデータは、同時にアクセスや変更ができます。分散データベースの主な利点は、物理的には別々のデータベースに格納されているデータを論理的に結合し、ネットワーク上のすべてのユーザーに対してアクセス可能にできることです。

分散データベース内のデータベースを管理する各コンピュータを「ノード」と呼びます。ユーザーが直接接続するデータベースを「ローカル」データベース、このユーザーがアクセスする他のデータベースを「リモート」データベースと呼びます。ローカル・データベースが情報を取得するためにリモート・データベースにアクセスするとき、ローカル・データベースはリモート・サーバーのクライアントになります（クライアント／サーバー・アーキテクチャについては前述しました）。

分散データベースでは、ネットワークを介した大量のデータに対するアクセスを増加させることができますが、その一方で、データの位置とネットワークをまたがるアクセスの複雑さを隠す機能も提供しなければなりません。分散データベース管理システムでは、さらに各ローカル・データベースを非分散データベースと同様に管理できるという特長も維持していなければなりません。

位置の透過性

データベース・システムのアプリケーションとユーザーがデータの物理的な位置を意識しないですむ（透過である）とき、「位置の透過性」が実現されます。ビュー、プロシージャおよびシノニムなど、Oracle のいくつかの機能を利用して位置の透過性が実現されます。たとえば、複数のデータベースの表データを結合するビューによって、位置の透過性が実現されます。つまり、この場合、ビューのユーザーはデータの物理的な位置を知る必要がありません。

サイト自律性

「サイト自律性」とは、分散データベースの一部となっている各データベースが、ネットワーク化されていないときと同じように、他のデータベースから独立して個別に管理されることを意味します。各データベースは他のデータベースと協調して動作しますが、それらは個別に管理される別々のシステムです。

分散データ操作

Oracle の分散データベース・アーキテクチャは、リモート表のデータの問合せ、挿入、更新および削除など、すべての DML 操作をサポートしています。リモート・データをアクセスするには、リモート・オブジェクトのグローバル・オブジェクト名を含めた参照を作成します。リモート・データにアクセスするための特別なコーディングや複雑な構文は必要ありません。

たとえば、リモート・データベース SALES の表 EMP を問い合わせるには、次のようにして表のグローバル・オブジェクト名を参照します。

```
SELECT * FROM emp@sales;
```

2 フェーズ・コミット

Oracle は、分散環境でも、非分散環境と同じようにデータの一貫性を保証できます。Oracle では、トランザクション・モデルと「2 フェーズ・コミット・メカニズム」を使用して、この一貫性を保証します。

分散システム以外のシステムと同様にトランザクションを慎重に計画し、論理的な SQL 文の集合を 1 単位としてトランザクションに含めることにより、それらがすべて成功するか、さもなければすべて失敗するというようにできます。Oracle の 2 フェーズ・コミット・メカニズムによって、発生したシステム障害やネットワーク障害のタイプに関係なく、分散トランザクションがすべての関連ノード上でコミットまたはロールバックされ、グローバルな分散データベースでのデータの一貫性を維持できることが保証されます。

データベース・ユーザーに対する完全な透過性 Oracle の 2 フェーズ・コミット・メカニズムは、分散トランザクションを発行するユーザーに対して完全に透過的です。トランザクションの終了を示す単純な COMMIT 文が、トランザクションをコミットするために 2 フェーズ・コミット・メカニズムを自動的に起動します。つまり、データベース・アプリケーションの本体に、分散トランザクションを含めるための特別なコーディングや複雑な文の構文は必要ありません。

システム障害やネットワーク障害からの自動回復 RECO (リカバラ) バックグラウンド・プロセスは、「インダウト分散トランザクション」(システム障害やネットワーク障害によってコミットが中断された分散トランザクション)の結果を自動的に解決します。障害が修復され、通信が再確立された後、各ローカル Oracle Server の RECO が、関係するすべてのノード上でインダウト分散トランザクションを一貫して自動的にコミットするか、またはロールバックします。

手動による問題解決の機能 長時間にわたる障害が発生した場合、Oracle では、各ローカル管理者が、障害によって発生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か、またはロールバックできます。このオプションによって、ローカル・データベースの管理者は、長時間にわたる障害の結果として無期限に拘束される可能性のあるロックされたりソースを解放できます。

分散解決のための機能 特定のデータベースを過去の特定の時点まで回復する必要がある場合、Oracle の回復機能を使用することによって、他のサイトのデータベース管理者は各自のデータベースをその過去の時点に戻すことができます。これによって、グローバル・データベースは一貫した状態を保つことができます。

表のレプリケーション

分散データベース・システムでは、ローカル・ユーザーが頻繁に問い合わせるリモート表がしばしばレプリケートされます。頻繁にアクセスされるデータのコピーを複数のノード上に置くことにより、分散データベースでネットワークを介して繰り返し情報を送信する必要がなくなるため、データベース・アプリケーションのパフォーマンスを最大化する上で役立ちます。

データは、マテリアライズド・ビュー (スナップショット) を使用してレプリケートできます。

Oracle と Net8

「Net8」は、ネットワークで使用される通信プロトコルとのインタフェースとなる Oracle のメカニズムであり、分散処理と分散データベースの実現を支援します。通信プロトコルにより、ネットワークでのデータの送受信方法が定義されます。ネットワーク環境では、Oracle データベース・サーバーは、Oracle ソフトウェアである Net8 を使用して、クライアント・ワークステーションや他の Oracle データベース・サーバーと通信します。

PC LAN でサポートされているプロトコルや大規模メインフレーム・コンピュータ・システムでサポートされているプロトコルなど、Net8 は主要なネットワーク・プロトコルによる通信を広い範囲ですべてサポートします。

Net8 を使用すると、アプリケーション開発者がデータベース・アプリケーションでネットワーク通信のサポートにかかわる必要はありません。新しいプロトコルを使用する場合も、データベース管理者がわずかな変更を行うだけです。アプリケーションは修正しなくても機能し続けます。

起動操作と停止操作

最初に Oracle Server を起動し、データベースを オープンしておかないと、ユーザーは Oracle データベースを利用できません。この操作は、データベース管理者が実行します。データベースを起動してシステム全体で使用可能にするには、次の 3 つのステップを実行します。

1. Oracle Server のインスタンスを起動します。
2. データベースをマウントします。
3. データベースをオープンします。

これらのステップの説明は、5-5 ページの「[インスタンスとデータベースの起動](#)」を参照してください。

Oracle Server が起動すると、初期化パラメータを含むパラメータ・ファイルが使用されます。これらのパラメータは、データベース名、割り当てるメモリー量、制御ファイルの名前および各種の制限や他のシステム・パラメータを指定します。パラメータ・ファイルの例は、5-3 ページの「[パラメータ・ファイル](#)」を参照してください。

追加情報： 初期化パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

インスタンス、およびインスタンスが接続されているデータベースを停止するには、次の 3 つのステップを実行します。

1. データベースをクローズします。
2. データベースをディスマウントします。
3. Oracle Server のインスタンスを停止します。

インスタンスを停止すると、この3つのステップがすべて自動的に実行されます。詳細は、5-9 ページの「[データベースとインスタンスの停止](#)」を参照してください。

データベース・セキュリティ

Oracle のようなマルチユーザー・データベース・システムには、データベースへのアクセスおよび使用方法を制御するためのセキュリティ機能が用意されています。たとえば、セキュリティ・メカニズムは次のように機能します。

- 権限のないデータベース・アクセスを防止する。
- 権限のないスキーマ・オブジェクトへのアクセスを防止する。
- ディスクの使用を制御する。
- システム・リソース（CPU 時間など）の使用を制御する。
- ユーザーのアクションを監査する。

「スキーマ」は、各データベース・ユーザーに、そのユーザーと同じ名前によって対応付けられます。スキーマは、データベース・オブジェクト（表、ビュー、順序、シノニム、索引、クラスタ、プロシージャ、ファンクション、パッケージおよびデータベース・リンク）の論理的な集合です。デフォルトでは、各データベース・ユーザーは、対応するスキーマ内のすべてのオブジェクトを作成およびアクセスする権利を持っています。

データベース・セキュリティは、システム・セキュリティとデータ・セキュリティという2つのカテゴリに分かれています。

「システム・セキュリティ」には、システム・レベルにおけるデータベースのアクセスと使用を制御するメカニズムが含まれています。たとえば、次のものが含まれます。

- 有効なユーザー名とパスワードの組合せ
- ユーザーのスキーマ・オブジェクトに対して使用可能なディスク領域の容量
- ユーザーに対するリソース制限

システム・セキュリティ・メカニズムでは、ユーザーがデータベースへの接続を認可されているかどうか、データベース監査がアクティブになっているかどうかと、ユーザーが実行できるシステム操作がチェックされます。

「データ・セキュリティ」には、スキーマ・オブジェクト・レベルにおけるデータベースのアクセスと使用を制御するメカニズムが含まれています。たとえば、データベース・セキュリティには次のものが含まれます。

- 特定のスキーマ・オブジェクトにアクセスできるユーザーと、そのスキーマ・オブジェクトに対して各ユーザーが許可されている特定のタイプのアクション（たとえば、ユーザー SCOTT は EMP 表に対して SELECT 文と INSERT 文を発行できるが、DELETE 文は発行できない）
- 各スキーマ・オブジェクトに対して監査されるアクション。

セキュリティのメカニズム

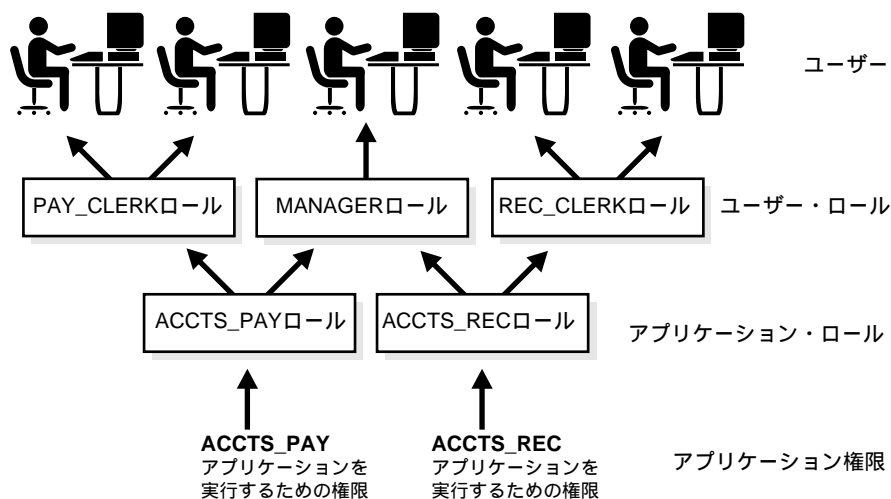
Oracle Server では、「任意アクセス制御」が可能です。任意アクセス制御とは、権限に基づいて情報へのアクセスを制限する方法です。ユーザーがスキーマ・オブジェクトにアクセスするには、そのユーザーは適切な権限を割り当てられている必要があります。適切な権限を付与されているユーザーは、他のユーザーに自分で任意に権限を付与することができます。このため、このタイプのセキュリティを「任意」セキュリティと呼びます。

Oracle は、次のようにいくつかの異なる機能を使用してデータベース・セキュリティを管理します。

- データベース・ユーザーとスキーマ
- 権限
- ロール
- 記憶領域の設定と割当て制限
- リソース制限
- 監査

[図 1-5](#) に、Oracle の各種セキュリティ機能の関連を示します。この後の項では、ユーザー、権限およびロールの概要について説明します。

図 1-5 Oracle のセキュリティ機能



データベース・ユーザーとスキーマ

それぞれの Oracle データベースは、ユーザー名のリストを持っています。データベースにアクセスするには、ユーザーはデータベース・アプリケーションを使用してデータベースの有効なユーザー名で接続する必要があります。無許可での使用を防止するために、各ユーザー名にはパスワードが対応付けられます。

セキュリティ・ドメイン 各ユーザーは「セキュリティ・ドメイン」、つまり、次のようなことを決定する一連のプロパティを持っています。

- ユーザーが利用できるアクション（権限とロール）
- ユーザーに対する表領域の割当て制限（利用可能なディスク領域）
- ユーザーのためのシステム・リソース制限（CPU 処理時間など）

ユーザーのセキュリティ・ドメインに寄与する各プロパティについては、この後の項で説明します。

権限

「権限」は、特定のタイプの SQL 文を実行するための権利です。たとえば、次のような権利を権限と呼びます。

- データベースに接続する権利（セッションを作成する権利）
- スキーマ内に表を作成する権利
- 他のユーザーの表から行を選択する権利
- 他のユーザーのストアド・プロシージャを実行する権利

Oracle データベースの権限は、システム権限とスキーマ・オブジェクト権限の 2 つに分類できます。

システム権限「システム権限」によって、ユーザーは特定のシステム全体にわたるアクションを実行したり、特定タイプのスキーマ・オブジェクトに対して特定のアクションを実行することができます。たとえば、表領域を作成する権限やデータベース内の任意の表の行を削除する権限はシステム権限です。多くのシステム権限は非常に強力であるため、それを使用できるのは管理者とアプリケーション開発者だけです。

スキーマ・オブジェクト権限「スキーマ・オブジェクト権限」によって、ユーザーは特定のスキーマ・オブジェクトに対して特定のアクションを実行できます。たとえば、特定の表の行を削除する権限は、オブジェクト権限です。エンドユーザーは、オブジェクト権限を付与される（割り当てられる）と、データベース・アプリケーションを使用して特定のタスクを完了できます。

権限の付与 ユーザーは、権限が付与されると、データベース内のデータにアクセスしたり修正できます。ユーザーは、次のような 2 通りの形態で権限を受け取ることができます。

- 権限は明示的にユーザーに対して付与できる。たとえば、EMP 表にレコードを挿入するための権限を、ユーザー SCOTT に明示的に付与できます。
- 権限は「ロール」（名前付き権限グループ）に対して付与できる。さらに、そのロールを 1 人以上のユーザーに対して付与できる。たとえば、EMP 表にレコードを挿入するための権限を、CLERK という名前のロールに付与できます。さらに、このロールをユーザー SCOTT や BRIAN に付与できます。

ロールによって、より簡単でより優れた権限管理が実現できるため、通常、権限は特定のユーザーではなくロールに対して付与します。次の項では、ロールとその使用について詳しく説明します。

ロール

Oracle では、ロールを使用することにより、制御された容易な権限管理を実現できます。「ロール」は、関連する権限のグループに名前を付けたもので、ユーザーまたは他のロールに付与されます。

ロールの次のような特徴によって、より簡便な権限管理を実現できます。

権限の付与件数の低減	データベース管理者は、同じ権限セットを多数のユーザーに明示的に付与するのではなく、関連するユーザーのグループに関する権限を 1 つのロールに付与できます。グループの各メンバーにそのロールを付与できます。
動的な権限管理	あるグループの権限を変更する必要がある場合は、そのロールの権限を修正するのみです。グループのロールを付与した全ユーザーのセキュリティ・ドメインは、そのロールに対して加えられる変更を自動的に反映します。
権限の選択的な可用性	ユーザーに付与したロールを、選択的に使用可能または使用禁止にすることができます。この機能によって、どのような状況でもユーザー権限を個々に制御できます。
アプリケーションによる認識	ユーザーがアプリケーションを使用するときに、選択したロールが自動的に使用可能または使用禁止にされるように、データベース・アプリケーションを設計できます。

データベース・アプリケーションについてのロールは、通常、データベース管理者が作成します。DBA は、アプリケーションの実行に必要なすべての権限をアプリケーション・ロールに付与します。その後、DBA は、アプリケーション・ロールを別のロールや特定のユーザーに付与できます。アプリケーションは複数の異なるロールを持つことができます。各ロールには、アプリケーションの使用時におけるデータ・アクセスの量を考慮して、異なる権限の集合を付与します。

DBA はパスワード付きのロールを作成し、そのロールに付与された権限が無許可で使用されるのを防止できます。通常、アプリケーションは起動時に適切なロールが使用可能になるように設計されます。そのため、アプリケーション・ユーザーは、アプリケーション・ロールのパスワードを知る必要がありません。

記憶領域の設定と割当て制限

Oracle は、データベースに割り当てられる各ユーザーのディスク領域の使用を管理また制御できます。これには、デフォルト表領域、一時表領域および表領域の割当てなどがあります。

デフォルト表領域 各ユーザーには、「デフォルト表領域」が対応付けられます。ユーザーがスキーマ・オブジェクトを作成する権限と、指定されたデフォルト表領域に対する割当て制限を持っている場合、表、索引またはクラスタの作成時にそのスキーマ・オブジェクトを物理的に格納する表領域を指定しなければ、そのユーザーのデフォルト表領域が使用されます。デフォルト表領域の機能によって、スキーマ・オブジェクトの位置が指定されていない状況で、領域使用を指示する情報が Oracle に与えられます。

一時表領域 各ユーザーは、「一時表領域」を持っています。ユーザーが一時セグメントの作成を必要とする SQL 文（索引の作成など）を実行するときには、そのユーザーの一時表領域が使用されます。すべてのユーザーの一時セグメントを別の表領域に割り当てることによって、一時表領域機能は、一時セグメントと他のタイプのセグメントとの間の I/O 競合を低減します。

表領域割当て制限 Oracle は、スキーマ内のオブジェクトに使用可能なディスク領域の容量を制限できます。「割当て制限」（領域制限）は、ユーザーが使用可能な表領域ごとに設定できます。表領域割当て制限によるセキュリティ機能によって、特定のスキーマのオブジェクトが消費するディスク領域の容量を選択的に制御できます。

プロファイルとリソース制限

各ユーザーには、そのユーザーが使用可能な複数のシステム・リソースに関する制限を指定した「プロファイル」が割り当てられます。プロファイルには次のような制限が指定されます。

- ユーザーが確立できる同時実行セッションの数
- CPU 処理時間
 - ユーザーのセッションで使用可能な時間
 - SQL 文による Oracle への 1 回のコールで使用可能な時間
- 論理 I/O の総量
 - ユーザーのセッションで使用可能な数
 - SQL 文による Oracle への 1 回のコールで使用可能な数
- ユーザーのセッションで許されるアイドル時間の総量
- ユーザーのセッションに許される接続時間の総量
- パスワード制限
 - 複数のログインが失敗したときにアカウントをロックする機能
 - パスワードの期限切れと猶予期間
 - パスワードの再使用と複雑さに関する制限

データベースの各ユーザーに対して、個別に異なるプロファイルを作成し、割り当てることができます。明示的にプロファイルを割り当てられていないすべてのユーザーには、デフォルト・プロファイルが提供されます。リソース制限機能は、グローバルなデータベース・システム・リソースの過剰消費を防止します。

監査

Oracle では、疑わしいデータベース使用の調査を支援するために、ユーザー・アクションを選択的に「監査」することができます（記録式の監視）。監査は、文監査、権限監査およびスキーマ・オブジェクト監査の3つのレベルで行うことができます。

文監査

文監査は、特定のスキーマ・オブジェクトとは無関係な、特定の SQL 文の監査です。（また、データベース管理者は、データベース・トリガーを使用して、Oracle に組み込まれている監査機能を拡張し、カスタマイズできます。）

文監査では、システムの全ユーザーの広範な監査、またはシステムの特定ユーザーの集中的な監査ができます。たとえば、ユーザーごとの文監査では、ユーザー SCOTT と LORI によるデータベースへの接続と切断を監査できます。

権限監査

権限監査は、特定のスキーマ・オブジェクトとは無関係な、強力なシステム権限の使用の監査です。権限監査では、全ユーザーの広範な監査、または特定ユーザーの集中的な監査ができます。

スキーマ・オブジェクト監査

スキーマ・オブジェクト監査は、ユーザーとは無関係な、特定のスキーマ・オブジェクトへのアクセスの監査です。オブジェクト監査では、特定の表に発行される SELECT 文や DELETE 文など、オブジェクト権限によって許可される文を監視します。

Oracle では、すべてのタイプの監査について、正常に終了した文の実行、失敗した文の実行またはその両方という選択的な監査ができます。これにより、文を発行しているユーザーがその文を発行するための適切な権限を持っているかどうかにかかわらず、疑わしい文を監査できます。

監査の結果は、「監査証跡」と呼ばれる表に記録されます。監査レコードを簡単に検索できるように、事前に定義された監査証跡のビューを使用できます。

データベースのバックアップとリカバリ

この項では、次のことを実現するために Oracle で使用されている構造とソフトウェアのメカニズムについて説明します。

- さまざまなタイプの障害が発生した場合に必要なデータベースの回復（リカバリ）
- どのような状況にも対応する柔軟な回復操作
- バックアップとリカバリの処理中のデータの可用性（システムのユーザーが作業を継続できるようにする）

回復が重要な理由

どのようなデータベース・システムでも、常にシステムやハードウェアの障害が発生する可能性があります。障害が発生し、データベースが影響を受けた場合は、データベースを回復する必要があります。障害発生後の目標は、コミットされたすべてのトランザクションの影響を、回復するデータベースに確実に反映させ、その障害によって生じた問題からユーザーを隔離しながら、できるだけ迅速に通常の運用状態に復帰させることです。

障害のタイプ

状況によっては、Oracle データベースの操作が中断されます。最も一般的な障害のタイプについて、次に説明します。

- | | |
|------------|---|
| ユーザー・エラー | ユーザー・エラーが発生した場合は、発生前の状態までデータベースを回復する必要があります。たとえば、ユーザーが誤って表を削除した場合を考えます。ユーザー・エラーからの回復を可能にし、その他の固有の回復要件を満たすために、Oracle は厳密な時間管理による回復機能を用意しています。たとえば、ユーザーが誤って表を削除した場合、データベースは表が削除された直前の状態に回復できます。 |
| 文障害とプロセス障害 | Oracle プログラム内の文の処理中に論理的な障害（たとえば、文が有効な SQL 構造になっていない場合）があると、文障害が発生します。文障害が発生すると、文による影響（それがあある場合）は Oracle によって自動的に取り消され、ユーザーに制御が戻されます。 |

「プロセス障害」とは、接続の異常切断やプロセスの異常終了など、Oracle にアクセスしているユーザー・プロセスでの障害です。障害を起こしたユーザー・プロセスは操作を継続できませんが、Oracle と他のユーザー・プロセスは継続できます。Oracle のバックグラウンド・プロセスである PMON は、障害ユーザー・プロセスを自動的に検出するか、または SQL*Net から通知を受け取ります。PMON は、ユーザー・プロセスのコミットされていないトランザクションをロールバックし、このプロセスが使用していたリソースを解放することによって問題を解決します。

SQL 文構造の誤りやユーザー・プロセスの中断など、一般的な問題が原因となっている場合、データベース・システム全体が中断されることはありません。さらに、Oracle は、システムや他のユーザーに与える影響を最小限に抑えながら、コミットされていないトランザクションの変更とロックされているリソースに対して、必要な回復処理を自動的に実行します。

インスタンス障害

インスタンス（システム・グローバル領域とバックグラウンド・プロセス）で作業を継続できないような問題が発生すると、インスタンス障害が発生します。インスタンス障害は、停電などのハードウェア問題、またはオペレーティング・システムのクラッシュなどのソフトウェア問題が原因で発生することがあります。インスタンス障害が発生した場合、システム・グローバル領域のバッファ内のデータは、データ・ファイルに書き込まれません。

インスタンス障害が発生した場合は、「クラッシュ回復」または「インスタンス回復」が必要になります。クラッシュ回復は、インスタンスの再起動時に Oracle によって自動的に実行されます。Oracle Parallel Server では、別のインスタンスの SMON プロセスによって、インスタンス回復が実行されます。インスタンス障害のために失われた、SGA のデータベース・バッファ内のコミット済みデータを回復するために、REDO ログが使用されます。

メディア（ディスク）障害

データベースを運用する上で必要なファイルに対して読み書きを実行しているときに、エラーが発生することがあります。ディスク上の物理ファイルの読み書きにかかわる物理的な問題が原因となるため、このような障害をディスク障害と呼びます。一般的な例は、ディスク・ヘッドのクラッシュです。この場合、ディスク・ドライブ上のファイルはすべて失われます。

データ・ファイル、REDO ログ・ファイルおよび制御ファイルなど、さまざまなファイルが、このタイプのディスク障害によって影響を受ける可能性があります。また、データベース・インスタンスは正常に機能し続けることができないため、システム・グローバル領域のデータベース・バッファ内のデータをデータ・ファイルに書き込むことができません。

ディスク障害では「メディア回復」が必要です。メディア回復は、障害のために失われたメモリー内のコミット済みデータなど、データベースのデータ・ファイル内の情報がディスク障害の起こる直前の状態になるようにデータ・ファイルを復元します。ディスク障害からの回復を完了するには、データベースのデータ・ファイルのバックアップ、すべてのオンライン REDO ログ・ファイルおよび必要なアーカイブ済み REDO ログ・ファイルを使用する必要があります。

Oracle では、ディスク・クラッシュなど、発生する可能性のあるすべてのタイプのハードウェア障害から、完全に迅速な回復を行うことができます。データベースを完全に回復するか、または部分的に特定の時点まで回復できるように、オプションが用意されています。

ディスク障害のためにいくつかのデータ・ファイルが破損しても、データベースの大部分は損なわれておらず、そのまま運用できるという場合に限り、必要な表領域を個別に回復する間、データベースをオープンしたままにしておくことができます。したがって、データベースのうち損害を受けていない部分は、破損した部分の回復中も日常業務に使用できます。

回復に使用される構造

Oracle は、いくつかの構造を使用して、インスタンス障害やディスク障害から完全に回復します。つまり、REDO ログ、ロールバック・セグメント、制御ファイルおよびデータベースのバックアップを使用します。

REDO ログ

1-12 ページの「**REDO ログ・ファイル**」で説明されているとおり、「REDO ログ」とはデータ・ファイルに書き込まれていない、メモリー内の変更済みデータベース・データを保護するファイルの集合です。REDO ログは、オンライン REDO ログとアーカイブ済み REDO ログの 2 つの部分から構成されます。

オンライン REDO ログ「オンライン REDO ログ」は、2 つ以上の「オンライン REDO ログ・ファイル」の集合であり、データベースに対して行われたすべての変更が、コミット済みかどうかを問わず記録されます。REDO エントリは、システム・グローバル領域の REDO ログ・バッファに一時的に格納され、バックグラウンド・プロセス LGWR はこの REDO ログ・エントリをオンライン REDO ログ・ファイルに順次書き込みます。LGWR は REDO エ

ントリを絶えず書き込みます。つまり、ユーザー・プロセスがトランザクションをコミットするたびに、コミット・レコードを書き込みます。

オンライン REDO ログ・ファイルは循環方式で使用されます。たとえば、オンライン REDO ログが 2 つのファイルで構成されている場合には、最初のファイルが満杯になり、2 番目のファイルも満杯になったら最初のファイルを再使用し、そのファイルが満杯になったら 2 番目のファイルを再使用するというようになります。ファイルが満杯になるたびに、REDO エントリの集合を識別するために「ログ順序番号」がそのファイルに割り当てられます。

1 箇所に障害が起きてもデータベースが失われないように、Oracle では複数のオンライン REDO ログ・ファイルのセットを維持できます。「多重化されたオンライン REDO ログ」は、物理的に別々のディスク上にあるオンライン REDO ログ・ファイルの複数のコピーから構成され、そのグループ内の各メンバーに加えられた変更は、すべてのメンバーに反映されます。

オンライン REDO ログ・ファイルが格納されているディスクで障害が生じて、他のコピーが破損することはなく、Oracle から使用できます。システム操作を中断せずに、破損していないコピーを使用して、失われたオンライン REDO ログ・ファイルを容易に回復できます。

アーカイブ済み REDO ログ 満杯になったオンライン REDO ログ・ファイルは、「アーカイブ済み REDO ログ」を作成して再使用の前にアーカイブすることもできます。アーカイブ済み REDO ログは、「アーカイブ済み（オフライン）REDO ログ・ファイル」によって構成されます。

アーカイブ済み REDO ログの有無は、REDO ログが使用しているモードで判断します。

ARCHIVELOG	満杯になったオンライン REDO ログ・ファイルは、循環方式で再使用される前にアーカイブされます。
------------	---

NOARCHIVELOG	満杯になったオンライン REDO ログ・ファイルはアーカイブされません。
--------------	--------------------------------------

ARCHIVELOG モードでは、インスタンス障害とディスク障害の両方から、データベースを完全に回復できます。また、データベースがオープンされており、使用可能な状態になっているときにも、データベースのバックアップを作成できます。ただし、アーカイブ済み REDO ログのメンテナンスのために、余分な管理作業が必要になります。

データベースの REDO ログを NOARCHIVELOG モードで運用している場合、データベースをインスタンス障害から完全に回復することはできませんが、ディスク障害からは回復できません。また、データベースが完全にクローズされている間のみ、データベースのバックアップを作成できます。アーカイブ済み REDO ログが作成されないため、データベース管理者による余分な作業は必要ありません。

LogMiner (SQL ベースのログ・アナライザ) LogMiner は、オンラインおよびアーカイブ済みログ・ファイルを、SQL を使用して読み込み、分析および解釈するためのリレーショナル・ツールです。LogMiner によるログ・ファイルの分析機能は、次の操作に使用できます。

- トランザクション、ユーザー、表、時刻などに基づいて、特定の変更の集合を追跡する。データベース・オブジェクトを変更したユーザーと、変更前後のオブジェクト・データの内容を判別できます。データベース変更をそのソースまでさかのぼって追跡し、監査して取り消すことによって、データを保護し、制御できます。
- データベースに不適切な変更が加えられた日時を特定する。これにより、データベース・レベルではなく、アプリケーション・レベルで論理回復できます。
- チューニングと容量計画のための補足情報を提供する。履歴分析を実行して、傾向やデータ・アクセス・パターンを判断することもできます。
- 複雑なアプリケーションをデバッグするうえで不可欠な情報を取り出す。

注意： LogMiner で読み込んで分析できるのは、Oracle8 またはそれ以降のログ・ファイルのみです。

追加情報： LogMiner の詳細は、『Oracle8i 管理者ガイド』を参照してください。

制御ファイル

データベースの「制御ファイル」には、主としてデータベースのファイル構造と、LGWR によって書き込まれている現行のログ順序番号についての情報が保管されます。通常の回復手順では、回復操作を自動的に進行させるために制御ファイル内の情報を使用します。

制御ファイルの多重化 Oracle は、多数の同じ制御ファイルをすべて同時に更新して、メンテナンスできます。

ロールバック・セグメント

1-10 ページの「[データ・ブロック、エクステンツおよびセグメント](#)」で説明されているように、ロールバック・セグメントには、Oracle のいくつかの機能で使用されるロールバック情報が記録されます。データベースの回復では、REDO ログに記録された変更がすべて適用された後、Oracle はロールバック・セグメントの情報を使用して、コミットされていないトランザクションを取り消します。ロールバック・セグメントはデータベース・バッファ内に格納されるため、この重要な回復情報は REDO ログによって自動的に保護されます。

データベースのバックアップ

ディスク障害では、1 つ以上のファイルが物理的に破損している可能性があるため、メディア回復では、データベースの最新のオペレーティング・システム・バックアップから、破損したファイルを復元する必要があります。データベースのファイルのバックアップを作成する方法は次の 2 つに分類されます。

全体データベース・バックアップ 全体データベース・バックアップは、Oracle データベースを構成するすべてのデータ・ファイル、オンライン REDO ログ・ファイルおよび制御ファイルの、オペレーティング・システム・バックアップです。データベースをクローズして使用できない状態にしているときに、全体バックアップを実行します。

部分バックアップ 部分バックアップは、データベースの一部のオペレーティング・システム・バックアップです。部分バックアップには、個々の表領域のデータ・ファイルのバックアップや、制御ファイルのバックアップなどがあります。データベースの REDO ログが ARCHIVELOG モードで運用されている場合に限り、部分バックアップが有効です。

バックアップ計画に応じて、さまざまな部分バックアップを作成できます。データベースがオープンされているかどうか、特定の表領域がオンラインになっているかどうかなど、さまざまな場合にデータ・ファイルと制御ファイルのバックアップを作成できます。REDO ログは ARCHIVELOG モードで運用されているため、REDO ログの追加バックアップは必要ありません。つまり、アーカイブ済み REDO ログは、満杯になったオンライン REDO ログ・ファイルのバックアップです。

基本的な回復手順

DBWn がデータ・ファイルにデータベース・バッファを書き込む方法によっては、ある時点では、コミットされていないトランザクションで仮に変更されたデータ・ブロックのいくつか、データ・ファイルに含まれている可能性があります。また、コミットされたトランザクションで変更されたいくつかのブロックが、データ・ファイルに含まれていない可能性があります。したがって、障害の発生後には次のような 2 つの状態が考えられます。

- コミット済みの変更を含んでいるデータ・ブロックが、データ・ファイルに書き込まれておらず、その変更は REDO ログのみに存在する。したがって、REDO ログには、データ・ファイルに適用が必要なコミット済みデータが含まれています。
- REDO ログには、コミットされていないデータが含まれている可能性があるため、REDO ログが回復中に適用した変更のうちコミットされていないトランザクションの変更は、データ・ファイルから消去しなければならない。

このような状態を解決するために、インスタンス障害やメディア障害からの回復では、ロールフォワードおよびロールバックという 2 つのステップが必ず使用されます。

ロールフォワード

回復の最初のステップは「ロールフォワード」です。これによって、REDO ログに記録されたすべての変更がデータ・ファイルに再適用されます。ロールフォワードは、必要なすべての REDO ログ・ファイルを使用して、必要な時点までデータ・ファイルの内容を戻します。

必要な REDO 情報がすべてオンラインになっている場合、Oracle はデータベースの起動時に自動的にロールフォワードします。ロールフォワードが行われた後、データ・ファイルには、すべてのコミット済みの変更と REDO ログに記録されたコミットされていない変更が含まれています。

ロールバック

ロールフォワードは、回復処理の半分にすぎません。ロールフォワードの実行後、コミットされていない変更をすべて取り消す必要があります。REDO ログ・ファイルが適用された後、コミットされなかったのに REDO ログに記録されていたトランザクションを識別して取り消すために、ロールバック・セグメントを使用します。このプロセスを「ロールバック」といいます。Oracle はこのステップを自動的に完了します。

Recovery Manager

Recovery Manager は、バックアップおよびリカバリの操作を管理する Oracle ユーティリティです。データベース・ファイル（データ・ファイル、制御ファイルおよびアーカイブ済み REDO ログ・ファイル）のバックアップを作成したり、バックアップを使用してデータベースの復元（リストア）や回復（リカバリ）を行います。

Recovery Manager は、「リカバリ・カタログ」というリポジトリを維持しています。リカバリ・カタログには、バックアップ・ファイルとアーカイブ済みログ・ファイルに関する情報が含まれています。Recovery Manager は、このリカバリ・カタログを使用して、復元操作とメディア回復の両方を自動化します。

リカバリ・カタログには、次の情報が含まれています。

- データ・ファイルとアーカイブ・ログのバックアップに関する情報
- データ・ファイル・コピーに関する情報
- アーカイブ済み REDO ログとそれらのログのコピーに関する情報
- ターゲット・データベースの物理スキーマに関する情報
- 「ストアド・スクリプト」と呼ばれる名前付きのコマンド・シーケンス

追加情報： Recovery Manager の詳細は、『Oracle8i 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。

データ・アクセス

この項では、次のような DBMS の一般要件に Oracle がどのように応えているかを説明します。

- データ・アクセス言語として業界で受け入れられている規格を遵守する。
- データの操作中にデータベース情報の一貫性を制御し、維持する。
- データベース情報の整合性を維持する規則を定義、施行するためのシステムを提供する。
- 高パフォーマンスを実現する。

SQL— 構造化問合せ言語

「SQL」は、簡単かつ強力なデータベース・アクセス言語であり、RDBMS（リレーショナル・データベース管理システム）の標準言語です。オラクル社が Oracle 向けにインプリメントした SQL は、ANSI/ISO 標準規格の SQL データ言語に 100 パーセント準拠しています。

SQL 文

Oracle データベース内の情報の操作は、すべて SQL 文を使用して実行されます。SQL 文とは、Oracle に与えられ、Oracle によって実行される SQL テキスト文字列です。次に示すように、文は、完全な「SQL センテンス」と等価でなければなりません。

```
SELECT ename, deptno FROM emp;
```

実行できるのは完全な SQL 文のみです。次のような「不完全文」を実行しようとすると、テキストが不足しているため SQL 文を実行できないことを示すエラーが発生します。

```
SELECT ename
```

SQL 文は非常に簡単な文ですが、強力なコンピュータ・プログラム、または命令と考えることができます。SQL 文は、次のカテゴリに分類されます。

- 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
- 埋込み SQL 文

データ定義文（DDL）「DDL 文」は、スキーマ・オブジェクトを定義およびメンテナンスしたり、不要になったスキーマ・オブジェクトを削除します。DDL 文には、データベースやデータベース内の特定のオブジェクトにアクセスする「権限」、つまり権利を、あるユーザーから他のユーザーに付与するための文も含まれます。（1-37 ページの「[データベース・セキュリティ](#)」を参照。）

データ操作文（DML）「DML 文」は、データベースのデータを操作します。たとえば、表の行の問合せ、挿入、更新および削除はすべて DML 操作です。また、表やビューのロック、SQL 文の実行計画の検査も DML 操作です。

トランザクション制御文「トランザクション制御」文は、DML 文によって行われる変更を管理します。これによって、ユーザーやアプリケーション開発者はいくつかの変更をグループ化して、論理トランザクションを作成できます。（1-52 ページの「[トランザクション](#)」を参照。）この文に含まれるのは、COMMIT、ROLLBACK および SAVEPOINT などです。

セッション制御文「セッション制御」文によって、ユーザーは自分の現行セッションのプロパティを制御できます。つまり、ロールを使用可能や使用禁止にしたり、言語設定を変更できます。セッション制御文には、ALTER SESSION と SET ROLE の 2 つがあります。

システム制御文 システム制御コマンドは、Oracle Server インスタンスのプロパティを変更します。システム制御コマンドは ALTER SYSTEM だけです。このコマンドは、共有サーバーの最小数などの設定の変更、セッションの停止およびその他の作業を実行します。

埋込み SQL 文 埋込み SQL 文は、DDL 文、DML 文、およびトランザクション制御文をプロシージャ型言語プログラム（Oracle プリコンパイラとともに使用されるプログラムなど）に組み込んだものです。たとえば、OPEN、CLOSE、FETCH、EXECUTE などがあります。

トランザクション

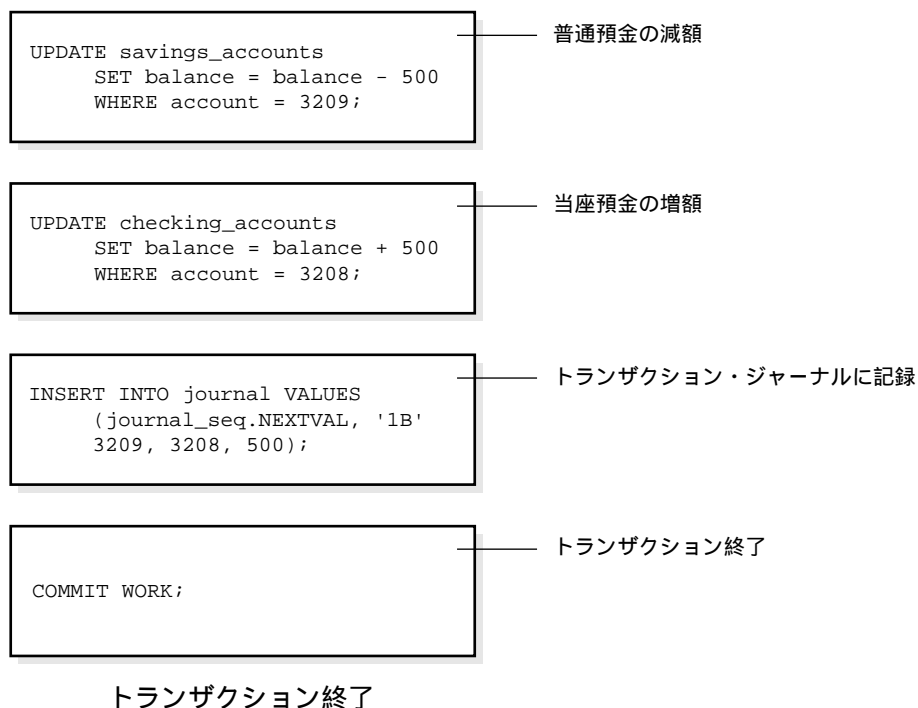
「トランザクション」は、単一のユーザーによって実行される 1 つ以上の SQL 文を含む論理作業単位です。Oracle が互換性を持つ ANSI/ISO SQL 規格によれば、トランザクションはユーザーの最初の実行可能 SQL 文で始まります。トランザクションは、ユーザーによって明示的にコミットまたはロールバックされるときに終了します（コミットとロールバックについては後述します）。

銀行のデータベースを考えてみます。顧客が普通預金口座から当座預金口座へ預金を移動するとき、トランザクションは、3 つの別々の操作、つまり普通預金口座の減額、当座預金口座の増額およびトランザクション・ジャーナルへのトランザクション記録を実行します。

Oracle は、3 つの SQL 文をすべて実行することによって口座の差引勘定が正しく維持されることを保証する必要があります。なんらかの問題（ハードウェア障害など）が発生してトランザクション内の文がどれか 1 つでも実行されなかった場合、トランザクションの他の文の実行を取り消さなければなりません。これを「ロールバック」と呼びます。2 つの更新のうちどちらかの実行中にエラーが発生した場合、どちらの更新も行われません。

図 1-6 に銀行のトランザクションの例を示します。

図 1-6 銀行のトランザクション



トランザクションのコミットとロールバック

トランザクションを構成する SQL 文によって加えられた変更は、コミットまたはロールバックできます。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場合、次のトランザクションは次の SQL 文から開始されます。

トランザクションを「コミット」すると、トランザクション内のすべての SQL 文による変更が確定されます。トランザクションの SQL 文による変更が、他のユーザー・セッションのトランザクションから参照可能になるのは、そのトランザクションがコミットされた後に開始されたトランザクションについてのみです。

トランザクションを「ロールバック」すると、トランザクション内の SQL 文による変更がすべて取り消されます。トランザクションがロールバックされると、トランザクション内の SQL 文が実行されなかった場合と同様に、影響を受けたデータは変更されないまま残ります。

セーブポイント

多数の SQL 文を含む大規模なトランザクションでは、中間の目印、つまり「セーブポイント」を宣言できます。「セーブポイント」は、トランザクションをより小さな部分に分割するために使用します。

セーブポイントを使用して、大規模なトランザクション内の任意の位置で、作業を記録できます。これにより、トランザクションの現在の位置から、トランザクション内で宣言したセーブポイントまでの間に行われたすべての作業を、選択的にロールバックできます。たとえば、大規模で複雑な一連の更新のどこにでもセーブポイントを設定できるため、エラーが発生してもすべての文を再発行する必要はありません。

トランザクションを使用したデータの一貫性

トランザクション内の SQL 文が論理的にグループ化される限り、トランザクションを使用して、データベース・ユーザーやアプリケーション開発者はデータへの一貫した変更を保証できます。

トランザクションには、1 つの論理作業単位に必要な部分を過不足なくすべて含める必要があります。トランザクションの開始から終了まで、参照される表データはすべて一貫した状態に維持されます。各トランザクションには、データに対して首尾一貫した 1 つの変更を加える SQL 文のみを含めてください。

たとえば、銀行の例を思い出してください。2 つの口座間の送金（トランザクション）には、一方の口座の預金高を増やすアクション（1 つの SQL 文）、他方の口座の預金高を減らすアクション（1 つの SQL 文）、およびトランザクション・ジャーナルに記録するアクション（1 つの SQL 文）が必要です。これらのアクションは、すべて失敗するか、すべて成功するかのどちらかです。つまり、借方が存在しないのに貸方が成立することはあり得ません。一方の口座に新しく預金するなど、無関係なアクションは、この送金トランザクションで実行しないでください。そのような文は、別のトランザクションに組み込みます。

PL/SQL

「PL/SQL」は、SQL に対する Oracle のプロシージャ型言語拡張機能です。PL/SQL は、SQL の持つ簡易性と柔軟性を、IF..THEN、WHILE、LOOP などの構造化プログラミング言語のプロシージャ的な機能と一体化します。

データベース・アプリケーションを設計するとき、開発者はストアド PL/SQL を使用することによる次のような利点を検討してください。

- PL/SQL コードはデータベースに集中的に格納できるため、アプリケーションとデータベースとの間のネットワーク通信量が低減され、アプリケーションとシステムのパフォーマンスが向上する。
- データ・アクセスをストアド PL/SQL コードによって制御できる。この場合（別のアクセス・ルートが与えられない限り）PL/SQL のユーザーはアプリケーション開発者が意図したとおりによりデータにアクセスできます。

- アプリケーションからデータベースに PL/SQL ブロックを送信できるため、大量のネットワーク通信を行わずに複雑な操作を実行できる。

PL/SQL がデータベースに格納されていない場合でも、アプリケーションは個々に SQL 文をデータベースに送信するかわりに、PL/SQL のブロックを送信できます。そのため、ネットワーク通信量が減少します。

この後の項では、データベース内に定義し、一括して格納できるさまざまなプログラム・ユニットについて説明します。

プロシージャとファンクション

「プロシージャ」と「ファンクション」を構成する一連の SQL 文および PL/SQL 文は、特定の問題を解決したり一連の関連タスクを実行することを目的として、1 つの単位としてグループ化されています。プロシージャは、作成後にコンパイル済みの形式でデータベース内に格納され、ユーザーまたはデータベース・アプリケーションによって実行されます。

プロシージャはコール元に値を戻さないのに対して、ファンクションはコール元に必ず 1 つの値を戻します。それ以外の点は同じです。

パッケージ

「パッケージ」によって、関連したプロシージャ、ファンクション、変数およびその他のパッケージ構成体をカプセル化し、データベースの単位としてまとめて格納できます。パッケージによって、管理者やアプリケーション開発者がこのようなルーチンを編成できる一方で、高機能（グローバル・パッケージ変数を宣言し、パッケージ内のプロシージャで使用できるなど）と高パフォーマンス（パッケージの全オブジェクトを一度に解析、コンパイルし、メモリーにロードするなど）が実現します。

データベース・トリガー

Oracle では、表に対する挿入、更新または削除の結果として、自動的に実行されるプロシージャを作成できます。これらのプロシージャを「データベース・トリガー」と呼びます。

データベースの情報管理のために、トリガーをいろいろな方法で使用できます。たとえば、データ生成の自動化、データ修正の監査、複雑な整合性制約の施行、複雑なセキュリティ許可のカスタマイズに使用できます。

メソッド

「メソッド」とは、ユーザー定義データ型（オブジェクト型、ネストした表または変数配列）の一部であるプロシージャまたはファンクションのことです。

注意： ユーザー定義のデータ型を使用できるのは、Oracle8i Enterprise Edition を購入した場合のみです。

メソッドは、次の2つの点でストアド・プロシージャと異なります。

- メソッドは、対応付けられた型のオブジェクトを参照することによって起動する。
- メソッドは、対応付けられたオブジェクトが持つ属性と、その型についての情報に完全にアクセスできる。

すべてのユーザー定義データ型には、データ型の仕様に基づいて新しいオブジェクトを作成するための、システム定義の「コンストラクタ・メソッド」があります。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。オブジェクト型の場合、コンストラクタ・メソッドのパラメータの名前と型は、オブジェクト型の属性の名前と型と同じです。コンストラクタ・メソッドは、新しいオブジェクトを値として戻すファンクションです。ネストした表および配列にも、コンストラクタ・メソッドがあります。

「比較メソッド」は、特定のオブジェクト型に属するオブジェクト間の順序関係を定義します。「マップ・メソッド」では、ビルトイン型を比較する Oracle の機能を使用します。たとえば、「四角形」という名のオブジェクト型に「高さ」と「幅」の属性があり、その四角形の「高さ」属性と「幅」属性の積を示す数値を戻すマップ・メソッド「面積」が定義されていれば、Oracle は2つの四角形の面積を比較することによって、その2つの四角形を比較できます。「順序メソッド」は、独自の内部論理を使用して、特定のオブジェクト型に属する2つのオブジェクトを比較します。次に、順序関係を符号化した値を戻します。たとえば、最初のオブジェクトのほうが小さければ -1 を戻し、どちらも同じ大きさであれば 0 を戻し、最初のオブジェクトのほうが大きければ 1 を戻します。

データの整合性

データがデータベース管理者やアプリケーション開発者によって決められたとおり、ある業務規則を遵守することを保証するのは非常に重要です。たとえば、業務規則によって、INVENTORY 表の SALE_DISCOUNT 列には 9 よりも大きな数値を入れないように指定されている場合を考えます。INSERT 文や UPDATE 文がこの整合性ルールに違反しようとする、Oracle はその無効な文をロールバックし、アプリケーションにエラーを戻します。Oracle は、データベースのデータ整合性ルールを管理するための解決策として、整合性制約とトリガーを提供します。

整合性制約

「整合性制約」は、表の列に対して業務規則を定義する宣言の方法です。整合性制約は表のデータに関する文であり、常に次のように機能します。

- 整合性制約を表に対して作成した場合に、いくつかの既存の表データがその制約を満たしていないと、その制約は施行できない。
- 制約が定義された後、DML 文の結果が整合性制約に違反した場合、その文はロールバックされ、エラーが戻される。

整合性制約は表に定義され、表の定義の一部として集中的にデータベースのデータ・ディクショナリに格納されるため、すべてのデータベース・アプリケーションは同じ一連の規則を遵守しなければなりません。規則が変更されても、整合性制約はデータベース・レベルで一度変更すればよいので、アプリケーションごとに何度も変更する必要はありません。

Oracle がサポートする整合性制約は次のとおりです。

NOT NULL	表の列に NULL（空の入力）を許可しない。
UNIQUE	列（または列の集合）に重複値を許可しない。
PRIMARY KEY	列（または列の集合）に重複値と NULL を許可しない。
FOREIGN KEY	列（または列の集合）の値が、関連する表の UNIQUE または PRIMARY KEY の値と一致している必要がある。 （また、FOREIGN KEY 整合性制約は、参照データが変更された場合に依存データを処理する方法を Oracle に指示する、参照整合性アクションも定義します。）
CHECK	制約の論理式を満たさない値を許可しない。

キー

「キー」という語は、いくつかのタイプの整合性制約の定義で使用されます。「キー」は、特定のタイプの整合性制約の定義に含まれる列または列の集合です。キーは、リレーショナル・データベースの別々の表と列の間の関連を記述するもので、次のようなタイプがあります。

主キー	表の PRIMARY KEY 制約の定義に含まれる列（または列の集合）です。主キーの値は、表内の各行を一意に識別します。主キーは、表あたり 1 つだけ定義できます。
一意キー	UNIQUE 制約の定義に含まれる列（または列の集合）です。
外部キー	参照整合性制約の定義に含まれる列（または列の集合）です。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。

キーの中の個々の値を、「キー値」と呼びます。

データベース・トリガー

集中的なアクションは、データベース・トリガーで PL/SQL コードを使用して定義できます（整合性制約のように宣言するわけではありません）。「データベース・トリガー」は、対応付けられた表に対して INSERT、UPDATE または DELETE 文が発行されたときに起動される（暗黙的に実行される）ストアード・プロシージャです。データベース・トリガーは、データベースの監査、および複雑なセキュリティ・チェックと整合性ルールの施行などの機能とともに使用して、データベース管理システムをカスタマイズできます。たとえば、通常の就業時間内に限って表を修正できるように、データベース・トリガーを作成できます。

注意： データベース・トリガーによって整合性ルールを定義または施行することはできますが、データベース・トリガーが整合性制約と同じ機能を持つわけではありません。特に、整合性ルールを施行するために定義されたデータベース・トリガーは、すでに表にロードされているデータをチェックしません。したがって、整合性制約によって整合性ルールを施行できない場合にのみ、データベース・トリガーを使用することを強くお勧めします。

第 II 部

データベースの構造

第 II 部では、Oracle データベースの基本的な構造上のアーキテクチャについて説明します。たとえば、物理的また論理的な記憶構造を取り上げます。第 II 部には、次の章が含まれています。

- [第 2 章「データ・ディクショナリ」](#)
- [第 3 章「表領域とデータ・ファイル」](#)
- [第 4 章「データ・ブロック、エクステントおよびセグメント」](#)

追加情報： その他の論理データベース構造については、次の各章を参照してください。

- [第 10 章「スキーマ・オブジェクト」](#)
- [第 11 章「パーティション表とパーティション索引」](#)

データ・ディクショナリ

LEXICOGRAPHER—A writer of dictionaries, a harmless drudge.

Samuel Johnson: *Dictionary*

この章では、「データ・ディクショナリ」と呼ばれる、各 Oracle データベースの読み込み専用の参照表とビューの中核的なセットについて説明します。この章の内容は、次のとおりです。

- データ・ディクショナリの基礎知識
- データ・ディクショナリの構造
- SYS、データ・ディクショナリの所有者
- データ・ディクショナリ使用方法
- 動的パフォーマンス表

データ・ディクショナリの基礎知識

「データ・ディクショナリ」は Oracle データベースで最も重要な部分の 1 つであり、対応付けられているデータベースに関する情報を提供する**読み込み専用**の表の集合です。データ・ディクショナリには次のものが含まれます。

- データベース内のすべてのスキーマ・オブジェクト（表、ビュー、索引、クラスタ、シノニム、順序、プロシージャ、ファンクション、パッケージ、トリガーなど）の定義
- スキーマ・オブジェクトに割り当てられている領域と、現在使用されている領域の容量
- 列のデフォルト値
- 整合性制約に関する情報
- Oracle ユーザーの名前
- それぞれのユーザーに付与されている権限とロール
- 監査情報。たとえば、各種スキーマ・オブジェクトにアクセスまたは更新したユーザーなど。
- その他の一般的なデータベース情報

データ・ディクショナリは、他のデータベース・データと同様に、表とビューによって構成されています。特定のデータベースのデータ・ディクショナリ表とビューは、すべてそのデータベースの SYSTEM 表領域に格納されます（3-7 ページの「**SYSTEM 表領域**」を参照してください）。

データ・ディクショナリは、あらゆる Oracle データベースにとって中核的な存在であるだけでなく、エンド・ユーザーからアプリケーション設計者やデータベース管理者まで、すべてのユーザーにとって重要なツールです。データ・ディクショナリにアクセスするには、SQL 文を使用します。データ・ディクショナリは読み込み専用なので、ユーザーはデータ・ディクショナリの表とビューに対して問合せ（SELECT 文）のみを発行できます。

データ・ディクショナリの構造

データベースのデータ・ディクショナリは、次の要素で構成されています。

実表

対応するデータベースについての情報を格納する基礎になる表。これらの表に読み書きできるのは Oracle だけです。これらの表は標準化され、データのほとんどは、暗号形式で格納されているので、ユーザーが直接アクセスすることはめったにありません。

ユーザー・アクセス可能ビュー	データ・ディクショナリの実表に格納されている情報を要約して表示するビュー。これらのビューは、実表にあるデータを、ユーザー名や表の名前などの実用的な情報にデコードし、結合と WHERE 句を使用して情報を簡略化します。ほとんどのユーザーには、実表ではなく、これらのビューへのアクセス権が与えられています。
----------------	---

SYS、データ・ディクショナリの所有者

データ・ディクショナリのすべての実表とユーザー・アクセス可能ビューは、Oracle ユーザー SYS が所有しています。したがって、Oracle ユーザーは、SYS スキーマに含まれている行またはスキーマ・オブジェクトを**決して**変更（更新、削除または挿入）しないでください。そのような操作により、データの整合性が損なわれることがあります。セキュリティ管理者は、このアカウントを厳しく管理する必要があります。

警告： 基礎となるデータ・ディクショナリ表のデータを変更したり操作したりすると、データベースの操作に永続的な悪影響を与えるおそれがあります。

データ・ディクショナリの使用法

データ・ディクショナリの主な使用法は次の 3 つです。

- Oracle は、ユーザーおよびスキーマ・オブジェクト、記憶構造に関する情報を検索するためにデータ・ディクショナリにアクセスする。
- Oracle は、データ定義言語（DDL）文が発行されるたびにデータ・ディクショナリを変更する。
- Oracle ユーザーは、データベースについての情報の読み専用リファレンスとしてデータ・ディクショナリを使用できる。

Oracle によるデータ・ディクショナリの使用法

データ・ディクショナリの実表内のデータは、**Oracle を機能させるために必要**です。したがって、データ・ディクショナリ情報を書き込んだり変更するのは、Oracle のみです。

データベースの操作時に、Oracle はデータ・ディクショナリを読み込んで、スキーマ・オブジェクトが存在しており、ユーザーにはアクセス権が正しく付与されていることを確認します。また Oracle は、データベース構造、監査、権限付与およびデータの変更を反映するように、継続的にデータ・ディクショナリを更新します。

たとえば、ユーザー KATHY が PARTS という表を作成すると、新しい表、列、セグメント、エクステントおよび KATHY がその表に対して持っている権限を反映するために、新しい行がデータ・ディクショナリに追加されます。この新しい情報は、次回ディクショナリ・ビューを問い合わせるときに表示されます。

データ・ディクショナリ・ビューのパブリック・シノニム

多くのデータ・ディクショナリ・ビューにユーザーが簡単にアクセスできるようにするため、Oracle はパブリック・シノニムを作成します。(セキュリティ管理者は、システム全体で使用するスキーマ・オブジェクトのパブリック・シノニムを作成して追加することもできます。) ユーザーは、パブリック・シノニムに使用されているのと同じ名前を、自分のスキーマ・オブジェクトに付けないようにする必要があります。

高速アクセスのためのデータ・ディクショナリのキャッシュ

ユーザー・アクセスの妥当性検査やスキーマ・オブジェクト状態の検証のために、Oracle はデータベース操作中に絶えずデータ・ディクショナリにアクセスするので、データ・ディクショナリ情報の大部分は SGA (ディクショナリ・キャッシュ) 内に格納されます。すべての情報は、LRU (最低使用頻度) アルゴリズムを使用してメモリーに格納されます。

通常、キャッシュに保持されるのは、解析処理に必要な情報です。表とそれらの列について記述している COMMENTS 列は、頻繁にアクセスされない限りキャッシュには保持されません。

他のプログラムとデータ・ディクショナリ

他の Oracle 製品は、既存のビューを参照したり、独自のデータ・ディクショナリ表またはビューを追加できます。データ・ディクショナリを参照するプログラムを記述するアプリケーション開発者は、基礎となる表ではなくパブリック・シノニムを参照する必要があります。これは、シノニムの方がソフトウェア・リリース間での変更が少ないからです。

新しいデータ・ディクショナリ項目の追加

新しい表またはビューをデータ・ディクショナリに追加できます。新しいオブジェクトをデータ・ディクショナリに追加する場合、その新しいオブジェクトの所有者はユーザー SYSTEM か、第 3 の Oracle ユーザーである必要があります。

注意： ユーザー SYS に属する新しいオブジェクトは絶対に作成しないでください。ただし、オラクル社が提供するスクリプトを実行して、データ・ディクショナリのオブジェクトを作成する場合は例外です。

データ・ディクショナリ項目の削除

データ・ディクショナリに対するすべての変更は、DDL 文への応答として Oracle が実行するので、**どのユーザーも、データ・ディクショナリの表にあるデータを絶対に削除または変更しないでください。**

この規則の唯一の例外は、表 SYS.AUD\$ です。監査が使用可能になっていると、この表は際限なく肥大化する可能性があります。AUDIT_TRAIL 表は削除すべきではありませんが、この表の行は情報を提供するのみのもので、Oracle の実行に必要なではないので、セキュリティ管理者はこの表のデータを削除できます。

ユーザーと DBA によるデータ・ディクショナリの使用法

データ・ディクショナリのビューは、すべてのデータベース・ユーザーのためのリファレンスとしての役目を果たします。データ・ディクショナリ・ビューには、SQL 言語を介してアクセスします。すべての Oracle ユーザーがアクセスできるビューもいくつかありますが、その他のビューはデータベース管理者のみが使用するように設計されています。

データ・ディクショナリは、データベースがオープンしていれば常に使用可能です。データ・ディクショナリは、常にオンライン状態にある SYSTEM 表領域にあります。

データ・ディクショナリは、ビューのセットによって構成されています。多くの場合、そのセットは、類似した情報が格納されている 3 つのビューで構成され、それぞれが接頭辞によって区別されます。

表 2-1 データ・ディクショナリ・ビューの接頭辞

接頭辞	有効範囲
USER	ユーザーのビュー（ユーザーのスキーマにある）
ALL	広義のユーザーのビュー（ユーザーがアクセスできる）
DBA	データベース管理者のビュー（ユーザー全員のスキーマの内容）

列の集合は、次の例外を除き、ビュー全体で同一です。

- 接頭辞が USER のビューには、通常、列 OWNER は含まれない。USER ビューでは、この列には、問合せを発行するユーザーが暗黙的に想定されます。
- 一部の DBA ビューには、管理者にとって有用な情報を含む列が追加されている。

追加情報： データ・ディクショナリ・ビューとその列の完全なリストは、『Oracle8i リファレンス・マニュアル』を参照してください。

接頭辞が USER のビュー

通常のデータベース・ユーザーにとって最も関心のあるのは、接頭辞が USER のビューでしょう。これらのビューには次のような特徴があります。

- ユーザーが作成したスキーマ・オブジェクトやユーザーによる権限付与に関する情報など、ユーザー独自のプライベートなデータベース環境を参照する。
- ユーザーに関係する行だけを表示する。
- 列 OWNER が暗黙的に想定される（現ユーザー）ことを除いて、他のビューと同一の列を持っている。
- ALL_ ビューにある情報のサブセットを戻す。
- 使用しやすいように短縮した PUBLIC シノニムを持つことができる。

たとえば、次の問合せは、自分のスキーマに入っているすべてのオブジェクトを戻します。

```
SELECT object_name, object_type FROM user_objects;
```

接頭辞が ALL のビュー

接頭辞が ALL のビューは、ユーザーのデータベース全体の概要を参照します。これらのビューは、ユーザーが所有しているスキーマ・オブジェクトに加えて、権限とロールの PUBLIC への付与や明示的な付与によってそのユーザーがアクセスできるようになったスキーマ・オブジェクトに関する情報を戻します。たとえば次の問合せは、アクセス権を持っているすべてのオブジェクトに関する情報を戻します。

```
SELECT owner, object_name, object_type FROM all_objects;
```

接頭辞が DBA のビュー

接頭辞が DBA のビューには、データベース全体のグローバル・ビューが示されます。したがって、これらのビューに問合せを発行できるのはデータベース管理者だけです。システム権限 SELECT ANY TABLE を付与されているユーザーは、データ・ディクショナリの接頭辞が DBA のビューに問合せを発行できます。

DBA ビューへの問合せは管理者のみが発行すべきであるため、これらのビューのシノニムは作成されません。このため、DBA ビューに問合せを発行するには、管理者は、次のようにして所有者 SYS をビューの名前の接頭辞として指定する必要があります。

```
SELECT owner, object_name, object_type FROM sys.dba_objects;
```

SELECT ANY TABLE システム権限を持つ管理者は、スクリプト・ファイル DBA_SYNONYMS.SQL を実行し、自分のアカウントに DBA ビューのプライベート・シノニムを作成できます。このスクリプトを実行して作成されるのは、現ユーザーのみのためのシノニムです。

DUAL 表

表 DUAL は、既知の結果を保証するために、Oracle とユーザー作成のプログラムによって参照されるデータ・ディクショナリ内の小さな表です。この表には DUMMY という 1 つの列と、値 "X" を格納する 1 つの行があります。

追加情報： DUAL 表の詳細は、『Oracle8i SQL リファレンス』の SELECT コマンドの説明を参照してください。

動的パフォーマンス表

Oracle は、操作中ずっと、現在のデータベース・アクティビティを記録する一連の仮想表を保持しています。これらの表のことを動的パフォーマンス表といいます。

動的パフォーマンス表は実際の表ではないので、ほとんどのユーザーはこれにアクセスするべきではありません。しかし、データベース管理者は、これらの表のビューに問合せを発行したり、ビューを作成したり、それらのビューにアクセスする権限を他のユーザーに付与できます。これらのビューは、データベース管理者が変更または削除できないので、固定ビューと呼ばれることもあります。

動的パフォーマンス表は SYS が所有しており、その名前はすべて V_\$ で始まります。これらの表のビューが作成され、そのビューのパブリック・シノニムが作成されます。これらのシノニム名は、V\$ で始まります。たとえば、V\$DATAFILE にはデータベースのデータ・ファイルに関する情報が格納され、V\$FIXED_TABLE にはデータベース内のすべての動的パフォーマンス表とビューに関する情報が格納されます。

追加情報： 動的パフォーマンス・ビューのシノニムと列の完全なリストは、『Oracle8i リファレンス・マニュアル』を参照してください。

表領域とデータ・ファイル

Space—the final frontier . . .

Gene Roddenberry: *Star Trek*

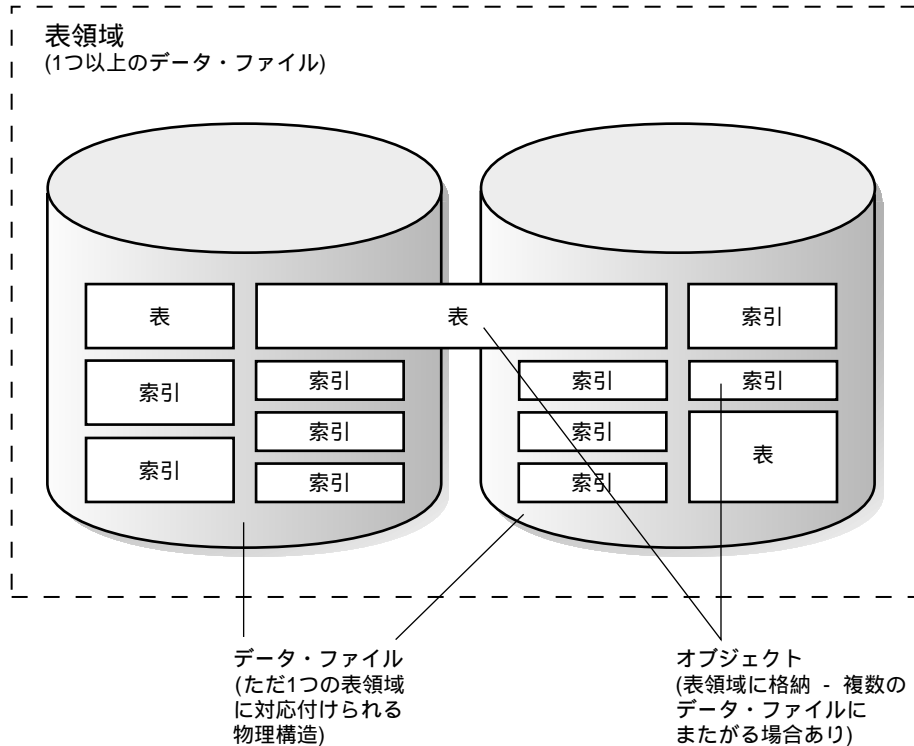
この章では、Oracle データベースの主要な論理データベース構造である表領域と、それぞれの表領域に対応する物理データ・ファイルについて説明します。この章の内容は、次のとおりです。

- データベース、表領域およびデータ・ファイル
- 表領域
- データ・ファイル

データベース、表領域およびデータ・ファイル

Oracle のデータは、論理的には表領域に格納され、物理的にはその表領域に対応するデータ・ファイルに格納されます。図 3-1 にこの関係を示します。

図 3-1 データ・ファイルと表領域



データベース、表領域およびデータ・ファイルは、それぞれ密接に関連し合っていますが、次のような重要な相違があります。

データベースと表領域	Oracle データベースは、データベースのすべてのデータがまとめて格納される、表領域と呼ばれる 1 つ以上の論理記憶単位からなっています。
表領域とデータ・ファイル	Oracle データベース内の各表領域は、データ・ファイルと呼ばれる 1 つ以上のファイルからなっています。データ・ファイルは、Oracle が稼働中のオペレーティング・システムに準拠する物理構造です。
データベースとデータ・ファイル	データベースのデータは、データベースの各表領域を構成するデータ・ファイル内にまとめて格納されます。たとえば、最も単純な Oracle データベースには、1 つのデータ・ファイルで構成される表領域が 1 つあります。他のデータベースは、それぞれ 2 つのデータ・ファイルから構成される 3 つの表領域を持つ場合もあります（合計で 6 つのデータ・ファイルになります）。

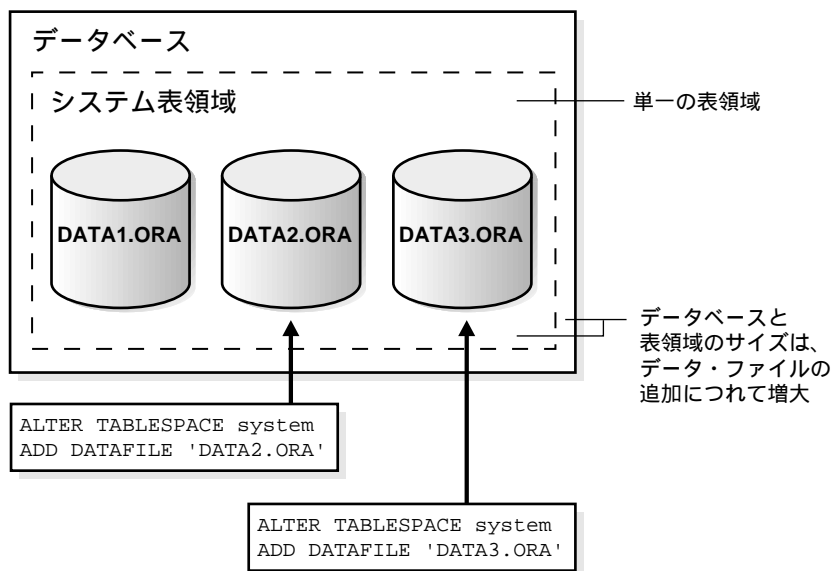
データベースへの多くの領域の割当て

データベースのサイズを拡張するには、次の3つの方法があります。

- データ・ファイルを表領域に追加する
- 新しい表領域を追加する
- データ・ファイルのサイズを大きくする

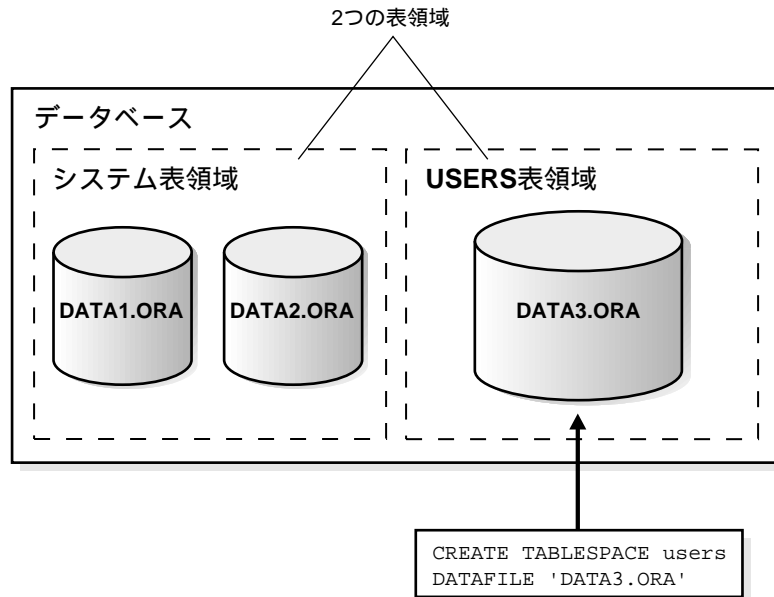
既存の表領域にデータ・ファイルを追加すると、その表領域に割り当てられているディスク領域の容量を増やすことになります。図 3-2 は、このようにして領域を拡大する方法を示しています。

図 3-2 表領域にデータ・ファイルを追加してデータベースを拡大する



別の方法として、新しい表領域を作成し（少なくとも1つのデータ・ファイルを作成）、データベースのサイズを大きくすることもできます。図 3-3 は、この方法を示しています。

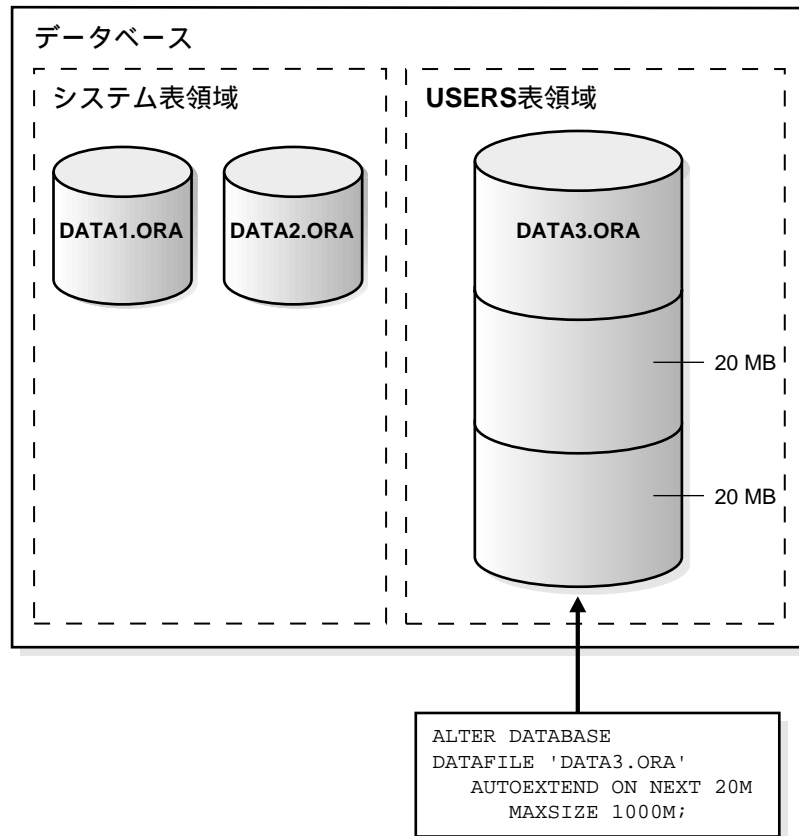
図 3-3 新しい表領域を追加してデータベースを拡大する



表領域のサイズは、表領域を構成する1つ以上のデータ・ファイルのサイズです。データベースのサイズは、データベースを構成する表領域の合計サイズです。

3番目の方法として、データ・ファイルのサイズを変更するか、必要領域の増加に応じて既存の表領域のデータ・ファイルが動的に拡張するように指定できます。そのためには、既存のファイルを変更するか、動的拡張プロパティを持つデータ・ファイルを追加します。[図 3-4](#)は、この方法を示しています。

図 3-4 データ・ファイルを動的にサイズ変更してデータベースを拡大する



追加情報： データベース内の領域を拡張する方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。

表領域

データベースは、表領域と呼ばれる 1 つ以上の論理的な記憶単位に分けられます。表領域はセグメントと呼ばれる論理記憶単位に分けられ、セグメントはさらにエクステントに分けられています（第 4 章「[データ・ブロック、エクステントおよびセグメント](#)」を参照）。

この項で表領域に関して取り上げるトピックは次のとおりです。

- [SYSTEM 表領域](#)
- [複数の表領域の使用方法](#)
- [表領域内の領域管理](#)
- [オンライン表領域とオフライン表領域](#)
- [読み込み専用表領域](#)
- [一時表領域](#)
- [データベース間での表領域のトランスポート](#)

SYSTEM 表領域

Oracle データベースには、データベースの作成時に自動的に作成される SYSTEM という表領域が含まれています。

注意： SYSTEM 表領域は、データベースのオープン中は常にオンラインにしておく必要があります。3-9 ページの「[オンライン表領域とオフライン表領域](#)」を参照してください。

データ・ディクショナリ

SYSTEM 表領域には、データベース全体のデータ・ディクショナリ表が必ず含まれます。データ・ディクショナリ表は、データ・ファイル 1 に格納されます。

PL/SQL プログラム・ユニット

ストアード PL/SQL プログラム・ユニット（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）のために格納されるデータは、すべて SYSTEM 表領域にあります。データベースにこれらのプログラム・ユニットを多数格納する場合、データベース管理者はこれらが SYSTEM 表領域内で使用する領域を確保しておく必要があります。

これらの PL/SQL プログラム・ユニットと、それらに必要な領域の詳細は、[第 18 章「プロシージャとパッケージ」](#)および[第 20 章「トリガー」](#)を参照してください。

複数の表領域の使用方法

小規模なデータベースでは、SYSTEM 表領域のみで十分な場合があります。ただし、ユーザー・データをデータ・ディクショナリ情報と切り離して格納するために、追加の表領域を少なくとも 1 つ作成することをお勧めします。これによって、各種のデータベース管理操作がより柔軟性に富んだものとなり、ディクショナリ・オブジェクトとスキーマ・オブジェクトに同一のデータ・ファイルからアクセスすることに伴う競合を減らすこともできます。

複数の表領域の用途は、次のとおりです。

- データベース・データに対するディスク領域の割当てを制御する。
- データベース・ユーザーに対し固有の領域割当て制限を設定する。
- 各表領域を個別にオンラインまたはオフラインにすることによって、データの可用性を制御する。
- データベースのバックアップ操作やリカバリ操作を部分的に実行する。
- パフォーマンスを改善するためにデータ記憶域を複数のデバイスにわたって割り当てる。

データベース管理者 (DBA) は、表領域の新規作成、表領域へのデータ・ファイルの追加、表領域内に作成されるセグメントに対するデフォルトのセグメント記憶設定値の設定や変更、表領域の読み専用または読み書き可能への変更、一時的または永続的表領域の作成および表領域の削除を行うことができます。

表領域内の領域管理

表領域では、領域がエクステント単位で割り当てられます (4-10 ページの「[エクステント](#)」を参照)。表領域では、次の 2 つの方法で空き領域と使用済み領域を追跡できます。

- データ・ディクショナリによるエクステント管理 (ディクショナリ管理の表領域)
- 表領域によるエクステント管理 (ローカル管理の表領域)

表領域の作成時に、この 2 つの領域管理方法からどちらかを選択できます。選択した方法を後で変更することはできません。

ディクショナリ管理の表領域

データ・ディクショナリを使用してエクステントが管理される表領域の場合、エクステントが割り当てられたり再使用のために解放されるたびに、Oracle はデータ・ディクショナリ内の適切な表を更新します。また、ディクショナリ表のそれぞれの更新に関するロールバック情報も格納されます。(4-19 ページの「[ロールバック・セグメント](#)」を参照。) ディクショナリ表とロールバック・セグメントはデータベースの一部であるため、これらが占める領域は、他のすべてのデータと同じ領域管理操作の対象となります。

これは、表領域でのデフォルトの領域管理方法です。リリース 8.0 またはそれ以前の Oracle では、この方法しか使用できませんでした。

ローカル管理の表領域

自身のエクステントを管理する表領域では、各データ・ファイルのビットマップがメンテナンスされ、そのデータ・ファイル内のブロックの解放または使用状態が追跡されます。ビットマップ内の各ビットは、1 ブロックまたは 1 ブロック・グループに対応します。エクステントが割り当てられるか、再使用のために解放されると、Oracle はブロックの新しい状態を示すためにビットマップ値を変更します。この変更では、データ・ディクショナリ内の表は更新されないため（表領域割当て制限情報などの特殊ケースを除く）、ロールバック情報は生成されません。

ローカル管理の表領域には、ディクショナリ管理の表領域に比べて次のような長所があります。

- エクステントのローカル管理によって、再帰的な領域管理操作が回避される。エクステント内の領域を消費または解放することによって、ロールバック・セグメントやデータ・ディクショナリ表の領域を消費または解放する他の操作が生じる場合に、ディクショナリ管理の表領域内でこのような領域管理操作が発生することがあります。
- エクステントのローカル管理によって、隣接する空き領域が自動的に追跡され、使用可能エクステントを合体する必要がなくなる。

ローカルに管理されるエクステントのサイズは、システムが動的に決定できます。また、ローカル管理の表領域内では、すべてのエクステントを同じサイズにすることもできます。詳細は、4-11 ページの「[ローカル管理のエクステント](#)」を参照してください。

この領域管理方法を指定するには、各種の CREATE コマンドで EXTENT MANAGEMENT 句の LOCAL オプションを使用します。

- SYSTEM 表領域の場合は、CREATE DATABASE コマンドで EXTENT MANGEMENT LOCAL を指定する。SYSTEM 表領域がローカルに管理される場合、データベース内の他の表領域はディクショナリ管理にすることができますが、すべてのロールバック・セグメントはローカル管理の表領域に作成する必要があります。
- SYSTEM 以外の永続表領域の場合は、CREATE TABLESPACE コマンドで EXTENT MANGEMENT LOCAL を指定する。
- 一時表領域の場合は、CREATE TEMPORARY TABLESPACE コマンドで EXTENT MANGEMENT LOCAL を指定する。（3-12 ページの「[一時表領域](#)」を参照。）

追加情報： これらのコマンドの詳細は、『Oracle8i SQL リファレンス』を参照してください。

オンライン表領域とオフライン表領域

データベース管理者は、Oracle データベースがオープンされているときはいつでも、その表領域（SYSTEM 表領域を除く）をオンライン（アクセス可能）またはオフライン（アクセス不能）にできます。SYSTEM 表領域は、データベースのオープン中は常にオンラインになっています。Oracle がデータ・ディクショナリを常に使用できるようになっている必要があります。

表領域は通常はオンラインになっており、データベース・ユーザーはその表領域内に入っているデータを使用できます。しかし、データベース管理者は、次のような理由で表領域をオフラインにすることがあります。

- データベースの一部を使用できないようにする。一方で、データベースの残りの部分は通常どおりアクセスできるようにする。
- 表領域のオフライン・バックアップを実行する（ただし、オンラインで使用中の表領域をバックアップすることもできます）。
- アプリケーションの更新やメンテナンスをする間、アプリケーションとその表のグループを一時的に使用できないようにする。

使用中のロールバック・セグメントを含む表領域をオフラインにすることはできません。詳細は、4-19 ページの「[ロールバック・セグメント](#)」を参照してください。

表領域がオフラインになった場合

表領域がオフラインになると、Oracle では後続の SQL 文でその表領域内に含まれるオブジェクトを参照できなくなります。表領域内のデータを参照する完了済みの文を含むアクティビティ・トランザクションは、トランザクション・レベルでは影響を受けません。それらの完了済みの文に対応するロールバック・データは、遅延ロールバック・セグメント（SYSTEM 表領域にある）に保存されます。表領域が再びオンラインになると、このロールバック・データは必要に応じて表領域に適用されます。

表領域がオフラインになったり、オンラインに戻ったときには、SYSTEM 表領域内のデータ・ディクショナリにそのことが記録されます。表領域がオフラインのときにデータベースを停止すると、後からそのデータベースをマウントして再オープンしたときにも、その表領域はオフラインのままです。

表領域をオンラインにできるのは、その表領域を作成したデータベース内のみに限られます。これは、必要なデータ・ディクショナリ情報がそのデータベースの SYSTEM 表領域内に維持されているためです。オフラインの表領域は、Oracle 以外のユーティリティで読み込んだり、編集したりできません。したがって、オフラインの表領域をデータベース間で転送することはできません。（オンラインの表領域をデータベース間で転送する方法は、3-12 ページの「[一時表領域](#)」を参照してください）。

追加情報： ツールを使用して Oracle データをデータベース間で転送する方法は、『Oracle8i ユーティリティ・ガイド』を参照してください。

ある種のエラーが発生すると（データベース・ライター・プロセス DBWn が、表領域のデータ・ファイルに書き込もうとして、何度か失敗した場合など）、Oracle は表領域をオンラインからオフラインへ自動的に切り替えます。オフラインの表領域にある表にアクセスしようとしたユーザーは、エラーを受け取ります。このようにディスク I/O が失敗してしまう原因がメディア障害の場合は、ハードウェアの問題を解決してから、表領域を回復する必要があります。

特殊なプロシージャに対する表領域の使用方法

複数の表領域を作成して異なる種類のデータを分離する場合、さまざまなプロシージャに合わせて特定の表領域をオフラインにします。他の表領域はオンラインのままなので、その中の情報は使用できる状態のままです。ただし、表領域をオフラインにすると、特殊な状況が発生することがあります。たとえば、2つの表領域を使用して表データを索引データから分離する場合は、次のようなことに注意してください。

- 索引を含む表領域がオフラインになっていても、問合せはその表データにアクセスできる。問合せは、表データへのアクセスには索引を必要としないためです。
- 表データを含む表領域がオフラインになっていると、データベース内のその表データにはアクセスできない。データへのアクセスには表が必要なためです。

つまり、ある文を実行するのに十分な情報がオンライン表領域内にあるれば、その文は実行されます。オフライン表領域内のデータが必要な場合、その文の実行は失敗します。

読み専用表領域

読み専用表領域の主な目的は、データベースの静的な部分を大量にバックアップしたりリカバリ（回復）しなくてもよいようにすることです。読み専用表領域のファイルは Oracle によって更新されることがないため、CD-ROM や WORM ドライブのような読み専用メディアに常駐させることができます。

注意： 表領域は、表領域が作成されたデータベースにおいてしかオンラインにできないため、読み専用表領域はアーカイブ要件やデータ発行要件を満たすとは限りません。

新規の表領域は、必ず読み書き可能なものとして作成されます。ALTER TABLESPACE コマンドの READ ONLY オプションにより、表領域を読み専用に変更するとともに、この表領域に関連したデータ・ファイルをすべて読み専用にすることができます。

ALTER TABLESPACE ... READ ONLY コマンドは、表領域を「読み専用推移」モードにし、既存のトランザクションが完了（コミットまたはロールバック）するまで待ちます。この推移状態では、前にそ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を除き、その表領域に対する後続の書き込み操作は禁止されます。したがって、推移中の表領域は、ROLLBACK を除くすべてのユーザー・コマンドに対して、読み専用の表領域と同じように動作します。既存のトランザクションがすべてコミットまたはロールバックされると、ALTER TABLESPACE ... READ ONLY コマンドが完了し、表領域は読み専用モードになります。

注意： 読み専用推移状態が発生するのは、初期化パラメータ COMPATIBLE の値が 8.1.0 またはそれ以上の場合のみです。パラメータ値が 8.1.0 未満の場合は、アクティブなトランザクションが存在すると、ALTER TABLESPACE ... READ ONLY コマンドは失敗します。

その後、ALTER TABLESPACE コマンドの READ WRITE オプションを使用して、読み専用表領域を再び読み書き可能にすることができます。

追加情報： 表領域を読み専用または読み書きモードに変更する方法の詳細は『Oracle8i 管理者ガイド』、ALTER TABLESPACE コマンドの詳細は『Oracle8i SQL リファレンス』を参照してください。

表領域を読み専用にしても、その表領域のオフライン / オンライン状態が変化することはありません。オフラインのデータ・ファイルにはアクセスできません。読み専用表領域に含まれるデータ・ファイルをオンラインにすると、読みのみが可能になります。このデータ・ファイルは、対応付けられている表領域が読み書き状態に戻るまで、書込みができません。読み専用表領域のデータ・ファイルは、ALTER DATABASE コマンドの DATAFILE オプションを使用すれば個別にオンラインまたはオフラインにすることができます。

読み専用表領域は修正できません。読み専用表領域を更新するには、最初に表領域を読み書き可能にします。表領域の更新後に、その表領域を読み専用に再設定します。

読み専用表領域は変更されないため、バックアップを繰り返す必要はありません。また、データベースを回復する必要が生じたときにも、読み専用表領域は修正されていないため、回復する必要がありません。ただし、インスタンス回復またはメディア回復時には、読み専用表領域が読み書き可能になっていたことがあるかどうかと、読み書き可能になっていた時期に応じて、注意を必要とする場合があります。

追加情報： 回復の詳細は、『Oracle8i 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。

表や索引などの項目は、オフライン表領域から削除できるのと同様に、読み専用表領域からも削除できます。ただし、読み専用表領域内でオブジェクトを作成または変更することはできません。

読み専用表領域へのデータ・ファイルの追加は、その表領域をオフラインにしても実行できません。データ・ファイルを追加すると、Oracle はファイル・ヘッダーを更新する必要がありますが、読み専用表領域ではその書込み操作が禁止されているからです。

一時表領域

ソート作用の領域は、ソート専用指定した「一時表領域」を使用すれば、さらに効率的に管理できます。そうすることによって、ソート領域の割当ておよび割当て解除に伴う領域管理操作の直列化を効率的に解消できます。

結合、索引作成、順序づけ（ORDER BY）、集計の計算（GROUP BY）および ANALYZE コマンドを使用したオプティマイザ統計の収集など、ソートを使用するすべての操作には、一時表領域の機能が役立ちます。これにより、Oracle Parallel Server 環境ではパフォーマンスが大幅に向上します。

ソート・セグメント

一時表領域は、ソート・セグメントにのみ使用できます。（セグメントの詳細は、[第4章「データ・ブロック、エクステントおよびセグメント」](#)を参照してください。）ユーザーが一時セグメント用に指定する表領域とは異なり、一時セグメント用に指定する表領域としては、ユーザーが使用可能な任意の表領域を使えます。永続スキーマ・オブジェクトを一時表領域に格納することはできません。

ソート・セグメントは、1つのセグメントが複数のソート操作によって共有されている場合に使用されます。特定の表領域内でソート操作を実行する各インスタンス内に、ソート・セグメントが1つずつ存在します。

メモリー容量を超える大規模なソート操作が複数ある場合には、一時表領域を使用するとパフォーマンスが改善されます。最初のソート操作の実行時に、所定の一時表領域のソート・セグメントが作成されます。このソート・セグメントは、セグメント・サイズが、そのインスタンスで実行中のアクティブなソート操作すべてに必要な記憶容量の合計以上になるまで、エクステントが割り当てられて拡大します。

一時表領域の作成および削除

一時表領域を作成するには、CREATE TABLESPACE または CREATE TEMPORARY TABLESPACE コマンドを使用します。

- ローカル管理の一時表領域の場合は、CREATE TEMPORARY TABLESPACE を使用する。このコマンドでは、DATAFILES ではなく TEMPFILES を指定します（3-17 ページの「[一時データ・ファイル](#)」を参照）。
- ディクショナリ管理の一時表領域の場合は、CREATE TABLESPACE の TEMPORARY オプションを使用する。

また、どの一時表領域の場合も（ローカル管理かディクショナリ管理かを問わず）、ALTER TABLESPACE コマンドを使用すると、表領域を PERMANENT から TEMPORARY に、またはその逆に変更できます。ローカル管理とディクショナリ管理の表領域の詳細は、3-8 ページの「[表領域内の領域管理](#)」を参照してください。

追加情報： CREATE TABLESPACE、CREATE TEMPORARY TABLESPACE および ALTER TABLESPACE コマンドの詳細は『Oracle8i SQL リファレンス』、ソートおよびハッシュ結合用に一時表領域を設定する方法は『Oracle8i チューニング』を参照してください。

データベース間での表領域のトランスポート

「トランスポートابل表領域」機能により、Oracle データベース間でそのサブセットを移動できます。ある表領域のクローンを作成して別のデータベースにプラグインする（データベース間で表領域をコピーする）方法と、ある Oracle データベースから表領域をアンプラグして他の Oracle データベースにプラグインする（データベース間で表領域を移動する）方法があります。

表領域をトランスポートする場合は、データ・ファイルをコピーして表領域のメタデータを統合するだけでよい。この方法によるデータ移動は同じデータをエクスポート／インポートまたはアンロード／ロードするよりも何倍も高速です。表領域をトランスポートする場合は、表データのインポート後またはロード後に索引を再構築しなくてもよいように、索引データも移動できます。

現行のリリースでは、表領域をトランスポートできるのは、両方の Oracle データベースが同じデータ・ブロック・サイズおよびキャラクタ・セットを使用し、同じハードウェア・ベンダーの互換プラットフォーム上で稼働している場合のみです。

表領域の別のデータベースへの移動またはコピー

表領域の集合を移動またはコピーするには、その表領域を読み込み専用にし、その中のデータ・ファイルをコピーし、エクスポート／インポートを使用してデータ・ディクショナリに格納されているデータベース情報（「メタデータ」）を移動する必要があります。データ・ファイルとメタデータのエクスポート・ファイルを、必ずターゲット・データベースにコピーします。これらのファイルをトランスポートするには、オペレーティング・システムのコピー機能、ftp または CD 上でのパブリッシングなど、フラット・ファイルのコピー機能を使用できます。

データ・ファイルをコピーしてメタデータをインポートした後は、必要に応じて表領域を読み書きモードにすることができます。

追加情報： 表領域を別のデータベースに移動またはコピーする方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。

トランスポートابل・データ・セット 1 つ以上の表領域で構成されているデータ・セットは、そこに含まれるスキーマ・オブジェクトの集合が自己完結型の場合に限りトランスポートできます（オブジェクト参照は例外です。13-9 ページの「REF」を参照してください）。BFILE へのポインタを含むデータ・セットをトランスポートする場合は、BFILE も移動して、ターゲット・データベース内のディレクトリを正しく設定する必要があります。

データ・セットにパーティション表が含まれている場合は、その表のすべてのパーティションを含めてください。パーティション表のサブセットを移送するために、あらかじめパーティションを表に変換できます。

表領域のメタデータ エクスポートするメタデータには、使用するエクスポート・オプションに応じて、トリガー、権限付与および制約に関する情報を含めることも、除外することもできます。主キー制約は常にエクスポートされます。

表領域のトランスポートによる利点

表領域のトランスポートは、次の場合に役立ちます。

- データ・ウェアハウス
- データ・マート
- データ・パブリケーション

また、表領域をトランスポートすると、互換性やリリース・レベルが異なる Oracle データベース間でも、データを移動またはコピーできます。

追加情報： Oracle のリリース間または互換性レベル間で表領域を移動またはコピーする方法の詳細は、『Oracle8i 移行ガイド』を参照してください。

データ・ウェアハウスとデータ・マート 企業の「データ・ウェアハウス」には、自社に関する詳細な履歴データが含まれています。通常、データは、1 つ以上のオンライン・トランザクション処理 (OLTP) データベースから、月次、週次または日次ベースでデータ・ウェアハウスに送られます。データは、一般に「ステージング・データベース」で処理されてから、データ・ウェアハウスに追加されます。

「データ・マート」には、特定の事業体、部門またはユーザー・グループにとって価値のある社内データのサブセットが含まれています。通常、データ・マートは企業データ・ウェアハウスから導出されます。

表領域のトランスポートは、データ・ウェアハウス環境でのさまざまな用途で役立ちます。

- データを OLTP データベースからステージング・データベースに移動する。ここでデータをクリーン・アップし、変換してからデータ・ウェアハウスに送ることができます。
- データをステージング・データベースから企業データ・ウェアハウスに移動する（新しいデータは、表をパーティションに変換することによって、履歴データのパーティションにすることができます）。
- データをデータ・ウェアハウスからデータ・マートに移動する。
- 古いデータをデータ・ウェアハウスからアーカイブし、必要に応じて表領域を復元できるように、アーカイブ・データをメタデータとともに保管する。

追加情報： データ・ウェアハウスとデータ・マートの詳細は、『Oracle8i チューニング』を参照してください。

データ・パブリケーション「コンテンツ・プロバイダ」は、データを入手して、それを有効な形式にして提供します。たとえば、コンテンツ・プロバイダが病院から統計データを入手して保険会社に提供したり、電話会社が大口顧客に請求データを CD で提供している場合があります。コンテンツ・プロバイダは、表領域をトランスポートして構造化データを CD や他のメディアの形で発行し、顧客が発行されたデータを各自の Oracle データベースに統合できるようにします。

データ・ファイル

Oracle データベース内の表領域は、1 つ以上の物理的な「データ・ファイル」から構成されます。1 つのデータ・ファイルは、1 つの表領域および 1 つのデータベースのみに対応付けることができます。

Oracle は、指定されたディスクの容量に、ファイル・ヘッダーに必要なオーバーヘッドの量を加えた容量のファイルを割り当てることによって、表領域のデータ・ファイルを作成します。データ・ファイルが作成されるとき、そのファイルが Oracle に割り当てられる前に、Oracle が実行されているオペレーティング・システムによってそのファイルから古い情報や許可が消去されます。ファイルが大きい場合には、この処理にかなりの時間がかかることがあります。

追加情報： 使用するオペレーティング・システムにおいてデータ・ファイルのファイル・ヘッダーに必要な領域の容量は、オペレーティング・システム固有の Oracle のマニュアルを参照してください。

どのデータベースでも最初の表領域は常に SYSTEM 表領域であるため、データベースの最初のデータ・ファイルは、データベースの作成時に自動的に SYSTEM 表領域に割り当てられます。

データ・ファイルの内容

データ・ファイルが最初に作成されるときに、割り当てられるディスク領域はフォーマットされますが、ユーザー・データは含まれていません。ただし、その領域は、対応する表領域の将来のセグメントのデータを保持するために確保されます。その領域は、Oracle のみを使用することになります。表領域内のデータが増加すると、Oracle は対応付けられたデータ・ファイル内の空き領域を使用してセグメントにエクステンツを割り当てます。詳細は、[第 4 章「データ・ブロック、エクステンツおよびセグメント」](#)を参照してください。

表領域内のスキーマ・オブジェクトに対応付けられたデータは、物理的には、表領域を構成する 1 つ以上のデータ・ファイルに格納されます。ただし、スキーマ・オブジェクトは、特定のデータ・ファイルには対応しません。データ・ファイルは、特定の表領域内の任意のスキーマ・オブジェクトのデータを保持するためのリポジトリです。スキーマ・オブジェクトに関連するデータのための領域は、表領域の 1 つ以上のデータ・ファイル内に割り当てられます。このため、1 つのスキーマ・オブジェクトが 1 つ以上のデータ・ファイルに「またがる」場合があります。表の「ストライプ化」(データが複数のディスク上に分散される)を使用しない限り、データベース管理者とエンド・ユーザーは、どのデータ・ファイルにスキーマ・オブジェクトが格納されるかを制御できません。

データ・ファイルのサイズ

データ・ファイルを作成した後でそのデータ・ファイルのサイズを変更することも、表領域内のスキーマ・オブジェクトのサイズが拡大するにつれてデータ・ファイルのサイズが動的に拡大するように指定することもできます。この機能によって、表領域当たりのデータ・ファイルの数を少なくして、データ・ファイルの管理を単純化できます。

追加情報： データ・ファイルのサイズ変更の詳細は、『Oracle8i 管理者ガイド』を参照してください。

オフライン・データ・ファイル

SYSTEM 以外の表領域は、いつでも「オフライン」（使用不能）にしたり、「オンライン」（使用可能）にすることができます。表領域をオフラインまたはオンラインにすると、その表領域を構成するすべてのデータ・ファイルは 1 単位としてオフラインまたはオンラインになります。

個々のデータ・ファイルをオフラインにすることもできます。ただし、それを行うのは通常、一部のデータベース回復手順を実行するときのみです。

一時データ・ファイル

ローカルに管理される一時表領域には、次の点を除き、通常のデータ・ファイルと同様の一時データ・ファイル（「テンポラリー・ファイル」）があります。

- テンポラリー・ファイルは、常に NOLOGGING モードに設定される。
- テンポラリー・ファイルを読込み専用にすることはできない。
- テンポラリー・ファイルは改名できない。
- ALTER DATABASE コマンドではテンポラリー・ファイルを作成できない。
- テンポラリー・ファイルはメディア回復では認識されない。
 - BACKUP CONTROLFILE ではテンポラリー・ファイルの情報は生成されない。
 - CREATE CONTROLFILE ではテンポラリー・ファイルに関する情報を指定できない。
- テンポラリー・ファイル情報は、ディクショナリ・ビュー DBA_TEMP_FILES と動的パフォーマンス・ビュー V\$TEMPFILE には表示されるが、DBA_DATA_FILES や V\$DATAFILE には表示されない。

ローカルに管理される表領域の詳細は、3-8 ページ「[表領域内の領域管理](#)」を参照してください。

データ・ブロック、エクステントおよびセグメント

He was not merely a chip of the old block, but the old block itself.

Edmund Burke: *On Pitt's first speech*

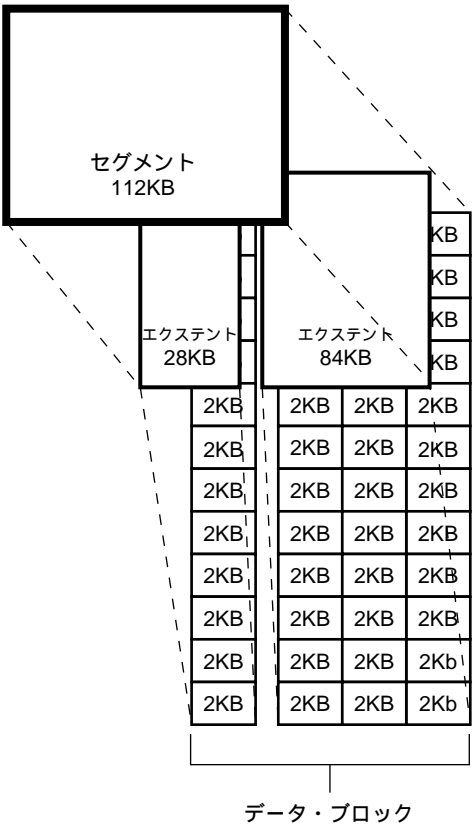
この章では、Oracle Server の論理的な記憶構造の性質と、それらの構造の相互関係について説明します。この章の内容は次のとおりです。

- データ・ブロック、エクステントおよびセグメントの関係
- データ・ブロック
- エクステント
- セグメント

データ・ブロック、エクステントおよびセグメントの関係

Oracle では、データベース内のすべてのデータに対して論理データベース領域が割り当てられます。データベース領域の割当て単位は、データ・ブロック、エクステントおよびセグメントです。次の図は、これらのデータ構造の間の関係を示しています。

図 4-1 セグメント、エクステントおよびデータ・ブロックの関係



最も細かいレベルでは、Oracle はデータを「データ・ブロック」(「論理ブロック」,「Oracle ブロック」または「ページ」とも呼ばれる)に格納します。1つのデータ・ブロックは、ディスク上の特定のバイト数の物理データベース領域に対応します。

論理的なデータベース領域の次のレベルは、「エクステント」と呼ばれます。エクステントは、特定のタイプの情報を格納するために割り当てられる、連続した一定数のデータ・ブロックです。

エクステントの上位に位置する論理的なデータベース記憶領域のレベルは、「セグメント」と呼ばれます。セグメントは、それぞれ特定のデータ構造体に割り当てられ、それら全体が同じ表領域に格納されている、エクステントの集合です。たとえば、各表のデータはそれぞれ専用の「データ・セグメント」に格納され、各索引のデータはそれぞれ専用の「索引セグメント」に格納されます。表や索引がパーティション化されている場合、各パーティションはそれぞれ専用のセグメント内に格納されます。

Oracle では、セグメントの領域は 1 エクステント単位で割り当てられます。あるセグメントの既存のエクステントが満杯であれば、そのセグメントには別のエクステントが割り当てられます。エクステントは必要に応じて割り当てられるため、1 つのセグメントの各エクステントはディスク上で連続している場合と連続していない場合があります。

1 つのセグメントとそのすべてのエクステントは、1 つの表領域内に格納されます。表領域内のセグメントは、複数のファイルからのエクステントを含むことがあります。つまり、セグメントは複数のデータ・ファイルにまたがる場合があります。ただし、各エクステントが含むことのできるデータは、1 つのデータ・ファイルからのデータのみです。

データ・ブロック

Oracle では、データベースのデータ・ファイル内の記憶領域は、「データ・ブロック」と呼ばれる単位で管理されます。データ・ブロックは、データベースで使用される I/O の最小単位です。これとは対照的に、物理的なオペレーティング・システム・レベルでは、すべてのデータはバイト単位で格納されます。各オペレーティング・システムには、「ブロック・サイズ」と呼ばれる単位があります。Oracle では、データはオペレーティング・システム・ブロックではなく、Oracle データ・ブロックの倍数を単位とする必要があります。

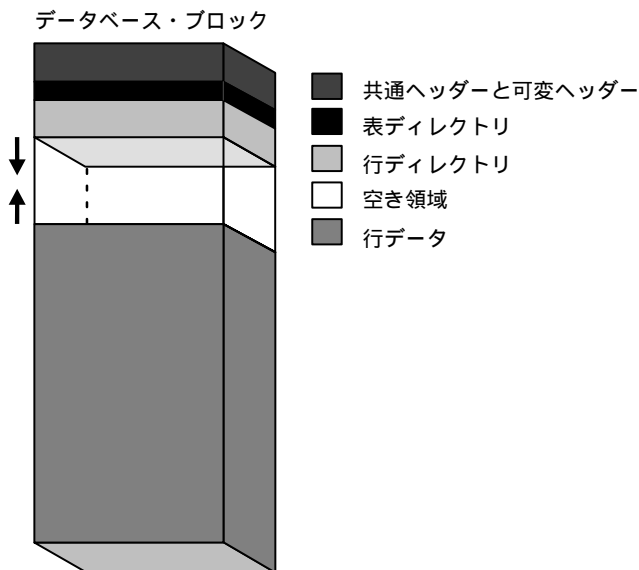
どの Oracle データベースについても、その作成時にデータ・ブロックのサイズを設定します。このデータ・ブロック・サイズは、不必要な I/O を避けるための最大値（オペレーティング・システムに固有の値）の範囲内で、オペレーティング・システムのブロック・サイズの倍数にしてください。Oracle データ・ブロックは、Oracle が使用および割当て可能な最小の格納単位です。

追加情報： データ・ブロック・サイズの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

データ・ブロックの形式

Oracle データ・ブロックの形式は、データ・ブロックに表、索引またはクラスタ化されたデータのどれが含まれるかにかかわらず類似しています。図 4-2 は、データ・ブロックの形式を示しています。

図 4-2 データ・ブロックの形式



ヘッダー（共通と可変）

ヘッダーには、ブロック・アドレスやセグメントのタイプ（たとえば、データ、索引またはロールバック）などの、一般的なブロック情報が収録されます。

表ディレクトリ

データ・ブロックのこの部分には、このブロックに行を持つ表についての情報が格納されます。

行ディレクトリ

データ・ブロックのこの部分には、ブロック内の実際の行に関する情報（行データ領域内での各行断片のアドレスを含む）が収録されます。

データ・ブロックのオーバーヘッドの行ディレクトリに領域が割り当てられると、その後は行が削除されてもこの領域は再利用されません。したがって、現在は空き状態でも、ある時点では最大 50 行が入っていたブロックでは、ヘッダー内の行ディレクトリに 100 バイトが割り当てられたままになっています。この領域は、新しい行がブロックに挿入されるときに限り再利用されます。

オーバーヘッド

データ・ブロック・ヘッダー、表ディレクトリおよび行ディレクトリのことを、まとめて「オーバーヘッド」と呼びます。一部のブロック・オーバーヘッドはサイズが固定ですが、ブロック・オーバーヘッド全体のサイズは可変です。データ・ブロック・オーバーヘッドの固定部と可変部の合計は、平均で 84 ~ 107 バイトになります。

行データ

データ・ブロックのこの部分には、表データまたは索引データが格納されます。行は、複数のブロックにまたがる場合があります。4-9 ページの「[行連鎖と移行](#)」を参照してください。

空き領域

空き領域は、新しい行の挿入や、追加の領域を必要とする行の更新（後続 NULL が NULL 以外の値に更新される場合など）に割り当てられます。発行された挿入が、あるデータ・ブロックで実際に実行されるかどうかは、そのデータ・ブロック内の現在の空き領域の容量と、領域管理パラメータ PCTFREE の値によって決まります。領域管理パラメータの詳細は、次の 4-5 ページの「[PCTFREE、PCTUSED および行連鎖の基礎知識](#)」を参照してください。

表やクラスタのデータ・セグメント、または索引の索引セグメントに割り当てられたデータ・ブロックでは、空き領域にトランザクション・エントリを格納することもできます。「トランザクション・エントリ」は、ブロック内の 1 つ以上の行にアクセスする INSERT、UPDATE、DELETE および SELECT...FOR UPDATE の各文について、ブロック内に必要です。トランザクション・エントリに必要な領域は、オペレーティング・システムによって異なります。ただし、多くのオペレーティング・システムでは、トランザクション・エントリのために約 23 バイトが必要です。

PCTFREE、PCTUSED および行連鎖の基礎知識

PCTFREE と PCTUSED という 2 つの領域管理パラメータにより、特定のセグメントのすべてのデータ・ブロック内での行の挿入および更新における空き領域の使用を制御できます。これらのパラメータは、表またはクラスタ（専用の「データ」セグメントを持つ）を作成または変更するときに指定します。記憶領域パラメータ PCTFREE は、索引（専用の「索引」セグメントを持つ）を作成または変更するときにも指定できます。

注意： この説明は、LOB データ型（BLOB、CLOB、NCLOB および BFILE）には適用されません。これらのデータ型では、PCTFREE 記憶領域パラメータや空きリストは使用されません。詳細は、12-11 ページの「[LOB データ型](#)」を参照してください。

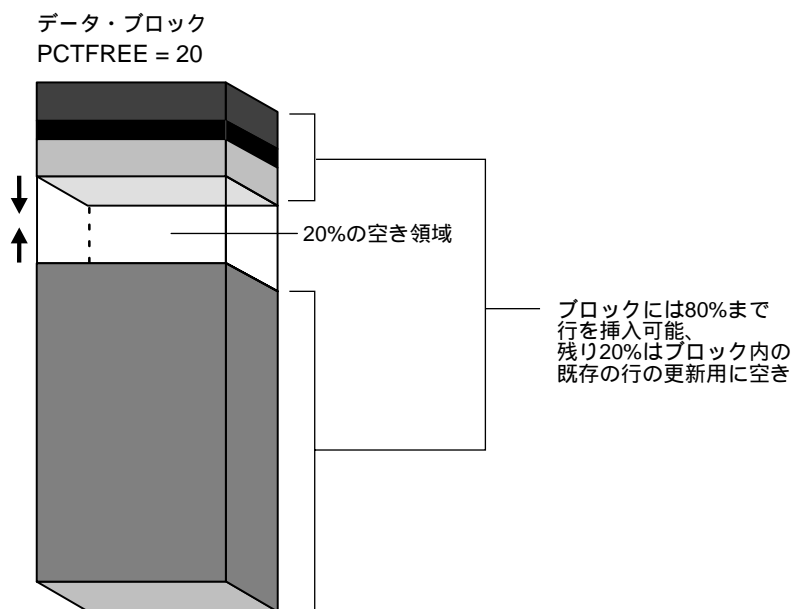
PCTFREE パラメータ

PCTFREE パラメータは、ブロック内の既存の行を更新する場合に備えてデータ・ブロック内で空き領域として「確保」される割合の最小値を設定します。たとえば、CREATE TABLE 文で次のようにパラメータを指定するとします。

```
PCTFREE 20
```

これによって、この表のデータ・セグメント内の各データ・ブロックの 20% が空き状態で維持されます。この空き領域は、各ブロック内の既存の行が更新される場合に使用されます。行データとオーバーヘッドの合計が合計ブロック・サイズの 80% になるまで、新規の行を行データ領域に追加し、それに対応する情報をオーバーヘッド領域の可変部分に追加できます。図 4-3 は、PCTFREE を示しています。

図 4-3 PCTFREE



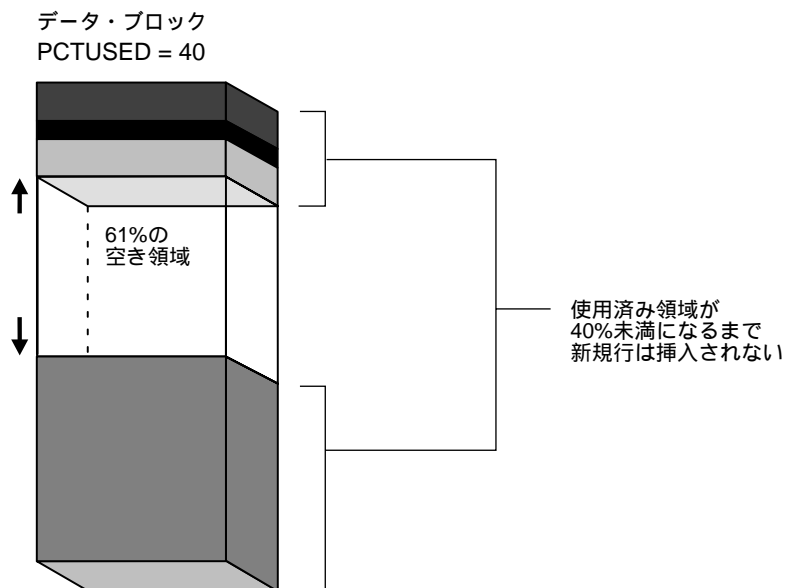
PCTUSED パラメータ

PCTUSED パラメータは、新しい行をブロックに追加するときに、行データとオーバーヘッドに「使用」できるブロックの割合の最小値を設定します。データ・ブロックが PCTFREE で指定した限界値まで満たされると、そのブロックにおける割合がパラメータ PCTUSED の値を下回るまで、Oracle はそのブロックを新しい行の挿入には使用できないものとみなします。この PCTUSED の値に到達するまでは、データ・ブロックの空き領域は、すでにデータ・ブロックに入っている行の更新にのみ使用されます。たとえば、CREATE TABLE 文で次のようにパラメータを指定したとします。

```
PCTUSED 40
```

この場合、この表のデータ・セグメントとして使用されるデータ・ブロックは、ブロックの使用領域の総量が 39% 以下になるまでは、新しい行の挿入に使用できないものとみなされます（ブロックの使用領域がそれ以前に PCTFREE に達していたと仮定します）。図 4-4 は、このことを示しています。

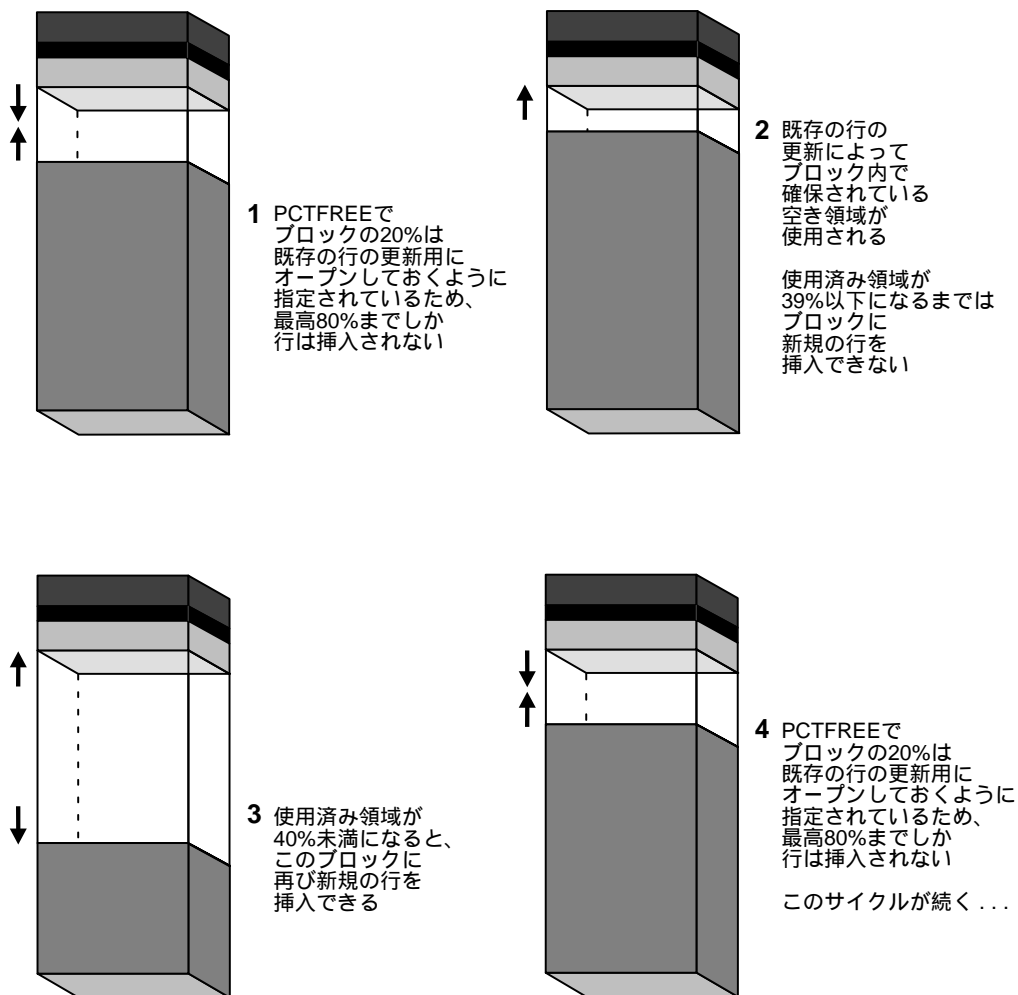
図 4-4 PCTUSED



PCTFREE と PCTUSED の機能

PCTFREE と PCTUSED が連動することにより、データ・セグメント内にあるエクステントのデータ・ブロック内の領域の使用が最適化されます。図 4-5 は、この 2 つのパラメータの相互作用を示しています。

図 4-5 PCTFREE と PCTUSED によるデータ・ブロックの空き領域の管理



新しく割り当てられたデータ・ブロックでは、挿入に使用できる領域は、ブロック・サイズからブロック・オーバーヘッドと空き領域（PCTFREE）の合計を引いた大きさです。既存データの更新では、そのブロック内の使用可能な領域がすべて使用できます。したがって、更新によって、ブロックの使用可能な領域が PCTFREE（更新用に確保され、挿入には使用できない領域）よりも小さくなる可能性があります。

各データ・セグメントと索引セグメントについて、Oracle は 1 つ以上の「空きリスト」を維持します。空きリストとは、セグメントのエクステントに割り当てられているデータ・ブロックで、PCTFREE よりも大きな空き領域があるブロックのリストのことです。これらのデータ・ブロックは、挿入に使用可能です。INSERT 文を発行すると、表の空きリスト内で最初に使用可能なデータ・ブロックがチェックされ、可能であれば使用されます。そのブロックの空き領域が足りないために INSERT 文に対応できず、そのブロックが PCTUSED 以上であれば、空きリストからはずされます。1 つのセグメントに複数の空きリストがあれば、複数の挿入が同時に行われるときに、空きリストの競合を軽減できます。

DELETE 文または UPDATE 文が発行されると、Oracle はその文を処理して、ブロック内の使用中の領域が PCTUSED よりも小さくなったかどうかを調べます。小さい場合、そのブロックはトランザクション空きリストの先頭に登録され、そのトランザクションで最初で使用される使用可能ブロックになります。トランザクションがコミットされると、ブロック内の空き領域は他のトランザクションで使用可能になります。

データ・ブロック内の空き領域の可用性と圧縮

1 つ以上のデータ・ブロックの空き領域を増加させる文は、DELETE 文と、既存の値を小さい値に更新する UPDATE 文の 2 つです。これらの文によって解放された領域は、次の条件を満たす後続の INSERT 文で使用できます。

- INSERT 文が同じトランザクション内に存在し、領域を解放した文の後にある場合、その INSERT 文では、使用可能になった領域を使用できる。
- INSERT 文と、領域を解放する文が別々のトランザクションに含まれている（おそらく別々のユーザーが実行している）場合、その INSERT 文では、もう一方のトランザクションがコミットされた後で、しかもその領域が必要とされる場合に限り、使用可能になった領域を使用できる。

解放された領域は、データ・ブロック内の空き領域の主な領域と連続しているとは限りません。データ・ブロックの空き領域が連続した領域にまとめられるのは、(1) INSERT 文または UPDATE 文が使用するブロックに新しい行断片が十分に収まる大きさの空き領域があり、かつ、(2) その空き領域が断片化しているために行断片をブロックの連続した部分に挿入できない場合に限られます。上記の場合にのみ圧縮が行われる理由は、そうでなければ、データ・ブロックの空き領域を絶えず圧縮しようとするによりデータベース・システムのパフォーマンスが低下するためです。

行連鎖と移行

表の行データが 1 つのデータ・ブロック内に収まらない状況が 2 つあります。1 番目の状況は、行を最初に挿入するときに、その行が大きすぎて 1 つのデータ・ブロック内に収まらない場合です。このような場合、Oracle はその行のデータを、そのセグメント用に確保されたデータ・ブロックの「連鎖」(1 つ以上) に格納します。行連鎖は、データ型 LONG または LONG RAW の列が入っている行など、大きな行で最も頻繁に発生します。そのような場合の行連鎖は、避けられません。

注意： 行の形式と行断片については、10-5 ページの「[行の形式とサイズ](#)」を参照してください。

それに対して、2 番目の状況は、1 つのデータ・ブロック内にすでに収まっていた行が更新されて、行全体の長さが増加したが、ブロックの空き領域がすでに満杯になっている場合です。この場合、Oracle はその行全体のデータを新しいデータ・ブロックに「移行」します（その行全体が新しいブロックに収まる場合）。移行された行の元の行断片は、行の移行先の新しいブロックを指すように保たれます。そのため、移行された行の ROWID は変わりません。ROWID の詳細は、10-7 ページの「[行断片の ROWID](#)」および 12-14 ページの「[物理 ROWID](#)」を参照してください。

行が連鎖または移行されると、その行に関連する I/O パフォーマンスは低下します。これは、その行の情報を取り出す際に、Oracle が複数のデータ・ブロックを走査する必要があるためです。

追加情報： 行の連鎖と移行を減らして I/O パフォーマンスを改善する方法は、『Oracle8i チューニング』を参照してください。

エクステント

エクステントは、複数の連続したデータ・ブロックで構成される、データベース記憶領域の割当ての論理単位です。1 つまたは複数のエクステントによって、セグメントが構成されます。セグメント内の既存の領域を使用し尽くすと、Oracle はそのセグメントに新しいエクステントを割り当てます。

エクステントが割り当てられる時期

表を作成する場合、Oracle はその表のデータ・セグメントに、指定された数のデータ・ブロックからなる「初期エクステント」を割り当てます。行はまだ挿入されていませんが、初期エクステントに対応する Oracle データ・ブロックは、その表の行のために確保されています。

セグメントの初期エクステントのデータ・ブロックが満杯になり、新しいデータを保持するためさらに多くの領域が必要な場合、Oracle はそのセグメントに「増分エクステント」を自動的に割り当てます。増分エクステントは、それより前にそのセグメントに割り当てられていたエクステントのサイズと同一またはそれ以上の後続エクステントです。（次の項では、増分エクステントのサイズを制御する要素について説明します。）

メンテナンスに役立つように、各セグメントのヘッダー・ブロックには、そのセグメント内のエクステントのディレクトリが含まれています。

ロールバック・セグメントには必ず 2 つ以上のエクステントがあります。詳細は、4-21 ページの「[エクステントの使用法とロールバック・セグメントへの割当て方法](#)」を参照してください。

注意： この章は、1つのサーバー・プロセスによって SQL 文を解析して実行する、シリアル操作に適用されます。複数のサーバー・プロセスを必要とするパラレル SQL 文では、エクステントは少し異なる方法で割り当てられます。詳細は、26-30 ページの「[空き領域とパラレル DDL](#)」を参照してください。

エクステントの数とサイズの決定

エクステントで指定される「記憶領域パラメータ」によって、すべてのセグメントが定義されます。記憶領域パラメータは、すべてのタイプのセグメントに適用されます。記憶領域パラメータにより、特定のセグメントに空きデータベース領域がどのように割り当てられるかが制御されます。たとえば、CREATE TABLE 文の STORAGE 句で記憶領域パラメータを指定することによって、表のデータ・セグメント用に最初に確保される領域の大きさを決定したり、表で割当て可能なエクステントの数を制限できます。表の記憶領域パラメータを指定しなければ、表領域のデフォルト記憶領域パラメータが使用されます。

表領域では、そのエクステントをローカルに、またはデータ・ディクショナリを介して管理できます（3-8 ページの「[表領域内の領域管理](#)」を参照）。記憶領域パラメータには、ディクショナリ管理の表領域内のエクステントにのみ適用されるものと、すべてのエクステントに適用されるものがあります。

ローカル管理のエクステント

エクステントをローカルに管理する表領域の場合は、均一のエクステント・サイズでも、システムによって自動的に決定される可変エクステント・サイズでもかまいません。表領域の作成時に、UNIFORM または AUTOALLOCATE（システム管理）オプションで割当てのタイプを指定します。

- システム管理のエクステントの場合は、初期エクステントのサイズを指定すると、他のエクステントの最適サイズが自動的に決定される。この場合、最小エクステント・サイズは 64KB です。これは、永続表領域のデフォルトです。
- 均一エクステントの場合は、エクステント・サイズを指定するか、またはデフォルト・サイズである 1MB を使用できる。エクステントがローカルに管理される一時表領域の場合は、この種の割当てしか使用できません。

ローカル管理のエクステントの場合、NEXT、PCTINCREASE、MINEXTENTS、MAXEXTENTS および DEFAULT STORAGE 記憶領域パラメータは無効です。

データ・ディクショナリ管理のエクステンツ

データ・ディクショナリを使用してエクステンツが管理される表領域の場合、エクステンツ・サイズは記憶領域パラメータ INITIAL、NEXT および PCTINCREASE によって決定されます。このような表領域内でスキーマ・オブジェクトを作成すると、その最初のエクステンツは INITIAL サイズで割り当てられます。さらに領域が必要になった場合は、NEXT および PCTINCREASE パラメータによって新しいエクステンツのサイズが決定されます。NEXT と PCTINCREASE の値は、スキーマ・オブジェクトの作成後に変更できます。

追加情報： 記憶領域パラメータの詳細は、『Oracle8i 管理者ガイド』および『Oracle8i SQL リファレンス』を参照してください。

エクステンツの割当て方法

Oracle がエクステンツを割り当てるときには、ローカルに管理されるかディクショナリによって管理されるかに応じて、異なるアルゴリズムが使用されます。

ローカル管理の表領域内でのエクステンツの割当て

ローカル管理の表領域では、Oracle は最初に表領域内で候補となるデータ・ファイルを判別します。次に、そのデータ・ファイルのビットマップ内で必要数の隣接する空きブロックを検索することによって、空き領域を検索し、新しいエクステンツを割り当てます。そのデータ・ファイルに十分な隣接する空き領域がなければ、Oracle は他のデータ・ファイルを調べます。

ディクショナリ管理の表領域内でのエクステンツの割当て

ディクショナリ管理の表領域の場合、Oracle は、特定のセグメントに対する増分エクステンツの割当てを次のように制御します。

1. 次のアルゴリズムにより、空き領域（そのセグメントを含む表領域内）を検索し、増分エクステンツより大きいサイズを持つ最初の連続した空きデータ・ブロックの集合を探します。
 - a. 内部的な断片化を少なくするために、新しいエクステンツのサイズに 1 ブロックを加えたサイズと一致する、連続したデータ・ブロックの集合を検索します。（必要であれば、そのサイズは表領域の最小エクステンツ・サイズの単位にまで切り上げられる。）たとえば、新しいエクステンツに 19 個のデータ・ブロックが必要な場合は、ちょうど 20 個の連続したデータ・ブロックを検索します。ただし、新しいエクステンツが 5 ブロック以下の場合は、必要なブロック数に余分のブロックは追加しません。

- b. 完全に一致するブロックの集合が見つからないときは、必要量より大きい連続データ・ブロックの集合を検索します。必要なエクステントのサイズより 5 ブロック以上大きい連続ブロックのグループが見つかったら、ブロック・グループをそれぞれ必要サイズの個々のエクステントに分割します。必要サイズより大きいブロック・グループが見つかったら、5 ブロック以上大きくなければ、すべての連続ブロックを新しいエクステントに割り当てます。

この例では、正確に 20 個の連続データ・ブロックの集合が見つからなければ、20 個を超える連続データ・ブロックの集合が検索されます。最初に見つかった集合に 25 個以上のブロックが含まれていれば、そのブロックが分割され、そのうちの 20 個が新しいエクステントに割り当てられ、残りの 5 個以上のブロックは空き領域として残ります。見つかった最初の集合に 21 ~ 24 個のブロックがある場合は、すべてのブロックを新しいエクステントに割り当てます。

- c. 必要なサイズ以上の連続したデータ・ブロックの集合が見つからない場合は、対応する表領域内の空いている隣接したデータ・ブロックを結合して、より大きな連続データ・ブロックの集合を形成します。(SMON バックグラウンド・プロセスも、連続した空き領域を定期的に結合します。) 表領域のデータ・ブロックを結合した後、1a および 1b で説明した検索を再度実行します。
- d. この 2 回目の検索の後でもエクステントを割り当てることができない場合には、自動拡張によりファイルのサイズ変更を試みます。ファイルのサイズ変更ができない場合、エラーが戻されます。

2. 表領域内に必要な空き領域を見つけて割り当てた後、Oracle は、増分エクステントのサイズに相当する空き領域の一部を割り当てます。エクステントに必要な容量よりも大きな空き領域が見つかった場合、残りは空き領域として残します (5 個以上の連続したブロックの場合)。
3. 新しいエクステントが割り当てられたことと、その割り当てられた領域が使用できなくなったことを示すように、セグメント・ヘッダーとデータ・ディクショナリを更新します。

新しく割り当てられたエクステントのブロックは、空き領域であったとしても、古いデータが残っている場合があります。通常、Oracle は、新しく割り当てられたエクステントを使用し始めるときに、そのエクステントのブロックをフォーマットします。ただし、これは必要な場合だけです (セグメントの空きリストにあるブロックから始めます)。例外として、データベース管理者が ALTER TABLE 文または ALTER CLUSTER 文に ALLOCATE EXTENT オプションを指定して増分エクステントの割当てを強制した場合などには、エクステントが割り当てられる時点でエクステントのブロックがフォーマットされます。

エクステントが割当て解除される時期

一般に、セグメントにデータが格納されているスキーマ・オブジェクトを (DROP TABLE 文または DROP CLUSTER 文によって) 削除するまで、そのセグメントのエクステントは表領域に戻されません。ただし、これには次のような例外があります。

- 表やクラスタの所有者、または DELETE ANY 権限を持つユーザーは、TRUNCATE...DROP STORAGE 文を使用して表やクラスタを切り捨てることができる。
- OPTIMAL サイズが指定されているロールバック・セグメントについては、定期的に 1 つ以上のエクステントの割当てが解除されることがある。
- データベース管理者 (DBA) は、次の SQL 構文を使用して未使用のエクステントの割当てを解除できる。

```
ALTER TABLE table_name DEALLOCATE UNUSED;
```

エクステントが解放されると、Oracle はデータ・ファイル内のビットマップを変更するか (ローカル管理の表領域の場合)、データ・ディクショナリを更新して (ディクショナリ管理の表領域の場合)、再度取得されたエクステントを使用可能領域として反映させます。解放されたエクステントのブロックにあるデータにはアクセスできなくなり、そのブロックが他のエクステント用に再利用される時点でそのデータが消去されます。

追加情報： エクステントの割当て解除の詳細は、『Oracle8i 管理者ガイド』および『Oracle8i SQL リファレンス』を参照してください。

クラスタ化されていない表のエクステント

クラスタ化されていない表が存在している限り、その表を切り捨てるまで、そのデータ・セグメントに割り当てられたデータ・ブロックは割当て解除されません。十分な領域があれば、Oracle はブロックに新しい行を挿入します。表の行をすべて削除しても、データ・ブロックが再生されて表領域内の他のオブジェクトがそれを使用できるようになることはありません。

クラスタ化されていない表を削除した後、他のエクステントが空き領域を必要とするときに、この領域を再生することができます。それらの表が存在していた表領域のデータ・セグメントおよび索引セグメントのすべてのエクステントが再生され、その表領域内の他のオブジェクトが使用できるようになります。

ディクショナリ管理の表領域の場合は、使用可能エクステントより大きいエクステントがセグメントに必要になると、Oracle は再生済みの連続したエクステントを識別し、それらを結合して大きいエクステントにします。これを、エクステントを「結合する」といいます。

ローカル管理の表領域の場合は、新しいエクステントが 1 つ以上のエクステントから再生されたかどうかに関係なく、すべての連続した空き領域が新しいエクステントへの割当てに使用可能なため、エクステントを結合する必要はありません。

クラスタ化表のエクステント

クラスタ化表では、クラスタ用に作成されたデータ・セグメントに情報が格納されます。したがって、クラスタ内の 1 つの表を削除した場合、データ・セグメントはクラスタ内の他の表が使用できるように残ります。エクステントが割当て解除されることはありません。また、エクステントを解放するために、クラスタ (ハッシュ・クラスタを除く) を切り捨てることもできます。

マテリアライズド・ビューとそのログのエクステント

Oracle は、マテリアライズド・ビューとマテリアライズド・ビュー・ログ（レプリケーション環境ではスナップショットとスナップショット・ログ）のエクステントを、表やクラスタの場合と同じ方法で割当て解除します。マテリアライズド・ビューとそのログの詳細は、10-17 ページの「[マテリアライズド・ビュー](#)」を参照してください。

索引内のエクステント

索引セグメントに割り当てられたエクステントはすべて、その索引が存在する限り、割り当てられたままになります。索引や、それに対応する表またはクラスタを削除すると、エクステントは再生されて、表領域内で他の目的に使用できるようになります。

ロールバック・セグメント内のエクステント

Oracle では、データベースのロールバック・セグメントが最適サイズを超えていないかが定期的にチェックされます。ロールバック・セグメントが最適サイズを超えている（つまり、エクステントが多すぎる）場合、ロールバック・セグメントから 1 つ以上のエクステントが自動的に割当て解除されます。詳細は、4-24 ページの「[ロールバック・セグメントからのエクステントの割当て解除](#)」を参照してください。

一時セグメント内のエクステント

一時セグメントを必要とする文の実行が完了すると、一時セグメントは自動的に削除され、そのセグメントに割り当てられていたエクステントは、対応する表領域に戻されます。単一のソートでは、その文を発行したユーザーの一時表領域に専用の一時セグメントが作成されます。その後、そのエクステントは表領域に戻されます。

それに対して複数のソートでは、ソート専用設定されている一時表領域のソート・セグメントを使用できます。これらのソート・セグメントは、インスタンスごとに 1 回だけ割り当てられ、ソート後には戻されずに残るので、他の複数ソートでそのセグメントを使用できます。詳細は、4-17 ページの「[一時セグメント](#)」を参照してください。

一時表の一時セグメントには、1 つのトランザクションまたはセッションの複数の文に関するデータが入っています（10-10 ページの「[一時表](#)」を参照）。Oracle は、トランザクションまたはセッションの終了時に一時セグメントを削除します。そのセグメントに割り当てられていたエクステントは、対応する表領域に戻されます。

セグメント

セグメントは、表領域内の特定の論理記憶構造のデータがすべて入っている、エクステントの集合です。たとえば、各表には、その表のデータ・セグメントを形成する 1 つ以上のエクステントが割り当てられ、各索引には、その索引セグメントを形成する 1 つ以上のエクステントが割り当てられます。

Oracle データベースでは、次の 4 タイプのセグメントが使用されます。

- [データ・セグメント](#)
- [索引セグメント](#)
- [一時セグメント](#)
- [ロールバック・セグメント](#)

データ・セグメント

Oracle データベース内の 1 つのデータ・セグメントには、次のいずれかのデータがすべて保持されます。

- パーティション化またはクラスタ化されていない表
- パーティション表のパーティション
- 表のクラスタ

このデータ・セグメントは、CREATE コマンドを使用して表またはクラスタを作成する時点で作成されます。

データ・セグメントのエクステントがどのように割り当てられるかは、表またはクラスタの記憶領域パラメータによって決定されます。これらの記憶領域パラメータは、適切な CREATE または ALTER コマンドによって直接設定できます。これらの記憶領域パラメータは、オブジェクトに対応するデータ・セグメントのデータ検索と格納の効率に影響を与えます。

注意： Oracle は、スナップショットとスナップショット・ログのセグメントを、表やクラスタの場合と同じ方法で作成します。

追加情報： スナップショットとスナップショット・ログの詳細は『Oracle8i レプリケーション・ガイド』、CREATE および ALTER コマンドの詳細は『Oracle8i SQL リファレンス』を参照してください。

索引セグメント

Oracle データベース内の各非パーティション索引には、すべてのデータを保持する索引セグメントが 1 つあります。パーティション索引の場合は、パーティションごとに、そのデータを保持する索引セグメントが 1 つずつあります。

索引または索引パーティションの索引セグメントは、CREATE INDEX コマンドを発行した時点で作成されます。このコマンドでは、索引セグメントのエクステントの記憶領域パラメータと、索引セグメントが作成される表領域を指定できます。(表のセグメントと、その表に関連する索引のセグメントは、同一の表領域に存在しなくてもかまいません。) 記憶領域パラメータを設定すると、データ検索と格納の効率に直接影響があります。

一時セグメント

Oracle では、問合せの処理中に、SQL 文の解析と実行の中間段階で、一時的な作業領域が必要になることがよくあります。Oracle は、「一時セグメント」と呼ばれるこのディスク領域を自動的に割り当てます。通常、一時セグメントは、ソート操作の作業領域として必要です。ソート操作がメモリー内で実行できる場合、または Oracle が索引を使用して操作を実行できる別の方法を見つけた場合、一時セグメントは作成されません。

一時セグメントを必要とする操作

次のコマンドでは、一時セグメントの使用が必要になる場合があります。

- CREATE INDEX
- SELECT ... ORDER BY
- SELECT DISTINCT ...
- SELECT ... GROUP BY
- SELECT ... UNION
- SELECT ... INTERSECT
- SELECT ... MINUS

一部の索引なしの結合および相関副問合せでも、一時セグメントの使用が必要になることがあります。たとえば、問合せに DISTINCT 句、GROUP BY および ORDER BY が含まれている場合、2 つの一時セグメントが必要になる可能性があります。アプリケーションで前述のコマンドを頻繁に発行する場合は、データベース管理者が初期化パラメータ SORT_AREA_SIZE を調整してパフォーマンスを改善してください。

追加情報： SORT_AREA_SIZE およびその他の初期化パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

一時表とその索引のセグメント

Oracle は、一時表と一時表に作成される索引用に一時セグメントを割り当てることがあります。一時表には、トランザクションまたはセッションの期間中にのみ存在するデータが保持されます（10-10 ページの「[一時表](#)」を参照）。

一時セグメントが割り当てられる方法

Oracle は、問合せと一時表には異なる方法で一時セグメントを割り当てます。

問合せ用の一時セグメントの割当て 一時セグメントは、ユーザー・セッション中に必要に応じて、文を発行したユーザーの一時表領域内に割り当てられます。一時表領域は、TEMPORARY TABLESPACE オプションを付けた CREATE USER または ALTER USER コマンドで指定します。ユーザーに定義されている一時表領域がない場合、SYSTEM 表領域がデフォルトの一時表領域として使用されます。一時セグメントのエクステントの記憶特性は、一時セグメントが作成された表領域のデフォルト値によって決まります。

一時セグメントは、その文が完了すると削除されます。

一時セグメントの割当てと割当て解除は頻繁に発生するため、一時セグメント専用の表領域を作成してください。これによって、ディスク・デバイス間で I/O を分散させるとともに、SYSTEM 表領域やその他の表領域に一時セグメントが保持されることによって生じるそれらの表領域の断片化を避けることもできます。

ユーザーの一時セグメント表領域を割り当てる方法は、[第 29 章「データベース・アクセスの制御」](#)を参照してください。

REDO ログには、ソート操作用の一時セグメントに対する変更のエントリは格納されません。ただし、例外として、一時セグメントに対する領域管理操作のエントリは格納されます。

一時表および索引用の一時セグメントの割当て Oracle は、一時表に対して最初の INSERT が発行された時点で、その表にセグメントを割り当てます。（これは、CREATE TABLE AS SELECT によって内部で発行される挿入操作の場合もあります。）一時表に対する最初の INSERT 文では、表とその索引にセグメントが割り当てられ、索引のルート・ページが作成され、LOB セグメントが割り当てられます。

一時表のセグメントは、その表を作成したユーザーの一時表領域内で割り当てられます。

トランザクションやセッションの終了時に、Oracle は、それぞれに固有の一時表のセグメントを削除します。その一時表の使用を他のトランザクションまたはセッションが共有している場合、そのデータを含むセグメントは表に残ります。

詳細は、10-10 ページの「[一時表](#)」を参照してください。

ロールバック・セグメント

各データベースには 1 つ以上のロールバック・セグメントが入っています。ロールバック・セグメントには、各トランザクションで（コミットされたかどうかに関係なく）変更されたデータの古い値が記録されます。ロールバック・セグメントは、読み込み一貫性の実現、トランザクションのロールバックおよびデータベースの回復のために使用されます。

ロールバック・セグメントがさまざまな状況下でどのように機能するかは、次に示す項を参照してください。

トピック	項
読み込み一貫性	27-4 ページの「 マルチバージョン一貫性制御 」
トランザクションのロールバック	17-6 ページの「 トランザクションのロールバック 」
データベースの回復	32-9 ページの「 ロールバック・セグメントとロールバック 」

ロールバック・セグメントの内容

ロールバック・セグメント内の情報は、いくつかの「ロールバック・エントリ」で構成されています。ロールバック・エントリには、たとえば、ブロック情報（変更されたデータのファイル番号とブロック ID）やトランザクション内の操作を実行する前のデータが入っています。同一トランザクションについてのロールバック・エントリはリンクされるため、トランザクションをロールバックする必要が生じたときには、ロールバック・エントリが容易に見つかります。

データベース・ユーザーもデータベース管理者も、ロールバック・セグメントに対してアクセスや読み込みを実行できません。Oracle のみが書き込みや読み込みを実行できます。（ロールバック・セグメントは、実際に作成したユーザーとは関係なく、ユーザー SYS によって所有されます。）

ロールバック・エントリのロギング

ロールバック・エントリはロールバック・セグメント内のデータ・ブロックを変更します。Oracle は、データ・ブロックのすべての変更（ロールバック・エントリを含む）を REDO ログに記録します。この 2 次的なロールバック情報の記録は、システム・クラッシュの時点でのアクティブ・トランザクション（まだコミットまたはロールバックされていないトランザクション）にとって非常に重要です。システム・クラッシュが発生すると、アクティブ・

トランザクションのロールバック・エントリを含むロールバック・セグメント情報は、インスタンス回復またはメディア回復の一部として自動的に復元されます。回復が完了すると、Oracle はシステム・クラッシュの時点でコミットまたはロールバックされていなかったトランザクションのロールバックを実行します。

ロールバック情報が必要になる時期

ロールバック・セグメントごとに、Oracle は「トランザクション表」を保持します。トランザクション表は、対応するロールバック・セグメントを使用するすべてのトランザクションと、それらのトランザクションによって行われた各変更についてのロールバック・エントリのリストです。ロールバック・セグメント内のロールバック・エントリは、トランザクションのロールバックを実行したり、問合せに対して読みみ一貫性を保った結果を作成するのに使用されます。

ロールバック・セグメントには、1 つのトランザクションごとに、変更前のデータが記録されます。各トランザクションについて、新しい変更がそれぞれ前の変更とリンクされます。トランザクションをロールバックする必要がある場合は、一連の変更が、データを直前の状態に復元するのに必要な順序でデータ・ブロックに適用されていきます。

同様に、問合せに対して読みみ一貫性のある結果の集合を用意する必要があるときは、Oracle は、ロールバック・セグメント内の情報によって、特定の時点に対応する一貫したデータの集合を作成できます。

トランザクションとロールバック・セグメント

ユーザーのトランザクションが開始されるたびに、そのトランザクションは次の 2 つの方法のどちらかによってロールバック・セグメントに割り当てられます。

- トランザクションは、使用可能な次のロールバック・セグメントに自動的に割り当てられる。トランザクションの割り当ては、トランザクション内の最初の DML 文または DDL 文が発行された時点で行われます。トランザクションが SET TRANSACTION READ ONLY 文で始まっているかどうかにかかわらず、読みみ専用トランザクション（問合せのみを含むトランザクション）はロールバック・セグメントに割り当てられません。
- トランザクションは、アプリケーションによって、特定のロールバック・セグメントに明示的に割り当てられる。トランザクションの開始時に、アプリケーションの開発者またはユーザーは、そのトランザクションの実行時に Oracle で使用される特定のロールバック・セグメントを指定できます。これによって、アプリケーションの開発者やユーザーは、それぞれのトランザクションに合わせて、サイズの異なるロールバック・セグメントを選択できます。

トランザクションの存続期間中に、対応するユーザー・プロセスは、割り当てられたロールバック・セグメントのみにロールバック情報を書き込みます。

トランザクションをコミットした時点で、ロールバック情報は解放されますが、すぐに破棄されるわけではありません。トランザクションがコミットされる前に開始された問合せに対して適切なデータの読み一貫ビューを作成できるようにするために、その情報はロールバック・セグメントに残っています。そのようなビューのためにロールバック・データを可能な限り長く使用可能にするために、Oracle は、ロールバック・セグメントのエクステントを順次方式で書き込みます。ロールバック・セグメントの最後のエクステントが満杯になると、折り返してセグメント内の最初のエクステントに上書きすることにより、ロールバック・データの書き込みが継続されます。長時間実行のトランザクション（アイドル状態またはアクティブ状態）では、新しいエクステントをロールバック・セグメントに割り当てる必要が生じることがあります。トランザクションでロールバック・セグメントのエクステントが使用される方法の詳細は、4-22 ページの図 4-6、4-23 ページの図 4-7 および 4-24 ページの図 4-8 を参照してください。

各ロールバック・セグメントは、1つのインスタンスからの一定の数のトランザクションを処理できます。トランザクションを特定のロールバック・セグメントに明示的に割り当てない限り、使用可能なロールバック・セグメントへのアクティブ・トランザクションの配分は、すべてのロールバック・セグメントにほぼ同じ数のアクティブ・トランザクションが割り当てられるように行われます。この配分は使用可能なロールバック・セグメントのサイズには依存**しません**。したがって、すべてのトランザクションで同じ量のロールバック情報が生成される場合は、ロールバック・セグメントはすべて同じサイズになります。

追加情報： ロールバック・セグメントで処理できるトランザクションの数は、オペレーティング・システムによって異なるデータ・ブロックのサイズに応じて決まります。詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

エクステントの使用方法和ロールバック・セグメントへの割当て方法

ロールバック・セグメントの作成時には、記憶領域パラメータを指定して、そのセグメントへのエクステントの割当てを制御できます。各ロールバック・セグメントには、最低 2 つのエクステントが割り当てられなければなりません。

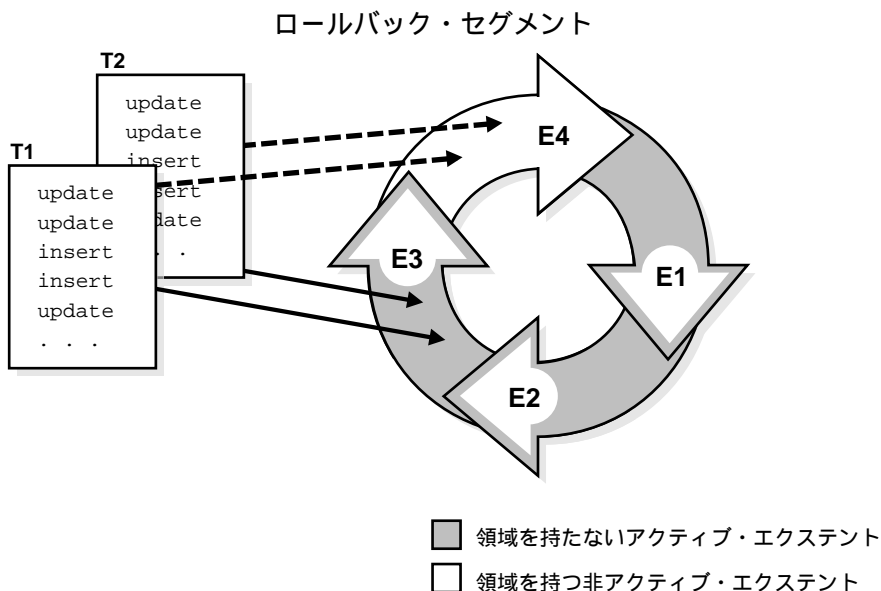
1 つのトランザクションは、1 つのロールバック・セグメントに順次方式で書き込みを行います。各トランザクションは、どの時点でもロールバック・セグメントの 1 つのエクステントにのみ書き込みます。多数のアクティブ・トランザクションが、1 つのロールバック・セグメントに同時に書き込むことができ、しかもロールバック・セグメントの同じエクステントに同時に書き込むこともできます。ただし、ロールバック・セグメントのエクステント内にある各データ・ブロックには、1 つのトランザクションについての情報しか格納できません。

トランザクションが現行のエクステント領域をすべて使用し尽くしても書き込みを継続する必要がある場合、Oracle は、次の 2 つの方法のどちらかによって、同じロールバック・セグメントから使用可能なエクステントを見つけます。

- すでにロールバック・セグメントに割り当てられているエクステントを再利用する。
- ロールバック・セグメント用に新しいエクステントを取得する（そして割り当てる）。

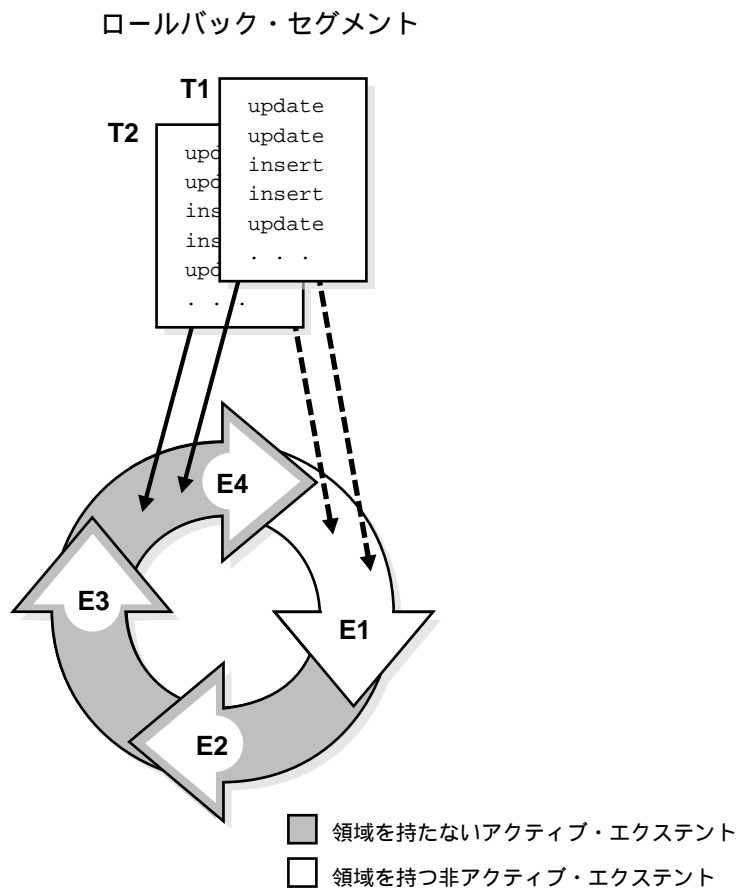
ロールバック領域をさらに取得する必要があるトランザクションは、ロールバック・セグメントの次のエクステントを調べます。ロールバック・セグメントの次のエクステントにアクティブなトランザクションからの情報が含まれていなければ、Oracle はそれを現行のエクステントにします。これにより、さらに領域を必要とするすべてのトランザクションは新しい現行のエクステントにロールバック情報を書き込めるようになります。図 4-6 は、ロールバック・セグメントの第 3 エクステント (E3) への書き込みを開始し、続いて第 4 エクステント (E4) に書き込む、2 つのトランザクション T1 および T2 を示しています。

図 4-6 ロールバック・セグメントに割り当てられたエクステントの使用



トランザクションが書き込みを継続し、現行のエクステントが満杯になると、Oracle はロールバック・セグメントにすでに割り当てられている次のエクステントを調べ、そのエクステントが使用可能かどうかを判断します。図 4-7 で、E4 が完全に満杯であるときに T1 と T2 がさらに書き込みをする場合は、ロールバック・セグメントに割り当てられている次に使用可能なエクステントに書き込みをします。この図では、E1 がそのエクステントに相当します。この図は、ロールバック・セグメント内のエクステントが循環的に使用されることを示しています。

図 4-7 ロールバック・セグメントに割り当てられたエクステントの循環的な使用

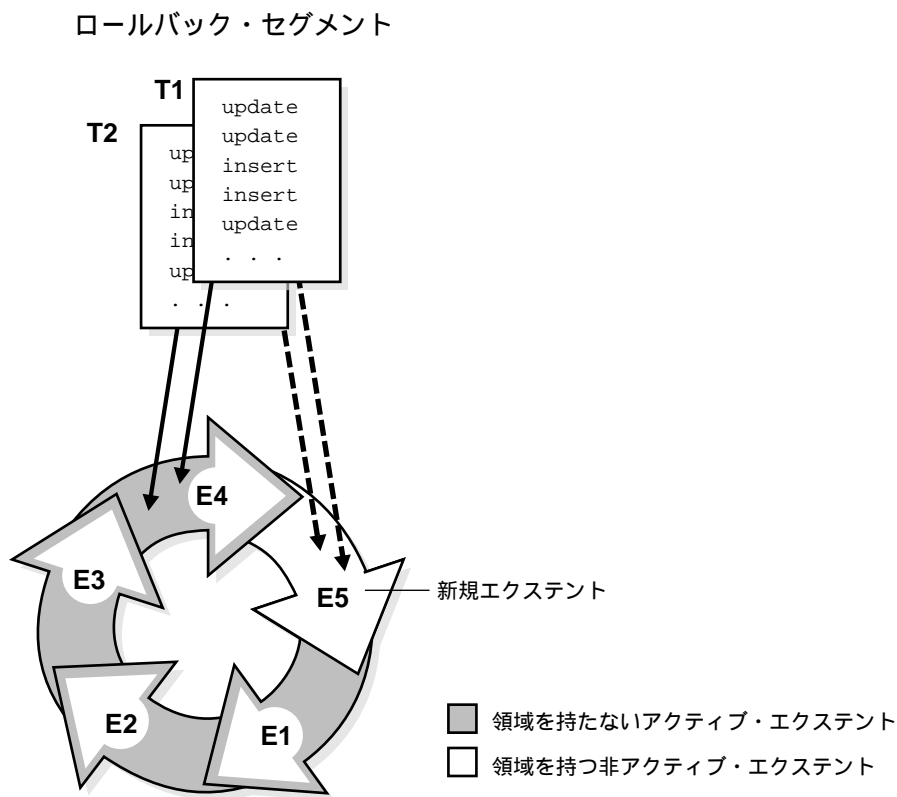


トランザクションのロールバック情報の書込みを継続するときに、Oracle は常にリング内の次のエクステントを最初に再利用しようとします。ただし、次のエクステントにアクティブ・トランザクションからのデータが含まれている場合は、新しいエクステントを割り当てる必要があります。エクステントの数が、ロールバック・セグメントの記憶領域パラメータ MAXEXTENTS に設定されている値に達するまで、Oracle は、ロールバック・セグメントに新しいエクステントを割り当てることができます。

図 4-8 に、ロールバック・セグメントに割り当てられた新しいエクステントを示します。コミットされていないトランザクションが長時間実行されています（アイドル状態、アクティブ状態または永続のインダウト分散トランザクション）。この時点で、トランザクションは

ロールバック・セグメントの4番目のエクステント（E4）に書き込んでいます。しかし、ロールバック・セグメントに割り当てられている次のエクステント（E1）にはアクティブなロールバック・エントリが含まれているため、E4が完全に満杯になると、トランザクションは書き込みを継続できません。そこで、Oracleは、このロールバック・セグメントに新しいエクステント（E5）を割り当てて、トランザクションはこの新しいエクステントへの書き込みを継続します。

図 4-8 ロールバック・セグメントへの新しいエクステントの割当て



ロールバック・セグメントからのエクステントの割当て解除

ロールバック・セグメントを削除すると、そのロールバック・セグメントのエクステントはすべてその表領域に戻されます。その後、戻されたエクステントは、表領域内の他のセグメントのために使用できるようになります。

ロールバック・セグメントを作成または変更するときには、記憶領域パラメータ OPTIMAL (ロールバック・セグメントにのみ適用される) を使用して、セグメントの最適サイズをバイト単位で指定できます。トランザクションが、ロールバック情報の書き込みをロールバック・セグメント内のあるエクステントから別のエクステントに継続する必要がある場合、Oracle は、ロールバック・セグメントの現在のサイズと最適サイズを比較します。ロールバック・セグメントが最適サイズよりも大きく、かつ満杯になったエクステントのすぐ後のエクステントがアクティブでない場合、ロールバック・セグメントの全体のサイズが、その最適サイズに等しいか、それに近い (が小さくはない) サイズになるまで、Oracle は、連続するアクティブでないエクステントの割当てをロールバック・セグメントから解除します。アクティブでないエクステントのうち最も古いものは、一貫した読み込みで使用される可能性が最も低いので、Oracle は常にその種のエクステントを解放します。

ロールバック・セグメントの OPTIMAL 設定は、セグメントの最小数のエクステントに割り当てられる領域より小さくすることはできません。

(INITIAL + NEXT + NEXT + ... up to MINEXTENTS) bytes

SYSTEM ロールバック・セグメント

データベースの作成時には、SYSTEM という名前の初期ロールバック・セグメントが必ず作成されます。この初期ロールバック・セグメントは SYSTEM 表領域内に作成され、SYSTEM 表領域のデフォルト記憶領域パラメータを使用します。SYSTEM ロールバック・セグメントは削除できません。インスタンスは、必要な他のロールバック・セグメントに加えて、SYSTEM ロールバック・セグメントをいつも取得します。

ロールバック・セグメントが複数ある場合、Oracle は特別のシステム・トランザクションにのみ SYSTEM ロールバック・セグメントを使用し、ユーザー・トランザクションを他のロールバック・セグメントに割り当てようとします。SYSTEM ロールバック・セグメント以外のロールバック・セグメントに対するトランザクションが多すぎる場合、Oracle は必要に応じて SYSTEM ロールバック・セグメントを使用します。一般には、データベース作成後に、SYSTEM 表領域内に追加のロールバック・セグメントを最低 1 つ作成する必要があります。

Oracle インスタンスとロールバック・セグメントのタイプ

Oracle インスタンスは、データベースをオープンするときに、その後のトランザクションによって生成されるロールバック情報を処理できるように、1 つ以上のロールバック・セグメントを取得する必要があります。インスタンスは、プライベート・ロールバック・セグメントとパブリック・ロールバック・セグメントの両方を取得できます。「プライベート・ロールバック・セグメント」は、インスタンスがデータベースをオープンするときに、そのインスタンスによって明示的に取得されます。「パブリック・ロールバック・セグメント」は、ロールバック・セグメントを必要とする任意のインスタンスが利用できる、ロールバック・セグメントのプールを形成します。

プライベート・ロールバック・セグメントとパブリック・ロールバック・セグメントは、データベース内にいくつ存在してもかまいません。インスタンスは、データベースをオープンするときに、次の規則に従って1つ以上のロールバック・セグメントを取得しようとします。

1. インスタンスは、最低1つのロールバック・セグメントを取得する必要があります。データベースにアクセスしているインスタンスが唯一のインスタンスの場合は、SYSTEM セグメントを取得します。Oracle Parallel Server 内のデータベースにアクセスしている複数のインスタンスの中の1つのインスタンスの場合は、SYSTEM ロールバック・セグメントと、それ以外に最低1つのロールバック・セグメントを取得します。これらのセグメントを取得できないと、エラーが戻されて、インスタンスはデータベースをオープンできません。
2. インスタンスは、少なくとも、次の初期化パラメータの値の比率と等しい数のロールバック・セグメントを常に取得しようとします。

```
CEIL (TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT)
```

CEIL は、入力の数値以上の最小の整数を戻す SQL 関数です。上記の例で、TRANSACTIONS が 155 で、TRANSACTIONS_PER_ROLLBACK_SEGMENT が 10 であれば、インスタンスは 16 個以上のロールバック・セグメントを取得しようとします。(ただし、インスタンスは、上の計算によって算出される数のロールバック・セグメントを取得できなくても、データベースをオープンすることはできます。)

注意： TRANSACTIONS_PER_ROLLBACK_SEGMENT パラメータによって、ロールバック・セグメントを使用できるトランザクションの数が制限されるわけではありません。このパラメータによって指定されるのは、インスタンスがデータベースをオープンする時点で取得しようとするロールバック・セグメントの数です。

3. インスタンスは、SYSTEM ロールバック・セグメントを取得した後、そのインスタンスの ROLLBACK_SEGMENTS パラメータに指定されているプライベート・ロールバック・セグメントをすべて取得しようとします。Oracle Parallel Server 内のインスタンスがデータベースをオープンして、別のインスタンスがすでに要求しているプライベート・ロールバック・セグメントを取得しようとした場合、ロールバック・セグメントを取得しようとしている2番目のインスタンスは起動時にエラーを受け取ります。また、存在しないプライベート・ロールバック・セグメントを取得しようとするインスタンスも、エラーを受け取ります。
4. インスタンスが、ステップ3で十分な数のロールバック・セグメントを取得できた場合は、それ以上のアクションは必要ありません。ただし、さらにロールバック・セグメントが必要な場合は、インスタンスはパブリック・ロールバック・セグメントを取得しようとします。

一度パブリック・ロールバック・セグメントがインスタンスによって要求されると、そのロールバック・セグメントがオフラインにされるか、またはそのロールバック・セグメントを要求しているインスタンスが停止するまで、他のインスタンスはこのセグメントを使用できません。

データベース内に十分な数のセグメントがあり、データベースをオープンする各インスタンスが、1つのSYSTEM ロールバック・セグメントを含む2つ以上のロールバック・セグメントを取得できれば、Oracle Parallel Server で使用されるデータベースはパブリック・セグメントのみを持ち、プライベート・セグメントは持つ必要はありません。ただし、Oracle Parallel Server の使用時には、プライベート・ロールバック・セグメントを使用する必要があることもあります。

追加情報： Oracle Parallel Server におけるロールバック・セグメントの使用方法の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

ロールバック・セグメントの状態

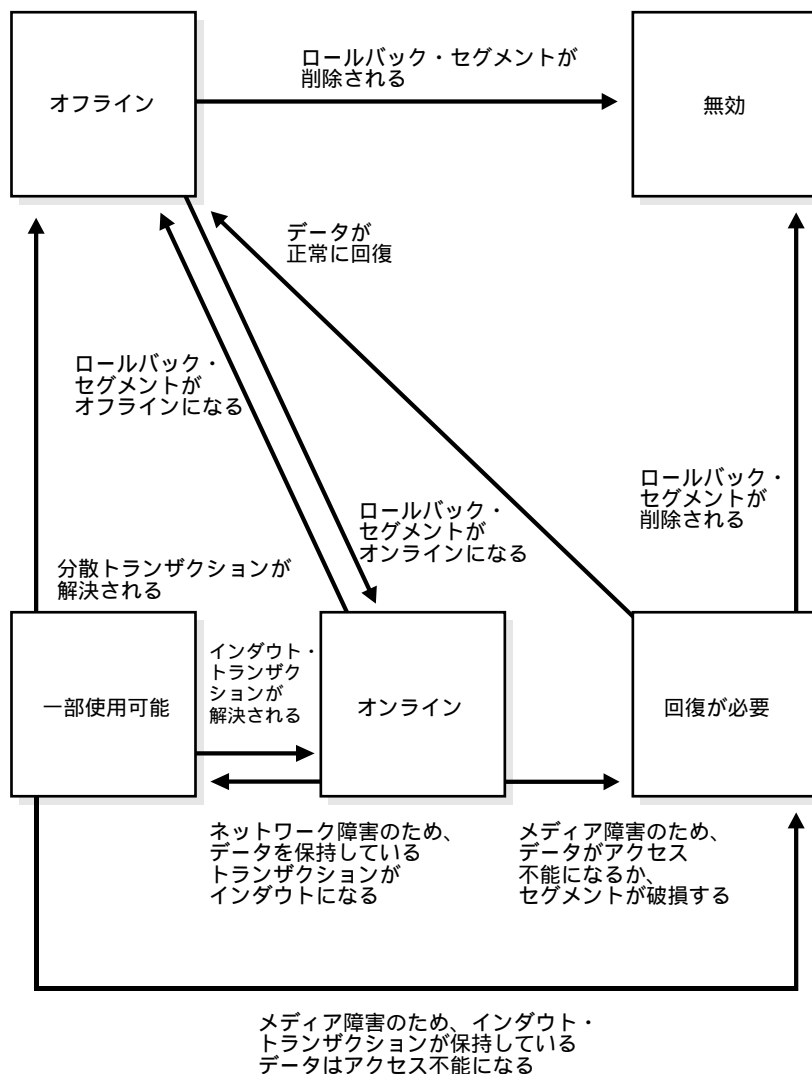
ロールバック・セグメントの状態は、常にいくつかの状態のうちの1つに該当します。たとえば、ロールバック・セグメントがオフラインになっているか、インスタンスによって取得されているか、未解決のトランザクションにかかわっているか、回復を必要としているか、削除されたかなど状況によって決まります。その状態に応じて、そのロールバック・セグメントをトランザクションで使用できるかどうか、および DBA がどの管理処理を実行できるかが決まります。

ロールバック・セグメントの状態は、次のとおりです。

OFFLINE	インスタンスによって取得されていない（オンラインになっていない）。
ONLINE	インスタンスによって取得されている（オンラインになっている）。アクティブ・トランザクションのデータが入っていることがあります。
NEEDS RECOVERY	ロールバックできず（関係するデータ・ファイルにアクセスできないため）、コミットされていないトランザクションのデータが入っているか、データが破損している。
PARTLY AVAILABLE	インダウト・トランザクション（つまり、未解決の分散トランザクション）のデータが含まれている。
INVALID	削除された（このロールバック・セグメントに割り当てられていた領域は、後の新しいロールバック・セグメントの作成時に使用されます。）

データ・ディクショナリ表 DBA_ROLLBACK_SEGS には、各ロールバック・セグメントの状態と、その他のロールバック情報がリストされます。図 4-9 に、ロールバック・セグメントの状態の変化を示します。

図 4-9 ロールバック・セグメントの状態とその推移



PARTLY AVAILABLE および NEEDS RECOVERY 状態のロールバック・セグメント PARTLY AVAILABLE 状態と NEEDS RECOVERY 状態は、ほとんど同じです。通常、どちらの状態のロールバック・セグメントにも、解決不能なトランザクションからのデータが含まれています。

- PARTLY AVAILABLE のロールバック・セグメントは、ネットワーク障害が原因で解決できないインダウト分散トランザクションで使用されている。NEEDS RECOVERY のロールバック・セグメントは、データ・ファイルの欠損や破損などのローカルなメディア障害、またはセグメント自体の破損が原因で解決できないトランザクション（ローカルまたは分散）で使用されています。
- Oracle または DBA は、PARTLY AVAILABLE のロールバック・セグメントをオンラインにすることができます。それに対して、NEEDS RECOVERY のロールバック・セグメントをオンラインにするには、そのセグメントをあらかじめ OFFLINE にしておく必要があります。（データベースを回復してトランザクションを解決すると、NEEDS RECOVERY のロールバック・セグメントの状態は自動的に OFFLINE に変更されます。）
- DBA は、NEEDS RECOVERY のロールバック・セグメントを削除できる。（これによって、DBA は破損したセグメントを削除できます。）PARTLY AVAILABLE のセグメントは削除できません。最初にインダウト・トランザクションを RECO プロセスで自動的に解決するか、手動で解決しておく必要があります。

追加情報： 分散トランザクションでの障害の詳細は、『Oracle8i 分散システム』を参照してください。

PARTLY AVAILABLE のロールバック・セグメントをオンラインにする（コマンドを使用するか、インスタンスの起動時に）と、そのセグメントは新しいトランザクションで使用できるようになります。ただし、インダウト・トランザクションがトランザクション表の一部のエントリをそのまま保持しているため、そのロールバック・セグメントを使用できる新しいトランザクションの数は制限されます。（トランザクション表の詳細は、4-20 ページの「[ロールバック情報が必要になる時期](#)」を参照してください）。

また、インダウト・トランザクションが解決されるまでは、そのトランザクションがロールバック・セグメント内に取得したエクステントは保持され続けるため、他のトランザクションはそれらのエクステントを使用することができません。そのため、そのロールバック・セグメントは、アクティブ・トランザクション用として新しいエクステントを取得する必要性が生じて、サイズが大きくなる可能性があります。ロールバック・セグメントの拡大を防ぐため、データベース管理者は、インダウト・トランザクションが解決されるまでは、PARTLY AVAILABLE のセグメントをオンラインにするよりも、トランザクション用に新しくロールバック・セグメントを作成することをお勧めします。

遅延ロールバック・セグメント

表領域がオフラインになったためにトランザクションをすぐにロールバックできない場合、Oracle は「遅延ロールバック・セグメント」に書き込みます。遅延ロールバック・セグメントには、表領域に適用できなかったロールバック・エントリが入っており、その表領域がオンラインに戻ったときに適用できるようになっています。表領域がオンラインに戻って回復されると、これらのセグメントは消滅します。遅延ロールバック・セグメントは、SYSTEM 表領域内に自動的に作成されます。

第 III 部

Oracle インスタンス

第 III 部では、Oracle インスタンスのアーキテクチャ、ネットワーク環境で利用できる各種のクライアント / サーバー構成、Oracle の起動および停止手順について説明します。

第 III 部には、次の章が含まれています。

- [第 5 章「データベースとインスタンスの起動と停止」](#)
- [第 6 章「分散処理」](#)
- [第 7 章「メモリー・アーキテクチャ」](#)
- [第 8 章「プロセスのアーキテクチャ」](#)
- [第 9 章「データベース・リソースの管理」](#)

データベースとインスタンスの起動と停止

Greetings, Prophet; The Great Work begins: The Messenger has arrived.

Tony Kushner: *Angels in America*, Part I

この章では、Oracle のインスタンスおよびデータベースの起動と停止に関連するプロセスを説明します。この章の内容は次のとおりです。

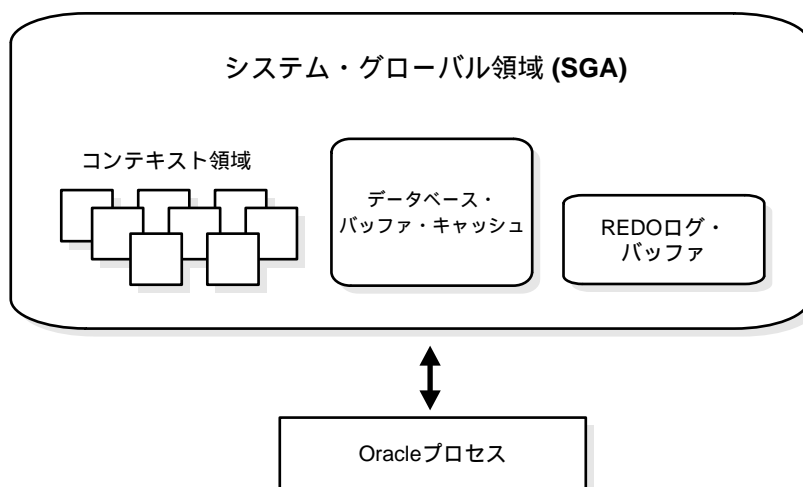
- Oracle インスタンスの概要
 - 管理者権限での接続
 - パラメータ・ファイル
- インスタンスとデータベースの起動
- データベースとインスタンスの停止

Oracle インスタンスの概要

稼働中のすべての Oracle データベースは、Oracle インスタンスに対応付けられます。データベース・サーバー上のデータベースが起動すると、コンピュータの種類にかかわらず、Oracle はシステム・グローバル領域 (SGA) というメモリー領域を割り当て、1 つ以上の Oracle プロセスを起動します。この SGA と Oracle プロセスの組合せのことを、「Oracle インスタンス」といいます。インスタンスのメモリーとプロセスは、対応付けられたデータベースのデータを効果的に管理し、データベースを使用する 1 人以上のユーザーのために機能します。

図 5-1 に Oracle インスタンスを示します。SGA と Oracle プロセスの詳細は、[第 7 章「メモリー・アーキテクチャ」](#) および [第 8 章「プロセスのアーキテクチャ」](#) を参照してください。

図 5-1 Oracle インスタンス



インスタンスとデータベース

インスタンスを起動した後、Oracle は指定されたデータベースにインスタンスを対応付けます。これをデータベースの「マウント」といいます。これでデータベースをいつでも「オープン」できるようになり、データベースをオープンすると許可されたユーザーがアクセスできるようになります。

複数のインスタンスを同時に同じコンピュータ上で実行し、それぞれ専用の物理データベースにアクセスさせることができます。クラスタ化された大規模な並列処理システム (MPP)

では、Oracle Parallel Server により、1 つのデータベースを複数のインスタンスがマウントできます。

追加情報： Oracle Parallel Server の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

データベース管理者だけが、インスタンスを起動してデータベースをオープンすることができます。データベースがオープンされている場合、データベース管理者はそのデータベースを停止してクローズすることができます。データベースが「クローズ」されている場合、ユーザーはそのデータベース内の情報にアクセスできません。

データベースの起動と停止のセキュリティは、管理者権限を使用して Oracle に接続することによって制御されます。一般のユーザーは、Oracle データベースの状態を制御できません。

管理者権限での接続

データベースの起動と停止は強力な管理オプションであり、管理者権限を使用して Oracle に接続するユーザーのみがこれを行うことができます。ユーザーの管理権限を確立するには、オペレーティング・システムに応じて、次のいずれかの条件が必要です。

- ユーザーのオペレーティング・システム権限により、そのユーザーが管理者権限を使用して接続できる。
- ユーザーが SYSDBA または SYSOPER 権限を付与されており、データベースがパスワード・ファイルを使用してデータベース管理者を認証している。
- データベースに INTERNAL ログインのパスワードがあり、ユーザーがそのパスワードを知っている。

追加のセキュリティとして、ユーザーが管理者権限で接続できるのは専用サーバーのみです（共有サーバーには接続できません）。

管理者権限で接続すると、そのユーザーは SYS が所有するスキーマ内に置かれます。これによって、SYS スキーマ内のすべてのオブジェクトにアクセスできるようになります。

データベース管理者を認証するときのパスワード・ファイルと認証方式の詳細は、[第 29 章「データベース・アクセスの制御」](#)を参照してください。

追加情報： 各オペレーティング・システムにおける管理者権限の機能の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

パラメータ・ファイル

インスタンスを起動するには、Oracle は「パラメータ・ファイル」を読み込む必要があります。パラメータ・ファイルとは、そのインスタンスとデータベースの構成パラメータ（「初期化パラメータ」）のリストを含むテキスト・ファイルです。これらのパラメータに特定の

値を設定して、Oracle インスタンスのメモリーと プロセスの設定の多くを初期化します。ほとんどの初期化パラメータは、次のグループのいずれかに属しています。

- 名前（ファイルなどの）を指定するパラメータ。
- 制限（最大数など）を設定するパラメータ。
- SGA のサイズなどの容量に影響するパラメータ。これを「可変パラメータ」といいます。

主に、初期化パラメータは Oracle に次のことを伝えます。

- インスタンスを起動するデータベースの名前
- SGA のメモリー構造に使用するメモリーの容量
- 満杯になったオンライン REDO ログ・ファイルの処理方法
- データベースの制御ファイルの名前と位置
- データベースのプライベート・ロールバック・セグメントの名前

パラメータ・ファイルの例

パラメータ・ファイルの例を次に示します。

```
db_block_buffers = 550
db_name = ORA8PROD
db_domain = US.ACME.COM
#
license_max_users = 64
#
control_files = filename1, filename2
#
log_archive_dest = c:\logarch
log_archive_format = arch%S.ora
log_archive_start = TRUE
log_buffer = 64512
log_checkpoint_interval = 256000
# rollback_segments = rs_one, rs_two
```

パラメータ値の変更

データベース管理者は、データベース・システムのパフォーマンスを改善するために、可変パラメータを調整できます。どのパラメータがシステムに最も強い影響を与えるかは、データベースのさまざまな特性や変数によって異なります。

変更されたパラメータ値は、インスタンスを起動してパラメータ・ファイルを読み込んだ後に初めて有効になります。一部のパラメータは、インスタンスの実行中に ALTER SESSION コマンドまたは ALTER SYSTEM コマンドを使用して「動的」に変更することもできます。

追加情報： すべての初期化パラメータの説明は、『Oracle8i リファレンス・マニュアル』を参照してください。SGA に影響するパラメータの詳細は、7-12 ページの「[SGA のサイズ](#)」を参照してください。

NLS パラメータ

Oracle は、このファイルの各国語サポート (NLS) パラメータに定義されている文字列リテラルを、データベースのキャラクタ・セットとして取り扱います。

追加情報： 各国語サポートの詳細は、『Oracle8i NLS ガイド』を参照してください。

インスタンスとデータベースの起動

Oracle データベースを起動して、システム全体で使用可能にするには、次の 3 つのステップを実行します。

1. インスタンスを起動します。
2. データベースをマウントします。
3. データベースをオープンします。

データベース管理者は、Oracle Enterprise Manager を使用してこれらのステップを実行できます。

追加情報： 『Oracle Enterprise Manager 管理者ガイド』を参照してください。

インスタンスを起動する

Oracle は、インスタンスを起動するときに、まずパラメータ・ファイルを読み込んで初期化パラメータの値を判別し、次に SGA (データベース情報のために使用される共有メモリー領域) を割り当てて、バックグラウンド・プロセスを作成します。この時点では、これらのメモリー構造とプロセスにデータベースは対応付けられていません。

SGA の詳細は第 7 章「[メモリー・アーキテクチャ](#)」を、バックグラウンド・プロセスの詳細は第 8 章「[プロセスのアーキテクチャ](#)」を参照してください。

インスタンスの起動の制限モード

インスタンスを制限モードで起動したり、既存のインスタンスを制限モードに変更したりできます。これにより、接続は RESTRICTED SESSION システム権限を付与されているユーザーのみに制限されます。

異常な状況でのインスタンスの強制起動

異常な状況では、たとえばインスタンスのプロセスの1つが正常に終了していないなど、直前のインスタンスが「きれいに」停止されていないことがあります。そのような状況では、次回インスタンスを通常起動するときにデータベースからエラーが戻されます。この問題を解決するには、直前のインスタンスの残りの Oracle プロセスをすべて終了してから、新しいインスタンスを起動する必要があります。

データベースをマウントする

インスタンスは、データベースをマウントして、データベースをそのインスタンスに関連付けます。データベースをマウントすると、そのインスタンスはデータベース制御ファイルを見つけてオープンします。(制御ファイルは、インスタンスの起動に使用したパラメータ・ファイルの CONTROL_FILES 初期化パラメータに指定されます。) その後で Oracle は制御ファイルを読み込み、データベースのデータ・ファイルと REDO ログ・ファイルの名前を取得します。

マウント直後のデータベースはまだクローズされているので、データベース管理者のみがそのデータベースにアクセスできます。データベース管理者は、特定のメンテナンス操作を完了するまでの間、データベースをクローズしたままにしておくことができます。ただし、データベースに対して通常の操作を行うことはまだできません。

Oracle Parallel Server によるデータベースのマウント

注意： この項で説明する機能は、Parallel Server Option 付きの Oracle8i Enterprise Edition を購入した場合にのみ利用できます。

Oracle で同時に複数のインスタンスが同じデータベースをマウントできる場合、データベース管理者は初期化パラメータ PARALLEL_SERVER を使用して、データベースを複数インスタンスに使用可能にすることができます。PARALLEL_SERVER パラメータのデフォルト値は、FALSE です。Parallel Server Option をサポートしていないバージョンの Oracle の場合、PARALLEL_SERVER は FALSE にしか設定できません。

データベースをマウントする最初のインスタンスの PARALLEL_SERVER が FALSE の場合は、そのインスタンスしかデータベースをマウントできません。ただし、PARALLEL_SERVER が TRUE に設定されていれば、他のインスタンスも PARALLEL_SERVER が TRUE に設定されている状態でデータベースをマウントできます。データベースをマウントできるインスタンスの数は、データベースの作成時に指定した最大数によって決まります。

追加情報： 1つのデータベースでの複数のインスタンスの使用の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

スタンバイ・データベースをマウントする

「スタンバイ・データベース」は、プライマリ・データベースの複製コピーをメンテナンスし、障害発生時にも引き続き使用できるようにします。

スタンバイ・データベースは、常に回復モードになっています。スタンバイ・データベースをメンテナンスするには、ALTER DATABASE コマンドを使用してスタンバイ・モードでマウントし、プライマリ・データベースによって生成されるアーカイブ済み REDO ログを適用する必要があります。スタンバイ・データベースと障害回復の詳細は、32-25 ページの「[耐障害性](#)」を参照してください。

スタンバイ・データベースは、読み専用モードでオープンして、一時的なレポート用データベースとして使用できます。(5-8 ページの「[データベースの読み専用モードでのオープン](#)」を参照。) スタンバイ・データベースを読み書きモードでオープンすることはできません。

クローン・データベースをマウントする

「クローン・データベース」は、表領域の Point-in-Time 回復に使用できる、データベースの特殊コピーです。表領域の Point-in-Time 回復を実行する場合は、クローン・データベースをマウントし、表領域を目的の時点まで回復します。次に、クローンからプライマリ・データベースにメタデータをエクスポートし、回復された表領域からデータ・ファイルをコピーします。

追加情報： クローン・データベースと表領域の Point-in-Time 回復の詳細は、『Oracle8i 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。

データベースのオープン

マウントされたデータベースをオープンすると、そのデータベースで通常のデータベース操作を実行できるようになります。有効なユーザーは、オープンされたデータベースに接続して、データベースの情報にアクセスできます。通常、データベース管理者は、データベースをオープンして一般に使用できる状態にします。

データベースをオープンすると、Oracle はオンライン・データ・ファイルとオンライン REDO ログ・ファイルをオープンします。データベースを前回停止したときに表領域がオフラインであった場合は、データベースを再オープンしたとき、その表領域と対応するデータ・ファイルはオフラインのままです。3-9 ページの「[オンライン表領域とオフライン表領域](#)」を参照してください。

データベースをオープンしようとしたときに、データ・ファイルまたは REDO ログ・ファイルのうち存在していないファイルがあると、Oracle からエラーが戻されます。そのデータベースをオープンするには、破損または欠落したファイルのバックアップから回復する必要があります。

クラッシュ回復

データベース管理者がインスタンスを異常終了させたため、または電源障害が発生したためにデータベースが異常にクローズされた場合、Oracle はデータベースの再オープン時にクラッシュ回復を自動的に実行します。32-3 ページの「[データベース・インスタンス障害](#)」を参照してください。

ロールバック・セグメントの取得

データベースをオープンするときに、インスタンスは 1 つ以上のロールバック・セグメントを取得しようとします。4-25 ページの「[SYSTEM ロールバック・セグメント](#)」および「[Oracle インスタンスとロールバック・セグメントのタイプ](#)」を参照してください。

インダウト分散トランザクションの解決

ときには、データベースが異常にクローズし、1 つ以上の分散トランザクションが「インダウト」(コミットもロールバックもされない状態)になることがあります。データベースを再オープンし、クラッシュ回復が完了すると、RECO バックグラウンド・プロセスがただちに自動的に実行されて、インダウト分散トランザクションが矛盾のないように解決されます。詳細は、[第 33 章「分散データベース」](#)を参照してください。

追加情報： 分散トランザクションの障害からの回復の詳細は、『Oracle8i Server 分散システム』を参照してください。

データベースの読み専用モードでのオープン

データベースは、ユーザー・トランザクションによってデータの内容が変更されるのを防ぐために、読み専用モードでオープンできます。読み専用モードでは、データベース・アクセスは読み専用トランザクションに限定され、データ・ファイルや REDO ログ・ファイルに書き込むことはできません。

制御ファイル、オペレーティング・システムの監査証跡、トレース・ファイルおよびアラート・ファイルなど、他のファイルへのディスク書込みは、読み専用モードでも実行できます。データベースを読み専用モードでオープンしても、ソート操作の一時表領域には影響しません。ただし、データベースを読み専用モードでオープンしている間は、永続表領域はオフラインにすることができません。読み専用モードでは、ジョブ・キューを使用できません。

データベース回復や、REDO データを生成しないでデータベースの状態を変更する操作には、制限はありません。たとえば、読み専用モードでは、次の操作を実行できます。

- データ・ファイルのオフラインとオンラインを切り替えることができる。
- オフラインのデータ・ファイルと表領域の回復を実行できる。
- 制御ファイルは、データベースの状態の更新に引き続き使用できる。

読み専用モードは、一時レポート用データベースとして機能しているスタンバイ・データベースに役立ちます。

Oracle Parallel Server では、すべてのインスタンスはデータベースを読み書きモードまたは読み専用モードでオープンする必要があります。

追加情報： データベースを読み専用モードでオープンする方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。

データベースとインスタンスの停止

データベースとそれに対応付けられたインスタンスを停止するには、次の 3 つのステップを実行します。

1. データベースをクローズします。
2. データベースをディスマウントします。
3. インスタンスを停止します。

データベース管理者は、これらのステップを Oracle Enterprise Manager を使用して実行できます。Oracle では、インスタンスの停止時にこの 3 つのステップが自動的に実行されます。

追加情報： 『Oracle Enterprise Manager 管理者ガイド』を参照してください。

データベースのクローズ

データベースをクローズすると、SGA にあるすべてのデータベース・データと回復のためのデータが、それぞれデータ・ファイルと REDO ログ・ファイルに書き込まれます。次に、Oracle がすべてのオンライン・データ・ファイルとオンライン REDO ログ・ファイルをクローズします。(オフライン表領域のオフライン・データ・ファイルは、すでにクローズされています。オフラインになっていた表領域とそのデータ・ファイルは、この後でデータベースを再オープンしたとき、それぞれオフラインでクローズされたままになっています。)この時点でデータベースはクローズされているので、通常の操作は実行できません。データベースがクローズされても、まだマウントされていれば、制御ファイルはオープンされたままになります。

インスタンスの異常終了によるデータベースのクローズ

万一の非常時には、オープンされているデータベースのインスタンスを異常終了(中断)させて、データベースをクローズし、すぐ完全に停止させることができます。SGA のバッファにあるすべてのデータをデータ・ファイルと REDO ログ・ファイルに書き込む操作が省略されるので、このプロセスは高速に実行されます。この後、データベースをリオープンすると、Oracle がクラッシュ回復を自動的に実行します。

注意： データベースがオープンされているときにシステム・クラッシュや電源障害が発生すると、インスタンスは事実上「異常終了」するため、データベースをリオープンした時点でクラッシュ回復が実行されます。

データベースのディスマウント

データベースがクローズされると、Oracle はデータベースをディスマウントしてインスタンスからそれを切り離します。この時点ではインスタンスはコンピュータのメモリーに残留しています。

データベースをディスマウントすると、データベースの制御ファイルはクローズされます。

インスタンスの停止

データベース停止の最後のステップは、インスタンスの停止です。インスタンスを停止すると、SGA がメモリーから削除され、バックグラウンド・プロセスが停止します。

インスタンスの異常停止

異常な状況では、すべてのメモリー構造がメモリーから削除されない、またはバックグラウンド・プロセスの 1 つが終了されないなど、インスタンスが正常に停止しないことがあります。前のインスタンスの残骸が残っていると、その後にインスタンスを起動しようとしても、ほとんどの場合に失敗します。このような状況では、データベース管理者は、最初に、残っている前のインスタンスを削除してから新しいインスタンスを起動するか、Oracle Enterprise Manager で SHUTDOWN ABORT コマンドを発行して、新しいインスタンスの起動を強制実行することができます。

追加情報： インスタンスとデータベースの起動および停止の詳細は、『Oracle8i 管理者ガイド』を参照してください。

We must try to trust one another. Stay and cooperate.

Jomo Kenyatta

この章では、分散処理について定義し、分散処理環境で Oracle Server とデータベースのアプリケーションがどのように機能するかを説明します。このマニュアルの内容は、ほとんどすべてのタイプの Oracle データベース・システム環境に適用されます。

この章の内容は、次のとおりです。

- [Oracle クライアント / サーバー・アーキテクチャ](#)
- [分散処理](#)
- [Net8](#)
- [複数層アーキテクチャ](#)

Oracle クライアント / サーバー・アーキテクチャ

Oracle データベース・システム環境は、データベース・アプリケーションとデータベースが、フロントエンド（クライアント）部とバックエンド（サーバー）部の 2 つの部分に分かれた、クライアント / サーバー・アーキテクチャになっています。クライアントでは、データベース情報にアクセスし、キーボード、スクリーンおよびマウスなどのポインティング・デバイスを使用してユーザーと対話する、データベース・アプリケーションが実行されます。サーバーでは、Oracle ソフトウェアが実行され、Oracle データベースへの同時実行の共有データ・アクセスに必要な機能が処理されます。

クライアント・アプリケーションと Oracle を同じコンピュータで実行することもできますが、クライアント部とサーバー部を別々のコンピュータで実行し、それらのコンピュータをネットワークで接続する方がさらに効率的です。この後の各項では、Oracle クライアント / サーバー・アーキテクチャのさまざまな形態について説明します。

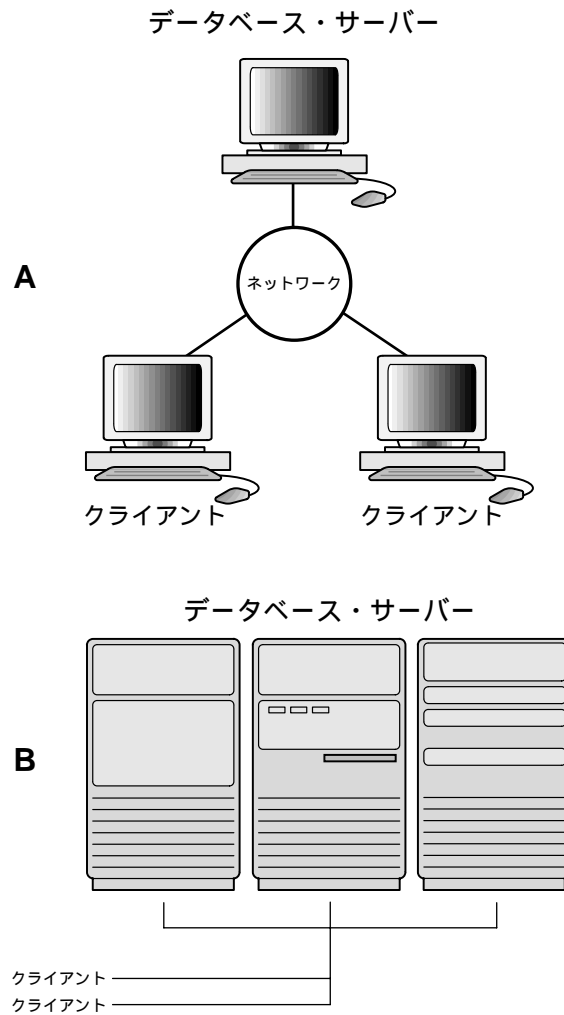
分散処理

「分散処理」とは、複数のプロセッサを使用して個々の作業を処理することです。[図 6-1](#) に、Oracle データベース・システムでの分散処理の例を示します。

- 図の A では、クライアントとサーバーを別々のコンピュータに配置し、各コンピュータをネットワークで接続しています。Oracle データベース・システムのサーバーとクライアントは、Oracle のネットワーク・インタフェースである Net8 を介して通信します。詳細は、6-4 ページの「[Net8](#)」を参照してください。
- B では、1 つのコンピュータに複数のプロセッサが含まれており、クライアント・アプリケーションと Oracle を別々のプロセッサで実行しています。

注意： この章の内容は、1 つのサーバー上に 1 つのデータベースがある環境に適用されます。「分散データベース」では、あるサーバー（Oracle）から別のサーバー上のデータベースへのアクセスが必要となる場合があります。分散データベースでのクライアントとサーバーの詳細は、[第 33 章「分散データベース」](#)を参照してください。

図 6-1 クライアント / サーバー・アーキテクチャと分散処理



分散処理環境での Oracle クライアント / サーバー・アーキテクチャには、次のような利点があります。

- クライアント・アプリケーションではデータ処理は実行されない。そのかわり、クライアント・アプリケーションは、ユーザーに入力を要求し、必要なデータをサーバーに要求し、そのデータを分析してクライアント・ワークステーションまたは端末の表示機能（グラフィックスやスプレッドシート）を使用して表示します。
- クライアント・アプリケーションはデータの物理的な位置に依存しない。データを他のデータベース・サーバーに移動したり分散しても、アプリケーションはほとんど、またはまったく変更することなく、機能を続行できます。
- Oracle は、基礎となるオペレーティング・システムのマルチタスク機能と共有メモリー機能を活用できる。その結果、クライアント・アプリケーションに最高度の同時実行性、データの整合性およびパフォーマンスが提供されます。
- クライアント・ワークステーションや端末はデータの表示用に最適化でき（グラフィックスやマウスのサポート提供など）、サーバーはデータの処理と格納用に最適化できる（大容量のメモリーとディスク領域の搭載など）。
- ネットワーク化された環境では、安価なクライアント・ワークステーションを使用して、サーバーのリモート・データに効率的にアクセスできる。
- Oracle は、システムの拡大とともに必要に応じて「拡張」できる。複数のサーバーを追加して、データベース処理の負荷をネットワーク全体に分散させたり（「水平拡張」）、Oracle をミニコンピュータやメインフレームなどに移動して、より大規模なシステムのパフォーマンス上の利点を活用（「垂直拡張」）できます。どちらの場合も、Oracle にはシステム間での移植性があるので、すべてのデータとアプリケーションをほとんど、またはまったく変更することなく維持できます。
- ネットワーク化された環境の場合、共有データは、システムのすべてのコンピュータに格納されるのではなく、サーバーに格納される。このため、同時アクセスの管理が簡単で効率的になります。
- ネットワーク化された環境では、クライアント・アプリケーションからサーバーにデータベース要求を送るときに SQL 文を使用できる。サーバーが受け取った SQL 文はそのサーバーで処理され、その結果がクライアント・アプリケーションに戻されます。ネットワークを介してやり取りされるのは要求と結果のみであるため、ネットワーク通信量は最小限ですみます。

Net8

Net8 は、ネットワーク・ワークステーションとサーバーで実行されている Oracle Tools から、他のサーバーのデータをアクセス、変更、共有および格納できるようにするための、Oracle ネットワーク・インタフェースです。Net8 は、ネットワーク通信におけるプログラム・インタフェースの一部とみなされます。プログラム・インタフェースの詳細は、[第 8 章「プロセスのアーキテクチャ」](#)を参照してください。

Net8 は、広範囲のネットワークでサポートされている通信プロトコルやアプリケーション・プログラム・インタフェース（API）を使用して、Oracle に分散データベースと分散処理の機能を提供します。

- 通信プロトコルとは、ネットワークを介したデータ伝送を管理する一連の規格のことで、ソフトウェアでインプリメントされる。
- API は、ネットワークの場合、通信プロトコルを介してリモート・プロセス間通信を確立する手段を提供する、一連のサブルーチンである。

通信プロトコルにより、ネットワーク上でのデータの送受信方法が定義されます。ネットワーク環境では、Oracle サーバーは Net8 を使用してクライアント・ワークステーションや他の Oracle サーバーと通信します。Net8 は、PC LAN でサポートされるプロトコルから、最も大規模なメインフレーム・コンピュータ・システムで使用されるプロトコルまで、主要なネットワーク・プロトコルすべてをサポートしています。

Net8 を使用しなければ、アプリケーション開発者は、ネットワーク化された分散環境で実行するアプリケーションのすべての通信ルーチンを、手でコーディングしなければなりません。ネットワークのハードウェア、トポロジまたはプロトコルが変更された場合は、それに応じてアプリケーションも変更する必要があります。

しかし、Net8 を使用すれば、アプリケーション開発者は、データベース・アプリケーションにおけるネットワーク通信をサポートする作業に煩わされずに済みます。基礎となるプロトコルが変更されても、データベース管理者がわずかな変更を加えるだけですみ、アプリケーションは、変更しなくても、機能を続行できます。

Net8 の機能

Net8 ドライバにより、データベース・サーバーで実行される Oracle プロセスと、ネットワーク上の他のコンピュータで実行される Oracle Tools のユーザー・プロセスとの間のインタフェースが提供されます。

Net8 ドライバは、Oracle Tools のインタフェースから SQL 文を取り出し、それらをパッケージ化してから、サポートされている業界標準の高レベルのプロトコルまたはプログラム・インタフェースを介して Oracle に送信します。また、Oracle からの回答を取り込んでパッケージ化し、同じ高レベルの通信メカニズムを介して Oracle Tools に送り返します。これらの機能はすべて、ネットワーク・オペレーティング・システムからは独立して実行されます。

追加情報： Oracle を実行するオペレーティング・システムによっては、データベース・サーバーの Net8 ソフトウェアにドライバ・ソフトウェアが含まれており、Net8 ソフトウェアによって Oracle バックグラウンド・プロセスが起動される場合があります。詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。また、Net8 の追加情報は、『Oracle8i Net8 管理者ガイド』を参照してください。

ネットワーク・リスナー

インスタンスの起動時には、「ネットワーク・リスナー・プロセス」によって Oracle への通信経路が確立されます。ユーザー・プロセスが接続要求を出すと、リスナーは共有サーバー・プロセスと専用サーバー・プロセスのどちらを使用するかを判断し、適切な接続を確立します。サーバー・プロセスの詳細は、8-16 ページの「[マルチスレッド・サーバー構成](#)」および 8-22 ページの「[専用サーバー構成](#)」を参照してください。

また、リスナー・プロセスは、データベース間の通信経路も確立します。Oracle Parallel Server など、単一のマシンで複数のデータベースやインスタンスが稼働している場合は、「サービス名」を使用すると、インスタンスを同じマシン上の他のリスナーに自動的に登録できます。サービス名では複数のインスタンスを識別でき、インスタンスは複数のサービスに属することができます。サービスに接続するクライアントは、必要なインスタンスを指定する必要があります。

自動インスタンス登録により、複数のデータベースやインスタンスの管理に伴うオーバーヘッドが低減します。ネットワーク上の他のインスタンスのシステム識別子（SID）は、LISTENER.ORA ファイルに登録する必要があります。

初期化パラメータ SERVICE_NAMES では、インスタンスが属するサービスが識別されます。起動時に、各インスタンスは、同じサービスに属している他のインスタンスのリスナーに登録します。データベース操作中に、各サービスのインスタンスは CPU 使用と現行の接続カウントに関する情報を、同じサービスのすべてのリスナーに渡します。これにより、動的なロード・バランシングと接続フェイルオーバーが使用可能になります。

追加情報： ネットワーク・リスナーの詳細は『Oracle8i Net8 管理者ガイド』、Oracle Parallel Server でのインスタンス登録とクライアント / サービス接続の詳細は『Oracle8i Parallel Server 概要および管理』を参照してください。

複数層アーキテクチャ

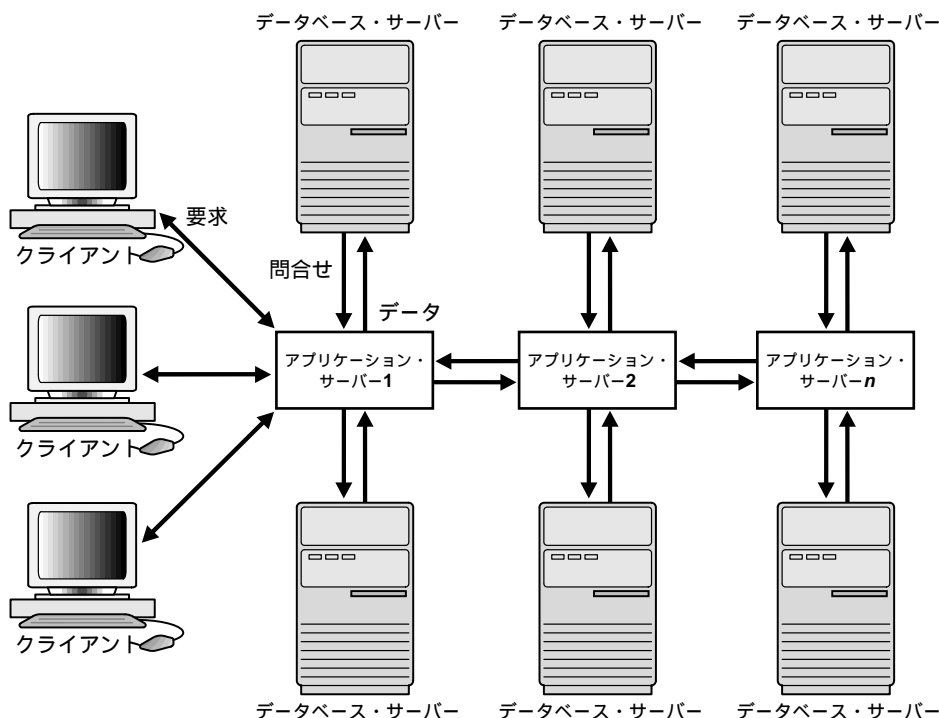
複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントとデータベース・サーバー間のインタフェースとして機能します。

このアーキテクチャでは、アプリケーション・サーバーを使用して次のことを実行できます。

- Web ブラウザなど、クライアントの資格証明を検証する。
- データベース・サーバーに接続する。
- 要求された操作を実行する。

[図 6-2](#) は、複数層アーキテクチャの例を示しています。

図 6-2 複数層アーキテクチャ環境の例



クライアント

クライアントは、データベース・サーバー上で実行される操作について要求を出します。このクライアントは、Web ブラウザでも他のエンド・ユーザー・プロセスでもかまいません。複数層アーキテクチャでは、クライアントは1つ以上のアプリケーション・サーバーを介してデータベース・サーバーに接続します。

アプリケーション・サーバー

アプリケーション・サーバーは、クライアントにデータ・アクセスを提供します。また、クライアントと、さらに高度なセキュリティ・レベルを提供する1つ以上のデータベース・サーバーとの間のインタフェースとして機能します。さらに、クライアントのために一部の問合せ処理も実行するため、データベース・サーバーの負荷がある程度軽減されます。

アプリケーション・サーバーは、クライアントのためにデータベース・サーバー上で操作を実行するとき、そのクライアントの認証を呈します。アプリケーション・サーバーの権限は、クライアント操作中に不要な操作や望ましくない操作を実行できないように制限されます。

データベース・サーバー

データベース・サーバーは、クライアントにかわってアプリケーション・サーバーから要求されたデータを提供します。残りのすべての問合せ処理は、データベース・サーバーによって実行されます。

Oracle データベース・サーバーは、個々のクライアントのかわりにアプリケーション・サーバーが実行した操作や、アプリケーション自体のために実行した操作を監査できます。たとえば、クライアント操作がクライアントに表示する情報の要求であったり、アプリケーション・サーバーの操作がデータベース・サーバーへの接続要求であったりします。

複数層環境におけるセキュリティの注意点の詳細は、29-8 ページの「[複数層の認証と許可](#)」を参照してください。

メモリー・アーキテクチャ

Yea, from the table of my memory I'll wipe away all trivial fond records.

Shakespeare: *Hamlet*

この章では、Oracle インスタンスのメモリー・アーキテクチャについて説明します。この章の内容は次のとおりです。

- Oracle メモリー構造の概要
- システム・グローバル領域 (SGA)
- プログラム・グローバル領域 (PGA)
- ソート領域
- 仮想メモリー
- ソフトウェア・コード領域

Oracle メモリー構造の概要

Oracle は、メモリーを使用してさまざまな情報を格納します。

- 実行中のプログラム・コード
- 現在はアクティブかどうかを問わず、接続されているセッションについての情報
- プログラムの実行時に必要な情報（行をフェッチしている問合せの現在の状態など）
- Oracle プロセス間で共有され、やり取りされる情報（ロック情報など）
- 周辺メモリーにも永続的に格納されるキャッシュ・データ（たとえば、データ・ブロックや REDO ログ・エントリ）

Oracle に関連する基本的なメモリー構造は、次のような領域で構成されます。

- [ソフトウェア・コード領域](#)
- [システム・グローバル領域 \(SGA\)](#) :
 - データベース・バッファ・キャッシュ
 - REDO ログ・バッファ
 - 共有プール
- [プログラム・グローバル領域 \(PGA\)](#) :
 - スタック領域
 - データ領域

システム・グローバル領域 (SGA)

システム・グローバル領域 (SGA) とは、共有メモリー構造のグループで、1 つの Oracle データベース・インスタンスのためのデータと制御情報が含まれています。複数のユーザーが同じインスタンスに同時に接続している場合、そのインスタンスの SGA 内のデータは複数のユーザー間で「共有」されます。このため、SGA は「共有グローバル領域」と呼ばれることもあります。

5-2 ページの「[Oracle インスタンスの概要](#)」で説明したように、SGA と Oracle プロセスが Oracle インスタンスを構成します。インスタンスを起動すると、Oracle が自動的に SGA 用のメモリーを割り当て、インスタンスを停止すると、オペレーティング・システムがそのメモリーの割当てを解除します。各インスタンスには、それぞれ専用の SGA があります。

SGA は読み書き可能です。マルチ・プロセス・データベースのインスタンスに接続しているすべてのユーザーは、そのインスタンスの SGA に含まれている情報を読み込むことができます。また、Oracle の稼動中に、複数のプロセスが SGA に書き込むことができます。

SGA には、次のようなデータ構造が含まれています。

- データベース・バッファ・キャッシュ
- REDO ログ・バッファ
- 共有プール
- 大規模プール (オプション)
- データ・ディクショナリ・キャッシュ
- その他の情報

SGA の一部には、バックグラウンド・プロセスがアクセスする必要のある、データベースとインスタンスの状態に関する一般的な情報が含まれています。この部分を「固定 SGA」と呼びます。ここには、ユーザー・データは格納されません。SGA には、ロック情報などのプロセス間でやりとりされる情報も格納されます。

システムがマルチスレッド・サーバー・アーキテクチャを使用している場合、要求キューと応答キュー、およびプログラム・グローバル領域の内容の一部は SGA に格納されます。(7-13 ページの「[プログラム・グローバル領域 \(PGA\)](#)」および 8-17 ページの「[ディスパッチャの要求キューと応答キュー](#)」を参照してください。)

データベース・バッファ・キャッシュ

データベース・バッファ・キャッシュは SGA の一部であり、データ・ファイルから読み込んだデータ・ブロックのコピーが保持される領域です。インスタンスに同時接続しているすべてのユーザー・プロセスは、データベース・バッファ・キャッシュへのアクセスを共有します。

データベース・バッファ・キャッシュと共有 SQL キャッシュは、論理的にセグメント化されて複数の集合になります。このように複数の集合へ編成することにより、マルチ・プロセス・システム上での競合が減少します。

データベース・バッファ・キャッシュの編成

キャッシュ内のバッファは、使用済みリストおよび最低使用頻度 (LRU) リストという 2 つのリストに編成されます。「使用済みリスト」は、「使用済みバッファ」を保持します。使用済みバッファとは、修正されたが、まだディスクに書き込まれていないデータを含むバッファのことです。「最低使用頻度 (LRU) リスト」は、空きバッファ、使用中バッファおよび使用済みリストに移動していない使用済みバッファを保持します。「空きバッファ」は、修正されておらず使用可能なバッファです。「使用中バッファ」は、現在アクセスされているバッファです。

Oracle プロセスがバッファにアクセスするとき、プロセスはそのバッファを LRU リストの最高使用頻度 (MRU) 側に移動します。さらに多くのバッファが LRU リストの MRU 側へ移動されるにつれて、使用済みバッファは LRU リストの LRU 側に向かって「古く」なります。

Oracle ユーザー・プロセスでデータの特定の部分が初めて必要になると、プロセスはデータベース・バッファ・キャッシュ内のデータを検索します。キャッシュ内にデータが見つかった場合（「キャッシュ・ヒット」）、プロセスはデータをメモリーから直接読み込むことができます。キャッシュ内にデータが見つからなかった場合（「キャッシュ・ミス」）、プロセスはデータにアクセスする前に、ディスク上のデータ・ファイルからキャッシュ内のバッファにデータ・ブロックをコピーする必要があります。キャッシュ・ヒットによるデータのアクセスは、キャッシュ・ミスによるデータのアクセスよりも高速です。

プロセスはキャッシュ内にデータ・ブロックを読み込む前に、空きバッファを見つける必要があります。プロセスは、LRU リストの最低使用頻度側から検索を開始します。空きバッファが見つかるか、検索したバッファ数が制限しきい値に達するまで、プロセスは検索を続けます。

ユーザー・プロセスが LRU リストの検索時に使用済みバッファを見つけた場合、プロセスはそのバッファを使用済みリストに移動してから検索を続けます。空きバッファが見つかったら、プロセスはディスクからバッファにデータ・ブロックを読み込んで、バッファを LRU リストの MRU 側に移動します。

空きバッファが見つからず、検索したバッファ数が制限しきい値に達すると、Oracle ユーザー・プロセスは LRU リストの検索を停止し、使用済みバッファの一部をディスクに書き込むように、DBW0 バックグラウンド・プロセスに信号を送ります。DBW0 プロセス（または複数の DBWn プロセス）の詳細は、8-8 ページの「[データベース・ライター \(DBWn\)](#)」を参照してください。

LRU アルゴリズムと全表走査

ユーザー・プロセスは、全表走査を実行するときに、表のブロックをバッファに読み込んで LRU リストの（MRU 側ではなく）LRU 側に入れます。通常、全表走査された表は一時的に必要なので、使用頻度の高いブロックをキャッシュ内に残しておくために、全表走査された表のブロックをすぐにキャッシュから移動する必要があるからです。

表走査に関連するブロックのこのデフォルトの動作は、表ごとに制御できます。全表走査時に表のブロックをリストの MRU 側に格納するように指定するには、表やクラスタの作成時または変更時に CACHE 句を使用します。それ以後の表へのアクセスで I/O を防止するために、小さな参照表や大きな静的履歴表にこの動作を指定することがあります。

追加情報： CACHE 句の詳細は、『Oracle8i SQL リファレンス』を参照してください。

データベース・バッファ・キャッシュのサイズ

初期化パラメータ DB_BLOCK_BUFFERS によって、データベース・バッファ・キャッシュ内のバッファ数を指定します。キャッシュ内の各バッファは、（初期化パラメータ DB_BLOCK_SIZE で指定される）1 つの Oracle データ・ブロックと同じサイズです。したがって、キャッシュ内の各データベース・バッファには、データ・ファイルから読み込まれたデータ・ブロックを 1 つ保持できます。

キャッシュのサイズには制限があるため、ディスク上のすべてのデータをキャッシュに入れられるとは限りません。キャッシュが満杯になった後でキャッシュ・ミスが発生すると、Oracle は新しいデータ用の領域を確保するために、すでにキャッシュ内にある使用済みデータをディスクに書き込みます。(バッファが使用済みでなければ、新しいブロックをバッファに読み込む前にディスクに書き込む必要はありません。) その後、ディスクに書き込まれたデータにアクセスすると、それもキャッシュ・ミスになります。

データを要求したときにキャッシュ・ヒットになる確率は、キャッシュのサイズによって決まります。キャッシュが大きければ、要求されたデータがキャッシュに入っている可能性は高くなります。キャッシュのサイズを大きくすれば、データ要求がキャッシュ・ヒットになる確率が高くなります。

追加情報： バッファ・キャッシュの詳細は、『Oracle8i チューニング』を参照してください。

複数のバッファ・プール

異なるバッファ・プールを持つデータベース・バッファ・キャッシュを構成して、バッファ・キャッシュ内にデータを保持するか、またはデータ・ブロックの使用直後に新しいデータがバッファを使用するかを指定できます。その後、特定のスキーマ・オブジェクト(表およびクラスタ、索引およびパーティション)を適切なバッファ・プールに割り当てて、キャッシュからデータ・ブロックのエージングを行う方法を制御できます。

- KEEP バッファ・プールは、スキーマ・オブジェクトのデータ・ブロックをメモリーに保持する。
- RECYCLE バッファ・プールは、データ・ブロックが不要になると、すぐにそれをメモリーから排除する。
- DEFAULT バッファ・プールには、どのバッファ・プールにも割り当てられていないスキーマ・オブジェクトのデータ・ブロックと、明示的に DEFAULT プールに割り当てられたスキーマ・オブジェクトが含まれる。

KEEP バッファ・プールと RECYCLE バッファ・プールを構成する初期化パラメータは、BUFFER_POOL_KEEP と BUFFER_POOL_RECYCLE です。

追加情報： バッファ・プールの詳細は『Oracle8i チューニング』、STORAGE 句の BUFFER_POOL オプションの構文は『Oracle8i SQL リファレンス』を参照してください。

REDO ログ・バッファ

「REDO ログ・バッファ」とは、SGA 内の循環バッファであり、データベースに加えられた変更についての情報を保持します。この情報は、「REDO エントリ」に格納されます。REDO エントリには、INSERT、UPDATE、DELETE、CREATE、ALTER または DROP の各操作によってデータベースに加えられた変更の再構築または再実行に必要な情報が含まれます。REDO エントリは、必要に応じてデータベースの回復時に使用されます。

REDO エントリは、Oracle のサーバー・プロセスによってユーザーのメモリー領域から SGA 内の REDO ログ・バッファにコピーされます。REDO エントリは、バッファ内で連続した順次領域を占めます。バックグラウンド・プロセス LGWR は、REDO ログ・バッファをディスク上のアクティブなオンライン REDO ログ・ファイル (または REDO ログ・ファイルのグループ) に書き込みます。

追加情報： REDO ログ・バッファがディスクに書き込まれる方法の詳細は 8-9 ページの「[ログ・ライター・プロセス \(LGWR\)](#)」、オンライン REDO ログ・ファイルとグループの詳細は『Oracle8i 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。

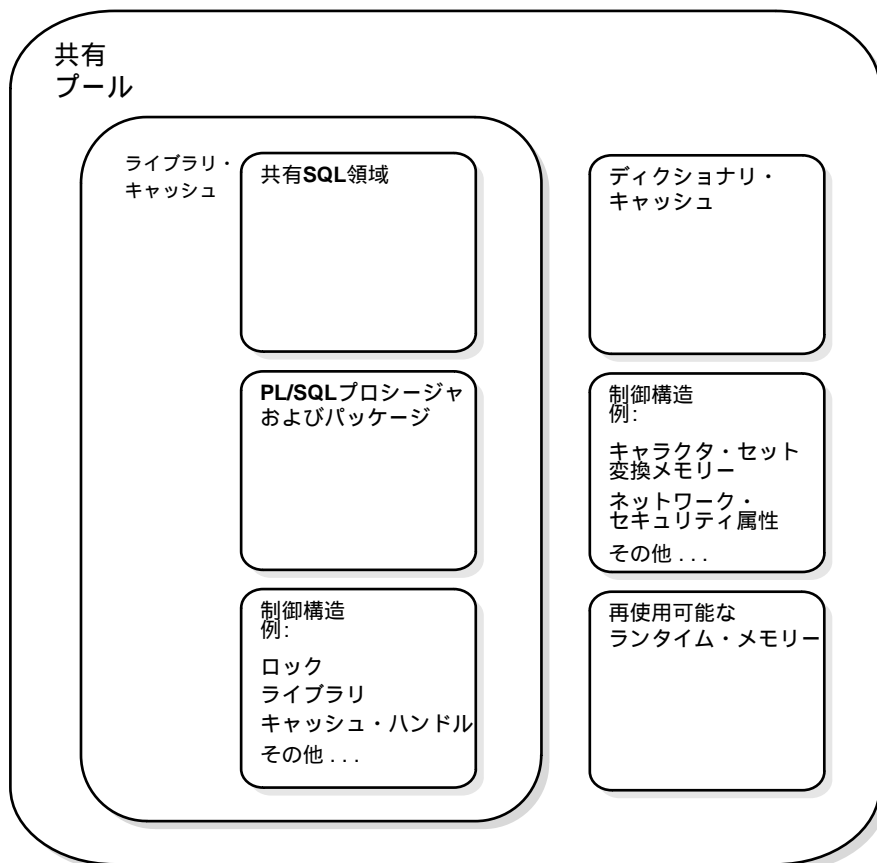
初期化パラメータ LOG_BUFFER は、REDO ログ・バッファのサイズをバイト単位で設定します。一般的に (トランザクションが長い、数が多い場合には特に) 設定する値を大きくするほどログ・ファイルの I/O は少なくなります。デフォルト設定は、ホスト・オペレーティング・システムの最大データ・ブロック・サイズの 4 倍です。

共有プール

SGA の共有プール部分には、3 つの主な領域 (ライブラリ・キャッシュ、ディクショナリ・キャッシュおよび制御構造) が含まれています。図 7-1 に、共有プールの内容を示します。

共有プールの合計サイズは、初期化パラメータ SHARED_POOL_SIZE によって決まります。このパラメータのデフォルト値は 3,500,000 バイトです。このパラメータの値を大きくすると、共有プール用に確保されるメモリーの量が増加し、したがって共有 SQL 領域用に確保される領域が増加します。

図 7-1 共有プールの内容



ライブラリ・キャッシュ

ライブラリ・キャッシュには、共有 SQL 領域、プライベート SQL 領域、PL/SQL プロシージャおよびパッケージのほか、ロックやライブラリ・キャッシュ・ハンドルなどの制御構造が含まれます。

複数のユーザーが共有 SQL 領域を使用できるようにするため、ライブラリ・キャッシュは SGA 内の共有プールに含まれます。ライブラリ・キャッシュのサイズは、データ・ディクショナリ・キャッシュと共に、共有プールのサイズによって制限されます。

共有 SQL 領域とプライベート SQL 領域

Oracle は、実行する各 SQL 文を、「共有 SQL 領域」と「プライベート SQL 領域」によって表現します。Oracle は、2 人のユーザーが同じ SQL 文を実行する場合にそれを認識し、これらのユーザーのために共有 SQL 領域を再使用します。ただし、各ユーザーは、その文のプライベート SQL 領域のコピーを各自で持つ必要があります。

共有 SQL 領域 共有 SQL 領域には、1 つの SQL 文または同一の SQL 文に関する解析ツリーと実行計画が含まれます。Oracle は、多数のユーザーが同じアプリケーションを実行するときには特に、複数の同一の DML 文に 1 つの共有 SQL 領域を使用することによってメモリーを節約します。共有 SQL 領域は、常に共有プール内にあります。

追加情報： 同一の SQL 文を判別するための基準の詳細は、『Oracle8i チューニング』を参照してください。

SQL 文を解析するときに、Oracle は共有プールからメモリーを割り当てます。このメモリーのサイズは、文の複雑度によって異なります。SQL 文が新しい共有 SQL 領域を必要とし、共有プール全体が割当て済みの場合、Oracle は、新しい文の共有 SQL 領域のための空き領域が十分になるまで、修正した最低使用頻度アルゴリズムを使用してその共有プールからエントリの割当てを解除できます。Oracle が共有 SQL 領域を割当て解除する場合、対応付けられていた SQL 文は次の実行時に再解析され、別の共有 SQL 領域に再び割り当てられます。

プライベート SQL 領域 プライベート SQL 領域は、バインド情報などのデータと実行時バッファを格納します。SQL 文を発行するそれぞれのセッションには、プライベート SQL 領域があります。同一の SQL 文を送る各ユーザーは、1 つの共有 SQL 領域を使用する自分のプライベート SQL 領域を持っています。つまり、多くのプライベート SQL 領域を、同じ共有 SQL 領域に対応付けることができます。(セッションの詳細は、8-4 ページの「[接続とセッション](#)」を参照してください。)

プライベート SQL 領域は、さらに持続領域と実行時領域に分けられます。

- 「**持続領域**」には、複数の実行にまたがって持続するバインド情報、データ型変換のコード（定義したデータ型が、選択した列のデータ型と異なる場合）およびその他の状況情報（再帰カーソルまたはリモート・カーソルの番号や、パラレル問合せの状況）が格納される。持続領域のサイズは、文で指定されているバインドおよび列の数によって異なります。たとえば、問合せで多くの列が指定されると、持続領域が大きくなります。
- 「**実行時領域**」には、SQL 文の実行時に使用する情報が含まれる。実行時領域のサイズは、実行する SQL 文のタイプと複雑度およびその文で処理する行のサイズによって異なります。通常、特に SELECT 文でソートを要求している場合（7-16 ページの「[ソート領域](#)」を参照）、実行時領域のサイズは、SELECT 文の場合よりも INSERT、UPDATE および DELETE 文の場合の方がやや小さくなります。

Oracle は、実行要求の最初のステップで実行時領域を作成します。INSERT 文、UPDATE 文および DELETE 文では、Oracle は文の実行後に実行時領域を解放します。問合せの場合、Oracle は、すべての行がフェッチされた後、または問合せが取り消された後にのみ、実行時領域を解放します。

プライベート SQL 領域の位置は、セッションのために確立される接続のタイプによって異なります。セッションが専用サーバーを介して接続されている場合、プライベート SQL 領域はユーザーの PGA 内にあります。ただし、セッションがマルチスレッド・サーバーを介して接続されている場合、持続領域および実行時領域 (SELECT 文の場合) は SGA 内にあります。

カーソルと SQL 領域 Oracle プリコンパイラ・プログラムや OCI プログラムのアプリケーション開発者は、「カーソル」、つまり特定のプライベート SQL 領域へのハンドルを明示的にオープンし、それらのカーソルをプログラムの実行中に名前付きリソースとして使用できます。Oracle が一部の SQL 文のために暗黙的に発行する再帰カーソルも共有 SQL 領域を使用します。詳細は、16-6 ページの「[カーソル](#)」を参照してください。

プライベート SQL 領域は、ユーザー・プロセスが管理します。ユーザー・プロセスが割り当てることのできるプライベート SQL 領域の数は初期化パラメータ OPEN_CURSORS によって制限されますが、プライベート SQL 領域の割当ておよび割当て解除は、使用するアプリケーション・ツールに大きく依存します。このパラメータのデフォルト値は 50 です。

プライベート SQL 領域は、対応するカーソルがクローズされるか、文ハンドルが解放されるまで存在します。Oracle は文の実行が完了した後に実行時領域を解放しますが、持続領域は待機し続けます。持続領域を解放し、アプリケーションのユーザーが必要とするメモリー容量を最小限に抑えるには、オープンされているカーソルのうち再使用しないものを、すべてアプリケーション開発者側でクローズする必要があります。

ソートを必要とする大量のデータ処理を行う問合せでは、部分的な 1 回のフェッチの結果でクライアントの要求が満たされるのであれば、アプリケーションの開発者側でその問合せを取り消す必要があります。たとえば、Oracle Office アプリケーションでは、メール・メッセージを作成するときに、ユーザーは 60 個以上のテンプレートのリストから選択できます。Oracle Office が最初の 10 個のテンプレート名を表示し、ユーザーがテンプレートの 1 つを選択した場合、アプリケーションはさらに他のテンプレート名を表示するのではなく、残りの問合せの処理を取り消す必要があります。

PL/SQL プログラム・ユニットと共有プール

Oracle は、PL/SQL プログラム・ユニット (プロシージャ、ファンクション、パッケージ、無名ブロックおよびデータベース・トリガー) を、個々の SQL 文の処理と同様の方法で処理します。プログラム・ユニットを解析およびコンパイル済みの形で保持するために、共有領域が割り当てられます。Oracle では、ローカル変数、グローバル変数、パッケージ変数 (パッケージ・インスタンスエーションとも呼ばれる) および SQL 実行用のバッファなど、プログラム・ユニットを実行するセッションに固有な値を保持するために、プライベート領域が割り当てられます。複数のユーザーが同じプログラム・ユニットを実行している場合、単一の共有領域はすべてのユーザーによって使用されますが、各ユーザーは、自分のセッションに固有の値を保持するプライベート SQL 領域のコピーを個別にメンテナンスします。

PL/SQL プログラム・ユニット内に含まれる個々の SQL 文は、先の項で説明したように処理されます。PL/SQL プログラム・ユニット内の起点には関係なく、これらの SQL 文は解析された表現を保持するために共有領域を使用し、その文を実行するセッションごとにプライベート SQL 領域を使用します。

ディクショナリ・キャッシュ

データ・ディクショナリとは、データベース、データベースの構造およびそのユーザーについての参照情報を含むデータベースの表およびビューの集合のことです。Oracle は、SQL 文の解析時にデータ・ディクショナリに頻繁にアクセスします。このアクセスは、Oracle の操作を続けるために不可欠です。詳細は、[第 2 章「データ・ディクショナリ」](#)を参照してください。

データ・ディクショナリは Oracle によって頻繁にアクセスされるので、ディクショナリ・データを保持するため、メモリー内に 2 箇所の特別な場所が指定されています。一方の領域は、データのブロック全体を保持するバッファのかわりに行としてデータを保持するので、「データ・ディクショナリ・キャッシュ」または「行キャッシュ」と呼ばれます。ディクショナリ・データを保持するメモリー内の他方の領域は、ライブラリ・キャッシュです (7-7 ページの「[ライブラリ・キャッシュ](#)」を参照)。すべての Oracle ユーザー・プロセスは、データ・ディクショナリ情報にアクセスするために、これら 2 つのキャッシュを共有します。

共有プール内のメモリーの割当てと再使用

一般に、共有プール内のあらゆる項目 (共有 SQL 領域やディクショナリ行) は、修正 LRU アルゴリズムに基づいてフラッシュされるまでは、そのまま存在します。新しい項目に領域を割り当てるために共有プール内にさらに領域が必要になると、定期的には使用されていない項目のメモリーが解放されます。修正 LRU アルゴリズムを使用することにより、多数のセッションが使用している共有プール項目は、その項目を作成したプロセスが終了しても、その項目が使用されている限りメモリーに残ります。その結果、マルチ・ユーザー Oracle システムに関連する SQL 文のオーバーヘッドと処理が、最小限に抑えられます。

SQL 文が実行のために Oracle に送られると、Oracle は次のメモリー割当てステップを自動的に実行します。

1. 同一の文について共有 SQL 領域がすでに存在しているかどうかを確認するため、共有プールをチェックします。その文のための共有 SQL 領域がすでに存在する場合、その領域は、その文の後続の新しいインスタンスを実行するために使用されます。一方、その文のための共有 SQL 領域が存在しない場合、Oracle は新しい共有 SQL 領域を共有プール内に割り当てます。どちらの場合も、ユーザーのプライベート SQL 領域が、その文を含む共有 SQL 領域に対応付けられます。

注意： 共有 SQL 領域がオープンしているカーソルに対応していても、しばらく使用されていないカーソルであれば、その共有 SQL 領域を共有プールからフラッシュできます。オープンしているカーソルが、その後その文を実行するために使用される場合、その文は再解析されて新しい共有 SQL 領域が共有プール内に割り当てられます。

2. セッションのためにプライベート SQL 領域を割り当てます。プライベート SQL 領域の正確な位置は、セッションのために確立された接続によって異なります (7-8 ページの「[共有 SQL 領域とプライベート SQL 領域](#)」を参照してください)。

Oracle は次のような場合にも共有 SQL 領域を共有プールからフラッシュします。

- 表、クスタまたは索引の統計を更新または削除するために ANALYZE コマンドを使用する場合、分析されたスキーマ・オブジェクトを参照する文を含んでいるすべての共有 SQL 領域は、共有プールからフラッシュされます。フラッシュされた文を次に実行するときには、その文はスキーマ・オブジェクトの新しい統計を反映するように新しい共有 SQL 領域で解析されます。
- スキーマ・オブジェクトが SQL 文で参照され、なんらかの方法で修正されると、共有 SQL 領域は「無効」になり (無効のマークが付けられる) 、その文を次に実行するときには解析する必要があります。SQL 文の無効化と依存性の問題の詳細は、[第 21 章「Oracle の依存性の管理」](#)を参照してください。
- データベースのグローバル・データベース名を変更すると、すべての情報が共有プールからフラッシュされます。
- 管理者はインスタンス起動後のパフォーマンス (データ・バッファ・キャッシュではなく、共有プールに関するパフォーマンス) を査定するために、共有プール内のすべての情報を手動でフラッシュできます。こうすれば、現行のインスタンスを停止する必要がありません。

大規模プール

データベース管理者は、次の目的で大量のメモリーを割り当てるために、「大規模プール」と呼ばれるオプションのメモリー領域を構成できます。

- マルチスレッド・サーバーと Oracle XA インタフェース用のセッション・メモリー
- I/O サーバー・プロセス
- Oracle のバックアップおよびリストア操作

Oracle は、マルチスレッド・サーバー用 (8-16 ページの「[マルチスレッド・サーバー構成](#)」を参照) または Oracle XA 用に、大規模プールからセッション・メモリーを割り当てることによって、主として共有 SQL のキャッシュとその圧縮によるパフォーマンス上のオーバーヘッドを回避する目的で、共有プールを使用できます。

追加情報： Oracle XA の詳細は、『Oracle8i 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。

Oracle のバックアップおよびリストア操作と I/O サーバー・プロセス用のメモリーは、数 100KB のバッファ内で割り当てられます。大規模プールの方が、共有プールよりも適切に、これらの要求を満たすことができます。

大規模プールには、LRU リストはありません。これに対して、共有プール内で確保されている領域では、その共有プールから割り当てられる他のメモリーと同じ LRU リストが使用されます。

追加情報： 大規模プール、共有プール内の領域確保および I/O サーバー・プロセスの詳細は、『Oracle8i チューニング』を参照してください。

SGA のサイズ

SGA のサイズは、インスタンスの起動時に決定されます。多くのシステムで最適なパフォーマンスを実現するには、SGA 全体が実メモリーに入らなければなりません。SGA 全体が実メモリーに収まらず、その一部を格納するのに仮想メモリーが使用される場合（7-17 ページの「**仮想メモリー**」を参照）オペレーティング・システムが SGA の一部についてページング（ディスクの読み書き）を実行するため、データベース・システム全体のパフォーマンスがかなり低下する可能性があります。SGA 内のすべての共有領域に専用に割り当てられるメモリーの量も、パフォーマンスに影響します。詳細は、『Oracle8i チューニング』を参照してください。

SGA のサイズは、いくつかの初期化パラメータによって決定されます。次のパラメータは、SGA サイズに最も影響を与えます。

DB_BLOCK_SIZE	1 つのデータ・ブロックおよびデータベース・バッファのサイズ（単位はバイト）。
DB_BLOCK_BUFFERS	SGA に割り当てられるデータベース・バッファの数。それぞれのバッファのサイズは、DB_BLOCK_SIZE で指定したサイズです。 SGA 内のデータベース・バッファ・キャッシュに割り当てられる領域の総量は、「DB_BLOCK_SIZE × DB_BLOCK_BUFFERS」になります。
LOG_BUFFER	REDO ログ・バッファに割り当てられるバイト数。
SHARED_POOL_SIZE	共有 SQL 文と共有 PL/SQL 文に割り当てられる領域のサイズ（単位はバイト）。

Oracle Enterprise Manager（または SQL*Plus）を使用しているときには、インスタンスの SGA に割り当てられたメモリーがインスタンスの起動時に表示されます。SQL*Plus コマンド SHOW で SGA オプションを指定して、現行のインスタンスの SGA サイズを表示することもできます。

追加情報： Oracle Enterprise Manager で SGA のサイズを表示する方法の詳細は、『Oracle Enterprise Manager 管理者ガイド』(SQL*Plus の場合は、『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。

前述の初期化パラメータと各パラメータが SGA に与える影響の詳細は、『Oracle8i チューニング』を参照してください。オペレーティング・システム固有の情報は、該当する Oracle インストレーション・ガイドまたはユーザーズ・ガイドも参照してください。

SGA のメモリーの使用方法の制御

複数の初期化パラメータを使用して、SGA によるメモリーの使用方法を制御できます。

物理メモリー

LOCK_SGA パラメータは、SGA を物理メモリーにロックします。

SGA 開始アドレス

SHARED_MEMORY_ADDRESS パラメータと HI_SHARED_MEMORY_ADDRESS パラメータは、実行時の SGA 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しないプラットフォーム上でのみ使用します。64 ビットのプラットフォームでは、HI_SHARED_MEMORY_ADDRESS は 64 ビット・アドレスの上位 32 ビットを指定します。

拡張バッファ・キャッシュ・メカニズム

USE_INDIRECT_DATA_BUFFERS パラメータは、4GB を超える物理メモリーのサポート機能を持った 32 ビット・プラットフォームで、拡張バッファ・キャッシュ・メカニズムを使用可能にします。

追加情報： これらのパラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。使用しているオペレーティング・システム固有の情報は、該当する Oracle インストレーション・ガイドまたはユーザーズ・ガイドも参照してください。

プログラム・グローバル領域 (PGA)

プログラム・グローバル領域 (PGA) とは、1 つのプロセス (サーバーまたはバックグラウンド) のデータと制御情報が含まれるメモリー領域のことです。そのため、PGA は「プロセス・グローバル領域」と呼ばれることもあります。

PGA は、プロセスが書き込める、非共有のメモリー領域です。サーバー・プロセスごとに 1 つの PGA が割り当てられます。この PGA はそのサーバー・プロセスに対して排他的であり、そのプロセスのために活動している Oracle コードによってのみ読み書きされます。

オペレーティング・システムと構成によって異なりますが、ユーザーが Oracle データベースに接続して、セッションが確立されると、Oracle によって PGA が割り当てられます。(セッションの詳細は、8-4 ページの「[接続とセッション](#)」を参照してください。)

PGA の内容

PGA の内容は、対応するインスタンスがマルチスレッド・サーバーを実行しているかどうかによって異なります。(マルチスレッド・サーバーの詳細は、8-16 ページの「[マルチスレッド・サーバー構成](#)」を参照してください。)

スタック領域

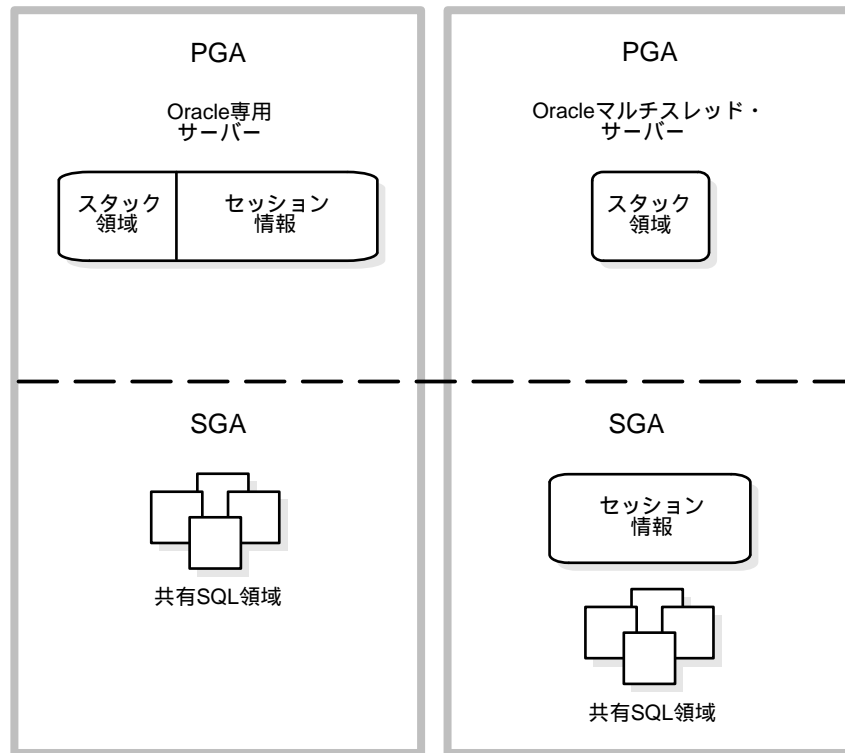
PGA には、「スタック領域」が必ず含まれます。スタック領域とは、セッションの変数、配列およびその他の情報を保持しておくために割り当てられるメモリーです。

セッション情報

インスタンスがマルチスレッド・サーバーなしで実行されている場合、PGA にはプライベート SQL 領域など、ユーザーのセッションに関する情報も格納されます。そのインスタンスがマルチスレッド・サーバー構成で実行されている場合、このセッション情報は PGA 内には存在しませんが、そのかわりに SGA 内に割り当てられます。

[図 7-2](#) に、さまざまな構成でセッション情報が格納される場所を示します。

図 7-2 マルチスレッドサーバーを含む構成と含まない構成のセッション情報の格納場所



PGA のサイズ

PGA の初期サイズは固定であり、オペレーティング・システムによって異なります。クライアントとサーバーが異なるマシン上に存在する場合、PGA は接続時にデータベース・サーバー上に割り当てられます。接続時に十分なメモリーが使用できなければ、そのオペレーティング・システムのエラーを示す範囲のエラー番号が付いた Oracle エラーが発生します。いったん接続されると、PGA 領域が使い果たされることはありません。つまり、接続時に十分なメモリーが存在しているか、メモリーが不足しているかのどちらかです。

初期化パラメータ `OPEN_LINKS` および `DB_FILES` は、PGA のサイズに影響します。Oracle バックグラウンド・プロセス (DBW0 や LGWR など) のために作成される各 PGA 内のスタック領域のサイズは、いくつかの追加パラメータの影響を受けます。

追加情報： PGA の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

ソート領域

ソートを実行するには、メモリー内に領域が必要です。Oracle がデータのソートを実行するメモリー内の部分を、「ソート領域」といいます。ソートには、それをユーザー・プロセスのために実行する Oracle サーバー・プロセスからのメモリーが使用されます。ただし、ソート領域の一部 (SORT_AREA_RETAINED_SIZE まで) は、プロセスのプライベート SQL 領域の実行時領域内に存在します。7-8 ページの「[プライベート SQL 領域](#)」で説明したように、SELECT 文の場合、プライベート SQL 領域内のこのメモリーは、接続構成に応じてさまざまな場所から取り込まれます。

- 専用サーバーを介した接続の場合は PGA
- マルチスレッド・サーバーを介した接続の場合は SGA

これらの構成の説明は、8-16 ページの「[マルチスレッド・サーバー構成](#)」および 8-22 ページの「[専用サーバー構成](#)」を参照してください。

ソート領域は、ソートされるデータの量を収容するために大きくなることがありますが、初期化パラメータ SORT_AREA_SIZE の値によって制限されます。バイト数で指定されるデフォルト値は、オペレーティング・システムによって異なります。

ソート処理中に、Oracle はソート領域内のデータを参照する必要のないタスクをいくつか実行することがあります。そのような場合、Oracle はデータの一部をディスク上の一時セグメントに書き込み、そのデータを含むソート領域の一部を割当て解除することにより、ソート領域のサイズを小さくします。たとえば、Oracle がアプリケーションに制御を戻す場合に、このような割当て解除が生じる可能性があります。

縮小後のソート領域のサイズは、初期化パラメータ SORT_AREA_RETAINED_SIZE によって決定されます。このパラメータのデフォルト値は、SORT_AREA_SIZE パラメータの値です。

ソート時に解放されたメモリーは、同一の Oracle プロセスで使用できますが、オペレーティング・システムに対して解放されるわけではありません。

ソートするデータの量がソート領域に収まらない場合、収まるように小さい断片に分割されます。続いて、各断片は個々にソートされます。個々にソートされた断片のことを、「ラン (run)」といいます。すべてのランがソートされた後に、マージされて最終結果が生成されます。

仮想メモリー

多くのオペレーティング・システム上で、Oracle は「仮想メモリー」を使用します。仮想メモリーとは、オペレーティング・システムの機能の 1 つであり、実メモリー単独の場合より多くの見かけ上のメモリーを提供することにより、主メモリーをより柔軟に使用できるようにする機能です。

仮想メモリーは、実（主）メモリーと 2 次記憶領域（通常はディスク領域）の組合せを使用して、メモリーをシミュレートします。オペレーティング・システムは、アプリケーション・プログラムに対して 2 次記憶領域を主メモリーのように見せることによって、仮想メモリーにアクセスします。

提案： 通常、実メモリー内に SGA 全体を置くと効率が最も良くなります。多くのプラットフォームでは、LOCK_SGA パラメータを使用して SGA を実メモリーにロックできます。

ソフトウェア・コード領域

「ソフトウェア・コード領域」とは、実行中または実行される可能性があるコードを格納するためのメモリー部分です。Oracle のコードはソフトウェア領域に格納されますが、これはユーザー・プログラムが格納される領域の位置とは異なる位置、つまり排他的で保護された位置になります。

ソフトウェア領域のサイズは通常固定されており、ソフトウェアの更新時か再インストール時に限り変化します。これらの領域に必要なサイズは、オペレーティング・システムによって異なります。

ソフトウェア領域は読取り専用であり、共有または非共有でインストールされます。可能なときには、メモリー内に Oracle コードの複数のコピーを持たずにすべての Oracle ユーザーが Oracle コードにアクセスできるようにするため、Oracle コードは共有されます。これにより、実際の主メモリーが節約され、全体のパフォーマンスが改善されます。

ユーザー・プログラムは共有でも非共有でもかまいません。SQL*Forms や SQL*Plus などの Oracle Tools およびユーティリティによっては、共有でインストールできるものもありますが、共有にできないものもあります。Oracle の複数のインスタンスが同じコンピュータ上で実行されている場合、それらのインスタンスは異なるデータベースにおいても同じ Oracle コード領域を使用できます。

追加情報： ソフトウェアを共有でインストールするオプションは、すべてのオペレーティング・システムで使用できるわけではありません（たとえば、MS-DOS が稼働している PC では使用できません）。詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

プロセスのアーキテクチャ

If the good people, in their wisdom, shall see fit to keep me in the background, I have been too familiar with disappointments to be very much chagrined.

Abraham Lincoln, *Address at New Salem (1832)*

この章では、Oracle データベース・システムのプロセスと、Oracle システムで使用可能な各種構成について説明します。この章の内容は次のとおりです。

- [プロセスの概要](#)
- [ユーザー・プロセス](#)
- [Oracle プロセス](#)
- [マルチスレッド・サーバー構成](#)
- [専用サーバー構成](#)
- [プログラム・インタフェース](#)

プロセスの概要

接続している Oracle ユーザーはすべて、Oracle データベース・インスタンスにアクセスするために、2 つのコード・モジュールを実行しなければなりません。

アプリケーションまたは Oracle Tools	データベース・ユーザーは、データベース・アプリケーション（プリコンパイラ・プログラムなど）や Oracle Tools（SQL*Plus など）を実行します。これらは、Oracle データベースに SQL 文を発行します。
--------------------------	---

Oracle Server コード	各ユーザーは自分のために Oracle Server コードをいくつか持ち、これがアプリケーションの SQL 文を解釈して処理します。
-------------------	---

これらのコード・モジュールは、プロセスによって実行されます。「プロセス」は「制御のスレッド」、つまり一連のステップを実行できるオペレーティング・システムのメカニズムです。（オペレーティング・システムによっては、「ジョブ」または「タスク」という用語を使用します）。プロセスには、通常、プロセスそのものを実行するための専用のプライベート・メモリー領域があります。

マルチ・プロセス Oracle システム

「マルチ・プロセス Oracle」（「マルチ・ユーザー Oracle」とも呼ばれる）は、Oracle コードの各部分とユーザーのためのその他のプロセスを実行するために、複数のプロセスを使用します。ユーザー用プロセスは、接続中のユーザーごとに個別のプロセスの場合や、複数のユーザーが共有する 1 つ以上のプロセスの場合があります。データベースの主な利点の 1 つが、複数のユーザーが同時に必要とするデータを管理できることにあるため、多くのデータベース・システムはマルチ・ユーザー・システムです。

Oracle インスタンスにおける各プロセスは、特定のジョブを実行します。Oracle とデータベース・アプリケーションの作業を複数のプロセスに分けることにより、システムの優れたパフォーマンスを維持しながら、複数のユーザーやアプリケーションが同時に 1 つのデータベース・インスタンスに接続できます。

プロセスのタイプ

Oracle システム内のプロセスは、次の 2 つの主要グループに分類できます。

- ユーザー・プロセスは、アプリケーションまたは Oracle Tool のコードを実行する（8-4 ページの「[ユーザー・プロセス](#)」を参照）。
- Oracle プロセスは、Oracle サーバーのコードを実行する。この種のプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります（8-5 ページの「[Oracle プロセス](#)」を参照）。

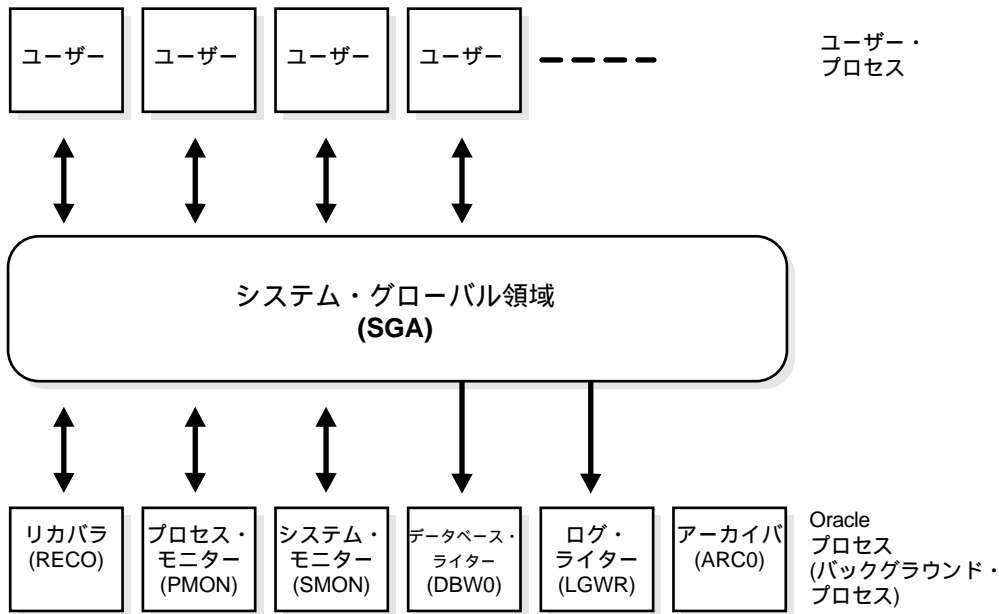
プロセス構造は、オペレーティング・システムと、選択する Oracle オプションによって、Oracle 構成ごとに異なります。接続中のユーザー用のコードは、次のどちらかの方法で構成できます。

専用サーバー (2 タスクの Oracle)	それぞれのユーザーについて、データベース・アプリケーションを実行するプロセス (ユーザー・プロセス) と、Oracle Server コードを実行するプロセス (専用サーバー・プロセス) を別々にする。8-22 ページの「 専用サーバー構成 」を参照してください。
マルチスレッド・サーバー	データベース・アプリケーションを実行するプロセス (ユーザー・プロセス) と、Oracle Server コードを実行するプロセスを別々にする。Oracle Server コードを実行する各サーバー・プロセス (「共有サーバー・プロセス」) は、複数のユーザー・プロセスを処理できます。8-16 ページの「 マルチスレッド・サーバー構成 」を参照してください。

追加情報： オペレーティング・システムによっては、構成を選択できるものもあります。このようなオプションの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

[図 8-1](#) に、専用サーバー構成を示します。接続中の各ユーザーは別々のユーザー・プロセスを持ち、複数のバックグラウンド・プロセスが Oracle を実行します。

図 8-1 マルチ・プロセス Oracle のインスタンス



この図は、Oracle システムと同じマシン上でアプリケーションを実行している複数の同時実行ユーザーを表しています。このような構成は通常、メインフレームやミニコンピュータ上で稼動します。

ユーザー・プロセス

ユーザーがアプリケーション・プログラム（Pro*C プログラムなど）または Oracle Tool（Oracle Enterprise Manager や SQL*Plus など）を実行すると、Oracle はユーザーのアプリケーションを実行するために「ユーザー・プロセス」を作成します。

接続とセッション

「接続」および「セッション」という用語は、「ユーザー・プロセス」という用語と密接に関連していますが、意味的には大きな差異があります。

「接続」とは、ユーザー・プロセスと Oracle インスタンスの間の通信路のことです。通信路は、使用可能なプロセス間通信メカニズム（1 台のコンピュータでユーザー・プロセスと Oracle の両方を実行する場合）またはネットワーク・ソフトウェア（別々のコンピュータがデータベース・アプリケーションと Oracle を実行し、ネットワークを介して通信する場合）を使用することにより確立されます。

「セッション」とは、ユーザーがユーザー・プロセスを介して Oracle インスタンスに接続するときの、特定の接続のことです。たとえば、ユーザーは、SQL*Plus を開始するときに、有効なユーザー名とパスワードを入力する必要があります。そうすることにより、そのユーザーのためのセッションが確立されます。セッションは、ユーザーが接続した時点から、接続を切断するかデータベース・アプリケーションを終了する時点まで続きます。

1 人の Oracle ユーザーに対して複数のセッションを作成し、それらを同じユーザー名で同時に存在させることができます。たとえば、SCOTT/TIGER というユーザー名 / パスワードを持つユーザーは、同じ Oracle インスタンスに何度も接続できます。

マルチスレッド・サーバーを使用していない構成では、Oracle はユーザー・セッションごとにサーバー・プロセスを 1 つずつ作成します。しかし、マルチスレッド・サーバーを使用すると、1 つのサーバー・プロセスを多数のユーザー・セッションで共有できます。詳細は、8-16 ページの「[マルチスレッド・サーバー構成](#)」を参照してください。

Oracle プロセス

この項では、Oracle Server コードを実行する 2 種類のプロセス（サーバー・プロセスとバックグラウンド・プロセス）と、Oracle プロセスのデータベース・イベントが記録されるトレース・ファイルとアラート・ファイルについて説明します。

サーバー・プロセス

Oracle では、インスタンスに接続されたユーザー・プロセスの要求を処理するために、「サーバー・プロセス」が作成されます。アプリケーションと Oracle が同一のマシン上で稼働しているときには、システムのオーバーヘッドを軽減するために、ユーザー・プロセスとそれに対応するサーバー・プロセスを 1 つのプロセスに結合できる場合もあります。しかし、アプリケーションと Oracle がそれぞれ別のマシン上で稼働している場合は、ユーザー・プロセスは常に独立したサーバー・プロセスを通じて Oracle と通信します。

各ユーザー・アプリケーションのために作成されたサーバー・プロセス（または、結合されたユーザー / サーバー・プロセスのサーバー部）は、次の 1 つ以上の操作を実行できます。

- アプリケーションを介して発行された SQL 文を解析し、実行する。
- 必要なデータ・ブロックが SGA 内に存在しない場合、そのブロックをディスク上のデータ・ファイルから SGA の共有データベース・バッファに読み込む。
- アプリケーションが情報を処理できるような方法で結果を戻す。

バックグラウンド・プロセス

パフォーマンスを最大にし、多数のユーザーが使用できるように、マルチ・プロセス Oracle システムでは、この他に「バックグラウンド・プロセス」という複数の Oracle プロセスを使用します。

1 つの Oracle インスタンスは、多数のバックグラウンド・プロセスを持つことができます。ただし、常にすべてが存在するわけではありません。Oracle インスタンスのバックグラウンド・プロセスには、次のものがあります。

- データベース・ライター (DBW0 または DBW n)
- ログ・ライター (LGWR)
- チェックポイント (CKPT)
- システム・モニター (SMON)
- プロセス・モニター (PMON)
- アーカイバ (ARC n)
- リカバラ (RECO)
- ロック (LCK0)
- ジョブ・キュー (SNP n)
- キュー・モニター (QMNP n)
- ディスパッチャ (Dnnn)
- サーバー (Snnn)

多くのオペレーティング・システムでは、インスタンスが起動するときに、バックグラウンド・プロセスが自動的に作成されます。

追加情報： プロセスが作成される方法の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

図 8-2 に、Oracle データベースの各部分とそれぞれのバックグラウンド・プロセスとの相互作用を示します。その後、各プロセスについて説明します。

追加情報： 図 8-2 には、Oracle Parallel Server は示されていません。詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

データベース・ライター (DBWn)

「データベース・ライター・プロセス (DBWn)」は、バッファの内容をデータ・ファイルに書き込みます。DBWn プロセスは、データベース・バッファ・キャッシュ内の変更された (使用済み) バッファをディスクに書き込みます。(7-3 ページの「[データベース・バッファ・キャッシュ](#)」を参照。) ほとんどのシステムでは 1 つのデータベース・ライター・プロセス (DBW0) があれば十分ですが、追加プロセス (DBW1 ~ DBW9) を構成して、システムでデータを頻繁に変更する場合に書き込みのパフォーマンスを改善できます。これらの追加 DBWn プロセスは、単一プロセッサ・システムでは効果がありません。

データベース・バッファ・キャッシュ内のバッファが変更されると、そのバッファには「使用済み」のマークが付きます。DBWn プロセスの主な機能は、使用済みバッファをディスクに書き込むことによって、バッファ・キャッシュを「クリーン」な状態に保つことです。ユーザー・プロセスによってバッファが使用済みの状態になると、空きバッファの数が減少します。空きバッファの数が少なすぎると、ディスクからキャッシュにブロックを読み込む必要があるユーザー・プロセスは、空きバッファを見つけることができません。ユーザー・プロセスが空きバッファを常に検出できるように、DBWn はバッファ・キャッシュを管理します。

DBWn プロセスは、最低使用頻度 (LRU) のバッファをディスクに書き込みます。最低使用頻度の使用済みバッファをディスクに書き込むことにより、DBWn は、使用頻度の高いバッファをメモリー内に保ちながら、空きバッファを検索するときのパフォーマンスを向上させます。たとえば、頻繁にアクセスされる小さな表または索引の一部であるブロックは、それらをディスクから再び読み込まなくても済むように、キャッシュ内に置かれます。LRU アルゴリズムは、バッファの内容をディスクに書き込むときに、すぐに使用されそうなデータが書き込まれてしまわないように、より頻繁にアクセスされるブロックをバッファ・キャッシュ内に保持します。

DBWn プロセスの個数は、初期化パラメータ DB_WRITER_PROCESSES で指定します。システムで複数の DBWn プロセスを使用している場合は、DB_BLOCK_LRU_LATCHES パラメータの値を調整して、各 DBWn プロセスが同じ個数のラッチ (LRU バッファ・リスト) を持つようにする必要があります。

追加情報： DB_WRITER_PROCESSES と DB_BLOCK_LRU_LATCHES の設定に関するアドバイスは、『Oracle8i チューニング』を参照してください。

DBWn プロセスは、次のような場合に使用済みバッファをディスクに書き込みます。

- サーバー・プロセスがしきい値の数までバッファを走査しても、クリーンな再使用可能バッファが見つからなければ、DBWn に書き込み信号が送られる。DBWn は、1 回の「マルチブロック書き込み」によって、使用済みバッファをディスクに書き込みます。
- DBWn は、定期的にバッファを書き込んで、REDO スレッド (ログ) 内のクラッシュまたはインスタンス回復を開始する必要がある位置 (「チェックポイント」) を進める。このログ位置は、バッファ・キャッシュ内の最も古い使用済みバッファによって決まります。詳細は、32-13 ページの「[ファースト・スタート・チェックポイント](#)」を参照してください。

どの場合でも、効率を向上させるために DBW_n はバッチ（マルチ・ブロック）書込みを実行します。マルチ・ブロック書込みで書き込まれるブロックの数は、オペレーティング・システムによって異なります。

追加情報： 単一の DBW0 プロセスまたは複数の DBW_n プロセスのパフォーマンスの監視と調整の方法は、『Oracle8i チューニング』を参照してください。

ログ・ライター・プロセス (LGWR)

「ログ・ライター・プロセス (LGWR)」は、REDO ログ・バッファ管理、つまりディスク上の REDO ログ・ファイルへの REDO ログ・バッファの書込みを行います（7-5 ページの「[REDO ログ・バッファ](#)」を参照）。LGWR は、最後の書込み以後にバッファにコピーされた REDO エントリすべてを、REDO ログ・ファイルに書き込みます。

REDO ログ・バッファは循環バッファです。LGWR が REDO ログ・バッファから REDO ログ・ファイルに REDO エントリを書き込んだ後、サーバー・プロセスはディスクに書き込まれた REDO ログ・バッファのエントリ上に新しいエントリをコピーできます。REDO ログへのアクセスが頻繁なときにも、新しいエントリを書き込めるよう常にバッファ内の領域を空けておくため、LGWR が行う書き込みは通常は非常に高速になります。

LGWR は、バッファの 1 つの連続した部分をディスクに書き込みます。LGWR は、次のものを書き込みます。

- ユーザー・プロセスがトランザクションをコミットしたときのコミット・レコード
- REDO ログ・バッファ
 - 3 秒ごとに書き込む
 - 3 分の 1 が満杯になったときに書き込む
 - DBW_n プロセスが修正済みのバッファをディスクに書き込むときに、必要に応じて書き込む

注意： DBW_n が修正済みバッファを書き込むには、バッファの変更に関連するすべての REDO レコードがディスクに書き込まれている必要があります（「事前書込みプロトコル」）。DBW_n は、いくつかの REDO レコードが書き込まれていないことを発見した場合に、REDO レコードをディスクに書き込むように LGWR に信号を送り、REDO ログ・バッファの書込みの完了を待機してからデータ・バッファを書き出します。

LGWR は、ミラー化されたアクティブなオンライン REDO ログ・ファイルのグループにも同時に書き込みます。グループ内のファイルのいずれかが破損している場合や使用できない場合、LGWR はそのグループ内の他のファイルへの書き込みを続け、このエラーのログを LGWR トレース・ファイルやシステム ALERT ファイルに記録します（8-15 ページの「[トレース・ファイルと ALERT ファイル](#)」を参照）。グループ内のすべてのファイルが破損している場合や、アーカイブされていないためにグループ全体が使用できない場合、LGWR は機能を続行できません。

ユーザーが COMMIT 文を発行すると、LGWR は REDO ログ・バッファ内にコミット・レコードを入れ、トランザクションの REDO エントリとともにそれを即時にディスクに書き込みます。対応するデータ・ブロックの変更は、それらをより効率的に書き込めるようになるまで延期されます。これを「高速コミット」メカニズムといいます。トランザクションのコミット・レコードを含む REDO エントリのアトミックな書き込みは、トランザクションがコミットされたかどうかを判別する 1 つのイベントです。Oracle は、データ・バッファがまだディスクに書き込まれていない場合でも、コミットしたトランザクションに成功コードを戻します。

注意： より多くのバッファ領域が必要な場合に、LGWR はトランザクションがコミットされる前に REDO ログ・エントリを書き込むこともあります。後でトランザクションがコミットされた場合にのみ、これらのエントリは確定します。

ユーザーがトランザクションをコミットすると、そのトランザクションには「システム変更番号（SCN）」が割り当てられます。Oracle は、このシステム変更番号を該当するトランザクションの REDO エントリとともに REDO ログに記録します。SCN は、Oracle Parallel Server 構成および分散データベースで回復操作を同期させることができるように、REDO ログに記録されます。

追加情報： SCN とその使用方法の詳細は、『Oracle8i Parallel Server 概要 および管理』および『Oracle8i 管理者ガイド』参照してください。

アクティビティが高いときには、LGWR は「グループ・コミット」を使用してオンライン REDO ログ・ファイルに書き込むこともあります。たとえば、あるユーザー・トランザクションをコミットする場合を考えます。LGWR はトランザクションの REDO エントリをディスクに書き込む必要があります。このときに、他のユーザーは COMMIT 文を発行します。しかし、LGWR は前の書き込み操作を完了するまで、他のユーザーのトランザクションをオンライン REDO ログ・ファイルに書き込んでコミットすることはできません。最初のトランザクションのエントリをオンライン REDO ログ・ファイルに書き込んだ後、待機中の（まだコミットされていない）トランザクションの REDO エントリのリスト全体を 1 回の操作でディスクに書き込むことができるので、トランザクションのエントリを個々に処理するときよりも、必要となる I/O を低減できます。そのため、ディスク I/O は最小になり、LGWR のパフォーマンスは最大になります。コミットの要求が高い頻度で続けると、REDO ログ・バッファからの（LGWR による）毎回の書き込みに複数のコミット・レコードが含まれることがあります。

追加情報： LGWR のパフォーマンスの監視およびチューニング方法の詳細は、『Oracle8i チューニング』を参照してください。

チェックポイント・プロセス (CKPT)

チェックポイントが発生すると、すべてのデータ・ファイルのヘッダーはそのチェックポイントの詳細を記録するために更新される必要があります。この処理は、CKPT プロセスによって実行されます。CKPT プロセスは、ブロックをディスクに書き込みません。この書き込みは、常に DBWn が実行します。

Oracle Enterprise Manager の System_Statistics モニターによって表示される「DBWR チェックポイント」の統計には、完了したチェックポイント要求の数が示されます。

追加情報： チェックポイント間隔を変更した場合の影響は、『Oracle8i 管理者ガイド』を参照してください。

Oracle Parallel Server の CKPT の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

システム・モニター (SMON)

「システム・モニター・プロセス (SMON)」は、インスタンスの起動時に必要に応じてクラッシュ回復を実行します。また、SMON は、使用されなくなった一時セグメントをクリーン・アップする操作と、ディクショナリ管理の表領域内で連続した使用可能エクステンツを 1 つに合せる操作を受け持ちます。ファイル読取りエラーやオフライン・エラーが原因で、クラッシュ回復またはインスタンス回復時にデッド・トランザクションがスキップされた場合、SMON はその表領域やファイルがオンラインに戻った時点で回復します。SMON は、処理が必要かどうかをチェックするために定期的に「起動」します。他のプロセスで SMON を起動する必要が検出された場合は、そのプロセスから SMON をコールできます。

Oracle Parallel Server 環境では、あるインスタンスの SMON プロセスは、障害が発生した CPU またはインスタンスについてもインスタンス回復を実行できます。

追加情報： SMON の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

プロセス・モニター (PMON)

ユーザー・プロセスが失敗すると、「プロセス・モニター (PMON)」がプロセスを回復します。PMON は、データベース・バッファ・キャッシュをクリーン・アップしたり、ユーザー・プロセスが使用していたリソースを解放したりします。たとえば、アクティブ・トランザクション表の状態をリセットしてロックを解除し、アクティブ・プロセスのリストからプロセス ID を削除します。

PMON は、ディスパッチャ・プロセスとサーバー・プロセスの状態も定期的にチェックし、消滅したサーバー・プロセスがあれば再起動します（ただし、Oracle が意図的に終了させたプロセスは除きます）。また、PMON は、インスタンスおよびディスパッチャ・プロセスに関する情報をネットワーク・リスナーに登録します。

SMON と同じように、PMON は、処理が必要かどうかをチェックするために定期的に「起動」し、別のプロセスで起動する必要が検出された場合にコールできます。

リカバラ・プロセス (RECO)

「リカバラ・プロセス (RECO)」は、分散データベース構成で使用されるバックグラウンド・プロセスであり、分散トランザクションに関連する障害を自動的に解決します。ノードの RECO プロセスは、インダウト分散トランザクションにかかわる他のデータベースに自動的に接続されます。RECO プロセスは、関係するデータベース・サーバー間の接続を再確立するときに、すべてのインダウト・トランザクションを自動的に解決し、解決されるインダウト・トランザクションに対応する行を各データベースの保留中のトランザクション表からすべて削除します。

RECO プロセスがリモート・サーバーとの接続に失敗した場合、RECO は決められた時間間隔で自動的に再接続しようとします。ただし、RECO が次の接続を試行するまで待機する時間は増えていきます（指数関数的に増加します）。

追加情報： 分散トランザクションの回復の詳細は、『Oracle8i Server 分散システム』を参照してください。

RECO プロセスは、インスタンスが分散トランザクションを許可している場合と DISTRIBUTED_TRANSACTIONS パラメータがゼロより大きい場合にのみ存在します。この初期化パラメータをゼロに設定すると、インスタンスの起動時に RECO は作成されません。

アーカイバ・プロセス (ARCn)

「アーカイバ・プロセス (ARCn)」は、オンライン REDO ログ・ファイルが満杯になると、ALTER SYSTEM SWITCH LOGFILE コマンドによって強制的なログ・スイッチが発生すると、そのファイルを指定された記憶デバイスにコピーします。ARCn プロセスが存在するのは、データベースが ARCHIVELOG モードで、かつ、自動アーカイブが使用可能になっている場合のみです（32-17 ページの「[データベースのアーカイブ・モード](#)」を参照）。

Oracle インスタンスは、最大 10 個の ARCn プロセス (ARC0 ~ ARC9) を持つことができます。LGWR プロセスは、現行の ARCn プロセス数では作業負荷を処理できなくなると、新しい ARCn プロセスを起動します。ALERT ファイルには、LGWR が新しい ARCn プロセスを開始した時刻が記録されます。（8-15 ページの「[トレース・ファイルと ALERT ファイル](#)」を参照。）

大量のデータのロード中など、アーカイブに伴う大量の作業負荷が予想される場合は、初期化パラメータ LOG_ARCHIVE_MAX_PROCESSES を使用して複数のアーカイバ・プロセスを指定できます。ALTER SYSTEM コマンドでは、このパラメータの値を動的に変更し、ARCn プロセス数を増減させることができます。ただし、データベースの作業負荷に対応しきれなくなると、必要な ARCn プロセス数がシステムによって自動的に判断され、LGWR が追加の ARCn プロセスを自動的に起動するので、このパラメータをデフォルト値の 1 から変更する必要はありません。

追加情報： オンライン REDO ログのアーカイブの詳細は、32-7 ページの「[REDO ログ](#)」および『Oracle8i 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。ARC*n* プロセスの使用方法是、オペレーティング・システム固有の Oracle マニュアルを参照してください。

ロック・プロセス (LCK0)

Oracle Parallel Server では、「ロック・プロセス」(LCK0) がインスタンス間ロックを提供します。

追加情報： このバックグラウンド・プロセスの詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

ジョブ・キュー・プロセス (SNP*n*)

分散データベース構成では、最大 36 個の「ジョブ・キュー・プロセス」(SNP0、...、SNP9、SNPA、...、SNPZ) が、表スナップショットを自動的にリフレッシュできます。これらのプロセスは定期的に起動され、自動リフレッシュがスケジュールされているスナップショットをリフレッシュします。複数のジョブ・キュー・プロセスを使用すると、プロセス間でスナップショットのリフレッシュ・タスクが共有されます。

他の Oracle バックグラウンド・プロセスとは異なり、SNP*n* プロセスの障害は、インスタンス障害の原因にはなりません。SNP*n* プロセスに障害が発生すると、Oracle はそれを再起動します。

これらのプロセスは、DBMS_JOB パッケージによって作成されるジョブ要求も実行し、キューイングされたメッセージを他のデータベースのキューに波及させます (19-3 ページの「[Oracle Advanced Queuing](#)」を参照してください)。

追加情報： このバックグラウンド・プロセスとジョブ・キューの詳細は、『Oracle8i 管理者ガイド』を参照してください。

キュー・モニター・プロセス (QMN*n*)

「キュー・モニター・プロセス」は、メッセージ・キューを監視する Oracle Advanced Queuing (Oracle AQ) のオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。これらのプロセスは、SNP*n* プロセスと同様に、プロセスの障害がインスタンス障害の原因とならない点で他の Oracle バックグラウンド・プロセスと異なります。

メッセージ・キューとキュー・モニター・プロセスの詳細は、19-3 ページの「[Oracle Advanced Queuing](#)」を参照してください。

ディスパッチャ・プロセス (Dnnn)

「ディスパッチャ・プロセス」は、ユーザー・プロセスが限定された数のサーバー・プロセスを共有できるようにすることで、マルチスレッド構成をサポートします。(8-16 ページの「[マルチスレッド・サーバー構成](#)」を参照。) マルチスレッド・サーバーでは、共有サーバー・プロセスの数がユーザーの数より少なくても済みます。そのため、特にクライアント・アプリケーションとサーバーが別々のマシン上で稼働するようなクライアント / サーバー環境では、マルチスレッド・サーバーによってより多くのユーザーをサポートできます。

1 つのデータベース・インスタンスに対して複数のディスパッチャ・プロセスを作成することができます。Oracle で使用するネットワーク・プロトコルごとに、少なくとも 1 つのディスパッチャを作成する必要があります。データベース管理者は、プロセスあたりの接続数についてのオペレーティング・システムの制限に応じて、ディスパッチャ・プロセスを適切な数だけ起動しなければなりません。また、インスタンスの実行中にディスパッチャ・プロセスを追加および削除できます。

注意： ディスパッチャに接続するそれぞれのユーザー・プロセスは、両方のプロセスが同一のマシン上で実行されている場合であっても、Net8 または SQL*Net バージョン 2 を介して接続する必要があります。

マルチスレッド・サーバー構成では、ネットワーク・リスナー・プロセスがクライアント・アプリケーションからの接続要求を待機し、それぞれのクライアント・アプリケーションをディスパッチャ・プロセスに経路指定します。クライアント・アプリケーションをディスパッチャに接続できない場合、リスナー・プロセスは専用サーバー・プロセスを起動し、クライアント・アプリケーションを専用サーバーに接続します。リスナー・プロセスは、Oracle インスタンスの一部ではなく、Oracle とともに作動するネットワーキング・プロセスの一部です。

追加情報： ネットワーク・リスナーの詳細は、8-16 ページの「[マルチスレッド・サーバー構成](#)」および『Oracle8i Net8 管理者ガイド』を参照してください。

共有サーバー・プロセス (Snnn)

マルチスレッド・サーバー構成では、各「共有サーバー・プロセス」は複数のクライアント要求を処理します。詳細は、8-19 ページの「[共有サーバー・プロセス](#)」を参照してください。

トレース・ファイルと ALERT ファイル

それぞれのサーバーとバックグラウンド・プロセスは、対応付けられた「トレース・ファイル」に書き込むことができます。プロセスによって内部エラーが検出されると、エラーについての情報がそのプロセスのトレース・ファイルに書き込まれます。内部エラーが発生して情報がトレース・ファイルに書き込まれた場合、管理者はオラクル社のカスタム・サポートに連絡してください。

追加情報： エラー・メッセージの詳細は、『Oracle8i エラー・メッセージ』を参照してください。

バックグラウンド・プロセスに対応付けられているすべてのトレース・ファイルのファイル名には、そのトレース・ファイルを生成したプロセスの名前が含まれます。唯一の例外は、ジョブ・キュー・プロセス (SNPn) が生成するトレース・ファイルです。

トレース・ファイル内の追加情報は、アプリケーションやインスタンスを調整するための手引きにもなります。バックグラウンド・プロセスは、該当する場合は、いつもトレース・ファイルにこの情報を書き込みます。しかし、インスタンスまたはセッションの初期化パラメータ SQL_TRACE が TRUE に設定されている場合に限り、サーバー・プロセスはトレース・ファイルに調整情報を書き込みます。(内部エラーに関する情報は、常にトレース・ファイルに書き込まれます。)

それぞれのセッションは、SQL_TRACE パラメータを指定した SQL コマンド ALTER SESSION を使用することにより、対応付けられたサーバー・プロセスのためにトレースのロギングを使用可能または使用禁止にできます。たとえば、次の文により、セッションのトレース・ファイルに情報が書き込まれるようになります。

```
ALTER SESSION SET SQL_TRACE = TRUE;
```

データベースには、それぞれ「ALERT ファイル」もあります。データベースの ALERT ファイルは、次のようなメッセージとエラーの履歴ログです。

- 発生したすべての内部エラー (ORA-600)、ブロック障害エラー (ORA-1578) およびデッドロック・エラー (ORA-60)
- 管理的な操作。たとえば CREATE/ALTER/DROP DATABASE/TABLESPACE/ROLLBACK SEGMENT の各 SQL 文と、Oracle Enterprise Manager または SQL*Plus 文の STARTUP、SHUTDOWN、ARCHIVE LOG、RECOVER など。
- 共有サーバーとディスパッチャ・プロセスの機能に関係するいくつかのメッセージとエラー
- スナップショットの自動リフレッシュにおけるエラー

Oracle は、これらのイベントの記録を保管するのに、オペレータのコンソール上に情報を表示するかわりに、ALERT ファイルを使用します。(多くのシステムではコンソール上にもこの情報が表示されます。) 管理操作が成功すると、タイムスタンプとともに「completed (完了)」というメッセージが ALERT ファイルに書き込まれます。

マルチスレッド・サーバー構成

マルチスレッド・サーバー構成では、ごく少数のサーバー・プロセスを多数のユーザー・プロセスが共有できます。ユーザー・プロセスはディスパッチャ・バックグラウンド・プロセスに接続し、ディスパッチャは次に利用できる共有サーバー・プロセスへクライアント要求を経路指定します。

マルチスレッド・サーバー構成の利点は、システムのオーバーヘッドが削減され、サポートできるユーザー数が増えることです。わずかな数の共有サーバー・プロセスで、多くの専用サーバー・プロセスが処理するのと同じ量の処理ができます。また、各ユーザーに必要なメモリーの容量も相対的に小さくなります。

マルチスレッド・サーバー・システムでは、さまざまなプロセスが多数必要です。

- ユーザー・プロセスをディスパッチャまたは専用サーバーに接続するネットワーク・リスナー・プロセス（リスナー・プロセスは、Oracle ではなく Net8 の一部です）
- 1 つ以上のディスパッチャ・プロセス
- 1 つ以上の共有サーバー・プロセス

マルチスレッド・サーバーでは、Net8 または SQL*Net バージョン 2 が必要です。

注意： 共有サーバーを使用するには、ユーザー・プロセスは、Oracle インスタンスと同一のマシン上で実行されている場合でも、Net8 または SQL*Net バージョン 2 を介して接続する必要があります。

インスタンスが起動されると、ネットワーク・リスナー・プロセスはユーザーが Oracle に接続するときの通信路をオープンして確立します。その後、各ディスパッチャ・プロセスは、接続要求をリスニングするアドレスをリスナー・プロセスに割り当てます。データベース・クライアントで使用するネットワーク・プロトコルごとに、最低 1 つ以上のディスパッチャ・プロセスを構成し起動する必要があります。

ユーザー・プロセスが接続を要求すると、リスナーはその要求を調べてユーザー・プロセスが共有サーバー・プロセスを使用できるかどうかを決定します。使用できる場合には、リスナー・プロセスは負荷が最も軽いディスパッチャ・プロセスのアドレスを戻し、ユーザー・プロセスはディスパッチャに直接接続します。

ユーザー・プロセスによっては、ディスパッチャと通信できない（たとえば、バージョン 2 より前の SQL*Net を使用して接続した場合）こともあるので、ネットワーク・リスナー・プロセスは、このようなユーザーをディスパッチャに接続できません。この場合、つまりユーザー・プロセスが専用サーバーを要求すると（8-20 ページの「**マルチスレッド・サーバーの限定的運用**」を参照）、リスナーは専用サーバーを作成して適切な接続を確立します。

追加情報： ネットワーク・リスナーの詳細は、『Oracle8i Net8 管理者ガイド』を参照してください。

ディスパッチャの要求キューと応答キュー

ユーザーからの要求は、ユーザーの SQL 文の一部である単一のプログラム・インタフェース・コールです。ユーザーがコールすると、そのディスパッチャが要求を「要求キュー」に入れ、次に使用可能な共有サーバー・プロセスがそこから要求を取り出します。

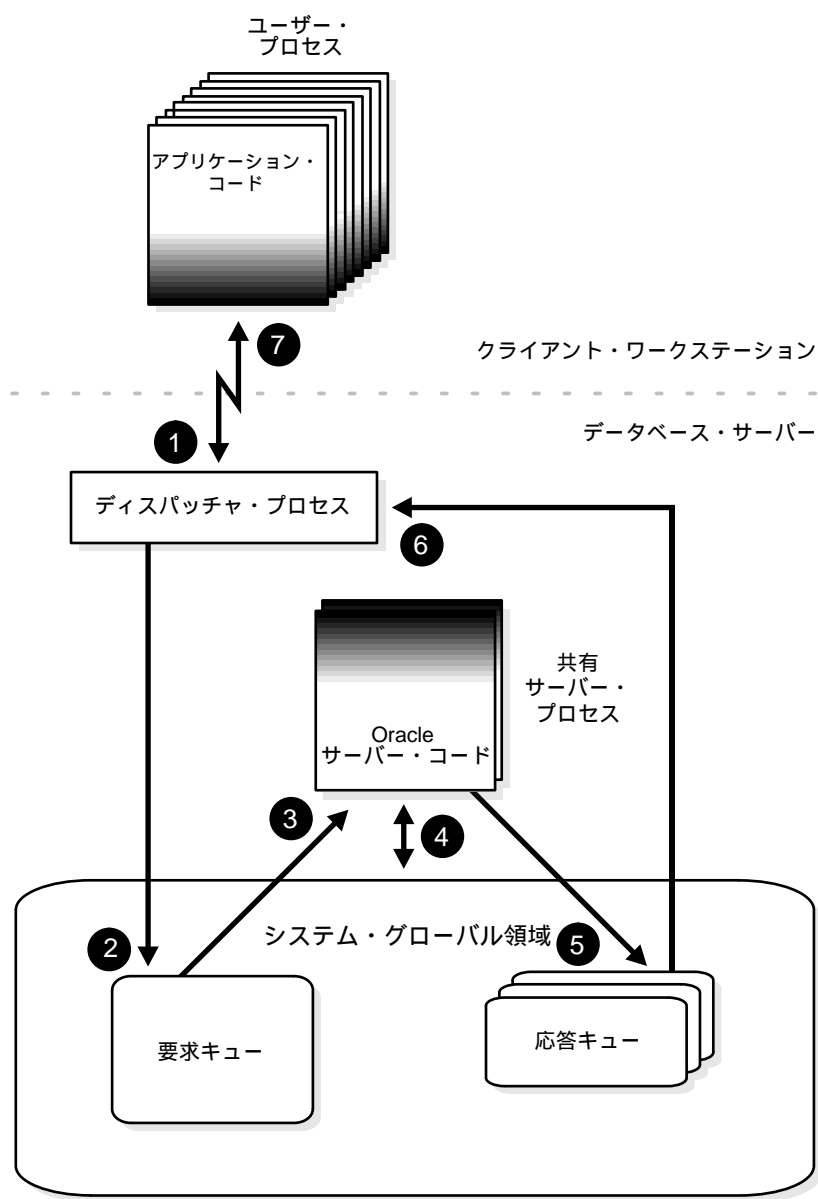
要求キューは SGA 内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。共有サーバー・プロセスは、共通の要求キューをチェックして新しい要求がないかどうかを調べ、先入れ先出し方式に基づいて新しい要求をピックアップします。1 つの共有サーバー・プロセスがキュー内の要求を 1 つピックアップし、その要求を完了するのに必要なデータベースに対するコールをすべて出します。

サーバーは、要求を完了すると、コール・ディスパッチャの「応答キュー」に応答を入れます。各ディスパッチャには、SGA 内に固有の応答キューがあります。ディスパッチャは、完了した要求を適切なユーザー・プロセスに戻します。

たとえば、注文入力システムでは、それぞれの事務担当のユーザー・プロセスがディスパッチャに接続し、事務担当が出したそれぞれの要求がそのディスパッチャに送られ、そしてディスパッチャがその要求を要求キューに入れます。次に使用可能な共有サーバー・プロセスは、要求をピックアップして処理し、応答キューに応答を入れます。事務担当の要求が完了しても、事務担当はディスパッチャに接続されたままですが、その要求を処理した共有サーバー・プロセスは解放されるので、別の要求で使用することができます。1 人の事務担当が顧客と話し合っている間に、別の事務担当は同じ共有サーバー・プロセスを使用できます。

図 8-3 に、ユーザー・プロセスがプログラム・インタフェースを介してディスパッチャと通信する方法と、ディスパッチャがユーザー要求を共有サーバー・プロセスに伝達する方法を示します。

図 8-3 マルチスレッド・サーバー構成と共有サーバー・プロセス



共有サーバー・プロセス

共有サーバー・プロセスと専用サーバー・プロセスには同じ機能がありますが、共有サーバー・プロセスは特定のユーザー・プロセスと対応付けられていません。むしろ、共有サーバー・プロセスは、マルチスレッド・サーバー構成におけるクライアント要求に対してサービスを提供します。

共有サーバー・プロセスの PGA には、(すべての共有サーバー・プロセスがアクセスする必要のある) ユーザーに関連したデータは含まれません。共有サーバー・プロセスの PGA には、スタック領域とプロセス固有の変数のみが含まれています。各種インスタンス構成における PGA の内容の詳細は、7-13 ページの「[プログラム・グローバル領域 \(PGA\)](#)」を参照してください。

セッション関連の情報はすべて SGA 内に入られます。サーバーがどのセッションからの要求でも処理できるように、すべてのセッションのデータ領域に各共有サーバー・プロセスからアクセスできなければなりません。セッションのデータ領域ごとに、SGA 内に領域が割り当てられます。ユーザーのプロファイル内のリソース制限 `PRIVATE_SGA` を必要な領域の容量に設定することにより、セッションが割り当てることのできる領域の容量を制限できます。リソース制限とプロファイルの詳細は、[第 29 章「データベース・アクセスの制御」](#)を参照してください。

Oracle は、要求キューの長さに基づいて、共有サーバー・プロセスの数を動的に調整します。作成できる共有サーバー・プロセスの数は、初期化パラメータ `MTS_SERVERS` と `MTS_MAX_SERVERS` の値で指定した範囲です。

人工デッドロック

共有サーバー・プロセスの数が制限されている場合、「人工」デッドロックが生じる可能性があります。次のような状況では、人工デッドロックになることがあります。

1. あるユーザーが、FOR UPDATE 句を指定した SELECT 文または LOCK TABLE 文を発行することによって、リソースに対する排他ロックを取得します。
2. 文の実行が完了すると、ロック要求を処理する共有サーバー・プロセスが解放されます。
3. 他のユーザーが、ロックされているリソースにアクセスしようとします。ロックされている必要なリソースが使用できるようになるまで、各共有サーバー・プロセスはユーザー・プロセスにバインドされたままです。最終的には、ロックされたリソースを待機しているユーザー・プロセスにすべての共有サーバーがバインドされる可能性があります。
4. 最初のユーザーが新しい要求 (COMMIT 文または ROLLBACK 文など) を送って、以前に取得したロックを解放しようとしますが、共有サーバー・プロセスがすべて使用中であるために送ることができません。

Oracle が人工デッドロックを検出すると、ロックされているリソース（人工デッドロックを引き起こしているリソース）を解放する要求を元のユーザーが送るまで、必要に応じて新しい共有サーバー・プロセスが自動的に作成されます。最大数の共有サーバー・プロセス（MTS_MAX_SERVERS パラメータで指定）がすでに起動されている場合、データベース管理者はユーザーの接続を切断することによってデッドロックを手動で解決しなければなりません。これにより、共有サーバー・プロセスが解放されるので、人工デッドロックが解決されます。

システム上で人工デッドロックが頻繁に発生する場合には、MTS_MAX_SERVERS の値を増やしてください。

マルチスレッド・サーバーの限定的運用

インスタンスの停止、インスタンスの起動、そしてメディア回復を含む、特定の管理アクティビティは、ディスパッチャ・プロセスに接続されている間は実行できません。ディスパッチャ・プロセスに接続されている間にこれらのアクティビティを実行しようとすると、エラー・メッセージが出されます。

これらのアクティビティは、一般的には管理者権限を使用して接続しているときに実行されます。マルチスレッド・サーバーにより構成されたシステムで管理者権限を使用して接続する場合は、ディスパッチャ・プロセスではなく専用サーバー・プロセスを使用することを接続文字列で明言してください（SRVR=DEDICATED）。

追加情報： 正しい接続文字列構文の詳細は、オペレーティング・システム固有の Oracle マニュアルまたは『Oracle8i Net8 管理者ガイド』を参照してください。

マルチスレッド・サーバーを使用する Oracle の例

次のステップは、マルチスレッド・サーバー構成で Oracle がどのように機能するかを示しています。これらのステップは、Oracle が実行する操作で最も基本的なレベルのものを示しています。

1. 現在、データベース・サーバーは、マルチスレッド・サーバー構成を使用して Oracle を実行しています。
2. クライアント・ワークステーション上のユーザー・プロセスは、SQL*Forms などのデータベース・アプリケーションを実行します。クライアント・アプリケーションは、適切な Net8 ドライバを使用して、データベース・サーバーへの接続を確立しようとします。
3. 現在、データベース・サーバー・マシンは、適切な Net8 ドライバを実行中です。データベース・サーバー上のネットワーク・リスナー・プロセスは、ユーザー・プロセスの接続要求を検出し、ユーザー・プロセスをどのように接続するかを決定します。ユーザーが Net8 または SQL*Net バージョン 2 を使用している場合、リスナーは、使用可能なディスパッチャ・プロセスのアドレスを使用して再接続するようにユーザー・プロセスに通知します。

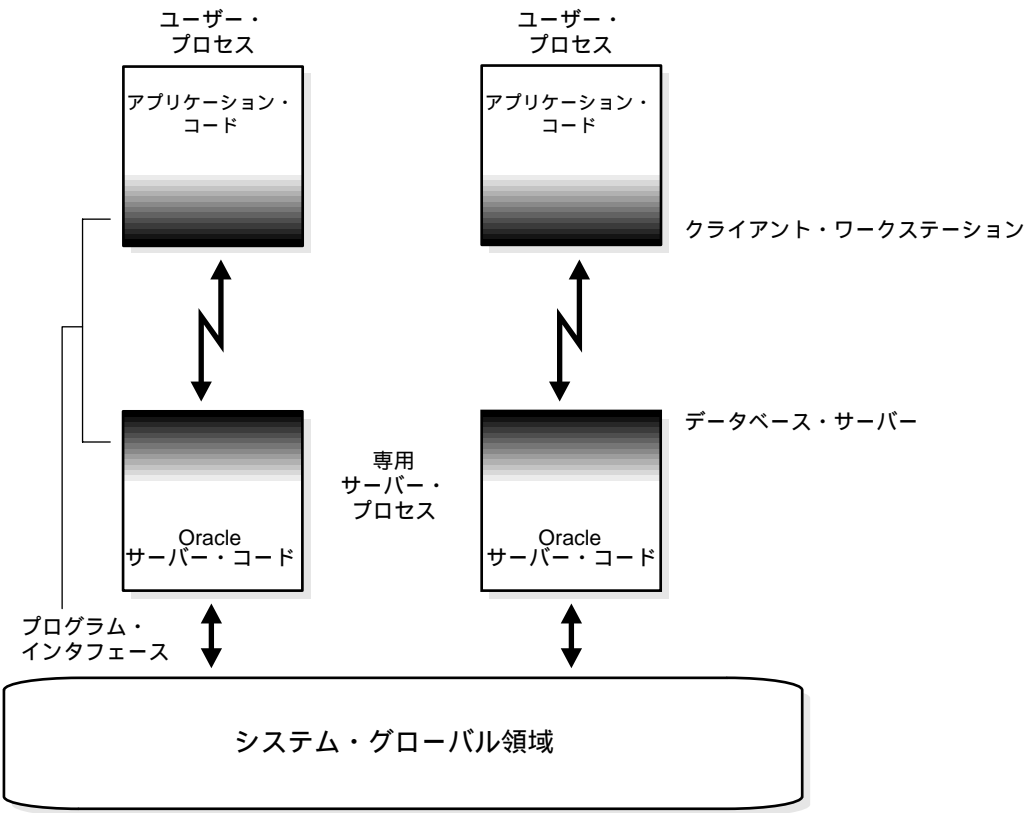
注意： ユーザー・プロセスが SQL*Net バージョン 1 または 1.1 に接続している場合には、SQL*Net のリスナーはユーザー・プロセスのために専用サーバー・プロセスを作成します。その後、後述の「専用サーバー・プロセスを使用する Oracle の例」で説明するように作動します。（ユーザー・プロセスは、共有サーバー・プロセスを使用するには Net8 または SQL*Net バージョン 2 に接続する必要があります。）

4. ユーザーが、表内の 1 行を更新するなど、1 つの SQL 文を発行します。
5. ディスパッチャ・プロセスはユーザー・プロセスの要求を要求キューに入れます。要求キューは SGA 内にあり、すべてのディスパッチャ・プロセスによって共有されています。
6. 利用可能な共有サーバー・プロセスは、共通のディスパッチャ要求キューをチェックして、キューにある次の SQL 文をピックアップします。この時点では、次の 2 つの方法で SQL 文の処理が継続されます。
 - 共有プールに同一の SQL 文のための共有 SQL 領域がある場合、サーバー・プロセスはクライアントの SQL 文を実行するために既存の共有 SQL 領域を使用します。
 - 共有プールに同一の SQL 文のための共有 SQL 領域がない場合、その文のために共有プール内に新しい共有 SQL 領域が割り当てられます。どちらの場合も、プライベート SQL 領域が（一部はセッションの PGA 内、一部は SGA 内に）作成され、共有サーバー・プロセスは、要求されたデータへのユーザーのアクセス権限をチェックします。
7. 共有サーバー・プロセスは、必要であれば、実際のデータ・ファイルからデータ・ブロックを検索します。または、すでにインスタンスの SGA にあるバッファ・キャッシュに格納されているデータ・ブロックを使用します。
8. 共有サーバー・プロセスは、共有 SQL 領域に格納されている SQL 文を実行します。データは最初に SGA 内で変更されます。DBW0 プロセスによって最も効率的だと判断された時期に、データはディスクに永続的に書き込まれます。LGWR プロセスは、ユーザーから次のコミット要求があった時点でのみ、オンライン REDO ログ・ファイル内にトランザクションを記録します。
9. 共有サーバー・プロセスは、SQL 文の処理を完了すると、要求を送ったディスパッチャ・プロセスの応答キューにその結果を入れます。
10. ディスパッチャ・プロセスは、その応答キューをチェックし、要求を出したユーザー・プロセスに完了済みの要求を送り返します。

専用サーバー構成

図 8-4 は、専用サーバー・アーキテクチャを使用し、2 台のコンピュータ上で稼働している Oracle を示しています。この構成では、データベース・アプリケーションが 1 台のマシン上のユーザー・プロセスによって実行され、対応付けられた Oracle Server がもう 1 台のマシン上のサーバー・プロセスによって実行されます。

図 8-4 専用サーバー・プロセスを使用する Oracle



ユーザー・プロセスとサーバー・プロセスは、相互に分離した別のプロセスです。ユーザー・プロセスごとに作成された別々のサーバー・プロセスは、このサーバー・プロセスが対応付けられたユーザー・プロセスのためにのみ活動するので、「専用サーバー・プロセス」（またはシャドウ・プロセス）といいます。

この構成では、ユーザー・プロセス数とサーバー・プロセス数の比率が1対1に維持されます。ユーザーが活発にデータベース要求をしていない場合でも、専用サーバー・プロセスはそのまま残ります（ただし、活動していない状態の場合、オペレーティング・システムによってはページアウトされることもある）。

図 8-4 に、ネットワークを介して接続されている別々のコンピュータ上で実行されるユーザー・プロセスとサーバー・プロセスを示します。専用サーバー・アーキテクチャは、同じコンピュータ上でクライアント・アプリケーションと Oracle Server コードの両方が実行される場合にも使用されますが、この2つのプログラムが1つのプロセスで実行されている場合は、ホスト・オペレーティング・システムはその2つの分離を維持できません。（UNIX はそのようなオペレーティング・システムの一例です。）

専用サーバー構成では、ユーザー・プロセスとサーバー・プロセスは異なるメカニズムを使用して通信します。

- ユーザー・プロセスと専用サーバー・プロセスを同じコンピュータで実行するようにシステムが構成される場合、プログラム・インタフェースは、ホスト・オペレーティング・システムのプロセス間通信メカニズムを使用してジョブを実行する。
- ユーザー・プロセスと専用サーバー・プロセスが別々のコンピュータ上で実行される場合、プログラム・インタフェースはプログラム間の通信メカニズム（ネットワーク・ソフトウェアや Net8 など）を提供する。

追加情報： このような通信リンクは、オペレーティング・システムとインストレーションに依存しています。詳細は、オペレーティング・システム固有の Oracle マニュアルと Net8 のマニュアルを参照してください。

専用サーバー・アーキテクチャによって、効率が低下する場合があります。専用サーバー・プロセスによる注文入力システムの例を考えてみます。顧客から注文が入り、事務担当がデータベースにその注文を入力します。トランザクションの大部分では、事務担当が顧客と話し合っており、事務担当のユーザー・プロセス専用のサーバー・プロセスはアイドル状態になっています。サーバー・プロセスは、トランザクションの大部分で不要であり、他の事務担当が注文を入力するときにシステムの処理速度は遅くなります。このようなアプリケーションでは、マルチスレッド・サーバー・アーキテクチャを選択してください。

専用サーバー・プロセスを使用する Oracle の例

次のステップは、専用サーバー構成で Oracle がどのように機能するかを示しています。これらのステップは、Oracle が実行する操作で最も基本的なレベルのもののみを示しています。

1. 現在、データベース・サーバー・マシンは、複数のバックグラウンド・プロセスを使用して Oracle を実行しています。
2. クライアント・ワークステーション上のユーザー・プロセスは、SQL*Plus などのデータベース・アプリケーションを実行します。クライアント・アプリケーションは、Net8 ドライバを使用して、サーバーへの接続を確立しようとします。

3. 現在、データベース・サーバーは、適切な Net8 ドライバを実行中です。データベース・サーバー上のネットワーク・リスナー・プロセスは、クライアント・データベース・アプリケーションからの接続要求を検出し、ユーザー・プロセスのためにデータベース・サーバー上に専用サーバー・プロセスを作成します。
4. ユーザーが SQL 文を 1 つ実行します。たとえば、表の中に行を挿入します。
5. 専用サーバー・プロセスはその文を受け取ります。この時点では、次の 2 つの方法で SQL 文の処理が継続されます。
 - 共有プールに同一の SQL 文のための共有 SQL 領域がある場合、サーバー・プロセスはクライアントの SQL 文を実行するために既存の共有 SQL 領域を使用する。
 - 共有プールに同一の SQL 文のための共有 SQL 領域がない場合、その文のために共有プール内に新しい共有 SQL 領域が割り当てられる。

どちらの場合も、プライベート SQL 領域はセッションの PGA 内に作成され、専用サーバー・プロセスは、要求されたデータへのユーザーのアクセス権限をチェックします。

6. サーバー・プロセスは、必要であれば、実際のデータ・ファイルからデータ・ブロックを検索します。または、すでにインスタンスの SGA にあるバッファ・キャッシュに格納されているデータ・ブロックを使用します。
7. サーバー・プロセスは、共有 SQL 領域に格納されている SQL 文を実行します。データは最初に SGA 内で変更されます。DBW0 プロセスによって最も効率的だと判断された時期に、データはディスクに永続的に書き込まれます。LGWR プロセスは、ユーザーから次のコミット要求があった時点でのみ、オンライン REDO ログ・ファイル内にトランザクションを記録します。
8. 要求が成功すると、サーバーはネットワークを介してユーザーにその旨のメッセージを送ります。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体にわたって、他のバックグラウンド・プロセスが稼働しており、介入が必要となる条件がないかどうか監視しています。さらに Oracle は、他のトランザクションを管理し、同じデータを要求する別のトランザクション間で競合が起こらないようにしています。

プログラム・インタフェース

「プログラム・インタフェース」とは、データベース・アプリケーションと Oracle の間のソフトウェア・レイヤーです。プログラム・インタフェースには次のような機能があります。

- セキュリティ・バリアを実現し、クライアント・ユーザー・プロセスによる SGA への破壊的なアクセスを防ぐ。
- 通信メカニズムとして機能し、情報要求の形式化、データの受渡し、およびエラーの検出と通知を行う。
- 特に異機種コンピュータ間、または外部ユーザー・プログラム・データ型に対しデータを変換し解釈する。

「Oracle コード」はサーバーとして機能し、データ・ブロックから行をフェッチするなど、「アプリケーション」(クライアント)のためにデータベース・タスクを実行します。Oracle コードは、Oracle ソフトウェアとオペレーティング・システム固有のソフトウェアの両方によって提供されるいくつかの部分で構成されます。

プログラム・インタフェースの構造

プログラム・インタフェースは、次の要素から構成されます。

- Oracle コール・インタフェース (OCI) または Oracle ランタイム・ライブラリ (SQLLIB)
- クライアント側またはユーザー側のプログラム・インタフェース (UPI と呼ばれる)
- 各種「Net8 ドライバ」(プロトコル固有の通信ソフトウェア)
- オペレーティング・システムの通信ソフトウェア
- サーバー側または Oracle 側のプログラム・インタフェース (OPI と呼ばれる)

ユーザー側と Oracle 側の両方のプログラム・インタフェースは、どちらもドライバのような仕組みで Oracle ソフトウェアを実行します。

Net8 は、プログラム・インタフェースの一部であり、これによってクライアント・アプリケーション・プログラムと Oracle Server が通信ネットワーク内の別々のコンピュータ上に常駐できるようになります。

プログラム・インタフェース・ドライバ

「ドライバ」とは、通常はネットワークを介してデータを転送するソフトウェアの一部です。ドライバは、接続、切断、エラーの通知、エラーのテストなどの操作を実行します。ドライバは通信プロトコルに固有のものです。デフォルト・ドライバが常に存在します。

複数のドライバ (非同期または DECnet ドライバなど) をインストールし、その 1 つをデフォルト・ドライバとして選択しますが、各ユーザーは接続の時点で必要なドライバを指定することにより、他のドライバを使用できます。プロセスが異なれば、異なるドライバを使用できます。1 つのプロセスにおいても、異なる Net8 ドライバを使用して、1 つのデータベースまたは複数のデータベース (ローカルまたはリモートのどちらでも) に同時に接続できます。

ドライバの選択とインストール、そしてインストール後の新規ドライバの追加の詳細は、使用しているシステムのインストール・ガイドおよび構成ガイド、Net8 のマニュアルに記載されています。Net8 のマニュアルでは、Oracle にアクセスしながら、実行時にドライバを選択する方法について説明しています。

追加情報： Net8 の詳細は、『Oracle8i Net8 管理者ガイド』を参照してください。

オペレーティング・システムの通信ソフトウェア

ユーザー側のプログラム・インタフェースと、Oracle 側のプログラム・インタフェースを接続している最下位層のソフトウェアは、ホスト・オペレーティング・システムによって提供される通信ソフトウェアです。たとえば、DECnet、TCP/IP、LU6.2、ASYNCR などです。

追加情報： 通信ソフトウェアはオラクル社から提供されることもありますが、通常、ハードウェア・ベンダーまたはサード・パーティーのソフトウェア会社から別個に購入します。システムの通信ソフトウェアの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

データベース・リソースの管理

Seek not, my soul, the life of the immortals; but enjoy to the full the resources that are within thy reach.

Pindar: *Pythian Odes*

この章では、データベース・リソース・マネージャを使用して、さまざまなユーザー・グループにリソースを割り当てる方法について説明します。この章の内容は、次のとおりです。

- データベース・リソース・マネージャの概要
- 資源消費グループとリソース・プラン
- 資源割当方法
- リソース・プラン・ダイレクティブ
- 例
- データベース・リソース・マネージャの使用方法

注意： この章で説明するデータベース・リソース・マネージャは、Oracle8i Enterprise Edition を購入した場合にのみ使用できます。

データベース・リソース・マネージャの概要

データベース・リソース・マネージャを使用すると、データベース管理者は、通常のようにオペレーティング・システムのリソース管理機能を単独で使用する場合よりも、きめ細かくリソースを管理できます。この機能を使用して実行できる操作は、次のとおりです。

- ユーザー・グループに、システム上の他のグループの負荷やユーザー数に関係なく、最低量の処理リソースを保証する。
- さまざまなユーザーとアプリケーションに、比率を指定して CPU 時間を割り当てて、使用可能な処理リソースを分配する。たとえば、データ・ウェアハウスでは、バッチ・ジョブよりも ROLAP アプリケーションに高い優先順位を与えることができます。
- 1 組のユーザーに許される並列度を制限する。
- 特定のリソース割当て計画を使用するようにインスタンスを構成する。データベース管理者は、インスタンスを停止して再起動しなくても、セットアップ作業を業務時間中から夜間に変更するなど、計画を動的に変更できます。

データベース・リソース・マネージャを使用するには、データベース管理者は次の事項を定義します。

資源消費グループ	同様の処理およびリソース使用要件を持ったユーザー・セッションをグループ化する手段。
リソース・プラン	リソースを消費グループ間で割り当てる手段。
資源割当て方法	特定のリソースを割り当てるときに使用する方針。資源割当て方法は、プランと消費グループの両方で使用されます。
リソース・プラン・ダイレクティブ	次の操作を行う手段。 <ul style="list-style-type: none">■ 消費グループまたはサブプランをリソース・プランに割り当てる。■ 各資源割当て方法のパラメータを指定し、プランに含まれる資源消費グループ間でリソースを割り当てる。

この後の各項では、これらの項目について詳しく説明します。

資源消費グループとリソース・プラン

資源消費グループとリソース・プランを使用して、さまざまなユーザー間で処理資源をパーティション化する方法を指定できます。現在、資源消費グループ・レベルで制御されるリソースは CPU だけです。リソース・プランは、現在は CPU および並列度の制限という 2 つのリソースの制御をサポートしています。

この項では、資源消費グループとプラン、および資源消費グループとリソース・プランを使用してリソースを制御する方法について説明します。

資源消費グループ

リソース消費を制御するために、ユーザー・セッションを「資源消費グループ」に割り当てることができます。資源消費グループでは、同様のリソース使用要件を持つ 1 組のユーザーを定義します。また、制御されるリソースごとに、資源割当方法も指定します。

資源消費グループとそれに対応付けられた属性は、データ・ディクショナリ・ビュー DBA_RSRC_CONSUMER_GROUPS に表示できます。各エントリに含まれる情報は、次のとおりです。

- 資源消費グループ名
- グループの CPU 資源割当方法
- コメント
- 状態（保留中またはアクティブ）
- 資源消費グループが必須（および、削除できない）かどうかを示す文

次に、DBA_RSRC_CONSUMER_GROUPS ビュー内の 2 つのサンプル・エントリを示します。

CONSUMER_GROUP	CPU_METHOD	COMMENTS	STATUS	MANDATORY
BUGUSERS	ROUND-ROBIN	Resource group/method for bug DB users	ACTIVE	0
PQ	ROUND-ROBIN	Resource group/method for PQ slaves	PENDING	0

各ユーザーは、デフォルトの資源消費グループを持ちます。デフォルトでは、ユーザーが所有するすべてのセッションは、そのユーザーのデフォルトの資源消費グループに属します。

ユーザーには、別の消費グループに切り替える権限を付与できます。適切な権限を持っている場合、PL/SQL プロシージャを使用すると、特定のセッションの資源消費グループを切り替えることができます。また、データベースの稼働中に、資源消費グループの資源割当方法を動的に変更することもできます。

データ・ディクショナリには、常に DEFAULT_CONSUMER_GROUP という消費グループが 1 つ存在します。どのグループにも明示的に属していないすべてのセッションは、DEFAULT_CONSUMER_GROUP に属します。

リソース・プラン

リソース割当ては、「リソース・プラン」で指定します。リソース・プランには、各資源消費グループに割り当てられるリソースを指定する「リソース・プラン・ダイレクティブ」が含まれています。

概念上は、資源消費グループの下に、各ユーザー要求の実際のセッションがあります。つまり、セッションは資源消費グループに属し、この資源消費グループをリソース・プランを使用して処理リソースの割当てが決定されます。

リソース・プランの用途は、次のとおりです。

- 資源消費グループや他のリソース・プランをグループ化する。
- グループ化した資源消費グループやプランの間でリソースをパーティション化する。
- 資源消費グループの資源割当て方法を指定する。

データベース内で複数のリソース・プランを定義し、プランごとに異なる方法で資源消費グループにリソースを割り当てることにより、リソース割当てに柔軟性を持たせることができます。ただし、1 インスタンスでアクティブ化できるのは1 プランだけです。たとえば、業務時間中のプラン、夜間のプランおよび週末のプランを定義できます。Oracle Parallel Server のインスタンスごとに、異なるリソース・プランを使用できます。

リソース・プランを階層方式で指定するには、「サブプラン」を使用します。プランをアクティブ化すると、そのすべてのサブプランもアクティブになります。

インスタンスの実行中に、最上位のアクティブ・プランを動的に切り替えることができます。これにより、さまざまな状況に合わせてリソース・プランを定義し、状況に応じてプランを変更できるようになります。

また、次の2 つの特殊な消費グループがあります。

- OTHER_GROUPS。アクティブなプラン・スキーマに含まれていない消費グループに属するすべてのセッションに適用されます。このグループ名は、常にデータ・ディクショナリに存在します。OTHER_GROUPS は、どのアクティブ・プランでも、プラン・スキーマのどこかに存在させる必要があります。

したがって、セッションが消費グループ 'C' に属し、インスタンスがプラン 'P' を使用している場合、サーバーは次のエントリを順番どおりに使用しようとしています。

1. 'P'、'C'、...
2. 'P'、'OTHER_GROUPS'、...

- DEFAULT_CONSUMER_GROUP。どのグループにも明示的に属していないすべてのセッションに適用されます。

これらの特殊な消費グループは、変更も削除もできません。

次のように、Oracle 提供の2 つの消費グループがあり、これらは環境に応じて変更または削除したり、現状のまま使用するか、使用しないかを選択できます。

- SYS_GROUP。ユーザー SYS および SYSTEM のデフォルト・グループです。
- LOW_GROUP。SYSTEM_PLAN で使用するために用意されています（表 9-1「SYSTEM_PLAN デフォルト・リソース・プラン」を参照）。スイッチ特権は、このグループの PUBLIC に付与されます。

リソース・プランとそれに対応付けられた属性は、データ・ディクショナリ・ビュー DBA_RSRC_PLANS に表示できます。各エントリに含まれる情報は、次のとおりです。

- プラン名
- プラン・ダイレクティブ数
- CPU の資源割当方法
- 並列度の制限方法
- コメント
- 状態（保留中またはアクティブ）
- プランが必須かどうかを示す文

次に、サンプル・リソース・プランのエントリを示します。

PLAN	NUM_PLAN_D	CPU_METHOD	PARALLEL_DEGREE_LIMIT_MTH	COMMENTS	STATUS	MANDATORY
MAILDB	3	EMPHASIS	PARALLEL_DEGREE_LIMIT_ABSOLUTE	Plan/method for mail users	ACTIVE	0
APPDB	3	ROUND-ROBIN	MAX_ACTIVE_SESS_ABSOLUTE	Plan/method for apps users	ACTIVE	0

リソース・プランに変更を加えると、すべてのインスタンス間で即時に有効になります。

追加情報： リソース・プランと資源消費グループに関連するデータ・ディクショナリ・ビューの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

資源割当方法

「資源割当方法」によって、データベース・リソース・マネージャが特定のリソースを割り当てるときに使用する方法や方針が決定され、これらは資源消費グループとリソース・プランの両方に使用されます。

CPU、最大並列度など、管理可能なリソースごとに 1 つずつ、デフォルトの資源割当方法が用意されています。この後の各項では、Oracle 提供のデフォルトの資源割当方法について説明します。

CPU 資源割当方法 : 強調

「強調」CPU 資源割当方法では、各消費グループのセッションに与える強調度が決定されます。これを指定するには、各消費グループに強調度を割り当てます。CPU の使用順位は、1 ~ 8 のレベルを使用して割り当てます。各レベルで CPU をパーティション化する方法は、パーセンテージで指定します。

強調資源割当方法に適用される規則は、次のとおりです。

- 上位レベルで 0% 以外の強調度が指定されている資源消費グループのセッションには、常に最初に行われる機会が与えられる。
- CPU リソースは、指定したパーセンテージに基づいた特定のレベルで分配される。資源消費グループに対して指定するリソースのパーセンテージは、その消費グループが使用可能な最大値となります。特定レベルのすべての資源消費グループに実行する機会が与えられても、CPU リソースが残っている場合、残りの CPU リソースは次の下位のレベルへと落ちます。消費グループが割り当てられたリソースを消費しなければ、そのリソースは同じレベルの他の消費グループに与えられるのではなく、次のレベルに渡されます。
- どのレベルの強調度の合計も、100 以内にする必要があります。
- 未使用の CPU 時間はリサイクルされる。つまり、割り当て量を（強調度による）即時に必要な消費グループがなければ、各消費グループにはレベル 1 から順番に割り当て量を使用する機会が再度与えられます。
- プラン・ダイレクティブが明示的に指定されていないレベルの場合、すべてのサブプラン / 消費グループについて暗黙的に 0% になる。

強調資源割当方法の長所は、次のとおりです。

- 強調度方式を使用すると、CPU をオンラインとオフラインの間で切り替えたり、サーバーを追加および削除できる。
- サーバー数に応じて CPU リソース量を比例配分しなくてもよい場合、サーバーが少数の場合にもきめ細かく制御できる。
- 強調度方式では、優先順位に関連したリソース欠乏の問題を回避できる。ユーザーは優先順位順に実行するのではなく、各自の資源消費グループに指定された強調度に基づいて実行します。また、強調度を使用して優先順位スキーマをシミュレーションできます。

最大並列度による資源割当方法 : 絶対

「並列度制限」リソース・ダイレクティブを使用すると、管理者は操作の並列度に対して制限を指定できます。このパラメータは、資源消費グループを参照するダイレクティブにのみ指定できます。最大並列度のデフォルトの資源割当方法は、絶対値です。

複数のプラン・ダイレクティブが同じサブプラン / 消費グループを参照している場合、そのサブプラン / 消費グループの配列度制限は、与えられるすべての値の中の「最小」値になります。

リソース・プラン・ダイレクティブ

リソース・プラン・ダイレクティブの用途は、次のとおりです。

- 消費グループまたはサブプランをリソース・プランに割り当てる。
- 各資源割当方法のパラメータを指定して、プランに含まれる資源消費グループ間でリソースを割り当てる。

プランのエントリごとに、リソース・プラン・ダイレクティブが1つずつ存在します。

例

この項では、資源消費グループ、リソース・プラン、資源割当方法およびリソース・プラン・ダイレクティブの使用例を示します。

資源消費グループとリソース・プランの使用方法

データベース・リソース・マネージャを使用する最初のステップは、資源消費グループとリソース・プランを使用してリソース要件を識別することです。

デフォルトのリソース・プラン SYSTEM_PLAN が用意されており、その定義は次のとおりです。

表 9-1 SYSTEM_PLAN デフォルト・リソース・プラン

エントリ	レベル1	レベル2	レベル3
SYS_GROUP	100%	0%	0%
OTHER_GROUPS	0%	100%	0%
LOW_GROUP	0%	0%	100%

SYS と SYSTEM のデフォルト消費グループは、SYS_GROUP ですが、これは変更できます。SYSTEM_PLAN は、システム・セッションに優先順位を与えます。また、SYS_GROUP および OTHER_GROUPS より下位の優先順位を持つ、グループ LOW_GROUP を指定します。どのユーザー・セッションを LOW_GROUP の一部にするかを指定する必要があります。環境に合っている場合は、この Oracle 提供の単純なプランを使用できます。

表 9-2 と表 9-3 は、BUGDB と MAILDB のサンプル・リソース・プランを示しています。

表 9-2 BUGDB サンプル・リソース・プラン

エントリ	レベル 1	レベル 2
Online 資源消費グループ	80%	0%
Batch 資源消費グループ	20%	0%
Bug_Maintenance 資源消費グループ	0%	100%

表 9-3 MAILDB サンプル・リソース・プラン

エントリ	レベル 1	レベル 2
Mailusers 資源消費グループ	0%	80%
Postman 資源消費グループ	40%	0%
Mail_Maintenance 資源消費グループ	0%	20%

BUGDB および MAILDB サンプル・リソース・プランのデータは、強調 CPU 資源割当方法によるものです。この場合、各資源消費グループに強調度を割り当てて、さまざまなグループのセッションの強調度を指定できます。

MAILDB プランで実行されるセッション数が無限であれば、Postman 資源消費グループは CPU 時間の 40% を消費し、Mailusers および Mail_Maintenance 消費グループは残りの CPU 時間を 80:20 の割合で分けます。したがって、Mailusers 資源消費グループは CPU 時間の 48%（60% の 80%）を消費し、Mail_Maintenance 資源消費グループは 12%（60% の 20%）を消費します。この例で、レベル 2 のエントリは、CPU リソースのうち最低で 60% を取得することが保証されています。ただし、Postman 資源消費グループが割当て分の 40% を使い果たすかどうかによっては、さらに割当て量が増えることがあります。

サブプランの使用方法

別のプランから参照されるリソース・プランを、「サブプラン」と呼びます。たとえば、表 9-4 は、2 つのサブプランのダイレクティブを含むプランを示しています。

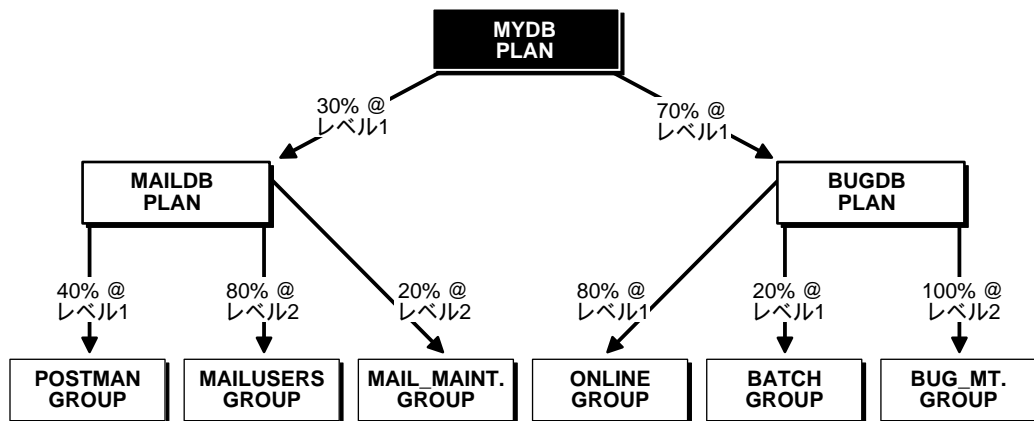
表 9-4 MYDB リソース・プラン、CPU プラン・ダイレクティブ

サブプラン / グループ	CPU_Level1
MAILDB プラン	30%
BUGDB プラン	70%

MYDB リソース・プランが有効で、すべての資源消費グループの実行可能ユーザー数が無制限であれば、MAILDB プランの有効時間は 30%、BUGDB プランの有効時間は 70% となります。

これをさらに分割すると、MAILDB プランでは Postman 資源消費グループにリソースの 40% が割り当てられ、BUGDB プランでは Online 資源消費グループにリソースの 80% が割り当てられます。Postman グループのユーザーは、この時間のうち 12% (30% の 40%) を占め、Online グループのユーザーは 56% (70% の 80%) を占めます。図 9-1 は、この使用例を示しています。

図 9-1 使用例：相互に参照するリソース・プラン



複数レベルのリソース・プランの使用方法

複数レベルのリソース・プランの方が、単一レベルのプランより強力です。資源消費グループが 1 レベルで割当てを使用しなければ、残りは次のレベルに渡され、そのレベルでの分配方法を明示的に指定できます。単一レベル・スキーマでは、未使用の時間を残りのすべての資源消費グループに一定比率で配分することしかできません。このような違いがあるため、最上位を除くレベルの強調度の合計が 100 にならなければ、複数レベル・スキーマを単一レベル・スキーマに縮小できません。

並列度制限によるリソース・ダイレクティブの使用方法

次の例で、Online グループから発行される操作の最大並列度は 0、Batch グループは 4、Bug_Maintenance グループは 4 になっています。この仕様は、並列度制限によるプラン・ダイレクティブを使用して、セッション・グループで実行中の並列操作を制限する方法を示しています。Online グループの並列度制限は 0 のため、そのすべての操作は直列で実行する必要があります。

表 9-5 最大並列度によるプラン・ダイレクティブ

サブプラン / グループ	parallel_degree_limit
Online グループ	0
Batch グループ	4
Bug_Maintenance グループ	4

まとめ

次の例では、BUGDB プランの例を使用して、前述のデフォルト資源割当方法のすべてのプラン・ダイレクティブを組み合わせています。

CPU リソース・プラン・ダイレクティブ (レベル 1 ~ 8) (明示的に指定する必要があるのは、ゼロ以外のダイレクティブを持つレベルのみであることに注意)				並列度制限による リソース・プラン・ ダイレクティブ
サブプラン / グループ	CPU_Level 1	CPU_Level 2	...	parallel_degree_limit_1
ONLINE グループ	80%	0%	0%	0
BATCH グループ	20%	0%	0%	4
BUG_MT グループ	0%	100%	0%	4

データベース・リソース・マネージャの使用方法

データベース・リソース・マネージャを使用するには、データベース管理者は次の手順で操作します。

1. PL/SQL パッケージ DBMS_RESOURCE_MANAGER を使用して、リソース・プランを作成します。
2. PL/SQL パッケージ DBMS_RESOURCE_MANAGER を使用して、資源消費グループを作成します。
3. PL/SQL パッケージ DBMS_RESOURCE_MANAGER を使用して、リソース・プラン・ダイレクティブを作成します。
4. PL/SQL パッケージ DBMS_RESOURCE_MANAGER_PRIVS を使用して、消費グループにユーザーを割り当てます。
5. インスタンスに使用されるプランを指定します。初期化パラメータ RESOURCE_MANAGER_PLAN で、特定のインスタンスに使用する最上位プランを指定します。データベース・リソース・マネージャにより、この最上位プランとそのすべての子孫 (サブプラン、ダイレクティブおよび消費グループ) がロードされます。

RESOURCE_MANAGER_PLAN パラメータを指定しなければ、データベース・リソース・マネージャは使用禁止になります。データベース管理者は、ALTER SYSTEM コマンドを使用してデータベース・リソース・マネージャを使用可能にするか（前に使用禁止にしている場合）、使用禁止にするか、または現行プランを変更して、このパラメータを動的に設定できます。

追加情報： これらの PL/SQL パッケージの使用方法は、『Oracle8i 管理者ガイド』を参照してください。

第 IV 部

オブジェクト・リレーショナル DBMS

第 IV 部では、データベースを管理するための Oracle リレーショナル・モデルと、そのモデルに対するオブジェクト拡張機能について説明します。

第 IV 部には、次の章が含まれています。

- [第 10 章「スキーマ・オブジェクト」](#)
- [第 11 章「パーティション表とパーティション索引」](#)
- [第 12 章「ビルトイン・データ型」](#)
- [第 13 章「ユーザー定義データ型」](#)
- [第 14 章「ユーザー定義データ型の使用方法」](#)
- [第 15 章「オブジェクト・ビュー」](#)

スキーマ・オブジェクト

My object all sublime I shall achieve in time—To let the punishment fit the crime.

Sir William Schwenck Gilbert: *The Mikado*

この章では、ユーザー・スキーマに含まれるさまざまな種類のデータベース・オブジェクトについて説明します。この章の内容は次のとおりです。

- [スキーマ・オブジェクトの概要](#)
- [表](#)
- [ビュー](#)
- [マテリアライズド・ビュー](#)
- [ディメンション](#)
- [シーケンス・ジェネレーター](#)
- [シノニム](#)
- [索引](#)
- [索引構成表](#)
- [アプリケーション・ドメイン索引](#)
- [クラスタ](#)
- [ハッシュ・クラスタ](#)

その他のスキーマ・オブジェクトの詳細は、33-5 ページの「[データベース・リンク](#)」、18-2 ページの「[ストアド・プロシージャとファンクション](#)」、18-11 ページの「[パッケージ](#)」および第 20 章「[トリガー](#)」を参照してください。

スキーマ・オブジェクトの概要

「スキーマ」は、各データベース・ユーザーに対応付けられます。スキーマは、スキーマ・オブジェクトの集合です。スキーマ・オブジェクトには、表、ビュー、順序、シノニム、索引、クラスタ、データベース・リンク、スナップショット、プロシージャ、ファンクションおよびパッケージなどが含まれます。

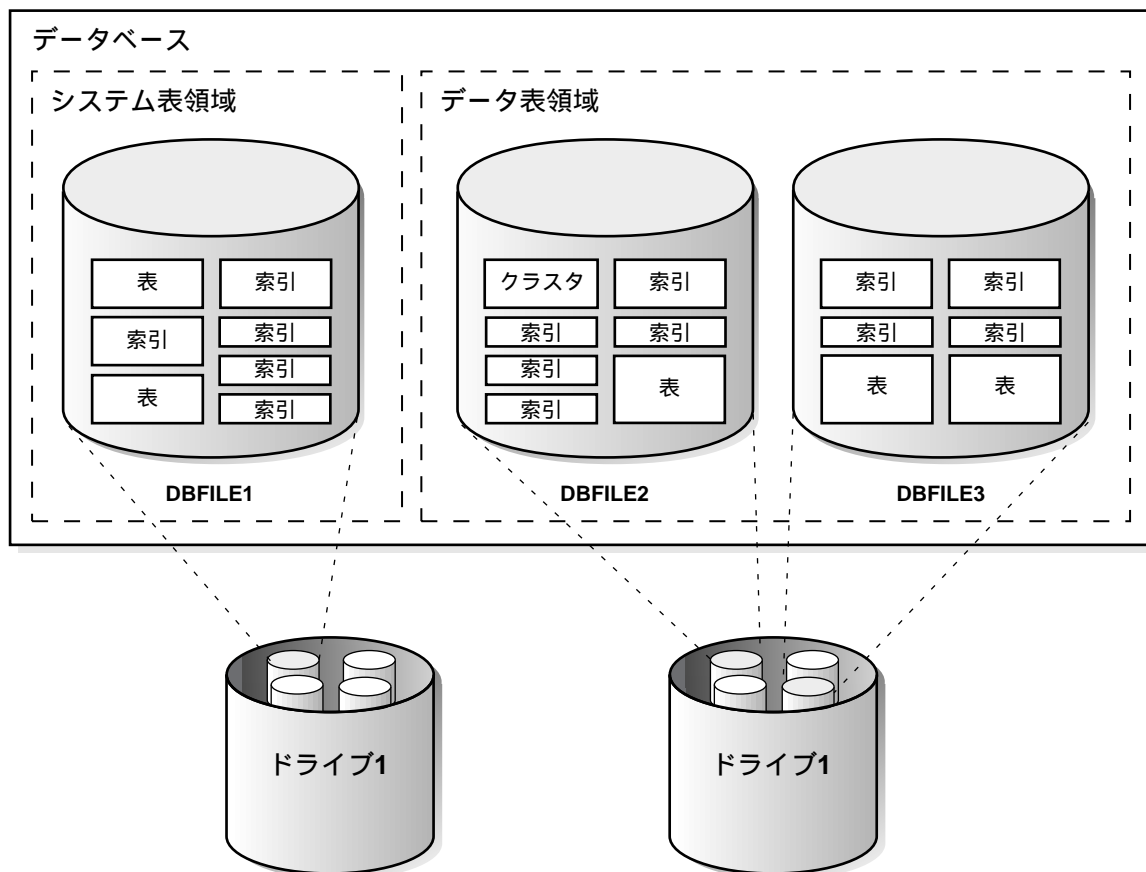
スキーマ・オブジェクトは、論理的なデータ記憶構造です。各スキーマ・オブジェクトは、その情報を格納するディスク上の物理ファイルと1対1では対応しませんが、データベースの表領域内に論理的に格納されます。各オブジェクトのデータは、物理的には、表領域の1つ以上のデータ・ファイルに格納されます。表、索引およびクラスタなど、いくつかのオブジェクトについては、表領域のデータ・ファイル内のオブジェクトに対して割り当てるディスク領域の容量を指定できます。

スキーマと表領域の間にはどんな関連もありません。異なるスキーマのオブジェクトが1つの表領域に含まれることもあれば、1つのスキーマのオブジェクトが異なる表領域に含まれることもあります。図 10-1 に、オブジェクト、表領域およびデータ・ファイルの関連を示します。

追加情報： この章では、表、ビュー、マテリアライズド・ビュー、順序、シノニム、索引およびクラスタについて説明します。その他の種類のスキーマ・オブジェクトについては、このマニュアルの別の章または他のマニュアルで説明しています。次のとおりです。

- プロシージャ、ファンクションおよびパッケージの説明は、[第 18 章「プロシージャとパッケージ」](#)を参照してください。
- トリガーの説明は、[第 20 章「トリガー」](#)を参照してください。
- データベース・リンクの説明は、33-1 ページの「[分散データベース](#)」を参照してください。

図 10-1 スキーマ・オブジェクト、表領域およびデータ・ファイル



表

「表」は、Oracle データベースにおけるデータ記憶の基本単位です。データは「行」と「列」に格納されます。すべての表は、「表名」(EMP など)と列の集合で定義されます。各列には、「列名」(EMPNO、ENAME および JOB など)「データ型」(VARCHAR2、DATE または NUMBER など)および「幅」(DATE のように、データ型によって事前に決定されていることもある)または「精度」と「位取り」(NUMBER データ型の列のみ)を指定します。行は、1 つのレコードに対応する列情報の集まりです。Oracle データ型の詳細は、[第 12 章「ビルトイン・データ型」](#)を参照してください。

表の各列に対してルールを任意に指定できます。これらのルールのことを「整合性制約」と呼びます。たとえば、整合性制約の1つに NOT NULL 整合性制約があります。この制約により、どの行の列にも必ず値が入るようになります。整合性制約の詳細は、第 28 章「データの整合性」を参照してください。

表の作成後に、SQL 文を使用してデータ行を挿入します。挿入した表データは、SQL を使用して問合せ、削除または更新できます。

図 10-2 に、EMP という名前のサンプル表を示します。

図 10-2 EMP 表

行

列

列名

	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-88	800.00	300.00	20
7499	ALLEN	SALESMAN	7698	20-FEB-88	1600.00	300.00	30
7521	WARD	SALESMAN	7698	22-FEB-88	1250.00	500.00	30
7566	JONES	MANAGER	7839	02-APR-88	2975.00		20

NULLが許されない列

NULLが許される列

表データの格納方法

表の作成時に、表の将来のデータを保持するために、データ・セグメントが表領域内に自動的に割り当てられます。(ただし、クラスタ化表と一時表は、このルールの例外です)。表のデータ・セグメントに対する領域の割当て、およびこの確保した領域の使用は、次の方法で制御できます。

- データ・セグメントに対して記憶領域パラメータを設定する。これにより、データ・セグメントのエクステントの領域の容量を制御できます。
- データ・セグメントに対して、PCTFREE パラメータと PCTUSED パラメータを設定する。これにより、データ・セグメントのエクステントを構成するデータ・ブロック内の空き領域の使用を制御できます。

クラスタ化した表のデータは、対応するクラスタに対して作成されたデータ・セグメント内に格納されます。クラスタ化した表を作成または変更するときに、記憶領域パラメータは指定できません。つまり、クラスタに対して設定された記憶領域パラメータが、クラスタ内のすべての表の記憶を制御します。

クラスタ化していない表のデータ・セグメントが含まれる表領域は、表所有者のデフォルト表領域または CREATE TABLE 文で明確に指定した表領域です。29-13 ページの「[ユーザー表領域の設定と割当て制限](#)」を参照してください。

行の形式とサイズ

Oracle では、データベース表の各行が 1 つ以上の行断片として格納されます。1 つの行全体を単一のデータ・ブロックに挿入できる場合、その行は 1 つの行断片として格納されます。ただし、1 行のデータを単一のデータ・ブロックには挿入できない場合や、既存の行の更新によって行がデータ・ブロックのサイズを超えてしまう場合は、複数の行断片を使用して行が格納されます。1 つのデータ・ブロックには、通常、行あたり 1 つの行断片のみが格納されます。1 つの行を複数の行断片に格納する必要がある場合は、複数のブロックにわたって行が「連鎖」されます。連鎖行の各断片は、それぞれの断片の ROWID を使用して連鎖されます。4-9 ページの「[行連鎖と移行](#)」を参照してください。

それぞれの行断片は、連鎖されていても連鎖されていなくても、行の全部または一部の列の「行ヘッダー」とデータを含みます。また、個々の列が複数の行断片にまたがり、結果として複数のデータ・ブロックにまたがっている場合もあります。[図 10-3](#) に、行断片の形式を示します。

「行ヘッダー」は、データの前に位置し、次のような情報を含んでいます。

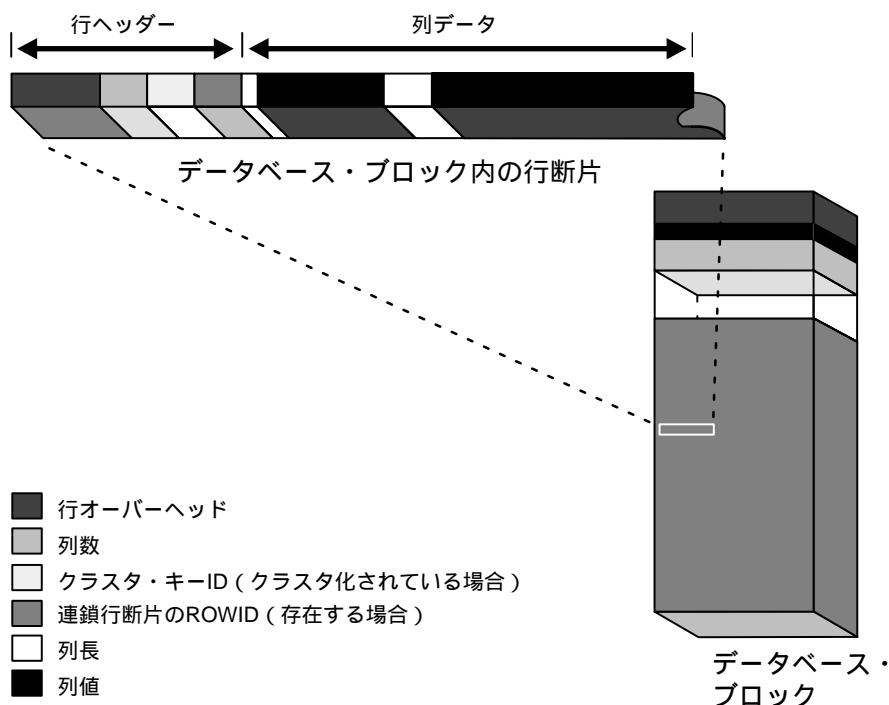
- 行断片
- 連鎖（連鎖された行断片の場合のみ）
- 行断片内の列
- クラスター・キー（クラスタ化されたデータの場合のみ）

1 つのブロックに完全に収まる行は、最低 3 バイトの行ヘッダーを持っています。各行は、行ヘッダー情報の後に列の長さとしてデータを格納します。列の長さは、250 バイト以下のデータを格納する列については 1 バイト、250 バイトよりも大きなデータを格納する列については 3 バイトを必要とし、列データの直前に位置します。列データのために必要となる領域は、データ型によって異なります。列のデータ型が可変長である場合、値を保持するために必要な領域は、データの更新によって変動します。

領域を節約するために、NULL を含む列は、列の長さ（ゼロ）のみを格納します。NULL 列のデータは格納されません。また、末尾にある NULL 列の場合は、列の長さも格納されません。10-7 ページの「[NULL](#)」を参照してください。

注意： また、各行は、データ・ブロック・ヘッダーの行ディレクトリで 2 バイトを使用します。4-4 ページの「[行ディレクトリ](#)」を参照してください。

図 10-3 行断片の形式



クラスタ化された行には、クラスタ化されていない行と同じ情報が格納されます。また、クラスタ化された行には、それらの行が属するクラスタ・キーを参照する情報も格納されます。10-46 ページの「[クラスタ化されたデータ・ブロックの形式](#)」を参照してください。

削除された列または未使用の列

ALTER TABLE コマンドの DROP COLUMN オプションを使用すると、表から列を削除できます。これにより、表記述から列が削除され、表の各行から列の長さとデータが削除され、データ・ブロック内の領域が解放されます。

大きい表から列を削除するには長時間かかります。かわりに、ALTER TABLE コマンドの SET UNUSED オプションを使用し、列に未使用のマークを設定する方が短時間ですみます。これにより、その列データは使用不可になりますが、データは表の各行に残ります。列に未使用のマークを設定した後は、同じ名前を持つ別の列をその表に追加できます。未使用の列は、後で列データが占める領域の再生が必要になった時点で削除できます。

追加情報： 列を削除するか未使用マークを設定する方法は『Oracle8i 管理者ガイド』、ALTER TABLE コマンドの詳細は『Oracle8i SQL リファレンス』を参照してください。

行断片の ROWID

「ROWID」は、それぞれの行断片を位置またはアドレスで識別します。ROWID が行断片に割り当てられると、対応する行の削除や、Export/Import コーティリティによるエクスポートまたはインポートが実行されるまで、行断片はその ROWID を保持します。クラスタ化された表の場合（10-44 ページの「[クラスタ](#)」を参照）、ある行のクラスタ・キー値が変化しても、その行はそれまでと同じ ROWID を保持しますが、同時に新しい値に関する追加のポインタ ROWID も取得します。

ROWID は、行断片の存続期間中は一定なので、SELECT、UPDATE および DELETE などの SQL 文で ROWID を参照すると便利です。詳細は、12-14 ページの「[物理 ROWID](#)」を参照してください。

列の順序

列の順序は、表内のすべての行について同一です。列は、通常、CREATE TABLE 文にリストした順序で格納されますが、そのことが保証されているわけではありません。たとえば、LONG データ型の列を持つ表を作成する場合、Oracle は常にこの列を最後に格納します。また、表を変更して新しい列を追加すると、その新しい列は、最後に格納されます。

一般的には、行に必要な領域を少なくするために、NULL を頻繁に格納する列を最後の列にする必要があります。ただし、作成する表に LONG 列が含まれている場合、NULL 列を最後に置く利点はなくなります。

NULL

「NULL」は、ある行のある列に値が入っていないことを意味します。NULL は、欠落しているデータ、不明なデータ、または適用できないデータを示します。他の値（ゼロなど）を暗示する目的では NULL を使用しないでください。NOT NULL または PRIMARY KEY 制約が列に対して定義されていない場合に限り、列に NULL 値を入力できます。これらの制約が列に対して定義されている場合、その列に値を持たない行は挿入できません。

データ値を持つ 2 つの列の間にはさまれた NULL はデータベースに格納されます。このような場合、NULL には列の長さ（ゼロ）を格納する 1 バイトのみが必要となります。

行内の末尾にある NULL には、格納領域は必要ありません。新しい行の行ヘッダーが、前の行の残りの列が NULL であることを知らせるためです（たとえば、表の最後の 3 列が NULL であれば、その 3 列には情報は格納されません）。列が多い表では、ディスク領域を節約するため、NULL を含む可能性の高い列は最後に定義する必要があります。

NULL とその他の値との比較の大部分は、定義によって TRUE にも FALSE にもならず、UNKNOWN となります。SQL 文の中で NULL を識別するには、IS NULL 述語を使用します。NULL を NULL 以外の値に変換するには、SQL 関数の NVL を使用します。

追加情報： IS NULL および NVL 関数を使用した比較の詳細は、『Oracle8i SQL リファレンス』を参照してください。

クラスタ・キー列の値が NULL の場合または索引がビットマップ索引の場合を除いて、NULL に索引を付けることはできません (10-23 ページの「[索引と NULL](#)」および 10-34 ページの「[ビットマップ索引と NULL](#)」を参照してください)。

列のデフォルト値

新しい行が挿入され、列に対する値が省略されたときに、デフォルト値が自動的に入力されるように、表の列にデフォルト値を割り当てることができます。列のデフォルト値は、実際に INSERT 文が値を指定している場合と同様に機能します。

正当なデフォルト値には、リテラルや、列、LEVEL、ROWNUM または PRIOR を参照しない式が含まれます。デフォルト値は、SQL 関数 SYSDATE、USER、USERENV および UID を含むことができます。デフォルトのリテラルまたは式のデータ型は、その列のデータ型に一致しているか、あるいはそれに変換可能である必要があります。

列に対してデフォルト値が明示的に定義されていない場合、その列のデフォルトは暗黙的に NULL に設定されます。

デフォルト値の挿入と整合性制約のチェック

デフォルト値を持つ行が挿入されると、整合性制約チェックが行われます。たとえば、[図 10-4](#) では、従業員の部門番号に対する値を持たない行が、EMP 表に挿入されています。従業員の部門番号に値が指定されていないため、DEPTNO 列のデフォルト値「20」が割り当てられます。デフォルト値が割り当てられた後で、DEPTNO 列に定義されている FOREIGN KEY 整合性制約がチェックされます。

整合性制約の詳細は、[第 28 章「データの整合性」](#)を参照してください。

図 10-4 デフォルトの列値

親キー

表DEPT		
DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

外部キー

表EMP							
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9000.00		20
7499	ALLEN	VP_SALES	7329	20-FEB-90	7500.00	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	5000.00	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90	2975.00	400.00	30
7691	OSTER	SALESMAN	7521	06-APR-90	2975.00	400.00	20

挿入される新規行
(DEPTNO列には値がない)INSERT
INTO

7691	OSTER	SALESMAN	7521	06-APR-90	2975.00	400.00	
------	-------	----------	------	-----------	---------	--------	--

デフォルト値
(この列の値を指定
しなければデフォルト
の20が使用される)

ネストした表

データ型として別の表を指定した表を作成できます。つまり、表の列値として別の表を「ネスト」できます。Oracle Server は、親表の行の「外に」ネストした表のデータを、ネストした表列に対応付けた「格納表」を使用して格納します。親行には、ネストした表のインスタンスに対応付けられた一意の集合識別子が入れます。

追加情報： 13-11 ページの「[ネストした表](#)」および『Oracle8i アプリケーション開発者ガイド 基礎編』参照してください。

一時表

Oracle では、永続的な表だけでなく、トランザクションまたはセッションの期間中にのみ存在するセッションのプライベート・データが保持される「一時表」を作成できます。

CREATE GLOBAL TEMPORARY TABLE コマンドでは、トランザクション固有またはセッション固有の一時表が作成されます。トランザクション固有の一時表の場合、データはトランザクションの存続期間中のみ存在し、セッション固有の一時表の場合、データはセッションの存続期間中のみ存在します。一時表には、セッションのプライベート・データが含まれます。各セッションで表示したり変更できるのは、そのセッションの独自データだけです。一時表のデータについては、DML ロックは取得されません。各セッションに固有のプライベート・データがあるため、一時表には LOCK コマンドは無効です。

セッション固有の一時表で TRUNCATE 文が発行されると、その固有セッションのデータが切り捨てられます。同じ表を使用する他のセッションのデータが切り捨てられることはありません。

一時表での DML 文では、データ変更の REDO ログは生成されません。ただし、データの UNDO ログと UNDO ログの REDO ログは生成されます。セッションの終了時には、一時表からデータが自動的に削除されます。たとえば、ユーザーがログオフした場合や、セッションまたはインスタンスのクラッシュ時など、セッションが異常終了した場合です。

CREATE INDEX コマンドを使用すると、一時表の索引を作成できます。一時表に作成された索引と、その索引のデータは、一時表のデータと同じセッションまたはトランザクション・スコープを持ちます。

一時表と永続表の両方にアクセスするビューを作成できます。また、一時表のトリガーを作成することも可能です。

Export および Import ユーティリティでは、一時表の定義をエクスポートおよびインポートできます。ただし、ROWS オプションを使用しても、データ行はエクスポートされません。同様に、一時表の定義はレプリケートできますが、そのデータはレプリケートできません。

セグメントの割当て

一時表は一時セグメントを使用します（4-15 ページの「[一時セグメント内のエクステント](#)」を参照）。永続表と違って、一時表とその索引の場合、作成時にセグメントが自動的に割り当てられることはありません。かわりに、最初の INSERT（または CREATE TABLE AS SELECT）の実行時にセグメントが割り当てられます。これは、最初の INSERT の前に SELECT、UPDATE または DELETE が実行されると、表が空のように見えることを意味します。

一時表で DDL コマンド（ALTER TABLE、DROP TABLE、CREATE INDEX など）を実行できるのは、バインドされているセッションがない場合だけです。セッションは、INSERT の実行時に一時表にバインドされます。セッションは、セッションの終了時に TRUNCATE によって、またはトランザクション固有の一時表の場合は COMMIT または ABORT によってアンバインドされます。

一時セグメントは、トランザクション固有の一時表の場合はトランザクションの終了時に、セッション固有の一時表の場合はセッションの終了時に割当て解除されます。

親トランザクションと子トランザクション

トランザクション固有の一時表には、ユーザー・トランザクションとその子トランザクションからアクセスできます。ただし、2つのトランザクションが、同じセッションで特定のトランザクション固有の一時表を同時に使用することはできません（異なるセッションであれば複数のトランザクションが使用できます）。

- ユーザー・トランザクションが一時表への INSERT を実行した後は、その子トランザクションは一時表を使用できなくなる。
- 子トランザクションが一時表への INSERT を実行すると、その子トランザクションの終了時に、一時表に対応するデータが削除される。その後は、ユーザー・トランザクションまたは他の任意の子トランザクションから、一時表にアクセスできます。

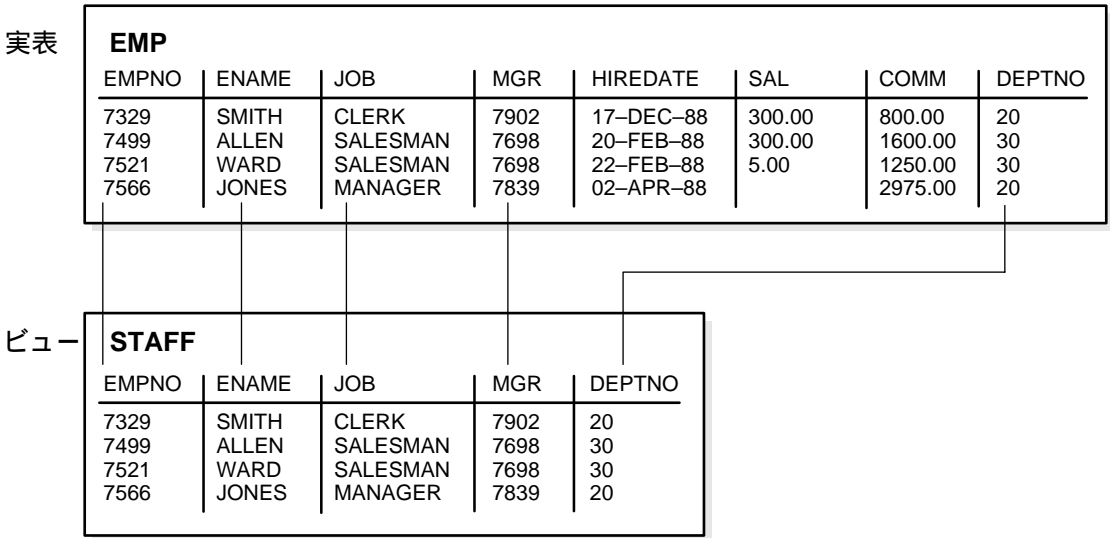
ビュー

ビューとは、1つ以上の表（または他のビュー）に含まれているデータを調整して表現したものです。ビューは問合せの結果を取り込んで、それを表として取り扱います。つまり、ビューは「格納された問合せ」または「仮想表」と考えることができます。多くの場合、ビューは表と同じように使用できます。

たとえば、EMP 表には、いくつかの列と多数の情報行があります。ユーザーにはそのうちの5つの列または特定の行しか参照させないようにする場合は、他のユーザーがアクセスできるように EMP 表のビューを作成できます。

[図 10-5](#) に、実表 EMP から導出された STAFF というビューの例を示します。ビューが、実表の5つの列のみを表示していることに注目してください。

図 10-5 ビューの例



ビューは表から導出されるので、ビューと表には多数の類似点があります。たとえば、ビューには表と同じように最大 1000 個の列を定義できます。ビューは問合せが可能なだけでなく、いくつかの制限付きでビューのデータを更新、挿入および削除できます。実際にビューに対して実行されるすべての操作は、そのビューの実表のデータに影響し、実表の整合性制約とトリガーの対象になります。

追加情報：『Oracle8i SQL リファレンス』を参照してください。

注意： ビューでは整合性制約とトリガーを明示的に定義できませんが、ビューから参照される実表に対しては定義できます。

ビューの記憶領域

表とは異なり、ビューには記憶領域は割り当てられず、ビューが実際にデータを持つことはありません。ビューは、そのビューで参照する表からデータの抽出または取出しを行う問合せによって定義されます。これらの表は「実表」と呼ばれます。実表としては、実際の表またはビュー（スナップショットを含む）を使用できます。ビューは他のオブジェクトを基盤としているため、データ・ディクショナリ内のビューの定義（格納された問合せ）以外には記憶領域が必要ありません。

ビューの使用法

ビューは、実表に含まれるデータをいろいろな表現形式で表現する手段を提供します。様々なユーザーに合わせてデータの表現形式を変化させることができるため、ビューは非常に強力な機能です。通常、ビューは次の目的で使用されます。

- 事前に定義された行または列（あるいはその両方）の集合のみにアクセスを限定することにより、表のセキュリティ・レベルを強化する。

たとえば、[図 10-5](#) は、STAFF ビューが実表 EMP の列 SAL と COMM をどのように隠すかを示しています。

- データの複雑さを隠す。

たとえば、複数の表の関連した列または行を 1 つにまとめる「結合処理」によって、単一のビューを定義できます。ただし、ビューからは、この情報が複数の表から抽出されたものであるということとはわかりません。

- ユーザー側のコマンドを単純化する。

たとえば、ユーザーが結合の方法を知らなくても、複数の表から情報を選択できるようにします。

- 実表とは異なる見方でデータを提示する。

たとえば、ビューが基礎にしている表に影響を与えずに、ビューの列名を変更できます。

- 実表の定義の変更からアプリケーションを分離する。

たとえば、ビューを定義している問合せが表の 4 つの列の中の 3 つの列を参照している場合、たとえ 5 番目の列が追加されてもビューの定義は影響を受けないため、ビューを使用しているすべてのアプリケーションも影響を受けません。

- ビューなしでは表現できなかった問合せを表現する。

たとえば、GROUP BY ビューと表を結合してビューを定義できます。また、UNION ビューと表を結合してビューを定義することもできます。

追加情報： GROUP BY と UNION の詳細は、『Oracle8i SQL リファレンス』を参照してください。

- 複雑な問合せを保存する。

たとえば、ある問合せで表情報に対して複雑な計算を実行したとします。この問合せをビューとして保存しておけば、そのビューに問合せを行うたびに同じ計算が実行されません。

ビューのメカニズム

ビューの定義は、ビューを定義する問合せのテキストとしてデータ・ディクショナリに格納されます。ビューが SQL 文で参照されると、Oracle はそのビューを参照する文を、ビューを定義する問合せとマージしてから、マージした文を共有 SQL 領域で解析し、実行します。新しい共有 SQL 領域内のビューを参照する文は、既存の共有 SQL 領域に同一の文が含まれていない場合にのみ解析されます。したがって、ビューを使用すると、共有 SQL に対応付けられているメモリーの使用量を減らせるという利点があります。

NLS パラメータ

文字列リテラルが含まれているビューや、NLS パラメータを引数として持つ SQL 関数 (TO_CHAR、TO_DATE および TO_NUMBER など) を評価する場合、Oracle はセッションの NLS パラメータからこれらのパラメータのデフォルト値を取り出します。これらのデフォルト値は、ビューの定義で NLS パラメータを明示的に指定することにより上書きできます。

追加情報： 各国語サポートの詳細は、『Oracle8i NLS ガイド』を参照してください。

索引の使用方法

Oracle がビューに対する問合せで索引を使用するかどうかは、元の問合せがビュー定義の問合せとマージされるときにどのように変換されるかによって決まります。

たとえば、次のビューについて考えてみます。

```
CREATE VIEW emp_view AS
  SELECT empno, ename, sal, loc
     FROM emp, dept
    WHERE emp.deptno = dept.deptno AND
           dept.deptno = 10;
```

ここで、ユーザーは次の問合せを発行します。

```
SELECT ename
   FROM emp_view
  WHERE empno = 9876;
```

Oracle によって作成される最終的な問合せは次のようになります。

```
SELECT ename
   FROM emp, dept
  WHERE emp.deptno = dept.deptno AND
        dept.deptno = 10 AND
        emp.empno = 9876;
```

可能であれば、Oracle はビューに対する問合せを、そのビューを定義している問合せとマージします（基礎となるビューの問合せもマージされます）。マージされた問合せは、ビューを参照しないで発行された場合と同じように最適化されます。そのため、列がビュー定義で参照されても、またはビューに対するユーザーの問合せで参照されても、参照される実表の列に対する索引を使用できます。

ユーザーの発行した問合せとビュー定義をマージできないこともあります。その場合は、参照された列の索引すべてを使用できるとは限りません。

問合せの最適化の詳細は、23-14 ページの「[ビューにアクセスする文の最適化](#)」を参照してください。

依存性とビュー

ビューは、他のオブジェクト（表、スナップショットまたは他のビュー）を参照する問合せによって定義されるため、参照されるオブジェクトに依存します。Oracle は、ビューの依存性を自動的に処理します。たとえば、ビューの実表が削除された後で再作成された場合、Oracle は、新しい実表がそのビューの既存の定義と合致するかどうかを判断します。データベースの依存性の詳細は、[第 21 章「Oracle の依存性の管理」](#)を参照してください。

更新可能な結合ビュー

「結合ビュー」とは、ビューの定義に FROM 句で複数の表またはビューが指定され、DISTINCT 句、AGGREGATION 句、GROUP BY 句、START WITH 句、CONNECT BY 句、ROWNUM 句および集合演算（UNION ALL、INTERSECT など）のいずれをも使用していないビューです。

「更新可能な結合ビュー」とは、2 つ以上の実表またはビューを含み、UPDATE、INSERT および DELETE の各操作が実行可能である結合ビューです。ビューのどの列が更新可能であるかを示す情報は、データ・ディクショナリ・ビュー ALL_UPDATABLE_COLUMNS、DBA_UPDATABLE_COLUMNS および USER_UPDATABLE_COLUMNS に含まれています。

[表 10-1](#) は、更新可能な結合ビューの規則を示しています。

表 10-1 結合ビューに対する INSERT、UPDATE および DELETE の規則

規則	説明
一般規則	結合ビューに対する INSERT、UPDATE または DELETE 操作では、そのビューの基礎となる表を一度に 1 つだけ変更できる。
UPDATE 規則	結合ビュー内の更新可能なすべての列は、キー保存表の各列に対応していなければならない。ビューを定義するときに WITH CHECK OPTION 句を指定すると、すべての結合列と繰返し表のすべての列は更新できなくなります。
DELETE 規則	結合に含まれているキー保存表が厳密に 1 つであれば、結合ビューの行を削除できる。ビューを定義するときに WITH CHECK OPTION 句を指定し、キー保存表を繰返し指定した場合は、ビューから行を削除できません。
INSERT 規則	INSERT 文では、明示的にも暗黙的にもキー保存表以外の列を参照してはならない。結合ビューを定義するときに WITH CHECK OPTION 句を指定すると、INSERT 文は使用できません。

更新可能でないビューは、INSTEAD OF トリガーを使用して変更できます。詳細は、20-11 ページの「[INSTEAD OF トリガー](#)」を参照してください。

オブジェクト・ビュー

Oracle オブジェクト・リレーショナル・データベースの場合、「オブジェクト・ビュー」を使用すると、リレーショナル・データを、それがオブジェクト型として格納されているかのように検索、更新、挿入および削除できます。また、オブジェクト、REF およびコレクション（ネストした表と VARRAY）などのオブジェクト・データ型である列を持つビューを定義できます。

追加情報： [第 15 章「オブジェクト・ビュー」](#) および『Oracle8i アプリケーション開発者ガイド 基礎編』参照してください。

インライン・ビュー

「インライン・ビュー」は、スキーマ・オブジェクトではなく、SQL 文でビューと同様に使用できる別名（相関名）を持つ副問合せです。

たとえば、次の問合せでは、集計表 SUMTAB を TIME 表で定義されているインライン・ビュー V に結合して T.YEAR を取得し、SUMTAB の集計を YEAR レベルにロールアップしています。

```
SELECT v.year, s.prod_name, SUM(s.sum_sales)
FROM sumtab s,
     (SELECT DISTINCT t.month, t.year FROM time t) v
WHERE s.month = v.month
GROUP BY v.year, s.prod_name;
```

追加情報： 副問合せの詳細は、『Oracle8i SQL リファレンス』を参照してください。

マテリアライズド・ビュー

マテリアライズド・ビューは、データの集計、事前計算、レプリケートおよび分配に使用できるスキーマ・オブジェクトです。この種のビューは、データ・ウェアハウジング、意思決定支援および分散コンピューティングやモバイル・コンピューティングなど、さまざまなコンピュータ環境に適しています。

- データ・ウェアハウスでは、マテリアライズド・ビューは、合計値や平均値など、集約されたデータを事前に計算して格納するために使用される。通常、このような環境では、マテリアライズド・ビューには集計データが格納されるので、「サマリー」と呼ばれます。また、マテリアライズド・ビューは、集合の有無に関係なく、結合の事前計算にも使用できます。

コストベース最適化では、マテリアライズド・ビューを使用して、要求を満たすことができる場合と、そのために使用すべき場合を自動的に認識して、問合せのパフォーマンスを改善できます。要求は、マテリアライズド・ビューを使用するように、最適化によって自動的に透過的にリライトされます。その後、問合せは、基礎となるディテール表やビューではなく、マテリアライズド・ビューに送られます。

- 分散環境では、マテリアライズド・ビュー（「スナップショット」と呼ばれることもある）は、分散サイトでデータをレプリケートし、複数のサイトで実行される更新を競合解決方法を用いて同期化するために使用される。レプリカとしてのマテリアライズド・ビューは、他の方法ではリモート・サイトからアクセスしなければならないデータへの、ローカル・アクセスを提供します。
- モバイル・コンピューティング環境では、マテリアライズド・ビューは、中央のサーバーからモバイル・クライアントにデータのサブセットをダウンロードし、中央のサーバーから定期的に取り替えられ、クライアントで実行された更新を中央のサーバーに波及させるために使用される。

マテリアライズド・ビューは、いくつかの点で索引に似ています。つまり、記憶領域を消費し、マスター表のデータに変更があった場合はリフレッシュする必要があります。また、クエリー・リライトに使用すると SQL の実行効率が改善され、その存在は SQL アプリケーションとユーザーに対して透過的です。（10-21 ページの「索引」を参照。）索引と違って、マテリアライズド・ビューには、SELECT 文を使用して直接アクセスできます。必要なリフレッシュのタイプによっては、INSERT、UPDATE または DELETE 文で直接アクセスすることも可能です。

マテリアライズド・ビューはパーティション化できます。また、マテリアライズド・ビューをパーティション表で定義したり、1つ以上の索引をマテリアライズド・ビューで定義することもできます。パーティション化の詳細は、[第 11 章「パーティション表とパーティション索引」](#)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

Oracle では、マテリアライズド・ビューをマスター表の変更後にリフレッシュすることによって、そこに含まれるデータがメンテナンスされます。リフレッシュ方法には、増分リフレッシュ（「高速リフレッシュ」）または完全リフレッシュがあります。高速リフレッシュ方法を使用する場合、マスター表に対する変更は「マテリアライズド・ビュー・ログ」または「ダイレクト・ローダー・ログ」に記録されます。

マテリアライズド・ビューは、必要時に、または定期的にリフレッシュできます。また、マスター表と同じデータベース内のマテリアライズド・ビューは、トランザクションによってマスター表の変更がコミットされるたびにリフレッシュできます。

マテリアライズド・ビュー・ログ

「マテリアライズド・ビュー・ログ」とは、マスター表に定義されているマテリアライズド・ビューの増分リフレッシュを実行できるように、マスター表の変更を記録するスキーマ・オブジェクトです。マテリアライズド・ビュー・ログは、「スナップショット・ログ」ともいいます。

マテリアライズド・ビュー・ログは、それぞれ 1 つのマスター表に対応付けられています。マテリアライズド・ビュー・ログは、マスター表と同じデータベースおよびスキーマに格納されます。

追加情報： ウェアハウジング環境におけるマテリアライズド・ビューとマテリアライズド・ビュー・ログの説明は『Oracle8i チューニング』、レプリケーションに使用されるマテリアライズド・ビュー（スナップショット）の説明は『Oracle8i レプリケーション・ガイド』を参照してください。

ディメンション

ディメンションとは、1 対の列または列セットの間の階層関係を定義するスキーマ・オブジェクトです。階層関係は、ある階層レベルから次のレベルへの「機能上の依存関係」です。ディメンションは、列相互の論理関係のコンテナで、それにはデータ記憶領域は割り当てられません。

CREATE DIMENSION 文では、次の事項を指定します。

- 複数の LEVEL 句。それぞれがディメンション内の 1 列または列の集合を識別します。
- 1 つ以上の HIERARCHY 句。隣接するレベル間の親子関係を指定します。

- オプションの ATTRIBUTE 句。それぞれが、個々のレベルに対応付けられている他の列または列セットを識別します。

ディメンション内の列は、同じ表のもの（「非正規化」）または複数の表のもの（「完全正規化または一部正規化」）でもかまいません。複数の表からの列によるディメンションを定義するには、HIERARCHY 句の JOIN KEY オプションを使用して表を接続します。

たとえば、正規化されている time ディメンションには、date 表、month 表および year 表と、各 date 行を month 行に、各 month 行を year 行に接続する結合条件を含めることができます。完全な非正規化の time ディメンションでは、date、month および year 列はすべて同じ表に格納されます。正規化されているかどうかに関係なく、列相互の階層関係を CREATE DIMENSION 文で指定する必要があります。

追加情報：『Oracle8i チューニング』には、ウェアハウジング環境におけるディメンションの使用方法が記載されています。

シーケンス・ジェネレーター

シーケンス・ジェネレーターは、連続的な数値を生成します。また、ディスク I/O やトランザクション・ロックなどのオーバーヘッドを発生させずに、一意の連続的な数を生成するため、マルチユーザー環境では特に有効です。したがって、このシーケンス・ジェネレーターによって「直列化」（2 つのトランザクションの文が連続番号を同時に生成する必要がある状態）が低減されます。この直列化を避けることによって、トランザクションのスループットが改善され、ユーザーの待ち時間が大幅に短縮されます。

順序番号とは、データベース内で定義される最大 38 桁の Oracle 整数です。順序の定義には、順序の名前、昇順または降順、数の増分値などの一般的な情報を指定します。順序定義で重要なことの 1 つは、生成される順序番号の集合をメモリー内にキャッシュするかどうかということです。

データベースごとにすべての順序定義が、SYSTEM 表領域内の単一のデータ・ディクショナリ表に行として格納されます。したがって、SYSTEM 表領域は常時オンラインであるため、すべての順序定義がいつでも使用できます。

順序番号は、順序を参照する SQL 文によって使用されます。文を発行して、新しい順序番号を生成することも、現行の順序番号を使用することもできます。ユーザーのセッションで文によって順序番号が生成されると、そのセッションでのみ固有の順序番号が使用可能になります。つまり、順序を参照する各ユーザーは、固有の現行順序番号にアクセスできます。

順序番号は表からは独立して生成されます。そのため、同じシーケンス・ジェネレーターを 1 つの表または複数の表に対して使用できます。順序番号の生成は、データに対して一意の主キーを自動的に生成する場合や、複数の行または表にまたがるキーを統合する場合に有効です。最終的にロールバックされたトランザクションで生成されて使用された個々の順序番号はスキップされます。必要であれば、アプリケーションは、これらの順序番号を受け取って再利用するように作成することもできます。

追加情報： 順序の使用がパフォーマンスに与える影響は、『Oracle8i 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。

シノニム

シノニムとは、表、ビュー、スナップショット、順序、プロシージャ、ファンクションまたはパッケージに対する別名です。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶領域は必要ありません。

シノニムは、セキュリティと便利さのために使用されます。たとえば、次のような利点があります。

- オブジェクトの名前と所有者を隠す。
- 分散データベースのリモート・オブジェクトの位置の透過性を実現する。
- データベース・ユーザー側の SQL 文を単純にする。

パブリック・シノニムとプライベート・シノニムの両方を作成できます。「パブリック」シノニムは、PUBLIC という名前の特別なユーザー・グループが所有しており、データベースのすべてのユーザーがアクセスできます。「プライベート」シノニムは、特定のユーザーのスキーマに含まれており、そのユーザーが他のユーザーに対するシノニムの使用許可を制御します。

シノニムは、分散システム内の位置を含めて、基礎になるオブジェクトの所在を隠すため、分散データベース環境でもそれ以外の環境でもきわめて役立ちます。基礎になるオブジェクトが改名または移動されても、シノニムを再定義する必要があるだけで、シノニムを基礎とするアプリケーションは修正せずにそのまま機能します。

また、分散データベース・システム内のユーザーのために、シノニムによって SQL 文を単純化できます。実表の所在を隠したり、SQL 文の複雑さを低減するために、データベース管理者はしばしばパブリック・シノニムを作成します。その理由と作成方法を次の例で説明します。この例では、次のような状況を想定します。

- ユーザー JWARD が所有するスキーマ内に SALES_DATA という表がある。
- SALES_DATA 表に対する SELECT 権限が PUBLIC に付与されている。

ここで、データベースのユーザーが、次のような SQL 文を使用して SALES_DATA 表を問い合わせます。

```
SELECT * FROM jward.sales_data;
```

問合せを実行するには、表を含むスキーマと表の名前の両方を含める必要があることに注意してください。

データベース管理者が次の SQL 文によって、パブリック・シノニムを作成するとします。

```
CREATE PUBLIC SYNONYM sales FOR jward.sales_data;
```


パブリック・シノニムが作成されると、ユーザーは、次のような単純な SQL 文で SALES_DATA 表を問い合わせることができます。

```
SELECT * FROM sales;
```

パブリック・シノニム SALES は、表 SALES_DATA の名前とその表を含むスキーマ名を隠していることに注意してください。

索引

索引は、表とクラスタに対応付けるオプションの構造体です。表の 1 つ以上の列に索引を作成し、その表に対する SQL 文の実行を高速化できます。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。同様に、Oracle の索引を使用すると表データに高速にアクセスできます。適切に使用すれば、索引はディスク I/O を低減する基本的な手段になります。

索引ごとに列の組合せが異なる限り、1 つの表に必要な数だけ索引を作成できます。列の組合せを個別に指定すれば、同じ列を使用して複数の索引を作成できます。たとえば、次の文では有効な組合せを指定しています。

```
CREATE INDEX emp_idx1 ON emp (ename, job);  
CREATE INDEX emp_idx2 ON emp (job, ename);
```

表の 1 列のみを参照する索引がすでに存在している場合、この種の索引は作成できません。

Oracle は、パフォーマンスの機能性を補足する複数の索引体系を提供しています。これには、B* ツリー索引（現在では最も一般的）、B* ツリー・クラスタ索引、ハッシュ・クラスタ索引、逆キー索引およびビットマップ索引があります。また、アプリケーションまたはカートリッジ固有のファンクションベース索引やドメイン索引もサポートしています。

索引の有無によって、SQL 文の表現を変更する必要はありません。索引は、データへの高速なアクセス・パスにすぎないからです。つまり、索引は実行速度にのみ影響を与えます。索引の付いたデータ値では、索引はその値を含んでいる行の位置を直接示すポインタとして機能します。

索引は、関連する表のデータから論理的にも物理的にも独立しています。索引は、実表や他の索引に影響を及ぼさずにいつでも作成または削除できます。索引が削除されてもすべてのアプリケーションは作動し続けますが、以前索引が付いていたデータにアクセスする場合、速度が遅くなる可能性があります。索引は独立した構造体であるため、記憶領域を必要とします。

作成された索引は、Oracle によって自動的にメンテナンスされ、使用されます。行の新規追加、更新または削除など、データに対する変更は、関連するすべての索引に自動的に反映され、ユーザーによる操作は不要です。

新たにいくつかの行が挿入されても、索引付きのデータ検索のパフォーマンスはほとんど一定です。ただし、表に対して数多くの索引が存在すると、更新、削除、挿入のパフォーマンスは低下します。これは、表に関連する索引も更新する必要があるためです。

オブティマイザは、既存の索引を使用して別の索引を作成できます。この結果、索引作成がはるかに高速になります。

一意索引と非一意索引

索引には、一意索引と非一意索引の 2 種類があります。一意索引によって、索引を定義された列に重複値を持つ行が複数存在しないことが保証されます。非一意索引の場合は、列値にこのような制限はありません。

表には、一意索引を明示的に定義しないことをお勧めします。一意性は厳密に論理的な概念であり、表の定義に関連付けるべきだからです。そのかわりに、必要な列に対して UNIQUE 整合性制約を定義します。Oracle は、一意キーに対して自動的に一意索引を定義することにより、UNIQUE 整合性制約を施行します。

複合索引

「複合索引」(「連結索引」とも呼ばれる) は、表の中の複数の列に対して作成される索引です。複合索引を構成する複数の列は、どんな順序で指定してもかまいません。また、複合索引の列は表の中で隣接している必要もありません。

SELECT 文の WHERE 句で複合索引のすべての列または列の先頭部分を参照する場合には、複合索引によってデータの検索速度を向上させることができます。したがって、定義の中で指定する列の順序が重要です。通常は、最も頻繁にアクセスされる列や選択される列を最初に定義します。

追加情報： 詳細は、『Oracle8i チューニング』を参照してください。


 **図 10-6** は、VENDOR_ID および PART_NO 列に複合索引を設定した VENDOR_PARTS 表を示しています。

図 10-6 索引、主キー、一意キーおよび外部キー

VENDOR_PARTS		
VEND ID	PART NO	UNIT COST
1012	10-440	.25
1012	10-441	.39
1012	457	4.95
1010	10-440	.27
1010	457	5.10
1220	08-300	1.33
1012	08-300	1.19
1292	457	5.28

連結索引
(複数列を持つ索引)

通常の複合索引は最大 32 列で形成されます。また、ビットマップ索引の場合、最大列数は 30 です。キー値は、データ・ブロックの利用可能なディスク領域のおよそ 2 分の 1 (オーバーヘッド分を差し引いたもの) を超えることができません。

索引とキー

「索引」と「キー」という 2 つの用語は同じ意味で使用されることがありますが、これらの用語の区別を理解しておく必要があります。「索引」は、データベース内に実際に格納される構造体であり、ユーザーは SQL 文を使用して作成、変更および削除します。作成された索引により、表データへの高速なアクセス・パスが提供されます。一方、「キー」は厳密に論理的な概念です。キーは、整合性制約に対応しています。これは、データベースの業務規則を施行する Oracle のもう 1 つの機能です (第 28 章「データの整合性」を参照)。

いくつかのタイプの整合性制約は、索引によって施行されるため、キーと索引という用語が同じ意味で使用されることもありますが、これらの用語を混同しないようにしてください。

索引と NULL

索引内に複数の NULL 値を持つ行が存在する場合、それぞれの行は別個のものと見なされます。ただし、索引内に NULL でない同一の値を持つ行が 2 行以上存在する場合は、それらの行は同一と見なされます。したがって、UNIQUE 索引では、NULL 値を含む行は同一のものとして扱われなくなります。この規則は、NULL 以外の値が存在しない場合、つまり、行全体が NULL の場合には適用されません。

表のうち、すべてのキー列が NULL の表の行には索引は設定されません。ただし、ビットマップ索引の場合 (10-34 ページの「ビットマップ索引と NULL」を参照) や、クラスタ・キーの列値が NULL の場合は例外です。

ファンクションベース索引

表に索引を設定する場合、その表の1つ以上の列を含むファンクションと式について、索引を作成できます。「ファンクションベース索引」では、ファンクションや式の値が事前に計算され、索引に格納されます。ファンクションベース索引は、B* ツリーまたはビットマップ索引として作成できます（10-30 ページの「[ビットマップ索引](#)」を参照）。

索引作成に使用するファンクションには、算術式や、PL/SQL 関数、パッケージ・ファンクション、C コールアウトまたは SQL 関数を含む式を使用できます。集計関数を含むことはできず、DETERMINISTIC にする必要があります（23-9 ページの「[DETERMINISTIC 関数](#)」を参照）。オブジェクト型を含む列の索引を作成する場合は、マップ・メソッドなど、そのオブジェクトのメソッドをファンクションとして使用できます。ただし、LOB 列、REF またはネストした表の列には、ファンクションベース索引を作成できません。また、オブジェクト型に LOB、REF またはネストした表が含まれる場合も作成できません。

ファンクションベース索引の使用方法

ファンクションベース索引は、WHERE 句にファンクションを含む文を効率的に評価するメカニズムを提供します。ファンクションベース索引を作成して、SELECT 文と DELETE 文を処理するときに式の値を計算しなくてもよいように、計算集中型の式を具象化できます。ただし、INSERT 文と UPDATE 文の処理中には、文を処理するために従来どおりファンクションを評価する必要があります。

たとえば、次の索引を作成する場合を考えます。

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
```

これにより、次のような問合せを処理するときに、この索引を使用できます。

```
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

UPPER(*column_name*) または LOWER(*column_name*) でファンクションベース索引を定義すると、大 / 小文字区別の検索が容易になります。たとえば、次の索引により、

```
CREATE INDEX uppercase_idx ON emp (UPPER(empname));
```

次のような問合せの処理が容易になります。

```
SELECT * FROM emp WHERE UPPER(empname) = 'RICHARD';
```

また、ファンクションベース索引は、SQL 文に効率的な言語照合機能を提供する NLS ソート索引にも使用できます。

追加情報： NLS ソート索引の詳細は、『Oracle8i NLS ガイド』を参照してください。

ファンクションベース索引による最適化

オブティマイザ用に、ファンクションベース索引の統計を収集する必要があります (22-9 ページの「[コストベース最適化の統計](#)」を参照)。この情報がなければ、この種の索引は SQL 文の処理に使用できません。ルールベース最適化には、ファンクションベース索引は使用されません。

コストベース最適化では、WHERE 句に式を持つ問合せに、ファンクションベース索引に対する索引範囲スキャンを使用できます。(23-34 ページの「[索引走査](#)」および 23-35 ページの「[アクセス・パス](#)」を参照してください。) たとえば、次の問合せでは、

```
select * from T where a + b < 10;
```

a+b の索引が作成されていれば、オブティマイザは索引範囲スキャンを使用できます。範囲スキャンのアクセス・パスは、述語 (WHERE 句) の選択性が低い場合に特に便利です。また、式がファンクションベース索引で具象化されていれば、より正確な式を必要とする述語の選択性をオブティマイザで見積もることができます。

オブティマイザは、SQL 文の式を解析し、文の式ツリーとファンクションベース索引を比較して、式のマッチングを実行します。この比較は大 / 小文字が区別され、ブランクは無視されます。オブティマイザによる式の評価方法の詳細は、23-4 ページの「[式と条件の評価](#)」を参照してください。

ファンクションベース索引の依存性

ファンクションベース索引は、その索引を定義している式に使用されたファンクションに依存しています。ファンクションが PL/SQL 関数またはパッケージ・ファンクションの場合は、ファンクションの仕様に変更があると索引は使用禁止になります。

ファンクションベース索引の定義には、PL/SQL 関数 DETERMINISTIC を使用する必要があります (23-9 ページの「[DETERMINISTIC 関数](#)」を参照)。索引所有者には、定義する関数に対する EXECUTE 権限が必要です。EXECUTE 権限が取り消されると、ファンクションベース索引に DISABLED マークが設定されます。

ファンクションベース索引の依存性と権限の詳細は、21-7 ページの「[ファンクションベース索引の依存性](#)」を参照してください。

索引の格納方法

索引を作成すると、索引のデータを保持するための索引セグメントが表領域内に自動的に割り当てられます。索引セグメントへの領域の割当てと、確保された領域の使用は、次の方法で制御できます。

- 索引セグメントのエクステントの割当ては、その索引セグメントに対して記憶領域パラメータを設定することにより制御できる。
- 索引セグメントのエクステントを構成するデータ・ブロック内の空き領域は、その索引セグメントの PCTFREE パラメータを設定することによって制御できる。

索引セグメントの表領域は、所有者のデフォルト表領域または CREATE INDEX 文で指定された表領域です。索引を、その索引に対応する表と同じ表領域に格納する必要はありません。また、Oracle では索引と表のデータの両方を並行して検索できるため、索引とその索引に対応する表をそれぞれ別のディスク・ドライブ上の異なる表領域内に格納すると、問合せのパフォーマンスが向上します。29-13 ページの「[ユーザー表領域の設定と割当て制限](#)」を参照してください。

索引ブロックの形式

索引データに対して使用可能な領域は、Oracle ブロック・サイズから、ブロック・オーバーヘッド、エントリ・オーバーヘッド、ROWID および索引を付ける値 1 つあたり 1 バイトの合計を引いたものです。索引ブロックのオーバーヘッドに必要なバイト数は、オペレーティング・システムによって異なります。

追加情報： 索引ブロックのオーバーヘッドの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

索引の作成時には、索引を付ける列がフェッチされてソートされます。次に、各行について ROWID と索引値と一緒に格納されます。その後、索引が一番下から順にロードされます。たとえば、次の文を考えてみます。

```
CREATE INDEX emp_ename ON emp(ename);
```

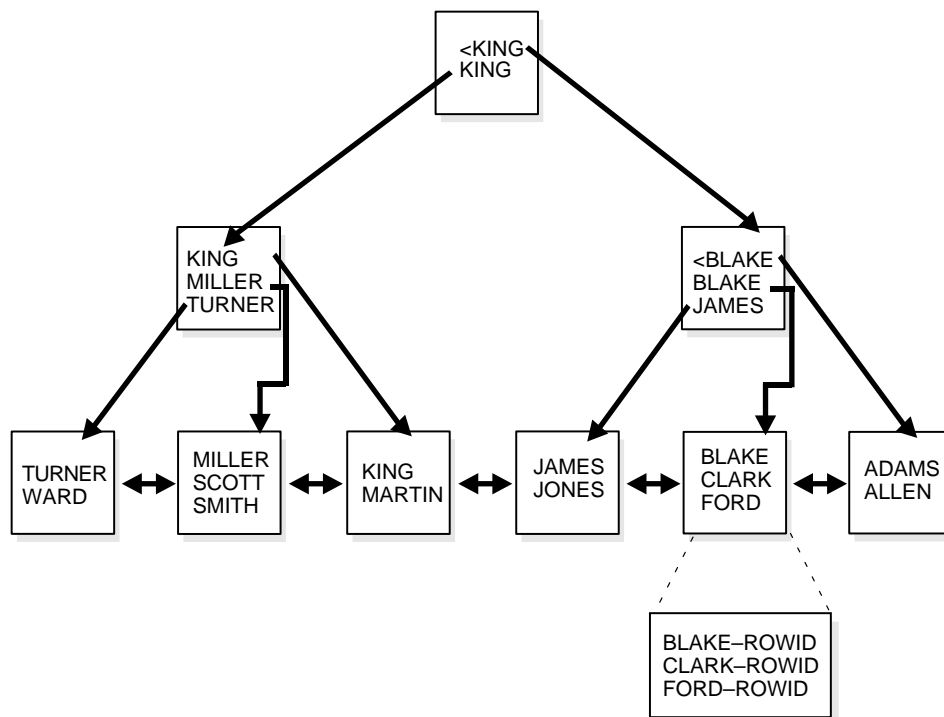
Oracle は、ENAME 列に基づいて EMP 表をソートし、次にこのソート順に、ENAME とそれに対応する ROWID という形で索引をロードします。索引を使用する場合、Oracle は、ソートされた ENAME 値を迅速に検索し、その ENAME 値に対応する ROWID 値を使用して、求める ENAME 列を持つ行を見つけます。

Oracle は CREATE INDEX コマンドのキーワードとして ASC、DESC、COMPRESS および NOCOMPRESS を受け入れますが、これらのキーワードは後方圧縮を使用してリーフ・ノードではなくブランチ・ノードに格納されている索引データには作用しません。

索引の内部構造

Oracle は、すべての行へのアクセス時間を均一化するために B* ツリー索引を使用します。B* ツリー索引の理論は、このマニュアルでは説明しません。詳細は、データ構造を扱ったコンピュータ・サイエンスの文献を参照してください。図 10-7 に、B* ツリー索引の構造を示します。

図 10-7 B* ツリー索引の内部構造



B* ツリー索引の上位ブロック（「ブランチ・ブロック」）には、下位レベルの索引ブロックを指す索引データが含まれます。最下位レベルの索引ブロック（「リーフ・ブロック」）には、すべての索引対象のデータ値と、実際の行を検出するための ROWID が含まれます。リーフ・ブロックは、二重にリンクされます。文字データを持つ列の索引は、データベースのキャラクター・セットにおける文字のバイナリ値を基盤にしています。

一意索引の場合、データ値ごとに ROWID が 1 つ存在します。非一意索引の場合、ROWID はソートするためのキーに含まれるため、非一意索引は索引キーと ROWID によってソートされます。クラスタ索引を除いて、NULL のみを含んでいるキーに索引は定義できません。2 つの行に NULL のみを含めても、一意索引には違反しません。

B* ツリー構造の利点

B* ツリーの利点は、次のとおりです。

- ツリーのすべてのリーフ・ブロックは同じ深さであるため、索引内のどの位置にあるレコードを検索する場合にも、所要時間はほぼ同じになる。
- B* ツリー索引は自動的にバランスが保たれる。
- B* ツリーのすべてのブロックの平均 4 分の 3 が満杯になる。
- 完全一致や範囲検索など、広範囲な問合せに対して、B* ツリーは優れた検索パフォーマンスを提供する。
- 挿入、更新および削除が効率的で、高速検索のためにキー順序が維持される。
- B* ツリーのパフォーマンスは、小さな表と大きな表のどちらでも優れているため、表のサイズが大きくなっても低下しない。

キー圧縮

キー圧縮により、索引（または索引構成表）内で主キーの列値の各部を圧縮し、繰り返される値による記憶領域のオーバーヘッドが低減します。

通常、索引のキーには、グループ化要素および一意要素という 2 つの要素があります。一意要素を持つようにキーを定義しなければ、グループ化要素に ROWID を追加して一意要素を提供します。キー圧縮は、複数の一意要素で共有できるように、グループ化要素を分割し、格納する方法です。

プレフィクス・エントリとサフィクス・エントリ

キー圧縮により、索引キーがプレフィクス・エントリ（グループ化要素）とサフィクス・エントリ（一意要素）に分割されます。圧縮するために、プレフィクス・エントリが索引ブロック内のサフィクス・エントリ間で共有されます。圧縮されるのは、B* ツリー索引のリーフ・ブロックのキーのみです。ブランチ・ブロックの場合、キーの接尾辞は切り捨てることができますが、キーは圧縮されません。

キー圧縮は、複数の索引ブロック間ではなく、1 つの索引ブロック内で実行されます。サフィクス・エントリは、索引行の圧縮版を形成します。各サフィクス・エントリは、同じ索引ブロックに格納されているプレフィクス・エントリを参照します。

デフォルトで、接頭辞は、最終列を除く、すべてのキー列で構成されます。たとえば、3 列（column1, column2, column3）からなるキーの場合、デフォルトの接頭辞は（column1, column2）です。値リスト（1,2,3）（1,2,4）（1,2,7）（1,3,5）（1,3,4）（1,4,4）の場合は、接頭辞内で重複する（1,2）（1,3）が圧縮されます。

また、接頭辞の長さは、接頭辞に含まれる列数として指定できます。たとえば、接頭辞の長さを 1 と指定すると、その接頭辞は column1、接尾辞は（column2, column3）となります。値リスト（1,2,3）（1,2,4）（1,2,7）（1,3,5）（1,3,4）（1,4,4）の場合は、接頭辞内で重複する 1 が圧縮されます。

非一意索引の接頭辞の最大長はキー列の数であり、一意索引の接頭辞の最大長は、キー列の数から 1 を差し引いた値です。

プレフィクス・エントリは、それと等しい値を持つプレフィクス・エントリが索引ブロックに含まれていない場合にのみ、索引ブロックに書き込まれます。プレフィクス・エントリは、索引ブロックに書き込まれた直後から共有可能になり、最後に削除した参照側のサフィクス・エントリが索引ブロックから消去されるまで使用可能のままです。

パフォーマンスと記憶領域に関する考慮事項

キー圧縮を使用すると、領域が大幅に節約になり、索引ブロックあたりで格納できるキー数が増え、I/O が減少してパフォーマンスが向上します。

ただし、索引の記憶領域必要量は減少しますが、索引スキャン中にキー列値を再構築するための CPU 時間が増大することがあります。また、各プレフィクス・エントリには、対応する 4 バイトのオーバーヘッドが生じるため、記憶領域のオーバーヘッドが大きくなります。

キー圧縮の使用方法

キー圧縮は、次のようにさまざまな状況で役立ちます。

- 通常の非一意索引の場合、重複キーが格納されて ROWID がキーに追加され、重複行が分割される。キー圧縮を使用すると、重複キーは ROWID を除きプレフィクス・エントリとして索引ブロックに格納されます。残りの行は、ROWID のみからなるサフィクス・エントリです。
- これと同じ動作は、(stock_ticker, transaction_time) など、(item, timestamp) 形式のキーを持つ一意索引にも見られる。多数の行が同じ stock_ticker 値を持ち、transaction_time によって一意性が保たれます。特定の索引ブロックでは、stock_ticker 値がプレフィクス・エントリとして 1 度だけ格納されます。索引ブロックの他のエントリでは、transaction_time 値が共通の stock_ticker プレフィクス・エントリを参照するサフィクス・エントリとして格納されます。
- VARRAY または NESTED TABLE データ型を含む索引構成表の場合は、コレクション・データ型の要素ごとにオブジェクト ID (OID) が繰り返されます。キー圧縮を使用すると、重複する OID 値を圧縮できます。(10-35 ページの「索引構成表」を参照。)

ただし、キー圧縮を使用できない場合があります。たとえば、単一の属性キーを持つ一意索引の場合、一意要素はありますが、共有するグループ化要素がないため、キー圧縮は使用できません。

逆キー索引

「逆キー索引」を作成する場合は、標準の索引とは違って、列の順序は保ちながら、索引が定義されている各列 (ROWID を除く) のバイトを逆にします。索引に対する変更が少数のリーフ・ブロックに集中する Oracle Parallel Server 環境では、この機能によりパフォーマンスの低下を防ぐことができます。索引のキーを逆にすることにより、挿入値は索引のリーフ・キー全体に分散されます。

逆キーを使用すると、その索引に対する索引範囲走査は実行できなくなります。逆キー索引では辞書的に連続しているキーが隣接して格納されないため、キー指定フェッチまたは全索引（表）走査しか実行できません。

状況によっては、逆キー索引を使った方が OLTP Oracle Parallel Server アプリケーションのパフォーマンスが高速になることもあります。たとえば、Oracle Office でメール・メッセージに対する索引を保持する場合、古いメッセージを保持するユーザーもいるので、索引は最新のメッセージだけでなく、それら古いメッセージに対するポインタもメンテナンスしなければなりません。

REVERSE キーワードは、逆キー索引を作成するための簡単なメカニズムを備えています。CREATE INDEX 文のオプションの索引仕様部に REVERSE キーワードを指定します。

```
CREATE INDEX i ON t (a,b,c) REVERSE;
```

逆キー索引を標準の索引に REBUILD（再構築）するには、キーワード NOREVERSE を指定します。

```
ALTER INDEX i REBUILD NOREVERSE;
```

NOREVERSE キーワードを指定しないで逆キー索引を再構築すると、逆キー索引が再構築されて生成されます。標準の索引を逆キー索引として再構築することはできません。その場合は、CREATE コマンドを使用してください。

ビットマップ索引

注意： ビットマップ索引は、Oracle8i Enterprise Edition を購入した場合にのみ利用できます。

索引を使用する目的は、特定のキー値が含まれる行へのポインタを提供することです。通常の索引では、そのために、キー値と、そのキー値を持つ行の ROWID の対応リストを格納します。（Oracle は、キー値とそれに対応する ROWID を繰り返し格納します。）「ビットマップ索引」では、ROWID のリストのかわりに、各キー値のビットマップを使用します。

ビットマップ中の各ビットは、該当する可能性のある ROWID に対応しています。あるビットが設定されていれば、それに対応する ROWID の行には該当するキー値が含まれています。マッピング機能がビット位置を実際の ROWID に変換するので、内部的には別の表現が使用されていても、ビットマップ索引は通常の索引と同じ機能を果たします。異なるキー値の数が多くなければ、ビットマップ索引は領域を有効に使用できます。

ビットマップ索引は、WHERE 句に指定されたいくつかの条件に対応する索引を効率的にマージします。一部の条件は満たしているがすべては満たしていない行については、表自体にアクセスする前に除外します。これによって、応答時間が短縮されます。多くの場合は劇的に改善されます。

データ・ウェアハウス・アプリケーションの場合の利点

ビットマップ索引は、大量のデータと非定型問合せを扱うものの、同時実行トランザクションのレベルは高くないデータ・ウェアハウス・アプリケーションに対して効果的です。この種のアプリケーションに対するビットマップ索引の利点は次のとおりです。

- かなりの種類の非定型問合せの応答時間が短縮される。
- 他の方式の索引と比べて使用領域が実質的に縮小される。
- 機能の低いハードウェアでもパフォーマンスが劇的に向上する。
- パラレル DML とロードを効果的に実行できる。

従来の B* ツリー索引を使用して大きな表に完全な索引を作成すると、領域という面で膨大なコストがかかります。索引は表中のデータの何倍にも膨れ上がることがあるからです。それに対して、ビットマップ索引は通常、表中で索引を付けるデータのサイズより小さくて済みます。

ビットマップ索引は、数多くの並行トランザクションがデータを更新する OLTP アプリケーションには適していません。むしろ、ユーザーがデータの更新より問合せを実行することが多い、データ・ウェアハウス・アプリケーションの意思決定支援システム (DSS) に適しています。

ビットマップ索引は、Oracle のコストベース最適化アプローチおよび実行エンジンと統合されます。また、他の Oracle 実行メソッドとシームレスに組み合わせて使用できます。たとえば、オプティマイザが 2 つの表をハッシュ結合することを決めたとします。片方の表ではビットマップ索引を、もう一方の表では通常の B* ツリー索引を使用します。オプティマイザはビットマップ索引と他の使用可能なアクセス方法 (通常の B* ツリー索引や全表走査など) を検討し、可能な場合にはパラレル化も考慮に入れながら、最適な方法を選択します。

パラレル問合せおよびパラレル DML は、従来の索引だけでなく、ビットマップ索引に対しても機能します。(パーティション表のビットマップ索引は、ローカル索引でなければなりません。詳細は、11-28 ページの「[索引のパーティション化](#)」を参照してください。) 索引のパラレル作成と連結索引もサポートされています。

カーディナリティ

カーディナリティの低い列ほど、ビットマップ索引を使ったときの効果は大きくなります。カーディナリティの低い列とは、表全体の行数と比べて個別の値の数が少ない列のことです。ある列に同じ値が 100 個以上含まれていれば、その列はビットマップ索引の候補です。値の繰返しが少ない (つまりカーディナリティが高い) 列でも、その列が問合せの WHERE 句で複雑な条件に含まれることが頻繁にあるなら、ビットマップ索引の候補になります。

たとえば、100 万行ある表で個別の値が 10,000 個ある列は、ビットマップ索引の候補です。この列に対しては、B* ツリー索引よりビットマップ索引の方が効果的です。この列を他の列と組み合わせて問い合わせることが多い場合は、特に効果的です。

B* ツリー索引は、カーディナリティの高いデータ、つまり個別の値が多いデータ (CUSTOMER_NAME や PHONE_NUMBER など) に対して有効です。通常の B* ツリー索引は、索引を付けるデータよりも数倍大きくなることがあります。一方、ビットマップ索引は、適切に使用すれば B* ツリー索引より相当小さなサイズですみます。

非定型の問合せ（またそれと同様の状況）では、ビットマップ索引を使用すると問合せのパフォーマンスが劇的に向上します。問合せの WHERE 句の AND 条件と OR 条件については、結果のビットマップを ROWID に変換する前に、対応するブール演算をビットマップに直接実行することによって解決をスピードアップできます。結果の行数が少なければ、全表走査を行う必要はないので、問合せはすぐに終了します。

ビットマップ索引の例

表 10-2 に、ある会社の顧客データの一部を示します。

表 10-2 ビットマップ索引の例

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

MARITAL_STATUS、REGION、GENDER および INCOME_LEVEL はすべてカーディナリティの低い列（最初の 2 つには値が 3 つ、3 番目には 2 つ、4 番目には 4 つしかない）なので、これらの列にはビットマップ索引が適しています。一方、CUSTOMER# はカーディナリティの高い列なので、ビットマップ索引は作成しません。この場合は、B* ツリー索引を作成する方が、表示と検索が効率的になります。

表 10-3 に、この例の REGION 列に対するビットマップ索引を示します。これは、各地域に対応する 3 つのビットマップからなっています。

表 10-3 サンプル・ビットマップ

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

ビットマップの各項目（または「ビット」）は CUSTOMER 表の 1 つの行に対応しており、各ビットの値は対応する行の値に依存しています。たとえば、REGION='east' というビットマップの最初のビットは 1 です。CUSTOMER 表の最初の行で region が "east" だからです。REGION の値が "east" の行は他にないので、ビットマップ REGION='east' の残りのビットは 0 です。

顧客の人口統計を調べているアナリストが、「既婚の顧客のうち central または west 地域に住んでいる人はどれくらいいるのか」尋ねてきたとします。この質問は、次の SQL 問合せで表現できます。

```
SELECT COUNT(*) FROM CUSTOMER
WHERE MARITAL_STATUS = 'married' AND REGION IN ('central','west');
```

ビットマップ索引を使用すると、図 10-8 に示すとおり、結果のビットマップから該当する値を数えるだけで、この問合せを非常に効果的に処理できます。基準に該当する特定の顧客を識別するときは、結果のビットマップを使用して表にアクセスできます。

図 10-8 ビットマップ索引を使用した問合せの実行

status = 'married'			region = 'central'			region = 'west'		
0			0		0	0		0
1			1		0	1		1
1			0		1	1		1
0	AND		0	OR		0	AND	
0			1		0	0		0
1			1		0	1		1

ビットマップ索引と NULL

他のほとんどのタイプの索引とは異なり、ビットマップ索引には NULL 値を持つ行が含まれます。NULL の索引作成は、集計関数 COUNT を使った問合せなど、いくつかのタイプの SQL 文に使用できます。

例 1

```
SELECT COUNT(*) FROM EMP;
```

NULL データを持つ行を含め、すべての表の行の索引が作成されるので、この問合せには任意のビットマップ索引を使用できます。NULL の索引が作成されていない場合、オプティマイザは NOT NULL 制約を持つ列の索引のみを使用できます。

例 2

```
SELECT COUNT(*) FROM EMP WHERE COMM IS NULL;
```

この問合せは、COMM のビットマップ索引によって最適化できます。

例 3

```
SELECT COUNT(*) FROM CUSTOMER WHERE GENDER = 'M' AND STATE != 'CA';
```

この問合せは、GENDER = 'M' のビットマップを検索し、STATE = 'CA' のビットマップを除くことによって回答されます。STATE に NULL 値が含まれる場合（つまり、NOT NULL 制約がない場合）は、STATE = 'NULL' のビットマップも結果から除く必要があります。

パーティション表に対するビットマップ索引

他の索引の場合と同じように、ビットマップ索引もパーティション表に対して作成できます。ただし、制限が1つだけあります。ビットマップ索引はパーティション表に対してローカルでなければなりません。グローバル索引にはできません。(グローバル・ビットマップ索引は、パーティション化されていない表でのみサポートされます。)

パーティション表と、ローカルおよびグローバル索引の詳細は、[第11章「パーティション表とパーティション索引」](#)を参照してください。

追加情報：『Oracle8i チューニング』には、ビットマップ索引の使用方法が記載されています。

索引構成表

「索引構成表」が通常の表と違うのは、表のデータが、対応付けられた索引に保持される点です。行の追加、更新または削除など、表データを変更しても、更新されるのは索引のみです。

索引構成表は、1つ以上の列に対して1つの索引が定義された通常の表と似ています。ただし、データベース・システムは、表本体とB* ツリー索引のために2つの領域をメンテナンスするのではなく、コード化されたキー値と、それに対応する行の関連列値が含まれた1つのB* ツリー索引をメンテナンスします。索引エントリの2番目の要素として行のROWIDを格納するかわりに、実際のデータ行をB* ツリー索引に格納します。データ行は表の主キーに対して作成され、各B* ツリー索引エントリには<primary_key_value, non_primary_key_column_values>が含まれます。

索引構成表は、主キーまたはその有効な接頭辞である任意のキーを使用してデータにアクセスするときに適しています。非キー列値はキーと一緒に格納されるため、キー値の重複はありません。他の列から効率的にアクセスできるように、2次索引を作成できます(10-37 ページの「[索引構成表の2次索引](#)」を参照)。

アプリケーションは、通常の表と同じように、SQL 文を使用して索引構成表を操作します。ただし、データベース・システムは、対応するB* ツリー索引を操作することですべての操作を実行します。

表 10-4 に、索引構成表と通常の表の違いをまとめます。

表 10-4 索引構成表と通常の表の比較

通常の表	索引構成表
ROWID が行を一意に識別する。任意で、主キーを指定できます。	主キーが行を一意に識別する。主キーは必ず指定する必要があります。
ROWID 疑似列の物理 ROWID で 2 次索引を作成できる。	ROWID 疑似列の論理 ROWID で 2 次索引を作成できる。
ROWID に基づいてアクセスする。	主キーに基づいてアクセスする。
順次走査はすべての行を戻す。	全索引走査は主キーの順序ですべての行を戻す。
UNIQUE 制約とトリガーを定義できる。	UNIQUE 制約は定義できないがトリガーは定義できる。
他の表とともにクラスタに格納できる。	クラスタには格納できない。
LONG データ型の列と LOB データ型の列を格納できる。	LOB 列は格納できるが、LONG 列は格納できない。
分散とレプリケーションがサポートされる。	分散とレプリケーションがサポートされない。

追加情報： 索引構成表を作成およびメンテナンスする方法は、『Oracle8i 管理者ガイド』を参照してください。

索引構成表の利点

索引構成表では、行が索引に格納されるので、完全一致、範囲検索またはその両方を使用した問合せの場合、キーを使用したデータ・アクセスが高速です。通常の表や索引のように、キー列が重複しないため、記憶領域必要量は軽減されます。キーとともに索引構成表に格納されるデータ行には、非キー列値のみが含まれます。また、キーとともにデータ行を格納することにより、通常の表の索引が必要とする物理 ROWID 用の追加の記憶領域が不要になります。物理 ROWID は、キー値を表中の対応する行にリンクします。

コレクション・データ型とキー圧縮

索引構成表には、コレクションの各要素のオブジェクト ID (OID) が格納される、コレクション・データ型 VARRAY および NESTED TABLE を含めることができます。OID はコレクションのすべての要素に繰り返されるため、OID に必要な要素あたり 16 バイトの記憶領域オーバーヘッドは不要です。キー圧縮により、索引構成表のリーフ・ブロック内で重複する OID 値を圧縮できます。10-28 ページの「[キー圧縮](#)」を参照してください。

行オーバーフロー領域付きの索引構成表

B* ツリー索引エントリは `<key, ROWID>` の組のみで成り立っているため、サイズは小さくすみます。これに対して、索引構成表の場合、B* ツリー索引エントリは `<key, non_key_column_values>` の組から成り立っているため、サイズは大きくなります。索引エントリが極端に大きくなると、リーフ・ノードは 1 行または行の断片を格納するだけで満杯になり、高い密度で格納されるという B* ツリー索引の特徴が失われます。

この問題を処理するために OVERFLOW 句が用意されています。オーバーフロー表領域としきい値を指定できます。しきい値は、ブロック・サイズのパーセント (PCTTHRESHOLD) で指定します。

指定されたしきい値よりも行サイズが大きくなると、その行の非キー列値は、指定されたオーバーフロー表領域に格納されます。その場合、索引エントリには `<key, rowhead>` の組が含まれます。rowhead は、残りの列の先頭部分を表します。通常の行断片と似ていますが、rowhead は残りの列値が含まれているオーバーフロー行断片を指すところが異なります。

追加情報： OVERFLOW 句の使用例は、『Oracle8i 管理者ガイド』を参照してください。

索引構成表の 2 次索引

索引構成表の 2 次索引がサポートされることにより、主キーでもその接頭辞でもない列を使用して、索引構成表に効率よくアクセスできます。

Oracle は、論理行識別子を使用して索引構成表を組み立てます。この識別子は、表の主キーに基づいており、「論理 ROWID」と呼ばれます。論理 ROWID には、必要に応じて、行のブロック位置を識別する「物理推測」が含まれます。Oracle は、これらの推測を使用して、主キー検索をバイパスし、索引構成表のリーフ・ブロックを直接ブロープできます。ただし、索引構成表の行には永続物理アドレスがないため、行が新しいブロックに移動すると推測が陳腐化することがあります。

通常の表の場合、2 次索引によるアクセスでは、行を含むデータ・ブロックをフェッチするために、2 次索引のスキャンと付加的な I/O が必要になります。索引構成表の場合、2 次索引によるアクセスは、次のように物理推測を使用するかどうかと、その正確さに応じて異なります。

- 推測を使用しない場合、アクセス時には、主キー索引のスキャンとそれに続く 2 次索引スキャンという 2 つの索引スキャンが実行される。
- 正確な推測を使用する場合、アクセス時には、2 次索引スキャンと追加の I/O によって行を含むデータ・ブロックがフェッチされる。
- 不正確な推測を使用する場合、アクセス時には 2 次索引スキャンと I/O によって間違ったデータ・ブロック（推測が示すブロック）がフェッチされてから、主キー索引がスキャンされる。

詳細は、12-18 ページの「[論理 ROWID](#)」を参照してください。

索引構成表のその他の機能

この項では、索引構成表を活用するための付加的機能について説明します。

索引構成表の再構築

索引構成表を再構築し、増分更新によって生じる断片化を削減できます。索引構成表を再構築するには、ALTER TABLE コマンドで MOVE オプションを指定します。

MOVE オプションを指定すると、索引構成表の主キー索引の B* ツリーが再構築されます。ただし、OVERFLOW 句を明示的に指定するか、ALTER TABLE 文の一部として PCTTHRESHOLD または INCLUDING 列の値を変更しなければ、オーバーフロー・データ・セグメントは再構築されません。また、LOB 列に対応する索引とデータ・セグメントは、LOB 列を明示的に指定しなければ再構築されません。

索引構成表のパラレル作成

CREATE TABLE ... AS SELECT 文を使用すると、索引構成表を作成し、基礎となる副問合せ (AS SELECT) に PARALLEL 句を使用して、それをパラレルにロードできます。この文は、SQL*Loader を使用するパラレル大量ロードの代替方法です。

索引構成表と 2 次索引のパーティション化

列値の範囲を指定して索引構成表をパーティション化できます (11-15 ページの「[レンジ・パーティション化](#)」を参照)。パーティション化列は、主キー列のサブセットを形成する必要があります。

索引構成表では、次のタイプの 2 次索引を列値の範囲でパーティション化できます。

- ローカルの同一キー索引
- ローカルの非同一次キー索引
- グローバルな同一キー索引

詳細は、11-28 ページの「[索引のパーティション化](#)」および 11-41 ページを参照してください。

索引構成表に適したアプリケーション

索引構成表は、特に次のようなアプリケーションに適しています。

- 情報検索 (IR) アプリケーション
- 空間データ・アプリケーション
- OLAP アプリケーション

情報検索アプリケーション

「情報検索 (IR) アプリケーション」は、ドキュメント・コレクションに対する内容ベース検索をサポートします。そのために、IR アプリケーションは、ドキュメント・コレクションに対する逆索引をメンテナンスします。「逆索引」には、通常、ドキュメントに含まれる個別のワードごとに、`<token, document_id, occurrence_data>` という形式のエントリが含まれています。内容ベース検索では、アプリケーションは、逆索引を走査して該当するトークン (検索する値) を探します。

逆索引をモデル化する通常の表を定義できます。表データの検索を高速化するために、トークンに対応する列に対して索引を定義することもできます。ただし、この方法には次のような問題点があります。

- 索引を使用して逆索引からオカレンス・データを取り出すと、行あたり 1 つの ROWID ベース・フェッチが余分に増える。通常、内容ベースの IR 問合せでは、指定された問合せ用語を探すためにすべての逆索引エントリをフェッチする必要があります。IR アプリケーションの場合、重複は例外ではなく標準であるため、1 つの問合せ用語につき何千もの重複値が存在することがあります。このため、行オーバーヘッドあたり 1 つの ROWID ベース・フェッチというのはきわめて重大で、IR の検索パフォーマンスに多大な影響を与えます。
- 表および索引にキー (トークン) 列の重複があると、無駄な領域が生じる。逆索引は巨大になることがあるため、記憶領域必要量が許容範囲を超えることがあります。

場合によっては、元になる表の複数列に連結索引を定義することにより、検索パフォーマンスが向上することがあります。連結索引を定義した場合、オカレンス・データが必要でなければ (つまり、ブール問合せ) 索引構成検索が可能です。そのような場合、逆表レコードの ROWID フェッチは行われません。問合せに近接述語 (たとえば、「Oracle Corporation」) が含まれている場合は、連結索引アプローチを使ったとしても、元になる表にアクセスしなければなりません。さらに、連結索引の作成とメンテナンスには、トークンに単一列の索引を定義した場合と比べて、より多くの時間がかかります。また、表と索引でキー (トークン) の複数列が重複するため、格納オーバーヘッドは高くなります。

索引構成表を使用して逆索引を生成すると、上記の問題を回避できます。次のとおりです。

- データ行がキーと一緒に格納されているため、索引のオカレンス・データを検索することには、索引を走査して、該当するリーフ・ノードからデータ行を取得することが含まれる。
- 索引には、非キー列値だけがキーと一緒に格納される。したがって、データの重複はありません。また、通常の表に対してメンテナンスされている索引に必要な、余分な ROWID 領域のオーバーヘッドは回避されます。

また、索引構成表はアプリケーションから参照できるため、協調的索引作成に適しています。協調的索引作成では、アプリケーションとデータベースが協調しながら、アプリケーション固有の索引を管理するからです。

空間データ・アプリケーション

空間データ・アプリケーションは、アプリケーション固有の索引をメンテナンスするためにある種の逆索引を使用するので、索引構成表に適しています。

空間データ・アプリケーションは、空間データ問合せを処理するために逆索引をメンテナンスします。たとえば、グリッドの集合の中に存在するオブジェクトに対する空間データ索引は、次の形式のエントリを持つ逆索引としてモデル化できます。

```
<grid_id, spatial_object_id, spatial_object_data>
```

索引構成表は、必要な検索パフォーマンスと低い領域コストを同時に実現するので、この種の逆索引をモデル化するのに適しています。

OLAP アプリケーション

オンライン分析処理 (OLAP) アプリケーションは、通常、複数ディメンション・ブロックを操作します。複数ディメンション・ブロックの一部分を高速に検索できるように、ディメンション値のセットをページのセットにマップする逆索引をメンテナンスします。

逆索引のエントリの形式は次のとおりです。

```
<dimension_value, list_of_pages>
```

OLAP アプリケーションがメンテナンスする逆索引は、索引構成表として簡単にモデル化できます。

アプリケーション・ドメイン索引

Oracle は、複合データ型 (文書、空間データ、イメージおよびビデオ・クリップなど) の索引に対応し、特化した索引作成テクニックを使用するために、「拡張索引作成機能」を提供します。この機能を使用すると、アプリケーション固有の索引管理ルーチンを「索引タイプ」スキーマ・オブジェクトとしてカプセル化し、オブジェクト型の表の列や属性の「ドメイン索引」(アプリケーション固有の索引) を定義できます。(オブジェクト型とその属性の詳細は、13-3 ページの「[ユーザー定義データ型](#)」を参照)。また、拡張索引作成機能により、アプリケーション固有の「演算子」を効率的に処理できます。

ドメイン索引の構造と内容は、「カートリッジ」と呼ばれるアプリケーション・ソフトウェアによって制御されます。Oracle Server は、ドメイン索引の作成、メンテナンスおよび検索のために、このアプリケーションと対話します。索引構造そのものは、索引構成表として Oracle データベースに格納するか、ファイルとして外部に格納できます。

索引タイプ

「索引タイプ」スキーマ・オブジェクトは、ドメイン索引を管理およびアクセスするルーチンの集合をカプセル化します。索引タイプの目的は、テキスト・データ、空間データ、イメージ・データおよび OLAP データなど、複合ドメインのファンクションを、外部アプリケーション・ソフトウェアを使用して効率的に検索し、取り出せるようにすることです。

索引設計者がインプリメントする必要のあるルーチンは、すべて Oracle Data Cartridge Interface (ODCIIndex) で指定されます。各ルーチンは、型のメソッド (13-3 ページの「[オブジェクト型](#)」を参照) としてインプリメントできます。

索引定義ルーチン

索引定義ルーチンは、CREATE INDEX 文で索引タイプが参照されるとドメイン索引を作成し、ALTER INDEX 文で索引タイプが変更されるとドメイン索引情報を変更します。また、DROP INDEX 文で索引タイプが削除されると索引情報を削除し、実表が切り捨てられると索引を切り捨てます。

索引メンテナンス・ルーチン

索引メンテナンス・ルーチンは、実表の行が挿入、削除、更新またはロードされると、ドメイン索引の内容をメンテナンスします。

索引スキャン・ルーチン

索引スキャン・ルーチンは、ビルトイン演算子またはユーザー定義オペレータを含んだ述語を満たす実表の行をアクセス側 SQL 文で取り出すために、ドメイン索引へのアクセスをインプリメントします。ドメイン索引のスキャンに使用できる演算子の詳細は、10-42 ページの「[ユーザー定義オペレータ](#)」を参照してください。

索引スキャンは、istart、ifetch および iclose という 3 つのルーチンを通じて指定されます。これらのルーチンは、データ構造を初期化し、述語を満たす行をフェッチし、それらの行がすべて戻されるとカーソルをクローズできます。

ドメイン索引

「ドメイン索引」スキーマ・オブジェクトは、索引タイプで指定されたルーチンによって作成、管理およびアクセスされる、アプリケーション固有の索引です。ドメイン索引と呼ばれるのは、アプリケーション固有のドメイン内のデータについて、索引を作成するからです。

現在サポートされているのは、単一列のドメイン索引のみです。単一列のドメイン索引は、スカラー、オブジェクトまたは LOB データ型を持つ列について作成できます。

同じ列に複数のドメイン索引を作成できるのは、索引タイプが異なる場合だけです。この点では、ビルトイン B* ツリー索引メソッドは、別の索引タイプと見なすことができます。

ドメイン索引の格納

ドメイン索引は、索引構成表または外部ファイルに格納できます。拡張可能索引作成用の SQL インタフェースでは、アプリケーションが索引の定義、メンテナンスおよび検索用のプロトコルに準拠していれば、索引データの位置に制限はありません。

ドメイン索引のメタデータ

B* ツリー索引の場合は、USER_INDEXES ビューを問い合わせることで索引情報を取得できます。ドメイン索引にも同様のサポートを提供するために、索引設計者は次の方法でドメイン固有のメタデータを追加できます。

- 索引設計者は、メタデータを含む 1 つ以上の表を定義できる。この表のキー列は、索引の一意識別子にする必要があります。この一意キーは、索引名 (schema.index) でもかまいません。列定義の残りの部分は、索引設計者の裁量で使用できます。
- システム定義のメタデータ表をドメイン索引のメタデータ表と結合するビューを作成し、ドメイン索引のインスタンスごとに包括的な情報の集合を提供できる。ビュー定義は、索引設計者が提供する必要があります。

ユーザー定義オペレータ

Oracle には、算術演算子 (+、-、*、/)、比較演算子 (=、>、<)、論理演算子 (NOT、AND、OR) および集合演算子 (UNION) を含む、ビルトイン演算子の集合が用意されています。これらの演算子は、入力として 1 つ以上の引数 (「オペランド」) をとり、結果を戻します。これらの演算子は、SQL 文では特殊文字 (+) またはキーワード (AND) で表されます。ユーザーとドメイン・カートリッジ作成者は、新しい演算子を定義してビルトイン演算子と同様に SQL 文に使用できます。

「ユーザー定義オペレータ」は、名前で識別されるスキーマ・オブジェクトです。この名前には、文字列、特殊文字または記号を使用できます。ビルトイン演算子と同様に、ユーザー定義オペレータは入力として一連のオペランドを取り、結果を戻します。このオペレータは、ユーザーまたはドメイン・カートリッジ作成者が実現する必要があります。

ユーザー定義オペレータは、ビルトイン演算子を使用できる場所、つまり、次のように問合せまたはデータ操作文の中で式を使用できる場所であれば、どこでも起動できます。

- SELECT コマンドまたはサブ問合せの選択リスト
- WHERE 句の条件
- ORDER BY 句と GROUP BY 句

たとえば、新しいオペレータ Contains を、入力としてテキスト文書とキーワードを取り、文書にそのキーワードが含まれていれば TRUE を戻すように定義する場合は、SQL 問合せを次のように記述できます。

```
SELECT * FROM employees WHERE Contains(resume, 'Oracle and UNIX');
```

オペレータを作成するには、CREATE OPERATOR 文でオペレータ名とそのバインディング（存在する場合）を指定します。オペレータの「バインディング」によって、オペレータはそれを実現するユーザー定義関数に対応付けられます。また、バインディングでは一意の「シグネチャ」（関数に指定する引数のデータ型の順序）を使用してオペレータも識別されます。

シグネチャが異なっていれば、1つのオペレータが複数のバインディングを持つことができます。特定のシグネチャを持つオペレータが起動されると、適切な関数が実行されます。スキーマ内で作成されたオペレータは、同じスキーマまたは異なるスキーマに定義されている関数を使用して評価できます。

オペレータにバインドされているユーザー定義関数は、次のとおりです。

- スタンドアロン関数
- パッケージ関数
- オブジェクト・メンバー・メソッド

たとえば、オペレータ Contains は、Ordsys スキーマ内で、2つのバインディングと、Text および Spatial ドメイン内で実現される、対応する関数を指定して作成できます。

```
CREATE OPERATOR Ordsys.Contains
BINDING
  (VARCHAR2, VARCHAR2) RETURN BOOLEAN USING text.contains,
  (Spatial.Geo, Spatial.Geo) RETURN BOOLEAN USING Spatial.contains;
```

戻り値のデータ型は、オペレータのバインディング宣言の一部として指定されますが、バインディングの一意性を決定するものではありません。ただし、指定する関数の引数と戻り値のデータ型は、オペレータ・バインディングと同じにする必要があります。

また、オペレータは索引を使用して評価できます。Oracle では、いくつかのビルトイン演算子を効率的に評価するために索引を使用します。たとえば、B* ツリー索引を使用すると、比較演算子 =、> および < を評価できます。同様に、ユーザー定義のドメイン索引を使用すると、ユーザー定義オペレータを効率よく評価できます。

索引タイプは、索引タイプ定義にリストされたオペレータの索引インプリメンテーションを提供します。Oracle Server は、索引タイプに指定されたルーチンを起動し、ドメイン索引を検索して候補となる行を識別し、さらに処理（行のフィルタリング、選択およびフェッチ）を実行します。

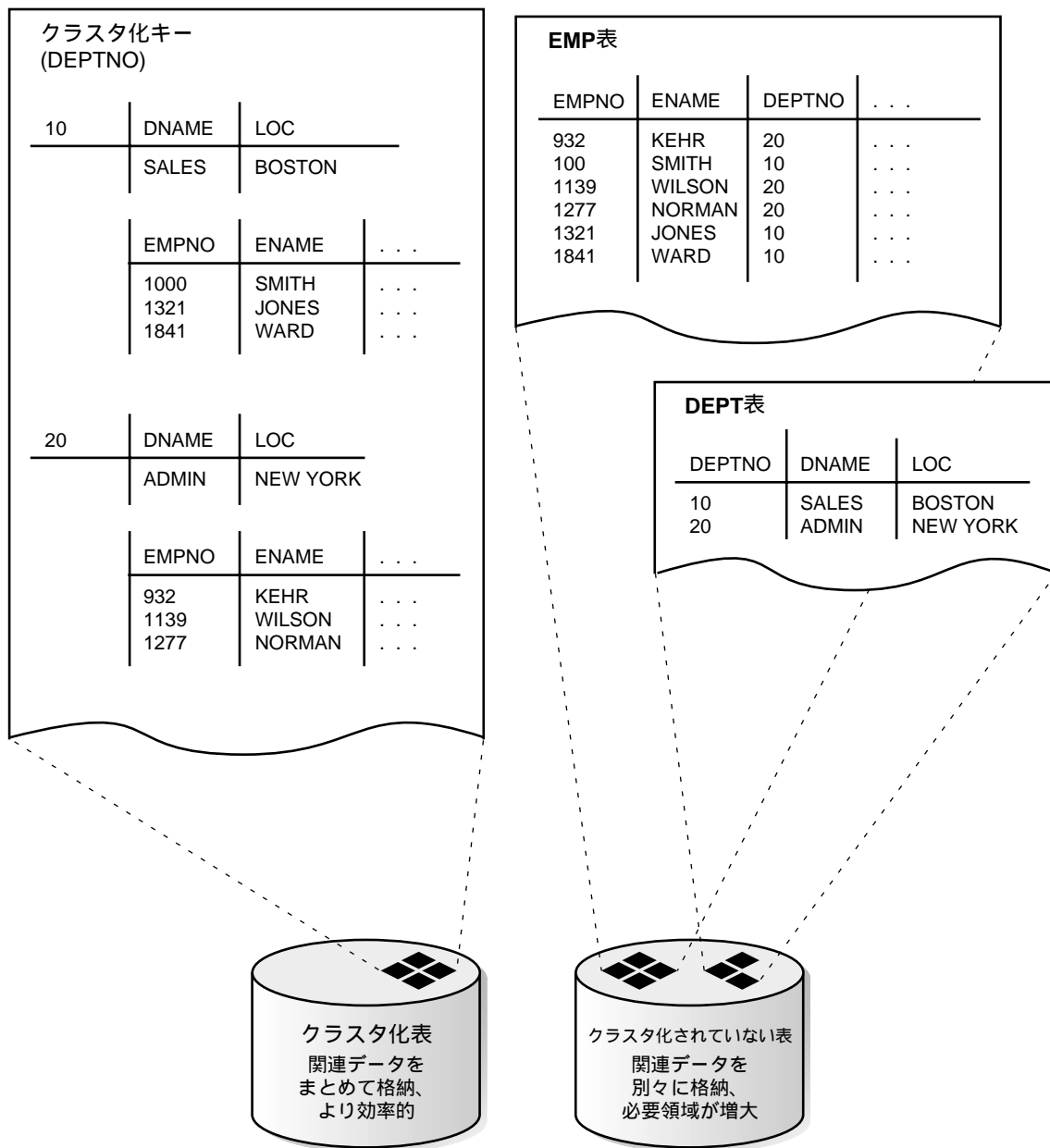
追加情報： 索引タイプ、ドメイン索引およびユーザー定義オペレータの詳細は、『Oracle8i Data Cartridge Developer's Guide』を参照してください。

クラスタ

「クラスタ」は、表データを格納するためのオプションの方法です。クラスタは同じデータ・ブロックを共有する表のグループです。これらの表グループは共通の列を共有し、頻繁に併用されます。

たとえば、EMP 表と DEPT 表は DEPTNO 列を共有しています。EMP 表と DEPT 表をクラスタ化すると（図 10-9 の「[クラスタ化表データ](#)」）、EMP 表と DEPT 表の両方に属する各部門のすべての行が物理的に同じデータ・ブロックに格納されます。

図 10-9 クラスタ化表データ



クラスタは、異なる表の関連した行を、同じデータ・ブロック内にまとめて格納するため、クラスタを適切に使用することには次の2つの基本的な利点があります。

- ディスク I/O が少なくなり、クラスタ化した表を結合するときのアクセス時間が改善される。
- クラスタでは、特定の行のクラスタ・キー列が「クラスタ・キー値」になり、各クラスタ・キー値は、その値を持つ行がいくつかの表に含まれているとしても、クラスタとクラスタ索引にそれぞれ一度だけ格納される。

そのため、クラスタとして関連付けた表データと索引データを格納するために必要な記憶領域は、クラスタ化していない表で必要な記憶領域よりも少なくなる場合があります。たとえば、図 10-9 では、各クラスタ・キー（各 DEPTNO）は、EMP 表と DEPT 表の両方で同じ値を持つ多数の行に対して一度だけ格納されています。

パフォーマンスの考慮事項

それぞれに索引を付けて表に別々に格納した場合と比較すると、クラスタによって INSERT 文のパフォーマンスが低下することがあります。このような欠点は、領域の使用方法和、表を走査するために確認する必要があるブロック数に関係があります。複数の表が各ブロック内にデータを持っているため、クラスタ化した表にデータを格納するには、その同じ表をクラスタ化しない場合よりも多くのブロックを使用する必要があります。

クラスタ化して格納する方がよいデータを識別するには、参照整合性制約によって関連付けられている表と、JOIN を使用して頻繁に一緒にアクセスされる表を見つけます。表データを結合するために使用される列に基づいて表をクラスタ化すると、問合せを処理するためにアクセスする必要があるデータ・ブロックの数が少なくなります。つまり、クラスタ・キーに基づく結合に必要な行がすべて同じブロック内に存在します。したがって、結合のパフォーマンスが改善されます。同様に、個々の表をクラスタ化すると有効な場合もあります。たとえば、EMP 表は、同じ部門の従業員に対する行をクラスタ化するために、DEPTNO 列に基づいてクラスタ化することもできます。アプリケーションが通常は部門ごとに行を処理する場合に、このクラスタ化は有効です。

索引と同じように、クラスタがアプリケーションの設計に影響することはありません。クラスタの存在は、ユーザーとアプリケーションに対して透明です。クラスタ化されていない表に格納されているデータと同様に、クラスタ化表に格納されているデータには SQL を介してアクセスできます。

追加情報： クラスタを使用した場合のパフォーマンスへの影響の詳細は、『Oracle8i チューニング』を参照してください。

クラスタ化されたデータ・ブロックの形式

一般に、クラスタ化されたデータ・ブロックは、クラスタ化されていないデータ・ブロックと同じ形式ですが、表ディレクトリに追加のデータがあります。ただし、同じクラスタ・キー値を共有するすべての行は、同一のデータ・ブロック内に格納されます。

クラスタの作成時に、1つのクラスタ・キー値に対するすべての行を格納するために必要な領域の平均容量を、CREATE CLUSTER コマンドの SIZE パラメータで指定します。SIZE によって、データ・ブロックごとに格納できるクラスタ・キーの最大数が決まります。

たとえば、データ・ブロックごとに使用可能な領域が 1700 バイトで、指定されたクラスタ・キー・サイズが 500 バイトの場合、各データ・ブロックは 3 つのクラスタ・キーに対する行を保持できる可能性があります。SIZE がデータ・ブロックごとに使用可能な領域の容量より大きい場合、各データ・ブロックは 1 つのクラスタ・キー値に対応する行のみを保持します。

データ・ブロックあたりのクラスタ・キー値の最大数は SIZE によって固定されますが、Oracle は各クラスタ・キー値に対する領域を実際に確保するわけではなく、確実にその数のクラスタ・キーを各ブロックに割り当てるわけでもありません。たとえば、SIZE の指定によってデータ・ブロックあたり 3 つのクラスタ・キー値を格納できる場合でも、1 つのクラスタ・キー値の行がブロック内の使用可能な領域をすべて占める可能性があります。特定のキーに対して、1 つのブロックに収まる行数よりも多くの行が存在する場合は、必要に応じてブロックが連鎖されます。

クラスタ・キー値は、データ・ブロックに 1 度だけ格納されます。

クラスタ・キー

「クラスタ・キー」は、クラスタ化表が共通に持つ列、または列のグループです。クラスタを作成するときに、クラスタ・キーの列を指定します。その後、クラスタに追加する表を作成するたびに、その同じ列を指定できます。

クラスタ・キーの一部として（クラスタの作成時に）指定される各列について、クラスタ内に作成するすべての表は、クラスタ・キー列のサイズとタイプに一致する列を持つ必要があります。クラスタ・キーは最大 16 列で形成されます。また、クラスタ・キー値は、データ・ブロック内の使用可能ディスク領域のおよそ 2 分の 1（オーバーヘッド分を差し引いたもの）を超えることができません。クラスタ・キーに、LONG 列または LONG RAW 列を含めることはできません。

表のクラスタ化列のデータ値は更新できます。しかし、データの配置はクラスタ・キーに依存しているため、行のクラスタ・キーを変更すると、Oracle によって行の物理的な再配置が実行されることがあります。そのため、頻繁に更新される列は、クラスタ・キーには適しません。

クラスタ索引

クラスタを作成した後、クラスタ・キー列に対して索引を作成しなければなりません。「クラスタ索引」は、特にクラスタのために定義される索引です。この索引は、各クラスタ・キー値のエントリを含みます。

クラスタ内の行を突き止めるために、クラスタ索引を使用してクラスタ・キー値が検索されます。クラスタ・キー値は、そのキー値に対応付けられているデータ・ブロックへのポイン

タとして機能します。そのため、各行は、最低 2 回の I/O によってアクセスされます（索引内を走査しなければならないレベルの数によっては、2 回より多くなる場合もあります）。

クラスタ索引を作成しないと、そのクラスタ化表に対する DML 文（INSERT 文と SELECT 文を含む）は実行できません。したがって、クラスタ化表のデータは、クラスタ索引が作成されるまでロードできません。

クラスタ索引は、表索引と同じように索引セグメントに格納されます。そのため、クラスタのある表領域に格納し、クラスタ索引を別の表領域に格納できます。

クラスタ索引は、次の点で表索引と異なります。

- すべて NULL で構成されるキーが、クラスタ索引内にエントリを持つ。
- 索引エントリは、特定のクラスタ・キー値に対する連鎖の先頭ブロックを指す。
- クラスタ索引は、クラスタ行あたり 1 エントリではなく、クラスタ・キー値あたり 1 エントリを収録する。
- 表索引がなくても表データにアクセスできるが、クラスタ索引がなければクラスタ化データにアクセスできない。

クラスタ索引を削除すると、クラスタ内のデータはそのまま残りますが、新しいクラスタ索引を作成するまではデータを使用できません。別の表領域にクラスタ索引を移動したり、記憶特性を変更するために、クラスタ索引を削除する場合があります。その場合は、そのクラスタの索引を再作成しなければ、クラスタ内のデータをアクセスできません。

ハッシュ・クラスタ

ハッシングは、データ検索のパフォーマンスを改善するために表データを格納するオプションの方法です。ハッシングを使用するには、「ハッシュ・クラスタ」を作成し、そのクラスタに表をロードします。表の行はハッシュ・クラスタ内に物理的に格納され、ハッシュ関数の結果に従って検索されます。

「ハッシュ関数」は、「ハッシュ値」と呼ばれる、数値の分布を表す値を生成するために使用します。この値は、特定のクラスタ・キー値に基づくものです。ハッシュ・クラスタのキーは、単一の列または複合キー（複数列のキー）にすることができます（索引クラスタのキーと同じ）。ハッシュ・クラスタ内の行を検索または格納するために、Oracle はその行のクラスタ・キー値に対してハッシュ関数を適用します。結果として得られるハッシュ値はクラスタ内の 1 つのデータ・ブロックに対応し、発行された文による読み書きはそのデータ・ブロックに対して実行されます。

ハッシュ・クラスタは、索引を持つクラスタ化されていない表、または索引クラスタにかわるものです。索引付きの表または索引クラスタでは、別個の索引に格納されているキー値に基づいて表の中で行が配置されます。

索引付きの表またはクラスタの行を検索または格納するには、最低 2 回の I/O が実行されます（通常は 2 回以上）。索引内でキー値を検索または格納するために 1 回以上の I/O、表またはクラスタの行を読み書きするためにさらに 1 回の I/O が実行されます。それとは対照的に、ハッシュ・クラスタでは、ハッシュ関数を使用してクラスタ内の行が位置付けられます（I/O は不要）。その結果、ハッシュ・クラスタ内の行を読み書きする場合は、最低 1 回の I/O 操作ですみます。

ハッシュ・クラスタへのデータの格納方法

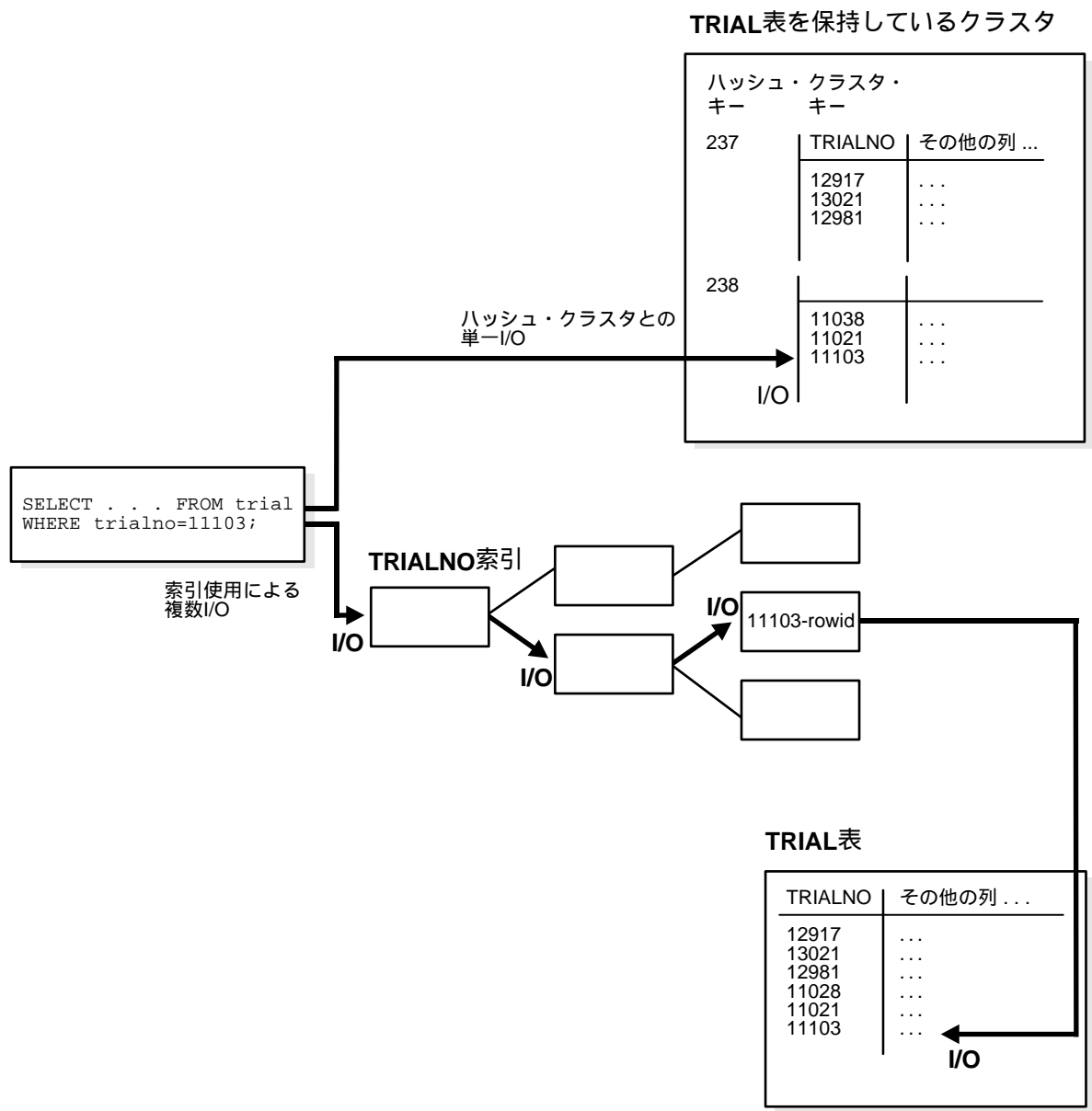
ハッシュ・クラスタは、関連する行をまとめて同じデータ・ブロック内に格納します。ハッシュ・クラスタ内の行は、そのハッシュ値に基づいてまとめて格納されます。

注意： それとは対照的に、索引クラスタは、各行のクラスタ・キー値に基づいて、クラスタ化表の関連する行をまとめて格納します。

ハッシュ・クラスタを作成すると、そのクラスタのデータ・セグメントに対して記憶領域の初期容量が割り当てられます。Oracle によってハッシュ・クラスタに最初に割り当てられる記憶領域は、クラスタ内のハッシュ・キーの行数とサイズの予測値に基づいて決定されます。

[図 10-10](#) は、索引を持つ表とハッシュ・クラスタ内の表のデータ検索を示しています。この後の項では、ハッシュ・クラスタ記憶領域の内部操作について詳しく説明します。

図 10-10 ハッシングと索引作成：データ格納と情報



ハッシュ・キー値

ハッシュ・クラスタ内で行を検索または格納するために、Oracle はハッシュ関数を行のクラスタ・キー値に適用します。結果として得られるハッシュ値は、クラスタ内の 1 つのデータ・ブロックに対応し、Oracle は発行された文による読み書きをそのデータ・ブロックに対して実行します。ハッシュ・クラスタに対するハッシュ値の数は、作成時に固定されており、CREATE CLUSTER コマンドの HASHKEYS パラメータによって決まります。

HASHKEYS の値は、クラスタに対して使用されるハッシュ関数によって生成できる一意のハッシュ値の数を制限します。HASHKEYS に指定した数値は、最も近い素数に丸められます。たとえば、HASHKEYS を 100 に設定すると、任意のクラスタ・キー値に対しハッシュ関数は 0 から 100 までのハッシュ値を生成します（つまり、ハッシュ値は 101 個あります）。

そのため、ハッシュ・クラスタ内の行の分布は、HASHKEYS パラメータに対して設定する値によって直接的に制御されます。所定の行数に対してハッシュ・キー数を増やすと、「衝突」（2 つのクラスタ・キー値が同一のハッシュ値を持っている状態）の可能性が低くなります。衝突したクラスタ・キー値を持つ複数の行を格納するためにオーバーフロー・ブロックが必要になる（つまり、余分な I/O が必要になる）ことがあるので、衝突の数をなるべく少なくすることが重要です。

データ・ブロックごとに割り当てられるハッシュ・キーの最大数は、CREATE CLUSTER コマンドの SIZE パラメータで指定します。SIZE は、各ハッシュ値に対応付けられる行の平均数を格納するために必要な領域の推定合計量（単位はバイト）です。たとえば、データ・ブロックあたりの使用可能な空き領域が 1700 バイトで、SIZE が 500 バイトに設定される場合、データ・ブロックあたり 3 つのハッシュ・キーが割り当てられます。

注意： ハッシュ・クラスタの SIZE パラメータの重要性は、索引クラスタの SIZE パラメータの重要性と類似しています。ただし、索引クラスタでは、同じハッシュ値ではなく同じクラスタ・キー値を持つ行に SIZE が適用されます。

データ・ブロックあたりのハッシュ・キー値の最大数は SIZE によって指定されますが、Oracle はそれぞれのハッシュ・キー値に対する領域をブロック内に実際に確保するわけではありません（ただし、例外は 10-55 ページの「[単一表ハッシュ・クラスタ](#)」を参照）。たとえば、SIZE の指定によればブロックあたり 3 つのハッシュ・キー値を格納できる場合でも、1 つのハッシュ・キー値の行がブロック内の使用可能領域をすべて占める可能性があります。1 つのブロックに収まる行数よりもハッシュ・キー値の行数の方が多い場合、必要に応じてブロックが連鎖されます。

各行のハッシュ値は行の一部としては格納されないので注意してください。各行のクラスタ・キー値が格納されます。したがって、SIZE の適切な値を決めるときには、格納するすべての行についてクラスタ・キー値を含めて考える必要があります。

ハッシュ関数

ハッシュ関数は、クラスタ・キー値に適用されてハッシュ値を戻す関数です。その後、Oracle はハッシュ値を使用して、ハッシュ・クラスタの適切なデータ・ブロック内に行を位置付けます。ハッシュ関数の役割は、クラスタの使用可能なハッシュ値の間で行をできるだけ分散させることです。そのために、ハッシュ関数は衝突の数を最小限に抑える必要があります。

Oracle の内部ハッシュ関数の使用方法

クラスタを作成するときには、Oracle の内部ハッシュ関数を使用しても、使用しなくてもかまいません。内部ハッシュ関数では、クラスタ・キーは単一の列または複合キーです。

さらに、クラスタ・キーは、任意のデータ型（LONG と LONG RAW を除く）の列で構成できます。内部ハッシュ関数は、使用可能なハッシュ・キー間でクラスタ・キー値を十分に分散させ、どんなタイプのクラスタ・キーについても衝突数を最小にします。

クラスタ・キーをハッシュ関数として指定

クラスタ・キーが、その範囲にわたって一様に分布している一意の識別子である場合、内部ハッシュ関数を使用しないで、単にハッシュする列を指定することもできます。

Oracle は、内部ハッシュ関数を使用してハッシュ値を生成するかわりに、クラスタ・キー値をチェックします。クラスタ・キー値が HASHKEYS より小さい場合は、そのクラスタ・キー値をハッシュ値とします。しかし、クラスタ・キー値が HASHKEYS 以上である場合は、そのクラスタ・キー値を HASHKEYS に指定された数値で除算し、その余りをハッシュ値とします。つまり、「ハッシュ値」=「クラスタ・キー値」MOD「ハッシュ・キー数」となります。

クラスタ・キー値がクラスタ内で均等に分散している場合は、CREATE CLUSTER コマンドの HASH IS パラメータを使用してクラスタ・キー列を指定します。指定するクラスタ・キーは、位取りがゼロの数（整数）だけを含む単一列で構成されていなければなりません。内部ハッシュ関数を使用しない場合に整数以外のクラスタ・キー値を指定すると、操作（INSERT または UPDATE 文）はロールバックされ、エラーが戻されます。

ユーザー定義のハッシュ関数の指定

ハッシュ・クラスタのためのハッシュ関数として SQL 式を指定することもできます。クラスタ・キー値がクラスタ内で均等に分散していない場合は、クラスタ行をハッシュ値に効果的に分散させるユーザー固有のハッシュ関数を作成することを考えてください。

たとえば、従業員情報が入っているハッシュ・クラスタがあり、クラスタ・キーが従業員の自宅の市外局番である場合、多くの従業員が同一のハッシュ値にハッシュされる可能性があります。この問題を解決するには、CREATE CLUSTER コマンドの HASH IS 句に次の式を記述します。

```
MOD((emp.home_area_code + emp.home_prefix + emp.home_suffix), 101)
```


この式は最初に市外局番の列を取り込み、これに電話番号の局番の列と番号の列を加算した値をハッシュ値の数（上の例では 101）で除算し、その余りをハッシュ値として使用します。その結果、クラスタ行はさまざまなハッシュ値に、より均等に分散されます。

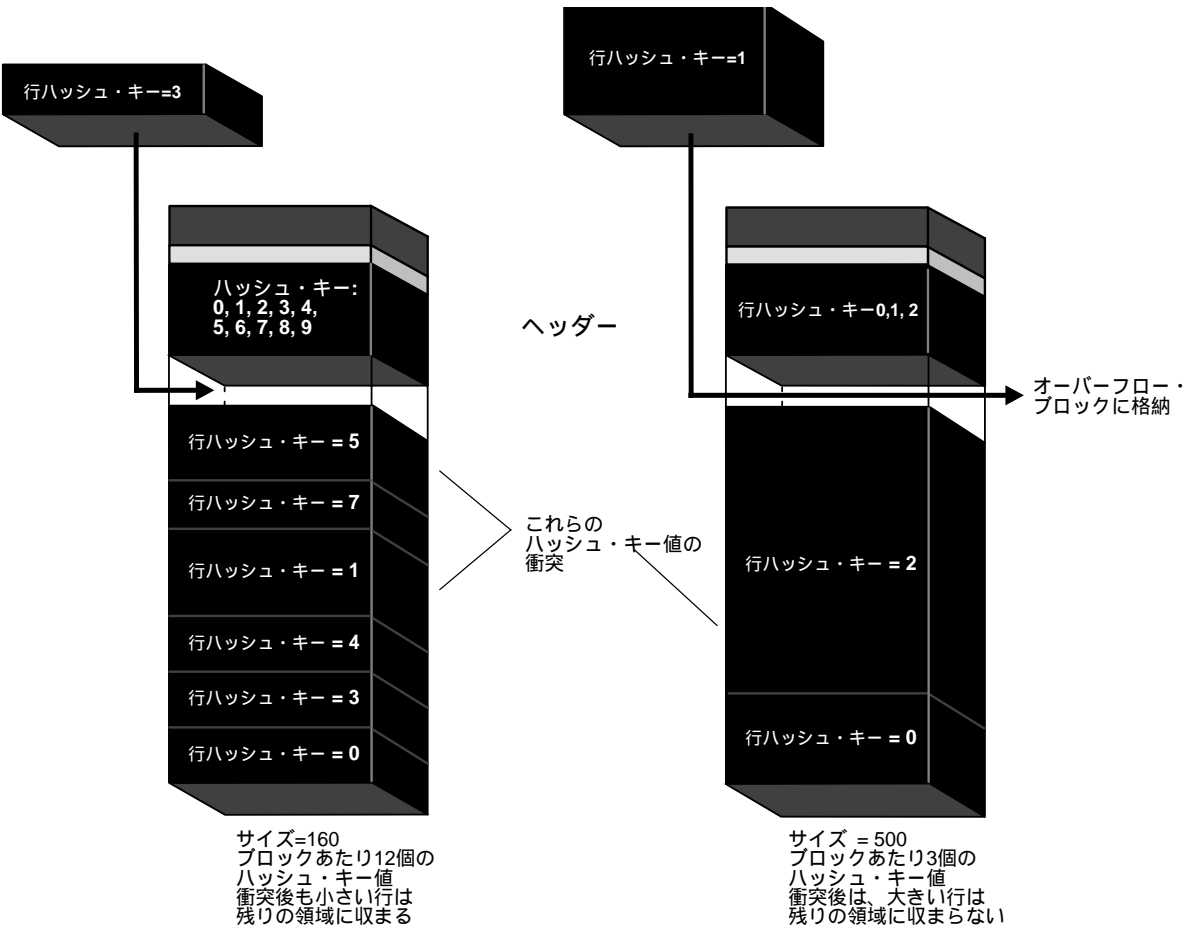
ハッシュ・クラスタに対する領域の割当て

他のタイプのセグメントと同じように、ハッシュ・クラスタの作成時のエクステントの割当ては、STORAGE 句の INITIAL、NEXT および MINEXTENTS パラメータによって制御されます。ただし、ハッシュ・クラスタでは、「ハッシュ表」と呼ばれる領域の最初の部分が作成時に割り当てられ、クラスタのすべてのハッシュ・キーをマップできるようになります（領域全体のサイズは $SIZE * HASHKEYS$ ）。そのため、ハッシュ・クラスタに対する領域の初期割当ては、SIZE と HASHKEYS の値にも依存しています。「 $SIZE * HASHKEYS$ 」と、STORAGE 句（INITIAL、NEXT など）によって指定された値のうち、どちらか大きい方が使用されます。

その後にハッシュ・クラスタに割り当てられる領域は、満杯になったデータ・ブロックからオーバーフローした行を保持するために使用されます。たとえば、特定のハッシュ・キーに対する最初のデータ・ブロックが満杯になっているとします。そのようなクラスタ化表にユーザーが行を挿入すると、行のクラスタ・キーは満杯のデータ・ブロックに格納されているハッシュ値にハッシュされます。したがって、そのハッシュ・キーに対して割り当てられた「ルート・ブロック」（元のブロック）に、行は挿入されません。そのかわり、ハッシュ・キーのルート・ブロックに連鎖されたオーバーフロー・ブロックに行が挿入されます。

衝突が頻繁に発生すると、ハッシュ・クラスタ内のオーバーフロー・ブロックが増加する（したがってデータ検索のパフォーマンスが低下する）場合もあり、増加しない場合もあります。衝突が発生し、ハッシュ・キーに対して割り当てられた最初のブロック内に空き領域がない場合は、新しい行を保持するためにオーバーフロー・ブロックを割り当てる必要があります。[図 10-11](#) に示されているように、このような可能性は、ハッシュ・クラスタの作成時に指定される、各ハッシュ・キー値とこれに対応するデータの平均サイズに大きく依存します。

図 10-11 ハッシュ・クラスタ内の衝突とオーバーフロー・ブロック



平均サイズが小さく、各行が一意的ハッシュ・キー値を持っている場合、データ・ブロックごとに多くのハッシュ・キー値を割り当てることができます。この場合、衝突している小さな行は、ハッシュ・キーに対するルート・ブロックの領域に収まる可能性が高くなります。ただし、ハッシュ・キー値の平均のサイズが大きいか、各ハッシュ・キー値が複数の行に対応する場合、データ・ブロックあたりで割当て可能なハッシュ・キー値はごくわずかです。この場合、大きな行は、ハッシュ・キー値に対して割り当てられたルート・ブロックに収まらず、オーバーフロー・ブロックが割り当てられる可能性があります。

単一表ハッシュ・クラスタ

単一表ハッシュ・クラスタにより、表の行への高速アクセスを提供できます。通常のハッシュ・クラスタでは、実際には一致するキーを持つ行が1つしかなくても、ブロック内で特定の表の行をすべて走査します。ただし、単一表ハッシュ・クラスタの場合は、ハッシュ・キーとデータ行に1対1のマッピングがあると、Oracle はデータ・ブロック内のすべての行を走査しなくても、1行を検出できます。

Oracle では、単一表ハッシュ・クラスタの作成時に、各ハッシュ・キー値の領域が事前に割り当てられます。ハッシュ値（基礎となるクラスタ・キー値ではなく）あたり複数の行は指定できず、ブロック内の行連鎖は許されません。さもないと、Oracle はクラスタ・キーと一致する行を判別するために、そのブロック内のすべての行を走査します。

追加情報： CREATE CLUSTER コマンドの SINGLE TABLE HASHKEYS オプションの詳細は、『Oracle8i SQL リファレンス』を参照してください。

パーティション表とパーティション索引

Like to a double cherry, seeming parted,

But yet an union in partition;

Two lovely berries molded on one stem.

Wm.Shakespeare: *A Midsummer-Night's Dream*

この章では、パーティション表とパーティション索引、およびパーティション化の管理における考慮事項について説明します。この章の内容は、次のとおりです。

- [パーティション化の基礎知識](#)
- [パーティション化の基本的なモデル](#)
- [表および索引をパーティション化するときのルール](#)
- [DML パーティション・ロックとサブパーティション・ロック](#)
- [メンテナンス操作](#)
- [索引の管理](#)
- [パーティション表およびパーティション索引についての権限](#)
- [パーティション表およびパーティション索引についての監査](#)
- [拡張パーティション表名と拡張サブパーティション表名](#)

注意： この章で説明している機能は、Partitioning Option 付きの Oracle8i Enterprise Edition を購入した場合にのみ使用できます。

パーティション化の基礎知識

この項では、Oracle データベースで大規模な表や索引を管理するときにパーティション化がどのように役立つかについて説明します。この項の内容は、次のとおりです。

- [パーティション化とは何か](#)
- [パーティション化の利点](#)
- [パーティション・ビューを使用した手動のパーティション化](#)

注意： Oracle がサポートしているのは、表、表の索引、マテリアライズド・ビューおよびその索引のパーティション化だけです。クラスタ化表やその索引のパーティション化はサポートしていません。

追加情報： マテリアライズド・ビューのパーティション化の詳細は、『Oracle8i チューニング』を参照してください。

パーティション化とは何か

「パーティション化」とは、大規模な表や索引を「パーティション」という、より小さくて管理しやすい部分に分割することで、この種の表や索引をサポートするときの主な問題に対処します。パーティションを定義すると、SQL 文は、表または索引全体ではなくパーティションをアクセスしたり操作できるようになります。パーティションは、一般に大量の履歴データを格納および分析するデータ・ウェアハウス・アプリケーションに特に有用です。

パーティション化の方法

パーティション化には、主として「レンジ・パーティション化」および「ハッシュ・パーティション化」という 2 つの方法があります。レンジ・パーティション化では、表や索引のデータが値の範囲に従ってパーティション化され、ハッシュ・パーティション化ではデータがハッシュ関数に従ってパーティション化されます。その他に「コンポジット・パーティション化」という方法があり、この方法ではデータは範囲別にパーティション化され、さらにハッシュ関数を使用して「サブパーティション」に分割されます。これらのパーティション化方法の詳細は、11-13 ページの「[パーティション化の基本的なモデル](#)」を参照してください。

論理属性と物理属性 表や索引のすべてのパーティションは、物理属性は異なっていることがありますが、論理属性はすべて同じです。たとえば、表のすべてのパーティションが同じ列および制約定義を共有し、索引のすべてのパーティションが同じ索引列を共有していても、記憶域仕様や、PCTFREE、PCTUSED、INTRANS および MAXTRANS などの他の物理属性は、同じ表や索引のパーティションごとに異なる場合があります。

パーティションと同様に、表や索引のすべてのサブパーティションも、同じ論理属性を持ちます。ただし、パーティションと違って、単一パーティションの各サブパーティションが異なる物理属性を持つことはできません。

パーティションとサブパーティションの格納 レンジ・パーティション化またはハッシュ・パーティション化された表や索引の各パーティション、およびコンポジット・パーティション化された表や索引の各サブパーティションは、別々のセグメントに格納されます。コンポジット・パーティション化された表や索引のパーティションは単なる論理構造であり、データはサブパーティションのセグメントに格納されるため、別のセグメントを占有しません。

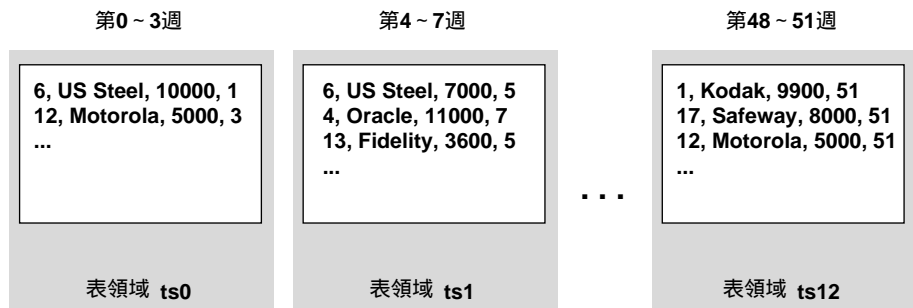
必要であれば、各パーティション（またはコンポジット・パーティション化された表や索引のサブパーティション）を、別々の表領域に格納できます。この方法には、次のような利点があります。

- データの破損による影響を抑えることができる。
- 各パーティション（またはサブパーティション）を個別にバックアップを作成したり回復できる。
- I/O の負荷平衡化のために、パーティション（またはサブパーティション）をディスク・ドライブにマップできる。

パーティション表の例

図 11-1 では、SALES 表の履歴データを、週の番号に基づいてそれぞれ 4 週間分を含む 13 個のパーティションに分けています。

図 11-1 週別にパーティション化された SALES 表



次の SQL 文では、図 11-1 に示すレンジ・パーティション表が作成されます。

```
CREATE TABLE sales ( acct_no          NUMBER(5),
                      acct_name        CHAR(30),
                      amount_of_sale   NUMBER(6),
                      week_no          INTEGER )
PARTITION BY RANGE ( week_no ) ...
(PARTITION sales1 VALUES LESS THAN ( 4 ) TABLESPACE ts0,
 PARTITION sales2 VALUES LESS THAN ( 8 ) TABLESPACE ts1,
 ...
 PARTITION sales13 VALUES LESS THAN ( 52 ) TABLESPACE ts12 );
```

追加情報： パーティション表の詳細例は、『Oracle8i 管理者ガイド』を参照してください。

パーティション・プルニング

Oracle Server には、パーティションとサブパーティションを明示的に認識するためのインテリジェント機能が組み込まれています。この知識は、アクセスを必要とするパーティションまたはサブパーティションをマークし、不要なパーティションやサブパーティションがこれらの SQL 文によるアクセスから「排除（プルニング）」されるようにするために、SQL 文の最適化に使用されます。

SQL 文ごとに、指定された選択基準に応じて不要なパーティションやサブパーティションが排除されます。たとえば、売上データ Q1（第 1 四半期）のみに関する問合せであれば、残りの 3 つに関するデータを取り出す必要はありません。このようなインテリジェント・プルニング機能によって、データの量を大幅に低減できるので、問合せのパフォーマンスが実質的に向上します。

オブティマイザは、アクセスされるパーティションまたはサブパーティションのすべての行がプルニングによって使用される選択基準を満たすかどうかを判断する場合に、パフォーマンスを向上させるために、評価のときにそれらの基準を述語リスト（WHERE 句）から削除します。ただし、SQL 文によってパーティション化される列に TO_DATE 以外の関数が適用されると、オブティマイザはパーティションをプルニングすることができません。（同様に、ファンクションベース索引でない限り、SQL 文によって索引付きの列に関数が適用されると、オブティマイザは索引を使用できません）。11-21 ページの「[DATE データ型](#)」を参照してください。

基礎となる表のパーティションを排除できない場合にも、索引と表が別々の列にパーティション化されていれば、パーティションのプルニングによって索引パーティションを排除できます。大規模な表に対する操作のパフォーマンスは、SQL 文がアクセスまたは変更する必要のあるデータ量を削減するパーティション索引を作成することによって改善できます。

不要なパーティションやサブパーティションを SQL 文からプルニングする機能によって、パーティション・レベルまたはサブパーティション・レベルのロード、除去、バックアップ、復元、再編成および索引作成など、さまざまな操作のパフォーマンスと可用性が向上します。

パーティション・ワイズ結合

パーティション表の最適化におけるもう 1 つの分野が、「パーティション・ワイズ結合」です。これは大規模な結合操作であり、順次、またはパラレルに実行される小さい結合に分かれています。

パーティション・ワイズ結合を使用するには、両方の表を同一レベルでパーティション化しておく必要があります（11-23 ページの「[同一レベル・パーティション化](#)」を参照）。

パーティション・ワイズ結合が使用されるのは、パフォーマンスが向上するとオプティマイザによって判断された場合です。場合によっては、オプティマイザがブルニングとパーティション・ワイズ結合を併用することもあります。

追加情報： パーティション・ワイズ結合の詳細は、『Oracle8i チューニング』を参照してください。

パーティション化の利点

この項では、パーティションを使用すると利点があるデータベースのクラスを明らかにし、それに伴う問題点を説明します。

- [大規模データベース（VLDB）](#)
- [定期メンテナンス操作による休止時間の短縮](#)
- [データ障害による休止時間の短縮](#)
- [DSS のパフォーマンス](#)
- [I/O のパフォーマンス](#)
- [ディスクのストライプ化：パフォーマンスと可用性](#)
- [パーティションの透過性](#)

大規模データベース（VLDB）

「大規模データベース（VLDB）」には、数百 GB から数 TB のデータが含まれています。パーティション化は、構造化されていないデータではなく、大部分が構造化されたデータが入っている VLDB をサポートできます。VLDB のサイズが大きいのは、一般に、非常に多くのデータ・オブジェクトがあるためではなく、数個の非常に大きなデータ・オブジェクト（表や索引）があるためです。

VLDB は、次の 2 つのカテゴリに大別できます。

- 「オンライン・トランザクション処理（OLTP）」データベースは、非常に多くの同時実行トランザクション向けに設計されている。各トランザクションは、少量のデータを処理する比較的簡単な操作です。
- 「意思決定支援システム（DSS）」は、大量のデータをアクセスして処理する必要のある、非常に複雑な問合せ向けに設計されている。

VLDB は、ほとんどの作業負荷が OLTP であれば、OLTP データベースであるといえます。同様に、ほとんどの作業負荷が DSS 問合せであれば、VLDB は DSS データベースであるといえます。

パーティション化は、OLTP VLDB と DSS VLDB をどちらも効果的にサポートします。

履歴データベース 履歴データベースは、典型的な種類の DSS VLDB です。この種のデータベースには、履歴データの表およびエンタープライズ・データの表という 2 種類の表が含まれます。

- 「履歴」表には、最近の特定の期間（過去 24 か月間など）における会社全体の業務取引が記録される。履歴表には、次の 2 種類があります。
 - 「実」表には、ベースライン情報（売上、伝票および注文など）が含まれる。
 - 「ロールアップ」表には、GROUP BY、AVERAGE および COUNT などの操作を使用してベース情報から導出したサマリー情報が含まれる。

履歴データの表に反映される期間は「ローリング・ウィンドウ」のように扱われます。データベース管理者（DBA）は、最も古いトランザクションを構成している一連の行を定期的に削除し、新しいトランザクションを構成する一連の行のために領域を割り当てます。たとえば、1997 年 4 月 30 日の業務が終了した時点で、DBA は 1995 年 5 月の業務を記録した行（およびサポートしている索引項目すべて）を削除し、1997 年 5 月の業務を記録するための領域を割り当てます。

履歴 VLDB のデータの大多数は、非常に大規模ないくつかの履歴データ表に格納されます。この表は、サイズが大きくなり、古いデータを削除して新しいデータを入力する操作を円滑に行う必要があるため、特に注意を要する問題が生じます。

- 「エンタープライズ」表には、会社全体のビジネス・エンティティ（部門、場所および製品など）が記録される。この情報は、時間の経過とともにゆっくりと変更が加えられます。定期的に修正されることはありません。エンタープライズ・データの表が大きい場合でも、履歴データの表と全社データの表を結合する操作を含む、長時間実行される多くの DSS 問合せのパフォーマンスに影響します。

パーティション化は、履歴データを時間に基づくパーティションに分割することにより、それらのパーティションを別々に管理したり自由に追加および削除して、大規模な履歴データ表やその索引をサポートするときの問題点に対処します。

ミッション・クリティカルなデータベース ミッション・クリティカルな OLTP データベースには、非常に大きいものでもなくとも、可用性とパフォーマンスについて特別の問題があります。たとえば、きわめて短時間（通常は 1 時間以内）で、10GB の表について予定のメンテナンス操作や回復操作を実行する必要があります。さらに、表や索引を複数のドライブに分散させている場合も、DBA はデータの配置をある程度制御できる必要があります。

メンテナンス操作の時間枠や回復時間を短縮し、障害による影響を軽減させる目的でクリティカルな表と索引がパーティションに分けられている場合、パーティション化により、ミッション・クリティカルなデータベースの可用性を高めることができます。パーティションごとにパフォーマンス・パラメータを制御することにより、クリティカルな表や索引へのアクセスのパフォーマンスも改善できます。

定期メンテナンス操作による休止時間の短縮

パーティションにより、データのロード、索引作成、データ・パージなどのデータ管理操作を表全体に対してではなくパーティション・レベルで実行できるので、これらの操作にかかる時間を大幅に軽減することができます。

パーティション化により、定期的なメンテナンス操作による休止時間の影響を大幅に軽減できます。

- 表または索引全体ではなく、個々のパーティションごとに「パーティションのメンテナンス操作」を実行する。
- 別々のパーティションに対してメンテナンス操作を同時に実行できるように、「パーティションを独立」させる。

注意： コンポジット・パーティション表および索引には、パーティションのメンテナンス操作だけでなく「サブパーティションのメンテナンス操作」、パーティションの独立だけでなく「サブパーティションの独立」があります。後述で使用する「パーティション」という一般用語は、パーティションとサブパーティションの両方を指します。

パーティションのメンテナンス操作「パーティションのメンテナンス操作」は、表全体または索引全体のメンテナンス操作よりも高速です。次の比率でスピードアップを実現できます。

$$(\text{表全体または索引全体のレコード数}) / (\text{パーティション内のレコード数})$$

これは、パーティションをまたがって格納される構成体（グローバル索引や参照整合性制約）がない場合の数字です。

休止時間をさらに短縮するために、パーティションのメンテナンス操作では、PARALLEL、NOLOGGING および DIRECT（または APPEND）オプションなど、表レベルや索引レベルのメンテナンス操作に使用可能なパフォーマンス機能を活用できます。

パーティションの独立 パーティションのメンテナンス操作のために「パーティションを独立」させると、同じ表または索引の別々のパーティションに対してメンテナンス操作を同時に実行したり、メンテナンス操作の影響を受けないパーティションに対して SELECT 操作や DML 操作を同時に実行できます。

たとえば、パーティション PA と PB に同時にダイレクト・パス・ロードを実行し、アプリケーションではその他のパーティションに対して標準の SQL SELECT 操作や DML 操作を実行することが可能です。

データの移動を含む操作では、パーティションの独立が特に重要です。そのような操作には非常に時間がかかります（数分、数時間または数日）。パーティション化すると、パーティションをまたいで格納される構成体（グローバル索引や参照整合性制約）がない場合に、データ移動を含む操作中に他のパーティションが使用できなくなる時間枠を数秒に短縮できます。

データの移動を伴わない操作であれば短時間で完了するため、パーティションを独立させる必要はありません。

データ障害による休止時間の短縮

メンテナンス操作のなかには、予定外の出来事によるものがあります。たとえば、ハードウェア障害またはソフトウェア障害によってデータが消失または破損し、回復操作が必要になることがあります。データベース、表領域またはデータ・ファイルに対して RECOVER コマンドを実行することにより、ハードウェア障害や多くのシステム・ソフトウェア障害から回復できます。回復中の表領域またはデータ・ファイルにレコードが含まれている表や索引は、回復中には使用できません。ミッション・クリティカルな OLTP データベースの場合には、可用性を高めることが特に重要です。

パーティションは互いに独立しているので、ある部分（またはある部分のサブセット）が使用できなくても、それによってデータの他の部分が使用できなくなることはありません。

パーティションを別々の表領域に格納することには、次のような利点があります。

- 回復の単位（表領域）が小さくなるため、RECOVER コマンドの実行による休止時間が短くなる。
- 回復の単位が小さくなるため、オフライン表領域（遅延ロールバック・セグメント）の回復に必要なディスク・リソースが少なくなる。
- 回復する表領域に格納されているパーティションのみをオフラインにすればよいため、使用できないデータの量が少なくなる。ユーザー・アプリケーションやメンテナンス操作は、他のパーティションにはアクセスできます。これは、パーティションの独立が役立つもう 1 つの例です。

DSS のパフォーマンス

非常に大きな表に対して DSS 問合せを実行すると、特別なパフォーマンス上の問題が生じます。表の走査が必要な非定型の問合せは、表内の各行を検査しなければならないため、時間がかかります。関係のない行のサブセットを識別してスキップする手段はありません。履歴データの表の場合は、多くの問合せが、最近生成された行に集中的にアクセスするため、この問題は特に重要です。

パーティションを作成すると、DSS のパフォーマンス上の問題を解決する助けになります。1 つのパーティション（または特定の範囲のパーティション）に対応する行のみを必要とする非定型問合せは、表の走査ではなくパーティションの走査を使用して実行できます。

たとえば、1997 年 10 月に生成されたデータを要求する問合せでは、何年にもわたるアクティビティによって生成された行ではなく、1997 年 10 月分のパーティションに格納された行のみを走査できます。これにより応答時間が速くなり、ソートが必要な問合せのための一時ディスク領域を実質的に減らすこともできます。

I/O のパフォーマンス

パーティション化により、データを複数の物理デバイス間にどのように分散させるかを制御できます。I/O 使用率のバランスをとるため、表や索引の各パーティションを格納する場所を指定できます。

このレベルの位置制御があれば、ディスクの競合を減らし、より速いデバイスを使用することにより、高速な応答時間が求められるアプリケーションで特殊な必要を満たすことができます。一方、アクセス頻度の低いデータ（古い履歴データなど）は、遅いディスクに移したり、階層的な記憶領域をサポートするサブシステムに格納できます。

ディスクのストライプ化：パフォーマンスと可用性

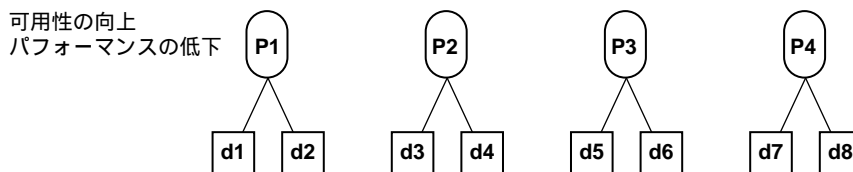
ディスクのストライプ化とパーティション化は、どちらもディスク・アームの競合を減らすことによりパフォーマンスを改善できるようにする手段です。どちらの手段を使用するか、または選択した手段をどの範囲に使用するかは、データベースを物理的に設計する段階で考慮する必要がある重要な問題です。これらの問題は、パフォーマンスだけではなく、可用性とパーティションの独立の観点からも考慮する必要があります。

図 11-2 は、パーティション化とストライプ化を結合した 2 つの極端な例を示しています。図 11-2 の (a) と (b) は、どちらも 4 つのパーティションを 8 台のディスクに分散させています。(a) は各パーティションを専用の 1 組のディスクにストライプ化しているのに対して (b) は各パーティションを 8 台すべてのディスクにストライプ化しています。

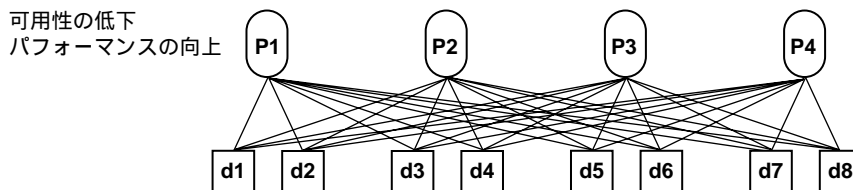
- (b) はパフォーマンス特性に優れているが、1 台でもディスクが障害が起こすと、すべてのパーティションが悪影響を受ける。
- (a) は、1 台のディスクが障害を起こしても 1 つのパーティションにしか影響がないため、可用性に優れている。

この 2 つの中間的な構成も可能であり、パーティションのサブセットをディスクのサブセットにストライプ化できます。

図 11-2 パーティションとディスクのストライプ化



(a) 各パーティションが少数のディスクのストライプに常駐



表と索引をどのようにパーティション化するか、および各パーティションを格納するディスクをどのようにストライプ化するかを決定するときには、パフォーマンスと可用性を比較検討する必要があります。

ミッション・クリティカルなデータベースの場合は、パーティションの独立と可用性を優先することをお勧めします。したがって、複数のディスクにストライプ化するパーティションは、それぞれ専用のディスク・ドライブ・セットにストライプ化してください。ディスク・ドライブのセットには十分な数のドライブを含めて、特定のパーティションへのアクセスに必要な I/O の並列性を達成します。

パーティションの透過性

大多数のアプリケーション・プログラムには「パーティションの透過性」が必要です。つまり、そのプログラムでアクセスするデータがパーティション化されているかどうか、どのようにパーティション化されているかを、プログラムからは意識しないで済むようにする必要があります。

しかし、一部のアプリケーション・プログラムでは、表全体ではなくて個々のパーティションにアクセスするよう明示的に要求することにより、パーティションを活用できます。たとえば、非常に大規模な表に対して実行する長いバッチ・ジョブを、個々のパーティションに対して夜間に行う一連の短いジョブに分けることができます。

パーティション・ビューを使用した手動のパーティション化

パーティション表を使用するかわりに、同一のテンプレートを使用して表をいくつか作成し、それらの表を結合（UNION）するビューを定義できます。この操作を「手動パーティション化」といい、このビューを「パーティション・ビュー」といいます。パーティション・ビューは、Oracle7、リリース 7.3 で使用できるパーティション化の唯一の形態です。Oracle8i の新しいアプリケーションに使用することはお薦めしません。

注意： Oracle8i は、Oracle7、リリース 7.3 との下位互換性のためにのみパーティション・ビューをサポートしています。Oracle リリース 9 から、パーティション・ビューはサポートされません。

Oracle7 データベースで作成されたパーティション・ビューは、ALTER TABLE コマンドの EXCHANGE PARTITION オプションを使用してパーティション表に変換できます。

追加情報： パーティション・ビューをパーティション表に変換する手順は、『Oracle8i 管理者ガイド』を参照してください。

パーティション・ビューの基本概念は、パーティション基準に WHERE 句または CHECK 制約を使用して大きな表を複数の物理的な表に分け、UNION ALL ビューを使用して小さな表を 1 つにまとめることです。その後、実表に対して同一のキー指定による「実索引」の集合を定義し、UNION ALL ビューを使用するときに索引の機能を提供します。（パーティション・ビューが正しく動作するためには、索引を作成する必要があります）。パーティション化されていない表に比べて、パーティション・ビューでは CPU オーバーヘッドが大幅に増大することはありません。キー範囲を使用してパーティション・ビューから選択を行う問合せがアクセスするのは、そのキー範囲内にある実表のみです。オプティマイザは、パーティション・ビューの実表に対して別々の実行計画を使用できます。（対照的に、オプティマイザはパーティション表内のすべてのパーティションに対して 1 つの実行計画を使用します。）

パーティション・ビューの短所

パーティション・ビューによる手動パーティション化には、パーティション表に比べて次のように多数の短所があります。

- 構成が複雑である。

データベース管理者は、パーティションに対応する実表と実索引を正しく定義し、これらの定義をメンテナンスする必要があります。パーティションをまたいでデータを移動する DDL 操作（split、move など）と等価の機能は、Export/Import ユーティリティまたは SQL スクリプトによってインプリメントする必要があります。

- パーティションの透過性が不足している。

一部の SQL 操作は、UNION ALL ビューではなく実表を使用して実行する必要があります。たとえば、INSERT では実表を参照するので、INSERT 文に指定する表名を取得するためにユーザー・コードが必要になります。

- パフォーマンスが低い。

UNION ALL ビューに対する SQL 操作の中には、オブティマイザが既存の実索引すべてを使用しないために、パフォーマンスが低いものがあります。

- メモリーの利用効率が悪い。

UNION ALL ビューに対して実行される SQL コンパイル済みの問合せは、そのビューをサポートするすべての表の記述情報を内部的にレプリケートします。

- DDL の制限。

グローバル索引と参照整合性制約は、UNION ALL ビューに対しては定義できません。

- ロードの制限。

UNION ALL ビューに対してダイレクト・ロードは実行できません。

パーティション・ビューのガイドライン

パーティション・ビューは、表 11-1 のガイドラインに従って作成し、メンテナンスしてください。

表 11-1 パーティション・ビューのガイドライン

パーティション・ビューを使用するには、PARTITION_VIEW_ENABLED パラメータを設定する必要があります。

DDL コマンドは、基礎となる表ごとに別々に発行する必要があります。たとえば、パーティション・ビューに索引を追加するには、基礎となるすべての表に索引を追加する必要があります。パーティション・ビューを分析するには、基礎となる表をすべて分析する必要があります。ただし、各パーティションの操作は並行して送ることができます。

管理操作は、パーティション・ビュー自体ではなく、その基礎となる表に対する操作として実行する必要があります。たとえば、分割操作は、1 つまたは 2 つの CREATE TABLE AS SELECT 操作（分割が「同所」分割の場合は 1 つ）と、パーティション・ビューのビュー・テキストの再定義からなっています。

基礎となる表の参照整合性制約は作成できるが、その制約をパーティション化に適用するには、主キーにパーティション列を含める必要がある。

同様に、基礎となる表の一意索引は作成できるが、その一意性をパーティション・ビューに適用するには、一意索引にパーティション列を含める必要がある。（一意索引は、1 つしか作成できません）

各プログラムに固有の索引があるため、スキップされないパーティションのすべての索引で索引参照を実行する必要がある。

パーティション・ビューを DML 文（UPDATE、INSERT または DELETE）のターゲットにすることはできない。

パーティション・ビューでは、連結されたパーティション化キーはサポートされない。

SQL*Loader では、パーティション・ビューはサポートされない。

パーティション化の基本的なモデル

この項では、次のパーティション化方法を含む基本的なパーティション化モデルについて説明します。

- レンジ・パーティション化
- ハッシュ・パーティション化
- コンポジット・パーティション化

また、次のトピックについても説明します。

- [パーティション名とサブパーティション名](#)
- [パーティション化およびサブパーティション化された列およびキー](#)
- [レンジ・パーティション化のパーティション・バウンド](#)
- [同一レベル・パーティション化](#)

CREATE TABLE または CREATE INDEX 文のオプションを指定して、表または索引をパーティション化できます。パーティション表またはパーティション索引を作成した後、そのパーティション化の属性を修正するには、ALTER TABLE または ALTER INDEX 文を使用します。

CREATE TABLE 文と CREATE INDEX 文のパーティション化の構文はよく似ています。CREATE TABLE 文では、次の事項を指定します。

1. 列や制約の定義などの、表の論理属性。
2. 表の物理属性。
 - 表が非パーティション表であれば、この属性はその表に対応付けられたセグメントの実際の物理属性である。
 - 表がパーティション表であれば、この表レベルの属性により、その表の個々のパーティションのデフォルトが指定される。
3. パーティション表の場合、「パーティションの指定」。次のものを指定できます。
 - 行をパーティションにマップするときに使用する表レベルのアルゴリズム
 - 表内の各パーティションにつき 1 つの、パーティション記述のリスト
 - パーティション記述のリスト（コンポジット・パーティション化の場合のみ）

各パーティション記述には、行をパーティションにマップするときのアルゴリズムについて補足的なパーティション・レベルの情報を定義する句が含まれます。この句では、そのパーティションのパーティション名と物理属性も指定できます。

各サブパーティションの記述（コンポジット・パーティション化の場合）では、サブパーティションの名前と表領域を指定できます。

データ型の制限 パーティション表には、LONG または LONG RAW データ型の列は格納できません。DATE データ型の列で表や索引をパーティション化する場合と、NLS 日付書式で世紀が指定されていない場合は、パーティション記述で TO_DATE 関数を使用して年を完全に指定しなければ、表や索引を作成できません。

例は、11-21 ページの「[DATE データ型](#)」を参照してください。

ビットマップの制限 パーティション表に対してビットマップ索引を作成できますが、ビットマップ索引はパーティション表に対してローカルでなければならないという制限があります。グローバル索引にすることはできません。（11-28 ページの「[索引のパーティション化](#)」を参照。）

コストベース最適化 コストベース最適化は、SQL 文がパーティション表またはパーティション索引にアクセスする場合に使用します。ルールベース最適化は、パーティションでは使用できません。パーティション表のすべてのパーティションに対して単一の実行計画を使用します。

DBMS_STATS パッケージまたは ANALYZE コマンドを使用すると、パーティションまたはサブパーティション別に統計を収集できます。特に、パーティション表内のデータの特性に大きな変化があったときは、統計情報の収集が重要になります。統計は、次のデータ・ディクショナリ・ビューで見ることができます。

ALL_TAB_PARTITIONS、DBA_TAB_PARTITIONS、USER_TAB_PARTITIONS

ALL_TAB_SUBPARTITIONS、DBA_TAB_SUBPARTITIONS、
USER_TAB_SUBPARTITIONS

ALL_IND_PARTITIONS、DBA_IND_PARTITIONS、USER_IND_PARTITIONS

ALL_IND_SUBPARTITIONS、DBA_IND_SUBPARTITIONS、
USER_IND_SUBPARTITIONS

ALL_PART_COL_STATISTICS、DBA_PART_COL_STATISTICS、
USER_PART_COL_STATISTICS

ALL_SUBPART_COL_STATISTICS、DBA_SUBPART_COL_STATISTICS、
USER_SUBPART_COL_STATISTICS

レンジ・パーティション化

「レンジ・パーティション化」は、列値の範囲に基づいて行をパーティションにマップします。レンジ・パーティション化は、表または索引についてパーティション化を指定することによって定義します。

```
PARTITION BY RANGE ( column_list )
```

さらに、個々のパーティションごとにパーティション化を指定します。

```
VALUES LESS THAN ( value_list )
```

各値の意味は次のとおりです。

- *column_list* は、列の順序付きリストであり、行または索引エントリがどのパーティションに属するかを指定する。
 - これらの列を「パーティション化列」という。
 - 特定の行のパーティション化列の値は、その行の「パーティション化キー」を構成する。
- *value_list* は、*column_list* に指定した列の値の順序付きリストである。

- *value_list* の各値は、リテラルか、定数を引数に持つ TO_DATE() または RPAD() 関数のいずれかでなければならない。(11-21 ページの「[DATE データ型](#)」を参照)
- 各パーティションのパーティション化を指定する *value_list* は、そのパーティションの上限 (その値は含まない) を定義する。この上限を「パーティション・バウンド」といいます。
- 各パーティションのパーティション・バウンドは、次のパーティションのパーティション・バウンドより小さくなければならない。

各パーティションでは、すべての行 (または索引エントリが指している行) の「パーティション・キー」は、そのパーティションの「パーティション・バウンド」未満になっています。表または索引の最初のパーティションでない限り、各パーティションのすべてのパーティション化キーも、直前のパーティションのパーティション・バウンド以上でなければなりません。パーティション化キーとパーティション・バウンドが比較される方法や、複数列からなるパーティション化キーが処理される方法の詳細は、11-20 ページの「[レンジ・パーティション化のパーティション・バウンド](#)」を参照してください。

たとえば、4 つのパーティション (各四半期の売上ごとに 1 つずつ) からなる次の表では、SALE_YEAR=1997、SALE_MONTH=7 および SALE_DAY=18 の行は、パーティション化キーが (1997, 7, 18) のため、第 3 パーティションになり、表領域 TSC に格納されます。SALE_YEAR=1997、SALE_MONTH=7 および SALE_DAY=1 の行は、パーティション化キーが (1997, 7, 1) のため、この行も第 3 パーティションになり表領域 TSC に格納されます。

```
CREATE TABLE sales
( invoice_no NUMBER,
  sale_year  INT NOT NULL,
  sale_month INT NOT NULL,
  sale_day   INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
( PARTITION sales_q1 VALUES LESS THAN (1997, 04, 01)
  TABLESPACE tsa,
  PARTITION sales_q2 VALUES LESS THAN (1997, 07, 01)
  TABLESPACE tsb,
  PARTITION sales_q3 VALUES LESS THAN (1997, 10, 01)
  TABLESPACE tsc,
  PARTITION sales_q4 VALUES LESS THAN (1998, 01, 01)
  TABLESPACE tsd );
```

ALTER TABLE MERGE PARTITIONS コマンドを使用すると、2 つの隣接するレンジ・パーティションの内容を、1 つのパーティションにマージできます。履歴データをより大きいパーティション内でオンライン状態に保つには、この操作が必要です。たとえば、日次パーティションがあり、最も古いパーティションが週次パーティションにロールアップされ、週次パーティションが月次パーティションにロールアップされるようにする必要があります。

ハッシュ・パーティション化

レンジ・パーティション化は、履歴データベースには適していますが、その他の用途には最善の選択肢とはいえない場合があります。パーティション化方法の1つである「ハッシュ・パーティション化」では、パーティション化列にハッシュ関数を使用してデータをパーティションにストライプ化します。ハッシュ・パーティション化では、パフォーマンス上の理由からレンジ・パーティション化に適さないデータを容易にパーティション化できます（パラレル DML、パーティション・プルニングおよびパーティション・ワイズ結合など）。

レンジ・パーティション化よりハッシュ・パーティション化の方が適しているのは、次のような場合です。

- 特定の範囲にマップされるデータ量が事前にわからない。
- レンジ・パーティションのサイズのばらつきがきわめて大きい。
- パーティション化キーによるパーティション・プルニングとパーティション・ワイズ結合が重要である（11-5 ページの「[パーティション・プルニング](#)」および「[パーティション・ワイズ結合](#)」を参照）。

最も均等なデータ分布を得るには、パーティション数を2の乗数（2、4、8など）にする必要があります。ハッシュ・パーティションは、名前を付けて特定の表領域に格納できます。ハッシュ・パーティションのローカル索引は、表データと同一レベルでパーティション化されます。ローカル索引パーティションの場合は、パーティション名と表領域を指定できません。

次の例は、ハッシュ・パーティションに名前を付け、特定の表領域に格納する表を作成しています。

```
CREATE TABLE product( ... )
  STORAGE (INITIAL 10M)
  PARTITION BY HASH(column_list)
  ( PARTITION p1 TABLESPACE h1,
    PARTITION p2 TABLESPACE h2 );
```

ハッシュ・パーティションには、パーティションの分割、削除およびマージの概念は当てはまりません。ただし、パーティション数は、ALTER TABLE を使用してハッシュ・パーティションを ADD または COALESCE すれば、増減させることができます。

コンポジット・パーティション化

「コンポジット・パーティション化」では、データはレンジ方式でパーティション化されてから、各パーティション内でハッシュ方式でサブパーティション化されます。このタイプのパーティション化では、パーティション・レベルでの履歴操作、並列性（パラレル DML）およびサブパーティション・レベルでのデータ配置がサポートされます。

コンポジット・パーティション化の特色は、次のとおりです。

- 管理が容易であるというレンジ・パーティション化の長所を持つ。

- データ配置と並列性というハッシュ・パーティション化の長所を持つ。
- サブパーティションに名前を付けて、特定の表領域に格納できる。
- コンポジット・パーティション化表の論理索引を作成できる。これは、デフォルトで表のサブパーティションと同じ表領域に格納されます。
- レンジ・パーティション化されたグローバルな索引を作成できる。
- 索引サブパーティションに名前を付けて、特定の表領域を指定できる。

コンポジット・パーティション化された表または索引のパーティションは、あくまでも論理構造であり、データはそのサブパーティションのセグメントに格納されます。

次の例では、コンポジット・パーティション化を使用する表を作成しています（NLS DATE 書式が DD-MON-YYYY の場合）。

```
CREATE TABLE orders(  
    ordid NUMBER,  
    orderdate DATE,  
    productid NUMBER,  
    quantity NUMBER)  
PARTITION BY RANGE(orderdate)  
SUBPARTITION BY HASH(productid) SUBPARTITIONS 8  
STORE IN(ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8)  
( PARTITION q1 VALUES LESS THAN('01-APR-1998'),  
  PARTITION q2 VALUES LESS THAN('01-JUL-1998'),  
  PARTITION q3 VALUES LESS THAN('01-OCT-1998'),  
  PARTITION q4 VALUES LESS THAN(MAXVALUE));
```

この例で、ORDERS 表は ORDERDATE キーに基づいて、年の四半期を表す 4 つの範囲にパーティション化されます。各レンジ・パーティションは、PRODUCTID キーに基づいて 8 個のサブパーティションにさらに分割され、サブパーティションは合計 32 個になります。各表領域には、各パーティションからのサブパーティションが 1 つ含まれます。

次の例では、コンポジット・パーティション化を使用する表が作成され、各サブパーティションに明示的に名前が付けられ、指定された表領域に格納されます。

```
CREATE TABLE orders( ... )  
PARTITION BY RANGE(orderdate)  
SUBPARTITION BY HASH(productid) SUBPARTITIONS 8  
STORE IN (ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8)  
( PARTITION q1 VALUES LESS THAN('01-APR-1998')  
  ( SUBPARTITION q1_h1 TABLESPACE ts1,  
    ...  
    SUBPARTITION q1_h7 TABLESPACE ts7,  
    SUBPARTITION q1_h8 TABLESPACE ts8)  
  PARTITION q2 VALUES LESS THAN('01-JUL-1998'), ... );
```

パーティション名とサブパーティション名

各パーティションまたはサブパーティションには名前があります。パーティション名は、スキーマ・オブジェクトとその部分を命名するときの通常のルールに従う必要があります。特に、次の点に注意してください。

- 表パーティションまたはサブパーティションの名前は、同じ親表に属するすべてのパーティションの中で一意にする。
- 索引パーティションまたはサブパーティションの名前は、同じ親索引に属するすべてのパーティションの中で一意にする。

コンポジット・パーティション化の場合、サブパーティションとパーティションの名前は同じ名前領域にあります。つまり、同じ親表または親索引に属するパーティションおよびサブパーティションには、同じ名前は使用できません。

パーティションまたはサブパーティションは改名できますが、パーティション名やサブパーティション名のシノニムを作成することはできません。

追加情報： スキーマ・オブジェクトの命名のルールの詳細は、『Oracle8i SQL リファレンス』を参照してください。

パーティションまたはサブパーティションの参照

パーティション名とサブパーティション名は、DDL 文や DML 文、および Import/Export や SQL*Loader などのユーティリティ文で参照できます。パーティション名の指定は、その親表または親索引の名前を指定するコンテキストでのみ可能であり、スキーマ名では修飾できません。（親表または親索引を修飾するためのスキーマ名は使用できます。）たとえば、次のとおりです。

```
ALTER TABLE admin.patient_visits DROP PARTITION pv_dec92;  
SELECT * FROM sales PARTITION (s_nov97) s WHERE s.amount_of_sale > 1000;
```

SQL 文でパーティションを参照する方法の詳細は、11-62 ページの「[拡張パーティション表名と拡張サブパーティション表名](#)」を参照してください。

パーティション化およびサブパーティション化された列およびキー

表または索引の「パーティション化列」（または「サブパーティション化列」）は、データのパーティション化（またはサブパーティション化）方法を決定する値を持つ列の順序付きリストです。このリストは 16 列以内であり、それには次のタイプの列を含めることはできません。

- LEVEL または ROWID 疑似列
- ROWID データ型の列
- ネストした表、VARRAY、オブジェクト型または REF 列
- LOB 列（BLOB、CLOB、NCLOB または BFILE データ型）

ある行の「パーティション化キー」は、パーティション化列の値を順番に並べたリストです。同様に、コンポジット・パーティション化では、ある行の「サブパーティション化キー」は、サブパーティション化列の値を順番に並べたリストです。Oracle では、各行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）にレンジまたはハッシュ方式が適用され、行が属するパーティション（またはサブパーティション）が判別されます。

レンジ・パーティション化のパーティション・バウンド

レンジ・パーティション化された表または索引の場合、各行のパーティション化キーが上限値および下限値と比較され、行が属するパーティションが判別されます（レンジ・パーティション化の概要は、11-15 ページの「[レンジ・パーティション化](#)」を参照してください）。

- レンジ・パーティション化された表と索引のすべてのパーティションには、VALUES LESS THAN 句によって指定された「上限値」（その値未満）がある。
- 最初のパーティションを除くすべてのパーティションには、1 つ前のパーティションの VALUES LESS THAN によって指定された「下限値」（その値以上）がある。

パーティション・バウンド全体で、表または索引内のパーティションの順序を定義します。「最初」のパーティションは、VALUES LESS THAN 句の値が最も低いパーティションであり、「最後」もしくは「最高」のパーティションは、VALUES LESS THAN 句の値が最も高いパーティションです。

パーティション化キーとパーティション・バウンドの比較

表に挿入する行のパーティション化キーが、表内で最上位のパーティションのパーティション・バウンド以上であれば、挿入は失敗します。

パーティション化キーとパーティション・バウンドの文字値の比較では、文字はそのバイナリ値に基づいて比較されます。ただし、1 文字が複数バイトからなる場合は、Oracle は文字ではなく各バイトのバイナリ値を比較します。この比較では、列のデータ型に対応する比較のルールも適用されます（たとえば、ANSI CHAR データ型の比較では、空白埋め比較が行われます）。NLS パラメータ、特に初期化パラメータ NLS_SORT および NLS_LANGUAGE と環境変数 NLS_LANG は、比較には影響を与えません。

パーティション化キーの比較の詳細は、11-22 ページの「[複数列からなるパーティション化キー](#)」を参照してください。

MAXVALUE

パーティション・バウンド *value_list* のどの値についても、キーワード MAXVALUE を指定できます。このキーワードは、仮想の「無限大」値を示しており、そのデータ型の他のどんな値（NULL 値を含む）よりも高い値としてソートされます。

たとえば、次のようなパーティション・バウンドを使用すると、STATE 列（CHAR(10) 列）に基づいて OFFICE 表を 3 つのパーティションに分けることができます。

- VALUES LESS THAN ('I'): A ~ H の文字で始まる州

- VALUES LESS THAN ('S'): I ~ R までの文字で始まる州。
- VALUES LESS THAN (MAXVALUE): S ~ Z までの文字で始まる州と、米国以外の特
殊コードで始まる地域。

NULL

パーティション・バウンド *value_list* に NULL は指定できません。パーティション・バウンド *value_list* の値として空の文字列を指定することもできません。データベース・サーバーでは空の文字列が NULL として処理されるからです。

行をパーティションに割り当てる目的で、Oracle は NULL 値を MAXVALUE 以外のすべての値より大きな値としてソートします。NULL は、MAXVALUE より小さい値としてソートされます。

したがって、NULL を受入れ可能な列に基づいて表をパーティション化する場合、その列に NULL 値が含まれるのであれば、その列の最も高いパーティションのパーティション・バウンドには MAXVALUE を指定してください。そうしないと、NULL 値が含まれる行が表の最も高いパーティションよりも上にマップされるため、挿入は失敗します。

DATE データ型

パーティション・キーに DATE データ型の列が含まれている場合に、NLS 日付書式で世紀が指定されていなければ、TO_DATE() 関数で年に 4 文字の書式マスクを使用してパーティション・バウンドを指定する必要があります。このように指定しなければ、表や索引を作成できません。

たとえば、DATE 列を使用して SALES 表を作成する場合を考えます。

```
CREATE TABLE sales
( invoice_no NUMBER,
  sale_date DATE NOT NULL )
PARTITION BY RANGE (sale_date)
( PARTITION sales_q1
  VALUES LESS THAN (TO_DATE('1997-04-01','YYYY-MM-DD'))
  TABLESPACE tsa,
  PARTITION sales_q2
  VALUES LESS THAN (TO_DATE('1997-07-01','YYYY-MM-DD'))
  TABLESPACE tsb,
  PARTITION sales_q3
  VALUES LESS THAN (TO_DATE('1997-10-01','YYYY-MM-DD'))
  TABLESPACE tsc,
  PARTITION sales_q4
  VALUES LESS THAN (TO_DATE('1998-01-01','YYYY-MM-DD'))
  TABLESPACE tsd );
```

データを問い合わせるか変更する場合は、日付情報の値をコンパイル時に判断できるように、WHERE 句に TO_DATE() 関数を使用することをお勧めします。ただし、次のように、別の書式を使用すると、オプティマイザは DATE 型のパーティション化列で選択条件を使用して、パーティションをプルニングすることができます。

```
SELECT * FROM sales
WHERE s_saledate BETWEEN TO_DATE('01-JUL-94', 'DD-MON-YY')
AND TO_DATE('01-OCT-94', 'DD-MON-YY');
```

```
SELECT * FROM sales
WHERE s_saledate BETWEEN '01-JUL-1994' AND '01-OCT-1994';
```

この場合、日付値は実行時にのみ完全なものになります。したがって、Oracle がアクセスしているパーティションは、通常、SQL 文の EXPLAIN PLAN コマンド出力の partition_start および partition_stop 列に表示されますが、この情報を見ることはできません。かわりに、両方の列にキーワード「KEY」が表示されます。

複数列からなるパーティション化キー

複数の列に基づいて表や索引を範囲別にパーティション化すると、各パーティション・バウンドとパーティション化キーは、値のリスト（ベクトル）になります。パーティション・バウンドとパーティション・キーの順序は、ANSI SQL2 ベクトル比較規則に従って設定されます。（また、複数列からなる索引キーの順序も、同じ方法で設定されます）

パーティション化キーをパーティション・バウンドと比較するには、それに対応する列の値を、等しくないペアが見つかるまで比較します。そのペアによって、どのベクトルの方が大きいかを判断します。残りの列の値は、この比較では無効です。

数学用語では、ベクトル $V1$ と $V2$ が同数の値を含んでいれば、 $Vx[i]$ は、 Vx にある i 番目の値を示します。さらに、 $V1[i]$ と $V2[i]$ のデータ型には互換性があるとします。このとき、比較の結果は次のようになります。

- $V1 = V2$ になるのは、すべての i について $V1[i] = V2[i]$ である場合。
- $V1 < V2$ になるのは、ある n について $V1[n] < V2[n]$ であり、 n より小さいすべての i について $V1[i] = V2[i]$ である場合。
- $V1 > V2$ になるのは、ある n について $V1[n] > V2[n]$ であり、 n より小さいすべての i について $V1[i] = V2[i]$ である場合。

たとえば、パーティション P のパーティション・バウンドが (7, 5, 10) であり、次に低いパーティションのパーティション・バウンドが (6, 7, 3) であれば、次のようになります。

- キー (6, 9, 11) はパーティション P に属する。その理由は、次のとおりです。
 - キー (6, x, x) は (7, x, x) より小さい。
 - キー (6, 9, x) は (6, 7, x) より大きい。

この比較では、2 列目に不一致が見つかったと 3 列目の値は考慮されないため、この例の (x, x, 11) と (x, x, 10) のように、キーの 3 列目の値がパーティション・バウンド内で対応する値より大きいことがあるので注意してください。

- キー (7, 3, 15) はパーティション P に属する。その理由は、次のとおりです。
 - キー (7, 3, x) は (7, 5, x) より小さい。
 - キー (7, x, x) は (6, x, x) より大きい。

このキーの 1 列目の値は、パーティション・バウンドの 1 列目の値と等しくなることがあるので注意してください。VALUES LESS THAN 句は、個々の列ではなく列の集合に対して適用されます。

- キー (6, 5, 0) と (7, 5, 11) は、P 以外のパーティションに属する。

パーティション・バウンド *value_list* の 1 つの要素に MAXVALUE がある場合は、それ以降の要素の値はすべて無関係になります。たとえば、パーティション・バウンド (10, MAXVALUE, 5) は、パーティション・バウンド (10, MAXVALUE, 6) またはパーティション・バウンド (10, MAXVALUE, MAXVALUE) と等価です。

複数列のパーティション化キーが役立つのは、表の主キーに複数の列が含まれているが、キーの中で最も重要な列に基づいて行を分配しても均等に分配できない場合です。たとえば、SUPPLIER_PARTS 表に仕入先とその提供部品に関する情報が含まれており、その表の主キーが (SUPPNUM, PARTNUM) の場合を考えます。一部の仕入先は数百～数千種類の部品を納入しているものの、その他の仕入先は数種類の特殊な部品しか納入していないため、SUPPNUM (仕入先番号) に基づいてパーティションを作成するだけでは不十分です。かわりに、表を (SUPPNUM, PARTNUM) に基づいてパーティション化します。

複数列のパーティション化キーは、日付を DATE 列ではなく、3 つの CHAR 列で表現している場合にも役立ちます。

パーティション・バウンドによって適用される暗黙的な制約

表内の最も高いパーティションとして MAXVALUE 以外のパーティション・バウンドを指定した場合、その表には暗黙の CHECK 制約が適用されます。この制約は、データ・ディクショナリ内には記録されません (しかし、パーティション・バウンドそのものは記録されます)。

同一レベル・パーティション化

2 つの表または索引は、次の場合に「同一レベル・パーティション化」されます。

- 両方のパーティション化方法 (レンジまたはハッシュ) パーティション化列、パーティション数およびパーティション・バウンド (レンジ・パーティション化の場合) が同一である。

- 一方または両方の表または索引がコンポジット・パーティション化されている場合、その表または索引が最低 1 つのパーティション化方法（レンジまたはハッシュ）で同一レベル・パーティション化されていれば、両方が同一レベル・パーティション化される。この場合、表または索引は「1 ディメンションで同一レベル・パーティション化」されます。

これらのスキーマ・オブジェクトの型は、同じでなくてもかまいません。たとえば、表と索引でも同一レベル・パーティション化を実施できます。

レンジ同一レベル・パーティション化

A と B がレンジ・パーティション化された表または索引で、A[i] が A の i 番目のパーティションで、B[i] が B の i 番目のパーティションである場合、次のすべての条件が満たされていれば、A と B には同一レベル・パーティション化が適用されています。

- パーティション数が同数 (N) である。
- パーティション化列が同数 (M) である。
- $1 \leq i \leq N$ のすべての i について、A[i] と B[i] には同じパーティション・バウンドが指定されている。

Apcol[i] が A の i 番目のパーティション化列、Bpcol[i] が B の i 番目のパーティション化列であれば、さらに次の条件も満たしている必要があります。

- $1 \leq i \leq M$ のすべての i について、Apcol[i] と Bpcol[i] のデータ型が、長さ、精度および位取りを含めて同じになっている。

A[i] と B[i] は、物理的な属性の点で異なっても構いません。特に、それらが同じ表領域内に存在する必要はありません。

同一レベル・パーティション化は、データベースの設計時に考慮することが重要です。

- パーティションのメンテナンス操作および表領域の回復操作の際に、休止時間と使用できないデータの量を減らす。たとえば、表とそのローカル索引が同一レベル・パーティション化されるため、パーティションを分割しても、その効果は 1 つの表パーティションとそれに対応する索引パーティションに限られます。表の索引がローカルでなければ、その表の 1 パーティションを分割すると、索引全体の再編成が必要になります。
- 同一レベル・パーティション化によって、パーティション・ワイズ結合が使用可能になる。
- データの関連サブセットにおける表領域の不完全回復（Point-in-Time 回復）を容易にする。たとえば、表とその主キー索引、または親表と子表に同一レベル・パーティション化を実施できます。これにより、対応するパーティションを特定の時点まで回復できます。

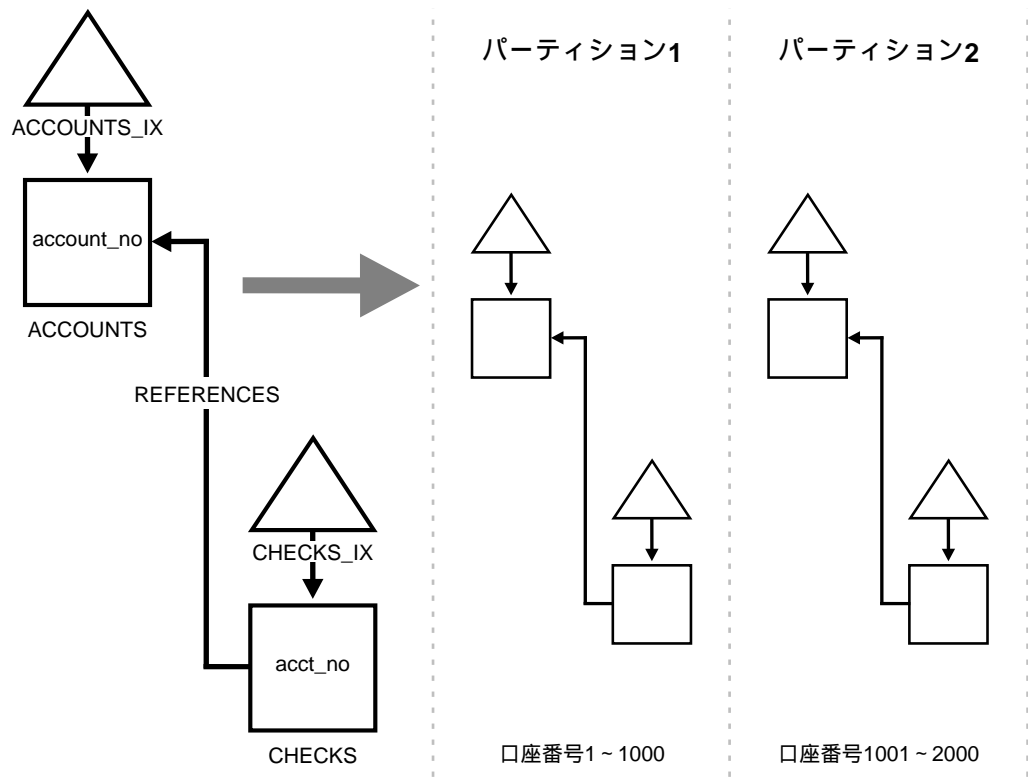
同一レベル・パーティション化の例

図 11-3 は、同一レベル・パーティション化を適用した 4 つの論理的に関連のあるスキーマ・オブジェクトを示しています。

- ACCOUNTS は、列 ACCOUNT_NO に基づいてレンジ・パーティション化された 2 つのパーティションを持つ表である。最初のパーティションはアカウント番号 1000 までを含み、2 番目のパーティションは 2000 までを含んでいます。
- ACCOUNTS_IX は、ACCOUNTS 表の列 ACCOUNT_NO に対する索引である。表と同様、索引も ACCOUNT_NO に基づいてレンジ・パーティション化されており、ACCOUNTS のパーティションと同じパーティション・バウンドを持つ 2 つのパーティションに分かれています。
- CHECKS は、列 ACCT_NO に基づいてレンジ・パーティション化された、2 つのパーティションがある表である。このパーティションには、ACCOUNTS 表のパーティションと同じパーティション・バウンドを指定します。ACCT_NO は、ACCOUNTS にある ACCOUNT_NO を参照する外部キーです。
- CHECKS_IX は、CHECKS の列 (ACCT_NO, CHECK_NO) に対する索引です。この索引には、ACCT_NO に基づいてレンジ・パーティション化を実施し、2 つのパーティションがあります。ACCOUNTS のパーティションと同じパーティション・バウンドを指定します。

これら 4 つのスキーマ・オブジェクト間の論理関係を、図 11-3 の左側に示します。物理的なパーティション化については、図の右側に示します。(三角形は索引を、長方形は表を表します。)

図 11-3 同一レベル・パーティション化された表と索引



表および索引をパーティション化するときのルール

この項では、パーティション表とパーティション索引を作成するときのルール、およびパーティションの物理的な属性を説明します。

表のパーティション化

表をパーティション化するときのルールは、次のように簡単なものです。

- 次のような場合に、表をパーティション化できる。
 - 表がクラスタの一部になっていない。

- LONG または LONG RAW データ型を含んでいない。
- パーティション索引および非パーティション索引を、パーティション表および非パーティション表と混在させることができる。
 - パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。
 - 非パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。（非パーティション表には、グローバル索引しか作成できません。11-32 ページの「[グローバル・パーティション索引](#)」を参照してください）。

表パーティションの物理属性

この項では、レンジ、ハッシュおよびコンポジット・パーティション化に関する表パーティションの物理属性について説明します。

レンジ・パーティション化とハッシュ・パーティション化 デフォルトの物理属性は、CREATE TABLE 文によってパーティション表を作成する時点で指定します。パーティション表そのものに対応するセグメントはないため、この属性はメンバー・パーティションの物理属性を導出するときに限り使用されます。デフォルトの物理属性は、ALTER TABLE MODIFY DEFAULT ATTRIBUTES を使用して修正できます。

ハッシュ・パーティション化の場合、すべてのパーティションは同じ物理特性を持つため、パーティションに指定できる物理属性はその表領域だけです。

CREATE TABLE または ALTER TABLE ADD PARTITION によって作成される表パーティションの物理属性は、次のようにして決まります。

- パーティション属性の値を指定しなければ、それに対応する実表に（明示的に、またはデフォルトで）指定されている物理属性の値が使用される。

ハッシュ・パーティション化の場合は、ALTER TABLE MOVE PARTITION を使用して、パーティションを異なる表領域に移動できます。レンジ・パーティション化の場合は、この文でパーティションを指定するか、その物理属性を変更できます。その後の属性は、次のようにして決まります。

- 新しい値を指定しなければ、この文を発行する前に存在していた値が使用される。

レンジ・パーティション化の場合、ALTER TABLE SPLIT PARTITION によって作成される表パーティションの物理属性は、次のようにして決まります。

- 新しい値を指定しなければ、分割するパーティションの物理属性値が使用される。（これは、グローバル索引の分割にも当てはまります。欠落している属性は、分割する索引パーティションから継承されます）。

表のすべてのパーティションの物理属性は、ALTER TABLE を使用して変更できます。たとえば、ALTER TABLE *tablename* NOLOGGING は、*tablename* のすべてのパーティションのロギング・モードを NOLOGGING に変更します。

LOB データ型を含む表パーティションの物理属性に関する追加情報は、11-39 ページの「[LOB データ・パーティションの表領域および記憶域属性](#)」を参照してください。

コンポジット・パーティション化 コンポジット・パーティション化の場合、パーティションでサブパーティションのデフォルト物理属性が指定されます。サブパーティションは、明示的に指定できる物理属性は表領域だけであるという点で、ハッシュ・パーティションに似ています。

デフォルトの物理属性は、CREATE TABLE 文によってコンポジット・パーティション表を作成する時点で最初に指定します。パーティションまたは表そのものに対応するセグメントはないため、この属性はメンバー・サブパーティションの物理属性を導出するときに限り使用されます。このデフォルト属性は、後で ALTER TABLE MODIFY DEFAULT ATTRIBUTES または ALTER TABLE MODIFY DEFAULT ATTRIBUTES FOR PARTITION を使用して変更できます。

CREATE TABLE または ALTER TABLE ADD PARTITION によって作成されるサブパーティションの物理属性は、次のようにして決まります。

- サブパーティションの表領域を明示的に指定しなければ、それに対応するパーティションに（明示的に、またはデフォルトで）指定された表領域が使用される。
- パーティションの物理属性値を指定しなければ、それに対応する実表に（明示的に、またはデフォルトで）指定した属性が使用される。

ALTER TABLE MOVE SUBPARTITION を使用すると、サブパーティションを別の表領域に移動できますが、サブパーティションの他の物理属性は変更されません。ALTER TABLE MODIFY PARTITION では、パーティション自体のデフォルトの物理属性と、その既存のすべてのサブパーティションの物理属性が変更されます。ALTER TABLE MODIFY PARTITION の FOR PARTITION 句を使用すると、既存のサブパーティションの属性変更を回避できます。表レベルで変更された属性は、表、パーティションおよびサブパーティションという 3 レベルすべてのデフォルトに影響します。

LOB データ型を含む表サブパーティションの物理属性に関する追加情報は、11-39 ページの「[LOB データ・パーティションの表領域および記憶域属性](#)」を参照してください。

索引のパーティション化

索引をパーティション化するときのルールは、表をパーティション化するときのルールと似ています。

- 次の条件を満たしていれば、索引をパーティション化できる。
 - クラスタ索引ではない。
 - クラスタ化表に対して定義された索引ではない。
 - パーティション表のビットマップ索引はローカル索引でなければならない。
- パーティション索引および非パーティション索引を、パーティション表および非パーティション表と混在させることができる。

- パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。
- 非パーティション表には、パーティション B* ツリー索引または非パーティション B* ツリー索引（あるいはその両方）を含めることができる。
- 非パーティション表のビットマップ索引は、パーティション化できない。

ただし、パーティション索引はパーティション表よりも複雑で、パーティション索引にはローカル同一キー索引、ローカル非同一次元キー索引、グローバル同一キー索引およびグローバル非同一次元キー索引の 4 種類があります。これらの種類については、この後説明します。Oracle は、この 4 種類のうち 3 種類をサポートしています（グローバル非同一次元索引は、実際のアプリケーションでは役に立ちません）。

ローカル・パーティション索引

「ローカル索引」では、特定の索引パーティションにあるすべてのキーが、基礎を形成する 1 つの表パーティションに格納された行のみを参照します。ローカル索引は、LOCAL 属性を指定することにより作成されます。

Oracle は、ローカル索引とその基礎になる表が同一レベルでパーティション化されるように、ローカル索引を組み立てます。Oracle は、基礎になる表と同じ列に基づいて索引をパーティション化し、基礎になる表と同数のパーティションまたはサブパーティションを作成し、対応するパーティションと同じパーティション・バウンドを指定します。

また、基礎となる表のパーティションが追加、削除、マージまたは分割されるか、ハッシュ・パーティションまたはサブパーティションが追加されたり結合されると、索引のパーティション化が自動的にメンテナンスされます。これによって、索引と表は同一レベルでパーティション化された状態を保ちます。

パーティション列が索引列のサブセットを形成している場合は、一意のローカル索引を作成できます。この制限により、同じ索引キーを持つ行は常に同じパーティションにマップされるため、一意性違反を検出できることが保証されます。

ローカル索引の長所は、次のとおりです。

- 基礎となる表パーティションのメンテナンス操作（SPLIT PARTITION、またはハッシュ・パーティションの場合は ADD PARTITION を除く）が実行される場合は、索引パーティションを 1 つ再構築するだけです。
 - パーティション表の索引がすべてローカルであれば、パーティションのメンテナンス操作の所要時間は、パーティション・サイズに比例する。
 - ローカル索引はパーティションの独立をサポートしている。
 - ローカル索引は、履歴データの表における古いデータのロールアウトと、新しいデータのロールインをスムーズにサポートする。
- Oracle は、ローカル索引と基礎になる表が同一レベルでパーティション化されているという状況を活用して、より効率的な問合せアクセス計画を生成できる。

- ローカル索引により、表領域の不完全回復の作業を簡略化できる。表のパーティションまたはサブパーティションを特定の時点まで回復するには、対応する索引エントリも同じ時点まで回復する必要があります。そのための唯一の方法は、ローカル索引を使用することです。これにより、対応する表と索引パーティションまたはサブパーティションを同時に回復できます。
- パーティション内並列性（つまり、各パーティションに対する複数の処理）を DBMS_PCLXUTIL パッケージの BUILD_PART_INDEX プロシージャと併用すると、パーティション表のローカル索引を作成または再構築できます。

追加情報： DBMS_PCLXUTIL パッケージの説明は、『Oracle8i パッケージ・プロシージャ・リファレンス』を参照してください。

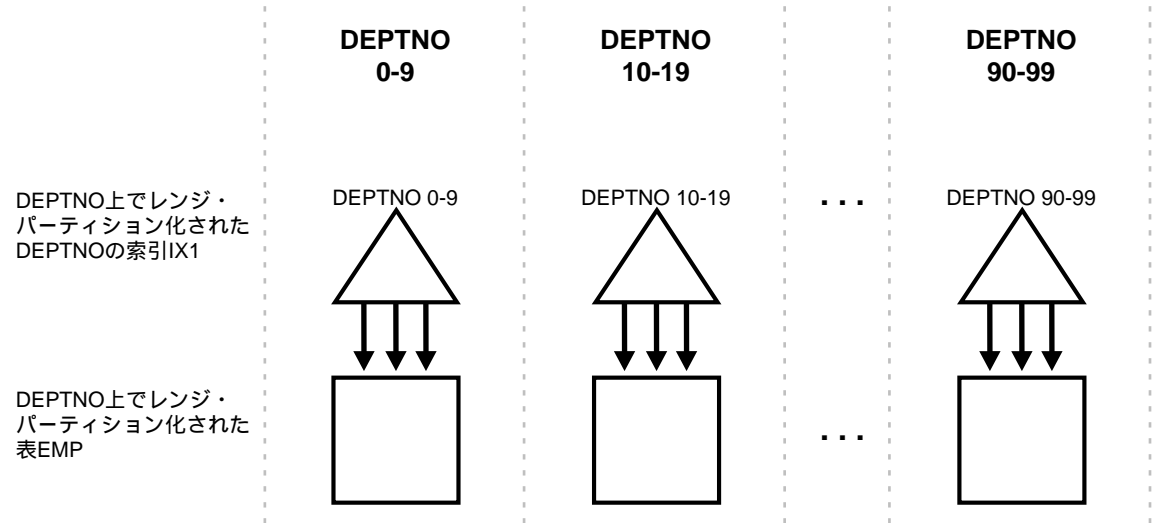
ローカル同一キー索引 索引列の左側のキー（prefix）に基づいてパーティション化されている場合、その索引はローカル「同一キー」（local prefixed）索引になります。

たとえば、SALES 表とそのローカル索引 SALES_IX が WEEK_NUM 列に基づいてパーティション化される場合、索引 SALES_IX は列 (WEEK_NUM,XACTION_NUM) で定義されていれば、「ローカル同一キー」索引になります。これに対して、索引 SALES_IX が列 PRODUCT_NUM で定義されていれば、同一キー索引にはなりません。

図 11-4 では、ローカル同一キー索引の別の例を示します。

ローカル同一キー索引は、一意の索引にも、一意でない索引にもできます。

図 11-4 ローカル同一キー索引

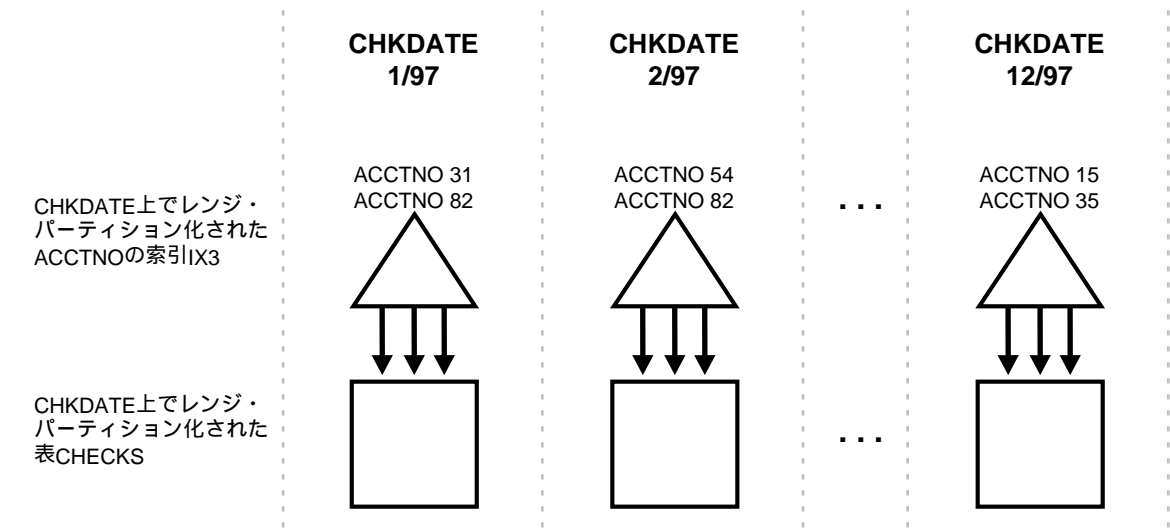


ローカル非同ーキー索引 索引列の左側のキー（prefix）に基づいてパーティション化されていない場合、その索引はローカル「非同ーキー」（local nonprefixed）索引になります。

パーティション化キーが索引キーのサブセットでない限り、一意のローカル非同ーキー索引は作成できません。

図 11-5 では、ローカル非同ーキー索引の例を示します。

図 11-5 ローカル非同一次索引



グローバル・パーティション索引

「グローバル・パーティション索引」では、特定の索引パーティションにあるキーが、基礎になる複数の表パーティションまたはサブパーティションに格納されている行を参照できます。グローバル索引にはレンジ・パーティション化しか適用できませんが、任意のタイプ（レンジ、ハッシュまたはコンポジット・パーティション化）のパーティション表に定義できます。

グローバル索引は、GLOBAL 属性を指定することにより作成されます。作成時にグローバル索引の初期パーティション化を定義する作業と、その後のパーティション化を維持していく作業は、データベース管理者の責任で行う必要があります。索引パーティションは、必要に応じてマージしたり分割できます。

通常、グローバル索引は、基礎になる表と同一レベルでパーティション化されることはありません。グローバル索引を基礎になる表と同一レベルでパーティション化することを妨げる理由は何もありませんが、問合せ計画の生成時やパーティションのメンテナンス操作の実行時には、同一レベル・パーティション化による利点はありません。このため、基礎になる表と同一レベルでパーティション化する索引は、LOCAL として作成してください。

グローバル・パーティション索引には、すべてのパーティション内のすべての行のエントリを持つ 1 つの B* ツリーが（概念的に）含まれています。各索引パーティションには、表内の多くの異なるパーティションまたはサブパーティションを参照するキーが含まれています。

グローバル索引の最も高位のパーティションのパーティション・バウンドには、すべての値に MAXVALUE を指定する必要があります。これにより、基礎になる表のすべての行を索引に含めることができます。

グローバルな同一キー・パーティション索引と非同ーキー・パーティション索引 索引列の左側のキーでパーティション化されている場合、そのグローバル・パーティション索引はグローバル「同一キー」パーティション索引です。(図 11-6 に例を示します。) 索引列の左側のキーでパーティション化されていない場合、そのグローバル・パーティション索引はグローバル「非同ーキー」パーティション索引です。Oracle は、グローバルな非同ーキー・パーティション索引をサポートしていません。

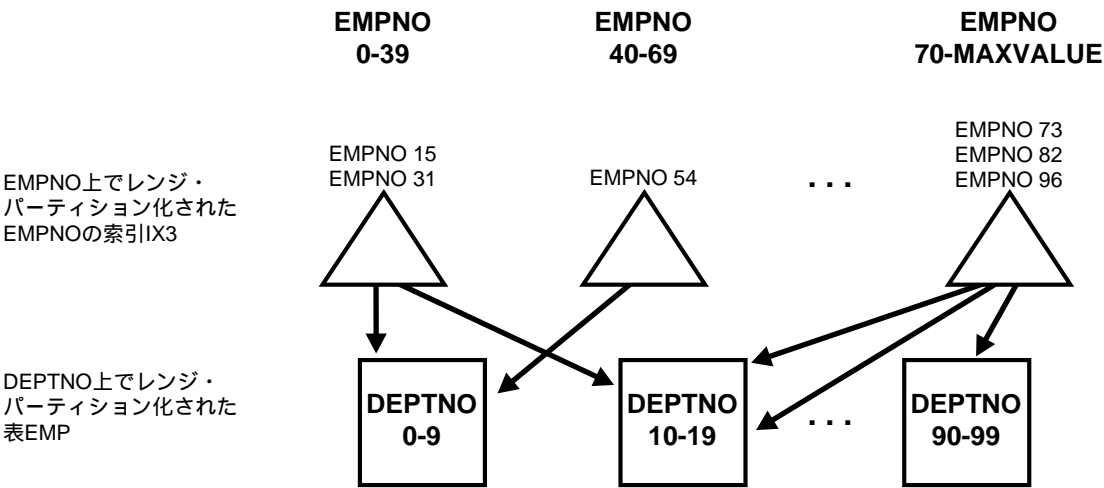
グローバル同一キー・パーティション索引は、一意索引にすることも、非一意索引にすることもできます。

非パーティション索引は、グローバル同一キー非パーティション索引として処理されます。

グローバル・パーティション索引の管理 グローバル・パーティション索引は、ローカル索引よりも次の点で管理が煩雑です。

- 基礎になる表パーティションのデータを移動したり削除したりすると (SPLIT、MOVE、DROP または TRUNCATE)、グローバル索引のすべてのパーティションが影響を受ける。その結果、グローバル索引では、パーティションのメンテナンス (グローバル索引や索引パーティションの再作成など) にパーティション・サイズではなく表サイズに比例した時間がかかり、パーティションの独立はサポートされません。
- 基礎になる表パーティションまたはサブパーティションを特定の時点まで回復する場合、グローバル索引内の対応するエントリもすべて同じ時点まで回復する必要がある。このエントリは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わたって散らばっている (回復の対象ではない別のパーティションまたはサブパーティションのエントリと混在している) ことがあるため、グローバル索引全体を再作成するしか方法がありません。

図 11-6 グローバル同一キー・パーティション索引



パーティション索引の種類のとめ

表 11-2 に、Oracle がサポートする 3 種類のパーティション索引をまとめま。

- 索引が「ローカル」であれば、基礎になる表と同一レベルでパーティション化されま。それ以外の場合、索引は「グローバル」です。
- 「同一キー」索引は、索引列の左側のキーに基づいてパーティション化されま。それ以外の場合、索引は「非同一キー」索引です。

表 11-2 パーティション索引のタイプ

索引のタイプ	表と同一レベルでパーティション化された索引	索引列の左側のキーに基づいてパーティション化された索引	UNIQUE 属性の可否	例		
				表パーティション・キー	索引列	索引パーティション・キー
ローカル同一キー (任意のパーティション化方法)	はい	はい	はい	A	A、B	A
ローカル非同ーキー (任意のパーティション化方法)	はい	いいえ	はい ¹	A	B	A
グローバル同一キー (レンジ・パーティション化のみ)	いいえ ²	はい	はい	A	B	B
グローバル非同ーキー ³	—	—	—	—	—	—

¹ 一意のローカル非同ーキー索引を作成するには、パーティション化キーを索引キーのサブセットにする必要があります。

² グローバル・パーティション索引は基礎になる表と同一レベルでパーティション化できますが、DROP や SPLIT PARTITION などのパーティション・メンテナンス操作を実行した後は、パーティション化の利点を活用したり、同一レベル・パーティション化を維持することはできません。

³ この種類の索引はサポートされていません。

非同ーキー索引の重要性

非同ーキー索引は、履歴データベースの場合に特に便利です。履歴データを含む表では、高速アクセスをサポートするために 1 つの列に対して索引を定義するのが普通ですが、古いデータをロールアウトして新しいデータにロールインする期間をサポートするために別の列（基礎になる表と同じ列）に基づいて索引をパーティション化します。

11-3 ページの図 11-1 の SALES 表を考えます（「週別にパーティション化された SALES 表」）。この表には、1 年分の売上データが、13 個のパーティションに分けて記録されています。WEEK_NO に基づいてレンジ・パーティション化されているので、1 つのパーティションに 4 週間分のデータが入ります。SALES に基づいてローカルの非同ーキー索引 SALES_IX を作成できます。アカウント番号によってデータに高速にアクセスする必要のある問合せがあるため、SALES_IX 索引は ACCT_NO に対して定義されます。ただし、この索引は SALES 表と一致するように WEEK_NO に基づいてパーティション化されます。4 週間ごとに、SALES と SALES_IX の最も古いパーティションが削除され、新しいパーティションが追加されます。

同一キー索引と非同ーキー索引のパフォーマンスへの影響

同一キー索引をブロープするよりも、非同ーキー索引をブロープする方がコストが高くなります。

索引キーが同一キー（ローカルまたはグローバル）で、Oracle に索引列を含む述語が提示された場合、パーティションのプルニングによってその述語の適用を索引パーティションのサブセットに限定できます。

たとえば、11-31 ページの図 11-4（「ローカル同一キー索引」）の場合、述語が DEPTNO=15 であれば、オブティマイザはその述語がその索引の 2 番目のパーティションだけに当てはまることを認識します。（述語にバインド変数が含まれる場合、オブティマイザは正確にどのパーティションであるかは認識できませんが、関係するパーティションが 1 つのみであることは認識します。その場合、実行時には 1 つの索引パーティションのみがアクセスされます。）

索引が非同ーキー索引である場合、Oracle は索引列を含む述語を N 個すべての索引パーティションに適用しなければならないことがあります。これには、1 つのキーを調べるか、索引範囲の走査を実行する必要があります。範囲を指定して走査する場合、Oracle は N 個の索引パーティションからの情報を組み合わせる処理も実行する必要があります。たとえば、11-32 ページの図 11-5（「ローカル非同ーキー索引」）の場合、ローカル索引は ACCTNO の索引キーによって CHKDATE にパーティション化されています。述語が ACCTNO=31 であれば、Oracle は 12 個の索引パーティションすべてを調査します。

パーティション化列に対する述語も含まれている場合には、複数の索引を調査しないですむ場合もあります。Oracle は、ローカル索引と基礎になる表が同一レベルでパーティション化されているという状況を活用して、パーティション・キーに基づいてパーティションを切り詰めます。たとえば、図 11-5 の述語が CHKDATE<3/97 であれば、Oracle が調査する必要のあるパーティションは 2 つだけです。

非同ーキー索引の場合、パーティション・キーが WHERE 句の一部である（ただし索引キーの一部ではない）場合、オブティマイザは、基礎になる表パーティションに基づいて、どの索引パーティションを調査するかを判断します。

ローカルな非同ーキー索引のキーを使用した多くの問合せや DML 文がすべての索引パーティションを調査しなければならない場合、そのような索引によって提供されるパーティションの独立の程度は低くなります。

索引のパーティション化のためのガイドライン

表の索引をどのようにパーティション化するかを決定するときには、その表をアクセスする必要があるいくつかのアプリケーションの組合せを考慮に入れる必要があります。パフォーマンスという面と、可用性と管理しやすさという面の両面を比較検討する必要があります。考慮すべきガイドラインは次のとおりです。

■ OLTP アプリケーションの場合

- グローバル索引とローカル同一キー索引を使用すると、索引パーティション・ブロープの数が最小になるため、ローカル非同ーキー索引を使用するよりもパフォーマンスが向上する。
- ローカル索引は、表に対してパーティションまたはサブパーティションのメンテナンス操作を実行するときに、より高い可用性をサポートする。ローカル非同ーキー索引は、履歴データベースの場合に非常に便利です。

- DSS アプリケーションの場合は、ローカル非同一次索引を使用すると、索引キーに基づくレンジ問合せにより、多くの索引パーティションをパラレルで走査できるため、パフォーマンスを改善できる。

たとえば、11-32 ページの図 11-5 (「ローカル非同一次索引」) の表 CHECKS に対して述語「ACCTNO between 40 and 45」を使用した問合せを実行すると、非同一次索引 IX3 のすべてのパーティションがパラレルで走査されます。一方、11-31 ページの図 11-4 (「ローカル同一次索引」) の表 DEPTNO に対して述語「DEPTNO between 40 and 45」を使用した問合せは、同一次索引 IX1 の 1 つのパーティションにしかアクセスしないため、パラレル化できません。

- 履歴データ表の場合、索引はできるだけローカルにする。これにより、通常の定期的なパーティションの削除操作の影響を抑えることができます。
- パーティション化列でない列の一意索引は、グローバルにする必要がある。キーにパーティション化キーが含まれていない一意のローカル非同一次索引はサポートされていないからです。

索引パーティションの物理属性

デフォルトの物理属性は、CREATE INDEX 文によってパーティション索引を作成する時点で最初に指定します。パーティション索引そのものに対応するセグメントはないため、この属性はメンバー・パーティションの物理属性を導出するときに限り使用されます。デフォルトの物理属性は、ALTER INDEX MODIFY DEFAULT ATTRIBUTES を使用して修正できます。

CREATE INDEX によって作成されるパーティションの物理属性は、次のようにして決まります。

- パーティションの属性が指定されていない場合は、索引に（明示的に、またはデフォルトで）指定された対応する物理属性の値が使用される。LOCAL 索引のパーティションの TABLESPACE 属性を扱うときには、このルールに関して重要な例外があります。つまり、ユーザーが TABLESPACE 値を指定しなかった場合には、基礎になる表の対応するパーティションの値が使用されます。

ALTER TABLE ADD PARTITION の処理過程で作成されたローカル索引のパーティションの物理属性（TABLESPACE を除く）には、各索引のデフォルトの物理属性が設定されます。

ALTER TABLE SPLIT PARTITION によって作成される索引パーティションの物理属性（TABLESPACE を除く）は、次のようにして決まります。

- 分割する索引パーティションの物理属性値が使用される。

既存の索引パーティションの物理属性は、ALTER INDEX MODIFY PARTITION および ALTER INDEX REBUILD PARTITION によって修正できます。その後の属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、文を発行する前のパーティションの属性値が使用される。ALTER INDEX REBUILD PARTITION は、パーティションが存在する表領域を変更するためにも使用できます。

ALTER INDEX SPLIT PARTITION によって作成されるグローバル索引パーティションの物理属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、分割するパーティションの属性値が使用される。

索引のすべてのパーティションの物理属性は（デフォルト値とともに）、ALTER INDEX を使用して変更できます。たとえば、ALTER INDEX *indexname* NOLOGGING は、*indexname* のすべてのパーティションのロギング・モードを NOLOGGING に変更します。

LOB 索引パーティションの物理属性に関する追加情報は、11-40 ページの「[LOB 索引パーティションの表領域および記憶域属性](#)」を参照してください。

LOB 列を持つ表のパーティション化

LOB 列を含む表（12-11 ページの「[LOB データ型](#)」を参照）はパーティション化できますが、パーティション化キーに LOB 列を含めることはできません。LOB 列の LOB データおよび LOB 索引セグメントは、実表と同一レベルでパーティション化されます。

注意： この項では、LOB データと LOB 索引を区別していますが、両者は別々のエントリではありません。LOB 索引は、システムによって暗黙的に作成およびメンテナンスされます。これは制御情報を含んでおり、LOB 列記憶領域の重要部分です。

LOB 列を含むパーティション表のすべてのパーティションには、LOB データ・パーティション用の LOB データ・セグメントと、LOB 索引パーティション用の LOB 索引セグメントがあります。これらのデータ・セグメントと索引セグメントには、そのパーティションの行に属する LOB が格納されます。

同様に、LOB 列を含むコンポジット・パーティション表のすべてのサブパーティションには、LOB データ・サブパーティション用の LOB データ・セグメントと、LOB 索引サブパーティション用の LOB 索引セグメントがあります。これらのデータ・セグメントと索引セグメントには、そのサブパーティションの行に属する LOB が格納されます。この後の説明では、「パーティション」は、レンジ・パーティション化またはハッシュ・パーティション化された表のパーティション、あるいは、コンポジット・パーティション化された表または索引のサブパーティションを指します。

LOB データ・セグメントと LOB 索引セグメントを同一レベルでパーティション化することによって、メンテナンス操作の効果がローカル化され、リソースの使用効率が向上し、データの可用性が改善されます。11-56 ページの「[LOB 列を持つ表のパーティション・メンテナンス操作](#)」を参照してください。

LOB データ・パーティションの表領域および記憶域属性

特定の LOB データ・パーティション用の表領域を決定するためのアルゴリズムは、LOCAL 索引パーティションの表領域を決定するためのアルゴリズムに似ています。LOB データ・パーティションについて、TABLESPACE 以外の物理記憶域属性の値を決定する場合は、表パーティションの物理属性値を決定するときと同じアルゴリズムが使用されます。特定の LOB 列の LOB 記憶域特性は、パーティション・レベルまたは表レベルで明示的に指定できるので注意してください。

LOB データ・パーティションの TABLESPACE 属性 LOB データ・パーティションの表領域を決定するルールは、次のとおりです。

1. 特定の LOB データ・パーティション用に表領域が指定されていれば、その値が使用されます。
2. 表領域が指定されていない場合、表の特定の LOB 列のすべての LOB データ・パーティションについて、「TABLESPACE DEFAULT」以外」のデフォルトの TABLESPACE 値が指定されていれば、その値が使用されます。
3. それ以外の場合、LOB データ・パーティションは、それに対応付けられている表パーティションと同じ位置に配置されます。

次の例は、これらのルールを示しています。

```
CREATE TABLE PT1 (A NUMBER, B BLOB, C CLOB, D CLOB)
  LOB (B,D) STORE AS (STORAGE (NEXT 15K))
  LOB (C) STORE AS (TABLESPACE TSB)
  PARTITION BY RANGE (A)
    (PARTITION P VALUES LESS THAN (MAXVALUE) TABLESPACE TS1
      LOB (B) STORE AS (TABLESPACE TSA),           (Rule 1)
      LOB (C) STORE AS (PCTVERSION 20),           (Rule 2)
      LOB (D) STORE AS (STORAGE (NEXT 10K)))       (Rule 3)
  TABLESPACE TSX;
```

この例では、パーティション P に対応する LOB データ・パーティションの表領域が、次のように決定されます。

- 列 B の LOB データ・パーティションは、表領域 TSA に配置される (ルール 1)。
- 列 C の LOB データ・パーティションは、表領域 TSB に配置される (ルール 2)。
- 列 D の LOB データ・パーティションは、表領域 TS1 に配置される (ルール 3)。

LOB 索引パーティションが配置される表領域の決定方法は、11-40 ページの「[LOB 索引パーティションの表領域および記憶域属性](#)」を参照してください。

LOB データ・パーティションのその他の記憶域属性 LOB データ・パーティションの記憶域属性 (TABLESPACE 以外) の値は、次のように決定されます。

1. 特定の LOB データ・パーティションの値が指定されていれば、その値が使用されます。

- 2. 値が指定されていない場合、表の特定の LOB 列のすべての LOB データ・パーティションについて、表レベルでデフォルト値が指定されていれば、その値が使用されます。
- 3. それ以外の場合は、システムまたは表領域のデフォルト値が使用されます。ただし、LOGGING の場合は、CACHE が明示的に指定されていると、表領域の値に関係なく LOGGING が使用されます (CACHE NOLOGGING がサポートされないため)。

LOB 索引パーティションの記憶域属性の決定方法は、次の項を参照してください。

LOB 索引パーティションの表領域および記憶域属性

LOB 索引パーティションは、常に対応する LOB データ・パーティションと同じ表領域に常駐します。つまり、LOB 索引パーティションは、LOB データ・パーティションと同じ位置に配置されます。LOB 索引パーティションの他のすべての属性は、対応する LOB データ・パーティションの属性と、LOB データおよび対応する LOB 索引パーティションの両方が常駐する表領域のデフォルト属性に基づいて決定されます。

注意： LOB 索引またはそのパーティションの属性は指定できません。

LOB 索引パーティションの TABLESPACE 属性 次の例は、LOB 索引パーティションが、それに対応する LOB データ・パーティションとともに、どのようにして同じ位置に配置されるかを示しています。

```
CREATE TABLE PT1 (A NUMBER, B BLOB, C CLOB, D CLOB)
  LOB (B,D) STORE AS (STORAGE (NEXT 15K))
  LOB (C) STORE AS (TABLESPACE TSB);
PARTITION BY RANGE (A)
(PARTITION P VALUES LESS THAN (MAXVALUE) TABLESPACE TS1
  LOB (B) STORE AS (TABLESPACE TSA),                (Rule 1)
  LOB (C) STORE AS (PCTVERSION 20),                (Rule 2)
  LOB (D) STORE AS (STORAGE (NEXT 10K)));            (Rule 3)
TABLESPACE TSX;
```

この例で、パーティション P に対応付けられている LOB データ・パーティションに対応する LOB 索引パーティションは、次の表領域に配置されます。

- 列 B の LOB 索引パーティションは、表領域 TSA に配置される (ルール 1)。
- 列 C の LOB 索引パーティションは、表領域 TSB に配置される (ルール 2)。
- 列 D の LOB 索引パーティションは、表領域 TS1 に配置される (ルール 3)。

LOB 索引パーティションのその他の記憶域属性 LOB 索引パーティションの記憶域属性 (TABLESPACE 以外) の値は、対応する LOB データ・パーティションの属性値と、LOB 索引パーティションが配置されている表領域のデフォルト属性に基づいて決定されます。

ビューとパーティション LOB

LOB 列を持つパーティション表の通常のビューは、LOB 列を持たない表の場合と同様に機能します。また、LOB 列を持つパーティション表の最上位に、オブジェクト・ビューを作成できます。

パーティション表のビューから選択された LOB には、非パーティション表から選択された LOB の場合と同じ、ビューベースの権限チェックが実行されます。ユーザーには、LOB ロケータの取得元 (SELECT される) のビューを介して LOB にアクセスするための権限が必要です。このビューベースの権限チェックは、スナップショット (つまり、レプリケーションに使用されるマテリアライズド・ビュー) に必要です。

パーティション表の BFILE

BFILE の場合、表には LOB ロケータのみが格納され、実際の BFILE データは外部のオペレーティング・システム・ファイルにあります。したがって、BFILE ロケータは、BFILE データではなく表の残りの部分とともにパーティション化されます。また、BFILE ロケータは可変長であり、ディレクトリの別名、ファイル名およびその他の制御情報が格納されます。したがって、パーティション表の BFILE 列は、パーティション表の VARCHAR2 列に似ています。

索引構成表と 2 次索引のパーティション化

列値の範囲を指定して索引構成表をパーティション化できます。索引構成表と通常の (ヒープ構成) 表の違いは、次のとおりです。

- 索引構成表には常に主キーがあるが、通常の表には主キーがない場合がある。
- 索引構成表の行は、主キー索引セグメントのリーフ・ブロックに、索引行の一部として格納される。
- 索引構成表は、必要に応じて、主キー索引セグメントだけでなく、行オーバーフロー・データ・セグメントも持つことができる。したがって、通常の表 (LOB なし) は単一データ・セグメントに格納されますが、索引構成表 (LOB なし) には、データを格納するための索引セグメントとオーバーフロー・データ・セグメント (オプション) が必要です。

詳細は、10-35 ページの「[索引構成表](#)」を参照してください。

索引構成表をパーティション化する場合の注意事項は、次のとおりです。

- レンジ・パーティション化しかサポートされない。
- パーティション列は、主キー列のサブセットにする必要がある。
- 2 次索引も、ローカルとグローバルの両方でパーティション化できる。
- OVERFLOW データ・セグメントは、常に表パーティションと同一レベルでパーティション化される。

- 表データとオーバーフロー・データの記憶域属性は、表レベルまたは個々のパーティション・レベルで指定できる。

レンジ・パーティション化と主キー列

パーティション化する列を主キー列のサブセットに限定することによって、パーティションに行を挿入するときに、そのパーティションを検索して主キーが一意かどうかを確実に検証できます。(この制限がなければ、他のパーティションも検索する必要があるため、パーティション相互の独立性がなくなります)。

パーティション化する列が主キー列の接頭辞を形成する場合、パーティション・バウンドは主キー順の順序を形成します。複数のパーティションからのデータを必要とする問合せの場合は、各パーティションから得られた行の単純連結によって、主キー順が保持されます。これは、索引構成表のパーティション化に最適の方法です。

パーティション化する列が主キー列の接頭辞になっていない場合、各パーティションのデータは主キー順にソートされますが、複数のパーティションから行を主キー順に選択するには、個別にソートされたパーティション行をマージする必要があります。

主キー列のサブセットでない列に基づいて索引構成表をパーティション化する場合は、次のようにします。

1. パーティション化する列を主キーの最後に追加して、主キー列の一部にします。
2. 元の主キー列に対して一意制約を定義します。

たとえば、列 A、B および C と主キー (A, B) を持つ索引構成表の場合、この表を列 C に基づいてパーティション化するには、主キーを (A, B, C) に変更し、(A, B) の一意制約を定義する必要があります。これにより、挿入操作では、行がターゲット・パーティションに挿入され、(A, B) のキー値が (A, B) の非パーティション索引に挿入されます。これにより、すべてのパーティション間で一意性が検証されます。

行オーバーフローなしの索引構成表

行オーバーフローなしのパーティション化された索引構成表を作成するには、表レベルでのみ ORGANIZATION INDEX を指定する必要があります。すべてのパーティションは、ORGANIZATION INDEX プロパティを表から継承します。

物理属性のデフォルト値は、表レベルで指定してパーティション・レベルで上書きできます。これらの属性は、パーティションごとに作成される主キー索引に適用されます。索引セグメントの表領域は、パーティション・レベルまたは表レベルで指定できます。どちらのレベルでも指定しなければ、ユーザーのデフォルト表領域が使用されます。

次の例は、行オーバーフローがない索引構成表の作成方法を示しています。

```
CREATE TABLE orders(  
    id NUMBER, odate DATE, ...  
    PRIMARY KEY(id, odate))  
    ORGANIZATION INDEX  
    PARTITION BY RANGE(odate)  
( PARTITION p1 ... TABLESPACE q1,  
    PARTITION p2 ... TABLESPACE q2);
```

この例で、索引構成表 ORDERS は ODATE 列に基づいてパーティション化され、各パーティションが専用の表領域に格納されます。オーバーフローは生じません。

行オーバーフロー付きの索引構成表

オーバーフロー・オプションを使用すると、行の末尾部分をオーバーフロー・データ・セグメントに格納できます。ここでは、オーバーフロー付きでパーティション化された索引構成表の概要を示します。

- オーバーフロー付きでパーティション化された索引構成表の場合、各パーティションは索引セグメントとオーバーフロー・データ・セグメントを持つ。
- オーバーフロー・データ・セグメントは、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る。
- 索引セグメントの物理属性と同様に、オーバーフロー・データ・セグメントの物理属性のデフォルト値を表レベルで指定し、パーティション・レベルの値を指定して上書きできる。
- OVERFLOW キーワードの前のすべての属性は、主キー索引セグメントに適用され、OVERFLOW キーワードの後のすべての属性は、オーバーフロー・データ・セグメントに適用される。
- PCTTHRESHOLD および INCLUDING *column* のデフォルト値は、表レベルでのみ指定できる。これらの句によって、非キー部分を先頭と末尾の行断片に分割する操作が制御されます。先頭の行断片は索引行に格納され、末尾の行断片はオーバーフロー・データ・セグメントに格納されます。
- オーバーフロー・データ・セグメントの表領域を指定しなければ、表レベル・デフォルトに設定される。表レベル・デフォルトが指定されていない場合は、対応するパーティションの索引セグメントの表領域が使用されます。
- 索引データ・セグメントとオーバーフロー・データ・セグメントのシステム生成名の書式は、それぞれ SYS_IOT_TOP_Pn と SYS_IOT_OVER_Pn である。

次の例は、パーティション化された索引構成表を作成し、パーティション化されたオーバーフローを単一の表領域に格納する操作を示しています。

```
CREATE TABLE orders(  
    id NUMBER, odate DATE, notes VARCHAR2(1000), ...  
    PRIMARY KEY(id, odate))  
    ORGANIZATION INDEX INCLUDING odate  
    OVERFLOW TABLESPACE all_overflow  
    PARTITION BY RANGE(odate)  
    ( PARTITION p1 ... TABLESPACE q1,  
      PARTITION p2 ... TABLESPACE q2);
```

この例で、表には、オーバーフロー・データ・セグメント用に別の表領域があります。オーバーフロー・データ・セグメントは、同じ物理表領域（ALL_OVERFLOW）に格納されていますが、索引構成表に使用されるのと同じパーティション列に基づいてパーティション化されます。INCLUDING ODATE 句が使用されていることに注目してください。これは、ODATE 列以後のすべてのデータがオーバーフローに格納されることを意味します。

次の例は、パーティション化された索引構成表を作成し、パーティション化されたオーバーフローを複数の表領域に格納する操作を示しています。

```
CREATE TABLE orders(  
    id NUMBER, odate DATE, notes VARCHAR2(1000), ...  
    PRIMARY KEY(id, odate))  
    ORGANIZATION INDEX INCLUDING odate  
    PARTITION BY RANGE(odate)  
    ( PARTITION p1 ... TABLESPACE q1  
      OVERFLOW TABLESPACE q1_overflow,  
      PARTITION p2 ... TABLESPACE q2  
      OVERFLOW TABLESPACE q2_overflow);
```

この例で、パーティション化されたオーバーフロー・セグメントは、それぞれ専用の表領域に格納されます。

索引構成表の 2 次パーティション索引

索引構成表には、ローカル同一キー、ローカル非同一次キーおよびグローバル同一キーというパーティション索引を作成できます。索引構成表の索引には、物理 ROWID ではなく主キーベースの（論理）ROWID が格納されます。また、2 次索引ベースのアクセスをスピードアップするために、ROWID に関して追加の「推測」データを含めることができます。詳細は、10-37 ページ「索引構成表の 2 次索引」および 12-18 ページの「論理 ROWID」を参照してください。

グローバル・パーティション索引によって索引構成表パーティションにアクセスするために、Oracle は論理 ROWID に基づいてパーティションを識別します。この操作が可能なのは、ROWID に主キー列が含まれ、主キー列にはすべてのパーティション化列が含まれているからです。パーティションが識別されると、Oracle は「推測」を使用して、索引行が保持されるリーフ・ブロックに直接アクセスできます。「推測」が無効であれば、関連するパーティションの B* ツリーに対して索引スキャンを実行する必要があります。

DML パーティション・ロックとサブパーティション・ロック

DML 表ロックは、DML 文 (INSERT、UPDATE および DELETE) と、DDL 文および LOCK TABLE 文とを同期化します。さらに、DML 表ロックは、DDL 文および LOCK TABLE 文も相互に同期化します。パーティション表またはサブパーティション表の場合、Oracle は DML パーティション・ロックまたは DML サブパーティション・ロックを使用して、DDL およびユーティリティ操作のために「パーティションを独立」させます。

- レンジ・パーティション化またはハッシュ・パーティション化された表には、「DML パーティション・ロック」が適用される。
- コンポジット・パーティション化を使用する表には、「DML サブパーティション・ロック」が適用される。

パーティションを独立 (またはサブパーティションを独立) させることによって、他のパーティションやサブパーティションのアクティビティを減少させずに、選択したパーティションまたはサブパーティションに対して、DDL およびユーティリティ操作を実行できます。

DML パーティション・ロック

パーティション・ロックは、パーティション表の個々のパーティション内のデータを保護すると同時に、複数のユーザーが表内のパーティションに同時にアクセスできるようにします。

パーティション・ロックは、DML ロックの階層内で表ロックと行ロックの中間に位置します。

- 表ロック
 - パーティション・ロック
 - * 行ロック

パーティション・ロックは、表ロックと同じモードで取得できます。共有 (S)、排他 (X)、行共有 (SS)、行排他 (SX) および共有行排他 (SSX) モードがあります。DML および DDL 文に対するパーティション・ロックの詳細は、11-49 ページの「[メンテナンス操作の同時実行モデル](#)」を参照してください。

LOB 列に対する DML 操作中のパーティション・ロック

LOB 全体またはその一部のみの更新（DBMS_LOB 操作を使用）中に、Oracle はパーティション表の DML SX ロックだけでなく、1 つ以上の表パーティションの DML SX ロックも取得します。

DML サブパーティション・ロック

DML サブパーティション・ロックにより、同じパーティションの他のサブパーティション（および、他のパーティションのサブパーティション）に対するアクティビティを減少させずに、選択したサブパーティションに対して DDL およびユーティリティ操作を実行できます。

サブパーティション・ロックは、個々のサブパーティション内のデータを保護するだけでなく、そのサブパーティション、同じパーティションの他のサブパーティションまたは表の他のパーティションの一部のサブパーティションに、複数のユーザーが同時にアクセスできるようにします。

コンポジット・パーティション表の DML または DDL 操作の実行時には、Oracle は DML パーティション・ロックを取得しません。

- 特定のサブパーティション内のデータにアクセスする DML 操作は、レンジ・パーティション化表またはハッシュ・パーティション化表のパーティション内のデータにアクセスする同様の DML 操作と同じモードで、そのサブパーティションの DML ロックを取得する。
- 「サブパーティションのメンテナンス操作」は、操作に関与するサブパーティションの DML ロックを取得する。
- コンポジット・パーティション表のパーティションに対するメンテナンス操作（SPLIT PARTITION または TRUNCATE PARTITION など）は、その操作に関与するパーティションに属するすべてのサブパーティションの DML ロックを取得する。

パーティション・ロックと同様に、サブパーティション・ロックは、DML ロックの階層内で表ロックと行ロックの中間に位置します。

- 表ロック
 - サブパーティション・ロック
 - * 行ロック

サブパーティションの DML ロックは、表およびパーティションの DML ロックと同じモードで取得できます。共有（S）、排他（X）、行共有（SS）、行排他（SX）および共有行排他（SSX）モードがあります。

Oracle Parallel Server のパフォーマンスの考慮事項

DML ロックの余分なレベルを実行すると、余分のメッセージが分散ロック・マネージャに送られるので、Oracle Parallel Server 環境において短いトランザクションのパフォーマンスに影響することがあります。

Oracle Parallel Server 環境でパフォーマンスを改善するために、ALTER TABLE DISABLE TABLE LOCK 文を使用して、選択した表の DML ロックをオフにできます。これにより、表およびパーティションの DML ロックがどちらも使用禁止になります。DML ロックが使用禁止のときには、DDL 文は使用できません。

追加情報：『Oracle8i Parallel Server 概要および管理』を参照してください。

メンテナンス操作

この項では、次の内容を取り上げます。

- [パーティションのメンテナンス操作](#)
- [索引の管理](#)
- [パーティション表およびパーティション索引についての権限](#)
- [パーティション表およびパーティション索引についての監査](#)

この章の目的上、「メンテナンス操作」とは、表や索引の定義を変更する DDL 文、またはデータの大量ロードやアンロードを実行するユーティリティ（Export、Import および SQL*Loader など）を指します。

非パーティション表と非パーティション索引に対する既存のメンテナンス操作は、すべてパーティション表とパーティション索引に対して実行できます。たとえば、DROP TABLE でパーティション表を削除したり、Export でパーティション表をエクスポートしたりできます。ただし、メンテナンス操作の中には、パーティション表またはパーティション索引全体ではなく、個々のパーティションに対して実行する必要があるものもあります。たとえば、ALTER TABLE ALLOCATE EXTENT は、レンジ・パーティション表に対して実行できません。そのため、パーティションに新規エクステントを追加するときは、そのパーティションに対して ALTER TABLE MODIFY PARTITION ALLOCATE EXTENT を使用します。

メンテナンス操作は、所要時間が操作対象のスキーマ・オブジェクトのサイズ（レコード数）の影響を受けなければ、高速であるとみなされます。高速なメンテナンス操作は、ディクショナリとセグメント・ヘッダーを変更するだけで、データを走査したり更新したりしません。その操作は、短時間（数秒）で完了します。たとえば、RENAME は高速な操作ですが、CREATE INDEX は高速な操作ではありません。

パーティションのメンテナンス操作

「パーティション・メンテナンス操作」は、パーティション表またはパーティション索引のパーティションの1つを修正します。たとえば、既存の表に新しいパーティションを追加したり、I/O 負荷のバランスを改善するためにパーティションを別の表領域に移動したり、パーティションをロードする操作があります。

パーティションのメンテナンス操作の中には、計画されたイベントであるものもあります。たとえば、履歴データベースでは、データベース管理者 (DBA) はデータベースから最も古いパーティションを定期的に削除し、一連の新しいパーティションを追加します。この削除操作と追加操作は、定期的な予定に基づいて実行されます。計画されたメンテナンス操作の別の例として、データを再クラスタ化して断片化を減らす目的で行う、定期的なエクスポート / インポートがあります。

パーティションのその他のメンテナンス操作は、計画されていないイベントであり、アプリケーションやシステムの問題から回復するために必要になるものです。たとえば、予期していなかったトランザクション・アクティビティが発生すると、DBA はもう一度 I/O 負荷のバランスをとるためにパーティションを分割したり、1 つ以上のパーティションを再作成する必要が生じることがあります。

パーティションのメンテナンス操作は、次のとおりです。

- パーティションまたはサブパーティションを既存の表に追加する。
- 表のパーティション (レンジ・パーティション化またはコンポジット・パーティション化) をマージする。
- 表のパーティション (ハッシュ・パーティション化) またはサブパーティション (コンポジット・パーティション化) を合わせる。これによりパーティションまたはサブパーティションの内容を、残りの 1 つ以上のパーティションまたはサブパーティションに再分配します。
- 既存のパーティションを 2 つのパーティションに分割する (レンジ・パーティション化またはコンポジット・パーティション化)。
- パーティション (レンジ・パーティション化またはコンポジット・パーティション化) を削除する。
- 表パーティションまたはサブパーティションを切り捨てる (領域の再生を要求する、または要求しない)。
- パーティションまたはサブパーティションを交換する。これにより、表パーティションまたはサブパーティションのデータ (および、可能であればローカル索引セグメント) を、非パーティション表のデータ (および索引セグメント) に変換します。
- パーティションまたはサブパーティションを変更する。その物理属性を変更します。
- パーティション (コンポジット・パーティション化) のデフォルト属性を変更する。パーティションの新しいサブパーティションのデフォルト属性を指定します。

- パーティションを移動する。パーティションを別の表領域に移動するか、再クラスタ化するか、パラメータ（作成時のパラメータを含む）を変更します。
- パーティションまたはサブパーティションを改名する。
- 表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、すべて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に、UNUSABLE マークを付ける。
- 索引パーティションまたはサブパーティションを再作成する。
- 1つの表パーティションまたはサブパーティションにデータをロードする。
- 1つの表パーティションまたはサブパーティションからデータをエクスポートする。
- 表パーティションまたはサブパーティションをインポートする。

LOB データのメンテナンスの詳細は、11-56 ページの「[LOB 列を持つ表のパーティション・メンテナンス操作](#)」を参照してください。

追加情報： パーティション・メンテナンス操作の詳細は、『Oracle8i SQL リファレンス』の ALTER TABLE および ALTER INDEX の項を参照してください。

メンテナンス操作の同時実行モデル

この項で説明する同時実行モデルは、複数の DDL 操作とユーティリティ操作を同じスキーマ・オブジェクトに対して同時に実行するための条件を定義します。また、どの問合せおよび DML 操作を DDL 操作やユーティリティ操作と同時に実行できるかについても定義しています。

このモデルは、すべての DDL 文に適用されます。また、SQL*Loader のようなユーティリティにも適用されます。

1ステップ操作と3ステップ操作 メンテナンス操作には、1ステップ操作と3ステップ操作の2種類があります。

1ステップ操作は、次のような操作です。

- この操作は、関係する表を排他（X）モードで DML ロックします。索引操作は、基礎になる表をロックします。さらに、操作の実行中は、排他ディクショナリ・ロックが保持されます。
- この操作には、高速で実行されるもの（レンジ・パーティション化のための ALTER TABLE ADD PARTITION など）と、他の操作を同時に実行できないもの（ALTER TABLE ADD column など）がある。
- 次の操作を除き、すべての索引操作は1ステップである。
 - CREATE INDEX および ALTER INDEX REBUILD

- 削除または分割するグローバル・パーティションが USABLE の場合の、ALTER INDEX DROP または SPLIT PARTITION
- 次の文を除き、すべての Oracle DDL 文は 1 ステップ操作である。
 - CREATE INDEX
 - MOVE、SPLIT、REBUILD PARTITION または SUBPARTITION
 - EXCHANGE PARTITION または SUBPARTITION WITH VALIDATION
 - ハッシュ・パーティション化の ADD PARTITION とコンポジット・パーティション化の ADD SUBPARTITION (処理方法は、SPLIT および MOVE PARTITION または SUBPARTITION 操作と同じ)
 - COALESCE PARTITION または SUBPARTITION
 - LOAD、EXPORT、IMPORT PARTITION または SUBPARTITION
- コンポジット・パーティション化された表およびローカル索引の場合、すべてのパーティション・メンテナンス操作には、レンジ・パーティション化された表およびローカル索引に対する同様の操作と同じプロトコルが適用される。

3 ステップ操作は、次のような操作です。

- この操作は、関係する表に対して制限の弱い DML ロックを取得する。排他 (X) モードで 1 つのパーティションまたはサブパーティションのみをロックするか、表全体をロックする場合でも、S モード、SS モード、SX モードのどれかでロックします。
- この操作には次の 3 つのステップがある。
 - ステップ 1: 共有ディクショナリ・ロックを保持した状態で、ディクショナリを読み込む。ステップ 1 は、短時間 (数秒) で実行されます。このステップの終了時に、適切な DML ロックが取得され、ディクショナリ・ロックが解放されます。
 - ステップ 2: 表または索引レコードを走査または更新する。ステップ 2 には、長時間かかることがあります (数分または数時間)。
 - ステップ 3: 排他ディクショナリ・ロックを保持した状態で、ディクショナリを更新する。ステップ 3 は、短時間 (数秒) で実行されます。
- これらの操作は長時間要するが、その他の操作を同時に実行できる。実際にどの操作を同時に実行できるかは、次に説明するとおり、文によって取得される特定の DML ロックによって異なります。
- レンジ・パーティション化された表またはローカル索引に対する 3 ステップ操作は、パーティションの DML ロックを取得する。コンポジット・パーティション化された表またはローカル索引に対する 3 ステップ操作も、操作に関与する各パーティションのすべてのサブパーティションのロックを取得します。
- 次の操作は 3 ステップ操作である。

- ALTER TABLE MOVE PARTITION または SUBPARTITION、ALTER TABLE SPLIT PARTITION、ALTER TABLE EXCHANGE PARTITION または SUBPARTITION WITH VALIDATION、表パーティションまたはサブパーティションのダイレクト・パス・ロード。これらの文は、表を行排他 (SX) モードでロックし、パーティションまたはサブパーティションを排他 (X) モードでロックします。
- CREATE INDEX および ALTER INDEX REBUILD PARTITION (グローバル索引の場合)。これらの文は、表を共有 (S) モードでロックします。
- ALTER INDEX REBUILD PARTITION または SUBPARTITION (ローカル索引の場合)。この文は、表を行共有 (SS) モードでロックし、パーティションまたはサブパーティションを共有 (S) モードでロックします。
- ALTER TABLE MODIFY PARTITION REBUILD UNUSABLE LOCAL INDEXES。この文は、表を行共有 (SS) モードでロックし、パーティションを共有 (S) モードでロックします。
- ALTER TABLE MODIFY PARTITION ADD SUBPARTITION。この文は、表を行排他 (SX) モードでロックし、サブパーティションを排他 (X) モードでロックします。
- ALTER TABLE COALESCE PARTITION。この文は、表を行排他 (SX) モードでロックし、表の最後のパーティションと、そのパーティションからの行がリフレッシュされるパーティションを排他 (X) モードでロックします。
- ALTER TABLE MODIFY PARTITION COALESCE SUBPARTITION。この文は、表を行排他 (SX) モードでロックし、パーティションの最後のサブパーティションと、そのパーティションからの行がリフレッシュされるサブパーティションを排他 (X) モードでロックします。
- LOAD PARTITION または SUBPARTITION。順番に実行されると、この文は表を行排他 (SX) モードでロックし、ロードされるパーティションまたはサブパーティションを排他 (X) モードでロックします。並列に実行されると、表を行共有 (SS) モードでロックし、ロードされるパーティションまたはサブパーティションを共有 (S) モードでロックします。
- EXPORT PARTITION または SUBPARTITION。この文は、DML ロックを取得しません (パーティションまたはサブパーティションからの SELECT を実行します)。
- IMPORT PARTITION または SUBPARTITION。この文は、表とデータのインポート先 (サブパーティションへの INSERT) となるパーティションまたはサブパーティションを、行排他 (SX) モードでロックします。

最後に、操作によっては、状況に応じて 1 ステップ・プロトコルと 3 ステップ・プロトコルのどちらかに従うものがあります。

■ ALTER TABLE DROP PARTITION および ALTER TABLE TRUNCATE PARTITION

変更する表にグローバル索引が定義されていない場合、または使用可能 (ENABLED) 参照制約によって表を参照する場合、このグループの文は 1 ステップ・プロトコルを使用して実行されるため、高速になります。それ以外の場合は、3 ステップ・プロトコルを使用して実行されます。後者の場合、実表は行排他 (SX) モードでロックされ、パーティションは排他 (X) モードでロックされます。

■ ALTER INDEX SPLIT PARTITION (グローバル索引の場合のみ)

分割するパーティションが USABLE であれば、文は 3 ステップ・プロトコルで実行され、SPLIT の結果として作成されるパーティションは USABLE になります。他方、分割するパーティションが UNUSABLE であれば、操作は 1 ステップ・プロトコルで実行され、その結果として作成されるパーティションも UNUSABLE になります。

■ ALTER INDEX DROP PARTITION (グローバル索引の場合のみ)

削除するパーティションが USABLE であれば、文は 3 ステップ・プロトコルで実行されます。それ以外の場合は、1 ステップ・プロトコルで実行されます。

従来型バスの SQL*Loader と Import では SQL INSERT が使用されるため、このモデルにおいては DML 操作として分類されます。Export では SQL SELECT が使用されるため、問合せ操作として分類されます。

同時に実行できる操作 この項のルールは、1 ステップ操作と 3 ステップ操作の定義から導き出せるものです。

1 ステップ操作が進行中の場合

- 表に対する問合せを実行できる。
- その他の操作 (DDL、ユーティリティまたは DML) は実行できない。

問合せ (READ 操作) には DML ロックが必要ないため、分割または移動するパーティションに対する問合せは、SPLIT や MOVE の処理中でも許可されます。ただし、操作の終了時に現行のセグメントが削除され、その領域が再使用されます。領域が再使用されると、エラーが通知されます。

ALTER TABLE MOVE PARTITION、ALTER TABLE SPLIT PARTITION、ALTER TABLE EXCHANGE PARTITION、または表パーティションのダイレクト・バス・ロードがパーティションに対して進行中の場合

- 同じ表内の別のパーティションを移動、分割またはダイレクト・バス・ロードできる。
- 表に対する問合せを実行できる。
- パーティションに書き込みを実行しない DML 操作であれば、表に対する DML 操作を実行できる。

- パーティションに対するローカル索引パーティション以外のローカル索引パーティションであれば、再作成できる。
- 前述以外のメンテナンス操作を表やその索引に対して実行することはできない。

USABLE パーティションに対する CREATE INDEX、ALTER INDEX REBUILD PARTITION または ALTER INDEX DROP/SPLIT PARTITION（グローバル索引の場合）が進行中の場合

- 基礎になる表に対する問合せを実行できる。
- 表に対して他の索引を作成したり、既存の索引にパーティションを再作成したり、既存の索引内の USABLE パーティションを削除または分割できる。
- 表に対する DML 操作は実行できない。また、前述以外のメンテナンス操作を表やその索引に対して実行することはできない。

ALTER INDEX REBUILD PARTITION（ローカル索引の場合）が、基礎になる表パーティションに対応するパーティション上で進行中の場合

- 基礎になる表のパーティション以外のパーティションを移動、分割またはダイレクト・パス・ロードできる。
- 表に対する問合せを実行できる。
- 基礎になる表パーティションに書き込みを実行しない DML 操作であれば、表に対する DML 操作を実行できる。
- 索引内の他のパーティションを再作成できる。また、表に対して別の索引を作成したり、別の索引内のパーティションを再作成できます。
- 前述以外のメンテナンス操作を表やその索引に対して実行することはできない。

表のパーティションに対するメンテナンス操作の中には、表パーティションまたは索引パーティションのグローバル索引を UNUSABLE にするものもあります。1 つの例は、ALTER TABLE MOVE PARTITION です。DBA は、パーティションのメンテナンス操作に加えて、グローバル索引の再作成操作を含むスクリプトを実行する必要があります。結果的にユーザー側から見ると、この操作は表全体へのシリアル・アクセスになります。ALTER TABLE MOVE/SPLIT PARTITION のような操作は、パーティション・グローバル索引のすべてのパーティションだけではなく、非パーティション・グローバル索引も UNUSABLE にします。

グローバル索引のすべてのパーティションをマークする表パーティション操作は、ローカル索引のうち 1 つのパーティション（操作される表パーティションに対応するパーティション）も UNUSABLE としてマークします。

同様に、パーティションのメンテナンス操作によっては、操作の前に参照整合性制約を使用禁止にして、操作後に再び使用可能にしなければならないものがあります。1 つの例として、空ではないパーティションの ALTER TABLE DROP PARTITION があります。DBA は、パーティションのメンテナンス操作に加えて、制約を再び使用可能にするスクリプトを実行する必要があります。結果的に、ユーザー側から見ると、この操作は表全体へのシリアル・アクセスになります。

表 11-3 は、サブパーティションのメンテナンス操作と同時に実行できる操作を示しています。

表 11-3 サブパーティションに対する同時操作

メンテナンス操作	同時に実行できる操作
ALTER TABLE/INDEX MODIFY DEFAULT ATTRIBUTES OF PARTITION	表の問合せ
ALTER TABLE EXCHANGE SUBPARTITION WITHOUT VALIDATION	
ALTER TABLE/INDEX MODIFY SUBPARTITION (ALLOCATE EXTENT を指定しない場合)	
ALTER TABLE/INDEX RENAME SUBPARTITION	
ALTER TABLE MODIFY PARTITION ADD SUBPARTITION	表の問合せ
ALTER TABLE MODIFY PARTITION COALESCE SUBPARTITION	DML (この文の影響を受けるサブパーティションの内容を変更しない場合のみ) 他のパーティションとそのサブパーティションのメンテナンス操作 そのパーティションの他のサブパーティションのメンテナンス操作 同じパーティションの他のサブパーティションまたは表の他のパーティションのサブパーティションに対応する、ローカル索引のサブパーティションに対する ALTER INDEX REBUILD SUBPARTITION
ALTER TABLE EXCHANGE SUBPARTITION WITH VALIDATION	表の問合せ
ALTER TABLE/INDEX MODIFY SUBPARTITION ALLOCATE EXTENT	DML (この文で参照するサブパーティションの内容を変更しない場合のみ)
ALTER TABLE MOVE SUBPARTITION	他のパーティションとそのサブパーティションのメンテナンス操作
LOAD SUBPARTITION	そのパーティションの他のサブパーティションのメンテナンス操作 同じパーティションの他のサブパーティションまたは表の他のパーティションのサブパーティションに対応する、ローカル索引のサブパーティションに対する ALTER INDEX REBUILD SUBPARTITION

表 11-3 サブパーティションに対する同時操作（続き）

メンテナンス操作	同時に実行できる操作
IMPORT SUBPARTITION	<p>表の問合せ</p> <p>表の DML</p> <p>他のパーティションとそのサブパーティションのメンテナンス操作</p> <p>そのパーティションの他のサブパーティションのメンテナンス操作</p> <p>同じパーティションの他のサブパーティションまたは表の他のパーティションのサブパーティションに対応する、ローカル索引のサブパーティションに対する ALTER INDEX REBUILD SUBPARTITION</p>
EXPORT PARTITION	<p>表とそこで定義されている索引、パーティションおよびサブパーティションに対する任意の操作</p>
ALTER (local) INDEX REBUILD SUBPARTITION	<p>表の問合せ</p> <p>DML（再作成する索引サブパーティションに対応するサブパーティションの内容を変更しない場合のみ）</p> <p>サブパーティションを再作成するパーティションを除き、索引のパーティションのサブパーティションのメンテナンス操作</p> <p>索引パーティションの他のサブパーティションのメンテナンス操作</p> <p>基礎となる表の新規索引の CREATE</p> <p>基礎となる表の既存の索引、そのパーティションおよびサブパーティション（該当する場合）のメンテナンス操作</p> <p>サブパーティションを再作成する索引パーティションに対応するパーティションを除き、基礎となる表のパーティションのメンテナンス操作</p> <p>サブパーティションを再作成する索引パーティションに対応するサブパーティションを除き、基礎となる表のサブパーティションのメンテナンス操作</p>

LOB 列を持つ表のパーティション・メンテナンス操作

表のパーティション・メンテナンス操作は、LOB 列を持つパーティション表を次のように処理します（11-38 ページの「[LOB 列を持つ表のパーティション化](#)」を参照）。

- ADD PARTITION: 表のすべての LOB 列について、新しい LOB データ・パーティションと LOB 索引パーティションが作成されます。新しい LOB データ・パーティションの物理属性を指定できます。
- DROP PARTITION: 表のすべての LOB 列について、削除する表パーティションに対応する LOB データ・パーティションと LOB 索引パーティションも削除されます。
- EXCHANGE PARTITION: 特定の非パーティション表をパーティション表のパーティションと交換できるかどうかを決定するためのアルゴリズムも、LOB 列を処理できます。
- IMPORT/EXPORT PARTITION: LOB 列を含む表のパーティションをインポート / エクスポートできます。
- LOAD PARTITION: データが表パーティションにロードされるだけでなく、それに対応する LOB データ・パーティションにもロードされます（LOB 索引パーティションが適切に変更されます）。
- MODIFY PARTITION: 特定の表パーティションに対応付けられている LOB データ・パーティションの属性を変更できます。LOB 索引パーティションの属性は指定できませんが、LOB データ・パーティションの属性を変更すると、それに対応付けられている LOB 索引パーティションの対応する属性を変更できます。
- MOVE PARTITION: 表のすべての LOB 列について、移動する表パーティションに対応する LOB データ・パーティションと LOB 索引パーティションも移動できます（ただし、LOB データ・パーティションが読取り専用デバイスにある場合などは、移動する必要はありません）。LOB データ・パーティションの新しい物理属性を指定できます。
- SPLIT PARTITION: 表のすべての LOB 列について、2 つの新しい LOB データ・パーティションと LOB 索引パーティションが作成されます。LOB インスタンスは、属している行のパーティション列の値に基づいて、新しいパーティション間で分割されます。新しい LOB データ・パーティションの物理属性を指定できます。
- TRUNCATE PARTITION: 表のすべての LOB 列について、切り捨てる表パーティションに対応する LOB データ・パーティションと LOB 索引パーティションも切り捨てられます。

パーティション表に LOB 列を追加しても、メンテナンス操作の同時実行モデルには影響しません（11-49 ページの「[メンテナンス操作の同時実行モデル](#)」を参照）。

問合せとパーティションのメンテナンス操作

パーティションのメンテナンス操作を開始する前、またはパーティションのメンテナンス操作でディクショナリを更新する前に実行を開始した問合せは、Consistent Read によって、問合せのスナップショット時に存在していたデータに正しくアクセスします。このような問合せは、正常終了するとスナップショット時に存在するすべての関連データを戻し、異常終了すると ORA-8103 または ORA-1410 エラーを戻します。このようなエラーが戻された場合は、アプリケーションで問合せを再発行する必要があります。

パーティション索引を使用する問合せで、INDEX UNUSABLE とマークされた索引パーティションのいくつかを使用して開始する問合せは、これらのパーティションの 1 つに実際に初めてアクセスするときエラーを戻します。これは、問合せの開始後にパーティションが UNUSABLE にされた場合でも発生します。

カーソルの無効化

新しい DDL 文の多くはパーティション・ベースですが、カーソルの無効化は表ベースのままです。つまり、DDL 文が表 T の 1 つのパーティション P にしか影響を与えず、かつカーソルがパーティション P にアクセスしない場合でも、表 T を修正する DDL 文は、表 T に依存しているすべてのカーソルを無効化するということです。

LOGGING 操作と NOLOGGING 操作

すべてのパーティション・メンテナンス操作は、LOGGING モードで実行できます。ただし、次に挙げる一部の操作は、NOLOGGING オプションをサポートします。

- パラレルの CREATE TABLE ... AS SELECT
- CREATE INDEX
- SPLIT、MOVE または REBUILD PARTITION
- ダイレクト・パスの SQL*Loader
- ダイレクト・ロード・インサート

デフォルトは LOGGING ですが、データベースが NOARCHIVELOG モードで動作している場合は、NOLOGGING がデフォルトになります。LOGGING/NOLOGGING オプションをサポートしていない DDL やユーティリティ文は、常に回復可能モード (LOGGING) で実行されます。

注意： LOGGING や NOLOGGING は操作ではなく物理オブジェクトの属性であるため、INSERT 文には指定できません。挿入操作に関連する表または索引のロギング・モードを変更する場合は、INSERT 文を発行する前に、ALTER TABLE/INDEX [NO]LOGGING を発行する必要があります。詳細は、25-5 ページの「[ロギング・モード](#)」を参照してください。

索引の管理

ローカル索引またはグローバル索引の改名、物理的な記憶領域の属性の変更またはパーティションの再作成は、いつでも実行できます。索引をパーティション化する方法を変更するときには、索引がローカルであるかグローバルであるかに応じて異なった処理が必要です。

ローカル索引

Oracle では、ローカル索引のパーティション化が、基礎になる表のパーティション化と必ず一致することが保証されます。基礎になる表が変更されると、Oracle は必要に応じて自動的に索引パーティションを作成したり削除することにより、2 つのパーティション化を一致させます。ローカル索引のパーティションは、明示的に追加、削除または分割することはできません。

ローカル索引ごとに、次のようになります。

- 基礎になる表にパーティションを追加すると、その新しい表パーティションと同じパーティション・バウンドを使用して新しい索引パーティションが自動的に作成される。
- 基礎になる表のパーティションを削除すると、対応する索引パーティションが自動的に削除される。
- 基礎になる表でパーティションを分割すると、対応する索引パーティションが自動的に分割される。この 2 つの新しい索引パーティションには、新しい表パーティションと同じパーティション・バウンドがあります。

親の表パーティションを分割した結果として作成されたローカル索引パーティションは、対応する表パーティションが空でないかぎり、UNUSABLE とマークされるので注意してください。

Oracle が（対応する表パーティションの ADD または SPLIT により）新しいローカル索引パーティションを作成する場合、次のように操作します。

- 索引パーティションに、対応する表パーティションと同じ名前を割り当てようとする。その名前を割り当てられない場合は、SYS_Pnnnn という形式の名前を生成します（11-19 ページの「[パーティション名とサブパーティション名](#)」を参照）。パーティションの名前は後で改名できます。
- ADD PARTITION については、Oracle は基になる索引のデフォルトの物理記憶領域属性を使用してセグメントを作成する。親索引に表領域（DEFAULT を除く。DEFAULT 表領域の場合、ローカル索引パーティションは、対応する実表パーティションと同じ位置に配置される）が指定されている場合、索引パーティションはその表領域内に配置されます。親索引に表領域が指定されていない場合は、新しい索引パーティションが存在する表領域が、基礎になる表の対応するパーティションの表領域になります。この属性は後で修正できます。（TABLESPACE を変更するには、再作成するしかありません）。

- SPLIT PARTITION の場合、分割する索引パーティションの属性が、分割した後の索引パーティションにも使用される。ただし、パーティション名は例外です（Oracle は、可能なときにのみそれらの表パーティション名を再使用します）。たとえば、TP という名前の表パーティションと IP という名前のローカル索引がある場合、TP を TP と TP1 に分割すると、ローカル索引パーティションの名前は IP と TP1（TP1 がすでに使用されている場合は、システムが生成する別の名前）になります。TP を TP1 と TP2 に分割すると、ローカル索引パーティションの名前は TP1 と TP2 になります。つまり、Oracle が表パーティション名を再使用するときは、ローカル索引パーティション名も再使用するということです。ただし、その他の属性は、分割する索引パーティションから継承されます。

ローカル索引パーティションを明示的に（データを対応する表パーティションにロードする前などに）削除するのではなく、表パーティションを非パーティション表に EXCHANGE し、その表の索引を削除し、ロード操作を実行してから、索引を作成し、INCLUDING INDEXES オプションを使用して表を元のパーティションに EXCHANGE できます。

グローバル・パーティション索引

グローバル索引のパーティション化を維持する作業は、DBA の責任で行います。DBA は、グローバル索引のパーティションを削除したり分割できます。ただし、グローバル索引の最高位のパーティションのパーティション・バウンドは必ず MAXVALUE に設定されているため、グローバル索引にパーティションを追加することはできません。グローバル索引の管理の詳細は、11-33 ページの「[グローバル・パーティション索引の管理](#)」を参照してください。

索引パーティションの再作成

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の 1 つのパーティションを再作成するには、ALTER INDEX REBUILD PARTITION 文を使用できます。これにより、DROP INDEX を実行してから CREATE INDEX を実行する（この操作は索引内のすべてのパーティションに影響を与える）という必要がなくなります。

ALTER INDEX REBUILD PARTITION には、4 つの重要な役割があります。

- 索引パーティションを再クラスタ化して、領域を回復し、パフォーマンスを改善する。
- 索引パーティションが存在するボリューム上でメディア障害が発生した場合、または索引パーティションを含むセグメントがソフトウェアによって破損された場合に、索引パーティションを修復する。
- Import または SQL*Loader を使用して基礎になる表パーティションをロードした後、ローカル索引パーティションを再生成する。これらのユーティリティは、索引のメンテナンスを回避するためのパフォーマンス・オプションを提供し、関係するパーティションを INDEX UNUSABLE としてマークし、後で DBA が再作成できるようにします。（INDEX UNUSABLE は、次の項で説明します。）つまり、「索引を削除してから索引を再作成する」という方針は、「索引パーティションを UNUSABLE としてマークしてから索引パーティションを再作成する」という方針に変更できます。

- 基礎になる表に対するパーティション・メンテナンス操作を使用して UNUSABLE とマークされた索引パーティションを再作成する。

INDEX UNUSABLE 属性

一部のメンテナンス操作は、索引を *INDEX UNUSABLE (IU)* とマークします。INDEX UNUSABLE (索引使用禁止状態) は、非パーティション索引の属性であり、パーティション索引内のパーティションの属性でもあります。索引または索引パーティションが IU とマークされている場合、その索引 (またはパーティション) を必要とする SELECT 文または DML 文を実行すると、エラーになります。

1 つの索引パーティションが IU とマークされている場合は、そのパーティションを使用する前に、そのパーティションを再作成して再び有効にする必要があります。しかし、1 つのパーティションが IU とマークされても、索引のその他のパーティションは有効であり、IU とマークされたパーティションにアクセスしない限り、その索引を必要とする SELECT 文または DML 文を実行できます。

IU パーティションを再作成する前に IU パーティションを分割または改名したり、GLOBAL 索引の IU パーティションを削除したりできます。

非パーティション索引に IU マークが付いていれば、その索引は削除できます。また、GLOBAL index.x の IU パーティションを削除して再作成したり、ALTER INDEX REBUILD を使用して非パーティション索引を再構築することもできます。

次の 6 種類のメンテナンス操作により、索引パーティションが INDEX UNUSABLE とマークされます。どの場合にも、操作が完了した後、その索引パーティションを再作成する必要があります。

- ローカル索引のメンテナンスをバイパスするオプションがある、パーティションのインポートまたは従来型バスの SQL*Loader などの操作。インポートが完了すると、関係するローカル索引パーティションは IU とマークされます。
- ダイレクト・バスの SQL*Loader が索引指定するデータの索引が期限切れの場合、関係するローカル索引パーティションとグローバル索引は IU 状態のままになる。(索引使用禁止状態 (INDEX UNUSABLE) は、以前はダイレクト・ロード状態と呼ばれていました。) 索引は次の 2 つの理由により期限切れになることがあります。
 - 領域管理でエラーが発生したため (たとえば、エクステント不足) 、ロード操作で索引をメンテナンスできない。
 - ユーザーが SKIP_INDEX_MAINTENANCE オプションを要求した。
- ROWID を変更する、ALTER TABLE MOVE PARTITION などのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティションとすべてのグローバル索引パーティションを IU とマークします。
- 表から行を削除する ALTER TABLE TRUNCATE PARTITION または DROP PARTITION などのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティションとすべてのグローバル索引パーティションを IU とマークします。

- ローカル索引のパーティション定義を修正するが、新しい定義に一致するように索引データを自動的に再作成しない、ALTER TABLE SPLIT PARTITION などのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティション（複数のこともある）を IU とマークします。（ALTER TABLE SPLIT PARTITION は、ROWID に変更を加えるため、すべてのグローバル索引パーティションも IU とマークします。）
- 索引のパーティション化の定義を修正するが、関係するパーティションを自動的に再作成しない、ALTER INDEX SPLIT PARTITION などの索引メンテナンス操作。この種の操作は、関係する索引パーティション（複数のこともある）を IU とマークします。ただし、グローバル索引の USABLE パーティションを分割すると、その結果として作成されるパーティションも USABLE になります。IU とマークされたパーティションを分割すると、その結果として作成されるパーティションも IU になります。（IU であるか、空でないグローバル索引のパーティションを削除すると、索引の次のパーティションは IU になるので注意してください。）

パーティション表およびパーティション索引についての権限

パーティションについての権限は、個々のパーティションごとではなく、親表または親索引に対して付与されます。パーティション単位で表へのアクセス権を付与する場合は、表のパーティションのビューを定義し、そのビューに対して権限を付与できます（11-62 ページの「[表としてのパーティションまたはサブパーティションの表示](#)」を参照）。

ユーザーまたはロールが、非パーティション表や非パーティション索引に対して Oracle の操作を実行するのに必要な権限（必要なリソース権限を含む）を持っている場合、その同じ Oracle の操作はパーティション表やパーティション索引に対しても実行できます。たとえば、次のとおりです。

- 非パーティション表を作成できる場合は、パーティション表も作成できる。
- 非パーティション索引を削除できる場合は、パーティション索引も削除できる。
- ALTER を使用して非パーティション表に列を追加できる場合は、ALTER を使用してパーティション表にも列を追加できる。

ユーザーまたはロールが、表または索引に対して ALTER 操作を実行するための権限を持っている場合は、いくつかの例外を除き、表または索引のパーティションに対して ALTER 操作を起動できます。

追加情報： ALTER TABLE コマンドと ALTER INDEX コマンドに関する権限の詳細は、『Oracle8i SQL リファレンス』を参照してください。

パーティション表およびパーティション索引についての監査

すべての ALL ALTER TABLE PARTITION 操作は、ALTER TABLE 操作と同様に監査されます。パーティションに対しては、新しい監査属性は使用されません。

拡張パーティション表名と拡張サブパーティション表名

一括操作を、パーティション・レベルまたはサブパーティション・レベルで実行できます。つまり、一括操作を特定のパーティションまたはサブパーティションの行のみに限定できます。たとえば、すべてのグローバル索引を UNUSABLE にしないでパーティションを削除する場合は、そのパーティションからのみすべての行を削除できます。

この種の操作は、拡張パーティション表名と拡張サブパーティション表名のための構文を提供する SQL 拡張要素によって、きわめて自然に表現されます。それと同じ操作を WHERE 句述語で表現しようとする、特にレンジ・パーティション化キーが複数の列を使用しているときなどには、非常に厄介になります。

PARTITION と SUBPARTITION の仕様

次の DML 文の表仕様構文には、パーティション表に関するオプションの PARTITION 仕様や、コンポジット・パーティション表に関するオプションの PARTITION または SUBPARTITION 仕様を含めることができます。

- SELECT
- INSERT
- UPDATE
- DELETE
- LOCK TABLE

たとえば、次のとおりです。

```
SELECT * FROM schema.table PARTITION part_name;
```

コンポジット・パーティション表の場合は、PARTITION 仕様を使用して、操作の対象を、指定したパーティションのすべてのサブパーティションに含まれるデータに限定します。

追加情報： DML 文の構文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

表としてのパーティションまたはサブパーティションの表示

表仕様の PARTITION または SUBPARTITION 構文を使用すると、個々のパーティションまたはサブパーティションを表として容易に表示できます。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名を使用すると、1 つのパーティションまたはサブパーティションからのみ選択するビューを作成し、表のかわりに使用できます。たとえば、次のとおりです。

```
CREATE VIEW sales_feb98_v1 AS
  SELECT * FROM sales SUBPARTITION (feb98_s1);

SELECT * FROM sales_feb98_v1;
```

そのようなビューを使用すると、このビューについての権限を他のユーザーやロールに付与したり取り消すことにより、パーティション・レベルまたはサブパーティション・レベルのアクセス制御メカニズムを作成することもできます。

注意： アプリケーションの移植性と ANSI 構文への準拠を考慮して、常にビューを使用し、Oracle のこの独自の拡張機能からアプリケーションを隔離する必要があります。

拡張パーティション表名と拡張サブパーティション表名の使用

この項では、表仕様に PARTITION および SUBPARTITION オプションを使用する場合の制限について説明し、この 2 つのオプションを含む SQL 文の例を示します。

拡張パーティション表名と拡張サブパーティション表名に関する制限

拡張パーティション表名と拡張サブパーティション表名の使用には、次のような制限があります。

1. 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は、リモート・スキーマ・オブジェクトを参照できません。

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・リンクや、変換するとデータベース・リンク付きの表になるシノニムを含めることはできません。リモート・パーティションまたはサブパーティションを使用する必要がある場合は、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の構文を使用するビューをリモート・サイトで作成し、そのリモート・ビューを参照できます。

2. 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の構文は、PL/SQL ではサポートされていません。

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構文を使用した SQL 文は、DBMS_SQL パッケージの動的 SQL では使用できますが、PL/SQL ブロックでは使用できません。また、PL/SQL ブロック内のパーティションまたはサブパーティションを参照する必要がある場合は、かわりに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構文を使用するビューを使用できます。

3. 実表しか使用できません。

パーティションまたはサブパーティションの拡張は、実表に指定する必要があります。シノニム、ビュー、その他のスキーマ・オブジェクトには指定できません。

PARTITION 仕様の使用例

次の文には、有効な拡張パーティション表名が含まれています。

```
SELECT * FROM sales PARTITION (nov97) s
  WHERE s.amount_of_sale > 1000;

UPDATE sales PARTITION (feb98) s
  SET s.account_name = UPPER(s.account_name);

DELETE FROM sales PARTITION (nov97)
  WHERE amount_of_sale != 0;

INSERT INTO sales PARTITION (oct97)
  SELECT * FROM latest_data;

INSERT INTO sales PARTITION (oct97)
  VALUES (...);

INSERT INTO sales PARTITION (oct97)
  (acct_no, ..., week_no)
  VALUES (...);

LOCK TABLE sales PARTITION (jun98) IN EXCLUSIVE MODE;

CREATE VIEW sales_feb98 AS
  SELECT * FROM sales PARTITION (feb98);
```

SUBPARTITION 仕様の使用例

次の文には、有効な拡張サブパーティション表名が含まれています。

```
SELECT * FROM sales SUBPARTITION (nov97_s1) s
  WHERE s.amount_of_sale > 1000;

UPDATE sales SUBPARTITION (feb98_s4) s
  SET s.account_name = UPPER(s.account_name);

DELETE FROM sales SUBPARTITION (nov97_s3)
  WHERE amount_of_sale != 0;

INSERT INTO sales SUBPARTITION (oct97_s5)
  SELECT * FROM latest_data;

INSERT INTO sales SUBPARTITION (oct97_s2)
  VALUES (...);

INSERT INTO sales SUBPARTITION (oct97_s4)
  (acct_no, ..., week_no)
```

```
VALUES (...);

LOCK TABLE sales SUBPARTITION (jun98_s1) IN EXCLUSIVE MODE;

CREATE VIEW sales_feb98_1 AS
  SELECT * FROM sales SUBPARTITION (feb98_s1);
```

ビルトイン・データ型

I am the voice of today, the herald of tomorrow.... I am the leaden army that conquers the world—I am TYPE.

Frederic William Goudy: *The Type Speaks*

この章では、Oracle ビルトイン・データ型とその特性、および Oracle データ型を Oracle 以外のデータ型にマップする方法について説明します。この章の内容は、次のとおりです。

- [Oracle データ型の概要](#)
- [文字データ型](#)
- [NUMBER データ型](#)
- [DATE データ型](#)
- [LOB データ型](#)
- [RAW および LONG RAW データ型](#)
- [ROWID および UROWID データ型](#)
- [ANSI データ型、DB2 データ型および SQL/DS データ型](#)
- [データ変換](#)

Oracle データ型の概要

SQL 文中の列値と定数は、それぞれ特定の記憶形式、制約および値の有効範囲に対応付けられた「データ型」を持っています。表の作成時には、その各列のデータ型を指定する必要があります。

Oracle のビルトイン・データ型は、次のとおりです。

- [文字データ型](#)
 - [CHAR データ型](#)
 - [VARCHAR2 および VARCHAR データ型](#)
 - [NCHAR および NVARCHAR2 データ型](#)
 - [LONG データ型](#)
- [NUMBER データ型](#)
- [DATE データ型](#)
- [LOB データ型](#)
 - [BLOB データ型](#)
 - [CLOB および NCLOB データ型](#)
 - [BFILE データ型](#)
- [RAW および LONG RAW データ型](#)
- [ROWID および UROWID データ型](#)
 - [物理 ROWID](#)
 - [論理 ROWID](#)
 - [Oracle 以外のデータベースの ROWID](#)

追加情報： PL/SQL には、その他のデータ型として、BOOLEAN、参照型、複合型（コレクションとレコード）およびユーザー定義サブタイプがあります。PL/SQL のデータ型の詳細は、『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。

表 12-1 は、各 Oracle データ型の特徴をまとめたものです。

表 12-1 Oracle ビルトイン・データ型の要約

データ型	説明	列の長さデフォルト
CHAR (<i>size</i>)	長さが <i>size</i> バイトの固定長文字データ。	表のどの行でも固定（後続空白あり）。最大サイズは行当たり 2000 バイト、デフォルト・サイズは行当たり 1 バイトです。 <i>size</i> を設定する前に、キャラクタ・セットについて検討してください（1 バイトまたはマルチバイトのどちらか）。
VARCHAR2 (<i>size</i>)	可変長文字データ。最大サイズ <i>size</i> を指定する必要があります。	行ごとに可変。最大サイズは行当たり 4000 バイトです。 <i>size</i> を設定する前に、キャラクタ・セットについて検討してください（1 バイトまたはマルチバイトのどちらか）。
NCHAR(<i>size</i>)	固定長文字データ。長さは、各国文字キャラクタ・セットに応じて <i>size</i> 文字または <i>size</i> です。	表のどの行でも固定（後続空白あり）。列の「サイズ」は、文字数（固定幅の各国文字キャラクタ・セットの場合）またはバイト数（可変幅の各国文字キャラクタ・セットの場合）のいずれかで指定します。最大「サイズ」は、1 つの文字を格納するのに必要なバイト数によって決まります。行当たりの上限は 2000 バイトです。デフォルトは 1 文字または 1 バイト（キャラクタ・セットによって変わる）です。
NVARCHAR2 (<i>size</i>)	各国文字キャラクタ・セットに応じて、長さが <i>size</i> 文字またはバイトの可変長文字データ。最大サイズ <i>size</i> を指定する必要があります。	行ごとに可変。列の「サイズ」は、文字数（固定幅の各国文字キャラクタ・セットの場合）またはバイト数（可変幅の各国文字キャラクタ・セットの場合）のいずれかで指定します。最大「サイズ」は、1 つの文字を格納するのに必要なバイト数によって決まります。行当たりの上限は 4000 バイトです。デフォルトは 1 文字または 1 バイト（キャラクタ・セットによって変わる）です。
LONG	可変長文字データ。	表の行ごとに可変。行当たりの最大サイズは $2^{31} - 1$ バイト、つまり 2GB です。
NUMBER (<i>p</i> , <i>s</i>)	可変長数値データ。精度 <i>p</i> や位取り <i>s</i> の最大値は 38 です。	行ごとに可変。1 つの列に必要な最大領域は行当たり 21 バイト。
DATE	固定長の日時データ。範囲は西暦前 4712 年 1 月 1 日から西暦 9999 年 12 月 31 日までです。	表のそれぞれの行の固定値は 7 バイト。デフォルト形式は、NLS_DATE_FORMAT パラメータで指定されている文字列（DD-MON-YY など）です。

表 12-1 Oracle ビルトイン・データ型の要約（続き）

データ型	説明	列の長さでデフォルト
RAW (size)	可変長ロー・バイナリ・データ。最大サイズ <i>size</i> を指定する必要があります。	表の行ごとに可変。行あたりの最大サイズは 2000 バイトです。
LONG RAW	可変長ロー・バイナリ・データ。	表の行ごとに可変。行あたりの最大サイズは $2^{31} - 1$ バイト、つまり 2GB です。
BLOB	バイナリ・データ。	最大 $2^{32} - 1$ バイト、つまり 4GB。
CLOB	シングルバイト文字データ。	最大 $2^{32} - 1$ バイト、つまり 4GB。
NCLOB	シングルバイト、固定長または可変長マルチバイトの各国文字キャラクター・セット (NCHAR) データ。	最大 $2^{32} - 1$ バイト、つまり 4GB。
BFILE	外部ファイルに格納されるバイナリ・データ。	最大 $2^{32} - 1$ バイト、つまり 4GB。
ROWID	物理行アドレスを表すバイナリ・データ。	表のどの行でも 10 バイト (拡張 ROWID) または 6 バイト (制限付き ROWID) に固定。
UROWID	行アドレスのタイプを表すバイナリ・データ。物理、論理または外部のいずれか。	最大 4000 バイト (ただし、論理 ROWID の場合、主キーには 3950 バイトしか使用できません)。デフォルトは 4000 バイトです。

この後の各項では、各ビルトイン・データ型について、さらに詳しく説明します。

追加情報： ビルトイン・データ型の使用方法の詳細は、『Oracle8i 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。

文字データ型

文字データ型は、文字コード体系（通常、キャラクター・セットまたはコード・ページと呼ばれる）に対応するバイト値によって、文字（英数字）データを文字列に格納します。

データベースのキャラクター・セットは、データベースの作成時に設定され、その後は変更されません。キャラクター・セットには、7 ビット ASCII（情報交換用米国標準コード）、EBCDIC（拡張 2 進化 10 進コード）、コード・ページ 500、および日本語拡張 UNIX コードなどがあります。Oracle は、シングルバイトとマルチバイトの両方の文字コード体系をサポートしています。

追加情報： 文字データ型を選択する方法の詳細は、『Oracle8i 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。

CHAR データ型

CHAR データ型は、**固定長**の文字列を格納します。CHAR 列を含む表を作成するときは、CHAR 列の文字列の長さを 1 から 2000 までの値で指定しなければなりません（単位は文字数ではなくバイト数）。（デフォルトは 1 です。）Oracle によって、次のことが保証されます。

- 表に行を挿入したり更新するとき、CHAR 列の値の長さは固定長になる。
- 短い値を与えると、固定長に合わせて空白が埋め込まれる。
- 末尾に空白が続く長い値を与えると、固定長に合わせて値から空白が切り捨てられる。
- 値が大きすぎると、Oracle からエラーが戻される。

Oracle は、空白埋め比較方法を使用して CHAR 値を比較します。

追加情報： 空白埋め比較方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。

VARCHAR2 および VARCHAR データ型

VARCHAR2 データ型には、可変長の文字列が格納されます。VARCHAR2 列を含む表を作成するときは、VARCHAR2 列の文字列の最大長を 1 から 4000 までの値で指定します（単位は文字数ではなくバイト数）。各行の値は、可変長フィールドとして列に格納されます（値が列の最大長を超えている場合は、Oracle からエラーが戻される）。

たとえば、列 VARCHAR2 を最大サイズ 50 文字で宣言するとします。シングルバイト・キャラクタ・セットで、特定の行の VARCHAR2 列に 10 文字を入れると、その行断片の列には 50 文字ではなく 10 文字（10 バイト）だけが格納されます。

Oracle は、非空白埋め比較方法を使用して VARCHAR2 値を比較します。

追加情報： 非空白埋め比較方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。

VARCHAR データ型

現在のところ、VARCHAR データ型は、VARCHAR2 データ型と同義です。ただし、Oracle の将来版では、VARCHAR データ型には異なった比較方法によって比較される可変長の文字列が格納される予定です。したがって、考えられる動作変更を回避するために、可変長文字列の格納には常に VARCHAR2 データ型を使用してください。

文字データ型と NLS キャラクタ・セットの列の長さ

Oracle の各国語サポート (NLS) 機能は、文字データ型にさまざまなキャラクタ・セットを使用できるようにします。各国語サポートを使用すると、シングルバイトおよびマルチバイト・キャラクタ・セットを処理したり、それらのキャラクタ・セットの間で変換できます。クライアント・セッションでは、データベース・キャラクタ・セットと異なる各国文字キャラクタ・セットを使用できます。

文字データ型の列の長さを指定する場合は、文字のサイズを考慮してください。この問題は、文字データを含む列を持つ表の領域を見積るときに考慮する必要があります。

追加情報： Oracle の NLS 機能の詳細は、『Oracle8i NLS ガイド』参照してください。

NCHAR および NVARCHAR2 データ型

NCHAR データ型と NVARCHAR2 データ型は NLS 文字データを格納します。NCHAR データ型は、固定長または可変長の各国文字キャラクタ・セットに対応する固定長の文字列を格納するのに対し、NVARCHAR2 データ型は可変長の文字列を格納します。

NCHAR または NVARCHAR2 の列を含む表を作成するときは、最大サイズとして文字数 (固定長の各国文字キャラクタ・セットの場合) またはバイト数 (可変長の各国文字キャラクタ・セットの場合) のいずれかを指定します。

- NCHAR 列の最大長は 2000 バイト、またはその 2000 バイトに格納できる文字数です。
- NVARCHAR2 列の最大長は 4000 バイト、またはその 4000 バイトに格納できる文字数です。

追加情報： NCHAR および NVARCHAR2 データ型の詳細は、『Oracle8i NLS ガイド』参照してください。

LOB 文字データ型

文字データの LOB データ型は CLOB と NCLOB で、最大 4GB の文字データ (CLOB) または各国文字キャラクタ・セット・データ (NCLOB) を格納できます。これらのデータ型の説明は、12-11 ページの「[LOB データ型](#)」を参照してください。

LONG データ型

LONG と定義されている列には、最大 2GB までの情報を含む可変長の文字データを格納できます。LONG データは、異なるシステム間で移動しても適切に変換されるテキスト・データです。

LONG データ型の列は、ビュー定義のテキストを格納するために、データ・ディクショナリで使用されます。LONG 列は、SELECT リスト、UPDATE 文の SET 句および INSERT 文の VALUES 句で使用できます。

注意： LONG データ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションでは、大量の文字データを格納するには CLOB および NCLOB データ型を使用してください。

追加情報： LONG データ型には多数の制限があります。『Oracle8i 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 RAW データ型の詳細は、12-13 ページの「[RAW および LONG RAW データ型](#)」も参照してください。

NUMBER データ型

NUMBER データ型には、固定長および浮動小数点の数値が格納されます。事実上どんな大きさの数値でも格納可能です。Oracle が稼働するシステムであれば、最大 38 桁の精度で、システム間の移植性が保証されています。

NUMBER 列には、次の数値を格納できます。

- $1 \times 10^{-130} \sim 9.99..9 \times 10^{125}$ の範囲の正の数（最大有効桁数は 38）
- $-1 \times 10^{-130} \sim 9.99..99 \times 10^{125}$ の範囲の負の数（最大有効桁数は 38）
- ゼロ。
- 正と負の無限大（Oracle バージョン 5 データベースからのインポート時にのみ生成される）

数値列の場合は、次のように列を指定できます。

```
column_name NUMBER
```

また、任意で次のように「精度」（全体の桁数）と「位取り」（小数点以下の桁数）を指定することもできます。

```
column_name NUMBER (precision, scale)
```

精度を指定しないと、値は与えられたとおりに列に格納されます。位取りを指定しないと、位取りはゼロになります。

Oracle では、38 桁以下の精度で数値の移植性が保証されています。次のように、位取りのみを指定して、精度は指定しないこともできます。

```
column_name NUMBER (*, scale)
```

この場合、精度は 38 になり、指定した位取りが保持されます。

数値フィールドを指定するときは、精度と位取りを指定してください。これにより、入力時の整合性チェックがさらに強化されます。

表 12-2 に、さまざまなスケール変更係数がデータの格納にどのように影響するかを示します。

表 12-2 スケール変更係数が数値データの格納に与える影響

入力データ	指定	格納方法
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(精度を超えているため、受け入れられない)
7,456,123.89	NUMBER(7,-2)	7456100

負の位取りを指定すると、小数点の左側の指定した桁数で実際のデータが丸められます。たとえば、(7,-2) という指定は、表 12-2 に示されているとおり、Oracle によって 100 の位で丸められることを意味します。

数値の入力と出力において、標準 Oracle のデフォルト小数点文字はピリオドです。たとえば、1234.56 というようになります。(小数点文字は、数値の整数部と小数部を分離する文字です。) デフォルトの小数点文字は、パラメータ NLS_NUMERIC_CHARACTERS を使用して変更できます。また、セッションの存続中の小数点文字は、ALTER SESSION 文でも変更できます。現行のデフォルト小数点文字を使用していない数値を入力するには、TO_NUMBER 関数を使用します。

内部数値形式

Oracle は、数値データを可変長形式で格納します。それぞれの値は科学表記法で格納され、指数を格納するために 1 バイト、仮数を格納するために最大 20 バイトが使用されます。(結果の数値の精度は 38 桁に制限されます。) Oracle は、先行ゼロと後続ゼロは格納しません。たとえば、数値 412 は 4.12×10^2 という形式で格納され、指数 (2) を格納するために 1 バイト、仮数の 3 桁の有効数字 (4、1、2) を格納するために 2 バイトが使用されます。

このことを考慮に入れると、値の精度を p とすると (位取りは無関係)、数値データ NUMBER(p) の列データ・サイズは、次の式を使用して計算できます。

1 バイト (指数部)
 + FLOOR(p/2)+1 バイト (仮数部)
 + 1 バイト (有効数字が 38 桁未満の負数の場合のみ)

データのバイト数

ゼロおよび正と負の無限大 (Oracle V5 データベースからのインポート時にのみ生成される) は、固有の表現で格納されます。ゼロと負の無限大にはそれぞれ 1 バイト、正の無限大には 2 バイトが必要です。

DATE データ型

DATE データ型には、ある時点の値 (日付と時刻) が表に格納されます。DATE データ型には、年 (世紀を含む) 月、日、時、分および秒 (真夜中から数える) が格納されます。

Oracle では、ユリウス暦の西暦前 4712 年 1 月 1 日から西暦 4712 年 12 月 31 日までの日付を格納できます。BCE (書式マスクでは 'BC') が明確に指定されない限り、デフォルトとして日付は西暦として扱われます。

Oracle は、日付を格納するために固有の内部形式を使用します。日付データは、それぞれ世紀、年、月、日、時、分および秒に対応する 7 バイトの固定長フィールドに格納されます。

日付の入出力に対する標準的な Oracle のデフォルト日付書式は、DD-MON-YY です。次に例を示します。

```
'13-NOV-92'
```

このデフォルト日付書式は、インスタンスごとにパラメータ NLS_DATE_FORMAT を使用して変更できます。ユーザー・セッションの存続中には、ALTER SESSION 文でも変更できます。標準 Oracle 日付書式ではない日付を入力するには、次のように書式マスク付きの TO_DATE 関数を使用してください。

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

注意： 標準的な日付書式 DD-MON-YY を使用した場合、YY は 20 世紀における年を表します (たとえば、31-DEC-92 は 1992 年 12 月 31 日を表します)。20 世紀以外の世紀年を表すには、前述のように異なる書式マスクを使用してください。

Oracle は、時刻を HH:MI:SS の 24 時間形式で格納します。デフォルトでは、時刻部分に何も指定しない場合、日付フィールドの時刻部分は 00:00:00 A.M. (真夜中) になります。時刻だけを入力した場合、日付部分のデフォルトは現在の月の 1 日になります。日付の時刻部分を入力するには、次のように時刻部分を表す書式マスクを指定して TO_DATE 関数を使用してください。

```
INSERT INTO birthdays (bname, bday) VALUES
('ANDY', TO_DATE('13-AUG-66 12:56 A.M.', 'DD-MON-YY HH:MI A.M.'));
```

ユリウス暦の使用方法

ユリウス暦を使用すると、共通の基準からの通算日数を算定できます。(01-01-4712 (西暦前) が基準になるので、現在の日付は 2,400,000 辺りになります。) ユリウス暦は、通常は整数ではなく、小数部が日付部分になります。Oracle は簡略化された方法を使用しているので、結果は整数値になります。ユリウス暦は、さまざまな計算方法と解釈があります。Oracle で使用している計算方法では、7 桁の数値 (最も一般的に使用される日付の場合) が生成されます。たとえば、08-APR-93 は 2449086 になります。

注意： Oracle のユリウス暦日付は、他の日付アルゴリズムで生成されるユリウス暦日付と互換性がない場合があります。

日付データをユリウス暦に変換するために、日付関数 (TO_DATE または TO_CHAR) に書式マスク「J」を指定できます。たとえば、次の問合せはユリウス暦の日付書式ですべての日付を戻します。

```
SELECT TO_CHAR (hiredate, 'J') FROM emp;
```

ユリウス暦を計算で使用する場合は、TO_NUMBER 関数を使用する必要があります。ユリウス暦を入力する場合は、TO_DATE 関数を使用できます。

```
INSERT INTO emp (hiredate) VALUES (TO_DATE(2448921, 'J'));
```

日付算術

Oracle 日付算術では、過去採用されてきたさまざまな暦の変則を考慮しています。たとえば、ユリウス暦からグレゴリウス暦への切替え (15-10-1582) では、その直前の 10 日間 (05-10-1582 から 14-10-1582 まで) が排除されています。0 年は存在しません。

存在しない日付でもデータベースに入力することができます。ただし、その日付は、日付算術では無視され、次の「実在する」日付として処理されます。たとえば、04-10-1582 の次の日は 15-10-1582 で、05-10-1582 の次の日も 15-10-1582 になります。

注意： 日付算術に関するこの説明は、すべての国 (アジア圏など) の日付規格に適用できるとは限りません。

世紀と西暦 2000 年

Oracle は、データを世紀情報と一緒に格納します。たとえば、Oracle データベースには、単に 96 または 01 ではなく、1996 または 2001 が格納されます。DATE データ型は常に 4 桁の年を内部に格納し、データベース内部に格納される他のすべての日付も 4 桁の年となります。インポート、エクスポートおよび回復などの Oracle ユーティリティでも、年を 4 桁で正しく処理しています。

ただし、年を仮定して（年は必ず 19xx のはず、など）作成されているアプリケーションもあります。このため、アプリケーション・プログラマは西暦 2000 年についてのコードを再検討し、テストする必要があります。

追加情報： 世紀と日付の書式マスクの詳細は、『Oracle8i 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。日付書式コードの一般情報は、『Oracle8i SQL リファレンス』を参照してください。

LOB データ型

LOB データ型 BLOB、CLOB、NCLOB および BFILE は、非構造化データ（テキスト、グラフィック・イメージ、ビデオ・クリップおよびサウンド・ウェーブなど）の大きなブロックを格納します。最大サイズは 4GB です。このデータ型を使用すれば、個々のデータに対する効率的なランダム・アクセスが可能です。

LOB 列に対してパラレル問合せ（パラレル DML または DDL ではなく）を実行できます。

LOB データ型は、いくつかの点で LONG および LONG RAW データ型と異なります。たとえば、次のとおりです。

- 表は複数の LOB 列を含むことができるが、LONG 列は 1 つしか含むことができない。
- 1 つ以上の LOB 列を含む表はパーティション化できるが、LONG 列を含む表はパーティション化できない。
- LOB の最大サイズは 4GB であるが、LONG の最大サイズは 2GB である。
- LOB はデータへのランダム・アクセスをサポートしているが、LONG は順次アクセスしかサポートしていない。
- LOB データ型（NCLOB を除く）をユーザー定義オブジェクト型の属性として指定できるが、LONG データ型には指定できない。
- ローカル変数のように機能する一時 LOB を使用して、LOB データの変換を実行できる。一時内部 LOB（BLOB、CLOB および NCLOB）は、ユーザーの一時表領域に作成され、表には依存しません。ただし、LONG データ型の場合、一時構造は使用できません。

追加情報： LOB データ型と LONG および LONG RAW データ型の相違点の詳細リストは、『Oracle8i SQL リファレンス』を参照してください。

SQL 文は、表に LOB 列を定義し、ユーザー定義オブジェクト型に LOB 属性を定義します。LOB 属性 LOGGING および NOLOGGING の詳細は、25-7 ページの「[デフォルトのロギング・モード](#)」を参照してください。表内に LOB を定義するときは、各 LOB について表領域と記憶特性を明示的に指定できます。

LOB データ型は、インライン（表の中）、ライン外（表領域の中。LOB ロケータを使用する）または外部ファイル（BFILE データ型）のいずれかに格納できます。

追加情報： LOB 記憶域と LOB ロケーションの詳細は、『Oracle8i 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。

BLOB データ型

BLOB データ型は、構造化されていないバイナリ・データをデータベースに格納します。格納できるバイナリ・データの最大サイズは、4GB です。

BLOB はトランザクションに全面的に参加します。DBMS_LOB パッケージ、PL/SQL または OCI によって BLOB 値に加えられる変更は、コミットまたはロールバックが可能です。ただし、BLOB ロケータは、複数のトランザクションまたはセッションにまたがることはできません。

CLOB および NCLOB データ型

CLOB および NCLOB データ型は、最大 4GB の文字データをデータベースに格納します。CLOB はシングルバイト・キャラクタ・セットのデータを格納し、NCLOB は固定幅および可変幅のマルチバイト各国文字キャラクタ・セットのデータ（NCHAR データ）を格納します。

CLOB と NCLOB はトランザクションに全面的に参加します。DBMS_LOB パッケージ、PL/SQL または OCI によって CLOB または NCLOB 値に加えられる変更は、コミットまたはロールバックが可能です。ただし、CLOB および NCLOB ロケータは複数のトランザクションやセッションにまたがることはできません。

CLOB および NCLOB 値は、固定幅の 2 バイトの Unicode キャラクタ・セットを使用して、データベースに格納されます。Oracle は、格納された Unicode 値をクライアントまたはサーバーに必要なキャラクタ・セットに変換します。このキャラクタ・セットは、固定幅の場合と可変幅の場合があります。可変幅キャラクタ・セットを使用して CLOB 列または NCLOB 列にデータを挿入すると、Oracle はそのデータを Unicode に変換してからデータベースに格納します。

NCLOB の属性を持ったオブジェクト型は作成できませんが、オブジェクト型のメソッドで NCLOB パラメータを指定することはできます。

追加情報： 各国文字キャラクタ・セットと Unicode キャラクタ・セットの詳細は、『Oracle8i NLS ガイド』を参照してください。

BFILE データ型

BFILE データ型は、構造化されていないバイナリ・データを、データベースの外にあるオペレーティング・システム・ファイルに格納します。BFILE 列または BFILE 属性は、データが含まれる外部ファイルを参照するファイル・ロケータを格納します。格納できるデータの最大サイズは、4GB です。

BFILE は読み取り専用のため、変更はできません。サポートしているのはランダム読み取りのみで（順次読み込みはサポートしていない）、トランザクションには参加しません。BFILE のファイルの整合性と耐久性を管理するのは、基礎になるオペレーティング・システムです。データベース管理者は、参照するファイルが存在していること、またそのファイルに対するオペレーティング・システムの読み取り許可が Oracle プロセスに与えられていることを確認する必要があります。

RAW および LONG RAW データ型

注意： RAW および LONG RAW データ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションの場合は、大量のバイナリ・データを格納するには BLOB および BFILE データ型を使用してください。

RAW データ型と LONG RAW データ型は、Oracle が解釈しない（異なるシステム間での移動時には変換されない）データに使用します。これらのデータ型は、バイナリ・データやバイト文字列用に用意されています。たとえば、図形データ、音声データ、文書およびバイナリ・データの配列を格納するために LONG RAW を使用できます。解釈は、用途によって異なります。

RAW は、VARCHAR2 文字データ型と同じく可変長のデータ型です。ただし、RAW または LONG RAW データの送信時に、Net8（ユーザー・セッションをインスタンスに接続）と Import/Export ユーティリティでは文字変換は実行されません。それとは対照的に、Net8 と Import/Export ユーティリティは、データベースのキャラクタ・セットとユーザー・セッションのキャラクタ・セットが異なる場合に、これらのキャラクタ・セット間（ALTER SESSION コマンドの NLS_LANGUAGE パラメータで設定）で CHAR、VARCHAR2 および LONG データを自動変換します。

Oracle が RAW または LONG RAW データと CHAR データとの間の自動変換を実行する場合、バイナリ・データは 16 進数で表され、これらの 16 進文字は 1 文字で RAW データ 4 ビット分のデータを表します。たとえば、1 バイトの RAW データのビットが 11001011 の場合、これは「CB」として表示または入力されます。

LONG RAW データに索引を付けることはできませんが、RAW データには索引を付けることができます。

追加情報： LONG RAW データ型の詳細とその他の制限は、『Oracle8i 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。

ROWID および UROWID データ型

ROWID データ型には、データベース内の各行のアドレス（「ROWID」）が格納されます。

- 「物理 ROWID」は、通常の表（索引構成表を除く）、クラスタ化表、表パーティションとサブパーティション、索引および索引パーティションとサブパーティションに、行のアドレスを格納する。
- 「論理 ROWID」は、索引構成表に行のアドレスを格納する。

「汎用 ROWID」または UROWID という単一のデータ型は、論理 ROWID と物理 ROWID だけでなく、外部表（ゲートウェイを介してアクセスされる Oracle 以外の表など（12-20 ページの「[Oracle 以外のデータベースの ROWID](#)」を参照）の ROWID もサポートしています。

UROWID データ型の列には、あらゆる種類の ROWID を格納できます。UROWID 列を使用するには、COMPATIBLE 初期化パラメータの値を 8.1 以上に設定する必要があります。

ROWID 疑似列

Oracle データベースのそれぞれの表は、ROWID という名前の「疑似列」を内部的に持っています。SQL*Plus で SELECT * FROM ... 文または DESCRIBE ... 文を実行することによって表の構造のリストを表示しても、この疑似列は表示されません。ただし、それぞれの行のアドレスは、予約語 ROWID を列名として使用することにより、SQL 問合せで取得できます。次に例を示します。

```
SELECT ROWID, ename FROM emp;
```

疑似列 ROWID の値は、INSERT または UPDATE 文で設定することはできません。また、ROWID の値は削除できません。Oracle は、索引を組み立てるために、疑似列 ROWID 内の ROWID を内部的に使用します（12-18 ページの「[ROWID の使用方法](#)」を参照）。

疑似列 ROWID の ROWID は他の表の列と同じように参照できますが（SELECT リストや WHERE 句で使用する）、その ROWID はデータベースに格納されているわけではなく、またそのデータベースのデータでもありません。ROWID データ型の列を含む表を作成することはできますが、そのような列の ROWID 値が有効であるかどうかは保証できません。

物理 ROWID

物理 ROWID は、特定の表の 1 行への最高速のアクセスを提供します。物理 ROWID には、1 行のアドレス（特定のブロックまで）が含まれ、実際には単一のブロック・アクセスでその行を取り出すことができます。その行が存在する限り、ROWID は変化しないことが保証されます。このようなパフォーマンスと安定性により、アプリケーションが行の集合を選択し、それに対してなんらかの操作を実行し、選択した一部の行に更新などの目的で再度アクセスする場合に、ROWID が役立ちます。

クラスタ化されてない表のすべての行には、行の行断片（行が複数の行断片にわたって連鎖している場合は、最初の行断片）の物理アドレスに対応する、一意の ROWID が割り当てられています。クラスタ化された表の場合、同じデータ・ブロックにある異なる表の行の ROWID が同一の場合もあります。

行に割り当てられている ROWID は、その行がエクスポートおよびインポート（Import および Export ユーティリティを使用）されない限り変更されません。表から行を削除する（および、その操作を含むトランザクションがコミットされる）と、削除された行に対応していた ROWID は、後続のトランザクションで挿入される行に割り当てられることがあります。

物理 ROWID のデータ型は、次の 2 つの書式のどちらかです。

- 「拡張 ROWID」形式は、表領域関連のデータ・ブロックのアドレスをサポートし、パーティション表と索引内の行や、パーティション化されていない索引の行を効率よく識別する。Oracle8i サーバーによって作成される表と索引は、常に拡張 ROWID を持ちます。
- Oracle7 以前のリリースで開発されたアプリケーションとの下位互換性を保つために、「制限付き ROWID」形式も使用できる。

拡張 ROWID

拡張 ROWID は、それぞれの選択された行の物理アドレスを基本 64 コード化で表現したものです。コード化文字は、A ~ Z、a ~ z、0 ~ 9、+ および / です。たとえば、次の問合せでは、

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

次のような行情報が戻されます。

```
ROWID          ENAME
-----
AAAAaoAATAAABrXAAA BORTINS
AAAAaoAATAAABrXAAE RUGGLES
AAAAaoAATAAABrXAAG CHEN
AAAAaoAATAAABrXAAN BLUMBERG
```

拡張 ROWID の書式は、OOOOOOFFBBBBBBRRR という 4 つの部分からなっています。

- OOOOOO: **データ・オブジェクト番号**。データベース・セグメント（例では AAAAao）を識別します。同じセグメント内のスキーマ・オブジェクト（表のクラスタなど）は、同じデータ・オブジェクト番号を持っています。
- FFF: 行を含むデータ・ファイルの表領域関連の**データ・ファイル番号**（例ではファイル AAT）。
- BBBBBB: その行を含む**データ・ブロック**（例ではブロック AAABrX）。ブロック番号は、表領域ではなくデータ・ファイルに対する相対値です。このため、同じブロック番号の行が、同じ表領域の 2 つの異なるデータ・ファイルに存在することもあります。

- RRR: ブロック内の行。

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 USER_OBJECTS、DBA_OBJECTS および ALL_OBJECTS から取得できます。たとえば、次の問合せは、スキーマ SCOTT の EMP 表のデータ・オブジェクト番号を戻します。

```
SELECT DATA_OBJECT_ID FROM DBA_OBJECTS
      WHERE OWNER = 'SCOTT' AND OBJECT_NAME = 'EMP';
```

DBMS_ROWID パッケージを使用して、拡張 ROWID から情報を抽出したり、ROWID を拡張形式から制限付き形式に変換（またはその逆も）できます。

追加情報： DBMS_ROWID パッケージの詳細は、『Oracle8i 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。

制限付き ROWID

制限付き ROWID は、それぞれの選択された行の物理アドレスをバイナリで表現したものです。SQL*Plus を使用して問合せを実行すると、このバイナリ表現が VARCHAR2/16 進数表現に変換されます。たとえば、次の問合せでは、

```
SELECT ROWID, ename FROM emp
      WHERE deptno = 30;
```

次のような行情報が戻されます。

ROWID	ENAME
00000DD5.0000.0001	KRISHNAN
00000DD5.0001.0001	ARBUCKLE
00000DD5.0002.0001	NGUYEN

前述のように、制限付き ROWID の VARCHAR2/16 進数表現は、*block.row.file* という 3 つの部分からなっています。

- その行を含む**データ・ブロック**（例ではブロック DD5）。ブロック番号は、表領域ではなくデータ・ファイルに対する相対値です。このため、同じブロック番号の行が、同じ表領域の 2 つの異なるデータ・ファイルに存在することもあります。
- その行を含むブロックの中の**行**（例では行 0、1、2）。各ブロックの行番号は、必ず 0 で始まります。
- その行を含む**データ・ファイル**（例ではファイル 1）。すべてのデータベースの最初のデータ・ファイルは必ずファイル 1 であり、ファイル番号はデータベース内で一意です。

ROWID の使用例

関数 SUBSTR を使用して、ROWID のデータをコンポーネントに分割できます。たとえば、SUBSTR を使用して拡張 ROWID を 4 つのコンポーネント（データベース・オブジェクト、ファイル、ブロックおよび行）に分割します。

```
SELECT ROWID,
       SUBSTR(ROWID,1,6) "OBJECT",
       SUBSTR(ROWID,7,3) "FIL",
       SUBSTR(ROWID,10,6) "BLOCK",
       SUBSTR(ROWID,16,3) "ROW"
FROM products;
```

ROWID	OBJECT	FIL	BLOCK	ROW
-----	-----	---	-----	----
AAAA8mAALAAAAQkAAA	AAAA8m	AAL	AAAAQk	AAA
AAAA8mAALAAAAQkAAF	AAAA8m	AAL	AAAAQk	AAF
AAAA8mAALAAAAQkAAI	AAAA8m	AAL	AAAAQk	AAI

または、SUBSTR を使用して制限付き ROWID を 3 つのコンポーネント（ブロック、行およびファイル）に分割します。

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE",
       SUBSTR(ROWID,1,8) "BLOCK",
       SUBSTR(ROWID,10,4) "ROW"
FROM products;
```

ROWID	FILE	BLOCK	ROW
-----	----	-----	----
00000DD5.0000.0001	0001	00000DD5	0000
00000DD5.0001.0001	0001	00000DD5	0001
00000DD5.0002.0001	0001	00000DD5	0002

ROWID は、表データの物理的な格納に関する情報を明らかにするのに便利です。たとえば、表の行の物理的な位置を知りたい場合（表のストライプ化の場合など）、次のように拡張 ROWID を問い合わせることによって、指定の表の行がいくつかのデータ・ファイルに含まれているかがわかります。

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID,7,3))) "FILES" FROM tablename;

FILES
-----
2
```

追加情報： ROWID の詳細な使用例は、『Oracle8i SQL リファレンス』、『PL/SQL ユーザーズ・ガイドおよびリファレンス』、『Oracle8i チューニング』および Oracle Tools とユーティリティに関する他のマニュアルを参照してください。

ROWID の使用方法

Oracle は、ROWID を内部的に使用して索引を構築します。索引のそれぞれのキーは、高速アクセスのために、対応する行のアドレスを指す ROWID に結び付けられています。エンド・ユーザーとアプリケーション開発者は、次のような重要な用途に ROWID を使用することもできます。

- ROWID は特定の行にアクセスする最も高速な方法である。
- ROWID は表の編成を把握するために使用できる。
- ROWID は表の中の行の一意識別子である。

DML 文の中で ROWID を使用する前に、ROWID が変更されないことを検証して確認する必要があります。目的の行を、削除されないようにロックしてください。無効な ROWID を使用してデータを要求すると、状況によっては、文が失敗します。

また、ROWID データ型を使用して定義した列を持つ表を作成することもできます。たとえば、データ型 ROWID の列を持つ例外表を定義して、整合性制約に違反するデータベース行の ROWID を格納できます。ROWID データ型を使用して定義されている列の動作は、表の他の列と類似しています。たとえば、値を更新できます。データ型 ROWID として定義されている列のそれぞれの値を、適切な列データとして格納するには 6 バイト必要です。

論理 ROWID

索引構成表の中の行は、永続的な物理アドレスを持たず、索引リーフに格納されます。挿入の結果として同一ブロック内で、または別のブロックに移動できます。したがって、物理アドレスに基づく ROWID は使用できません。かわりに、Oracle は論理行識別子を持つ索引構成表を提供します。この識別子は、表の主キーに基づいており、「論理 ROWID」と呼ばれます。Oracle は、これらの論理 ROWID を使用して、索引構成表の 2 次索引を構築します。

2 次索引に使用される論理 ROWID には、「物理推測」を含めることができます。これは、推測時（つまり、2 次索引の作成時または再構築時）に、索引構成表の中の行のブロック位置を識別するものです。

Oracle は、推測を使用して、全キー検索を回避し、リーフ・ブロックを直接ブロープできます。これにより、不揮発性の索引構成表の ROWID アクセスで、通常の表の物理 ROWID アクセスに匹敵するパフォーマンスを得られることが保証されます。ただし、揮発性の表では、推測が陳腐化するとブロープが失敗することがあります。その場合は、主キー検索を実行する必要があります。

2 つの論理 ROWID の値は、同じ主キー値および異なる推測を持つ場合に同一と見なされません。

論理 ROWID と物理 ROWID の比較

論理 ROWID と物理 ROWID の類似点は、次のとおりです。

- 論理 ROWID には、ROWID 疑似列を介してアクセスできる。
 - ROWID 疑似列を使用して、索引構成表から論理 ROWID を選択できる。SELECT ROWID 文は不透明構造を戻します。この構造は、表の主キーと行の物理推測（存在する場合）およびなんらかの制御情報で内部的に構成されています。
 - WHERE ROWID = *value* 形式の述語を使用して行にアクセスできる。この場合、*value* は SELECT ROWID から戻される不透明構造です。
- 論理 ROWID を介してアクセスするのは、特定の行へのアクセス方法としては最も高速であるが、複数のブロック・アクセスを必要とする場合がある。
- 行の論理 ROWID は、主キー値に変更がなければ変化しない。このため、行に対するすべての更新を通じて変化しない物理 ROWID に劣らず陳腐化しにくくなっています。
- 論理 ROWID は、UROWID データ型の列に格納できる（12-14 ページの「[ROWID および UROWID データ型](#)」を参照）。

物理 ROWID と論理 ROWID の違いの 1 つは、論理 ROWID を使用しても表の構成方法を表示できないことです。

論理 ROWID での推測

行の物理位置が変化した場合、論理 ROWID は推測が含まれていても引き続き有効ですが、推測は陳腐化し、その行へのアクセスが低速になることがあります。推測情報を動的に更新することはできません。ただし、索引構成表の 2 次索引に対し、索引を再構築して最新の推測を取得できます。索引構成表の 2 次索引を再構築するには、通常の表の索引を再構築する場合と違って、実表を読み込む必要があるので注意してください。

Oracle で推測が不用意に使用されないように、DBMS_STATS パッケージまたは ANALYZE コマンドで索引統計を収集し、推測の陳腐化を追跡する必要があります。特に、UROWID 列に推測を使用した ROWID を永続的に格納し、その ROWID を後から取り出して行のフェッチに使用するアプリケーションの場合は、この作業が重要になります。

DBMS_STATS パッケージまたは ANALYZE コマンドを使用して索引の統計を収集すると、既存の推測がまだ有効かどうかチェックされ、陳腐化した推測と有効な推測の比率がデータ・ディクショナリに記録されます。2 次索引を再構築（推測を再コンパイル）した後に、索引統計を収集しなおす必要があります。統計の収集方法の詳細は、22-9 ページの「[コストベース最適化の統計](#)」を参照してください。

通常、論理 ROWID に推論が付いていない場合に、揮発性の高い表へのアクセスが最も高速になります。表が静的な場合、または ROWID を取得してから使用するまでの時間が短いため行移動がない場合は、推測付きの論理 ROWID によるアクセスが最も高速です。

Oracle 以外のデータベースの ROWID

Oracle データベース・アプリケーションは、SQL*Connect または Oracle Open Gateway を使用して、Oracle 以外のデータベース・サーバーに対しても実行可能です。そのような場合、ROWID の形式は、その Oracle 以外のシステムの特性に応じて異なります。また、VARCHAR2/16 進形式への標準変換も使用できません。プログラムで ROWID データ型を使用することもできますが、最大 256 バイトまでの 16 進形式への非標準変換を使用する必要があります。

Oracle 以外のデータベースの ROWID を、UROWID データ型の列に格納できます (12-14 ページの「[ROWID および UROWID データ型](#)」を参照)。

追加情報： Oracle 以外のシステムで ROWID を処理する場合の詳細は、OCI またはプリコンパイラの関連マニュアルを参照してください。

ANSI データ型、DB2 データ型および SQL/DS データ型

[表 12-3](#) に、ANSI データ型 から Oracle データ型への変換を示します。ANSI/ISO データ型 NUMERIC、DECIMAL および DEC では、固定小数点の数値のみを指定できます。これらのデータ型の場合、s (スケール) はデフォルトの 0 に設定されます。

表 12-3 ANSI データ型から Oracle データ型への変換

ANSI SQL データ型	Oracle データ型
CHARACTER (n)、CHAR(n)	CHAR(n)
NUMERIC (p,s)、DECIMAL (p,s)、DEC (p,s)	NUMBER (p,s)
INTEGER、INT、SMALLINT	NUMBER (38)
FLOAT (p)	FLOAT (p)
REAL	FLOAT (63)
DOUBLE PRECISION	FLOAT (126)
CHARACTER VARYING(n)、CHAR VARYING(n)	VARCHAR2 (n)

IBM 製品 SQL/DS と DB2 のデータ型 TIME、TIMESTAMP、GRAPHIC、VARGRAPHIC および LONG VARGRAPHIC は、対応する Oracle データ型が存在しないので使用できません。TIME データ型と TIMESTAMP データ型は、Oracle データ型 DATE の部分構成要素です。

表 12-4 に、DB2 と SQL/DS の変換を示します。

表 12-4 SQL/DS、DB2 データ型から Oracle データ型への変換

DB2 または SQL/DS データ型	Oracle データ型
CHARACTER (n)	CHAR(n)
VARCHAR(n)	VARCHAR2 (n)
LONG VARCHAR	LONG
DECIMAL (p,s)	NUMBER (p,s)
INTEGER, SMALLINT	NUMBER (38)
FLOAT (p)	FLOAT (p)
DATE	DATE

データ変換

場合によっては、予期しているのとは異なるデータ型が与えられることがあります。そのような値は、Oracle が次の関数の 1 つを使用して、データを必要なデータ型に変換できるとき、データ変換が自動的に行われます。

- TO_NUMBER()
- TO_CHAR()
- TO_DATE()
- CHARTOROWID()
- ROWIDTOCHAR()
- HEXTORAW()
- RAWTOHEX()
- REFTOHEX()

追加情報： 暗黙的なデータ型変換の規則は、『Oracle8i 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。

ユーザー定義データ型

The Beautiful arises from the perceived harmony of an object, whether sight or sound, with the inborn and constitutive rules of the judgment and imagination:and it is always intuitive.

Samuel Taylor Coleridge, *Genial Criticism*

オブジェクト型とその他のユーザー定義データ型を使用すると、アプリケーションのデータの構造と動作をモデル化するデータ型を定義できます。

この章の内容は、次のとおりです。

- [基礎知識](#)
- [ユーザー定義データ型](#)
- [アプリケーション・インタフェース](#)

注意： この章で説明している機能は、Oracle8i Enterprise Edition を購入した場合にのみ使用できます。この章に記載されている「Oracle Server」という用語は、Oracle8i Enterprise Edition を指しています。

基礎知識

リレーショナル・データベース管理システム（RDBMS）は、ビジネス・データを管理する標準的なツールです。これを使用することにより、世界中で毎日行われている幾百万ものビジネスについての膨大なデータに、高速、効率的、かつ十分な信頼性をもってアクセスできます。

Oracle8i は、「オブジェクト・リレーショナル」データベース管理システム（ORDBMS）です。これは、ユーザーが各種のデータを追加分定義し（データの構造とその構造に対する操作の両方を指定する）、リレーショナル・モデル内でそれらのデータ型を使用できることを意味しています。このアプローチにより、データベースに格納されているデータにさらに価値が付加されます。ユーザー定義データ型によって、アプリケーション開発者は、イメージ、オーディオおよびビデオなどの複合データを容易に処理できます。オブジェクト型では、構造化されたビジネス・データが自然な形で格納され、アプリケーションもそれらのデータを自然な形で取り出すことができます。これにより、オブジェクト指向プログラミング技法を使用して開発されたアプリケーションを効率よく処理できます。

複合データ・モデル

Oracle Server では、複合ビジネス・モデルを SQL で定義し、それをデータベース・スキーマの一部にすることができます。データを管理し共有するアプリケーションに入っている必要があるのは、データ・ロジックではなく、アプリケーション・ロジックだけです。

例

たとえば、発注情報を使用して、購買、買掛管理、出荷および売掛管理の機能を編成しているとします。

1 件の発注情報には、関連する供給業者または顧客と、不特定数の品目が含まれています。さらに、アプリケーションでは、発注に関して状況情報を動的に計算する機能がしばしば必要になります。たとえば、出荷済みまたは未出荷の品目の現在数量が必要になることがあります。

アプリケーションですべての発注データのテンプレートとして使用することになる、「オブジェクト型」と呼ばれるスキーマ・オブジェクトを定義する方法は、この章の後半で扱います。オブジェクト型では、発注などの構造化されたデータ単位を形成する個々の要素を指定します。この要素のことを「属性」といいます。属性そのものが、別の構造化されたデータ単位になっていることもあります（明細項目リストなど）。オブジェクト型では、1 件の発注の合計金額を計算するなど、そのデータ単位に対して実行できる操作も指定します。この操作のことを「メソッド」といいます。

テンプレートと合致する発注情報を作成し、数値や日付とまったく同じように表の列に格納することができます。

また、発注情報を「オブジェクト表」に格納することもできます。その場合、表の各行が 1 件の発注情報に対応し、表の各列が発注情報の属性に対応します。

発注情報の構造と動作のロジックはスキーマに入っているため、アプリケーションはその詳細を知る必要はなく、最新の変更内容を反映する必要もありません。

Oracle では、オブジェクト型に関するスキーマ情報を使用して、実質的な伝送効率を向上させます。クライアント側のアプリケーションは、サーバーに発注情報を要求したとき、すべての関連データを 1 回の伝送で受信できます。その後、アプリケーションは、データが格納されている位置や実現方法の詳細を知らなくても、サーバーからのそれ以上の伝送なしで、関連データ項目間をナビゲートできます。

マルチメディア・データ型

数値、日付および文字などの基本的なデータ型の管理を最適化することにより、データベース・システムはかなりの程度まで効率が良くなります。また、値の比較、値の分散状況の判別、効率的な索引の作成およびその他の最適化を実行したりするための機能があります。

これらの基本的なデータ型に適合しない重要なビジネス・エンティティの例には、テキスト、ビデオ、サウンド、グラフィックスおよび空間データなどがあります。Oracle8i Enterprise Edition は、これらの複合データ型のモデル化と実現をサポートしています。

ユーザー定義データ型

第 12 章「ビルトイン・データ型」では、Oracle のビルトイン・データ型について説明しています。ユーザー定義データ型には、その他に次の 2 つのカテゴリがあります。

- オブジェクト型
- コレクション型

ユーザー定義データ型では、ビルトイン・データ型と他のユーザー定義データ型を、アプリケーション・データの構造と動作をモデル化するデータ型の構築ブロックとして使用します。

ユーザー定義データ型は、スキーマ・オブジェクトです。これらのオブジェクトの使用は、他のスキーマ・オブジェクトと同様の管理制御を受けます（第 14 章「ユーザー定義データ型の使用法」を参照）。

オブジェクト型

オブジェクト型は、アプリケーション・プログラムが取り扱う実社会のエンティティ（発注など）を抽象化したものです。オブジェクト型は、次のような 3 種類のコンポーネントのあるスキーマ・オブジェクトです。

- 「名前」。これは、そのスキーマ内でオブジェクト型を一意に識別するものです。
- 「属性」。実社会のエンティティの構造と状態をモデル化します。属性は、ビルトイン型か他のユーザー定義型です。

- 「メソッド」。これは、PL/SQL で作成されてデータベースに格納されたファンクションやプロシージャ、または、C などの言語で作成されて外部に格納されたファンクション（関数）やプロシージャのことです。メソッドは、実社会のエンティティに対してアプリケーションが実行できる操作をインプリメントします。

オブジェクト型はテンプレートです。テンプレートに合わせて構造化されたデータ単位を「オブジェクト」といいます。

発注情報の例

ここで説明する例では、EXTERNAL_PERSON、LINEITEM および PURCHASE_ORDER というオブジェクト型を定義する方法を示します。

オブジェクト型 EXTERNAL_PERSON および LINEITEM は、ビルトイン型の属性を持ちます。オブジェクト型 PURCHASE_ORDER の構造はもっと複雑で、実際の発注情報の構造と厳密に一致するものになっています。

PURCHASE_ORDER の属性は、ID、CONTACT および LINEITEMS です。属性 CONTACT はオブジェクトで、属性 LINEITEMS はネストした表です（13-11 ページの「[ネストした表](#)」を参照）。

```
CREATE TYPE external_person AS OBJECT (  
    name          VARCHAR2(30),  
    phone         VARCHAR2(20) );  
  
CREATE TYPE lineitem AS OBJECT (  
    item_name     VARCHAR2(30),  
    quantity      NUMBER,  
    unit_price    NUMBER(12,2) );  
  
CREATE TYPE lineitem_table AS TABLE OF lineitem;  
  
CREATE TYPE purchase_order AS OBJECT (  
    id            NUMBER,  
    contact       external_person,  
    lineitems     lineitem_table,  
  
    MEMBER FUNCTION  
    get_value     RETURN NUMBER );
```

ここに記載したコード例は簡略化してあります。メソッド GET_VALUE の本体を指定する方法は示してありません。また、実際の発注情報の複雑さも完全には反映していません。

追加情報： 発注例の詳細は、『Oracle8i 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。

オブジェクト型はテンプレートです。オブジェクト型を定義しても、記憶領域は割り当てられません。SQL 文で NUMBER や VARCHAR2 などのデータ型を使用可能な場所であれば、ほとんどの場所に LINEITEM、EXTERNAL_PERSON または PURCHASE_ORDER を使用できます。

たとえば、次のようなリレーショナル表を定義して、発注先の担当者（contact）を追跡し記録できます。

```
CREATE TABLE contacts (
  contact      external_person
  date        DATE );
```

CONTACT 表は、列の 1 つがオブジェクト型で定義されているリレーショナル表です。リレーショナル表の列を占めるオブジェクトを「列オブジェクト」といいます（13-8 ページの「[行オブジェクトと列オブジェクト](#)」を参照）。

メソッド

オブジェクト型のメソッドは、オブジェクトの動作をモデル化したもので、大きく分けるとメンバー、静的および比較という 3 つのカテゴリに分類できます。

メンバー・メソッドは、最初のパラメータとして常に暗黙の SELF パラメータを持ち、その型にオブジェクト型を含むファンクションまたはプロシージャです。この種のメソッドは、OBJECT.METHOD() の様な、「自己参照的なスタイル」で起動できます。メンバー・メソッドは、オブザーバ・メソッドやミューテータ・メソッドを記述するときに役立ちます。

静的メソッドは、暗黙の SELF パラメータを持たないファンクションまたはプロシージャです。この種のメソッドは、TYPE_NAME.METHOD() のように、メソッドを型名で修飾して起動できます。静的メソッドは、ユーザー定義のコンストラクタやキャスト・メソッドを指定するときに役立ちます。

比較メソッドは、オブジェクトのインスタンスを比較するときに使用します。

Oracle は、PL/SQL、JAVA および C で型メソッドのインプリメントの選択をサポートしています。

この例の場合、PURCHASE_ORDER には GET_VALUE というメソッドがあります。それぞれの発注情報オブジェクトには、専用の GET_VALUE メソッドがあります。たとえば、X と Y が発注情報オブジェクトを保持する PL/SQL 変数で、W と Z が数値を保持する変数である場合、次の 2 つの文による W と Z の結果値は異なる可能性があります。

```
w = x.get_value();
z = y.get_value();
```

これらの文を実行すると、W には変数 X で表される発注情報の値、Z には変数 Y で表される発注情報の値が代入されます。

X.GET_VALUE () の項により、メソッド GET_VALUE が起動されます。メソッド定義にパラメータを組み込むこともできますが、GET_VALUE メソッドにパラメータは必要ありません。

ん。起動時に結び付けられたオブジェクトの属性からすべての引数を検索できるからです。つまり、1 番目のサンプル文では発注情報 X の属性を使用して値を計算し、2 番目のサンプル文では発注情報 Y の属性を使用して値を計算します。これを「自己参照的なスタイル」のメソッドの起動といいます。

すべてのオブジェクト型には、特定のオブジェクトに結び付けられない、暗黙に定義されるメソッド、つまりオブジェクト型のコンストラクタ・メソッドも 1 つあります。

オブジェクト型のコンストラクタ・メソッド すべてのオブジェクト型には、オブジェクト型の仕様に基づいて新しいオブジェクトを作成する、システム定義の「コンストラクタ・メソッド」があります。コンストラクタ・メソッドの名前は、そのオブジェクト型の名前と同じであり、コンストラクタ・メソッドのパラメータの名前と型は、オブジェクト型の属性の名前と型です。コンストラクタ・メソッドはファンクションであり、新しいオブジェクトを値として戻します。

たとえば、次のような式は、

```
purchase_order(
    1000376,
    external_person ("John Smith", "1-800-555-1212"),
    NULL )
```

次のような属性をもつ発注情報オブジェクトを表します。

```
id          1000376
contact     external_person("John Smith", "1-800-555-1212")
lineitems   NULL
```

式 `external_person ("John Smith", "1-800-555-1212")` により、オブジェクト型 `EXTERNAL_PERSON` のコンストラクタ・ファンクションが起動されます。これにより戻されるオブジェクトが、発注情報の `contact` 属性になります。

`NULL` オブジェクトと `NULL` 属性の説明は、14-3 ページの「[NULL](#)」を参照してください。

比較メソッド メソッドは、オブジェクトの比較でも役割を果たします。Oracle には、特定のビルトイン型の 2 つのデータ項目（たとえば 2 つの数値）を比較し、一方が他方より大きい、等しい、または小さいかどうかを判別する機能があります。しかし、Oracle では、定義者がさらに指示を与えないかぎり、任意のユーザー定義型の 2 つの項目を比較することができません。Oracle は、特定のオブジェクト型のオブジェクト間の順序関係を定義するための 2 つの方法として、「マップ」メソッドと「順序」メソッドを提供しています。

マップ・メソッドでは、ビルトイン型を比較する Oracle の機能を使用します。たとえば、「`RECTANGLE` (四角形)」というオブジェクト型に「`HEIGHT` (高さ)」および「`WIDTH` (幅)」という属性を定義した場合を考えます。「`area` (面積)」というマップ・メソッドを定義して、数値、つまり四角形の「`HEIGHT`」属性と「`WIDTH`」属性の積を戻すようにします。この場合は、面積によって 2 つの四角形を比較できます。

順序メソッドは、より一般的な方法です。順序メソッドは、独自の内部論理を使用して、特定のオブジェクト型に属する 2 つのオブジェクトを比較します。それは、順序関係を符号化した値を戻します。たとえば、最初のオブジェクトのほうが小さければ -1 を戻し、どちらも同じ大きさであれば 0 を戻し、最初のオブジェクトのほうが大きければ 1 を戻す、などです。

たとえば、「ADDRESS (住所)」というオブジェクト型に「STREET (番地)」、「CITY (市町村)」、「STATE (都道府県)」および「ZIP (郵便番号)」という属性を定義した場合を考えます。アプリケーションで住所を比較して「より大きい」か「より小さい」かを判別するのは無意味ですが、2 つの住所が等しいかどうかを判別するには複雑な計算をする必要があるかもしれません。

オブジェクト型を定義する際、マップ・メソッドか順序メソッドのどちらかを指定できますが、両方は指定できません。オブジェクト型に比較メソッドがない場合、Oracle はその型の 2 つのオブジェクト間の大小関係を判別できません。ただし、その型の 2 つのオブジェクトが等しいかどうかの判別は試行できます。

比較メソッドを持たない型の 2 つのオブジェクトを比較する場合、Oracle は、対応する属性を比較します。

- すべての属性が NULL 以外であり、等しい場合、Oracle はその 2 つのオブジェクトが等しいと報告する。
- 2 つのオブジェクトに等しくない NULL 以外の値がある場合、Oracle はその 2 つのオブジェクトが等しくないと報告する。
- その他の場合、Oracle は比較できないことを報告する (NULL)。

追加情報： 比較メソッドを指定して使用する例は、『Oracle8i 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。

オブジェクト表

「オブジェクト表」は特殊な種類の表で、オブジェクトを保持し、各オブジェクトの属性のリレーショナル・ビューを提供します。

たとえば、次の文は、以前に定義した EXTERNAL_PERSON 型のオブジェクトのオブジェクト表を定義します。

```
CREATE TABLE external_person_table OF external_person;
```

Oracle では、次の 2 つの方法でこの表を表示できます。

- 単一列の表。それぞれのエントリが EXTERNAL_PERSON オブジェクトになっています。
- 複数列の表。オブジェクト型 EXTERNAL_PERSON のそれぞれの属性、つまり NAME と PHONE が列を 1 つずつ占有します。

たとえば、次の命令を実行できます。

```
INSERT INTO external_person_table VALUES (  
    "John Smith",  
    "1-800-555-1212" );  
  
SELECT VALUE(p) FROM external_person_table p  
    WHERE p.name = "John Smith";
```

最初の命令は、複数列の表としての EXTERNAL_PERSON_TABLE に EXTERNAL_PERSON オブジェクトを挿入します。2 番目の命令は、単一系列の表としての EXTERNAL_PERSON_TABLE から選択します。

行オブジェクトと列オブジェクト オブジェクト表に表示されるオブジェクトのことを「行オブジェクト」といいます。表の列に表示されるオブジェクト、または他のオブジェクトの属性として表示されるオブジェクトのことを「列オブジェクト」といいます。

オブジェクト識別子

オブジェクト表のすべての行オブジェクトには、論理オブジェクト識別子（OID）が対応付けられています。デフォルトで、各行オブジェクトには OID として長さ 16 バイトの一意的システム生成識別子が割り当てられます。Oracle は、オブジェクト識別子の内部構造の説明や、内部構造へのアクセス手段を提供していません。内部構造は、随時変更されることがあります。

オブジェクト表の OID 列は、非表示列です。OID 値そのものはオブジェクト・リレーショナル・アプリケーションにほとんど意味を持ちませんが、Oracle はこの値を使用して行オブジェクトへのオブジェクト参照を組み立てます。後述のように、アプリケーションは、オブジェクトのフェッチとナビゲートに使用されるオブジェクト参照のみに注意しなければなりません。

行オブジェクトの OID の目的は、そのオブジェクトをオブジェクト表内で一意に識別することです。そのために、Oracle はオブジェクト表の OID 列に索引を暗黙的に作成してメンテナンスします。システム生成の一意識別子には、分散およびレプリケート環境でオブジェクトが明確に識別されること以外にも、多数の利点があります。

主キーに基づくオブジェクト識別子 システム生成のグローバル一意識別子が提供する機能性を必要としないアプリケーションでは、各オブジェクトとともに余分の 16 バイトを格納し、その索引をメンテナンスしても、効率的でない場合があります。Oracle では、行オブジェクトのオブジェクト ID としてその主キー値を指定する方法を選択できます。

また、主キーに基づく識別子には、より効率的かつ容易にオブジェクト表をロードできるという利点もあります。これに対して、システム生成のオブジェクト識別子は、特にその参照も永続的に格納される場合には、なんらかのユーザー指定キーを使用して再マップする必要があります。

オブジェクト・ビュー

オブジェクト・ビュー（第 15 章「オブジェクト・ビュー」を参照）は、仮想的なオブジェクト表です。オブジェクト・ビューの行は、行オブジェクトです。Oracle は、永続的には格納しないオブジェクト識別子を、基礎を形成する表やビューの主キーから具象化します。

REF

リレーショナル・モデルの場合、外部キーは多対 1 の関係を表します。Oracle オブジェクト型では、多対 1 の関係で「1」の側が行オブジェクトである場合に、この関係を表すための効率的な方法が提供されています。

Oracle は、指定したオブジェクト型の行オブジェクトの参照をカプセル化するために、REF というビルトイン・データ型を提供します。モデル化の観点から見ると、REF を使用すると 2 つの行オブジェクト間の関連付けを獲得できます。Oracle は、この種の REF を組み立てるためにオブジェクト識別子を使用します。

REF を使用して、参照先のオブジェクトを検査したり更新したりすることができます。また、参照先のオブジェクトのコピーを取得するためにも使用できます。REF に対して実行できる変更は、同じオブジェクト型の異なるオブジェクトへの参照に内容を置き換えるか、NULL 値を割り当てるという変更だけです。

有効範囲付き REF 列タイプまたはコレクション要素、オブジェクト型の属性を REF として宣言する際、REF に制約を適用して、指定したオブジェクト表への参照のみを含めることができます。このような REF は「有効範囲付き REF」といいます。有効範囲付き REF は記憶領域の所要量が少なく、有効範囲なしの REF よりも効率的にアクセスできます。

DANGLING REF REF で識別されるオブジェクトが削除されたり権限が変更されて、そのオブジェクトを使用できなくなる可能性があります。このような REF を「DANGLING REF」といいます。Oracle の SQL は、この条件が当てはまるかどうか REF をテストする述語（IS DANGLING）を提供しています。

REF の参照解除 REF で参照しているオブジェクトにアクセスすることを、REF の「参照解除」といいます。Oracle には、そのための DEREF 演算子が用意されています。DANGLING REF への参照を解除すると、結果は NULL オブジェクトになります。

Oracle は、REF の「暗黙的な参照解除」を提供しています。たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (
  name    VARCHAR2(30),
  manager REF person );
```

x が PERSON 型のオブジェクトを表している場合、次の式は、

```
x.manager.name
```

X の MANAGER 属性が参照する PERSON オブジェクトの NAME 属性を含む文字列を表します。つまり、前述の式は、次の式の短縮形です。

```
y.name, where y = Deref(x.manager)
```

REF の取得 行オブジェクトへの REF は、オブジェクト表からそのオブジェクトを選択し、REF 演算子を適用することによって取得できます。たとえば、識別番号 1000376 の発注情報への REF は、次のようにして取得できます。

```
DECLARE OrderRef REF to purchase_order;  
  
SELECT REF(po) INTO OrderRef  
      FROM purchase_order_table po  
      WHERE po.id = 1000376;
```

オブジェクトと REF の記憶領域の詳細は、14-9 ページの「[コレクション](#)」を参照してください。

追加情報： REF の使用例は、『Oracle8i 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。

コレクション型

それぞれのコレクション型は、すべて同じデータ型の不特定数の要素で構成されるデータ単位を記述します。コレクション型には、「配列型」と「表型」があります。

配列型および表型は、スキーマ・オブジェクトです。それぞれに対応するデータ単位を、「VARRAY」および「ネストした表」といいます。混乱の恐れがない場合は、コレクション型のことを単に VARRAY およびネストした表と呼ぶこともよくあります。

コレクション型には、コンストラクタ・メソッドがあります。コンストラクタ・メソッドの名前はコレクション型の名前と同じで、その引数は新しいコレクションの要素をカンマで区切ったリストです。コンストラクタ・メソッドはファンクションです。新しいコレクションを値として返します。

コレクション型の名前に空のカッコを付けた式は、その型を持つ空のコレクションを作成するコンストラクタ・メソッドのコールを表します。空のコレクションは、NULL コレクションとは異なります。

VARRAY

「配列」とは、データ「要素」を順番に並べた集合のことです。特定の配列のすべての要素は、同じデータ型で構成されます。各要素には「索引」があり、これは配列内におけるその要素の位置に対応する番号です。

配列内の要素の数が、配列の「サイズ」になります。Oracle の配列のサイズは可変 (variable) なので、VARRAY と呼ばれます。配列型を宣言するときは、最大サイズを指定する必要があります。

たとえば、次のような文で配列型を宣言します。

```
CREATE TYPE prices AS VARRAY(10) OF NUMBER(12,2);
```

これは、PRICES 型の VARRAY で、最大 10 個までの要素を入れることができ、各要素のデータ型は NUMBER(12,2) です。

配列型を作成しても、領域は割り当てられません。単にデータ型を定義しているだけです。このデータ型は、次の用途に使用できます。

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

通常、VARRAY はひとまとめに格納されます。つまり、行に入っているその他のデータと同じ表領域に格納されます。ただし、サイズが十分に大きい場合には、Oracle はこれを BLOB として格納します（14-19 ページの「[ユーザー定義型の Import/Export](#)」を参照）。

追加情報： VARRAY の使用方法の詳細は、『Oracle8i 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。

ネストした表

「ネストした表」は、すべて同じデータ型のデータ「要素」を順不同で並べた集合です。この表には単一の列があり、この列の型はビルトイン型かオブジェクト型です。オブジェクト型の場合は、この表を、オブジェクト型の各属性に対応する列のある、複数列の表とみなすこともできます。

たとえば、発注情報の例では、次の文により、品目を入れるネストした表のために使用する表型を宣言します。

```
CREATE TYPE lineitem_table AS TABLE OF lineitem;
```

表型を定義しても、領域は割り当てられません。単に型を定義しているだけです。この型は、次の用途に使用できます。

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

表型が、リレーショナル表の列の型、またはオブジェクト表の基礎を形成するオブジェクト型の属性として使用された場合、Oracle はネストした表のすべてのデータを単一の表に格納します。Oracle は、その表を、それを含むリレーショナル表またはオブジェクト表と対応付けます（14-18 ページの「[ネストした表](#)」を参照）。たとえば、次の文は、オブジェクト型 PURCHASE_ORDER に対してオブジェクト表を定義します。

```
CREATE TABLE purchase_order_table OF purchase_order  
  NESTED TABLE lineitems STORE AS lineitems_table;
```

2 番目の行は、PURCHASE_ORDER_TABLE にあるすべての PURCHASE_ORDER オブジェクトの LINEITEMS 属性を格納する表として、LINEITEMS_TABLE を指定しています。

ネストした表の要素に個々にアクセスする簡便な方法は、ネストしたカーソルを使用することです。

追加情報： ネストしたカーソルの詳細は『Oracle8i リファレンス・マニユアル』、ネストした表の使用方法的詳細は『Oracle8i 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。

アプリケーション・インタフェース

Oracle には、アプリケーション・プログラムでユーザー定義データ型を使用するための機能が多数あります。

- [SQL](#)
- [PL/SQL](#)
- [Pro*C/C++](#)
- [OCI](#)
- [OTT](#)
- [JPublisher](#)
- [JDBC](#)
- [SQLJ](#)

SQL

Oracle SQL DDL は、ユーザー定義データ型のために次のようなサポートを提供しています。

- オブジェクト型およびネストした表、配列の定義。
- 権限の指定。
- ユーザー定義型の表の列の指定。
- オブジェクト表の作成。

Oracle SQL DML は、ユーザー定義データ型のために次のようなサポートを提供しています。

- オブジェクトとコレクションの問合せおよび更新。
- REF の操作。

追加情報： Oracle SQL 構文の詳細説明は、『Oracle8i SQL リファレンス』を参照してください。

PL/SQL

PL/SQL は、SQL を拡張するプロシージャ型言語です。パッケージおよびデータのカプセル化、情報の隠ぺい、オーバーロード、例外処理など、最新のソフトウェア・エンジニアリング機能が提供されています。ほとんどのストアード・プロシージャは、PL/SQL で記述されています。

PL/SQL では、ファンクションおよびプロシージャ内から、ユーザー定義型をサポートする SQL 機能を使用することができます。

PL/SQL ファンクションとプロシージャのパラメータと変数は、ユーザー定義型にすることができます。

PL/SQL は、オブジェクト型に結び付けるメソッドをインプリメントするのに必要なすべての機能を備えています。これらのメソッド（ファンクションとプロシージャ）は、ユーザーのスキーマの一部としてサーバーに格納されます。

追加情報： PL/SQL の詳細説明は、『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。

Pro*C/C++

Oracle Pro*C/C++ プリコンパイラを使用すると、プログラマは C および C++ プログラムでユーザー定義データ型を使用できます。

Pro*C を使用する開発者は、Object Type Translator を使用して、Oracle のオブジェクト型とコレクションを C のデータ型にマップし、Pro*C アプリケーションで使用できます。

Pro*C は、オブジェクト型とコレクションのコンパイル時型チェック機能と、データベース型から C データ型への自動型変換を提供しています。

Pro*C は、オブジェクトを作成したり破棄したりするための EXEC SQL 構文と、サーバーにあるオブジェクトにアクセスする 2 つの方法を提供しています。

- Pro*C プログラムに埋め込まれている SQL 文と PL/SQL ファンクションまたはプロシージャ。
- オブジェクト・キャッシュへの単純なインタフェース（13-14 ページの「OCI」の項で説明します）。網羅用の（traversing）ポインタでオブジェクトにアクセスし、その後サーバー上で変更および更新します。

追加情報： Pro*C プリコンパイラの詳細説明は、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

OCI

Oracle コール・インタフェース (OCI) は、Oracle Server への C 言語インタフェースの集合です。OCI を使用すると、プログラマはサーバーの機能を使用するときに融通を利かせることができます。

OCI の重要な要素は、オブジェクト・キャッシュと呼ばれる作業領域をアプリケーション・プログラムから使用できるようにするコールの集合です。「オブジェクト・キャッシュ」は、クライアント側にあるメモリー・ブロックで、プログラムがオブジェクト全体を格納し、サーバーとの間を往復せずにオブジェクト間をナビゲートできるようにします。

オブジェクト・キャッシュは、そのオブジェクト・キャッシュを使用するアプリケーション・プログラムが完全に制御および管理します。Oracle Server は、オブジェクト・キャッシュにアクセスできません。オブジェクト・キャッシュを使用するアプリケーション・プログラムは、サーバーとのデータの一貫性を維持しながら、同時発生のアクセスによる衝突から作業領域を保護する必要があります。

OCI は、次の機能を持つ関数を提供しています。

- SQL を使用してサーバー上のオブジェクトにアクセスする。
- 網羅用の (traversing) ポインタまたは REF によってオブジェクト・キャッシュ内のオブジェクトにアクセスし、操作し、管理する。
- Oracle の日付および文字列、数値を C データ型に変換する。
- オブジェクト・キャッシュのメモリーのサイズを管理する。

OCI の同時実行性が向上し、個々のオブジェクトにロックをかけることができるようになりました。また、複合オブジェクト検索をサポートすることにより、パフォーマンスが向上しています。

OCI を使用する開発者は、Object Type Translator を使用して、Oracle オブジェクト型に対応する C データ型を生成できます。

追加情報： OCI の詳細説明は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

OTT

Oracle Type Translator (OTT) は、オブジェクト型に対応する C 言語の構造体の宣言を自動生成するプログラムです。OTT は、Pro*C プリコンパイラと OCI サーバー・アクセス・パッケージを使用するときの手助けをします。

追加情報： OTT の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』および『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

JPublisher

Java Publisher (JPublisher) は、データベース内のユーザー定義型に対応する Java クラス定義を自動的に生成するプログラムです。JPublisher は、SQLJ および JDBC サーバー・アクセス・パッケージの使用を容易にします。

追加情報： JPublisher の詳細は、『Oracle8i JPublisher User's Guide』を参照してください。

JDBC

JDBC (Java Database Connectivity) は、Oracle サーバーへの Java インタフェースの集合です。Oracle の JDBC の機能は、次のとおりです。

- 動的 SQL を通じて Java プログラム内のオブジェクト型とコレクション型 (データベース内で定義) にアクセスできるようにする。
- データベース内で定義されている型を、デフォルトまたはカスタマイズ可能なマッピングを通じて Java のクラスに変換する。

追加情報： JDBC の詳細は、『Oracle8i 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。

SQLJ

SQLJ により、開発者は Java プログラムでユーザー定義データ型を使用できます。開発者は、JPublisher を使用して、Oracle のオブジェクト型とコレクション型をアプリケーションで使用される Java クラスにマップできます。

SQLJ は、Java コードに埋め込まれた SQL 文を使用して、サーバー・オブジェクトへのアクセスを提供します。また、コンパイル時には、SQL 文中のオブジェクト型とコレクションの型チェックを提供します。

追加情報： SQLJ の詳細は、『Oracle8i Java 開発者ガイド』を参照してください。

ユーザー定義データ型の使用方法

It is not enough to have a good mind. The main thing is to use it well.

René Descartes, *Le Discours de la Méthode*

この章では、ユーザー定義データ型を使用するにあたって理解しておく必要のある主要な概念を説明します。この章の内容は、次のとおりです。

- [オブジェクト型と参照](#)
- [コレクション](#)
- [ユーザー定義型の権限とそのメソッド](#)
- [依存性と不完全な型](#)
- [ユーザー定義型の記憶領域](#)
- [ユーティリティ](#)

注意： この章で説明している機能は、Oracle8i Enterprise Edition を購入した場合にのみ使用できます。

基礎知識

どんなアプリケーションのデータ・モデルも、エンティティ、エンティティ相互の対応付けおよびエンティティと対応付けの両方を記述する属性からなっています。

たとえば、発注処理アプリケーションには、CUSTOMER、ORDER および LINE-ITEM などのエンティティが含まれています。顧客が実社会で複数の明細項目からなる注文を出すと、これらの関連によってデータ・モデル内のエンティティ間の対応付けが表されます。これらの各エンティティは、1 つ以上の属性を持つことができます。たとえば、エンティティ CUSTOMER は CUSTOMER NAME や CUSTOMER ADDRESS などの属性を持ち、LINE-ITEMS は QUANTITY や PRICE などの属性を持つ場合があります。

属性そのものは、構造化されていても、複数の値を持つものでもかまいません。たとえば、NAME は FIRSTNAME と LASTNAME で構成され、ADDRESS は STREET、CITY、STATE および ZIP で構成されることがあります。同様に、LINE-ITEMS は LINE-ITEMS の集合になっていることがあります。

Oracle8i リリース 8.1.x では、この種のエンティティとその構造化属性の両方をモデル化するオブジェクト型を定義できます。コレクション型を使用して複数の値を持つ属性をモデル化し、オブジェクト参照またはそのコレクション型を使用して、エンティティ相互の対応付けをモデル化できます。対応付けに参加するエンティティのオブジェクト参照を含むオブジェクト型を使用して対応付けをモデル化する場合は、そのオブジェクト型で表されている属性で対応付けが記述されます。

オブジェクト型と参照

この項では、次のようにオブジェクト型と参照について説明します。

- [オブジェクト属性のプロパティ](#)
- [オブジェクト参照](#)
- [名前変換](#)

オブジェクト属性のプロパティ

Oracle では、オブジェクト属性のいくつかのプロパティを指定できます。

- [NULL](#)
- [デフォルト](#)
- [制約](#)
- [索引](#)
- [トリガー](#)

NULL

表の列、オブジェクト、オブジェクト属性、コレクションまたはコレクション要素の特性の 1 つは、それを NULL にすることができるという点です。つまり、その項目を NULL に初期化したり、未初期化状態のまますることができます。通常、これは、その項目の値は今不明だが、後で使用可能になるということを意味します。

値が NULL であるオブジェクトのことを「アトミック NULL」といいます。さらに、オブジェクトの属性を NULL にすることもできます。これら 2 つの NULL の使用方法は異なります。

たとえば、次のように定義される CONTACTS 表を考えてみます。

```
CREATE TYPE external_person AS OBJECT (  
    name          VARCHAR2(30),  
    phone         VARCHAR2(20) );  
  
CREATE TABLE contacts (  
    contact        external_person  
    date          DATE );
```

このとき、次の文は、

```
INSERT INTO contacts VALUES (  
    external_person (NULL, NULL),  
    '24 Jun 1997' );
```

次の文とは異なる結果になります。

```
INSERT INTO contacts VALUES (  
    NULL,  
    '24 Jun 1997' );
```

どちらの場合も、Oracle は CONTACTS 表に新しい行のための領域を割り当てて、DATE 列に指定された値を設定します。最初の文の場合、Oracle は EXTERNAL_PERSON 列にオブジェクトのための領域を割り当てて、その各属性に NULL を設定します。2 番目の文の場合は、EXTERNAL_PERSON 列に NULL を設定し、オブジェクトのための領域は割り当てません。

表の行は NULL にすることができません。このため、行オブジェクトを NULL に設定することはできません。同様に、オブジェクトのネストした表にも、NULL 値の要素を入れることはできません。

ネストした表または配列は、NULL にすることができます。NULL コレクションは、空のコレクション、つまり要素が入っていないコレクションとは異なります。

デフォルト

表の列をオブジェクト型またはコレクション型として宣言する場合は、DEFAULT 句を含めることができます。DEFAULT 句には、その列の値を明示的に指定しなかった場合に使用される値を指定します。DEFAULT 句には、そのオブジェクトまたはコレクションのコンストラクタ・メソッドの「リテラル起動」を含める必要があります。

コンストラクタ・メソッドの「リテラル起動」は、リテラルまたはコンストラクタ・メソッドのリテラル起動のどちらかを引数とするコンストラクタ・メソッドの起動として再帰的に定義します。

たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (  
    id          NUMBER  
    name        VARCHAR2(30),  
    address     VARCHAR2(30) );  
  
CREATE TYPE people AS TABLE OF person;
```

このとき、次の文は、ネストした表型 PEOPLE のコンストラクタ・メソッドのリテラル起動です。

```
people ( person(1, 'John Smith', '5 Cherry Lane'),  
          person(2, 'Diane Smith', NULL) )
```


次の例は、コンストラクタ・メソッドのリテラル起動を使用してデフォルトを指定する方法を示しています。

```
CREATE TABLE department (
  d_no      CHAR(5) PRIMARY KEY,
  d_name    CHAR(20),
  d_mgr     person DEFAULT person(1, 'John Doe', NULL),
  d_emps    people DEFAULT people() )
NESTED TABLE d_emps STORE AS d_emps_tab;
```

PEOPLE() の項が、空の PEOPLE 表を作成するためのコンストラクタ・メソッドのリテラル起動になっていることに注意してください。

制約

他の表の場合と同様に、オブジェクト表に対しても制約を定義できます。

列オブジェクトのリーフ・レベル・スカラー属性に制約を定義できます。ただし、有効範囲なしの REF には定義できません（13-9 ページの「[有効範囲付き REF](#)」を参照）。

次に、いくつかの例を示します。

最初の例では、オブジェクト表 PERSON_EXTENT の SSNO 列に主キー制約を定義します。

```
CREATE TYPE location (
  building_no NUMBER,
  city         VARCHAR2(40) );

CREATE TYPE person (
  ssno        NUMBER,
  name        VARCHAR2(100),
  address     VARCHAR2(100),
  office      location );

CREATE TABLE person_extent OF person (
  ssno        PRIMARY KEY );
```

次の例の DEPARTMENT 表には、前の例で定義したオブジェクト型 LOCATION を型とする列があります。この例では、表の DEPT_LOC 列に入れる LOCATION オブジェクトのスカラー属性に制約を定義します。

```
CREATE TABLE department (
  deptno     CHAR(5) PRIMARY KEY,
  dept_name  CHAR(20),
  dept_mgr   person,
  dept_loc   location,
  CONSTRAINT dept_loc_cons1
    UNIQUE (dept_loc.building_no, dept_loc.city),
  CONSTRAINT dept_loc_cons2
```

```
CHECK (dept_loc.city IS NOT NULL) );
```

索引

他の表の場合と同様に、オブジェクト表や、ネストした表の列または属性の格納表にも、索引を定義できます。

索引は、列オブジェクトのリーフ・レベル・スカラー属性に対して定義できます。ただし、REF に対しては、REF が有効範囲付きの場合に限り、REF 属性または列に索引を定義できます（13-9 ページの「[有効範囲付き REF](#)」を参照）。

次の例では、オブジェクト列の属性に索引を定義します。

```
CREATE TABLE department (  
    deptno      CHAR(5) PRIMARY KEY,  
    dept_name   CHAR(20),  
    dept_addr   address );  
  
CREATE INDEX i_dept_addr1  
    ON department (dept_addr.city);
```

このコーディングにより、department の address の city 属性に索引が作成されます。

索引の定義において列名を指定できる位置には、オブジェクト列のスカラー属性を指定することもできます。

トリガー

他の表の場合と同様に、オブジェクト表にもトリガーを定義できます。ネストした表の列または属性の格納表にはトリガーを定義できません。

トリガー本体の LOB 値は変更できません。その他には、ユーザー定義型に対してトリガーを使用する場合の制限事項は特にありません。

次の例では、前の項で定義した PERSON_EXTENT 表にトリガーを定義します。

```
CREATE TABLE movement (  
    ssno        NUMBER,  
    old_office  location,  
    new_office  location );  
  
CREATE TRIGGER trig1  
    BEFORE UPDATE  
        OF office  
        ON person_extent  
    FOR EACH ROW  
        WHEN new.office.city = 'REDWOOD SHORES'  
    BEGIN  
        IF :new.office.building_no = 600 THEN  
            INSERT INTO movement (ssno, old_office, new_office)
```

```
VALUES (:old.ssno, :old.office, :new.office);
END IF;
END;
```

オブジェクト参照

Oracle では、SCOPE 句または参照制約句を使用して、REF または属性を制約なしにしたり、制約付きにすることができます。REF 列が制約なしの場合は、オブジェクト参照を、それに対応するオブジェクト型の任意のオブジェクト表に含まれている行オブジェクトに格納できます。

Oracle では、この種の列に格納されているオブジェクト参照が、既存の有効な行オブジェクトを「指す」という意味で有効かどうかは保証されません。したがって、REF 列には、既存の行オブジェクトを「指さない」オブジェクト参照が含まれる可能性があります。この種の REF 値を「DANGLING REF」といいます。現在、Oracle では、主キー・ベースの OID を含むオブジェクト参照を、制約なしの REF 列に格納することは許されません。

REF 列は、特定のオブジェクト表への「有効範囲付き」になるように制約を付けることができます。SCOPE 制約付きの列に格納されているすべての REF 値は、SCOPE 句で指定された表の行オブジェクトを「指し」ます。ただし、REF 値を「DANGLING」にすることもできます。

REF 列には、外部キーの仕様部のように、REFERENTIAL 制約で制約を付けることができます。この種の列には、参照制約のルールが適用されます。つまり、これらのオブジェクトに格納されているオブジェクト参照は、指定されたオブジェクト表内の既存の有効な行オブジェクトを指す必要があります。

REF 列に UNIQUE または PRIMARY KEY 制約を指定することはできません。ただし、この種の列に NOT NULL 制約を指定することはできます。

名前変換

Oracle SQL は、使用しやすさを目指して設計されています。たとえば、PROJECTS が ASSIGNMENT という列を持つ表で、DEPTS が ASSIGNMENT という列を含まない表であれば、次のように記述できます。

```
SELECT *
FROM projects
WHERE EXISTS
  (SELECT * FROM depts
   WHERE assignment = task);
```

Oracle は、それぞれの列がどの表に属するかを判断します。次のように列名を表名で修飾することは可能ですが、その必要はありません。

```
SELECT *
FROM projects
WHERE EXISTS
```

```
(SELECT * FROM depts
WHERE projects.assignment = depts.task);
```

次のように列名を表の別名で修飾することは可能ですが、その必要はありません。

```
SELECT *
FROM projects pj
WHERE EXISTS
  (SELECT * FROM depts dp
   WHERE pj.assignment = dp.task);
```

表の別名

上記の最初の SELECT 文の形式は最も記述しやすく理解しやすい形式ですが、後で DEPTS 表に ASSIGNMENT 列を追加して、問合せを変更し忘れると、予期しない結果になる可能性があります。Oracle は自動的にその問合せを再コンパイルし、新しいバージョンでは DEPTS 表の ASSIGNMENT 列を使用します。このような状況のことを「内側獲得」といいます。

内側獲得や、それに類似した SQL 文の解釈間違いを回避するには、表の別名を使用して、オブジェクトのメソッドや属性への参照を修飾する必要があります。これは、REF による属性参照にも適用されます。このような要件のことを「獲得回避規則」といいます。

たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (ssno VARCHAR(20));
CREATE TABLE ptab1 OF person;
CREATE TABLE ptab2 (c1 person);
```

これらの文は、オブジェクト型 PERSON と、2 つの表を定義します。最初の表は、オブジェクト型 PERSON のためのオブジェクト表です。2 番目の表には、型 PERSON の列が 1 つあります。

このとき、次の文を考えてみます。

```
SELECT      ssno FROM ptab1   ;  --Correct
SELECT  c1.ssno FROM ptab2   ;  --Wrong
SELECT p.c1.ssno FROM ptab2 p ;  --Correct
```

- 最初の SELECT 文で、SSNO は PTAB1 の列の名前である。それ以上の修飾は不要です。
- 2 番目の SELECT 文で、SSNO は、C1 という列に入っている PERSON オブジェクトの属性の名前である。この参照には、表の別名が必要です。
- 3 番目の SELECT 文は 2 番目と同じであるが、必要な表の別名 P が指定されている。

オブジェクトの属性への参照を、表の別名ではなく表の名前で修飾することは、その表名をさらにスキーマ名で修飾したとしても、この要件を満たすことにはなりません。

たとえば、問合せで次のような文を使用しても、

```
scott.projects.assignment.duedate
```

SCOTT スキーマの PROJECTS 表の ASSIGNMENT 列の DUEDATE 属性は参照できません。

表の別名は、問合せの中で一意になっている必要があり、問合せの中に指定できるスキーマ名と同じ名前は指定しないでください。

注意： オラクル社では、すべての UPDATE 文、DELETE 文、SELECT 文および副問合せの中で表の別名を定義し、列にオブジェクト型が入っているかどうかに関係なく、列の参照を表の別名で修飾することをお薦めしています。

引数なしのメソッド・コール

メソッドは、ファンクションまたはサブルーチンです。メソッドを起動する正しい構文では、メソッド名の後にカッコを使用して、コール引数を囲みます。あいまいさを避けるため、Oracle では、引数がないメソッド・コールにも空のカッコが必要です。

たとえば、オブジェクト型 T の列 C を持つ表 TB があり、T に引数を取らないメソッド m がある場合、次のような問合せは正しい構文の一例です。

```
SELECT p.c.m() FROM tb p;
```

これは、PL/SQL の規則とは異なっています。PL/SQL のファンクションおよびプロシージャでは、引数がないコールの場合はカッコを省略できます。

コレクション

この項では、次のようなコレクションの使用方法について説明します。

- [コレクションの問合せ](#)
- [コレクションのネスト解除](#)
- [ネストした表のロケータ](#)
- [コレクションの DML](#)

コレクションの問合せ

Oracle8i では、TABLE 式を使用してコレクション列を問合せできます。たとえば、表 (EMPLOYEES) のネストした表の列 (PROJECTS) は、次のように問合せできます。

```
SELECT * FROM TABLE(SELECT t.projects FROM employees t WHERE t.eno = 1000);
```

```
SELECT t.eno, CURSOR(SELECT * FROM TABLE(t.projects)) FROM employees t;
```

TABLE 式を使用すると、変数やパラメータなど、一時的な値を含むコレクション値表現を問合せできます。

注意： TABLE 式は、旧リリースで導入された副問合せにかわるものです。副問合せ式は、最終的には廃止されます。

コレクションのネスト解除

多くのツールやアプリケーションにはコレクション型を取り扱う機能は組み込まれておらず、データのフラット化したビューが必要です。これらのツールを使用して Oracle スキーマ内でコレクション・データを表示するには、行のコレクション属性のネストを解除するか、1 つ以上のリレーショナル行にフラット化する必要があります。次のオブジェクト・リレーショナル・スキーマを考えてみます。

```
CREATE TYPE emp_set_t IS NESTED TABLE of emp_t;  
CREATE TYPE dept_t(deptno NUMBER, emps emp_set_t);  
CREATE TABLE depts OF dept_t NESTED TABLE emps STORE AS depts_emps;
```

次の問合せでは、EMPS 列のデータが DEPT 表に関してネストされなくなり、EMPS の各行には親 DEPTS 行の引数が与えられます。

```
SELECT d.deptno, e.* FROM depts d, TABLE(d.emps) e;
```

また、Oracle8i は、外部結合の結果を生成するための次の構文をサポートしています。

```
SELECT d.*, e.* FROM depts d, TABLE(d.emps)(+) e;
```

(+) は、DEPTS と D.EMPS の間の「依存」結合に NULL の引数が必要であることを示します。つまり、D.EMPS が NULL または空になっている出力には DEPTS の行があり、D.EMPS に対応する列の NULL 値が入ります。

ネストした表のロケータ

Oracle8i では、コレクション型を持つ値はカプセル化されます。そのため、クライアントは Oracle 提供のインタフェースを介してコレクションの内容にアクセスする必要があります。

通常は、クライアントが（含まれるオブジェクトをフェッチして）ネストした表に明示的または暗黙的にアクセスすると、Oracle はクライアント・プロセスに集合値全体を戻します。ただし、パフォーマンス上の理由で、クライアントは、コレクションの要素がクライアント側で具象化されるタイミングを制御する必要があります。これは、特に集合値が極端に大きい場合に関連します。この種の操作を容易にするために、Oracle はネストした表の値をロケータとして戻す機能をサポートしています。

ネストした表のロケータは、集合値のハンドルに似ています。この種のロケータは、取出し時のデータベース・スナップショットを含めることによって、ネストした表の値またはコピー・セマンティクスを保存しようとしています。スナップショットにより、データベースは後でロケータを使用してコレクションの要素がフェッチされるときに、ネストした表の値の正しいインスタンスを取り出すことができます。ロケータの有効範囲は1つのセッションに限定され、セッション間では使用できません。データベース・スナップショットが使用されるため、ネストした表の更新率が高い場合は、"snapshot too old (スナップショットが古すぎます)"というエラーが戻されることがあります。LOB ロケータと違って、ネストした表のロケータは実際にロケータであり、コレクション・インスタンスの変更には使用できません。

コレクションの DML

Oracle は、ネストした表の列での次の DML 操作をサポートしています。

- コレクション全体の新しい値を提供する挿入と更新
- ピース単位更新
 - コレクションへの新規要素の挿入
 - コレクションからの要素の削除
 - コレクションの要素の更新

Oracle は、VARRAY 列のピース単位更新はサポートしていません。ただし、VARRAY 列はアトミック単位として挿入または更新できます。

ネストした表の列のピース単位更新の場合、DML 文は TABLE 式を使用して、操作されるネストした表の値を識別します。ネストした表の値の DML 操作は、直列化されるので注意してください。つまり、ネストした表の値がトランザクション内の DML 文によって操作されると、同じネストした表の値に対する他のトランザクションからの変更は、最初のトランザクションが終了するまでブロックされます。

次の DML 文は、ネストした表の列に対するピース単位操作を示しています。

```
INSERT INTO TABLE(SELECT e.projects
                     FROM      employees e
                     WHERE     e.eno = 100)
VALUES (1, 'Project Neptune');

UPDATE TABLE(SELECT e.projects
               FROM      employees e
               WHERE     e.eno = 100) p
SET VALUE(p) = project_t(1, 'Project Pluto')
WHERE p.pno = 1;

DELETE FROM TABLE(SELECT e.projects
                    FROM      employee e
```

```
WHERE          e.eno = 100) p
WHERE p.pno = 1;
```

ユーザー定義型の権限とそのメソッド

ユーザー定義データ型の権限は、システム・レベルとスキーマ・オブジェクト・レベルが存在します。

システム権限

Oracle は、ユーザー定義型に対して次のシステム権限を定義しています。

- CREATE TYPE。自分のスキーマ内にユーザー定義型を作成できます。
- CREATE ANY TYPE。任意のスキーマ内にユーザー定義型を作成できます。
- ALTER ANY TYPE。任意のスキーマ内のユーザー定義型を変更できます。
- DROP ANY TYPE。任意のスキーマ内の、指定した型を削除できます。
- EXECUTE ANY TYPE。任意のスキーマ内の、指定した型を使用したり参照できます。

CONNECT ロールおよび RESOURCE ロールには、CREATE TYPE システム権限が含まれています。DBA ロールには、前述の権限すべてが含まれています。

スキーマ・オブジェクト権限

ユーザー定義型に適用されるスキーマ・オブジェクト権限は、EXECUTE だけです。

ユーザー定義型に対する EXECUTE 権限があると、その型を次の目的で使用できます。

- 表を定義する。
- リレーショナル表に列を定義する。
- 名前を指定した型の変数またはパラメータを宣言する。

EXECUTE 権限があれば、コンストラクタなど、その型のメソッドを起動できます。

メソッドの実行とそれに関連する許可は、PL/SQL ストアド・プロシージャの場合と同じです。

新しい型または表での型の使用方法

これまで説明した許可の他にも、次の場合には特定の権限が必要です。

- 他のユーザーが作成した型を使用して、型または表を作成する。
- 自分が作成した新しい型または表の使用権限を他のユーザーに付与する。

新しい型または表を定義する場合は、EXECUTE ANY TYPE システム権限か、その定義で使用する型に対する EXECUTE オブジェクト権限が必要です。これらの権限は、ロールを通してではなく、明示的に受け取らなくてはなりません。

新しい型または表へのアクセス権限を他のユーザーに付与する場合は、必要な型に対する GRANT オプション付きの EXECUTE オブジェクト権限か、オプション WITH ADMIN OPTION 付きの EXECUTE ANY TYPE システム権限が必要です。これらの権限は、ロールを通してではなく、明示的に受け取らなくてはなりません。

例

CONNECT および RESOURCE ロールを持つ USER1、USER2 および USER3 という 3 人のユーザーが存在する場合を考えます。

USER1 は、USER1 スキーマで次の DDL を実行します。

```
CREATE TYPE type1 AS OBJECT ( attr1 NUMBER );
CREATE TYPE type2 AS OBJECT ( attr2 NUMBER );
GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

USER2 は、USER2 スキーマで次の DDL を実行します。

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT ( attr3 user1.type2 );
CREATE TABLE tab2 (col1 user1.type2 );
```

次の文は、正常に実行されます。USER2 は、USER1 の TYPE2 に対して GRANT オプション付きの EXECUTE 権限を持っているからです。

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT on tab2 TO user3;
```

ただし、次の権限付与は正しく実行されません。USER2 は、USER1.TYPE1 に対して GRANT オプション付きの EXECUTE 権限を持っていないからです。

```
GRANT SELECT ON tab1 TO user3;
```

USER3 は、次のアクションを正しく実行できます。

```
CREATE TYPE type4 AS OBJECT (attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

型アクセスとオブジェクト・アクセスについての権限

表の使用を規定する権限は、オブジェクト表にも等しく適用されます。

- SELECT 権限があれば、その表からオブジェクトとその属性にアクセスできる。

- UPDATE 権限があれば、その表にあるオブジェクトの属性を変更できる。
- INSERT 権限があれば、その表に新しいオブジェクトを追加できる。
- DELETE 権限があれば、その表からオブジェクトを削除できる。

表および列の同様の権限により、ユーザー定義型の表の列の使用方法が規定されます。

オブジェクト表の列は、オブジェクト表の型に対する権限がなくても選択できます。ただし、行オブジェクト全体を選択する場合は、この権限が必要です。

次のスキーマを考えてみます。

```
CREATE TYPE emp_type as object (  
    eno    NUMBER,  
    ename  CHAR(31),  
    eaddr  addr_t );
```

```
CREATE TABLE emp OF emp_type;
```

さらに、次の2つの問合せについて考えます。

```
SELECT VALUE(e) FROM emp e;  
SELECT eno, ename FROM emp;
```

どちらの問合せの場合も、Oracle は、EMP 表に対するユーザーの SELECT 権限をチェックします。最初の問合せの場合、ユーザーは、EMP_TYPE の型情報を取得してデータを解釈する必要があります。問合せによって EMP_TYPE 型がアクセスされると、Oracle はユーザーの EXECUTE 権限をチェックします。

しかし、2 番目の問合せを実行しても、指定された型が含まれていないので、Oracle は型の権限をチェックしません。

さらに、前の項のスキーマを使用して、USER3 は、次の問合せを実行できます。

```
SELECT tab1.col1.attr2 from user2.tab1 tab1;  
SELECT t.attr4.attr3.attr2 FROM tab3 t;
```

USER3 は基礎となる型に対する権限を持っていませんが、USER3 によるどちらの SELECT 文も成功します。これは、それらの型および表の所有者が GRANT オプション付きの必要な権限を持っているからです。

Oracle は、次の要求に対する権限をチェックし、要求側がそのアクションに必要な権限を持っていないとエラーを戻します。

- オブジェクトの REF 値を使用してオブジェクト・キャッシュ内でオブジェクトをピンしようとする、Oracle は、そのオブジェクトを含んでいるオブジェクト表に対する SELECT 権限と、オブジェクト型に対する EXECUTE 権限をチェックする。

- 既存のオブジェクトを修正したり、オブジェクト・キャッシュからオブジェクトをフラッシュしようとする、Oracle は目的のオブジェクト表に対する UPDATE 権限をチェックする。新しいオブジェクトをフラッシュしようとする、Oracle は目的のオブジェクト表に対する INSERT 権限をチェックする。
- オブジェクトを削除しようとする、Oracle は目的の表に対する DELETE 権限をチェックする。
- メソッドを起動すると、Oracle は対応するオブジェクト型に対する EXECUTE 権限をチェックする。

Oracle は、オブジェクト表に対する列レベルの権限は提供していません。

依存性と不完全な型

型定義は、相互に依存し合うことがあります。たとえば、オブジェクト型 EMPLOYEE（従業員）および DEPARTMENT（部門）を定義するときに、EMPLOYEE の 1 つの属性をその従業員が所属する部門にし、DEPARTMENT の 1 つの属性をその部門を管理する従業員にする場合を考えます。

このように、直接か中間の型を介するかを問わず、相互に依存している型のことを「相互に依存する型」といいいます。相互に依存する型のダイアグラムでは、依存性が矢印によって表され、矢印のパスは必ずどちらかの型のところで開始し終了しています。

Oracle で、このように循環する依存性が許可されるのは、そのサイクルの少なくとも 1 つのブランチで REF が使用されている場合だけです。

たとえば、次のような型を定義できます。

```
CREATE TYPE department;
```

```
CREATE TYPE employee AS OBJECT (
    name    VARCHAR2(30),
    dept    REF department,
    supv    REF employee );
```

```
CREATE TYPE emp_list AS TABLE OF employee;
```

```
CREATE TYPE department AS OBJECT (
    name    VARCHAR2(30),
    mgr     REF employee,
    staff   emp_list );
```

この定義は、相互に依存する型の適正な組であり、SQL DDL 文の順序も適正です。これらの定義文は、Oracle によりエラーなしでコンパイルされます。先頭の文は次のようになっています。

```
CREATE TYPE department;
```

この文は、任意指定です。この文により、コンパイル処理がエラーなしで進行します。この文は、DEPARTMENT を「不完全なオブジェクト型」として設定します。不完全なオブジェクト型への REF はエラーなしでコンパイルされるので、EMPLOYEE のコンパイルもエラーなしで進行します。

Oracle が最後の文まで到達すると、DEPARTMENT の定義が完了し、DEPARTMENT のすべてのコンポーネントが正しくコンパイルされたので、コンパイル処理はエラーなしで終了します。

DEPARTMENT を不完全な型として宣言しなければ、EMPLOYEE のコンパイル時にエラーが起きます。その後、Oracle は、EMPLOYEE を不完全なオブジェクト型としてスキーマ・オブジェクトのライブラリに自動的に追加します。この処理により、EMP_LIST および DEPARTMENT の宣言はエラーなしでコンパイルされます。EMP_LIST と DEPARTMENT が完了した後、EMPLOYEE が再コンパイルされると、エラーなしでコンパイルされて完全なオブジェクト型になります。

不完全な型を完全にする方法

不完全なオブジェクト型を宣言したら、その型をオブジェクト型として完全なものにする必要があります。たとえば、その型を表型や配列型に宣言することはできません。代替処置としては、その型を削除するしかありません。

このことは、Oracle によってその型が不完全なオブジェクト型にされた場合（前の項の例で、EMPLOYEE のコンパイルが失敗した場合など）も同じです。

表の型の依存性

SQL コマンド REVOKE および DROP TYPE に指定した型が、その型に依存する表またはその他の型を持つものであると、エラーが戻されて異常終了します。

どちらのコマンドの場合も、FORCE オプションを付ければ、この動作が上書きされます。コマンドは正しく実行され、影響を受けた表または型は無効になります。

アクセス時に型定義に依存するデータが表に含まれている場合、その型を変更すると、その表のデータにアクセスできなくなります。このような状況になるのは、その型に必要な権限が取り消されたり、型またはその型が依存する型が削除された場合です。その場合、表は無効になり、アクセスできなくなります。

必要な権限が再び付与されると、権限がないために無効になった表は自動的に有効になり、アクセスできるようになります。

依存する型が削除されたために無効になった表へのアクセスは、二度と回復できません。許可される唯一のアクションは、その表の削除だけです。

ユーザー定義型の記憶領域

Oracle は、ユーザー定義型のデータを表に格納して管理します。ユーザー定義型の複雑な構造が、自動的に表のシンプルな四角形構造にマップされます。

リーフ・レベル属性

オブジェクト型の構造は、ツリー構造に似ています。幹から伸びているブランチ（枝）が属性を表します。属性がオブジェクト型である場合は、このブランチから、新しいオブジェクト型の属性としてサブブランチが分岐します。

それぞれのブランチは、最終的には、ビルトイン型またはコレクション型の属性で終わります。これらの属性を、元のオブジェクト型の「リーフ・レベル属性」といいます。Oracle は、それぞれのリーフ・レベル属性用の列を表の中に用意します。

コレクション型でないリーフ・レベル属性を、オブジェクト型の「リーフ・レベル・スカラー属性」といいます。

行オブジェクト

オブジェクト表には、すべてのリーフ・レベル・スカラー属性または REF 属性について、Oracle が実際のデータを格納する列があります。これは、VARRAY の場合も（非常に大きい場合を除き）同じです（14-18 ページの「[VARRAY](#)」を参照）。Oracle は、表型のリーフ・レベル属性を、オブジェクト表に対応付けられた別々の表に格納します。これらの表を、オブジェクト表の宣言の一部として宣言する必要があります（14-18 ページの「[ネストした表](#)」を参照）。

オブジェクト表にあるオブジェクトの個々の属性にアクセスするには、単に表の列にアクセスします。オブジェクトの値そのものにアクセスすると、オブジェクト表の列を引数として、その型のデフォルト・コンストラクタが起動されます。つまり、Oracle はそのオブジェクトのコピーを提供します。

Oracle は、システム生成のオブジェクト識別子を非表示列に格納します。Oracle は、このオブジェクト識別子を使用して、オブジェクトへの REF を組み立てます。

列オブジェクト

表にオブジェクト型の列が定義されている場合、Oracle は、そのオブジェクト型のリーフ・レベル属性のための非表示列を表に追加します。追加された列には、そのオブジェクトの NULL 情報（つまり、トップレベルのオブジェクトとネストされたオブジェクトのアトミック NULL）が格納されます。

REF

Oracle は、行オブジェクトに対してビルトイン・ファンクション REF を起動することにより、行オブジェクトへの REF を組み立てます。組み立てられた REF は、オブジェクト識別子、オブジェクト表のメタデータおよび ROWID（オプション）で構成されています。

REF 型の列の REF のサイズは、その列と対応付けられている記憶プロパティに応じて異なります。たとえば、列が REF WITH ROWID として宣言されている場合、Oracle は ROWID を REF 列に格納します。制約付きの REF 列でのオブジェクト参照の場合、ROWID のヒントは無視されます。

列を SCOPE 句を指定した REF として宣言すると、Oracle は、オブジェクト表のメタデータと ROWID を列に格納しません。有効範囲付き REF の長さは 16 バイトです。

Oracle8i では、制約なしの REF 値は単一の列に格納されます。REF 列が有効範囲付きであるか、システム生成 OID を持つオブジェクト表への参照制約付きの場合、Oracle はその OID 値を格納するために 1 つの列を作成します。ただし、OID が主キー・ベースの場合、主キーを構成する列数に応じて、Oracle は主キー値を格納するために 1 つ以上の内部列を作成する場合があります。主キー・ベースの OID を含んでいる行オブジェクトの参照は、有効範囲付きまたは参照制約付きの REF 列には格納できません。

ネストした表

ネストした表の行は、別々の格納表に格納されます。ネストした表の列ごとに、それに対応付けられた記憶域表には、ネストした表のすべてのインスタンスの要素が（親表のすべての行に）含まれます。記憶域表には非表示の NESTED_TABLE_ID 列があり、この列には特定のネストした表の値の行を識別するために Oracle が使用するシステム生成値が格納されます。

Oracle では、記憶域表を索引構成表にすることができます。記憶域表を索引のみで構成すると、ネストした表の値の行をすべてクラスタ化できます。

Oracle は、ネストした表の型について、次の 2 つの形式の要素型をサポートしています。

- オブジェクト型
- NUMBER、VARCHAR2、REF などのスカラー型

要素の型がオブジェクト型の場合、オブジェクト型の最上位属性が記憶域表の列になるという点で、記憶域表はオブジェクト表によく似ています。ただし、ネストした表の行に対応付けられている OID 値はないため、OID 列は存在しません。このため、ネストした表内のオブジェクトは参照できなくなります。

要素型がスカラーのネストした表の記憶域表には、スカラー値を含む COLUMN_VALUE という 1 列が含まれます。

VARRAY

VARRAY のすべての要素は、1 つの列に格納されます。配列のサイズが 4000 バイト未満の場合、Oracle はその配列をひとまとめにして格納します。4000 バイトより大きい場合は、BLOB に格納します。

ユーティリティ

Oracle では、Oracle データベース内のデータを複数の方法で移動し、ロードできます。

ユーザー定義型の Import/Export

Export ユーティリティと Import ユーティリティは、Oracle データベースと外部との間でデータを移動します。また、データのバックアップを作成したりアーカイブしたりして、Oracle RDBMS の異なるリリースへの移行を支援することもできます。

Export と Import では、ユーザー定義型がサポートされています。Export は、ユーザー定義型の定義とそれに対応するすべてのデータを、ダンプ・ファイルに書き込みます。Import は、ダンプ・ファイルからそれらの項目を再作成します。

追加情報： Export と Import の詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

ユーザー定義型のロード

SQL*Loader は、行オブジェクト、列オブジェクトおよびコレクションと参照を持つオブジェクトのロードをサポートしています。オブジェクトの場合、Oracle8i では従来型パスのロードしかサポートされません。

追加情報： SQL*Loader の詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

オブジェクト・ビュー

The choice of a point of view is the initial act of a culture.

José Ortega y Gasset, *The Modern Theme*

この章では、オブジェクト・ビューについて説明します。この章の内容は、次のとおりです。

- 基礎知識
- オブジェクト・ビューの定義
- オブジェクト・ビューの使用方法
- オブジェクト・ビューの更新

注意： この章で説明している機能は、Oracle8i Enterprise Edition を購入した場合にのみ使用できます。この章に記載されている「Oracle Server」という用語は、Oracle8i Enterprise Edition を指しています。

基礎知識

ビューが仮想表であるのと同様に、「オブジェクト・ビュー」は仮想的なオブジェクト表です。

Oracle は、基本的なリレーショナル・ビュー・メカニズムへの機能拡張としてオブジェクト・ビューを提供します。オブジェクト・ビューを使用すると、データベースのリレーショナル表またはオブジェクト表の列に格納されているデータ（ビルトイン型またはユーザー定義型）から、仮想的なオブジェクト表を作成できます。

オブジェクト・ビューは、データベース内のデータおよびオブジェクトへの特化または制限付きのアクセスを提供する機能を実現します。たとえば、オブジェクト・ビューを使用して、機密データを含む属性がなく、削除メソッドがないバージョンの従業員オブジェクト表を提供できます。

オブジェクト・ビューにより、オブジェクト指向アプリケーションでリレーショナル・データを使用できるようになります。ユーザーは、オブジェクト・ビューを使用して次の操作ができます。

- 既存の表を変換せずにオブジェクト指向プログラミング技法を試してみる。
- リレーショナル表からオブジェクト・リレーショナル表へデータを段階的かつ透過的に変換する。
- 既存のオブジェクト指向アプリケーションで従来の RDBMS データを使用する。

オブジェクト・ビューの利点

オブジェクト・ビューを使用すると、パフォーマンスを改善できます。オブジェクト・ビューの 1 行を構成するリレーショナル・データは、1 つの単位としてネットワークを行き来するので、何度も往復する必要がありません。

リレーショナル・データをクライアント側のオブジェクト・キャッシュにフェッチして、「C」または「C++」の構造体にマップできるので、3GL アプリケーションはそのデータをネイティブな構造体と同じように処理できます。

オブジェクト・ビューは、従来のデータを段階的に移行していくための手段になります。

オブジェクト・ビューを使用すると、リレーショナル・アプリケーションとオブジェクト指向アプリケーションを共存させることができます。したがって、あるパラダイムを別のパラダイムに急激に切替えなくても、既存のリレーショナル・データにオブジェクト指向アプリケーションを簡単に適用できます。

オブジェクト・ビューには、同じリレーショナル・データまたはオブジェクト・データを複数の方法で表示できるという柔軟性もあります。したがって、データベースにデータを格納する方法を変えなくても、さまざまなアプリケーションに応じて異なるメモリー内オブジェクト表現を使用することができます。

オブジェクト・ビューの定義

概念上は、オブジェクト・ビューを定義する手順は簡単です。

1. オブジェクト・ビューの行によって表現するオブジェクト型を定義します。
2. どのリレーショナル表のどのデータに、その型のオブジェクトの属性が入っているかを指定する問合せを作成します。
3. オブジェクト・ビューのオブジェクト（行）を REF で参照できるように、基礎となるデータの属性に基づいてオブジェクト識別子を指定します。

オブジェクト識別子は、Oracle がオブジェクト表の行として自動生成する一意のオブジェクト識別子と対応しています。ただし、オブジェクト・ビューの場合は、基礎を形成するデータの中でなんらかの一意の値（主キーなど）を宣言する必要があります。

表または別のオブジェクト・ビューに基づくオブジェクト・ビューにオブジェクト識別子が指定されない場合、Oracle は、元の表またはオブジェクト・ビューのオブジェクト識別子を使用します。

複合オブジェクト・ビューを更新できるようにするには、さらに次のステップを実行する必要があります。

4. アプリケーション・プログラムがオブジェクト・ビューのデータを更新しようとするたびに Oracle で実行されるように、INSTEAD OF トリガー・プロシージャ（15-5 ページの「[オブジェクト・ビューの更新](#)」を参照）を作成します。

これらのステップを実行すれば、オブジェクト・ビューをオブジェクト表と同じように使用できます。

たとえば、次の SQL 文は、オブジェクト・ビューを定義します。

```
CREATE TABLE emp_table (
    empnum    NUMBER (5),
    ename     VARCHAR2 (20),
    salary    NUMBER (9, 2),
    job       VARCHAR2 (20) );

CREATE TYPE employee_t (
    empno     NUMBER (5),
    ename     VARCHAR2 (20),
    salary    NUMBER (9, 2),
    job       VARCHAR2 (20) );

CREATE VIEW emp_view1 OF employee_t
    WITH OBJECT OID (empno) AS
    SELECT    e.empnum, e.ename, e.salary, e.job
    FROM      emp_table e
    WHERE     job = 'Developer';
```

このオブジェクト・ビューは、ユーザーには、基礎を形成する型が `employee_t` であるオブジェクト表のように見えます。それぞれの行には `employee_t` 型のオブジェクトが含まれており、それぞれの行のオブジェクト識別子は一意です。

Oracle は、指定されたキーに基づいてオブジェクト識別子を構築します。ほとんどの場合、実表の主キーを指定します。ただし、オブジェクト・ビューを定義する問合せに結合が含まれている場合は、オブジェクト・ビューの行を一意に識別できるように、その結合に含まれるすべての表からのキーを指定する必要があります。

注意： WITH OBJECT OID 句の列（この例では `empno`）は、基礎を形成するオブジェクト型（この例では `employee_t`）の属性にもなっている必要があります。これにより、トリガー・プログラムが実表の対応する行を一意に識別しやすくなります。

オブジェクト・ビューの使用法

オブジェクト・ビューの行データが 2 つ以上の表から取られている場合もありますが、そのオブジェクトは 1 回の操作でネットワークを通過します。インスタンスがクライアント側のオブジェクト・キャッシュに入っているときは、プログラマにとってそのインスタンスは C または C++ の構造体、あるいは PL/SQL のオブジェクト変数のように見えます。このインスタンスは、他のネイティブな構造体と同じように処理できます。

SQL 文の中で、オブジェクト・ビューは、オブジェクト表を参照するときと同じ方法で参照できます。たとえば、オブジェクト・ビューは、SELECT リスト、UPDATE-SET 句または WHERE 句に指定できます。

オブジェクト・ビューに対するオブジェクト・ビューを定義することもできます。

オブジェクト表のオブジェクトに使用するときと同じ OCI コールを使用して、クライアント側でオブジェクト・ビューのデータにアクセスできます。たとえば、REF をピンするときに `OCIObjectPin()` を使用し、オブジェクトをサーバーにフラッシュするときに `OCIObjectFlush()` を使用できます。オブジェクト・ビュー内のオブジェクトをサーバーに更新またはフラッシュすると、Oracle はオブジェクト・ビューを更新します。

追加情報： OCI コールの詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

オブジェクト・ビューの更新

オブジェクト・ビューのデータは、オブジェクト表に使用すると同じ SQL DML を使用して更新、挿入および削除できます。あいまいさがなければ、Oracle はオブジェクト・ビューの実表を更新します。

ビューの問合せに、結合、集合演算子、集計関数、GROUP BY または DISTINCT が含まれる場合、そのビューは更新不可能です。ビューの問合せに疑似列、つまり式が含まれている場合、対応するビューの列は更新不可能です。オブジェクト・ビューには、しばしば結合が含まれています。

このような問題を克服するために、Oracle は「INSTEAD OF トリガー」(第 20 章「トリガー」を参照)を提供しています。Oracle は実際の DML 文のかわりにこのトリガー本体を実行するので、これらのトリガーのことを INSTEAD OF トリガーと呼びます。

INSTEAD OF トリガーにより、透過的な方法でオブジェクト・ビューまたはリレーショナル・ビューを更新できます。オブジェクト表の場合と同じ SQL DML (INSERT、DELETE および UPDATE) 文を記述します。Oracle は、SQL 文のかわりに該当するトリガーを起動し、そのトリガー本体に指定されているアクションを実行します。

追加情報： INSTEAD OF トリガーを使用する発注 / 明細項目の詳細は、『Oracle8i 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。

ビュー内でネストした表の列の更新

ネストした表を変更するには、新しい要素を挿入して更新するか、既存の要素を削除します。ビュー内と同様に、ネストした表の列が仮想または合成の場合、通常その列は更新できません。この種の列を更新するために、Oracle ではこれらの列に INSTEAD OF トリガーを作成できます。

(ビューの) ネストした表の列で定義されている INSTEAD OF トリガーは、その列が変更されると起動されます。コレクション全体が(親行の更新によって)置き換えられると、ネストした表の INSTEAD OF トリガーは起動されないので注意してください。

追加情報： ネストした表の列で INSTEAD OF トリガーを使用する発注 / 明細項目の例は、『Oracle8i 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。

数字

- 2 タスク・モード, 8-3
 - ネットワーク通信, 8-23
 - プログラム・インタフェース, 8-23
 - リスナー・プロセス, 8-14
- 2 フェーズ・コミット
 - トランザクションの管理, 17-7
 - 手動による問題解決, 1-35
 - 説明, 1-34, 33-12
 - トリガー, 20-20
 - パラレル DML, 26-37
- 3 値論理 (TRUE、FALSE、UNKNOWN)
 - NULL のために生じる, 10-7

A

- ADMIN OPTION
 - EXECUTE ANY TYPE, 14-13
 - システム権限, 30-3
 - ロール, 30-18
- ADT、オブジェクト型を参照
- Advanced Security, 33-17
- AFTER トリガー, 20-9
 - 起動されるタイミング, 20-20
 - 定義, 20-9
- ALERT ファイル, 8-15
 - ARCn プロセス, 8-12
 - REDO ログ, 8-10
- ALL, 23-6
- ALL_ROWS ヒント, 23-32
- ALL_UPDATABLE_COLUMNS ビュー, 10-15
- ALL_ ビュー, 2-6
- ALTER ANY TYPE 権限, 14-12
 - 権限も参照

- ALTER DATABASE コマンド
 - スタンバイ・データベース, 5-7
- ALTER INDEX コマンド
 - REBUILD PARTITION, 11-59
 - SPLIT PARTITION のためのロギングなしモード, 11-57, 25-7
 - パーティション属性, 11-37
- ALTER SESSION
 - FORCE PARALLEL DDL, 26-22, 26-25
 - CREATE TABLE AS SELECT, 26-23, 26-25
 - 索引の作成または再構築, 26-22, 26-25
 - パーティションの移動または分割, 26-23, 26-25
 - FORCE PARALLEL DML
 - 更新と削除, 26-20, 26-24
 - 挿入, 26-21, 26-24
- ALTER SESSION コマンド, 16-5
 - ENABLE PARALLEL DML, 26-35
 - HASH_JOIN_ENABLED, 24-7
 - OPTIMIZER_GOAL, 23-31
 - SET CONSTRAINTS DEFERRED, 28-21
 - トランザクション分離レベル, 27-8, 27-31
 - 動的パラメータ, 5-4
- ALTER SYSTEM コマンド, 16-5
 - SWITCH LOGFILE オプション, 8-12
 - 動的パラメータ, 5-4
 - LOG_ARCHIVE_MAX_PROCESSES, 8-12, 32-19
- ALTER TABLESPACE コマンド
 - READ ONLY, 3-11
 - READ WRITE, 3-12
 - TEMPORARY または PERMANENT, 3-13
- ALTER TABLE コマンド
 - CACHE 句, 7-4
 - DEALLOCATE UNUSED, 4-14
 - DROP COLUMN, 10-6

EXCHANGE PARTITION, 11-11
MERGE PARTITIONS, 11-16
MODIFY CONSTRAINT, 28-23
SPLIT PARTITION のためのロギングなしモード, 11-57, 25-7
UNUSED 列, 10-6
VALIDATE または NOVALIDATE 制約, 28-21
監査, 31-7
制約を使用禁止または使用可能にする, 28-21
トリガー, 20-6
ハッシュ・パーティションの追加または結合, 11-17
パーティション属性, 11-27
ALTER USER コマンド
一時セグメント, 4-18
ALTER コマンド, 16-4
パーティションの監査, 11-61
ALWAYS_ANTI_JOIN パラメータ, 24-13
ALWAYS_SEMI_JOIN パラメータ, 24-13
ANALYZE コマンド, 16-4
共有プール, 7-11
推定による統計, 22-14
パーティションの統計, 11-15
ヒストグラムの作成, 22-11
ANSI SQL 規格
Oracle が準拠, 1-3
データ型, 12-20
ANSI/ISO SQL 規格, 1-3
データ同時実行性, 27-2
複合外部キー, 28-16
分離レベル, 27-11
ANY, 23-5
AQ
キュー表, 19-4
キュー表のエクスポート, 19-11
キュー・モニター・プロセス, 1-19, 8-13, 19-6
間隔統計, 19-11
実行の時間枠, 19-7
パブリッシュ / サブスクライブのサポート, 19-10
イベントの発行, 20-18
メッセージ・キューイング, 19-2
リモート・データベース, 19-9
レシビエント, 19-5
サブスクライバ・リスト, 19-5
ルールベースのサブスクリプション, 19-5, 19-6
AQ_ADMINISTRATOR ロール, 19-6
AQ_TM_PROCESS パラメータ, 19-6, 19-7

ARCHIVELOG モード
アーカイバ・プロセス (ARCn), 1-18, 8-12, 32-17
概要, 1-47
全体データベース・バックアップ, 32-23
定義, 32-17
部分データベース・バックアップ, 1-49, 32-24
ARCn バックグラウンド・プロセス, 1-18, 8-12
アーカイバ・プロセスも参照
AUDIT コマンド, 16-4
ロック, 27-28

B

B* ツリー索引, 10-26
索引構成表, 10-35
ビットマップ索引と対比, 10-30, 10-31
BEFORE トリガー, 20-9
起動されるタイミング, 20-20
定義, 20-9
BETWEEN, 23-7
BFILE データ型, 12-13
BLOB, 12-12
Block サンプリング, 22-14
BOOLEAN データ型, 12-2
BSP バックグラウンド・プロセス, 27-7
BUFFER_POOL_KEEP パラメータ, 7-5
BUFFER_POOL_RECYCLE パラメータ, 7-5
BUILD_PART_INDEX プロシージャ, 11-30

C

CACHE 句, 7-4
CASCADE アクション
DELETE 文, 28-16
CHARTOROWID 関数, 12-21
CHAR データ型, 12-5
空白埋め比較方法, 12-5
CHECK 制約, 28-17
1 つの列に対する複数の制約, 28-18
一部が NULL の外部キー, 28-16
チェックのメカニズム, 28-19
定義, 28-17
パーティション・ビュー, 11-11
副問合せは指定禁止, 28-17
CHOOSE ヒント, 23-32
CKPT バックグラウンド・プロセス, 1-18, 8-11
CLOB データ型, 12-12

COMMENT コマンド, 16-4
 COMMIT コマンド, 16-5
 2 フェーズ・コミット, 17-7, 33-13
 DDL による暗黙, 17-2, 17-4
 高速コミット, 8-10
 トランザクションの終了, 17-2, 17-4
 パラレル DML での 2 フェーズ・コミット, 26-37
 COMPATIBLE パラメータ
 読み込み専用表領域, 3-12
 CONNECT BY 句
 ビュー問合せの最適化, 23-15
 CONNECT INTERNAL, 5-3
 CONNECT ロール, 30-20
 ユーザー定義型, 14-12, 14-13
 CPU 時間の制限, 29-16
 CREATE ANY TYPE 権限, 14-12
 権限も参照
 CREATE CLUSTER コマンド
 HASHKEYS 句, 10-51, 10-55
 SINGLE TABLE HASHKEYS, 10-55
 CREATE CLUSTETR コマンド
 記憶領域パラメータ, 4-16
 CREATE FUNCTION コマンド, 18-17
 CREATE INDEX コマンド
 一時セグメント, 4-17
 オブジェクト型, 14-6
 記憶領域パラメータ, 4-17
 パーティション属性, 11-37
 パラレル化のルール, 26-22
 ロギングなしモード, 11-57, 25-7
 CREATE OUTLINE 文, 22-7
 CREATE PACKAGE BODY コマンド, 18-11, 18-17
 CREATE PACKAGE コマンド
 パッケージ名, 18-17
 例, 18-11, 20-10
 ロック, 27-28
 CREATE PROCEDURE コマンド
 プロシージャ名, 18-17
 例, 18-6
 ロック, 27-28
 CREATE SYNONYM コマンド
 ロック, 27-28
 CREATE TABLE AS SELECT
 パラレル化のルール
 索引構成表, 26-28
 CREATE TABLESPACE コマンド
 TEMPORARY 句, 3-13
 CREATE TABLE コマンド
 AS SELECT
 意思決定支援システム, 26-28
 一時記憶域, 26-30
 領域の断片化, 26-30
 ダイレクト・ロード・インサートと対比, 25-2
 パラレル化のルール, 26-23
 ロギングなしモード, 11-57, 25-7
 CACHE 句, 7-4
 監査, 31-6, 31-7, 31-9
 記憶領域パラメータ, 4-16
 制約を使用可能または使用禁止にする, 28-21
 トリガー, 20-6
 パーティション属性, 11-27
 パラレル化, 26-28
 索引構成表, 26-28
 例
 オブジェクト表, 13-7, 13-12, 14-5, 14-8
 ネストした表, 13-12
 列オブジェクト, 13-5, 14-8
 ロック, 27-28
 CREATE TEMPORARY TABLESPACE コマンド, 3-13
 CREATE TEMPORARY TABLE コマンド, 10-10
 CREATE TRIGGER コマンド
 コンパイルと格納, 20-23
 例, 20-10, 20-14, 20-22
 オブジェクト表, 14-6
 ロック, 27-28
 CREATE TYPE 権限, 14-12
 権限も参照
 CREATE TYPE コマンド
 VARRAY, 13-11
 オブジェクト型, 13-4, 14-3, 14-4, 14-8
 オブジェクト・ビュー, 15-3
 ネストした表, 13-4, 13-11, 14-4
 不完全な型, 14-15
 CREATE USER コマンド
 一時セグメント, 4-18
 CREATE VIEW コマンド
 例, 20-13
 オブジェクト・ビュー, 15-3
 ロック, 27-28
 CREATE_STORED_OUTLINES セッション・パラメータ, 22-7
 CREATE コマンド, 16-4

D

DANGLING REF, 13-9

DATE データ型, 12-9

深夜, 12-10

デフォルト書式の変更, 12-9

パーティション化, 11-14, 11-21

パーティション・プルニング, 11-22

日付算術, 12-10

ユリウス暦, 12-10

DB_BLOCK_BUFFERS パラメータ

システム・グローバル領域のサイズ, 7-12

バッファ・キャッシュ, 7-4

DB_BLOCK_LRU_LATCHES パラメータ, 8-8

DB_BLOCK_SIZE パラメータ

システム・グローバル領域のサイズ, 7-12

バッファ・キャッシュ, 7-4

DB_FILE_MULTIBLOCK_READ_COUNT パラメータ,
23-49

コストベース最適化, 24-9

DB_FILES パラメータ, 7-15

DB_NAME パラメータ, 32-21

DB_WRITER_PROCESSES パラメータ, 1-17, 8-8

DBA_QUEUE_SCHEDULES ビュー, 19-9

DBA_SYNONYMS.SQL スクリプト

使用方法, 2-6

DBA_UPDATABLE_COLUMNS ビュー, 10-15

DBA_ ビュー, 2-6

DBA ロール, 30-20

ユーザー定義型, 14-12

DBMS, 1-2

一般要件, 1-50

オブジェクト・リレーショナル DBMS, 13-2

DBMS_AQADM パッケージ, 19-4, 19-7

DBMS_AQ パッケージ, 19-4

DBMS_JOB パッケージ, 8-13

Oracle が提供するパッケージ, 18-16

DBMS_LOCK パッケージ, 27-38

Oracle が提供するパッケージ, 18-16

DBMS_PCLXUTIL パッケージ, 11-30

DBMS_RLS パッケージ

セキュリティ・ポリシー, 30-21

定義者権限を使用, 30-8

DBMS_SQL パッケージ, 16-19

DDL 文の解析, 16-19

Oracle が提供するパッケージ, 18-16

DBMS_STATS パッケージ, 22-12

推定による統計, 22-14

パーティションの統計, 11-15

ヒストグラムの作成, 22-11

DBWn バックグラウンド・プロセス, 8-8

データベース・ライター・プロセスも参照

DDL, 1-51, 16-4

データ定義言語も参照

DELETE CASCADE 制約, 28-16

DELETE No Action 制約, 28-16

DELETE コマンド, 16-3

外部キー参照, 28-16

ロギングなしモード

LOB, 25-7

データ・ブロック内の領域の解放, 4-9

トリガー, 20-2, 20-6

パラレル DELETE, 26-19

ロギングなしモード, 25-7

DETERMINISTIC, 23-9

DETERMINISTIC 関数, 23-9

ファンクションベース索引, 21-7

Digital の POLYCENTER Manager on NetView, 33-19

ディレクトリ対応の Oracle Security Manager, 29-5

DISABLED 索引, 21-7, 21-8

DISABLE 制約, 28-21

DISTINCT 演算子

ビューの最適化, 23-15

DISTRIBUTED_TRANSACTIONS パラメータ, 8-12

DML, 1-51, 16-3

データ操作言語も参照

DML サブパーティション・ロック, 11-46

Dnnn バックグラウンド・プロセス, 1-19, 8-14

ディスパッチャ・プロセスも参照

DROP ANY TYPE 権限, 14-12

権限も参照

DROP COLUMN 句, 10-6

DROP TABLE コマンド

監査, 31-6, 31-7

トリガー, 20-6

DROP TYPE コマンド

FORCE オプション, 14-16

依存性, 14-16

DROP コマンド, 16-4

DSS データベース

索引のパーティション化, 11-37

スコア表, 26-34

ディスクのストライプ化, 26-45

パーティション, 11-6

パフォーマンス, 11-8
パラレル DML, 26-34
DUAL 表, 2-7

E

ENABLE 制約, 28-21
Enterprise Manager, 33-18
 ALERT ファイル, 8-15
 PL/SQL, 16-17, 16-18
 SGA のサイズの表示, 7-12
 SQL 文, 16-2
 起動, 5-5
 システム権限の付与, 30-3
 スキーマ・オブジェクト権限, 30-4
 チェックポイント統計, 8-11
 停止, 5-9, 5-10
 統計モニター, 29-18
 パッケージの実行, 18-6
 パラレル回復, 32-11
 分散データベース, 33-18
 プロシージャの実行, 18-4
 ロールの付与, 30-17
 ロックとラッチのモニター, 27-29
EXCHANGE PARTITION, 11-11
EXECUTE ANY TYPE 権限, 14-12, 14-13
 権限も参照
EXECUTE 権限
 権限も参照
 ユーザー・アクセスの検査, 18-18
 ユーザー定義型, 14-13
EXECUTE 権限の GRANT オプション, 14-13
EXECUTE ユーザー定義型, 14-12
EXP_FULL_DATABASE ロール, 30-20
EXPLAIN PLAN コマンド, 16-3
 アクセス・パス, 23-37, 23-38, 23-39, 23-40, 23-41, 23-42, 23-43, 23-44, 23-45, 23-46, 23-47, 23-48
 スター型変換, 24-18
 スター問合せ, 24-16
 パーティション・プルニング, 11-22
Export ユーティリティ, 1-5
 統計のコピー, 22-9
 バックアップでの使用方法, 32-25
 パーティションのメンテナンス操作, 11-47
 ユーザー定義型, 14-19

F

FAST_START_IO_TARGET パラメータ, 32-13
FIPS 規格, 16-6
FIRST_ROWS ヒント, 23-32
FORCE PARALLEL DDL オプション, 26-22, 26-25
 CREATE TABLE AS SELECT, 26-23, 26-25
 索引の作成または再構築, 26-22, 26-25
 パーティションの移動または分割, 26-23, 26-25
FORCE PARALLEL DML オプション
 更新と削除, 26-20, 26-24
 挿入, 26-21, 26-24
FORCE オプション
 オブジェクト型の依存性, 14-16
FOREIGN KEY 制約
 NULL, 28-15
 親キー値の変更, 28-16
 親キー表の更新, 28-16
 親表の行の削除, 28-16
 制約チェック, 28-19
 列の最大数, 28-13

G

GRANT ANY PRIVILEGE システム権限, 30-3
GRANT コマンド, 16-4
 ロック, 27-28
GROUP BY 句
 一時表領域, 3-13
 ビューの最適化, 23-15

H

HASH_AJ ヒント, 24-13
HASH_AREA_SIZE パラメータ, 24-8
HASH_JOIN_ENABLED パラメータ, 24-7
HASH_MULTIBLOCK_IO_COUNT パラメータ, 24-8
HASH_SJ ヒント, 24-13
HASHKEYS パラメータ, 10-51, 10-55
HEXTORAW 関数, 12-21
HI_SHARED_MEMORY_ADDRESS パラメータ, 7-13
HIGH_VALUE 統計, 23-50
HP の OpenView, 33-19

I

IBM の NetView/6000, 33-19

ILMS, 16-19
IMP_FULL_DATABASE ロール, 30-20
Import ユーティリティ, 1-6
 回復での使用方法, 32-25
 統計のコピー, 22-9
 パーティションのメンテナンス操作, 11-47
 ユーザー定義型, 14-19
INDEX_FFS ヒント, 23-34
INDEX_JOIN ヒント, 23-35
INIT.ORA ファイル, 5-4, 5-5
INSERT コマンド, 16-3
 INSERT ... SELECT のパラレル化, 26-21
 空きリスト, 4-9
 ダイレクト・ロード・インサート, 25-2
 ロギングなしモード, 11-57, 25-5, 25-7
 トリガー, 20-2, 20-6
 BEFORE トリガー, 20-9
 パラレル INSERT の記憶領域, 25-8
INSTEAD OF トリガー, 20-11
 オブジェクト・ビュー, 15-5
 ネストした表, 15-5
Inter-Language Method Services (ILMS), 16-19
INTERNAL 接続, 5-3
 監査されない文の実行, 31-4
INTERSECT 演算子
 ビュー問合せの最適化, 23-15
 複合問合せ, 23-3
 例, 23-28
INVALID 状態, 21-2
IN 演算子, 23-5
 ビューのマージ, 23-16
IN 副問合せ, 23-15
IS NULL 述語, 10-7
ISO SQL 規格, 1-3, 12-20
 複合外部キー, 28-16

J

Java
 トリガー, 20-1, 20-7
JOB_QUEUE_PROCESSES パラメータ, 19-9

L

LCK0 バックグラウンド・プロセス, 1-19, 8-13
LGWR バックグラウンド・プロセス, 1-18, 8-9
 ログ・ライター・プロセスも参照

LICENSE_MAX_SESSIONS パラメータ, 29-19
LICENSE_SESSIONS_WARNING パラメータ, 29-19
LIKE, 23-5
LISTENER.ORA ファイル, 6-6
LISTENER.ORA ファイル内の SID, 6-6
LOB データ型, 12-11
 BFILE, 12-13
 BLOB, 12-12
 CLOB および NCLOB, 12-12
 NOLOGGING モード, 25-7
 制限
 パラレル DDL, 26-28
 パラレル DML, 26-41
 デフォルトのロギング・モード, 25-7
LOCK TABLE コマンド, 16-4
LOCK_SGA パラメータ, 7-13, 7-17
LOG_ARCHIVE_MAX_PROCESSES パラメータ,
 1-18, 8-12
 自動アーカイブ, 32-19
LOG_ARCHIVE_START パラメータ, 32-19
LOG_BUFFER パラメータ, 7-6
 システム・グローバル領域のサイズ, 7-12
LOG_CHECKPOINT_INTERVAL パラメータ, 32-13
LOG_CHECKPOINT_TIMEOUT パラメータ, 32-13
LONG RAW データ型, 12-13
 LONG データ型との類似点, 12-13
 索引の設定は禁止, 12-13
 パーティション化の制限事項, 11-14
LONG データ型
 記憶領域, 10-7
 自動的に最後の列になる, 10-7
 定義, 12-6
 パーティション化の制限事項, 11-14
LOW_VALUE 統計, 23-50
LRU, 7-3, 7-4, 8-8
 共有 SQL プール, 7-8, 7-10
 ディクショナリ・キャッシュ, 2-4
 ラッチ, 8-8

M

MAXEXTENTS UNLIMITED 記憶領域パラメータ,
 26-36
MAXVALUE
 パーティション表とパーティション索引, 11-20
MERGE_AJ ヒント, 24-13
MERGE_SJ ヒント, 24-13

MERGE ヒント, 23-16
MINIMUM EXTENT
 パラレル DML, 25-9, 25-10
MINIMUM EXTENT パラメータ, 26-30
MINUS 演算子
 ビュー問合せの最適化, 23-15
 複合問合せ, 23-3
MODIFY CONSTRAINT オプション, 28-23
MOVE PARTITION コマンド
 パラレル化のルール, 26-23
 ロギングなしモード, 11-57, 25-7
MPP
 大量パラレル処理を参照
MTS_MAX_SERVERS パラメータ, 8-19
 人工デッドロック, 8-20
MTS_SERVERS パラメータ, 8-19

N

NCHAR データ型, 12-6
NCLOB データ型, 12-12
Net8, 1-7, 1-36, 6-4, 33-4
 Advanced Security, 33-17
 アプリケーション, 6-5
 概要, 6-4
 クライアント / サーバー・システムでの使用, 6-4
 マルチスレッド・サーバーの要件, 8-14, 8-16
NEXT 記憶領域パラメータ
 パラレル・ダイレクト・ロード・インサート, 25-8
 値の計算, 25-9
NLS
 各国語サポートを参照
NLS_DATE_FORMAT パラメータ, 12-9
NLS_LANGUAGE パラメータ, 11-20
NLS_LANG 環境変数, 11-20
NLS_NUMERIC_CHARACTERS パラメータ, 12-8
NLS_SORT パラメータ
 ORDER BY アクセス・パス, 23-46
 パーティション化キーに影響を与えない, 11-20
NOARCHIVELOG モード, 32-17
 LOGGING モードとの関係, 25-5
 回復用のデータベース・バックアップ, 32-23
 概要, 1-47
 定義, 32-17
NOAUDIT コマンド, 16-4
 ロック, 27-28
NOLOGGING モード

 影響を受ける SQL 操作, 25-7
 ダイレクト・ロード・インサート, 25-5
 パーティション, 11-57
 パラレル DDL, 26-28, 26-29
NOT, 23-7
NOT IN 副問合せ, 24-13
NOT NULL 制約
 PRIMARY KEY による暗黙, 28-12
 UNIQUE キー, 28-11
 制約チェック, 28-19
 定義, 28-7
NOVALIDATE 制約, 28-21
Novell の NetWare Management System, 33-19
NULL
 NULL 以外の値, 10-7, 24-11
 UNIQUE キー制約, 28-11
 UNIQUE キーでの不一致, 28-11
 値に変換, 10-7
 最適化, 24-11
 アトミック, 14-3
 オブジェクト型, 14-3
 格納方法, 10-7
 外部キー, 28-15, 28-16
 禁止, 28-7
 索引, 10-8, 10-23, 10-34
 主キーでは使用禁止, 28-11
 定義, 10-7
 デフォルト値, 10-8
 パーティション表とパーティション索引, 11-21
 比較では不明扱い, 10-7
 列の順序, 10-7
NUM_DISTINCT 列
 USER_TAB_COLUMNS ビュー, 23-50
NUM_ROWS 列
 USER_TABLES ビュー, 23-50
NUMBER データ型, 12-7
 内部形式, 12-8
 丸め, 12-8
NVARCHAR2 データ型, 12-6
NVL 関数, 10-7

O

OCI, 8-25
 OCIObjectFlush, 15-4
 OCIObjectPin, 15-4
 オブジェクト・キャッシュ, 13-14

- ストアド・プロシージャ, 16-18
- バインド変数, 16-12
- 無名ブロック, 16-17
- ODCIIndex, 10-41
- OID, 14-17, 15-3, 15-4
 - WITH OBJECT OID 句, 15-3, 15-4
 - コレクション
 - キー圧縮, 10-29, 10-36
- OLTP データベース, 11-5
 - 索引のパーティション化, 11-36
 - バッチ・ジョブ, 26-35
 - パーティション, 11-6
 - パラレル DML, 26-34
- OPEN_CURSORS パラメータ, 16-6
 - プライベート SQL 領域の管理, 7-9
- OPEN_LINKS パラメータ, 7-15
- OPTIMAL 記憶領域パラメータ, 4-24
- OPTIMIZER_FEATURES_ENABLE パラメータ,
 - 23-16, 23-34, 23-35, 24-12
- OPTIMIZER_GOAL オプション, 23-31
- OPTIMIZER_MODE, 23-30
 - 影響を与えるヒント, 23-32
- OPTIMIZER_PERCENT_PARALLEL パラメータ, 22-8
- Oracle
 - Oracle Server, 1-4
 - Parallel Server オプション, 1-8
 - Parallel Server も参照
 - SQL 処理, 16-8
 - アーキテクチャ, 1-8, 1-13
 - 移植性, 1-3
 - インスタンス, 1-6, 1-15, 5-2
 - 拡張性, 6-4
 - 機能, 1-2
 - クライアント / サーバー・アーキテクチャ, 6-2
 - 構成, 8-2
 - マルチ・プロセス Oracle, 8-2
 - 互換性, 1-3
 - 互換性レベル, 3-15
 - 準拠する規格, 1-3
 - 整合性制約, 28-5
 - 接続性, 1-2
 - 異なる Oracle バージョン, 33-6
 - データ・アクセス, 1-50
 - 動作例, 1-20
 - 専用サーバー, 8-23
 - マルチスレッド・サーバー, 8-20
 - ネットワークでの使用, 1-2, 1-36
 - プロセス, 1-16, 8-5
 - ライセンス, 29-18
- Oracle AQ, 19-1
 - キュー表, 19-4
 - キュー表のエクスポート, 19-11
 - キュー・モニター・プロセス, 1-19, 8-13, 19-6
 - 間隔統計, 19-11
 - 実行の時間枠, 19-7
 - パブリッシュ / サブスクライブのサポート, 19-10
 - イベントの発行, 20-18
 - メッセージ・キューイング, 19-2
 - リモート・データベース, 19-9
 - レシipient, 19-5
 - サブスクライバ・リスト, 19-5
 - ルールベースのサブスクリプション, 19-5, 19-6
- Oracle Data Cartridge Interface, 10-41
- Oracle Enterprise Manager
 - Enterprise Manager を参照
- Oracle Forms
 - PL/SQL, 16-16
 - オブジェクト依存性, 21-12
- Oracle Internet Directory, 29-5
- Oracle Names
 - グローバルなディレクトリ・サービス, 33-4
- Oracle Open Gateways, 33-8
- Oracle Parallel Server, 1-8
 - Parallel Server も参照
- Oracle Replication Manager, 34-15
- Oracle Security Manager, 29-5, 33-17
- Oracle Server, 1-4
 - Oracle も参照
- Oracle Type Translator (OTT), 13-14
- Oracle Wallet Manager, 29-5
- Oracle XA
 - 大規模プール内のセッション・メモリー, 7-11
- Oracle Wallet, 29-5
- Oracle コード, 8-2, 8-24
- Oracle コール・インタフェース (OCI), 8-25
 - OCIObjectFlush, 15-4
 - OCIObjectPin, 15-4
 - オブジェクト・キャッシュ, 13-14
 - ストアド・プロシージャ, 16-18
 - バインド変数, 16-12
 - 無名ブロック, 16-17
- Oracle 認証局, 29-5
- Oracle ブロック, 1-10, 4-2
 - データ・ブロックも参照

Oracle プリコンパイラ
 FIPS フラガー, 16-6
 埋込み SQL, 16-5
 カーソル, 16-11
 ストアド・プロシージャ, 16-18
 バインド変数, 16-12
 無名ブロック, 16-17
Oracle プログラム・インタフェース (OPI), 8-25
ORDBMS, 1-21, 13-2
ORDERED ヒント, 24-10
OTT, 13-14
OUTLN スキーマ
 DBA 権限, 22-7

P

Parallel Server, 1-8
 DML ロックとパフォーマンス, 11-47
 PCM ロック, 27-20
 一時表領域, 3-13
 インスタンス・グループ, 26-17
 共有モード
 ロールバック・セグメント, 4-27
 逆キー索引, 10-29
 システム変更番号, 8-10
 システム・モニター・プロセス, 8-11, 26-38
 ディスクの親和性, 26-44
 データベースとインスタンス, 5-3
 データベースのマウント, 5-6
 同時実行の制限, 29-20
 名前付きユーザー・ライセンス, 29-20
 排他モード
 ロールバック・セグメント, 4-27
 パラレル SQL, 26-1
 ファイルとログの管理ロック, 27-30
 分散ロック, 27-20
 分離レベル, 27-12
 読み込み一貫性, 27-7
 ロック・プロセス, 1-19, 8-13
PARALLEL SERVER パラメータ, 5-6
PARALLEL_INDEX ヒント, 26-15
PARALLEL_MAX_SERVERS パラメータ, 26-8
PARALLEL_MIN_PERCENT パラメータ, 26-17
PARALLEL_MIN_SERVERS パラメータ, 26-7, 26-8
PARALLEL 句
 パラレル化ルール, 26-18
PARALLEL ヒント, 26-15

 UPDATE と DELETE, 26-19
 パラレル化ルール, 26-18
PARTITION_VIEW_ENABLED パラメータ, 11-13
PARTITION オプション, 11-62
PCTFREE 記憶領域パラメータ
 PCTUSED, 4-7
 仕組み, 4-6
PCTINCREASE 記憶領域パラメータ
 パラレル DML, 25-8, 25-9
PCTUSED 記憶領域パラメータ
 PCTFREE, 4-7
 仕組み, 4-7
PGA, 1-16, 7-13
 マルチスレッド・サーバー, 8-19
PKI, 29-5
PL/SQL, 16-14
 DDL 文の解析, 16-19
 PL/SQL エンジン, 16-15, 18-2
 コンパイラ, 18-16
 製品, 16-16
 プロシージャの実行, 18-18
 オブジェクト・ビュー, 15-4
 オブティマイザの目標, 23-32
 解析ロック, 27-29
 拡張パーティション表名, 11-63
 外部プロシージャ, 16-19, 18-11
 概要, 1-54, 16-14
 言語要素, 16-17
 実行, 16-15, 18-17, 18-18
 ストアド・プロシージャ, 1-24, 16-15, 18-2, 18-6
 データ型, 12-2
 データベース・トリガー, 1-58, 20-1
 動的 SQL, 16-19
 バインド変数
 ユーザー定義型, 13-13
 パッケージ, 18-4, 18-11
 文の監査, 31-4
 プログラム・ユニット, 1-24, 7-9, 16-14, 18-2
 共有 SQL 領域, 7-9
 コンパイル済み, 16-16, 18-9, 18-16
 プロシージャ内のロール, 30-18
 無名ブロック, 16-14, 18-9
 ユーザー定義データ型, 13-13
 ユーザー・ロック, 27-38
 例外処理, 16-18
PMON バックグラウンド・プロセス, 1-18, 8-11
 プロセス・モニター・プロセスも参照

Point-in-Time 回復
クローン・データベース, 5-7
PRIMARY KEY 制約, 28-11
暗黙の NOT NULL 制約, 28-12
施行に索引が使用される, 28-12
名前, 28-12
制約チェック, 28-19
説明, 28-11
列の最大数, 28-12
Pro*C/C++
SQL 文の処理, 16-10
ユーザー定義データ型, 13-13
PUBLIC ユーザー・グループ, 29-14, 30-18
プロシージャの有効性, 18-18
PUSH_JOIN_PRED ヒント, 24-12
P コード, 18-17

Q

QMN_n バックグラウンド・プロセス, 1-19, 8-13, 19-6
間隔統計, 19-11
実行の時間枠, 19-7

R

RADIUS, 29-6
RAWTOHEX 関数, 12-21
RAW データ型, 12-13
RDBMS, 1-21
Oracle も参照
オブジェクト・リレーショナル DBMS, 1-21, 13-2
READ ONLY オプション
ALTER TABLESPACE, 3-11
READ WRITE オプション
ALTER TABLESPACE, 3-12
REBUILD INDEX PARTITION コマンド, 11-59
パラレル化のルール, 26-22
ロギングなしモード, 25-7
REBUILD INDEX コマンド
パラレル化のルール, 26-22
ロギングなしモード, 11-57, 25-7
Recovery Manager, 1-50, 32-14
カタログを使用しない操作, 32-15
パラレル回復, 32-11
パラレル操作, 32-16
リカバリ・カタログ, 32-15

レポートの生成, 32-16
REDO エントリ, 1-12, 32-9
REDO レコード, 1-12
REDO ログ, 1-12, 32-9
アーカイブ・モード, 32-17
ロールフォワード
インスタンス障害, 32-4
REDO ログ・エントリ
コミットされたデータ, 32-8, 32-9
コミットされていないデータ, 32-9
REDO ログ・バッファ, 1-15, 7-5
書込み, 8-9
サイズ, 7-6
循環, 8-9
トランザクションのコミット, 8-10
ログ・ライター・プロセス, 7-6
REDO ログ・ファイル, 1-12, 32-7
REDO エントリ, 1-12, 32-9
アーカイバ・プロセス (ARC_n), 1-18, 8-12
アーカイブ済み, 1-47, 32-17
アーカイブ時のエラー, 32-19
手動, 32-19
自動, 32-19
一時セグメントが関係する場合, 4-18
オンラインまたはオフライン, 1-47, 32-7
回復, 32-7
概要, 1-12, 1-46
制御ファイルに名前がある, 32-20
多重化, 1-47
用途, 1-12
トランザクションのコミット前の書込み, 8-10
バッファ管理, 8-9
パラレル回復, 32-10
物理データベース構造, 1-5
モード, 1-47
ロールフォワード, 32-9
ログ順序番号, 1-47
制御ファイルに記録される, 32-21
ログ・スイッチ
ALTER SYSTEM SWITCH LOGFILE, 8-12
アーカイバ・プロセス, 1-18, 8-12
ログ・ライター・プロセス, 8-9
REF
DANGLING, 13-9
暗黙的な解除, 13-9
オブジェクト識別子からの組立て, 14-17, 14-18
オブジェクト・ビューの行, 15-3

- サイズ, 14-18
- 索引, 14-6
- 参照解除, 13-9
- 相互に依存する型, 14-15
- 表の別名の使用, 14-8
- ピン, 14-14, 15-4
- 有効範囲付き, 13-9, 14-18
- REFERENCES 権限
 - ロールを介して付与された場合, 30-19
- REFTOHEX 関数, 12-21
- REMOTE_DEPENDENCIES_MODE パラメータ, 21-10
- RENAME コマンド, 16-4
- Replication Manager, 34-15
- RESOURCE ロール, 30-20
 - ユーザー定義型, 14-12, 14-13
- RESTRICTED SESSION 権限, 29-19
- REVOKE コマンド, 16-4
 - FORCE オプション, 14-16
 - オブジェクト型と依存性, 14-16
 - ロック, 27-28
- ROLLBACK コマンド, 16-5
- ROWID, 10-7
 - Oracle 以外のデータベース, 12-20
 - アクセス, 12-14
 - クラスタ化された行, 10-7
 - 索引のソートでの使用, 10-27
 - 内部使用, 12-14, 12-18
 - 汎用, 12-14
 - 表アクセス, 23-33
 - 物理, 12-14
 - 変更, 12-15
 - 論理, 12-14
 - 論理 ROWID, 12-18
 - 索引構成表の索引, 10-37
 - 推測の陳腐化, 12-19
 - 推測の統計, 12-19
 - 物理推測, 10-37, 12-18
- ROWIDTOCHAR 関数, 12-21
- ROWID データ型, 12-14
 - 拡張 ROWID 形式, 12-15
 - 制限付き ROWID 形式, 12-16
- ROWNUM 疑似列
 - 索引を使用できない, 23-47
 - ビュー問合せの最適化, 23-15, 23-24
- ROW サンプルング, 22-14
- RPC, 33-11

- RULE ヒント
 - OPTIMIZER_MODE, 23-32

S

- SAMPLE BLOCK オプション, 23-33
 - アクセス・パス, 23-47
 - ヒントで上書きできない, 23-49
- SAMPLE オプション, 23-33
 - アクセス・パス, 23-47
 - ヒントで上書きできない, 23-49
- SAMPLE 句
 - コストベース最適化, 22-16
- SAVEPOINT コマンド, 16-5
- SCN, 17-5
 - システム変更番号も参照
- SELECT コマンド, 16-3
 - SAMPLE オプション, 23-33
 - アクセス・パス, 23-47, 23-49
- SAMPLE 句
 - コストベース最適化, 22-16
 - 問合せも参照
 - 複合索引, 10-22
 - 副問合せ, 16-12
- Server Manager
 - PL/SQL, 16-17, 16-18
 - SQL 文, 16-2
- SERVICE_NAMES パラメータ, 6-6
- SESSION_ROLES ビュー
 - PL/SQL ブロックからの問合せ, 30-19
- SET CONSTRAINTS コマンド
 - DEFERRABLE または IMMEDIATE, 28-20
- SET ROLE コマンド, 16-5
- SET TRANSACTION コマンド, 16-5
 - ISOLATION LEVEL, 27-8, 27-31
 - READ ONLY, 4-20
- SGA
 - システム・グローバル領域を参照
- SHARED_MEMORY_ADDRESS パラメータ, 7-13
- SHARED_POOL_SIZE パラメータ, 7-6
 - システム・グローバル領域のサイズ, 7-12
- SHUTDOWN ABORT コマンド, 5-10
 - 必要なクラッシュ回復, 32-4
- SINGLE TABLE HASHKEYS, 10-55
- SKIP_UNUSABLE_INDEXES パラメータ, 21-8
- SMON バックグラウンド・プロセス, 1-18, 8-11
 - システム・モニター・プロセスも参照

- SMP アーキテクチャ
 - ディスクの親和性, 26-45
- SNMP サポート
 - データベース管理, 33-19
- Snnn* バックグラウンド・プロセス, 8-14
- SNPn* バックグラウンド・プロセス, 1-19, 8-13
 - メッセージの伝播, 19-9
- SOME, 23-5
- SORT_AREA_RETAINED_SIZE* パラメータ, 7-16
- SORT_AREA_SIZE* パラメータ, 4-18, 7-16
 - コストベース最適化, 24-9
- SPLIT PARTITION コマンド
 - パラレル化のルール, 26-23
 - ロギングなしモード, 11-57, 25-7
- SQL, 16-2
 - PL/SQL, 1-54, 16-14
 - 埋込み, 1-52, 16-5
 - ユーザー定義データ型, 13-13
 - カーソル, 16-6
 - 解析, 16-7
 - 拡張要素
 - パーティション名またはサブパーティション名, 11-62
- 関数, 16-2
 - CHECK 制約, 28-17
 - COUNT, 10-34
 - NVL, 10-7
 - 列のデフォルト値, 10-8
- 概要, 1-51, 16-2
- 共有 SQL, 16-7
- 再帰, 16-6
 - カーソル, 16-6
- システム制御文, 16-5
- セッション制御文, 16-5
- データ操作言語 (DML), 16-3
- データ定義言語 (DDL), 16-4
- トランザクション, 1-52, 17-2, 17-5
- トランザクション制御文, 16-5
- 動的 SQL, 16-19
- パラレル実行, 26-2
- 文のタイプ, 1-51, 16-3
 - 最適化, 23-4
- 文レベルのロールバック, 17-4
- メモリー割当て, 7-10
- ユーザー定義データ型, 13-12, 14-7
 - OCI, 13-14
 - 埋込み SQL, 13-13
- 予約語, 16-3
- 関数
 - ビュー問合せの最適化, 23-22
- SQL*Loader, 1-6
 - ダイレクト・ロード
 - NOLOGGING モード, 11-57, 25-7
 - ダイレクト・ロード・インサートに類似, 25-2
 - パラレル・ダイレクト・ロード, 25-2
 - パーティション操作, 11-47, 11-49
- SQL*Menu
 - PL/SQL, 16-16
- SQL*Module
 - FIPS フラガー, 16-6
 - ストアド・プロシージャ, 16-18
- SQL*Net
 - Net8 を参照
- SQL*Plus
 - ALERT ファイル, 8-15
 - SGA のサイズの表示, 7-12
 - SQL 文, 16-2
 - ストアド・プロシージャ, 16-18
 - セッション変数, 16-17
 - 接続, 29-4
 - 統計モニター, 29-18
 - パッケージの実行, 18-6
 - パラレル回復, 32-11
 - プロシージャの実行, 18-4
 - 無名ブロック, 16-17
 - ロックとラッチのモニター, 27-29
- SQL_TRACE パラメータ, 8-15
- SQL92, 27-2
- SQL 文, 1-51, 16-3, 16-8
 - 1 つの SQL 文で起動されるトリガーの数, 20-20
 - 依存オブジェクトの参照, 21-4
 - 埋込み, 16-5
 - カーソルの作成, 16-11
 - 解析, 16-11
 - 解析ロック, 27-29
 - 監査, 31-6, 31-9
 - 概要, 1-43
 - レコードが生成される場合, 31-4
 - 概要, 1-51
 - 再帰
 - OPTIMIZER_GOAL は影響を与えない, 23-31
 - 最適化
 - 複合文, 23-12
 - 文のタイプ, 23-4

- 障害, 32-2
- 実行, 16-8, 16-13
- 実行計画, 22-2
- 正常な実行, 17-3
- タイプ, 1-51, 16-3, 23-4
- 単純, 23-3
- ディクショナリ・キャッシュ・ロック, 27-30
- トランザクション, 16-14
- トリガー, 20-2, 20-8
 - トリガー・イベント, 20-6
- 配列処理, 16-13
- ハンドル, 1-16
- パラレル化, 26-2, 26-10
- パラレル実行, 26-2
- 必要な権限, 30-3
- 複合, 23-3, 23-12
 - 最適化, 23-12
- 分散
 - 最適化, 23-29
 - 定義, 23-4, 33-10
 - ノードへのルーティング, 16-11
- 変換
 - 例, 23-9
- リソース制限, 29-16
- リモート
 - 定義, 23-4, 33-10
- SQL 文のハンドル, 1-16, 7-9
- SQL 領域
 - 共有, 1-15, 7-8, 16-7
 - プライベート, 7-8
 - 持続, 7-8
 - 実行時, 7-8
- STAR_TRANSFORMATION_ENABLED パラメータ, 24-19
- STAR_TRANSFORMATION ヒント, 24-19
- STARTUP FORCE コマンド
 - 必要なクラッシュ回復, 32-4
- STAR ヒント, 24-15
- STORAGE 句
 - 使用, 4-11
 - パラレル実行, 26-30
- SUBPARTITION オプション, 11-62
- SunSoft の SunNet Manager, 33-19
- SYS.AUD\$ ビュー
 - 削除, 2-5
- SYSDBA 権限, 5-3
- SYSOPER 権限, 5-3

- SYSTEM 表領域, 3-7
 - オンライン要件, 3-9
 - 格納されているデータ・ディクショナリ, 2-2, 2-5, 3-7
 - 格納されるプロシージャ, 3-7, 18-17
 - データ・ファイル 1, 3-16
 - メディア障害, 32-6
- SYSTEM ユーザー名
 - セキュリティ・ドメイン, 29-3
- SYSTEM ロールバック・セグメント, 4-25
- SYS ユーザー名
 - V\$ ビュー, 2-7
 - 監査されない文の実行, 31-4
 - 所有する一時スキーマ・オブジェクト, 29-14
 - セキュリティ・ドメイン, 29-3
 - データ・ディクショナリ表の所有, 2-3

T

- TAF, 32-14
- TO_CHAR 関数
 - CHECK 制約のデフォルト NLS, 28-17
 - データ変換, 12-21
 - ビューのデフォルト NLS, 10-14
 - ユリウス暦, 12-10
- TO_DATE 関数, 12-9
 - CHECK 制約のデフォルト NLS, 28-17
 - データ変換, 12-21
 - パーティション, 11-14, 11-21
 - ビューのデフォルト NLS, 10-14
 - ユリウス暦, 12-10
- TO_NUMBER 関数, 12-8
 - CHECK 制約のデフォルト NLS, 28-17
 - データ変換, 12-21
 - ビューのデフォルト NLS, 10-14
 - ユリウス暦, 12-10
- TRANSACTIONS_PER_ROLLBACK_SEGMENT パラメータ, 4-26
- TRANSACTIONS パラメータ, 4-26
- TRUNCATE コマンド, 16-4

U

- UNION ALL 演算子
 - OR からの変換, 23-10
 - ビュー問合せの最適化, 23-15
 - 例, 23-10, 23-12, 23-26

UNION ALL ビュー , 11-11
UNION 演算子
 ビュー問合せの最適化 , 23-15
 複合問合せ , 23-3
 例 , 23-17 , 23-27
UNIQUE キー制約 , 28-8
 NOT NULL 制約 , 28-11
 NULL , 28-11
 サイズの制限 , 28-10
 施行に索引が使用される , 28-10
 制約チェック , 28-19
 複合キー , 28-9 , 28-11
 列の最大数 , 28-10
UNLIMITED エクステンツ , 26-36
UNUSED 索引
 ファンクションベース , 21-8
UNUSED 列 , 10-6
UPDATE No Action 制約 , 28-16
UPDATE コマンド , 16-3
 外部キー参照 , 28-16
 データ・ブロック内の領域の解放 , 4-9
 トリガー , 20-2 , 20-6
 BEFORE トリガー , 20-9
 パラレル UPDATE , 26-19
 ロギングなしモード , 25-7
 LOB , 25-7
UROWID データ型 , 12-14
USE_INDIRECT_DATA_BUFFERS パラメータ , 7-13
USE_STORED_OUTLINES セッション・パラメータ , 22-7
USER_TAB_COL_STATISTICS ビュー , 23-50
USER_TAB_COLUMNS ビュー , 23-50
USER_TABLES ビュー , 23-50
USER_UPDATABLE_COLUMNS ビュー , 10-15
USER_ ビュー , 2-6
USER 疑似列 , 30-6

V

V_\$ ビューと VS\$ ビュー , 2-7
 VSLICENSE , 29-20
VALIDATE 制約 , 28-21
VALUES LESS THAN 句 , 11-20
 DATE データ型 , 11-21
 MAXVALUE , 11-21 , 11-23
 複数列からなるキー , 11-23
 例 , 11-16 , 11-18

VARCHAR2 データ型 , 12-5
 RAW データ型との類似点 , 12-13
 非空白埋め比較方法 , 12-5
VARCHAR データ型 , 12-5
VARRAY , 13-10
 索引構成表 , 10-36
 キー圧縮 , 10-29
VLDB
 パーティション , 11-5
 パラレル SQL , 26-2

W

Wallet , 29-5
Wallet Manager , 29-5
WITH OBJECT OID 句 , 15-3 , 15-4

X

X.509 証明書 , 29-5
XA
 大規模プール内のセッション・メモリー , 7-11

あ

アーカイバ・プロセス (ARC_n)
 手動アーカイブに使用しない , 32-20
 自動アーカイブ , 32-19
 説明 , 1-18 , 8-12
 トレース・ファイル , 32-19
 マルチ・プロセス , 1-18 , 8-12
 例 , 32-18
アーカイブ済み REDO ログ , 1-47
 使用可能にする , 32-17
 手動アーカイブ , 32-19
 自動アーカイブ , 32-19
アーキテクチャ
 MPP , 26-45
 Oracle , 1-13
 SMP , 26-45
 クライアント / サーバー , 1-32
空きリスト , 4-9
空き領域
 空きリスト , 4-9
 エクステンツの結合 , 4-13
 SMON プロセス , 1-18 , 8-11
 データ・ブロック内での結合 , 4-9

- データ・ブロックのセクション, 4-5
- データ・ブロック用のパラメータ, 4-5
- 空き領域の結合
 - エクステント, 4-13
 - SMON プロセス, 1-18, 8-11
 - データ・ブロック内, 4-9
- アクセス制御, 30-2
 - 権限, 30-2
 - 任意, 1-38
 - パスワード暗号化, 29-7
 - ファイン・グレイン・アクセス・コントロール, 30-21
 - ルール, 30-15
- アクセス・パス
 - ROWID による単一行アクセス, 23-37
 - 一意キーまたは主キーによる単一行アクセス, 23-39
 - クラスタ結合, 23-39
 - クラスタ結合による単一行アクセス, 23-37
 - 最適化, 23-32
 - 索引クラスタ・キー, 23-40
 - 定義, 22-5
 - ハッシュ・クラスタ・キー, 23-40
 - ハッシュ・クラスタ・キー（一意キーの指定付き）による単一行アクセス, 23-38
 - 複合索引, 23-41
 - リスト, 23-35
- アクセス方法, 23-32
 - クラスタ走査, 23-33
 - 索引走査, 23-34
 - 実行計画, 22-2
 - ハッシュ走査, 23-33
 - 表走査, 23-32
- 「アクセスを逐次化できません。», 27-11
- 圧縮、索引キー, 10-28
- アトミック NULL, 14-3
- アドバンスト・キューイング（Oracle AQ）, 19-1
 - キュー表, 19-4
 - キュー表のエクスポート, 19-11
 - キュー・モニター・プロセス, 1-19, 8-13, 19-6
 - 間隔統計, 19-11
 - 実行の時間枠, 19-7
 - パブリッシュ / サブスクライブのサポート, 19-10
 - イベントの発行, 20-18
 - メッセージ・キューイング, 19-2
 - リモート・データベース, 19-9
 - 例外処理, 19-11
 - レシピエント, 19-5
 - サブスクライバ・リスト, 19-5
 - ルールベースのサブスクリプション, 19-5, 19-6
- アドバンスト・レプリケーション
 - 使用方法, 34-6, 34-12
 - 同期伝播, 34-16
 - ハイブリッド構成, 34-13
 - プロシージャ・レプリケーション, 34-16
 - マルチマスター構成, 34-6
- アプリケーション
 - アプリケーション・トリガーとデータベース・トリガー, 20-3
 - 意思決定支援システム（DSS）, 10-31
 - パラレル SQL, 26-2, 26-28
 - 依存性, 21-10
 - オブジェクト依存性, 21-12
 - オンライン・トランザクション処理（OLTP）
 - 逆キー索引, 10-30
 - オンライン分析処理（OLAP）, 10-40
 - 空間データ・アプリケーション, 10-40
 - コードの共有, 7-17
 - 索引構成表, 10-38
 - 情報検索（IR）, 10-39
 - 制約違反を検出可能, 28-6
 - セキュリティ
 - アプリケーション・コンテキスト, 30-23
 - セキュリティの強化, 1-41, 28-5
 - ダイレクト・ロード・インサート, 26-35
 - データ・ウェアハウジング, 10-31
 - スター問合せ, 24-14
 - データ・ディクショナリの参照, 2-4
 - データベース・アクセス, 8-2
 - トランザクションの終了, 17-5
 - ネットワーク通信, 6-5
 - パラレル DML, 26-34
 - プログラム・インタフェース, 8-24
 - プロセス, 8-4
 - 離散トランザクション, 17-8
 - ルール, 30-16
- アプリケーション・コンテキスト, 30-23
- アプリケーション負荷の分散
 - アドバンスト・レプリケーションと同様, 34-7
- アンチ・ジョイン, 24-13
- 暗黙的な参照解除, 13-9

い

- 異機種間サービス, 33-8
 - エージェント, 33-8
- 異機種間分散データベース, 33-8
- 意思決定支援アプリケーション
 - 基本レプリケーション, 34-11
- 意思決定支援システム (DSS), 11-5
 - スコア表, 26-34
 - ディスクのストライプ化, 26-45
 - パーティション, 11-5
 - パフォーマンス, 11-8, 26-34
 - パラレル DML, 26-34
 - パラレル SQL, 26-2, 26-28, 26-34
 - ビットマップ索引, 10-31
 - マテリアライズド・ビュー, 10-17
- 移植性, 1-3
- 依存性, 21-1
 - Oracle Forms トリガー, 21-12
 - オブジェクト型の定義, 14-15, 14-16
 - 管理, 21-1
 - 共有ブール, 21-9
 - 権限, 21-6
 - スキーマ・オブジェクト間, 21-2
 - 存在しない参照オブジェクト, 21-8
 - 存在しない他のオブジェクト, 21-8
 - ファンクションベース索引, 10-25, 21-7
 - リモート・オブジェクト, 21-10
 - ローカル, 21-10
- 一意キー, 1-57, 28-9
 - 検索, 23-39
 - 最適化, 23-13
- 一意索引, 10-22
- 一時セグメント, 4-15, 4-18, 10-10
 - REDO ログに記録されない場合, 4-18
 - エクステンツの割当て解除, 4-15
 - 削除, 4-15
 - 操作, 4-17
 - 問合せ用の割当て, 4-18
 - パラレル DDL, 26-30
 - パラレル INSERT, 25-8
 - 表領域, 4-15, 4-18
 - 割当て, 4-18
 - 割当て制限は無視される, 29-14
- 一時表, 10-10
- 一時表領域, 3-12
- 位置の透過性, 1-34

- インスタンス, 1-6
 - 異常終了, 5-9, 32-4
 - インスタンス・グループ, 26-17
 - 回復, 5-10, 32-4
 - SMON プロセス, 8-11
 - データベースのオープン, 5-8
 - ファースト・スタート・チェックポイント, 32-13
 - 仮想メモリー, 7-17
- 概要, 1-6
- 起動, 5-5
 - 監査レコード, 31-5
- サービス名, 6-6
- システム識別子 (SID), 6-6
- 障害, 1-45, 32-4
- 制限モード, 5-5
- 説明, 5-2
- 定義, 1-15
- 停止, 5-9, 5-10
 - 監査レコード, 31-5
- データベースに対応付け, 5-2, 5-6
- データベースの共有, 1-8
- プロセスの構造, 8-2
- マルチ・プロセス, 8-2
- メモリー構造, 7-2
- ロールバック・セグメントの取得, 4-26

- インスタンス・グループ、パラレル操作, 26-17
- インスタンスの異常終了, 5-9, 32-4
- インスタンスの回復, 32-4
 - SMON プロセス, 1-18, 8-11, 26-38
 - クラッシュ回復も参照
- インスタンス障害, 1-45, 32-4
- 読み込み専用表領域, 32-6
- インターオペレータ並行性, 26-12
- インダウト・トランザクション, 4-24, 5-8
- イントラオペレータ並行性, 26-12
- インライン・ビュー, 10-16
 - 例, 10-16

う

- ウェアハウス
 - データ・ウェアハウジングも参照
 - 表データのリフレッシュ, 26-34
 - マテリアライズド・ビュー, 10-17
- 内側獲得, 14-8
- 埋込み SQL 文, 1-52, 16-5

PL/SQL の動的 SQL, 16-19

え

永続キューイング, 19-2

エクステント

管理, 4-11

概要, 4-10

結合, 4-14

増分, 4-10

定義, 4-3

ディクショナリ管理, 3-8

データ・ブロックの集合, 4-10

データ・ブロックの割当て, 4-12

パラレル DDL, 26-30

パラレル INSERT

記憶領域パラメータ, 25-8

マテリアライズド・ビュー, 4-15

ローカル管理, 3-9

ロールバック・セグメント内

現行の変更, 4-22

ロールバック・セグメントの削除, 4-24

ロールバック・セグメントへの割当て

セグメント作成後, 4-24

セグメント作成時, 4-21

割当て, 4-12

割当て解除

実行される時期, 4-13

ロールバック・セグメント, 4-24

割当て方法, 4-12

エクステントの結合, 4-14

エクステントの割当て解除, 4-13

エラー

埋込み SQL, 16-5

トレース・ファイルに記録, 8-15

お

応答キュー, 8-17

応答時間, 22-8

コストベースのアプローチ, 23-30

オフライン REDO ログ・ファイル, 1-47, 32-7

オフライン・バックアップ

全体データベース・バックアップ, 32-22

オブジェクト

権限, 30-10

オブジェクト型, 1-21, 13-2, 13-3

Oracle Type Translator, 13-14

オブジェクト・ビュー, 10-16

キャッシュでのロック, 13-14

行オブジェクト, 13-8

コンストラクタ・メソッド, 1-56, 13-6, 14-17

制限

パラレル DDL, 26-28

パラレル DML, 26-41

相互に依存, 14-15

属性, 13-2, 13-3, 13-4

発注の例, 13-2, 13-4

パラレル問合せ, 26-27

制限, 26-27

比較メソッド, 13-6

表の別名の使用, 14-8

不完全, 14-16

メソッド, 1-55, 13-4

PL/SQL, 13-13

発注の例, 13-2, 13-5

メソッド・コール, 14-9

メッセージ・キューイング, 19-6

列オブジェクト, 13-8

索引, 14-6

オブジェクト型のコンパイル, 14-15

オブジェクト型の属性, 13-2, 13-3, 13-4

オブジェクト型のメソッド, 1-55, 13-4

PL/SQL, 13-13

空のカッコの使用, 14-9

コンストラクタ・メソッド, 1-56, 14-17

自己参照的なスタイルによる起動, 13-6

実行権限, 14-12

順序メソッド, 1-56, 13-7

発注の例, 13-2, 13-5

マップ・メソッド, 1-56, 13-6

オブジェクト・キャッシュ

OCI, 13-14

Pro*C, 13-13

オブジェクト・ビュー, 15-4

権限, 14-14

オブジェクト権限, 30-3

スキーマ・オブジェクト権限も参照

オブジェクト識別子, 15-3

WITH OBJECT OID 句, 15-3, 15-4

オブジェクト型, 14-17

オブジェクト・ビュー, 15-3, 15-4

オブジェクト表, 13-2, 13-7

仮想的なオブジェクト表, 15-2

- 行オブジェクト, 13-8
- 索引, 14-6
- 制約, 14-5
- トリガー, 14-6
- オブジェクト表の DELETE 権限, 14-14, 14-15
- オブジェクト表の INSERT 権限, 14-14, 14-15
- オブジェクト表の SELECT 権限, 14-13, 14-14
- オブジェクト表の UPDATE 権限, 14-14, 14-15
- オブジェクト・ビュー, 10-16, 15-1
 - INSTEAD OF トリガーの使用方法, 15-5
 - オブジェクト識別子, 15-3, 15-4
 - 更新, 15-5
 - 定義, 15-3
 - ネストした表, 15-5
 - 変更の問題, 20-12
 - 利点, 15-2
- オブジェクト・リレーショナル DBMS (ORDBMS), 1-21, 13-2
- オペレーティング・システム
 - 管理者の権限, 5-3
 - 通信ソフトウェア, 8-26
 - 認証, 29-4
 - ブロック・サイズ, 4-3
 - ルール, 30-20
- オンライン REDO ログ, 1-47, 32-7
 - アーカイブ, 32-17, 32-19
 - 制御ファイルに記録される, 32-20
 - 多重化, 32-5
 - チェックポイント, 32-21
 - メディア障害, 32-5
- オンライン・トランザクション処理 (OLTP), 11-5
 - 逆キー索引, 10-30
- オンライン分析処理 (OLAP)
 - 索引構成表, 10-40

か

- カーソル
 - 埋込み SQL, 16-5
 - オープン, 7-9, 16-6
 - オブジェクト依存性, 21-9
 - 概要, 1-16
 - 再帰, 16-6
 - 再帰 SQL, 16-6
 - 最大数, 16-6
 - 作成, 16-11
 - ストアド・プロシージャ, 16-17

- 定義, 16-6
- プライベート SQL 領域, 7-9, 16-6
- カーディナリティ, 10-31
- 解析, 16-11
 - DBMS_SQL パッケージ, 16-19
 - SQL 文, 16-11, 16-19
 - 埋込み SQL, 16-5
 - 解析コール, 16-7
 - 解析ロック, 16-11, 27-29
 - 実行, 16-7
- 解析ツリー, 18-17
 - 共有 SQL 領域, 7-8
 - 構築, 16-7
 - データベースへの格納, 18-17
- 階層, 1-28, 10-18
 - 結合キー, 1-28, 10-19
 - レベル, 1-28, 10-18
- 回復
 - Point-in-Time
 - クローン・データベース, 5-7
 - Recovery Manager, 1-50, 32-14
 - インスタンスの回復, 32-4
 - SMON プロセス, 1-18, 8-11, 26-38
 - インスタンス障害, 1-45, 32-4
 - パラレル DML, 26-38
 - ファースト・スタート・チェックポイント, 32-13
 - 読み込み専用表領域, 32-6
 - 概要, 1-44, 32-8
 - 基本的な手順, 1-49, 32-9
 - クラッシュ回復, 1-45, 32-4, 32-13
 - SMON プロセス, 1-18, 8-11
 - インスタンス障害, 5-10
 - インスタンスの異常終了後に必要, 5-9
 - データベースのオープン, 5-8
 - 読み込み専用表領域, 32-6
 - 障害回復, 32-25
 - 使用される構造, 1-46, 32-6
 - 推奨事項, 32-13
 - スタンバイ・データベース, 32-26
 - 全体データベース・バックアップ, 32-23
 - データベース・パッチ, 32-8
 - デッド・トランザクション, 32-4
 - パラレル DML, 26-37
 - パラレル回復, 32-10
 - パラレル復旧, 32-16
 - ブロック・レベルの回復, 27-21, 32-14

- 分散処理, 8-12
- 分散トランザクション, 5-8
- 文障害, 32-3
- プロセスの回復, 8-11, 32-3
- メディア回復
 - 使用可能または使用禁止, 32-17
 - ディスパッチャ・プロセス, 8-20
- ロールバック・トランザクション, 32-9
- ロールフォワード, 32-9
- 回復時のロールバック, 32-9
- 回復時のロールフォワード, 1-49, 32-9
- 書込みが読込みを阻止するか, 27-11
- 拡張 ROWID 形式, 12-15
- 拡張可能な最適化, 22-16
 - ユーザー定義コスト, 22-17
 - ユーザー定義選択性, 22-17
 - ユーザー定義統計, 22-17
- 拡張性
 - クライアント / サーバー・アーキテクチャ, 6-4
 - パッチ・ジョブ, 26-35
 - パラレル DML, 26-34
 - パラレル SQL 実行, 26-2
- 獲得回避規則, 14-8
- 仮想表, 1-22
- 仮想メモリー, 7-17
- 型
 - データ型、オブジェクト型を参照
 - 権限, 30-10
- カタログ、レプリケーション, 34-14
- カッコ、メソッド・コールでの使用, 14-9
- 各国語サポート (NLS)
 - CHECK 制約, 28-17
 - DATE データ型とパーティション, 11-14, 11-21
 - NCHAR および NVARCHAR2 データ型, 12-6
 - NCLOB データ型, 12-12
 - キャラクタ・セット, 12-6
 - クライアントとサーバーで個別に選択可能, 33-19
 - パラメータ, 5-5
 - ビュー, 10-14
- 仮読込み, 27-3, 27-11
- 監査, 1-43, 31-1
 - DDL 文, 31-6
 - DML 文, 31-6
 - アクセス別, 31-10
 - 必須, 31-11
 - オプションが有効になる時期, 31-5
 - 監査オプション, 31-3
 - 監査証跡, 31-3
 - オペレーティング・システム, 31-5, 31-6
 - データベース, 31-3
 - 監査レコード, 31-3
 - 管理者権限での接続, 31-5
 - 起動と停止, 31-5
 - 権限の使用, 31-2, 31-7
 - 失敗した実行, 31-9
 - 使用するデータ・ディクショナリ, 2-5
 - スキーマ・オブジェクト, 31-2, 31-7
 - 成功した実行, 31-9
 - セキュリティ, 31-6
 - セッション別, 31-9
 - 禁止, 31-11
 - 説明, 1-43, 31-2
 - 対象範囲, 31-3, 31-9
 - タイプ, 31-2
 - データベースと OS のユーザー名, 29-4
 - トランザクションに依存しない, 31-4
 - パーティション表とパーティション索引, 11-61
 - 文, 31-2, 31-6
 - 分散データベース, 31-6
 - ユーザー, 31-11
- 監査証跡
 - ディクショナリ内のデータの削除, 2-5
- 関数
 - Java
 - パラレル実行, 26-43
 - PL/SQL, 18-2, 18-6
 - DETERMINISTIC, 21-7, 23-9
 - プロシージャも参照
 - 権限, 30-7
 - パラレル実行, 26-43
 - ロール, 30-18
 - SQL, 16-2
 - CHECK 制約, 28-17
 - COUNT, 10-34
 - NVL, 10-7
 - デフォルトの列値, 10-8
 - ビューでの使用, 10-14
 - ビュー問合せの最適化, 23-22
 - ハッシュ関数, 10-52
 - ファンクションベース索引, 10-24
 - ユーザー定義
 - 拡張可能な最適化, 22-16
 - 管理者権限, 5-3
 - OUTLN スキーマ, 22-7

- 監査されない文の実行, 31-4
- 監査される接続, 31-5
- 外部キー, 1-57
 - 一部が NULL, 28-16
 - 親キーを使用するための権限, 30-5
 - 定義, 1-57
- 外部キーのマッチング
 - 完全一致、部分一致または不一致, 28-16
- 外部結合
 - NULL に対する NULL 以外の値, 24-11
 - 定義, 23-3
- 外部参照, 18-10
 - 名前変換, 18-19
- 外部プロシージャ, 16-19, 18-11

き

- キー
 - 値の最大記憶領域, 10-23
 - 一意, 28-8
 - 複合, 28-9, 28-11
 - 親, 28-13, 28-15
 - 外部キー, 28-12, 28-13
 - キー値, 1-57
 - 逆キー索引, 10-29
 - クラスタ, 1-25, 10-46
 - 検索, 23-38
 - 索引, 10-23
 - PRIMARY KEY 制約, 28-12
 - UNIQUE 制約, 28-10
 - 圧縮, 10-28
 - 逆キー, 10-29
 - 参照, 1-57, 28-13
 - 主キー, 28-11
 - 制約における, 1-57
 - 定義, 28-9
 - ハッシュ, 10-51, 10-55
- 一意キー
 - 複合, 28-9, 28-11
- キー圧縮, 10-28
- 記憶域
 - REF, 14-18
 - オブジェクト表, 14-17
 - データ・ファイル, 3-16
 - ネストした表, 14-18
 - ユーザーごとの割当て制限, 1-41
 - 論理構造, 3-7

- 記憶領域
 - NULL, 10-7
 - クラスタ, 10-46
 - 索引, 10-25
 - 索引パーティション, 11-37
 - トリガー, 20-2, 20-23
 - ハッシュ・クラスタ, 10-49
 - パラレル DDL の断片化, 26-30
 - パラレル INSERT, 25-8
 - 表パーティション, 11-27
 - 表領域の取消し, 29-14
 - 表領域割当て制限, 29-13
 - ビュー定義, 10-14
 - ユーザーに対する制限, 29-13
 - 論理構造, 10-2
- 記憶領域パラメータ
 - MAXEXTENTS UNLIMITED, 26-36
 - NEXT, 25-8
 - 計算, 25-9
 - OPTIMAL (ロールバック・セグメント), 4-24, 26-36
 - PCTINCREASE, 25-8, 25-9
 - 設定, 4-11
 - パラレル・ダイレクト・ロード・インサート, 25-8
- 規格, 1-3
 - ANSI/ISO, 1-3, 28-5, 28-16
 - 分離レベル, 27-2, 27-11
 - FIPS, 16-6
 - Oracle が準拠, 1-3
 - 整合性制約, 28-5, 28-16
- 規格化されていない機能のフラグ付け, 16-6
- 起動, 5-2, 5-5
 - SGA の割当て, 7-2
 - 開始アドレス, 7-13
 - 回復, 32-4
 - 監査レコード, 31-5
 - 強制実行, 5-6
 - ステップ, 5-5
 - 制限モード, 5-5
 - ディスパッチャ・プロセスでは禁止, 8-20
- 起動者権限, 18-9
 - 提供されるパッケージ, 30-8
 - 名前変換, 18-19
 - プロシージャのセキュリティ, 30-8
- 基本レプリケーション, 34-12
 - 使用方法, 34-11
- キャッシュ

- オブジェクト・キャッシュ, 13-13, 13-14, 14-14
 - オブジェクト・ビュー, 15-4
- キャッシュ・ヒット, 7-4
- キャッシュ・ミス, 7-4
- 共有 SQL 領域, 7-6, 7-8
- データ・ディクショナリ, 2-4, 7-10
 - 位置, 7-6
- データベース・バッファ, 1-15
- バッファ・キャッシュ, 7-3
 - 複数のバッファ・プール, 7-5
- バッファの書き込み, 8-8
- プライベート SQL 領域, 7-8
- ライブラリ・キャッシュ, 7-6, 7-7, 7-10
- キャッシュ・フュージョン
 - 読み込み一貫性, 27-7
- キャラクタ・セット
 - CLOB および NCLOB データ型, 12-12
 - NCHAR および NVARCHAR2, 12-6
 - 各国語, 5-5
 - 列の長さ, 12-6
- キューイング, 19-2
 - インスタンス親和性, 19-10
 - キュー表, 19-4, 19-11
 - キュー表のエクスポート, 19-11
 - キュー・モニター・プロセス, 1-19, 8-13, 19-6
 - 間隔統計, 19-11
 - 実行の時間枠, 19-7
 - キュー・レベルのアクセス制御, 19-9
 - パブリッシュ / サブスクライプのサポート, 19-10
 - イベントの発行, 20-18
 - リモート・データベース, 19-9
 - 例外処理, 19-11
 - レシビエント, 19-5
 - サブスクライバ・リスト, 19-5
 - ルールベースのサブスクリプション, 19-5, 19-6
- キューイングのエージェント, 19-4
- キュー・モニター・プロセス (QMN n), 1-19, 8-13, 19-6
 - 間隔統計, 19-11
 - 実行の時間枠, 19-7
- 競合
 - データ
 - デッドロック, 8-19, 27-17
 - ロックの段階的拡大が発生しない, 27-17
 - プロシージャ・レプリケーション, 34-16
 - ロールバック・セグメント, 4-21
- 共有 SQL 領域, 7-8, 16-7
- ANALYZE コマンド, 7-11
- SQL のロード, 16-11
- 依存性管理, 7-11
- 解析ロック, 27-29
- 概要, 1-15, 16-7
- サイズ, 7-8
- 説明, 7-8
- プロシージャ、パッケージ、トリガー, 7-9
- 共有グローバル領域 (SGA), 7-2
 - システム・グローバル領域も参照
- 共有サーバー, 1-17
 - 管理者権限で接続できない, 5-3
- 共有サーバー・プロセス (Snnn), 8-14, 8-19
 - 説明, 8-19
- 共有プール, 7-6
 - ANALYZE コマンド, 7-11
 - 依存性管理, 7-11
 - オブジェクト依存性, 21-9
 - 概要, 1-15
 - 行キャッシュ, 7-10
 - サイズ, 7-6
 - 説明, 7-6
 - フラッシュ, 7-11
 - プロシージャとパッケージ, 18-16
 - 割当て, 7-10
- 共有モード
 - ロールバック・セグメント, 4-27
- 共有ロック
 - 共有表ロック (S), 27-24
- 疑似コード, 18-17
- トリガー, 20-23
- 疑似列
 - CHECK 制約が禁止する
 - LEVEL および ROWNUM, 28-17
 - ROWID, 12-14
 - ROWNUM
 - 索引を使用できない, 23-47
 - ビュー問合せの最適化, 23-15, 23-24
 - USER, 30-6
 - ビューの変更, 20-13
- 逆キー索引, 10-29
- 行, 1-22, 10-3
 - ROWID が変更される場合, 12-15
 - ROWID での表示, 12-15, 12-16
 - ROWID を使用する検索, 23-33, 23-37
 - アドレス, 10-7
 - 格納形式, 10-5

- 行オブジェクト, 13-8
- 行ソース, 22-4
- 行レベルのセキュリティ, 30-21
- クラスタ化, 10-6
 - ROWID, 10-7
- サイズ, 10-5
- 索引構成表での行オーバーフロー, 10-37
- 新規ブロックへの移行, 4-10
- 説明, 10-3
- 断片, 10-5
- 定義, 1-22
- データ・ブロック内での形式, 4-4
- トリガー, 20-8
- フェッチ, 16-11
- ブロックにまたがる連鎖, 4-9, 10-5
- ヘッダー, 10-5
- ロック, 11-45, 27-11, 27-20, 27-23
- 論理 ROWID, 12-18
 - 索引構成表, 10-37
- 行 ID
 - 行の移行, 4-10
- 行オブジェクト, 13-8
- 行キャッシュ, 7-10
- 行ソース, 22-4
- 行断片, 10-5
 - 識別方法, 10-7
- ヘッダー, 10-5
- 行ディレクトリ, 4-4
- 行データ (データ・ブロックのセクション), 4-5
- 行トリガー, 20-8
 - トリガーも参照
 - 起動されるタイミング, 20-20
- 行の連鎖, 4-9, 10-5
- 業務規則
 - アプリケーション・コードで施行, 28-5
 - ストアド・プロシージャを使用して施行, 28-5
 - 制約を使用して施行, 1-56, 28-1
 - 利点, 28-5
 - トリガーを使用して施行, 1-58
- 行レベル・ロック, 27-11, 27-20
- 行ロック, 27-11, 27-20
 - 直列可能トランザクション, 27-8
 - ブロック・レベルの回復, 27-21, 32-14



空間データ・アプリケーション

- 索引構成表, 10-40
- クエリー・リライト, 10-17
 - セキュリティ・ポリシー内の動的述語, 30-21
- クライアント / サーバー・アーキテクチャ, 6-2
 - 概要, 1-32, 6-2
 - クライアント, 1-32
 - 図, 6-2
 - 直接接続と間接接続, 33-2
 - 分散処理, 6-2
 - 分散データベース, 33-2
 - プログラム・インタフェース, 8-24
- クラスタ
 - ROWID, 10-7
 - 格納形式, 10-46
 - 概要, 10-44
 - キー, 1-25, 10-46, 10-47
 - NULL の索引付けへの影響, 10-8
 - 記憶領域パラメータ, 10-4
 - クラスタ化するデータの選択, 10-46
 - 結合, 10-46, 24-5
 - 索引, 10-21, 10-47
 - 走査, 23-40
 - ハッシュと対比, 10-48
 - パーティション化できない, 11-2
 - 走査, 7-4, 23-33
 - 定義, 1-25
 - ディクショナリ・ロック, 27-29
- ハッシュ, 10-48
 - 記憶領域, 10-49
 - 索引と対比, 10-48
 - 衝突の解決, 10-51
 - 走査, 23-33
 - 単一表, 10-55
 - 領域の割当て, 10-53
 - ルート・ブロック, 10-53
 - パーティション化できない, 11-2
 - パフォーマンスの考慮事項, 10-46
 - パラメータの設定, 10-47
- クラスタ化コンピュータ・システム
 - Oracle Parallel Server, 5-3
- クラスタ・キー, 1-25, 10-46
- クラスタ結合, 24-5
- クラッシュ回復, 32-4, 32-13
 - SMON プロセス, 1-18, 8-11
 - インスタンス障害, 1-45, 5-10, 32-4
 - インスタンスの異常終了後に必要, 5-9
 - データベースのオープン, 5-8

- 読み専用表領域, 32-6
- クローン・データベース
 - マウント, 5-7
- クロス結合, 23-3
- グループ・インスタンス, 26-17
- グループ・コミット, 8-10
- グローバル索引
 - パーティション化, 11-32
 - 索引タイプのまとめ, 11-34
 - パーティションの管理, 11-33, 11-59
- グローバル・スキーマ・オブジェクト名, 1-28, 33-6
- グローバル・データベース名
 - 共有プール, 7-11
- グローバルなユーザー
 - 現ユーザーのリンク, 18-20

け

計画

- OR 演算子, 23-11
- SQL の実行, 16-3, 16-11
- 結合, 24-2, 24-8
- スター型変換, 24-18
- ビューの結合, 23-24
- ビューへのアクセス, 23-18, 23-20, 23-22
- 複合問合せ, 23-26, 23-27, 23-28
- 複合文, 23-13

結合

- アンチ・ジョイン, 24-13
- 外部, 23-3
 - NULL に対する NULL 以外の値, 24-11
- クラスタ, 10-46, 23-37, 24-5
 - 検索, 23-39
- クロス, 23-3
- 結合順序
 - 実行計画, 22-2
 - 述語の選択性, 22-9, 22-17
- 最適化, 24-10
- 索引結合, 23-35, 23-48
- サンプル表スキャンがサポートされない, 23-33
- 実行計画, 24-2
- スター型結合, 24-14
- スター問合せ, 24-14
- セミ・ジョイン, 24-13
- 選択表示結合ビュー, 23-14
- ソート / マージ, 24-4
 - コストベース最適化, 24-9

- 例, 23-45
- 直積, 23-3
- 定義, 23-3
- 等価結合, 23-3
- ネスト・ループ, 24-2
 - コストベース最適化, 24-9
- ハッシュ結合, 24-7
- パーティション・ワイズ, 11-5
- 非同レベル結合, 23-3
- ビュー, 1-23, 10-15
- ビューにカプセル化, 1-23, 10-13
- 副問合せへの変換, 23-12
- 結合ビュー, 10-15

権限

- RESTRICTED SESSION, 29-19
- 解析時にチェックされる, 16-11
- 監査の使用方法, 1-43, 31-7
- 管理者, 5-3
 - OUTLN スキーマ, 22-7
 - 監査されない文の実行, 31-4
 - 監査される接続, 31-5
- 概要, 1-40, 30-2
- システム, 30-2
 - 概要, 1-40
 - 付与と取消し, 30-3
 - ユーザー定義型, 14-12
- スキーマ・オブジェクト, 30-3
 - DML 操作と DDL 操作, 30-4
 - 概要, 1-40
 - パッケージ, 30-9
 - 付与と取消し, 30-4
 - プロシージャ, 30-7
- データベースの起動または停止, 5-3
- トリガー権限, 30-7
- 取消し, 30-3, 30-4
 - オブジェクト依存性, 21-6
- パーティション表とパーティション索引, 11-61
- ビュー, 30-5
 - 作成, 30-5
 - 使用, 30-6
- ファンクションベース索引, 10-25, 21-7
- 付与, 1-40, 30-3, 30-4
 - 例, 30-9
- プロシージャ, 30-7
 - 作成と変更, 30-8
 - 実行, 18-18, 30-7
 - パッケージ内, 30-9

ユーザー定義型

- ADMIN OPTION 付きの EXECUTE ANY TYPE , 14-13
- ALTER ANY TYPE , 14-12
- CREATE ANY TYPE , 14-12
- CREATE TYPE , 14-12
- DELETE , 14-14 , 14-15
- DROP ANY TYPE , 14-12
- EXECUTE , 14-12 , 14-13
- EXECUTE ANY TYPE , 14-12 , 14-13
- GRANT オプション付きの EXECUTE , 14-13
- INSERT , 14-14 , 14-15
- SELECT , 14-13 , 14-14
- UPDATE , 14-14 , 14-15
- オブジェクト表の列レベル , 14-15
- システム権限 , 14-12
- 使用 , 14-12 , 14-16
- ピン時のチェック , 14-14
- ロールによって取得 , 14-12
- ロール , 30-15
 - 制限 , 30-19
- ロールとしてグループ化 , 1-40
- ゲートウェイ , 33-8
- 現ユーザー , 18-10

こ

公開鍵インフラストラクチャ , 29-5

更新

- 位置の透過性 , 33-14
- オブジェクト・ビュー , 15-5
- オブジェクト・ビューの更新可能性 , 15-5
- 更新可能な結合ビュー , 10-15
- 頻繁に更新が行われる環境 , 27-9
- ビューの更新可能性 , 10-15 , 20-11 , 20-13
- 分散 , 33-11

更新可能スナップショット , 34-9

高水位標

- ダイレクト・ロード・インサート , 25-3

高速コミット , 8-10

高速全索引走査 , 23-34

高速リフレッシュ , 10-18

構造

- データ・ディクショナリ , 1-28 , 2-1
- データ・ファイル
 - ROWID での表示 , 12-16
- データ・ブロック

ROWID での表示 , 12-16

物理 , 1-5 , 1-11

REDO ログ・ファイル , 1-12 , 32-7

制御ファイル , 1-12 , 32-20

データ・ファイル , 1-11 , 3-1

プロセス , 1-13 , 1-16 , 8-1

メモリー , 1-13 , 7-1

ロック , 27-27

論理 , 1-5 , 1-8 , 4-1

エクステンツ , 1-10 , 4-2 , 4-10

スキーマ・オブジェクト , 1-10 , 10-2

セグメント , 1-10 , 4-2 , 4-16

データ・ブロック , 1-10 , 4-2 , 4-3

表領域 , 1-9 , 3-1 , 3-7

構造化問合せ言語 (SQL) , 1-51 , 16-2

SQL も参照

構造体

物理

データ・ファイル , 3-16

コール

Oracle コール・インタフェース , 8-25

リモート・プロシージャ , 33-11

コストベース最適化 , 22-8 , 33-10

拡張可能な最適化 , 22-16

クエリー・リライト , 10-17

述語の選択性 , 22-9

ヒストグラム , 22-9 , 22-11

ユーザー定義 , 22-17

スター問合せ , 24-14

統計 , 22-9 , 23-31

ユーザー定義 , 22-17

ヒストグラム , 22-9

ユーザー定義コスト , 22-17

固定ビュー , 2-7

コミット読み込み分離 , 27-7 , 27-8

コレクション , 13-10

可変配列 (VARRAY) , 13-10

索引構成表 , 10-36

キー圧縮 , 10-29

ネストした表 , 13-11

コレクションのメソッド

コンストラクタ・メソッド , 1-56

コンストラクタ・メソッド , 1-56 , 13-6 , 14-17

リテラル起動 , 14-4

コンパイル済み PL/SQL , 18-16

共有プール , 16-16

疑似コード , 18-17 , 20-23

- 再コンパイル, 18-18
- トリガー, 20-23
- プロシージャ, 18-9
- 利点, 18-7
- コンパイル済みトリガー, 20-23
- 互換性, 1-3
- 互換性レベル
 - トランスポートابل表領域, 3-15

さ

- サーバー, 1-32
 - 共有, 1-17
 - クライアント / サーバー・アーキテクチャ, 6-2
 - 専用, 1-17, 8-22
 - マルチスレッドと対比, 8-16
 - 専用サーバー・アーキテクチャ, 8-3
 - 定義, 1-33
 - プロセス, 1-17
 - マルチスレッド, 1-17
 - アーキテクチャ, 8-3, 8-16
 - 専用と対比, 8-16
 - プロセス, 8-14, 8-16, 8-19
- サーバー・プロセス, 1-17, 8-5
 - リスナー・プロセス, 6-6
- サービス
 - 異機種, 33-8
- サービス名, 6-6
- 再帰 SQL
 - カーソル, 16-6
- 最低使用頻度アルゴリズム (LRU)
 - 共有 SQL プール, 7-8, 7-10
 - 全表走査, 7-4
 - ディクショナリ・キャッシュ, 2-4
 - データベース・バッファ, 7-3
 - ラッチ, 8-8
- 最適化, 22-2
 - DISTINCT, 23-15
 - GROUP BY ビュー, 23-15
 - NULL に対する NULL 以外の値, 24-11
 - PL/SQL, 23-32
 - SQL 文のタイプ, 23-4
 - アプローチの選択, 23-30
 - 拡張可能オブティマイザ, 22-16
 - クエリー・リライト, 10-17
 - セキュリティ・ポリシー内, 30-21
 - コストベース, 22-8, 24-9

- アクセス・パスの選択, 23-48
- スター問合せ, 24-14
- ヒストグラム, 22-9
- ユーザー定義コスト, 22-17
- リモート・データベース, 23-29
 - 例, 23-50
- 索引作成, 10-22
- 式と述語の変換, 23-4
- 手動, 23-32
- 実行される操作, 23-2
- 述語の選択性, 22-9
 - ヒストグラム, 22-9, 22-11
 - ユーザー定義, 22-17
- 推移律, 23-7
- 説明, 22-2
- セミ・ジョイン, 24-13
- 問合せの選択性, 23-49
- 統計, 22-9, 23-31
 - ユーザー定義, 22-17
- パーティション索引, 11-35
- パーティションの実行計画, 11-11, 11-15
- パーティション・プルニング, 11-4
 - 索引, 11-36
- パーティション・プルニング (排除), 11-4
- パーティション・ワイズ結合, 11-5
- パラレル SQL, 26-10
- ヒント, 23-32, 23-34, 23-35
- ファンクションベース索引, 10-25
- 複合ビューのマージ, 23-15
- 分散型の SQL 文, 23-29
- 文へのビューのマージ, 23-14
- マージなし, 23-24
- ルールベース, 22-18, 24-10
 - アクセス・パスの選択, 23-52
 - 例, 23-52
- サイト自律性, 1-34, 33-15
- 作業負荷の不整, 26-17
- 索引, 1-25, 10-21
 - B* ツリー構造, 10-26
 - LONG RAW データ型では禁止, 12-13
 - NULL, 10-8, 10-23, 10-34
 - REF, 14-6
 - ROWID, 10-27
 - 位置, 10-26
 - 一意, 10-22
 - 一意走査, 23-34
 - オブジェクト列の属性, 14-6

- カーディナリティ, 10-31
- 拡張可能, 10-40
- 格納形式, 10-26
- 概要, 1-25, 10-21
- キー, 10-23
 - UNIQUE キー制約, 28-10
 - 主キー制約, 28-12
- キー圧縮, 10-28
- 逆キー索引, 10-29
- クラスタ, 10-47
 - 削除, 10-48
 - パーティション化できない, 11-2
 - 表と対比, 10-48
- グローバル・パーティション索引, 11-32
 - パーティションの管理, 11-33, 11-59
- 高速全走査, 23-34
- 最適化, 23-10
- 索引結合, 23-35, 23-48
- 索引構成表, 10-35
 - 2 次索引, 10-37
 - 論理 ROWID, 10-37, 12-18
- 索引使用禁止状態 (IU), 11-60
- 作成
 - 既存の索引を使用, 10-22
- 整合性制約の施行, 28-10, 28-12
- 説明, 1-25, 10-21
- 走査, 23-34
 - MAX または MIN, 23-45
 - ORDER BY, 23-46
 - 制限, 23-46
 - 単一列, 23-41
 - 非有界範囲, 23-44
 - 有界範囲, 23-43
- ダイレクト・ロード・インサートの後の再作成, 25-8
- ドメイン, 10-40
- ドメイン索引
 - 拡張可能な最適化, 22-16
 - ユーザー定義統計, 22-17
- 内部構造, 10-26
- 範囲走査, 23-34
- パーティション, 11-2, 11-28
- パーティション化のガイドライン, 11-36
- パーティションについての権限, 11-61
- パーティションの監査, 11-61
- パーティションの管理, 11-58
- パーティションの再作成, 11-59
- パーティション表, 10-35
- パーティション・プルニング, 11-4
- パフォーマンス, 10-21
- パラレル DDL 記憶領域, 26-30
- パラレルの索引走査, 26-5
- 非一意, 10-22
- ビットマップ索引, 10-30, 10-35
 - NULL, 10-8
 - パラレル問合せおよび DML, 10-31
- ビューでの使用, 10-14
- ファンクションベース, 10-24
 - DETERMINISTIC 関数, 21-7
 - DISABLED, 21-8
 - 依存性, 10-25, 21-7, 21-9
 - 権限, 10-25, 21-7
 - 最適化, 10-25
- 複合, 10-22
- 複合データ型, 10-40
- ブランチ・ブロック, 10-27
- 文の変換, 23-10
- ユーザー定義型, 14-6
- リーフ・ブロック, 10-27
- 連結, 10-22
- ローカル索引, 11-29, 11-58
 - パーティションを並列に作成, 11-30
- ロギングなしモード, 25-7
- 索引結合, 23-35, 23-48
- 索引構成表, 10-35
 - 2 次索引, 10-37
 - 2 次パーティション索引, 11-44
- アプリケーション, 10-38
- キー圧縮, 10-29, 10-36
- キュー表, 19-12
- 行オーバーフロー領域, 10-37
- 再構築, 10-38
- パーティション, 11-41
- パラレル CREATE, 26-28
- パラレル問合せ, 26-26
- 利点, 10-36
- 論理 ROWID, 10-37, 12-18
- 索引セグメント, 1-10, 4-17
- 索引タイプ, 10-41
- 索引の NOREVERSE オプション, 10-30
- 索引の REVERSE オプション, 10-30
- サブスクリプション
 - ルールベース, 19-5, 19-6
- サブパーティション

- 統計, 22-12
- サブパーティション・ロック
 - DML, 11-46
- 参照
 - オブジェクト
 - 依存性, 21-2
 - 外部参照, 18-10, 18-19
 - キー, 1-57, 28-13
 - パーティション, 11-19
- 参照解除, 13-9
 - 暗黙的, 13-9
- 参照整合性, 27-12, 28-12, 28-13
 - CASCADE 規則, 28-3
 - PRIMARY KEY 制約, 28-11
 - RESTRICT 規則, 28-3
 - SET TO DEFAULT 規則, 28-3
 - SET TO NULL 規則, 28-3
 - 一部が NULL の外部キー, 28-16
 - 自己参照型制約, 28-15, 28-18
 - 例, 28-18
- 参照表
 - スター問合せ, 24-14
- サンプル表スキャン, 23-33, 23-47
 - ヒントで上書きできない, 23-49

し

- システム・グローバル領域 (SGA), 7-2
 - REDO ログ・バッファ, 7-5, 17-5
 - いつ割り当てられるか, 7-2
 - 概要, 1-15, 7-2
 - 共有および書き込み可能, 7-2
 - 共有プール, 7-6
 - 固定, 7-3
 - サイズ, 7-12
 - 可変パラメータ, 5-4
 - 図解, 5-2
 - 大規模プール, 7-11
 - データ・ディクショナリのキャッシュ, 2-4, 7-10
 - データベース・バッファ・キャッシュ, 7-3
 - 内容, 7-3
 - プライベート SQL 領域の制限, 29-17
 - ロールバック・セグメント, 17-5
 - 割当て, 5-5
- システム権限, 30-2
 - ADMIN OPTION, 14-13, 30-3
 - 権限も参照

- 説明, 30-2
- 付与と取消し, 30-3
- ユーザー定義型, 14-12
- システム制御文, 1-52, 16-5
- システム変更番号 (SCN)
 - REDO ログ, 8-10
 - 決定される時期, 27-5
 - 定義, 17-5
 - トランザクションのコミット, 17-5
 - 読みみ一貫性, 27-5, 27-6
- システム・モニター・プロセス (SMON), 8-11
 - Parallel Server, 8-11, 26-38
 - 一時セグメントのクリーン・アップ, 8-11
 - インスタンス回復, 32-4
 - 定義, 1-18, 8-11
 - パラレル DML インスタンス回復, 26-38
 - パラレル DML システム回復, 26-38
 - ロールバック・トランザクション, 32-10
- シノニム, 21-8
 - オブジェクトからの権限の継承, 30-3
 - 拡張パーティション表名, 11-63
 - 概要, 1-24
 - 使用方法, 10-20
 - 制約が間接的に影響する, 28-5
 - 説明, 10-20
 - データ・ディクショナリ・ビュー, 2-4
 - パブリック, 10-20
 - プライベート, 10-20
- シャドウ・プロセス, 8-22
- 主キー, 1-57, 28-11
 - 検索, 23-39
 - 最適化, 23-13
 - 定義, 28-3
 - 利点, 28-11
- 手動ロック, 1-32, 27-30
- 障害, 32-2
 - REDO ログファイルのアーカイブ, 32-19
 - 回復も参照
 - インスタンス, 1-45, 32-4
 - 回復, 5-8, 5-10, 32-4
 - 説明, 1-44, 32-2
 - 耐障害性, 32-25
 - 提供されている保護対策, 32-6
 - データベース・バッファ, 32-8
 - 内部エラー
 - トレース・ファイルに記録, 8-15
 - ネットワーク, 32-3

文障害とプロセス障害, 1-44, 8-11, 32-2
 メディア, 1-45, 32-5
 ユーザー・エラー, 1-44, 32-2
 障害回復, 32-25
 初期化パラメータ
 ALWAYS_ANTI_JOIN, 24-13
 ALWAYS_SEMI_JOIN, 24-13
 AQ_TM_PROCESS, 19-6, 19-7
 BUFFER_POOL_KEEP, 7-5
 BUFFER_POOL_RECYCLE, 7-5
 COMPATIBLE, 3-12
 DB_BLOCK_BUFFERS, 7-4, 7-12
 DB_BLOCK_LRU_LATCHES, 8-8
 DB_BLOCK_SIZE, 7-4, 7-12
 DB_FILE_MULTIBLOCK_READ_COUNT, 23-49, 24-9
 DB_FILES, 7-15
 DB_NAME, 32-21
 DB_WRITER_PROCESSES, 1-17, 8-8
 DISTRIBUTED_TRANSACTIONS, 8-12
 FAST_START_IO_TARGET, 32-13
 HASH_AREA_SIZE, 24-8
 HASH_JOIN_ENABLED, 24-7
 HASH_MULTIBLOCK_IO_COUNT, 24-8
 HI_SHARED_MEMORY_ADDRESS, 7-13
 JOB_QUEUE_PROCESSES, 19-9
 LICENSE_MAX_SESSIONS, 29-19
 LICENSE_SESSIONS_WARNING, 29-19
 LOCK_SGA, 7-13, 7-17
 LOG_ARCHIVE_MAX_PROCESSES, 1-18, 8-12, 32-19
 LOG_ARCHIVE_START, 32-19
 LOG_BUFFER, 7-6, 7-12
 LOG_CHECKPOINT_INTERVAL, 32-13
 LOG_CHECKPOINT_TIMEOUT, 32-13
 MTS_MAX_SERVERS, 8-19, 8-20
 MTS_SERVERS, 8-19
 NLS_LANGUAGE, 11-20
 NLS_NUMERIC_CHARACTERS, 12-8
 NLS_SORT, 11-20
 OPEN_CURSORS, 7-9, 16-6
 OPEN_LINKS, 7-15
 OPTIMIZER_FEATURES_ENABLE, 23-16, 23-34, 23-35, 24-12
 OPTIMIZER_MODE, 23-30
 OPTIMIZER_PERCENT_PARALLEL, 22-8
 PARALLEL_MAX_SERVERS, 26-8
 PARALLEL_MIN_PERCENT, 26-17
 PARALLEL_MIN_SERVERS, 26-7, 26-8
 PARALLEL_SERVER, 5-6
 REMOTE_DEPENDENCIES_MODE, 21-10
 ROLLBACK_SEGMENTS, 4-26
 SERVICE_NAMES, 6-6
 SHARED_MEMORY_ADDRESS, 7-13
 SHARED_POOL_SIZE, 7-6, 7-12
 SKIP_UNUSABLE_INDEXES, 21-8
 SORT_AREA_RETAINED_SIZE, 7-16
 SORT_AREA_SIZE, 4-18, 7-16, 24-9
 SQL_TRACE, 8-15
 STAR_TRANSFORMATION_ENABLED, 24-19
 TRANSACTIONS, 4-26
 TRANSACTIONS_PER_ROLLBACK_SEGMENT, 4-26
 USE_INDIRECT_DATA_BUFFERS, 7-13
 初期即時制約, 28-20
 初期遅延制約, 28-20
 署名のチェック, 21-10
 処理
 DDL 文, 16-14
 DML 文, 16-10
 概要, 16-8
 問合せ, 16-11
 パラレル SQL, 26-2
 分散, 1-32
 使用済みバッファ, 7-3
 増分チェックポイント, 8-8
 ファースト・スタート・チェックポイント, 32-13
 シンプル・ネットワーク管理プロトコル (SNMP)
 データベース管理, 33-19
 シンプル・ネットワーク管理プロトコル (SNMP) のサポート
 データベース管理, 33-19
 親和性
 パーティション, 26-44
 パラレル DML, 26-45
 自己参照的なスタイルのメソッドの起動, 13-6
 事実表
 スター型結合, 24-14
 スター問合せ, 24-14
 事前書込み, 8-9
 持続領域, 7-8
 実行計画
 EXPLAIN PLAN, 16-3
 OR 演算子, 23-11

- SQL の解析, 16-11
- 位置, 7-8
- 概要, 22-2
- 結合, 24-2, 24-8
- 実行順序, 22-6
- スター型変換, 24-18
- パーティションとパーティション・ビュー, 11-11, 11-15
- 表示, 22-5
- ビューの結合, 23-24
- ビューへのアクセス, 23-18, 23-20, 23-22
- 複合問合せ, 23-26, 23-27, 23-28
- 複合文, 23-13
- プラン・スタビリティ, 22-7
- 例, 23-13
- 実行時領域, 7-8
- 実表, 1-23
 - ビューも参照
 - データ・ディクショナリ, 2-2
- 自動化されたスタンバイ・データベース, 32-26
- 述語
 - 選択性, 22-9
 - ヒストグラム, 22-9, 22-11
 - ユーザー定義, 22-17
 - 動的
 - セキュリティ・ポリシー内, 30-21
 - パーティション・プルニング, 11-4
 - 索引, 11-36
 - ビュー問合せの最適化, 23-14
 - ビューへの追加, 23-17, 23-22
 - 例, 23-17, 23-20
- 述語の選択性, 22-9
 - ヒストグラム, 22-9, 22-11
 - ユーザー定義選択性, 22-17
- 順序, 1-24, 10-19
 - CHECK 制約が禁止する, 28-17
 - 監査, 31-7
 - 数値の生成, 10-19
 - 数値の長さ, 10-19
 - 表からの独立, 10-19
- 順序メソッド, 1-56, 13-7
- 情報
 - オフロード
 - 基本レプリケーション, 34-11
 - 配布
 - 基本レプリケーション, 34-11
 - 輸送, 34-12

- 情報検索 (IR) アプリケーション
 - 索引構成表, 10-39
- 情報の配布
 - 基本レプリケーション, 34-11
- ジョブ, 8-2
- ジョブ・キュー・プロセス (SNPn), 1-19, 8-13
- メッセージの伝播, 19-9

す

- スキーマ, 29-2
 - OUTLN, 22-7
 - オブジェクト, 10-2
 - スター・スキーマ, 24-14, 24-15
 - 正規化された表, 24-15
 - 定義, 29-2
 - 内容, 10-2
 - 表領域と対比, 10-2
 - ユーザー定義データ型, 13-13
 - ユーザーとの対応付け, 1-37, 10-2
- スキーマ・オブジェクト, 10-1
 - INVALID 状態, 21-2
 - 依存性, 21-2
 - 存在しない他のオブジェクト, 21-8
 - トリガー管理, 20-20
 - ビュー, 10-15
 - 分散データベース, 21-12
 - 失われた権限に依存, 21-6
 - 監査, 1-43, 31-7
 - 概要, 1-10, 1-22, 10-2
 - グローバル名, 33-6
 - 権限, 30-3
 - 索引タイプ, 10-41
 - 作成
 - 表領域割当て制限が必要, 29-13
 - 定義, 1-5
 - ディメンション, 10-18
 - データ・ディクショナリ内の情報, 2-2
 - データ・ファイルとの関連, 3-16, 10-2
 - デフォルトの表領域, 29-13
 - トリガーの依存性, 20-24
 - 取り消した表領域内, 29-14
 - ドメイン索引, 10-41
 - 分散データベースにおける名前, 33-6
 - 分散データベースの命名規則, 33-6
 - マテリアライズド・ビュー, 10-17
 - ユーザー定義オペレータ, 10-42

- ユーザー定義型, 13-3
- スキーマ・オブジェクト権限, 30-3
 - DML 操作と DDL 操作, 30-4
 - 概要, 1-40
 - ビュー, 30-5
 - 付与と取消し, 30-4
- スキーマ名
 - データベース内で一意, 33-6
 - 分散データベースにおける, 33-6
 - 列名の修飾, 14-8
- スター型変換, 24-16
- スター型結合, 24-14
- スター型変換
 - 制限, 24-19
 - 例, 24-16
- スター・スキーマ
 - 非正規化ビュー, 24-15
- スター問合せ, 24-14
 - 拡張スター・スキーマ, 24-15
 - 索引, 24-15
 - スター型変換, 24-16
 - チューニング, 24-15
 - 非正規化ビュー, 24-15
 - ヒント, 24-15
- スタック領域, 7-14
- スタンバイ・データベース
 - 耐障害性, 32-25
 - マウント, 5-7
- ストアド・ファンクション, 1-24, 18-2, 18-6
- ストアド・プロシージャ, 1-24, 16-15, 18-2, 18-6
 - プロシージャも参照
 - コール, 16-18
 - トリガーと対比, 20-2
 - 変数と定数, 16-17
 - 無名ブロックと対比, 18-9
- スナップショット
 - グループ, 34-6
 - 更新可能, 34-9
 - サイト, 34-6
 - マテリアライズド・ビューと同義, 1-23, 34-3
 - 読み込み専用, 34-8
 - リフレッシュ, 8-13
- スナップショット読み込み時期, 27-11
- スナップショット・リフレッシュ
 - ジョブ・キュー・プロセス (SNPn), 1-19, 8-13
- スループット, 22-8
 - コストベースのアプローチ, 23-30

- スレッド
 - マルチスレッド・サーバー, 8-14, 8-16

せ

- 世紀, 12-11
- 正規化された表, 1-28, 10-19
 - スター・スキーマ, 24-15
- 正規化されていない表, 1-28, 10-19
- 制御ファイル, 1-12, 32-20
 - 回復, 1-48
 - 概要, 1-12, 32-20
 - 記録される変更内容, 32-21
 - 指定方法, 5-4
 - 多重化, 1-48, 32-21
 - チェックポイント, 32-21
 - データベースのマウントでの使用, 5-6
 - 内容, 32-20
 - バックアップ, 32-24
 - 物理データベース構造, 1-5
- 制限
 - 関数のパラレル実行, 26-43
 - ダイレクト・ロード・インサート, 25-11, 26-40
 - ネストした表, 26-27
 - パーティション
 - 拡張パーティション表名, 11-63
 - データ型, 11-14, 11-21
 - ビットマップ索引, 11-14
 - パーティション・ビュー, 11-12
 - パラレル DDL, 26-28
 - リモート・トランザクション, 26-26
 - パラレル DML, 26-40
 - リモート・トランザクション, 26-26, 26-42
- 制限付き ROWID 形式, 12-16
- 制限モード
 - インスタンスの起動, 5-5
- 整合性制約, 28-2
 - 制約も参照
 - デフォルトの列値, 10-8
- 整合性ルール, 1-21
 - パラレル DML の制限事項, 26-41
- 制約, 1-56
 - CHECK, 28-17
 - ENABLE または DISABLE, 28-21
 - FOREIGN KEY, 1-57, 28-12
 - NOT NULL, 28-7, 28-11
 - PRIMARY KEY, 1-57, 28-11

- UNIQUE キー, 1-57, 28-8
 - 一部が NULL, 28-11
- VALIDATE または NOVALIDATE, 28-21
- アプリケーションが違反を検出可能, 28-6
- 一時的な使用禁止, 28-6
- 違反した場合の処理, 28-5
- オブジェクト表, 14-5
- 概要, 1-56
- 索引による施行, 10-23
 - PRIMARY KEY, 28-12
 - UNIQUE, 28-10
- 参照制約
 - 自己参照型, 28-15
 - 更新の効果, 28-16
- 施行のメカニズム, 28-18
- タイプのリスト, 1-57, 28-1
- 代替処置, 28-5
- 定義, 10-4
- デフォルト値, 28-19
- トリガーと対比, 20-5
- トリガーは違反できない, 20-20
- パフォーマンスに対する効果, 28-6
- パラレル CREATE TABLE, 26-23
- 評価されるタイミング, 10-8
- ビューでは定義不可, 10-12
- 変更, 28-23
- 西暦 2000 年, 12-11
- セーブポイント, 1-54, 17-7
 - 暗黙的, 17-4
 - 概要, 1-54
 - 説明, 17-7
 - ロールバック, 17-6
- セキュリティ, 1-40, 29-2
 - アプリケーションを使用して施行, 1-41
 - 監査, 31-2, 31-6
 - 監査データの削除, 2-5
 - 管理者権限, 5-3
 - 起動者権限, 18-9, 18-19
 - 施行のメカニズム, 1-38
 - システム, 1-37, 2-3
 - セキュリティ・ポリシー, 30-21
 - 説明, 1-37
 - 定義者権限, 18-9, 18-19
 - データ, 1-37
 - 動的述語, 30-21
 - ドメイン, 1-39, 29-2
 - 任意アクセス制御, 1-38, 29-2
 - パスワード, 29-7
 - ビュー, 10-13
 - ビューによる強化, 30-6
 - ファイン・グレイン・アクセス・コントロール, 30-21
 - 分散データベース, 33-16
 - プログラム・インタフェースでの実施, 8-24
 - プロシージャによる強化, 30-7
 - ポリシー
 - インプリメント, 30-23
 - メッセージ・キュー, 19-6
 - ユーザー・アクションの監査, 1-43
- セキュリティ・ドメイン, 1-39, 29-2
 - 使用可能なロール, 30-17
 - 表領域の割当て制限, 29-13
- セグメント, 1-10, 4-16
 - 一時, 1-11, 4-17, 10-10
 - SMON によるクリーン・アップ, 8-11
 - 削除, 4-15
 - 操作, 4-17
 - パラレル INSERT, 25-8
 - 表領域, 4-15, 4-18
 - 割当て, 4-17
 - 割当て制限は無視される, 29-14
- エクステントの割当て解除, 4-13
- 概要, 1-10, 4-16
- 索引, 4-17
- 定義, 4-3
- データ, 4-16
- 表
 - 高水位標, 25-3
- ヘッダー・ブロック, 4-10
- ロールバック, 4-19
- セッション
 - PARALLEL DML の使用可能化, 26-35
 - PGA 内のスタック領域, 7-14
 - 監査オプションが有効になる時期, 31-5
 - 現ユーザー, 18-10
 - 時間制限, 29-17
 - 情報の格納場所, 7-14
 - 接続と対比, 8-5
 - 大規模プール内のメモリー割当て, 7-11
 - 定義, 8-5, 31-9
 - トランザクション分離レベル, 27-31
 - 同時セッションに対する制限, 1-42
 - ライセンスによる, 29-19
 - パッケージの状態, 21-6

- 別監査, 31-9
- ユーザー当たりの制限, 29-17
- リソース制限, 29-15
- セッション制御文, 1-52, 16-5
- 接続
 - 埋込み SQL, 16-5
 - 管理者権限での, 5-3
 - 監査レコード, 31-5
 - 制限, 5-5
 - セッションとの比較, 8-5
 - 定義, 8-5
 - ユーザー名, 29-2
 - リスナー・プロセス, 6-6, 8-14
- 接続性, 1-2
- セミ・ジョイン, 24-13
- 選択表示結合ビュー, 23-14
- 専用サーバー, 8-22
 - 使用例, 8-23
 - 定義, 1-17
 - マルチスレッド・サーバーと対比, 8-16
- 全索引走査, 23-34
- 全体データベース・バックアップ, 1-49, 32-22
- 全表走査, 23-32, 23-46
 - LRU アルゴリズム, 7-4
 - 選択性, 23-49
 - パラレル実行, 26-5, 26-6
 - マルチブロック読み込み, 23-49
 - ルールベースのオプティマイザ, 23-52

そ

- 関連名
 - インライン・ビュー, 10-16
- 走査, 23-32
 - 一意, 23-34, 23-39
 - クラスタ, 23-37, 23-39
 - 高速全索引走査, 23-34
 - 索引, 23-34
 - MAX または MIN, 23-45
 - ORDER BY, 23-46
 - 制限, 23-46
 - 選択性, 23-49
 - 非有界範囲, 23-44
 - ビットマップ, 23-35
 - 有界範囲, 23-43
 - 索引結合, 23-35, 23-48
 - サンプル表, 23-33, 23-47

- ヒントで上書きできない, 23-49
- 全表, 23-32, 23-46
 - LRU アルゴリズム, 7-4
 - パラレル問合せ, 26-5
 - マルチブロック読み込み, 23-49
 - ルールベースのオプティマイザ, 23-52
- 範囲, 23-34
 - MAX または MIN, 23-45
 - ORDER BY, 23-46
 - 非有界, 23-44
 - 有界, 23-43
 - 表走査と CACHE 句, 7-4
- ソート・セグメント, 3-13
- ソート操作, 3-13
- ソート / マージ結合, 24-4
 - アクセス・パス, 23-45
 - コストベース最適化, 24-9
 - 例, 23-45
- ソート領域, 7-16
- 即時制約, 28-20
- 阻止しているトランザクション, 27-11
- 阻止しているトランザクションを待機するか, 27-11
- ソフトウェア・コード領域, 7-17
 - プログラムとユーティリティによる共有, 7-17
- 増分チェックポイント, 8-8
- 増分リフレッシュ, 10-18
- 属性
 - リーフ・レベル, 14-17
 - リーフ・レベル・スカラー, 14-17

た

- 耐障害性, 32-25
- タイムスタンプのチェック, 21-10
- 大量パラレル処理 (MPP)
 - 親和性, 26-6, 26-44, 26-45
 - パラレル SQL 実行, 26-2
 - 複数の Oracle インスタンス, 5-3
- 多重化
 - REDO ログ・ファイル, 1-47
 - 回復, 32-5
 - 制御ファイル, 1-48, 32-21
- タスク, 8-2
- 単一表ハッシュ・クラスタ, 10-55
- 大規模データベース (VLDB), 11-5
 - パーティション, 11-5
 - パラレル SQL, 26-2

- 大規模プール, 7-11
 - 概要, 1-15
- ダイレクト・ロード・インサート, 25-2
 - シリアル INSERT, 25-3
 - 制限, 25-11, 26-40
 - パラレル INSERT, 25-3
 - パラレル・ロードとパラレル INSERT の比較, 25-2
 - 領域管理, 25-8
 - ロギング・モード, 25-5
- ダンプ・ファイル
 - Export と Import, 14-19
- 断片化
 - パラレル DDL, 26-30

ち

- チェックポイント
 - DBWn プロセス, 8-8, 8-11
 - 制御ファイル, 32-21
 - 増分, 8-8
 - チェックポイント・プロセス (CKPT), 1-18, 8-11
 - 統計, 8-11
 - ファースト・スタート・チェックポイント, 32-13
- チェックポイント・プロセス (CKPT), 1-18, 8-11
- 遅延制約
 - 初期遅延または初期即時, 28-20
 - 遅延可能または遅延不可, 28-20
- 直積, 23-3

つ

- 通信プロトコル, 6-5

て

- 提供されるパッケージ, 18-16
 - 起動者権限または定義者権限, 30-8
- 定義者権限, 18-9
 - 名前変換, 18-19
 - プロシージャのセキュリティ, 30-7
- 停止, 5-9, 5-10
 - SGA の割当て解除, 7-2
 - 異常, 5-6, 5-10
 - 監査レコード, 31-5
 - ステップ, 5-9
 - ディスパッチャ・プロセスでは禁止, 8-20
- 定数

- 計算されるとき, 23-4
- 式の評価, 23-4
- ストアド・プロシージャ内, 16-17
- 比較, 23-4
- テンポラリ・ファイル, 3-17
- ディクショナリ
 - データ・ディクショナリを参照
- ディクショナリ管理の表領域, 3-8
- ディクショナリ・キャッシュ・ロック, 27-30
- ディスク障害, 1-45
- ディスクの親和性
 - パーティション, 26-44
 - パラレル DML, 26-45
- ディスクのストライプ化
 - 親和性, 26-44
 - パーティション, 11-9
- ディスク領域
 - データ・ファイルへの割当て, 3-16
 - 表への割当ての制御, 10-4
- ディスパッチャ・プロセス (Dnnn)
 - Net8 を介したユーザー・プロセスの接続, 8-14, 8-16
 - 応答キュー, 8-17
 - 起動と停止の禁止, 8-20
 - セッション当りの SGA 領域の制限, 29-17
 - 説明, 8-14
 - 定義, 1-19
 - ネットワーク・プロトコル, 8-14
 - リスナー・プロセス, 8-14
- ディメンション, 1-28, 10-18
 - 階層, 1-28, 10-18
 - 結合キー, 1-28, 10-19
 - スター型結合, 24-14
 - スター問合せ, 24-14
 - 正規化された表と正規化されていない表, 1-28, 10-19
 - 属性, 1-28, 10-19
- ディスク障害, 32-5
- データ
 - アクセス, 1-50
 - 制御, 29-2
 - セキュリティ・ドメイン, 29-2
 - ファイン・グレイン・アクセス・コントロール, 30-21
 - メッセージ・キュー, 19-6
 - 一貫性
 - 基礎となる原理, 27-15

- 手動ロック, 27-30
 - 定義, 1-54
 - トランザクション・レベル, 27-6
 - 反復可能読み込み, 27-6
 - 読み込み一貫性, 1-30
 - ロック, 27-3
 - ロック動作の例, 27-31
- 整合性, 1-29, 10-4, 28-2
 - 2 フェーズ・コミット, 1-34
 - CHECK 制約, 28-17
 - 概要, 1-56
 - 参照制約, 28-3
 - 施行, 28-4, 28-5
 - タイプ, 28-3
 - パラレル DML の制限事項, 26-41
- 同時アクセス, 27-2
- 表への格納方法, 10-4
- 分散操作, 1-34
- レプリケーション, 1-35
- ロック, 27-20
- データ・ファイル
 - バックアップ, 32-24
- データ・ウェアハウジング
 - 階層, 1-28, 10-18
 - 基本レプリケーション, 34-12
 - スター問合せ, 24-14
 - ディメンション, 24-14
 - ディメンション・スキーマ・オブジェクト, 1-28, 10-18
 - 表データのリフレッシュ, 26-34
 - ビットマップ索引, 10-31
 - マテリアライズド・ビュー, 10-17
 - 要約, 10-17
- データ・オブジェクト番号
 - 拡張 ROWID, 12-15
- データ型, 12-2, 12-3
 - ANSI, 12-20
 - BOOLEAN, 12-2
 - CHAR, 12-5
 - DATE, 12-9
 - DB2, 12-20
 - LOB データ型, 12-11
 - BFILE, 12-13
 - BLOB, 12-12
 - CLOB および NCLOB, 12-12
 - デフォルトのロギング・モード, 25-7
 - LONG, 12-6
 - 記憶領域, 10-7
 - NCHAR および NVARCHAR2, 12-6
 - NUMBER, 12-7
 - PL/SQL, 12-2
 - RAW および LONG RAW, 12-13
 - ROWID, 12-14
 - SQL/DS, 12-20
 - VARCHAR, 12-5
 - VARCHAR2, 12-5
 - オブジェクト型, 1-21, 13-3
 - コレクション, 13-10
 - 使用可能なデータ型のリスト, 12-2
 - ネストした表, 10-9, 13-11
 - 配列型, 13-10
 - 表との関連, 10-3
 - 変換
 - Oracle 以外の型, 12-20
 - Oracle データ型から別の Oracle データ型へ, 12-21
 - プログラム・インタフェース, 8-24
- マルチメディア, 13-3
- 文字, 12-4, 12-12
- ユーザー定義, 13-1, 13-3
 - 統計, 22-17
- 要約, 12-3
- 列, 1-22
- データ・セグメント, 1-10, 4-16, 10-4
- データ操作言語 (DML)
 - 監査, 31-6
 - 権限制御, 30-4
 - 取得されるロック, 27-25
 - 説明, 16-3
 - 定義, 1-51
 - 副問合せの直列可能分離, 27-14
 - トリガー, 20-3, 20-22
 - パーティション・ロック, 11-45
 - パラレル DML, 26-3, 26-32
 - パラレル DML のためのトランザクション・モデル, 26-36
 - 分散トランザクション, 33-10
 - 文の処理, 16-10
- データ定義言語 (DDL)
 - DBMS_SQL での解析, 16-19
 - PL/SQL への埋込み, 16-19
 - 暗黙のコミット, 17-4
 - 監査, 31-6
 - 説明, 16-4

- 定義, 1-51
- パラレル DDL, 26-3
- 文の処理, 16-14
- ロールと権限, 30-19
- ロック, 27-27
- データ・ディクショナリ
 - DUAL 表, 2-7
 - SYSTEM 表領域, 2-2, 2-5, 3-7
 - アクセス, 2-2
 - 依存性の追跡, 21-3
 - オブジェクトの追加, 2-4
 - 監査証跡 (SYS.AUD\$), 2-5
 - キャッシュ, 7-10
 - 位置, 7-6
 - 行キャッシュ, 7-10
 - 更新, 2-5
 - 構造, 2-2
 - 最適化で使用するビュー, 22-15
 - 所有者, 2-3
 - 使用方法, 2-3
 - 表と列の定義, 16-11
 - 接頭辞が ALL のビュー, 2-6
 - 接頭辞が DBA のビュー, 2-6
 - 接頭辞が USER のビュー, 2-6
 - 定義, 1-28, 2-2
 - ディクショナリ管理の表領域, 3-8
 - データ・ファイル 1, 3-7, 32-24
 - 統計, 22-15
 - 統計データ, 23-31
 - パーティションの統計, 11-15
 - 動的パフォーマンス表, 2-7
 - 内容, 2-2, 7-10
 - プロシージャ, 18-17
 - バックアップ, 32-24
 - パブリック・シノニム, 2-4
 - ビューの接頭辞, 2-5
 - プロシージャの有効性, 18-18
 - ロック, 27-27
- データ・ディクショナリ・ビューの接頭辞, 2-5
- データの一貫性, 1-54
- データの整合性
 - 読み込み一貫性 (read consistency) も参照
 - マルチバージョンの一貫性モデル, 1-30
- データの変換
 - ANSI データ型, 12-20
 - SQL/DS および DB2 データ型, 12-20
 - プログラム・インタフェース, 8-24
- データ・ファイル
 - ROWID での表示, 12-15, 12-16
 - 一時, 3-17
 - オフライン化, 3-17
 - オンライン表領域またはオフライン表領域, 3-17
 - 回復不能, 32-17
 - 概要, 1-9, 1-11, 3-16
 - 制御ファイルに名前がある, 32-20
 - データ・ファイル 1, 3-7, 3-16
 - SYSTEM 表領域, 3-7, 3-16
 - バックアップ, 32-24
 - 内容, 3-16
 - パラレル回復, 32-11
 - 表領域との関連, 3-2
 - 物理データベース構造, 1-5
 - 読み込み専用, 3-11
 - 回復, 32-6
 - 読み込み専用表領域, 3-12
- データ・ファイル 1, 3-16
 - SYSTEM 表領域, 3-7, 3-16
 - データ・ディクショナリ, 3-7, 32-24
 - バックアップ, 32-24
- データ・ブロック, 1-10, 4-2
 - ROWID での表示, 12-15, 12-16
 - 空きリスト, 4-9
 - 空き領域の制御, 4-5
 - エクステントの結合, 4-13
 - エクステントへの割当て, 4-12
 - 概要, 4-2
 - 行ディレクトリ, 10-5
 - 行の格納方法, 10-5
 - 行を挿入できる領域, 4-8
 - クラスタ化, 10-46
 - クラスタによる共有, 10-44
 - 形式, 4-3
 - ディスクへの書き込み, 8-8
 - ハッシュ・キー, 10-53
 - バッファ・キャッシュに格納, 7-3
 - ブロック内の空き領域の結合, 4-9
 - ブロック・レベルの回復, 32-14
 - メモリーにキャッシュ, 8-8
 - 読み込み専用トランザクション, 27-31
- データ・ブロック内の空き領域の圧縮, 4-9
- データ変換
 - ANSI データ型, 12-20
 - SQL/DS および DB2 データ型, 12-20
 - プログラム・インタフェース, 8-24

データベース

アーカイブのモード, 32-17

アクセス制御

概要, 1-50

セキュリティ・ドメイン, 29-2

パスワード暗号化, 29-7

オープン, 5-7

ロールバック・セグメントの取得, 4-26

オープンとクローズ, 5-2

回復, 1-44, 32-2

拡張性, 6-4, 26-2, 26-34

管理

Enterprise Manager, 33-18

起動, 5-2

強制実行, 5-10

クローズ, 5-9

インスタンスの異常終了, 5-9, 32-4

クローン・データベース, 5-7

グローバル・データベース名, 33-4

構成, 5-4

構造

REDO ログ・ファイル, 1-12, 32-7

ROWID を使用して明確化, 12-16

エクステント, 1-10, 4-2, 4-10

スキーマ・オブジェクト, 1-10, 10-2

制御ファイル, 1-12, 32-20

セグメント, 1-10, 4-2, 4-16

データ・ディクショナリ, 1-28

データ・ファイル, 1-11, 3-1, 3-16

データ・ブロック, 1-10, 4-2, 4-3

表領域, 1-9, 3-1, 3-7

物理, 1-5, 1-11

プロセス, 1-13, 1-16, 8-1

メモリー, 1-13, 7-1

論理, 1-5, 1-8, 4-1

サイズ

判別する方法, 3-5

使用の制限, 29-15

スキーマが組み込まれている, 29-2

スタンバイ, 5-7, 32-25

制御ファイルに格納される名前, 32-20

定義, 1-8

停止, 5-9

ディスマウント, 5-10

バックアップ, 1-49, 32-22

分散, 1-34, 33-1

2 フェーズ・コミット, 1-34

概要, 1-32, 1-33, 33-1

グローバル・データベース名の変更, 7-11

サイト自律性, 33-15

ノード, 1-34

表のレプリケーション, 1-35

文の最適化, 23-29

マウント, 5-6

読み込み専用のオープン, 5-8

ネットワーク

分散データベース, 33-4

データベース管理システム (DBMS), 1-2

Oracle Server, 1-4

オブジェクト・リレーショナル DBMS, 13-2

原理, 1-21

データベース管理者 (DBA)

DBA ロール, 14-12, 30-20

データ・ディクショナリ・ビュー, 2-6

認証, 29-11

バックアップおよびリカバリに対する責任, 32-2

パスワード・ファイル, 29-12

データベース構造

REDO ログ・ファイル, 1-12, 32-7

ROWID を使用して明確化, 12-16

エクステント, 1-10, 4-2, 4-10

スキーマ・オブジェクト, 1-10, 10-2

制御ファイル, 1-12, 32-20

セグメント, 1-10, 4-2, 4-16

データ・ディクショナリ, 1-28, 2-1

データ・ファイル, 1-11, 3-1

データ・ブロック, 1-10, 4-2, 4-3

表領域, 1-9, 3-1, 3-7

物理, 1-5

プロセス, 1-13, 1-16, 8-1

メモリー, 1-13, 7-1

論理, 1-5, 1-8

データベース構造体

データ・ファイル, 3-16

データベース・スキーマのオブジェクト, 1-5

スキーマ・オブジェクトも参照

データベース・トリガー, 1-58, 20-1

トリガーも参照

データベースの構成

パラメータ・ファイル, 5-4

プロセスの構造, 8-2

データベース・バッファ

空き, 7-3

書き込み, 8-8

- キャッシュのサイズ, 7-4
- クリーン, 8-8
- 使用済み, 7-3, 8-8
- 使用中, 7-3
- 定義, 1-15, 7-3
- トランザクションのコミット, 8-10
- トランザクションをコミットした後, 17-5
- バッファ・キャッシュ, 7-3, 8-8
- 複数のバッファ・プール, 7-5
- データベース・ライター・プロセス (DBW_n), 8-8
 - 概要, 1-17
 - アクティブになるとき, 8-8
 - 最低使用頻度アルゴリズム (LRU), 8-8
 - 事前書込み, 8-9
 - チェックポイント, 8-8
 - チェックポイント時のディスクへの書込み, 8-11
 - 定義, 8-8
 - 複数の DBW_n プロセス, 8-8
 - メディア障害, 32-6
 - トレース・ファイル, 32-5
- データベース・リンク, 1-28
 - 拡張パーティション表名, 11-63
 - 概要, 33-5
 - 定義, 1-28
- データ・モデル, 1-21
- データ・ロック
 - 存続期間, 27-16
 - 段階的な拡大, 27-17
 - 変換, 27-17
- デッド・トランザクション, 32-4
 - ブロック・レベルの回復, 32-14
- デッドロック
 - 回避, 27-19
 - 検出, 27-18
 - 人工, 8-19
 - 定義, 27-17
 - 分散トランザクション, 27-19
- デフォルト値, 10-8
 - 制約が与える影響, 10-8, 28-19
 - ユーザー定義型, 14-4
- 伝播スケジューリング機能, 19-10

と

- 問合せ
 - DML, 16-3
 - IN 副問合せの最適化, 23-15

- ROWID による表走査の平行化, 26-4
- SAMPLE 句
 - コストベース最適化, 22-16
 - 一時セグメント, 4-18, 16-12
 - 位置の透過性, 33-14
 - インライン・ビュー, 10-16
 - 記述フェーズ, 16-12
 - 行のフェッチ, 16-11
 - 処理, 16-11
 - スター問合せ, 24-14
 - 選択性, 23-49
 - 定義, 23-3
 - 定義フェーズ, 16-12
 - デフォルト・ロック, 27-26
 - トリガーの使用法, 20-22
 - パーティションによって平行化される索引走査, 26-5
 - 平行処理, 26-2
 - 非定型, 26-28
 - ビュー問合せとのマージ, 10-14
 - ビュー問合せの最適化, 23-14
 - ビューとして格納, 1-22, 10-11
 - フェーズ, 27-5
- 複合
 - OR からの変換, 23-10
 - 最適化, 23-26
 - 定義, 23-3
- 複合索引, 10-22
 - 分散またはリモート, 33-10
 - 読み込み一貫性, 1-31, 27-6
- 問合せの処理の記述フェーズ, 16-12
- 問合せの処理の定義フェーズ, 16-12
- 問合せの行のフェッチ, 16-13
 - 埋込み SQL, 16-5
- 問合せの選択性, 23-49
- 等価結合
 - クラスタ結合, 24-5
 - ソート / マージ, 24-4
 - 定義, 23-3
 - ハッシュ結合, 24-7
- 透過的アプリケーション・フェイルオーバー, 32-14
- 統計
 - ANALYZE による, 22-13
 - B* ツリー索引またはビットマップ索引から, 22-13
 - DBMS_STATS を使用した生成と管理, 22-12
 - HIGH_VALUE および LOW_VALUE, 23-50
 - エクスポートとインポート, 22-9

- オブティマイザでの使用, 22-8, 22-9, 23-31
- オブティマイザの目標, 23-31
- オブティマイザ・モード, 23-30
- 拡張可能な最適化, 22-16
- キューイング, 19-10
- 述語の選択性, 22-9
 - ヒストグラム, 22-9, 22-11
 - ユーザー定義, 22-17
- 推定, 22-14
 - Block サンプルング, 22-14
 - ROW サンプルング, 22-14
- チェックポイント, 8-11
- パーティションとサブパーティション, 22-12
- パーティション表とパーティション索引, 11-15
- ユーザー定義統計, 22-17
- トランザクション, 1-52, 17-1
 - アドバンスト・キューイング, 19-3
 - アプリケーションの終了, 17-5
 - インダウト
 - 手動による解決, 1-35
 - 自動解決, 1-35, 5-8, 17-7
 - 部分的に使用可能なセグメントの使用, 4-29
 - ロールバック・セグメント, 4-24
 - ロールバック・セグメントのアクセスの制限, 4-29
 - 開始, 17-4
 - 回復, 32-4
 - 概要, 1-52
 - コミット, 1-53, 8-10, 17-3, 17-5
 - グループ・コミット, 8-10
 - ロールバック・セグメントの使用, 4-21
 - コミット前に書き込まれる REDO ログ・ファイル, 8-10
 - システム変更番号, 8-10
 - システム変更番号の割当て, 17-5
 - 終了, 17-4
 - 一貫したデータ, 16-14
 - 手動ロック, 27-31
 - 自律型, 17-8
 - PL/SQL ブロック内, 17-9
 - セーブポイント, 1-54, 17-7
 - 説明, 17-2
 - 直列可能, 27-7
 - 定義と制御, 16-14
 - データ・ブロック内で使用される領域, 4-5
 - デッド, 32-4
 - デッドロック, 17-4
 - トランザクション制御文, 16-5
 - トランザクションの制御, 16-14
 - トリガー, 20-22
 - 同時実行性, 27-16
 - パラレル DML での 2 フェーズ・コミット, 26-37
 - 非同期処理, 19-2
 - ブロック・レベルの回復, 27-21, 32-14
 - 分散, 1-31
 - 2 フェーズ・コミット, 1-34, 17-7, 33-13
 - 自動解決, 8-12
 - デッドロック, 27-19
 - パラレル DDL の制限事項, 26-26
 - パラレル DML の制限事項, 26-26, 26-42
 - 文レベルのロールバック, 17-4
 - 読み込み一貫性, 1-30, 27-6
 - 読み込み専用, 1-31, 27-7
 - ロールバック・セグメントには割り当てられない, 4-20
 - 離散トランザクション, 16-14, 17-8
 - デッドロック, 27-17
 - ロールバック, 1-53, 17-6
 - オフライン表領域, 4-30
 - 部分的な, 17-6
 - ロールバック・セグメントの使用, 4-20
 - ロールバック・セグメント, 4-20
 - ロールバック・セグメントへの書き込み, 4-21
 - ロールバック・セグメントへの配分, 4-21
 - ロールバック・セグメントへの割当て, 4-20
 - トランザクション集合の一貫性, 27-10, 27-11
 - トランザクション制御文, 1-51, 16-5
 - 自律型 PL/SQL ブロック内, 17-10
 - トランザクションのコミット
 - 概要, 1-53
 - グループ・コミット, 8-10
 - 高速コミット, 8-10
 - 実現, 8-10
 - 定義, 17-2
 - パラレル DML, 26-37
 - トランザクション表, 4-20
 - 回復時にリセット, 8-11
 - トランスポータブル表領域, 3-14
 - トリガー, 1-58, 20-1, 21-8
 - AFTER トリガー, 20-9
 - BEFORE トリガー, 20-9
 - INSTEAD OF トリガー, 20-11
 - オブジェクト・ビュー, 15-5
 - INVALID 状態, 21-2, 21-6

- Java, 20-7
- Oracle Forms トリガーと対比, 20-3
- UNKNOWN では起動されない, 20-7
- アクション, 20-7
 - タイミング, 20-9
- 依存性の管理, 20-24, 21-6
 - 使用可能なトリガー, 20-20
- イベント, 20-6
- 各部分, 20-5
- カスケード, 20-4
- 監査, 31-8
- 概要, 1-58, 20-2
- 記憶領域, 20-23
- 起動 (実行), 20-2, 20-23
 - 実行されるステップ, 20-20
 - タイミング, 20-20
 - 必要な権限, 20-23
- 共有 SQL 領域, 7-9
- 行, 20-8
- 使用可能または使用禁止, 20-20
- 使用方法, 20-3
- 実行する権限, 30-7
 - ルール, 30-18
- スキーマ・オブジェクトの依存性, 20-20, 20-24
- 制限, 20-7, 26-42
 - ダイレクト・ロード・インサート, 25-11
 - パラレル DML, 26-41
- 制約が適用される, 20-20
- 制約と対比, 20-5
- タイプ, 20-8
- データ・アクセス, 20-22
- データ整合性の維持, 1-58
- データ整合性の施行, 28-4
- パブリッシュ / サブスクライブのサポート, 20-17
- ビューでは定義不可, 10-12
- 複数トリガーの起動順序, 20-20
- 文, 20-8
- プログラム・ユニット, 1-55
- プロシージャと対比, 20-2
- ユーザー定義型, 14-6
- 例, 20-10, 20-13, 20-22

取消し, 1-10

- ロールバックも参照

トレース・ファイル, 8-15

- ARCn トレース・ファイル, 32-19
- DBWn トレース・ファイル, 32-5
- LGWR トレース・ファイル, 8-10

- トレース・ファイルに記録される内部エラー, 8-15
- 同一キー索引, 11-30, 11-34
- 同一行の書込みが書込みを阻止するか, 27-11
- 同一レベル・パーティション化, 11-23
 - 1 ディメンション, 11-24
 - LOB 列, 11-38
 - 索引構成表のオーバーフロー, 11-41, 11-43
 - 例, 11-25, 11-30, 11-32
 - レンジ・パーティション化, 11-24
 - ローカル索引, 11-29
- 同期データ伝播, 34-16
- 同時実行性
 - 制限, 1-42, 25-11
 - データベース当り, 29-19
 - ユーザー当たりの, 29-17
 - 説明, 27-2
 - ダイレクト・ロード・インサート, 25-11
 - 定義, 1-29
 - トランザクション, 27-16
 - ロックを使用して施行, 1-31
 - パーティションのメンテナンス, 11-49
- 動的 SQL
 - DBMS_SQL パッケージ, 16-19
 - 埋込み, 16-19
 - 名前変換, 18-19
- 動的述語
 - セキュリティ・ポリシー内, 30-21
- 動的パーティション化, 26-6
- 動的パフォーマンス表 (VS 表), 2-7
- ドメイン索引, 10-41
 - 拡張可能な最適化, 22-16
 - ユーザー定義統計, 22-17
- ドライバ, 8-25

な

- 内容を保証しない書込み, 27-11
- 内容を保証しない読込み, 27-2, 27-11
- 名前付きユーザー・ライセンス, 29-20

に

- 任意アクセス制御, 1-38, 29-2
- 認証
 - Oracle, 29-7
 - オペレーティング・システム, 29-4
 - 公開鍵インフラストラクチャ, 29-5

- 説明, 29-3
- データベース管理者, 29-11
- ネットワーク, 29-4
- 複数層, 29-8
- リモート, 29-6
- 認証局, 29-5

ね

- ネストした表, 10-9, 13-11
 - INSTEAD OF トリガー, 15-5
 - 索引, 14-6
 - 索引構成表, 10-36
 - キー圧縮, 10-29
 - 制限, 26-27
 - ビュー内で更新, 15-5
- ネスト・ループ・ジョイン, 24-2
 - コストベース最適化, 24-9
- ネットワーク
 - 2 タスク・モード, 8-23
 - Net8, 6-4, 33-4
 - Oracle Names, 33-4
 - Oracle の使用, 1-7, 1-36
 - クライアント / サーバー・アーキテクチャでの使用, 6-2
 - 障害, 32-3
 - 通信プロトコル, 6-5, 8-25, 8-26
 - ディスパッチャ・プロセス, 8-14, 8-16
 - ドライバ, 8-25
 - ネットワーク認証サービス, 29-4
 - 分散データベースの使用, 33-2
 - リスナー・プロセス, 6-6, 8-14
- ネットワーク・リスナー・プロセス, 6-6
 - サービス名, 6-6
 - 接続要求, 8-14, 8-16
 - 専用サーバーの例, 8-24
 - マルチスレッド・サーバーの例, 8-20

の

- ノード
 - パラレル・サーバーでのディスク親和性, 26-44
 - 分散データベース, 1-34

は

- 排他モード, 4-27

- 排他ロック
 - RX ロック, 27-23
 - 行ロック (TX), 27-20
 - 表ロック (TM), 27-21
- ハイブリッド構成
 - アドバンスド・レプリケーション, 34-13
- 配列
 - VARRAY のサイズ, 13-10
 - 可変 (VARRAY), 13-10
- 配列処理, 16-13
- 発行
 - DDL 文, 20-19
 - DML 文, 20-19
 - システム・イベント
 - 起動と停止, 20-18
 - サーバー・エラー, 20-18
 - トリガーの使用, 20-17
 - ログオンおよびログオフ・イベント, 20-18
- ハッシュ・クラスタ, 1-28, 10-48
 - 概要, 1-28
 - 記憶領域, 10-49
 - 索引と対比, 10-48
 - 衝突の解決, 10-51
 - 走査, 23-33, 23-38, 23-40
 - 単一表ハッシュ・クラスタ, 10-55
 - 領域の割当て, 10-53
 - ルート・ブロック, 10-53
- ハッシュ結合, 24-7
 - HASH_AREA_SIZE パラメータ, 24-8
 - HASH_MULTIBLOCK_IO_COUNT パラメータ, 24-8
 - 索引結合, 23-35, 23-48
- 発注の例
 - オブジェクト型, 13-2, 13-4
- 反復可能読み込み, 27-3
- 反復不能読み込み, 27-3, 27-11
- バイナリ・データ
 - BFILE, 12-13
 - BLOB, 12-12
 - RAW および LONG RAW, 12-13
- バインド変数
 - 最適化, 23-50
 - ユーザー定義型, 13-13
- バックアップ
 - Recovery Manager, 1-50, 32-14
 - 概要, 1-44, 32-22
 - 制御ファイル, 32-24

- 全体データベース・バックアップ, 1-49, 32-22
 - タイプ, 1-48
 - データ・ファイル, 32-24
 - パラレル, 32-16
 - 部分, 1-49, 32-23
 - 読み込み専用表領域, 32-25
- バックエンド, 6-2
- バックグラウンド・プロセス, 1-17, 8-6
 - プロセスも参照
 - 概要, 1-17
 - 図, 8-6
 - 説明, 8-6
 - トレース・ファイル, 8-15
- バックアップ
 - Export を使用した補助, 32-25
- バッファ
 - REDO ログ・バッファ, 1-15, 7-5
 - データベース・バッファ・キャッシュ, 1-15, 7-3, 8-8
 - 増分チェックポイント, 8-8
 - ファースト・スタート・チェックポイント, 32-13
- バッファ・キャッシュ, 7-3, 8-8
 - 拡張バッファ・キャッシュ (32 ビット), 7-13
 - 複数のバッファ・プール, 7-5
- バッファ・プール, 7-5
- パーティション, 11-2, 11-13
 - DATE データ型, 11-14, 11-21
 - DML パーティション・ロック, 11-45
 - EXCHANGE PARTITION, 11-11
 - LONG および LONG RAW の制限事項, 11-14
 - OLTP データベース, 11-6
 - VLDB, 11-5
 - 拡張パーティション表名, 11-62
 - グローバル索引, 11-32, 11-59
 - 索引のパーティション化, 11-28, 11-36
 - 親和性, 26-44
 - 実行計画, 11-11, 11-15
 - 制限
 - 拡張パーティション表名, 11-63
 - データ型, 11-14, 11-21
 - ビットマップ索引, 11-14
 - セグメント, 4-16, 4-17
 - 統計, 11-15, 22-12
 - 同一キー索引, 11-30
 - 同一レベル・パーティション化, 11-23
 - 1 ディメンション, 11-24
 - LOB 列, 11-38
 - 索引構成表のオーバーフロー, 11-41, 11-43
 - 例, 11-25, 11-30, 11-32
 - レンジ・パーティション化, 11-24
 - ローカル索引, 11-29
 - 同時実行のメンテナンス操作, 11-49
 - 動的パーティション化, 26-6
 - ハッシュ・パーティション化, 11-17
 - パーティション化キー, 11-15, 11-19
 - パーティション化の基本的なモデル, 11-13
 - パーティション絞込み, 11-4
 - パーティションの再作成, 11-59
 - パーティションの参照, 11-19
 - パーティションの透過性, 11-10
 - パーティション・バウンド, 11-20
 - パーティション・ブルニング, 11-4
 - DATE データ型, 11-22
 - 索引, 11-36
 - ディスクのストライプ化, 26-45
 - ブロック範囲によるパラレル化, 26-4
 - パーティション名, 11-19
 - パーティション・ワイズ結合, 11-5
 - パラレル DDL, 26-28
 - パラレル化のルール, 26-23, 26-24
 - パラレル問合せ, 26-4
 - 非同一キー索引, 11-31, 11-35
 - 表のパーティション化, 11-26
 - ビットマップ索引, 10-35
 - 物理属性, 11-27, 11-37
 - マージ, 11-16
 - マテリアライズド・ビュー, 10-18, 11-2
 - メンテナンス操作, 11-47
 - 利点, 11-5, 11-7
 - レンジ・パーティション化, 11-15
 - ディスクのストライプ化, 26-45
 - ローカル索引, 11-29, 11-58
 - 並列に作成, 11-30
 - ロギングなしモード, 25-7
- パーティション化
 - LOB
 - DML ロック, 11-46
 - メンテナンス操作, 11-56
 - LOB 列を持つ表, 11-38
- パーティション化キー, 11-15, 11-19
 - 複数列からなるキー, 11-22
- パーティション絞込み, 11-4
- パーティションのマージ, 11-16

- パーティション・ビュー, 11-11
- パーティション・プルニング, 11-4, 26-4, 26-45
 - DATE データ型, 11-22
 - EXPLAIN PLAN, 11-22
 - 索引, 11-36
 - 索引パーティション, 11-4
- パーティション・ワイズ結合, 11-5
- パスワード
 - アカウントのロック, 29-7
 - 暗号化, 29-7
 - 管理者権限, 5-3
 - 時間切れ, 29-7
 - 接続, 8-5
 - データベース・ユーザーの認証, 29-7
 - パスワードの再使用, 29-8
 - パスワード・ファイル, 29-12
 - パスワードを指定しない接続, 29-4
 - 複雑度の検証, 29-8
 - ロールでの使用, 1-41
- パッケージ, 18-4, 18-11
 - OUTLN_PKG, 22-7
 - 格納, 18-16
 - 監査, 31-7
 - 概要, 1-24
 - キューイング, 19-4
 - 共有 SQL 領域, 7-9
 - 権限
 - 構成体ごとに分割, 30-9
 - 実行, 30-7, 30-9
 - 実行, 16-16, 18-17
 - セッションの状態, 21-6
 - 提供されるパッケージ, 18-16
 - 起動者権限または定義者権限, 30-8
 - 動的 SQL, 16-19
 - パブリック, 18-15
 - プライベート, 18-15
 - プログラム・ユニット, 1-55
 - 有効性, 18-18
 - 利点, 18-14
 - 例, 18-11, 30-9
 - ロック用, 27-37
- パフォーマンス
 - DSS データベース, 11-8, 26-34
 - I/O, 11-9
 - Oracle Parallel Server と DML ロック, 11-47
 - SGA のサイズ, 7-12
 - 回復, 32-13
 - クラスタ, 10-46
 - グループ・コミット, 8-10
 - 向上させる構造体, 1-25
 - 索引作成, 10-22
 - 実行計画の表示, 22-5
 - 制約が与える影響, 28-6
 - ソート操作, 3-13
 - 同一キー索引と非同一次元索引, 11-35
 - 動的パフォーマンス表 (VS), 2-7
 - パーティション, 11-8
 - パッケージ, 18-15
 - パラレル回復, 32-11
 - リソース制限, 29-15
- パブリック・ロールバック・セグメント, 4-25
- パブリッシュ / サブスクライブのサポート, 19-10
 - イベントの発行, 20-18
 - トリガー, 20-17
 - 非同期通知, 19-11
 - メッセージの伝播, 19-9
 - リスニング機能, 19-11
 - ルールベースのサブスクライバ, 19-6
- パラメータ
 - 各国語サポート, 5-5
 - 記憶域, 4-5
 - 記憶領域, 4-11
 - 初期化, 5-4
 - 初期化パラメータも参照
 - ロック動作, 27-19
- パラメータ・ファイル, 5-4
 - 起動時の使用, 5-5
 - 例, 5-4
- パラレル DDL, 26-27
 - エクステンツの割当て, 26-30
 - 関数, 26-43
 - 制限
 - LOB, 26-28
 - オブジェクト型, 26-27, 26-28
- パーティション表とパーティション索引, 26-28
 - ローカル索引の作成, 11-30
- パラレル化ルール, 26-18
 - 並行性のタイプ, 26-3
- パラレル DELETE, 26-19
- パラレル DML, 26-32
 - PARALLEL DML の使用可能化, 26-35
 - アプリケーション, 26-34
 - 回復, 26-37
 - 関数, 26-43

- 制限, 26-40
 - オブジェクト型, 26-27, 26-41
 - リモート・トランザクション, 26-42
 - トランザクション・モデル, 26-36
- パラレル化ルール, 26-18
- ビットマップ索引, 10-31
- 並行性のタイプ, 26-3
- 並行度, 26-18, 26-20
- リソースのロックとエンキュー, 26-38
- ロールバック・セグメント, 26-36
- パラレル SQL, 26-2
 - Parallel Server, 26-1
- パラレル実行も参照
- インスタンス・グループ, 26-17
- オブティマイザ, 26-10
- コーディネータ・プロセス, 26-6
 - ダイレクト・ロード・インサート, 25-3
- サーバー・プロセス, 26-6
 - NEXT エクステント・サイズ, 25-9
 - ダイレクト・ロード・インサート, 25-3, 25-8
- サマリー表またはロールアップ表, 26-28
- 実行計画にある操作, 26-10
- パラレル化ルール, 26-18
- パラレル実行サーバーの数, 26-7
- パラレル実行サーバーへの行の割当て, 26-10
- 並行度, 26-15
- マルチスレッド・サーバー, 26-8
- パラレル UPDATE, 26-19
- パラレル回復, 32-10, 32-16
- パラレル実行, 26-2
 - パラレル SQL も参照
 - インターオペレータ並行性, 26-12
 - イントラオペレータ並行性, 26-12
 - コーディネータ, 25-3, 26-6
 - サーバー, 25-3, 26-6
 - NEXT エクステント・サイズ, 25-9
 - 一時セグメント, 25-8
 - 索引メンテナンス, 25-8
 - 全表走査, 26-5
 - パーティション表とパーティション索引, 26-4
- パラレル実行コーディネータ, 26-6
 - ダイレクト・ロード・インサート, 25-3
- パラレル実行サーバー, 26-6
 - ダイレクト・ロード・インサート, 25-3
 - NEXT エクステント・サイズ, 25-9
 - 一時セグメント, 25-8
 - 索引メンテナンス, 25-8

- パラレル問合せ, 26-25
 - オブジェクト型, 26-27
 - 制限, 26-27
- 関数, 26-43
- 索引構成表, 26-26
- パラレル化ルール, 26-18
- ビットマップ索引, 10-31
- パラレル・バックアップ操作, 32-16

ひ

- 非一意索引, 10-22
- 非永続キュー, 19-10
- 比較メソッド, 13-6
- 非コミット読み込み, 27-3
- ヒストグラム, 22-9
- 非正規化ビュー
 - スター・スキーマ, 24-15
- 非接続環境
 - アドバンスド・レプリケーションと同様, 34-12
- 非同一キー索引, 11-31, 11-35
 - グローバル・パーティション索引, 11-33
- 非同一レベル結合
 - 定義, 23-3
- 非同期 I/O
 - パラレル回復, 32-11
- 非同期処理, 19-2
- 表
 - DUAL, 2-7
 - LOB 列
 - パーティション化, 11-38
 - PARTITION オプション, 11-62
 - SUBPARTITION オプション, 11-62
 - VALIDATE または NOVALIDATE 制約, 28-21
 - 依存ビューに対する影響, 21-5
 - 一時, 10-10
 - セグメント, 4-18
 - オブジェクト表, 13-2, 13-7
 - 仮想, 15-2
 - 索引, 14-6
 - 制約, 14-5
 - トリガー, 14-6
 - 拡張パーティション表名, 11-62
 - 仮想またはビュー, 1-23
 - 監査, 11-61, 31-7
 - 概要, 1-22, 10-3
 - キュー表, 19-4, 19-11

- クラスタ化, 10-44
- 権限, 30-4
- 索引, 10-21
- 索引構成
 - キー圧縮, 10-29, 10-36
- 索引構成表, 10-35
 - 論理 ROWID, 10-37, 12-18
- サマリー表またはロールアップ表, 26-28
- 参照表, 24-14
- 使用するトリガー, 20-2
- 事実表
 - スター問合せ, 24-14
- 実表, 1-23
 - データ・ディクショナリでの使用, 2-2
 - ビューとの関連, 10-12
- 正規化
 - スター・スキーマ, 24-15
- 正規化された表と正規化されていない表, 1-28, 10-19
- 整合性制約, 28-2, 28-5
- 整合性制約を含む, 1-57
- 制約を使用可能または使用禁止にする, 28-21
- 全表走査とバッファ・キャッシュ, 7-4
- 単一表ハッシュ・クラスタ, 10-55
- ディメンション
 - スター問合せ, 24-14
- データ・ウェアハウスでのリフレッシュ, 26-34
- データの格納方法, 10-4
- 動的パーティション化, 26-6
- ネストした表, 10-9, 13-11
 - 索引, 14-6
- ハッシュ, 10-53
- パーティション, 11-2, 11-26
- パーティションについての権限, 11-61
- パラレル DDL 記憶領域, 26-30
- パラレル作成, 26-28
- パラレル実行での STORAGE 句, 26-30
- パラレルの表走査, 26-4
- 表の別名, 14-8
- 表名
 - 列名の修飾, 14-7, 14-8
- 表領域に含まれる, 10-5
- 表領域の指定, 10-5
- ビューでの提示, 10-11
- 領域割当ての制御, 10-4, 25-8
- 履歴データ, 26-35
- 列の最大数, 10-12

- レプリケーション, 1-35
- ロギングなしモード, 25-7
- ロック, 11-45, 27-21, 27-23, 27-24
- 表ディレクトリ, 4-4
- 表領域, 3-7
 - MINIMUM EXTENT
 - パラレル DML, 25-10
 - SYSTEM 表領域を参照
 - 一時, 1-42, 3-12
 - ユーザーのデフォルト, 29-13
 - 一時セグメントに使用, 4-15, 4-18
 - オフライン, 1-9, 3-9, 3-17
 - 再マウント時にオフラインのまま, 3-10
 - 索引データとの関係, 3-11
 - 読み込み専用にできない, 3-12
 - オブジェクト作成のデフォルト, 1-41, 29-13
 - オンライン, 1-9, 3-9, 3-17
 - 概要, 1-9, 3-7
 - サイズ, 3-5
 - スキーマと対比, 10-2
 - 説明, 3-7
 - 他のデータベースへの移動またはコピー, 3-14
 - ディクショナリ管理, 3-8
 - データ・ファイルとの関連, 3-2
 - トランспортаブル, 3-14
 - 表に対して指定する方法, 10-5
 - ユーザーからのアクセスの取消し, 29-14
 - 読み込み専用, 3-11
 - オブジェクトの削除, 3-12
 - 推移モード, 3-11
 - 読み込み専用推移モード, 3-11
 - 領域割当て, 3-8
 - ローカル管理, 3-9
 - 一時表領域, 3-13
 - ロギングなしモード, 25-7
 - ロック, 27-30
 - 割当て制限, 1-41, 1-42, 29-13
 - 制限ありと制限なし, 29-13
 - デフォルトなし, 29-13
- 表領域の Point-in-Time 回復
 - クローン・データベース, 5-7
- ヒント
 - INDEX, 24-15
 - INDEX_FFS, 23-34
 - INDEX_JOIN, 23-35
 - MERGE, 23-16
 - MERGE_AJ および HASH_AJ, 24-13

- MERGE_SJ および HASH_SJ, 24-13
- OPTIMIZER_MODE と OPTIMIZER_GOAL を上書き, 23-32
- ORDERED, 24-10, 24-15
- PARALLEL, 26-15
- PARALLEL_INDEX, 26-15
- PUSH_JOIN_PRED, 24-12
- STAR, 24-15
- USE_HASH, 24-7
- オブティマイザによる選択の上書き, 23-49
- 拡張可能な最適化, 22-17
- サンプル・アクセス・パスを上書きできない, 23-49
- ビットマップ索引, 10-30
- NULL, 10-8, 10-34
- カーディナリティ, 10-31
- スター型変換, 24-16
- 走査, 23-35
- パーティション表, 11-14
- パラレル問合せおよび DML, 10-31
- ビュー, 1-22, 10-11
 - INSTEAD OF トリガー, 20-11
 - INVALID 状態, 21-2
 - NLS パラメータ, 10-14
 - NULL に対する NULL 以外の値, 24-11
 - SQL 関数, 10-14
 - 依存性の状態, 21-5
 - インライン・ビュー, 10-16
 - オブジェクト・ビュー, 10-16, 15-1
 - 更新可能性, 15-5
 - 格納方法, 10-12
 - 監査, 31-7, 31-8
 - 概要, 1-22, 10-11
 - 疑似列, 20-13
 - 権限, 30-5
 - 更新可能性, 10-15, 15-5, 20-13
 - 固定ビュー, 2-7
 - コンパイルの前提条件, 21-5
 - 最適化, 23-14
 - 索引, 10-14
 - 式を含む, 20-13
 - 使用方法, 10-13
 - 実表, 1-23
 - 実表の変更, 21-5
 - スキーマ・オブジェクトの依存性, 10-15, 21-4, 21-8
 - 制約が間接的に影響する, 28-5

- 制約とトリガーは定義不可, 10-12
- セキュリティ・アプリケーション, 30-6
- 選択表示結合ビュー, 23-14
- 定義の展開, 21-5
- データ・ディクショナリ
 - 更新可能な列, 10-15
 - ユーザー・アクセス可能ビュー, 2-3
- 統計, 22-15
- パーティションの統計, 11-15
- パーティション・ビュー, 11-11
- ヒストグラム, 22-11
- 非正規化
 - スター・スキーマ, 24-15
- 複合ビューのマージ, 23-15
- 変更, 20-12
 - 変更可能, 20-13
 - 本質的に変更可能, 20-13
- マテリアライズド・ビュー, 1-23, 10-17
 - スナップショットと同義, 1-23, 34-3
- 列の最大数, 10-12

ふ

- ファースト・スタート・オン・デマンド・ロールバック, 32-10
- ファースト・スタート・チェックポイント, 32-13
- ファースト・スタート・パラレル・ロールバック, 32-14
- ファースト・スタート・リカバリ, 32-13
- ファイル
 - ALERT およびトレース・ファイル, 8-10, 8-15
 - LISTENER.ORA, 6-6
 - Oracle データベース, 1-9, 1-11, 32-6
 - 制御ファイル、データ・ファイル、REDO ログ・ファイルも参照
 - オペレーティング・システム, 1-5
 - 初期化パラメータ, 5-4, 5-5
 - ダンプ・ファイルの Export と Import, 14-19
 - パスワード, 29-12
 - 管理者権限, 5-3
- ファイル管理ロック, 27-30
- ファイン・グレイン・アクセス・コントロール, 30-21
- ファジー読み込み, 27-3
- ファンクション
 - PL/SQL
 - プロシージャと対比, 1-55
- 関数

- PL/SQL
 - プロシージャと対比, 18-2
- ファンクションベース索引, 10-24
 - DISABLED, 21-7, 21-8
 - UNUSABLE, 21-8
 - 依存性, 10-25, 21-7
 - 権限, 10-25, 21-7
- フェイルオーバー・サイト
 - アドバンスド・レプリケーションと同様, 34-7
- 不完全なオブジェクト型, 14-16
- 複合索引, 10-22
- 複合ビューのマージ, 23-15
- 副問合せ, 16-12
 - CHECK 制約が禁止する, 28-17
 - DDL 文, 26-28
 - DML 文
 - 直列可能分離, 27-14
 - IN 副問合せの最適化, 23-15
 - NOT IN, 24-13
 - 問合せも参照
 - インライン・ビュー, 10-16
 - 結合への変換, 23-12
 - 問合せ処理, 16-12
 - リモート更新, 33-11
- 不整なパラレル DML 作業負荷, 26-17
- 付与
 - EXECUTE ユーザー定義型, 14-13
 - 権限とロール, 30-3
- フロントエンド, 6-2
- 物理データベース構造, 1-5, 1-11
 - REDO ログ・ファイル, 1-12, 32-7
 - 制御ファイル, 1-12, 32-20
 - データ・ファイル, 1-11, 3-16
- 部分バックアップ, 32-23
- ブランチ・ブロック, 10-27
- ブロック
 - データベース, 4-3
 - データ・ブロックも参照
 - ブロック・レベルの回復, 32-14
 - 無名, 16-14, 18-9
- ブロック・サーバー・プロセス (BSP), 27-7
- ブロック・レベルの回復, 27-21, 32-14
- 文
 - SQL 文を参照
- 分散処理環境
 - クライアント / サーバー・アーキテクチャ, 1-32, 6-2

- 説明, 1-32, 6-2
- データ操作言語, 16-10
- 分散データベース, 33-7
- マテリアライズド・ビュー (スナップショット), 10-17
- 分散データベース, 33-1
 - 2 フェーズ・コミット, 1-34, 33-12
 - 異機種間, 33-8
 - 依存スキーマ・オブジェクト, 21-10
 - 監査, 31-6
 - 管理ツール, 33-18
 - 概要, 1-33, 33-2
 - クライアント / サーバー・アーキテクチャ, 6-2
 - グローバル・オブジェクト名, 33-6
 - 異なる Oracle バージョン, 33-6
 - サーバーをクライアントとしても使用可能, 6-2
 - サイト自律性, 33-15
 - ジョブ・キュー・プロセス (SNPn), 1-19, 8-13
 - 図, 33-2
 - データベース・リンク, 33-5
 - デッドロック, 27-19
 - 透過性, 33-13
 - ノード, 33-2
 - 表のレプリケーション, 1-35
 - 分散更新, 33-11
 - 分散問合せ, 33-11
 - 文の最適化, 23-29
 - メッセージの伝播, 19-9
 - リカバラ・プロセス (RECO), 8-12
 - リモート依存性, 21-10
 - リモート問合せと更新, 33-10
- 分散データベースでのネーム変換, 33-6
- 分散問合せの最適化, 33-10
- 分散トランザクション
 - 2 フェーズ・コミット, 1-34, 17-7
 - 最適化, 23-29
 - サンプル表スキャンがサポートされない, 23-33
 - 定義, 33-12
 - ノードへの文のルーティング, 16-11
 - パラレル DDL の制限事項, 26-26
 - パラレル DML の制限事項, 26-26, 26-42
 - 分散型の文, 23-4
- 文トリガー, 20-8
 - トリガーも参照
 - 起動されるタイミング, 20-20
 - 説明, 20-8
- 文へのビューのマージ, 23-14

- 分離レベル
 - コミット読み込み, 27-8
 - 設定, 27-8, 27-31
 - 選択, 27-13
- 文レベルの読み込み一貫性, 27-6
- プライベート SQL 領域
 - カーソル, 7-9
 - 持続領域, 7-8
 - 実行時領域, 7-8
 - 説明, 7-8
 - どのように管理されるか, 7-9
- プライベート・ロールバック・セグメント, 4-25
- プリコンパイラ
 - FIPS フラガー, 16-6
 - 埋込み SQL, 16-5
 - カーソル, 16-11
 - ストアド・プロシージャ, 16-18
 - バインド変数, 16-12
 - 無名ブロック, 16-17
- プログラム・インタフェース, 8-24
 - 2 タスク・モード, 8-23
 - Oracle 側 (OPI), 8-25
 - 概要, 1-19
 - 構造, 8-25
 - ユーザー側 (UPI), 8-25
- プログラム・グローバル領域 (PGA), 1-16, 7-13
 - サイズ, 7-15
 - 内容, 7-14
 - 非共有と書き込み可能, 7-13
 - マルチスレッド・サーバー, 8-19
 - 割当て, 7-14
- プログラム・ユニット, 1-24, 16-14, 18-2
 - 共有ブール, 7-9
 - コンパイルの前提条件, 21-5
- プロシージャ, 16-15, 18-1, 18-6, 21-8
 - INVALID 状態, 21-2, 21-6
 - 依存性の追跡, 21-6
 - カーソル, 16-17
 - 格納, 18-16
 - 監査, 31-7, 31-8
 - 外部参照, 18-10, 18-19
 - 外部プロシージャ, 16-19, 18-11
 - 起動者権限, 18-9, 30-8
 - 使用されるロール, 30-19
 - 提供されるパッケージ, 30-8
 - 共有 SQL 領域, 7-9
 - 権限
 - 作成または変更, 30-8
 - 実行, 30-7
 - パッケージ内での実行, 30-9
- 現ユーザー, 18-10
- コンパイルの前提条件, 21-5
- 実行, 16-16, 18-17
- ストアド・プロシージャ, 16-15, 16-18, 18-2
- セキュリティの強化, 18-7, 30-7
- 提供されるパッケージ, 18-16
 - 起動者権限または定義者権限, 30-8
- 定義者権限, 18-9, 30-7
 - ロールは使用禁止, 30-18
- トリガー, 20-2
- ファンクションと対比, 1-55, 18-2
- 無名ブロックと対比, 18-9
- 有効性, 18-18
- 利点, 18-7
- リモート・プロシージャ・コール, 33-11
- 例, 18-6, 30-9
- プロシージャ・コール
 - リモート, 33-11
- プロシージャの名前変換, 18-19
- プロシージャ・レプリケーション, 34-16
 - 競合の検出, 34-16
 - ラッパー, 34-16
- プロセス, 8-2
 - Oracle, 1-16, 8-5
 - アーカイバ (ARCn), 1-18, 8-12, 32-19
 - 回復時, 32-12
 - 概要, 1-16
 - キュー・モニター (QMNn), 1-19, 8-13, 19-6
 - 構造, 8-2
 - サーバー, 1-17, 1-33, 8-5
 - 共有, 8-14, 8-19
 - 専用, 8-22
 - システム・モニター (SMON), 1-18, 8-11
 - シャドウ, 8-22
 - 障害, 32-3
 - ジョブ・キュー (SNPN), 1-19, 8-13
 - メッセージの伝播, 19-9
 - 専用サーバー, 8-19
 - チェックポイント, 8-8
 - チェックポイント (CKPT), 1-18, 8-11
 - ディスパッチャ (Dnnn), 1-19, 8-14
 - データベース・ライター (DBWn), 1-17, 8-8
 - トレース・ファイル, 8-15
 - バックグラウンド, 1-17, 8-6

図, 8-6
パラレル実行コーディネータ, 26-6
ダイレクト・ロード・インサート, 25-3
パラレル実行サーバー, 26-6
NEXT エクステント・サイズ, 25-9
ダイレクト・ロード・インサート, 25-3, 25-8
ブロック・サーバー (BSP), 27-7
分散トランザクションの解決, 8-12
プロセス・モニター (PMON), 1-18, 8-11
マルチスレッド・サーバー, 8-16
クライアントの要求, 8-17
人工デッドロック, 8-19
マルチ・プロセス Oracle, 8-2
ユーザー, 1-16, 8-4
PGA の割当て, 7-14
サーバー・プロセスの共有, 8-14
手動アーカイブ, 32-20
障害からの回復, 8-11
リカバラ (RECO), 1-19, 8-12
インダウト・トランザクション, 1-35
リスナー, 6-6, 8-14
共有サーバー, 8-16
ログ・ライター (LGWR), 1-18, 8-9
ロック (LCK0), 1-19, 8-13
プロセス・グローバル領域 (PGA), 7-13
プログラム・グローバル領域も参照
プロセス・モニター・プロセス (PMON)
説明, 1-18, 8-11
タイムアウト・セッションのクリーン・アップ,
29-17
ネットワーク障害, 32-3
パラレル DML プロセス回復, 26-37
プロセス障害, 32-3
プロファイル
概要, 1-42
使用する場合, 29-18
パスワード管理, 29-7

へ

並行度, 26-18, 26-20
問合せ操作間, 26-12
パラレル SQL, 26-7, 26-15
ヘッダー
行断片, 10-5
データ・ブロック, 4-4
変更エラーとトリガー, 20-21

変数

埋込み SQL, 16-5
オブジェクト変数, 15-4
ストアド・プロシージャ内, 16-17
バインド変数
最適化, 23-50
ユーザー定義型, 13-13
別の行の書込みが書込みを阻止するか, 27-11
別名
副問合せを修飾 (インライン・ビュー), 10-16
列名の修飾, 14-8
ページ, 4-2

ま

マスター・グループ, 34-5
マスター・サイト, 34-5
マスター定義サイト, 34-5
マップ・メソッド, 1-56, 13-6
マテリアライズド・ビュー, 10-17
エクステントの割当て解除, 4-15
概要, 1-23
スナップショットと同義, 1-23, 34-3
パーティション化, 10-18, 11-2
マテリアライズド・ビュー・ログ, 10-18
リフレッシュ, 10-18
マテリアライズド・ビュー・ログ, 10-18
マルチスレッド・サーバー, 8-16
Net8 または SQL*Net V2 が必要, 8-14, 8-16
共有サーバー・プロセス, 8-14, 8-19
限定的運用, 8-20
サーバー・プロセス, 8-14, 8-19
使用例, 8-20
人工デッドロック, 8-19
説明, 8-3, 8-16
専用サーバーと対比, 8-16
大規模プール内のセッション・メモリー, 7-11
ディスパッチャ・プロセス, 8-14
パラレル SQL 実行, 26-8
必要なプロセス, 8-16
プライベート SQL 領域の制限, 29-17
マルチスレッドサーバー
サーバー・プロセス, 1-17
セッション情報, 7-14
ディスパッチャ・プロセス, 1-19
プライベート SQL 領域, 7-9
ソート領域, 7-16

マルチバージョン同時実行性制御, 27-6
マルチバージョンの一貫性モデル, 1-30
マルチブロック書込み, 8-8
マルチ・プロセス・システム (マルチ・ユーザー・システム), 8-2
マルチマスター・レプリケーション, 34-6
マルチメディア・データ型, 13-3
マルチユーザー環境, 1-2, 8-2

む

無名 PL/SQL ブロック, 16-14, 18-9
アプリケーション, 16-17
ストアド・プロシージャと対比, 18-9
ストアド・プロシージャのコール, 16-18
動的 SQL, 16-19
パフォーマンス, 18-9

め

明示的ロック, 27-30
メソッド
権限, 30-10
コンストラクタ・メソッド, 13-6
リテラル起動, 14-4
比較メソッド, 13-6
メッセージ snapshot too old, 27-5
メッセージ・キューイング, 19-2
キュー表, 19-4
キュー表のエクスポート, 19-11
キュー・モニター・プロセス, 1-19, 8-13, 19-6
間隔統計, 19-11
実行の時間枠, 19-7
パブリッシュ / サブスクライブのサポート, 19-10
イベントの発行, 20-18
メッセージ, 19-4
リモート・データベース, 19-9
レシピエント, 19-5
サブスクライバ・リスト, 19-5
ルールベースのサブスクリプション, 19-5, 19-6
メディア障害, 1-45, 32-5
メモリー
SQL 文のための割当て, 7-10
システム・グローバル領域も参照
カーソル (文ハンドル), 1-16
拡張バッファ・キャッシュ (32 ビット), 7-13
仮想, 7-17

共有 SQL 領域, 7-8
構造, 7-2
構造の概要, 1-13
システム・グローバル領域 (SGA)
SGA のサイズ, 7-12
開始アドレス, 7-13
初期化パラメータ, 7-12, 7-13
物理メモリーへのロック, 7-13, 7-17
割当て, 7-2
ストアド・プロシージャ, 18-8, 18-16
ソート領域, 7-16
ソフトウェア・コード領域, 7-17
内容, 7-2
プロセスでの使用, 8-2

も

モード
2 タスク, 8-3
アーカイブ・ログ, 32-17
表ロック, 27-22
モバイル・コンピューティング環境
マテリアライズド・ビュー, 10-17

ゆ

有効範囲付き REF, 13-9, 14-18
ユーザー, 29-2
PUBLIC ユーザー・グループ, 29-14, 30-18
アクセス権, 29-2
一時表領域, 1-42, 4-18, 29-13
監査, 31-11
権限, 1-40
現ユーザー, 18-10
スキーマ, 1-37, 29-2
スキーマに対応付け, 10-2
セキュリティ・ドメイン, 1-39, 29-2, 30-18
専用サーバー, 8-22
データ・ディクショナリにリスト, 2-2
同時実行動作の調整, 1-29
認証, 29-3
パスワード暗号化, 29-7
デフォルトの表領域, 29-13
表領域, 1-41
表領域割当て制限, 1-42, 29-13
分散データベース, 33-16
プロセス, 1-16, 8-4

- プロフィール, 1-42, 29-18
- マルチユーザー環境, 1-2, 8-2
- ユーザー数によるライセンス, 29-20
- ユーザー名, 1-39, 29-2
 - セッションと接続, 8-5
- ライセンス, 29-18
- リソース使用に対する制限, 1-41
- リソース制限, 29-15
- ロール, 30-15
 - ユーザーのタイプ, 30-17
- ユーザー・アクションの監視, 1-43, 31-2
- ユーザー定義オペレータ, 10-42
- ユーザー定義コスト, 22-17
- ユーザー定義データ型, 13-1, 13-3, 14-1
 - Export と Import, 14-19
 - オブジェクト型, 13-2, 13-3
 - 表の別名の使用, 14-8
 - オブジェクト・リレーショナル・モデル, 1-21
 - 記憶域, 14-17
 - 権限, 14-12
 - コレクション, 13-10
 - 可変配列 (VARRAY), 13-10
 - ネストした表, 13-11
 - 不完全な型, 14-15
- ユーザー・プログラム・インタフェース (UPI), 8-25
- ユーザー・プロセス
 - PGA の割当て, 7-14
 - 共有サーバー・プロセス, 8-19
 - 手動アーカイブ, 32-20
 - セッション, 8-5
 - 接続, 8-5
 - 専用サーバー・プロセス, 8-22
- ユーザー・ロック, 27-38

よ

要約, 10-17

読み込み

- データ・ブロック
 - 制限, 29-16
 - 内容を保証しない, 27-2
 - 反復可能, 27-6
- 読み込み一貫性, 27-2, 27-4
 - DML の副問合せ, 27-14
 - Oracle Parallel Server, 27-7
 - 仮読み込み, 27-3, 27-11
 - キャッシュ・フュージョン, 27-7

- 定義, 1-30
- 問合せ, 16-12, 27-4
- トランザクション, 1-30, 27-4, 27-6
- トリガー, 20-20, 20-22
- 内容を保証しない読み込み, 27-2, 27-11
- 反復不能読み込み, 27-3, 27-11
- 文レベル, 27-6
- マルチバージョンの一貫性モデル, 1-30, 27-4
- メッセージ snapshot too old, 27-5
- ロールバック・セグメント, 4-20
- 読み込みが書き込みを阻止するか, 27-11
- 読み込み専用推移表領域, 3-11
- 読み込み専用スナップショット, 34-8
- 読み込み専用データベース
 - オープン, 5-8
- 読み込み専用トランザクション, 1-31
- 読み込み専用表領域
 - 制限, 3-12
 - 説明, 3-11
 - バックアップ, 32-25
 - 読み込み専用推移モード, 3-11
- 読み込み専用レプリケーション
 - 使用方法, 34-11
- 予約語, 16-3

ら

ライセンス

- 現行の制限の表, 29-20
- 同時使用, 29-19
- 名前付きユーザー, 29-20

ライブラリ・キャッシュ, 7-6, 7-7, 7-10

ラッチ

- LRU, 8-8
- 説明, 27-29

ラッパー

- プロシージャ・レプリケーション, 34-16

り

リアルタイム

- データ・レプリケーション, 34-16
- レプリケーション, 34-16

リーフ・ブロック, 10-27

リーフ・レベル・スカラー属性, 14-17

リーフ・レベル属性, 14-17

リカバラ・プロセス (RECO), 1-19, 8-12

- インダウト・トランザクション, 1-35, 5-8, 17-7
- 離散トランザクションの管理
 - 要約, 17-8
- リスナー・プロセス, 6-6, 8-14
 - サービス名, 6-6
- リソース制限
 - CPU 時間の制限, 29-16
 - 値の決定, 29-18
 - 概要, 1-42
 - コール・レベル, 29-16
 - セッション当たりのアイドル時間, 29-17
 - セッション当りの接続時間, 29-17
 - セッション当りのプライベート SGA 領域, 29-17
 - セッション・レベル, 29-15
 - ユーザー当たりのセッション数, 29-17
 - 論理読み込みの制限, 29-16
- リテラル起動
 - コンストラクタ・メソッド, 14-4
- リフレッシュ
 - ジョブ・キュー・プロセス (SNPn), 1-19, 8-13
 - 増分, 10-18
 - マテリアライズド・ビュー, 10-18
- リモート依存性, 21-10
- リモート・データベース, 1-34
 - データベース・リンク, 33-5
- リモート・トランザクション, 33-12
 - パラレル DML および DDL の制限事項, 26-26
- リモート・プロシージャ・コール, 33-11
- リモート・プロシージャ・コール (RPC), 33-11
- 領域管理
 - MINIMUM EXTENT パラメータ, 26-30
 - PCTFREE, 4-6
 - PCTUSED, 4-7
 - エクステント, 4-10
 - 行連鎖, 4-9
 - セグメント, 4-16
 - ダイレクト・ロード・インサート, 25-8
 - データ・ブロック, 4-5
 - パラレル DDL, 26-30
 - ブロック内の空き領域の圧縮, 4-9
- リライト
 - セキュリティ・ポリシー内の述語, 30-21
 - マテリアライズド・ビューを使用, 10-17
- リレーショナル DBMS (RDBMS)
 - 原理, 1-21
 - SQL, 16-2
 - Oracle も参照

- オブジェクト・リレーショナル DBMS, 13-2
- リレーショナル・データベースの操作, 1-21
- リレーション, 1-22
- 履歴データベース
 - パーティション, 11-6
 - メンテナンス操作, 11-48
- リンク, 33-5

る

- ルート・ブロック, 10-53
- ルールベース最適化, 22-18
- ルールベースのサブスクリプション, 19-5, 19-6

れ

- 例外
 - ストアド・プロシージャ, 16-18
 - トリガー実行中, 20-21
 - 発生, 16-18
- レシビエント, 19-5
 - サブスクライバ・リスト, 19-5
- 列
 - NULL の使用禁止, 28-7
 - カーディナリティ, 10-31
 - 疑似列
 - ROWID, 12-14
 - ROWNUM, 23-15, 23-24, 23-47
 - USER, 30-6
 - 削除, 10-6
 - 順序, 10-7
 - 整合性制約, 10-4, 10-8, 28-4, 28-7
 - 説明, 10-3
 - 選択性, 22-9
 - ヒストグラム, 22-9, 22-11
 - 定義, 1-22
 - デフォルト値, 10-8
 - ネストした表, 10-9
 - ビューまたは表での最大数, 10-12
 - 未使用, 10-6
 - 列オブジェクト, 13-8
 - 索引, 14-6
 - 列名
 - 問合せでの修飾, 14-7, 14-8
 - 連結索引での最大数, 10-23
- 列の SET UNUSED オプション, 10-6
- 列のパーティション化, 11-15

- レプリケーション
 - アドバンスド、使用方法, 34-6
 - オブジェクト, 34-2
 - カタログ, 34-14
 - 競合
 - プロシージャ・レプリケーション, 34-16
 - グループ, 34-2
 - サイト, 34-5
 - 制限
 - ダイレクト・ロード・インサート, 25-11
 - パラレル DML, 26-41
 - 定義, 34-2
 - 分散データベース, 33-7
 - プロシージャ, 34-16
 - マテリアライズド・ビュー（スナップショット）, 10-17
 - 読み込み専用レプリケーションの使用方法, 34-11
 - リアルタイム, 34-16
- レプリケーション管理 API, 34-14
 - 管理要求, 34-14
- 連結索引, 10-22
- レンジ・パーティション化, 11-15
 - キーの比較, 11-20, 11-22
 - 主キー列, 11-42
 - 同一レベル・パーティション化, 11-24
 - パーティション・バウンド, 11-20

ろ

- ローカル管理の表領域, 3-9
 - 一時表領域, 3-13
- ローカル管理表領域, 3-9
- ローカル索引, 11-29, 11-34
 - 同一レベル・パーティション化, 11-29
 - パーティションの管理, 11-58
 - パーティションを並列に作成, 11-30
 - ビットマップ索引
 - パーティション表, 10-35
 - パラレル問合せおよび DML, 10-31
- ローカル・データベース, 1-34
- ロール, 1-40, 30-15
 - CONNECT ロール, 14-12, 14-13, 30-20
 - DBA ロール, 14-12, 30-20
 - DDL 文, 30-19
 - EXP_FULL_DATABASE ロール, 30-20
 - IMP_FULL_DATABASE ロール, 30-20
 - PL/SQL ブロック内で設定, 30-19

- RESOURCE ロール, 14-12, 14-13, 30-20
- アプリケーション, 30-16
- アプリケーションにおける, 1-41
- 依存性管理, 30-19
- オペレーティング・システムを介した管理, 30-20
- 概要, 1-40
- 起動者権限プロシージャで使用, 30-19
- 機能性, 30-2
- キュー管理者, 19-6
- グローバル認証サービス, 33-16
- 権限に関する制限, 30-19
- 使用可能または使用禁止, 30-17
- 使用方法, 30-16
- 事前定義済み, 30-20
- スキーマには含まれない, 30-18
- セキュリティ・ドメイン, 30-18
- 定義者権限プロシージャでは使用禁止, 30-18
- 取消し, 30-17
- パスワードの使用, 1-41
- 付与, 30-3, 30-17
- 付与できるユーザー, 30-18
- 分散データベース・アプリケーション, 33-16
- 命名, 30-18
- ユーザー, 30-17

- ロールバック, 4-19, 17-6
 - 回復時, 1-50, 32-9
 - セーブポイントまで, 17-6
 - 説明, 17-6
 - 定義, 1-52
 - トランザクションの終了, 17-2, 17-4, 17-6
 - 文レベル, 17-4
- ロールバック・エントリ, 4-19
- ロールバック・セグメント, 1-10, 4-19
 - 1 つ当りのトランザクション数, 4-21
 - MAXEXTENTS UNLIMITED, 26-36
 - OPTIMAL, 26-36
 - SYSTEM ロールバック・セグメント, 4-25
 - アクセス, 4-19
 - いつ取得されるか, 4-26
 - いつ使用されるか, 4-20
 - インダウト分散トランザクション, 4-24
 - エクステントの割当て, 4-21
 - 新しいエクステント, 4-24
 - エクステントの割当て解除, 4-24
 - オフライン, 4-27, 4-29
 - オフライン表領域, 4-30
 - オンライン, 4-27, 4-29

- 回復が必要な状態, 4-27
- 回復での使用, 1-48, 32-9
- 概要, 4-19, 32-8
- 起動時に取得する, 5-8
- 競合, 4-21
- 削除, 4-24
 - 制限, 4-29
- 取得時のクラッシュ, 4-26
- 循環方式による書込み, 4-21
- 状態, 4-27
- 遅延, 4-30
- 次エクステンツへの移動, 4-22
- 定義, 1-10
- トランザクション, 4-20
- トランザクションからどのように書き込まれるか, 4-21
- トランザクションのコミット, 4-21
- パブリック, 4-25
- パラレル DML, 26-36
- パラレル回復, 32-10
- 部分的に使用可能な状態, 4-27, 32-4
- プライベート, 4-25
- 無効な状態, 4-27
- 読み込み一貫性, 1-30, 4-20, 27-4
- ロック, 27-30
- ロールバック・トランザクション, 1-53, 17-2, 17-6, 32-4
- ロギング・モード
 - NOARCHIVELOG モードとの関係, 25-5
 - 影響を受ける SQL 操作, 25-7
 - ダイレクト・ロード・インサート, 25-5
 - パーティション, 11-57
 - パラレル DDL, 26-28, 26-29
- ログ・エントリ, 1-12, 32-9
 - REDO ログ・ファイルも参照, 1-12
- ログ管理ロック, 27-30
- ログ順序番号, 1-47
- ログ・スイッチ
 - ALTER SYSTEM SWITCH LOGFILE, 8-12
 - アーカイバ・プロセス, 1-18, 8-12
- REDO ログ
 - ロールフォワード, 32-8, 32-9
- ログ・ライター・プロセス (LGWR), 1-18, 8-9
 - REDO ログ・バッファ, 7-6
 - アーカイブ・モード, 32-17
 - 新しい ARCn プロセスの起動, 8-12
 - グループ・コミット, 8-10
- システム変更番号, 17-5
- 手動アーカイブ, 32-20
- 事前書込み, 8-9
- ロック, 1-31, 27-3
 - DML が取得, 27-27
 - 図, 27-25
 - DML パーティション・ロック, 11-45
 - Oracle での使用方法, 27-15
 - Oracle のロック管理サービス, 27-37
 - オブジェクト・レベルのロック, 13-14
 - 解析, 16-11, 27-29
 - 概要, 1-31, 27-3
 - 共有行排他ロック (SRX), 27-24
 - 共有表ロック (S), 27-24
 - 共有副排他ロック (SSX), 27-24
 - 行 (TX), 27-20
 - ブロック・レベルの回復, 32-14
 - 行共有表ロック (RS), 27-23
 - 行排他ロック (RX), 27-23
 - 手動, 1-32, 27-30
 - 動作の例, 27-31
 - 自動, 1-31, 27-15, 27-19
 - タイプ, 27-19
 - 段階的拡大が発生しない, 27-17
 - ディクショナリ, 27-27
 - クラスタ, 27-29
 - 存続期間, 27-29
 - ディクショナリ・キャッシュ, 27-30
 - データ, 27-20
 - 存続期間, 27-16
 - デッドロック, 27-17, 27-18
 - 回避, 27-19
 - トランザクションをコミットした後, 17-5
 - 内部, 27-29
 - 排他表ロック (X), 27-25
 - パラレル DML, 26-38
 - パラレル・キャッシュ管理 (PCM), 27-20
 - 表 (TM), 27-21
 - 表領域, 27-30
 - 表ロックのモード, 27-22
 - ファイル管理ロック, 27-30
 - 副共有表ロック SS, 27-23
 - 副排他表ロック (SX), 27-23
 - 変換, 27-17
 - ラッチ, 27-29
 - ロールバック・セグメント, 27-30
 - ログ管理ロック, 27-30

- ロック・プロセス (LCK0), 8-13
- ロック (LCK0), 1-19
- 論理 ROWID, 12-18
 - 索引構成表の索引, 10-37
 - 推測の陳腐化, 12-19
 - 推測の統計, 12-19
 - 物理推測, 10-37, 12-18
- 論理 ROWID での推測, 12-18
- 論理 ROWID での物理推測, 12-18
 - 陳腐化, 12-19
 - 統計, 12-19
- 論理データベース構造, 1-5, 1-8
 - 表領域, 3-7
- 論理ブロック, 4-2
- 論理読み込みの制限, 29-16

わ

- 割当て制限
 - SYS ユーザーには適用されない, 29-14
 - ゼロに設定, 29-14
- 表領域, 1-42, 29-13
 - 一時セグメントでは無視される, 29-13
- 表領域アクセスの取消し, 29-14