

Oracle8*i*

概要

リリース 8.1

2000 年 2 月

Oracle8i 概要, リリース 8.1

原本名: Concepts, Release 2(8.1.6)

原本著者: Lefty Leverenz, Diana Rehfield, Cathy Baird

原本協力者: Lance Ashdown, Steve Bobrowski, Cynthia Chin-Lee, Cindy Closkey, Bill Creekbaum, Jason Durbin, John Frazzini, Richard Mateosian, Denis Raphaely, John Russell, Danny Sokolsky, Randy Urbano, Richard Allen, David Anderson, Andre Bakker, Mark Bauer, Ruth Baylis, Bill Bridge, Atif Chaudry, Jeff Cohen, Michele Cyran, Benoit Dageville, Mary Ann Davidson, Sandy Dreskin, Ahmed Ezzat, Jim Finnerty, Diana Lorentz, Anurag Gupta, Gary Hallmark, Michael Hartstein, Terry Hayes, Alex Ho, Chin Hong, Ken Jacobs, Sandeep Jain, Amit Jasuja, Hakan Jakobsson, Bob Jenkins, 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, Bhagat Nainani, Denise Oertel, Bruce Olsen, Robert Pang, Mark Porter, Maria Pratt, Tuomas Pystynen, Ann Rhee, Patrick Ritto, Hasan Rizvi, Sriram Samu, Hari Sankar, Gordon Smith, Mark 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（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

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

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

危険な用途への使用について

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

Restricted Rights Notice

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

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

目次

はじめに	xxi
------------	-----

第 I 部 Oracle の概要

1 Oracle Server の概要

データベースと情報管理の概要	1-2
Oracle Server	1-4
データベースの構造と領域管理	1-5
論理データベース構造	1-5
物理データベース構造	1-8
メモリー構造とプロセス	1-11
メモリー構造	1-12
プロセスのアーキテクチャ	1-15
プログラム・インタフェース	1-19
Oracle の動作例	1-19
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	1-20
リレーショナル・モデル	1-20
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	1-21
データ・ディクショナリ	1-27
データの同時実行性と一貫性	1-28
同時実行性	1-28
読取り一貫性	1-29
ロックのメカニズム	1-30
分散処理と分散データベース	1-31

クライアント / サーバー・アーキテクチャ：分散処理	1-31
複数層アーキテクチャ：アプリケーション・サーバー	1-32
分散データベース	1-32
起動操作と停止操作	1-35
データベース・セキュリティ	1-36
セキュリティのメカニズム	1-37
権限	1-39
データベースのバックアップとリカバリ	1-42
リカバリが重要な理由	1-42
障害のタイプ	1-43
リカバリに使用される構造	1-45
基本的なリカバリ手順	1-48
Recovery Manager	1-49
データ・アクセス	1-49
SQL— 構造化問合せ言語	1-49
トランザクション	1-51
PL/SQL	1-53
データの整合性	1-55

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・ディクショナリの概要	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方法	2-3
Oracle によるデータ・ディクショナリの使用方法	2-3
ユーザーと DBA によるデータ・ディクショナリの使用方法	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・ファイルの概要	3-2
データベースへの多くの領域の割当て	3-3
表領域	3-5
SYSTEM 表領域	3-6

複数の表領域の使用方法	3-7
表領域内の領域管理	3-7
オンライン表領域とオフライン表領域	3-9
読取り専用表領域	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	3-13
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	3-16
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概要	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、行連鎖の概要	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	5-4
インスタンスとデータベースの起動	5-5
インスタンスの起動	5-5
データベースのマウント	5-6
データベースのオープン	5-8
データベースとインスタンスの停止	5-10
データベースのクローズ	5-10
データベースのディスマウント	5-11
インスタンスの停止	5-11

6 分散処理

Oracle クライアント / サーバー・アーキテクチャの概要	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	6-7
クライアント	6-8
アプリケーション・サーバー	6-8
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	7-6
共有プール	7-6
大規模プール	7-12
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)	7-14
PGA の内容	7-14
PGA のサイズ	7-16
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・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	8-5
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	8-24
事前生成済み専用プロセス	8-25
プログラム・インタフェース	8-25
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	8-26
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
リソース・コンシューマ・グループとリソース・プラン	9-3
リソース・コンシューマ・グループ	9-3
リソース・プラン	9-4
リソース割当て方法	9-5
リソース・プランへの CPU 割当て: 強調方式	9-5
リソース・プランの並列度制限: 絶対方式	9-6
リソース・プラン・ダイレクティブ	9-7

例	9-7
リソース・コンシューマ・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・プラン・ダイレクティブの使用方法	9-10
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	9-11

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	10-8
列のデフォルト値	10-9
NESTED TABLE	10-10
一時表	10-11
ビュー	10-12
ビューの記憶域	10-13
ビューの使用方法	10-14
ビューのメカニズム	10-15
依存性とビュー	10-16
更新可能な結合ビュー	10-16
オブジェクト・ビュー	10-17
インライン・ビュー	10-17
マテリアライズド・ビュー	10-18
マテリアライズド・ビューのリフレッシュ	10-19
マテリアライズド・ビュー・ログ	10-19
ディメンション	10-20
シーケンス・ジェネレータ	10-21
シノニム	10-22
索引	10-23
一意索引と非一意索引	10-24
コンポジット索引	10-24

索引とキー	10-25
索引と NULL	10-25
ファンクション索引	10-26
索引の格納方法	10-28
キー圧縮	10-31
逆キー索引	10-33
ビットマップ索引	10-34
索引構成表	10-39
索引構成表の利点	10-41
行オーバーフロー領域付きの索引構成表	10-41
索引構成表の2次索引	10-42
索引構成表のその他の機能	10-42
索引構成表に適したアプリケーション	10-43
アプリケーション・ドメイン索引	10-45
索引タイプ	10-46
ドメイン索引	10-47
ユーザー定義オペレータ	10-48
クラスタ	10-49
パフォーマンスの考慮事項	10-51
クラスタ化されたデータ・ブロックの形式	10-52
クラスタ・キー	10-52
クラスタ索引	10-53
ハッシュ・クラスタ	10-53
ハッシュ・クラスタへのデータの格納方法	10-54
ハッシュ・キー値	10-56
ハッシュ関数	10-57
ハッシュ・クラスタに対する領域の割当て	10-58
単一表ハッシュ・クラスタ	10-60

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概要	11-2
パーティション化について	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	11-11

レンジ・パーティション化	11-13
ハッシュ・パーティション化	11-15
コンポジット・パーティション化	11-16
パーティション名とサブパーティション名	11-17
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	11-18
レンジ・パーティション化のパーティション・バウンド	11-19
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	11-37
索引構成表と2次索引のパーティション化	11-40
DML パーティション・ロックとサブパーティション・ロック	11-44
DML パーティション・ロック	11-44
DML サブパーティション・ロック	11-45
Oracle Parallel Server のパフォーマンスの考慮事項	11-46
メンテナンス操作	11-46
パーティションのメンテナンス操作	11-47
索引の管理	11-57
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 組込みデータ型

Oracle データ型の概要	12-2
文字データ型	12-5
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	12-7
LONG データ型	12-7

NUMBER データ型	12-7
内部数値形式	12-9
DATE データ型	12-9
ユリウス暦の使用方式	12-10
日付算術	12-11
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	12-13
BFILE データ型	12-13
RAW および LONG RAW データ型	12-14
ROWID および UROWID データ型	12-15
ROWID 疑似列	12-15
物理 ROWID	12-15
論理 ROWID	12-19
Oracle 以外のデータベースの ROWID	12-21
ANSI データ型、DB2 データ型および SQL/DS データ型	12-21
データ変換	12-22

13 ユーザー定義データ型

ユーザー定義データ型の概要	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	13-13
PL/SQL	13-13
Pro*C/C++	13-13
OCI	13-14
OTT	13-15
JPublisher	13-15
JDBC	13-15
SQLJ	13-15

14 オブジェクト・ビュー

オブジェクト・ビューの概要	14-2
オブジェクト・ビューの利点	14-2
オブジェクト・ビューの定義	14-3
オブジェクト・ビューの使用方法	14-4
オブジェクト・ビューの更新	14-5
ビュー内の NESTED TABLE の列の更新	14-5

第 V 部 データ・アクセス

15 SQL と PL/SQL

構造化問合せ言語の概要	15-2
SQL 文	15-3
非標準 SQL の識別	15-6
再帰 SQL	15-7
カーソル	15-7
共有 SQL	15-7
解析	15-8
SQL の処理	15-9
SQL 文の実行の概要	15-9
DML 文の処理	15-11
DDL 文の処理	15-15
トランザクションの制御	15-15
PL/SQL	15-16
PL/SQL が実行される方法	15-16
PL/SQL の言語構文	15-18
ストアド・プロシージャ	15-19
外部プロシージャ	15-21
PL/SQL Server Pages	15-21

16 トランザクションの管理

トランザクションの概要	16-2
文の実行とトランザクションの制御	16-3
文レベルのロールバック	16-4

Oracle とトランザクションの管理	16-5
トランザクションのコミット	16-5
トランザクションのロールバック	16-6
セーブポイント	16-7
2 フェーズ・コミット・メカニズム	16-8
ディスクリット・トランザクションの管理	16-9
自律型トランザクション	16-10
自律型 PL/SQL ブロック	16-10
自律型ブロック内のトランザクション制御文	16-11

17 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概要	17-2
ストアド・プロシージャとファンクション	17-2
パッケージ	17-4
プロシージャとファンクション	17-6
定義者権限と実行者権限	17-7
プロシージャの利点	17-8
プロシージャのガイドライン	17-10
無名 PL/SQL ブロックとストアド・プロシージャ	17-11
スタンドアロン・プロシージャ	17-11
ストアド・プロシージャの依存性の追跡	17-11
外部プロシージャ	17-12
パッケージ	17-12
パッケージの利点	17-16
パッケージの依存性の追跡	17-17
オラクル社が提供するパッケージ	17-17
Oracle がプロシージャとパッケージを格納する方法	17-17
プロシージャとパッケージのコンパイル	17-17
コンパイル済コードのメモリーへの格納	17-18
プロシージャとパッケージのデータベースへの格納	17-18
Oracle がプロシージャとパッケージを実行する方法	17-19
ユーザー・アクセスの検証	17-19
プロシージャの妥当性の検証	17-19
プロシージャの実行	17-20

18 アドバンスト・キューイング

メッセージ・キューイングの概要	18-2
Oracle Advanced Queuing	18-3
キューイング・モデル	18-4
キューイング・エンティティ	18-5
アドバンスト・キューイングの機能	18-9

19 トリガー

トリガーの概要	19-2
トリガーの使用方法	19-3
トリガーの各部分	19-6
トリガー・イベントまたはトリガーを実行する文	19-7
トリガー条件	19-8
トリガー・アクション	19-8
トリガーのタイプ	19-9
行トリガーと文トリガー	19-9
BEFORE トリガーと AFTER トリガー	19-10
INSTEAD OF トリガー	19-13
システム・イベントとユーザー・イベントのトリガー	19-19
トリガーの実行	19-22
トリガーの実行モデルと整合性制約のチェック	19-23
トリガーのデータ・アクセス	19-24
PL/SQL トリガーの記憶域	19-26
トリガーの実行	19-26
トリガーの依存性のメンテナンス	19-26

20 Oracle の依存性の管理

依存性の問題の概要	20-2
スキーマ・オブジェクトの依存性の解決	20-4
ビューと PL/SQL プログラム・ユニットのコンパイル	20-5
ファンクション索引の依存性	20-7
依存性の管理と存在しないスキーマ・オブジェクト	20-8
共有 SQL の依存性管理	20-10
ローカルおよびリモートの依存性の管理	20-10
ローカル依存性の管理	20-10

リモート依存性の管理	20-11
------------------	-------

第 VI 部 SQL 文の最適化

21 オプティマイザ

最適化の概要	21-2
実行計画	21-2
実行の順序	21-6
オプティマイザのプラン・スタビリティ	21-7
コストベース最適化	21-8
コストベース・アプローチの目標	21-8
コストベース最適化の統計	21-9
コストベースのアプローチを使用する場合	21-17
拡張可能な最適化	21-18
ユーザー定義統計	21-18
ユーザー定義選択性	21-19
ユーザー定義コスト	21-19
ルールベース最適化	21-20

第 VII 部 パラレル SQL とダイレクト・ロード・インサート

22 ダイレクト・ロード・インサート

ダイレクト・ロード・インサートの概要	22-2
ダイレクト・ロード・インサートの利点	22-2
INSERT ... SELECT 文	22-3
ダイレクト・ロード・インサート文の種類	22-3
シリアル INSERT とパラレル INSERT	22-3
ロギング・モード	22-5
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	22-8
索引のメンテナンス	22-8
領域についての考慮事項	22-9
ロックについての考慮事項	22-10
ダイレクト・ロード・インサートの制限事項	22-10

23 SQL 文の平行実行

SQL 文の平行実行の概要	23-2
平行化できる操作	23-2
Oracle が操作を平行化する方法	23-3
平行実行のプロセス・アーキテクチャ	23-5
平行実行サーバー・プール	23-7
平行実行サーバーの通信方法	23-9
SQL 文の平行化	23-10
並列度の設定	23-15
Oracle による操作の並列度の決定方法	23-16
作業負荷の均衡化	23-19
SQL 文の平行化ルール	23-20
平行問合せ	23-28
索引構成表の平行問合せ	23-29
オブジェクト型の平行問合せ	23-30
平行 DDL	23-31
平行化できる DDL 文	23-31
平行の CREATE TABLE ... AS SELECT	23-32
リカバリ可能性と平行 DDL	23-33
平行 DDL の領域管理	23-34
平行 DML	23-36
手動による平行化と比較したときの平行 DML の利点	23-37
平行 DML を使用する場合	23-38
平行 DML の使用可能化	23-39
平行 DML のためのトランザクション・モデル	23-40
平行 DML のリカバリ	23-41
平行 DML の領域に関する考慮事項	23-42
平行 DML のためのリソースのロックとエンキュー	23-43
平行 DML の制限事項	23-45
関数の平行実行	23-47
親和性	23-49
親和性と平行問合せ	23-49
親和性と平行 DML	23-50
その他の平行化	23-51

第 VIII 部 データの保護

24 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の概要	24-2
回避可能な現象とトランザクション分離レベル	24-2
ロックのメカニズム	24-3
データの同時実行性と一貫性の管理方法	24-4
マルチバージョン一貫性制御	24-4
文レベルの読取り一貫性	24-6
トランザクション・レベルの読取り一貫性	24-6
Oracle Parallel Server における読取り一貫性	24-7
Oracle の分離レベル	24-7
コミット読込み分離とシリアル化が可能な分離の比較	24-10
分離レベルの選択	24-13
データをロックする方法	24-15
トランザクションとデータ同時実行性	24-16
デッドロック	24-17
ロックの種類	24-19
DML ロック	24-20
DDL ロック	24-29
ラッチと内部ロック	24-30
明示的（手動）データ・ロック	24-32
Oracle のロック・マネージメント・サービス	24-40

25 データの整合性

データ整合性の概要	25-2
データ整合性のタイプ	25-3
Oracle がデータの整合性を規定する方法	25-4
整合性制約の概要	25-5
整合性制約の利点	25-5
整合性制約のパフォーマンス・コスト	25-7
整合性制約のタイプ	25-7
NOT NULL 整合性制約	25-7
UNIQUE キー整合性制約	25-8

PRIMARY KEY 整合性制約	25-11
参照整合性制約	25-13
CHECK 整合性制約	25-21
制約チェックのメカニズム	25-21
デフォルト列値と整合性制約チェック	25-24
遅延制約チェック	25-24
制約の属性	25-24
SET CONSTRAINTS モード	25-25
一意制約と一意索引	25-25
制約の状態	25-26
制約の状態変更	25-27

26 データベース・アクセスの制御

データベース・セキュリティの概要	26-2
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	26-2
ユーザーの認証	26-3
オペレーティング・システムによる認証	26-4
ネットワークによる認証	26-4
Oracle データベースによる認証	26-7
複数層の認証と許可	26-9
Secure Socket Layer プロトコルによる認証	26-12
データベース管理者の認証	26-13
ユーザー表領域の設定と割当て制限	26-14
デフォルト表領域	26-14
一時表領域	26-15
表領域のアクセスと割当て制限	26-15
ユーザー・グループ PUBLIC	26-16
ユーザー・リソースの制限とプロファイル	26-17
システム・リソースのタイプと制限	26-17
プロファイル	26-20
ライセンス	26-20
同時使用ライセンス	26-21
名前付きユーザー・ライセンス	26-22

27 権限、ロールおよびセキュリティ・ルール

権限の概要	27-2
システム権限	27-2
スキーマ・オブジェクト権限	27-3
表のセキュリティに関するトピック	27-5
ビューのセキュリティに関するトピック	27-6
プロシージャのセキュリティに関するトピック	27-7
型のセキュリティに関するトピック	27-12
ロール	27-17
ロールの一般的な使用方法	27-18
ロールのメカニズム	27-19
ロールの付与と取消し	27-19
ロールの付与と取消しを実行できるユーザー	27-20
ロールの命名	27-20
ロールとユーザーのセキュリティ・ドメイン	27-20
PL/SQL ブロックとロール	27-20
データ定義言語の文とロール	27-21
事前定義済のロール	27-22
オペレーティング・システムとロール	27-23
分散環境におけるロール	27-23
ファイングレイン・アクセス・コントロール	27-23
動的な述語	27-24
セキュリティ・ルールの例	27-24
アプリケーション・コンテキスト	27-25

28 監査

監査の概要	28-2
監査機能	28-2
監査のメカニズム	28-4
文監査	28-7
権限監査	28-7
スキーマ・オブジェクト監査	28-8
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	28-8
文、権限およびスキーマ・オブジェクトの監査対象の限定	28-9
文の正常な実行と失敗した実行の監査	28-9

BY SESSION 監査と BY ACCESS 監査	28-10
ユーザー別の監査	28-12

29 データベースのリカバリ

データベース・リカバリの概要	29-2
エラーと障害	29-2
データベースのリカバリで使用する構造	29-6
データベースのバックアップ	29-7
REDO ログ	29-7
ロールバック・セグメント	29-8
制御ファイル	29-8
ロールフォワードとロールバック	29-8
REDO ログとロールフォワード	29-9
ロールバック・セグメントとロールバック	29-9
リカバリ・パフォーマンスの改善	29-10
リカバリのパラレル実行	29-11
ファースト・スタート・リカバリ	29-14
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	29-15
Recovery Manager	29-16
リカバリ・カタログ	29-16
パラレル化	29-17
レポートの生成	29-18
データベースのアーカイブ・モード	29-18
NOARCHIVELOG モード（メディア・リカバリが使用禁止）	29-18
ARCHIVELOG モード（メディア・リカバリが使用可能）	29-19
制御ファイル	29-22
制御ファイルの内容	29-22
多重制御ファイル	29-23
データベースのバックアップ	29-24
データベース全体のバックアップ	29-24
部分データベース・バックアップ	29-26
Export および Import ユーティリティ	29-27
読取り専用表領域とバックアップ	29-27
耐障害性	29-27
障害リカバリの計画	29-28

管理されたスタンバイ・データベース	29-28
-------------------------	-------

第 IX 部 分散データベースとレプリケーション

30 分散データベースの概念

分散データベース・アーキテクチャの概要	30-2
同種分散データベース・システム	30-2
異種分散データベース・システム	30-5
クライアント / サーバー・データベース・アーキテクチャ	30-6
データベース・リンク	30-9
データベース・リンク	30-10
データベース・リンクを使用する理由	30-13
データベース・リンク内のグローバル・データベース名	30-13
データベース・リンク名	30-15
データベース・リンクのタイプ	30-16
データベース・リンクのユーザー	30-17
データベース・リンクの作成：例	30-20
スキーマ・オブジェクトとデータベース・リンク	30-21
データベース・リンクの制限事項	30-23
分散データベースの管理	30-23
サイト自律性	30-24
分散データベースのセキュリティ	30-24
データベース・リンクの監査	30-31
管理ツール	30-31
分散システムでのトランザクション処理	30-33
リモート SQL 文	30-34
分散 SQL 文	30-34
リモート文と分散文のための共有 SQL	30-35
リモート・トランザクション	30-35
分散トランザクション	30-35
2 フェーズ・コミット・メカニズム	30-36
データベース・リンクの解決	30-36
スキーマ・オブジェクトの名前解決	30-39
ビュー、シノニムおよびプロシージャでのグローバル・ネーム解決	30-42
分散データベース・アプリケーションの開発	30-44

分散データベース・システムにおける透過性	30-44
リモート・プロシージャ・コール (RPC)	30-46
分散問合せの最適化	30-47
各国語サポート	30-47
クライアント / サーバー環境	30-48
同種分散環境	30-48
異種分散環境	30-49

31 レプリケーション

レプリケーションの概要	31-2
レプリケーションを使用するアプリケーション	31-3
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	31-4
レプリケーション環境のタイプ	31-5
マルチマスター・レプリケーション	31-5
スナップショット・レプリケーション	31-7
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	31-11
レプリケーション環境の管理ツール	31-12
Oracle Replication Manager	31-13
レプリケーション・マネージメント API	31-14
レプリケーション・カタログ	31-14
分散スキーマ管理	31-14
レプリケーションの競合	31-15
マルチマスター・レプリケーションに関するその他のオプション	31-16
同期レプリケーション	31-16
プロシージャ型レプリケーション	31-16

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（標準エディションとも呼びます）および Oracle8i Enterprise Edition 製品の機能の詳細は、『Oracle8i 概要』を参照してください。Oracle8i および Oracle8i Enterprise Edition は、同じ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、アプリケーション・フェイルオーバー機能を使用するには、Enterprise Edition と Parallel Server オプションが必要です。

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、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 の概要」](#)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。

インストレーションや移行について

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、主としてインストレーションに関心のある方は、使用するオペレーティング・システムを対象にした Oracle 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、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』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](#)のみでなく、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。

パフォーマンスの最適化について

このマニュアルには、Oracle の効率的な運用、メンテナンスおよびパフォーマンスに役立つ情報が記載されています。詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

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 の概要」

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、この章をお読みください。

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取り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、エクステンツおよびセグメント」

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・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、名前付き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「組み込みデータ型」

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、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「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 15 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 16 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 17 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、パッケージと呼ばれるプロシージャ型言語の構文について説明します。

第 18 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 19 章「トリガー」

この章では、データベース・トリガーについて説明します。データベース・トリガーとは、PL/SQL、Java または C で記述され、データベースに格納されているプロシージャであり、表またはビューが変更された場合、またはなんらかのユーザー・アクションやデータベース・システム・アクションが発生した場合に、暗黙的に実行（起動）されます。

第 20 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 21 章「オプティマイザ」

オプティマイザ、つまり各 SQL 文を最も効率的に実行する方法を選択する Oracle の機能について説明します。

第 VII 部：パラレル SQL とダイレクト・ロード・インサート

第 22 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 句について説明します。

第 23 章「SQL 文のパラレル実行」

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 24 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するための、Oracle が自動的に実行するメカニズムについて説明します。

第 25 章「データの整合性」

データの整合性と、整合性を規定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 26 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 27 章「権限、ロールおよびセキュリティ・ルール」

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 28 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 29 章「データベースのリカバリ」

データベースのリカバリ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 30 章「分散データベースの概念」

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 31 章「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方法

すべての読者は、必ず第 1 章「Oracle Server の概要」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 26 章「データベース・アクセスの制御」、第 27 章「権限、ロールおよびセキュリティ・ルール」および第 28 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。

大文字

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 および Oracle Enterprise Manager の行モード (Server Manager)、SQL*Plus のコマンドや文は、固定幅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、カ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。例の中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icolon (;) 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icolonまたはその他の終了記号が必要な場合と、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、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「[Oracle Server の概要](#)」

Oracle Server の概要

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

- データベースと情報管理の概要
- 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の概要

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、障害のリカバリ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>
データベースに規定される整合性	<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」が規定されます。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、様々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種オプ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。

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）のコードを実行します。
- 「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・プロセスがあります。

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して、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。

関連項目： Oracle Parallel Server の詳細は、『Oracle8i Parallel Server 概要』を参照してください。

データベースの構造と領域管理

「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。物理構造と論理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域を管理できます。

論理データベース構造

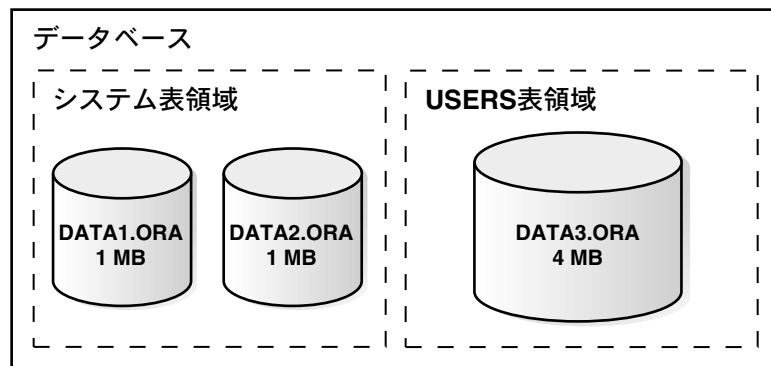
Oracle データベースの論理構造には、表領域、スキーマ・オブジェクト、データ・ブロック、エクステンツおよびセグメントが含まれます。

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 1-1 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 1-1 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ます。
- 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ます。
- 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ります (SYSTEM 表領域は 2MB、USERS 表領域は 4MB の記憶容量を持ちます)。
- 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ります (6MB)。

オンライン表領域とオフライン表領域 表領域は、「オンライン化」(アクセス可能) または「オフライン化」(アクセス不能) に設定できます。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化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。

注意： 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。

関連項目： スキーマ・オブジェクトの詳細は、1-21 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。

関連項目： [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・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
- データベース・リカバリ中
- 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いっぱいになった時点で、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いっぱい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。

関連項目：

- 1-29 ページの「[読取り一貫性](#)」
- 1-42 ページの「[データベースのバックアップとリカバリ](#)」

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ます。
- 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ます。
- 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ます。

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 DBWn バックグラウンド・プロセスによって決定されます。

関連項目： 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、1-11 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 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、データベースをリカバリ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停止した場合、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リカバリされます。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リストアします。

リカバリ操作中に 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。

関連項目： REDO ログ・ファイルの詳細は、1-42 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・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方法 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・ファイルや REDO ログ・ファイルの新規作成など）、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。

制御ファイルは、データベースのリカバリが必要な場合にも使用されます。

関連項目：

- 1-42 ページの「データベースのバックアップとリカバリ」
- 第 29 章「データベースのリカバリ」

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。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 文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) 文を使用します。DDL 文には、データベースや表を作成および変更する文が含まれます。

また、表の中のデータの更新、削除または取出しには、データ操作言語 (DML) を使用します。DML 文には、データの変更やフェッチを行う文が含まれます。最も使用頻度の高い SQL 文は、SELECT 文です。この文は、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 文の他に、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・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、表などのオブジェクトとその関連オブジェクトが抽出されてから、Export のダンプ・ファイルに書き込まれます。

Import Import ユーティリティは、ある Oracle データベースから Export ユーティリティによって抽出されたデータ・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・インスタンスエーションなど、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。

関連項目： 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 は様々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。

関連項目： 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。

メモリー構造とプロセス

この項では、データベースを管理するために Oracle Server が使用するメモリー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、メモリー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。すべてのメモリー構造は、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・メモリー内に存在します。

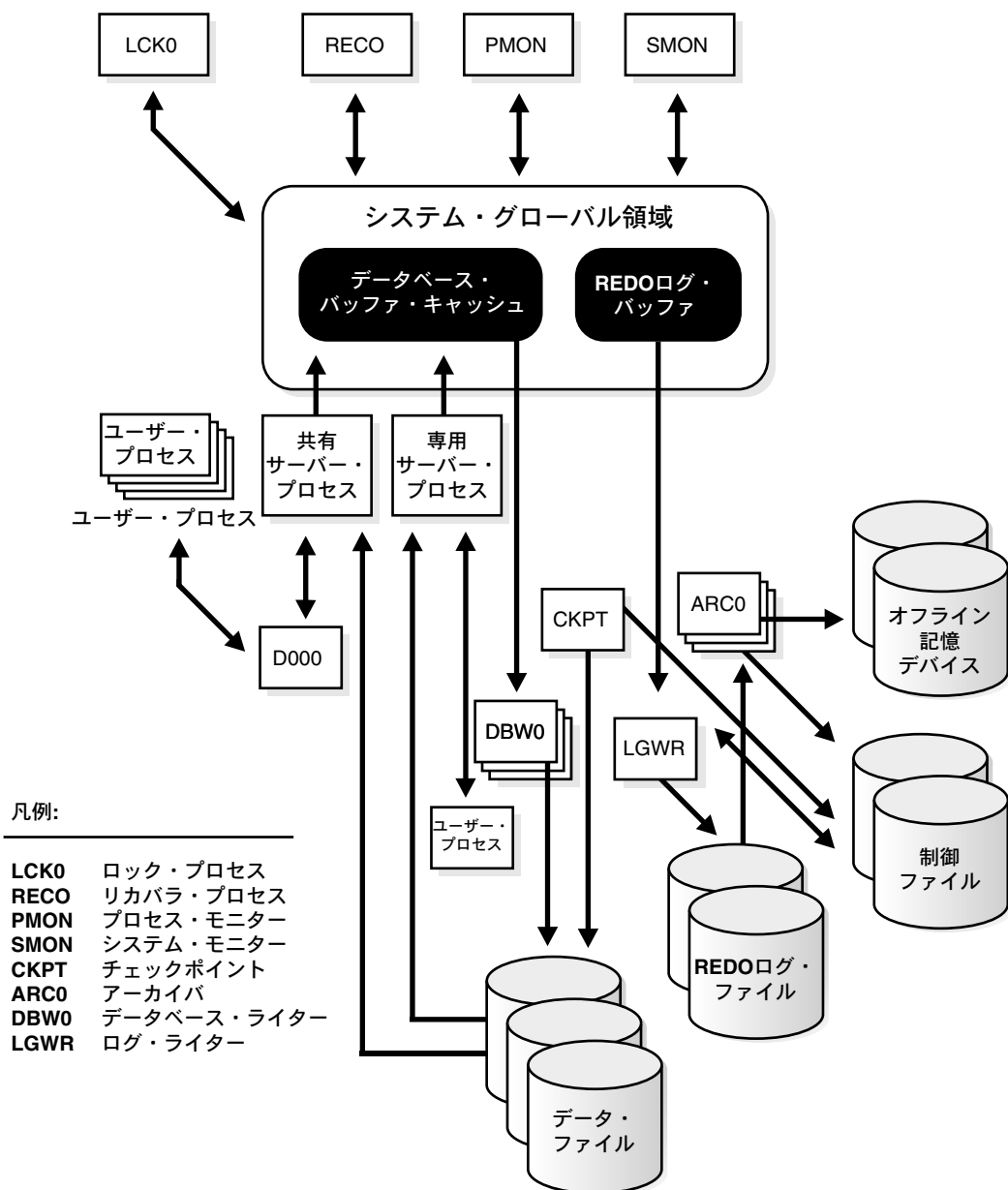
「プロセス」とは、これらのコンピュータのメモリー内で作業を行うジョブまたはタスクです。

1-13 ページの図 1-2 に、Oracle Server のメモリーとプロセスの構造の代表的な例を示します。

メモリー構造

Oracle はメモリー構造を作成して使用し、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、メモリーが使用されます。2つの基本的なメモリー構造、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、それぞれのメモリー領域について詳しく説明します。

図 1-2 Oracle のメモリー構造とプロセス



システム・グローバル領域

「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ー領域です。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。

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り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。

関連項目：

- 1-4 ページの「[Oracle インスタンス](#)」
- SGA および Oracle バックグラウンド・プロセスの詳細は、1-16 ページの「[バックグラウンド・プロセス](#)」を参照してください。

データベース・バッファ・キャッシュ 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り、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、データベースのリカバリ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。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。

関連項目： 共有 SQL 領域の詳細は、1-50 ページの「[SQL 文](#)」を参照してください。

大規模プール Oracle のバックアップおよびリストア操作、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 OracleXA 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）。ほとんどの 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、アプリケーションの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ある必要があります。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り、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。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、DBW n 、LGWR、CKPT、SMON、PMON、ARC n 、RECO、Dnnn、LCK0、SNP n および QMN n です。

関連項目：

- 1-4 ページの「[Oracle インスタンス](#)」
- SGA の詳細は、1-14 ページの「[システム・グローバル領域](#)」を参照してください。

データベース・ライター (DBW n) 「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。DBW n 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ため、DBW_n 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。そのかわりに、DBW_n 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、DBW_n 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。LRU のデータから先に、データ・ファイルに書き込まれます。DBW_n 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。

関連項目： コミットの詳細は、1-51 ページの「**トランザクション**」を参照してください。

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、DBW_n 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 DBW_n 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・リカバリを実行します。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・リカバリを実行できます。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・リカバリ時にファイル読み込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リカバリします。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 によってリカバリされます。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」がプロセスのリカバリを実行します。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 (ARC_n) 「アーカイバ」は、オンライン REDO ログ・ファイルがいっぱいになるか、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 ARC_n プロセス (ARC0) で十分ですが、動的初期化パラメータ LOG_ARCHIVE_MAX_PROCESSES を使用すると最高 10 個の ARC_n プロセスを指定できます。作業負荷が現行の ARC_n 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。

関連項目： アーカイバの詳細は、1-45 ページの「[REDO ログ](#)」を参照してください。

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」(LCK0) プロセスは、Oracle Parallel Server でインスタンス間ロックに使用されます。

関連項目： ロック・プロセスの構成の詳細は、1-4 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。

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 のインタフェースです。

関連項目： 1-34 ページの「[Oracle と Net8](#)」

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トラップと返送を行います。
- 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。

関連項目： Oracle 構成の詳細は、[第 8 章「プロセス・アーキテクチャ」](#)を参照してください。

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	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	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	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。

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3種類の要素から構成されます。

- 名前 — オブジェクト型を一意に識別するものです。
- 属性 — 組み込みデータ型、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 — 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、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実装します。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。）

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「スケール」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。

表のデータに対して定義されたビジネス・ルールを規定するために、整合性制約とトリガーを表に対して定義できます。

関連項目：

- 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。
- ビジネス・ルールの詳細は、1-55 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、「ストアド・クエリ」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「ベース表」と呼びます。ベース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、すべてベース表にも影響します。

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して、表のセキュリティ・レベルを強化します。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します。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も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します。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。
- ベース表とは異なる視点からデータを提示します。たとえば、ベース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します。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くと、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。

関連項目： 結合ビューの更新方法の詳細は、10-16 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して、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つ以上のベース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、ベース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。集計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。SQL 文では、キーワード SNAPSHOT および MATERIALIZED VIEW は同義です。

関連項目： リモート・データベース内のデータをレプリケートする方法
の詳細は、1-34 ページの「[表複製](#)」を参照してください。

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ため、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ため、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、様々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。

関連項目： SQL と PL/SQL のプロシージャ、ファンクションおよびパッケージの詳細は、1-49 ページの「[データ・アクセス](#)」を参照してください。

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、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します。
- 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します。
- 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します。
- データベース・ユーザー側の SQL 文を単純化します。

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、当人のみが見ることができる「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われるようになります。

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）使います。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、使われます。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。

索引は、パーティションに分割できます。

関連項目： 索引の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。

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。

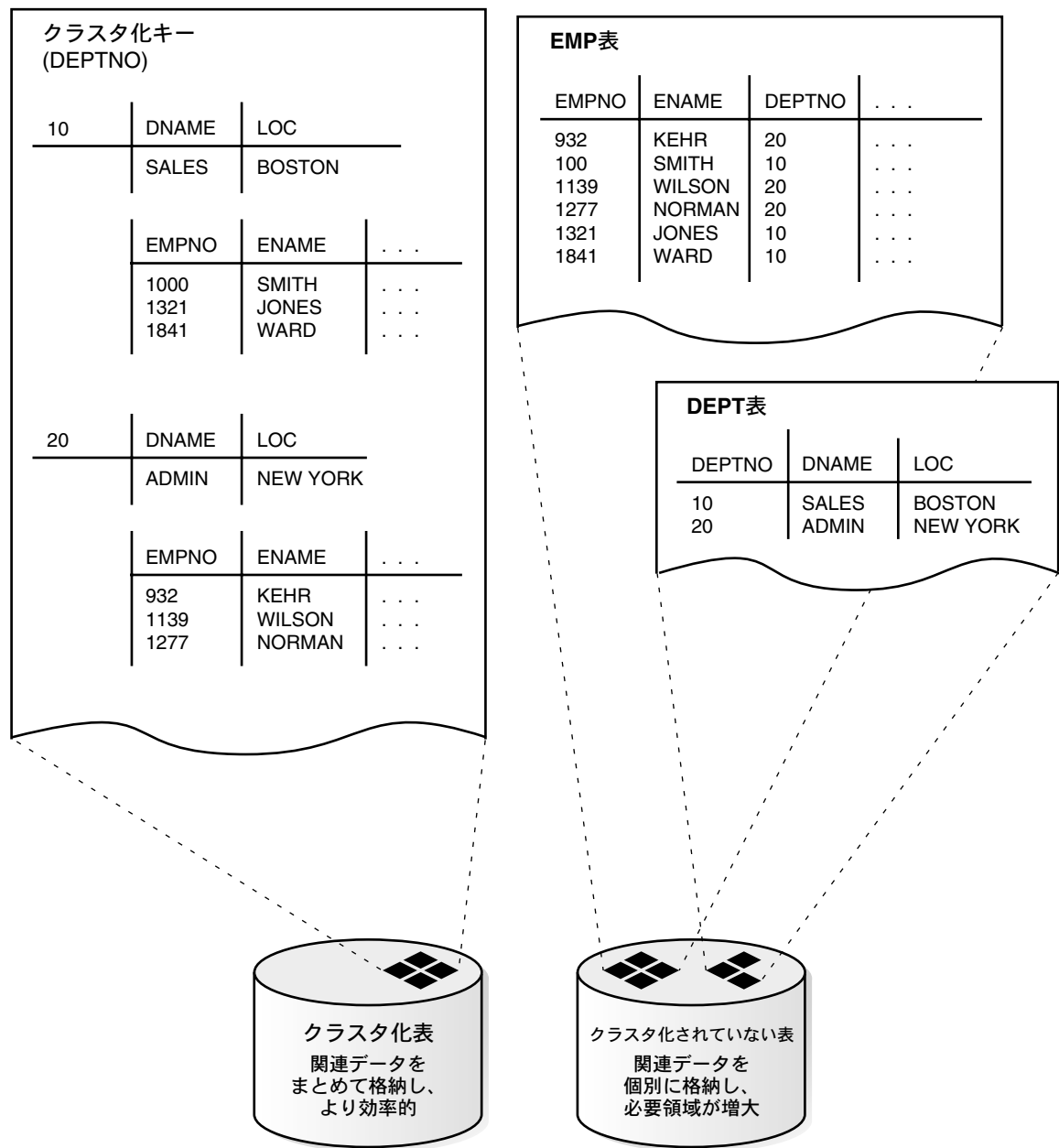
クラスタ内の表にあって相互に関連する列のことを「クラスタ・キー」と呼びます。クラスタの行が最小限の I/O で検索できるように、クラスタ・キーには索引が設定されています。索引クラスタ（非ハッシュ・クラスタ）のクラスタ・キーに含まれるデータは、複数の表について 1 回だけ格納すればよいいため、表が個別に格納されている（クラスタ化されていない）場合と比べると、クラスタは一連の表をより効率的に格納できます。

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 操作で検索されるため、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 1-3 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。

図 1-3 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 列または列の集合を識別します。
- 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、同じ表のもの（「非正規化」）でも複数の表のもの（「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 句を使用して表を内部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。

関連項目：

- 1-32 ページの「[分散データベース](#)」
- [第 30 章「分散データベースの概念」](#)

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「読取り専用」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に、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ためです。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。

関連項目： 第2章「データ・ディクショナリ」

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、修正します。
- 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ます。
- 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します。

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 DML 文を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、様々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。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。

関連項目： 同時実行性および一貫性機能の詳細は、1-53 ページの「[トランザクションを使用したデータの一貫性](#)」を参照してください。

読取り一貫性

Oracle がサポートする読取り一貫性は、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ません（文レベルの読取り一貫性）。
- 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ません。
- 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ません。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します。

Oracle の読取り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）。

読取り一貫性、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（読込み）や同時更新（書込み）の実行時に Oracle は読取り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、変更されたデータに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「読取り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」開始された文のみが、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、読取り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（またはコミットされていない）の 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。

- 読込みユーザーのために生成される、読取り一貫性を備えたビューの開始点を示します。
- データベースの他のトランザクションが、変更データを読込みまたは更新のために参照できる時期を制御します。

読取り専用トランザクション

デフォルトでは、文レベルの読取り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・レベルの読取り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取り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、「読取り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取り専用であると指定した後は、それぞれの問合せの結果が同一時点について読取り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」を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えるまで、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します。
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します。

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で、同一データへの競合が最小限に抑えられます。

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・ロックをオーバーライドできます。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る）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」を使用することで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つの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り、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の検証
- Oracle データベース・サーバーに接続
- クライアントにかわって要求された操作を実行

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する必要があります。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る必要があります。

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を実現されます。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、位置の透過性を実現されます。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・データにアクセスするには、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リカバリ 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リソースを解放できます。

分散リカバリのための機能 特定のデータベースを過去の特定の時点までリカバリする必要がある場合、Oracle のリカバリ機能を使用すると、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。これにより、グローバル・データベースは一貫した状態に保たれます。

表複製

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 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。

関連項目：『Oracle8i Net8 管理者ガイド』

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。

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。

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。

関連項目：

- データベースの起動方法の詳細は、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。
- 初期化パラメータ・ファイルの例については、5-4 ページの「[パラメータ・ファイル](#)」を参照してください。
- 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その接続先データベースの停止方法の詳細は、5-10 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・アクセスを防止します。
- 権限のないスキーマ・オブジェクトへのアクセスを防止します。
- ディスクの使用を制御します。
- システム・リソース（CPU タイムなど）の使用を制御します。
- ユーザーのアクションを監査します。

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。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 文を発行できますが、DELETE 文は発行できません）。
- 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

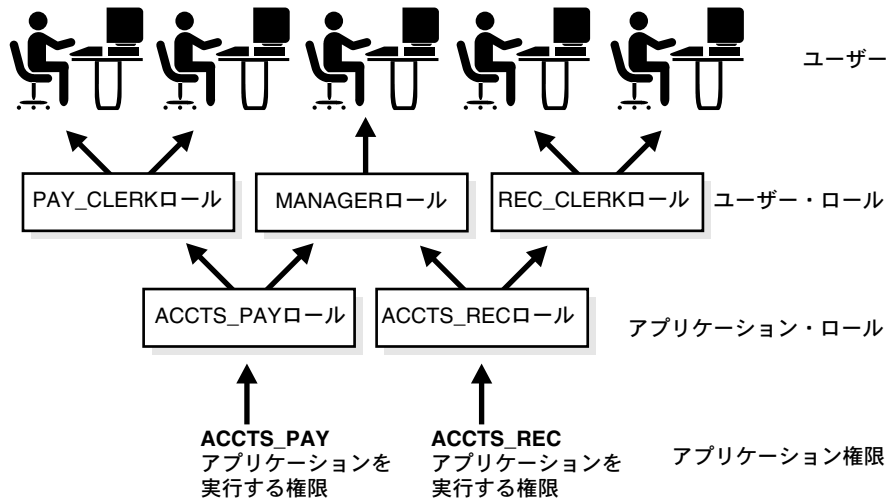
Oracle Server では、「任意アクセス制御」が可能です。任意アクセス制御とは、権限に基づいて情報へのアクセスを制限する方法です。ユーザーがスキーマ・オブジェクトにアクセスするには、そのユーザーは適切な権限を割り当てられている必要があります。適切な権限が付与されているユーザーは、他のユーザーに自分で任意に権限を付与できます。このため、このタイプのセキュリティを「任意」セキュリティと呼びます。

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
- 記憶域の設定と割当て制限
- リソース制限
- 監査

図 1-4 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 1-4 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、次のことをする権利が権限です。

- 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変更できます。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ます。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）に対して付与できます。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ます。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で簡単かつ制御された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。そのロールをグループの各メンバーに付与できます。
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・ドメインには、そのロールに対して加えられる変更が自動的に反映されます。
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量
 - SQL 文による Oracle への 1 回のコールで使用可能な量
- 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が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」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	<p>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。)</p> <p>文監査では、システム的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。</p>
権限監査	<p>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。</p>
スキーマ・オブジェクト監査	<p>スキーマ・オブジェクト監査は、ユーザーとは無関係の、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。</p>

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。

- 様々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リカバリ
- どのような状況にも対応する柔軟なリカバリ操作
- 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）

リカバリが重要な理由

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、データベースをリカ

バリ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、リカバリ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リカバリ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・エラーからのリカバリを可能にし、その他の固有のリカバリ要件を満たすために、Oracle は厳密な時間管理によるリカバリ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リカバリできます。

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し、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、必要なリカバリ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・リカバリ」または「インスタンス・リカバリ」が必要になります。クラッシュ・リカバリは、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 環境では、別のインスタンスの SMON プロセスによって、インスタンス・リカバリが実行されます。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データをリカバリするために、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、様々なファイルが、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・リカバリ」が必要です。メディア・リカバリは、障害のために失われたメモリー内のコミット済データなど、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・ファイルをリストアします。ディスク障害からのリカバリを完了するには、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 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、完全に迅速なリカバリを行うことができます。データベースを完全にリカ

バリするか、または部分的に特定の時点までリカバリできるように、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リカバリする間、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、破損した部分のリカバリ中も日常業務に使用できます。

リカバリに使用される構造

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リカバリします。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

「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 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、最初のファイルがいっぱいになり、2番目のファイルもいっぱいになれば最初のファイルを再使用し、そのファイルがいっぱいになれば2番目のファイルを再使用するというようになります。ファイルがいっぱいになるたびに、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も、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・ファイルを容易にリカバリできます。

アーカイブ REDO ログ いっぱいになったオンライン REDO ログ・ファイルは、「アーカイブ 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 REDO ログは、「アーカイブ (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 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 いっぱい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 いっぱい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リカバリできます。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 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リカバリすることはできませんが、ディスク障害からはリカバリできません。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 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・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します。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します。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・レベルで論理リカバリできます。
- チューニングと容量計画のための補足情報を提供します。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します。

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。

関連項目：

- LogMiner の詳細は、『Oracle8i 管理者ガイド』を参照してください。
- REDO ログ・ファイルの詳細は、1-9 ページの「[REDO ログ・ファイル](#)」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カレント・ログ順序番号についての情報が保管されます。通常のリカバリ手順では、リカバリ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。

多重制御ファイル 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

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するロールバック情報が記録されます。データベースのリカバリでは、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、この重要なリカバリ情報は REDO ログによって自動的に保護されます。

関連項目： ロールバック・セグメントの詳細は、1-6 ページの「[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。

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・リカバリでは、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リストア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。

データベース全体のバックアップ データベース全体のバックアップは、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、様々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、様々な場合にデータ・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 REDO ログは、いっぱい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。

基本的なリカバリ手順

DBW_n 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もあります。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 ログのみに存在します。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 ログがリカバリ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・ファイルから消去する必要があります。

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リカバリでは、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

リカバリ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、リカバリ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」と呼びます。Oracle はこのステップを自動的に完了します。

Recovery Manager

Recovery Manager (RMAN) 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 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リストアやりカバリを行います。

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・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、リストア操作とメディア・リカバリ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 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文の列

関連項目： 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の遵守
- データの操作中にデータベース情報の一貫性の制御、維持
- データベース情報の整合性を維持する規則を定義、規定するためのシステムの提供
- 高パフォーマンスの実現

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 向けに実装した 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」と等価である必要があります。

```
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、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

データ定義言語 (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。

関連項目： 権限の詳細は、1-36 ページの「[データベース・セキュリティ](#)」を参照してください。

データ操作言語 (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。この文に含まれるのは、COMMIT、ROLLBACK および SAVEPOINT などです。

関連項目： トランザクション制御文の詳細は、1-51 ページの「[トランザクション](#)」を参照してください。

セッション制御文「セッション制御文」によって、ユーザーは自分のカレント・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 および SET ROLE の 2 つがあります。

システム制御文「システム制御文」は、Oracle サーバー・インスタンスのプロパティを変更します。システム制御文は ALTER SYSTEM のみです。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

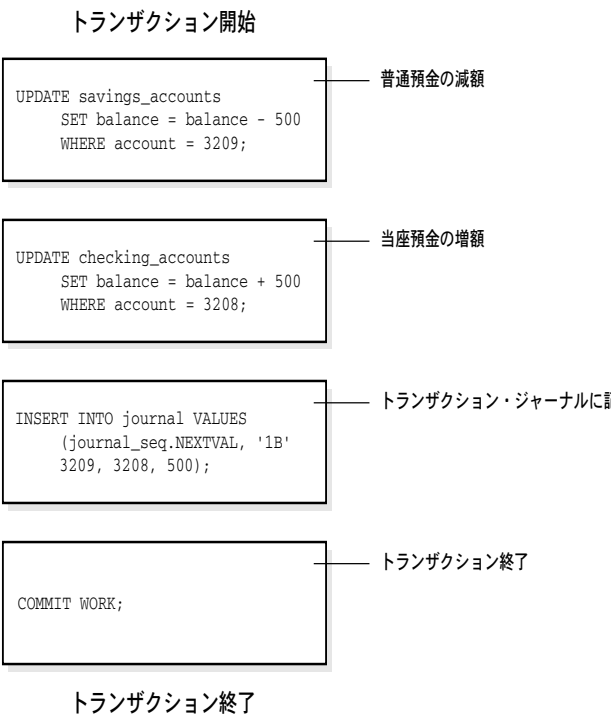
「トランザクション」は、単一のユーザーによって実行される 1 つ以上の SQL 文を含む論理作業単位です。Oracle が互換性を持つ ANSI/ISO SQL 規格によれば、トランザクションはユーザーの最初の実行可能 SQL 文で始まります。トランザクションは、ユーザーによって明示的にコミットまたはロールバックされるときに終了します（コミットとロールバックについては後述します）。

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り、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は、トランザクションの他の文の実行を取り消す必要があります。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 1-5 に銀行のトランザクションの例を示します。

図 1-5 銀行のトランザクション



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カレント位置から、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実行さ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 文のみを含めます。

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。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に、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します。
- 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ます。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のみ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ます。

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様々なプログラム・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。

関連項目：

- 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』
- 『Oracle8i Java パッケージ・プロシージャ リファレンス』

データベース・トリガー

Oracle では、表またはビューが変更された場合、またはなんらかのユーザー・アクションやデータベース・システム・アクションが発生した場合に暗黙的に実行（起動）されるプロシージャを、PL/SQL、Java または C で記述できます。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規定、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、NESTED TABLE 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。

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。

- メソッドを起動するには、対応付けられた型のオブジェクトを参照します。
- 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ます。

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。NESTED TABLE 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、組込み型を比較する 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と、その2つの四角形を比較できます。「オーダー・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。

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、あるビジネス・ルール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、ビジネス・ルールによって、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、表の列に対してビジネス・ルール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、その制約は規定できません。
- 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、エラーが戻されます。

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する必要があります。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ため、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）を許可しません。
UNIQUE	列（または列の集合）に重複値を許可しません。
PRIMARY KEY	列（または列の集合）に重複値と NULL を許可しません。
FOREIGN KEY	列または列の集合の値が、それぞれ関連する表の UNIQUE または PRIMARY KEY の値と一致している必要があります。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ません。

キー

「キー」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

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規定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、整合性ルールを規定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規定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、物理的または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」](#)

関連項目： 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

この章では、「データ・ディクショナリ」と呼ばれる、各 Oracle データベースの読取り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・ディクショナリの概要](#)
- [データ・ディクショナリの使用方法](#)
- [動的パフォーマンス表](#)

データ・ディクショナリの概要

「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**読取り専用**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。

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取り専用のため、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。

関連項目： SYSTEM 表領域の詳細は、3-6 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。

ベース表	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ため、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・ディクショナリのベース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、ベース表にあるデータを、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、ベース表ではなく、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ベース表とユーザー・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、スキーマ・オブジェクトおよび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します。
- 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します。
- Oracle ユーザーは、データベースについての情報の読取り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ます。

Oracle によるデータ・ディクショナリの使用法

データ・ディクショナリのベース表内のデータは、**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 のみにする必要があります。

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・アクセスの妥当性チェックやスキーマ・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ため、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 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 が実行するため、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 表は削除することはできませんが、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ため、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 は含まれません。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ます。

関連項目： 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します。
- ユーザーに関係する行のみを表示します。
- 列 OWNER が暗黙的に想定される（カレント・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ます。
- ALL_ ビューにある情報のサブセットを戻します。
- 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ます。

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 ビューへの問合せを発行するのは管理者のみであるため、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、カレント・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。

関連項目： DUAL 表の詳細は、『Oracle8i SQL リファレンス』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、カレント・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呼びます。

動的パフォーマンス表は実際の表ではないため、ほとんどのユーザーはこれにアクセスすることはありません。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ため、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 ビュー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 ビュー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。

関連項目： 動的性能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

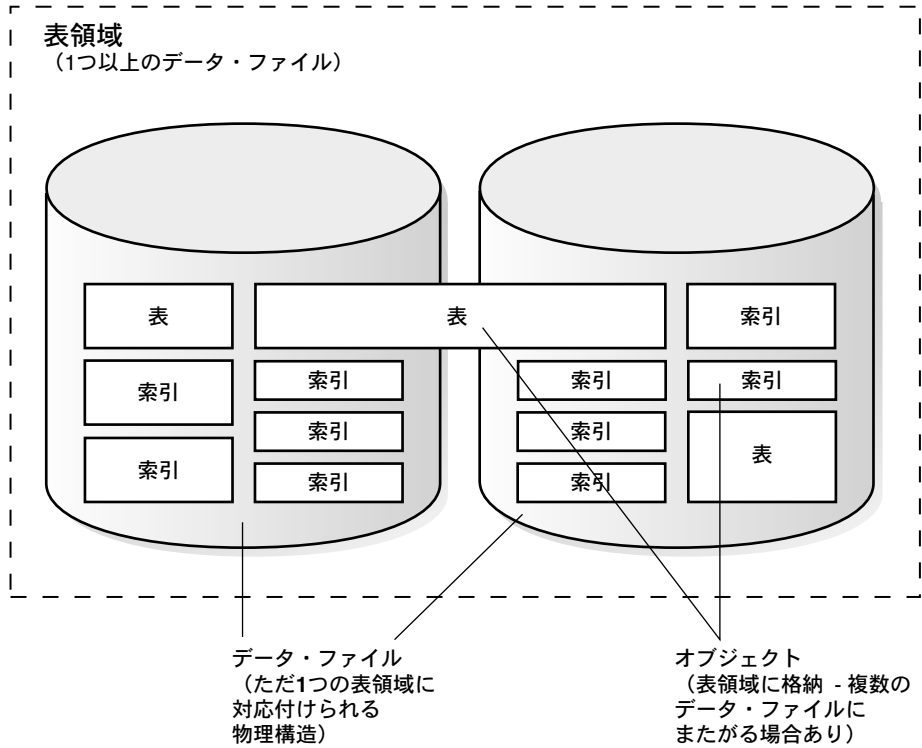
この章では、Oracle データベースの主要な論理データベース構造である表領域と、それぞれの表領域に対応する物理データ・ファイルについて説明します。この章の内容は、次のとおりです。

- データベース、表領域およびデータ・ファイルの概要
- 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・ファイルの概要

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）。

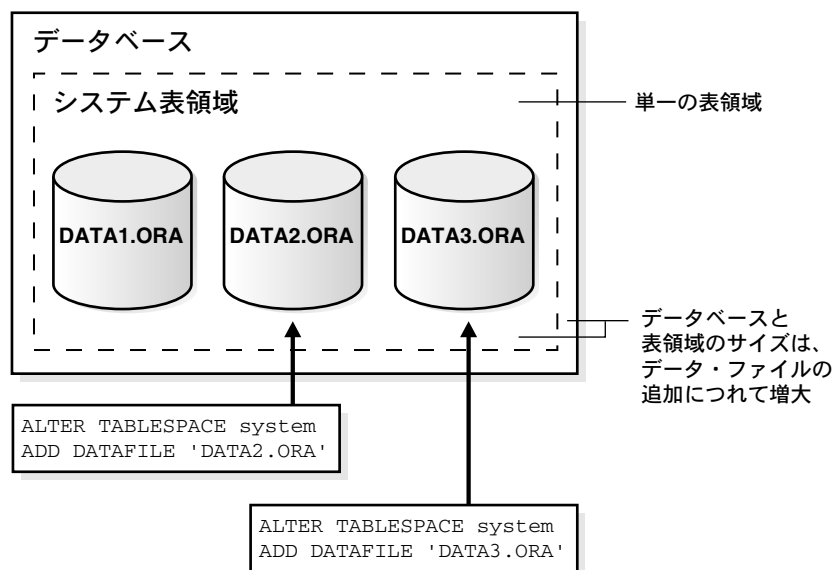
データベースへの多くの領域の割当て

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します。
- 新しい表領域を追加します。
- データ・ファイルのサイズを大きくします。

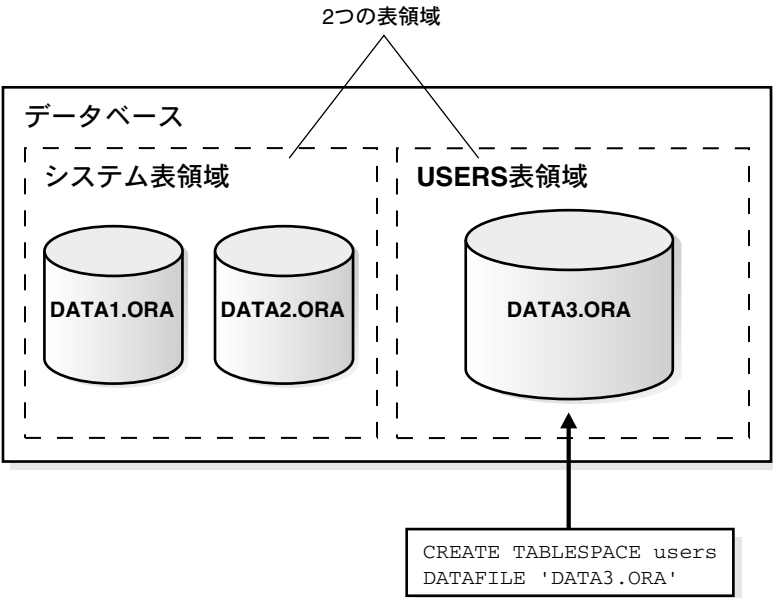
既存の表領域にデータ・ファイルを追加すると、その表領域に割り当てられているディスク領域の容量を増やすことになります。図 3-2 は、このようにして領域を拡大する方法を示しています。

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。

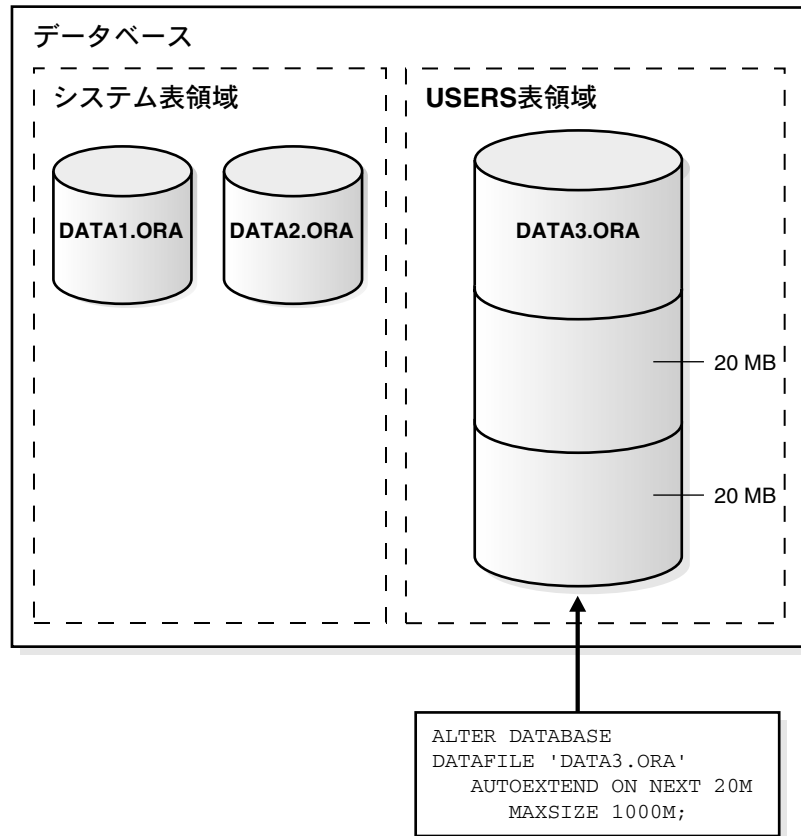
図 3-3 新しい表領域を追加してデータベースを拡大する



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



関連項目： 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。

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](#)
- [読取り専用表領域](#)
- [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](#)

関連項目： セグメントとエクステントの詳細は、[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。

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なっています。

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。

関連項目：

- SYSTEM 表領域の永続的なオンライン条件の詳細は、[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。
- PL/SQL プログラム・ユニットの領域要件の詳細は、[第 17 章「プロシージャとパッケージ」](#)および[第 19 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します。
- 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します。
- 各表領域を個別にオンライン化またはオフライン化して、データの可用性を制御します。
- 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します。
- 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ます。

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、表領域の読取り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。表領域では、次の2つの方法で空き領域と使用済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。

関連項目： 4-10 ページの「[エクステント](#)」

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。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 以前の Oracle では、この方法しか使用できませんでした。

関連項目： ディクショナリ表に関するロールバック情報の格納について
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。

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ます。エクステント内の領域を消費または解放することにより、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わせる必要がなくなります。

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。

この領域管理方法を指定するには、各種の CREATE 文の EXTENT MANAGEMENT 句で LOCAL 句を使用します。

- 永続表領域の場合は、CREATE TABLESPACE 文で EXTENT MANGEMENT LOCAL を指定します。
- 一時表領域の場合は、CREATE TEMPORARY TABLESPACE 文で EXTENT MANAGEMENT LOCAL を指定します。
- 現在、SYSTEM 表領域をローカル管理として作成する機能はサポートされていません。(Bug #809225)

関連項目：

- SQL 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- エクステンツ・サイズの詳細は、4-11 ページの「[ローカル管理のエクステンツ](#)」を参照してください。
- 一時表領域の詳細は、3-12 ページの「[一時表領域](#)」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。ただし、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化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します。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します。
- 表領域のオフライン・バックアップを実行します（ただし、オンラインで使用中の表領域のバックアップを実行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します。

使用中のロールバック・セグメントを含む表領域はオフライン化できません。

関連項目： ロールバック・セグメントの詳細は、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の文を含むアクティブ・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。それらの完了済の文に対応するロールバック・データは、SYSTEM 表領域にある遅延ロールバック・セグメントに保存されます。表領域が再びオンライン化されると、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化できるのは、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。

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、表領域をリカバリする必要があります。

関連項目：

- オンライン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。
- データ転送ツールの詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

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、様々なプロシージャに合わせて特定の表領域をオフライン化します。他の表領域はオンラインのままのため、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化すると、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ます。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。

読取り専用表領域

読取り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しなくてもよいようにすることです。読取り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取り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化できないため、読取り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 文の READ ONLY 句により、表領域を読取り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取り専用にすることができます。

ALTER TABLESPACE ... READ ONLY 文は、表領域を「読取り専用推移」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文に対して、読取り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 文が完了し、表領域は読取り専用モードになります。

注意： 読取り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 文は失敗します。

その後、ALTER TABLESPACE 文の READ WRITE 句を使用して、読取り専用表領域を再び読み書き可能にすることができます。

表領域を読取り専用にしても、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。読取り専用表領域に含まれるデータ・ファイルをオンライン化すると、読込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、書き込みができません。読取り専用表領域のデータ・ファイルは、ALTER DATABASE 文の DATAFILE 句を使用すれば、個別にオンライン化またはオフライン化できます。

読取り専用表領域は修正できません。読取り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、その表領域を読取り専用に再設定します。

読取り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、データベースをリカバリする必要があるときにも、読取り専用表領域は修正されていないため、リカバリする必要があるありません。ただし、インスタンス・リカバリまたはメディア・

リカバリ時には、読取り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。

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、読取り専用表領域からも削除できます。ただし、読取り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。

読取り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化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、読取り専用表領域ではその書き込み操作が禁止されているためです。

関連項目：

- 表領域を読取り専用または読取り / 書き込みモードに変更する方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。
- ALTER TABLESPACE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- リカバリの詳細は、『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。

一時表領域

ソート操作用の領域は、ソート専用指定した「一時表領域」を使用すると、さらに効率的に管理できます。それにより、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シリアル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 文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。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用できます。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、エクステンツが割り当てられて拡大します。

関連項目： セグメントの詳細は、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。

一時表領域の作成および変更

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

- ディクショナリ管理の一時表領域の場合は、CREATE TABLESPACE の TEMPORARY 句を使用します。
- ローカル管理の一時表領域の場合は、CREATE TEMPORARY TABLESPACE を使用します。この文では、DATAFILES ではなく TEMPFILES を指定します。

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 文を使用すると、表領域を PERMANENT から TEMPORARY に、またはその逆に変更できます。

関連項目：

- TEMPFILES の詳細は、3-17 ページの「一時データ・ファイル」を参照してください。
- ローカル管理とディクショナリ管理の表領域の詳細は、3-7 ページの「表領域内の領域管理」を参照してください。
- CREATE TABLESPACE、CREATE TEMPORARY TABLESPACE および ALTER TABLESPACE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- 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の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』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作成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、その表領域を読み取り専用にし、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。

関連項目： 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。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。

関連項目： オブジェクト参照の移動方法については、13-9 ページの「REF」を参照してください。

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。

関連項目： 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様々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します。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します。表をパーティションに変換することにより、新しいデータを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・マートに移動します。
- 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リストアできるように、アーカイブ・データをメタデータとともに保管します。

関連項目： データ・ウェアハウスとデータ・マートの詳細は、『Oracle8i データ・ウェアハウス』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。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 SYSTEM 表領域に自動的に割り当てられます。

関連項目： オペレーティング・システムでデータ・ファイルのファイル・ヘッダーに必要な領域の量については、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・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。

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。

関連項目： 領域使用の詳細は、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。

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、各表領域に含まれるデータ・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。

関連項目： 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）化または「オンライン」（使用可能）化できます。表領域をオフライン化またはオンライン化すると、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化またはオンライン化されます。

個々のデータ・ファイルをオフライン化することもできます。ただし、通常、この操作を行うのは、一部のデータベース・リカバリ手順を実行する場合のみです。

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・ファイル」）があります。

- テンポラリ・ファイルは、常に NOLOGGING モードに設定されます。
- テンポラリ・ファイルを読み取り専用にすることはできません。
- テンポラリ・ファイルは改名できません。
- ALTER DATABASE 文ではテンポラリ・ファイルを作成できません。
- テンポラリ・ファイルはメディア・リカバリでは認識されません。
 - BACKUP CONTROLFILE ではテンポラリ・ファイルの情報は生成されません。
 - CREATE CONTROLFILE ではテンポラリ・ファイルに関する情報を指定できません。
- テンポラリ・ファイル情報は、ディクショナリ・ビュー DBA_TEMP_FILES と動的性能ビュー V\$TEMPFILE には表示されますが、DBA_DATA_FILES や V\$DATAFILE ビューには表示されません。

関連項目： ローカル管理の表領域の詳細は、3-7 ページの「[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

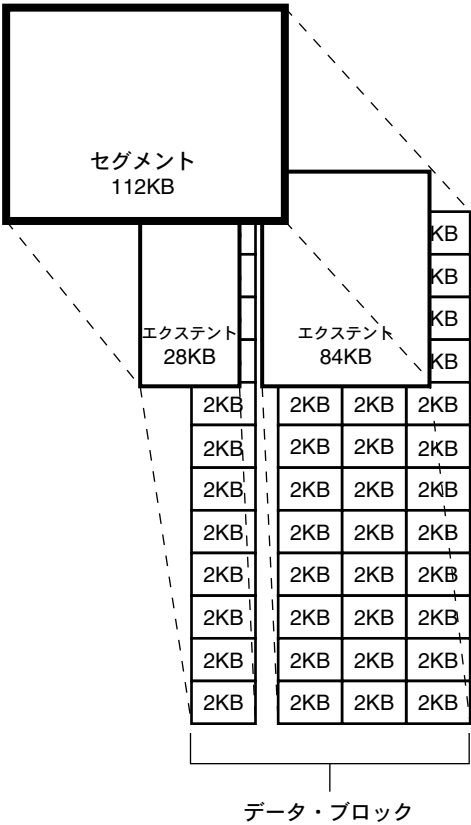
この章では、Oracle Server の論理的な記憶構造の性質と、それらの構造の相互関係について説明します。この章の内容は、次のとおりです。

- データ・ブロック、エクステントおよびセグメントの概要
- 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概要

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いっぱいであれば、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。

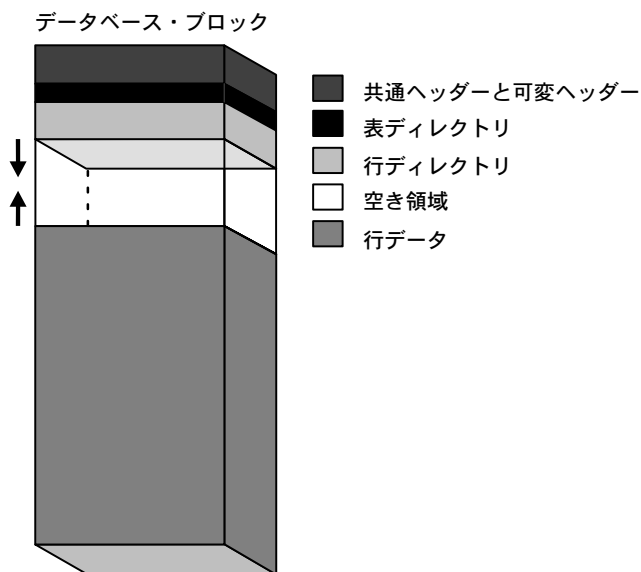
どの Oracle データベースについても、その作成時にデータ・ブロックのサイズを設定します。このデータ・ブロック・サイズは、不必要な I/O を避けるための最大値の範囲内で、オペレーティング・システムのブロック・サイズの倍数です。Oracle データ・ブロックは、Oracle が使用および割当て可能な最小の格納単位です。

関連項目： 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。

関連項目： 4-9 ページの「[行連鎖と移行](#)」

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・ブロックで実際に行われるかどうかは、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。

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。

関連項目： 領域管理パラメータの詳細は、4-5 ページの「[PCTFREE、PCTUSED、行連鎖の概要](#)」を参照してください。

PCTFREE、PCTUSED、行連鎖の概要

PCTFREE および 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。

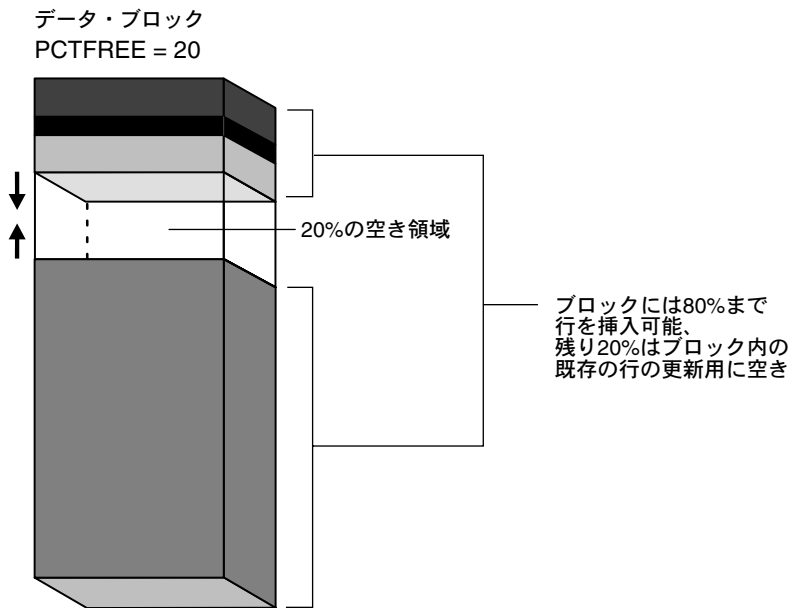
PCTFREE パラメータ

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

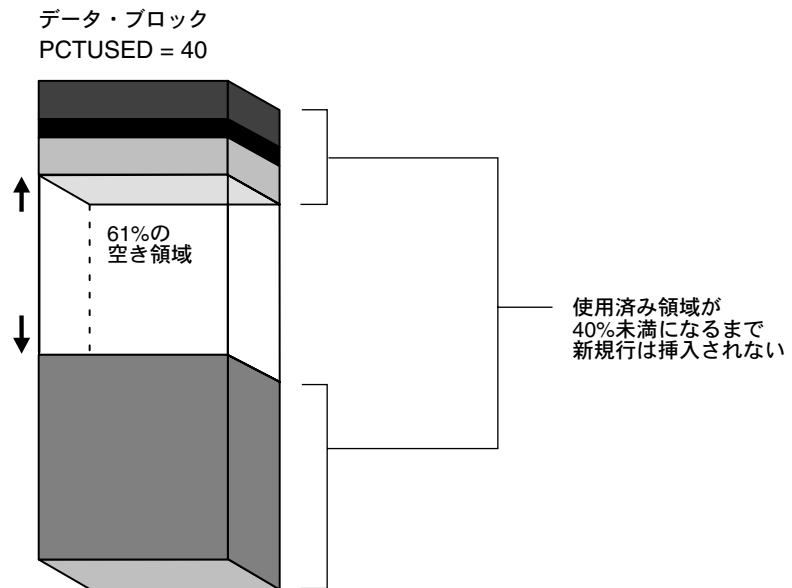
PCTUSED パラメータは、新しい行をブロックに追加するときに、行データとオーバーヘッドに「使用」できるブロックの割合の最小値を設定します。データ・ブロックが PCTFREE で指定した限界値まで満たされると、そのブロックにおける割合がパラメータ PCTUSED の値を下回るまで、Oracle はそのブロックを新しい行の挿入には使用できないものとみなしま

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。

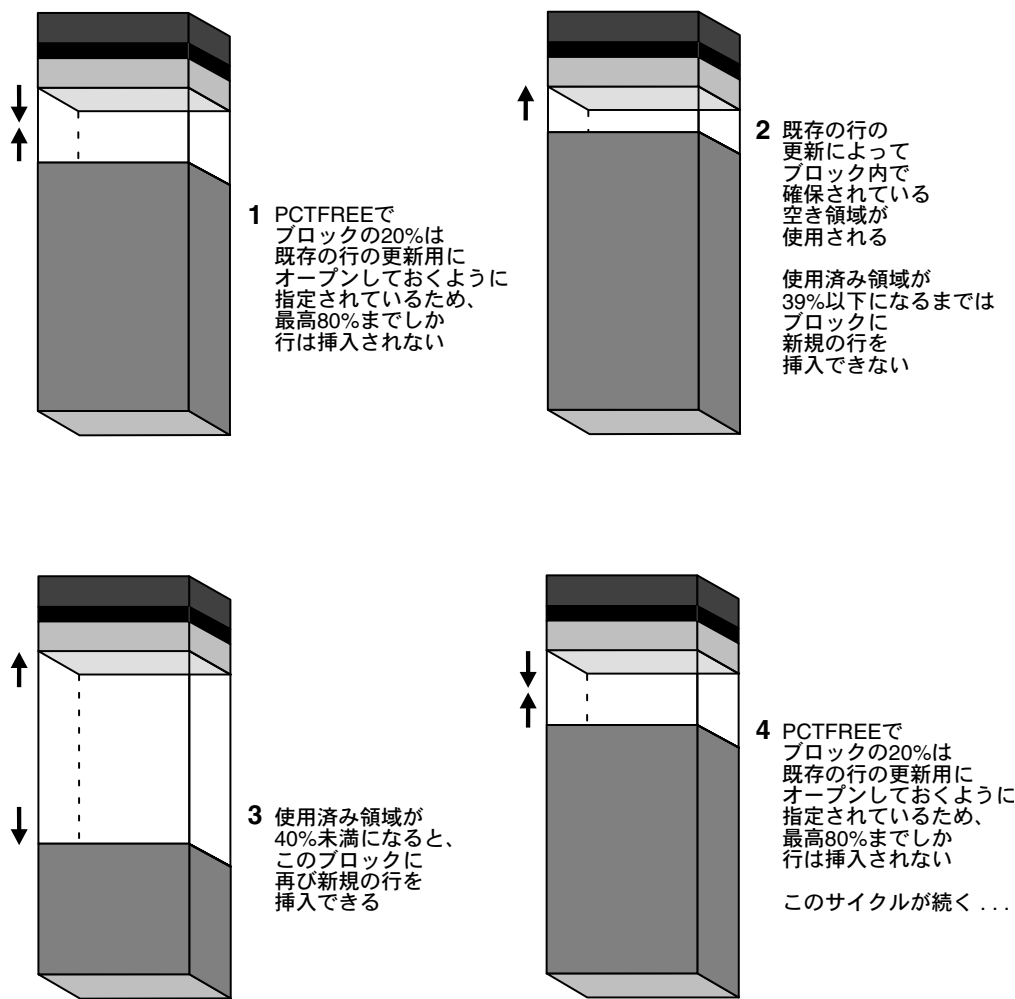
図 4-4 PCTUSED



PCTFREE と PCTUSED の機能

PCTFREE と PCTUSED の連動により、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ツ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、空きリストから外されます。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ます。
- 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ます。

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。前述の場合にのみ圧縮が行われるのは、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ため、データベース・システムのパフォーマンスが低下するからです。

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。

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いっぱい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。

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・ブロックをスキャンする必要があるためです。

関連項目：

- 行および行断片の形式の詳細は、10-5 ページの「[行の形式とサイズ](#)」を参照してください。
- ROWID の詳細は、10-8 ページの「[行断片の ROWID](#)」を参照してください。
- ROWID については、12-15 ページの「[物理 ROWID](#)」を参照してください。
- 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 パフォーマンスのための設計およびチューニング』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・ブロックがいっぱいになり、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。

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。

関連項目：

- 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](#)」
- 23-34 ページの「[空き領域とパラレル DDL](#)」

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 句で記憶域パラメータを指定すると、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。記憶域パラメータには、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。

関連項目： 3-7 ページの「[表領域内の領域管理](#)」

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）句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ます。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 を使用できます。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ツの場合、記憶域パラメータ NEXT、PCTINCREASE、MINEXTENTS、MAXEXTENTS および DEFAULT STORAGE 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。

関連項目：

記憶域パラメータの詳細は、次のマニュアルを参照してください。

- 『Oracle8i 管理者ガイド』
- 『Oracle8i SQL リファレンス』

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し、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ます。）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、正確に 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る、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て、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 句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ます。
- 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ります。
- 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ます。

```
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。

関連項目：

エクステントの割当て解除の詳細は、次のマニュアルを参照してください。

- 『Oracle8i 管理者ガイド』
- 『Oracle8i SQL リファレンス』

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する限り、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、エクステントを合わせると呼びます。

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、エクステントを合わせる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。

関連項目： マテリアライズド・ビューとそのログの詳細は、10-18 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どうか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。

関連項目： 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](#)」

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、ソート専用を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ため、他の複数ソートでそのセグメントを使用できます。

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。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。

関連項目：

- 4-17 ページの「一時セグメント」
- 10-11 ページの「一時表」

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 文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域パラメータによって決定されます。これらの記憶域パラメータは、適切な CREATE または ALTER 文によって直接設定できます。これらの記憶域パラメータは、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。

関連項目：

- スナップショットおよびスナップショット・ログの詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。
- CREATE 文および ALTER 文の詳細は、『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 文を発行した時点で作成されます。この文では、索引セグメントのエクステントの記憶域パラメータと、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

次の文では、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

文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。

関連項目： 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。

関連項目： 10-11 ページの「一時表」

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 句を付けた CREATE USER または ALTER USER 文で指定します。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ツの記憶特性は、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。

REDO ログには、ソート操作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。

関連項目： ユーザーの一時セグメント表領域を割り当てて方法の詳細は、第 26 章「データベース・アクセスの制御」を参照してください。

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。

関連項目： 10-11 ページの「一時表」

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、読取り一貫性の実現、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リカバリのために使用されます。

関連項目：

- 読取り一貫性の詳細は、24-4 ページの「[マルチバージョン一貫性制御](#)」を参照してください。
- 16-6 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・リカバリの詳細は、29-9 ページの「[ロールバック・セグメントとロールバック](#)」を参照してください。

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込みを実行できません。Oracle のみが書込みや読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・リカバリまたはメディア・リカバリの一部として自動的にリストアされます。リ

カバリが完了すると、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、問合せに対して読取り一貫性を保った結果を作成するために使用されます。

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リストア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、問合せに対して読取り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ます。トランザクションの割当ては、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、読取り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ます。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取り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

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いっぱいになると、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してロールバック・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。

注意： 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。

ロールバック・セグメントの作成時には、記憶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る必要があります。

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カレント・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。

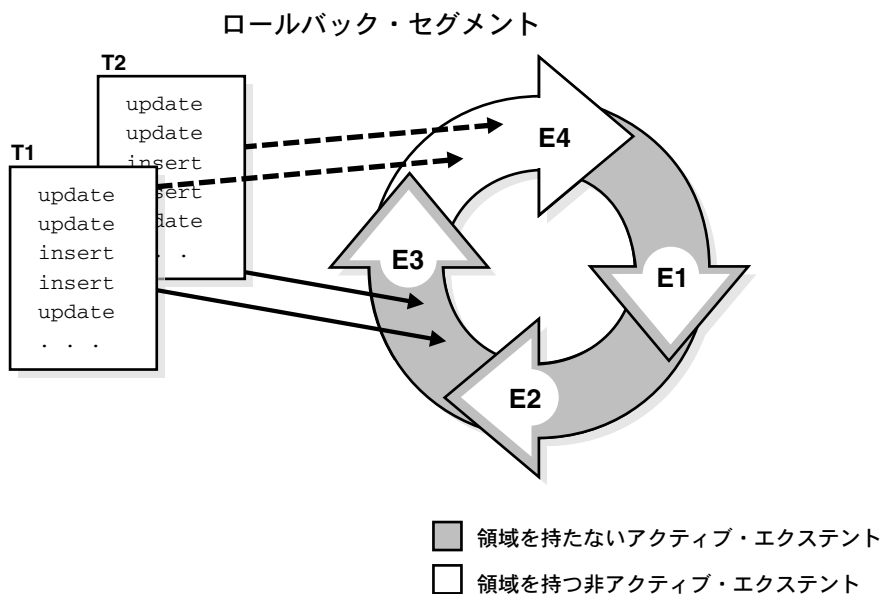
- すでにロールバック・セグメントに割り当てられているエクステントを再使用します。
- ロールバック・セグメント用に新しいエクステントを取得します（そして割り当てます）。

ロールバック領域をさらに取得する必要の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 はそれをカレント・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、新しいカレント・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。

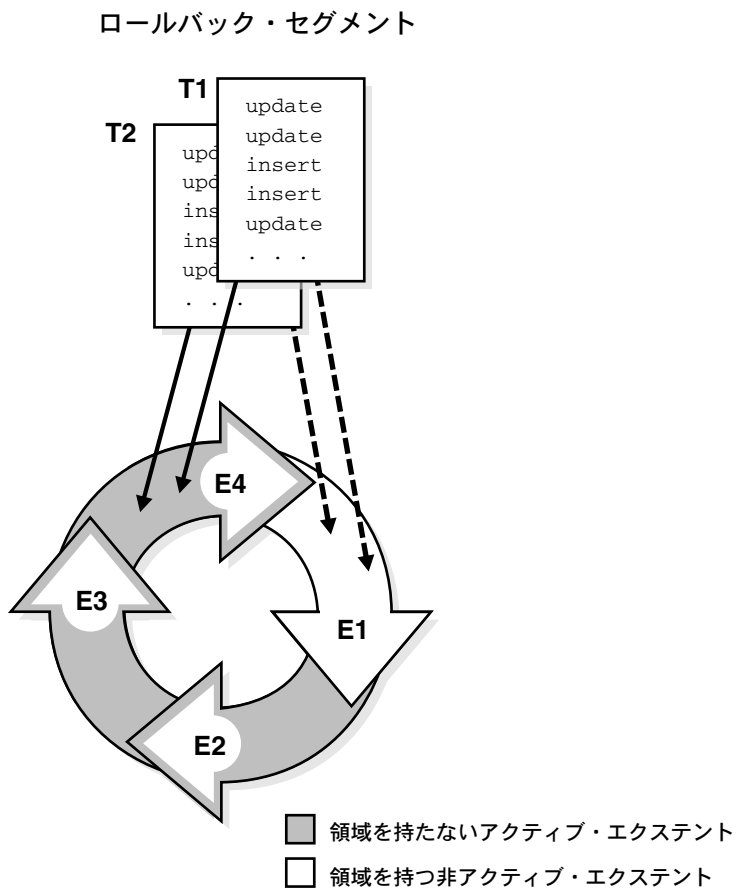
図 4-6 は、ロールバック・セグメントの第 3 エクステント（E3）への書き込みを開始し、続いて第 4 エクステント（E4）に書き込む、2 つのトランザクション T1 および T2 を示しています。

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、カレント・エクステントがいっぱいになると、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 が完全にいっぱいであるときに 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

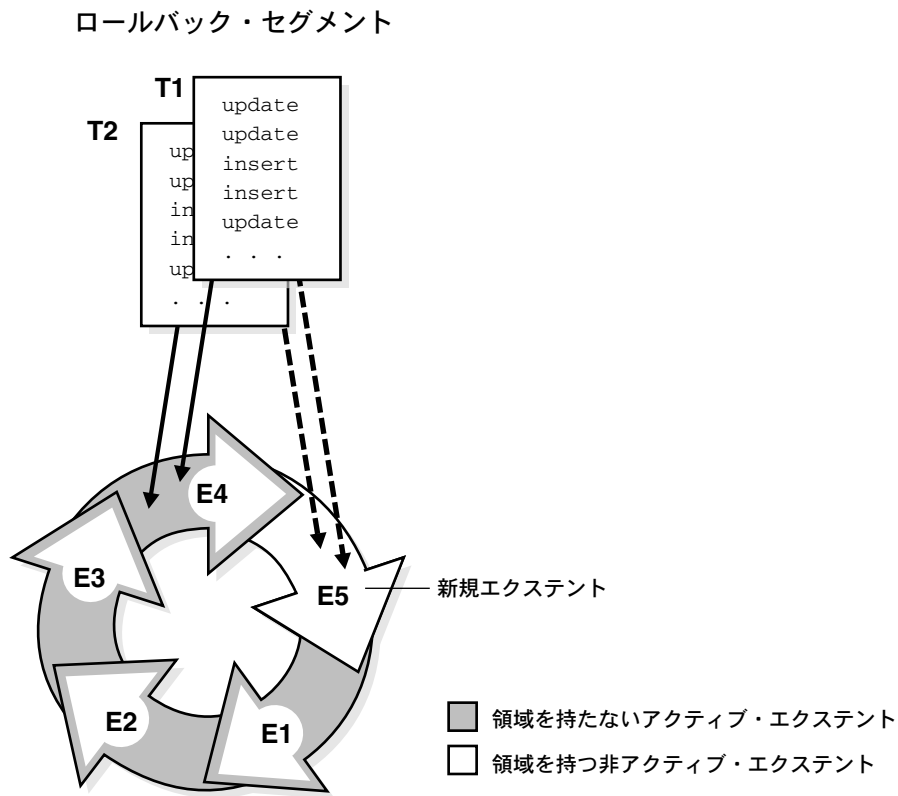


トランザクションのロールバック情報の書込みを継続するときに、Oracle は常にリング内の次のエクステントを最初に再利用しようとします。ただし、次のエクステントにアクティブ・トランザクションからのデータが含まれている場合は、新しいエクステントを割り当てる必要があります。エクステントの数が、ロールバック・セグメントの記憶域パラメータ MAXEXTENTS に設定されている値に達するまで、Oracle は、ロールバック・セグメントに新しいエクステントを割り当てることができます。

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が完全にいっぱいになると、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・セグメントのカレント・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、かついっぱい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、一貫した読取りで使用される可能性が最も低いいため、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。たとえば、次のとおりです。

(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・セグメントを常に取得します。

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使用できる、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 環境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。前述の例で、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化されるか、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ります。

関連項目：

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、次のマニュアルを参照してください。

- 『Oracle8i Parallel Server 概要』
- 『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

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、リカバリ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利用できるかどうか、および DBA がどの管理処理を実行できるかが決まります。

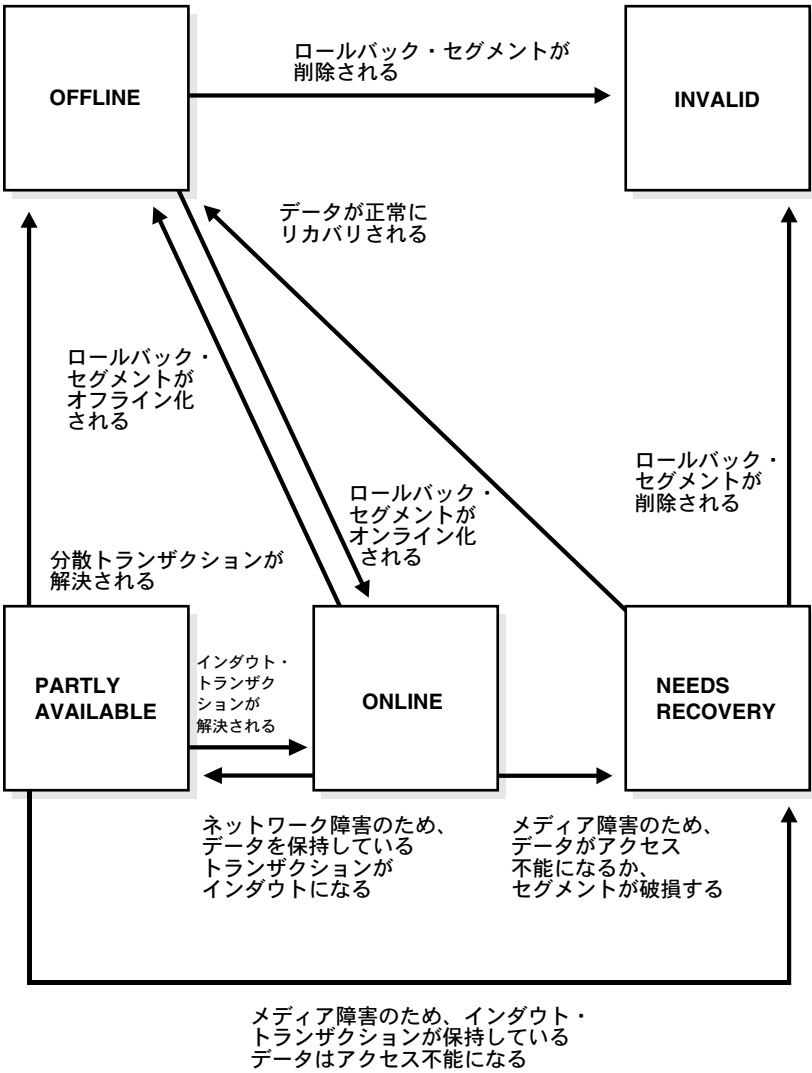
ロールバック・セグメントの状態は、次のとおりです。

OFFLINE	インスタンスによって取得されていません（オンライン化されていません）。
ONLINE	インスタンスによって取得されています（オンライン化されています）。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ません。
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ます。
INVALID	削除されています（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ます。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化できます。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化するには、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。（データベースをリカバリ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ます。（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。

PARTLY AVAILABLE のロールバック・セグメントをオンライン化する（文を使用するか、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。

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化するよりも、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。

関連項目：

- 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。
- 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。

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リカバリされると、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・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

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シー
ジャを説明します。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

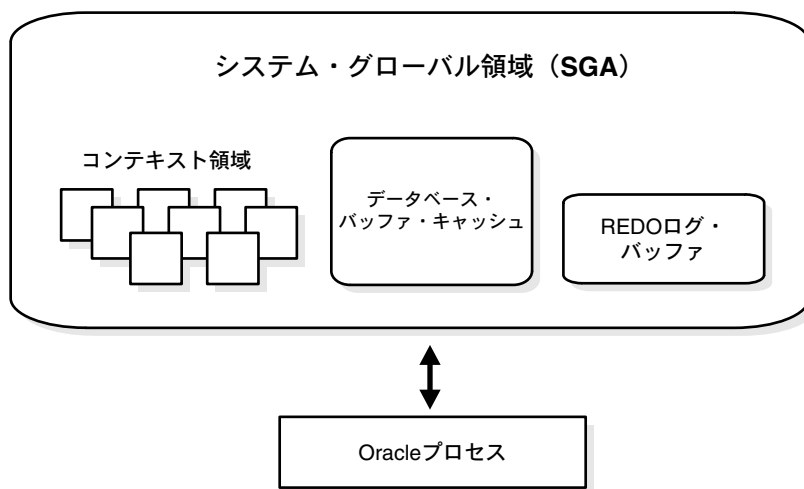
稼動中のすべての Oracle データベースは、Oracle インスタンスに対応付けられます。データベース・サーバー上のデータベースが起動すると、コンピュータの種類にかかわらず、Oracle はシステム・グローバル領域（SGA）というメモリー領域を割り当て、1 つ以上の Oracle プロセスを起動します。この SGA と Oracle プロセスの組合せのことを、「Oracle インスタンス」と呼びます。インスタンスのメモリーとプロセスは、対応付けられたデータベースのデータを効果的に管理し、データベースを使用する 1 人以上のユーザーのために機能します。

図 5-1 に Oracle インスタンスを示します。

関連項目：

- [第 7 章「メモリー・アーキテクチャ」](#)
- [第 8 章「プロセス・アーキテクチャ」](#)

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」と呼びます。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。クラスタ化された超並列システム（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。

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 データベースの現在の状態を制御できません。

関連項目： Oracle Parallel Server の詳細は、『Oracle8i Parallel Server 概要』を参照してください。

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こと。
- 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こと。
- 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こと。

SYSDBA または SYSOPER 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。SYSOPER 権限は、SYSDBA 権限のサブセットです。

関連項目：

- 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。
- 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 26 章「データベース・アクセスの制御」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」と呼びます。

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
- いっぱい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、データベースの様々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 文または ALTER SYSTEM 文を使用して「動的」に変更することもできます。

関連項目：

- 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、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。

関連項目： 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。

関連項目：『Oracle Enterprise Manager 管理者ガイド』

インスタンスの起動

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。

関連項目：

- SGAの詳細は、第7章「メモリー・アーキテクチャ」を参照してください。
- バックグラウンド・プロセスの詳細は、第8章「プロセス・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。

データベースのマウント

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ため、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： この項で説明している機能は、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。この PARALLEL_SERVER パラメータが TRUE に設定されていれば、他のインスタンスも PARALLEL_SERVER パラメータが 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。

関連項目：

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、次のマニュアルを参照してください。

- 『Oracle8i Parallel Server 管理、配置およびパフォーマンス』
- 『Oracle8i Parallel Server セットアップおよび構成ガイド』

スタンバイ・データベースのマウント

「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、常にリカバリ・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 文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 REDO ログを適用する必要があります。

スタンバイ・データベースは、読取り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。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。

関連項目：

- スタンバイ・データベースと障害リカバリの詳細は、29-27 ページの「[耐障害性](#)」を参照してください。
- スタンバイ・データベースを読取り専用モードでオープンする方法については、5-9 ページの「[データベースの読取り専用モードでのオープン](#)」を参照してください。

クローン・データベースのマウント

「クローン・データベース」は、表領域の Point-in-Time リカバリに使用できる、データベースの特殊コピーです。表領域の Point-in-Time リカバリを実行する場合は、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リカバリします。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、リカバリされた表領域からデータ・ファイルをコピーします。

関連項目： クローン・データベースと表領域の Point-in-Time リカバリの詳細は、『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に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。

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リカバリする必要があります。

関連項目： オフライン表領域をオープンする方法については、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。

クラッシュ・リカバリ

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・リカバリを自動的に実行します。

関連項目： 29-4 ページの「[データベース・インスタンス障害](#)」

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。

関連項目： 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](#)」

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・リカバリが完了すると、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。

関連項目：

分散トランザクションの障害からのリカバリの詳細は、次の章およびマニュアルを参照してください。

- [第 30 章「分散データベースの概念」](#)
- 『Oracle8i 分散システム』

データベースの読取り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、読取り専用モードでオープンできます。読取り専用モードでは、データベース・アクセスは読取り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き込みは、読取り専用モードでも実行できます。データベースを読取り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、データベースを読取り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化できません。読取り専用モードでは、ジョブ・キューを使用できません。

データベース・リカバリや、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、読取り専用モードでは、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ます。
- オフラインのデータ・ファイルと表領域のリカバリを実行できます。
- 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ます。

読取り専用モードは、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取り専用モードでオープンする必要があります。

関連項目： データベースを読取り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せん。

関連項目：『Oracle Enterprise Manager 管理者ガイド』

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・データとリカバリのためのデータが、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ため、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ため、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・リカバリ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・リカバリが実行されません。

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 文を発行することにより、新しいインスタンスの起動を強制実行できます。

関連項目： 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。

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・アーキテクチャの概要](#)
- [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・アーキテクチャの概要

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・アーキテクチャの様々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。

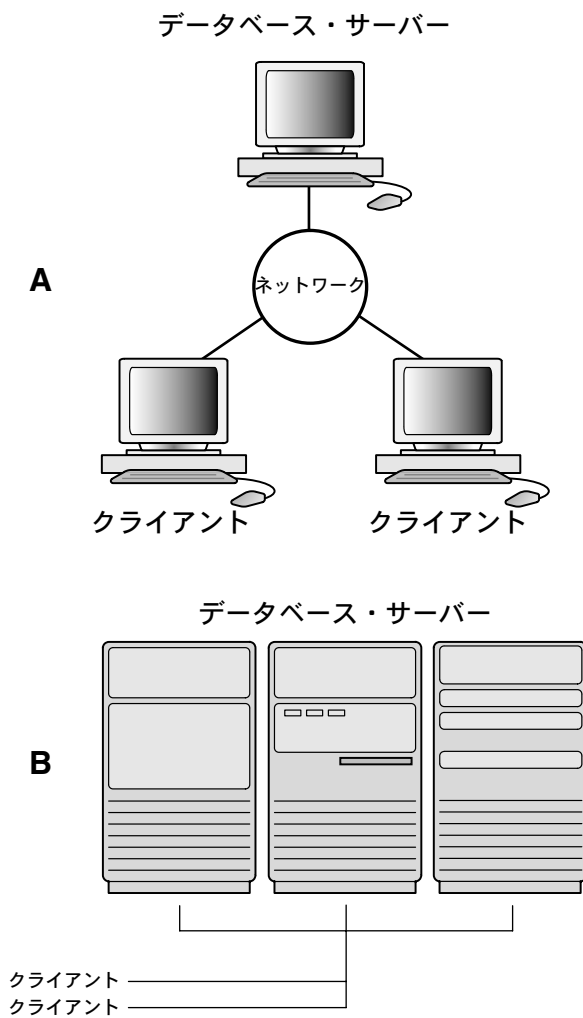
- 図の A では、クライアントとサーバーを別々のコンピュータに配置し、各コンピュータをネットワークで接続しています。Oracle データベース・システムのサーバーとクライアントは、Oracle のネットワーク・インタフェースである Net8 を介して通信します。
- B では、1 つのコンピュータに複数のプロセッサが含まれており、クライアント・アプリケーションと Oracle を別々のプロセッサで実行しています。

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。

関連項目：

- Net8 の詳細は、6-4 ページの「[Net8](#)」を参照してください。
- 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 30 章「分散データベースの概念」](#)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ません。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ません。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キング機能と共有メモリ機能を活用できます。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ます（大容量のメモリ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ます。
- Oracle は、システムの拡大とともに必要に応じて「スケール」できます。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（「水平スケール」）、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（「垂直スケール」）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ため、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ます。このため、同時アクセスの管理が簡単に効率的になります。
- 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 文を使用できます。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。

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、ソフトウェアで実装されます。
- 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す。

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 Server 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 Server 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、手でコーディングする必要があります。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。

関連項目： プログラム・インタフェースの詳細は、[第 8 章「プロセス・アーキテクチャ」](#)を参照してください。

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 からの応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。

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、ドライバ・ソフトウェアによって追加の 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。

関連項目：

- Net8 の動作の詳細は、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。
- 『Oracle8i Net8 管理者ガイド』

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。

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せん。

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・バランス機能と接続フェイルオーバー機能が使用可能になります。

関連項目：

- 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](#)」を参照してください。
- サーバー・プロセスの詳細は、8-22 ページの「[専用サーバー構成](#)」を参照してください。
- 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。
- Oracle Parallel Server でのインスタンス登録およびクライアント / サーバー接続の詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』および『Oracle8i Parallel Server 管理、配置およびパフォーマンス』を参照してください。

複数層アーキテクチャ

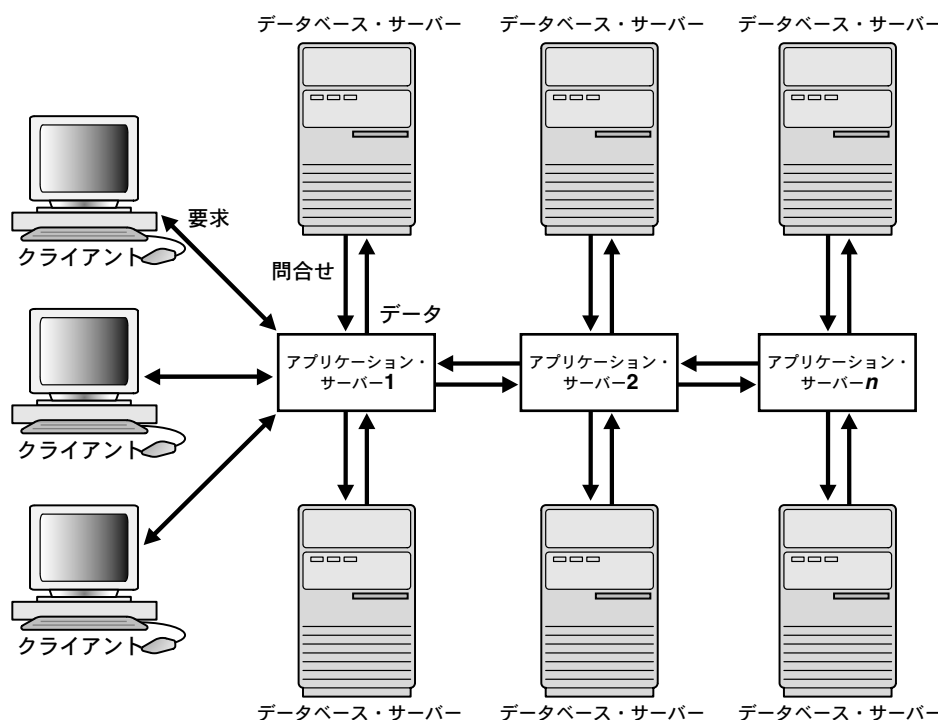
複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントとデータベース・サーバー間のインターフェースとして機能します。

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します。
- データベース・サーバーに接続します。
- 要求された操作を実行します。

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想定します。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。

関連項目： 複数層環境におけるセキュリティの注意点の詳細は、26-9 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

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次のよう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（データ・ブロックや REDO ログ・エントリなど）

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。

SGA と Oracle 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。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がある、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。

関連項目：

- Oracle インスタンスの詳細は、5-2 ページの「[Oracle インスタンスの概要](#)」を参照してください。
- 7-14 ページの「[プログラム・グローバル領域 \(PGA\)](#)」
- 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](#)」

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と、マルチ・プロセッサ・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、書込みリストおよび LRU リストという 2 つのリストに編成されます。「書込みリスト」は、「使用済バッファ」を保持します。使用済バッファとは、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。「LRU リスト」は、使用可能バッファ、使用中バッファおよび書込みリストに移動していない使用済バッファを保持します。「使用可能バッファ」は、有効なデータが含まれておらず、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、使用可能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 LRU 側から検索を開始します。使用可能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バッファを見つけた場合、プロセスはそのバッファを書込みリストに移動してから検索を続けます。使用可能バッファが見つかる、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。

使用可能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。

関連項目： 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 アルゴリズムとフル・テーブル・スキャン

ユーザー・プロセスは、フル・テーブル・スキャンを実行するときに、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、フル・テーブル・スキャン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、フル・テーブル・スキャン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ためです。

表スキャン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。フル・テーブル・スキャン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できます。

関連項目： 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。キャッシュがいっぱい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データをディスクに書き込みます。(バッファが使用済でなければ、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ると、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。

関連項目： バッファ・キャッシュの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』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します。
- 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します。
- 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ます。

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。

関連項目：

- バッファ・プールの詳細は、『Oracle8i パフォーマンスのための設計 およびチューニング』を参照してください。
- STORAGE 句の BUFFER_POOL 句の構文については、『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リカバリ時に使用されます。

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。

関連項目：

- 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 \(LGWR\) プロセス](#)」を参照してください。
- 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。

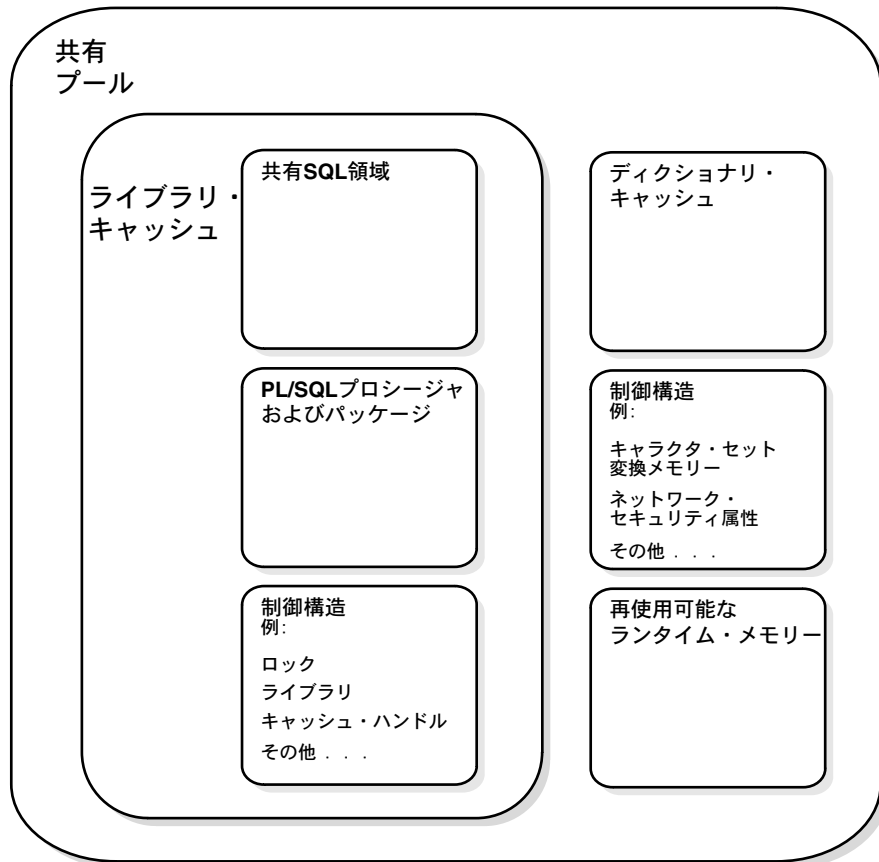
初期化パラメータ LOG_BUFFER は、REDO ログ・バッファのサイズをバイト単位で設定します。一般的に（トランザクションが長い、数が多い場合には特に）、設定する値を大きくするほどログ・ファイルの I/O は少なくなります。デフォルト設定は、ホスト・オペレーティング・システムの最大データ・ブロック・サイズの 4 倍です。

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他、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・キャッシュとともに、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 文または類似する 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、類似する複数の DML 文に 1 つの共有 SQL 領域を使用し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。

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 LRU 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。

関連項目：『Oracle8i パフォーマンスのための設計およびチューニング』

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ランタイム・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。

プライベート SQL 領域は、さらに持続領域とランタイム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）およびその他の状態情報（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態）が格納されます。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。
- 「ランタイム領域」には、SQL 文の実行時に使用する情報が含まれます。ランタイム領域のサイズは、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、ランタイム領域のサイズは、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ランタイム領域を作成します。INSERT 文、UPDATE 文および DELETE 文では、Oracle は文の実行後にランタイム領域を解放します。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、ランタイム領域を解放します。

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、持続領域およびランタイム領域 (SELECT 文の場合) は SGA 内にあります。

関連項目：

- 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。
- SELECT 文でソート中のランタイム領域の詳細は、7-16 ページの「[ソート領域](#)」を参照してください。

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。

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ランタイム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、電子メール・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、ユーザーはテンプレートのリストから選択できます。電子メール・アプリケーションが最初の 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。

関連項目： 15-7 ページの「[カーソル](#)」**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。

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に、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ため、「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。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。

関連項目：

- 第 2 章「データ・ディクショナリ」
- 7-7 ページの「ライブラリ・キャッシュ」

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と、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。

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、クラスタまたは索引の統計を更新または削除するために ANALYZE 文を使用する場合、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。
- 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、カレント・インスタンスを停止する必要がありません。

関連項目：

- プライベート SQL 領域の位置の詳細は、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。
- SQL 文の無効化と依存性の問題の詳細は、[第 20 章「Oracle の依存性の管理」](#)を参照してください。

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

大規模プールからセッション・メモリーをマルチスレッド・サーバー用または Oracle XA 用に割り当てることにより、共有プールを主に共有 SQL のキャッシュに使用して、共有 SQL キャッシュの縮小によるパフォーマンスのオーバーヘッドを回避できます。

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。

関連項目：

- マルチスレッド・サーバー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方法の詳細は、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。
- 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。
- 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 パフォーマンスのための設計およびチューニング』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収まる必要があります。SGA 全体が実メモリーに収まらず、その一部を格納するために仮想メモリーが使用される場合、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大幅に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。

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 の SHOW 文に SGA 句を指定して、カレント・インスタンスの SGA サイズを表示することもできます。

関連項目：

- 7-17 ページの「[仮想メモリー](#)」
- 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』を参照してください。
- SQL*Plus で SGA のサイズを表示する方法の詳細は、『Oracle8i 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 パフォーマンスのための設計およびチューニング』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 インストール・ガイドまたはユーザーズ・ガイドを参照してください。

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。

関連項目：

- USE_INDIRECT_DATA_BUFFERS 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。
- オペレーティング・システム固有の情報は、該当する Oracle インストール・ガイドまたはユーザーズ・ガイドを参照してください。

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。

関連項目： セッションについては、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。

関連項目： 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

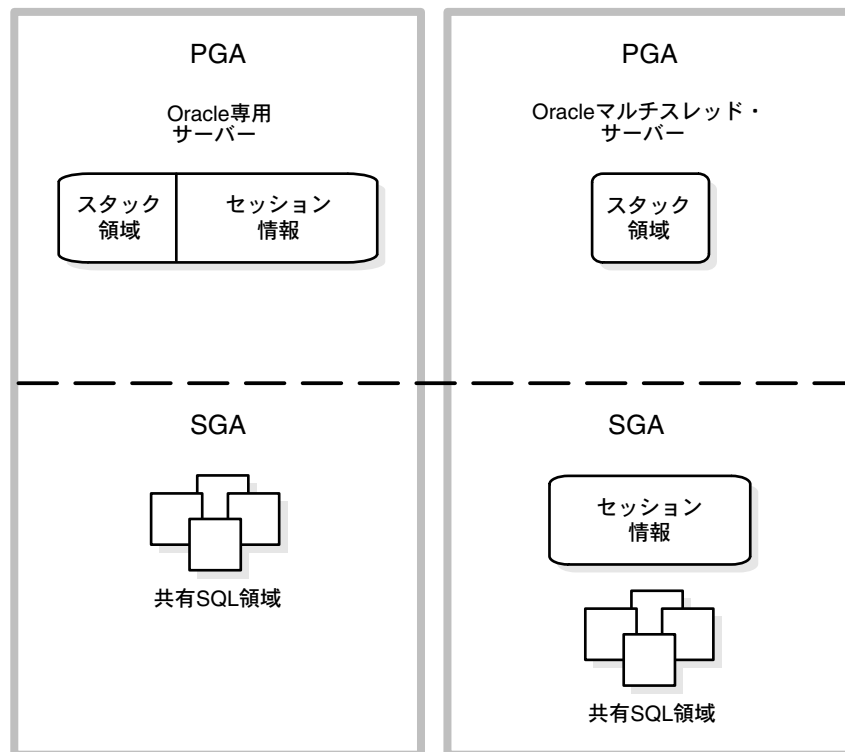
PGA には、「スタック領域」が必ず含まれます。スタック領域とは、セッションの変数、配列およびその他の情報を保持しておくために割り当てられるメモリーです。

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、様々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 領域がすべて使用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。

関連項目： 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」と呼びます。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 領域のランタイム領域内に存在します。SELECT 文の場合、プライベート SQL 領域内のこのメモリーは、接続構成に応じて様々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

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して、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)」と呼びます。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。

関連項目：

- プライベート SQL メモリーがどこから取り込まれるかの詳細は、7-8 ページの「[プライベート SQL 領域](#)」を参照してください。
- 専用サーバー構成でのプライベート SQL 領域の詳細は、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・サーバー構成でのプライベート SQL 領域の詳細は、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。

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して、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よくなります。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。

注意： 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・アーキテクチャ

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・モジュールを実行する必要があります。

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 サーバー・コード	各ユーザーは自分のために Oracle サーバー・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。

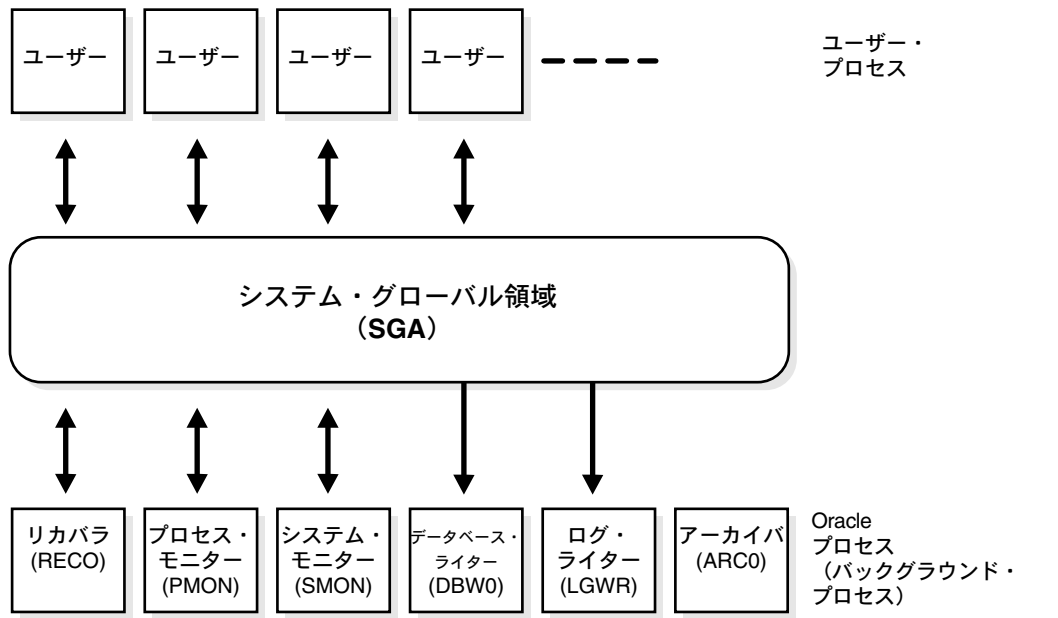
- ユーザー・プロセスは、アプリケーションまたは Oracle Tool のコードを実行します。
- Oracle プロセスは、Oracle Server のコードを実行します。それらのプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。

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。接続ユーザー用のコードを構成するには、次の3つの方法があります。

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 サーバー・コードを実行するプロセス（専用サーバー・プロセス）を別々にします。
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 サーバー・コードを実行するプロセスを別々にします。Oracle サーバー・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。
事前生成済み専用サーバー	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と、Oracle サーバー・コードを実行するプロセスを別々にします。ユーザー・プロセスは、ユーザーが要求する前に構成の一部として起動します。

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。

関連項目：

- 8-4 ページの「ユーザー・プロセス」
- 8-5 ページの「Oracle プロセス」
- 8-22 ページの「専用サーバー構成」
- 8-16 ページの「マルチスレッド・サーバー構成」
- 構成の選択肢の詳細は、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。

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経路のことです。通信経路は、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）を使用して確立されます。

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。

関連項目： 8-16 ページの「マルチスレッド・サーバー構成」

Oracle プロセス

この項では、Oracle サーバー・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。ただし、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、実行します。
- 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・バッファに読み込みます。
- 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します。

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 (QMN*n*)
- ディスパッチャ (Dmn*n*)
- サーバー (Smm)

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。

[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。

関連項目：

- 『Oracle8i Parallel Server 概要』。[図 8-2](#) には、Oracle Parallel Server は示されていません。
- 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。

データベース・ライター (DBW_n)

「データベース・ライター (DBW_n) プロセス」は、バッファの内容をデータ・ファイルに書き込みます。DBW_n 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) バッファをディスクに書き込みます。ほとんどのシステムでは1つのデータベース・ライター (DBW0) プロセス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 DBW_n 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」のマークが付きます。「コールド」バッファとは、LRU アルゴリズムに従って最近使用されたことがないバッファです。DBW_n プロセスは、ユーザー・プロセスが新規ブロックをキャッシュに読み込むために使用できるクリーンなコールド・バッファを検出できるように、使用済のコールド・バッファをディスクに書き込みます。ユーザー・プロセスによってバッファが使用済の状態になると、使用可能バッファの数が減少します。使用可能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、使用可能バッファを検出できません。ユーザー・プロセスが使用可能バッファを常に検出できるように、DBW_n はバッファ・キャッシュを管理します。

使用済のコールド・バッファをディスクに書き込むことにより、DBW_n は、使用頻度の高いバッファをメモリー内に保ちながら、使用可能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。

DBW_n 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 DBW_n 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 DBW_n 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。DBW_n プロセスは、次のような場合に使用済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スキャンしても、クリーンな再使用可能バッファが見つからなければ、DBW_n に書き込み信号が送られます。DBW_n は、他の処理中に使用済バッファをディスクに非同期に書き込みます。
- DBW_n 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・リカバリを開始する必要がある位置 (「チェックポイント」) を進めます。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バッファによって決まります。

どの場合でも、効率を向上させるために DBW_n はパッチ (マルチブロック) READ を実行します。マルチブロック WRITE 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。

関連項目：

- 7-3 ページの「データベース・バッファ・キャッシュ」
- DB_WRITER_PROCESSES と DB_BLOCK_LRU_LATCHES の設定に関するアドバイスは、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- 29-14 ページの「ファースト・スタート・チェックポイント」
- 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チューニングの方法は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

ログ・ライター (LGWR) プロセス

「ログ・ライター (LGWR) プロセス」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。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 秒ごとに書き込みます。
 - 3 分の 1 がいっぱいになったときに書き込みます。
 - DBW n プロセスが修正済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みます。

注意： DBW n が修正済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。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」メカニズムと呼びます。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リカバリ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。

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。ただし、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ため、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。したがって、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。

関連項目：

- 7-6 ページの「[REDO ログ・バッファ](#)」
- 8-15 ページの「[トレース・ファイルと ALERT ファイル](#)」
- SCN およびその使用方法の詳細は、『Oracle8i Parallel Server 管理、配置およびパフォーマンス』を参照してください。
- SCN およびその使用方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。
- 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 パフォーマンスのための設計およびチューニング』を参照してください。

チェックポイント (CKPT) プロセス

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。この書込みは、常に DBW*n* 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。

関連項目： Oracle Parallel Server 環境での CKPT の詳細は、『Oracle8i Parallel Server 管理、配置およびパフォーマンス』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 (SMON) プロセス」は、インスタンスの起動時に必要に応じてクラッシュ・リカバリを実行します。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・リカバリまたはインスタンス・リカバリ時にデッド・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リカバリします。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・リカバリを実行できます。

関連項目： SMON の詳細は、『Oracle8i Parallel Server 管理、配置およびパフォーマンス』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)」がプロセスをリカバリします。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 (RECO) プロセス

「リカバラ (RECO) プロセス」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）。

関連項目： 分散トランザクションのリカバリの詳細は、『Oracle8i 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 (ARCn) プロセス

「アーカイバ (ARCn) プロセス」は、オンライン REDO ログ・ファイルがいっぱいになるか、ALTER SYSTEM SWITCH LOGFILE 文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。

Oracle インスタンスは、最大 10 個の ARC n 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 ARC n プロセス数では作業負荷を処理できなくなると、新しい ARC n プロセスを起動します。ALERT ファイルには、LGWR が新しい ARC n プロセスを開始した時刻が記録されます。

データのバルクロード中など、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 文では、このパラメータの値を動的に変更し、ARC n 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 ARC n 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 ARC n プロセスを自動的に起動するため、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。

関連項目：

- 29-18 ページの「[データベースのアーカイブ・モード](#)」
- 8-15 ページの「[トレース・ファイルと ALERT ファイル](#)」
- オンライン REDO ログのアーカイブの詳細は、29-7 ページの「[REDO ログ](#)」を参照してください。
- 『Oracle8i バックアップおよびリカバリ・ガイド』
- ARC n プロセスの使用方法の詳細は、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。

ロック (LCK0) プロセス

Oracle Parallel Server では、「ロック (LCK0) プロセス」がインスタンス間ロックを提供します。

関連項目： このバックグラウンド・プロセスの詳細は、『Oracle8i Parallel Server 管理、配置およびパフォーマンス』を参照してください。

ジョブ・キュー (SNP n) プロセス

分散データベース構成では、最大 36 個の「ジョブ・キュー (SNP0、..., SNP9, SNPA、..., SNPZ) プロセス」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。

関連項目：

- 18-3 ページの「[Oracle Advanced Queuing](#)」
- 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 (QMNn) プロセス

「キュー・モニター (QMNn) プロセス」は、メッセージ・キューを監視する Oracle Advanced Queuing 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、SNPn 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。

関連項目： 18-3 ページの「[Oracle Advanced Queuing](#)」

ディスパッチャ (Dnnn) プロセス

「ディスパッチャ (Dnnn) 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。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すみません。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する必要があります。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・プロセスにルーティングします。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。

関連項目：

- 8-16 ページの「マルチスレッド・サーバー構成」
- 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。

共有サーバー（Snnn）プロセス

マルチスレッド・サーバー構成では、各「共有サーバー（Snnn）プロセス」は複数のクライアント要求を処理します。

関連項目： 8-19 ページの「共有サーバー・プロセス」

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カスタマ・サポート・センターに連絡してください。

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（SNPn）プロセス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チューニング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。

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-00600）、ブロック障害エラー（ORA-01578）およびデッドロック・エラー（ORA-00060）。
- 管理的な操作。たとえば、SQL 文の CREATE/ALTER/DROP DATABASE/TABLESPACE/ROLLBACK SEGMENT 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。

関連項目：

- SQL のトレース機能を使用可能にする方法については、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- 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。

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ルーティングします。

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、様々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経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）こともあるため、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、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。

関連項目：

- 8-20 ページの「マルチスレッド・サーバーの限定的運用」
- 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。

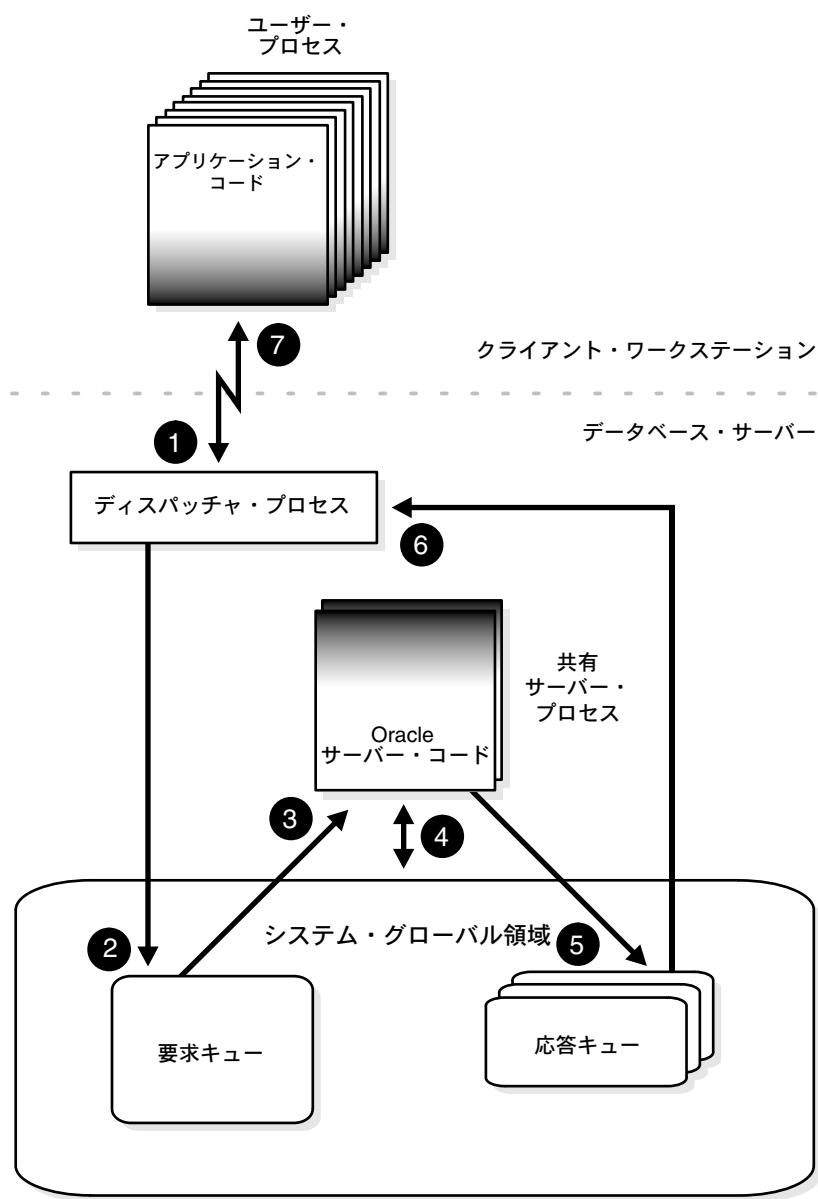
要求キューは SGA 内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。共有サーバー・プロセスは、共通の要求キューをチェックして新しい要求がないかどうかを調べ、先入れ先出し方式に基づいて新しい要求をピックアップします。1 つの共有サーバー・プロセスがキュー内の要求を 1 つピックアップし、その要求を完了するのに必要なデータベースに対するコールをすべて出します。

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ため、別の要求で使用できます。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・プロセスには専用サーバー・プロセスと同じ機能がありますが、特定のユーザー・プロセスとは対応付けられていません。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、ユーザーに関係した（すべての共有サーバー・プロセスからアクセス可能であることが必要な）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。

セッション関連の情報はすべて SGA 内に入れます。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る必要があります。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と、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。

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。

関連項目：

- 各種インスタンス構成における PGA の内容の詳細は、7-14 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。
- リソース制限とプロファイルの詳細は、第 26 章「[データベース・アクセスの制御](#)」を参照してください。

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` 文を発行して、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してデッドロックを手動で解決する必要があります。これにより、共有サーバー・プロセスが解放されるため、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・リカバリを含む、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、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）。

関連項目：

正しい接続文字列の構文については、次のマニュアルを参照してください。

- オペレーティング・システム固有の Oracle マニュアル
- 『Oracle8i Net8 管理者ガイド』

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のみを示しています。

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。）

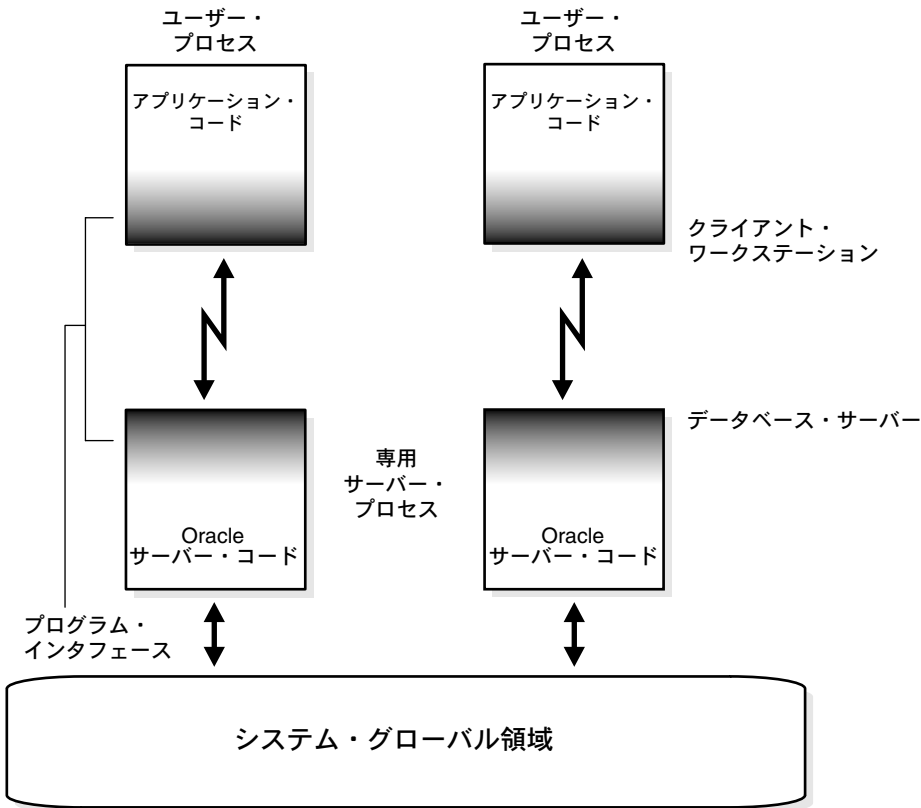
4. ユーザーが、表内の 1 行を更新するなど、1 つの SQL 文を発行します。
5. ディスパッチャ・プロセスはユーザー・プロセスの要求を要求キューに入れます。要求キューは SGA 内にあり、すべてのディスパッチャ・プロセスによって共有されています。
6. 利用可能な共有サーバー・プロセスは、共通のディスパッチャ要求キューをチェックして、キューにある次の SQL 文をピックアップします。この時点では、次の 2 つの方法で SQL 文の処理が継続されます。
 - 共有プールに類似する SQL 文のための共有 SQL 領域がある場合、サーバー・プロセスはクライアントの SQL 文を実行するために既存の共有 SQL 領域を使用します。
 - 共有プールに類似する 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の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・プロセスに完了済の要求を送り返します。

関連項目： 共有 SQL 領域の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

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ため、「専用サーバー・プロセス」（または「シャドウ・プロセス」）と呼ばれます。

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ります）。

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 サーバー・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します。
- 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）を提供します。
- 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。

関連項目：

通信リンクの詳細は、次のマニュアルを参照してください。

- オペレーティング・システム固有の Oracle マニュアル
- 『Oracle8i Net8 管理者ガイド』

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。
 - 共有プールに類似する 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。
 - 共有プールに類似する 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。

関連項目： 共有 SQL 領域の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

事前生成済み専用プロセス

「事前生成済みプロセス」は、アプリケーションが要求する前に起動されるユーザー・プロセスです。事前生成済みプロセスにより、専用サーバーとの接続時間を改善できます。これは、大量の負荷が発生するシステムでマルチスレッド・サーバーを使用しない場合に特に効果的です。事前生成済みプロセスが使用可能になっている場合、リスナーは接続要求を受け取った後、待ち時間なしで既存のプロセスに接続を渡すことができます。ユーザーが接続されると、リスナーは次の接続用に次のシャドウ・プロセスを作成します。

事前生成済みユーザー・プロセスは、リソースやシステムへのアクセスを制御する場合にも役立ちます。事前生成できるシャドウ・プロセスの数には、上限を設定できます。この上限に達した後、すべての新規接続は専用となります。

関連項目： 事前生成済み専用プロセスの構成については、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

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ぎます。
- 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、エラーのトラップと通知を行います。
- 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・データ型に対してデータを変換し解釈します。

「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（SLLIB）
- 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と、他のドライバを使用できます。プロセスが異なる場合は、異なるドライバを使用できます。1 つのプロセスでも、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。

関連項目：

- ドライバの選択、インストールおよび追加方法の詳細は、システムのインストールおよび構成ガイドを参照してください。
- Oracle へのアクセス中にドライバを実行時に選択する方法については、システムの Net8 マニュアルを参照してください。
- 『Oracle8i Net8 管理者ガイド』

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。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。

関連項目： 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

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、様々なユーザー・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](#)
- [リソース・コンシューマ・グループとリソース・プラン](#)
- [リソース割当て方法](#)
- [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・マネージャの機能は、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。リソース管理の改善により、アプリケーションのパフォーマンスと可用性が向上します。データベース管理者は、データベース・リソース・マネージャを使用して次の操作を実行できます。

- 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します。
- 様々なユーザーとアプリケーションに、比率を指定して CPU タイムを割り当てて、使用可能な処理リソースを分配します。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します。
- 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します。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。

リソース・コンシューマ・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・セッションのグループ
リソース・プラン	リソースをコンシューマ・グループ間で割り当てる手段
リソース割当て方法	特定のリソースを割り当てるときに使用する方針 リソース割当て方法は、プランとコンシューマ・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">■ コンシューマ・グループまたはサブプランをリソース・プランに割り当てます。■ 各リソース割当て方法のパラメータを指定し、プランに含まれるリソース・コンシューマ・グループ間でリソースを割り当てます。

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。

リソース・コンシューマ・グループとリソース・プラン

リソース・コンシューマ・グループとリソース・プランを使用して、様々なユーザー間で処理リソースを割り当てる方法を指定できます。現在、リソース・コンシューマ・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、リソース・コンシューマ・グループとリソース・プランについて説明します。

リソース・コンシューマ・グループ

リソース使用を制御するために、ユーザー・セッションをリソース・コンシューマ・グループに割り当てることができます。リソース・コンシューマ・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、セッション間で CPU を割り当てるためのリソース割当て方法も指定します。現在、リソース・コンシューマ・グループ内のセッション間で CPU の割当てに使用されるリソース割当て方法は、「ラウンドロビン」方式のみです。

ユーザーには、別のコンシューマ・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れば、PL/SQL プロシージャを使用すると、特定のセッションのリソース・コンシューマ・グループを切り替えることができます。

次の 2 つは、変更も削除もできない特殊なコンシューマ・グループです。

- OTHER_GROUPS

アクティブなプラン・スキーマに含まれていないコンシューマ・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 には、アクティブなプランのスキーマ内でリソース・プラン・ダイレクティブを指定する必要があります。

- DEFAULT_CONSUMER_GROUP

初期のコンシューマ・グループに明示的に割り当てられていない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。

次のように、Oracle 提供の 2 つのコンシューマ・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 の初期のコンシューマ・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 9-2 「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

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、各リソース・コンシューマ・グループに割り当て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、セッションはリソース・コンシューマ・グループに属し、このリソース・コンシューマ・グループをリソース・プランで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。

- リソース・コンシューマ・グループや他のリソース・プランをグループ化します。
- グループ化したリソース・コンシューマ・グループやプランの間でリソースを割り当てます。
- リソース割当てを指定します。（現在、リソース・コンシューマ・グループ内のセッション間で CPU の割当てに使用されるリソース割当て方法は、「強調」方式のみです。並列度を制限するためのリソース割当て方法は、「絶対」方式のみです。この2つの方式については後述します。）

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リソース・コンシューマ・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、一度にアクティブにできるのは1プランのみです。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にすると、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、様々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。

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 のリソース割当て方法
- 並列度制限方法
- 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
APPD	3	EMPHASIS	PARALLEL_DEGREE_LIMIT_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。

関連項目： リソース・プランとリソース・コンシューマ・グループに対
 応付けられたデータ・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。

リソース割当て方法

「リソース割当て方法」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リソース・コンシューマ・グループやリソース・プランに割り当てるときに使用される方法や方針が決まります。

Oracle では、リソース・コンシューマ・グループまたはリソース・プランへのリソース割当て方法として、表 9-1 に示す方法が用意されています。

表 9-1 リソース割当て方法

方法	リソース	リソース・レシピエント
ラウンドロビン方式	セッションへの CPU 割当て	リソース・コンシューマ・グループ
強調方式	コンシューマ・グループへの CPU 割当て	リソース・プラン
絶対方式	並列度制限	リソース・プラン

リソース・プランへの CPU 割当て：強調方式

強調 CPU 割当て方法では、リソース・プランに含まれる各コンシューマ・グループのセッションに与える強調度が決定されます。CPU の使用順位は、レベル 1 を最上位としてレベル 1 ～ 8 を使用して割り当てます。各レベルで CPU を各コンシューマ・グループに割り当てる方法は、パーセンテージで指定します。

強調リソース割当て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% 以外のパーセンテージが指定されているリソース・コンシューマ・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ます。
- 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ます。リソース・コンシューマ・グループに対して指定する CPU のパーセンテージは、そのコンシューマ・グループが特定のレベルで使用可能な最大値となります。特定レベルのすべてのリソース・コンシューマ・グループに実行され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 リソースは次のレベルへと落ちます。コンシューマ・グループが割り当てられたリソースを使用しなければ、そのリソースは同じレベルの他のコンシューマ・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 タイムはリサイクルされます。つまり、指定された CPU タイム（強調度による）を即時に必要とするコンシューマ・グループがなければ、各コンシューマ・グループにはレベル 1 から順番に割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 / コンシューマ・グループについてデフォルトが 0% になります。

強調方式の例については後述します。

強調リソース割当て方法の長所は、次のとおりです。

- 強調度を設定すると、CPU をオンライン / オフラインの間で切り替えたり、CPU の強調度を変更せずにサーバーを追加および削除できます。
- 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ため、サーバーが少数の場合にもきめ細かく制御できます。
- 強調度を設定すると、優先順位に関連したリソース不足の問題を回避できます。ユーザーは優先順位順に実行するのではなく、各自のリソース・コンシューマ・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ムをシミュレーションできます。

リソース・プランの並列度制限：絶対方式

「並列度制限」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、リソース・コンシューマ・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。現在、並列度を制限するためのリソース割当て方法は、「絶対」方式のみです。「絶対」とは、操作に割当て可能なプロセス数を数値（パーセンテージや分数ではなく）で指定することを意味します。

複数のプラン・ダイレクティブが同じサブプラン / コンシューマ・グループを参照している場合、そのサブプラン / コンシューマ・グループの並列度制限は、与えられるすべての値の中の「最小」値になります。

関連項目： パラレル実行については、[第 23 章「SQL 文のパラレル実行」](#)を参照してください。

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。

- コンシューマ・グループまたはサブプランをリソース・プランに割り当てます。
- 各リソース割当て方法のパラメータを指定して、プランに含まれるリソース・コンシューマ・グループ間でリソースを割り当てます。

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、リソース・コンシューマ・グループ、リソース・プラン、リソース割当て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。

リソース・コンシューマ・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、リソース・コンシューマ・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。

リソース・プラン `SYSTEM_PLAN` が用意されています。その定義を[表 9-2](#) に示します。

表 9-2 `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
<code>SYS_GROUP</code>	100%	0%	0%
<code>OTHER_GROUPS</code>	0%	100%	0%
<code>LOW_GROUP</code>	0%	0%	100%

ユーザー `SYS` および `SYSTEM` のコンシューマ・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 9-3](#) と [表 9-4](#) は、`BUGDB` と `MAILDB` のサンプル・リソース・プランを示しています。

表 9-3 BUGDB サンプル・リソース・プラン

エントリ	レベル 1	レベル 2	レベル 3
Online リソース・コンシューマ・グループ	80%	0%	0%
Batch リソース・コンシューマ・グループ	20%	0%	0%
Bug_Maintenance リソース・コンシューマ・グループ	0%	100%	0%
OTHER_GROUPS	0%	0%	100%

表 9-4 MAILDB サンプル・リソース・プラン

エントリ	レベル 1	レベル 2	レベル 3
Mailusers リソース・コンシューマ・グループ	0%	80%	0%
Postman リソース・コンシューマ・グループ	40%	0%	0%
Mail_Maintenance リソース・コンシューマ・グループ	0%	20%	0%
OTHER_GROUPS	0%	0%	100%

BUGDB および MAILDB サンプル・リソース・プランのデータは、強調 CPU リソース割当て方法によるものです。この場合、各リソース・コンシューマ・グループに強調度を割り当てて、様々なグループのセッションの強調度を指定できます。

MAILDB プランでは、Postman リソース・コンシューマ・グループは 40% の CPU が保証されます。残りの CPU（Postman が割当て分の 40% をすべて必要としない場合は 60% 以上）は、Mailusers および Mail_Maintenance リソース・コンシューマ・グループの間で 80 : 20 の割合で分割されます。Mailusers と Mail_Maintenance が CPU タイムの残りの 60% を使い果たさなければ、残りの CPU タイムは OTHER_GROUPS に割り当てられます。OTHER_GROUPS が使用する機会を与えられても CPU タイムが残っている場合は、9-5 ページに示すルールに従って、すべてのコンシューマ・グループに残りの CPU タイムを使用する機会が与えられます。

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 9-5 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。

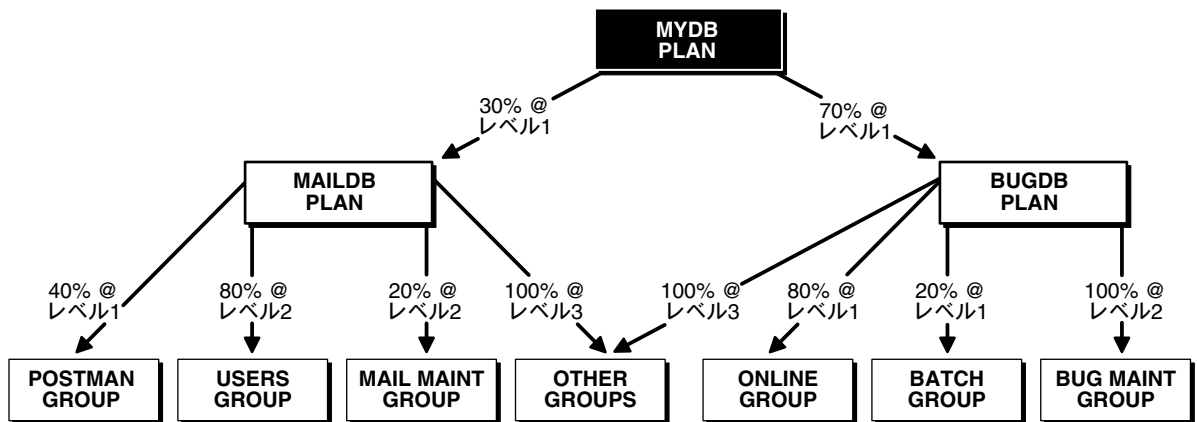
表 9-5 MYDB リソース・プラン、CPU プラン・ダイレクティブ

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、すべてのリソース・コンシューマ・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 リソース・コンシューマ・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 リソース・コンシューマ・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。リソース・コンシューマ・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リソース・コンシューマ・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 にならないければ、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・プラン・ダイレクティブの使用方法

表 9-6 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、OTHER_GROUPS 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シリアルで実行する必要があります。

表 9-6 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
OTHER_GROUPS	4

まとめ

表 9-7 では、表 9-3 の BUGDB プラン情報を使用して、デフォルトのリソース割当て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。

表 9-7 リソース・プラン・ダイレクティブ

サブプラン/グループ	レベル 1 ~ 8 の CPU リソース・プラン・ダイレクティブ (明示的な指定が必要なのは、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
	CPU_Level 1	CPU_Level 2	CPU_Level 3	...	parallel_degree_limit_1
ONLINE グループ	80%	0%	0%	0%	0
BATCH グループ	20%	0%	0%	0%	4
BUG_MT グループ	0%	100%	0%	0%	4
OTHER_GROUPS	0%	0%	100%	0%	4

データベース・リソース・マネージャの使用法

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、リソース・コンシューマ・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、コンシューマ・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トップレベルの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、このトップレベルの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コンシューマ・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いと、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、このパラメータを ALTER SYSTEM 文で動的に設定して、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できます。

関連項目：

これらの PL/SQL パッケージの使用法は、次のマニュアルを参照してください。

- 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』
- 『Oracle8i 管理者ガイド』

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「組み込みデータ型」](#)
- [第 13 章「ユーザー定義データ型」](#)
- [第 14 章「オブジェクト・ビュー」](#)

スキーマ・オブジェクト

この章では、ユーザー・スキーマに含まれる様々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](#)
- [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](#)

関連項目：

追加のスキーマ・オブジェクトの詳細は、次を参照してください。

- [30-9 ページの「データベース・リンク」](#)
- [17-2 ページの「ストアド・プロシージャとファンクション」](#)
- [第 17 章「プロシージャとパッケージ」](#)
- [第 19 章「トリガー」](#)

スキーマ・オブジェクトの概要

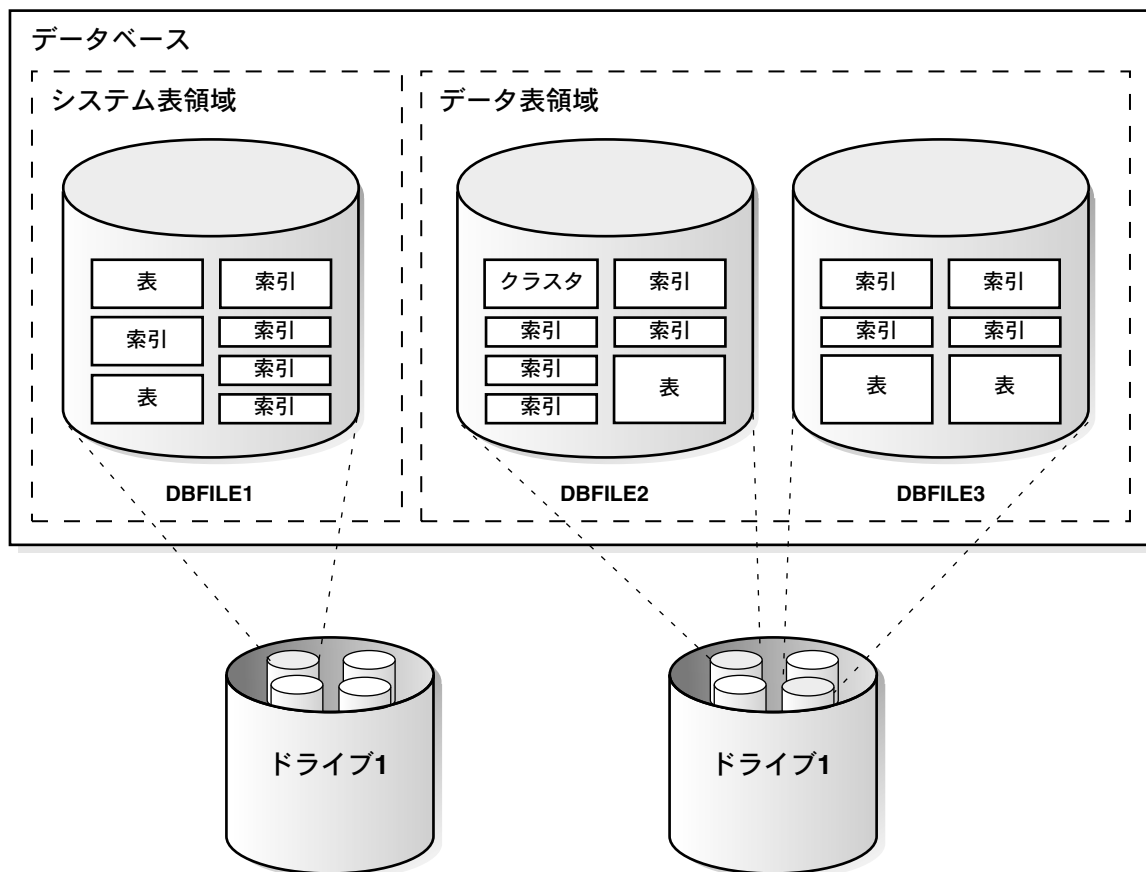
「スキーマ」は、各データベース・ユーザーに対応付けられます。スキーマは、スキーマ・オブジェクトの集合です。スキーマ・オブジェクトには、表、ビュー、順序、シノニム、索引、クラスタ、データベース・リンク、スナップショット、プロシージャ、ファンクションおよびパッケージなどが含まれます。

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。

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域の基本単位です。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」を割り当てます。幅は、DATE の場合のようにデータ型によって事前に決定されている場合があります。列が NUMBER データ型の場合は、幅のかわりに「精度」と「スケール」を定義します。行は、1 つのレコードに対応する列情報の集まりです。

表の各列に対してルールを指定できます。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。

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。

関連項目：

- Oracle データ型の詳細は、第 12 章「組み込みデータ型」を参照してください。
- 整合性制約の詳細は、第 25 章「データの整合性」を参照してください。

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。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域パラメータを設定します。これにより、データ・セグメントのエクステンツ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 パラメータを設定します。これにより、データ・セグメントのエクステンツを構成するデータ・ブロック内の空き領域の使用を制御できます。

クラスタ化表のデータは、表領域のデータ・セグメントではなく、クラスタ用に作成されたデータ・セグメントに格納されます。クラスタ化表を作成または変更するときには、記憶域パラメータは指定できません。クラスタに対して設定された記憶域パラメータが、常にそのクラスタ内のすべての表の記憶域を制御します。

クラスタ化され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。

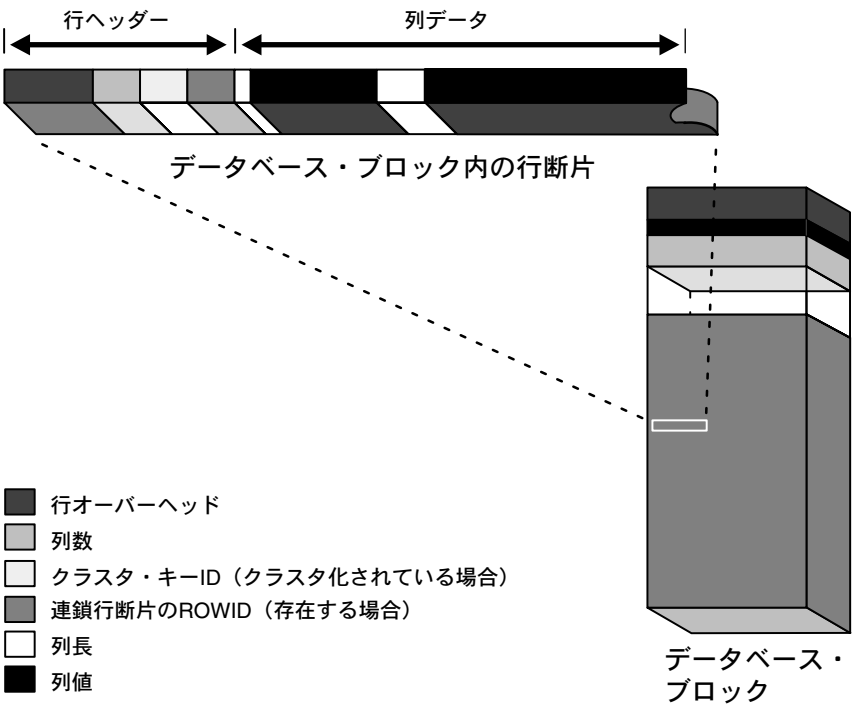
関連項目： 26-14 ページの「[ユーザー表領域の設定と割当て制限](#)」

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、各行に 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。

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。

図 10-3 行断片の形式



「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・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で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。

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。

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せん。

関連項目：

- 4-9 ページの「[行連鎖と移行](#)」
- 10-8 ページの「[NULL](#)」
- 4-4 ページの「[行ディレクトリ](#)」
- 10-52 ページの「[クラスタ化されたデータ・ブロックの形式](#)」

削除された列または未使用の列

ALTER TABLE 文の DROP COLUMN 句を使用すると、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 文の SET UNUSED 句を使用し、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。

関連項目：

- 列削除の方法と未使用のマークを設定する方法については、『Oracle8i 管理者ガイド』を参照してください。
- ALTER TABLE 文の詳細は、『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 ユ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。クラスタ化表の場合、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であるため、SELECT、UPDATE および DELETE などの SQL 文で ROWID を参照すると便利です。

関連項目：

- 10-49 ページの「[クラスタ](#)」
- 12-15 ページの「[物理 ROWID](#)」

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。

行の末尾にある NULL には、記憶域は不要です。これは、新しい行のヘッダーが、前行の残りの列が 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。

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。

関連項目：

- 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。
- 10-25 ページの「[索引と NULL](#)」
- 10-38 ページの「[ビットマップ索引と NULL](#)」

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。

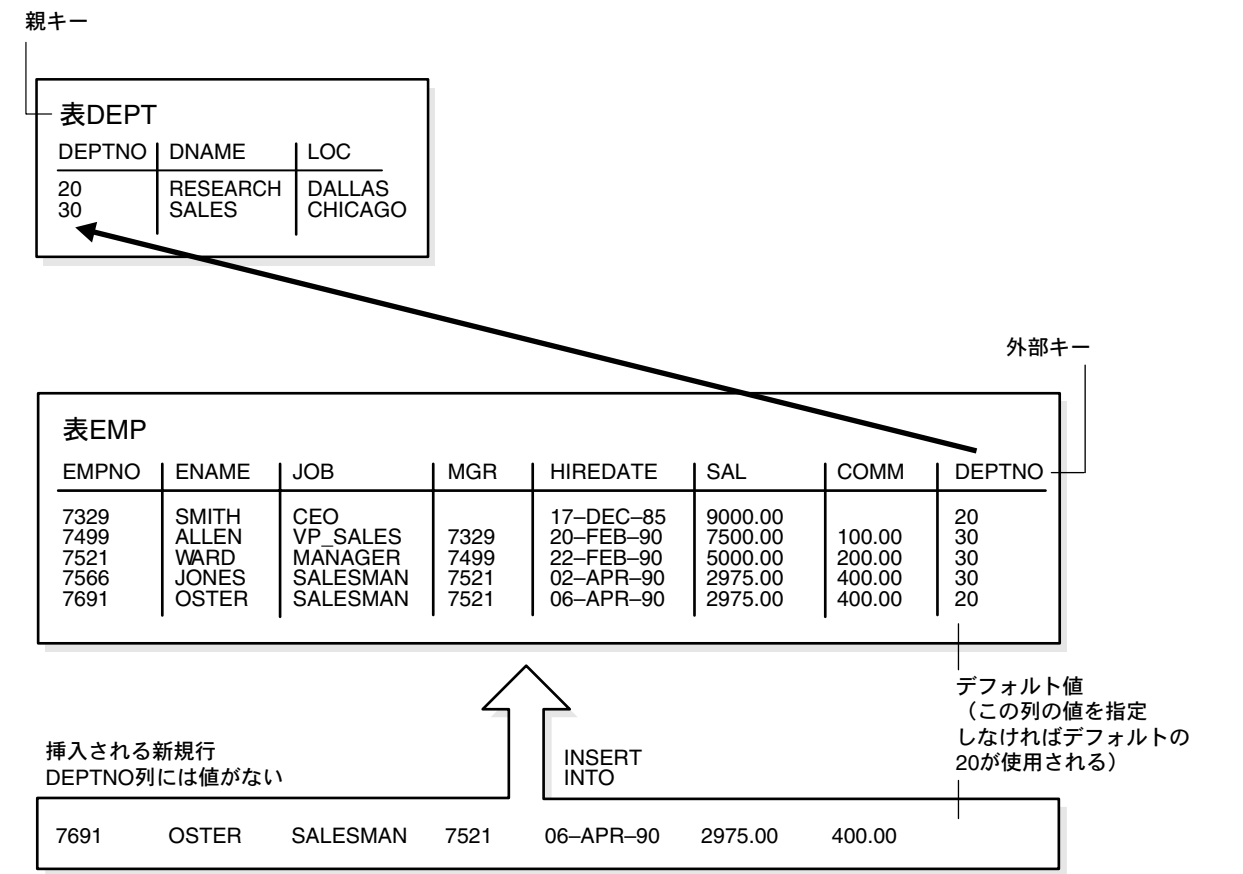
列に対してデフォルト値が明示的に定義されていない場合、その列のデフォルトは暗黙的に NULL に設定されます。

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。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 が挿入されます。デフォルト値の挿入後に、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。

関連項目： 整合性制約の詳細は、[第 25 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



NESTED TABLE

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、NESTED TABLE 列に対応付けた「格納表」を使用して、親表の行の外に NESTED TABLE のデータを格納します。親行には、NESTED TABLE のインスタンスと対応付けられた一意の集合識別子が入れられます。

関連項目：

- 13-11 ページの「NESTED TABLE」
- 『Oracle8i アプリケーション開発者ガイド 基礎編』

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 文では、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のみです。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 文は無効です。

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 文を使用すると、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 句を使用しても、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。永続表とは異なり、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 文（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のみです。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。

関連項目： 4-15 ページの「一時セグメント内のエクステンツ」

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ります。

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ます。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「ストアド・クエリ」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、ベース表 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、ベース表の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

ベース
表

EMP							
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-88	300.00	800.00	20
7499	ALLEN	SALESMAN	7698	20-FEB-88	300.00	1600.00	30
7521	WARD	SALESMAN	7698	22-FEB-88	5.00	1250.00	30
7566	JONES	MANAGER	7839	02-APR-88		2975.00	20

ビュー

STAFF				
EMPNO	ENAME	JOB	MGR	DEPTNO
7329	SMITH	CLERK	7902	20
7499	ALLEN	SALESMAN	7698	30
7521	WARD	SALESMAN	7698	30
7566	JONES	MANAGER	7839	20

ビューは表から導出されるため、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、そのビューのベース表のデータに影響し、ベース表の整合性制約とトリガーの対象になります。

関連項目：『Oracle8i SQL リファレンス』

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ベース表に対しては定義できます。

ビューの記憶域

表とは異なり、ビューには記憶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導出を行う問合せによって定義されます。これらの表は「ベース表」と呼ばれます。ベース表としては、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（ストアド・クエリ）以外には記憶域が必要ありません。

ビューの使用法

ビューは、ベース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。

- 事前に定義された表の行または列の集合のみにアクセスを限定し、表のセキュリティ・レベルを強化します。

たとえば、図 10-5 は、STAFF ビューがベース表 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します。

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はわかりません。

- ユーザー側の文を単純化します。

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。

- ベース表とは異なる見方でデータを提示します。

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。

- ベース表の定義の変更からアプリケーションを分離します。

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します。

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。

- 複雑な問合せを保存します。

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くと、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す。

関連項目： GROUP BY および UNION ビューの詳細は、『Oracle8i SQL リファレンス』を参照してください。

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。SQL 文でビューを参照すると、Oracle では次の操作が実行されます。

1. ビューを参照する文が、そのビューを定義している問合せとマージされます。
2. マージ済みの文が共有 SQL 領域内で解析されます。
3. 文が実行されます。

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 領域に同様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して上書きできます。

関連項目： 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および基礎となるビューの問合せとマージします。マージされた問合せは、ビューを参照せずに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、参照されるベース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。

関連項目： 問合せの最適化の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』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、ビューのベース表が削除された後で再作成された場合、Oracle は、新しいベース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。

関連項目： データベースの依存性の詳細は、[第 20 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 つ以上のベース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

ルール	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 つのみ変更できます。
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る必要があります。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ます。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ないでください。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。

関連項目： 19-13 ページの「[INSTEAD OF トリガー](#)」

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（NESTED TABLE 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。

関連項目：

- [第 14 章「オブジェクト・ビュー」](#)
- 『Oracle8i アプリケーション開発者ガイド 基礎編』

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;
```

関連項目： 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、スナップショットと呼ばれることもあり、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・ウェアハウス、意思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、様々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、集計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ます。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ため、「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、可能かつ必要な場合を自動的に認識して、マテリアライズド・ビューを使用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、オプティマイザ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消方法に従って同期化するために使用されます。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・サイトからのアクセスを必要とするデータへの、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リフレッシュし、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ます。

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。

- 記憶領域を消費します。
- 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。

- 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ます。
- 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。

索引とは異なり、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。

関連項目：

- 10-23 ページの「[索引](#)」
- [第 11 章「パーティション表とパーティション索引」](#)

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・ビューがマスター表の変更後にリフレッシュされ、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」とも呼ばれます。

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。

関連項目：

- ウェアハウス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・ログの詳細は、『Oracle8i データ・ウェアハウス』を参照してください。
- 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）の詳細は、『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」）でも複数の表のもの（「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 句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。

関連項目： ウェアハウス環境におけるディメンションの使用方法については、『Oracle8i データ・ウェアハウス』を参照してください。

シーケンス・ジェネレータ

シーケンス・ジェネレータは、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・ジェネレータによってシリアル化（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。このシリアル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、次のように一般的な情報を指定します。

- 順序の名前
- 昇順または降順
- 数値の増分値
- 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

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、カレント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、固有のカレント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・ジェネレータを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使用するように作成することもできます。

関連項目： 順序の使用がパフォーマンスに与える影響については、『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します。
- 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します。
- データベース・ユーザー側の SQL 文を単純化します。

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。ベース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パブリック・シノニムを作成する場合があります。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 という表があります。
- SALES_DATA 表に対する SELECT 権限が PUBLIC に付与されています。

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ると、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。

- B-tree 索引
- B-tree クラスタ索引
- ハッシュ・クラスタ索引
- 逆キー索引
- ビットマップ索引

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・パスにすぎないためです。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、ベース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。ただし、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。

オプティマイザは、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、表の定義に対応付ける必要があるためです。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して、UNIQUE 整合性制約を規定します。

コンポジット索引

「コンポジット索引」（「連結索引」とも呼ばれる）は、表内の複数の列に対して作成される索引です。コンポジット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、コンポジット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 句でコンポジット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、コンポジット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。

図 10-6 は、VENDOR_ID および PART_NO 列にコンポジット索引を設定した VENDOR_PARTS 表を示しています。

図 10-6 コンポジット索引の例

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)

通常のコンポジット索引は最大 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。

関連項目：『Oracle8i パフォーマンスのための設計およびチューニング』

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、データベースのビジネス・ルールを規定する Oracle のもう 1 つの機能です。

いくつかのタイプの整合性制約は、索引によって規定されるため、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。

関連項目： 第 25 章「データの整合性」

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や、クラスタ・キーの列値が NULL の場合は例外です。

関連項目： 10-38 ページの「[ビットマップ索引と NULL](#)」

ファンクション索引

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。ファンクション索引は、B-tree またはビットマップ索引として作成できます。

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。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 NESTED TABLE の列には、ファンクション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 NESTED TABLE が含まれる場合も作成できません。

関連項目：

- 「[ビットマップ索引](#)」
- 『Oracle8i パフォーマンスのための設計およびチューニング』

ファンクション索引の使用方法

ファンクション索引は、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。

関連項目： NLS ソート索引の詳細は、『Oracle8i NLS ガイド』を参照してください。

ファンクション索引による最適化

オプティマイザ用に、ファンクション索引の統計を収集する必要があります。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、ファンクション索引は使用されません。

コストベース最適化では、WHERE 句の式による問合せに、ファンクション索引に対する索引レンジ・スキャンを使用できます。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、オプティマイザは索引レンジ・スキャンを使用できます。レンジ・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プティマイザで見積もることができます。

オプティマイザは、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。

関連項目：

- 統計収集の詳細は、21-9 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。
- オプティマイザの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

ファンクション索引の依存性

ファンクション索引は、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。

ファンクション索引の定義には、PL/SQL 関数 DETERMINISTIC を使用する必要があります。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、ファンクション索引に DISABLED マークが設定されます。

関連項目：

- PL/SQL 関数 DETERMINISTIC の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- ファンクション索引の依存性と権限の詳細は、20-7 ページの「[ファンクション索引の依存性](#)」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域パラメータを設定すると制御できます。
- 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と制御できます。

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パラレルに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。

関連項目： 26-14 ページの「[ユーザー表領域の設定と割当て制限](#)」

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。

関連項目： 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。

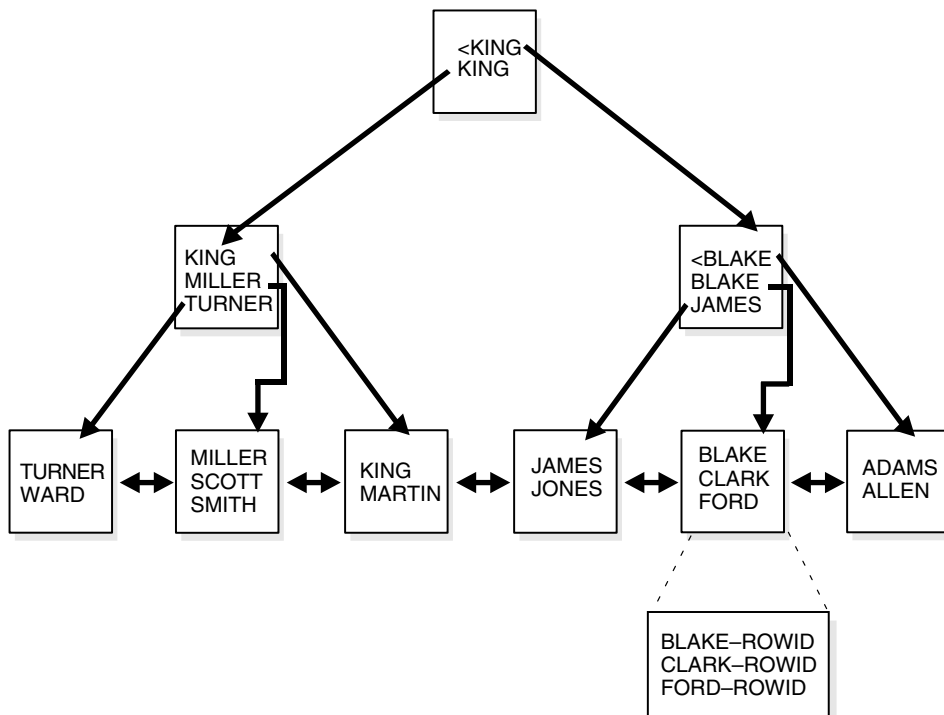
Oracle は CREATE INDEX 文のキーワードとして ASC、DESC、COMPRESS および NOCOMPRESS を受け入れますが、これらのキーワードは、後方圧縮を使用してリーフ・ノードではなくブランチ・ノードに格納されている索引データには作用しません。

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-tree 索引を使用します。B-tree 索引の理論は、このマニュアルでは説明しません。

図 10-7 に、B-tree 索引の構造を示します。

図 10-7 B-tree 索引の内部構造



B-tree 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-tree 構造の利点

B-tree 構造の利点は、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ります。
- B-tree 索引は自動的にバランスが保たれます。
- B-tree のすべてのブロックの平均 4 分の 3 がいっぱいになります。
- 完全一致や範囲検索など、広範囲な問合せに対して、B-tree は優れた検索パフォーマンスを提供します。
- 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ます。
- B-tree 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ません。

関連項目： B-tree 索引の詳細は、コンピュータ・サイエンスのテキストを参照してください。

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。

接頭辞エントリと接尾辞エントリ

キー圧縮により、索引キーが接頭辞エントリ（グループ化要素）と接尾辞エントリ（一意要素）に分割されます。圧縮するために、接頭辞エントリが索引ブロック内の接尾辞エントリ間で共有されます。圧縮されるのは、B-tree 索引のリーフ・ブロックのキーのみです。プランチ・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。接尾辞エントリは、索引行の圧縮版を形成します。各接尾辞エントリは、同じ索引ブロックに格納されている接頭辞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。

接頭辞エントリは、それと等しい値を持つ接頭辞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。接頭辞エントリは、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接尾辞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、各索引ブロックに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 タイムが増大することがあります。また、各接頭辞エントリには、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、次のように様々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、重複行が分割されます。キー圧縮を使用すると、重複キーは ROWID を除き接頭辞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 のみからなる接尾辞エントリです。
- 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ます。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` 値が接頭辞エントリとして 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` 接頭辞エントリを参照する接尾辞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。

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。

関連項目： 10-39 ページの「索引構成表」

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レンジ・スキャン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）スキャンしか実行できません。

状況によっては、逆キー索引を使用する方が、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、電子メール・アプリケーションでメール・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ため、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する必要があります。

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（再作成）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作成すると、逆キー索引が再作成されて生成されます。標準の索引を逆キー索引として再作成することはできません。その場合は、CREATE 文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使用可能です。

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る場合、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 に変換するため、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い場合、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。

- 多くの種類の非定型問合せの応答時間が短縮されます。
- 他の方式の索引と比べて使用領域が実質的に縮小されます。
- 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します。
- パラレル DML とロードを効果的に実行できます。

従来の B-tree 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ためです。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、数多くの並列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ように決めたとします。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-tree 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-tree 索引またはフル・テーブル・スキャンなど）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、ローカル索引であることが必要です。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。

関連項目： 11-28 ページの「[索引のパーティション化](#)」

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用し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 個以上含まれている場合、その列はビットマップ索引の候補です。値の繰返しが少なく、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場合は、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-tree 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合せることが多い場合は、特に効果的です。

B-tree 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-tree 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-tree 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と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、フル・テーブル・スキャンを行う必要はないため、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です。最初の 2 つには値が 3 つ、3 番目には 2 つ、4 番目には 4 つしかないため、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であるため、ビットマップ索引は作成しません。この場合は、一意の B-tree 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" であるためです。REGION の値が "east" の行は他にないため、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て、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		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		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 による問合せなど、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ため、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あることが必要です。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、非パーティション表でのみサポートされます。

関連項目：

- 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。
- ビットマップ索引の使用の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

索引構成表

「索引構成表」が通常の表と異なるのは、表のデータが、対応付けられた索引に保持されるためです。行の追加、更新または削除など、表データを変更すると、索引が更新されます。

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に似ています。ただし、データベース・システムは、表本体と B-tree 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-tree 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-tree 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-tree 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。

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-tree 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 が行を一意に識別します。任意で、主キーを指定できます。	主キーが行を一意に識別します。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 次索引を作成できます。	ROWID 疑似列の論理 ROWID で 2 次索引を作成できます。
ROWID に基づいてアクセスします。	主キーに基づいてアクセスします。
順次スキャンはすべての行を戻します。	全索引スキャンは主キーの順序ですべての行を戻します。
UNIQUE 制約とトリガーを定義できます。	UNIQUE 制約は定義できませんがトリガーは定義できます。
他の表とともにクラスタに格納できます。	クラスタには格納できません。
LONG データ型の列と LOB データ型の列を格納できます。	LOB 列は格納できますが、LONG 列は格納できません。
分散とレプリケーションがサポートされます。	分散とレプリケーションがサポートされません。

関連項目：

- 10-42 ページの「[索引構成表の 2 次索引](#)」
- 索引構成表の作成とメンテナンスの詳細は、『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ため、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と、通常の表の索引が必要とする物理 ROWID 用の追加の記憶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。

関連項目： 10-31 ページの「[キー圧縮](#)」

行オーバーフロー領域付きの索引構成表

B-tree 索引エントリは <key, ROWID> のペア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-tree 索引エントリは <key, non_key_column_values> のペア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いっぱいになり、高い密度で格納されるという B-tree 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。

関連項目： 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。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ます。
- 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ます。
- 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 によって間違っただデータ・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ます。

関連項目： 12-19 ページの「[論理 ROWID](#)」

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。

索引構成表の再作成

索引構成表を再作成し、増分更新によって生じる断片化を削減できます。索引構成表を再作成するには、ALTER TABLE 文で MOVE 句を指定します。

MOVE 句を指定すると、索引構成表の主キー索引の B-tree が再作成されます。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・セグメントは再作成されません。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作成されません。

索引構成表の平行作成

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、それを平行にロードできます。この文は、SQL*Loader を使用する平行・バルク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。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

関連項目：

- 11-13 ページの「[レンジ・パーティション化](#)」
- 11-28 ページの「[索引のパーティション化](#)」

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、逆索引をスキャン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ます。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、無駄な領域が生じます。逆索引は巨大になることがあるため、記憶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と、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、プール問合せ)、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用したとしても、元になる表にアクセスする必要があります。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、記憶域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、前述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・データの取出しには、索引をスキャンして、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ます。
- 索引には、非キー列値のみがキーと一緒に格納されます。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ためです。

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ため、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。また、拡張可能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。

関連項目： 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](#)」を参照してください。

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。

索引デザイナーが実装する必要があるルーチンは、すべて Oracle Data Cartridge Interface (ODCIIndex) で指定されます。各ルーチンは、型のメソッドとして実装できます。

関連項目： 13-3 ページの「[オブジェクト型](#)」

索引定義ルーチン

索引定義ルーチンの機能は、次のとおりです。

- 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ます。
- 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。
- DROP INDEX 文で索引タイプが削除されると、索引情報を削除します。
- ベース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、ベース表の行が挿入、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、組込み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ベース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実装します。

索引スキャンは、次の 3 つのルーチンを通じて指定されます。

- **istart** – データ構造を初期化します。
- **ifetch** – 述語を満たす行をフェッチします。
- **iclose** – 述語を満たすすべての行が戻された後に、カーソルをクローズします。

関連項目： ドメイン索引のスキャンに使用できるオペレータの詳細は、10-48 ページの「[ユーザー定義オペレータ](#)」を参照してください。

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、索引を作成するためです。

現在サポートされているのは、単一系列のドメイン索引のみです。単一系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のみです。この点では、組込み B-tree 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-tree 索引の場合は、USER_INDEXES ビューを問い合わせ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、索引デザイナー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。

- 索引デザイナーは、メタデータを含む 1 つ以上の表を定義できます。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、索引デザイナー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ます。ビュー定義は、索引デザイナー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、組込み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、結果を返します。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組込み演算子と同様に 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。組込み演算子と同様に、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、結果を返します。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、組込み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 DML 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 文または副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` を返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ド (存在する場合) を指定します。オペレータの「バインド」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、バインドでは一意の「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ド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つのバインドと、`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ド宣言の一部として指定されますが、バインド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・バインド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、いくつかの組込み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-tree` 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、行のフィルタリング、選択およびフェッチなど、さらに処理を実行します。

関連項目： 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 データ・カートリッジ開発者ガイド』を参照してください。

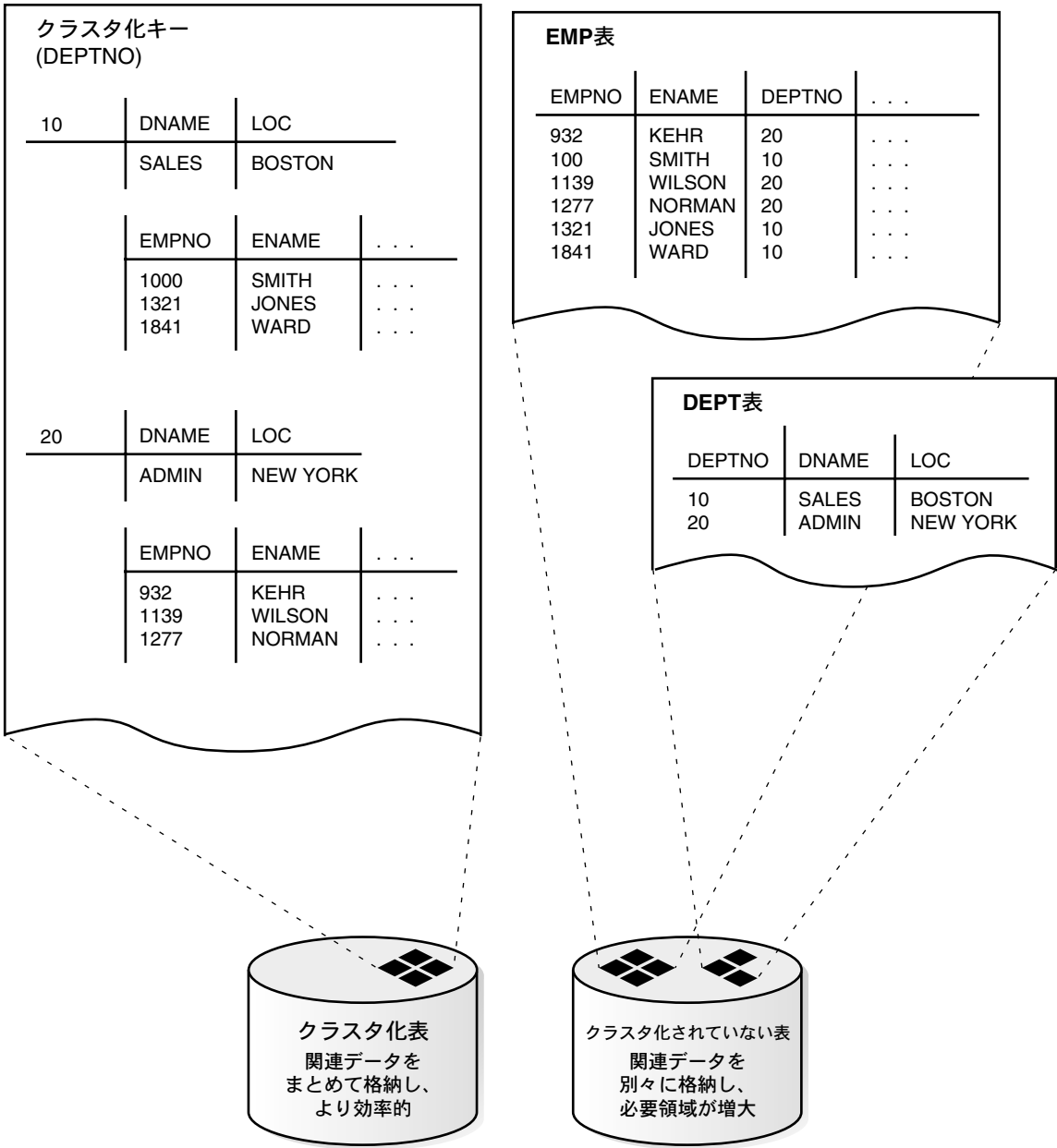
クラスタ

「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、`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」に、`EMP` 表と `DEPT` 表をクラスタ化した場合の結果を示します。

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と、次のような利点があります。

- クラスタ化表を結合する場合のディスク I/O が減少します。
- クラスタ化表を結合場合のアクセス時間が改善されます。
- 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ます。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域は、クラスタ化していない表で必要な記憶域よりも少なくなります。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、表をスキャン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。表データの結合に使用される列に基づいて表をクラスタ化すると、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することになり、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。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。

関連項目： 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 パフォーマンスのための設計およびチューニング』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 文の 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てもありません。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。

クラスタの作成時にクラスタ・キーの一部として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。ただし、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する必要があります。「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。索引内で横断を必要とする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ちます。
- 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します。
- 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 エントリを収録します。
- 表索引がなくても表データにアクセスできますが、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ません。

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に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、索引クラスタのキーと同様に、単一の列またはコンポジット・キー（複数列のキー）にすることができます。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、次のように最低 2 回の I/O を実行する必要があります。

- 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
- 表またはクラスタ内の行を読み書きするためにさらに 1 回の I/O

それとは対照的に、Oracle では、ハッシュ関数を使用してハッシュ・クラスタ内の行が位置付けられます。I/O は不要です。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

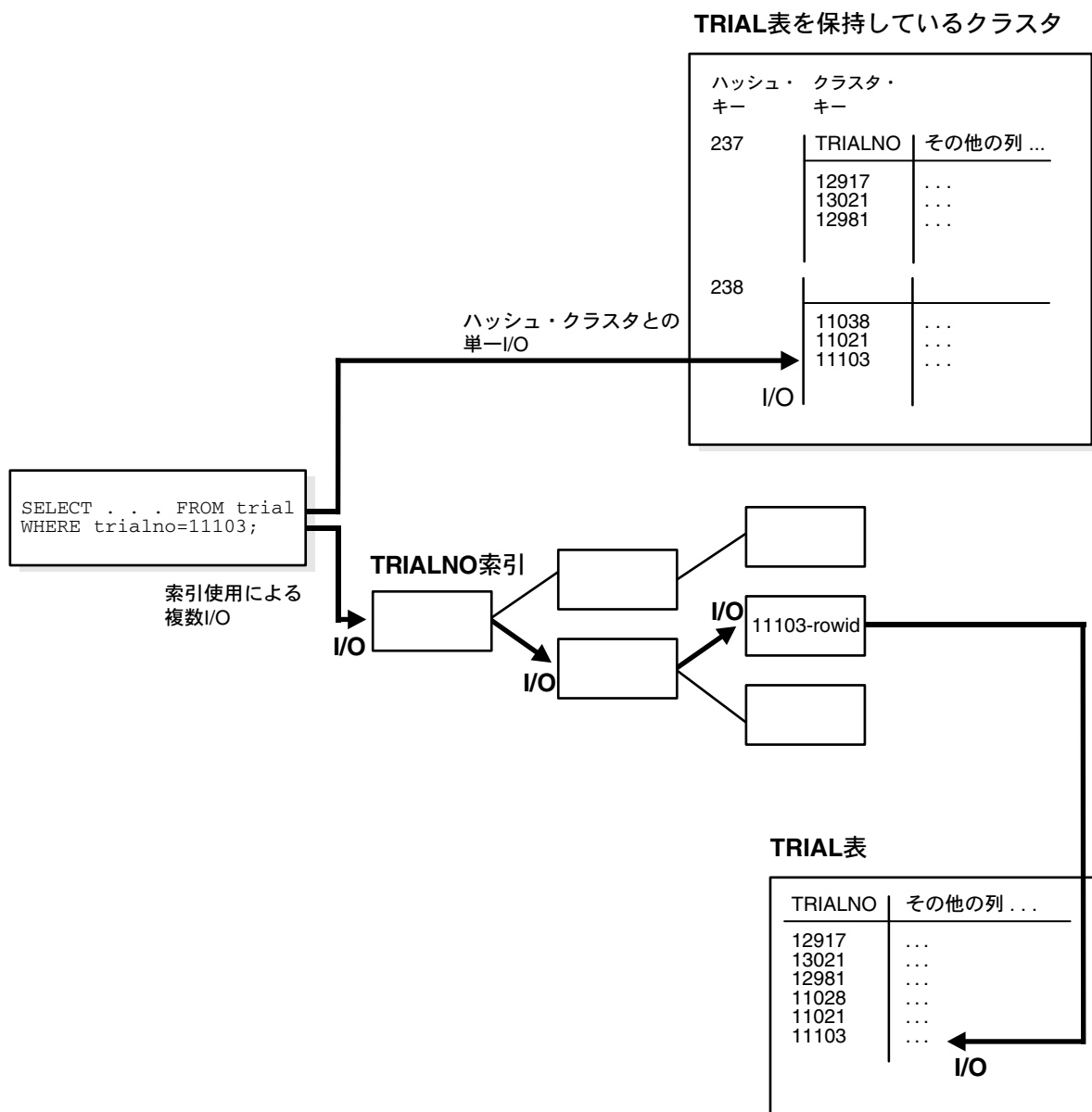
ハッシュ・クラスタは、関連する行をまとめて同じデータ・ブロック内に格納します。ハッシュ・クラスタ内の行は、そのハッシュ値に基づいてまとめて格納されます。

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 文の 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、最も近い素数にラウンドされます。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 文の 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ため注意してください。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。

関連項目： 各ハッシュ・キー値に対する領域が確保されないルールの例外については、10-60 ページの「[単一表ハッシュ・クラスタ](#)」を参照してください。

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コンポジット・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せずに、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。ただし、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 文の 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、スケールがゼロの数（整数）のみを含む単一列で構成されている必要があります。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 文の 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、クラスタ行は様々なハッシュ値に、より均等に分散されます。

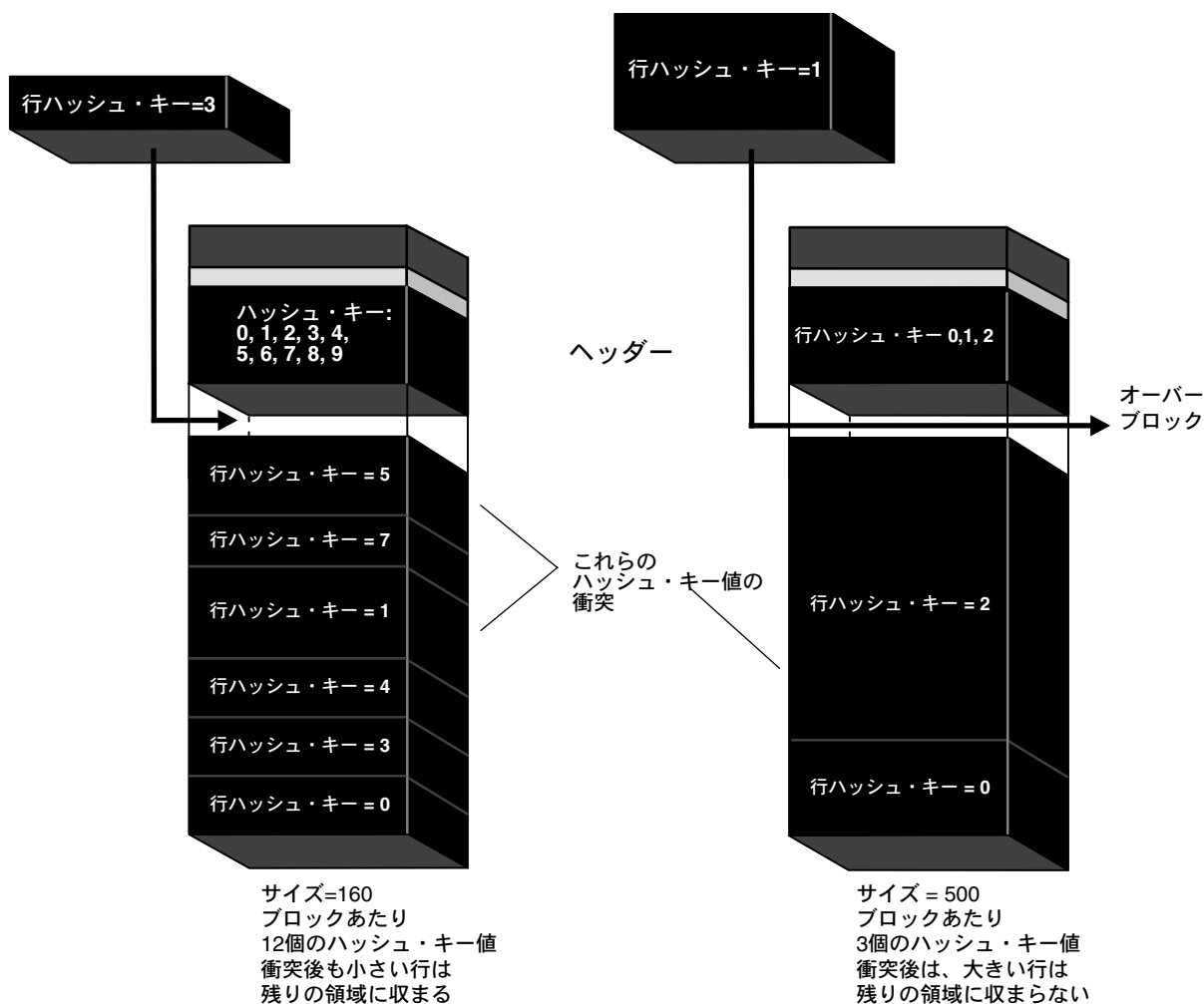
ハッシュ・クラスタに対する領域の割当て

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、いっぱいになったデータ・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・ブロックがいっぱい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・キーはいっぱいのデータ・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・ブロックが増加して、データ検索のパフォーマンスが低下する場合があります。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する必要があります。[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・キー値および対応するデータの平均サイズに依存します。

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、各行が一意のハッシュ・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スキャンします。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スキャンしなくても、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スキャンします。

関連項目： CREATE CLUSTER 文の SINGLE TABLE HASHKEYS 句の詳細は、『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

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概要](#)
- [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](#)

パーティション化の概要

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。

- [パーティション化について](#)
- [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のみです。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。

関連項目： 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 パフォーマンスのための設計およびチューニング』を参照してください。

パーティション化について

「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して、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。

関連項目： パーティション化方法の詳細は、11-11 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、INITRANS 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、パーティションとは異なり、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。

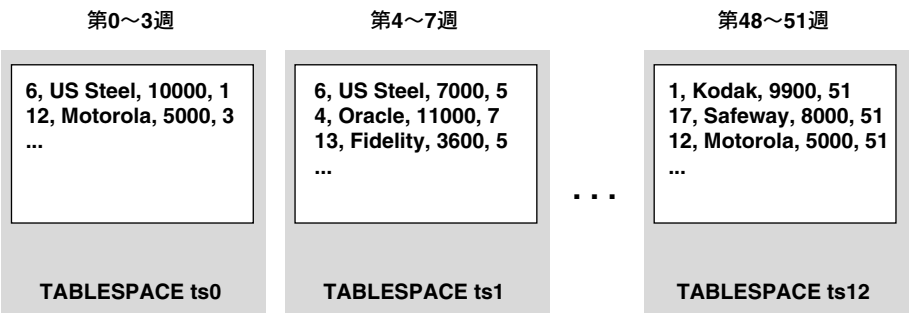
必要であれば、各パーティション（またはコンポジット・パーティション化された表や索引のサブパーティション）を、別々の表領域に格納できます。この方法には、次のような利点があります。

- データの破損による影響を抑えることができます。
- 各パーティションまたはサブパーティションのバックアップを個別に作成したりリカバリできます。
- I/O の負荷を均衡化するために、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ます。

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 );
```

関連項目： 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・ブルーニング

Oracle Server では、パーティションとサブパーティションを明示的に認識できます。この知識は、不要なパーティションやサブパーティションを「ブルーニング」して SQL 文を最適化するために使用されます。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・ブルーニング機能によって、データの量を大幅に低減できるため、問合せのパフォーマンスが実質的に向上します。

オプティマイザは、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ブルー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、オプティマイザはパーティションをブルーニングできません。同様に、ファンクション索引でない限り、SQL 文によって索引付きの列に関数が適用されると、オプティマイザは索引を使用できません。

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、パーティションのブルー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と改善できます。

不要なパーティションやサブパーティションを SQL 文からブルーニングする機能によって、次のように多数の用途で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・レベルまたはサブパーティション・レベルのロード
- ページ
- バックアップ
- リストア
- 再編成
- 索引作成

関連項目： SQL 文によってパーティション化される列に TO_DATE 関数が適用された結果の詳細は、11-20 ページの「[DATE データ型](#)」を参照してください。

パーティション・ワイズ・ジョイン

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・ジョイン」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・ジョインを使用するには、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。

パーティション・ワイズ・ジョインが使用されるのは、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ーニングとパーティション・ワイズ・ジョインを併用することもあります。

関連項目：

- 11-23 ページの「[同一レベル・パーティション化](#)」
- 『Oracle8i パフォーマンスのための設計およびチューニング』

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ます。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、大量のデータにアクセスして処理する必要がある、非常に複雑な問合せ向けに設計されています。

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。履歴データベースには、次の 2 つのクラスの表があります。

- 「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ます。履歴表には、次の 2 種類があります。
 - － 「ベース」表には、売上、伝票および注文などのベースライン情報が含まれます。
 - － 「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ます。

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 のほとんどのデータは、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、部門、場所および製品など、会社全体のビジネス・エンティティが記録されます。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。全社データの表が大きい場合でも、履歴データの表とエンタープライズ・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し、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

メンテナンス操作やリカバリ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リカバリ時間を短縮し、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と、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・ペ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ため、これらの操作にかかる時間を大幅に軽減できます。

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」を実行します。
- 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」させます。

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。後述で使用される、「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、グローバル索引や参照整合性制約など、パーティションをまたがって格納される構成体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) 句など、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、グローバル索引や参照整合性制約など、複数のパーティションにまたがって格納される構成体がない場合に、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、リカバリ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 文を実行すると、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リカバリできます。リカバリ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、リカバリ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ため、ある部分またはそ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。

- リカバリの単位（表領域）が小さくなるため、RECOVER 文の実行による休止時間が短くなります。
- リカバリ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）のリカバリに必要なディスク・リソースが少なくなります。
- リカバリ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化すればよいいため、使用できないデータの量が少なくなります。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。表スキャンが必要な問合せは、表内の各行を検査するため、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問合せは、表スキャンではなくパーティション・スキャン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スキャンできます。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して、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。

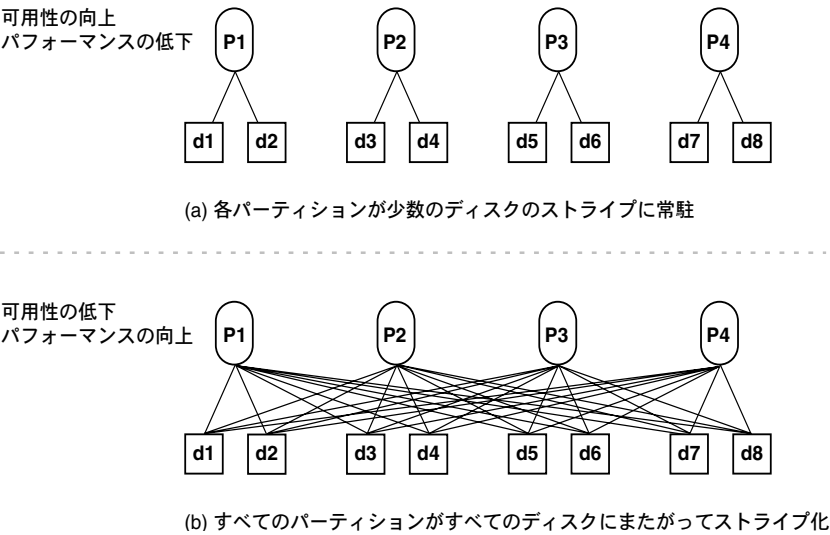
図 11-2 は、パーティション化とストライプ化を結合した 2 つの極端な例を示しています。図 11-2 の (a) と (b) は、どちらも 4 つのパーティションを 8 台のディスクに分散させています。(a) は各パーティションを専用の 1 組のディスクにストライプ化しているのに対して (b) は各パーティションを 8 台すべてのディスクにストライプ化しています。

(b) はパフォーマンス特性に優れていますが、1 台でもディスクが障害が起こると、すべてのパーティションが悪影響を受けます。

(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、可用性に優れています。

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



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薦めします。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すむ必要があります。

ただし、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して、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実行する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」と呼び、このビューを「パーティション・ビュー」と呼びます。パーティション・ビューは、Oracle7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 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。パーティション・ビューのかわりに、パーティション表を使用することをお薦めします。

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 文の EXCHANGE PARTITION 句を使用してパーティション表に変換できます。

関連項目： 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 文の句を指定して、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。

- 列や制約の定義などの、表の論理属性
- 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す。

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ます。

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。

コンポジット・パーティション化の場合、各サブパーティションの記述では、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。

関連項目： 日付書式の指定例は、11-20 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。

関連項目： 11-28 ページの「[索引のパーティション化](#)」

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 文を使用すると、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して定義します。

```
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します。これらの列を「パーティション化列」と呼びます。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」を構成します。
- *value_list* は、*column_list* に指定した列の値の順序付きリストです。*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() 関数のいずれかである必要があります。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）を定義します。この上限を「パーティション・バウンド」と呼びます。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・バウンドより小さい必要があります。

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・バウンド以上である必要があります。

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` 文を使用すると、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。

`ALTER TABLE EXCHANGE PARTITION` 文を使用すると、次のデータおよび索引セグメントを変換できます。

- レンジ・パーティションまたはサブパーティションと非パーティション表
- 非パーティション表とパーティション表のレンジ・パーティションまたはサブパーティション

関連項目：

- 11-20 ページの「[DATE データ型](#)」
- パーティション化キーがパーティション・バウンドと比較される方法の詳細は、11-19 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。

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、パラレル DML、パーティション・プルーニングおよびパーティション・ワイズ・ジョインなど、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。

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りません。
- 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くなります。
- パーティション化キーによるパーティション・プルーニングとパーティション・ワイズ・ジョインが重要です。

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す。記憶域パラメータは指定できません。

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。また、ハッシュ・パーティション化に使用するキーを選択する場合は、カーディナリティがきわめて重要です。たとえば、16個のハッシュ・パーティションが必要で、25個の個別値を含む列を選択したとします。ハッシュ・パーティションのうち8個には1つの個別値が含まれ、8個には2つの個別値が含まれています。したがって、ハッシュ・パーティションのサイズには大きな違いがあります。このように個別値が少数の場合は、個別値を含まないハッシュ・パーティションと、3個以上の個別値を含むハッシュ・パーティションができることも考えられます。ハッシュ・パーティション化を使用する理由の1つは、パーティションのサイズを均等化することであるため、カーディナリティの低いキー列にはハッシュ・パーティション化を使用しないようお勧めします。

通常、ハッシュ・パーティション化キーは、一意またはほぼ一意にする必要があります。表にはほぼ一意のキーが複数含まれ、複数がハッシュ・パーティション化の候補となることがあるため、パーティション化キーを選択する場合にはアクセス・プロファイルも考慮する必要があります。ハッシュ・パーティション化では、単一キー検索（`SELECT * FROM emp WHERE empno = 123` など）のパフォーマンスは改善できますが、範囲検索（`SELECT * FROM emp WHERE sal > 5000` など）のパフォーマンスは改善されません。さらに重要なのは、ハッシュ・パーティション化により、2つの表が結合キーでハッシュ・パーティション化されている場合に、パーティション・ワイズ・ジョインが使用可能になることです。これは、ハッシュ・パーティション化キーを選択するときに重要な、パフォーマンス関連の考慮事項です。

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。

ALTER TABLE EXCHANGE PARTITION 文を使用すると、次のデータおよび索引セグメントを変換できます。

- ハッシュ・パーティションまたはサブパーティションと、非パーティション表
- 非パーティション表と、パーティション表のハッシュ・パーティションまたはサブパーティション
- ハッシュ・パーティション化された表と、コンポジット・パーティション化された表のハッシュ・サブパーティション
- コンポジット・パーティション化された表のハッシュ・サブパーティションと、ハッシュ・パーティション化された表

関連項目： 11-5 ページの「[パーティション・プルーニング](#)」および「[パーティション・ワイズ・ジョイン](#)」

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っています。
- 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っています。
- 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ます。
- 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ます。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索引を作成できます。
- 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ます。

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、NLS DATE 書式が DD-MON-YYYY の場合に、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。

```
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します。

- インデックスパーティションまたはサブパーティションの名前は、同じ親インデックスに属するすべてのパーティションの中で一意にします。

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空間にあります。つまり、同じ親表または親インデックス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。

関連項目： 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、その親表または親インデックス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。親表または親インデックス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;
```

関連項目： SQL 文でパーティションとサブ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

表またはインデックスの「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
- NESTED TABLE、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）があります。
- 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）があります。

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 句の値が最も小さいパーティションであり、最後または最上位のパーティションは、VALUES LESS THAN 句の値が最も大きいパーティションです。

関連項目： 11-13 ページの「レンジ・パーティション化」

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規則も適用されます。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。

文字データのバイナリ値は、使用されるキャラクタ・セット（ASCII または EBCDIC など）によって異なります。たとえば、ASCII では、文字 A ～ Z は文字 a ～ z より小さいものとして定義されていますが、EBCDIC では、A ～ Z は a ～ z より大きいものとして定義されています。したがって、一方の順序に合せて設計されたパーティションは、他方の順序には機能しません。異なるキャラクタ・セットを使用している表からインポートした後に、その表を再パーティション化する必要があります。

関連項目：

- パーティション化キーの比較の詳細は、11-21 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。
- パーティション・バウンドの詳細は、『Oracle8i 管理者ガイド』を参照してください。
- キャラクタ・セットのサポートの詳細は、『Oracle8i NLS ガイド』を参照してください。
- インポート後に再パーティション化を必要とする表の例については、『Oracle8i ユーティリティ・ガイド』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 ('T'): A ~ H の文字で始まる州
- VALUES LESS THAN ('S'): I ~ R の文字で始まる州
- VALUES LESS THAN (MAXVALUE): S ~ Z 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 として処理されるためです。

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ーニングできます。

```
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 文出力の 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 $V1[i] = V2[i]$ である場合のみ
- $V1 < V2$ 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 $V1[i] = V2[i]$ である場合のみ
- $V1 > V2$ 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 $V1[i] = V2[i]$ である場合のみ

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 に属します。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る 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 に属します。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ため注意してください。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します。

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 $(10, \text{MAXVALUE}, 5)$ は、パーティション・バウンド $(10, \text{MAXVALUE}, 6)$ またはパーティション・バウンド $(10, \text{MAXVALUE}, \text{MAXVALUE})$ 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が、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）が同一である場合。
- 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ます。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・オブジェクトのタイプは、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) であること。
- パーティション化列が同数 (M) であること。
- $1 \leq i \leq N$ 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こと。

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、精度およびスケールを含めて同じになっていること。

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リカバリ操作の際に、休止時間と使用できないデータの量を減らします。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・ジョインが使用可能になります。
- データの関連サブセットにおける表領域の不完全リカバリ（Point-in-Time リカバリ）を容易にします。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リカバリできます。

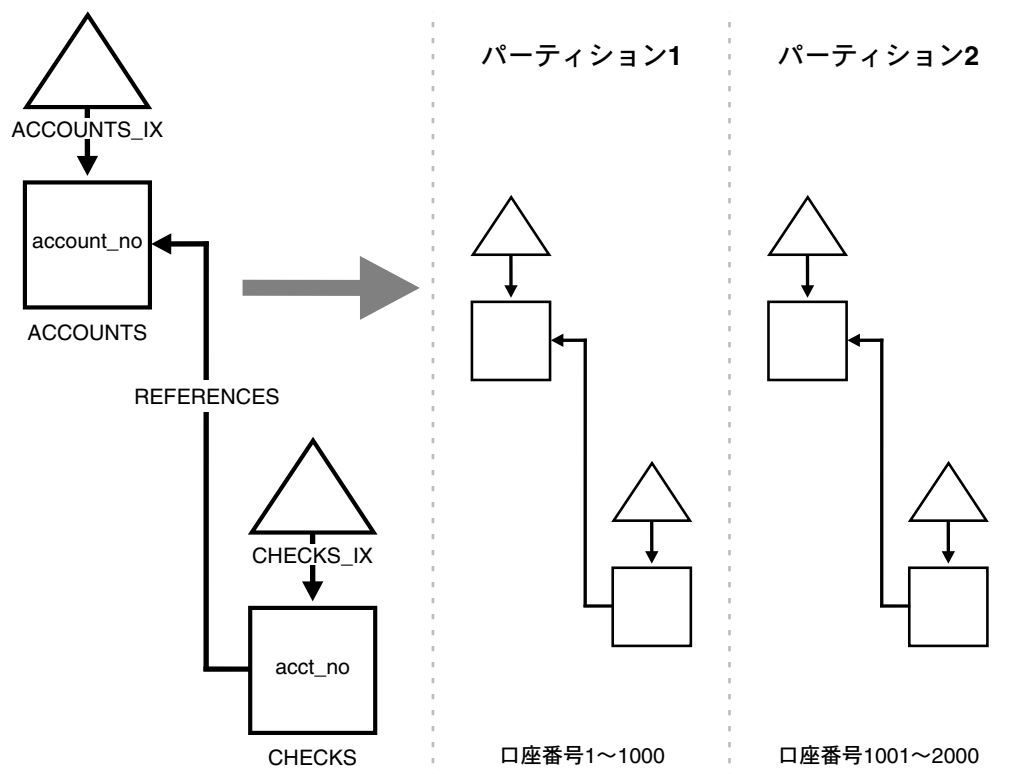
同一レベル・パーティション化の例

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す。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 に対する索引です。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す。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ます。
 - 表がクラスタの一部になっていない場合
 - LONG または LONG RAW データ型を含んでいない場合
- 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ます。
 - 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ます。
 - 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ます。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。

関連項目： 11-31 ページの「[グローバル・パーティション索引](#)」

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、それに対応するベース表に（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ます。

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ます。

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ます。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。

関連項目： 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、11-38 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その表領域だけあるという点で、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ます。
- パーティションの物理属性値を指定しなければ、それに対応するベース表に（明示的に、またはデフォルトで）指定した属性が使用されます。

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。

関連項目： 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、11-38 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ます。
 - － クラスタ索引ではないこと。
 - － クラスタ化表に対して定義された索引ではないこと。
- 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ます。
 - － 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ます。
 - － 非パーティション表には、パーティション B-tree 索引または非パーティション B-tree 索引（あるいはその両方）を含めることができます。
- 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ません。
- 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あることが必要です。

ただし、パーティション索引には次の4種類があるため、パーティション表よりも複雑です。

- 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
- グローバル非同一次元キー索引

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キー索引は、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 属性を指定して作成されます。

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に対して、SPLIT PARTITION または ADD PARTITION 以外のメンテナンス操作を実行する場合は、索引パーティションを 1 つ再作成するだけですみます。
- 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・サイズに比例します。
- 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ます。
- 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します。
- 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、より効率的な問合せアクセス計画を生成できます。
- ローカル索引により、表領域の不完全リカバリの作業を簡略化できます。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リカバリするには、対応する索引エントリも同じ時点までリカバリ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リカバリできます。
- 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作成できます。

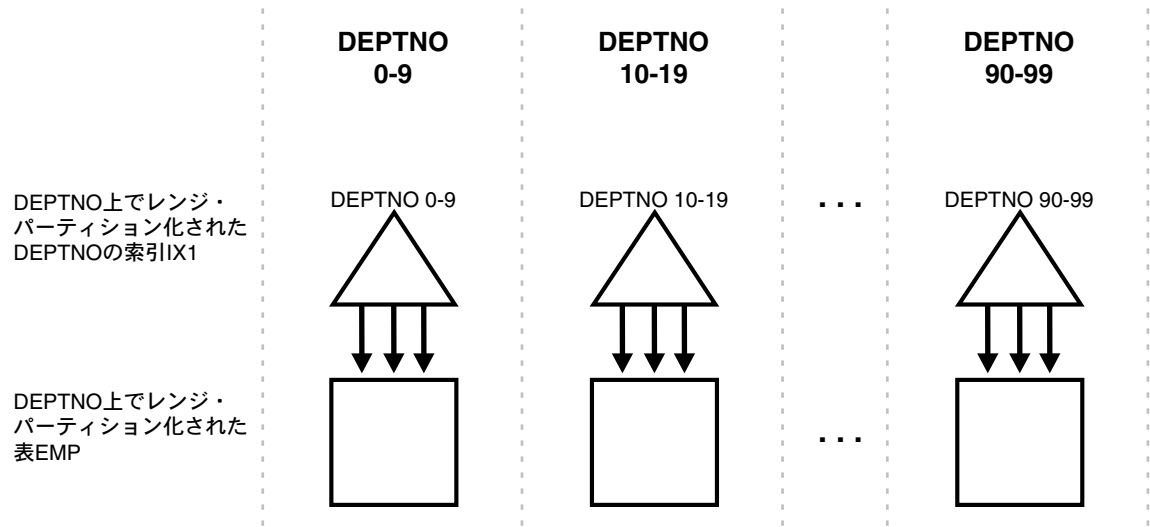
関連項目： DBMS_PCLXUTIL パッケージの説明は、『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』を参照してください。

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。

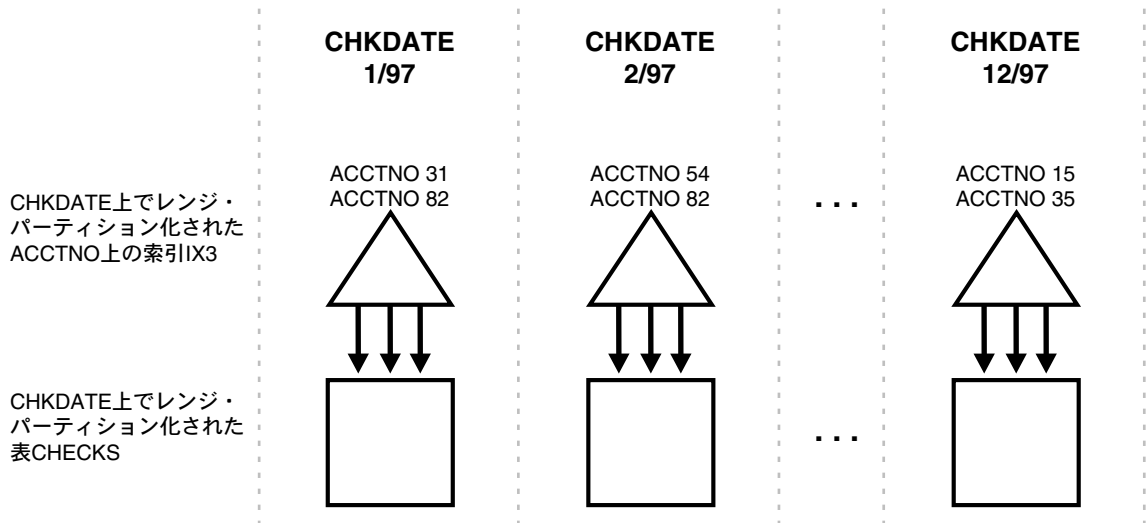
図 11-4 ローカル同一キー索引



ローカル非同一次元索引 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「非同一次元」(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一次元索引は作成できません。

図 11-5 では、ローカル非同一次元索引の別の例を示します。

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、基礎と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 属性を指定して作成されます。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-tree が含まれています。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一キー・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「非同一キー」パーティション索引です。Oracle は、グローバルな非同一キー・パーティション索引をサポートしていません。

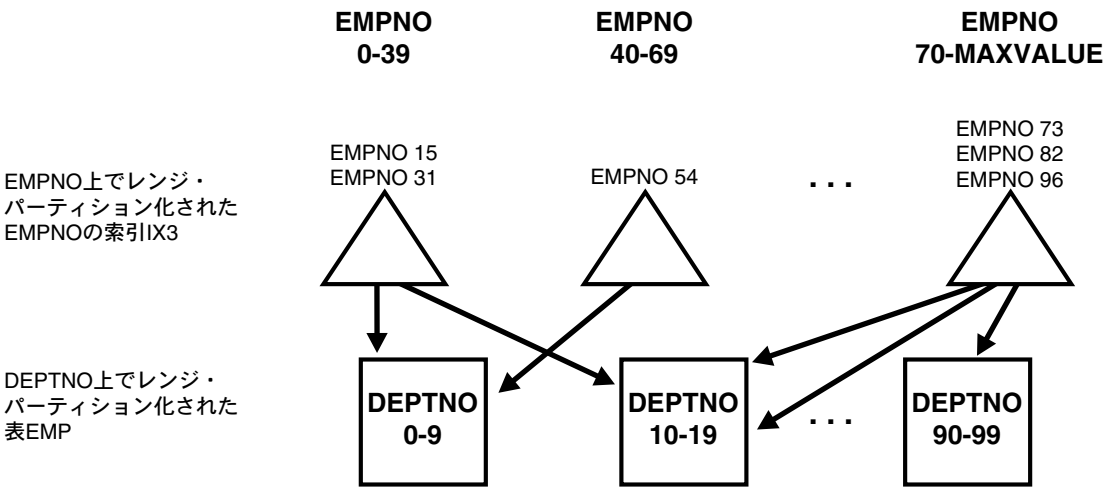
グローバル同一キー・パーティション索引は、一意索引にすることも、非一意索引にすることもできます。

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ます。その結果、グローバル索引では、グローバル索引や索引パーティションの再作成などの、パーティションのメンテナンスに、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リカバリする場合、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リカバリする必要があります。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 (リカバリ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まとめ

表 11-1 に、Oracle がサポートするパーティション索引のタイプをまとめます。

- 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す。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す。それ以外の場合、索引は「非同一キー」索引です。

表 11-1 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「週別にパーティション化された SALES 表」の SALES 表を考えます。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ため、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。

同ーキー索引と非同ーキー索引のパフォーマンスへの影響

同ーキー索引をプローブするよりも、非同ーキー索引をプローブ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・プルー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、11-30 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、オプ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、オプ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。）

索引が非同一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する必要がある場合があります。これには、1 つのキーを調べるか、索引レンジ・スキャンを実行する必要があります。範囲を指定してスキャンする場合、Oracle は N 個の索引パーティションからの情報を組み合せる処理も実行する必要があります。たとえば、11-31 ページの図 11-5（「ローカル非同一キー索引」）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プルーニングします。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。

非同一キー索引の場合、パーティション・キーが WHERE 句の一部であっても、索引キーの一部でなければ、オプティマイザは、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。

ローカルな非同一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する必要がある場合、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。考慮する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・プルーブの数が最小になるため、ローカル非同一キー索引を使用するよりもパフォーマンスが向上します。
- 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します。ローカル非同一キー索引は、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、ローカル非同一キー索引を使用すると、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スキャンできるため、パフォーマンスを改善できます。

たとえば、11-31 ページの図 11-5（「ローカル非同一キー索引」）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、非同一キー索引 IX3 のすべてのパーティションがパラレルでスキャンされます。一方、11-30 ページの図 11-4（「ローカル同一キー索引」）の表 DEPTNO に対して述語「DEPTNO between 40 and 45」を使用した問合せは、同一キー索引 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。

- 履歴表の場合、索引はできるだけローカルにします。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ります。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キー索引はサポートされていないためです。

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、後で 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ます。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ます。

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ます。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ます。

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。

関連項目： LOB 索引パーティションの物理属性に関する追加情報は、11-39 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 列を含む表はパーティション化できます。ただし、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、ベース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または索引のパーティション、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と、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。

関連項目：

- 12-11 ページの「LOB データ型」
- 11-55 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」

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ため注意してください。

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 に配置されます（ルール 1）。
- 列 C の LOB データ・パーティションは、表領域 TSB に配置されます（ルール 2）。
- 列 D の LOB データ・パーティションは、表領域 TS1 に配置されます（ルール 3）。

関連項目： 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、11-39 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. それ以外の場合、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 に配置されます (ルール 1)。
- 列 C の LOB 索引パーティションは、表領域 TSB に配置されます (ルール 2)。

- 列 D の LOB 索引パーティションは、表領域 TS1 に配置されます（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りますが、通常の表には主キーがない場合があります。
- 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ます。
- 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ます。したがって、LOB を持たないヒープ構成表は単一データ・セグメントに格納されますが、LOB を持たない索引構成表には、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。

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ません。
- パーティション列は、主キー列のサブセットであることが必要です。

- 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ます。
- 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ます。
- 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・レベルで指定できます。

関連項目： 10-39 ページの「[索引構成表](#)」

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り、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・セグメントを持ちます。
- 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ます。
- 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ます。
- 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ます。
- 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ます。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ます。表レベル・デフォルトが指定されていないければ、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_P*n* と SYS_IOT_OVER_P*n* です。

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 に関して追加の見積りデータを含めることができます。

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ためです。パーティションが識別されると、Oracle は見積りを使用して、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。見積りが無効な場合は、関連するパーティションの B-tree に対して索引スキャンを実行する必要があります。

関連項目：

- 10-42 ページの「[索引構成表の 2 次索引](#)」
- 12-19 ページの「[論理 ROWID](#)」

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」が適用されます。
- 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」が適用されます。

パーティション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。

関連項目： DML および DDL 文に対するパーティション・ロックの詳細は、11-48 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 ロックを取得します。
- 「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 ロックを取得します。
- 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 ロックを取得します。

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ため、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。

関連項目：『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

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、またはデータのバルク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、データをスキャンしたり更新しません。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のロード・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リカバリ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します。
- 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）をマージします。
- 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）を合わせます。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します（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）を削除します。
- 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ます（領域の再生を要求するか、または要求しません）。
- パーティションまたはサブパーティションを交換します。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します。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します。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します。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します。
- 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 マークを付けます。
- 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します。
- 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します。
- 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します。
- 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します。

関連項目：

- 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』を参照してください。
- LOB データのメンテナンスの詳細は、11-55 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。

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、どの間合せおよび 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）があります。
- 次の操作を除き、すべての索引操作は 1 ステップです。
 - 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 ステップ操作です。
 - 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 操作と同じです。
 - 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ます。

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 ロックを取得します。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 つのステップがあります。
 - 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みます。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スキャンまたは更新します。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します。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を要しますが、その他の操作を同時に実行できます。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 ロックを取得します。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 ステップ操作です。
 - 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、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、ベース表は行排他 (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。

従来型パスの 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ます。
- その他の操作 (DDL、ユーティリティまたは DML) は実行できません。

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、操作の終了時にカレント・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・パス・ロードがパーティションに対して進行中の場合

- ハッシュ・パーティション化された表との変換を実行する場合を除き、同じ表内の別のパーティションの移動、分割、変換またはダイレクト・パス・ロードを実行できます。この場合、ハッシュ・パーティション化された表に対する DDL または DML は実行できません。
- 表に対する問合せを実行できます。

- パーティションに書込みを実行しない DML 操作であれば、表に対する DML 操作を実行できます。
- パーティションに対応するローカル索引パーティション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ません。

グローバル索引について、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 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ます。
- 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ます。
- 表に対する DML 操作は実行できません。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ません。

ローカル索引について、ALTER INDEX REBUILD PARTITION 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・ロードできます。
- 表に対する問合せを実行できます。
- 基礎になる表パーティションに書込みを実行しない DML 操作であれば、表に対する DML 操作を実行できます。
- 索引内の他のパーティションを再作成できます。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ません。

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する必要があるものがあります。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

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 11-2 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 11-2 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

表 11-2 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
ALTER TABLE EXCHANGE SUBPARTITION WITH VALIDATION	表の間合せ
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	変換するハッシュ・パーティションを除く、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	変換するハッシュ・パーティションを除く他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
	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
IMPORT SUBPARTITION	表の間合せ
	表の DML
	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
EXPORT PARTITION	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

表 11-2 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
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。

- 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、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み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。

関連項目：

- 11-37 ページの「[LOB 列を持つ表のパーティション化](#)」
- 11-48 ページの「[メンテナンス操作の同時実行モデル](#)」

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にエラーを戻します。これは、問合せの開始後にパーティションが **USABLE** 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 句をサポートします。

- 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 句をサポートしていない DDL 文やユーティリティ文は、常にリカバリ可能モード (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、22-5 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、物理記憶域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して、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ます。
- 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ます。

- 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ます。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 とマークされるため注意してください。

Oracle が、対応する表パーティションの ADD または SPLIT 操作により、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します。その名前を割り当てられない場合は、SYS_Pnnn という形式の名前を生成します。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域属性を使用してセグメントを作成します。DEFAULT の場合、ローカル索引パーティションは、対応するベース表パーティションと同じ位置に配置されます。親索引に DEFAULT 以外の表領域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ます。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、次の操作を実行できます。

1. 表パーティションを非パーティション表、ハッシュ・パーティション表またはコンポジット・パーティション表に EXCHANGE します。
2. その表の索引を削除し、ロード操作を実行します。
3. 索引を作成し、INCLUDING INDEXES 句を使用して表を元のパーティションに EXCHANGE します。

関連項目： 新しいローカル索引パーティションの命名方法の詳細は、11-17 ページの「パーティション名とサブパーティション名」を参照してください。

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。

関連項目： グローバル索引の管理の詳細は、11-32 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、領域をリカバリし、パフォーマンスを改善します。
- 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します。
- 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します。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します。

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作成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パスの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・パスの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状態のままになります。(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ません。
 - － ユーザーがSKIP_INDEX_MAINTENANCE句を要求しました。
- 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 になることに注意してください。

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にではなく、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。

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ます。
- 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ます。
- 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ます。

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。

関連項目：

- 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](#)」
- ALTER TABLE 文と ALTER INDEX 文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

バルク操作を、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、バルク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。

関連項目： 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 句を使用する場合の制限について説明し、この 2 つの句を含む 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。

- 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。

- 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。

- ベース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、ベース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);
```

組込みデータ型

この章では、Oracle 組込み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 の組み込みデータ型は、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

注意： PL/SQL には、その他のデータ型として、BOOLEAN、参照型、コンポジット型（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、『Oracle8i 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 組込みデータ型の要約

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、各国語キャラクタ・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（固定幅の各国語キャラクタ・セットの場合）またはバイト数（可変幅の各国語キャラクタ・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>)	各国語キャラクタ・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（固定幅の各国語キャラクタ・セットの場合）またはバイト数（可変幅の各国語キャラクタ・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> やスケール <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 4712 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 組込みデータ型の要約（続き）

データ型	説明	列の長さデフォルト
RAW (<i>size</i>)	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語キャラクタ・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、各組込みデータ型について、さらに詳しく説明します。

関連項目： 組込み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・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・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・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。

関連項目：

- 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。
- 文字データ変換の詳細は、『Oracle8i NLS ガイド』を参照してください。あるキャラクタ・セットから別のキャラクタ・セットにデータを変換する場合は、[CHAR | VARCHAR] のデータが適切な書式の文字列で構成されていることを確認する必要があります。

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 までの値で指定する必要があります（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ります。
- 短い値を与えると、固定長に合せて空白が埋め込まれます。
- 末尾に空白が続く長い値を与えると、固定長に合せて値から空白が切り捨てられます。
- 値が大きすぎると、Oracle からエラーが戻されます。

Oracle は、空白埋め比較方法を使用して CHAR 値を比較します。

関連項目： 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ます。

たとえば、列 VARCHAR2 を最大サイズ 50 文字で宣言するとします。シングルのバイト・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）のみが格納されます。

Oracle は、非空白埋め比較方法を使用して VARCHAR2 値を比較します。

関連項目： 非空白埋め比較の詳細は、『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、文字データ型に様々なキャラクタ・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、シングルの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語キャラクタ・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。

関連項目： 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語キャラクタ・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（固定長の各国語キャラクタ・セットの場合）またはバイト数（可変長の各国語キャラクタ・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。

関連項目： 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）または各国語キャラクタ・セット・データ（NCLOB）を格納できます。

関連項目： 12-11 ページの「[LOB データ型](#)」

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われます。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の場合、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。

関連項目：

- LONG データ型の制限の詳細は、『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。
- LONG RAW データ型の詳細は、12-14 ページの「[RAW および LONG RAW データ型](#)」を参照してください。

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 です。
- $-1 \times 10^{-130} \sim 9.99..99 \times 10^{125}$ の範囲の負の数。最大有効桁数は 38 です。
- 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ます）。

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「スケール」（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。スケールを指定しないと、スケールはゼロになります。

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、スケールのみを指定して、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、指定したスケールが保持されます。

数値フィールドを指定するときは、精度とスケール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、様々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

負のスケールを指定すると、小数点の左側の指定した桁数で実際のデータがラウンドされます。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 の位でラウンドさ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。カレント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。負数の長さには、符号が含まれます。

このことを考慮に入れると、値の精度が p のとき、数値データ `NUMBER(p)` の列データ・サイズ (バイト数) は、次の式を使用して計算できます。

$$\text{ROUND}((\text{length}(p) + s) / 2) + 1$$

数値が正数であれば s はゼロとなり、数値が負数であれば s は 1 となります。

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）が基準になるため、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ため、結果は整数値になります。ユリウス暦は、様々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様々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リカバリなどの 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。

関連項目：

- 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。
- 日付書式コードの詳細は、『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ると、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ますが、LONG 列は 1 つしか含むことができません。

- 1 つ以上の LOB 列を含む表はパーティション化できますが、LONG 列を含む表はパーティション化できません。
- LOB の最大サイズは 4GB ですが、LONG の最大サイズは 2GB です。
- LOB はデータへのランダム・アクセスをサポートしていますが、LONG は順次アクセスしかサポートしていません。
- 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ますが、LONG データ型には指定できません。
- ローカル変数のように機能するテンポラリ LOB を使用して、LOB データの変換を実行できます。テンポラリ内部 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。

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。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。

関連項目：

- LOB 属性 LOGGING および NOLOGGING の詳細は、22-8 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。
- 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。
- LOB 記憶域と LOB ロケータの詳細は、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。

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語キャラクタ・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。

関連項目： 各国語キャラクタ・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取りのみで (順次読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： 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（ALTER SESSION 文の NLS_LANGUAGE パラメータで設定）が異なる場合に、これらのキャラクタ・セット間で CHAR、VARCHAR2 および LONG データを自動的に変換します。

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。

関連項目： LONG RAW データ型に関する他の制限の詳細は、『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。

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します。
- 「論理 ROWID」は、索引構成表に行のアドレスを格納します。

「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 のみでなく、ゲートウェイを介してアクセスされる Oracle 以外の表など、外部表の 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。

関連項目： 12-21 ページの「[Oracle 以外のデータベースの ROWID](#)」

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 ... 文を実行し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と、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 文では設定できません。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 値を内部的に使用します。

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。

関連項目： 12-19 ページの「[ROWID の使用方法](#)」

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 Import および Export ユーティリティを使用してエクスポートおよびインポートされない限り変更されません。表から行を削除し、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します。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」形式も使用できます。

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
-----	-----
AAAAaoAATAABrXAAA	BORTINS
AAAAaoAATAABrXAAE	RUGGLES
AAAAaoAATAABrXAAG	CHEN
AAAAaoAATAABrXAAN	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）。
- BBBB: 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。

関連項目： 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確認する場合（表のストライプ化の場合など）、次のように拡張 ROWID を問い合せると、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

関連項目：

ROWID の使用例の詳細は、次のマニュアルを参照してください。

- 『Oracle8i SQL リファレンス』
- 『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle8i パフォーマンスのための設計およびチューニング』

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す。
- ROWID は表の編成を把握するために使用できます。
- ROWID は表の中の行の一意識別子です。

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 ROWID 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 ごとに、「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作成時に、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・ブロックを直接プローブできます。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プローブ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ます。

ROWID 疑似列を使用して、索引構成表から論理 ROWID を選択できます。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ます。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すが、複数のブロック・アクセスを必要とする場合があります。
- 行の論理 ROWID は、主キー値に変更がなければ変化しません。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ます。

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。

関連項目： 12-15 ページの「[ROWID および UROWID データ型](#)」

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、索引を再作成して最新の推測を取得できます。索引構成表の 2 次索引を再作成するには、通常の表の索引を再作成する場合とは異なり、ベース表を読み込む必要があるため注意してください。

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 文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 文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 次索引を再作成（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。

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。

関連項目： 統計収集の詳細は、21-9 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。

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。

関連項目：

- 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。
- 12-15 ページの「[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

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)

表 12-3 ANSI データ型から Oracle データ型への変換（続き）

ANSI SQL データ型	Oracle データ型
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、GRAPHIC、VARGRAPHIC および LONG VARGRAPHIC は、対応する Oracle データ型が存在しないため使用できません。TIME 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。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場合は、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()

関連項目： 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

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。

- [ユーザー定義データ型の概要](#)
- [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](#)

ユーザー定義データ型の概要

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。このツールを使用すると、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・ロジックのみです。

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必要になることがあります。たとえば、出荷済または未出荷の品目の現行の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」と呼びます。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」と呼びます。

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などを実行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

ユーザー定義データ型には、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、組込み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。

関連項目：

- [第12章「組込みデータ型」](#)
- 『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーションアル機能』

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、組込み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実装します。

オブジェクト型はテンプレートです。テンプレートに合せて構造化されたデータ単位を「オブジェクト」と呼びます。

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、組込み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 NESTED TABLE です。

```
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。

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」と呼びます。

関連項目：

- 13-11 ページの「[NESTED TABLE](#)」
- 発注例の詳細は、『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーショナル機能』を参照してください。
- 13-8 ページの「[行オブジェクトと列オブジェクト](#)」

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、次の 3 つのカテゴリに大別できます。

- 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() のような、「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 で型メソッドの実装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。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ためです。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呼びます。

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す。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 )
```

次のような属性を持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。

関連項目： NULL オブジェクトと NULL 属性の説明は、『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーショナル機能』を参照してください。

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、特定の組込み型の 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限り、任意のユーザー定義型の 2 つの項目を比較することはできません。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・メソッドとオーダー・メソッドを提供しています。

「マップ」メソッドでは、組込み型を比較する 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。

「オーダー」メソッドは、より一般的な方法です。オーダー・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必要とする場合が考えられます。

オブジェクト型を定義する際、マップ・メソッドかオーダー・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します。
- 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します。
- その他の場合、Oracle は比較できないことを報告します（NULL）。

関連項目： 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 オブジェクト・リレーショナル機能』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、単一列表としての 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」と呼びます。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」と呼びます。

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 バイトの一意の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する必要があります。

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。

関連項目： 第 14 章「オブジェクト・ビュー」

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 という組み込み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、コレクション要素または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」と呼びます。有効範囲付き REF は記憶域必要量が少なく、有効範囲なしの REF よりも効率的にアクセスできます。

参照先がない 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「参照先がない REF」と呼びます。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」と呼びます。Oracle には、そのための DEREF 演算子が用意されています。参照先がない 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「暗黙の参照解除」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 演算子を適用すると取得できます。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;
```

関連項目： REF の使用例は、『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーショナル機能』を参照してください。

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「NESTED TABLE」と呼びます。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 NESTED TABLE 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）であるため、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。

関連項目： 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 オブジェクト・リレーショナル機能』を参照してください。

NESTED TABLE

「NESTED TABLE」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、この列の型は組込み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 NESTED TABLE 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 NESTED TABLE 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。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。

NESTED TABLE 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。

関連項目：

- ネストしたカーソルについては、『Oracle8i リファレンス・マニュアル』を参照してください。
- NESTED TABLE の使用方法の詳細は、『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーショナル機能』を参照してください。

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複数あります。

- [SQL](#)
- [PL/SQL](#)
- [Pro*C/C++](#)
- [OCI](#)
- [OTT](#)
- [JPublisher](#)
- [JDBC](#)
- [SQLJ](#)

SQL

Oracle SQL データ定義言語は、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 NESTED TABLE、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間合せおよび更新。
- REF の操作。

関連項目： 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、情報の非表示、オーバーロード、例外処理などの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実装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。

関連項目： PL/SQL の詳細説明は、『Oracle8i 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。横断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。

関連項目：

- 13-14 ページの「OCI」
- Pro*C プリコンパイラの詳細説明は、『Oracle8i 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、同時発生へ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します。
- 横断ポインタまたは REF によってオブジェクト・キャッシュ内のオブジェクトをアクセスし、操作し、管理します。
- Oracle の日付および文字列、数値を C データ型に変換します。
- オブジェクト・キャッシュのメモリーのサイズを管理します。

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。

関連項目：『Oracle8i コール・インタフェース・プログラマーズ・ガイド』

OTT

Object 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。

関連項目：

- 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』
- 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』

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。Java Publisher 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。

関連項目：『Oracle8i JPublisher ユーザーズ・ガイド』

JDBC

Java Database Connectivity (JDBC) は、Oracle Server 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 プログラムからデータベース内で定義されたオブジェクト型とコレクション型にアクセスできるようにします。
- 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 のクラスに変換します。

関連項目：『Oracle8i JDBC 開発者ガイドおよびリファレンス』

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。

関連項目：『Oracle8i Java 開発者ガイド』

オブジェクト・ビュー

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。

- [オブジェクト・ビューの概要](#)
- [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](#)

関連項目：『Oracle8i アプリケーション開発者ガイド オブジェクト・リレーショナル機能』

オブジェクト・ビューの概要

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（組込み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ます。
- 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します。
- 既存のオブジェクト指向アプリケーションで従来の RDBMS データを使用します。

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横断するため、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ため、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、様々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的に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、ベース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・プログラムがベース表の対応する行を一意に識別しやすくなります。

関連項目： INSTEAD OF トリガーの記述方法の詳細は、14-5 ページの「[オブジェクト・ビューの更新](#)」を参照してください。

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横断します。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。

関連項目： 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のと同じ 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・ビューのベース表を更新します。

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」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ため、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。

関連項目：

- 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 オブジェクト・リレーショナル機能』を参照してください。
- [第 19 章「トリガー」](#)

ビュー内の NESTED TABLE の列の更新

NESTED TABLE 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、NESTED TABLE 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 NESTED TABLE 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します。親行の更新によってコレクション全体が置き換えられると、NESTED TABLE 列の INSTEAD OF トリガーは起動しないことに注意してください。

- 関連項目：** NESTED TABLE 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。

第 V 部

データ・アクセス

第 V 部では、SQL 文からなるトランザクションを使用して、Oracle データベース内のデータにアクセスする方法について説明します。また、データ・アクセスのための追加機能を提供するプロシージャ言語の構文についても説明します。

第 V 部には、次の章が含まれています。

- [第 15 章「SQL と PL/SQL」](#)
- [第 16 章「トランザクションの管理」](#)
- [第 17 章「プロシージャとパッケージ」](#)
- [第 18 章「アドバンスト・キューイング」](#)
- [第 19 章「トリガー」](#)
- [第 20 章「Oracle の依存性の管理」](#)

この章では、構造化問合せ言語（SQL）と、SQL に対する Oracle のプロシージャ型拡張機能である PL/SQL の概要について説明します。この章の内容は、次のとおりです。

- [構造化問合せ言語の概要](#)
- [SQL の処理](#)
- [PL/SQL](#)
- [PL/SQL Server Pages](#)

関連項目：

- 『Oracle8i SQL リファレンス』
- 『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

構造化問合せ言語の概要

SQL は、シンプルかつ強力なデータベース・アクセス言語です。SQL は非プロシージャ型言語です。つまり、ユーザーが実行する処理内容を SQL で記述すると、データベースをナビゲートして指定されたタスクを実行するためのプロシージャが、SQL 言語コンパイラによって自動的に生成されます。

SQL は、IBM 社の研究所で開発、定義され、リレーショナル・データベース管理システムの標準言語として ANSI/ISO により改良されました。SQL-99 の最低規格合致性レベルをコアと呼びます。コア SQL-99 は、SQL-92 のエントリ・レベルの仕様のスーパーセットです。Oracle8i は、SQL-99 コア仕様に対して広範囲の互換性を持っています。ただし、SQL-99 コア機能には、現在のところ Oracle8i に実装されていないものや、Oracle8i の実装とは異なるものがあります。オラクル社は、将来のリリースで SQL-99 コア機能を全面的にサポートすることを目指すとともに、既存アプリケーションの上位互換性を提供しています。

Oracle SQL には、ANSI/ISO 規格 SQL 言語に対応する多くの拡張機能が組み込まれており、Oracle Tools とアプリケーションでは、追加の文を提供します。SQL*Plus、Oracle Enterprise Manager および Server Manager などの Oracle Tools を使用すると、Oracle データベースに対して ANSI/ISO 規格の SQL 文、またはこれらのツールで使用可能な追加の文や機能を実行できます。

いくつかの Oracle Tools およびアプリケーションでは、SQL の使用は簡略化またはマスクされていますが、すべてのデータベース操作は SQL を使用して実行されます。その他のデータ・アクセス方法を使用すると、Oracle に組み込まれているセキュリティが活用されず、データのセキュリティと整合性が損なわれる可能性があります。

この項で取り上げるトピックは次のとおりです。

- [SQL 文](#)
- [非標準 SQL の識別](#)
- [再帰 SQL](#)
- [カーソル](#)
- [共有 SQL](#)
- [解析](#)

関連項目：

- SQL 文および SQL のその他の部分（演算子、関数および書式モデルなど）の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- Oracle Enterprise Manager の詳細は、『Oracle Enterprise Manager 管理者ガイド』を参照してください。
- SQL 文との相違点など、SQL*Plus の文の詳細は、『Oracle8i 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。

SQL 文

Oracle データベースの情報に対するすべての操作は、SQL「文」を使用して実行します。SQL 文は、有効な「SQL 文」の特定のインスタンスです。文の一部は、SQL「予約語」で構成されます。SQL 予約語は、SQL で特別な意味があり、他の目的には使用できません。たとえば、SELECT と UPDATE は予約語のため、表名には使用できません。

SQL 文は簡単な文ですが、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。次に示すように、文は、完全な SQL 文と同等のものであることが必要です。

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

実行できるのは完全な SQL 文のみです。次のような断片部分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
```

Oracle SQL 文は、次のカテゴリに分類できます。

- データ操作言語文 (DML)
- データ定義言語文 (DDL)
- 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

関連項目：

PL/SQL プログラム・ユニットでの SQL 文の使用の詳細は、次の章を参照してください。

- [第 17 章「プロシージャとパッケージ」](#)
- [第 19 章「トリガー」](#)

データ操作言語文

データ操作言語（DML）文は、既存のスキーマ・オブジェクト内のデータの問合せや操作を実行します。次のことを実行できます。

- 1 つ以上の表またはビューからデータを取り出します（SELECT）。
- 表またはビューに新しいデータ行を追加します（INSERT）。
- 表またはビューの既存の行の列値を変更します（UPDATE）。
- 表またはビューから行を削除します（DELETE）。
- SQL 文の実行計画を表示します（EXPLAIN PLAN）。
- 表またはビューをロックして、一時的に他のユーザーのアクセスを制限します（LOCK TABLE）。

DML 文は、最も頻繁に使用する SQL 文です。次に、DML 文の例をいくつか示します。

```
SELECT ename, mgr, comm + sal FROM emp;
```

```
INSERT INTO emp VALUES  
(1234, 'DAVIS', 'SALESMAN', 7698, '14-FEB-1988', 1600, 500, 30);
```

```
DELETE FROM emp WHERE ename IN ('WARD','JONES');
```

データ定義言語文

データ定義言語（DDL）文は、スキーマ・オブジェクトに対し、定義、構造の変更、および削除を実行します。DDL 文によって、次のことを実行できます。

- スキーマ・オブジェクトとデータベース構造（データベースそのものとデータベース・ユーザーを含む）を作成、変更および削除します（CREATE、ALTER、DROP）。
- スキーマ・オブジェクトの名前を変更します（RENAME）。
- スキーマ・オブジェクトの構造を削除することなく、それらのオブジェクトの中のすべてのデータを削除します（TRUNCATE）。

- スキーマ・オブジェクトに関する統計情報の収集、オブジェクト構造の妥当性チェック、オブジェクト内の連鎖行のリスト表示を行います（ANALYZE）。
- 権限とロールの付与および取消しを行います（GRANT、REVOKE）。
- 監査オプションのオン / オフを切り換えます（AUDIT、NOAUDIT）。
- データ・ディクショナリにコメントを追加します（COMMENT）。

DDL 文は、先行するトランザクションを暗黙的にコミットし、新しいトランザクションを開始します。次に、DDL 文の例をいくつか示します。

```
CREATE TABLE plants  
  (COMMON_NAME VARCHAR2 (15), LATIN_NAME VARCHAR2 (40));
```

```
DROP TABLE plants;
```

```
GRANT SELECT ON emp TO scott;
```

```
REVOKE DELETE ON emp FROM scott;
```

関連項目：

データベース・アクセスに対応する DDL 文の詳細は、次の章を参照してください。

- [第 26 章「データベース・アクセスの制御」](#)
- [第 27 章「権限、ロールおよびセキュリティ・ルール」](#)
- [第 28 章「監査」](#)

トランザクション制御文

トランザクション制御文は、DML 文による変更の内容を管理し、一連の DML 文をトランザクションとしてグループ化します。次のことを実行できます。

- トランザクションの変更を確定します（COMMIT）。
- トランザクションの開始以降、またはセーブポイント以降に実行されたトランザクション内での変更を取り消します（ROLLBACK）。
- ロールバックできるポイントを設定します（SAVEPOINT）。
- トランザクションのプロパティを設定します（SET TRANSACTION）。

セッション制御文

セッション制御文は、特定のユーザー・セッションのプロパティを管理します。たとえば、次の操作を実行できます。

- SQL トレース機能の使用可能および使用禁止の切換えなど、特化された機能を実行してカレント・セッションを変更します (ALTER SESSION)。
- カレント・セッションのロール (権限のグループ) を使用可能または使用禁止にします (SET ROLE)。

システム制御文

システム制御文は、Oracle Server インスタンスのプロパティを変更します。

システム制御文は、ALTER SYSTEM のみです。この文は、設定値 (共有サーバーの最小数など) の変更、セッションのキル (停止) およびその他の作業のために使用します。

埋込み SQL 文

埋込み SQL 文は、プロシージャ型言語プログラム内に DDL、DML およびトランザクション制御文を取り込みます。これらの文は、Oracle プリコンパイラで使用されます。埋込み SQL 文によって、次のことを実行できます。

- カーソルを定義し、割り当て、解放します (DECLARE CURSOR、OPEN、CLOSE)。
- データベースを指定し、Oracle に接続します (DECLARE DATABASE、CONNECT)。
- 変数名を割り当てます (DECLARE STATEMENT)。
- 記述子を初期化します (DESCRIBE)。
- エラー条件と警告の処理方法を指定します (WHENEVER)。
- SQL 文を解析して実行します (PREPARE、EXECUTE、EXECUTE IMMEDIATE)。
- データベースからデータを取り出します (FETCH)。

非標準 SQL の識別

Oracle には、整合性拡張を伴う標準 SQL データベース言語への拡張機能が用意されています。SQL に関する連邦情報処理規格 (FIPS 127-2) によれば、ベンダーはこの拡張機能を使用する SQL 文を識別する手段を提供する必要があります。Oracle 拡張機能は、対話型 SQL、Oracle プリコンパイラまたは SQL*Module で FIPS フラガーを使用して「フラグを立てる」ことによって識別できます。

SQL の他の処理系へのアプリケーションの移植性に関心がある場合には、FIPS フラガーを使用します。

関連項目：

- 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』
- 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』
- 『Programmer's Guide to SQL*Module for Ada』

再帰 SQL

DDL 文が発行されると、Oracle はデータ・ディクショナリ情報を修正する「再帰 SQL 文」を暗黙的に発行します。ユーザーは、Oracle が内部的に実行する再帰 SQL を考慮する必要はありません。

カーソル

カーソルとは、解析済の文とその文の処理に使用する、その他の情報が格納されるメモリー内の領域（プライベート SQL 領域）のハンドルまたは名前のことです。

ほとんどの 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、アプリケーションの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。アプリケーション開発の場合、カーソルはプログラムが使用できる名前付きのリソースです。特にアプリケーションに埋め込まれた SQL 文を解析するために使用できます。

ユーザー・セッションごとに、初期化パラメータ OPEN_CURSORS で設定された値を上限として、複数のカーソルをオープンできます。ただし、システム・メモリーを節約するには、不必要なカーソルをアプリケーション側でクローズする必要があります。カーソル数の制限のためにカーソルをオープンできない場合、データベース管理者は OPEN_CURSORS 初期化パラメータを変更できます。

一部の文（主として DDL 文）では、Oracle は暗黙的に再帰 SQL 文を発行する必要があります。その場合には「再帰カーソル」も必要になります。たとえば、CREATE TABLE 文を使用すると、新しい表と列を記録するために、各種データ・ディクショナリ表に多数の更新が加えられます。それらの再帰カーソルに対して「再帰コール」が出されます。1つのカーソルで複数の再帰コールが実行されることもあります。それらの再帰カーソルでは、「共有 SQL 領域」も使用します。

共有 SQL

複数のアプリケーションがデータベースに対して同じ SQL 文を送信すると、Oracle はそのことを自動的に認識します。その文が最初に出現したときの処理に使用された SQL 領域は「共有」されます。つまり、その後に同じ文が出現すると、それを処理するためにこの領域が使用されます。したがって、一意の文に対しては1つの共有 SQL 領域しか存在しません。共有 SQL 領域は共有メモリー領域であるため、どの Oracle プロセスも共有 SQL 領域を使用

できます。SQL 領域を共有することで、データベース・サーバーのメモリー使用量が節約され、システムのスループットが向上します。

文が同一であるかどうかを評価するときに、Oracle は、ユーザーとアプリケーションが直接発行した SQL 文と、DDL 文によって内部的に発行された再帰 SQL 文を評価します。

関連項目： 共有 SQL 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。

解析

「解析」は、SQL 文を処理するときの 1 つの段階です。アプリケーションが SQL 文を発行すると、アプリケーションは Oracle に解析コールを出します。解析コールでは、Oracle は次のことを実行します。

- 文の構文上および意味上の妥当性をチェックします。
- 文を発行したプロセスにその文を実行する権限があるかどうかを判断します。
- 文にプライベート SQL 領域を割り当てます。

また、Oracle は、ライブラリ・キャッシュにその文の解析済の表現を含んでいる既存の共有 SQL 領域が存在するかどうかも判別します。存在する場合、ユーザー・プロセスはこの解析済の表現を使用して、すぐにその文を実行します。存在しない場合、Oracle はその文の解析済の表現を生成し、ユーザー・プロセスは、ライブラリ・キャッシュの中にその文の共有 SQL 領域を割り当て、そこに解析済の表現を格納します。

アプリケーションが SQL 文の解析コールを出すことと、Oracle が実際にその文を解析することには、次のような違いがあります。アプリケーションが解析コールを出すと、SQL 文はプライベート SQL 領域に対応付けられます。この対応付けにより、アプリケーションが解析コールを出さなくても、その文を繰り返し実行できます。Oracle が「解析操作」を実行した場合は、SQL 文に共有 SQL 領域が割り当てられます。文に共有 SQL 領域が割り当てられた後は、再解析しなくてもその文を繰り返し実行できます。

解析コールと解析は、実行に比べてコストが高いため、できるだけ回数を少なくしてください。

これらの説明は、PL/SQL ブロックの解析と PL/SQL 領域の割当てにも当てはまります。ストアド・プロシージャ、ファンクションおよびパッケージとトリガーには、PL/SQL 領域が割り当てられます。Oracle は、PL/SQL ブロック内のそれぞれの SQL 文にも、共有 SQL 領域とプライベート SQL 領域を割り当てます。

関連項目： 15-16 ページの「[PL/SQL](#)」

SQL の処理

この項では、SQL 処理の基本について説明します。この章の内容は、次のとおりです。

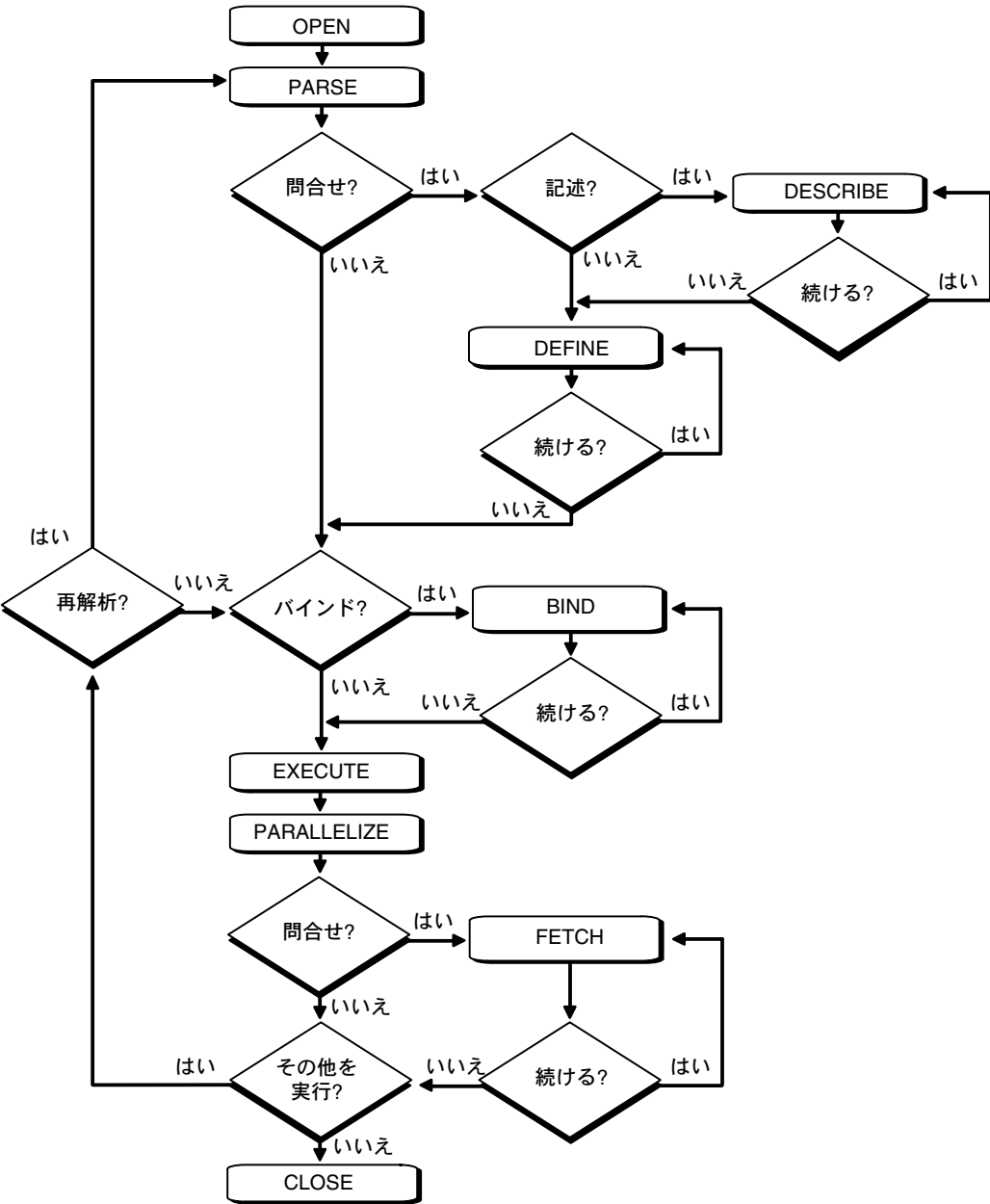
- [SQL 文の実行の概要](#)
- [DML 文の処理](#)
- [DDL 文の処理](#)
- [トランザクションの制御](#)

SQL 文の実行の概要

[図 15-1](#) に、SQL 文を処理して実行するための一般的な段階を示します。場合によっては、順序が少し異なることもあります。たとえば、「定義」の段階は、コーディングの仕方によっては「フェッチ」の直前にくることがあります。

多くの Oracle Tools では、これらの段階のいくつかが自動的に実行されます。ほとんどのユーザーには、ここまでの詳細は必要ありません。ただし、Oracle アプリケーションを作成する場合は、この情報が参考になることがあります。

図 15-1 SQL 文の処理の段階



DML 文の処理

ここでは、SQL 文の実行中の処理内容、特に DML 文処理の各段階で実行される内容について説明します。

Pro*C プログラムを使用して、ある部門の従業員全員の給与を増額する処理を実行するとします。現在使用しているプログラムから Oracle への接続は確立されており、EMP 表を更新するための適切なスキーマに接続しているとします。この場合、プログラムに次の SQL 文を埋め込むことができます。

```
EXEC SQL UPDATE emp SET sal = 1.10 * sal
      WHERE deptno = :dept_number;
```

DEPT_NUMBER は、部門番号の値を含むプログラム変数です。SQL 文の実行時には、アプリケーション・プログラムから提供される DEPT_NUMBER 値が使用されます。

各タイプの文の処理では、次の段階が必要です。

- 段階 1: カーソルの作成
- 段階 2: 文の解析
- 段階 5: 変数のバインド
- 段階 7: 文の実行
- 段階 9: カーソルのクローズ

オプションとして、次の段階を含めることもできます。

- 段階 6: 文の平行化

図 15-1 に示されているとおり、問合せ（SELECT）では、さらにいくつかの段階が必要です。

- 段階 3: 問合せの結果の記述
- 段階 4: 問合せの出力の定義
- 段階 8: 問合せの行のフェッチ
- 段階 9: カーソルのクローズ

関連項目： 15-13 ページの「問合せの処理」

段階 1: カーソルの作成

プログラム・インタフェース・コールによってカーソルが作成されます。カーソルは SQL 文からは独立した状態で任意の SQL 文を想定して作成されます。ほとんどのアプリケーションでは、カーソルは自動的に作成されます。ただし、プリコンパイラ・プログラムでは、暗黙的にカーソルが作成されたり、カーソル作成が明示的に宣言されることもあります。

段階 2: 文の解析

解析段階では、SQL 文がユーザー・プロセスから Oracle に渡され、解析済の表現が共有 SQL 領域にロードされます。文処理のこの段階では、多くのエラーを捕捉できます。

解析には、次のような処理が関係しています。

- SQL 文を変換し、その妥当性をチェックします。
- データ・ディクショナリを照合し、表と列の定義をチェックします。
- 必要なオブジェクトに解析ロックをかけて、文の解析中に定義が変更されないようにします。
- 参照先のスキーマ・オブジェクトへのアクセス権限をチェックします。
- 文の最適な実行計画を決定します。
- その実行計画を共有 SQL 領域にロードします。
- 分散型の文のすべてまたは一部を、参照データがあるリモート・ノードにルーティングします。

SQL 文が解析されるのは、同じ SQL 文の共有 SQL 領域が共有プールに存在しない場合のみです。その場合は、新しい共有 SQL 領域が割り当てられて、文が解析されます。

解析段階には、文の実行回数に関係なく 1 度だけ実行する必要がある処理要件があります。Oracle はそれぞれの SQL 文を 1 度だけ変換し、後でその文が参照されると、解析済文を再実行します。

SQL 文を解析するとその文の妥当性がチェックされますが、解析では**文を実行する前に**検出可能なエラーしか識別されません。したがって、解析で捕捉できないエラーもあります。たとえば、データ変換エラーまたはデータ・エラー（主キーに重複値を入力しようとした場合など）およびデッドロックは、実行段階に入ってからでなければ検出も報告もされません。

関連項目： 共有 SQL 領域の詳細は、15-7 ページの「[共有 SQL](#)」を参照してください。

問合せの処理

問合せは、正常に実行された場合に結果としてデータを戻すという点で、他のタイプの SQL 文とは異なります。他の文は単に成功か失敗かを戻すのみですが、問合せは 1 行または数千もの行を戻します。問合せの結果は、**常に表形式です**。結果の行は、1 行ごとに、またはグループ単位で「フェッチ」され（取り出され）ます。

問合せ処理のみに関連する問題がいくつかあります。明示的な SELECT 文のみでなく、他の SQL 文に含まれる暗黙の問合せ（副問合せ）もあります。たとえば、次のそれぞれの文では、実行の一部として問合せが必要になります。

```
INSERT INTO table SELECT...
```

```
UPDATE table SET x = y WHERE...
```

```
DELETE FROM table WHERE...
```

```
CREATE table AS SELECT...
```

問合せには、特に次のような特徴があります。

- 「読取り一貫性」が必要です。
- 中間処理に一時セグメントを使用できます。
- SQL 文処理の記述、定義およびフェッチ段階が必要になることがあります。

段階 3: 問合せの結果の記述

記述段階が必要なのは、問合せがユーザーにより対話式に入力された場合など、問合せ結果の特性が不明な場合のみです。

この場合は、記述段階で問合せ結果の特性（データ型、長さおよび名前）がわかります。

段階 4: 問合せの出力の定義

問合せの定義段階では、フェッチされた各値を受け取るために定義された変数の位置、サイズおよびデータ型を指定します。Oracle は、必要に応じてデータ型変換を実行します。

段階 5: 変数のバインド

この時点で、Oracle は SQL 文の意味を認識していますが、文を実行するための情報がまだ不足しています。Oracle は、文に含まれている変数の値を必要とします。例では、DEPT_NUMBER の値が必要です。これらの値を取得する処理のことを、「変数のバインド」と呼びます。

プログラムでは、値を検出できる位置（メモリー・アドレス）を指定する必要があります。Oracle ユーティリティはアプリケーションのエンド・ユーザーに新しい値の入力を要求するプロンプトを表示するのみであるため、エンド・ユーザーはバインド変数を指定しているということを認識していない可能性があります。

位置を指定（参照によるバインド）すると、再実行の前に変数を再バインドする必要はありません。変数の値は変更可能です。Oracle は、実行のたびに、メモリー・アドレスを使用して変数の値を調べます。

Oracle でデータ型変換を実行する必要がある場合は、暗黙的にまたはデフォルトで指定されていない限り、それぞれの値のデータ型と長さも指定する必要があります。

関連項目：

値のデータ型と長さを指定する方法の詳細は、次のマニュアルを参照してください。

- 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』
- 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』
（「動的 SQL メソッド 4」を参照）
- 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』
（「動的 SQL メソッド 4」を参照）

段階 6: 文のパラレル化

Oracle では、問合せ（SELECT）、INSERT、UPDATE、DELETE および特定の DDL 操作（索引作成、副問合せでの表の作成、およびパーティション上での操作など）をパラレル化できます。パラレル化すると、複数のサーバー・プロセスが SQL 文の処理を実行するため、処理を高速に完了できます。

関連項目： パラレル SQL の詳細は、[第 23 章「SQL 文のパラレル実行」](#)を参照してください。

段階 7: 文の実行

この時点で、Oracle には必要なすべての情報とリソースが揃ったため、文を実行できます。文が問合せや INSERT 文の場合は、変更されるデータがないため、行をロックする必要はありません。ただし、UPDATE 文または DELETE 文の場合、その文の影響を受けるすべての行は、トランザクションの COMMIT、ROLLBACK または SAVEPOINT が次に実行される時点まで、データベースの他のユーザーが使用できないようにロックされます。この処理により、データの整合性を確実に維持できます。

文によっては、実行回数を指定できる場合があります。このような処理を配列処理と呼びます。 n 回の実行回数が指定された場合、バインドと定義の位置はサイズ n の配列の開始点とみなされます。

段階 8: 問合せの行のフェッチ

フェッチ段階では、行が選択され、順序付け（問合せで要求された場合）されます。最後の行がフェッチされるまで、毎回のフェッチで結果の行が連続して取り出されます。

段階 9: カーソルのクローズ

SQL 文の処理の最後の段階は、カーソルのクローズです。

DDL 文の処理

DDL 文を正常実行するにはデータ・ディクショナリへの書込みアクセスが必要であるため、DDL 文の実行は DML 文や問合せの実行とは異なります。これらの文の解析（段階 2）には、実際には解析、データ・ディクショナリの参照および実行が含まれます。

トランザクション管理、セッション管理およびシステム管理の SQL 文は、解析および実行段階を使用して処理されます。それらを再実行するには、実行段階をもう一度実行します。

トランザクションの制御

一般に、1つのトランザクションとしてまとめるアクションのタイプに配慮するのは、Oracle へのプログラム・インタフェースを使用するアプリケーション設計者のみです。作業が論理的な単位として完遂され、データの一貫性が保たれるように、トランザクションは適切に定義する必要があります。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前と終了後に、すべての参照表のデータは必ず一貫した状態になっている必要があります。
- 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 文のみを含めます。

たとえば、口座間の振替操作（トランザクションまたは論理作業単位）の場合は、片方の口座からの引出し（1つの SQL 文）と、もう一方の口座への預入れ（1つの SQL 文）を含める必要があります。どちらのアクションも、1つの論理作業単位として一緒に失敗または一緒に成功する必要があります。貸方がないのに借方があることはありません。また、ある口座に新しく預金するなど、関連のないその他のアクションを、振替トランザクションに含めることはできません。

アプリケーションを設計するときは、トランザクションにどのタイプのアクションを含めるかのみでなく、短い非分散型のトランザクションのパフォーマンスを上げるために、BEGIN_DISCRETE_TRANSACTION プロシージャをいつ使用するかも判断する必要があります。

関連項目： 16-9 ページの「[ディスクリット・トランザクションの管理](#)」

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 を使用すると、SQL 文をプロシージャ型の言語構文と混合して使用できます。さらに PL/SQL では、プロシージャ、ファンクションおよびパッケージなどの PL/SQL プログラム・ユニットを定義して実行できます。

PL/SQL プログラム・ユニットは、一般に、無名ブロックとストアド・プロシージャに分類されます。

「無名ブロック」とは、アプリケーション内にあり、名前が付いていないか、データベースに格納されていない PL/SQL ブロックです。多くのアプリケーションでは、SQL 文を記述できるところであればどこにでも PL/SQL ブロックを記述できます。

「ストアド・プロシージャ」とは、Oracle によってデータベースに格納され、アプリケーションから名前でもコールできる PL/SQL ブロックのことです。ストアド・プロシージャを作成すると、Oracle はそのプロシージャを解析し、その解析済の表現をデータベースに格納します。さらに Oracle では、ファンクション（プロシージャによく似ているもの）やパッケージ（プロシージャとファンクションをまとめたもの）を作成して保管することもできます。

関連項目：

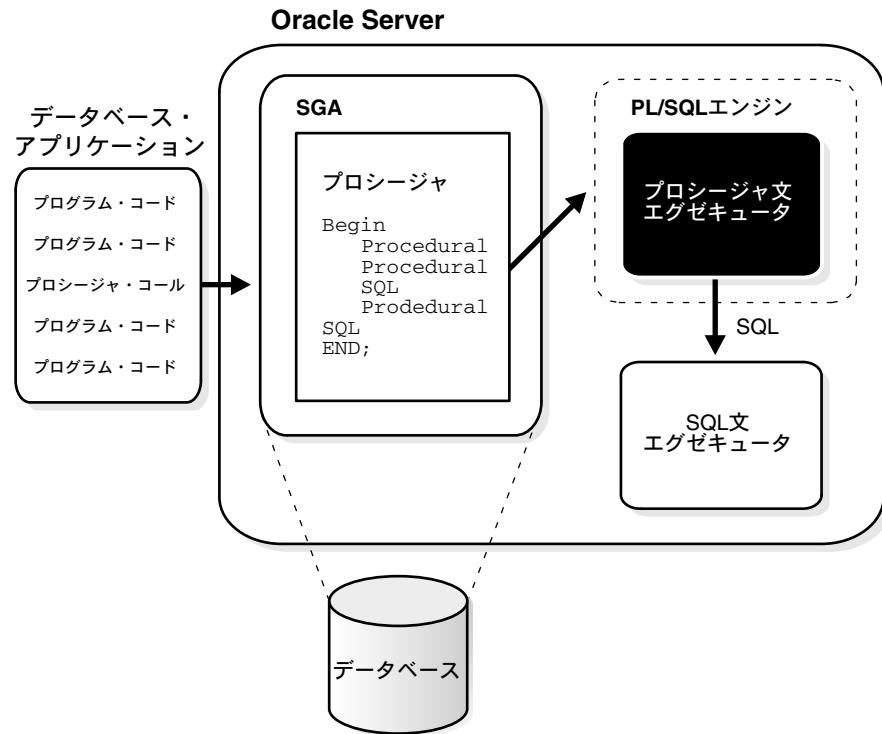
- [第 17 章「プロシージャとパッケージ」](#)
- [第 19 章「トリガー」](#)

PL/SQL が実行される方法

PL/SQL プログラム・ユニットを処理する「PL/SQL エンジン」は、Oracle Server をはじめとする多数の Oracle 製品に組み込まれている特別なコンポーネントです。

[図 15-2](#) に、Oracle Server に含まれている PL/SQL エンジンを示します。

図 15-2 PL/SQL エンジンと Oracle Server



プロシージャ（またはパッケージ）は、データベースに格納されます。アプリケーションがデータベースに格納されているプロシージャをコールすると、Oracle はコンパイル済のプロシージャ（またはパッケージ）をシステム・グローバル領域（SGA）の共有プールにロードし、PL/SQL 文エグゼキュータと SQL 文エグゼキュータが連動してプロシージャ内の文を処理します。

PL/SQL エンジンは、次の Oracle 製品に組み込まれています。

- Oracle Server
- Oracle Forms（バージョン 3 以降）
- SQL*Menu（バージョン 5 以降）
- Oracle Reports（バージョン 2 以降）
- Oracle Graphics（バージョン 2 以降）

別の PL/SQL ブロック（無名ブロックまたは別のストアド・プロシージャ）からストアド・プロシージャをコールすることもできます。たとえば、Oracle Forms（バージョン 3 以降）からストアド・プロシージャをコールできます。

また、無名ブロックを、次のツールで開発したアプリケーションから Oracle に渡すこともできます。

- Oracle プリコンパイラ（ユーザー・イグジットを含む）
- Oracle コール・インタフェース（OCI）
- SQL*Plus
- Server Manager
- Oracle Enterprise Manager

PL/SQL の言語構文

PL/SQL ブロックには、次の PL/SQL 言語構文を組み込むことができます。

- 変数と定数
- カーソル
- 例外

この項では、それぞれの要素について概説します。

関連項目：『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

変数と定数

プロシージャ、ファンクションまたはパッケージの中では、変数および定数を宣言できます。変数や定数は、必要になった時点で値を受け渡すために SQL 文や PL/SQL 文で使用できます。

注意： SQL*Plus などの対話形式のツールを使用すると、カレント・セッションで変数を定義できます。この方法で宣言した変数は、プロシージャやパッケージの中で宣言した変数と同じように使用できます。

カーソル

「カーソル」は、Oracle データのレコード単位での処理を容易にするために、プロシージャ、ファンクションまたはパッケージの中で明示的に宣言できます。また、カーソルは、PL/SQL エンジンによって（他のデータ操作アクションをサポートするため）暗黙的に宣言されることもあります。

例外

PL/SQL では、PL/SQL コードの処理中に発生する、「例外」と呼ばれる内部的なエラー条件とユーザー定義のエラー条件を明示的に処理できます。内部例外は、0 による除算などの不正な操作や、PL/SQL に戻された Oracle エラーが原因で発生します。ユーザー定義例外は、アプリケーション固有のエラー（借方に記入するだけで、差引勘定を負のままにするなど）の処理を制御するために、PL/SQL ブロック内で明示的に定義され、通知されます。

例外状況が発生する（通知される）と、通常の PL/SQL コードの実行は停止され、例外ハンドラというルーチンが起動します。それぞれの内部例外やユーザー定義例外を処理するための特定の例外ハンドラを作成できます。

ストアド・プロシージャ

Oracle では、ストアド・プロシージャを作成してコールすることもできます。アプリケーションがストアド・プロシージャをコールすると、そのプロシージャの解析済の表現がデータベースから検索され、Oracle の PL/SQL エンジンによって処理されます。

注意： 多くの Oracle 製品に PL/SQL コンポーネントが含まれていますが、このマニュアルでは、Oracle データベースの中に保存でき、Oracle Server の PL/SQL エンジンを使用して処理できるプロシージャとパッケージのみを対象にしています。

ストアド・プロシージャは、次のツールを使用して開発したアプリケーションからコールできます。

- Oracle プリコンパイラ（ユーザー・イグジットを含む）
- Oracle コール・インタフェース（OCI）
- SQL*Module
- SQL*Plus
- Server Manager
- Oracle Enterprise Manager

別の PL/SQL ブロック（無名ブロックまたは別のストアド・プロシージャ）からストアド・プロシージャをコールすることもできます。

関連項目：

- ストアド・プロシージャを他の PL/SQL ブロックからコールする方法の詳細は、[第 17 章「プロシージャとパッケージ」](#)を参照してください。
- C または C++ のストアド・プロシージャをコールする方法の詳細は、『Oracle8i 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。
- COBOL のストアド・プロシージャをコールする方法の詳細は、『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照してください。
- 各種アプリケーションのストアド・プロシージャをコールする方法の詳細は、個々のプログラマーズ・ガイドを参照してください。

PL/SQL の動的 SQL

PL/SQL では、完全なテキストが実行時まで認識されない「動的 SQL 文」を実行できます。動的 SQL 文は、実行時に、プログラムに入力されたり、またはプログラムにより作成される文字列に格納されます。これにより、汎用プロシージャを作成できます。たとえば、動的 SQL を使用すると、実行時までには名前がわからない表を操作するプロシージャを作成できます。

動的 SQL を含むストアド・プロシージャと無名 PL/SQL ブロックを記述するには、次の 2 つの方法があります。

- 動的 SQL 文を PL/SQL ブロックに埋め込みます。
- DBMS_SQL パッケージを使用します。

また、動的 SQL を使用して、データ操作言語（DML）またはデータ定義言語（DDL）の文を発行できます。この方法は、PL/SQL に DDL 文を静的に埋め込むことができないという問題を解決するために使用できます。たとえば、EXECUTE IMMEDIATE 文または DBMS_SQL パッケージによって提供される PARSE プロシージャを使用して、ストアド・プロシージャから DROP TABLE 文を発行できるようになります。

関連項目：

- 動的 SQL のための 2 つのアプローチの比較については、『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。
- 動的 SQL の詳細は、『Oracle8i 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。

外部プロシージャ

Oracle Server 上で実行される PL/SQL プロシージャからは、C プログラミング言語で作成して共有ライブラリに格納した外部プロシージャや外部ファンクションをコールできます。C ルーチンは、Oracle Server とは別のアドレス空間で実行されます。

関連項目： 外部プロシージャと Inter-Language Method Services 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。

PL/SQL Server Pages

PL/SQL Server Pages (PSP) は、特殊なタグにより PL/SQL スクリプトが埋め込まれているサーバー側の web ページ (HTML または XML 形式) です。インターネットの発展により、豊富な最新情報に対するニーズが高まっています。動的な web ページを生成するために、通常、開発者は C または Perl を使用して、同じプログラム内でデータをフェッチし、web ページ全体を生成する CGI プログラムを記述しています。この種の動的ページの開発とメンテナンスは、コストと時間がかかる作業です。

スクリプト化により、動的な web ページの迅速な開発というニーズに対処できます。HTML ページには、元の HTML の可読性を変えずに小型スクリプトを埋め込むことができます。スクリプトには、HTML ページの動的部分を生成し、ユーザーが要求したときに実行される論理が含まれています。

スクリプトの習得や使用は容易です。HTML の内容をアプリケーションの論理から切り離すことにより、スクリプト・ページの開発、デバッグおよびメンテナンスが容易になります。開発モデルが簡素化されており、通常のスクリプト作成言語はプログラミング・スキルをあまり必要としないため、web ページの作成者が動的 web ページを開発できます。

HTML ページに埋め込まれるスクリプトには、クライアント側スクリプトとサーバー側スクリプトの 2 種類があります。「クライアント側スクリプト」は、HTML ページの一部として戻され、ブラウザ内で実行されます。このスクリプトは、主としてクライアント側で HTML ページのナビゲーションやデータの妥当性チェックに使用されます。「サーバー側スクリプト」も HTML ページに埋め込まれますが、サーバー側で実行されます。このスクリプトは、データをフェッチして操作し、ページの一部として戻される HTML の内容を生成します。PSP スクリプトは、サーバー側スクリプトです。

PL/SQL ゲートウェイは、HTTP クライアントから HTTP 要求を受信し、URL で指定された PL/SQL ストアド・プロシージャをコールし、HTTP 出力をクライアントに戻します。PL/SQL Server Pages は、PSP コンパイラによって PL/SQL ストアド・プロシージャにコンパイルされます。ゲートウェイによりプロシージャが実行されると、動的な内容を持つ web ページが生成されます。PSP は、次の 2 つの既存の PL/SQL ゲートウェイのどちらかと共に使用します。

- Oracle Application Server の PL/SQL Cartridge
- WebDB

関連項目： PL/SQL Server Pages 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。

トランザクションの管理

この章では、トランザクションを定義し、トランザクションを使用して作業を管理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。

- トランザクションの概要
- Oracle とトランザクションの管理
- ディスクリット・トランザクションの管理
- 自律型トランザクション

トランザクションの概要

「トランザクション」は、1つ以上の SQL 文を含む論理作業単位です。トランザクションはアトミック（基本）な単位です。つまり、トランザクション内のすべての SQL 文は、すべて「コミット」（データベースに適用）されるか、すべて「ロールバック」（データベースから取消し）されるかのどちらかになります。

トランザクションは、最初の実行可能な SQL 文から開始します。トランザクションは、COMMIT 文または ROLLBACK 文を使用して明示的に、または DDL 文を発行して暗黙的にコミットまたはロールバックされた時点で終了します。

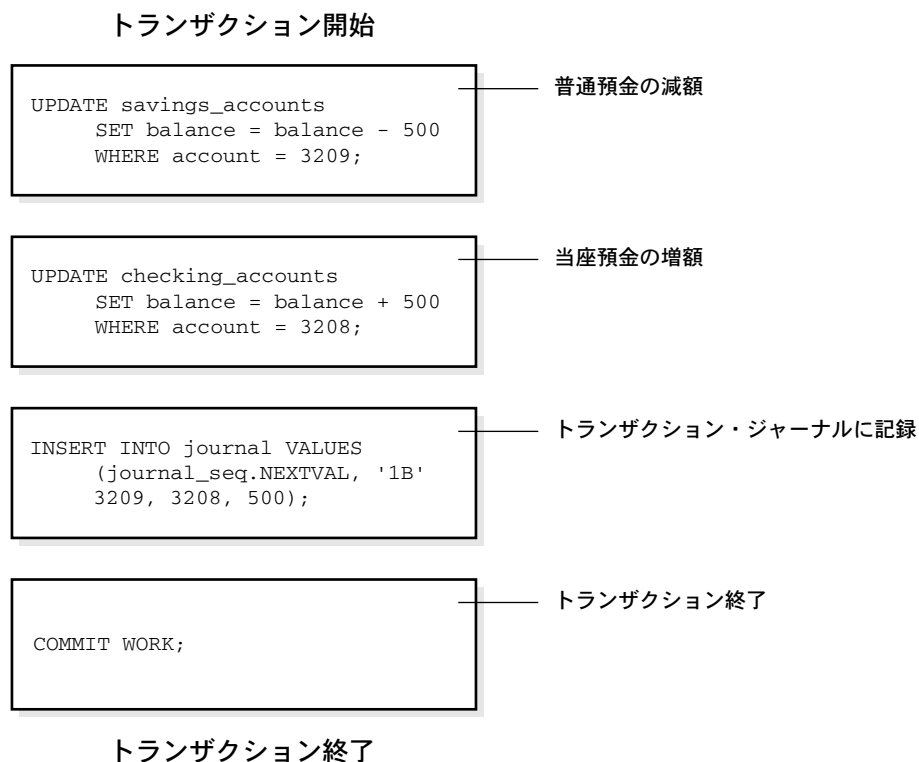
トランザクションの概念を具体的に説明するため、銀行業務データベースについて考えてみます。銀行の顧客が普通預金口座から当座預金口座へ預金を振り替える場合、トランザクションは次の3つの別々の操作で構成されます。

- 普通預金口座の減額
- 当座預金口座の増額
- トランザクション・ジャーナルへのトランザクションの記録

Oracle では、2つの状況が考慮されます。3つの SQL 文がすべて実行されて、口座の差引残高の帳尻が合えば、トランザクションの結果をデータベースに適用できます。ただし、預金額の不足、口座番号が無効、またはハードウェア障害などの問題が原因で、トランザクション内の1つまたは2つの文が完了しない場合は、すべての口座の差引残高が正しく維持されるようにトランザクション全体をロールバックする必要があります。

図 16-1 に銀行のトランザクションの例を示します。

図 16-1 銀行のトランザクション



文の実行とトランザクションの制御

正常に実行される SQL 文とコミットされるトランザクションとは異なります。

「正常に実行される」とは、単一の文が次の条件を満たしたことを意味します。

- 解析されたこと。
- 有効な SQL 構文であることが確認されたこと。
- アトミックな単位としてエラーなしで実行されたこと。たとえば、複数行の更新ですべての行が変更された場合などです。

ただし、その文が含まれているトランザクションがコミットされるまでは、トランザクションをロールバックしてその文のすべての変更を取消しできます。正常に実行される（成功する）という表現は、トランザクションではなく、文に対して使用します。

「コミット」とは、ユーザーが明示的または暗黙的に、このトランザクションの変更内容を確定するように要求したことを意味します。明示的な要求とは、ユーザーが **COMMIT** 文を発行することを意味します。暗黙的な要求は、アプリケーションの正常終了を介して、またはデータ定義言語などを介して出すことができます。トランザクションがコミットされて初めて、トランザクションの SQL 文による変更内容が確定され、他のユーザーがこれらの変更を参照できるようになります。このトランザクションよりも後に開始したトランザクションのみが、コミットされた変更を参照できます。

文レベルのロールバック

実行中に SQL 文のエラーが発生すると、その文のすべての効果はロールバックされます。ロールバックの効果は、その文がまったく実行されなかった場合と同じ状態にすることです。この操作が「文レベルのロールバック」です。

SQL 文の実行中にエラーが検出されると、文レベルのロールバックが発生します。この種のエラーの一例として、主キーに重複値を挿入しようとした場合があります。解析中に構文エラーなどのエラーが検出された場合、まだ SQL 文は実行されていないため、文レベルのロールバックは発生しません。1 つのデッドロック（同じデータに関する競合）に単一 SQL 文が複数関係している場合も、文レベルのロールバックが発生します。

ある SQL 文が失敗しても、その損失はその文自体が実行した作業にしか及ばず、カレント・トランザクションの中でその文より前に実行された作業は無駄になりません。文が DDL 文の場合、その文の直前に実行された暗黙のコミットは取り消されません。

文レベルのロールバックを要求するには、**ROLLBACK** 文を発行する方法もあります。

注意： ユーザーは、ロールバック文の中で暗黙のセーブポイントを直接参照できません。

関連項目： 24-17 ページの「**デッドロック**」

Oracle とトランザクションの管理

Oracle でのトランザクションは、最初の実行可能 SQL 文が検出された時点で開始されます。「実行可能 SQL 文」は、DML 文や DDL 文など、インスタンスへのコールを生成する SQL 文です。

トランザクションが開始されると、Oracle はそのトランザクションを使用可能なロールバック・セグメントに割り当てて、新しいトランザクションのロールバック・エントリを記録します。

トランザクションは、次のいずれかの状況が発生すると終了します。

- COMMIT 文または ROLLBACK 文が、SAVEPOINT 句なしで発行される場合。
- CREATE、DROP、RENAME、ALTER などの DDL 文が実行される場合。カレント・トランザクションに DML 文が含まれている場合、Oracle は最初にそのトランザクションをコミットし、新しい単一文トランザクションとしてその DDL 文を実行し、コミットします。
- ユーザーが Oracle の接続を切断する場合。カレント・トランザクションはコミットされます。
- ユーザー・プロセスが異常終了する場合。カレント・トランザクションはロールバックされます。

1 つのトランザクションが終了すると、次の実行可能 SQL 文によって次のトランザクションが自動的に開始されます。

注意： アプリケーションでは、プログラムを終了する前に、必ず明示的にトランザクションをコミットまたはロールバックする必要があります。

関連項目： 4-20 ページの「[トランザクションとロールバック・セグメント](#)」

トランザクションのコミット

トランザクションを「コミット」するとは、トランザクション内の SQL 文によって実行された変更を確定する操作のことです。

データを修正したトランザクションがコミットされる前に、次の処理が完了しているはずで

- Oracle により、システム・グローバル領域 (SGA) のロールバック・セグメント・バッファの中にロールバック・セグメント・レコードが生成されます。このロールバック情報には、トランザクションの SQL 文によって変更された古いデータ値が入れられます。
- Oracle により、SGA の REDO ログ・バッファに REDO ログ・エントリが生成されます。REDO ログ・レコードには、データ・ブロックおよびロールバック・ブロックに対する

変更内容が含まれています。これらの変更は、トランザクションがコミットされる前にディスクに移動することがあります。

- SGA のデータベース・バッファに変更が加えられます。これらの変更は、トランザクションが実際にコミットされる前にディスクに移動することがあります。

注意： コミットされたトランザクションのデータ変更は、SGA のデータベース・バッファに格納されており、必ずしも即座にデータベース・ライター (DBW n) のバックグラウンド・プロセスによってデータ・ファイルに書き込まれるわけではありません。データベースにとって最も効率的な時点で書き込まれます。この書込みは、トランザクションがコミットされる前に実行されたり、トランザクションがコミットされた後しばらくしてから実行されます。

トランザクションをコミットすると、次の処理が実行されます。

1. 対応付けられたロールバック・セグメントの内部トランザクション表にトランザクションがコミットされたことが記録され、トランザクションの対応する一意のシステム変更番号 (SCN) が割り当てられ、その表に記録されます。
2. ログ・ライター (LGWR) プロセスにより、SGA の REDO ログ・バッファの REDO ログ・エントリがオンライン REDO ログ・ファイルに書き込まれます。また、トランザクションの SCN も、LGWR によってオンライン REDO ログ・ファイルに書き込まれます。これが、トランザクションのコミットを構成するアトミック・イベントです。
3. Oracle により、行と表に対して保持されているロックが解放されます。
4. Oracle により、トランザクションに完了のマークが付けられます。

関連項目：

- 24-3 ページの「[ロックのメカニズム](#)」
- ロールバック・セグメントを変更するバックグラウンド・プロセス LGWR および DBW n の詳細は、8-5 ページの「[Oracle プロセス](#)」を参照してください。

トランザクションのロールバック

「ロールバックする」とは、コミットされていないトランザクション内の SQL 文によって実行されたデータ変更を取り消すことを意味します。Oracle は、ロールバック・セグメントを使用して古い値を格納します。REDO ログには、変更の履歴が記録されます。

Oracle では、コミットされていないトランザクション全体をロールバックできます。また、コミットされていないトランザクションの後半部分をセーブポイントと呼ばれるマーカーまでロールバックすることもできます。

次のすべてのタイプのロールバックで、同じ手順が使用されます。

- 文レベルのロールバック（文またはデッドロックの実行エラーによる）
- セーブポイントへのロールバック
- ユーザー要求によるトランザクションのロールバック
- プロセスの異常終了によるトランザクションのロールバック
- インスタンスが異常終了したときの、すべての保留中のトランザクションのロールバック
- リカバリのときの、不完全なトランザクションのロールバック

セーブポイントを参照しないで、**トランザクション全体**をロールバックした場合、次の処理が実行されます。

1. Oracle により、トランザクションのすべての SQL 文によるすべての変更が、対応するロールバック・セグメントを使用して取り消されます。
2. Oracle により、そのトランザクションのすべてのデータ・ロックが解放されます。
3. トランザクションが終了します。

関連項目：

- 16-7 ページの「[セーブポイント](#)」
- 24-3 ページの「[ロックのメカニズム](#)」
- リカバリ中にコミット済かどうかを問わず変更に対して実行される処理の詳細は、『Oracle8i Recovery Manager 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。

セーブポイント

トランザクションのコンテキスト内で、「セーブポイント」と呼ばれる中間マーカーを宣言できます。セーブポイントは、長いトランザクションをいくつかの小さい部分に分割します。

セーブポイントを使用すると、長いトランザクション内の任意のポイントで作業に任意にマークを設定できます。これにより、そのトランザクション内の宣言されたセーブポイントからトランザクション内の現時点までの間に実行された作業を、後でロールバックできるようになります。たとえば、大規模で複雑な一連の更新でセーブポイントを使用すると、エラーが発生してもすべての文を再送信する必要がなくなります。

セーブポイントは、アプリケーション・プログラム内でも使用できます。プロシージャにいくつかのファンクションが含まれている場合は、それぞれのファンクションの開始前にセーブポイントを作成できます。そうすれば、あるファンクションが失敗しても、そのファンクション開始前の状態にデータを復帰し、パラメータを修正してから再実行したり、リカバリ作業を実行するのが容易になります。

セーブポイントまでロールバックすると、Oracle はロールバック文が取得したデータ・ロックを解放します。以前にロックされていたリソースを待機していた他のトランザクションは、続行できます。以前にロックされていた行を更新しようとする他のトランザクションは、それらの行を更新できます。

トランザクションを**セーブポイントまで**ロールバックした場合、次の処理が実行されます。

1. Oracle により、セーブポイントの後に実行された文のみがロールバックされます。
2. 指定されたセーブポイントは保持されますが、そのセーブポイントより後に設定されたセーブポイントはすべて失われます。
3. Oracle により、そのセーブポイントの後に取得された表と行のすべてのロックが解放されますが、そのセーブポイントより前に取得されたすべてのデータ・ロックは保持されます。

トランザクションはアクティブであり、継続できます。

2 フェーズ・コミット・メカニズム

分散データベースでは、ネットワーク全体でトランザクション制御を調整し、ネットワーク障害やシステム障害が発生した場合にもデータの一貫性が保たれるようにする必要があります。

「分散トランザクション」とは、分散データベースにおいて複数の異なるノード上でデータを更新する、1 つ以上の文を含むトランザクションのことです。

「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に関係するすべてのデータベース・サーバーが、そのトランザクション内の文のすべてをコミットするか、またはすべてをロールバックするかのどちらか一方のみになるように保証するものです。また、2 フェーズ・コミット・メカニズムは、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙の DML 操作も保護します。

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からはまったく意識されません。事実、ユーザーは、トランザクションが分散していることも認識する必要はありません。トランザクションの終了を示す COMMIT 文があると、トランザクションをコミットするための 2 フェーズ・コミット・メカニズムが自動的に起動されます。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構文は必要ありません。

リカバラ (RECO) 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 プロセスが、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自動的にコミットするか、またはロールバックします。

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、またはロールバックできます。このオプションによって、ローカルのデータベース管理者は、長時間にわたる障害の結果として無期限にロックされる可能性のあるリソースを解放できます。

あるデータベースを過去のある時点までリカバリする必要がある場合、Oracle のリカバリ機能を使用すると、他のサイトのデータベース管理者は、自分のデータベースも過去のその同じ時点に戻すことができます。この操作により、グローバル・データベースは一貫した状態に保たれます。

関連項目：

- 30-35 ページの「[分散トランザクション](#)」
- 『Oracle8i 分散システム』

ディスクリート・トランザクションの管理

アプリケーション開発者は、BEGIN_DISCRETE_TRANSACTION プロシージャを使用して、小規模な非分散型トランザクションのパフォーマンスを向上させることができます。このプロシージャにより、トランザクション処理の効率が改善されるため、小規模なトランザクションをより高速に実行できるようになります。

ディスクリート・トランザクションでは、すべてのデータ変更がトランザクションのコミット時まで遅延されます。当然ながら、同時に実行されているその他のトランザクションは、ディスクリート・トランザクションであるかどうかにかかわらず、コミットされていない変更の参照はできません。

ディスクリート・トランザクション中には、次のイベントが発生します。

1. REDO 情報が生成され、メモリー内の別個の位置に格納されます。
2. トランザクションがコミット要求を発行すると、REDO 情報が他のグループ・コミットと一緒に REDO ログ・ファイルに書き込まれます。
3. データベース・ブロックに対する変更がそのブロックに直接適用されます。
4. コミットが完了すると、アプリケーションに制御が戻されます。

このトランザクション設計により、トランザクションがコミットされるまでブロックは実際には修正されず、REDO 情報は REDO ログ・バッファに格納されるため、取消し情報を生成する必要がなくなります。

常に REDO が生成されるディスクリート・トランザクションと、ダイレクト・パス操作にのみ適用される NOLOGGING モードとの間には、相互作用はありません。したがって、ディスクリート・トランザクションは、NOLOGGING 属性が設定されている表に対して発行できます。

関連項目：

- 22-5 ページの「[ロギング・モード](#)」
- ディスクリット・トランザクションの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- BEGIN_DISCRETE_TRANSACTION プロシージャの詳細は、『Oracle8i PL/SQL 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。

自律型トランザクション

自律型トランザクションは、他のトランザクションからコールできる、独立したトランザクションです。自律型トランザクションにより、コール側トランザクションのコンテキストから出て、なんらかの SQL 操作を実行し、その操作をコミットまたはロールバックしてから、コール側トランザクションのコンテキストに戻って引き続き実行できます。

起動後の自律型トランザクションは、コール側のメイン・トランザクションの影響をまったく受けません。メイン・トランザクションによって加えられたがコミットされていない変更は取り扱わず、ロックやリソースをメイン・トランザクションと共有することもあります。自律型トランザクションによって加えられた変更は、自律型トランザクションのコミット時に他のトランザクションから見えるようになります。

自律型トランザクションは、相互にコールし合うことができます。自律型トランザクションをコールできるレベル数には、リソース制限以外の制限はありません。

自律型トランザクションとコール側トランザクションの間で、デッドロックが発生することがあります。Oracle は、この種のデッドロックを検出するとエラーを戻します。アプリケーション開発者は、デッドロック状況を回避する責任があります。

自律型トランザクションは、トランザクションのロギングや再試行カウンタなど、コール側トランザクションでコミットまたはロールバックするかどうかに関係なく、独立して実行する必要があるアクションを実装する場合に便利です。

自律型 PL/SQL ブロック

PL/SQL ブロックから自律型トランザクションをコールできます。プラグマ AUTONOMOUS_TRANSACTION を使用してください。プラグマは、コンパイラ・ディレクティブです。次の種類の PL/SQL ブロックを、自律型として宣言できます。

- ストアド・プロシージャまたはファンクション
- ローカル・プロシージャまたはファンクション
- パッケージ
- 型メソッド
- 最上位レベルの無名ブロック

自律型 PL/SQL ブロックに入ると、コール側のトランザクション・コンテキストは中断されます。この操作により、このブロック（または、そこからコールされる他のブロック）で実行される SQL 操作は、コール側のトランザクション・コンテキストの状態に依存も影響もしないことが保証されます。

自律型ブロックが他の自律型ブロックまたはそれ自身を起動する場合、コールされたブロックが、コール側ブロックとトランザクション・コンテキストを共有することはありません。ただし、自立型ブロックが自律型でないブロック（つまり、自律型として宣言されていないブロック）を起動する場合、コールされたブロックはコール側の自律型ブロックのトランザクション・コンテキストを継承します。

関連項目：『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

自律型ブロック内のトランザクション制御文

自律型 PL/SQL ブロック内のトランザクション制御文は、現在アクティブになっている自律型トランザクションにのみ適用されます。この種の文の例は、次のとおりです。

- SET TRANSACTION
- COMMIT
- ROLLBACK
- SAVEPOINT
- ROLLBACK TO SAVEPOINT

同様に、メイン・トランザクション内のトランザクション制御文は、そのトランザクションにのみ適用され、コールされる自律型トランザクションには適用されません。たとえば、メイン・トランザクションを自律型トランザクションの開始前までロールバックしても、自律型トランザクションはロールバックされません。

関連項目：『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

プロシージャとパッケージ

この章では、Oracle のプロシージャ機能について説明します。この章の内容は次のとおりです。

- [ストアド・プロシージャとパッケージの概要](#)
- [プロシージャとファンクション](#)
- [パッケージ](#)
- [Oracle がプロシージャとパッケージを格納する方法](#)
- [Oracle がプロシージャとパッケージを実行する方法](#)

関連項目： プロシージャ、ファンクションおよびパッケージの間の依存性と、Oracle がそれらの依存性を管理する方法の詳細は、[第 20 章「Oracle の依存性の管理」](#)を参照してください。

ストアド・プロシージャとパッケージの概要

Oracle では、「PL/SQL プログラム・ユニット」というプロシージャ型スキーマ・オブジェクトを使用してデータベース情報にアクセスしたり処理できます。プロシージャ、ファンクションおよびパッケージは、すべて PL/SQL プログラム・ユニットの例です。

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、フロー制御と、複雑なプログラムの記述を可能にする他の文を装備して、SQL を拡張したものです。「PL/SQL エンジン」は、PL/SQL プログラム・ユニットの定義、コンパイルおよび実行に使用するツールです。このエンジンは、Oracle Server をはじめとする多数の Oracle 製品に組み込まれている特別なコンポーネントです。

多くの Oracle 製品に PL/SQL コンポーネントが含まれていますが、この章では、Oracle データベースに格納でき、Oracle Server の PL/SQL エンジンを使用して処理できるプロシージャとパッケージを対象にして説明します。それぞれの Oracle Tool の PL/SQL 機能については、該当する Oracle Tool のマニュアルに説明があります。

関連項目： 15-16 ページの「[PL/SQL](#)」

ストアド・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は、特定の作業を実行するための SQL および PL/SQL プログラミング言語文の集合を論理的にグループ化したスキーマ・オブジェクトです。プロシージャとファンクションは、ユーザーのスキーマ内に作成され、繰り返し使用するためにデータベースに格納されます。プロシージャまたはファンクションは、次の方法で対話型で実行できます。

- SQL*Plus などの Oracle Tool を使用します。
- Oracle Forms やプリコンパイラのアプリケーションなど、データベース・アプリケーションのコード内で明示的にコールします。
- 別のプロシージャやトリガーのコード内で明示的にコールします。

[図 17-1](#) に、データベースに格納され、いくつかの異なるデータベース・アプリケーションによってコールされる単純なプロシージャを示します。

プロシージャとファンクションはほとんど同じものですが、プロシージャはコール側に値を戻さないのに対して、ファンクションは必ず 1 つの値を戻すという違いがあります。簡単にするため、この章では「プロシージャ」という語で「プロシージャとファンクション」の両方を指すものとします。

図 17-1 ストアド・プロシージャ

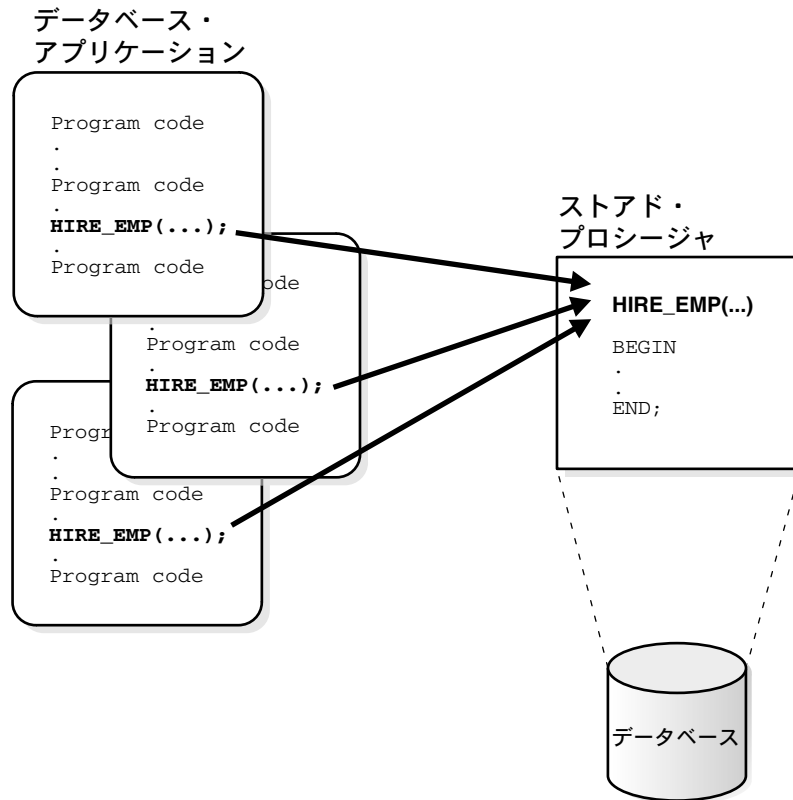


図 17-1 のストアド・プロシージャは、従業員レコードを EMP 表に挿入します。このプロシージャを図 17-2 に示します。

図 17-2 HIRE_EMP プロシージャ

```
Procedure HIRE_EMP (name VARCHAR2, job VARCHAR2,  
mgr NUMBER, hiredate DATE, sal NUMBER,  
comm NUMBER, deptno NUMBER)
```

```
BEGIN  
.  
.  
INSERT INTO emp VALUES  
    (emp_sequence.NEXTVAL, name, job, mgr  
    hiredate, sal, comm, deptno);  
.  
.  
END;
```

図 17-1 のデータベース・アプリケーションはすべて、HIRE_EMP プロシージャをコールしています。また、権限を付与されたユーザーは、Oracle Enterprise Manager または SQL*Plus を使用して、次の文で HIRE_EMP プロシージャを実行できます。

```
EXECUTE hire_emp ('TSMITH', 'CLERK', 1037, SYSDATE, \  
                  500, NULL, 20);
```

この文により、EMP 表に TSMITH のための新しい従業員レコードが挿入されます。

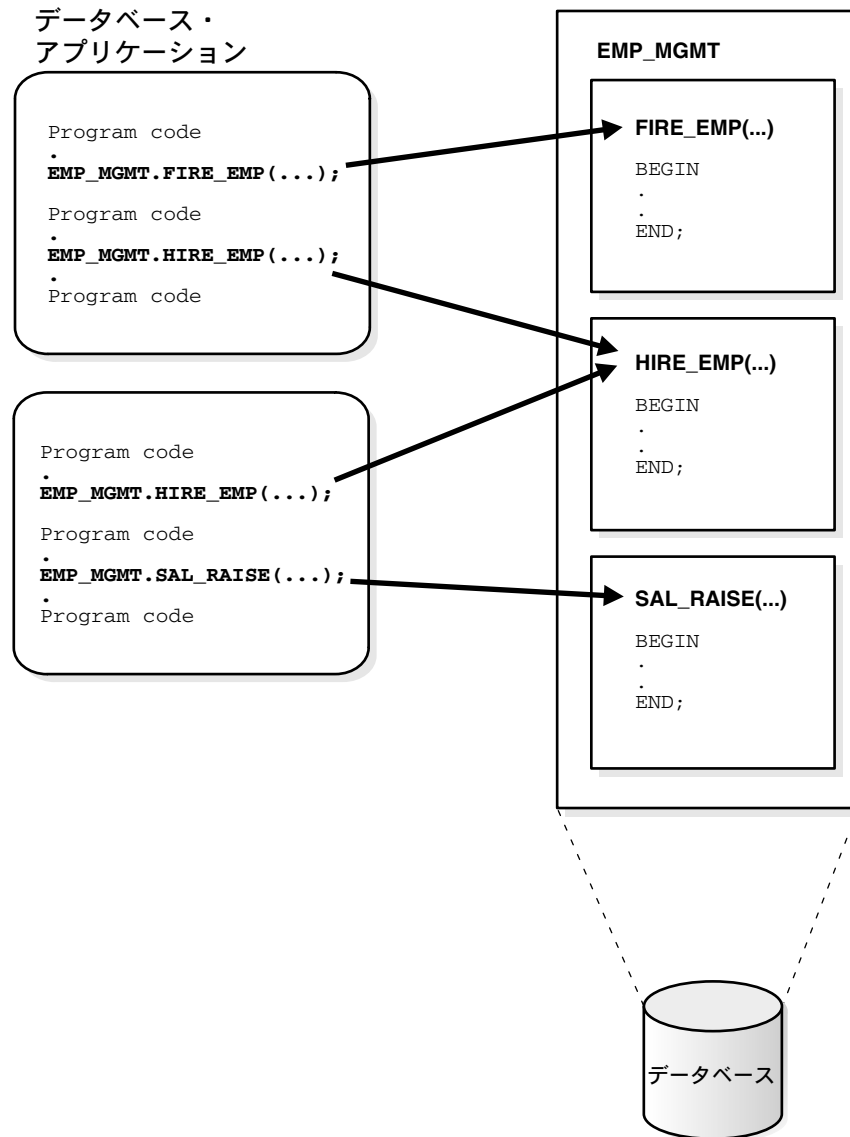
関連項目：『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

パッケージ

パッケージとは、関連するプロシージャとファンクション、およびこれらのプロシージャとファンクションが使用するカーソルと変数をグループとしてまとめたもので、1 単位として継続的に使用できるようにデータベースに格納されています。スタンドアロン・プロシージャやファンクションと同様に、パッケージ・プロシージャとファンクションは、アプリケーションやユーザーが明示的にコールできます。

図 17-3 に、従業員データベースの管理に使用するいくつかのプロシージャをカプセル化するパッケージを示します。

図 17-3 ストアド・パッケージ



データベース・アプリケーションは、必要に応じて明示的にパッケージ・プロシージャをコールします。EMP_MGMT パッケージに対する権限を付与されたユーザーは、そこに含まれている任意のプロシージャを明示的に実行できます。たとえば、Oracle Enterprise Manager または SQL*Plus では、次の文を発行して HIRE_EMP パッケージ・プロシージャを実行できます。

```
EXECUTE emp_mgmt.hire_emp ('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

スタンドアロンのストアード・プロシージャに比べて、パッケージには開発上およびパフォーマンス上のいくつかの利点があります。

関連項目： 17-12 ページの「[パッケージ](#)」

プロシージャとファンクション

「プロシージャ」や「ファンクション」は、SQL 文やその他の PL/SQL 構成体のセットで構成され、グループとしてまとめてデータベースに格納されるスキーマ・オブジェクトです。特定の問題を解決したり、関連する一連のタスクを実行するために、1 単位として実行されます。プロシージャとファンクションには、コール側から、入力のみ、出力のみまたは入出力の値が入るパラメータを指定できます。

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あわせ持つことができます。たとえば、次の文は、預金口座に入金する CREDIT_ACCOUNT プロシージャを作成します。

```
CREATE PROCEDURE credit_account
    (acct NUMBER, credit NUMBER) AS
/* This procedure accepts two arguments: an account number and an
   amount of money to credit to the specified account. If the
   specified account does not exist, a new account is created. */

    old_balance  NUMBER;
    new_balance  NUMBER;
BEGIN
    SELECT balance INTO old_balance FROM accounts
        WHERE acct_id = acct
        FOR UPDATE OF balance;

    new_balance := old_balance + credit;
    UPDATE accounts SET balance = new_balance
        WHERE acct_id = acct;
    COMMIT;
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    INSERT INTO accounts (acct_id, balance)
      VALUES(acct, credit);
  WHEN OTHERS THEN
    ROLLBACK;
END credit_account;
```

このサンプル・プロシージャには、SQL 文と PL/SQL 文の両方が含まれていることに注意してください。

関連項目：『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』

定義者権限と実行者権限

「PL/SQL プロシージャ」は、プロシージャ定義に応じて、その所有者の権限（定義者権限）またはカレント・ユーザーの権限（実行者権限）で実行できます。

- 定義者権限プロシージャは、その所有者の権限で実行されます。定義者権限プロシージャと、定義者権限プロシージャによって直接または間接的に実行されるプロシージャでは、ロールは使用禁止になります。
- 実行者権限プロシージャは、現在使用可能になっているロールも含め、すべての実行者の権限で実行されます。実行者権限プロシージャが定義者権限プロシージャによって直接または間接的に実行された場合、ロールは使用禁止になります。

関連項目：

- 権限の詳細は、27-7 ページの「[プロシージャのセキュリティに関するトピック](#)」を参照してください。
- ロールの詳細は、27-20 ページの「[PL/SQL ブロックとロール](#)」を参照してください。

カレント・ユーザー

「カレント・ユーザー」は、始めはセッション・ユーザーです。これは、ログインしたユーザーの場合と、リモート・プロシージャ・コール・セッションに対応付けられたユーザーの場合があります。カレント・ユーザーは、実行者権限プロシージャや無名ブロックをコールまたは終了しても変わりません。

定義者権限プロシージャをコールすると、そのプロシージャの所有者がカレント・ユーザーになります。定義者権限プロシージャを終了すると、カレント・ユーザーは前のカレント・ユーザー、つまり、定義者権限プロシージャをコールする前のカレント・ユーザーに戻ります。

外部参照の解決

PL/SQL プロシージャ内の「外部参照」とは、プログラム・ユニットの外側にあるオブジェクトを参照する名前です。

- 定義者権限プロシージャの場合、すべての外部参照は、そのプロシージャを含むスキーマ内で解決されます。
- 実行者権限プロシージャの場合、外部参照の解決方法は、それを含む文の種類に応じて異なります。次の外部参照は、カレント・ユーザーに対応付けられたスキーマ内で実行時に解決されます。実行時には、これらの外部参照によるデータ・アクセスも、カレント・ユーザーの権限と比較チェックされます。
 - 表、ビュー、順序およびファンクションの参照など、DML 文中の外部参照
 - 動的 SQL 文中と DBMS_SQL 文中の外部参照

ダイレクト PL/SQL プロシージャ・コールを含む、他の PL/SQL 文中のすべての外部参照は、実行者権限プロシージャを含むスキーマ内で解決されます。

実行者のスキーマ内で名前が解決されるため、アプリケーションはスキーマを指定しなくても、ユーザー固有の表にアクセスできます。

関連項目： 17-20 ページの「データベース・オブジェクトとプログラム・ユニットの名前解決」

プロシージャの利点

プロシージャは、次の分野で特長を発揮します。

- 定義者権限プロシージャでのセキュリティ
- 実行者権限プロシージャで継承される権限およびスキーマのコンテキスト
- パフォーマンス
- メモリー割当て
- 生産性
- 整合性

定義者権限プロシージャでのセキュリティ

ストアド・プロシージャは、データ・セキュリティの規定に役立ちます。定義者の権限で実行するプロシージャとファンクションを介してのみユーザーがデータにアクセスできるようにすると、ユーザーが実行できるデータベース操作を制限できます。たとえば、表を更新するプロシージャへのアクセス権は付与しても、その表自体へのアクセス権は付与しないこともできます。ユーザーがプロシージャを呼び出すと、そのプロシージャがプロシージャの所有者の権限で操作を実行します。プロシージャの実行権限しかなく、表データの問合せ、更

新または削除の権限を持たないユーザーは、プロシージャを呼び出すことはできても、表のデータをそれ以外の方法では操作できません。

関連項目： 17-11 ページの「[ストアド・プロシージャの依存性の追跡](#)」

実行者権限プロシージャで継承される権限およびスキーマのコンテキスト

実行者権限プロシージャは、それをコールしたプロシージャから権限とスキーマのコンテキストを継承します。つまり、実行者権限プロシージャは特定のユーザーやスキーマに結び付けられておらず、実行者権限プロシージャの各コールはカレント・ユーザーの権限でカレント・ユーザーのスキーマ内で実行されます。アプリケーション開発者の場合は、実行者権限プロシージャにより、基礎となるデータが複数のユーザー・スキーマに分割されていても、アプリケーションの論理を容易に一元化できます。

たとえば、管理者として EMP 表の更新プロシージャを実行するユーザーは給与を更新できますが、同じプロシージャを担当者として実行するユーザーが実行できる操作は、住所データの更新に制限できます。

パフォーマンス

ストアド・プロシージャを使用すると、次の方法でデータベースのパフォーマンスが向上します。

- 情報は 1 回しか送信されず、それ以降は使用するとき呼び出されるため、個々の SQL 文を発行したり、PL/SQL ブロック全体のテキストを Oracle に送信する場合に比べて、ネットワークを介して送信する必要のある情報量が少なくなります。
- データベース内でプロシージャのコンパイル済形式をそのまま使用できるため、実行時にコンパイルする必要がありません。
- プロシージャがすでに SGA の共有プール内にある場合は、ディスクから検索する必要がないため、ただちに実行を開始できます。

メモリー割当て

ストアド・プロシージャは、Oracle の共有メモリー機能を活用しているため、何人かのユーザーが実行する場合も、プロシージャのコピーを 1 つメモリーにロードするだけですみます。同じコードを何人ものユーザーが共有すれば、アプリケーションに必要な Oracle メモリーを実質的に削減できます。

生産性

ストアド・プロシージャにより、開発の生産性が向上します。共通のプロシージャを中心に、アプリケーションを設計することにより、冗長なコーディングを回避し、生産性を向上させることができます。

たとえば、EMP 表の従業員レコードを挿入、更新または削除するためのプロシージャを記述できます。これらのプロシージャは、各タスクの実行に必要な SQL 文を書き換えることなく、どんなアプリケーションからでもコールできます。データ管理の方法に変更があった場合も、修正する必要があるのはプロシージャのみで、プロシージャを使用するすべてのアプリケーションを修正する必要はありません。

整合性

ストアド・プロシージャにより、アプリケーションの整合性と一貫性が向上します。共通のプロシージャ群を中心としてのアプリケーションを開発すれば、コーディング・エラーの発生回数を少なくすることができます。

たとえば、プロシージャやファンクションが正確な結果を戻すかどうかをテストし、検証後は、そのプロシージャとファンクションを必要な数のアプリケーションで再使用できます。再びテストする必要はありません。そのプロシージャが参照するデータの構造になんらかの変更があった場合は、そのプロシージャのみを再コンパイルすれば十分です。そのプロシージャをコールする側のアプリケーションを修正する必要はありません。

プロシージャのガイドライン

ストアド・プロシージャの設計時には、次のガイドラインに従ってください。

- プロシージャは、単一の限定的なタスクを実行するように定義します。複数の個別のサブタスクを含む長いプロシージャは定義しないでください。多数のプロシージャに共通のサブタスクが存在すると、複数のプロシージャ・コードに不必要な重複が発生するためです。
- Oracle の他の機能によってすでに提供されている機能性を重複して提供するプロシージャは定義しないでください。たとえば、整合性制約を宣言すれば簡単に規定できる単純なデータ整合性規則の場合、それを規定するプロシージャは定義しないでください。

無名 PL/SQL ブロックとストアド・プロシージャ

ストアド・プロシージャを作成し、それをスキーマ・オブジェクトとしてデータベースに格納できます。いったん作成してコンパイルしたプロシージャは、再コンパイルしなくても実行可能な名前付きのオブジェクトになります。さらに、依存性情報がデータ・ディクショナリに格納されるため、それぞれのストアド・プロシージャの妥当性が保証されます。

ストアド・プロシージャとは別の方法として、無名 PL/SQL ブロックを Oracle Tools やアプリケーションから Oracle Server に送信し、無名 PL/SQL ブロックを作成できます。Oracle によって PL/SQL ブロックがコンパイルされ、SGA の共有プールにそのコンパイル済バージョンが入れられます。ただし、そのソース・コードやコンパイル済バージョンは、カレント・インスタンスの後にも再使用できるようにデータベースに格納されることはありません。共有 SQL を使用すると、共有プールに入っている無名 PL/SQL ブロックがその共有プールからフラッシュされるまでの間は、そのブロックを再使用および共有できます。

どちらの場合でも、データベース・アプリケーションからデータベースまたはメモリーに格納されているデータベース・プロシージャに、PL/SQL ブロックを移すことにより、Oracle が実行時に不必要な再コンパイルを実行することがなくなり、アプリケーションと Oracle の全体的なパフォーマンスが向上します。

関連項目： 17-17 ページの「[Oracle がプロシージャとパッケージを格納する方法](#)」

スタンドアロン・プロシージャ

パッケージのコンテキスト内で定義されていないストアド・プロシージャのことを、「スタンドアロン・プロシージャ」と呼びます。パッケージ内で定義されているプロシージャは、そのパッケージの一部とみなされます。

関連項目： パッケージの利点の詳細は、17-12 ページの「[パッケージ](#)」を参照してください。

ストアド・プロシージャの依存性の追跡

ストアド・プロシージャは、その本体で参照するオブジェクトに依存します。Oracle は、そのような依存性を自動的に追跡し、管理します。たとえば、プロシージャが参照している表の定義を変更した場合は、そのプロシージャを再コンパイルして、引き続き設計どおりに機能することを確認する必要があります。通常、Oracle はこのような依存性の管理を自動的に処理します。

関連項目： 依存性の追跡の詳細は、第 20 章「[Oracle の依存性の管理](#)」を参照してください。

外部プロシージャ

Oracle Server 上で実行される PL/SQL プロシージャからは、C プログラミング言語で作成して共有ライブラリに格納した外部プロシージャや外部ファンクションをコールできます。C ルーチンは、Oracle Server とは別のアドレス空間で実行されます。

関連項目： 外部プロシージャと Inter-Language Method Services 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。

パッケージ

パッケージは、相互に関連するプロシージャ、ファンクションおよびそれに関するカーソルや変数を 1 単位としてカプセル化してデータベースに入れたものです。

パッケージは、仕様部および本体という 2 つの部分で構成されています。パッケージの「仕様部」ではパッケージのすべてのパブリックな構成体を宣言し、「本体」ではパッケージのすべての構成体（パブリックとプライベート）を定義します。パッケージを 2 つに分けることには、次のような利点があります。

- 開発者は開発サイクルを柔軟なものにすることができます。パッケージ本体を実際には作成せずに、仕様部を作成し、パブリック・プロシージャを参照するようにできます。
- パッケージ仕様部にパブリックに宣言されている仕様部とは別個に、パッケージ本体に含まれているプロシージャ本体を変更できます。プロシージャ仕様部が変更されない限り、パッケージの変更されたプロシージャを参照するオブジェクトに無効のマーク、つまり、プロシージャの再コンパイルが必要であることを示すマーク、が付けられることはありません。

次の例では、銀行業務の取引きを処理する複数のプロシージャとファンクションを含むパッケージの仕様部と本体を作成します。

```
CREATE PACKAGE bank_transactions (null) AS
    minimum_balance CONSTANT NUMBER := 100.00;
    PROCEDURE apply_transactions;
    PROCEDURE enter_transaction (acct NUMBER,
                                kind CHAR,
                                amount NUMBER);
END bank_transactions;

CREATE PACKAGE BODY bank_transactions AS

/* Package to input bank transactions */
```

```
new_status CHAR(20); /* Global variable to record status
                        of transaction being applied. Used
                        for update in APPLY_TRANSACTIONS. */

PROCEDURE do_journal_entry (acct NUMBER,
                           kind CHAR) IS

/* Records a journal entry for each bank transaction applied
   by the APPLY_TRANSACTIONS procedure. */

BEGIN
    INSERT INTO journal
        VALUES (acct, kind, sysdate);
    IF kind = 'D' THEN
        new_status := 'Debit applied';
    ELSIF kind = 'C' THEN
        new_status := 'Credit applied';
    ELSE
        new_status := 'New account';
    END IF;
END do_journal_entry;

PROCEDURE credit_account (acct NUMBER, credit NUMBER) IS

/* Credits a bank account the specified amount. If the account
   does not exist, the procedure creates a new account first. */

    old_balance NUMBER;
    new_balance NUMBER;

BEGIN
    SELECT balance INTO old_balance FROM accounts
        WHERE acct_id = acct
        FOR UPDATE OF balance; /* Locks account for credit update */

    new_balance := old_balance + credit;
    UPDATE accounts SET balance = new_balance
        WHERE acct_id = acct;
    do_journal_entry(acct, 'C');

EXCEPTION
    WHEN NO_DATA_FOUND THEN /* Create new account if not found */
        INSERT INTO accounts (acct_id, balance)
            VALUES(acct, credit);
```

```

        do_journal_entry(acct, 'N');
    WHEN OTHERS THEN /* Return other errors to application */
        new_status := 'Error: ' || SQLERRM(SQLCODE);
END credit_account;

PROCEDURE debit_account (acct NUMBER, debit NUMBER) IS

/* Debits an existing account if result is greater than the
   allowed minimum balance. */

    old_balance      NUMBER;
    new_balance      NUMBER;
    insufficient_funds EXCEPTION;

BEGIN
    SELECT balance INTO old_balance FROM accounts
        WHERE acct_id = acct
        FOR UPDATE OF balance;
    new_balance := old_balance - debit;
    IF new_balance >= minimum_balance THEN
        UPDATE accounts SET balance = new_balance

            WHERE acct_id = acct;
    do_journal_entry(acct, 'D');
    ELSE
        RAISE insufficient_funds;
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        new_status := 'Nonexistent account';
    WHEN insufficient_funds THEN
        new_status := 'Insufficient funds';
    WHEN OTHERS THEN /* Returns other errors to application */
        new_status := 'Error: ' || SQLERRM(SQLCODE);
END debit_account;

PROCEDURE apply_transactions IS

/* Applies pending transactions in the table TRANSACTIONS to the
   ACCOUNTS table. Used at regular intervals to update bank
   accounts without interfering with input of new transactions. */

/* Cursor fetches and locks all rows from the TRANSACTIONS
   table with a status of 'Pending'. Locks released after all
   pending transactions have been applied. */

```

```

CURSOR trans_cursor IS
    SELECT acct_id, kind, amount FROM transactions
        WHERE status = 'Pending'
        ORDER BY time_tag
        FOR UPDATE OF status;

BEGIN
    FOR trans IN trans_cursor LOOP    /* implicit open and fetch */
        IF trans.kind = 'D' THEN
            debit_account(trans.acct_id, trans.amount);
        ELSIF trans.kind = 'C' THEN
            credit_account(trans.acct_id, trans.amount);
        ELSE
            new_status := 'Rejected';
        END IF;
        /* Update TRANSACTIONS table to return result of applying
           this transaction. */
        UPDATE transactions SET status = new_status
            WHERE CURRENT OF trans_cursor;
    END LOOP;
    COMMIT; /* Release row locks in TRANSACTIONS table. */
END apply_transactions;

PROCEDURE enter_transaction (acct   NUMBER,
                             kind    CHAR,
                             amount  NUMBER) IS

/* Enters a bank transaction into the TRANSACTIONS table. A new
transaction is always put into this 'queue' before being
applied to the specified account by the APPLY_TRANSACTIONS
procedure. Therefore, many transactions can be simultaneously
input without interference. */

BEGIN
    INSERT INTO transactions
        VALUES (acct, kind, amount, 'Pending', sysdate);
    COMMIT;
END enter_transaction;

END bank_transactions;

```

データベース管理者やアプリケーション開発者は、パッケージを使用することによって、多数の類似したルーチンを編成できます。さらに、機能性とデータベースのパフォーマンスが向上します。

関連項目： [第 20 章「Oracle の依存性の管理」](#)

パッケージの利点

パッケージは、互いに関連するプロシージャ、変数およびカーソルの定義に使用します。次のような分野で、その長所を発揮します。

- 関連するプロシージャと変数のカプセル化
- パブリック・プロシージャ、プライベート・プロシージャ、変数、定数およびカーソルの宣言
- 優れたパフォーマンス

カプセル化

ストアド・パッケージを使用すると、ストアド・プロシージャ、変数、データ型などをカプセル化、つまりグループ化し、名前を付けて 1 単位としてデータベースに格納できます。この方針は、開発プロセスにおける優れた構成を提供します。

プロシージャ型の構成体を 1 つのパッケージにカプセル化すると、権限の管理も簡単になります。パッケージを使用する権限を付与すると、権限を付与されたユーザーは、そのパッケージのすべての構成体にアクセスできるようになります。

パブリックおよびプライベートなデータとプロシージャ

パッケージの定義方法により、変数、カーソルおよびプロシージャを次のどちらかに指定できます。

パブリック パッケージのユーザーが直接アクセスできます。

プライベート パッケージのユーザーから隠されます。

たとえば、パッケージに 10 個のプロシージャが含まれているとします。このうち、3 つのみはユーザーが実行できるようにパブリックにし、残りのプロシージャはパッケージ内のプロシージャしかアクセスできないようにプライベートにするという形で、パッケージを定義できます。

パブリックおよびプライベートなパッケージ変数を、PUBLIC への権限付与と混同しないください。

関連項目： PUBLIC への権限付与の詳細は、[第 26 章「データベース・アクセスの制御」](#)を参照してください。

パフォーマンスの向上

パッケージ内のプロシージャが初めてコールされた時点で、そのパッケージ全体がメモリーにロードされます。スタンドアロン・プロシージャは個別にロードする必要があるのと対照的に、このロードは 1 回の操作で完了します。そのため、関連するパッケージ・プロシージャがコールされても、すでにメモリーにあるコンパイル済のコードを実行すればよく、ディスク I/O は発生しません。

パッケージ本体は、仕様部に影響を与えずに置換したり再コンパイルできます。その結果、パッケージ仕様部も置き換えるのであれば、パッケージの構成体を参照するスキーマ・オブジェクト（常に仕様部を介してアクセス）を再コンパイルする必要はありません。パッケージを使用すると、不必要な再コンパイルが最小限に抑えられるため、全体的なデータベース・パフォーマンスへの影響は少なくなります。

パッケージの依存性の追跡

パッケージは、その本体に定義されているプロシージャとファンクションが参照するオブジェクトに依存しています。Oracle は、そのような依存性を自動的に追跡し、管理します。

関連項目： 第 20 章「Oracle の依存性の管理」

オラクル社が提供するパッケージ

オラクル社は、データベースや PL/SQL の機能を拡張する機能を含んだ多数の PL/SQL パッケージを提供します。これらのほとんどのパッケージは、DBMS_SQL、DBMS_LOCK および DBMS_JOB のように、名前に DBMS_ 接頭辞が付いています。また、UTL_HTTP や UTL_FILE のように UTL_ 接頭辞が付いているパッケージや、DEBUG_ または OUTLN_ など、他の接頭辞が付いているパッケージもあります。

関連項目： オラクル社が提供するパッケージの詳細は、『Oracle8i PL/SQL 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。

Oracle がプロシージャとパッケージを格納する方法

プロシージャまたはパッケージを作成すると、次のステップが実行されます。

- プロシージャまたはパッケージをコンパイルします。
- コンパイル済コードをメモリーに格納します。
- プロシージャまたはパッケージをデータベースに格納します。

プロシージャとパッケージのコンパイル

PL/SQL コンパイラは、ソース・コードをコンパイルします。PL/SQL コンパイラは、Oracle に組み込まれている PL/SQL エンジンの一部です。コンパイル時にエラーが発生すると、メッセージが戻されます。プログラム内のエラー・ハンドラを使用してエラーに備えることができるため、Oracle がプロシージャを終了させるのを防止できます。

関連項目： コンパイル・エラーの識別方法は、『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。

コンパイル済コードのメモリーへの格納

Oracle は、コンパイル済のプロシージャまたはパッケージをシステム・グローバル領域 (SGA) の共有プールにキャッシュします。これにより、コードをただちに実行し、多数のユーザーの間で共有できます。プロシージャまたはパッケージのコンパイル済バージョンは、最初にそのプロシージャをコールしたユーザーが自分のセッションを終了しても、共有プールが使用した、修正された LRU アルゴリズムに従って共有プールに残ります。

関連項目： 共有プール・バッファに固有の情報は、7-6 ページの「[共有プール](#)」を参照してください。

プロシージャとパッケージのデータベースへの格納

作成時とコンパイル時に、Oracle はストアド・プロシージャまたはパッケージに関する次の情報を自動的にデータベースに格納します。

スキーマ・オブジェクト名	この名前を使用して、プロシージャまたはパッケージを識別します。この名前は、CREATE PROCEDURE 文、CREATE FUNCTION 文、CREATE PACKAGE 文または CREATE PACKAGE BODY 文に指定します。
ソース・コードと解析ツリー	PL/SQL コンパイラはソース・コードを解析して、「解析ツリー」と呼ばれるソース・コードの解析済の表現を生成します。
疑似コード (P コード)	PL/SQL コンパイラは、解析後のコードに基づいて、「疑似コード」(P コード) を生成します。プロシージャまたはパッケージが呼び出されると、PL/SQL エンジンはこのコードを実行します。
エラー・メッセージ	Oracle は、プロシージャまたはパッケージのコンパイル時にエラーを生成する場合があります。

プロシージャやパッケージの不必要な再コンパイルを回避するため、解析ツリーおよびオブジェクトの P コードがデータベースに格納されます。そのため、プロシージャまたはパッケージがその起動時に SGA 内に存在しなければ、そのプロシージャまたはパッケージのコンパイル済のバージョンが PL/SQL エンジンによって SGA の共有プール・バッファに読み込まれます。解析ツリーは、そのプロシージャをコールするコードをコンパイルするときに使用されます。

データベース・プロシージャのすべての部分は、対応するデータベースの (SYSTEM 表領域にある) データ・ディクショナリに格納されます。データベース管理者は、SYSTEM 表領域のサイズを計画する際に、この表領域に格納されるすべてのストアド・プロシージャに必要な領域を考慮に入れてください。

Oracle がプロシージャとパッケージを実行する方法

スタンドアロン・プロシージャまたはパッケージ・プロシージャを起動すると、Oracle により次のタスクが実行されます。

- [ユーザー・アクセスの検証](#)
- [プロシージャの妥当性の検証](#)
- [プロシージャの実行](#)

定義者権限プロシージャと実行者権限プロシージャでは、検証方法および実行方法が異なります。

関連項目： 17-11 ページの「[ストアド・プロシージャの依存性の追跡](#)」

ユーザー・アクセスの検証

Oracle は、コールしたユーザーがそのプロシージャまたはカプセル化されたパッケージに対して EXECUTE 権限を所有しているかどうかを検証します。プロシージャを実行するユーザーには、プロシージャ内で参照されているプロシージャやオブジェクトに対するアクセス権は必要ありません。参照先のスキーマ・オブジェクトへのアクセス権が必要なのは、プロシージャまたはパッケージの作成者のみです。

プロシージャの妥当性の検証

Oracle は、プロシージャまたはパッケージの状態が有効か無効かをデータ・ディクショナリで調べます。プロシージャまたはパッケージは、最後にコンパイルされた後に次のいずれかの状況が発生すると無効になります。

- 表、ビューまたは他のプロシージャなど、プロシージャまたはパッケージ内で参照されている 1 つ以上のスキーマ・オブジェクトが、変更または削除された場合。たとえば、ユーザーが表に列を追加した場合などです。
- パッケージまたはプロシージャに必要なシステム権限が、PUBLIC から、またはプロシージャかパッケージの所有者から取り消された場合。
- プロシージャまたはパッケージが参照する 1 つ以上のスキーマ・オブジェクトに対する必要なスキーマ・オブジェクト権限が、PUBLIC から、またはプロシージャかパッケージの所有者から取り消された場合。

前述のいずれかの操作によって無効にされていない場合、そのプロシージャは「有効」です。有効なスタンドアロン・プロシージャまたはパッケージ・プロシージャがコールされると、コンパイル済コードが実行されます。無効なスタンドアロン・プロシージャまたはパッケージ・プロシージャがコールされると、実行される前に自動的に再コンパイルされます。

関連項目： プロシージャとパッケージの有効と無効、プロシージャの再コンパイルおよび依存性の問題の詳細は、[第 20 章「Oracle の依存性の管理」](#)を参照してください。

プロシージャの実行

PL/SQL エンジン は、状況に応じて異なるステップでプロシージャまたはパッケージを実行します。

- 有効なプロシージャがメモリーにあれば、PL/SQL エンジン は単に P コードを実行します。
- 有効なプロシージャがメモリーになければ、PL/SQL エンジン はコンパイル済の P コードをディスクからメモリーにロードして実行します。パッケージの場合、パッケージのすべての構成体（すべてのプロシージャ、変数など、実行可能なコード断片としてコンパイルされるもの）は 1 単位としてロードされます。

15-17 ページの [図 15-2](#) のように、PL/SQL エンジン は、プロシージャ型の文はすべて自分で処理し、SQL 文は SQL 文エグゼキュータに渡すことによって、プロシージャの文を文単位で処理します。

データベース・オブジェクトとプログラム・ユニットの名前解決

定義者権限プロシージャの場合、すべての外部参照は、定義者のスキーマ内で解決されます。実行者権限プロシージャの場合、外部参照の解決方法は、それが含まれる文の種類に応じて異なります。

- DML 文中と動的 SQL 文中の外部参照は実行者のスキーマ内で解決され、Oracle は実行時に実行者権限を使用してアクセス権限をチェックします。このルールは、次の各文に含まれるデータベース・オブジェクト（表、ビューおよび順序など）の名前に適用されます。
 - SELECT、UPDATE、INSERT および DELETE 文
 - OPEN カーソル（カーソル宣言内の SELECT 文は、OPEN 時に解決されます）
 - LOCK TABLE 文
 - 動的 SQL 文 EXECUTE IMMEDIATE および PREPARE
 - DBMS_SQL 文、つまり DBMS_SQL.PARSE() を使用して解析される文

スキーマ・オブジェクトの名前は実行時に解決されますが、コンパイラは名前を定義者のスキーマにある一時オブジェクトに一時的に解決して、この種の各参照のタイプをコンパイル時に識別します。

- DML または動的 SQL 以外のすべての文中の外部参照は、定義者のスキーマ内で解決され、Oracle コンパイル時に定義者の権限を使用してアクセス権限をチェックします。このルールは、プロシージャがコールする他のプログラム・ユニット（パッケージ、プロシージャ、ファンクションおよびタイプなど）の名前に適用されます。

たとえば、外部ファンクション・コール "x := func(1)" を伴う代入文の場合、ファンクション名はプロシージャの定義者のスキーマ内で解決され、コンパイル時には定義者の権限で "func" へのアクセスがチェックされます。

データベース・リンクの名前解決

PL/SQL プロシージャ内のデータベース・リンク名は、前項のルールに従って解決されます。リモート・データベースへの接続には、次のいずれかの認可 ID が使用されます。

1. 固定ユーザー・リンクの場合、Oracle はリンク内で指定されたユーザー名を使用して、リモート・データベースに接続します。この動作は、定義者権限プロシージャおよび実行者権限プロシージャと同じです。

```
CREATE DATABASE LINK link1
  CONNECT TO scott IDENTIFIED BY tiger
  USING connect_string;
```

JOE が所有するプロシージャで LINK1 が使用される場合、このリンクで指定されている名前は SCOTT であるため、そのプロシージャの実行者に関係なく SCOTT として接続されます。

2. 接続ユーザー・リンクの場合、Oracle はセッション・ユーザー名を使用してリモート・データベースに接続します。この動作は、定義者権限プロシージャおよび実行者権限プロシージャと同じです。

```
CREATE DATABASE LINK link2
  USING connect_string;
```

JOE が所有するプロシージャで接続ユーザー・リンク LINK2 が使用される場合に、ユーザー SCOTT がこのプロシージャを起動すると、リモート・データベースには SCOTT で接続されます。

3. カレント・ユーザーのリンクの場合、定義者権限プロシージャと実行者権限プロシージャでは動作が異なります。
 - 実行者権限プロシージャの場合、Oracle は実行者の許可 ID を使用してリモート・ユーザーで接続します。

```
CREATE DATABASE LINK link3
  CONNECT TO CURRENT_USER
  USING connect_string;
```

JOE が所有する実行者権限プロシージャをグローバル・ユーザー SCOTT が起動する場合、カレント・ユーザーは SCOTT であるため、LINK3 はリモート・データベースにユーザー SCOTT で接続します。

- 定義者権限プロシージャの場合、Oracle は所有者の許可 ID を使用してリモート・ユーザーで接続します。JOE が定義者権限プロシージャを所有している場合は、JOE がカレント・ユーザーになるため、LINK3 はリモート・データベースにグローバル・ユーザー JOE で接続します。

関連項目：『Oracle8i 分散システム』

アドバンスト・キューイング

この章では、Oracle アドバンスト・キューイング機能について説明します。この章の内容は、次のとおりです。

- [メッセージ・キューイングの概要](#)
- [Oracle Advanced Queuing](#)
 - [キューイング・モデル](#)
 - [キューイング・エンティティ](#)
 - [アドバンスト・キューイングの機能](#)

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。

関連項目： 『Oracle8i アプリケーション開発者ガイド アドバンスト・キューイング』

メッセージ・キューイングの概要

典型的なオンライン販売ビジネスを考えます。これには、受入力、オーダー処理、請求処理および顧客サービスの各アプリケーションが含まれます。実際の出荷部門は、各地の倉庫に配置されています。請求処理部門と顧客サービス部門も、各地に散在している場合があります。

この状況では、分散コンピューティング環境にある複数のクライアント間の通信が必要です。メッセージは、クライアントとサーバー間、および異なるサーバー上のプロセス間でやりとりされます。効率的なメッセージ・システムには、内容ベースのルーティング、サブスクリプションおよび問合せを実装することが必要です。

メッセージ・システムは、次の2つのタイプのどちらかに分類できます。

- **同期通信**
- **非同期通信**

同期通信

同期通信は、要求 / 応答のパラダイムに基づくもので、あるプログラムが別のプログラムに要求を送信し、応答が到着するまで待機するというものです。

この通信モデル（「オンライン通信」または「接続通信」とも呼ぶ）は、作業を進めるために応答を受信する必要があるプログラムに適しています。従来のクライアント / サーバーのアーキテクチャは、このモデルに基づいています。

同期通信モデルの大きな欠点は、コール側アプリケーションが動作するために、要求を受信するプログラムを使用可能かつ実行中の状態にしておく必要があるという点です。

非同期通信

「切断」モデルまたは「遅延」モデルでは、プログラムは非同期に通信し、要求をキューに入れてから次の作業に進みます。

たとえば、特定の条件が満たされた後で、アプリケーションへのデータ入力や、操作の実行を必要とする場合があります。要求を受けるプログラムは、キューから要求を取り出し、その要求に従って動作します。このモデルは、要求をキューに入れた後でも作業を続けられるアプリケーションに適しています。そのようなアプリケーションでは、応答を待機するために動作が止まることはありません。

ネットワーク、マシンおよびアプリケーションに障害が発生しても正しく動作する遅延実行の場合、要求は永続的に格納されて、1度のみ処理される必要があります。このことは、永続キューイングとトランザクションの保護を組み合わせることによって実現できます。

通常、クライアント / サーバーの要求を「1 度のみ」処理することが、トランザクションの統合性またはフローを保つうえで重要になります。たとえば、特定の価格で株の買い注文を要求する場合、その要求の伝送時、受信時または実行時にネットワーク障害あるいはシステム障害が発生したとしても、要求をゼロ回または 2 回実行することは認められません。

Oracle Advanced Queuing

Oracle Advanced Queuing は、分散アプリケーションがメッセージを使用して非同期に通信するための基盤を提供します。メッセージは Oracle Server による遅延取出しおよび遅延処理のために、キューに格納されます。これにより、トランザクション処理モニターやメッセージ指向のミドルウェアなどの追加ソフトウェアがなくても、信頼できる効率のよいキューイング・システムが提供されます。

Oracle Advanced Queuing は、次の機能を提供します。

- Point-to-Point 通信モデルおよびパブリッシュ / サブスクライブ通信モデル
- メッセージ用の構造化ペイロード
- キュー内のメッセージの優先順位と順序付け
- 各メッセージための実行時間枠を指定する機能
- 標準 SQL を使用してキューを問い合わせる機能
- アプリケーションの開発と管理を単純化する統合されたトランザクション
- 複数のメッセージをまとめてデキューする機能
- 複数のレシピエントを指定する機能
- ローカルまたはリモートの Oracle データベース内のキューにメッセージを伝播させる機能
- 内容ベースのフィルタリングによるルールベースのパブリッシュ / サブスクライブ
- 複数キューでメッセージを待機する機能
- 永続キューイングと非永続キューイング
- キューに格納されたメッセージと他のキューに伝播したメッセージに関する統計
- 分析のためのメッセージとメッセージ履歴の保存
- キュー・レベルのアクセス制御
- パフォーマンス向上を目的とした Oracle Parallel Server 環境のサポート
- コールバック関数を使用した非同期通知

Oracle Advanced Queuing キューはデータベースの表内で実装されるため、高い可用性、高い拡張性および高い信頼性など、運用上のすべての利点がキュー・データにも当てはまります。さらに、キューに対してデータベース開発ツールやデータベース管理ツールを使用できます。

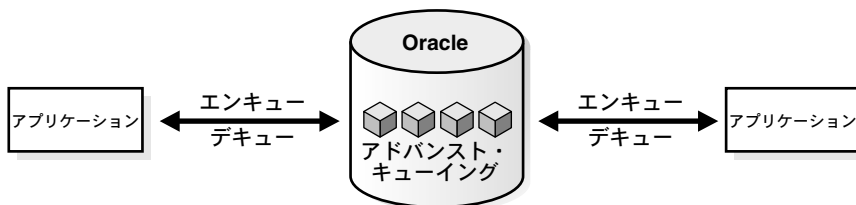
アプリケーションは、Oracle Advanced Queuing のネイティブ・インタフェース（PL/SQL、C/C++、Java および Visual Basic で定義）または Oracle Advanced Queuing の Java Messaging Service インタフェース（Java Messaging Service 規格に準拠した Java API）を介して、キューイング機能にアクセスできます。

キューイング・モデル

Oracle Advanced Queuing は、Point-to-Point モデルまたはパブリッシュ / サブスクライプ・モデルに従ってセットアップできます。

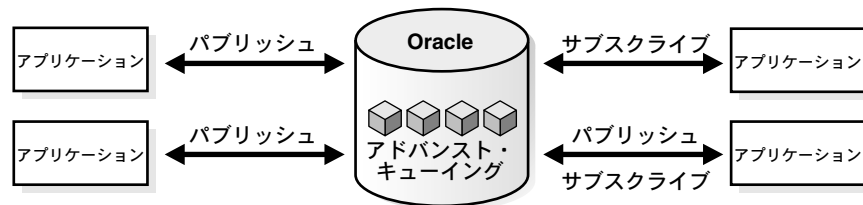
「Point-to-Point モデル」では、メッセージ交換に使用する共通キューを送信側と受信側が決定します。各メッセージは、単一のレシピエントのみが使用します。[図 18-1 「Point-to-Point モデル」](#)は、各アプリケーションが専用メッセージ・キューを持っていることを示しています。

図 18-1 Point-to-Point モデル



「パブリッシュ / サブスクライブ・モデル」は、トピックに基づいています。レシピエントはトピックにサブスクライブします。各メッセージを複数のレシピエントが使用できます。パブリッシャが必ずサブスクライバを認識しているわけではありません。図 18-2「パブリッシュ / サブスクライブ・モデル」は、各アプリケーションが特定のトピックをパブリッシュまたはサブスクライブできることを示しています。

図 18-2 パブリッシュ / サブスクライブ・モデル



キューイング・エンティティ

Oracle Advanced Queuing には、次のような基本エンティティがあります。

- メッセージ
- キュー
- キュー表
- エージェント
- レシピエント
- サブスクライバ
- レシピエント・リストおよびサブスクライバ・リスト
- ルール
- ルールベースのサブスクライバ
- キュー・モニター

メッセージ

「メッセージ」は、キューに挿入されたりキューから取り出される情報の最小単位です。通常、メッセージはビジネス・データとイベントをカプセル化します。メッセージは、制御情報とペイロード・データで構成されます。制御情報とは、メッセージを管理するために Oracle Advanced Queuing が使用するメッセージ・プロパティです。ペイロード・データとは、キューに格納されている情報であり、Oracle Advanced Queuing 側からは見えません。ペイロードのデータ型は、RAW またはオブジェクト型のどちらかです。

1 つのメッセージは、1 つのキューにしか存在できません。メッセージはエンキュー・コールによって作成され、デキュー・コールによって使用されます。エンキュー・コールとデキュー・コールは、DBMS_AQ パッケージの一部です。

キュー

「キュー」は、メッセージのリポジトリです。キューには、ユーザー・キューと例外キューの 2 種類があります。ユーザー・キューは通常のメッセージ処理用で、シングル・コンシューマ・キューとマルチ・コンシューマ・キューの 2 種類があります。「シングル・コンシューマ・キュー」は Point-to-Point モデルで使用され、「マルチ・コンシューマ・キュー」はパブリッシュ / サブスクライブ・モデルで使用されます。メッセージを取り出して処理することがなんらかの理由で不可能な場合、そのメッセージはユーザー・キューから例外キューに転送されます。

キュー内のすべてのメッセージは同じデータ型にする必要があります。

キューを作成、変更、開始、停止および削除するには、DBMS_AQADM パッケージを使用します。

キュー表

キューは「キュー表」に格納されます。各キュー表はデータベース表であり、1 つ以上のキューが含まれています。各キュー表にはデフォルトの例外キューが含まれます。

キュー表を作成すると、約 25 列を含むデータベース表が作成されます。それらの列には、Oracle Advanced Queuing メタデータとユーザー定義ペイロードが格納されます。

このキュー表に対して、1 つのビューと 2 つの索引が作成されます。ビューを使用することにより、メッセージ・データを問い合わせることができます。索引は、メッセージ・データへのアクセスを速めるために使用します。

エージェント

「エージェント」は、キューの使用者です。エージェントには、キューにメッセージを入れる（エンキューする）プロデューサと、キューからメッセージを取り出す（デキューする）コンシューマの 2 種類があります。任意の数のプロデューサとコンシューマが同時に 1 つのキューにアクセスできます。

エージェントは、名前、アドレスおよびプロトコルによって識別されます。リモート・データベース上のエージェントに対して現在サポートされているプロトコルは、`queue_name@dblink` 形式のアドレスを使用する Oracle データベース・リンクのみです。

レシピエント

メッセージの「レシピエント」は、その名前のみで指定することができます。その場合、レシピエントはメッセージがエンキューされたキューから、そのメッセージをデキューする必要があります。レシピエントは、名前とプロトコル値 0 のアドレスで指定できます。このアドレスは、同じデータベース内、または（データベース・リンクで識別する）別の Oracle8 データベース内の別のキューの名前にする必要があります。別の Oracle8 データベースの場合は、メッセージが指定のキューに伝播され、指定の名前を持つコンシューマがデキューできます。

レシピエント名が NULL であれば、そのメッセージはアドレスで指定したキューに伝播し、アドレスで指定したキューのサブスクライバがデキューできます。プロトコル・フィールドがゼロ以外の値であれば、名前フィールドとアドレス・フィールドはシステムで解釈されず、メッセージは特殊なコンシューマがデキューできます。

関連項目： 18-12 ページの「[他のデータベースへのメッセージの伝播](#)」

サブスクライバ

キュー管理者は、マルチ・コンシューマ・キューからメッセージを受信するレシピエントのデフォルト・リストを定義できます。このようなレシピエントを、サブスクライバと呼びます。キューごとに異なるサブスクライバを指定し、同じレシピエントが複数のキューのサブスクライバになることができます。

レシピエント・リストおよびサブスクライバ・リスト

1 つのメッセージを、複数のコンシューマで使用するように設計できます。これには、次の 2 つの方法があります。

- エンキュー元は、メッセージのレシピエントとしてメッセージを受信するコンシューマを明示的に指定できます。レシピエントは、名前、アドレスおよびプロトコルで指定するエージェントです。
- キュー管理者は、サブスクライバのリストを指定できます。レシピエントを指定せずにメッセージをエンキューすると、そのメッセージはそのキューのすべてのサブスクライバに暗黙的に送信されます。

キュー内の特定のメッセージを、そのキューのサブスクライバかどうかに関係なく、特定のレシピエントに送信できます。この場合サブスクライバ・リストは無効です。

ルール

「ルール」は、その規則に準拠するメッセージへのサブスクライブに関係のある 1 人以上のサブスクライバを定義するものです。この基準を満たしているメッセージは、関係のあるサブスクライバに配信されます。もう 1 つの方法では、ルールによって、サブスクライバが関係のある主題について、キュー内のメッセージにフィルタがかけられます。

ルールは、SQL 問合せの WHERE 句に似た構文を使用して、ブール式（TRUE または FALSE に評価される式）として指定します。このブール式には、次の条件を含めることができます。

- メッセージ・プロパティ（優先度と corrid）
- ユーザー・データのプロパティ（オブジェクト・ペイロードのみ）
- 関数（SQL 問合せの WHERE 句で指定）

ルールベースのサブスクライバ

「ルールベースのサブスクライバ」とは、デフォルトのレシピエント・リストに対応付けられたルールを持つサブスクライバです。ルールベース・サブスクライバには、そのメッセージについて対応するルールが TRUE に評価された場合に、レシピエントが明示的に指定されていないメッセージが送信されます。

キュー・モニター

「キュー・モニター」は、キュー内のメッセージを監視するオプションのバックグラウンド・プロセスです。キュー・モニターは、メッセージの時間切れ、再試行および遅延を管理するメカニズムを提供し、これによって、定期的な統計を収集できます。

キュー・モニター・プロセスは、プロセスの障害がインスタンスの障害の原因にならないという点が、他のほとんどの Oracle バックグラウンド・プロセスとは異なります。

初期化パラメータ AQ_TM_PROCESSES では、インスタンスの起動時に、1 つ以上のキュー・モニター・プロセスの作成を指定します。

関連項目：

- 18-9 ページの「[実行の時間枠](#)」
- 18-13 ページの「[キューイングの統計](#)」

アドバンスト・キューイングの機能

ここでは、Oracle アドバンスト・キューイングの主な機能を説明します。

構造化ペイロード

ペイロードを構造化し管理するために、オブジェクト型を使用できます。(RAW データ型は、非構造化ペイロードに使用できます。)

統合されたデータベース・レベルの操作サポート

Oracle Advanced Queuing では、メッセージは表の中に格納されます。すべての標準データベース機能がサポートされます。

SQL によるアクセス

メッセージはデータベース・レコードとして格納されます。SQL を使用することにより、メッセージのプロパティ、メッセージの履歴およびペイロードにアクセスできます。索引などの使用可能なすべての SQL テクノロジを使用することにより、メッセージへのアクセスを最適化できます。

AQ_ADMINISTRATOR ロールにより、キューに関する情報にアクセスできます。

関連項目： ロールについては、『Oracle8i 管理者ガイド』を参照してください。

実行の時間枠

特定の時間枠内にメッセージを消費するように指定できます。メッセージには、指定した時間（遅延時間）の経過後に限って処理できること、および指定した制限時間が切れる前に消費する必要があることを示すマークを設定できます。

AQ_TM_PROCESS 初期化パラメータを指定すると、キュー・メッセージを時間で監視できるようになります。これは、遅延プロパティおよび時間切れプロパティを指定するメッセージのために使用されます。定期的な統計を収集する場合は、時間の監視も使用可能にしておく必要があります。

このパラメータを 1 に設定すると、Oracle は、メッセージを監視するバックグラウンド・プロセスとして、「キュー・モニター・プロセス」(QMN0) を 1 つ作成します。このパラメータを 2 ～ 10 に設定すると、Oracle は指定された数の QMN n プロセスを作成します。このパラメータを指定しないか 0 に設定すると、キュー・モニター・プロセスは作成されません。キュー・モニター操作を開始および停止する DBMS_AQADM パッケージ内のプロシージャは、このパラメータによって少なくとも 1 つのキュー・モニター・プロセスがインスタンス起動の一部として起動された場合にのみ有効になります。

関連項目：

- 18-13 ページの「[キューイングの統計](#)」
- AQ_TM_PROCESS 初期化パラメータについては、『Oracle8i リファレンス・マニュアル』を参照してください。

1 メッセージに対して複数のコンシューマ

1つのメッセージを、複数のコンシューマで使用できます。マルチ・コンシューマ・キューを使用すると、1つのメッセージを複数のコンシューマが使用できます。

ナビゲーション

キューからメッセージを選択するには、いくつかの方法があります。最初のメッセージを選択できますが、メッセージを選択して一貫読みスナップショットを確定すると、カレント・スナップショットに基づいて次のメッセージを取り出せます。新しい一貫読みスナップショットは、キューから最初のメッセージを選択するたびに取得されます。

さらに、メッセージの相関識別子を使用して特定のメッセージを取り出すこともできます。

メッセージの優先順位と順序付け

メッセージの消費順序を指定するには、ソート順（キュー内のすべてのメッセージの順序を決めるために使用するプロパティを指定します）、優先順位（各メッセージに割り当てられます）および順序逸脱（あるメッセージを他のメッセージとの関係で配置します）の3種類のオプションがあります。

複数のコンシューマが同じキューに対するものである場合、ひとりのコンシューマはすぐに使用できる最初のメッセージを取り出します。別のコンシューマが使用中のメッセージはスキップします。

DEQUEUE のモード

DEQUEUE 要求により、メッセージをブラウズしたり取消したりできます。メッセージをブラウズしても、その後の処理に引き続き使用できます。メッセージを取り消すと、DEQUEUE 要求には使用できなくなります。キューのプロパティによっては、取り消されたメッセージがキュー表内に保持されることもあります。

メッセージ着信の待機

空のキューに対しても DEQUEUE を発行できます。要求がメッセージの着信を待機するかどうかと、待機する場合にはその待機時間を指定できます。

遅延による再試行

メッセージの使用は、1 度かぎりであることが必要です。メッセージのデキューに失敗し、トランザクションがロールバックされた場合、メッセージは、ユーザー指定の遅延時間経過後に再処理できるようになります。指定した限度まで再処理が試行されます。

例外キュー

特定の制約内、つまり実行時間枠または再試行の制限内に、メッセージを使用できないことがあります。このような条件が生じると、メッセージはユーザー指定の例外キューに移されます。

可視性

ENQUEUE/DEQUEUE 要求は、多くの場合、その要求を含むトランザクションの一部となっています。これによりトランザクションは適切に動作します。ただし、要求の結果を他のトランザクションからすぐに参照できるようにするために、要求自体をトランザクションとして指定することもできます。

メッセージのグループ化

特定のキューに入っているメッセージを、一度に 1 人のユーザーのみが使用できる集合となるようグループ化できます。そのためには、メッセージのグループ化の可能なキュー表内にキューを作成する必要があります。

1 つのグループに属するメッセージはすべて同じトランザクション内に作成される必要があります。1 つのトランザクション内に作成されたメッセージはすべて同じグループに属します。この機能により、複雑なメッセージを単純なメッセージにセグメント化できるようになります。たとえば、送付状が入っているキューに送信されるメッセージの場合、ヘッダー・メッセージで開始し、その後に明細を示すメッセージ、その後に後書きメッセージが続くメッセージ・グループとして構成できます。

保存

メッセージは、使用後もそれを保存するよう指定できます。これにより、関連したメッセージの履歴を保存できます。履歴は、追跡操作、データ・ウェアハウスの操作およびデータ探索操作のために使用できます。

メッセージの履歴

Oracle Advanced Queuing は、各メッセージの履歴の情報を保存します。この情報には、ENQUEUE/DEQUEUE の時間が記録されており、各要求を実行したトランザクションが示されています。

追跡

メッセージを保持する場合、メッセージを相互に関連付けることができます。たとえば、メッセージ *m1* の使用結果としてメッセージ *m2* が生成された場合、*m1* は *m2* に関連付けられます。それにより、関連付けられた一連のメッセージを追跡できるようになります。このメッセージ列は、アプリケーションによって組み立てられることの多い「イベント・ジャーナル」を表すものです。Oracle Advanced Queuing は、アプリケーションが自動的にイベント・ジャーナルを作成するように設計されています。

キュー・レベルのアクセス制御

Oracle 8i では、8.1 形式のキューの所有者が、キューに対するキュー・レベルの権限を付与または取消しできます。DBA は、任意のデータベース・ユーザーに、新しい AQ システム・レベルの権限を付与したり、取り消したりできます。また、任意のデータベース・ユーザーを AQ 管理者にすることもできます。

他のデータベースへのメッセージの伝播

あるデータベースにエンキューされたメッセージは、別のデータベースのキューに伝播できます。ソース・キューと宛先キューのデータ型は一致する必要があります。

メッセージの伝播により、アプリケーションは、同じデータベースや同じキューに接続していなくても、相互に通信できます。伝播では、ローカルまたはリモート・データベース間のデータベース・リンクと Net8 を使用します。どちらも Oracle Advanced Queuing が使用可能になっている必要があります。

メッセージの伝播をスケジューリングまたはスケジュール解除したり、開始時刻、時間枠および以降の伝播の時間枠の日付関数を定期的なスケジュールで指定したりできます。データ・ディクショナリ・ビュー DBA_QUEUE_SCHEDULES は、メッセージ伝播の現行のスケジュールを示します。

ジョブ・キュー (SNP*n*) バックグラウンド・プロセスは、メッセージの伝播を処理します。伝播を使用可能にするには、初期化パラメータ JOB_QUEUE_PROCESSES を使用して少なくとも 1 つのジョブ・キュー・プロセスを起動する必要があります。

伝播に関する統計

伝播統計は、スケジュールの範囲内で伝播された合計メッセージ数 / 平均メッセージ数 / バイト数の形式で取得できます。この情報を使用して、メッセージ伝播のパフォーマンスをチューニングできます。

非永続キュー

AQ は、非永続メッセージをサブスクライバに非同期に送信できます。これらのメッセージはイベント・ドリブン方式であり、システム（またはインスタンス）障害の発生後は存続しません。AQ では、共通 API を通じて永続メッセージと非永続メッセージがサポートされます。

パブリッシュ / サブスクライブのサポート

アプリケーション間でパブリッシュ / サブスクライブ方式のメッセージングを行えるように、一連の機能の組合せが導入されています。これらの機能には、ルールベース・サブスクライバ、メッセージ伝播、リスニング機能および通知機能があります。トリガーを使用すると、システム・イベントとユーザー・イベントを発行できます。

関連項目： 19-19 ページの「[システム・イベントとユーザー・イベントのトリガー](#)」

Oracle Parallel Server 環境のサポート

アプリケーションは、キュー表に対するインスタンス親和性を指定できます。AQ をパラレル・サーバーおよび複数インスタンスと併用する場合は、キュー・モニターのスケジューリング用にインスタンス間でキュー表をパーティション化するために、この情報が使用されます。キュー表は、ユーザーが指定したインスタンスのキュー・モニターによって監視されます。インスタンスの親和性を指定しなければ、キュー表は使用可能なインスタンス間で任意に分割されます。キュー表にアクセスするアプリケーションと、それを監視するキュー・モニターの間で ping できます。このインスタンス親和性を指定すると、アプリケーションは他のインスタンスからのキュー表とそのキューへのアクセスが可能となります。

この機能により、異なるインスタンスで実行中のキュー・モニターと AQ 伝播ジョブの間での ping が防止されます。Oracle8i リリース 8.1.5 では、キュー表についてインスタンス親和性（1 次および 2 次）を指定できます。AQ をパラレル・サーバーおよび複数インスタンスと併用する場合は、キュー・モニタースケジューリングおよび伝播用に、インスタンス間でキュー表をパーティション化するために、この情報が使用されます。キュー表は、常に 1 つのインスタンスとの親和性を持ちます。親和性を明示的に指定しなければ、使用可能な任意のインスタンスがキュー表の所有者となります。キュー表の所有者がいなくなると、2 次インスタンスまたは使用可能な他のインスタンスが、そのキュー表の所有権を引き継ぎます。

キューイングの統計

Oracle Advanced Queuing は、キューイング・システムの現在の状態に関する統計と定期的な統計を、動的表 GV\$AQ に保持します。

キューイング・システムの現行の状態に関する統計には、準備完了、待ち状態および期限切れのメッセージの個数が含まれます。

1 つ以上のキュー・モニター・プロセスを起動して、次のような定期的な統計を保持する必要があります。

- 各状態（準備完了、待ち状態および期限切れ）にあるメッセージの個数
- 待ち状態メッセージの平均待ち時間
- 待ち状態メッセージの合計待ち時間

非同期通知

クライアントは、ブロックするデキュー・コールを使用するかわりに、そのクライアントの選択条件と一致するメッセージがキューに入った時点で通知を受信するように選択できます。

Java Messaging Service クライアントは、`MessageListeners` に登録してメッセージを非同期に受信できます。Oracle Advanced Queuing は、そのクライアントに対するメッセージを受信すると、`onMessage` コールバックを実行します。

OCI クライアントは、新しいコールである `OCISubscriptionRegister` を使用して、メッセージ通知のコールバックを登録できます。クライアントは、サブスクリプション名とコールバックを指定する登録コールを発行します。サブスクリプション・メッセージが受信されると、コールバックが起動されます。コールバックは、メッセージを取り出すために明示的なデキューを発行できます。

リスニング機能（複数キューでのメッセージ待機）

リスニング・コールを使用すると、複数のキュー上のメッセージを待機できます。ゲートウェイ・アプリケーションは、このコールを使用して 1 組のキューを監視できます。また、サブスクリプション・リストのメッセージを待機することもできます。Oracle Advanced Queuing のネイティブ・インタフェース（PL/SQL または OCI）の場合、リスニング・コールはブロック化コールです。リスニングによって正常に値が戻される場合は、デキューを使用してメッセージを取り出す必要があります。

Java Messaging Service インタフェースでは、`Session Message Listener` を使用して複数のキュー上のメッセージをリスニングできます。これは、ブロックしません。メッセージがセッション中にコンシューマのいずれかに現れると、`MessageListener` の `onMessage` コールバックが起動されます。

相関識別子

各メッセージには識別子を割り当てることができます。この識別子を使用することによって、特定のメッセージを取り出せます。

インポート/エクスポート

キューをインポートまたはエクスポートすると、その基礎となるキュー表に関連付けられたディクショナリ表がインポートまたはエクスポートされます。キューのインポートおよびエクスポートは、キュー表単位でしか実行できません。

キュー表をエクスポートすると、表定義情報とキュー・データの両方がエクスポートされます。キュー表をインポートすると、エクスポート・アクションの手順によりキュー・ディクショナリが維持されます。キュー表のデータもエクスポートされるため、キュー表のデータのトランスポート時には、アプリケーション・レベルでのデータの整合性が維持されるようユーザー側で注意する必要があります。

複数のレスピエントをサポートするキュー表には、重要なキューのメタデータが含まれる索引構成表があります。このメタデータはキューの操作の基本となるものであるため、インポート後にこの表に含まれるキューが動作するためには、キュー表のみではなく、その索引構成表もエクスポートしインポートする必要があります。

キュー表を含むスキーマをエクスポートすると、索引構成表も自動的にエクスポートされます。インポートの場合も、同じようになります。メタデータ表にはキュー表にあるいくつかの行の ROWID が含まれるため、インポート実行時には、メタデータ表をインポートする時点で古くなった ROWID について通知されます。キューイング・システムは、インポート・プロセスの一部として古い ROWID を自動的に訂正するため、このメッセージは無視できます。ただし、インポート時に別の問題（ロールバック・セグメントの領域をすべて使用したなど）が生じた場合は、その問題を訂正してからインポートを再実行する必要があります。

関連項目：

- 『Oracle8i アプリケーション開発者ガイド アドバンスト・キューイング』
- 『Oracle8i ユーティリティ・ガイド』

この章では、データベース・トリガーについて説明します。データベース・トリガーとは、PL/SQL、Java または C で記述され、データベースに格納されているプロシージャであり、表またはビューが変更された場合、またはなんらかのユーザー・アクションやデータベース・システム・アクションが発生した場合に、暗黙的に実行（起動）されます。トリガーが起動されるのは、特定のスキーマ・オブジェクトでの DML 文、スキーマまたはデータベース内で発行された DDL 文、ユーザーのログオンまたはログオフ・イベント、サーバー・エラー、データベース起動またはインスタンス停止のいずれかが発生した場合です。

この章の内容は、次のとおりです。

- [トリガーの概要](#)
- [トリガーの各部分](#)
- [トリガーのタイプ](#)
- [トリガーの実行](#)

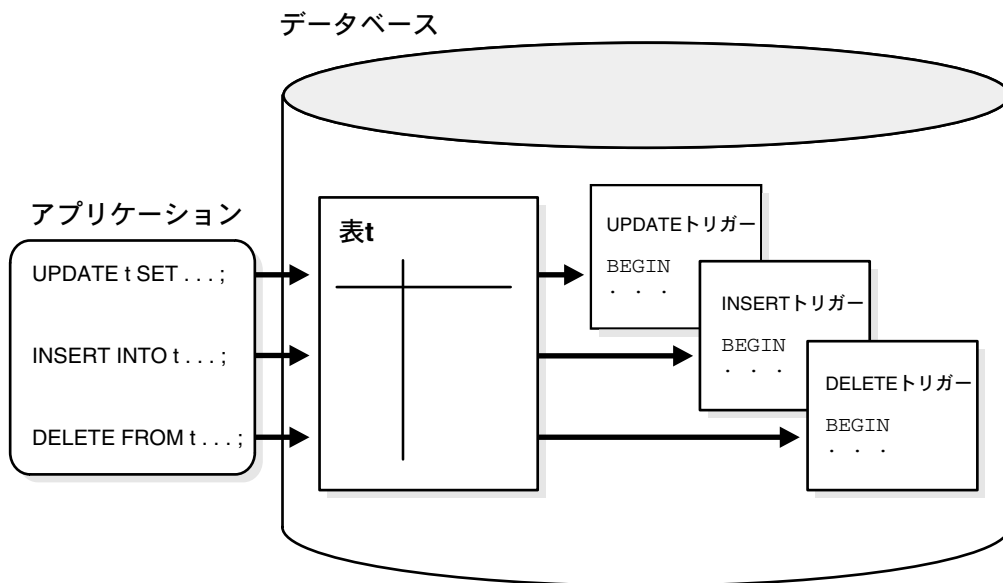
トリガーの概要

Oracle では、表（または、場合によってはビュー）に対して INSERT 文、UPDATE 文または DELETE 文を発行するか、データベース・システム・アクションが発生したときに暗黙的に実行されるプロシージャを定義できます。このようなプロシージャを「トリガー」と呼びます。これらのプロシージャは、PL/SQL または Java で記述してデータベースに格納するか、C のコールアウトとして記述できます。

トリガーは、ストアド・プロシージャに似ています。データベースに格納されているトリガーには、1 単位として実行可能な SQL や PL/SQL 文を格納することができ、また、トリガーは他のストアド・プロシージャを起動できます。ただし、トリガーとプロシージャとは、起動する方法が異なります。プロシージャは、ユーザー、アプリケーションまたはトリガーによって明示的に実行されます。トリガーは、接続ユーザーや使用中のアプリケーションに関係なく、トリガー・イベントが発生した時点で暗黙的に起動されます。

図 19-1 に示すデータベース・アプリケーションには、データベースに格納された複数のトリガーを暗黙のうちに起動する SQL 文がいくつか含まれています。データベースには、トリガーとそれに対応付けられている表が別々に格納されていることに注目してください。

図 19-1 トリガー



また、トリガーは C プロシージャをコールアウトし、計算集中型の操作に使用できます。

トリガーを起動するイベントは、次のとおりです。

- 表中のデータを変更する DML 文（INSERT、UPDATE または DELETE）
- DDL 文
- 起動、停止およびエラー・メッセージなどのシステム・イベント
- ログオンやログオフなどのユーザー・イベント

注意： Oracle Forms でも、種類の違うトリガーを定義、格納および実行できます。ただし、この章で説明するトリガーと Oracle Forms トリガーを混同しないでください。

関連項目：

- トリガーとストアド・プロシージャの類似点の詳細は、[第 17 章「プロシージャとパッケージ」](#)を参照してください。
- 19-7 ページの「[トリガー・イベントまたはトリガーを実行する文](#)」

トリガーの使用方法

トリガーは Oracle の標準機能を補い、高度にカスタマイズされたデータベース管理システムを提供します。たとえば、トリガーによって、表に対する DML 操作を通常の業務時間内のみに制限できます。また、トリガーを使用すると、平日の特定の時間にしか DML 操作が実行されないようにも制限できます。トリガーには、その他に次のような使用方法があります。

- 導出列の値の自動的な生成
- 不正なトランザクションの防止
- 複雑なセキュリティ認可の規定
- 分散データベース内の複数ノードにまたがる参照整合性の規定
- 複雑なビジネス・ルールの規定
- 透過的なイベント・ロギングの実現
- 高度な監査の実現
- 表レプリケーションの同期の維持
- 表アクセスについての統計情報の収集
- ビューに対して DML 文が発行された場合の表データの変更

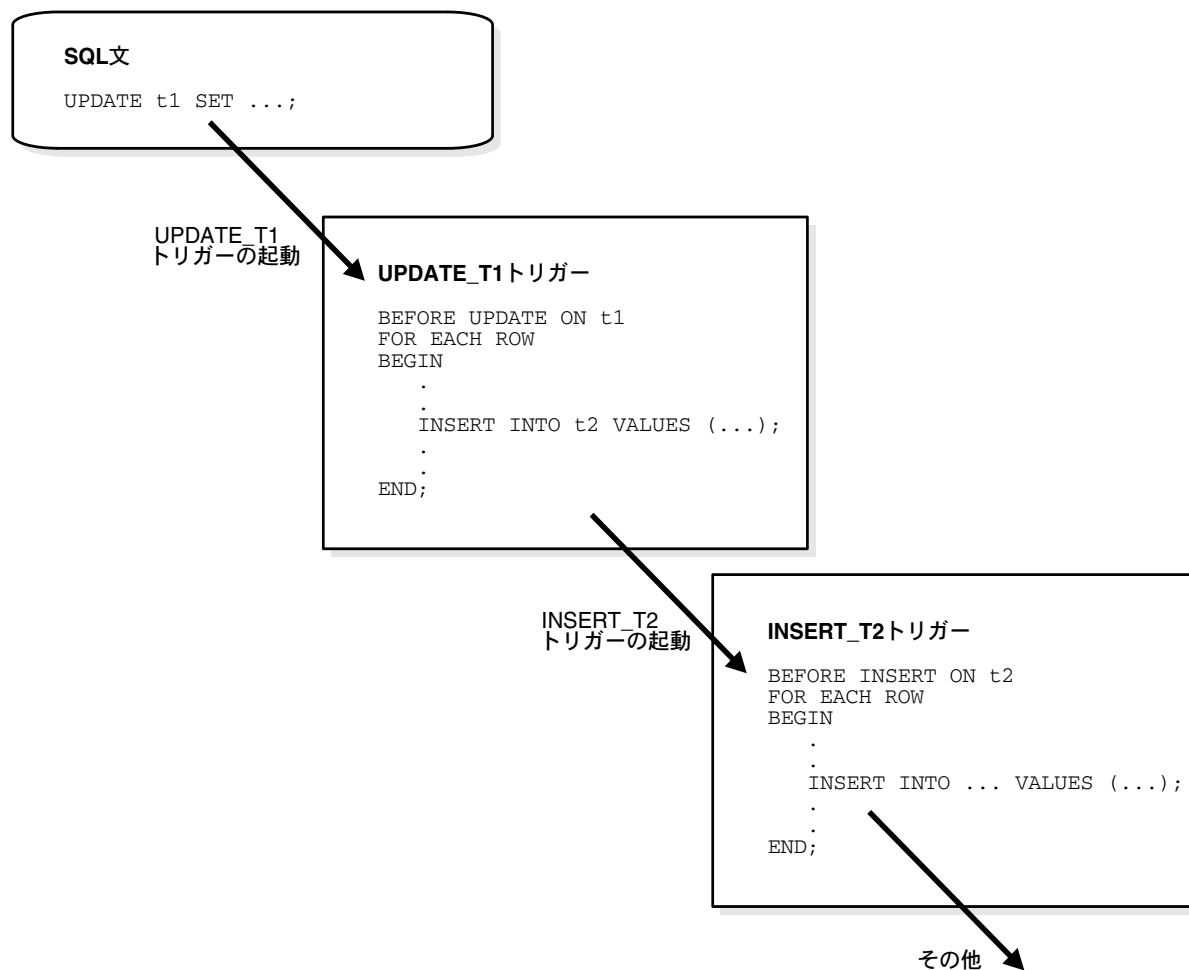
- データベース・イベント、ユーザー・イベントおよび SQL 文に関する情報の、サブスクライバ・アプリケーションに対する発行

関連項目： トリガーの使用例は、『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。

トリガーの使用上の注意

トリガーはデータベースのカスタマイズに使用すると便利ですが、必要な場合にのみ使用してください。トリガーを使用しすぎると相互依存性が複雑になるため、大規模アプリケーションのメンテナンスが困難になります。たとえば、トリガーが起動すると、そのトリガー・アクションに含まれる SQL 文によって他のトリガーが起動され、「トリガーが連鎖状態になる」ことがあります。これにより、意図しない影響を生じるおそれがあります。[図 19-2](#) に、この状態を示します。

図 19-2 連鎖的なトリガー



トリガーと宣言整合性制約

トリガーと整合性制約を併用すると、任意の整合性規則を定義して規定できます。ただし、トリガーを使用してデータ入力に制約を適用するのは、次の場合に限定してください。

- 子表と親表が分散データベースの別々のノードにあるときに、参照整合性を規定する場合
- 整合性制約では定義できない複雑なビジネス・ルールを規定する場合
- 必要な参照整合性規則を、次の整合性制約を使用して規定できない場合
 - NOT NULL、UNIQUE キー
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK
 - DELETE CASCADE
 - DELETE SET NULL

関連項目： 整合性制約の詳細は、25-4 ページの「[Oracle がデータの整合性を規定する方法](#)」を参照してください。

トリガーの各部分

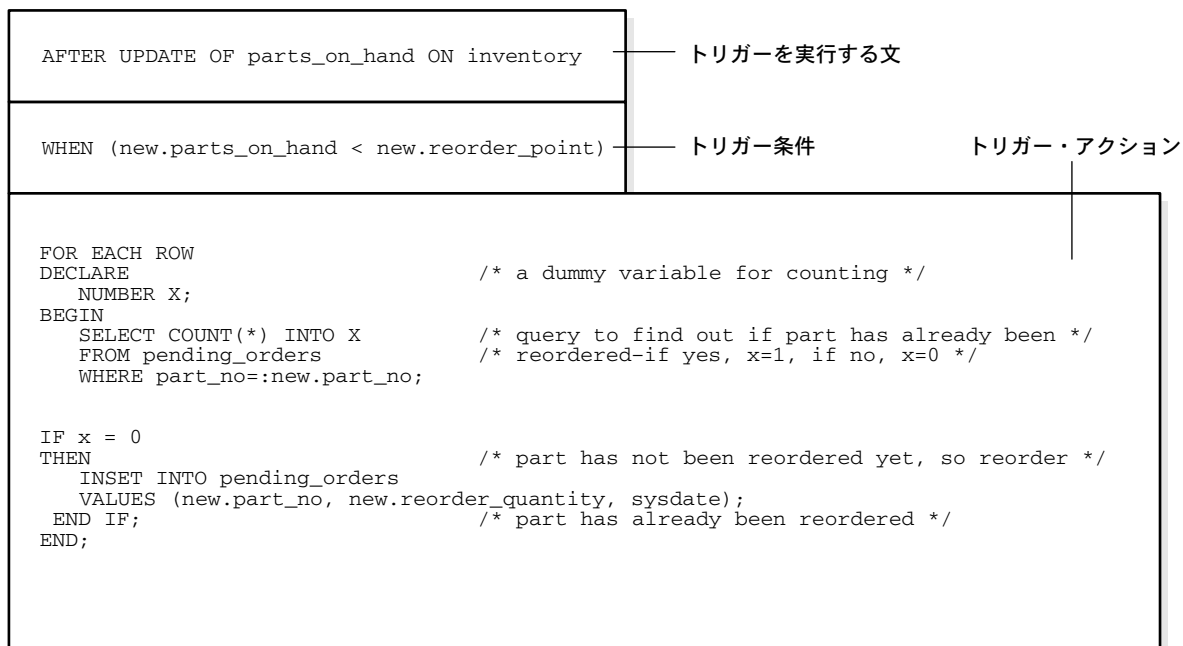
トリガーには次の3つの基本部分があります。

- トリガー・イベントまたはトリガーを実行する文
- トリガー条件
- トリガー・アクション

[図 19-3](#) に、トリガーの各部分を示します。この図は、正確な構文を示すためのものではありません。トリガーの各部分については、この後で詳しく説明します。

図 19-3 REORDER トリガー

REORDER トリガー



トリガー・イベントまたはトリガーを実行する文

トリガー・イベントまたはトリガーを実行する文とは、トリガーを起動させる SQL 文、データベース・イベントまたはユーザー・イベントのことです。トリガー・イベントは、次に示すものの 1 つ以上からなります。

- 特定の表（または、場合によってはビュー）に対する INSERT 文、UPDATE 文または DELETE 文
- スキーマ・オブジェクトに対する CREATE 文、ALTER 文または DROP 文
- データベース起動またはインスタンス停止
- 特定または任意のエラー・メッセージ
- ユーザーのログオンまたはログオフ

たとえば、図 19-3 のトリガーを実行する文は次のとおりです。

```
... UPDATE OF parts_on_hand ON inventory ...
```

この文は、INVENTORY 表の PARTS_ON_HAND 列を更新すると、トリガーが起動することを意味します。なお、トリガー・イベントが UPDATE 文の場合は、どの列が更新されるとトリガーが起動されるかを示す列リストを指定できます。INSERT 文と DELETE 文では情報行全体が処理の対象になるため、それらの文には列のリストを指定できません。

トリガー・イベントには、次のように複数の SQL 文を指定できます。

```
... INSERT OR UPDATE OR DELETE OF inventory ...
```

この場合、INVENTORY 表に対して INSERT 文、UPDATE 文または DELETE 文を発行すると、トリガーが起動されます。複数のタイプの SQL 文でトリガーが起動される場合は、条件述語を使用して、トリガーを実行する文のタイプを検出できます。これにより、トリガーを起動した文のタイプに応じて異なるコードを実行するトリガーを作成できます。

トリガー条件

トリガー条件にはブール式を指定します。トリガーが起動されるには、このブール式が TRUE になる必要があります。トリガー条件の評価が FALSE または UNKNOWN になった場合、トリガー・アクションは実行されません。この例では、次のようにしてトリガーを制限しています。

```
new.parts_on_hand < new.reorder_point
```

この場合、引当可能部品の数量が現行の再発注数量より少なくなるまで、トリガーは起動しません。

トリガー・アクション

トリガー・アクションは、SQL 文またはコードを含むプロシージャ（PL/SQL ブロック、Java プログラムまたは C コールアウト）です。これらの SQL 文やコードは、次のイベントが発生すると実行されます。

- トリガーを実行する文が発行されたとき
- トリガー条件が TRUE と評価されたとき

ストアド・プロシージャと同様に、トリガー・アクションでは次のことが可能です。

- SQL 文、PL/SQL 文または Java 文の使用
- 変数、定数、カーソル、例外など、PL/SQL の言語構文の定義
- Java 言語構文の定義
- ストアド・プロシージャのコール

トリガーが行トリガーの場合、トリガー・アクション内の文から、トリガーによって処理されている行の列値にアクセスできます。各列の古い値と新しい値には、相関名によってアクセスできます。

トリガーのタイプ

この項では、様々なタイプのトリガーについて説明します。

- 行トリガーと文トリガー
- BEFORE トリガーと AFTER トリガー
- INSTEAD OF トリガー
- システム・イベントとユーザー・イベントのトリガー

行トリガーと文トリガー

トリガーを定義するときに、次のようにトリガー・アクションの実行回数を指定できます。

- 多数の行を更新する UPDATE 文によって起動されるトリガーなどの場合、トリガーを実行する文の影響を受ける行ごとに 1 回ずつ
- 影響を受ける行数に関係なく、トリガーを実行する文ごとに 1 回ずつ

行トリガー

「行トリガー」は、表がトリガーを実行する文の処理の影響を受けるたびに実行されます。たとえば、UPDATE 文が表の複数の行を更新した場合、UPDATE 文が処理する行ごとに 1 つずつ行トリガーが起動されます。トリガーを実行する文の影響を受ける行がなければ、行トリガーは実行されません。

トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータまたは影響を受ける行数に依存する場合には、行トリガーが便利です。たとえば、[図 19-3](#) は、トリガーを実行する文の影響を受ける各行の値を使用する行トリガーの一例です。

文トリガー

「文トリガー」は、トリガーを実行する文の影響を受ける表の行数とは無関係に、また、影響を受ける行がない場合にも、トリガーを実行する文の実行毎に 1 回起動されます。たとえば、DELETE 文が表の複数の行を削除すると、文レベルの DELETE トリガーが 1 回のみ起動されます。

トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータや、影響を受ける行に依存しない場合には、文トリガーが便利です。文トリガーの使用例を次に示します。

- 現在時刻やカレント・ユーザーについて複雑なセキュリティ・チェックを実行する場合
- 単一の監査レコードを生成する場合

BEFORE トリガーと AFTER トリガー

トリガーを定義するときに、「トリガーのタイミング」を指定できます。これは、トリガー・アクションを、トリガーを実行する文の前に実行するのか、後に実行するのかを指定するものです。BEFORE トリガーと AFTER トリガーは、文トリガーと行トリガーの両方に適用されます。

DML 文によって起動される BEFORE トリガーと AFTER トリガーは、表にのみ定義でき、ビューには定義できません。ただし、ビューに対して INSERT 文、UPDATE 文または DELETE 文を発行すると、そのビューのベース表のトリガーが起動されます。DDL 文によって起動される BEFORE トリガーと AFTER トリガーは、データベースまたはスキーマにのみ定義でき、特定の表には定義できません。

関連項目：

- 19-13 ページの「[INSTEAD OF トリガー](#)」
- BEFORE および AFTER トリガーを使用して DML 文と DDL 文に関する情報を発行する方法は、19-19 ページの「[システム・イベントとユーザー・イベントのトリガー](#)」を参照してください。

BEFORE トリガー

BEFORE トリガーは、トリガー文を実行する前にトリガー・アクションを実行します。このタイプのトリガーは、次のような場合によく使用します。

- トリガー文を実行してよいかどうかを、トリガー・アクションによって決める場合。
BEFORE トリガーを使用することにより、トリガー・アクションで例外が発生した場合に、トリガー文で無駄な処理を実行して結果的にロールバックすることになるのを避けられます。
- トリガーを実行する INSERT または UPDATE 文を実行する前に、特定の列値を調べる場合。

AFTER トリガー

AFTER トリガーは、トリガー文を実行した後でトリガー・アクションを実行します。

トリガー・タイプの組合せ

これまでに説明したオプションを使用して、次の 4 タイプの行トリガーと文トリガーを作成できます。

- **BEFORE 文トリガー**

トリガー文を実行する前にトリガー・アクションが実行されます。

- **BEFORE 行トリガー**

トリガーを実行する文の影響を受ける各行が修正される前、かつ該当する整合性制約の検査の前に、トリガー条件に違反していない限りトリガー・アクションが実行されます。

- **AFTER 行トリガー**

トリガーを実行する文の影響を受ける各行が修正され、該当する整合性制約が適用された後で、トリガー条件に違反していない限りカレント行に対してトリガー・アクションが実行されます。BEFORE 行トリガーと異なり、AFTER 行トリガーでは行がロックされます。

- **AFTER 文トリガー**

トリガー文を実行し、遅延整合性制約を適用した後に、トリガー・アクションが実行されます。

ある表に対する 1 つの文に、同じタイプのトリガーを複数指定できます。たとえば、EMP 表に対する UPDATE 文に 2 つの BEFORE 文トリガーを指定できます。同じタイプのトリガーを複数指定することにより、同じ表に対してトリガーを持つ複数のアプリケーションをモジュール化してインストールできます。また、Oracle スナップショット・ログは AFTER 行トリガーを使用するため、Oracle によって定義される AFTER 行トリガーに加えて、ユーザー独自の AFTER 行トリガーを設計できます。

各種 DML 文（INSERT、UPDATE または DELETE）に対して、前述のトリガーを必要な数だけ作成できます。

たとえば、表 SAL で、表がアクセスされる時期と発行される問合せのタイプを確認する場合を考えます。次の例は、表 SAL に対するアクション（UPDATE、DELETE または INSERT など）の時間とタイプ別にその情報を記録するサンプルのパッケージとトリガーを示しています。グローバル・セッション変数 STAT.ROWCNT は、BEFORE 文トリガーによって 0 に初期化されます。その後、行トリガーが実行されるたびに増やされます。最終的に、AFTER 文トリガーによって、統計情報が表 STAT_TAB に保存されます。

SAL 表のサンプル・パッケージおよびトリガー

```
DROP TABLE stat_tab;
CREATE TABLE stat_tab(utype CHAR(8),
                      rowcnt INTEGER, uhour INTEGER);

CREATE OR REPLACE PACKAGE stat IS
  rowcnt INTEGER;
END;
/

CREATE TRIGGER bt BEFORE UPDATE OR DELETE OR INSERT ON sal
BEGIN
  stat.rowcnt := 0;
END;
/

CREATE TRIGGER rt BEFORE UPDATE OR DELETE OR INSERT ON sal
FOR EACH ROW BEGIN
  stat.rowcnt := stat.rowcnt + 1;
END;
/

CREATE TRIGGER at AFTER UPDATE OR DELETE OR INSERT ON sal
DECLARE
  typ CHAR(8);
  hour NUMBER;
BEGIN
  IF updating
  THEN typ := 'update'; END IF;
  IF deleting THEN typ := 'delete'; END IF;
  IF inserting THEN typ := 'insert'; END IF;

  hour := TRUNC((SYSDATE - TRUNC(SYSDATE)) * 24);
  UPDATE stat_tab
    SET rowcnt = rowcnt + stat.rowcnt
    WHERE utype = typ
    AND uhour = hour;
  IF SQL%ROWCOUNT = 0 THEN
    INSERT INTO stat_tab VALUES (typ, stat.rowcnt, hour);
  END IF;

EXCEPTION
  WHEN dup_val_on_index THEN
    UPDATE stat_tab
      SET rowcnt = rowcnt + stat.rowcnt
```

```
WHERE utype = typ
      AND uhour = hour;
END;
/
```

INSTEAD OF トリガー

注意： INSTEAD OF トリガーを使用できるのは、Oracle8i Enterprise Edition を購入した場合のみです。これらのトリガーは、リレーショナル・ビューとオブジェクト・ビューで使用できます。

INSTEAD OF トリガーを使用すると、DML 文（INSERT、UPDATE および DELETE）で直接変更できないビューを透過的に変更できるようになります。他のタイプのトリガーとは異なり、Oracle はトリガー文を実行する「かわりに（instead of）」このトリガーを起動するため、このようなトリガーを INSTEAD OF トリガーと呼びます。

通常の INSERT、UPDATE および DELETE 文をビューに対して記述し、それに応じて基礎となる表が更新されるように INSTEAD OF トリガーを起動させることができます。INSTEAD OF トリガーは、変更があったビューの行ごとにアクティブ化されます。

ビューの変更

ビューを変更する操作では、次のように曖昧な結果を生じることがあります。

- ビューで行を削除する操作は、ベース表から行を実際に削除する操作と、列値をいくつか更新してその行がビューに選択されないようにする操作のどちらかを意味します。
- ビューに行を挿入する操作は、ベース表に新しい行を実際に挿入する操作と、既存の行を更新してビューに表示されるようにする操作のどちらかを意味します。
- 結合が含まれるビューで列を更新すると、ビューに表示されない別の列の意味が変わってしまうことがあります。

オブジェクト・ビューには、その他にも問題があります。たとえば、オブジェクト・ビューの主な使用法は、マスターとディテールの関連を表すことです。この操作で結合が関係してくるのは避けられませんが、結合の変更は本質的に曖昧です。

その結果、変更可能なビューについては、たくさんの制限事項があります（次の項を参照）。INSTEAD OF トリガーは、それ以外の手段では変更できないリレーショナル・ビューのみでなく、オブジェクト・ビューに対しても使用できます。

ビューが本来は変更可能であっても、挿入、更新または削除する値の妥当性検査を実行する必要があります。この場合にも、INSTEAD OF トリガーを使用できます。変更される行の妥当性チェックがトリガー・コードによって実行され、有効であれば、基礎となる表に変更が伝播されます。

INSTEAD OF トリガーにより、OCI を使用してクライアント側のオブジェクト・ビューのインスタンスを変更することもできます。オブジェクト・ビューによって具象化されたオブジェクトを、クライアント側のオブジェクト・キャッシュ内で修正し、それを永続的な格納領域にフラッシュ・バックするには、オブジェクト・ビューが変更可能でない限り、INSTEAD OF トリガーを指定する必要があります。ただし、オブジェクト・キャッシュ内のビュー・オブジェクトをピンして読み込むのみであれば、これらのトリガーを定義する必要はありません。

関連項目：

- [第 14 章「オブジェクト・ビュー」](#)
- 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』

変更不可能なビュー

INSTEAD OF トリガーを使用せずにデータを挿入、更新または削除でき、かつ後述の条件に合致する場合、このビューは「本質的に変更可能」です。ビュー問合せに次の言語構文が含まれている場合、そのビューは本質的に変更不可能であるため、そのビューに対しては挿入、更新または削除を実行できません。

- 集合演算子
- 集計関数
- GROUP BY、CONNECT BY または START WITH 句
- DISTINCT 演算子
- 結合（ただし、一部の結合ビューは更新可能）

ビューに疑似列または式が含まれる場合、そのビューを更新するには、その疑似列または式を参照しない UPDATE 文を使用する方法しかありません。

関連項目： [10-16 ページの「更新可能な結合ビュー」](#)

INSTEAD OF トリガーの例

次の例は、すべての部門とそのマネージャをリストする manager_info ビューの各行を更新する INSTEAD OF トリガーを示しています。

dept は、部門リストを含むリレーショナル表であるとしてします。

```
CREATE TABLE dept (  
    deptno NUMBER PRIMARY KEY,  
    deptname VARCHAR2(20),  
    manager_num NUMBER  
);
```

emp は、従業員とその所属部門のリストを含むリレーショナル表であるとしします。

```
CREATE TABLE emp (  
    empno NUMBER PRIMARY KEY,  
    empname VARCHAR2(20),  
    deptno NUMBER REFERENCES dept(deptno),  
    startdate DATE  
);  
ALTER TABLE dept ADD (FOREIGN KEY(manager_num) REFERENCES emp(empno));
```

各部門のすべてのマネージャをリストする manager_info ビューを作成します。

```
CREATE VIEW manager_info AS  
    SELECT d.deptno, d.deptname, e.empno, e.empname  
    FROM   emp e, dept d  
    WHERE  e.empno = d.manager_num;
```

次に、このビューへの挿入を処理する INSTEAD OF トリガーを定義します。
manager_info ビューへの挿入を、dept 表の manager_num 列の更新に変換できます。

このトリガーでは、だれかを部門のマネージャにするには、その部門に最低 1 人は従業員が所属する必要があるという制約も規定できます。

```
CREATE TRIGGER manager_info_insert  
    INSTEAD OF INSERT ON manager_info  
    REFERENCING NEW AS n          -- new manager information  
    FOR EACH ROW  
    DECLARE  
        empCount NUMBER;  
    BEGIN  
  
        /* First check to make sure that the number of employees  
         * in the department is greater than one */  
        SELECT COUNT(*) INTO empCount  
        FROM emp e  
        WHERE e.deptno = :n.deptno;
```

```
/* If there are enough employees then make him or her the manager */
IF empCount >= 1 THEN

    UPDATE dept d
        SET manager_num = :n.empno
        WHERE d.deptno = :n.deptno;

    END IF;
END;
/
```

manager_info ビューへの次のような挿入を考えます。

```
INSERT INTO manager_info VALUES (200, 'Sports', 1002, 'Jack');
```

この文によって manager_info_insert トリガーが起動され、基礎となる表が更新されます。ビューに対する INSERT および DELETE の場合も、類似のトリガーを使用して適切なアクションを指定できます。

使用上の注意事項

CREATE TRIGGER 文の INSTEAD OF 句を指定できるのは、ビューに作成するトリガーの場合のみです。ビューに作成するトリガーには、BEFORE および AFTER 句は使用できません。

ビューに対する挿入または更新に INSTEAD OF トリガーが使用される場合、ビューの CHECK 句は実行されません。このチェックは、INSTEAD OF トリガー本体で規定します。

関連項目：

- 『Oracle8i アプリケーション開発者ガイド 基礎編』
- 『Oracle8i SQL リファレンス』の CREATE TRIGGER 文

NESTED TABLE に対する INSTEAD OF トリガー

ビュー内の NESTED TABLE 列の要素は、TABLE 句で直接変更することはできません。ただし、ビューの NESTED TABLE の列に INSTEAD OF トリガーを定義すると、要素を変更できます。NESTED TABLE のトリガーは、その表の要素が更新、挿入または削除すると起動し、基礎となる表に対する実際の変更を処理します。

ここでも、部門と従業員の例を考えます。部門ビューに、部門のリストと各部門の従業員の集合が含まれているものとします。次の例は、部門ビュー内で従業員オブジェクトの NESTED TABLE の要素を、INSTEAD OF トリガーを使用して変更する方法を示しています。

```

/* Create an employee type */
CREATE TYPE emp_t AS OBJECT
(
    empno NUMBER,
    empname VARCHAR2(20),
    days_worked NUMBER
);
/
/* Create a nested table type of employees */
CREATE TYPE emplist_t AS TABLE OF emp_t;
/
/* Now, create the department type */
CREATE TYPE dept_t AS OBJECT
(
    deptno NUMBER,
    deptname VARCHAR2(20),
    emplist emplist_t
);
/
/* The dept_view can now be created based on the dept (department) and emp
* (employee) tables. */
CREATE VIEW dept_view OF dept_t WITH OBJECT OID(deptno)
AS SELECT d.deptno, d.deptname, -- department number and name
        CAST (MULTISET (
            SELECT e.empno, e.empname, (SYSDATE - e.startdate)
            FROM   emp e
            WHERE  e.deptno = d.deptno)
        AS emplist_t) -- emplist - nested table of employees
FROM dept d;

```

TABLE 構文を使用して、ビュー内で NESTED TABLE emplist への挿入を可能にするには、次のように指定します。

```

INSERT INTO TABLE
    (SELECT d.emplist FROM dept_view d WHERE d.deptno = 10)
VALUES (10, 'Harry Mart', 334);

```

次のように、NESTED TABLE emplist に対して、挿入を処理する INSTEAD OF トリガーを定義できます。

```

CREATE TRIGGER dept_empinstr INSTEAD OF INSERT ON
    NESTED TABLE emplist OF dept_view FOR EACH ROW
BEGIN
    INSERT INTO emp VALUES (:NEW.empno, :NEW.empname,
                            :PARENT.deptno, SYSDATE - :NEW.days_worked);
END;
/

```

同様に、NESTED TABLE の要素に対する更新と削除を処理するトリガーも定義できます。

NESTED TABLE のトリガー内からの親行の属性へのアクセス 通常のトリガーでは、カレント行の値には NEW 修飾子および OLD 修飾子を使用してアクセスできます。ビューの NESTED TABLE の列のトリガーでは、これらの修飾子は変更する NESTED TABLE 要素の属性を参照します。この NESTED TABLE の列を含む親行の値にアクセスするには、PARENT 修飾子を使用できます。

この修飾子を使用できるのは、このように NESTED TABLE のトリガー内からのみです。この PARENT 修飾子を使用して取得された親行の値は、変更できません（つまり、読取り専用です）。

ここでは、前述の `dept_empinstr` トリガーの例を考えます。NEW 修飾子は、NESTED TABLE に挿入される行を参照し（つまり、`empno`、`empname` および `days_worked` を含む）、従業員が所属する部門番号（`deptno`）は含めません。ただし、トリガー内で従業員表に部門番号を挿入する必要があります。この値は、PARENT 修飾子を使用して、従業員リストを含む親行から取得できます。

NESTED TABLE とビューのトリガーの起動 前述のように、ビュー内の NESTED TABLE の列に INSTEAD OF トリガーが定義されている場合は、その表の要素が挿入、更新または削除されるときに、トリガーが起動されて実際の変更を行います。

この処理のために、NESTED TABLE の列を含むビューに INSTEAD OF トリガーを定義する必要はありません。ビューに定義されたトリガーは、NESTED TABLE の要素に対する変更時には起動しません。

逆に、ビュー内の行を変更する文では、そのビューの NESTED TABLE の列上のトリガーではなく、ビューに定義されたトリガーのみが起動します。たとえば、`emplist` NESTED TABLE の列が、次のように `dept_view` を通じて更新されるとします。

```
UPDATE dept_view SET emplist = emplist_t(emp_t(1001,'John',234));
```

この場合は、`dept_empinstr` NESTED TABLE のトリガーではなく、`dept_view` に定義された INSTEAD OF 更新トリガーが起動します。

関連項目：『Oracle8i アプリケーション開発者ガイド 基礎編』

システム・イベントとユーザー・イベントのトリガー

トリガーを使用すると、データベース・イベントに関する情報を、サブスクライバに対して発行できます。アプリケーションは、他のアプリケーションからのメッセージにサブスクライプするのと同様に、データベース・イベントにサブスクライプできます。これらのデータベース・イベントは、次のとおりです。

- システム・イベント
 - データベースの起動と停止
 - サーバー・エラー・メッセージ・イベント
- ユーザー・イベント
 - ユーザーのログオンとログオフ
 - DDL 文 (CREATE、ALTER および DROP)
 - DML 文 (INSERT、DELETE および UPDATE)

システム・イベントのトリガーは、データベース・レベルまたはスキーマ・レベルで定義できます。たとえば、データベース停止トリガーは、データベース・レベルで次のように定義します。

```
CREATE TRIGGER register_shutdown
ON DATABASE
SHUTDOWN
BEGIN
...
DBMS_AQ.ENQUEUE(...);
...
END;
```

DDL 文またはログオン・イベントおよびログオフ・イベントのトリガーも、データベース・レベルまたはスキーマ・レベルで定義できます。DML 文のトリガーは、表またはビューに定義できます。データベース・レベルで定義されたトリガーはすべてのユーザーに対して起動し、スキーマ・レベルまたは表レベルで定義されたトリガーは、トリガー・イベントがそのスキーマまたは表に関連する場合にのみ起動します。

イベントの発行

イベントの発行では、Oracle Advanced Queuing のパブリッシュ / サブスクライプ・メカニズムが使用されます。「キュー」は、各種サブスクライバに必要な主題を含むメッセージ・リポジトリの役割を果たします。トリガーは、特定のシステム・イベントまたはユーザー・イベントの発生時に、DBMS_AQ パッケージを使用してメッセージをエンキューします。

関連項目： [第 18 章「アドバンスド・キューイング」](#)

イベントの属性

各イベントでは、トリガー・テキスト内で属性を使用できます。たとえば、データベース起動および停止トリガーは、インスタンス番号およびデータベース名の属性を持ち、ログオン・トリガーおよびログオフ・トリガーは、ユーザー名の属性を持ちます。属性をイベント発生時に発行する場合は、トリガーの作成時に、その属性と同じ名前のファンクションを指定できます。属性値は、トリガーの起動時にファンクションまたはペイロードに渡されます。DML 文のトリガーの場合は、:OLD 列の値によって属性値が :NEW 列の値に渡されます。

システム・イベント

トリガーを起動できるシステム・イベントは、インスタンスの起動と停止およびエラー・メッセージに関連しています。起動および停止イベントに対して作成するトリガーは、データベースに対応付ける必要があります。エラー・イベントに対して作成するトリガーは、データベースまたはスキーマに対応付けることができます。

- **STARTUP** トリガーは、インスタンスによってデータベースがオープンされるときに起動します。この属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- **SHUTDOWN** トリガーは、サーバーがインスタンスの停止操作を開始する直前に起動します。このトリガーを使用して、データベースの停止時にサブスクライバ・アプリケーションを完全に停止できます。インスタンスが異常停止する場合、このトリガーは起動しません。SHUTDOWN トリガーの属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- **SERVERERROR** トリガーは、特定のエラーの発生時、または、エラー番号を指定しなければ任意のエラーの発生時に起動します。このトリガーの属性には、システム・イベントとエラー番号があります。

ユーザー・イベント

トリガーを起動できるユーザー・イベントは、ユーザー・ログオンとログオフ、DDL 文および DML 文に関連しています。

LOGON イベントおよび LOGOFF イベントのトリガー LOGON トリガーおよび LOGOFF トリガーは、データベースまたはスキーマに対応付けることができます。その属性には、システム・イベントとユーザー名があり、USERID と USERNAME について簡単な条件を指定できます。

- **LOGON** トリガーは、ユーザーのログオン成功後に起動します。
- **LOGOFF** トリガーは、ユーザー・ログオフの開始時に起動します。

DDL 文のトリガー DDL トリガーは、データベースまたはスキーマに対応付けることができます。その属性には、システム・イベント、スキーマ・オブジェクトのタイプおよび名前があります。ここでは、スキーマ・オブジェクトの型と名前のみでなく、USERID や USERNAME などの関数についても簡単な条件を指定できます。DDL トリガーには、次のタイプのトリガーがあります。

- BEFORE CREATE トリガーおよび AFTER CREATE トリガーは、データベースまたはスキーマ内でスキーマ・オブジェクトが作成されるときに起動します。
- BEFORE ALTER トリガーおよび AFTER ALTER トリガーは、データベースまたはスキーマ内でスキーマ・オブジェクトが変更されるときに起動します。
- BEFORE DROP トリガーおよび AFTER DROP トリガーは、データベースまたはスキーマからスキーマ・オブジェクトが削除されるときに起動します。

DML 文のトリガー イベント発行用の DML トリガーは、表に対応付けられます。指定した DML 操作が発生する各行に対して起動するように、BEFORE トリガーまたは AFTER トリガーを使用できます。DML 文に関連するイベントの発行には、INSTEAD OF トリガーを使用できません。かわりに、INSTEAD OF トリガーによってビューの基礎となる表に生じる DML 操作について、BEFORE トリガーまたは AFTER トリガーを使用してイベントを発行できます。

イベント発行用の DML トリガーの属性には、システム・イベントと、SELECT リストにユーザーが定義する列があります。これにより、スキーマ・オブジェクトの型と名前のみでなく、関数 (UID、USER、USERENV および SYSDATE)、疑似列および列についても、簡単な条件を指定できます。列には、接頭辞として新旧の値を示す :OLD と :NEW を使用できます。DML 文のトリガーを次に示します。

- BEFORE INSERT および AFTER INSERT トリガーは、表に挿入される行ごとに起動します。
- BEFORE UPDATE および AFTER UPDATE トリガーは、表中で更新される行ごとに起動します。
- BEFORE DELETE および AFTER DELETE トリガーは、表から削除される行ごとに起動します。

関連項目：

- 19-9 ページの「[行トリガー](#)」
- 19-10 ページの「[BEFORE トリガーと AFTER トリガー](#)」
- システム・イベントとユーザー・イベントのトリガーを使用したイベント発行の詳細は、『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。

トリガーの実行

トリガーには次の 2 つのモードがあります。

使用可能	使用可能にされているトリガーは、トリガーを実行する文が発行され、トリガー条件（指定されている場合）の評価が TRUE である場合に限り、そのトリガー・アクションを実行します。
使用禁止	使用禁止にされているトリガーは、トリガーを実行する文が発行され、トリガー条件（指定されている場合）の評価が TRUE であっても、そのトリガー・アクションを実行しません。

使用可能なトリガーの場合、Oracle は次のアクションを自動的に実行します。

- 1 つの SQL 文により複数のトリガーが起動される場合は、指定どおりの起動順序で各トリガーを実行します。
- 様々なタイプのトリガーに対して、指定された時点で整合性制約のチェックを実行し、整合性制約の違反を防止します。
- 問合せと制約に対して、読取り一貫性ビューを提供します。
- トリガー・アクションのコードで参照されるトリガーとスキーマ・オブジェクトの間の依存性を管理します。
- トリガーが分散データベースのリモート表を更新する場合には、2 フェーズ・コミットを使用します。
- 特定の 1 つの文について同じタイプのトリガーが複数存在する場合、順序不定で各トリガーが起動されます。

トリガーの実行モデルと整合性制約のチェック

1 つの SQL 文は最大 4 種類のトリガーを起動します。

- BEFORE 行トリガー
- BEFORE 文トリガー
- AFTER 行トリガー
- AFTER 文トリガー

トリガーを実行する文またはトリガー内の文によって、1 つ以上の整合性制約のチェックが実施されます。またトリガーには、他のトリガーを起動する文を含めることもできます（連鎖的なトリガー）。

Oracle では、次の実行モデルを使用して、複数のトリガーと制約チェックの適切な起動順序を維持します。

1. その文に適用されるすべての BEFORE 文トリガーを実行します。
2. その SQL 文の影響を受ける各行をループします。
 - a. その文に適用されるすべての BEFORE 行トリガーを実行します。
 - b. 行をロックして変更し、整合性制約チェックを実施します。（トランザクションがコミットされるまでロックは解除されません）。
 - c. その文に適用されるすべての AFTER 行トリガーを実行します。
3. 遅延整合性制約チェックを完了します。
4. その文に適用されるすべての AFTER 文トリガーを実行します。

実行モデルの定義は再帰的です。たとえば、ある SQL 文によって BEFORE 行トリガーを起動し、整合性制約をチェックできます。次に、この BEFORE 行トリガーが実行する更新によって、整合性制約をチェックし、AFTER 文トリガーを起動できます。この AFTER 文トリガーによって、整合性制約がチェックされます。この場合、実行モデルでは、次のステップが再帰的に繰り返し実行されます。

1. 元の SQL 文が発行されます。
2. BEFORE 行トリガーが起動されます。
3. BEFORE 行トリガー内の UPDATE 文により、AFTER 文トリガーが起動します。
4. AFTER 文トリガーの文が実行されます。
5. AFTER 文トリガーによって変更された表の整合性制約がチェックされます。
6. BEFORE 行トリガーの文が実行されます。
7. BEFORE 行トリガーによって変更された表の整合性制約がチェックされます。
8. SQL 文が実行されます。

9. SQL 文から整合性制約がチェックされます。

この再帰には、次の 2 つの例外があります。

- トリガーを実行する文が参照制約内で 1 つの表（主キーまたは外部キー表）を変更し、トリガーされた文が他方の表を変更すると、トリガー文のみが整合性制約をチェックします。これにより、行トリガーで参照整合性を強化できます。
- DELETE CASCADE および DELETE SET NULL によって起動されトリガーを実行する文は、個々の規定文の前後ではなく、ユーザーの DELETE 文の前後に起動されます。これにより、この種の文トリガーによる変更エラーの発生が防止されます。

実行モデルの重要なプロパティは、SQL 文の結果として実行されるアクションとチェックがすべて成功する必要があることです。トリガー内で発生した例外を明示的に処理できない場合、元の SQL 文によって実行されたすべてのアクションが、起動されたトリガーによって実行されたアクションを含めて、ロールバックされます。したがって、トリガーが整合性制約に違反することはありません。実行モデルは整合性制約を考慮し、宣言整合性制約に違反するトリガーを認めません。

たとえば、前述のステップ 1～8 を正しく実行し、ステップ 9 で整合性制約に違反があったとします。その違反の結果として、SQL 文によるすべての変更（ステップ 8）、起動された BEFORE 行トリガー（ステップ 6）および起動された AFTER 文トリガー（ステップ 4）がロールバックされます。

注意： 各種トリガーは特定の順序で起動されますが、同じ文に対する同じタイプのトリガーは、特定の順序で起動されるとは限りません。たとえば、1 つの UPDATE 文に対して定義されている BEFORE 行トリガーすべてが、毎回同じ順序で起動されるとは限りません。同じタイプのトリガーが複数ある場合は、その起動順序に依存しないようアプリケーションを設計してください。

トリガーのデータ・アクセス

トリガーが起動されると、トリガー・アクション内で参照される表は、他のユーザーのトランザクション内の SQL 文によって変更されている最中である可能性があります。トリガー内で実行される SQL 文は、常に単独の SQL 文と同じ規則に従います。起動されたトリガーが読み込み（問合せ）または書き込み（更新）の対象にする値が、まだコミットされていないトリガーによって変更されたものである場合、起動されたトリガーの本体にある SQL 文は、次のガイドラインに従います。

- 問合せは、参照先の表の読取り一貫性のあるカレント・スナップショットと、同一トランザクション内で変更されたデータを参照します。
- 更新は、既存のデータのロックが解除されるまで待機してから、処理を続行します。

次に、これらを具体的に説明する例を示します。

例 1: SALARY_CHECK トリガー（本体）に、次の SELECT 文が含まれているとします。

```
SELECT minsal, maxsal INTO minsal, maxsal
  FROM salgrade
 WHERE job_classification = :new.job_classification;
```

この例で、トランザクション T1 に SALGRADE 表の MAXSAL 列に対する更新が含まれているとします。その更新の時点で、トランザクション T2 内の文によって SALARY_CHECK トリガーが起動されます。起動されたトリガー内の SELECT 文（T2 に由来）は、コミットされていないトランザクション T1 による更新を参照しません。そのトリガーの問合せは、トランザクション T2 の読取り一貫性ポイントの値として、MAXSAL の古い値を戻します。

例 2: TOTAL_SALARY トリガーが、部門内のメンバー全員の給与総額が格納されている導出列をメンテナンスするものとします。

```
CREATE TRIGGER total_salary
AFTER DELETE OR INSERT OR UPDATE OF deptno, sal ON emp
FOR EACH ROW BEGIN
  /* assume that DEPTNO and SAL are non-null fields */
  IF DELETING OR (UPDATING AND :old.deptno != :new.deptno)
    THEN UPDATE dept
      SET total_sal = total_sal - :old.sal
      WHERE deptno = :old.deptno;
  END IF;
  IF INSERTING OR (UPDATING AND :old.deptno != :new.deptno)
    THEN UPDATE dept
      SET total_sal = total_sal + :new.sal
      WHERE deptno = :new.deptno;
  END IF;
  IF (UPDATING AND :old.deptno = :new.deptno AND
      :old.sal != :new.sal )
    THEN UPDATE dept
      SET total_sal = total_sal - :old.sal + :new.sal
      WHERE deptno = :new.deptno;
  END IF;
END;
```

この例で、コミットされていない第一のユーザーのトランザクションが、DEPT 表の 1 つの行の TOTAL_SAL 列に対する更新が含まれていたとします。その更新の時点で、第二のユーザーの SQL 文によって TOTAL_SALARY トリガーが起動されます。第一のユーザーのコミットされていないトランザクションに、TOTAL_SAL 列に含まれる関連する値の更新が含まれている（つまり行ロックがかけられている）ため、行ロックを保持しているトランザクションがコミットまたはロールバックされるまで、TOTAL_SALARY トリガーによる更新は実行されません。このため、第二のユーザーは、第一のユーザーのコミット・ポイントまたはロールバック・ポイントまで待機します。

PL/SQL トリガーの記憶域

PL/SQL トリガーは、ストアド・プロシージャと同じように、コンパイル済の形式で格納されます。CREATE TRIGGER 文がコミットされると、P コード（疑似コード）と呼ばれるコンパイル済 PL/SQL コードがデータベースに格納され、トリガーのソース・コードは共有プールからフラッシュされます。

関連項目： PL/SQL コードのコンパイルと格納の詳細は、17-17 ページの「[Oracle がプロシージャとパッケージを格納する方法](#)」を参照してください。

トリガーの実行

Oracle は、プロシージャの実行と同じステップを使用してトリガーを内部的に実行します。両者の唯一の違いは、ユーザーにトリガーを起動する権限が付与されるのは、ユーザーがトリガー文を実行する権限を持っている場合ということです。この点を除けば、トリガーはストアド・プロシージャと同じ方法でチェックされ、実行されます。

関連項目： 17-19 ページの「[Oracle がプロシージャとパッケージを実行する方法](#)」

トリガーの依存性のメンテナンス

プロシージャと同様に、トリガーも参照しているオブジェクトに依存します。トリガー・アクションで参照されるスキーマ・オブジェクトに対するトリガーの依存性は、Oracle によって自動的に管理されます。トリガーの依存性についての問題は、ストアド・プロシージャの依存性についての問題と同じです。トリガーはストアド・プロシージャと同じように扱われ、データ・ディクショナリに挿入されます。

関連項目： 第 20 章「[Oracle の依存性の管理](#)」

Oracle の依存性の管理

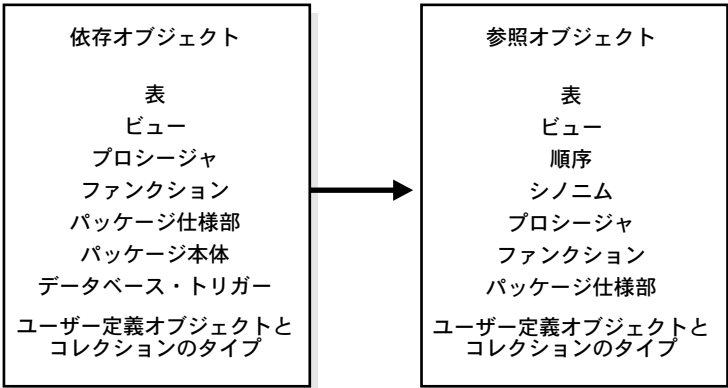
ビューやプロシージャなどのオブジェクトの定義では、他のオブジェクト（表など）が参照されます。したがって、定義するオブジェクトは、その定義で参照する他のオブジェクトに依存しています。この章では、スキーマ・オブジェクトの相互間の依存性と、Oracle がそのような依存性を自動的に追跡し管理する方法について説明します。この章の内容は次のとおりです。

- 依存性の問題の概要
- スキーマ・オブジェクトの依存性の解決
- 依存性の管理と存在しないスキーマ・オブジェクト
- 共有 SQL の依存性管理
- ローカルおよびリモートの依存性の管理

依存性の問題の概要

スキーマ・オブジェクトのタイプによっては、定義の一部として他のオブジェクトを参照できるものがあります。たとえば、ビューは表や他のビューを参照する問合せによって定義されます。また、プロシージャ本体には、データベースの他のオブジェクトを参照する SQL 文を組み込みます。定義の一部として他のオブジェクトを参照するオブジェクトを「依存」オブジェクトと呼び、参照されるオブジェクトを「参照」オブジェクトと呼びます。図 20-1 に、各種の依存オブジェクトと参照オブジェクトを示します。

図 20-1 様々な依存スキーマ・オブジェクトと参照スキーマ・オブジェクト



参照オブジェクトの定義を変更した場合、その変更の内容によって、依存オブジェクトがエラーなしで機能し続ける場合と、そうでない場合があります。たとえば、表を削除した場合、その表に基づくビューは使用できなくなります。

Oracle ではオブジェクト間の依存性が自動的に記録されるため、データベース管理者とユーザーは依存性管理の複雑な作業から解放されます。たとえば、複数のストアド・プロシージャが依存している表を変更すると、依存プロシージャは、次に参照された（再度実行またはコンパイルされた）時点で自動的に再コンパイルされます。

スキーマ・オブジェクト間の依存性を管理するために、データベース内のすべてのスキーマ・オブジェクトは次のいずれかの状態になります。

VALID スキーマ・オブジェクトはコンパイルされており、参照するとただちに使用できます。

INVALID

スキーマ・オブジェクトを使用するには、その前にコンパイルする必要があります。

- プロシージャ、ファンクションおよびパッケージの場合は、スキーマ・オブジェクトをコンパイルする必要があることを意味します。
- また、ビューの場合は、データ・ディクショナリ内のカレント定義を使用して、ビューを再解析する必要があることを意味します。

INVALID になる可能性があるのは、依存しているオブジェクトのみです。表、順序およびシノニムは、常に VALID です。

ビュー、プロシージャ、ファンクションまたはパッケージが INVALID である場合、Oracle がそのオブジェクトをコンパイルしようとしても、そのオブジェクトに関連したエラーが発生します。たとえば、ビューのコンパイル時に、ベース表の 1 つが存在しなかったり、ベース表の正当な権限がなかったりする場合があります。パッケージのコンパイル時には、PL/SQL または SQL の構文エラーが発生したり、参照されているオブジェクトについての適切な権限が存在しない場合があります。このような問題があるスキーマ・オブジェクトは INVALID のままです。

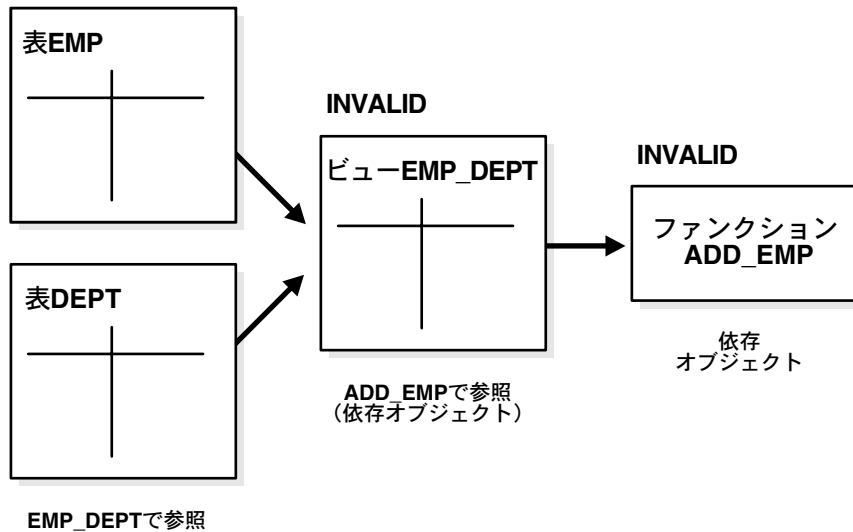
Oracle はデータベースの変更を自動的に追跡し、関係するオブジェクトの状態をデータ・ディクショナリに記録します。

状態の記録は再帰的な処理です。参照オブジェクトの状態が変更されると、直接的に依存するオブジェクトの状態が変更されるだけでなく、間接的に依存するオブジェクトの状態も変更されます。

たとえば、ビューを直接参照するストアド・プロシージャを考えます。ストアド・プロシージャは、このビューのベース表を間接的に参照します。このため、ベース表を変更するとビューは無効になり、さらにストアド・プロシージャも無効になります。[図 20-2](#) に、間接的な依存性を示します。

図 20-2 間接的な依存性

ALTER TABLE emp ...;



スキーマ・オブジェクトの依存性の解決

スキーマ・オブジェクトが SQL 文によって直接的に、または依存オブジェクトの参照を介して間接的に参照されるときに、Oracle は SQL 文に明示的に指定されているオブジェクトと参照オブジェクトの状態を、必要に応じてチェックします。Oracle のアクションは、SQL 文で直接および間接的に参照されるオブジェクトの状態に応じて異なります。

- 参照オブジェクトがすべて有効であれば、追加的な処理なしで SQL 文が即時に実行されます。
- 参照先のビューやプロシージャ（ファンクションやパッケージを含む）が無効な状態になっていると、Oracle はその参照オブジェクトを自動的にコンパイルしようとします。
 - 無効な参照オブジェクトは、そのすべてを正常にコンパイルできる場合には問題なくコンパイルされ、SQL 文は正常に実行されます。
 - 無効なオブジェクトは、正常にコンパイルできなければ無効のままになります。Oracle はエラーを戻し、その SQL 文を含むトランザクションをロールバックします。

注意： Oracle は、無効であると検出された後に置き換えられていないオブジェクトのみを動的に再コンパイルします。これにより、不要な再コンパイルが省略されます。

ビューと PL/SQL プログラム・ユニットのコンパイル

次の条件が満たされている場合は、ビューまたは PL/SQL プログラム・ユニットをコンパイルして、その状態を VALID にすることができます。

- ビューまたはプログラム・ユニットの定義が正しいこと。すべての SQL 文と PL/SQL 文が適切な構文である必要があります。
- 参照オブジェクトが存在し、適切な構成であること。たとえば、ビューを定義する問合せに列が含まれる場合、その列がベース表に存在している必要があります。
- ビューまたはプログラム・ユニットの所有者に、参照オブジェクトに対する必要な権限が付与されていること。たとえば、プロシージャ内の SQL 文によって表に行が挿入される場合、プロシージャの所有者にはその参照表に対する INSERT 権限が必要です。

ビューとベース表

ビューは、そのビューの定義問合せで参照されているベース表またはビューに依存しています。たとえば、`SELECT * FROM table` のように、ビューの問合せで列が明示的に定義されていないければ、定義問合せはデータ・ディクショナリへの格納時に展開され、参照先のベース表内の現行のすべての列が含まれるようになります。

ビューのベース表またはビューを変更、改名または削除すると、そのビューは無効になりますが、その定義は無効なビューを参照する権限、シノニム、他のオブジェクトおよび他のビューとともに、データ・ディクショナリ内に残ります。

無効 (INVALID) なビューを使用すると、そのビューは動的に再コンパイルされます。再コンパイルされたビューは、次に示す条件に応じて有効または無効になります。

- ビューの定義問合せが参照しているすべてのベース表が存在している必要があります。ビューのベース表を改名または削除すると、そのビューは無効になり、使用できなくなります。無効なビューを参照すると、その参照文は失敗します。ビューをコンパイルできるのは、ベース表を元の名前に改名した場合、またはベース表を再作成した場合のみです。
- ベース表を変更したり、同じ列を含むベース表を再作成した場合に、そのベース表の 1 つ以上の列のデータ型が変更されていても、依存ビューは正常に再コンパイルできます。

- ビューのベース表が、少なくとも同じ列セットを含むように変更または再作成された場合、そのビューは有効にすることができます。ベース表が新しい列を含むように再作成され、その結果の表中にもはや存在しない列をビューが参照していると、そのビューは有効になりません。後者のケースは、`SELECT * FROM table` の問合せで定義されたビューに特に当てはまります。このような定義問合せはビューの作成時にデータ・ディクショナリ内で展開され、そのまま永続的に格納されるためです。

プログラム・ユニットと参照オブジェクト

参照オブジェクトの定義を変更すると、プログラム・ユニットは自動的に無効になります。たとえば、スタンドアロン・プロシージャに表、ビュー、別のスタンドアロン・プロシージャおよびパブリック・パッケージ・プロシージャを参照する複数の文が含まれているとします。このような場合は、次のような条件が成立します。

- 参照表を変更すると、依存プロシージャは無効になります。
- 参照ビューのベース表を変更すると、ビューと依存プロシージャが無効になります。
- 参照スタンドアロン・プロシージャを置き換えると、依存プロシージャが無効になります。
- 参照パッケージの「本体」を置き換えても、依存プロシージャには影響しません。ただし、参照パッケージの「仕様部」を置き換えると、依存プロシージャは無効になります。

この最後の条件は、パッケージを使用することによってプロシージャおよび参照オブジェクトの間の依存性を最小にするメカニズムを示しています。

セッションの状態と参照パッケージ

パッケージ構成体を参照するセッションごとに、各パッケージの独自のインスタンスが存在します。このインスタンスには、パブリック変数とプライベート変数、カーソルおよび定数の永続的な状態が含まれます。その後、セッションのインスタンス化されたパッケージのいずれかが無効にされて再コンパイルされると、セッションのパッケージのすべてのインスタンスエーションが、状態も含めて失われる可能性があります。

セキュリティ認可

ユーザーや PUBLIC に対して DML オブジェクト権限やシステム権限の付与または取消しが実行された場合、Oracle はその旨を通知して、その所有者のすべての依存オブジェクトを自動的に無効にします。Oracle は依存オブジェクトを無効にして、その依存オブジェクトの所有者が参照オブジェクトに対する必要な権限を引き続き持っていることを検証します。Oracle は内部的に、そのようなオブジェクトを再コンパイルする必要はないことを確認します。つまり、オブジェクト構造を検証するのではなく、セキュリティ認可のみを検証します。このように最適化すれば、不要な再コンパイルが省略され、依存オブジェクトのタイムスタンプを変更する必要もありません。

関連項目： 無効なビューとプログラム・ユニットを強制的に再コンパイルする方法の詳細は、『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。

ファンクション索引の依存性

ファンクション索引は、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。PL/SQL ファンクションまたはパッケージ・ファンクションに変更があると、索引は使用禁止になります。

ここでは、ファンクション索引の要件と、削除される時期や使用権限が取り消される時期など、なんらかの方法でファンクションが変更された場合の操作について説明します。

要件

ファンクション索引を作成する手順は次のとおりです。

- 次の初期化パラメータを定義する必要があります。
 - QUERY_REWRITE_INTEGRITY を TRUSTED に設定します。
 - QUERY_REWRITE_ENABLED を TRUE に設定します。
 - COMPATIBLE を 8.1.0.0.0 以上の値に設定します。
- ユーザーには、CREATE INDEX と QUERY REWRITE、または CREATE ANY INDEX と GLOBAL QUERY REWRITE を付与する必要があります。

ファンクション索引を使用する手順は次のとおりです。

- 表は、索引の作成後に分析する必要があります。
- NULL 値は索引に格納されないため、問合せには索引付きの式からの NULL 値を必要としないことを保証する必要があります。

この後の各項では、追加の要件について説明します。

関連項目： 10-26 ページの「[ファンクション索引](#)」

DETERMINISTIC 関数

ファンクション索引に使用するためにユーザーが記述する関数は、現在も将来も入力引数値の集合に対して常に同じ出力戻り値を戻すことを示すために、DETERMINISTIC キーワードで宣言する必要があります。

関連項目：『Oracle8i パフォーマンスのための設計およびチューニング』

定義する関数に対する権限

索引所有者は、ファンクション索引の定義に使用する関数に対して EXECUTE 権限を持つ必要があります。EXECUTE 権限が取り消されると、ファンクション索引に DISABLED マークが設定されます。索引所有者は、この関数に対する EXECUTE WITH GRANT OPTION 権限がなくても、基礎となる表の SELECT 権限を付与できます。

ファンクション索引の依存性の解決

ファンクション索引は、それを使用するファンクションに依存します。ファンクション、またはそのファンクションを含むパッケージの仕様部が再定義されると（または、索引所有者の EXECUTE 権限が取り消されると）、次の条件が成立します。

- その索引に DISABLED マークが設定されます。
- DISABLED 索引の問合せは、オプティマイザが索引を使用するために選択すると、失敗します。
- DISABLED 索引の DML 操作は、索引にさらに UNUSABLE マークが設定され、初期化パラメータ SKIP_UNUSABLE_INDEXES が TRUE に設定されていなければ、失敗します。

ファンクションの変更後に索引を再使用可能にするには、ALTER INDEX ... ENABLE 文を使用します。

依存性の管理と存在しないスキーマ・オブジェクト

依存オブジェクトが作成されると、Oracle は次のステップを実行します。

1. Oracle は、最初にカレント・スキーマを検索し、すべての参照を解決しようとします。
2. カレント・スキーマ内に参照オブジェクトが見つからない場合、Oracle は、同じスキーマ内のプライベート・シノニムを検索して参照を解決しようとします。
3. プライベート・シノニムがなければ、パブリック・シノニムが検索されます。
4. パブリック・シノニムが存在しない場合、Oracle はオブジェクト名の先頭部分に一致するスキーマ名を検索します。

5. 一致するスキーマ名が存在すると、そのスキーマ内でオブジェクトが検索されます。
6. スキーマが見つからない場合には、エラーが戻されます。

Oracle による参照の解決では、オブジェクトが、他のオブジェクトが存在しないという事実
に依存する場合があります。このような状況は、依存オブジェクトが使用している参照が、
別のオブジェクトが存在する場合に異なって解釈される可能性がある場合に発生します。た
とえば、次のような場合です。

- 現時点では、COMPANY スキーマには EMP という名前の表が含まれています。
- COMPANY.EMP のために EMP という名前の PUBLIC シノニムが作成されており、
COMPANY.EMP に対する SELECT 権限が PUBLIC ロールに付与されています。
- JWARD スキーマには、EMP という名前の表またはプライベート・シノニムは含まれて
いません。
- ユーザー JWARD は、次の文を使用して自分のスキーマ内にビューを作成します。

```
CREATE VIEW dept_salaries AS
  SELECT deptno, MIN(sal), AVG(sal), MAX(sal) FROM emp
  GROUP BY deptno
  ORDER BY deptno;
```

JWARD が DEPT_SALARIES ビューを作成する際、EMP への参照を解決するために、表、
ビューおよびプライベート・シノニムとしての JWARD.EMP が最初に検索されます。いず
れも見つからないため、次に EMP という名前のパブリック・シノニムとして検索されて、
見つかります。結果的に、JWARD.DEPT_SALARIES は、JWARD.EMP が存在しないこと
と、PUBLIC.EMP が存在していることに依存することになります。

JWARD が、次の文を使用して EMP という名前の新しいビューを自分のスキーマ内に作成
するとします。

```
CREATE VIEW emp AS
  SELECT empno, ename, mgr, deptno
  FROM company.emp;
```

JWARD.EMP と COMPANY.EMP の列構成が異なることに注意してください。

オブジェクト定義に含まれる参照を解決しようとするときに、新しい依存オブジェクトが存
在しないはずのオブジェクトに依存していることが Oracle 内部で考慮されます。存在しな
いはずのオブジェクトとは、そのオブジェクトが存在していると、新しく作成するオブジ
ェクトの定義の解釈が変わってしまうスキーマ・オブジェクトのことです。存在しないはずの
オブジェクトが後に作成された場合に備えるため、そのような依存性があるということを知
覚しておく必要があるので注意してください。存在しないはずのオブジェクトが実際に作成さ
れた場合、その依存オブジェクトを再コンパイルして有効にするには、すべての依存オブ
ジェクトを無効にして、すべての依存ファンクション索引に UNUSABLE マークを設定する
必要があります。

したがって、前述の例では、JWARD.EMP が作成されると、JWARD.DEPT_SALARIES は JWARD.EMP に依存しているため無効になります。JWARD.DEPT_SALARIES が使用されると、Oracle はビューを再コンパイルします。EMP の参照は解決され、JWARD.EMP が見つかります（PUBLIC.EMP は参照オブジェクトではなくなります）。ただし、JWARD.EMP には SAL 列が含まれないため、ビューを置き換える際にエラーが発生し、無効のままになります。

要約すると、オブジェクトを解決する際に存在しないはずのオブジェクトが見つかった場合、存在しないはずのオブジェクトが後で実際に作成される場合に備えて、そのオブジェクトに対する依存性を管理する必要があります。

共有 SQL の依存性管理

スキーマ・オブジェクト相互間の依存性管理に加えて、Oracle は共有プール内の各共有 SQL 領域の依存性も管理します。表またはビュー、シノニム、順序を生成、変更または削除したり、プロシージャまたはパッケージの仕様部を再コンパイルすると、それらに依存する共有 SQL 領域もすべて無効になります。共有 SQL 領域が無効になった後で、その領域に対応するカーソルを実行すると、Oracle によって SQL 文が再解析され、共有 SQL 領域が再生成されます。

ローカルおよびリモートの依存性の管理

依存性の追跡や必要な再コンパイルは、Oracle によって自動的に実行されます。「ローカル依存性の管理」が発生するのは、Oracle が単一データベース内のオブジェクト間の依存性を管理する場合です。たとえば、プロシージャ内の文が、同じデータベース内の表を参照する場合があります。

「リモート依存性の管理」が発生するのは、Oracle がネットワークを介した分散環境で依存性を管理する場合です。たとえば、Oracle Forms トリガーは、データベース内のスキーマ・オブジェクトに依存する場合があります。また、分散データベースにおいて、ローカル・ビューの定義問合せがリモート表を参照する場合があります。

ローカル依存性の管理

Oracle は、データベース内部の依存性表を使用してローカル依存性を管理します。この表は、スキーマ・オブジェクトごとに依存オブジェクトを追跡し、記録するものです。参照オブジェクトが修正されると、この依存性表を使用して依存オブジェクトが識別され、それらの依存オブジェクトが無効にされます。

たとえば、ストアド・プロシージャ UPDATE_SAL が表 JWARD.EMP を参照している場合を考えます。この表の定義になんらかの変更があると、ストアド・プロシージャ UPDATE_SAL を含めて、JWARD.EMP を参照するすべてのオブジェクトの状態が INVALID に変更されます。そのため、このプロシージャは、再コンパイルされて有効にされるまで実行できません。同様に、ユーザーの DML 権限を取り消すと、そのユーザーのスキーマ内の依存オブジェクトはすべて無効になります。ただし、認可が取り消されたために

無効になったオブジェクトは、「再認可」によって有効にできます。その場合は、完全に再コンパイルする必要はありません。

リモート依存性の管理

Oracle は、アプリケーションからデータベースへの依存性と分散データベースの依存性も管理します。たとえば、Oracle Forms アプリケーションには表を参照するトリガーが含まれていたり、ローカルなストアド・プロシージャから分散データベース・システム内のリモート・プロシージャがコールされることがあります。データベース・システムでは、このようなオブジェクト間の依存性を管理する必要があります。関係するオブジェクトの種類によっては、リモート依存性を管理するために別のメカニズムが使用されます。

ローカルおよびリモートのデータベース・プロシージャ間の依存性

ファンクション、パッケージおよびトリガーなど、ストアド・プロシージャの相互間の依存性は、分散データベース・システムにおいては「タイムスタンプのチェック」または「シグネチャのチェック」を使用して管理されます。

動的初期化パラメータ `REMOTE_DEPENDENCIES_MODE` は、タイム・スタンプとシグネチャのどちらでリモート依存性を管理するかを決定します。

関連項目： タイムスタンプまたはシグネチャによるリモート依存性管理の詳細は、『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。

タイムスタンプのチェック タイムスタンプ・チェック依存性モデルでは、プロシージャがコンパイルまたは再コンパイルされるたびに、その「タイムスタンプ」（作成、変更または置換の時刻）がデータ・ディクショナリに記録されます。タイムスタンプは、プロシージャが作成、変更または置換された時刻の記録です。さらに、コンパイル済プロシージャには、それが参照するリモート・プロシージャごとに、スキーマ、パッケージ名、プロシージャ名、タイムスタンプなどの情報も含まれています。

依存プロシージャが使用されると、Oracle は、コンパイル時に記録されたりリモート・タイムスタンプと、リモートで参照されたプロシージャのカレント・タイムスタンプを比較します。比較結果に応じて、次の 2 つの処理が発生します。

- タイムスタンプが一致すると、ローカル・プロシージャとリモート・プロシージャはコンパイルされずに実行されます。
- リモートで参照されたプロシージャのいずれかの比較結果が一致しなければ、ローカル・プロシージャは無効となり、コール元の環境にエラーが戻されます。さらに、新しいタイムスタンプを持つそのリモート・プロシージャに依存する他のすべてのローカル・プロシージャも無効になります。たとえば、いくつかのローカル・プロシージャがリモート・プロシージャをコールしており、そのリモート・プロシージャが再コンパイルされたとします。ローカル・プロシージャの 1 つが実行され、リモート・プロシ

ジャのタイムスタンプと一致しないことがわかると、そのリモート・プロシージャに依存するローカル・プロシージャはすべて無効になります。

実際のタイムスタンプの比較は、ローカル・プロシージャ本体内の文がリモート・プロシージャを実行する時点で実行されます。この時点では、分散データベースの通信リンクを介してタイムスタンプの比較のみが実行されます。したがって、ローカル・プロシージャの文のうち無効なプロシージャ・コールより前にあるすべての文は、正常に実行される可能性があります。無効なプロシージャ・コールより後の文はまったく実行されません。コンパイルが必要です。ただし、無効なプロシージャ・コールより前に実行された DML 文があると、すべてロールバックされます。

シグネチャのチェック Oracle には「シグネチャ」を使用する「リモート依存性」の機能もあります。このシグネチャ機能の影響を受けるのは、リモート依存性のみです。ローカル依存性は、その環境でいつでも再コンパイルできるため影響を受けません。

プロシージャのシグネチャには、次の項目に関する情報が含まれます。

- パッケージ、プロシージャまたはファンクションの名前
- パラメータのベース型
- パラメータのモード (IN、OUT および IN OUT)

注意： 重要なのはパラメータの型とモードのみです。パラメータの名前はシグネチャには影響しません。

シグネチャ依存性モデルが有効である場合に、依存単位に親単位のプロシージャへのコールが含まれており、かつ、そのプロシージャのシグネチャが互換性のない方法で変更されると、リモート・プログラム・ユニットへの依存性により、その依存単位が無効になります。プログラム・ユニットは、パッケージ、ストアド・プロシージャ、ストアド・ファンクションまたはトリガーのいずれかです。

その他のリモート・スキーマ・オブジェクトの間の依存性

Oracle では、ローカル・プロシージャとリモート・プロシージャの間の依存性を除いては、リモート・スキーマ・オブジェクト間の依存性は管理されません。

たとえば、リモート表を参照する問合せによって、ローカル・ビューを作成して定義するとします。また、ローカル・プロシージャに、同じリモート表を参照する SQL 文が含まれているとします。その後、表の定義が変更されたとします。

表の変更後にビューとプロシージャが使用され、そのビューとプロシージャがエラーとなった場合でも、ローカル・ビューとプロシージャは無効になりません。この場合、ビューとプロシージャは、エラーが戻されないように手動で変更する必要があります。このような場合は、依存オブジェクトを不必要に再コンパイルするより、依存性を管理しないことをお勧めします。

アプリケーションの依存性

データベース・アプリケーションのコードでは、接続されているデータベース内のオブジェクトを参照できます。たとえば、OCI、プリコンパイラおよび SQL*Module アプリケーションは、無名 PL/SQL ブロックを送信できます。また、Oracle Forms アプリケーションのトリガーは、スキーマ・オブジェクトを参照できます。

このようなアプリケーションは、参照するスキーマ・オブジェクトに依存しています。依存性管理の方法は、開発環境によって異なります。データベース・アプリケーション内でリモート依存性を管理する方法の詳細は、アプリケーション開発ツール固有およびオペレーティング・システム固有の該当するマニュアルを参照してください。

第 VI 部

SQL 文の最適化

第 VI 部では、各 SQL 文を実行するための最も効率的な方法を選択するオプティマイザについて説明します。

第 VI 部に含まれる章は、次のとおりです。

- [第 21 章「オプティマイザ」](#)

21

オプティマイザ

この章では、Oracle オプティマイザの概要を説明します。この章の内容は次のとおりです。

- [最適化の概要](#)
- [コストベース最適化](#)
- [拡張可能な最適化](#)
- [ルールベース最適化](#)

関連項目： クエリー・リライトにマテリアライズド・ビューを使用する方法など、オプティマイザの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

最適化の概要

最適化とは、SQL 文を実行するのに最も効率的な方法を選択する処理です。この処理は、SELECT、INSERT、UPDATE または DELETE など、データ操作言語（DML）文の処理において重要なステップです。SQL 文を実行する場合、表や索引にアクセスする順序などによって、実行方法が様々になることがあります。文を実行するときに使用する手順は、文の実行速度に大きく影響します。

「オプティマイザ」は、SQL 文の最も効率的な実行方法を算定します。オプティマイザは、多数の要因を評価して代替アクセス・パスを選択します。コストベースまたはルールベースのアプローチを使用します。

注意： オプティマイザは、Oracle のバージョンが変わると、同じ決定をしない可能性があります。最近のバージョンでは、さらに改善され洗練された情報に基づいて実行方法を決定するようになっています。

オプティマイザのアプローチと目標を設定し、コストベース最適化の統計を収集すると、オプティマイズによる選択を調整できます。特定のアプリケーションのデータについて、オプティマイザよりもさらに詳細な知識を持つアプリケーション・デザイナのほうが、SQL 文をより効率的に実行する方法を選択できることもあります。アプリケーション・デザイナは、SQL 文内にヒントを使用して文の実行方法を指定できます。

関連項目：

- 21-8 ページの「[コストベース最適化](#)」
- 21-20 ページの「[ルールベース最適化](#)」
- SQL 文のヒントの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

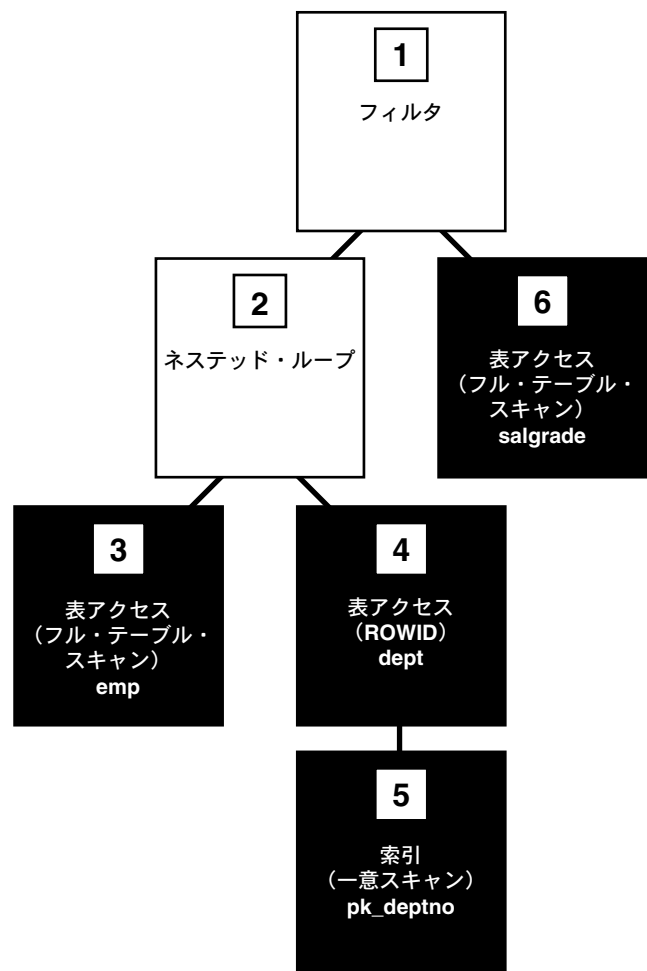
実行計画

DML 文を実行するために、Oracle では数多くのステップを必要とする場合があります。それらの各ステップでは、データベースからデータ行を物理的に検索するか、文を発行したユーザーのためになんらかの方法でデータ行を準備します。Oracle が文の実行に使用するステップの組合せを、「実行計画」と呼びます。実行計画には、文がアクセスする表ごとの「アクセス方法」と表の「結合順序」が含まれています。

図 21-1 に、次の SQL 文の実行計画を示します。この文は、給与が基準範囲内でないすべての従業員の名前 (ename)、職種 (job)、給与 (sal) および部門名 (dname) を選択します。

```
SELECT ename, job, sal, dname
  FROM emp, dept
 WHERE emp.deptno = dept.deptno
    AND NOT EXISTS
      (SELECT *
        FROM salgrade
       WHERE emp.sal BETWEEN losal AND hisal);
```

図 21-1 実行計画



実行計画のステップ

実行計画の各ステップでは、次のステップで使用する行のセットを戻すか、最後のステップの場合は、SQL 文を発行したユーザーまたはアプリケーションに行のセットを戻します。1つのステップから戻される行のセットのことを、「行ソース」と呼びます。

図 21-1 は、あるステップから別のステップへの行ソースの流れを示す階層図です。各ステップに付けられている番号は、EXPLAIN PLAN 文によって実行計画が表示される順序と対応しています（詳細はこの後の部分を参照）。多くの場合、この順序はステップが実際に実行される順序とは異なっています。実行計画の各ステップでは、入力として、データベースから行を検索するか、1 つ以上の行ソースから行を受け取ります。

- 黒のボックスで示されているステップでは、データベースのオブジェクトから物理的にデータを取り出します。このようなステップを「アクセス・パス」と呼びます。
 - ステップ 3 とステップ 6 では、それぞれ EMP 表と SALGRADE 表のすべての行を読み込みます。
 - ステップ 5 では、ステップ 3 で戻された各 DEPTNO 値を、PK_DEPTNO 索引の中から検索します。次に、DEPT 表内の対応する行の ROWID を検索します。
 - ステップ 4 では、ステップ 5 で戻された ROWID が入っている行を DEPT 表から取り出します。
- 白のボックスで示されているステップでは、行ソースを操作します。
 - ステップ 2 では、ネステッド・ループ操作を実行します。ステップ 3 と 4 から行ソースを受け取り、ステップ 3 のソースから受け取った各行をステップ 4 の対応する行と結合し、結果の行をステップ 1 に戻します。
 - ステップ 1 では、フィルタ操作を実行します。ステップ 2 と 6 から行ソースを受け取り、ステップ 2 からの行でステップ 6 に対応する行が入っている行を削除し、ステップ 2 のその他の行を、文を発行したユーザーまたはアプリケーションに戻します。

関連項目：

- 21-6 ページの「実行の順序」
- アクセス・パスおよび Oracle で行ソースを結合する方法の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

EXPLAIN PLAN 文

EXPLAIN PLAN 文を使用すると、オプティマイザが選択する SQL 文の実行計画を調べることができます。この文は、オプティマイザに実行計画を選択させ、その計画を記述したデータをデータベース表に挿入します。

たとえば、前項に示した文の記述の出力表は、次のようになります。

ID	OPERATION	OPTIONS	OBJECT_NAME

0	SELECT STATEMENT		
1	FILTER		
2	NESTED LOOPS		

3	TABLE ACCESS	FULL	EMP
4	TABLE ACCESS	BY ROWID	DEPT
5	INDEX	UNIQUE SCAN	PK_DEPTNO
6	TABLE ACCESS	FULL	SALGRADE

図 21-1 のボックスと出力表の行は、それぞれ実行計画の単一のステップに対応しています。リストに示されている各行の ID 列の値は、図 21-1 の対応するボックスに示されている値です。

このようなリストは、EXPLAIN PLAN 文を使用した後、その出力表に問い合せて取得できます。

関連項目： EXPLAIN PLAN の使用方法は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

実行の順序

実行計画のステップは、図に示されている番号の順序で実行されるわけではありません。Oracle は実行計画のツリー構造の図でリーフ・ノードとして示されているステップ（図 21-1 のステップ 3、5 および 6）を最初に実行します。それぞれのステップから戻された行が、その親ステップの行ソースになります。この後、親ステップを実行します。

たとえば、図 21-1 の文を実行する場合、Oracle は次の順序でステップを実行します。

- ステップ 3 を実行し、結果の行を 1 つずつステップ 2 に戻します。
- ステップ 3 から戻されるそれぞれの行について、次のステップを実行します。
 1. ステップ 5 を実行し、結果の ROWID をステップ 4 に戻します。
 2. ステップ 4 を実行し、結果の行をステップ 2 に戻します。
 3. ステップ 2 を実行します。ステップ 3 からの 1 行とステップ 4 からの 1 行を結合し、ステップ 1 に行を 1 つ戻します。
 4. ステップ 6 を実行し、結果の行があればステップ 1 に戻します。
 5. ステップ 1 を実行します。ステップ 6 から行が戻されなければ、SQL 文を発行したユーザーにステップ 2 から受け取った行を戻します。

ステップ 3 で戻された行ごとに、ステップ 5、4、2、6 および 1 が 1 回ずつ実行されるため注意してください。親ステップが、実行前に必要とするのが子ステップからの 1 行のみであれば、Oracle は子ステップから 1 行が戻されると、ただちに親ステップ（および実行計画の残りの部分）を実行します。行が 1 つ戻されると親ステップの親をアクティブにできる場合は、その親も実行されます。

こうして、ツリーの上位へと連鎖的に実行できます。Oracle は、子ステップで順次それぞれの行が取り出されるたびに、親ステップとすべてのカスケード・ステップを 1 回実行します。子ステップから各行が戻されるたびに起動される親ステップには、表アクセス、索引アクセス、ネステッド・ループ・ジョインおよびフィルタがあります。

子ステップからの行がすべてないと親ステップを実行できない場合、すべての行が子ステップから戻されるまで、親ステップを実行できません。このような親ステップとしては、ソート、ソート・マージ結合および集計関数があります。

オブティマイザのプラン・スタビリティ

アプリケーションを慎重にチューニングした後で、同じ SQL 文が実行されると常に同じ実行計画がオブティマイザで生成されるかどうかを確認する必要があります。「プラン・スタビリティ」により、データベースに対する変更に関係なく、同じ SQL 文について同じ実行計画を維持できます。データベースに対する変更には、次のものがあります。

- 表の再分析
- データの追加または削除
- 表の列、定数または索引の変更
- システム構成の変更
- オブティマイザの新バージョンへのアップグレード

CREATE OUTLINE は、オブティマイザで実行計画の作成に使用される属性の集合を含む、「ストアド・アウトライン」を作成します。ストアド・アウトラインは、システム・パラメータ CREATE_STORED_OUTLINES を TRUE に設定して自動的に作成する方法もあります。

システム・パラメータ USE_STORED_OUTLINES は、TRUE、FALSE またはカテゴリ名に設定し、実行する問合せに既存のストアド・アウトラインを使用するかどうかを指定できます。OUTLN_PKG パッケージは、ストアド・アウトラインの管理に使用するプロシージャを提供します。

プラン・スタビリティの実装によって、DBA 権限を持つ新しいスキーマ OUTLN が作成されます。データベース管理者は、SYS および SYSTEM スキーマの場合と同様に、OUTLN スキーマのパスワードを変更する必要があります。

関連項目：

- プラン・スタビリティの使用法および CREATE OUTLINE 文の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- OUTLN_PKG パッケージの詳細は、『Oracle8i PL/SQL 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。

コストベース最適化

コストベースのアプローチを使用すると、オプティマイザは使用可能なアクセス・パスと、SQL 文がアクセスするスキーマ・オブジェクト（表または索引）に関する統計情報の要素を考慮し、最も効率的な実行計画を判断します。コストベースのアプローチでは、ヒント、つまり文のコメントとして示されている最適化に関する提案も考慮に入れられます。

コストベースのアプローチは、概念的に次のようなステップで構成されています。

1. オプティマイザは、使用可能なアクセス・パスとヒントに基づいて、SQL 文の実行計画の集合を生成します。
2. オプティマイザは、文がアクセスする表、索引およびパーティションのデータ分布および記憶特性に関するデータ・ディクショナリ内の統計に基づいて、各実行プランのコストを見積ります。

「コスト」は、特定の実行計画を使用して文を実行するために必要な、予想されるリソースの使用量に比例する推定値です。オプティマイザは、計画を使用して文を実行するために必要な I/O、CPU 時間およびメモリーなど、コンピュータ・リソースの推定に基づいて、考えられる各アクセス方法と結合順序のコストを計算します。

コストが高いシリアル実行計画には、コストが低い実行計画よりも実行に時間がかかります。ただし、パラレル実行計画を使用する場合、リソースの使用状況と経過時間の間に直接的な関係はありません。

3. オプティマイザは、複数の実行計画のコストを比較して、コストが最も低い実行計画を選択します。

コストベース・アプローチの目標

コストベースのアプローチの目標は、デフォルトでは、最大の「スループット」を達成すること、つまり文がアクセスするすべての行を処理するために必要なリソース使用量を最小にすることです。

また、Oracle では、「応答時間」を最短に、つまり SQL 文がアクセスする最初の行を処理するのに必要なリソース使用量を最小に抑えることを目標にして、文を最適化することもできます。

SQL 文の平行実行の場合、オプティマイザはリソース使用と引換えに、経過時間を最小にするよう選択できます。オプティマイザで実行の平行化をどの程度試行するかを指定するには、初期化パラメータ `OPTIMIZER_PERCENT_PARALLEL` を使用します。

関連項目： `OPTIMIZER_PERCENT_PARALLEL` パラメータの使用方法和、オプティマイザが最適化のアプローチおよび目標を選択する方法の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

コストベース最適化の統計

コストベースのアプローチでは、統計を使用して述語の選択性を計算し、各実行計画のコストを推定します。「選択性」は、SQL 文の述語によって表内で選択される行の割合です。オプティマイザは、述語の選択性を使用して、特定のアクセス方法のコストを推定し、最適の結合順序を判断します。

統計データは、表、列、索引およびパーティションのデータ分布と記憶特性を数量化したものです。オプティマイザは、特定の実行計画を使用して SQL 文を実行する場合に、どの程度の I/O および CPU 時間、メモリーが必要になるかを、この統計情報を使用して推定します。統計はデータ・ディクショナリに格納されており、あるデータベースからエクスポートして別のデータベースにインポートできます。たとえば、テスト・システムにデータのサンプルが少数しかなくても、本番の統計をテスト・システムに転送して現実の環境をシミュレーションできます。

オプティマイザにスキーマ・オブジェクト関連の情報を提供するには、統計を定期的に収集する必要があります。スキーマ・オブジェクトのデータや構造が変更されたために、前の統計が不正確になる場合は、新しい統計を収集してください。たとえば、きわめて多数の行を表にロードした後は、行数に関して新しい統計を収集します。表内のデータを更新した後は、行数について新しい統計を収集する必要はありませんが、行の平均の長さについての新しい統計が必要になる場合があります。

関連項目： 21-13 ページの「統計収集」

コストベース最適化のヒストグラム

コストベース最適化では、データ値ヒストグラムを使用して列データの分布を正確に推定します。「ヒストグラム」は、帯に含まれるすべての列値が同じ範囲に含まれるように、列値を帯状にパーティション化したものです。ヒストグラムを使用すると、データの偏りが存在する場合の選択性の推定を改善でき、その結果、不均一なデータ分布がある場合でも最適の実行計画が作成されます。ヒストグラム内の範囲を「バケット」と呼びます。

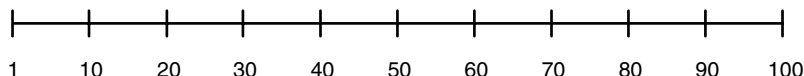
コストベース最適化の基本的な機能の1つは、問合せに含まれている述語の選択性を決定することです。選択性の推定は、索引の使用時期を決定するときと、表を結合する順序を決定するときに使用します。ほとんどの属性ドメイン（表の列）は、均一に分布していません。

コストベース最適化は、不均一なドメインの分布を示すために、指定した属性に対して「高さ調整ヒストグラム」を使用します。高さ調整ヒストグラムでは、列値は各帯にほぼ同数の値が含まれるように、複数の帯に分割されます。その後で、ヒストグラムから得られた各エンドポイントが入る値の範囲を有効な情報として活用します。

高さ調整ヒストグラムと幅調整ヒストグラムの比較 Oracle では、「幅調整」とは反対の「高さ調整」ヒストグラムが使用されます。この2種類のヒストグラムの相違点は、次のとおりです。

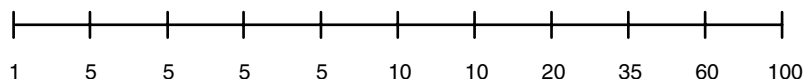
- 幅調整ヒストグラムでは、それぞれの値の範囲が同じ幅になるように、データを特定の数に分けてから、それぞれの範囲内にある値の数をカウントします。
- 高さ調整ヒストグラムでは、各範囲にほぼ同数の値が入るようにし、その範囲に入っている値の数によって範囲のエンドポイントが決められます。

値が1～100までの列Cと、10個の枠（バケット）からなるヒストグラムがあるとします。Cのデータが均一に分布している場合、このヒストグラムは次のようになります。数値はエンドポイント値です。



それぞれのバケットに入っている行数は、表の合計行数の10分の1です。分布が均一なこの例では、全体の4割の行が60～100の範囲の値になっています。

データが均一に分布していない場合、ヒストグラムは次のようになります。



この場合、ほとんどの行でこの列の値は5です。この例では、値が60～100の行が全体の1割しかありません。

たとえば、1000行を含む表の1つの列に入っている値の範囲が1～100である場合に、10バケットのヒストグラムを作成するとします。幅調整ヒストグラムでは、すべてのバケットが同じ幅（1-10、11-20、21-30、など）になり、各バケットはそのバケットの範囲内にある行数をカウントします。高さ調整ヒストグラムでは、各バケットは同じ高さであり（この場合は100行）、列内の個々の値の密度に基づいて各バケットのエンドポイントが決まります。

高さのバランスを調整するアプローチは、データが非常に不整であるときに威力を発揮します。1000行を含む表のうち800行に値5が入っており、残りの200行は1～100に均一に分布しているとします。幅調整ヒストグラムでは、バケット1～10に820行が入り、他の各バケットに約20行ずつ入ります。高さベースのヒストグラムの場合、1～5のバケットが1つ、5～50のバケットが7つ、50～100のバケットが1つになります。

表内の行のうち値が5の行の数を知りたい場合、高さ調整ヒストグラムを見ると、約80%の行がその値であることがわかります。ただし、幅調整ヒストグラムの場合、5の値と6の値を区別するための手段がありません。このヒストグラムで計算した場合、値が5の行は全体の8%の行のみになります。したがって、列値の選択性を判断するには、高さ調整ヒストグラムのほうが適しています。

ヒストグラムを使用する場合 パフォーマンスに影響を与える可能性があるため、ヒストグラムを使用するのは、問合せ計画を実際に改善できる場合のみにしてください。通常は、問合せのWHERE句での使用頻度が高く、非常に不整なデータ分布を持つ列について、ヒストグラムを作成する必要があります。多くのアプリケーションの場合、一般に索引列はWHERE句で最も使用頻度の高い列であるため、適切な方法はすべての索引列についてヒストグラムを作成することです。

ヒストグラムは永続オブジェクトであるため、使用するとメンテナンスと領域に関するコストが生じます。ヒストグラムは、データ分布が非常に不整であることがわかっている列についてのみ計算してください。均一に分布するデータの場合、コストベース最適化では、ヒストグラムを使用しなくても、特定の文の実行コストを正確に推定できます。

ヒストグラムは、他のすべてのオブティマイザ統計と同様に静的です。ヒストグラムが有用なのは、ある列の現在のデータ分布がヒストグラムに反映される場合のみです。データの「分布」が一定であれば、列データを変更できます。列のデータ分布が頻繁に変化する場合は、そのヒストグラムを頻繁に再計算する必要があります。

列に次のような特性がある場合、ヒストグラムは役立ちません。

- その列のすべての述語でバインド変数を使用している場合。
- その列のデータが均一に分布している場合。
- 問合せの WHERE 句でその列が使用されていない場合。
- その列が一意で、等価述語のみで使用されている場合。

ヒストグラムを生成するには、DBMS_STATS パッケージまたは ANALYZE 文を使用します。表またはパーティションの列のヒストグラムを生成できます。ヒストグラムの統計は、パラレルには収集されません。

ヒストグラム情報は、次のデータ・ディクショナリ・ビューを使用して表示できます。

- USER_HISTOGRAMS、ALL_HISTOGRAMS および DBA_HISTOGRAMS
- USER_PART_HISTOGRAMS、ALL_PART_HISTOGRAMS および DBA_PART_HISTOGRAMS
- USER_SUBPART_HISTOGRAMS、ALL_SUBPART_HISTOGRAMS および DBA_SUBPART_HISTOGRAMS
- TAB_COLUMNS

関連項目：

- 21-13 ページの「統計収集」
- ヒストグラムの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』

パーティション化されたスキーマ・オブジェクトの統計

パーティション化されたスキーマ・オブジェクトは、統計の集合を複数含むことができます。つまり、次を参照する統計を含むことができます。

- スキーマ・オブジェクト全体（グローバル統計）
- 個々のパーティション
- コンポジット・パーティション・オブジェクトの個々のサブパーティション

問合せの述語によって対象を単一パーティションに限定しなければ、オブティマイザはグローバル統計を使用します。ほとんどの問合せはこのように限定的ではないため、正確なグローバル統計を生成することが最も重要になります。パーティション・レベルの統計からグローバル統計を生成するのは簡単なように思われますが、これは一部の統計にしか当てはまりません。たとえば、各パーティション内の各値の数から 1 列の個別値の数を検出することは、値がオーバーラップしている可能性があるためきわめて困難です。したがって、実際にグローバル統計を収集するときには、ANALYZE 文を使用して計算するのではなく、DBMS_STATS パッケージを使用することをお勧めします。

注意： 現在、Oracle はグローバル・ヒストグラム統計を収集しません。

関連項目： 21-15 ページの「[ANALYZE 文](#)」

統計収集

この項では、様々な統計収集方法について説明します。

DBMS_STATS パッケージ PL/SQL パッケージ DBMS_STATS を使用すると、コストベース最適化のための統計を生成して管理できます。このパッケージでは、統計を収集、変更、表示および削除できます。また、このパッケージを使用して統計の集合を格納することも可能です。

DBMS_STATS パッケージでは、索引、表、列およびパーティションの統計や、スキーマまたはオブジェクト内のすべてのスキーマ・オブジェクトの統計を収集できます。クラスタ統計は収集されませんが、DBMS_STATS を使用するとクラスタ全体ではなく個々の表の統計を収集できます。

統計収集操作は、シリアルまたはパラレルに実行できます。可能であれば、DBMS_STATS はパラレル問合せをコールして、指定した並列度で統計を収集します。それ以外の場合は、シリアル問合せまたは ANALYZE 文をコールします。索引統計は、パラレルには収集されません。

この統計は、正確に計算するか、行またはブロックのランダムなサンプリングから推定できます。

パーティション表および索引の場合、DBMS_STATS ではパーティションごとに別々の統計を収集したり、表または索引全体のグローバル統計を収集できます。同様に、コンポジット・パーティション化の場合、DBMS_STATS ではサブパーティション、パーティションおよび表または索引全体について、別々の統計を収集できます。最適化する SQL 文に応じて、オブティマイザはパーティション（またはサブパーティション）の統計とグローバル統計のどちらを使用するかを選択します。

DBMS_STATS が収集するのはコストベース最適化のための統計のみで、他の統計は収集しません。たとえば、DBMS_STATS は、次の表統計を収集します。

- 行数
- その時点でデータが含まれているブロック数
- 行の長さの平均

DBMS_STATS は、次の表統計は収集しません。

- 連鎖した行数
- 空き領域の平均
- 未使用のデータ・ブロック数

関連項目：

- 21-16 ページの「[統計表](#)」
- 21-15 ページの「[正確な統計と推定による統計](#)」
- DBMS_STATS パッケージを使用した統計の収集方法の例は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

索引用の COMPUTE STATISTICS 句 Oracle は、B-tree 索引またはビットマップ索引の作成または再作成中に、一部の統計を自動的に収集できます。この統計収集を使用可能にするには、CREATE INDEX または ALTER INDEX ... REBUILD の COMPUTE STATISTICS 句を使用します。

COMPUTE STATISTICS 句を指定した場合に Oracle によって収集される統計は、索引がパーティション化されているかどうかによって異なります。

- 非パーティション索引の場合、索引の作成または再作成中には、索引、表および列の統計が収集されます。連結キー索引の場合、列統計はキーの先行列のみを参照します。
- パーティション索引の場合、索引の作成中またはそのパーティションの再作成中には、表または列の統計は収集されません。
 - － パーティション索引の作成時には、各パーティションと索引全体の索引統計が収集されます。索引がコンポジット・パーティション化を使用している場合は、各サブパーティションの統計も収集されます。
 - － 索引のパーティションまたはサブパーティションの再作成中には、そのパーティションまたはサブパーティションの索引統計のみが収集されます。

統計の正確さを保証するために、COMPUTE STATISTICS 句を指定して索引を作成するときには、索引作成に他の索引が使用可能な場合でも、常にベース表が使用されます。

関連項目： CREATE INDEX および ALTER INDEX 文の COMPUTE STATISTICS 句の詳細は、『Oracle8i SQL リファレンス』を参照してください。

ANALYZE 文 ANALYZE 文を使用すると、コストベース最適化のための統計も生成できます。次のように各種の制限を伴うため、この目的に ANALYZE 文を使用することはお薦めしません。

- ANALYZE は常にシリアルに実行されます。
- ANALYZE では、パーティション表とパーティション索引のグローバル統計が、直接収集されないで計算されます。このため、個別値の数など、一部の統計が不正確になる可能性があります。
 - － パーティション表とパーティション索引の場合、ANALYZE では個々のパーティションの統計が収集された後に、パーティション統計からグローバル統計が計算されます。
 - － コンポジット・パーティション化の場合、ANALYZE ではサブパーティションの統計が収集された後に、サブパーティション統計からパーティション統計とグローバル統計が計算されます。
- ANALYZE では、DBMS_STATS によって収集された統計値の一部を上書きも削除もできません。

ANALYZE では、連鎖行に関する情報や、索引、表およびクラスタの構造の一貫性に関する情報など、オブティマイザが使用しない追加情報を収集できます。この情報は、DBMS_STATS では収集されません。

関連項目： ANALYZE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

正確な統計と推定による統計 DBMS_STATS パッケージまたは ANALYZE 文によって収集される統計には、正確な統計と推定による統計があります。(索引の作成または再作成の COMPUTE STATISTICS 句を指定すると、常に正確な統計が収集されます。)

正確な統計を計算するには、Oracle は索引、表、パーティションまたはスキーマ内のすべてのデータを読み込む必要があります。その時点で表にデータが含まれているデータ・ブロック数や、ルート・ブロックからリーフ・ブロックまでの索引の深さなど、一部の統計は常に正確に計算されます。

統計を推定するために、Oracle はデータのサンプルをランダムに選択します。サンプリング率と、サンプリングの基礎を行にするかブロックにするかを指定できます。

「ROW サンプリング」では、ディスク上の物理位置に関係なく行が読み込まれます。この方法では、推定用に最もランダムなデータが得られますが、必要以上のデータが読み込まれることがあります。たとえば、最悪の場合は、行サンプルによって各ブロックから 1 行ずつ選択され、フル・テーブル・スキャンや全索引スキャンが必要になることがあります。

「Block サンプリング」では、ブロックのサンプルがランダムに読み込まれ、そのブロック内のすべての行が推定に使用されます。この方法では、特定のサンプル・サイズに対する I/O アクティビティは減少しますが、行がディスク上にランダムに分布していないと、サンプルのランダム性が低下するおそれがあります。Block サンプリングは、索引統計には使用できません。

統計の管理

この項では、統計表と、データ・ディクショナリに格納された統計情報を表示するビューについて説明します。

統計表 DBMS_STATS パッケージを使用すると、統計を「統計表」に格納できます。列、表、索引またはスキーマに関する統計を統計表に転送し、後からデータ・ディクショナリにリストアできます。オプティマイザは、統計表に格納されている統計は使用しません。

統計表により、様々な統計の集合を試験的に使用できます。たとえば、統計の集合のバックアップを作成してから、オリジナルの統計を削除または変更したり、新規に生成できます。その後、様々な統計の集合にあわせて最適化された SQL 文のパフォーマンスを比較し、表に格納されている統計が最高のパフォーマンスを発揮した場合は、それをデータ・ディクショナリにリストアできます。

その場合、統計の集合を 1 つの統計表にまとめて保存する方法と、複数の統計表を作成して別々に格納する方法があります。

統計の表示 DBMS_STATS パッケージを使用すると、データ・ディクショナリや統計表に格納されている統計を表示できます。

また、データ・ディクショナリ内の統計を問い合わせるときには、次のデータ・ディクショナリ・ビューも使用できます。

- USER_TABLES、ALL_TABLES および DBA_TABLES
- USER_TAB_COLUMNS、ALL_TAB_COLUMNS および DBA_TAB_COLUMNS
- USER_INDEXES、ALL_INDEXES および DBA_INDEXES
- USER_CLUSTERS および DBA_CLUSTERS
- USER_TAB_PARTITIONS、ALL_TAB_PARTITIONS および DBA_TAB_PARTITIONS
- USER_TAB_SUBPARTITIONS、ALL_TAB_SUBPARTITIONS および DBA_TAB_SUBPARTITIONS

- USER_IND_PARTITIONS、ALL_IND_PARTITIONS および DBA_IND_PARTITIONS
- USER_IND_SUBPARTITIONS、ALL_IND_SUBPARTITIONS および DBA_IND_SUBPARTITIONS
- USER_PART_COL_STATISTICS、ALL_PART_COL_STATISTICS および DBA_PART_COL_STATISTICS
- USER_SUBPART_COL_STATISTICS、ALL_SUBPART_COL_STATISTICS および DBA_SUBPART_COL_STATISTICS

関連項目： これらのビュー内の統計の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

コストベースのアプローチを使用する場合

一般に、新しいアプリケーションには、すべてコストベースのアプローチを使用してください。ルールベースのアプローチは、コストベースの最適化が使用できるようになる前に作成されたアプリケーションのためのものです。コストベースの最適化は、リレーショナル・データとオブジェクト型のどちらにも使用できます。

次の機能に対しては、コストベースの最適化しか使用できません。

- パーティション表
- パーティション・ビュー
- 索引構成表
- 逆キー索引
- ビットマップ索引
- ファンクション索引
- SELECT 文の SAMPLE 句
- パラレル問合せおよびパラレル DML
- スター型変換
- スター型結合
- 拡張可能な最適化

関連項目： コストベース・アプローチをどのようなときに使用するかの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

拡張可能な最適化

拡張可能な最適化により、ユーザー定義関数とドメイン索引の作成者は、コストベース最適化で実行プランの選択に使用される、統計、選択性およびコスト評価という3つの主要コンポーネントを制御できます。

拡張可能な最適化により、次のことが可能になります。

- コスト関数とデフォルト・コストを、ドメイン索引、索引タイプ、パッケージおよびスタンドアロン・ファンクションに対応付けます。
- 選択関数とデフォルトの選択性を、オブジェクト型、パッケージ・ファンクションおよびスタンドアロン・ファンクションのメソッドに対応付けます。
- 統計収集関数を表のドメイン索引および列に対応付けます。
- コストに基づいて述語を関数で順序付けします。
- アクセス・コストに基づいて表のユーザー定義アクセス方法（ドメイン索引）を選択します。
- ANALYZE 文を使用して、ユーザー定義の統計収集関数および統計削除関数を起動します。
- 新しいデータ・ディクショナリ・ビューを使用して、列、ドメイン索引、索引タイプまたは関数に対応付けた統計収集関数、コスト関数または選択関数に関する情報を含めます。
- ヒントを追加して、関数の述語の評価順を保ちます。

関連項目： 拡張可能な最適化の詳細は、『Oracle8i データ・カートリッジ 開発者ガイド』を参照してください。

ユーザー定義統計

ドメイン索引、表の個々の列およびユーザー定義データ型に対して、「統計収集関数」を定義できます。

統計を収集するためにドメイン索引が分析されるたびに、Oracle はそれに対応する統計収集関数をコールします。表列が分析されると、Oracle はその列の標準統計を収集し、対応する統計収集関数をコールします。データ型の統計収集関数が存在する場合、Oracle は分析される表内でそのデータ型を持つ列ごとに、統計収集関数をコールします。

ユーザー定義選択性

SQL 文の述語の選択性は、特定のアクセス方法のコストを推定するため、および最適の結合順序を決定するために使用されます。ユーザー定義オペレータに関する情報はなため、この種のオペレータを含む述語については、正確な選択性を計算できません。

ユーザー定義オペレータ、スタンドアロン・ファンクション、パッケージ・ファンクションまたは型のメソッドを含む述語に対して、「選択関数」を定義できます。オブティマイザは、定数 <、<=、=、>=、> または LIKE のいずれかに関連してオペレータ、関数またはメソッドを含む述語を検出すると、ユーザー定義の選択関数をコールします。

ユーザー定義コスト

オブティマイザは、ドメイン索引の内部記憶構造を認識しないため、この種の索引のコストの正確な推定を計算できません。また、オブティマイザは、次のいずれかの特性を持つユーザー定義関数については、コストを過小に推定する場合があります。

- PL/SQL の起動
- 再帰 SQL の使用
- BFILE へのアクセス
- CPU の大量使用

次のコストを定義できます。

- ドメイン索引
- ユーザー定義のスタンドアロン・ファンクション
- パッケージ・ファンクション
- 型のメソッド

これらのユーザー定義コストは、オブティマイザが単に参照するのみのデフォルト・コスト形式で使用方法と、オブティマイザがコストを計算するためにコールする完全なコスト関数として使用方法があります。

ルールベース最適化

ルールベースのアプローチを使用すると、オプティマイザは、使用可能なアクセス・パスと、そのアクセス・パスのランクに基づいて実行計画を選択します。ルールベース最適化は、リレーショナル・データへのアクセスにも、オブジェクト型へのアクセスにも使用できます。

Oracle のアクセス・パスのランク付けは、ヒューリスティックな手法です。SQL 文を実行する方法が 2 つ以上ある場合、ルールベースのアプローチでは、常にランクが低いほうの操作を使用します。通常、ランクの低い操作は、高いほうのランクの構成体に対応付けされている操作よりも高速に実行されます。

注意： Oracle8i の一部の拡張機能には、ルールベース最適化を使用できません。

関連項目：

- 『Oracle8i パフォーマンスのための設計およびチューニング』
- ルールベース最適化を使用できない機能のリストは、21-17 ページの「[コストベースのアプローチを使用する場合](#)」を参照してください。

第 VII 部

パラレル SQL とダイレクト・ロード・ インサート

第 VII 部では、SQL 文のパラレル実行とダイレクト・ロード・インサート機能について説明します。含まれる章は、次のとおりです。

- [第 22 章「ダイレクト・ロード・インサート」](#)
- [第 23 章「SQL 文のパラレル実行」](#)

ダイレクト・ロード・インサート

この章では、シリアル・インサートまたはパラレル・インサート用の Oracle ダイレクト・ロード・インサート機能について説明します。ダイレクト・ロード・インサートといくつかの DDL 文で使用可能な NOLOGGING 機能についても説明します。この章のトピックは次のとおりです。

- [ダイレクト・ロード・インサートの概要](#)
- [ダイレクト・ロード・インサート文の種類](#)
 - [シリアル INSERT とパラレル INSERT](#)
 - [ロギング・モード](#)
- [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](#)
- [ダイレクト・ロード・インサートの制限事項](#)

注意： この章で説明するパラレル・ダイレクト・ロード・インサート機能は、Oracle8i Enterprise Edition を購入した場合にのみ使用可能です。

関連項目：

- パラレル実行インサートの注意点の詳細は、[第 23 章「SQL 文のパラレル実行」](#)を参照してください。
- パラレル・ダイレクト・ロード・インサートのチューニング方法の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

ダイレクト・ロード・インサートの概要

「ダイレクト・ロード・インサート」は、データをフォーマットして Oracle データ・ファイルに書き込む操作を、バッファ・キャッシュを使用せずに直接実行することにより、インサート操作のパフォーマンスを改善します。この機能は、ダイレクト・ローダー・ユーティリティ (SQL*Loader) の機能と似ています。

ダイレクト・ロード・インサートは、挿入するデータを、表中の既存のデータの後に追加します。既存のデータ内の空き領域は再利用されません。データは、パーティション表または非パーティション表にパラレルまたはシリアルで挿入できます。

パラレル化、表のパーティション化およびロギングに関して、ダイレクト・ロード・インサートにはいくつかのオプションがあります。

関連項目：

- 22-3 ページの「[ダイレクト・ロード・インサート文の種類](#)」
- ダイレクト・ロード・インサートのパラレル化オプションと Partitioning Option の詳細は、[第 23 章「SQL 文のパラレル実行」](#)を参照してください。

ダイレクト・ロード・インサートの利点

ダイレクト・ロード・インサートを使用することの主な利点は、REDO エントリまたは UNDO エントリのログを記録せずにデータをロードできることです。これにより、挿入時のパフォーマンスが著しく改善されます。シリアルとパラレルのどちらでも、ダイレクト・ロード・インサートには、従来型パス INSERT と比べてパフォーマンス上の利点があります。

対照的に、従来型パス INSERT を使用した場合は、オブジェクト内の空き領域が再利用され、参照整合性が維持されます。挿入のための従来型パスは、パラレル化できません。

CREATE TABLE ... AS SELECT との比較

ダイレクト・ロード・インサートを使用すると、新しい表を作成するかわりに、既存の表にデータを挿入できます。ダイレクト・ロード・インサートは表の索引を更新しますが、CREATE TABLE ... AS SELECT は索引のない新しい表を作成するのみです。

関連項目： 23-32 ページの「[パラレルの CREATE TABLE ... AS SELECT](#)」

パラレル・ダイレクト・ロード（SQL*Loader）と比較した場合の利点

パラレル INSERT を使用する場合は、必ずトランザクションが最小単位になります。複数のパラレル・ロードを使用する場合には、最小単位としての実行が保証されません。さらに、パラレル・ロードでは、索引の更新中にエラーが発生した場合に、ロード終了時に表の索引のいくつかが UNUSABLE 状態のままになることがあります。対照的に、パラレル INSERT では、表と索引がアトミックに更新されます（つまり、索引の更新中にエラーが発生すると、文はロールバックされます）。

関連項目： パラレル・ロードの詳細は、『Oracle8i ユーティリティ・ガイド』参照してください。

INSERT ... SELECT 文

ダイレクト・ロード・インサート（シリアルまたはパラレル）でサポートされるのは、INSERT 文の INSERT ... SELECT 構文のみで、INSERT... *values* の構文はサポートされていません。INSERT ... SELECT のパラレル化は、パラレル・ヒントまたはパラレルの表定義句のどちらかによって決定されます。

関連項目： INSERT ... SELECT 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。

ダイレクト・ロード・インサート文の種類

ダイレクト・ロード・インサートは、次の方法で実行できます。

- シリアルまたはパラレル
- パーティション表または非パーティション表に対して
- REDO データのロギングありまたはロギングなし

シリアル INSERT とパラレル INSERT

ダイレクト・ロード・インサートはパーティション表または非パーティション表に対して実行でき、シリアルでもパラレルでも実行できます。

- **パーティション表または非パーティション表へのシリアル・ダイレクト・ロード・インサート。** データは、表のセグメントまたは各パーティションのセグメントの現行の高水位標を超えて挿入されます。「高水位標」は、ブロックがデータ受入れ用にフォーマットされたことのないレベルです。コミットを実行すると、高水位標が新しい値に更新され、他のプロセスからもデータが見えるようになります。
- **非パーティション表へのパラレル・ダイレクト・ロード・インサート。** 各パラレル実行サーバーが新しい一時セグメントを割り当てて、そこにデータを挿入します。コミットを実行すると、パラレル実行コーディネータは新しい一時セグメントを 1 次表セグメントにマージします。

- **パーティション表へのパラレル・ダイレクト・ロード・インサート。**各パラレル実行サーバーに1つ以上のパーティションが割り当てられ、どのパーティションでも1つのプロセスしか作動していません。パラレル・サーバー・プロセスは、割り当てられたパーティション・セグメントの現行の高水位標の後にデータを挿入します。コミットを実行すると、各パーティション・セグメントの高水位標がパラレル実行コーディネータによって新しい値に更新され、他のプロセスからもデータが見えるようになります。

いずれにしても、高水位標の更新または一時セグメントのマージは、実行されるとすぐに他のプロセスからもデータが見えてしまうので、コミットが発行されるまで待ってから実行されます。つまり、挿入操作をコミットしてしまいます。

関連項目： 23-5 ページの「[パラレル実行のプロセス・アーキテクチャ](#)」

シリアル・ダイレクト・ロード・インサートとパラレル・ダイレクト・ロード・インサートの指定

シリアル・ダイレクト・ロード・インサートを使用するには、APPEND ヒントが必要です。パラレル・ダイレクト・ロード・インサートの場合、文中の PARALLEL ヒント、または表定義の PARALLEL 句が必要です。APPEND ヒントはオプションです。パラレル・ダイレクト・ロード・インサートでは、パラレル DML を ALTER SESSION ENABLE/FORCE PARALLEL DML 文を使用してオンにする必要もあります。

[表 22-1](#) に、ダイレクト・ロード・インサートの要件および従来型 INSERT との比較をまとめます。

表 22-1 シリアルおよびパラレルの INSERT ... SELECT 文のまとめ

挿入のタイプ	シリアル	パラレル
ダイレクト・ロード・インサート	可能。SQL 文での APPEND ヒントが必要。	可能。次のものが必要。 <ul style="list-style-type: none">■ ALTER SESSION ENABLE/FORCE PARALLEL DML■ 表の PARALLEL 属性または文の PARALLEL ヒント■ APPEND ヒントはオプション
従来型 INSERT	可能（デフォルト）。	不可。

シリアル・ダイレクト・ロード・インサートとパラレル・ダイレクト・ロード・インサートの例

シリアル・ダイレクト・ロード・インサートは、APPEND ヒントで指定できます。次に例を示します。

```
INSERT /*+ APPEND */ INTO emp
      SELECT * FROM t_emp;
COMMIT;
```

パラレル・ダイレクト・ロード・インサートは、行の挿入先の表の PARALLEL 属性を設定すると指定できます。次に例を示します。

```
ALTER TABLE emp PARALLEL (10);
ALTER SESSION ENABLE PARALLEL DML;
INSERT INTO emp
      SELECT * FROM t_emp;
COMMIT;
```

行を選択する選択元の表の PARALLEL 属性を設定すると、次のように、SELECT 操作のパラレル化も指定できます。

```
ALTER TABLE emp PARALLEL (10);
ALTER TABLE t_emp PARALLEL (10);
ALTER SESSION ENABLE PARALLEL DML;
INSERT INTO emp
      SELECT * FROM t_emp;
COMMIT;
```

INSERT 操作または SELECT 操作のための PARALLEL ヒントは、表の PARALLEL 属性より優先されます。たとえば、次の INSERT ... SELECT 文の並列度は、EMP 表と T_EMP 表に PARALLEL 属性が設定されているかどうかには関係なく、12 になります。

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL(emp,12) */ INTO emp
      SELECT /*+ PARALLEL(t_emp,12) */ * FROM t_emp;
COMMIT;
```

関連項目： 23-23 ページの「[INSERT ... SELECT のパラレル化のルール](#)」

ロギング・モード

ダイレクト・ロード・インサートの操作は、REDO 情報のロギングありでもロギングなしでも実行できます。データ挿入先の表、パーティションまたは索引にロギングなしのモードを指定するには、ALTER TABLE 文、ALTER INDEX 文または ALTER TABLESPACE 文を使用します。

- **ロギングありのダイレクト・ロード・インサート。**このモードでは、インスタンスおよびメディア・リカバリ用に完全な REDO ログが生成されます。「ロギングあり」がデフォルトのモードです。オンライン REDO ログをテープにアーカイブするには、データベースを ARCHIVELOG モードにする必要があります。そうしないと、インスタンス・クラッシュはリカバリ可能ですが、ディスク障害はリカバリできません。
- **ロギングなしのダイレクト・ロード・インサート。**このモードでは、データは挿入されますが、REDO または UNDO ログは生成されません。(ただし、新しいエクステントが無効であることをマークするために最小限のロギングが実行されます。またディクショナリの変更については常に完全なログが記録されます。) メディア・リカバリ時にこれが適用されると、REDO データのログが記録されていないため、「エクステント無効」レコードは特定のブロック範囲を論理的に破損しているものとしてマークします。

ロギングなしモードにすると、生成されるログ・データが非常に少なくなるため、パフォーマンスが向上します。メディア・リカバリを可能にするには、ロギングなしの挿入操作を実行した後、ユーザーの責任でデータのバックアップを作成する必要があります。

ロギングなしモードとディスクリット・トランザクションの間に相互作用はなく、ディスクリット・トランザクションでは常に REDO 情報が生成されます。ディスクリット・トランザクションは、NOLOGGING 属性を使用する表に対して発行できます。

注意： ロギングありモードとロギングなしモードは、表、パーティションまたは索引の永続的な属性ではありません。挿入先のデータベース・オブジェクトにデータを挿入し、バックアップを作成した後は、オブジェクトの状況をロギングありモードにして、それ以降の変更のログが記録されるように設定できます。

表 22-2 に、ダイレクト・ロード・インサートと従来型 INSERT について、それぞれの LOGGING モードと NOLOGGING モードとの比較を示します。

表 22-2 LOGGING 句と NOLOGGING 句のまとめ

挿入のタイプ	LOGGING	NOLOGGING
ダイレクト・ロード・インサート	可能。リカバリ可能にするには ARCHIVELOG データベース・モードが必要。	可能。表領域、表、パーティションまたは索引の NOLOGGING 属性が必要。
従来型 INSERT	可能 (デフォルト)。リカバリ可能にするには ARCHIVELOG データベース・モードが必要。	不可。

関連項目： 16-9 ページの「ディスクリット・トランザクションの管理」

ロギングなしモードの例

行の挿入先の表の NOLOGGING 属性を設定すると、ダイレクト・ロード・インサートにロギングなしモードを指定できます。次に例を示します

```
ALTER TABLE emp NOLOGGING;  
ALTER SESSION ENABLE PARALLEL DML;  
INSERT /*+ PARALLEL(emp,12) */ INTO emp  
      SELECT /*+ PARALLEL(t_emp,12) */ * FROM t_emp;  
COMMIT;
```

パーティション、表領域または索引にも、NOLOGGING 属性を設定できます。次に例を示します。

```
ALTER TABLE emp MODIFY PARTITION emp_lmnp NOLOGGING;
```

```
ALTER TABLESPACE personnel NOLOGGING;
```

```
ALTER INDEX emp_ix NOLOGGING;
```

```
ALTER INDEX emp_ix MODIFY PARTITION eix_lmnp NOLOGGING;
```

ロギングなしモードを使用できる SQL 文

表、パーティション、索引または表領域に NOLOGGING 属性を設定できますが、ロギングなしモードは、NOLOGGING 属性を設定したスキーマ・オブジェクトに対して実行される操作のすべてに適用されるわけではありません。ロギングなしモードを使用できるのは、次の操作のみです。

- ダイレクト・ロード (SQL*Loader)
- ダイレクト・ロード・インサート
- CREATE TABLE ...AS SELECT
- CREATE INDEX
- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER INDEX ... SPLIT PARTITION
- ALTER INDEX ... REBUILD
- ALTER INDEX ...REBUILD PARTITION
- オフラインで格納される NOCACHE NOLOGGING モードの LOB の INSERT、UPDATE および DELETE

これらの SQL 文はすべてパラレル化できます。これらにより、シリアル実行とパラレル実行の両方を、ロギングありモードまたはロギングなしモードで実行できます。

その他の SQL 文は、スキーマ・オブジェクトの NOLOGGING 属性の影響を受けません。たとえば、このような SQL 文には、UPDATE と DELETE（前述のように一部の LOB を除く）、従来型パスの INSERT および前述のリストにない各種の DDL 文があります。

関連項目： 第 23 章「SQL 文のパラレル実行」

デフォルトのロギング・モード

LOGGING 句または NOLOGGING 句が指定されていない場合、表、パーティションまたは索引のロギング属性のデフォルトは、それが存在する表領域のロギング属性になります。

LOB の場合に LOGGING 句または NOLOGGING 句が省略されると、次の規則が適用されます。

- LOB は CACHE NOLOGGING になり得ないため、CACHE が指定されている場合は LOGGING が使用されます。
- それ以外の場合、デフォルトは、LOB 値が存在する表領域から取得されます。

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

この項では、ダイレクト・ロード・インサートのための索引のメンテナンス、領域の割当ておよびデータのロックについて説明します。

索引のメンテナンス

ローカル索引またはグローバル索引を持つ非パーティション表またはパーティション表へのダイレクト・ロード・インサートの場合、INSERT 操作の終わりに索引のメンテナンスが行われます。この索引のメンテナンスは、パーティション表または非パーティション表に対するパラレル・ダイレクト・ロード・インサートの場合にはパラレル実行サーバーによって行われ、シリアル・ダイレクト・ロード・インサートの場合には単一のプロセスによって行われます。

ダイレクト・ロード・インサートによって表内のデータのほとんどが変更される場合は、インサートの前に索引を削除し、後で再作成すると、索引のメンテナンスによるパフォーマンスの影響を回避できます。

領域についての考慮事項

ダイレクト・ロード・インサートは、セグメントの空きリスト内の既存の領域を無視するため、従来型パス INSERT より多くの領域が必要です。非パーティション表へのパラレル・ダイレクト・ロード・インサートの場合、表セグメントの高水位標より後にある空きブロックも無視されます。ダイレクト・ロード・インサートを使用する場合は、さらにいくつかの領域要件を考慮する必要があります。

非パーティション表に対してパラレル・ダイレクト・ロード・インサートを実行すると、一時セグメント（並列度ごとに1つのセグメント）が作成されます。たとえば、並列度を4に設定して非パーティション表に対してパラレル INSERT を使用すると、一時セグメントが4つ作成されます。

各パラレル実行サーバーは、最初にデータを一時セグメントに挿入し、最後に、すべての一時セグメントに入れられているデータを表に追加します。（CREATE TABLE... AS SELECT と同じメカニズムです。）

パーティション表に対するパラレル INSERT の場合は、一時セグメントは作成されません。各パラレル実行サーバーは、データを高水位標の後のパーティションに挿入するのみです。

ローカルに管理されておらず、自動モードでない非パーティション表に対してパラレル INSERT を実行し、次のパラメータの値を変更すると、不要になった領域を無駄にせず、一時セグメント用の記憶領域を十分に確保できます。

NEXT	オブジェクトに割り当てられる、そのオブジェクトの次のエクステントのサイズ（単位はバイト）
PCTINCREASE	3 番目以降のエクステントが既存のエクステントに対して増大する比率（単位は %）
MINIMUM EXTENT	表領域内の各使用済エクステントや使用可能エクステントのサイズが、最低でも指定した値以上で、かつその値の倍数になるようにして、表領域内の空き領域の断片化を制御

これらのパラメータには、次のような値を選択してください。

- 各エクステントが小さくなりすぎない程度のサイズ（1MB 以上）。このサイズは、オブジェクト内の合計エクステント数に影響します。
- 各エクステントが、パラレル INSERT によって必要以上に大きいセグメントの領域を無駄にすることのない程度のサイズ。

NEXT および PCTINCREASE パラメータの値を変更するには、ALTER TABLE 文で STORAGE 句を指定します。MINIMUM EXTENT パラメータの値は、ALTER TABLESPACE 文で変更できます。パラレル DML 文を実行した後、NEXT、PCTINCREASE および MINIMUM EXTENT 記憶域パラメータを、非パラレル操作に適した設定に戻すことができます。

ロックについての考慮事項

ダイレクト・ロード・インサートでは、表（またはパーティション表のすべてのパーティション）に対して排他ロックが取得され、その表に対して挿入、更新または削除を同時に実行できないようにします。ただし、問合せを同時に実行することは許されているため、インサートを開始する前の表中のデータのみ見えます。また、このようなロックがあるため、索引の同時作成または同時再作成はできません。このことは、表の同時実行性に影響してくるため、ダイレクト・ロード・インサートを使用する前に考慮しておく必要があります。

関連項目： 23-43 ページの「[パラレル DML のためのリソースのロックとエンキュー](#)」

ダイレクト・ロード・インサートの制限事項

ダイレクト・ロード・インサートの制限事項は、SQL*Loader を使用したダイレクト・パス・パラレル・ロードの場合と同じです。どちらも同じメカニズムが基礎になっているためです。さらに、一般的なパラレル DML の制限事項は、ダイレクト・ロード・インサートにも当てはまります。

シリアル・ダイレクト・ロード・インサートとパラレル・ダイレクト・ロード・インサートには、次の制限事項があります。

- トランザクションでは、複数のダイレクト・ロードの INSERT 文（またはダイレクト・ロードの INSERT 文と、パラレルの UPDATE 文または DELETE 文の両方）を含めることができますが、これらの文の 1 つで表を変更した後では、そのトランザクション内の他の SQL 文で同じ表にアクセスすることはできません。
 - － 同じ表にアクセスする問合せは、ダイレクト・ロードの INSERT 文より前には行うことができますが、後に行うことはできません。
 - － 同じトランザクション内のダイレクト・ロードの INSERT（またはパラレル DML）によって変更済の表にアクセスしようとするシリアル文またはパラレル文は、エラー・メッセージ付きで拒否されます。
- 初期化パラメータが ROW_LOCKING = INTENT の場合、ダイレクト・ロード・パスによる挿入を実行できません。
- ダイレクト・ロード・インサートは、参照整合性をサポートしていません。
- ダイレクト・ロード・インサート操作では、トリガーはサポートされません。
- ダイレクト・ロード・インサートではレプリケーション機能はサポートされません。
- オブジェクト列や LOB 列がある表、または索引構成表に対しては、ダイレクト・ロード・インサートを実行できません。
- ダイレクト・ロード・インサート操作に含まれるトランザクションを分散トランザクションにすることはできません。
- クラスタ化された表はサポートされません。

これらの制限に違反していると、警告やエラー・メッセージは出されずに、その文は従来型の挿入パスを使用してシリアルに実行されます。1つのトランザクションで同じ表に複数回アクセスする文に関する制限は例外で、この場合はエラー・メッセージが出されます。

たとえば、表にトリガーまたは参照整合性がある場合にダイレクト・ロード・インサート（シリアルまたはパラレル）を使用しようとすると、PARALLEL ヒントまたは句（存在する場合）のみでなく APPEND ヒントも無視されます。

関連項目： 23-45 ページの「[パラレル DML の制限事項](#)」

SQL 文の平行実行

この章では、SQL 文の平行実行について説明します。この章の内容は、次のとおりです。

- SQL 文の平行実行の概要
- 平行実行のプロセス・アーキテクチャ
- 並列度の設定
- 平行問合せ
- 平行 DDL
- 平行 DML
- 関数の平行実行
- 親和性
- その他の平行化

注意： この章で説明する平行実行機能は、Oracle8i Enterprise Edition を購入した場合にのみ使用できます。

また、平行実行は Oracle Parallel Server と同じではありません。Parallel Server オプションは、SQL 文を平行実行する際には必要ありません。ただし、平行実行のいくつかの面は、Oracle Parallel Server にのみ適用されます。

SQL 文の паралел実行の概要

Oracle が SQL 文を паралелに実行していない場合、各 SQL 文は単一のプロセスにより順次実行されます。しかし、паралел実行では、複数のプロセスが同時に作業を実行して、単一の SQL 文を実行します。1 つの文を実行するのに必要な作業を複数のプロセスに分けることにより、1 つのプロセスで文を実行する場合に比べて、Oracle はさらに高速で実行できます。

паралел実行により、意思決定支援（DSS）アプリケーションや大規模データベース環境に関連したデータ集中型の操作のパフォーマンスがめざましく改善されます。対称マルチプロセッシング（SMP）、クラスタ化されたシステムおよび超並列システム（MPP）においては、паралел実行を活用して最大のパフォーマンスを引き出せます。1 つの Oracle システムにある複数の CPU に文の処理を分散させることができるためです。

паралел実行によって、システムはハードウェア・リソースの使用が最適化され、パフォーマンスが一段と向上します。システムの CPU とディスク・コントローラの負荷がすでに大きい場合は、パフォーマンスを改善するため、паралел実行を使用する前にシステムの負荷を軽減するか、またはハードウェア・リソースを増やす必要があります。

関連項目： パラメータ・ファイルとデータベースをチューニングして、паралел実行を活用できるようにする方法の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

паралел化できる操作

Oracle Server では、次の操作に паралел実行を利用できます。

- 表スキャン
- ネステッド・ループ・ジョイン
- ソート / マージ結合
- ハッシュ結合
- NOT IN
- GROUP BY
- SELECT DISTINCT
- UNION と UNION ALL
- 集計（操作）
- SQL からコールされる PL/SQL ファンクション

- ORDER BY
- CREATE TABLE AS SELECT
- CREATE INDEX
- 索引の再作成
- 索引パーティションの再作成
- パーティションの移動
- パーティションの分割
- 更新
- 削除
- INSERT ... SELECT
- 制約の使用可能化（表スキンの平行化）
- スター型変換
- キューブ
- ロールアップ

Oracle が操作を平行化する方法

SELECT 文には、問合せのみが含まれています。通常、DML 文または DDL 文には、問合せ部分と DML または DDL 部分が含まれており、それぞれ平行化できます。

注意： 通常、データ操作言語（DML）に問合せも含まれますが、この章の「DML」は INSERT、UPDATE および DELETE のみを指します。

Oracle は基本的に次の方法で SQL 文を平行化します。

1. スキャン操作（DML 文と DDL 文の中の SELECT および副問合せ）のために、ブロック範囲により平行化します。
2. パーティション表とパーティション索引に対する DDL および DML 操作のため、パーティションによって平行化します。
3. 非パーティション表にのみ挿入するため、平行実行サーバーによって平行化します。

ブロック範囲による平行化

Oracle は、実行時に問合せを動的に平行化します。「動的平行化」は、表または索引をデータベース・ブロックのいくつかの範囲（「ROWID 範囲」）に分割し、異なる範囲に対して操作を平行に実行します。データの分散や位置に変更がある場合、SQL 文の問合せ部分を実行するたびに平行化が自動的に最適化されます。

ブロック範囲による平行・スキャンでは、表または索引が ROWID 値の上限と下限で範囲を決められた断片に分けられます。表または索引は、パーティション化されたものもそうでないものも使用できます。

パーティション表とパーティション索引の場合、1 つのパーティションに複数の ROWID 範囲を含めることはできますが、ROWID 範囲がパーティションにまたがることは不可能です。Oracle は、ROWID 範囲を使用してパーティション番号を送信することによって、パーティション・マップ参照を回避します。パーティション化列に対するコンパイル時述語および実行時述語は、ROWID 範囲を関係のあるパーティションに限定し、不要なパーティションがスキャンされないようにします（「パーティション・プルーニング」）。

したがって、表スキャンによってパーティション表にアクセスする平行問合せの作業量は、全体として、非パーティション表に対する同じ問合せの作業量と同じか、それ以下になります。ディスクのアクセス回数の合計はパーティション・プルーニングによって減少しますが、パーティション表に対する問合せは同等に平行化されて実行されます。

表および索引に対する操作のうち、ブロック範囲（ROWID 範囲）によって平行化できるものは、次のとおりです。

- 表スキャンを使用した問合せ（DML および DDL 文の問合せを含む）
- パーティションの移動
- パーティションの分割
- 索引パーティションの再作成
- CREATE INDEX（非パーティション索引）
- CREATE TABLE ... AS SELECT（非パーティション表）

パーティションによる平行化

パーティションは、実行時間の長い操作を、パーティションごとに平行に実行される、より小さい操作に分けるため、表と索引を静的に論理分割したものです。平行化の最小単位はパーティションです。次の場合を除き、パーティション内でさらに平行化することはありません。

- 問合せ。前述のようにブロック範囲によって平行化できます。
- コンポジット・パーティション化。平行化の粒度（最小単位）はサブパーティションです。

パーティション表と索引に対する操作は、表または索引の各パーティションに異なるパラレル実行サーバーを割り当てることによって、パラレルで実行されます。コンパイル時述語および実行時述語は、操作がパーティション化列を参照するときに、パーティションを制限します。コンパイル時述語または実行時述語が、操作を1つのパーティションに制限している場合、この操作はシリアルで実行されます。

パラレル操作には、アクセスされるパーティションの数より少ない数のパラレル実行サーバーが使用されることもあります（リソース制限、ヒントまたは表属性による）、各パーティションは1つのパラレル実行サーバーによってアクセスされます。ただし、1つのパラレル実行サーバーから複数のパーティションへのアクセスは可能です。

パーティション表とパーティション索引に対する操作がパラレルに実行されるのは、複数のパーティションにアクセスする場合のみです。

パーティション表とパーティション索引に対する操作のうち、パーティションによってパラレル化できるものは次のとおりです。

- CREATE INDEX
- CREATE TABLE ...AS SELECT
- UPDATE
- DELETE
- INSERT ... SELECT
- ALTER INDEX ... REBUILD
- パーティション索引でのレンジ・スキャンを使用する問合せ

関連項目： 11-16 ページの「[コンポジット・パーティション化](#)」

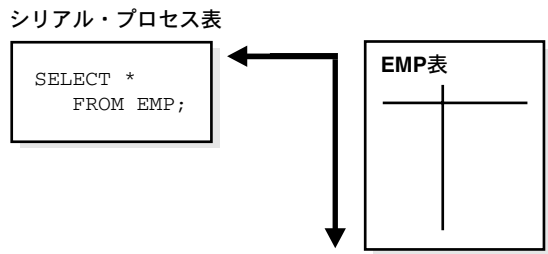
パラレル実行サーバーによるパラレル化

非パーティション表の場合のみ、Oracle は複数のパラレル実行サーバーの間で作業を分けることによって INSERT 操作をパラレル化します。新しい行には ROWID がないため、行は複数のパラレル実行サーバーの間で分配されて、空き領域に挿入されます。

パラレル実行のプロセス・アーキテクチャ

パラレル実行を使用しない場合は、SQL 文の順次実行に必要な処理すべてを1つのサーバー・プロセスが実行します。たとえば、フル・テーブル・スキャン (SELECT * FROM EMP など) を実行するには、[図 23-1](#) に示すように、1つのプロセスで操作全体を実行します。

図 23-1 シリアルなフル・テーブル・スキャン



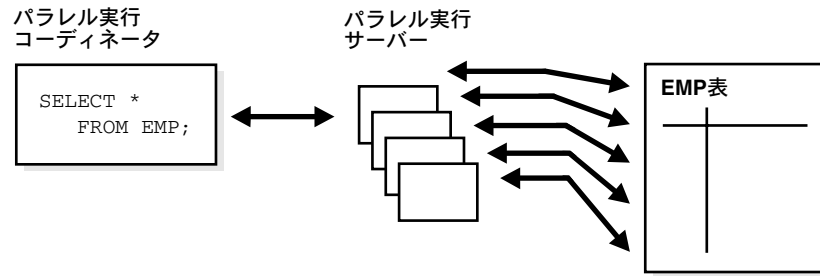
パラレル実行では、複数の「パラレル・プロセス」を使用して操作がパラレルで実行されます。「パラレル実行コーディネータ」という1つのプロセスが、文の実行を複数の「パラレル実行サーバー」に分配し、すべてのサーバー・プロセスからの結果を調整してユーザーに送り返します。

注意：「パラレル実行サーバー」という用語は、Oracle Parallel Server のプロセスを意味するのではなく、操作を他のプロセスとパラレルで実行するプロセスを意味します。Oracle Parallel Server では、パラレル実行サーバーが複数インスタンスにまたがる場合があります。パラレル実行サーバーは「スレーブ・プロセス」とも呼ばれます。

大量パラレル処理（MPP）構成でのパラレル実行のために操作が断片に分割される場合、Oracle はその操作に使用する表または索引の断片に対応するプロセスの「親和性」を考慮することにより、操作の特定の断片をパラレル実行サーバーに割り当てます。パーティション表および索引の物理的なレイアウトは、パラレル実行サーバーに作業を割り当てるのに使用される親和性に影響します。

図 23-2 に、EMP 表がブロック範囲によって動的に分割され（「動的パーティション化」）、その表の部分的なスキャンを複数のパラレル実行サーバーが同時に実行する様子を示します。パラレル実行サーバーは結果をパラレル実行コーディネータに送り返し、それぞれの結果が組み立てられて、最終的なフル・テーブル・スキャンになります。

図 23-2 パラレルのフル・テーブル・スキャン



パラレル実行コーディネータは、実行する機能をパラレルな断片に分割し、その後、パラレル実行サーバーによって処理された各断片（結果）を再び統合します。1つの操作に割り当てられたパラレル実行サーバーの数が、その操作の「並列度」（DOP）となります。同じ SQL 文中の複数の操作の並列度は、すべて同じです。

関連項目：

- 23-49 ページの「親和性」
- 23-16 ページの「[Oracle による操作の並列度の決定方法](#)」

パラレル実行サーバー・プール

インスタンスを起動すると、どのパラレル操作にも使用可能なパラレル実行サーバーのプールが作成されます。初期化パラメータ PARALLEL_MIN_SERVERS は、インスタンスの起動時に作成されるパラレル実行サーバーの数を指定します。

パラレル操作の実行中、パラレル実行コーディネータはプールからパラレル実行サーバーを取得して操作に割り当てます。必要であれば、Oracle はその操作のために追加のパラレル実行サーバーを作成できます。それらのパラレル実行サーバーは、ジョブの実行中は1つの操作に対応付けられたままになり、その後、他の操作に使用できるようになります。文が完全に処理された後、パラレル実行サーバーはプールに戻ります。

注意： パラレル実行コーディネータとパラレル実行サーバーは、一度に 1 つの文にしか作用しません。パラレル実行コーディネータは、たとえばパラレル問合せとパラレル DML 文を同時には調整できません。

ユーザーが SQL 文を発行すると、オプティマイザは操作をパラレルで実行するかどうかを決定し、各操作の並列度を決定します。操作に必要なパラレル実行サーバーの数は、様々な方法で指定できます。

オプティマイザがパラレル処理のために文を処理する場合、次のことが順に行われます。

1. SQL 文のフォアグラウンド・プロセスがパラレル実行コーディネータになります。
2. パラレル実行コーディネータが、サーバー・プールから必要な数のパラレル実行サーバー（並列度によって判断されます）を取得します。
3. Oracle が、一連の操作として文を実行します。可能であれば、各操作はパラレルで実行されます。
4. 文の処理が完了すると、コーディネータは文を発行したユーザー・プロセスに結果のデータを返し、さらにパラレル実行サーバーをサーバー・プールに戻します。

パラレル実行コーディネータは、SQL 文の解析時ではなく実行時に、パラレル実行サーバーをコールします。したがって、マルチスレッド・サーバーでパラレル実行を使用する場合には、ユーザーの文の EXECUTE コールを処理するサーバー・プロセスがその文のパラレル実行コーディネータになります。

関連項目： 23-15 ページの「[並列度の設定](#)」

パラレル実行サーバーの数の変動

1 つのインスタンスによって同時に処理されるパラレル操作の数が大幅に変動する場合、Oracle はプール中のパラレル実行サーバーの数を自動的に変更します。

パラレル操作の数が多くなると、入ってくる要求を処理するために追加のパラレル実行サーバーが作成されます。ただし、初期化パラメータ PARALLEL_MAX_SERVERS で指定されたより多数のパラレル実行サーバーがインスタンス用に作成されることはありません。

パラレル操作の数が減少すると、Oracle はしきい値で指定された時間アイドル状態になっていたパラレル実行サーバーを終了させます。パラレル実行サーバーがアイドル状態になっていた時間がいくら長くても、プール・サイズを PARALLEL_MIN_SERVERS の値より小さくすることはありません。

十分なパラレル実行サーバーがない場合の処理

Oracle は、要求された数より少ないプロセスでパラレル操作を実行できます。

プール中のすべてのパラレル実行サーバーが占有されており、最大数のパラレル実行サーバーがすでに開始されている場合、パラレル実行コーディネータはシリアル処理に切り替わります。

関連項目：

- `PARALLEL_MIN_PERCENT` 初期化パラメータで最小値を指定する方法の詳細は、23-18 ページの「[パラレル実行サーバーの最小数](#)」を参照してください。
- パラレル実行サーバーのインスタンスのプールの監視、および初期化パラメータの適切な値の決定の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

パラレル実行サーバーの通信方法

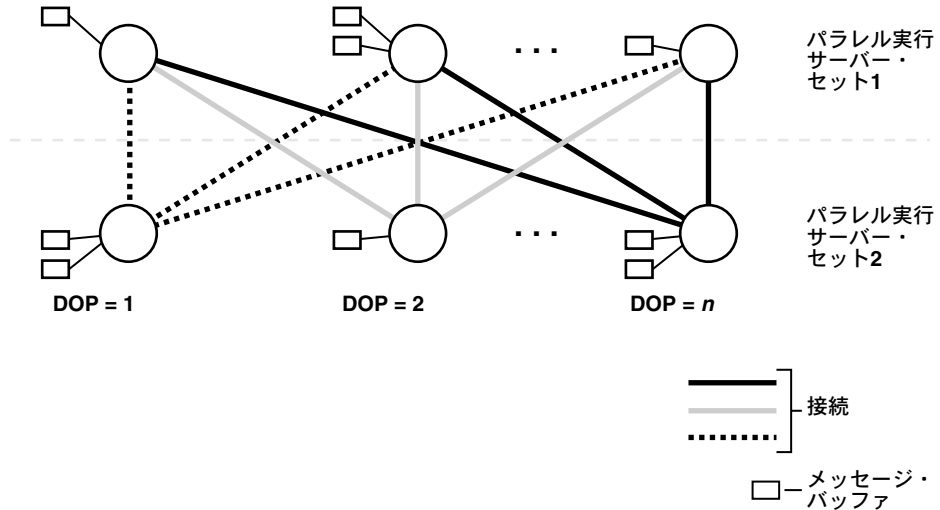
問合せをパラレルで実行するために、通常、Oracle はプロデューサ・キュー・サーバーとコンシューマ・サーバーを作成します。プロデューサ・キュー・サーバーは表から行を取り出し、コンシューマ・サーバーはこれらの行に対して結合、ソート、DML および DDL などの操作を実行します。プロデューサ実行プロセス・セット内の各サーバーは、コンシューマ・セット内の各サーバーに接続します。これは、パラレル実行サーバー間の仮想接続数が、並列度の 2 乗の倍率で増加することを意味します。

各通信チャンネルには、1 ～ 4 個のメモリー・バッファがあります。複数のメモリー・バッファがあれば、パラレル実行サーバー相互の非同期通信が容易になります。

単一インスタンス環境では、通信チャンネルごとに最大 3 個のバッファが使用されます。Oracle Parallel Server 環境では、チャンネルあたり最大 4 個のバッファが使用されます。

図 23-3 に、メッセージ・バッファと、プロデューサ・パラレル実行サーバーがコンシューマ・パラレル実行サーバーに接続する方法を示します。

図 23-3 パラレル実行サーバーの接続とバッファ



同じインスタンス内の 2 つのプロセス間に接続があれば、サーバーはバッファをやりとりして通信します。異なるインスタンス内のプロセス間に接続があれば、外部の高速ネットワーク・プロトコルを使用してメッセージが送信されます。図 23-3 では、並列度はパラレル実行サーバー数（この場合は "n"）と同数です。図 23-3 には、パラレル実行コーディネータは示されていません。各パラレル実行サーバーは、実際にはパラレル実行コーディネータへの追加の接続を持っています。

SQL 文のパラレル化

各 SQL 文ごとに、解析時に最適化プロセスおよびパラレル化プロセスが実行されます。そのため、データが変更されて、さらに最適な実行計画やパラレル化計画が使用可能になると、Oracle はその新しい状況に自動的に対応できます。

オブティマイザが文の実行計画を決定した後で、パラレル実行コーディネータは、実行計画に含まれる各操作のパラレル化の方法を決定します。たとえば、ブロック範囲によるフル・テーブル・スキャンをパラレル化したり、パーティション毎に索引レンジ・スキャンをパラレル化します。コーディネータは、操作をパラレルで実行できるかどうかと、実行する場合のパラレル実行サーバーの数を決定する必要があります。このパラレル実行サーバーの数を、「並列度」と呼びます。

関連項目：

- [第 21 章「オブティマイザ」](#)
- 23-15 ページの「[並列度の設定](#)」
- 23-20 ページの「[SQL 文のパラレル化ルール](#)」

複数のパラレル実行サーバー間での作業の分割

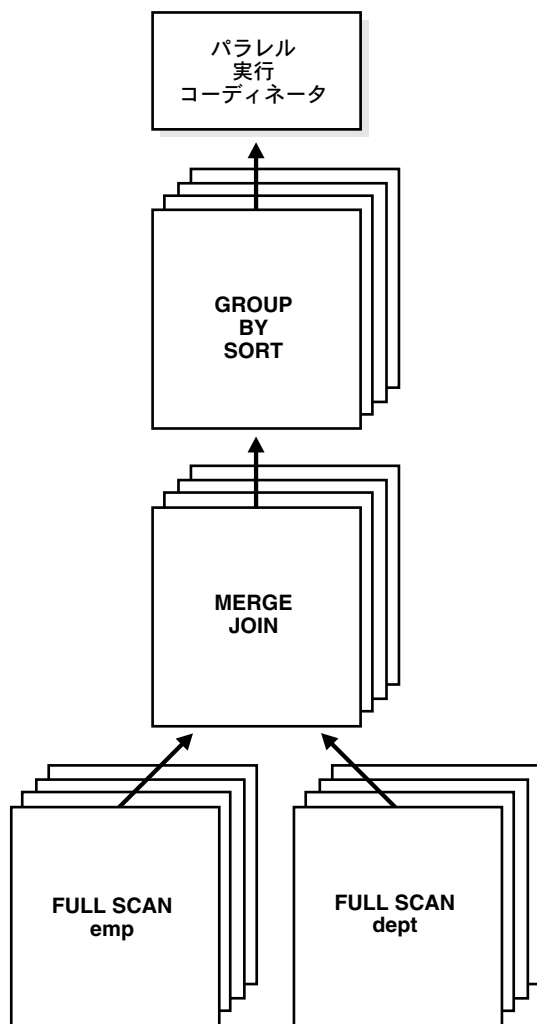
パラレル実行コーディネータは、各操作の再分散化要件を調べます。操作の「再分散化要件」とは、操作によって処理する行を複数のパラレル実行サーバーの間で分配するための方法です。

実行計画の中の各操作の再分散化要件を決定した後、オブティマイザは実行計画の中の各操作を実行する順序を決定します。この情報が決まれば、オブティマイザは文のデータ・フローを決定できます。

[図 23-4](#) に、次の問合せのデータ・フローを示します。

```
SELECT dname, MAX(sal), AVG(sal)
FROM emp, dept
WHERE emp.deptno = dept.deptno
GROUP BY dname;
```

図 23-4 EMP 表と DEPT 表の結合のデータ・フロー図



操作間のパラレル化

他の操作の出力を必要とする操作のことを、「親操作」と呼びます。図 23-4 では、GROUP BY SORT 操作が MERGE JOIN 操作の出力を必要とするため、GROUP BY SORT 操作は MERGE JOIN 操作の親操作です。

子操作によって行が生成されると、親操作はすぐにその行の使用を開始できます。前の例では、パラレル実行サーバーが FULL SCAN DEPT 操作で行を生成している間に、別の一連のパラレル実行サーバーのセットが MERGE JOIN 操作の実行を開始してその行を使用します。

同時に実行される前述の 2 つの操作には、それぞれ専用のパラレル実行サーバーの集合が用意されます。したがって、問合せ操作と、データ・フロー・ツリーそのものの両方にパラレル化が指定されていることになります。個々の操作のパラレル化のことを「イントラオペレーション（操作内）並行性」と呼び、データ・フロー・ツリーの各操作間でのパラレル化のことを「インターオペレーション（操作間）並行性」と呼びます。

Oracle Server の操作にはプロデューサ（作成者）およびコンシューマ（使用者）という特性があるため、特定のツリーに含まれる操作のうち、実行時間を最小にするために同時に実行する必要がある操作は 2 つのみです。

イントラオペレーション並行性とインターオペレーション並行性を理解するために、次の文を考えてみます。

```
SELECT * FROM emp ORDER BY ename;
```

実行計画では、EMP 表の全スキャンと、その後に行う、取り出した行の ENAME 列の値に基づくソート操作が実装されます。この例では、ENAME 列には索引が作成されていないものとします。また、この問合せの並列度は 4 に設定されているとします。つまり、それぞれの操作について 4 つのパラレル実行サーバーをアクティブにできることになります。

図 23-5 に、この例の問合せのパラレル実行を示します。

図 23-5 インターオペレーション並行性と動的パーティション化

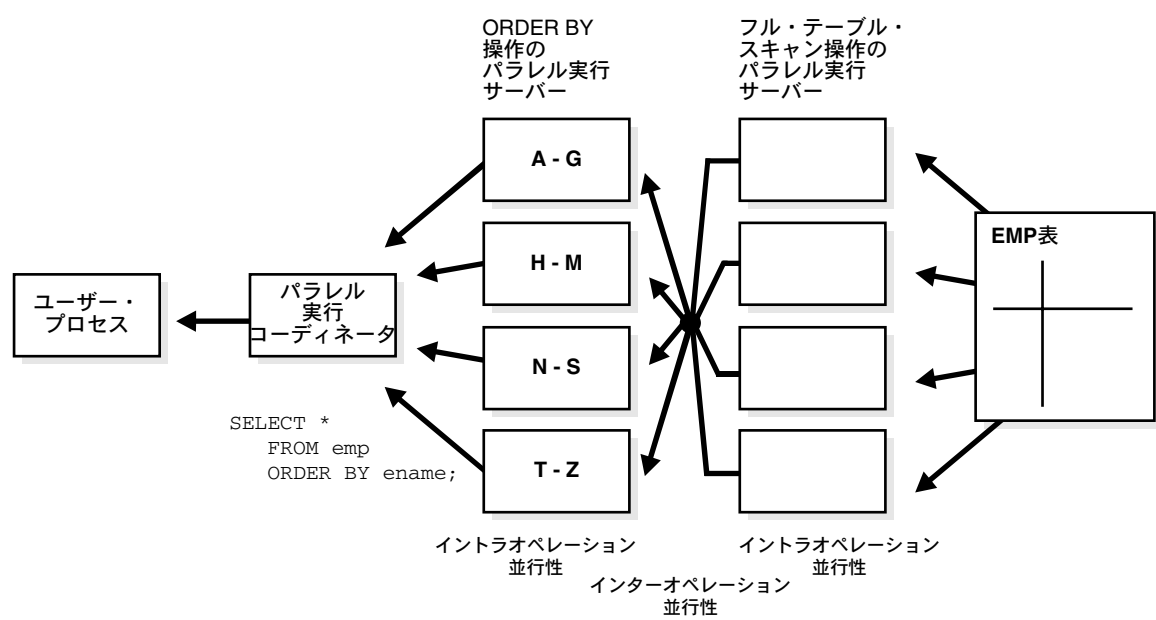


図 23-5 からわかるとおり、並列度は 4 ですが、問合せに実際に関与しているのは 8 つのパラレル実行サーバーです。これは、親オペレータと子オペレータを同時に実行できるためです（インターオペレーション並行性）。

スキャン操作に含まれるすべてのパラレル実行サーバーが、ソート操作を実行するパラレル実行サーバーのうちの適切なプロセスに行を送っていることにも注目してください。パラレル実行サーバーによってスキャンされた行の ENAME 列の内容が A ～ G の間の値であれば、その行は最初の ORDER BY パラレル実行サーバーへ送られます。スキャン操作が完了すると、ソート・プロセスはソート結果をコーディネータへ戻し、コーディネータは問合せ結果の全体をユーザーに戻します。

注意： 一連の並列実行サーバーの操作が完了すると、データ・フローの中で次の順位にある操作に移ります。たとえば、前述の図で、ORDER BY の後にもう 1 つ別の ORDER BY 操作が続いている場合は、表スキャンを実行している並列実行サーバーが、表スキャン完了後に 2 番目の ORDER BY 操作を実行します。

並列度の設定

並列実行コーディネータは、1 つの SQL 文の処理のために、そのインスタンスの並列実行サーバーを 2 つ以上登録することがあります。1 つの操作に関連付けられた並列実行サーバーの数を「並列度」と呼びます。

並列度を指定するには、次の方法があります。

- 文レベルで
 - ヒントを使用
 - PARALLEL 句を使用
- 表の定義を使用して表レベルで
- 索引の定義を使用して索引レベルで
- CPU 数に基づくデフォルトで

次の例は、表に対する並列度を 4 に設定する文を示しています。

```
ALTER TABLE emp PARALLEL 4;
```

次の例では、索引に対する並列度を設定しています。

```
ALTER INDEX iemp PARALLEL;
```

最後の例では、問合せのヒントを 4 に設定しています。

```
SELECT /*+ PARALLEL(emp,4) */ COUNT(*) FROM emp ;
```

この並列度が直接適用されるのは、イントラオペレーション並行性のみです。インターオペレーション並行性が可能であれば、並列実行サーバーの合計数は、指定した並列度の 2 倍になることがあります。同時に実行できる操作は、2 つまでです。

パラレル実行は、複数の CPU とディスクを効率的に使用して問合せに迅速に応答するように設計されています。複数のユーザーが同時にパラレル実行を使用すると、使用可能な CPU、メモリーおよびディスク・リソースは急速に消費されます。Oracle には、次のように、パラレル実行に伴うリソース使用状況を取り扱うために、複数の方法が用意されています。

- マルチユーザー問合せ調整アルゴリズム。システムにかかる負荷が大きくなるにつれて、並列度を低下させます。このオプションをオンにするには、初期化ファイル内で ALTER SYSTEM 文の PARALLEL_ADAPTIVE_MULTI_USER パラメータを使用します。
- ユーザーのリソース制限とプロファイル。ユーザーのセキュリティ・ドメインの一部として、各ユーザーが使用できる各種のシステム・リソースの容量に制限を設定できます。
- データベース・リソース・マネージャ。様々なユーザー・グループにリソースを割当てできます。

関連項目：

- SELECT 文と ALTER 文の構文の詳細は、『Oracle8i リファレンス・マニユアル』および『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- 26-17 ページの「ユーザー・リソースの制限とプロファイル」
- [第 9 章「データベース・リソースの管理」](#)
- ALTER SYSTEM SQL 文の構文は、『Oracle8i SQL リファレンス』を参照してください。

Oracle による操作の並列度の決定方法

パラレル実行コーディネータは、いくつかの指定を考慮して並列度を決定します。その手順は、次のとおりです。

1. SQL 文そのもので指定されたヒントまたは PARALLEL 句をチェックします。
2. 表または索引の定義を調べます。
3. デフォルトの並列度をチェックします。

これらの指定のうちのどれかで並列度が決まると、その並列度がその操作の並列度になります。

ヒント、PARALLEL 句、表または索引定義およびデフォルト値は、特定の操作に関してコーディネータが要求するパラレル実行サーバーの数を決定するのみです。実際に使用されるパラレル実行サーバーの数は、パラレル・サーバー・プール中で使用可能なプロセス数、および操作の同時実行が可能かどうかによって異なります。

関連項目：

- 23-17 ページの「デフォルトの並列度」
- 23-20 ページの「SQL 文のバラレル化ルール」
- 23-7 ページの「バラレル実行サーバー・プール」
- 23-13 ページの「操作間のバラレル化」

ヒント

SQL 文でヒントを指定すると、表または索引に対して並列度を設定したり、操作のキャッシュ動作を設定できます。

- PARALLEL ヒントは、表に対する操作にのみ使用されます。このヒントを使用して、問合せおよび DML 文（INSERT、UPDATE および DELETE）をバラレル化できます。
- PARALLEL_INDEX ヒントは、パーティション索引の索引レンジ・スキャンをバラレル化します。（索引操作では、PARALLEL ヒントは無効であり、無視されます。）

関連項目： SQL 文でのヒントの使用方法の概要と、PARALLEL、NOPARALLEL、PARALLEL_INDEX、CACHE および NOCACHE の各ヒントの明確な構文は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

表定義および索引定義

並列度は、表定義または索引定義の中にも指定できます。表または索引の並列度を設定するには、SQL 文 CREATE TABLE、ALTER TABLE、CREATE INDEX または ALTER INDEX のうちの 1 つを使用します。

関連項目： SQL 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。

デフォルトの並列度

操作のバラレル化を要求しながら、ヒントにも、表または索引の定義にも並列度を指定しなければ、デフォルトの並列度が使用されます。ほとんどのアプリケーションには、デフォルトの並列度が適しています。

SQL 文のデフォルトの並列度は、次の要因によって決定されます。

- システム内のすべての Oracle Parallel Server インスタンス用の CPU 数と、パラメータ PARALLEL_THREADS_PER_CPU の値。
- パーティションによるパラレル化の場合、パーティション・プルーニングに基づく、アクセスされるパーティションの数。
- グローバルな索引メンテナンスでのパラレル DML 操作の場合、更新するすべてのグローバル索引間でのトランザクション空きリストの最小数。パーティション化されたグローバル索引のためのトランザクション空きリストの最小数は、索引パーティションすべてにわたる最小数です。これは、自己デッドロックを防ぐための要件となります。

注意： Oracle は、CPU 情報をオペレーティング・システムから取得します。

前述の要素により、使用されるパラレル実行サーバーのデフォルトの数が決定されます。ただし、実際に使用されるプロセス数は、要求したインスタンス上で実行時にそれらがどの程度使用可能かによって制限されます。1 つのインスタンスに許されるパラレル実行サーバーの合計数の上限は、初期化パラメータ PARALLEL_MAX_SERVERS によって設定されます。

必要なパラレル実行サーバーの最小数（初期化パラメータ PARALLEL_MIN_PERCENT で指定）が使用できなければ、ユーザー・エラーが発生します。その場合、ユーザーは並列度を低くして問合せを再試行できます。

関連項目： 並列度の調整の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

マルチユーザー問合せ調整アルゴリズム

マルチユーザー問合せ調整アルゴリズムが使用可能になっている場合、パラレル実行コーディネータはシステム負荷に応じて並列度を変更します。負荷は、データベース・リソース・マネージャによって計算された割当て済スレッド数によって決定されます。使用可能な CPU 数について、現在の割当て済スレッド数が最適なスレッドを超えていれば、このアルゴリズムは並列度を下げます。これにより、リソースの過剰割当てが回避され、スループットが改善されます。

パラレル実行サーバーの最小数

最低 2 つのパラレル実行サーバーが使用可能であれば、パラレル操作を実行できます。使用可能なパラレル実行サーバーが少なすぎると、SQL 文の実行に予想以上の時間がかかる場合があります。要求されたパラレル実行サーバーのうち、一定の割合以上が使用可能になっていなければ操作を実行しないようにも指定できます。この方針により、最低限受入れ可能なパラレルのパフォーマンスで SQL 文が実行されるようになります。要求されたパラレル・サーバーのうち、使用可能なものの割合が最小割合未満の場合は、SQL 文は実行されず、エラーが戻されます。

要求されたパラレル実行サーバーに必要な最小パーセンテージは、初期化パラメータ `PARALLEL_MIN_PERCENT` に指定します。このパラメータは、問合せのみでなく DML および DDL の操作にも影響します。

たとえば、このパラメータに 50 を指定した場合、操作が正しく実行されるためには、パラレル操作のために要求されたパラレル実行サーバーの少なくとも 50% が使用可能である必要があります。20 個のパラレル実行サーバーが要求されると、少なくとも 10 個のサーバーが使用可能でなければ、ユーザーにエラーが戻されます。`PARALLEL_MIN_PERCENT` を NULL に設定すると、処理のために使用できるパラレル実行サーバーが少なくとも 2 つあれば、すべてのパラレル操作が続行されます。

使用可能なインスタンス数の制限

Oracle Parallel Server においては、インスタンス・グループを使用して、パラレル操作に関与するインスタンスの数を制限できます。それぞれが 1 つ以上のインスタンスからなるインスタンス・グループを必要な数だけ作成できます。その後、パラレル操作の一部またはすべてに対して、どのインスタンス・グループを使用するかを指定できます。パラレル実行サーバーは、指定されたインスタンス・グループのメンバーであるインスタンスにのみ使用されます。

関連項目： インスタンス・グループの詳細は、『Oracle8i Parallel Server 概要』を参照してください。

作業負荷の均衡化

パフォーマンスを最適化するには、すべてのパラレル実行サーバーに同等の作業負荷がかかるようにする必要があります。ブロック範囲またはパラレル実行サーバーによりパラレル化される SQL 文の場合、作業負荷はパラレル実行サーバーの間で動的に分割されます。これにより、一部のパラレル実行サーバーが他のプロセスよりも大量の作業を実行する場合に発生する「作業負荷の不整」を最小限に抑えることができます。

パーティションによりパラレル化される SQL 文の場合、作業負荷が各パーティションに均等に分散していれば、パラレル実行サーバーの数とパーティションの数を一致させるか、またはパーティションの数がプロセスの数の倍数になるように並列度を選択して、パフォーマンスを最適化できます。

たとえば、1つの表に10個のパーティションがあり、パラレル操作によって作業が各パーティションに均等に分配される場合、10個のパラレル実行サーバーを使用する（並列度 = 10）と、1つのプロセスで作業を実行した場合の時間の10分の1の時間で作業を完了できます。5つのプロセスを使用すれば5分の1の時間、2つのプロセスを使用すれば2分の1の時間で作業を完了できます。

ただし、9個のプロセスを使用して10個のパーティションに対する作業を実行すると、1つのパーティションで作業を終了した最初のプロセスが、10番目のパーティションで作業を開始します。その他のプロセスは、作業を終了した後にアイドル状態になります。この場合、各パーティションに均等に作業が分散されていると、パフォーマンスはあまり改善されません。作業の分け方が均等でない場合、最後に残されたパーティションの作業量が他のパーティションと比べて多いか少ないかによってパフォーマンスが異なります。

同じように、4個のプロセスを使用して10個のパーティションに対する作業を実行し、かつその作業が均等に分けられている場合、各プロセスは最初のパーティションの処理を終了した後に2番目のパーティションを処理します。そしてプロセスのうち2つは3番目のパーティションを処理しますが、残りの2つのプロセスはアイドル状態になります。

一般に、P個のパラレル実行サーバーを使用してN個のパーティションに対してパラレル操作を実行する場合の時間は、必ずしもN/Pではありません。一部のプロセスは、他のプロセスが最後のパーティションの作業を終了するまで待機させられる場合もあるためです。ただし、並列度を適切に選択すると、作業負荷の不整を最小限に抑えてパフォーマンスを最適化できます。

関連項目： ディスク親和性がある場合に作業負荷のバランスを調整する方法の詳細は、23-50 ページの「[親和性とパラレル DML](#)」を参照してください。

SQL 文のパラレル化ルール

SQL 文にパラレル・ヒントが含まれていたり、操作の対象となる表または索引が CREATE 文か ALTER 文で PARALLEL と宣言されている場合は、SQL 文をパラレル化できます。さらに、データ定義言語（DDL）文は、PARALLEL 句を使用してパラレル化できます。ただし、これらの方法すべてがどのタイプの SQL 文にも適用されるとは限りません。

パラレル化には、パラレル化するかどうかの決定および並列度という2つの要素が関係しています。これらの要素は、問合せ、DDL 操作および DML 操作について、それぞれ異なる方法で判断されます。

並列度を決定するために、Oracle は「参照オブジェクト」を調べます。

- パラレル問合せは、問合せのうちパラレル化する部分に含まれる各表および索引を調べて、どれが参照表であるかを判断します。並列度が最大の表または索引を選択するのが基本的なルールです。

- パラレル DML（挿入、更新および削除）の場合、並列度を決定する参照オブジェクトは、挿入、更新または削除操作によって変更される表です。パラレル DML では、デッドロックを防ぐため、並列度にある程度の制限も追加されます。パラレル DML 文に副問合せが含まれる場合、その副問合せの並列度は DML 操作と同じです。
- パラレル DDL の場合、並列度を決定する参照オブジェクトは、作成、再作成、分割または移動の対象となる表、索引またはパーティションです。パラレル DDL 文に副問合せが含まれる場合、その副問合せの並列度は DDL 操作と同じです。

問合せのパラレル化のルール

パラレル化の決定 SELECT 文は、次の条件が満たされている場合に限りパラレル化できません。

1. 問合せに PARALLEL ヒント指定（PARALLEL または PARALLEL_INDEX）が含まれている場合、または問合せで参照されているスキーマ・オブジェクトに PARALLEL 宣言が対応付けられている場合。
2. 問合せの中で指定されている表の少なくとも 1 つで、次のどちらかが必要。
 - フル・テーブル・スキャン
 - 複数のパーティションにまたがる索引レンジ・スキャン

並列度 問合せの並列度は、次のルールによって決定されます。

1. 問合せでは、問合せに含まれるすべての表宣言と、問合せを満たす候補となるすべての索引（「参照オブジェクト」）から取得した並列度の最大値を使用します。つまり、最大の並列度を持つ表または索引によって、その問合せの並列度が決まります（「最大問合せディレクティブ」）。
2. 問合せの中に表の PARALLEL ヒント指定が含まれ、かつその表の表指定に PARALLEL 宣言が含まれている場合、ヒント指定のほう PARALLEL 宣言仕様部より優先されます。

UPDATE と DELETE のパラレル化のルール

更新操作と削除操作は、パーティションまたはサブパーティションによってパラレル化されます。パラレル化できるのは、パーティション表に対する更新と削除のみです。1 つのパーティション内で、また非パーティション表上で、更新または削除のパラレル化はできません。

UPDATE 操作と DELETE 操作にパラレル・ディレクティブを指定するには、次の 2 つの方法があります (PARALLEL DML モードが使用可能であるとします)。

1. 更新または削除する表の定義に PARALLEL 句を指定します (参照オブジェクト)。
2. 更新パラレル・ヒントまたは削除パラレル・ヒントを文に指定します。

パラレル・ヒントは、UPDATE 文および DELETE 文の UPDATE または DELETE キーワードのすぐ後に指定します。また、ヒントは、変更する表に対する基礎的なスキャンにも適用されます。

CREATE TABLE 文と ALTER TABLE 文の PARALLEL 句によって、表のパラレル化を指定します。表定義に PARALLEL 句が含まれる場合には、問合せのパラレル化のみでなく DML 文のパラレル化も決定されます。ただし、DML 文に表のための明示的なパラレル・ヒントが含まれていれば、その表の PARALLEL 句による指定はパラレル・ヒントによって上書きされます。

ALTER SESSION FORCE PARALLEL DML 文を使用すると、セッションの後続の UPDATE および DELETE 文の PARALLEL 句を上書きできます。UPDATE 文や DELETE 文に PARALLEL ヒントが含まれている場合、ALTER SESSION FORCE PARALLEL DML 文が上書きされます。

関連項目： サブパーティションが存在する場合の更新および削除操作の詳細は、11-16 ページの「[コンポジット・パーティション化](#)」を参照してください。

パラレル化の決定 UPDATE または DELETE 文で更新または削除操作をパラレル化するかどうかは、次の規則によって決まります。

UPDATE または DELETE 操作は、次の条件の少なくとも 1 つが当てはまる場合にのみパラレル化されます

- 更新または削除される表に PARALLEL 仕様部がある場合。
- DML 文に PARALLEL ヒントが指定されている場合。
- このセッションで、前に ALTER SESSION FORCE PARALLEL DML 文が発行されている場合。

文に副問合せまたは更新可能なビューが含まれている場合、それら独自の PARALLEL ヒントや PARALLEL 句が指定されていることがあります。それらのパラレル化ディレクティブは更新または削除をパラレル化するかどうかの決定には影響を与えません。

表に対する PARALLEL ヒントまたは PARALLEL 句は、問合せ部分と更新または削除部分の両方についてパラレル化するかどうかの決定に使用されます。ただし、更新または削除部分をパラレル化するかどうかは、問合せ部分とは関係なく決定され、その逆も同様です。

並列度 並列度は、問合せと同じ規則によって決められます。更新操作と削除操作の場合、変更の対象となる 1 つの表（唯一の参照オブジェクト）しか関係しません。

更新または削除操作の並列度を決定する優先順位の規則では、更新または削除の PARALLEL ヒント指定が、ターゲット表の PARALLEL 宣言仕様部より優先されます。

更新 / 削除 ヒント > ターゲット表の PARALLEL 宣言仕様部

達成可能な最大の並列度は、表のパーティション数（または、コンポジット・サブパーティションの場合はサブパーティション数）と同じです。1 つの PARALLEL 実行サーバーから複数のパーティションのデータを更新したり削除できますが、各パーティションを更新または削除できるのは 1 つの PARALLEL 実行サーバーからのみです。

並列度がパーティションの数よりも少ない場合、1 つのパーティションに対する作業を終了した最初のプロセスが別のパーティションに対する作業を続け、すべてのパーティションに対する作業が完了するまでそれが繰り返されます。操作に関するパーティションの数より並列度のほうが大きい場合は、超過分の PARALLEL 実行サーバーは何も作業を実行しません。

例 1: `UPDATE tbl_1 SET c1=c1+1 WHERE c1>100;`

TBL_1 がパーティション表で、その表定義に PARALLEL 句が含まれている場合、表のスキャンが順次スキャンである（索引スキャンなど）としても、更新操作はパラレル化されます（この表に C1 が 100 を超えるパーティションが 2 つ以上あるものとします）。

例 2: `UPDATE /*+ PARALLEL(tbl_2,4) */ tbl_2 SET c1=c1+1;`

TBL_2 に対するスキャン操作と更新操作が、並列度 4 でパラレル化されます。

INSERT ... SELECT のパラレル化のルール

INSERT ... SELECT 文では、挿入操作と選択操作がそれぞれ独立してパラレル化されます（並列度は除く）。

PARALLEL ヒントは、INSERT ... SELECT 文の INSERT キーワードの後に指定できます。多くの場合、問合せ先の表は挿入先の表とは異なるため、ヒントでは、特に挿入操作のための PARALLEL・ディレクティブを指定できます。

INSERT... SELECT 文に PARALLEL・ディレクティブを指定するには、次の 4 つの方法があります（PARALLEL DML モードが使用可能になっている場合）。

- その文に SELECT パラレル・ヒント（複数可）を指定します。
- 選択対象の表の定義に PARALLEL 句を指定します。
- その文に INSERT パラレル・ヒントを指定します。
- 挿入先の表の定義に PARALLEL 句を指定します。

ALTER SESSION FORCE PARALLEL DML 文を使用すると、セッションの後続の挿入操作の PARALLEL 句を上書きできます。挿入操作にパラレル・ヒントが含まれている場合、ALTER SESSION FORCE PARALLEL DML 文が上書きされます。

パラレル化の決定 INSERT... SELECT 文で挿入操作をパラレル化するかどうかは、次の規則によって決まります。

INSERT は、次の条件の少なくとも 1 つが当てはまる場合にのみパラレル化されます。

- DML 文で INSERT の後に PARALLEL ヒントが指定されている場合。
- 挿入する表（参照オブジェクト）に PARALLEL 宣言仕様部がある場合。
- このセッションで、前に ALTER SESSION FORCE PARALLEL DML 文が発行されている場合。

このように、挿入操作をパラレル化するかどうかの決定は、選択操作には依存せず、その逆も同様です。

並列度 選択操作または挿入操作（あるいはその両方）をパラレル化することが決まると、次の優先順位ルールを使用して、文全体の「並列度」を決定するためのパラレル・ディレクティブが 1 つ選択されます。

挿入ヒント・ディレクティブ > 挿入先となる表のパラレル宣言仕様部 > 最大問合せディレクティブ

「最大問合せディレクティブ」とは、複数の表と索引のうち並列度が最大の表または索引が、問合せ操作のパラレル化を決定することを意味します。

選択したパラレル・ディレクティブは、選択操作と挿入操作の両方に適用されます。

例：次の例で使用されている並列度は 2 です。これは、INSERT のヒントに指定されている並列度です。

```
INSERT /*+ PARALLEL(tbl_ins,2) */ INTO tbl_ins
  SELECT /*+ PARALLEL(tbl_sel,4) */ * FROM tbl_sel;
```

DDL 文の平行化のルール

平行化の決定 DDL 操作は、構文中に PARALLEL 句（「宣言」）が指定されている場合に平行化できます。CREATE INDEX および ALTER INDEX ...REBUILD または ALTER INDEX ... REBUILD PARTITION の場合、PARALLEL 宣言はデータ・ディクショナリに格納されます。

ALTER SESSION FORCE PARALLEL DDL 文を使用すると、セッション中の後続の DDL 文の PARALLEL 句を上書きできます。

並列度 並列度は、ALTER SESSION FORCE PARALLEL DDL 文で上書きされない限り、PARALLEL 句の仕様部によって決まります。パーティション索引の再作成が平行化されることはありません。

索引の作成、索引の再作成およびパーティションの移動 / 分割の平行化のルール

CREATE INDEX または ALTER INDEX ... REBUILD の平行化 CREATE INDEX および ALTER INDEX ... REBUILD 文は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でのみ平行化できます。

ALTER INDEX ... REBUILD は、非パーティション索引の場合しか平行化できませんが、ALTER INDEX ... REBUILD PARTITION は PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL によって平行化できます。

ALTER INDEX ... REBUILD（非パーティション）、ALTER INDEX ... REBUILD PARTITION および CREATE INDEX のスキャン操作の平行化は、REBUILD または CREATE 操作と同じであり、並列度も同じです。REBUILD または CREATE に並列度が指定されていない場合、デフォルトは CPU の数になります。

MOVE PARTITION または SPLIT PARTITION の平行化 ALTER INDEX ... MOVE PARTITION および ALTER INDEX ... SPLIT PARTITION 文は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でしか平行化できません。これらの文のスキャン操作の平行化は、対応する MOVE または SPLIT 操作と同じです。並列度が指定されていない場合、デフォルトは CPU 数になります。

CREATE TABLE AS SELECT の平行化のルール

CREATE TABLE ... AS SELECT 文には、CREATE 部（DDL）および SELECT 部（問合せ）という 2 つの部分が含まれています。Oracle では、文の中のこれら 2 つの部分を両方とも平行化できます。CREATE 部は、その他の DDL 操作と同じルールに従います。

パラレル化するかどうかの決定（問合せ部） CREATE TABLE ... AS SELECT 文の問合せ部をパラレル化できるのは、次の条件が満たされる場合のみです。

1. 問合せに PARALLEL ヒント指定（PARALLEL または PARALLEL_INDEX）が含まれている場合、その文の CREATE 部に PARALLEL 句が指定されている場合、または問合せで参照されているスキーマ・オブジェクトに PARALLEL 宣言が対応付けられている場合。
2. 問合せの中で指定されている表の少なくとも 1 つで、次のどちらかが必要。
 - フル・テーブル・スキャン
 - 複数のパーティションにまたがる索引レンジ・スキャン

並列度（問合せ部） CREATE TABLE ... AS SELECT 文の問合せ部の並列度は、次のルールの 1 つによって決定されます。

- 問合せ部では、CREATE 部の PARALLEL 句で指定された値を使用します。
- PARALLEL 句が指定されていない場合のデフォルトの並列度は CPU 数です。
- CREATE がシリアルの場合、並列度は問合せによって決定されます。

パラレル化のヒントに指定する値は、すべて無視されます。

関連項目： 23-21 ページの「[問合せのパラレル化のルール](#)」

パラレル化するかどうかの決定（作成部） CREATE TABLE ... AS SELECT の CREATE 操作は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でのみパラレル化できます。

CREATE TABLE ... AS SELECT の CREATE 操作がパラレル化されると、可能であればスキャン操作もパラレル化されます。スキャン操作は、次のような場合にはパラレル化できません。

- SELECT 句に NOPARALLEL ヒントがある場合
- 操作が非パーティション表の索引をスキャンする場合

CREATE 操作がパラレル化されない場合、SELECT 操作に PARALLEL ヒントが指定されているか、選択した表（またはパーティション索引）に PARALLEL 宣言が指定されている場合は、その SELECT 操作をパラレル化できます。

並列度（作成部） CREATE 操作の並列度と SELECT 操作の並列度（パラレル化される場合）は、ALTER SESSION FORCE PARALLEL DDL 文で上書きされない限り、CREATE 文の PARALLEL 句によって指定します。PARALLEL 句に並列度を指定しない場合のデフォルト値は CPU 数です。

パラレル化ルールのまとめ

表 23-1 に、様々なタイプの SQL 文をパラレル化する方法と、パラレル化を指定するときの各方法の優先順位を示します。

- 優先順位（1）の指定は、優先順位（2）と優先順位（3）の指定に優先します。
- 優先順位（2）の指定は、優先順位（3）の指定に優先します。

関連項目： SQL 文の PARALLEL 句とパラレル・ヒントの詳細は、『Oracle8i SQL リファレンス』を参照してください。

表 23-1 パラレル化ルール

パラレル操作	句、ヒント、または基礎となる表 / 索引宣言によるパラレル化 (優先順位： 1、2、3)			
	パラレル・ヒント	パラレル句	ALTER SESSION	パラレル宣言
パラレル問合せ表スキャン（パーティション表または非パーティション表）	(1) PARALLEL		(2) FORCE PARALLEL QUERY	(3) 表の宣言
パラレル問合せ索引レンジ・スキャン（パーティション索引）	(1) PARALLEL_INDEX		(2) FORCE PARALLEL QUERY	(2) 索引の宣言
パラレル UPDATE/DELETE（パーティション表のみ）	(1) PARALLEL		(2) FORCE PARALLEL DML	(3) 更新または削除の対象となる表の宣言
パラレル INSERT... SELECT の INSERT 操作（パーティション表または非パーティション表）	(1) INSERT 部の PARALLEL		(2) FORCE PARALLEL DML	(3) 挿入先の表の宣言
INSERT ... SELECT の SELECT 操作 INSERT がパラレルのとき	INSERT 文からの並列度			
INSERT ... SELECT の SELECT 操作 INSERT がシリアルのとき	(1) PARALLEL			(2) 選択先の表の宣言
パラレル CREATE TABLE ... AS SELECT の CREATE 操作（パーティション表または非パーティション表）	(注意： SELECT 句のヒントは CREATE 操作には影響しません。)	(2)	(1) FORCE PARALLEL DDL	

表 23-1 パラレル化ルール（続き）

パラレル操作	句、ヒント、または基礎となる表 / 索引宣言によるパラレル化 (優先順位: 1、2、3)			
	パラレル・ヒント	パラレル句	ALTER SESSION	パラレル宣言
CREATE TABLE ... AS SELECT の SELECT 操作 CREATE がパラレルのとき	CREATE 文からの並列度			
CREATE TABLE ... AS SELECT の SELECT 操作 CREATE がシリアルのとき	(1) PARALLEL または PARALLEL_ INDEX			(2) 問合せする表ま たはパーティション 索引の宣言
パラレル CREATE INDEX（パー ティション表または非パーティシ ョン表）		(2)	(1) FORCE PARALLEL DDL	
パラレル REBUILD INDEX（非パー ティション索引）		(2)	(1) FORCE PARALLEL DDL	
REBUILD INDEX（パーティション 索引） — パラレル化されない			—	—
パラレル REBUILD INDEX パー ティション		(2)	(1) FORCE PARALLEL DDL	
パーティションのパラレル MOVE/ SPLIT		(2)	(1) FORCE PARALLEL DDL	

パラレル問合せ

問合せと副問合せは、SELECT 文のみでなく、DDL 文と DML 文（INSERT、UPDATE およ
 び DELETE）の問合せ部でパラレル化できます。

ただし、リモート・オブジェクトを参照している DDL 文や DML 文の問合せ部は、パラレ
 ル化できません。問合せ部でリモート・オブジェクトを参照するパラレルの DML 文または
 DDL 文を発行すると、操作は通知なしにシリアルで実行されます。

関連項目：

- Oracle がパラレル化できる問合せ操作のリストは、23-2 ページの「[パラレル化できる操作](#)」を参照してください。
- ROWID の範囲による動的パラレル化については、23-3 ページの「[Oracle が操作をパラレル化する方法](#)」を参照してください。
- パラレル問合せを実行するプロセスについては、23-5 ページの「[パラレル実行のプロセス・アーキテクチャ](#)」を参照してください。
- プロセスによるパラレル問合せの実行方法は、23-10 ページの「[SQL 文のパラレル化](#)」を参照してください。
- 問合せのパラレル化に関する条件と、並列度を決定する要因については、23-21 ページの「[問合せのパラレル化のルール](#)」を参照してください。
- リモート・オブジェクトを参照する問合せの例は、23-47 ページの「[分散トランザクションの制限事項](#)」を参照してください。

索引構成表のパラレル問合せ

索引構成表の場合は、次のパラレル・スキャン方法がサポートされています。

- 非パーティション索引構成表のパラレル高速全スキャン
- パーティション索引構成表のパラレル高速全スキャン
- パーティション索引構成表のパラレル索引レンジ・スキャン

これらのスキャン方法は、オーバーフロー領域を持つ索引構成表と、LOB を含む索引構成表に使用できます。

非パーティション索引構成表

非パーティション索引構成表のパラレル問合せには、パラレル高速全スキャンを使用します。並列度は、次のものによって降順の優先順位で決定されます。

1. PARALLEL ヒント（存在する場合）
2. CREATE TABLE または ALTER TABLE 文で並列度を指定した場合は、表に対応する並列度

作業の割当ては、索引セグメントを適切な数のブロック範囲に分割し、各ブロック範囲を必要に応じてパラレル実行サーバーに割り当てることによって行われます。いずれかの行に対応するオーバーフロー・ブロックには、その行を所有するプロセスのみが必要に応じてアクセスします。

パーティション索引構成表

索引レンジ・スキャンと高速全スキャンの両方をパラレルで実行できます。パラレル高速全スキャンのパラレル化は、非パーティション索引構成表の場合とまったく同じです。パーティション索引構成表に対するパラレル索引レンジ・スキャンの場合、並列度は（パラレル高速全スキャンの場合と同様に）前述の優先順位リストから選択された最小並列度と、索引構成表のパーティション数です。並列度に応じて、各パラレル実行サーバーは1つ以上の（必要に応じて割り当てられる）パーティションを取得します。各パーティションには、主キー索引セグメントおよび対応するオーバーフロー・セグメント（存在する場合）が含まれています。

オブジェクト型のパラレル問合せ

オブジェクト型の表とオブジェクト型の列を含む表に対して、パラレル問合せを実行できます。オブジェクト型のパラレル問合せでは、次のようにオブジェクト型の順次問合せに使用可能な機能をすべてサポートしています。

- オブジェクト型のメソッド
- オブジェクト型の属性アクセス
- オブジェクト型のインスタンスを作成するコンストラクタ
- オブジェクト・ビュー
- オブジェクト型に対する PL/SQL および OCI 問合せ

パラレル問合せには、オブジェクト型のサイズ制限はありません。

オブジェクト型に対するパラレル問合せの使用には、次の制限が適用されます。

- 結合とソート（ORDER BY、GROUP BY または集合演算を使用）を伴う問合せをパラレル化するには、MAP 関数が必要です。MAP 関数がなければ、問合せは自動的にシリアルで実行されます。
- NESTED TABLE のパラレル問合せはサポートされません。表またはパラレル・ヒントにパラレル属性があっても、問合せはシリアルで実行されます。
- オブジェクト型では、パラレル DML とパラレル DDL はサポートされません。DML 文と DDL 文は常にシリアルで実行されます。

これらの条件が1つでも当てはまるために問合せをパラレルで実行できない場合は、問合せ全体がシリアルで実行され、エラー・メッセージは表示されません。

パラレル DDL

この項では、データ定義言語（DDL）文のパラレル化に関する次のトピックを扱います。

- [パラレル化できる DDL 文](#)
- [パラレルの CREATE TABLE ... AS SELECT](#)
- [リカバリ可能性とパラレル DDL](#)
- [パラレル DDL の領域管理](#)

パラレル化できる DDL 文

パーティション化されているかどうかに関係なく、表と索引に対する DDL 文をパラレル化できます。DDL 文でパラレル化できる操作のまとめは、23-27 ページの[表 23-1](#)を参照してください。

非パーティション表および索引のパラレル DDL 文は次のとおりです。

- CREATE INDEX
- CREATE TABLE ...AS SELECT
- ALTER INDEX ... REBUILD

パーティション表とパーティション索引のパラレル DDL 文は次のとおりです。

- CREATE INDEX
- CREATE TABLE ...AS SELECT
- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER TABLE ... COALESCE PARTITION
- ALTER INDEX ...REBUILD PARTITION
- ALTER INDEX ... SPLIT PARTITION— 分割する（グローバル）索引パーティションが使用可能な場合のみ

これらのすべての DDL 操作は、パラレル実行でもシリアル実行でも、非ロギング・モードで実行できます。

索引構成表の CREATE TABLE は、AS SELECT 句を使用してもしなくてもパラレル化できます。

操作ごとに異なるパラレル化が使用されます（23-27 ページの表 23-1 を参照）。パーティション表のパラレル CREATE TABLE AS SELECT およびパーティション索引のパラレル CREATE INDEX は、パーティション数と同じ並列度で実行されます。

パーティション表全体のパラレル分析は複数のユーザー・セッションを使用して構築できるため、ANALYZE {TABLE, INDEX} PARTITION 文でパーティションのパラレル分析表を作成する必要はありません。

パラレル DDL は、オブジェクト列または LOB 列を含む表には使用できません。

関連項目：

- 22-5 ページの「[ロギング・モード](#)」
- パラレル DDL 文の構文と使用方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- LOB の制限事項は、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。

パラレルの CREATE TABLE ... AS SELECT

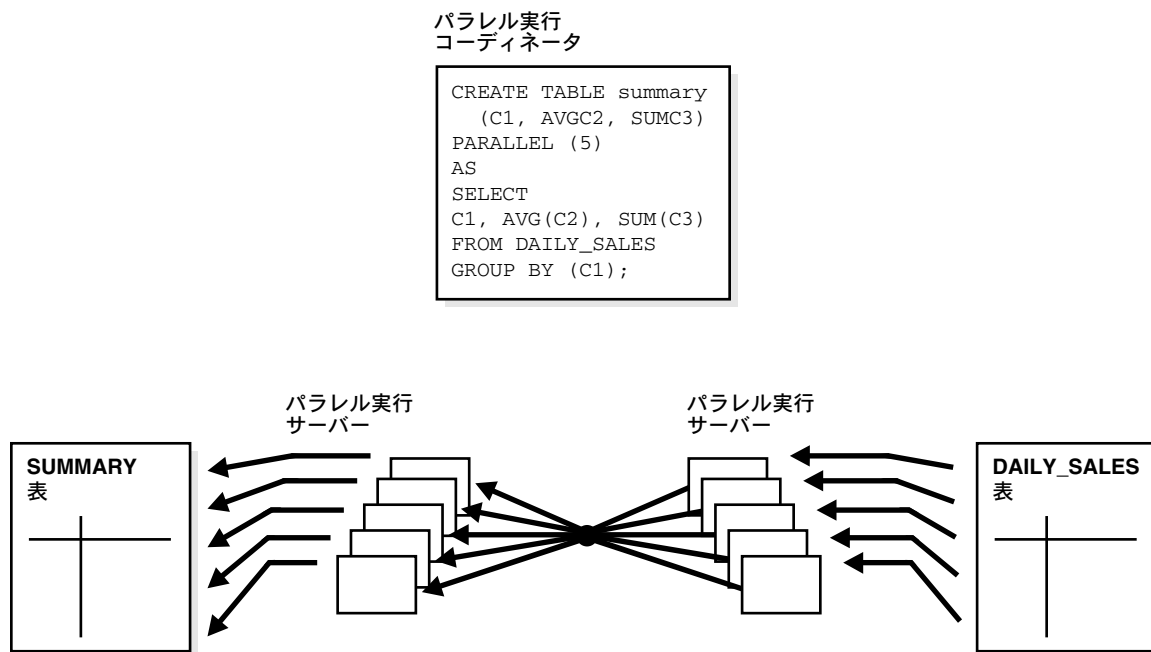
意思決定支援アプリケーションでは、大量のデータを非定型の意思決定支援問合せで使用するために、パフォーマンス上の理由でそれらのデータをより小さな表に「ロールアップ」することが必要な場合があります。このロールアップは、定期的に（毎晩または毎週など）、システムがアクティブでない短い期間に実行されます。

パラレル実行を使用すると、別の表または表の集合からの副問合せとして表を作成するための、問合せ操作と作成操作をパラレル化できます。

図 23-6 に、副問合せからパラレルで表を作成する様子を示します。

注意： クラスタ化表は、パラレルでは作成も挿入もできません。

図 23-6 パラレルでのサマリー表の作成



リカバリ可能性とパラレル DDL

サマリー表のデータを他の表のデータから導出する場合、サマリー表は小さいものであるため、メディア障害からのリカバリ可能性がそれほど重要でない場合があります。サマリー表の作成操作中はリカバリ可能性をオフにすることもできます。

パラレル表の作成（またはその他のパラレル DDL 操作）中のログギングをオフにした場合は、メディア障害によって表が失われないようにするため、表が作成された後で、その表を含む表領域のバックアップを作成してください。

UNDO ログと REDO ログの生成をオフにするには、CREATE/ALTER TABLE/INDEX 文の NOLOGGING 句を使用します。

関連項目：

- 22-5 ページの「[ログギング・モード](#)」
- パラレルで作成した表のリカバリ可能性の詳細は、『Oracle8i 管理者ガイド』を参照してください。

パラレル DDL の領域管理

表または索引のパラレル作成には、パラレル操作中に必要となる記憶領域と、表または索引が作成された後にできる空き領域の両方に影響する、領域管理の問題が関係しています。

CREATE TABLE ... AS SELECT および CREATE INDEX の記憶領域

パラレルで表または索引を作成すると、各パラレル実行サーバーは CREATE 文の STORAGE 句の値を使用して、行を格納するための一時的なセグメントを作成します。したがって、INITIAL として 5MB、PARALLEL DEGREE として 12 を指定して作成した表の場合、各プロセスが 5MB のエクステントで開始されるため、表を作成するには少なくとも 60MB の記憶領域が使用されます。パラレル実行コーディネータがセグメントを結合する時点でセグメントの一部が切り捨てられることがあるため、結果として生成される表は、要求された 60MB より小さいことがあります。

関連項目： CREATE TABLE 文の構文の説明は、『Oracle8i SQL リファレンス』を参照してください。

空き領域とパラレル DDL

パラレルで索引および表を作成すると、各パラレル実行サーバーは新しいエクステントを割り当てて、そのエクステントに表または索引のデータを挿入します。したがって、並列度 3 で索引を作成すると、当初、その索引用に少なくとも 3 つのエクステントが割り当てられます。この説明は、パラレルでの索引の再作成、およびパラレルでのパーティションの移動、分割または再作成にも適用されます。

シリアル操作では、スキーマ・オブジェクトに少なくとも 1 つのエクステントが含まれている必要があります。パラレル CREATE では、表または索引に、少なくともそのスキーマ・オブジェクトを作成するパラレル実行サーバーにあるのと同数のエクステントが含まれている必要があります。

表または索引をパラレルで作成すると、空き領域の飛び地（外部または内部断片化のどちらか）になることがあります。これは、パラレル実行サーバーが使用する一時セグメントが、行を格納するのに必要なサイズより大きい場合に生じます。

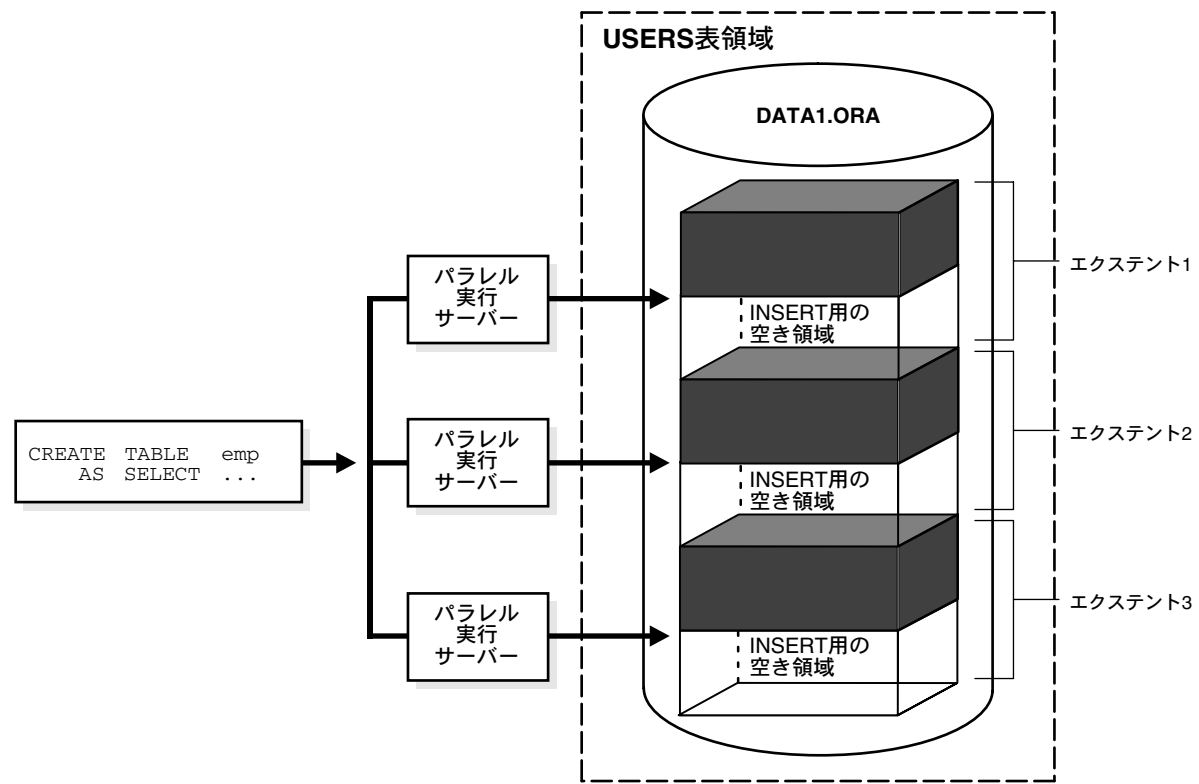
- 各一時セグメントの未使用領域が、表領域レベルで設定された MINIMUM EXTENT パラメータの値より大きい場合、すべての一時セグメントからの行をマージして表または索引にするとときに、未使用領域が切り捨てられます。未使用領域はシステムの空き領域に戻され、新しいエクステントに割り当てることができますが、これは連続した領域ではないため（「外部断片化」）、それらを合わせても 1 つの大きなセグメントにはなりません。
- 各一時セグメントの中の未使用領域が、MINIMUM EXTENT パラメータの値より小さい場合、一時セグメントの中の行をマージして表または索引にするとときに未使用領域を切り捨てることはできません。この未使用領域はシステムの空き領域には戻されず、表または索引の一部になり（「内部断片化」）、追加の領域を必要とする後続の挿入または更新でしか使用できません。

たとえば、CREATE TABLE ... AS SELECT 文で並列度 3 を指定した場合、表領域にデータ・ファイルが 1 つしかないとき、図 23-7 に示すような内部断片化が発生する可能性があります。データ・ファイルの内部表エクステント内にある空き領域の飛び地を他の空き領域と合わせて、エクステントとして割り当てることはできません。

関連項目：

- 空き領域を合わせる方法は、第 3 章「表領域とデータ・ファイル」を参照してください。
- パラレルでの表と索引の作成の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

図 23-7 未使用の空き領域（内部断片化）



パラレル DML

「パラレル DML」（パラレル INSERT、パラレル UPDATE およびパラレル DELETE）では、大規模データベース表や索引に対する大規模な DML 操作の速度を上げたり、規模をさらに大きくするために、パラレル実行メカニズムが使用されます。

注意： 一般にはデータ操作言語（DML）に問合せも含まれますが、この章の「DML」という用語は挿入、更新および削除のみを指します。

ここで扱うパラレル DML に関するトピックは、次のとおりです。

- [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](#)
- [パラレル DML を使用する場合](#)
- [パラレル DML の使用可能化](#)
- [パラレル DML のためのトランザクション・モデル](#)
- [パラレル DML のリカバリ](#)
- [パラレル DML の領域に関する考慮事項](#)
- [パラレル DML のためのリソースのロックとエンキュー](#)
- [パラレル DML の制限事項](#)

関連項目： パラレル INSERT 文の詳細は、[第 22 章「ダイレクト・ロード・インサート」](#)を参照してください。

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

異なるデータ集合に対して複数の DML 文を同時に発行すると、手動で DML 操作をパラレル化できます。たとえば、次のようにして手動でパラレル化できます。

- 複数の空きリスト・ブロックにある空き領域を活用するため、Oracle Parallel Server の複数のインスタンスに対して複数の INSERT 文を発行します。
- キー値の範囲または ROWID の範囲が異なる、複数の UPDATE 文および DELETE 文を発行します。

ただし、手動によるパラレル化には次のような短所もあります。

- 使用しにくいこと。複数のセッションをオープンし（おそらく異なるインスタンス上で）、複数の文を発行する必要があります。
- トランザクション的なプロパティがないこと。DML 文が同時に発行されないため、データに変更が加えられた場合、データベースのスナップショットに一貫性がありません。最小単位で処理するには、様々な文のコミットまたはロールバックを手動により調整する必要があります（おそらくインスタンス間で）。
- 作業分割が複雑であること。作業を正確に分割するには、ROWID の範囲またはキー値の範囲を検索するために表を問い合わせる必要があります。
- 親和性およびリソースの情報が不足していること。Oracle Parallel Server を稼働しているときには、正しいインスタンスに対して正しい DML 文を発行するために親和性の情報が必要になります。また、インスタンス間で作業量のバランスを取るため、カレントのリソースの使用状況を知る必要もあります。

パラレル DML は、INSERT、UPDATE および DELETE をパラレルで自動的に実行することにより、これらの欠点を補います。

パラレル DML を使用する場合

パラレル DML は、主として、大規模なデータベース・オブジェクトに対する規模の大きな DML 操作の速度を向上させるために使用されます。規模が大きいオブジェクトへのアクセスのパフォーマンスや拡張性が重要となっている意思決定支援システム（DSS）環境において、このパラレル DML は有用です。パラレル DML は、DSS データベースに問合せ機能と更新機能の両方を提供するという点で、パラレル問合せを補完するものとなっています。

パラレル化を設定することによるオーバーヘッドのため、パラレル DML 操作は短い OLTP トランザクションには適しません。ただし、OLTP データベース内で実行されるバッチ・ジョブは、パラレル DML 操作によって実行速度を上げることができます。

データ・ウェアハウス・システムの表のリフレッシュ

データ・ウェアハウス・システムの大規模な表は、実働システムの新しいまたは変更されたデータに基づいて定期的に「リフレッシュ」（更新）する必要があります。これは、更新可能な結合ビューと結合されるパラレル DML を使用することによって効果的に実行できます。

リフレッシュが必要なデータは、一般に、リフレッシュ・プロセスが開始する前に一時表にロードされます。この表には、新しい行またはデータ・ウェアハウスの最後のリフレッシュより後に更新された行が入れます。パラレル UPDATE を使用する更新可能な結合ビューを使用することによって、更新された行をリフレッシュしたり、パラレル INSERT を使用する非ハッシュ結合を使用して新しい行をリフレッシュできます。

関連項目：『Oracle8i パフォーマンスのための設計およびチューニング』

中間的なサマリー表

DSS 環境では、多くのアプリケーションで複雑な計算をする必要があり、この計算には多数の大規模な中間的なサマリー表の組立てや処理が含まれます。これらのサマリー表は一時的な表であることが多いため、ログをとる必要がない場合がほとんどです。パラレル DML を使用すると、これらの規模の大きい中間表に対する操作の速度が向上します。利点の 1 つは、中間表に少しずつ結果を入れることができ、パラレル UPDATE を実行できることです。

さらに、サマリー表には累積情報または比較情報が入れられることもあります。そのような情報は複数のアプリケーション・セッション間で持続させる必要があるため、一時表は適切ではありません。パラレル DML 操作により、これらの大規模なサマリー表への変更の速度が向上します。

スコア表

多くの DSS アプリケーションは、一連の基準に基づいて顧客に周期的にスコアを付けます。通常、このスコアは、大規模な DSS 表に格納されます。このスコア情報は、意思を決定するとき（たとえば、メーリング・リストへの追加など）に使用されます。

スコアを付ける作業では、大規模な表の多くの行を問い合せて更新します。パラレル DML を使用すると、これら規模の大きい表に対する操作の速度が向上します。

履歴データの表

履歴データの表には、最近までの特定の期間における会社全体の業務取引が記録されます。DBA は、定期的に表の中の一連の古い行を削除し、一連の新しい行を挿入します。パラレルの INSERT... SELECT 操作とパラレルの DELETE 操作により、この入れ替え作業の速度が向上します。

外部ソースからバルクデータを挿入する場合、パラレル・ダイレクト・ローダー (SQL*Loader) を使用することもできますが、データベース内の別の表にすでに存在しているデータを挿入する場合はパラレル INSERT... SELECT のほうが高速です。

古い行を削除するためにパーティションを削除することもできますが、そのためには表が日付に基づいて適切な期間でパーティション化されている必要があります。

バッチ・ジョブ

終業後に OLTP データベースで実行されるバッチ・ジョブは、一定の時間枠の範囲内で完了させる必要があります。ジョブを時間内に確実に完了させるには、操作をパラレル化するのが適切な方法です。作業量が増加するにつれて、さらに多くのマシン・リソースを追加できます。パラレル操作の拡張性という特性を活用すれば、時間的な制約を満たせます。

パラレル DML の使用可能化

ALTER SESSION 文に ENABLE PARALLEL DML 句を指定して、セッションのパラレル DML を明示的に使用可能にすれば DML 文をパラレル化できます。パラレル DML とシリアル DML ではロック要件、トランザクション要件およびディスク領域要件が異なるため、このモードが必要になります。

セッションのデフォルト・モードは DISABLE PARALLEL DML です。PARALLEL DML がオフの場合、PARALLEL ヒントまたは PARALLEL 句を使用しても、DML はパラレル実行されません。

セッション内で PARALLEL DML を使用可能にすると、そのセッション内のすべての DML 文をパラレル実行するものとみなされます。ただし、PARALLEL DML を使用可能にしても、パラレル・ヒントや PARALLEL 句が指定されていない場合や、パラレル操作の制限事項に違反する場合は、DML 操作がシリアルに実行されることがあります。

セッションを PARALLEL DML モードにしても、SELECT 文、DDL 文および DML 文の間合せ部分のパラレル化には影響がありません。したがって、このモードを設定していない場合、DML 操作はパラレル化されませんが、DML 文中のスキャン操作または結合操作はパラレル化されることがあります。

関連項目：

- 23-42 ページの「[パラレル DML の領域に関する考慮事項](#)」
- 23-43 ページの「[パラレル DML のためのリソースのロックとエンキュー](#)」
- 23-45 ページの「[パラレル DML の制限事項](#)」

PARALLEL DML を使用可能にしたトランザクション

PARALLEL DML を使用可能にしたセッションでは、セッション内のトランザクションが次のような特別なモードになることがあります。つまり、トランザクションの DML 文が表をパラレルで変更する場合は、後続のシリアルまたはパラレルの間合せや DML 文がそのトランザクションで同じ表に再度アクセスすることはできません。したがって、トランザクション中にパラレル変更の結果を見ることはできません。

同じトランザクション内でパラレルで変更された表にアクセスしようとするシリアル文またはパラレル文は、エラー・メッセージ付きで拒否されます。

PARALLEL DML を使用可能にしたセッションで PL/SQL プロシージャまたはブロックが実行される場合、そのプロシージャまたはブロック内の文に、ここで説明したすべてのルールが適用されます。

パラレル DML のためのトランザクション・モデル

DML 操作をパラレルで実行するために、パラレル実行コーディネータはパラレル実行サーバーを取得または生成し、各パラレル実行サーバーは作業の一部分をそれ自身のパラレル・プロセス・トランザクションとして実行します。

- 各パラレル実行サーバーごとに、それぞれ異なるパラレル・プロセス・トランザクションを作成します。
- ロールバック・セグメントでの競合を少なくするため、同じロールバック・セグメントに入れるパラレル・プロセス・トランザクションを少なくします（次の項を参照）。

コーディネータにもそれ自身のコーディネータ・トランザクションがあり、コーディネータ・トランザクションにはそれ自身のロールバック・セグメントがあります。

ロールバック・セグメント

Oracle は、アクティブ・トランザクションが最少のロールバック・セグメントにトランザクションを割り当てます。先送り操作と取消し操作の両方をスピードアップするには、十分なロールバック・セグメントを作成し、それをオンライン化して、1つのロールバック・セグメントに割り当てられるパラレル・プロセス・トランザクションが2個以下になるようにします。

必要に応じて拡張できるように、十分な領域を含む表領域内にロールバック・セグメントを作成し、それらのロールバック・セグメントの MAXEXTENTS 記憶域パラメータを UNLIMITED に設定します。また、パラレル DML トランザクションのコミット後に、ロールバック・セグメントが OPTIMAL サイズに縮小されるように、ロールバック・セグメントの OPTIMAL 値を設定します。

2 フェーズ・コミット

1つのパラレル DML 操作は、2つ以上の独立したパラレル・プロセス・トランザクションによって実行されます。ユーザー・レベルのトランザクションがアトミックであるようにするため、コーディネータは2フェーズ・コミット・プロトコルを使用することにより、パラレル・プロセス・トランザクションによって実行される変更をコミットします。

この2フェーズ・コミット・プロトコルは、簡略化されたバージョンであり、共有ディスク・アーキテクチャを最大限に利用して、特にトランザクションのリカバリ時にトランザクション状態を調査する速度を速めます。Oracle XA ライブラリは必要ありません。インダウト・トランザクションは、ユーザーからは決して見えません。

パラレル DML のリカバリ

パラレル DML 操作をロールバックする時間は、フォワード操作を実行する時間とほぼ同じです。

Oracle は、トランザクション障害とプロセス障害の発生後、およびインスタンス障害とシステム障害の発生後の、パラレル・ロールバックをサポートしています。Oracle では、トランザクション・リカバリのロールフォワード段階とロールバック段階をパラレル化できます。

関連項目： パラレル・ロールバックの詳細は、『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。

ユーザーが発行したロールバックのトランザクション・リカバリ

文のエラーのためのトランザクション障害でユーザーが発行したロールバックは、パラレル実行コーディネータとパラレル実行サーバーによってパラレルで実行されます。そのロールバックにかかる時間は、フォワード・トランザクションの場合とほとんど同じです。

プロセス・リカバリ

パラレル DML コーディネータまたはパラレル実行サーバーの障害からのリカバリは、PMON プロセスによって実行されます。PMON は、次のタスクを実行します。

- 1 つのパラレル実行サーバーで障害が発生すると、PMON はそのプロセスの作業をロールバックし、その他のすべてのパラレル実行サーバーはそれぞれ自分の作業をロールバックします。
- 複数のパラレル実行サーバーで障害が発生すると、PMON はそれらのプロセスの作業すべてをシリアルにロールバックします。
- パラレル実行コーディネータで障害が発生すると、PMON はコーディネータをリカバリし、すべてのパラレル実行サーバーはそれぞれ自分の作業をパラレルでロールバックします。

システム・リカバリ

システム障害からリカバリするには、新しく起動する必要があります。リカバリは、SMON プロセスと SMON によって作成されたリカバリ・サーバー・プロセスによって実行されます。パラレル DML 文は、パラレル・ロールバックを使用してパラレルでリカバリできます。初期化パラメータ COMPATIBLE を 8.1.3 以上に設定していると、「ファースト・スタート・オン・デマンド・ロールバック」により、必要に応じてデッド・トランザクションを 1 度に 1 ブロックずつリカバリできます。

関連項目： 29-15 ページの「[ファースト・スタート・オン・デマンド・ロールバック](#)」

インスタンス・リカバリ (Oracle Parallel Server)

Oracle Parallel Server におけるインスタンス障害からのリカバリは、他の有効なインスタンスのリカバリ・プロセス（つまり、SMON プロセスと、SMON によって作成されたリカバリ・サーバー・プロセス）によって実行されます。有効なインスタンスの各リカバリ・プロセスは、障害が発生したパラレル実行コーディネータまたはインスタンスのパラレル実行サーバー・トランザクション（あるいはその両方）を独立してリカバリします。

パラレル DML の領域に関する考慮事項

パラレル UPDATE では、既存のオブジェクト内の領域が使用されます。この点で、データ用に新しいセグメントを取得するダイレクト・ロード・インサートとは対照的です。

パラレル操作ではオブジェクトを修正する同時実行の子トランザクションが複数あるため、文が順番に実行される場合とパラレル環境とでは領域使用の特性が異なる場合があります。

関連項目： ダイレクト・ロード・インサートの領域の詳細は、22-9 ページの「[領域についての考慮事項](#)」を参照してください。

パラレル DML のためのリソースのロックとエンキュー

パラレル DML 操作の場合のリソースのロックとエンキューについての要件は、シリアル DML の要件とはかなり異なります。パラレル DML は多数のロックを保持するため、ENQUEUE_RESOURCES パラメータと DML_LOCKS パラメータの値を大きくする必要があります。

パラレル UPDATE、DELETE または INSERT 文のプロセスで取得するロックは、次のとおりです。

- パラレル実行コーディネータが取得するロック
 - 表ロック SX を 1 つ
 - パーティションまたはサブパーティションごとにパーティション・ロック X を 1 つ

パーティション表へのパラレル INSERT の場合、コーディネータはすべてのパーティションについてパーティション・ロックを取得します。パラレル UPDATE または DELETE の場合、関係するパーティションが WHERE 句で制限されていない限り、コーディネータはすべてのパーティションについてパーティション・ロックを取得します。
- 各パラレル実行サーバーが取得するロック
 - 表ロック SX を 1 つ
 - パーティションまたはサブパーティションごとにパーティション・ロック NULL を 1 つ
 - パーティションまたはサブパーティションごとにパーティション待機ロック X を 1 つ

1 つのパラレル実行サーバーは 1 つ以上のパーティションに対して処理を実行できますが、1 つのパーティションは 1 つのパラレル実行サーバーによってしか処理されません。

たとえば、600 個のパーティションがある表を並列度 100 で処理している場合、パラレル DML 文で必要なロックは、次のとおりです（その文にすべてのパーティションが関係する場合）。

- コーディネータは、表ロック SX を 1 つとパーティション・ロック X を 600 個取得します。
- パラレル実行サーバー全体として、表ロック SX を 100 個、パーティション・ロック NULL を 600 個、パーティション待機ロック X を 600 個取得します。

「ROW 移行パラレルの更新」と呼ばれる特殊なタイプのパラレル UPDATE が存在します。このパラレル更新方法を使用するのは、表移動句を使用可能にして表が定義されており、行を別のパーティションまたはサブパーティションに移動できる場合のみです。

表 23-2 に、異なるタイプのパラレル DML 文ごとに、コーディネータとパラレル実行サーバーが取得するロックのタイプをまとめます。

表 23-2 パラレル DML 文が取得するロック

文のタイプ	パラレル実行コーディネータが取得するロック	各パラレル実行サーバーが取得するロック
パーティション表のパラレル UPDATE または DELETE。 WHERE 句はパーティションまたはサブパーティションのサブセットにブルーニングされます。	表ロック SX を 1 つ ブルーニングされた（サブ）パーティションごとにパーティション・ロック X を 1 つ	表ロック SX を 1 つ パラレル実行サーバーが所有する、ブルーニングされた（サブ）パーティションごとにパーティション・ロック NULL を 1 つ パラレル実行サーバーが所有する、ブルーニングされた（サブ）パーティションごとにパーティション待機ロック S を 1 つ
パーティション表のパラレル移行 UPDATE。WHERE 句は（サブ）パーティションのサブセットにブルーニングされます。	表ロック SX を 1 つ ブルーニングされた（サブ）パーティションごとにパーティション・ロック X を 1 つ 他のすべての（サブ）パーティション用にパーティション・ロック SX を 1 つ	表ロック SX を 1 つ パラレル実行サーバーが所有する、ブルーニングされた（サブ）パーティションごとにパーティション・ロック NULL を 1 つ パラレル実行サーバーが所有する、ブルーニングされたパーティションごとにパーティション待機ロック S を 1 つ 他のすべての（サブ）パーティション用にパーティション・ロック SX を 1 つ
パーティション表のパラレル UPDATE、DELETE または INSERT	表ロック SX を 1 つ すべての（サブ）パーティション用にパーティション・ロック X を 1 つ	表ロック SX を 1 つ パラレル実行サーバーが所有する（サブ）パーティションごとにパーティション・ロック NULL を 1 つ パラレル実行サーバーが所有する（サブ）パーティションごとにパーティション待機ロック S を 1 つ
非パーティション表へのパラレル INSERT	表ロック X を 1 つ	なし

パラレル DML の制限事項

パラレル DML（ダイレクト・ロード・インサートを含む）には次の制限が適用されます。

- 非パーティション表に対する更新操作と削除操作はパラレル化されません。
- トランザクションは、それぞれ異なる表を変更する複数のパラレル DML 文を含むことができますが、パラレル DML 文が表を変更した後では、後続のシリアルまたはパラレル文（DML または問合せ）がそのトランザクション内で同じ表に再度アクセスすることはできません。
 - － この制限は、シリアル・ダイレクト・ロード INSERT 文の後も存続します。後続の SQL 文（DML または問合せ）は、そのトランザクション中では変更された表にアクセスできません。
 - － 同じ表をアクセスする問合せは、パラレル DML またはダイレクト・ロード INSERT 文より前には行うことができますが、後に行うことはできません。
 - － 同じトランザクション内でパラレル UPDATE またはパラレル DELETE、ダイレクト・ロード INSERT によって変更された表をアクセスしようとするシリアル文またはパラレル文は、エラー・メッセージ付きで拒否されます。
- 初期化パラメータが ROW_LOCKING = INTENT である場合、INSERT、UPDATE および DELETE はパラレル化されません（シリアル化が可能なモードとは無関係）。
- パラレル DML 操作ではトリガーはサポートされません。
- パラレル DML ではレプリケーション機能はサポートされません。
- 特定の制約（自己参照型の整合性、削除カスケードおよび遅延整合性）があると、パラレル DML は実行されません。さらに、ダイレクト・ロード・インサートでは、参照整合性はサポートされません。
- オブジェクト列や LOB 列がある表、または索引構成表に対しては、パラレル DML を実行できません。
- パラレル DML 操作に関与するトランザクションは、分散トランザクションであったり、分散トランザクションにすることはできません。
- クラスタ化された表はサポートされません。

これらの制限に違反していると、警告やエラー・メッセージは出されずに、その文はシリアルで実行されます（1 つのトランザクションで同じ表を複数回アクセスする文に関する制限は例外で、この場合はエラー・メッセージが出ます）。たとえば、非パーティション表にある UPDATE はシリアル化されます。

この後の項では、制限事項についてさらに詳しく説明します。

関連項目： LOB の制限事項の詳細は、『Oracle®i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。

パーティション化キーについての制限事項

パーティション表のパーティション化キーを新しい値に更新できるのは、行移動句を使用可能にして表が定義されていない限り、更新を実行しても行が新しいパーティションへ移動しない場合のみです。

ファンクションの制限事項

パラレル DML に関するファンクションの制限事項は、パラレル DDL およびパラレル問合せの場合と同じです。

関連項目： 23-47 ページの「関数のパラレル実行」

データの整合性についての制限事項

ここでは、整合性制約とパラレル DML 文の相互関係について説明します。

NOT NULL と CHECK このタイプの整合性制約は許可されています。これらは列および行レベルで規定されるものであるため、パラレル DML にとって問題とはなりません。

UNIQUE と PRIMARY KEY このタイプの整合性制約は許可されています。

FOREIGN KEY（参照整合性） ある表に対する DML 操作によって別の表で再帰的 DML 操作が実行される場合、または、整合性検査を実行するために、修正対象のオブジェクトに加えられたすべての変更を同時に見る必要がある場合には、参照整合性についての制限があります。

表 23-3 に、参照整合性制約に関係する表に対して実行できるすべての操作を示します。

表 23-3 参照整合性制約

DML 文	親に対して発行	子に対して発行	自己参照型
INSERT	(適用されない)	パラレル化されない	パラレル化されない
UPDATE No Action	サポートされる	サポートされる	パラレル化されない
DELETE No Action	サポートされる	サポートされる	パラレル化されない
DELETE Cascade	パラレル化されない	(適用されない)	パラレル化されない

DELETE Cascade パラレル実行サーバーは複数のパーティション（親表および子表）から行を削除しようとするため、DELETE カスケードを使用する外部キーを持つ表に対する削除はパラレル化されません。

自己参照整合性 参照キー（主キー）が関係する場合、自己参照整合性制約がある表に対する DML はパラレル化されません。他のすべての列に対する DML は、パラレル化できます。

遅延可能整合性制約 操作対象の表に遅延可能制約がある場合、DML 操作はパラレル化されません。

トリガーの制限事項

DML 操作の影響を受ける表に、その DML 文によって起動される可能性のある有効なトリガーがある場合、その DML 操作はパラレル化されません。これは、レプリケートされる表に対する DML 文はパラレル化されないということにもなります。

そのような表に対する DML をパラレル化するには、関連するトリガーを使用禁止にする必要があります。トリガーを使用可能または使用禁止にすると、依存する共有カーソルは無効になることに注意してください。

分散トランザクションの制限事項

DML 操作が分散トランザクション内にある場合、または DML 操作または問合せ操作の対象がリモート・オブジェクトである場合、その DML 操作はパラレル化できません。

例 1: リモート・オブジェクトを問い合わせる DML 文

```
INSERT /* APPEND PARALLEL (t3,2) */ INTO t3 SELECT * FROM t4@dblink;
```

この問合せ操作はリモート・オブジェクトを参照しているため、シリアルで実行され、通知はありません。

例 2: リモート・オブジェクトに対する DML 操作

```
DELETE /*+ PARALLEL (t1, 2) */ FROM t1@dblink;
```

この DELETE 操作はリモート・オブジェクトを参照しているため、パラレル化されません。

例 3: 分散トランザクション

```
SELECT * FROM t1@dblink;  
DELETE /*+ PARALLEL (t2,2) */ FROM t2;  
COMMIT;
```

この DELETE 操作は、分散トランザクション（SELECT 文で開始）で発生しているため、パラレル化されません。

関数のパラレル実行

PL/SQL や Java で記述されたか、または C で外部プロシージャとして記述されたユーザー記述関数の実行は、パラレル化できます。関数に使用される PL/SQL のパッケージ変数または Java の静的属性全体を、個々のパラレル実行プロセスに対してプライベートにすることができます。ただし、これらは、元のセッションからコピーされるのではなく、各パラレル

実行プロセスの開始時に新しく初期化されます。このため、パラレルで実行した場合に、すべての関数で正しい結果が生成されるとは限りません。

ユーザー記述関数をパラレルで実行できるようにするには、その関数を CREATE FUNCTION または CREATE PACKAGE 文で宣言するときに、PARALLEL_ENABLE キーワードを使用します。

パラレル問合せにおける関数

SELECT 文や、DML または DDL 文の副問合せでユーザー記述関数をパラレルで実行できるのは、PARALLEL_ENABLE キーワードで宣言されている場合、パッケージまたは型で宣言されており、すべての WNDS、RNPS および WNPS を示す PRAGMA RESTRICT_REFERENCES がある場合、または、CREATE FUNCTION で宣言されており、システムが PL/SQL コードの本体を分析し、そのコードがデータベースへの書込みも、パッケージ変数の読み込みや変更もしないことを判断できる場合です。

問合せまたは副問合せの他の部分は、その関数はそのままシリアルで実行する必要がある場合にも、パラレルで実行できる場合があります。

関連項目：

- プラグマ RESTRICT_REFERENCES 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。
- CREATE FUNCTION の詳細は、『Oracle8i SQL リファレンス』を参照してください。

パラレルの DML 文と DDL 文の関数

パラレルの DML 文または DDL 文で、パラレル問合せのようにユーザー記述関数をパラレルで実行できるのは、PARALLEL_ENABLE キーワードで宣言されている場合、パッケージまたは型で宣言されており、すべての RNDS、WNDS、RNPS および WNPS を示す PRAGMA RESTRICT_REFERENCES がある場合、または、CREATE FUNCTION で宣言されており、システムが PL/SQL コードの本体を分析し、そのコードがデータベースの読書きも、パッケージ変数の読み込みや変更もしないことを判断できる場合です。

パラレル DML 文の場合は、パラレルで実行できないファンクション・コールがあると、DML 文全体がシリアルで実行されます。

INSERT ... SELECT または CREATE TABLE ... AS SELECT 文の場合、問合せ部のファンクション・コールは、前述のパラレル問合せルールに従ってパラレル化されます。問合せは、文の残りの部分をシリアルに実行する必要がある場合にもパラレル化でき、その逆も可能です。

親和性

注意： この項で説明している機能は、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。

共有ディスク・クラスタまたは大量パラレル処理（MPP）構成では、Oracle Parallel Server のインスタンスを実行している 1 つまたは複数のプロセッサが直接にデバイスにアクセスする場合、そのインスタンスはそのデバイスに対して「親和性」があると呼びます。同様に、ファイルが格納されているデバイスに対して「親和性」のあるインスタンスには、そのファイルに対する親和性があります。

親和性の決定には、複数のデバイスにわたってストライプ化されているファイルについての任意の決定が含まれることがあります。さらに言えば、インスタンスが表領域内の最初のファイルに対して親和性を持っているなら、そのインスタンスはその表領域（またはその表領域内の表あるいは索引のパーティション）に対して親和性があるといえます。

Oracle では、パラレル実行サーバーに作業を割り当てるときに親和性が考慮されます。SQL 文のパラレル実行に親和性が使用されることは、ユーザーからはわかりません。

親和性とパラレル問合せ

パラレル問合せで親和性の機能を使用すると、データに近いプロセッサでスキャンを実行するため、ディスクからデータをスキャンするときの速度が速くなります。これにより、本来は共有ディスクをサポートしていないマシンにおいても、実質的にパフォーマンスが向上します。

親和性の最も一般的な使用法は、表パーティションまたは索引パーティションを 1 つのデバイス上の 1 つのファイルに格納することです。この構成を使用すると、デバイス障害による損傷を限定することによる高い可用性が約束され、パーティションによるパラレルの索引スキャンを最大限に活用できます。

DSS の使用者は、表パーティションを複数のデバイス（多くの場合、デバイスの合計数のサブセット）にわたってストライプ化することをお勧めします。このようにすると、一部の問合せにおいて、パーティション化基準を使用してアクセスするデータの合計量をプルーニングすることができ、しかも ROWID 範囲によるパラレル表（パーティション）スキャンによってパラレル化も実現できます。デバイスが RAID として構成される場合は、可用性もよくなります。DSS 用に使用する場合であっても、索引は個々のデバイス上でパーティション化します。

その他の構成（たとえば、複数のデバイスにまたがってストライプ化されたファイル内に複数のパーティションがある構成）を使用しても正しい問合せ結果が得られますが、正しい並列度を選択するにはヒントを使用するかオブジェクト属性を明示的に設定する必要があります。

親和性とパラレル DML

パラレル DML（INSERT、UPDATE および DELETE）の場合、親和性の拡張機能により、DML 操作をそのパーティションに対する親和性があるノードにルーティングできるため、キャッシュのパフォーマンスが向上します。

DML 操作をパラレルで実行するために、一連のインスタンスまたはパラレル実行サーバー（あるいはその両方）の間で作業を分配する方法は、親和性によって決められます。親和性を利用すると、次のように問合せのパフォーマンスが改善されます。

- 特定の MPP アーキテクチャでは、Oracle は、パラレル実行サーバーをどのノードで生成するか（パラレル・プロセスの割当て）、そして特定のノードにどの程度の作業グラニュル（ROWID の範囲またはパーティション）を送るか（作業の割当て）を決定するのに、デバイスとノード間の親和性情報を使用します。パフォーマンスを改善するには、各ノードが主としてローカル・デバイスをアクセスするようにし、各ノードのバッファ・キャッシュ・ヒット率を高め、ネットワークのオーバーヘッドと I/O 待ち時間を少なくします。
- SMP 共有ディスク・クラスタの場合、Oracle は、ラウンドロビン・メカニズムを使用してデバイスをノードに割り当てます。項目 1 と同じように、パラレル・プロセスの割当てや作業の割当てを決定するときに、このデバイスとノード間の親和性が使用されます。
- SMP およびクラスタ、MPP の各アーキテクチャでは、プロセス - デバイス間の親和性を使用してデバイスの分離が実現されます。これにより、複数のパラレル実行サーバーが同じデバイスに同時にアクセスする機会が少なくなります。このプロセス - デバイス間の親和性情報は、プロセス間でのスチール処理を実装するのにも使用されます。

パーティション表とパーティション表索引では、パーティションとノード間の親和性情報により、プロセスの割当てと作業の割当てが決定されます。非共有 MPP システムの場合、Oracle Parallel Server はパーティションをインスタンスに割り当て際に、パーティションのディスクへの親和性を考慮に入れます。共有ディスク MPP とクラスタ・システムの場合、パーティションはラウンドロビン方式でインスタンスに割り当てられます。

パラレル DML で親和性を利用できるのは、Oracle Parallel Server 構成で実行する場合のみです。複数の文にわたって持続する親和性情報は、バッファ・キャッシュ・ヒット率を改善し、インスタンス間でのブロックの ping を少なくします。

関連項目：『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

その他のパラレル化

Oracle では、パラレル SQL 実行以外に、次のタイプの操作にもパラレル化を使用できます。

- パラレル・リカバリ
- パラレル伝播（レプリケーション）
- パラレル・ロード（SQL*Loader ユーティリティ）

パラレル・リカバリとパラレル伝播は、パラレル SQL と同じように、パラレル実行コーディネータと複数のパラレル実行サーバーによって実行されます。ただし、パラレル・ロードは、異なるメカニズムを使用します。

パラレル実行コーディネータとパラレル実行サーバーの動作は、どんな種類の操作（SQL、リカバリまたは伝播）を実行するかによって異なる場合があります。たとえば、プール中のすべてのパラレル実行サーバーが使用中で、すでに最大数のパラレル実行サーバーが開始されている場合の動作は、次のとおりです。

- パラレル SQL ロールでは、パラレル実行コーディネータはシリアル処理に切り替えます。
- パラレル伝播ロールでは、パラレル実行コーディネータはエラーを戻します。

特定のセッションごとに、パラレル実行コーディネータは 1 種類の操作のみを調整します。パラレル実行コーディネータは、たとえばパラレル SQL とパラレル伝播またはパラレル・リカバリを同時には調整できません。

関連項目：

- パラレル・ロードの詳細と SQL*Loader の概要は、『Oracle8i ユーティリティ・ガイド』を参照してください。
- パラレル・ロードを使用する場合のアドバイスについては、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。
- 29-11 ページの「[リカバリのパラレル実行](#)」
- パラレル・リカバリの詳細は、『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。
- パラレル伝播の詳細は、『Oracle8i レプリケーション・ガイド』参照してください。

第 VIII 部

データの保護

第 VIII 部では、Oracle のデータベース内のデータを保護する方法、およびデータベース管理者がデータ保護をさらに強化するために行えることについて説明します。

第 VIII 部に含まれる章は、次のとおりです。

- [第 24 章「データの同時実行性と一貫性」](#)
- [第 25 章「データの整合性」](#)
- [第 26 章「データベース・アクセスの制御」](#)
- [第 27 章「権限、ロールおよびセキュリティ・ルール」](#)
- [第 28 章「監査」](#)
- [第 29 章「データベースのリカバリ」](#)

データの同時実行性と一貫性

この章では、マルチユーザー・データベース環境で Oracle がどのようにデータの一貫性を維持するかについて説明します。この章の内容は、次のとおりです。

- 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の概要
- データの同時実行性と一貫性の管理方法
- データをロックする方法

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の概要

シングル・ユーザーのデータベースでは、他のユーザーが同時に同じデータを変更するということがないため、ユーザーは何も心配せずにデータベース内のデータを変更できます。ただし、マルチユーザー・データベースでは、複数のトランザクション内の文によって、同じデータが同時に更新される可能性があります。同時に実行される複数のトランザクションで、意味のある一貫した結果が得られなければならないことが必要です。したがって、マルチユーザー・データベースではデータの同時実行性と一貫性の制御が重要になります。

- 「データ同時実行性」とは、多数のユーザーが同時にデータにアクセスできることを意味します。
- 「データ一貫性」は、各ユーザーにデータの一貫したビューが表示され、その中にユーザー自身のトランザクションや他のユーザーのトランザクションによる参照可能な変更も含まれることを意味します。

複数のトランザクションが同時に実行される場合のトランザクションの首尾一貫した動作を記述するために、データベース研究者は「シリアル化の可能性」と呼ばれるトランザクションの分離モデルを定義してきました。トランザクション動作のシリアル化が可能なモードでは、複数のトランザクションは、同時にではなく1つずつ（シリアルに）実行されるように見えます。

一般に、この程度までのトランザクションの分離が望ましいとされますが、このモードで多数のアプリケーションを実行すると、アプリケーションのスループットがかなり低下する可能性があります。同時に実行される各トランザクションの完全な分離とは、あるトランザクションが問合せを実行している表に対して、他のトランザクションが挿入を実行できない状態を指します。つまり、現実には、トランザクションの完全な分離とパフォーマンスとの間の妥協点を考慮する必要があります。

Oracle には2つの分離レベルがあり、これによってアプリケーションの開発者は、一貫性を保ちながら高いパフォーマンスを実現する各操作モードを使用できます。

関連項目： データベースに対応付けられたビジネス・ルールを規定する
データ整合性の詳細は、[第25章「データの整合性」](#)を参照してください。

回避可能な現象とトランザクション分離レベル

ANSI/ISO SQL 規格 (SQL92) ではトランザクションの4つの分離レベルが定義されており、それぞれトランザクション処理のスループットに与える影響の度合いが異なります。これらの分離レベルは、複数のトランザクションを同時に実行する場合に防ぐ必要がある3つの現象という観点から定義されています。

3つの防止可能な現象とは、次のものです。

内容を保証しない読み込み	まだコミットされていないトランザクションが書き込んだデータを、別のトランザクションが読み込む現象
非リピータブル・リード (ファジー読み込み)	あるトランザクションが以前に読み込んだデータをもう一度読み込んだときに、コミットされた別のトランザクションによってそのデータが変更または削除されたことが明らかになる現象
仮読み込み	あるトランザクションが検索条件を満たす一連の行を戻す問合せを2度実行する間に、コミットされた別のトランザクションによってその条件を満たす新しい行が挿入されたことが明らかになる現象

SQL92 では、それぞれの分離レベルで実行されるトランザクションで起こり得る現象という観点で4つの分離レベルを定義しています。表 24-1 に、各分離レベルを示します。

表 24-1 分離レベル別による回避可能な読み込み現象

分離レベル	内容を保証しない読み込み	非リピータブル・リード	仮読み込み
非コミット読み込み	あり	あり	あり
コミット読み込み	なし	あり	あり
リピータブル・リード	なし	なし	あり
シリアル化が可能	なし	なし	なし

Oracle で使用できる分離レベルは、SQL92 の一部ではない読取り専用モードと、「コミット読み込み」および「シリアル化が可能」です。デフォルトはコミット読み込みであり、Oracle リリース 7.3 より前では自動分離レベルとしてコミット読み込みのみが提供されていました。

関連項目： コミット読み込みとシリアル化が可能な分離レベルの詳細は、24-4 ページの「[データの同時実行性と一貫性の管理方法](#)」を参照してください。

ロックのメカニズム

一般に、マルチユーザー・データベースでは、なんらかのデータ・ロックを使用してデータの同時実行性、一貫性、および整合性の問題を解決しています。「ロック」は、同じリソースにアクセスしている複数のトランザクションの間で破壊的な相互作用が起きないようにするメカニズムです。

リソースには、次の 2 つの一般的なタイプのオブジェクトが含まれます。

- ユーザー・オブジェクト。たとえば、表と行（構造体とデータ）。
- ユーザーには見えないシステム・オブジェクト。たとえば、メモリー内の共有データ構造やデータ・ディクショナリ行。

関連項目： ロックの詳細は、24-15 ページの「[データをロックする方法](#)」を参照してください。

データの同時実行性と一貫性の管理方法

Oracle は、マルチバージョン一貫性モデルや、様々なタイプのロックおよびトランザクションを使用することによって、マルチユーザー環境でのデータの一貫性を維持します。この項のトピックは、次のとおりです。

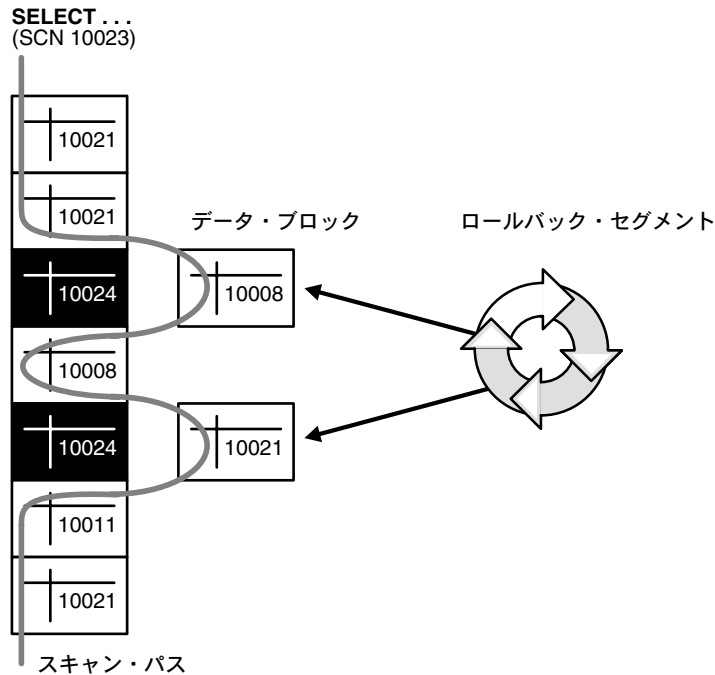
- [マルチバージョン一貫性制御](#)
- [文レベルの読取り一貫性](#)
- [トランザクション・レベルの読取り一貫性](#)
- [Oracle Parallel Server における読取り一貫性](#)
- [Oracle の分離レベル](#)
- [コミット読込み分離とシリアル化が可能な分離の比較](#)
- [分離レベルの選択](#)

マルチバージョン一貫性制御

Oracle では、ある問合せで参照されるデータのすべてが同一時点でのデータになるように、問合せに対して自動的に読取り一貫性が提供されます（文レベルの読取り一貫性）。読取り一貫性は、1 つのトランザクション内のすべての問合せに対しても提供できます（トランザクション・レベルの読取り一貫性）。

Oracle は、これらの一貫したデータの参照を実現するために、ロールバック・セグメント内に保持される情報を使用します。ロールバック・セグメントは、コミットされていないトランザクション、または最近コミットされたトランザクションによって変更されたデータの古い値が含まれています。図 24-1 に、Oracle がロールバック・セグメント内のデータを使用して、文レベルの読取り一貫性をどのように提供するかを示します。

図 24-1 トランザクションと読取り一貫性



問合せが実行段階に入ると現行のシステム変更番号（SCN）が判別されます。図 24-1 では、このシステム変更番号が 10023 になっています。問合せのためにデータ・ブロックを読み込むと、観察されたシステム変更番号（SCN）が書き込まれたブロックのみが使用されます。変更されたデータを含むブロック（もっと後の SCN）があると、それらはロールバック・セグメント内のデータに基づいて再構成され、問合せには再構成されたデータが戻されます。そのため、問合せを実行するたびに、問合せの実行開始時点で記録された SCN に関してコミット済のすべてのデータが戻されます。問合せの実行中に発生する他のトランザクションの変更は参照されません。このようにして、各問合せに対して一貫性のあるデータが戻されることが保証されます。

「スナップショットが古すぎます」というメッセージ

長時間実行の問合せでは、ごくまれに Oracle が一貫した結果セット（スナップショット）を戻せないことがあります。これが起こるのは、より古いデータを再構成するのに十分な情報がロールバック・セグメント内に残っていないためです。通常、このエラーが発生するのは、多くの更新アクティビティによってロールバック・セグメントが折り返され、長時間実行する問合せが必要とするデータの再構築に必要な変更が上書きされてしまう場合です。このイベントでは、次のようなエラーが発生します（*string* には文字列が入ります）。

ORA-1555: スナップショットが古すぎます (ロールバック・セグメント番号 *string*, 名前 *string* が小さすぎます)。

このエラー状態は、より多くの、またはより大きなロールバック・セグメントを作成することによって回避できます。また、同時実行トランザクションの数が少ないときに長時間実行の問合せを発行するという方法や、問合せの対象となる表について共有ロックを取得してトランザクションの実行中には他の排他ロックを禁止するという方法でも、このエラーを回避できます。

文レベルの読取り一貫性

Oracle では、常に「文レベル」の読取り一貫性が保たれています。これによって、1つの問合せによって戻されるすべてのデータは、その問合せが開始された時点のものであることが保証されます。そのため、問合せは、内容が保証されないデータも、その問合せの実行中にコミットされたトランザクションによって変更されたデータもまったく参照しません。問合せの実行中にその問合せで参照できるのは、その問合せが開始される前にコミットされたデータのみです。問合せは、文の実行の開始後にコミットされた変更は参照しません。

あらゆる問合せに対して一貫した一連の結果が保証され、これによって、データの一貫性が保証されます。この際、ユーザーは何もする必要がありません。SQL 文の SELECT、副問合せを伴う INSERT、UPDATE および DELETE は、いずれも明示的または暗黙的にデータの問合せを実行し、一貫性のあるデータを戻します。これらの文は、問合せを使用して、処理 (SELECT、INSERT、UPDATE または DELETE) の対象となるデータを判別します。

SELECT 文は明示的な問合せであり、ネストした問合せや結合操作を持つこともあります。INSERT 文では、ネストした問合せを使用できます。UPDATE 文と DELETE 文では、WHERE 句や副問合せを使用し、表のすべての行ではなく、一部の行を処理の対象にすることができます。

INSERT 文、UPDATE 文および DELETE 文で使用される問合せでは、一貫した結果セットの取得が保証されます。ただし、その DML 文そのものによって加えられる変更は見えません。つまり、そのような操作での問合せは、その操作によって変更される前のデータの状態を参照します。

トランザクション・レベルの読取り一貫性

Oracle では、「トランザクション・レベルの読取り一貫性」を保つこともできます。シリアル化が可能なモード (下記参照) でトランザクションが実行されている場合は、そのトランザクションが開始された時点でのデータベースの状態が、そのすべてのデータ・アクセスに反映されます。つまり、シリアル化が可能なトランザクションが発行した問合せではそのトランザクション自身が実行した変更が参照されるという点を除けば、同一のトランザクション内にあるどの問合せで参照されるデータも、1つの時点を基準とした一貫性が保たれているということです。トランザクション・レベルの読取り一貫性によってリピータブル・リードが実現され、問合せで実体のないデータが検索されることもありません。

Oracle Parallel Server における読取り一貫性

Oracle Parallel Server では、単一データベースにアクセスする複数インスタンス間でデータ整合性を保証するために、「キャッシュ・フュージョン」というパラレル・キャッシュ管理テクニックが採用されています。読取り一貫性を備えたブロックに対するインスタンス間の要求によって、読込み側と書込み側にキャッシュ一貫性矛盾が生じると、キャッシュ・フュージョンはブロック・サーバー・プロセスを使用して、ブロックを保持しているインスタンスのメモリー・キャッシュから要求側インスタンスのメモリー・キャッシュに、ブロックを直接コピーします。ブロックを保持しているインスタンスは、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、ブロックをディスクに書き込まないで要求側に直接送ります。このブロックの状態は、要求側ノードで要求が発行された時点で一貫性を備えています。

関連項目：

キャッシュ・フュージョンおよびブロック・サーバー・プロセスのプロセスの詳細は、次のマニュアルを参照してください。

- 『Oracle8i Parallel Server 概要』
- 『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

Oracle の分離レベル

Oracle には次のトランザクション分離レベルがあります。

コミット読込み	デフォルトのトランザクション分離レベル。あるトランザクションによって実行されるそれぞれの問合せで参照されるのは、その問合せ（トランザクションではなく）が開始される前にコミットされたデータのみです。Oracle の問合せでは、内容が保証されない（コミットされていない）データが読み込まれることはありません。 Oracle では、問合せで読み込まれたデータを他のトランザクションで変更できるため、問合せを 2 回実行する間に最初の実行でデータを他のトランザクションが変更する可能性があります。このため、1 つの問合せを 2 回実行するトランザクションでは、非リピータブル・リードと仮読込み（実体のない読込み）の現象の両方が起こり得ます。
シリアル化が可能なトランザクション	シリアル化が可能なトランザクションでは、そのトランザクションを開始した時点でコミット済の変更と、そのトランザクション自身が INSERT 文、UPDATE 文および DELETE 文で実行した変更のみが参照されます。シリアル化が可能なトランザクションでは、非リピータブル・リードや仮読込みの現象は起きません。

読取り専用

読取り専用トランザクションでは、そのトランザクションを開始した時点でコミット済の変更のみが参照され、INSERT 文、UPDATE 文および DELETE 文は使用できません。

分離レベルの設定

アプリケーション・デザイナー、アプリケーション開発者およびデータベース管理者は、アプリケーションと作業負荷に応じて、それぞれのトランザクションに適した分離レベルを選択できます。トランザクションの分離レベルは、トランザクションの開始時に次の文のいずれかを使用して設定できます。

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
SET TRANSACTION ISOLATION LEVEL READ ONLY;
```

各トランザクションを SET TRANSACTION 文で開始する場合のネットワークングおよび処理のコストを節約するために、ALTER SESSION 文を使用して、後続のすべてのトランザクションのトランザクション分離レベルを設定することもできます。

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
```

```
ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;
```

関連項目： これらの SQL 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

コミット読込みによる分離

Oracle のデフォルト分離レベルは、コミット読込みです。このレベルの分離は、競合するトランザクションの数が少ない環境に適しています。問合せごとに、それぞれ独自のスナップショット時間を基準として実行されます。このため、ある問合せを複数回実行する間に非リピータブル・リードと仮読込みが発生することもあります。高いスループットが得られます。コミット読込みによる分離は、競合するトランザクションの数が少ない環境に適した分離レベルです。

シリアル化が可能な分離

シリアル化が可能な分離は、次のような環境に適しています。

- 大規模データベースを使用し、短いトランザクションでごく少数の行しか更新しない環境。
- 2つの同時実行トランザクションが同一の行を更新するようなことが比較的少ない環境。
- 比較的長時間にわたって実行されるトランザクションが主に読取り専用である環境。

シリアル化が可能な分離では、同時実行トランザクションが実行できる変更は、トランザクションが順に1つずつ実行される場合に実行できるデータベース変更のみです。特に、シリアル化が可能なトランザクションでデータ行を変更できるのは、その行に対するそれ以前の変更が、シリアル化が可能なトランザクションの開始時にすでにコミットされていたトランザクションによるものであると判別できる場合のみです。

この判別の効率をよくするために、データ・ブロック内に格納されている制御情報、つまりブロック内の行のうち、どの行にコミットされた変更が含まれていて、どの行にコミットされてない変更が含まれているかを示す情報が使用されます。ある意味で、ブロックには、ブロック内の各行に影響を与えたトランザクションの最新の履歴が含まれていると考えられます。ここに保存される履歴の量は、CREATE TABLE および ALTER TABLE の INITRANS パラメータによって制御されます。

場合によっては、ごく最近のトランザクションで行が更新されたかどうかを判別するには履歴情報が不十分である場合もあります。これは、多数のトランザクションが同時に同じデータ・ブロックを変更している場合や、非常に短い期間にそのような操作が実行されている場合に起きることがあります。多数のトランザクションが同一のブロックを変更するような表については、INITRANS の値を大きく設定しておくことにより、この状況を回避できます。それにより、ブロックにアクセスした最新のトランザクションの履歴を記録するのに十分な記憶域が各ブロックに割り当てられます。

シリアル化が可能なトランザクションが、その開始後にコミットされたトランザクションによって変更されたデータを更新または削除しようとする、次のようなエラーが生成されます。

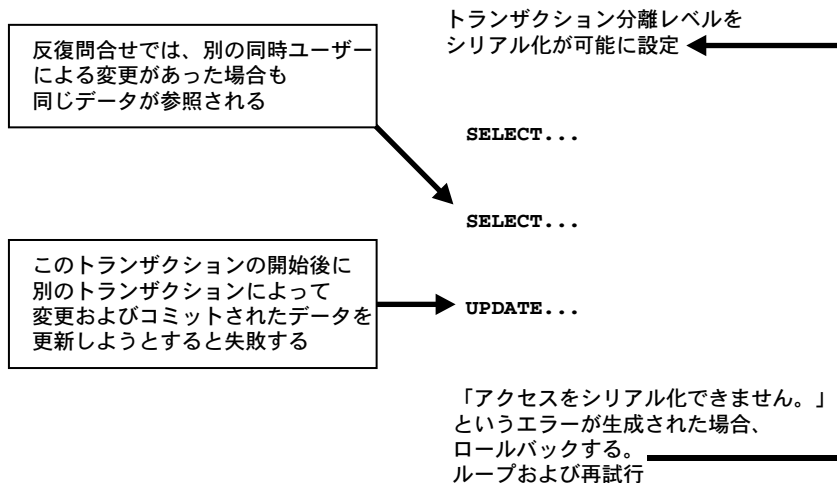
ORA-08177: このトランザクションのアクセスをシリアル化できません。

シリアル化が可能なトランザクションが失敗し、「アクセスをシリアル化できません。」というエラーが出た場合、アプリケーションは次のいずれかのアクションを実行できます。

- 実行した作業をそのポイントまでコミットします。
- 追加の（異なる）文を実行します（トランザクション内で以前に設定したセーブポイントまでロールバックした後など）。
- トランザクション全体をロールバックします。

図 24-2 に、「アクセスをシリアル化できません。」というエラーになったトランザクションをロールバックして再試行するアプリケーションの例を示します。

図 24-2 シリアル化が可能なトランザクションの失敗



コミット読み分離とシリアル化が可能な分離の比較

Oracle では、異なる特性を持つ 2 つのトランザクション分離レベルのどちらかを選択できます。コミット読み分離とシリアル化が可能な分離レベルでも、高度な一貫性と同時実行性が提供されます。どちらの分離レベルでも、Oracle の読取り一貫性によるマルチバージョン同時実行性制御モデルと、排他的な行レベルのロックングが実装されることによって競合が削減されます。これらの分離レベルは、実際のアプリケーションの配置を考慮して設計されています。

トランザクション集合の一貫性

Oracle におけるコミット読み分離とシリアル化が可能な分離レベルを理解するため、次のような使用例を考えます。データベースの表の集合（またはデータの任意の集合）があり、それらの表内の行を特定の順序で何度か読み込み、一連のトランザクションをある特定の時点でコミットするとします。ある操作（問合せまたはトランザクション）のどの読み込みでも、コミット済みのトランザクションの同一の集合によって書き込まれたデータが戻される場合、その操作には「トランザクション集合の一貫性」があります。一部の読み込みにはあるトランザクション集合による変更が反映されており、他の読み込みには別のトランザクション集合による変更が反映されている場合、その操作にはトランザクション集合の一貫性がありません。トランザクション集合の一貫性のない操作では、事実上、コミット済みの 1 つのトランザクション集合が反映された状態のデータベースを一貫して参照できません。

Oracle では、コミット読みモードで実行されるトランザクションには文単位でのトランザクション集合の一貫性が提供されます。シリアル化が可能なモードでは、トランザクション単位でのトランザクション集合の一貫性が提供されます。

表 24-2 に、Oracle での コミット読みトランザクションとシリアル化が可能なトランザクションの主な違いをまとめます。

表 24-2 コミット読みトランザクションとシリアル化が可能なトランザクション

	コミット読み	シリアル化が可能
内容を保証しない書き込み	なし	なし
内容を保証しない読み	なし	なし
非リピータブル・リード	あり	なし
仮読み	あり	なし
ANSI/ISO SQL 92 への準拠	はい	はい
スナップショット読み時間	文	トランザクション
トランザクション集合の一貫性	文レベル	トランザクション・レベル
行レベル・ロック	はい	はい
読みが書き込みを阻止するか	いいえ	いいえ
書き込みが読みを阻止するか	いいえ	いいえ
別の行の書き込みが書き込みを阻止するか	いいえ	いいえ
同一行の書き込みが書き込みを阻止するか	はい	はい
阻止しているトランザクションを待機するか	はい	はい
「アクセスをシリアル化できません。」が発生するか	いいえ	はい
阻止しているトランザクションの異常終了後にエラーが発生するか	いいえ	いいえ
阻止しているトランザクションのコミット後にエラーが発生するか	いいえ	はい

行レベル・ロック

コミット読みトランザクションとシリアル化が可能なトランザクションは、どちらも行レベルのロックを使用します。また、コミットされていない同時実行のトランザクションによって更新された行を変更しようとする、待ち状態になります。ある行を 2 番目に更新しようとしたトランザクションは、最初のトランザクションがコミットまたはロールバックされてロックが解除されるまで、待ち状態になります。この最初のトランザクションがロールバックした場合、どの分離モードでも、待ち状態のトランザクションは最初のトランザクションが存在しなかったかのように、以前にロックされた行を変更できるようになります。

ただし、阻止している最初のトランザクションがコミットされてロックが解除された場合、コミット読み込みトランザクションは、目的の更新処理を続行します。また、シリアル化が可能なトランザクションの場合には、そのシリアル化が可能なトランザクションの開始後になされた変更を他のトランザクションがコミットしているため、エラー、「アクセスをシリアル化できません。」を出して失敗します。

参照整合性

Oracle は、読み取り一貫性トランザクションでもシリアル化が可能なトランザクションでも読み込みロックを使用しないため、あるトランザクションで読み込んだデータが別のトランザクションによって上書きされる可能性があります。アプリケーション・レベルでデータベースの一貫性のチェックを実行するトランザクションでは、データの変更がトランザクションから見えなくても、読み込んだデータがトランザクションの実行中に変更されることはないとは想定しないでください。このことを考慮してアプリケーション・レベルの一貫性チェックをコーディングしない限り、シリアル化が可能なトランザクションを使用しても、データベースの一貫性が損なわれることがあります。

関連項目： 参照整合性とシリアル化が可能なトランザクションの詳細は、『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。

Oracle Parallel Server

Oracle Parallel Server では、コミット読み込みトランザクションとシリアル化が可能なトランザクションの両方の分離レベルを使用できます。Oracle Parallel Server 環境では、1つのデータベースに対して複数の Oracle インスタンスが実行されます。

分散トランザクション

分散データベース環境では、一部のノードのみがコミットされることがないように、複数の物理データベースが2 フェーズ・コミットによって保護され、1つのトランザクションによって複数の物理データベースのデータが更新されます。そのような環境では、Oracle かどうかを問わず、シリアル化が可能なトランザクションに関与するすべてのサーバーで、シリアル化が可能な分離モードがサポートされている必要があります。

シリアル化が可能なトランザクションをサポートしていないサーバーによって管理されているデータベース内のデータをシリアル化が可能なトランザクションが更新しようとする、そのトランザクションにはエラーが戻されます。トランザクションがロールバックして再試行できるのは、リモート・サーバーがシリアル化が可能なトランザクションをサポートしている場合のみです。

これとは対照的に、コミット読み込みトランザクションでは、シリアル化が可能なトランザクションをサポートしていないサーバーで分散トランザクションを実行できます。

関連項目： 『Oracle8i 分散システム』

分離レベルの選択

アプリケーション・デザイナーおよびアプリケーション開発者は、アプリケーションのコーディング以外にアプリケーションのパフォーマンスと一貫性のニーズに基づいて分離レベルを選択する必要があります。

多数の同時実行ユーザーが次々にトランザクションを送る環境では、トランザクションの予想到着率および応答時間の要求の面から、トランザクションのパフォーマンス要件を評価する必要があります。高いパフォーマンスが必要な環境では、分離レベルを選択する際に一貫性と同時実行性とのバランスを考慮する必要があります。

データベースの一貫性のチェックを実行するアプリケーションのロジックでは、どちらのモードでも読み込みによっては書き込みが阻止されないという点を考慮する必要があります。

Oracle の 2 つの分離モードは、行レベル・ロックと Oracle のマルチバージョン同時実行性制御システムとの組合せによって、高水準の一貫性、同時実行性およびパフォーマンスを提供します。Oracle では、読み込み側と書き込み側が互いに処理を阻止しないため、問合せで一貫性のあるデータが参照される一方で、コミット読み込みとシリアル化が可能などちらの分離レベルでも、コミットされていない（内容が保証されない）データを読むことなく、高パフォーマンスを実現する高水準の同時実行性を提供します。

コミット読み込み分離モードの選択

多くのアプリケーションでは、コミット読み込みが最適な分離レベルです。これは、リリース 7.3 より前の Oracle 上で実行されるアプリケーションが使用する分離レベルです。

コミット読み込みによる分離では、同時実行性が大幅に向上する一方で、一部のトランザクションでは、仮読み込みや非リピータブル・リードのために、一貫性のない結果が生じる危険性が多少高くなります。

トランザクションの到着率が高く、高いパフォーマンスが求められる多くの環境では、シリアル化が可能な分離レベルによって実現されるよりも大きなスループットと高速な応答時間が必要になる場合があります。サポートするユーザーが少数で、トランザクション到着率が非常に低い環境では、仮読み込みおよび非リピータブル・リードによって間違った結果が生じる危険性が非常に低くなります。コミット読み込み分離は、いずれの環境にも適しています。

Oracle のコミット読み込み分離では、すべての問合せについてトランザクション集合の一貫性が提供されます。つまり、どの問合せでも一貫性のある状態のデータが参照されます。したがって、マルチバージョン同時実行性制御を使用しない他のデータベース管理システム上で実行される場合にはさらに高水準の分離を必要とするようなアプリケーションでも、多くの場合、コミット読み込み分離で十分に対応できます。

コミット読み込み分離モードでは、アプリケーションのロジックで「アクセスをシリアル化できません。」というエラーをトラップしたり、処理をロールバックしてトランザクションを再起動する必要はありません。大部分のアプリケーションでは、同じ問合せを 2 回繰り返して発行するという機能的必要があるトランザクションはごく少数のため、仮読み込みおよび非リピータブル・リードの防止は重要ではありません。したがって、前述のようなエラー・チェックや再試行のコードを各トランザクションで記述するのを避けるために、多くの開発者はコミット読み込みを選択します。

シリアル化が可能な分離モードの選択

Oracle のシリアル化が可能な分離は、2 つの同時実行トランザクションが同じ行を更新する機会が比較的少なく、比較的長時間にわたって実行されるトランザクションが主に読取り専用である環境に適しています。この分離モードが最も適しているのは、大規模なデータベースを使用し、短いトランザクションでごく少数の行のみが更新される環境です。

シリアル化が可能な分離モードは、仮読込みと非リピータブル・リードを防止することによってある程度まで一貫性を向上できるため、読込み / 書込みトランザクションが 1 つの問合せを複数回実行する場合に重要になることがあります。

他の処理系のシリアル化が可能な分離では書込みのみでなく読込みについてもブロックがロックされるのに対し、Oracle では非ブロッキング問合せときめ細かい行レベル・ロックが提供され、それらによって書込み / 読込みの競合が少なくなります。読込み / 書込みの競合が多く発生するアプリケーションでは、Oracle のシリアル化が可能な分離は、他のシステムより大幅に高いスループットを提供します。このため、Oracle 上ではシリアル化が可能な分離に適していても、他のシステムではシリアル化が可能な分離に適さないアプリケーションもあります。

Oracle のシリアル化が可能なトランザクション内のすべての問合せは、一時点でのデータベースを参照するため、読込み / 書込みトランザクションで複数の一貫した問合せを発行する必要がある場合は、この分離レベルが適しています。シリアル化が可能なモードでは、READ ONLY トランザクションで実現される一貫性が確保される一方で、INSERT、UPDATE および DELETE トランザクションも実行できるため、サマリー・データを生成してそのデータをデータベースに格納する報告書作成アプリケーションでは、シリアル化が可能なモードを使用してください。

注意： 副問合せを持つ DML 文を含むトランザクションは、コミット読込みを保証するためにシリアル化が可能な分離を使用する必要があります。

アプリケーション開発者がシリアル化が可能なトランザクションをコーディングする場合には、「アクセスをシリアル化できません。」のエラーのチェックとトランザクションのロールバックおよび再試行のために、追加の作業が必要になります。他のデータベース管理システムでデッドロックを管理する際にも、同様の追加コーディングが必要です。社内標準を遵守する場合や、複数のデータベース管理システム上で実行するアプリケーションを作成する場合には、シリアル化が可能なモードに対応したトランザクションの設計が必要なことがあります。シリアル化の可能性エラーをチェックして、再試行するトランザクションは、Oracle のコミット読込みモード、つまりシリアル化の可能性エラーを生成しないモードで使用できます。

シリアル化が可能なモードにあまり適さないのは、大量の短い更新トランザクションでアクセスするのと同じ行を、比較的長いトランザクションで更新する環境です。このような環境では、長時間実行のトランザクションが行を最初に更新するという可能性は低いため、頻繁なロールバックが必要となり、作業が無駄になります。従来型の読込みをロックしたシリアル化が可能なモードの悲観的インプリメンテーションであり、この環境には適していません。長時間実行されるトランザクションは、読込みトランザクションであっても、短い更新トランザクションの処理を阻止したり、逆に短いトランザクションが長時間実行のトランザクションの処理を阻止する場合があるためです。

アプリケーション開発者は、シリアル化が可能なモードを使用するときはトランザクションのロールバックと再試行に要するコストを考慮する必要があります。デッドロックが頻繁に発生する読込みロックのシステムと同様に、シリアル化が可能なモードを使用するときには、異常終了したトランザクションによって実行された処理をロールバックし、再実行する必要があります。競合が頻繁に発生する環境では、このアクティビティでリソースが著しく消費されます。

ほとんどの環境では、「アクセスをシリアル化できません。」のエラーを受け取った後で再起動されたトランザクションと別のトランザクションの間で、2度目の競合が発生することはめったにありません。このため、他のトランザクションと競合する可能性の高い文は、シリアル化が可能なトランザクション内の可能な限り前の部分で実行するとよい場合があります。ただし、そのトランザクションが正常に完了する保証はないため、再試行の回数を制限するようにアプリケーションをコーディングしておく必要があります。

Oracle のシリアル化が可能なモードには SQL92 との互換性があり、読込みロックの処理系と比較して多くの利点がありますが、意味としてはそれらのシステムと同じではありません。アプリケーション・デザイナーは、Oracle での読込みは、他のシステムとは違って、書込みを阻止しないという点を考慮する必要があります。データベースの一貫性をアプリケーション・レベルでチェックするトランザクションでは、SELECT FOR UPDATE などのコーディング技法を使用することが必要な場合があります。シリアル化が可能なモードを使用しているアプリケーションを他の環境から Oracle に移植する場合には、この問題を検討する必要があります。

データをロックする方法

「ロック」は、同じリソース、つまりユーザー・オブジェクト（表や行など）またはユーザーには見えないシステム・オブジェクト（メモリー内の共有データ構造やデータ・ディクショナリ行など）のどちらかにアクセスする複数のトランザクションの間で、破壊的な相互作用を回避するためのメカニズムです。

どのような状況でも、SQL 文が実行されると、Oracle によって必要なロックが自動的に取得されます。そのため、ユーザーがそのような詳細事項にかかわる必要はありません。Oracle は、最も高度なデータ同時実行性とフェイルセーフなデータの整合性を同時に実現するために、最も低いレベルの制限でデータを自動的にロックします。また、必要に応じて、手動でもデータをロックできます。

関連項目： 24-19 ページの「[ロックの種類](#)」

トランザクションとデータ同時実行性

Oracle ではロック・メカニズムを使用して、トランザクション間のデータの同時実行性と一貫性を実現します。Oracle のロック・メカニズムはトランザクション制御と密接に結び付けられているため、アプリケーション・デザイナーがトランザクションを適切に定義するだけで、ロックは Oracle によって自動的に管理されます。

Oracle のロックは完全に自動化されていて、ユーザー・アクションを必要としません。すべての SQL 文について暗黙のロックが発生するため、データベース・ユーザーがリソースを明示的にロックする必要はありません。Oracle のデフォルトのロック・メカニズムは、最高度のデータ同時実行性を実現しながら、データの整合性を保証するために、最も低い制限レベルでデータをロックします。

関連項目： 24-32 ページの「[明示的（手動）データ・ロック](#)」

ロックのモード

Oracle では、マルチユーザー・データベースで2つのロック・モードを使用します。

- | | |
|-----------|---|
| 排他ロック・モード | 関連リソースが共有されないようにします。このロック・モードはデータを変更するために取得します。リソースを排他的にロックした最初のトランザクションは、その排他ロックが解除されるまで、そのリソースを変更できる唯一のトランザクションになります。 |
| 共有ロック・モード | 操作の種類に応じて、関連するリソースの共有が可能です。データの読み込みを実行する複数のユーザーはそのデータを共有でき、また共有ロックを保持することによって、排他ロックを必要とする書き込みユーザーの同時アクセスを防ぎます。複数のトランザクションが同じリソースについて共有ロックを取得できます。 |

ロックの期間

あるトランザクション内の文によって取得されたすべてのロックは、そのトランザクションの存続期間中は保持され、同時実行のトランザクションによる有害な干渉（内容を保証しない読み込み、更新内容の消失、有害な DDL 操作など）が防止されます。あるトランザクションの SQL 文によって加えられた変更を参照できるのは、そのトランザクションがコミットされた後に開始される別のトランザクションのみです。

トランザクションをコミットまたはロールバックすると、そのトランザクション内の文によって取得されたすべてのロックが解除されます。また、セーブポイントまでロールバックした場合には、そのセーブポイントより後で取得されたロックが解除されます。ただし、ロックを解除されて使用可能となったリソースに対してロックを取得できるのは、ロック中だったリソースを待機していなかったトランザクションのみです。ロック中のリソースを待機していたトランザクションは、元のトランザクションが完全にコミットされるかロールバックされるまで待機し続けます。

データ・ロック変換とロックの段階的拡大の比較

トランザクションは、トランザクション内で挿入、更新または削除の対象になる行すべてに対して排他ロックを保持します。行ロックは、制限の最も高い程度で取得されるため、ロック変換はまったく必要なく、実行されません。

Oracle は、低い制限の表ロックを、より高い制限の適切な表ロックに自動的に変換します。たとえば、トランザクションが表の行をロックするために、FOR UPDATE 句を指定した SELECT 文を使用する場合を考えます。その結果、排他行ロックとその表に対する行共有表ロックが取得されます。後でトランザクションがロック中の 1 つ以上の行を更新する場合、行共有表ロックは自動的に行排他表ロックに変換されます。

ロックの段階的拡大が発生するのは、あるレベル（行レベルなど）で多数のロックが保持されている場合に、各ロックをデータベースが上位レベル（表レベルなど）の別のロックに変更した場合です。たとえば、あるユーザーが表の中の多数の行をロックした場合、データベースによっては、そのユーザーが保持する複数の行ロックを単一の表ロックに自動的に拡大する場合があります。ロックの数は少なくなりますが、ロックされている対象の制限は大きくなります。

Oracle では、ロックの段階的拡大が発生することはありません。ロックの段階的拡大はデッドロックの可能性を大幅に増します。たとえば、システムがトランザクション T1 のためにロックを拡大しようとしませんが、トランザクション T2 によって保持されているロックのために拡大できないものとします。トランザクション T2 の処理を進めるのに同じデータのロックの段階的拡大が必要な場合、デッドロックが発生します。

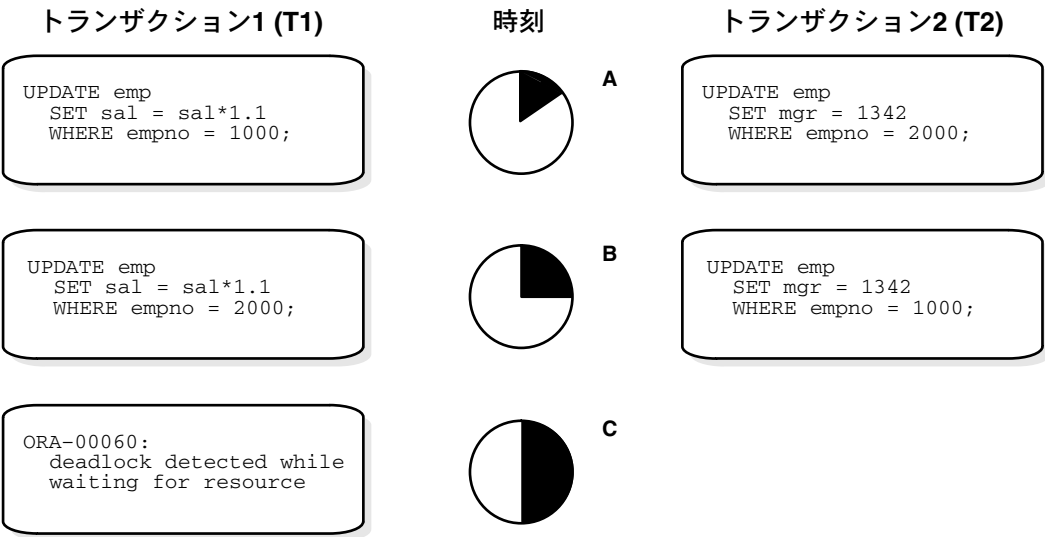
関連項目： 24-22 ページの「[表ロック \(TM\)](#)」

デッドロック

「デッドロック」が発生するのは、2 人以上のユーザーが、相手がロックしているデータを待機している場合です。デッドロックが発生すると、一部のトランザクションは処理を継続できなくなります。図 24-3 に、デッドロック状態になっている 2 つのトランザクションを示します。

図 24-3 において、各トランザクションはそれ自体が更新しようとする行に対して行ロックを保持するため、時刻 A では問題は存在しません。各トランザクションの処理は、終了せずに進行します。ただし、各トランザクションは、次に、他方のトランザクションが現在保持している行を更新しようとします。したがって、どちらのトランザクションも処理の進行や終了に必要なリソースを取得できないため、結局、時刻 B でデッドロックが発生します。デッドロックになるのは、それぞれのトランザクションがいくら待機しても、競合するロックが保持されるためです。

図 24-3 デッドロック状態の 2 つのトランザクション



デッドロックの検出

Oracle は、デッドロック状況を自動的に検出し、そのデッドロックに含まれる文の 1 つをロールバックして、競合する一連の行ロックを解放することにより、デッドロックを解決します。また、対応するメッセージが、文レベルのロールバックの対象となったトランザクションに戻されます。ロールバックされる文は、デッドロックが検出されたトランザクションに属しています。通常、通知されたトランザクションは明示的にロールバックする必要がありますが、待機した後でロールバック文を再試行できます。

注意： 分散トランザクションの場合、ローカル・デッドロックは待機グラフを分析することによって検出され、グローバル・デッドロックはタイムアウトによって検出されます。いったん検出されると、非分散デッドロックも分散デッドロックも、データベースとアプリケーションによって同じように処理されます。

デッドロックが最も頻繁に発生するのは、トランザクションが Oracle のデフォルト・ロックを明示的に上書きする場合です。Oracle 自体はロックの段階的拡大を実行せず、また、問合せに読み込みロックを使用せず、また行レベルのロック（ページ・レベルのロックではありません）を使用するため、Oracle ではデッドロックはまれにしか起こりません。

関連項目： 手動によるロックの取得の詳細とデッドロック状況の例は、24-32 ページの「[明示的（手動）データ・ロック](#)」を参照してください。

デッドロックの回避

多くの場合、複数表のデッドロックは、複数の同じ表にアクセスする複数のトランザクションが、暗黙のロックまたは明示的のロックを通じて各表を同じ順序でロックすれば回避できます。たとえば、すべてのアプリケーション開発者は、マスター表とディテール表の両方を更新するときに、まずマスター表をロックしてからディテール表をロックするという規則に従ってください。この規則を適切に設計し、すべてのアプリケーションがそれに従えば、デッドロックが起こる可能性はかなり低くなります。

あるトランザクションについて一連のロックが必要となることがわかっているときは、最も排他的な（最も共存不能な）ロックが最初に取得されるように考慮してください。

ロックの種類

Oracle は、データへの同時アクセスを制御し、各ユーザー間の有害な干渉を防止するために、様々なタイプのロックを自動的に使用します。Oracle は、トランザクションのためにリソースを自動的にロックし、他のトランザクションが同じリソースの排他アクセスを要求する処理を行うことを防止します。なんらかのイベントが発生し、トランザクションがそのリソースを必要としなくなると、ロックは自動的に解除されます。

全操作を通じて、Oracle はロックされるリソースと実行される操作に応じて、様々な制限レベルの様々なタイプのロックを自動的に取得します。

Oracle のロックは次のいずれか 1 つに分類されます。

DML ロック (データ・ロック)	DML ロックはデータを保護します。たとえば、表ロックは表全体をロックし、行ロックは選択された行をロックします。
DDL ロック (ディクショナリ・ロック)	DDL ロックは、スキーマ・オブジェクトの構造、たとえば表とビューの定義を保護します。
内部ロックとラッチ	内部ロックとラッチは、データ・ファイルなどの内部データベース構造を保護します。内部ロックとラッチは完全に自動的です。
分散ロック	分散ロックは、Oracle Parallel Server の複数のインスタンス間に分散されたデータや他のリソースが一貫性を維持していることを保証します。分散ロックは、トランザクションではなくインスタンスによって保持され、Oracle Parallel Server のインスタンス間でリソースの現在の状態を伝達します。
パラレル・キャッシュ管理 (PCM) ロック	パラレル・キャッシュ管理ロックは、バッファ・キャッシュ内の 1 つ以上のデータ・ブロック（表ブロックまたは索引ブロック）を対象とする分散ロックです。PCM ロックは、トランザクションのためには行をロックしません。

この章では、DML ロック、DDL ロックおよび内部ロックについて個別に説明します。

関連項目：

分散ロックと PCM ロックの詳細は、次のマニュアルを参照してください。

- 『Oracle8i Parallel Server 概要』
- 『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

DML ロック

DML（データ）ロックの目的は、複数のユーザーが同時にアクセスするデータの一貫性を保証することです。DML ロックは、同時に実行された矛盾する複数の DML 操作または DDL 操作（あるいはその両方）の有害な干渉を防ぎます。たとえば、Oracle の DML ロックでは、一度に 1 つのトランザクションのみが表の中の特定の行を更新すること、また、コミットされていないトランザクションに表への挿入操作が含まれている場合はその表が削除されないことが保証されます。

DML 操作では、2 つの異なるレベルでデータ・ロックを取得できます。1 つは特定の行に対するロック、もう 1 つは表全体に対するロックです。この後の各項では、行ロックと表ロックについて説明します。

注意： これ以降、それぞれのタイプのロックまたはロック・モードの後のカッコ内にある頭字語は、Oracle Enterprise Manager の Locks Monitor で使用される略称です。Oracle Enterprise Manager では、表ロックのモード (RS や SRX など) が示されるかわりに、すべての表ロックに TM と表示される可能性があります。

行ロック (TX)

Oracle が自動的に取得する DML ロックは、行レベルのロックのみです。1 つの文またはトランザクションで保持できる行ロックの数に制限はありません。また、Oracle が行レベルのロックをさらに粗い単位のロックに拡大することはありません。行ロックでは、最もきめの細かいロックが実現されるため、最高の同時実行性とスループットが得られます。

マルチバージョン同時実行性制御と行レベル・ロックを組み合わせると、同じデータに関して複数のユーザーが競合するのは、同じ行にアクセスする場合のみになります。特に、次のような場合です。

- データの読み込みが、同じデータ行の書き込みを待機しない場合。
- データの書き込みが、同じデータ行の読み込みを待機しない場合。ただし、読み込みのロックを要求する SELECT... FOR UPDATE を使用していない場合のみ。
- 他の書き込みが同時に同じ行を更新しようとする場合に限り、書き込みは他の書き込みを待機します。

注意： 保留中の分散トランザクションにおける非常に特殊な状況では、データを読み込むために、同じデータ・ブロックへの書き込みの待機が必要になることもあります。

INSERT 文、UPDATE 文、DELETE 文および FOR UPDATE 句を含む SELECT 文のいずれかで変更された行ごとに、トランザクションは排他 DML ロックを取得します。

変更された行は、ロックを保持しているトランザクションがコミットされるかロールバックされるまで、他のユーザーがその行を変更できないように**常に**排他的にロックされます。ただし、インスタンス障害のためにトランザクションが終了すると、トランザクション全体がリカバリされる前に、ブロック・レベルのリカバリによって行が使用可能になります。前述の文の結果として、行ロックは、常に Oracle によって自動的に取得されます。

ある行について行ロックを取得したトランザクションは、その行に対応する表についての表ロックも取得します。表ロックがあると、カレント・トランザクション内のデータ変更を上書きする DDL 操作の競合が回避されます。

関連項目：

- 29-4 ページの「データベース・インスタンス障害」
- 24-29 ページの「DDL ロック」

表ロック (TM)

INSERT 文、UPDATE 文、DELETE 文、FOR UPDATE 句付きの SELECT 文および LOCK TABLE 文の各 DML 文で表が変更される場合、トランザクションは表ロックを取得します。これらの DML 操作が表ロックを必要とするのは、トランザクションのために表への DML アクセスを確保する、およびトランザクションと競合する可能性がある DDL 操作を防止するという 2 つの目的のためです。すべての表ロックは同じ表に対する排他 DDL ロックを防止するため、そのようなロックを必要とする DDL 操作も防止されます。たとえば、コミットされていないトランザクションが、ある表について表ロックを保持している場合、その表を変更したり削除することはできません。

表ロックは、行共有 (RS)、行排他 (RX)、共有 (S)、共有行排他 (SRX) および排他 (X) のいずれかのモードで保持できます。表ロックのモードの制限は、他の表ロックが同じ表について取得し、保持できるモードを決定します。

表 24-3 に、文が取得する表ロックのモードと、各ロックで許可されている操作と禁止されている操作を示します。

表 24-3 表ロックのまとめ

SQL 文	表ロックのモード	許可されるロック・モード				
		RS	RX	S	SRX	X
SELECT...FROM table...	なし	可	可	可	可	可
INSERT INTO table ...	RX	可	可	不可	不可	不可
UPDATE table ...	RX	可 *	可 *	不可	不可	不可
DELETE FROM table ...	RX	可 *	可 *	不可	不可	不可
SELECT ... FROM table FOR UPDATE OF ...	RS	可 *	可 *	可 *	可 *	不可
LOCK TABLE table IN ROW SHARE MODE	RS	可	可	可	可	不可
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	可	可	不可	不可	不可
LOCK TABLE table IN SHARE MODE	S	可	不可	可	不可	不可
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	可	不可	不可	不可	不可
LOCK TABLE table IN EXCLUSIVE MODE	X	不可	不可	不可	不可	不可
		RS: 行共有 RX: 行排他 S: 共有 SRX: 共有行排他 X: 排他				
		* 別のトランザクションによって、競合する行ロックが保持されていない場合。そうでない場合は待機が発生。				

この後、制限レベルの低いものから高いものという順序で、表ロックの各モードについて説明します。それぞれの項では、表ロックのモード、トランザクションがそのモードで表ロックを取得する原因となるアクション、そのモードのロックで他のトランザクションに許可されるアクションと禁止されるアクションについて説明します。

関連項目： 24-32 ページの「明示的（手動）データ・ロック」

行共有表ロック (RS) 行共有表ロック（副共有表ロック、SS、とも呼ぶ）は、この表ロックを保持しているトランザクションが、その表の行をロックし、それらの行を更新する予定であることを示します。次の SQL 文のいずれかが実行された場合は、表の行共有表ロックが自動的に取得されます。

```
SELECT . . . FROM table . . . FOR UPDATE OF . . . ;
```

```
LOCK TABLE table IN ROW SHARE MODE;
```

行共有表ロックは、最も制限の緩やかなモードの表ロックであり、最高度の同時実行性を表に提供します。

許可される操作：トランザクションによって保持される行共有表ロックでは、他のトランザクションは、同じ表の中の行に対して問合せ、挿入、更新、削除またはロックを同時に実行できます。そのため他のトランザクションは、同じ表について同時に行共有ロック、行排他ロック、共有ロックおよび共有行排他ロックを取得できます。

禁止される操作：トランザクションによって保持される行共有表ロックは、他のトランザクションが次の文のみを使用して同じ表に排他書き込みアクセスを実行するのを防止します。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

行排他表ロック (RX) 行排他表ロック（副排他表ロック、SX、とも呼ぶ）は、一般に、このロックを保持しているトランザクションが、表の中の行に対して1つ以上の更新を行ったことを示します。行排他表ロックは、次のタイプの文によって修正される表について自動的に取得されます。

```
INSERT INTO table . . . ;
```

```
UPDATE table . . . ;
```

```
DELETE FROM table . . . ;
```

```
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

行排他表ロックでは、行共有表ロックよりも少し多くの制限が課されます。

許可される操作：トランザクションによって保持される行排他表ロックでは、他のトランザクションは、同じ表の中の行に対して問合せ、挿入、更新、削除またはロックを同時に実行できます。そのため、行排他表ロックによって、複数のトランザクションが同時に、同じ表について行排他ロックと行共有表ロックを取得できます。

禁止される操作：トランザクションによって保持される行排他表ロックは、他のトランザクションが排他的な読み込みや書き込みを実行するために表を手動でロックすることを防止します。そのため、他のトランザクションは、次の文を使用して同時に表をロックすることはできません。

```
LOCK TABLE table IN SHARE MODE;
```

```
LOCK TABLE table IN SHARE EXCLUSIVE MODE;
```

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

共有表ロック (S) 共有表ロックは、次の文で指定される表について自動的に取得されます。

```
LOCK TABLE table IN SHARE MODE;
```

許可される操作：トランザクションによって保持される共有表ロックでは、他のトランザクションは、表の問合せ、SELECT...FOR UPDATE 文による特定行のロック、または LOCK TABLE...IN SHARE MODE 文の実行のみが許可されます。したがって、他のトランザクションによる更新は許可されません。複数のトランザクションが、同じ表について同時に共有表ロックを保持できます。ただし、この場合、トランザクションは表を更新できません (FOR UPDATE 句を指定した SELECT 文の結果として行ロックを保持できたとしても更新できません)。そのため、共有表ロックを保持しているトランザクションがその表を更新できるのは、他のトランザクションが同じ表について共有表ロックを保持していない場合のみです。

禁止される操作：トランザクションによって保持される共有表ロックは、他のトランザクションが同じ表を変更するのを禁止します。また、他のトランザクションが次の文を実行することも禁止します。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

```
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

共有行排他表ロック (SRX) 共有行排他表ロック（共有副排他表ロック、SSX、とも呼ぶ）は、共有表ロックよりも多くの制限を課します。共有行排他表ロックは、次の文で指定された表について取得されます。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

許可される操作：一度に1つのトランザクションのみが、特定の表の共有行排他表ロックを取得できます。トランザクションによって保持される共有行排他表ロックでは、他のトランザクションは、問合せ、または FOR UPDATE 句を指定した SELECT 文による特定行のロックを許可されますが、表の更新は許可されません。

禁止される操作：トランザクションによって保持される共有行排他表ロックは、他のトランザクションによる行排他表ロックの取得、および同じ表の変更を防止します。また、共有行排他表ロックでは、他のトランザクションが共有ロック、共有行排他ロックおよび排他表ロックを取得することが禁止されます。つまり、他のトランザクションが次の文を実行できなくなります。

```
LOCK TABLE table IN SHARE MODE;
```

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

```
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

排他表ロック (X) 排他表ロックは、最も多くの制限が課されるモードの表ロックであり、ロックを保持するトランザクションが表に排他的に書込みアクセスすることを許可します。排他表ロックは、次の文で指定された「表」について取得されます。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

許可される操作：1つのトランザクションのみが、表の排他表ロックを取得できます。排他表ロックでは、他のトランザクションは同じ表に対する問合せのみを実行できます。

禁止される操作：トランザクションによって保持されている排他表ロックは、他のトランザクションによる DML 文の実行とその表に対するロックの適用を禁止します。

DML 文について自動的に取得される DML ロック

これまで、各種データ・ロックのタイプと、どのモードで保持できるか、いつ取得できるか、いつ取得されるか、何を禁止するかについて説明しました。この後の項では、各種の DML 操作を実行するために Oracle がどのようにデータを自動的にロックするかをまとめます。

表 24-4 に、この後の項で説明する情報をまとめておきます。

表 24-4 DML 文が取得するロック

DML 文	行ロック	表ロックのモード
SELECT ... FROM table		
INSERT INTO table ...	X	RX
UPDATE table ...	X	RX
DELETE FROM table ...	X	RX
SELECT ... FROM table ... FOR UPDATE OF ...	X	RS
LOCK TABLE table IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X
	X: 排他 RX: 行排他	RS: 行共有 S: 共有 SRX: 共有行排他

問合せのデフォルト・ロック 問合せはデータを読み込むのみであるため、他の SQL 文に対してほとんど干渉しない SQL 文です。INSERT 文、UPDATE 文および DELETE 文には、文の一部として暗黙の問合せが含まれている場合があります。問合せには、次の種類の文が含まれます。

```
SELECT

INSERT . . . SELECT . . . ;

UPDATE . . . ;

DELETE . . . ;

次の文は含まれません。

SELECT . . . FOR UPDATE OF . . . ;
```

次の特性は、FOR UPDATE 句を使用しないすべての問合せに当てはまります。

- 問合せはデータ・ロックを取得しません。そのため、特定の行に対して問合せが実行される場合も含め、問合せが実行されている表を、他のトランザクションが問い合わせで更新することができます。FOR UPDATE 句が指定されていない問合せでは、データ・ロックが取得されないため他の操作はブロックされないため、Oracle ではこの問合せを「非ブロック化問合せ」と呼ぶことがあります。
- 問合せでは、データ・ロックの解除を待つ必要がなく、常に処理を進めることができます。(待ち状態の分散トランザクションがある場合、ごくまれに、問合せにデータ・ロックの待機が必要な場合があります。)

INSERT、UPDATE、DELETE および SELECT ... FOR UPDATE のデフォルト・ロック INSERT 文、UPDATE 文、DELETE 文および SELECT ... FOR UPDATE 文のロックの特性は次のとおりです。

- DML 文を含むトランザクションは、その文の修正対象の行に対して排他行ロックを取得します。ロックしているトランザクションがコミットまたはロールバックされるまで、その他のトランザクションは、ロックされている行の更新や削除は実行できません。
- DML 文を含むトランザクションは、副問合せや暗黙の問合せ（WHERE 句の中にある問合せ）によって選択される行に対して行ロックを取得する必要がありません。DML 文の中の副問合せや暗黙の問合せは、問合せの開始時点での一貫性が保証され、元の DML 文自体の影響を参照しません。
- トランザクション内の問合せは、同じトランザクション内の前の位置にある DML 文によって加えられた変更を参照できますが、そのトランザクションより後で開始されたその他のトランザクションの変更は参照できません。
- DML 文を含むトランザクションは、必要な排他行ロックに加えて、処理の対象となる行を含む表に対して少なくとも行排他表ロックを取得します。トランザクションがその表について、共有ロック、共有行排他ロックまたは排他表ロックをすでに保持している場合、行排他表ロックは取得されません。トランザクションが行共有表ロックをすでに保持している場合、Oracle はこのロックを行排他表ロックに自動的に変換します。

DDL ロック

処理中のデータ・ディクショナリ・ロック（DDL）操作によってスキーマ・オブジェクトが影響を受けたり参照される間、そのオブジェクトの定義は DDL ロックによって保護されます。DDL 文がトランザクションを暗黙のうちにコミットすることに注意してください。たとえば、ユーザーがプロシージャを作成する場合を考えます。そのユーザーの単一文トランザクションのために、Oracle はプロシージャ定義で参照されるすべてのスキーマ・オブジェクトについての DDL ロックを自動的に取得します。その DDL ロックにより、プロシージャのコンパイル完了前に、プロシージャで参照されるオブジェクトの変更または削除が防止されます。

Oracle は、ディクショナリ・ロックを必要とする DDL トランザクションのために、ディクショナリ・ロックを自動的に取得します。ユーザーは DDL ロックを明示的に要求できません。DDL 操作中には、修正や参照の対象となる個々のスキーマ・オブジェクトのみがロックされます。データ・ディクショナリ全体がロックされることはありません。

DDL ロックは、排他 DDL ロック、共有 DDL ロックおよびブレイク可能解析ロックの 3 つに分類されます。

排他 DDL ロック

次の項「共有 DDL ロック」に示されている DDL 操作を除き、ほとんどの DDL 操作では、同じスキーマ・オブジェクトの変更や参照を実行する可能性のある他の DDL 操作によって破壊的な干渉が起きないように、リソースの排他 DDL ロックが必要です。たとえば ALTER TABLE 操作で表に列を追加している間は、DROP TABLE 操作でその表を削除できません。その逆も同様です。

排他 DDL ロックの取得では、別の操作によってすでにそのスキーマ・オブジェクトに対して別の DDL ロックが保持されている場合、その取得は古い DDL ロックが解除されるまで待機し、その後に処理されます。

また、DDL 操作は変更対象のスキーマ・オブジェクトに対する DML ロック（データ・ロック）も取得します。

共有 DDL ロック

ある種の DDL 操作では、競合する DDL 操作によって破壊的な干渉が起きないようにし、かつ一方では類似の DDL 操作についてデータの同時実行性を確保するため、リソースの共有 DDL ロックが必要です。たとえば CREATE PROCEDURE 文の実行時には、その文を含むトランザクションは、参照されるすべての表について共有 DDL ロックを取得します。他のトランザクションは同じ表を参照するプロシージャを同時に作成できるため、同じ表について同時実行の共有 DDL ロックを取得できます。ただし、参照中の表について排他 DDL ロックを取得できるトランザクションはありません。また、参照中の表を変更または削除できるトランザクションはありません。その結果、共有 DDL ロックを保持するトランザクションでは、参照中のスキーマ・オブジェクトの定義がそのトランザクションの存続期間にわたって変化しないことが保証されます。

スキーマ・オブジェクトに対する共有 DDL ロックは、次のような DDL 文に対して取得されます。AUDIT、NOAUDIT、COMMENT、CREATE [OR REPLACE] VIEW/PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/TRIGGER、CREATE SYNONYM および CREATE TABLE（CLUSTER パラメータが含まれていない場合）

ブレーク可能解析ロック

共有プール内の SQL 文（または PL/SQL プログラム・ユニット）は、参照される各スキーマ・オブジェクトについて解析ロックを保持します。参照オブジェクトが変更されたり削除されたりした場合に、対応する共有 SQL 領域を無効にできるようにするため、解析ロックが取得されます。解析ロックは、どのような DDL 操作も拒否せず、矛盾する DDL 操作も許可するためにブレーク（中断）できるため、「ブレーク可能解析ロック」と呼ばれます。

解析ロックは SQL 文の実行の解析フェーズで取得され、共有プールの中にその文の共有 SQL 領域が残っている限り保持されます。

関連項目： [第 20 章「Oracle の依存性の管理」](#)

DDL ロックの存続期間

DDL ロックの存続期間は、DDL ロックの種類によって異なります。排他ロックと共有 DDL ロックは、DDL 文実行の継続中と自動コミット中ずっと存続します。解析ロックは、対応する SQL 文が共有プールに残っている限り存続します。

DDL ロックとクラスタ

クラスタに対する DDL 操作は、クラスタおよびそのクラスタ内のすべての表とスナップショットに対して排他 DDL ロックを取得します。クラスタ内の表やスナップショットに対する DDL 操作は、その表やスナップショットに対する共有 DDL ロックまたは排他 DDL ロックに加えて、そのクラスタに対する共有ロックも取得します。クラスタに対する共有 DDL ロックは、最初の操作の処理中に、別の操作がそのクラスタを削除するのを防止します。

ラッチと内部ロック

ラッチと内部ロックは、内部データベース構造とメモリー構造を保護します。ユーザーは、それらの出現や存続期間を制御する必要がないため、そのいずれもユーザーからはアクセスできません。次の説明内容は、Oracle Enterprise Manager または SQL*Plus の LOCKS モニターと LATCHES モニターについて理解する上で役立ちます。

ラッチ

ラッチは、システム・グローバル領域（SGA）内の共有データ構造を保護するための単純な下位レベルのシリアル化メカニズムです。たとえば、ラッチは現在データベースにアクセスしているユーザーのリストと、バッファ・キャッシュ内のブロックを説明しているデータ構造を保護します。サーバー・プロセスまたはバックグラウンド・プロセスは、これらの構造の1つを操作したり参照する、非常に短い間のみラッチを取得します。ラッチのインプリメンテーション（特に、プロセスがラッチを待機するかどうか、およびどれくらいの時間待機するか）は、オペレーティング・システムに依存します。

内部ロック

内部ロックは、ラッチよりも高水準で複雑なメカニズムであり、いろいろな目的のために機能します。

ディクショナリ・キャッシュ・ロック これらのロックの存続期間は非常に短く、ディクショナリ・キャッシュ内のエントリが修正されたり使用されている間、そのエントリについて保持されます。これらのロックは、解析されている文が矛盾したオブジェクト定義を参照しないことを保証します。

ディクショナリ・キャッシュ・ロックは、共有または排他で保持されます。共有ロックは、解析が完了すると解除されます。排他ロックは、DDL 操作が完了すると解除されます。

ファイルとログの管理ロック これらのロックは様々なファイルを保護します。たとえば、あるロックは制御ファイルを保護し、一度に1つのプロセスのみがそれを変更できるようにします。別のロックは、REDO ログ・ファイルの使用とアーカイブを調整します。データベースを複数インスタンスが共有モードでマウントしたり、1つのインスタンスが排他モードでマウントすることを保証するために、データ・ファイルがロックされます。ファイルとログのロックはファイルの状態を示すため、必然的に長い間保持されます。

ファイルとログのロックは、Oracle Parallel Server を使用している場合は特に重要です。

関連項目： ロックの詳細は、『Oracle8i Parallel Server 管理、配置およびパフォーマンス』を参照してください。

表領域とロールバック・セグメントのロック これらのロックは、表領域とロールバック・セグメントを保護します。たとえば、データベースにアクセスするすべてのインスタンスは、表領域のオンライン / オフラインの状態について一致することが必要です。ロールバック・セグメントは、1つのセグメントに1つのインスタンスしか書き込めないようにロックされています。

明示的（手動）データ・ロック

データの同時実行性、整合性および文レベルの読取り一貫性を確保するために、Oracle は常に自動的にロックを実行します。ただし、デフォルトのロック・メカニズムを置き換えることもできます。デフォルト・ロックの置換えは、次のような状況で有効です。

- アプリケーションで、トランザクション・レベルの読取り一貫性または「リピータブル・リード」が必要な場合。つまり、アプリケーション内の問合せによって作成されるデータが、他のトランザクションによる変更を反映せず、そのトランザクションの存続期間にわたって一貫性を維持する必要がある場合。トランザクション・レベルの読取り一貫性は、明示的ロック、読取り専用トランザクションまたはシリアル化が可能なトランザクションを使用するか、デフォルトのロックを変更すると実現できます。
- アプリケーションで、あるトランザクションが他のトランザクションの完了まで待機せずに済むように、そのトランザクションがリソースに排他アクセスできるようにする必要があります。

Oracle の自動ロックは、次の 2 つのレベルで置換できます。

トランザクション Oracle のデフォルト・ロックは、次の SQL 文を含むトランザクションによって上書きされます。

- SET TRANSACTION ISOLATION LEVEL 文
- LOCK TABLE 文（表をロックする文、またはビューを使用するときにその基礎となるベース表をロックする文）
- SELECT... FOR UPDATE 文

これらの文で取得されたロックは、トランザクションがコミットまたはロールバックされた後で解除されます。

セッション セッションでは、ALTER SESSION 文によって必要なトランザクション分離レベルを設定できます。

注意： Oracle のデフォルト・ロックを任意のレベルで置き換える場合、データベース管理者やアプリケーション開発者は、ロック置換の手順が確実に正しく実行されるようにしてください。ロックの手順は、データの整合性が保証される、データの同時実行性が許容範囲内である、デッドロックは発生する可能性がない、または適切に処理されるなどの基準を満たすものにする必要があります。

関連項目： SQL 文 LOCK TABLE および SELECT ... FOR UPDATE の詳細は、『Oracle8i SQL リファレンス』を参照してください。

明示的ロックの下でのデータ同時実行性の例

LOCK TABLE 文や FOR UPDATE 句を指定した SELECT 文が使用されるときに、Oracle がデータの同時実行性、整合性および一貫性をどのように維持するかを次に示します。

注意： 簡単にするため、ORA-00054 のメッセージ・テキストは記載しません。そのメッセージ・テキストは「リソースビジー、NOWAIT が指定されていました。」というものです。ユーザーが入力するテキストは太字になっています。

トランザクション 1	時点	トランザクション 2
LOCK TABLE scott.dept IN ROW SHARE MODE; Statement processed	1	
	2	DROP TABLE scott.dept; DROP TABLE scott.dept * ORA-00054 (exclusive DDL lock not possible because of T1's table lock)
	3	LOCK TABLE scott.dept IN EXCLUSIVE MODE NOWAIT; ORA-00054
	4	SELECT LOC FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 row selected
UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (waits because T2 has locked same rows)	5	

トランザクション 1	時点	トランザクション 2
	6	ROLLBACK; (releases row locks)
1 row processed. ROLLBACK;	7	
LOCK TABLE scott.dept IN ROW EXCLUSIVE MODE; Statement processed.	8	
	9	LOCK TABLE scott.dept IN EXCLUSIVE MODE NOWAIT; ORA-00054
	10	LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	11	LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	12	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; 1 row processed.
	13	ROLLBACK;
SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 row selected.	14	
	15	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (waits because T1 has locked same rows)

トランザクション 1	時点	トランザクション 2
ROLLBACK;	16	
	17	1 row processed. (conflicting locks were released) ROLLBACK;
LOCK TABLE scott.dept IN SHARE MODE Statement processed	18	
	19	LOCK TABLE scott.dept IN EXCLUSIVE MODE NOWAIT; ORA-00054
	20	LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	21	LOCK TABLE scott.dept IN SHARE MODE; Statement processed.
	22	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 row selected.
	23	SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 row selected.
	24	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (waits because T1 holds conflicting table lock)

トランザクション 1	時点	トランザクション 2
ROLLBACK;	25	
	26	1 row processed. (conflicting table lock released) ROLLBACK;
LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE; Statement processed.	27	
	28	LOCK TABLE scott.dept IN EXCLUSIVE MODE NOWAIT; ORA-00054
	29	LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	30	LOCK TABLE scott.dept IN SHARE MODE NOWAIT; ORA-00054
	31	LOCK TABLE scott.dept IN ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	32	LOCK TABLE scott.dept IN SHARE MODE NOWAIT; ORA-00054
	33	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 row selected.

トランザクション 1	時点	トランザクション 2
	34	<code>SELECT loc</code> <code>FROM scott.dept</code> <code>WHERE deptno = 20</code> <code>FOR UPDATE OF loc;</code> LOC - - - - - DALLAS 1 row selected.
	35	<code>UPDATE scott.dept</code> <code>SET loc = 'NEW YORK'</code> <code>WHERE deptno = 20;</code> <i>(waits because T1 holds conflicting table lock)</i>
<code>UPDATE scott.dept</code> <code>SET loc = 'NEW YORK'</code> <code>WHERE deptno = 20;</code> <i>(waits because T2 has locked same rows)</i>	36	<i>(deadlock)</i>
Cancel operation <code>ROLLBACK;</code>	37	
	38	1 row processed.
<code>LOCK TABLE scott.dept</code> <code>IN EXCLUSIVE MODE;</code>	39	
	40	<code>LOCK TABLE scott.dept</code> <code>IN EXCLUSIVE MODE;</code> ORA-00054
	41	<code>LOCK TABLE scott.dept</code> <code>IN ROW EXCLUSIVE MODE</code> <code>NOWAIT;</code> ORA-00054
	42	<code>LOCK TABLE scott.dept</code> <code>IN SHARE MODE;</code> ORA-00054
	43	<code>LOCK TABLE scott.dept</code> <code>IN ROW EXCLUSIVE</code> <code>MODE NOWAIT;</code> ORA-00054

トランザクション 1	時点	トランザクション 2
	44	LOCK TABLE scott.dept IN ROW SHARE MODE NOWAIT; ORA-00054
	45	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 row selected.
	46	SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; (waits because T1 has conflicting table lock)
UPDATE scott.dept SET deptno = 30 WHERE deptno = 20; 1 row processed.	47	
COMMIT;	48	
	49	0 rows selected. (T1 released conflicting lock)
SET TRANSACTION READ ONLY;	50	
SELECT loc FROM scott.dept WHERE deptno = 10; LOC - - - - - BOSTON	51	
	52	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 10; 1 row processed.

トランザクション 1	時点	トランザクション 2
SELECT loc FROM scott.dept WHERE deptno = 10; LOC - - - - - BOSTON (T1 does not see uncommitted data)	53	
	54	COMMIT;
SELECT loc FROM scott.dept WHERE deptno = 10; LOC - - - - - (same results seen even after T2 commits)	55	
COMMIT;	56	
SELECT loc FROM scott.dept WHERE deptno = 10; LOC - - - - - NEW YORK (committed data is seen)	57	

Oracle のロック・マネージメント・サービス

アプリケーションの開発者は、Oracle のロック・マネージメント・サービスを使用して次のような処理をする文を PL/SQL ブロックに含めることができます。

- 特定のタイプのロックを要求します。
- そのロックに、同一のインスタンスまたは別のインスタンス内にある別のプロシージャからも識別できる、一意の名前を指定します。
- ロック・タイプを変更します。
- ロックを解除します。

確保したユーザー・ロックは、Oracle ロックと同一とみなされるため、デッドロックの検出などの Oracle ロックの機能をすべて備えています。ユーザー・ロックは接頭辞「UL」で識別されるため、Oracle ロックと矛盾することはありません。

Oracle のロック・マネージメント・サービスは、DBMS_LOCK パッケージ内のプロシージャを介して利用できます。

関連項目： Oracle ロック・マネージメント・サービスの詳細は、『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。

データの整合性

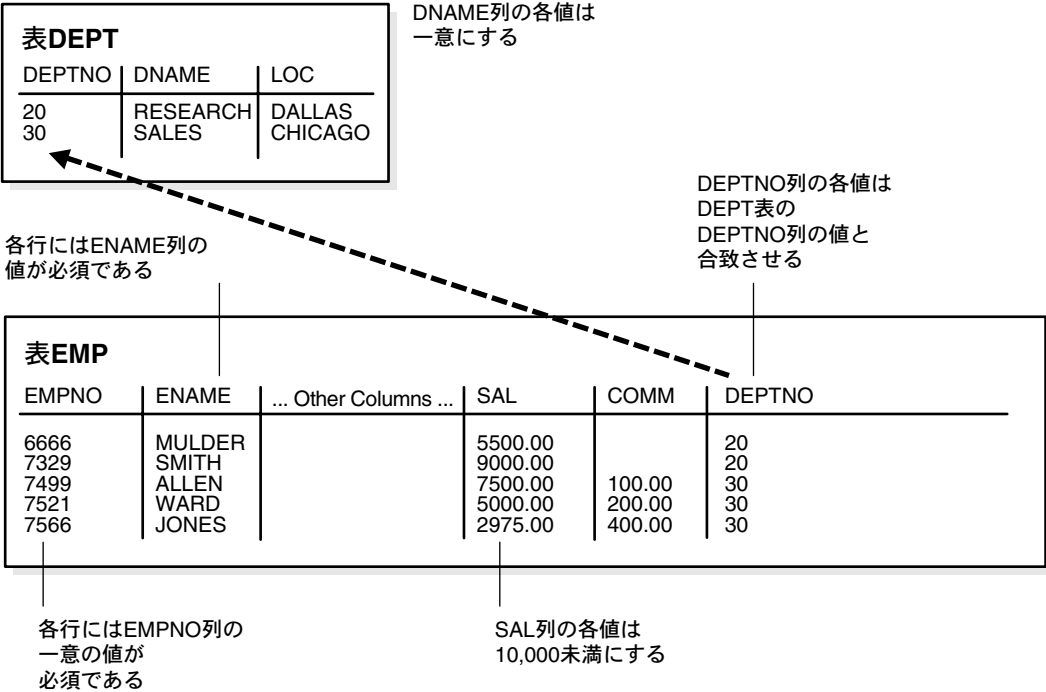
この章では、データベースに関連するビジネス・ルールを規定し、表への無効な情報入力を防止する方法について説明します。この章の内容は、次のとおりです。

- [データ整合性の概要](#)
- [整合性制約の概要](#)
- [整合性制約のタイプ](#)
- [制約チェックのメカニズム](#)
- [遅延制約チェック](#)
- [制約の状態](#)

データ整合性の概要

データが、データベース管理者やアプリケーションの開発者によって事前に定義された規則に準拠しているのは重要なことです。データの整合性の例として、図 25-1 に示す EMP 表と DEPT 表、およびこれらの各表の情報に関するビジネス・ルールについて考えてみましょう。

図 25-1 データ整合性の例



それぞれの表のある列には、その列に入っているデータに制約をかける固有の規則があることに注意してください。

データ整合性のタイプ

この項では、様々な種類のデータの整合性を規定するために表の列に適用できる規則について説明します。

NULL

NULL は、単一の列に対して定義される規則で、その列に NULL が入っている（値がない）行の挿入や更新を許可または禁止します。

一意の列値

列（または列の集合）に対して定義される一意の列値は、その列（または列の集合）に含まれる値が一意である場合に限って、行の挿入または更新を許可します。

主キー値

キー（列または列の集合）に対して定義される主キー値は、表の中の各行がキーの値によって一意に識別できることを指定します。

参照整合性

1 つの表のキー（列または列の集合）に対して定義される規則であり、そのキーの値が関連する表のキーの値（参照値）と一致することを保証します。

また、参照整合性には、参照先のデータに対してどのようなタイプのデータ操作を許可するか、およびその操作の結果として依存データがどのような影響を受けるかについて指示する規則が含まれています。参照整合性に関連する規則は、次のとおりです。

RESTRICT (制限)	参照先のデータの更新または削除を禁止します。
SET TO NULL (NULL 設定)	参照先のデータが更新または削除されると、対応する依存データがすべて NULL に設定されます。
SET TO DEFAULT (デフォルト設定)	参照先のデータが更新または削除されると、対応する依存データがすべてデフォルト値に設定されます。
CASCADE (カスケード)	参照先のデータが更新されると、対応するすべての依存データもそれに応じて更新され、参照先の行が削除されると、対応するすべての依存行も削除されます。
No Action	参照先のデータの更新または削除を禁止します。これは、文の最後にチェックされるか、または制約が遅延されている場合はトランザクションの最後にチェックされるという点が RESTRICT とは異なります。(Oracle はデフォルト・アクションとして No Action を使用します。)

複雑な整合性チェック

複雑な整合性チェックは、列（または列の集合）に対して設定されるユーザー定義の規則であり、その列の値に基づいて、行の挿入、更新、削除を許可または禁止します。

Oracle がデータの整合性を規定する方法

Oracle では、前述のデータ整合性規則をそれぞれ定義し、規定できます。これらの規則のほとんどは、整合性制約またはデータベース・トリガーを使用して簡単に定義できます。

整合性制約

整合性制約は、表の列に規則を定義する宣言手法です。Oracle では、次の整合性制約をサポートしています。

- NOT NULL 制約。列に含まれる NULL に関連した規則です。
- UNIQUE キー制約。一意の列値に関連した規則です。
- PRIMARY KEY 制約。1 次識別値に関連した規則です。
- FOREIGN KEY 制約。参照整合性に対応付けられた規則です。現在、Oracle では、次の参照整合性アクションを定義する際に FOREIGN KEY 整合性制約の使用をサポートしています。
 - UPDATE No Action と DELETE No Action
 - DELETE CASCADE
 - DELETE SET NULL
- CHECK 制約。複雑な整合性規則の場合に使用します。

注意： 子表と親表が分散データベースの別のノードにある場合、宣言整合性制約を使用して参照整合性の規定はできません。ただし、データベース・トリガーを使用して、分散データベースで参照整合性を規定することはできます（後述）。

データベース・トリガー

Oracle では、データベース・トリガー（挿入、更新または削除の各操作で自動的に起動されるストアド・データベース・プロシージャ）を使用することによって、宣言以外の形の整合性制約を規定できます。

関連項目： データの整合性の規定に使用されるトリガーの例は、[第 19 章「トリガー」](#)を参照してください。

整合性制約の概要

Oracle では、データベースのベース表に無効なデータが入力されないようにするため、整合性制約を使用します。整合性制約を定義することによって、データベースの情報に関連付けるビジネス・ルールを規定できます。DML 文を実行した結果が整合性制約に違反すると、Oracle はその文をロールバックし、エラーを戻します。

注意： ビュー（および表のシノニム）に対する操作は、基礎となるベース表に定義されている整合性制約に従います。

たとえば、EMP 表の SAL 列に整合性制約を定義するとします。この表では、どの行でもこの列に 10,000 より大きい数値を入れないという規則を、整合性制約として規定します。INSERT 文または UPDATE 文がこの整合性制約に違反すると、Oracle はその文をロールバックし、通知エラー・メッセージを戻します。

Oracle で実装されている整合性制約は、ANSI X3.135-1989 と ISO 9075-1989 の標準規格に完全に準拠しています。

整合性制約の利点

この項では、整合性制約が他の代替方法よりも優れている点をいくつか説明します。代替方法には次のようなものがあります。

- データベース・アプリケーションのコードでビジネス・ルールを規定します。
- ストアド・プロシージャを使用してデータへのアクセスを完全に制御します。
- トリガーされるストアド・データベース・プロシージャを使用してビジネス・ルールを規定します。

関連項目： [第 19 章「トリガー」](#)

宣言の容易さ

整合性制約は、SQL 文を使用して定義します。表を定義したり変更したりするときに追加のプログラミングをする必要はありません。SQL 文は簡単に記述でき、プログラミング・エラーが少なくすみ、Oracle が整合性制約の機能を制御します。これらの理由で、宣言整合性制約は、アプリケーション・コードやデータベース・トリガーよりも優れています。また、ストアド・プロシージャを使用してデータ・アクセスを制御することによってデータの整合性を解決するという方法もありますが、整合性制約の場合は非定型のデータ・アクセスという柔軟性を犠牲にすることがないため、宣言アプローチを使用するほうがストアド・プロシージャより優れています。

規則の集中化

整合性制約は、表に対して（アプリケーションに対してではなく）定義され、データ・ディクショナリに格納されます。どのアプリケーションから入力されるデータも、表に対応付けられている同じ整合性制約を遵守する必要があります。ビジネス・ルールをアプリケーション・コードから集中管理された整合性制約に移行することにより、どのデータベース・アプリケーションが情報を操作したとしても、データベース表には有効なデータが入っていることが保証されます。ストアド・プロシージャには、集中管理された規則を表に格納する場合のような利点がありません。また、データベース・トリガーでは、規則を集中管理できるという利点がありますが、それを実現する方法は、整合性制約で使用されている宣言アプローチよりはるかに複雑です。

アプリケーション開発の生産性の最大化

整合性制約によって規定されるビジネス・ルールを変更する場合、管理者が整合性制約を変更するだけで、すべてのアプリケーションは変更後の制約を自動的に遵守するようになります。それに対して、それぞれのデータベース・アプリケーションのコードでビジネス・ルールを規定する場合、開発者はすべてのアプリケーションのソース・コードを修正して、その修正したアプリケーションを再コンパイルし、デバッグしてテストする必要があります。

ユーザーへの即時フィードバック

Oracle は、各整合性制約ごとに、特定の情報をデータ・ディクショナリに格納します。Oracle が SQL 文を実行しチェックする前でも、データベース・アプリケーションは、その情報を使用して整合性制約の違反をユーザーに即時にフィードバックするように、データベース・アプリケーションを設計できます。たとえば SQL*Forms アプリケーションは、文を発行する前でも、データ・ディクショナリに格納されている整合性制約の定義を使用して、フォームのフィールドに入力される値の違反をチェックできます。

優れたパフォーマンス

整合性制約宣言の意味は明確に定義されており、個々の宣言規則ごとにパフォーマンスの最適化が実現されます。Oracle 問合せオプティマイザは、宣言を使用して、データをより詳細に認識し、問合せのパフォーマンスを全体として向上させます。（また、アプリケーション・コードとデータベース・トリガーから整合性規則を取り去ることによって、必要なときのみチェックが行われることが保証されます。）

データ・ロードの柔軟性と整合性違反の識別

大量のデータをロードする場合、制約チェックによるオーバーヘッドをなくすために、整合性制約を一時的に使用禁止にできます。データのロードが完了した後、整合性制約を使用可能にするのは簡単であり、整合性制約に違反した新しい行を別の例外表に自動的に報告させることもできます。

整合性制約のパフォーマンス・コスト

データ整合性の規則を規定することによる利点は、パフォーマンスを多少犠牲にして初めて獲得できます。一般に、整合性制約を組み込むことの「コスト」は、多くても、制約を評価する SQL 文を実行するのと同じです。

整合性制約のタイプ

列値の入力時の制限として課すことのできる整合性制約には、次のものがあります。

- NOT NULL 整合性制約
- UNIQUE キー整合性制約
- PRIMARY KEY 整合性制約
- 参照整合性制約
- CHECK 整合性制約

NOT NULL 整合性制約

デフォルトでは、表のすべての列で NULL を使用できます。「NULL」は、値がないことを意味します。NOT NULL 制約がある場合、表の列値が NULL でないということが求められます。たとえば、EMP 表のすべての行で ENAME 列に必ず値を入力するように求める NOT NULL 制約を定義できます。

図 25-2 に、NOT NULL 整合性制約を示します。

図 25-2 NOT NULL 整合性制約

表EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9,000.00		20
7499	ALLEN	VP_SALES	7329	20-FEB-90	7,500.00	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	5,000.00	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90	2,975.00	400.00	30

NOT NULL CONSTRAINT
(行のこの列にはNULLを
使用できない)

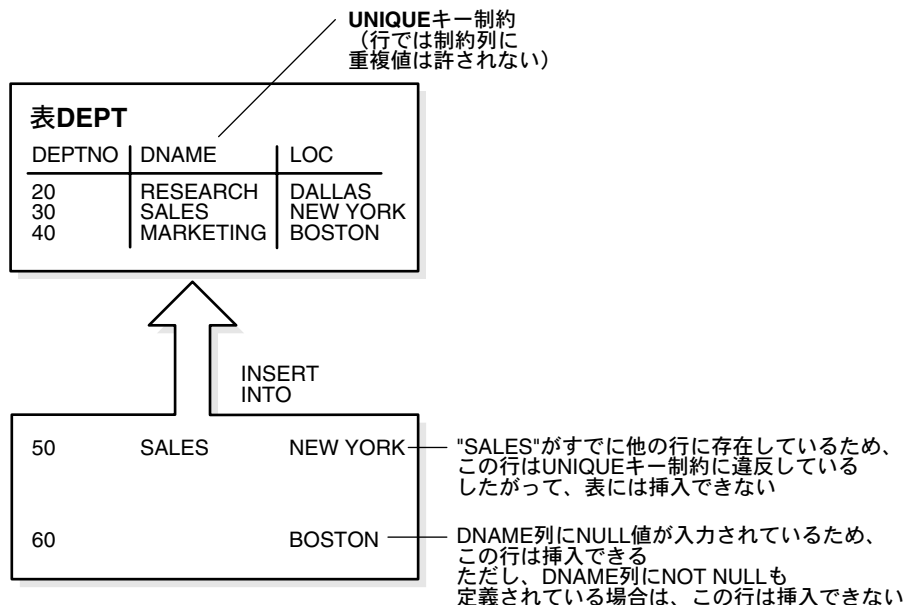
NOT NULL制約なし
(どの行でもこの列に
NULLを使用できる)

UNIQUE キー整合性制約

UNIQUE キー整合性制約では、列または列の集合（キー）のすべての値が一意である必要があります。つまり、指定した列または列の集合について、表の 2 つの行の値が重複することは許されません。

たとえば、図 25-3 では、DEPT 表の DNAME 列に UNIQUE キー制約が定義されており、複数の行での部門名の重複を禁止しています。

図 25-3 UNIQUE キー制約

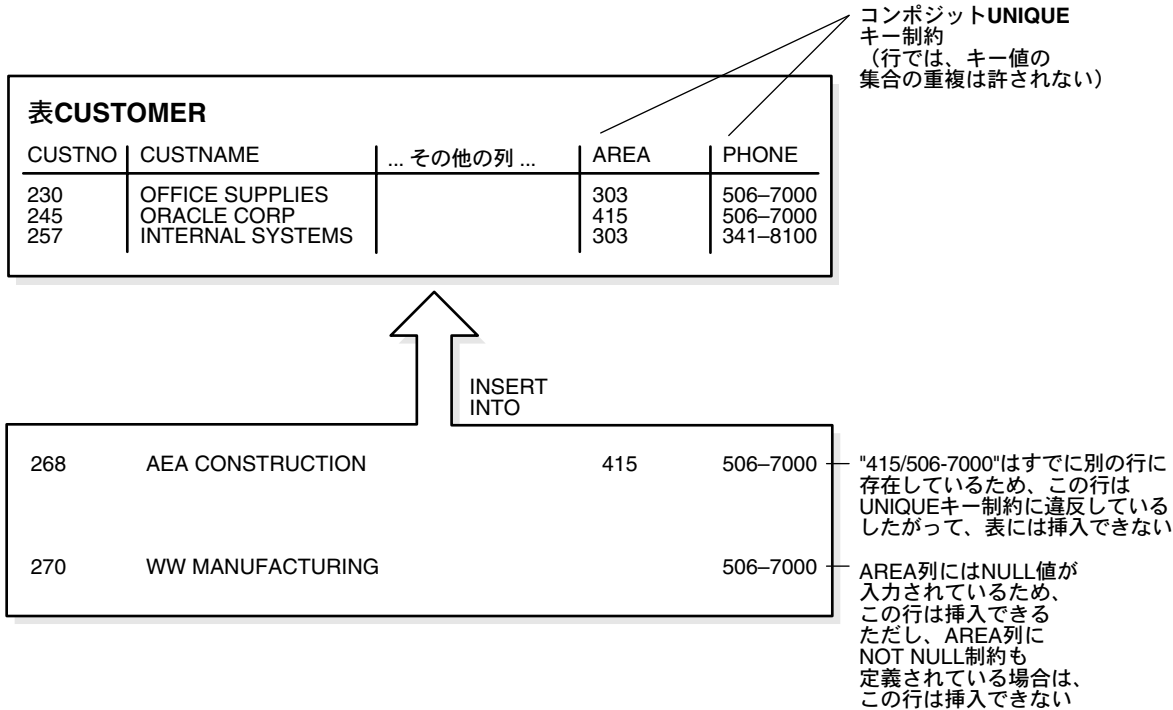


一意キー

UNIQUE キー制約の定義に含まれている列を、「一意キー」と呼びます。「一意キー」という用語は、誤って「UNIQUE キー制約」や「UNIQUE 索引」の同義語として使用されることがありますが、「キー」という用語は整合性制約の定義に使用されている列または列の集合のみを指して使用されるため、注意してください。

UNIQUE キーが2つ以上の列で構成されている場合、列のそのグループのことを「コンポジット一意キー」と呼びます。たとえば、図 25-4 の例では、CUSTOMER 表のコンポジット一意キー (AREA 列と PHONE 列) に対して UNIQUE キー制約が定義されています。

図 25-4 コンポジット UNIQUE キー制約



この場合の UNIQUE キー制約では、同じ市外局番と電話番号を何回でも入力できますが、市外局番と電話番号の特定の「組合せ」を同じ表の中に重複しては入力できません。これにより、誤って電話番号が重複するのを避けられます。

UNIQUE キー制約と索引

Oracle は、索引を使用して一意の整合性制約を規定します。たとえば、図 25-4 で、Oracle は、コンポジット一意キーに対する一意索引を暗黙的に作成することにより、UNIQUE キー制約を規定しています。したがって、コンポジット UNIQUE キー制約には、コンポジット索引に対して課されているのと同じ制限があります。コンポジット一意キーを構成する最大列数は 32 個で、キー値の合計バイト数は、データベースのブロック・サイズの約半分を超えることはできません。UNIQUE キー制約が作成されたときに使用できる索引がある場合、制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

UNIQUE キー整合性制約と NOT NULL 整合性制約を組み合わせる

図 25-3 と図 25-4 では、同じ列に NOT NULL 制約も定義するのでない限り、UNIQUE キー制約は NULL の入力を許可します。実際、NULL はどの値とも等しいとみなされることがないため、NOT NULL 制約のない列では、その列が NULL である行が何行存在してもかまいません。1 つの列に NULL がある（またはコンポジット UNIQUE キーのすべての列に NULL がある）場合は、UNIQUE キー制約はいつも満たされます。

一意キーと NOT NULL 整合性制約の両方が指定された列は、一般的に使用されます。これらの組合せにより、ユーザーは一意キーに必ず値を入力することになり、さらに新しい行データが既存の行データと衝突すること也不再行します。

注意： 2 つ以上の列に対する UNIQUE 制約の検索メカニズムにより、一部が NULL のコンポジット UNIQUE キー制約の非 NULL 列で同一の値は許されません。

PRIMARY KEY 整合性制約

データベースのそれぞれの表には、最大 1 つの PRIMARY KEY 制約を指定できます。この制約が指定されている 1 つ以上の列グループの値は、その行の一意識別子を構成します。つまり、それぞれの行は、その主キー値によって指定されます。

Oracle に実装されている PRIMARY KEY 整合性制約では、次の 2 つのことが保証されます。

- 表の指定された列または列の集合の中に、2 つの行が重複する値を持つことはありません。
- 主キー列では、NULL は許可されません。つまり、それぞれの行の主キー列には値が必ず存在することが必要です。

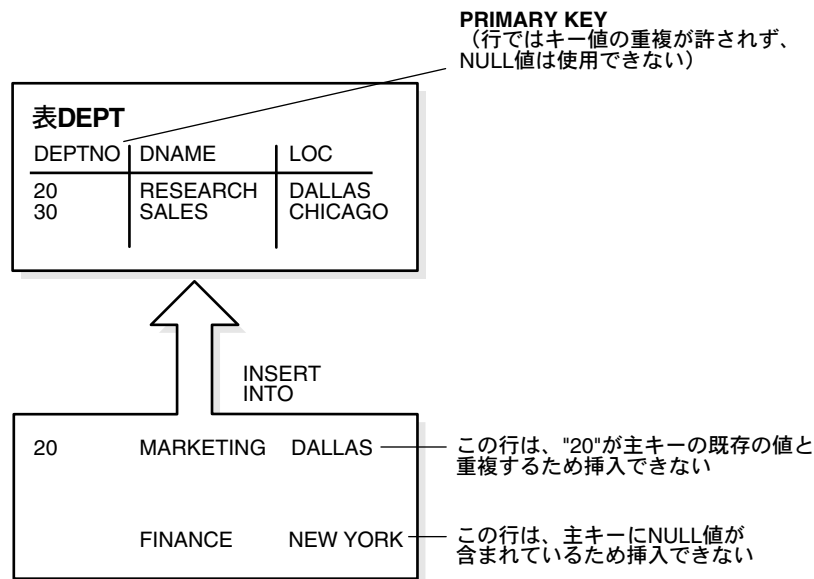
主キー

表の PRIMARY KEY 整合性制約の定義に含まれている列を、「主キー」と呼びます。主キーは必須ではありませんが、次の利点を考慮して、すべての表に主キーを指定してください。

- 表のそれぞれの行を一意に識別できます。
- 重複する行が表に存在しません。

図 25-5 に、DEPT 表での PRIMARY KEY 制約と、制約に違反する行の例を示します。

図 25-5 主キー制約



PRIMARY KEY 制約と索引

Oracle では、すべての PRIMARY KEY 制約が索引を使用して規定されます。図 25-5 では、DEPTNO 列に対して作成された主キー制約は、次のものの暗黙的な作成によって規定されます。

- その列に対して一意の索引
- その列に対する NOT NULL 制約

Oracle は、索引を使用して主キー制約を規定し、コンポジット主キー制約は、コンポジット索引に課されるのと同じ 32 以下の列という制限を受けます。この索引の名前は、制約の名前と同じ名前です。また、制約を作成に使用する CREATE TABLE 文または ALTER TABLE 文に ENABLE 句を組み込んで、この索引に格納オプションを指定することもできます。主キー制約の作成時に使用できる索引がある場合、主キー制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

参照整合性制約

リレーショナル・データベースの中の異なる表は共通の列で関連付けることができ、列の関係を管理する規則が守られていることが必要です。参照整合性規則により、これらの関係が保たれることが保証されます。

参照整合性制約に関連する用語は、次のとおりです。

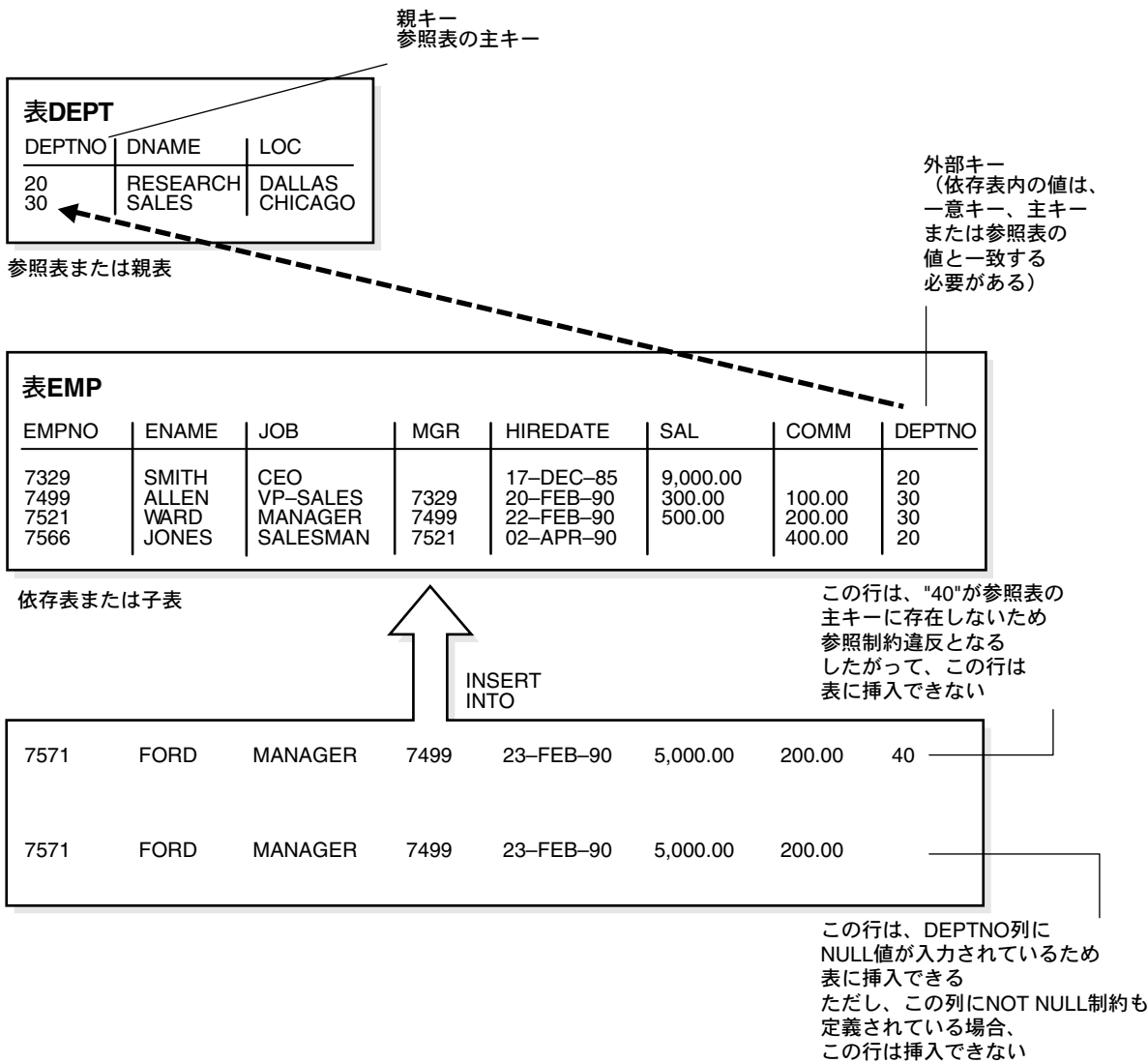
外部キー	参照整合性制約の定義に含まれている列または列の集合のうち、参照キーを参照するもの。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。

参照整合性制約では、表の各行の外部キー値は親表の値と一致している必要があります。

図 25-6 に、EMP 表の DEPTNO 列に対して定義された外部キーを示します。これにより、この列のすべての値が DEPT 表の主キー（すなわち DEPTNO 列）の値と一致することが保証されます。このため、EMP 表の DEPTNO 列に間違った部門番号が存在することはありません。

外部キーは、複数列として定義できます。ただし、コンポジット外部キーの列数とデータ型は、参照先のコンポジット主キーまたは一意キーと同じであることが必要です。コンポジット主キーおよび一意キーは 32 列までに制限されているため、コンポジット外部キーも 32 列までに制限されます。

図 25-6 参照整合性制約

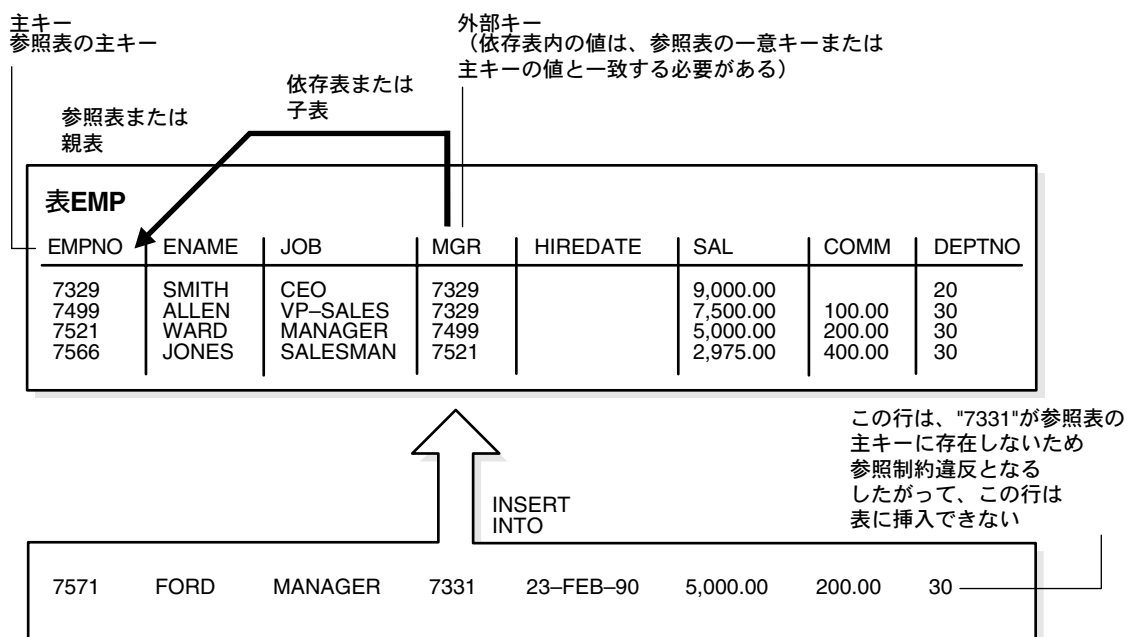


自己参照型整合性制約

図 25-7 に示す別のタイプの参照整合性制約のことを、自己参照型整合性制約と呼びます。このタイプの外部キーは、同じ表の親キーを参照します。

図 25-7 の例では、参照整合性制約によって、EMP 表の MGR 列のすべての値は、必ずしも同じ行でなくても、同じ表の EMPNO 列に現在存在する値と一致することが保証されます。これは、すべての管理職は従業員でもある必要があるためです。この整合性制約により、間違った従業員番号が MGR 列に存在する可能性はなくなります。

図 25-7 単一表の参照制約



NULL と外部キー

リレーショナル・モデルでは、外部キーの値が、参照主キーか一意キーの値、または NULL のどちらかに一致することが許されています。コンポジット（複数列）外部キーが関係している場合、リレーショナル・モデルのこの基本規則には、いくつかの解釈が可能になります。

ANSI/ISO SQL92（エントリ・レベル）規格では、コンポジット外部キーの非 NULL 列には、それ以外の列が NULL であるなら、任意の値を入れることができます。その非 NULL 列の値が参照キーになくてもかまいません。一部が NULL になっている外部キー処理は、NOT NULL 制約、CHECK 制約など、他の制約を使用してデフォルトの処理から変更できます。

コンポジット外部キーの値としては、NULL、すべて非 NULL または一部 NULL という値を指定できます。次の用語は、コンポジット外部キーに関する 3 つの互いに異なる一致規則を定義するものです。

完全一致	一部が NULL の外部キーは許可されません。外部キーのすべての要素が NULL であるか、または外部キーの値の組合せが、参照表の単一行の主キーまたは一意キーの値と一致している必要があります。
部分一致	一部が NULL のコンポジット外部キーが許可されます。外部キーのすべての要素が NULL、または外部キーの NULL 以外の値の組合せが、参照表の単一行の主キーまたは一意キーの対応部分と一致している必要があります。
不一致	一部が NULL のコンポジット外部キーが許可されます。コンポジット外部キーのいずれかの列が NULL の場合、そのキーの NULL 以外の部分は、親キーの対応部分と一致している必要はありません。

参照整合性制約によって定義されるアクション

参照整合性制約では、参照先の親キー値が修正された場合に子表の依存行に対して実行される特定のアクションを指定できます。Oracle の FOREIGN KEY 整合性制約によってサポートされる参照アクションは、UPDATE No Action、DELETE No Action および DELETE CASCADE です。

注意： Oracle の FOREIGN KEY 整合性制約でサポートされていない他の参照アクションは、データベース・トリガーを使用して規定できます。

詳細は、[第 19 章「トリガー」](#)を参照してください。

UPDATE No Action と DELETE No Action No Action（デフォルト）オプションは、結果のデータが参照整合性制約に違反する場合に参照キー値を更新または削除できないことを指定します。たとえば、主キー値が外部キーの中の値によって参照されている場合は、依存データであるため、参照先の主キー値を削除できません。

DELETE Cascade 参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行も削除（DELETE CASCADE）されます。たとえば、親表の行が削除され、この行の主キー値が子表の1つ以上の外部キー値によって参照されている場合は、子表の中のこの主キー値を参照する行も子表から削除されます。

DELETE Set Null 参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行の値が NULL に設定（SET NULL）されます。たとえば、EMPNO が TMP 表内の MGR を参照する場合は、管理者を削除すると、その管理者の部下である従業員全員の行の MGR 値が NULL に設定されます。

参照アクションに関する DML 制限 表 25-1 に、親表の主キー値または一意キー値および子表の外部キー値に対する異なる参照アクションごとに可能な DML 文の概要を示します。

表 25-1 UPDATE No Action と DELETE No Action で許される DML 文

DML 文	親表に対して発行	子表に対して発行
INSERT	親キー値が一意であれば常に発行できます。	外部キーの値が親キーに存在するか、外部キーの一部またはすべてが NULL の場合にのみ発行できます。
UPDATE No Action	文の実行後に、参照される親キー値のない行が子表内に残らない場合は発行できます。	文の実行後も新しい外部キー値によって参照キー値が参照される場合は発行できます。
DELETE No Action	子表のどの行も親キー値を参照していない場合は発行できます。	常に発行できます。
DELETE Cascade	常に発行できます。	常に発行できます。
DELETE Set Null	常に発行できます。	常に発行できます。

同時実行性の制御、索引および外部キー

外部キーには、ほぼ常に索引を付ける必要があります。唯一の例外は、対応する一意キーまたは主キーの更新や削除が発生しないことが確実な場合です。

Oracle は、親キーと依存している外部キー値に関して、同時実行性の制御を最適化します。両者の関連性の維持に使用される同時実行性のためのメカニズムを制御できるので、状況によっては、これが大きな利点をもたらすことがあります。ここでは、考えられる状況と個々の推奨事項について説明します。

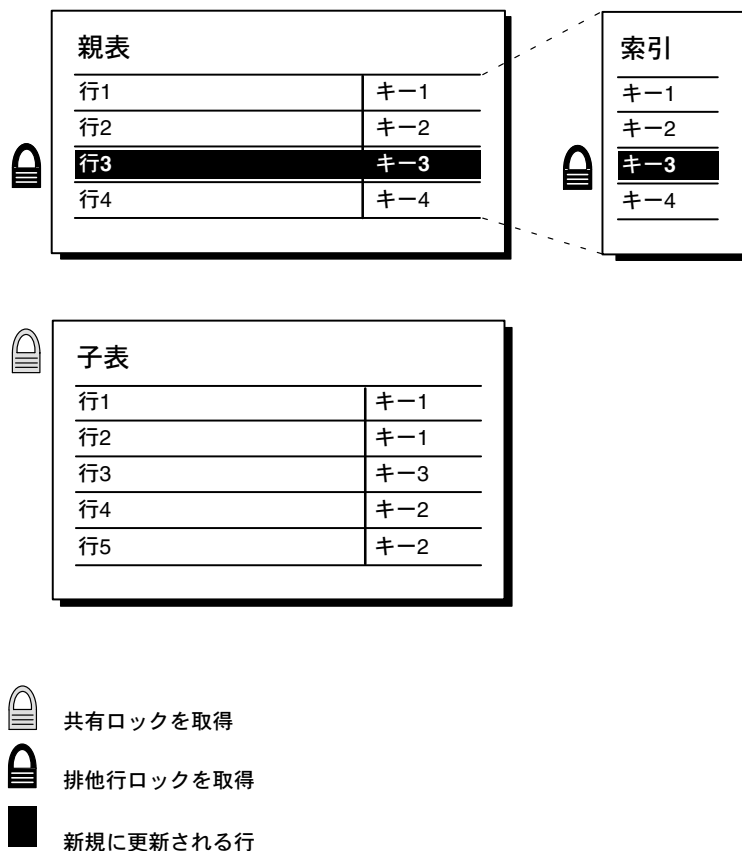
外部キーの索引がない場合 図 25-8 は、外部キーの索引が定義されていない場合と、親表の行が更新または削除される場合に、Oracle で使用されるロック・メカニズムを示しています。親表に対する挿入の場合、子表のロックは必要ありません。

親表に対する DELETE 文を含むトランザクションがコミットされるまでは、子表全体の共有ロックが必要なことに注意してください。外部キーで ON DELETE CASCADE が指定されている場合、DELETE 文では子表に表レベルの共有副排他ロックが発生します。また、子表で参照される列に影響する親表に対して UPDATE 文を発行する場合も、子表全体の共有ロックが必要です。共有ロックでは読みみしかできないため、UPDATE または DELETE を含むトランザクションがコミットされるまでは、子表に対して INSERT、UPDATE または DELETE 文は発行できません。子表に対する問合せはできます。

この状況は、親表に対する更新と削除を回避できる場合は問題はありません。

子表に対する INSERT、UPDATE および DELETE 文は親表のロックを取得しませんが、INSERT 文と UPDATE 文は親表の索引の行ロックが解放されるまで待機します。

図 25-8 外部キーに索引が定義されていない場合のロック・メカニズム

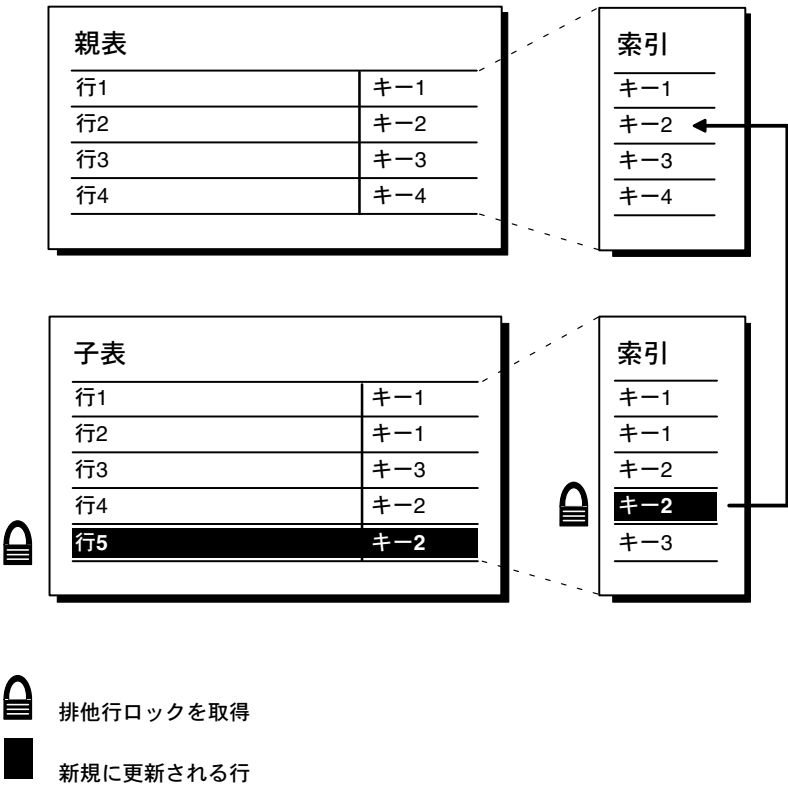


外部キーの索引 図 25-9 は、外部キーの索引が定義されており、子表に新規行が挿入されるか、更新または削除される場合に、Oracle で使用されるロック・メカニズムを示しています。

挿入、更新または削除が発生しても、親表またはその索引については、どんな種類の表ロックも取得されないことに注目してください。したがって、親表には、INSERT、UPDATE、DELETE および問合せなど、任意の DML 文を発行できます。

この状況が望ましいのは、子表に対する更新処理の発生中に、親表に対する更新または削除処理が発生する場合です。親表に対する INSERT、UPDATE および DELETE には子表のロックは必要ありませんが、UPDATE および DELETE は、子表の索引に対する行レベルのロックが解放されるまで待機します。

図 25-9 外部キーに索引が定義されている場合のロック・メカニズム



子表で ON DELETE CASCADE が指定されている場合、親表から削除すると子表からも削除されることがあります。この場合、待機とロックに関するルールは、親表からの DELETE を実行した後に子表から自身を削除した場合と同じです。

CHECK 整合性制約

列または列の集合に対する CHECK 整合性制約では、その表のすべての行について、指定した条件が TRUE または UNKNOWN であることが必要です。DML 文の結果が、CHECK 制約の条件が FALSE に評価される場合、その文はロールバックされます。

チェック条件

CHECK 制約を使用すると、チェック条件を指定することによって、特定の目的の、または洗練された整合性規則を規定できます。CHECK 制約の条件には次のような制限があります。

- 挿入または更新する行の値を使用して評価されるブール式であることが必要です。
- 副問合せ、順序、SQL 関数の SYSDATE、UID、USER または USERENV、あるいは疑似列 LEVEL または ROWNUM を含むことはできません。

文字列リテラルまたは NLS パラメータを引数に指定した SQL ファンクション (TO_CHAR、TO_DATE および TO_NUMBER など) が組み込まれている CHECK 制約を評価する際、デフォルトでは Oracle はデータベースの NLS 設定を使用します。これらのデフォルトは、CHECK 制約定義の中でそれらのファンクションに NLS パラメータを明示的に指定すれば、上書きできます。

関連項目： NLS 機能の詳細は、『Oracle8i NLS ガイド』を参照してください。

複数の CHECK 制約

単一の列に、定義の中でその列を参照する複数の CHECK 制約を指定できます。1 つの列に対して定義できる CHECK 制約の数に制限はありません。

単一の列に対して複数の CHECK 制約を作成する場合は、その目的が競合しないように慎重に設計してください。また、条件が特定の順序で評価されるとは考えないでください。CHECK の条件が矛盾しないかどうかは検証されません。

制約チェックのメカニズム

制約が存在する場合に許可されるアクションのタイプを把握する上で、Oracle が実際に制約をチェックするタイミングを理解しておく役立ちます。このことを具体的に説明するため、いくつかの例を紹介します。この例では、次のような状況を想定します。

- EMP 表は、25-15 ページの図 25-7 で定義されたものです。
- 自己参照型制約によって、MGR 列のエントリが EMPNO 列の値に依存しています。わかりやすくするため、これ以降の説明では、EMP 表の EMPNO 列と MGR 列のみを対象にします。

EMP 表に最初の行を挿入する場合を考えます。現在は行が存在しません。MGR 列が EMPNO 列の既存の値を参照できない場合、どうすれば行を入力できるでしょうか？この操作を実行するには、次の 3 つの方法が考えられます。

- MGR 列に NOT NULL 制約が定義されていない場合には、第 1 行の MGR 列に NULL を入力できます。外部キーには NULL が許されるため、この行は表に正しく挿入されます。
- EMPNO 列と MGR 列の両方に同じ値を入力できます。このことから、Oracle が文の実行完了「後」に制約チェックを実行することがわかります。親キーと外部キーに同じ値を指定した行の入力を許可するために、Oracle は、文を実行（つまり、新しい行を挿入）してから、その新しい行の MGR 列に対応する EMPNO が入っている行がその表に含まれているかどうかをチェックします。
- SELECT 文のネストを伴う INSERT 文など、複数行を挿入する INSERT 文により、相互に参照しあう行を挿入できます。たとえば、最初の行は EMPNO 列が 200 で MGR 列が 300、2 番目の行は EMPNO 列が 300 で MGR 列が 200 という挿入を実行できます。

このことから、制約チェックは文の実行が完了するまで遅延されていることがわかります。すべての行がまず挿入されてから、制約違反がないかどうかすべての行が調べられます。また、トランザクションが完了するまで制約のチェックを遅延させることもできます。

同じ自己参照型の整合性制約に関して、次の使用例を考えます。

会社を買収された。この買収に伴い、すべての従業員番号の現在の設定値に 5000 を加算して、新しい会社の従業員番号と調和させる必要があります。管理職番号は実際には従業員番号であるため、これらの値にも 5000 を加算する必要があります。

この時点では、この表は図 25-10 のような状態になっています。

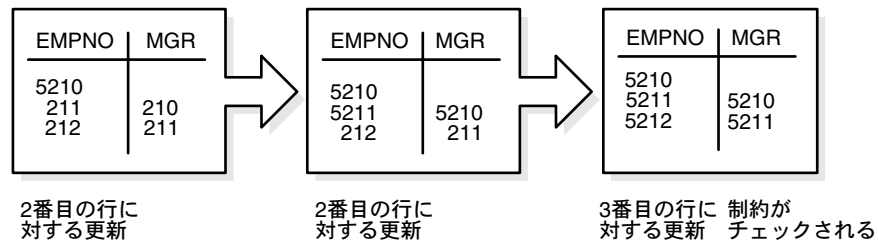
図 25-10 更新前の EMP 表

EMPNO	MGR
210	
211	210
212	211

```
UPDATE emp
  SET empno = empno + 5000,
      mgr = mgr + 5000;
```

制約は、各 MGR 値が EMPNO 値と一致するかどうかを検査するように定義されていますが、Oracle では文の完了後に制約チェックを効率的に実行されるため、この文は有効です。図 25-11 に、制約チェックの前に SQL 文全体のアクションが実行される流れを示します。

図 25-11 制約チェック



この項の例は、INSERT 文と UPDATE 文を実行した場合の制約チェックのメカニズムを示しています。UPDATE 文、INSERT 文および DELETE 文など、すべてのタイプの DML 文でも、同じメカニズムを使用しています。

また、これらの例は、自己参照型整合性制約を使用してチェックのメカニズムを示すものでした。これと同じメカニズムが、次に挙げるすべてのタイプの制約に使用されます。

- NOT NULL
- UNIQUE キー
- PRIMARY KEY
- すべてのタイプの FOREIGN KEY 制約
- CHECK 制約

関連項目： 25-24 ページの「[遅延制約チェック](#)」

デフォルト列値と整合性制約チェック

デフォルト値は、文の解析前に INSERT 文の一部として組み込まれます。このため、デフォルトの列値はすべての整合性制約チェックの対象になります。

遅延制約チェック

制約の妥当性のチェックは、トランザクション終了時まで「遅延」できます。

- 「遅延制約」では、制約が満たされているかどうかのチェックがコミット時にしか実行されません。遅延制約違反があると、コミット時にそのトランザクションはロールバックされます。
- 「即時制約」の場合（つまり遅延制約でない場合）、チェックはそれぞれの文が終わった時点で実行されます。制約違反があると、その文は即時にロールバックされます。

制約によって「アクション」（DELETE CASCADE など）が発生する場合は、遅延制約か即時制約かに関係なく、そのアクションは、アクションを発生させた文の一部として実行されます。

制約の属性

制約は、「遅延可能」または「遅延不可」、および「初期遅延」または「初期即時」のどちらかに定義できます。これらの属性は、制約ごとに異なるものを指定できます。それらの定義は、CONSTRAINT 句の中で次のキーワードを使用して指定します。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE

制約について、追加、削除、使用可能と使用禁止の切替え、または妥当性チェックを実行できます。また、制約の属性も変更できます。

関連項目：

- 制約属性とそのデフォルト値の詳細は、『Oracle8i SQL リファレンス』を参照してください。
- 25-26 ページの「[制約の状態](#)」
- 25-27 ページの「[制約の状態変更](#)」

SET CONSTRAINTS モード

SET CONSTRAINTS 文は、特定のトランザクションのために、制約を DEFERRED または IMMEDIATE のどちらかに指定します（構文的にも意味的にも ANSI SQL92 規格に準拠）。この文を使用すると、制約名のリストまたはすべての制約（ALL）を指定して、そのモードを設定できます。

SET CONSTRAINTS モードは、トランザクションの存続期間中、または別の SET CONSTRAINTS 文によってモードが再設定されるまで有効です。

SET CONSTRAINTS ... IMMEDIATE は、指定された制約を各制約文の実行直後にチェックするように指示します。チェック制約が一貫しており、他に SET CONSTRAINTS 文が発行されない場合、Oracle はまずトランザクション内で事前に遅延されている制約をチェックし、次にそのトランザクション内のそれ以降の文の制約を即時にチェックします。制約チェックが失敗すると、エラーが通知されます。この場合、COMMIT を発行するとトランザクション全体がロールバックされることになります。

ALTER SESSION 文にも、SET CONSTRAINTS IMMEDIATE または DEFERRED の句があります。これらの句では、暗黙的にすべての遅延制約が設定されます（つまり、制約名のリストは指定できません）。これらのオプションを指定することは、カレント・セッションで各トランザクションの最初に SET CONSTRAINTS 文を発行することと同じ意味を持ちます。

COMMIT が成功するかどうかを調べる 1 つの方法は、トランザクションの最後に「即時」制約を設定することです。トランザクションの最後の文で制約を IMMEDIATE に設定すれば、予期しないロールバックを回避できます。いずれかの制約がチェックを通らなかった場合は、エラーを訂正してから、トランザクションをコミットできます。

SET CONSTRAINTS 文は、トリガー内では許可されていません。

SET CONSTRAINTS は分散型の文としても有効です。SET CONSTRAINTS ALL 文が発生すると、処理中のトランザクションがある既存のデータベース・リンクにそのことが知らされ、新しいリンクはトランザクションを開始すると同時にその文が発生したことを認識します。

一意制約と一意索引

あるユーザーのトランザクションで制約の矛盾（一意索引内に重複値が含まれているなど）が生成された場合、そのユーザーにはそのような制約の矛盾がわかります。

スナップショットに対して遅延一意制約および遅延外部キー制約を設定すれば、リフレッシュ操作を高速かつ完全に完了できます。

遅延可能な一意制約は、常に一意でない索引を使用します。遅延可能制約を削除しても、その索引は残ります。制約を使用禁止にしても格納情報は残るため、この方法は便利です。遅延可能でない一意制約と主キーは、制約が規定される前に一意でない索引がキー列に置かれる場合には、一意でない索引も使用します。

制約の状態

CREATE TABLE 文または ALTER TABLE 文を使用すると、整合性制約を表レベルで使用可能または使用禁止にできます。また、次のように、制約を VALIDATE または NOVALIDATE に設定し、ENABLE または DISABLE と組み合わせることもできます。

- ENABLE を指定すると、入ってくるすべてのデータが制約に準拠していることが保証されます。
- DISABLE を指定すると、入ってくるデータは、制約に準拠しているかどうかに関係なく許可されます。
- VALIDATE を指定すると、既存のデータが制約に準拠していることが保証されます。
- NOVALIDATE は、一部の既存データが制約に準拠していない可能性があることを意味します。

さらに、次のようになります。

- ENABLE VALIDATE は ENABLE と同じです。制約がチェックされ、すべての行が保持されることが保証されます。
- ENABLE NOVALIDATE は、制約はチェックされますが、すべての行について TRUE でなくても許されることを意味します。これにより、既存の行が制約に違反することは許され、しかも、新しい行や変更した行に対してはすべて有効であることが保証されます。

ALTER TABLE 文の ENABLE NOVALIDATE オプションを使用すると、表内のすべてのデータの妥当性を最初にチェックせずに、使用禁止にした制約について制約チェックを再開できます。

- DISABLE NOVALIDATE は DISABLE と同じです。制約はチェックされず、TRUE でなくともかまいません。
- DISABLE VALIDATE を指定すると制約が使用禁止になり、制約の索引が削除され、対象となる列の変更は許されなくなります。

UNIQUE 制約の場合、DISABLE VALIDATE 状態では、ALTER TABLE 文の EXCHANGE PARTITION 句を使用して、非パーティション表からパーティション表にデータを効率よくロードできます。

これらの状態間の遷移は、次の規則で制御されます。

- NOVALIDATE が指定されていない限り、ENABLE は暗黙的に VALIDATE を意味します。
- VALIDATE が指定されていない限り、DISABLE は暗黙的に NOVALIDATE を意味します。
- VALIDATE と NOVALIDATE には、ENABLE 状態と DISABLE 状態に関するデフォルトの含意はありません。

- 一意キーまたは主キーが DISABLE 状態から ENABLE 状態に移る場合に、既存の索引がなければ一意キーが自動的に作成されます。同様に、一意キーまたは主キーが ENABLE から DISABLE に移る場合に、一意索引で使用可能にされていれば、一意索引は削除されます。
- 制約が NOVALIDATE 状態から VALIDATE 状態に移った場合は、すべてのデータをチェックする必要があります。（この操作は、きわめて低速の場合があります。）ただし、VALIDATE から NOVALIDATE に移っても、データがチェックされたことが無視されるだけです。
- 単一の制約を ENABLE NOVALIDATE 状態から ENABLE VALIDATE 状態に移動すると、読み込み、書き込みまたは他の DDL 文は阻止されません。パラレルに実行できます。

関連項目： ENABLE、DISABLE、VALIDATE および NOVALIDATE
CONSTRAINT 句の使用方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。

制約の状態変更

ALTER TABLE 文の MODIFY CONSTRAINT 句を使用すると、次の制約の状態を変更できます。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE
- RELY または NORELY
- USING INDEX ...
- ENABLE または DISABLE
- VALIDATE または NOVALIDATE
- EXCEPTIONS INTO ...

関連項目： これらの制約の状態の詳細は、『Oracle8i SQL リファレンス』を参照してください。

データベース・アクセスの制御

この章では、Oracle データベースへのアクセスを制御する方法を説明します。この項の内容は、次のとおりです。

- [データベース・セキュリティの概要](#)
- [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](#)
- [ユーザーの認証](#)
- [ユーザー表領域の設定と割当て制限](#)
- [ユーザー・グループ PUBLIC](#)
- [ユーザー・リソースの制限とプロファイル](#)
- [ライセンス](#)

データベース・セキュリティの概要

データベース・セキュリティには、データベースとデータベースに格納されているオブジェクトに対するユーザー・アクションを許可したり禁止する機能が関係しています。Oracle では、スキーマとセキュリティ・ドメインを使用して、データへのアクセスを制御したり、様々なデータベース・リソースの使用を制限します。

Oracle では、包括的な任意アクセス制御が提供されています。「任意アクセス制御」は、特定のオブジェクトに対するすべてのユーザー・アクセスを、権限によって調整します。権限とは、特定のオブジェクトに規定の方法でアクセスするための許可です。たとえば、表への問合せを実行する許可はその一例です。権限は、他のユーザーの裁量で任意に付与されるものであるため、「任意セキュリティ機能」という語を使用しています。

関連項目： 第 27 章「権限、ロールおよびセキュリティ・ルール」

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

「ユーザー」（「ユーザー名」とも呼ぶ）は、データベース内で定義されている名前であり、この名前を使用してオブジェクトに接続したりアクセスできます。「スキーマ」は、表、ビュー、クラスタ、プロシージャおよびパッケージなどのオブジェクトの集合に名前を付けたものです。スキーマとユーザーは、データベース管理者がデータベース・セキュリティを管理するのに役立ちます。

「エンタープライズ・ユーザー」は、ディレクトリ内で管理されるユーザーで、各データベース内でアカウントやスキーマを作成しなくても、複数のスキーマおよびデータベースへのアクセス権を取得できます。これはユーザーや DBA にとってより簡単であり、その権限を一元的に変更できるため、セキュリティが改善されます。

新たにデータベースを作成したり、既存のデータベースを変更する場合、セキュリティ管理者は、ユーザーのセキュリティ・ドメインについていくつか決定する必要があります。たとえば、次のようなことを決定します。

- ユーザー認証情報を、データベース、オペレーティング・システムまたはネットワークの認証サービスのどれを使用して維持管理するか。
- ユーザーのデフォルト表領域と一時表領域の設定。
- ユーザーがアクセス可能な表領域のリスト（存在する場合）と、そのリストに含まれている表領域に対応付ける割当て制限。
- ユーザーのリソース制限プロファイル。ユーザーが使用できるシステム・リソースの量を制限します。
- データベース操作の実行に必要な、オブジェクトへの適切なアクセス権をユーザーに提供する権限、ロールおよびセキュリティ・ルール。

この章では、このうち最初の4つのセキュリティ・ドメイン・オプションについて説明します。

注意： この章の情報は、ユーザー定義データベースのすべてのユーザーに適用されます。特殊なデータベース・ユーザーである SYS および SYSTEM には適用されません。これらの特殊なユーザーのセキュリティ・ドメインの設定は、絶対に変更しないでください。

関連項目：

- [第 27 章「権限、ロールおよびセキュリティ・ルール」](#)
- エンタープライズ・ユーザーの詳細は、『Oracle8i Advanced Security 管理者ガイド』を参照してください。
- 特別なユーザーである SYS および SYSTEM の詳細と、セキュリティ管理者の詳細は、『Oracle8i 管理者ガイド』を参照してください。

ユーザーの認証

データベース・ユーザー名が無許可で使用されないようにするため、Oracle では次のように複数の異なる方法で標準データベース・ユーザーの妥当性をチェックします。

- オペレーティング・システム
- ネットワーク・サービス
- 対応付けられた Oracle データベース
- ユーザーに代ってトランザクションを実行する中間層アプリケーションの Oracle データベース
- Secure Socket Layer (SSL) プロトコル

通常は、検査を簡素化するため、前述のどれか1つの方法を使用してデータベースのすべてのユーザーを認証します。ただし、Oracle では、同じデータベース・インスタンス内で前述の方法をすべて使用できます。

さらに、Oracle では、ネットワークのセキュリティを確実にするため、送信時にパスワードが暗号化されます。

データベース管理者は特別なデータベース操作を実行するため、データベース管理者に対しては特別な認証手順が必要になります。

オペレーティング・システムによる認証

一部のオペレーティング・システムでは、オペレーティング・システムの維持している情報を、Oracle がユーザーの認証のために使用できます。オペレーティング・システムによる認証の利点は、次のとおりです。

- ユーザーがより簡単に Oracle に接続できます。ユーザー名やパスワードを指定する必要はありません。たとえば、ユーザーは SQL*Plus を起動する際に次のように入力して、ユーザー名とパスワードのプロンプトを省略できます。

```
SQLPLUS /
```

- ユーザーの認可をオペレーティング・システムが集中管理できます。Oracle がユーザー・パスワードを格納したり管理する必要がなくなります。ただし、ユーザー名は Oracle データベース内に保持されます。
- データベースのユーザー名エントリと、オペレーティング・システムの監査証跡が対応します。

データベース・ユーザーの認証にオペレーティング・システムを使用する場合は、分散データベース環境とデータベース・リンクに関していくつかの特別な考慮事項があります。

関連項目：

- [第 30 章「分散データベースの概念」](#)
- オペレーティング・システムによる認証の詳細は、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。

ネットワークによる認証

Oracle は、後述のようにネットワークによる複数の認証方式をサポートしています。

サードパーティベースの認証テクノロジー

ネットワーク認証サービス（DCE、Kerberos または SESAME など）を使用できる場合、Oracle はこれらのネットワーク・サービスによる認証を受け入れることができます。Oracle でネットワーク認証サービスを使用するには、Oracle8i Enterprise Edition と Oracle Advanced Security が必要です。

関連項目：

- ネットワーク認証の詳細は、『Oracle8i 分散システム』を参照してください。ネットワーク認証サービスを使用する場合は、ネットワーク・ロールとデータベース・リンクについて特別な考慮事項があります。
- Oracle Advanced Security オプションの詳細は、『Oracle8i Advanced Security 管理者ガイド』を参照してください。

公開鍵インフラストラクチャベースの認証

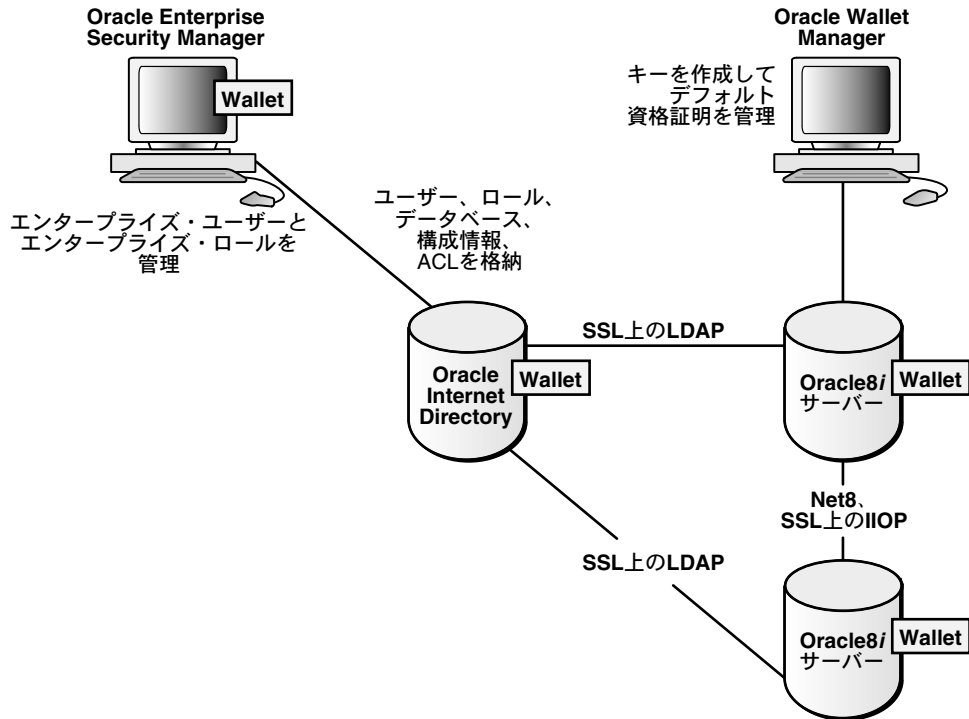
公開鍵暗号化システムに基づく認証システムは、ユーザー・クライアントにデジタル証明書を発行します。ユーザー・クライアントはそれを使用し、認証サーバーを直接関与させないで、社内のサーバーを直接認証します。Oracle は、公開鍵と証明書を使用するための公開鍵インフラストラクチャ（PKI）を提供します。PKI のコンポーネントは、次のとおりです。

- Secure Sockets Layer（SSL）による認証および保護セッション・キー管理。
- Oracle Call Interface（OCI）および PL/SQL 関数。ユーザー指定のデータに秘密鍵と証明書を使用してシグネチャを付け、証明書とトラスト・ポイントを使用してデータのシグネチャを検証します。
- 「Oracle Wallet」。これは、ユーザーの秘密鍵、ユーザー証明書および一連のトラスト・ポイント（ユーザーが信頼するルート証明書のリスト）を含むデータ構造です。
- 「Oracle Wallet Manager」。Oracle のクライアントとサーバー上でユーザー・キーを保護し、X.509v3 証明書を管理します。
- X.509v3 証明書。Oracle 外部の認証局から取得します。証明書が Oracle Wallet にロードされ、認証が使用可能になります。
- 「Oracle Enterprise Security Manager」。管理を容易にし、セキュリティ・レベルを高めるために、集中的な権限管理を提供します。ロールが Lightweight Directory Access Protocol（LDAP）をサポートしている場合は、Oracle Enterprise Security Manager により、Oracle Internet Directory にロールを格納し、取り出すことができます。また、ロールが Oracle スキーマと関連アクセス制御リストのインストールをサポートする場合は、そのロールを他の LDAP v3 準拠のディレクトリ・サーバーに格納できます。
- 「Oracle Internet Directory」。Oracle8i データベース上に作成された LDAP v3 準拠のディレクトリです。X.509 証明書を使用して認証されたユーザーについて、セキュリティ属性や権限など、ユーザーおよびシステム構成環境を管理できます。Oracle Internet Directory は認証レベルのアクセス制御を規定します。これにより、特定のユーザー（社内のセキュリティ管理者など）のみに、ディレクトリの特定の属性の読み込み、書込みまたは更新権限を限定できます。また、SSL 暗号化を介したディレクトリの問合せと応答の保護と認証もサポートしています。

- 「Oracle Enterprise Login Assistant」。ユーザーのサインオンを単純化します。Wallet を事前に Oracle Wallet Manager で構成する必要があります。

図 26-1 に、Oracle の公開鍵インフラストラクチャを示します。

図 26-1 Oracle の公開鍵インフラストラクチャ



注意： Oracle で公開鍵インフラストラクチャベースの認証を使用するには、Oracle8i Enterprise Edition と Oracle Advanced Security が必要です。

リモート認証

Oracle は、Remote Dial-In User Service (RADIUS) を介したユーザーのリモート認証をサポートしています。RADIUS は、ユーザー認証、許可およびアカウント処理に使用される標準軽量プロトコルです。Oracle で RADIUS を介したユーザーのリモート認証を使用するには、Oracle8i Enterprise Edition と Oracle Advanced Security が必要です。

関連項目： Oracle Advanced Security の詳細は、『Oracle8i Advanced Security 管理者ガイド』を参照してください。

Oracle データベースによる認証

Oracle では、データベースに格納されている情報を使用して、データベースに接続しようとするユーザーを認証できます。

Oracle でデータベースによる認証を使用する場合は、対応するパスワードを指定してそれぞれのユーザーを作成します。ユーザーは、接続の確立時に正しいパスワードを入力します。この方法により、データベースの無許可使用が防止されます。ユーザーのパスワードは、暗号化された形式でデータ・ディクショナリに格納されます。ユーザーは、いつでも自分のパスワードを変更できます。

接続時のパスワード暗号化

パスワードの機密性を保護するため、Oracle ではネットワーク（クライアント / サーバーおよびサーバー / サーバー）接続時にパスワードを暗号化できます。クライアントおよびサーバーのマシンでこの機能を使用可能にすると、パスワードは修正 DES（データ暗号化規格）アルゴリズムを使用して暗号化され、その後ネットワーク経由で送信されます。パスワードをネットワークへの侵入者から保護するために、接続のパスワード暗号化を使用可能にすることをお勧めします。

関連項目： ネットワーク・システムにおけるパスワード暗号化の詳細は、『Oracle8i 分散システム』を参照してください。

アカウントのロック

指定された試行回数の範囲内でユーザーがシステムにログインできない場合、ユーザーのアカウントがロックされることがあります。アカウントの構成方法によって、一定の時間の後に自動的にロックが解除されるか、またはデータベース管理者がロックを解除する必要があります。

CREATE PROFILE 文は、ユーザーが試行できるログインの失敗回数、および自動ロック解除前にアカウントがロックされたままになっている時間を構成するために使用します。

データベース管理者は、手動でアカウントをロックすることもできます。その場合、アカウントは自動的にロック解除されないため、データベース管理者による明示的なロック解除が必要です。

関連項目： 26-20 ページの「[プロファイル](#)」

パスワードの存続期間と時間切れ

パスワードの存続期間と時間切れのオプションを使用すると、データベース管理者はパスワードの存続期間を指定できます。その期間を過ぎると、パスワードは期限切れになり、そのアカウントにログインする前にパスワードを変更する必要があります。パスワードの期限が切れた後にデータベース・アカウントへの最初のログイン試行で、ユーザーのアカウントは猶予期間に入り、その猶予期間が終わるまで、ユーザーがログインするたびに警告メッセージが発行されます。

ユーザーは、猶予期間内にパスワードを変更するよう求められます。パスワードが猶予期間内に変更されなければ、そのアカウントはロックされ、それ以降、そのアカウントにはデータベース管理者の介入がなければログインできなくなります。

また、データベース管理者がパスワードの状態を時間切れに設定することもできます。この場合は、ユーザーのアカウントの状態が時間切れに変更され、そのユーザーがデータベースにログインするには、自分で、またはデータベース管理者に依頼してパスワードを変更する必要があります。

パスワード履歴

パスワード履歴オプションは、新しく指定される各パスワードを調べて、指定された期間内に、または指定されたパスワード変更回数の間に、同じパスワードが再使用されないようにするためのものです。データベース管理者は、CREATE PROFILE 文を使用してパスワードの再使用のルールを構成できます。

パスワードの複雑度の検証

複雑度の検証は、パスワードを推定してシステムに入ろうとする侵入者に対して、各パスワードが十分保護可能な複雑なものであることをチェックすることです。

Oracle において、デフォルトのパスワード複雑度検証ルーチンでは、各パスワードに次の条件が求められます。

- 4 文字以上の長さ。
- ユーザー ID と同じでないこと。
- 少なくとも 1 文字のアルファベット、1 文字の数字および 1 文字の句読点が含まれていること。
- welcome、account、database、user など、簡単な語からなる内部リストにある単語と一致しないこと。
- 前のパスワードと 3 文字以上異なっていること。

複数層の認証と許可

複数層環境では、Oracle は権限を制限し、すべての層を通じてクライアント認証を保持し、クライアントのために実行されるアクションを監査することによって、中間層アプリケーションのセキュリティを制御します。トランザクション処理モニターなど、大規模な中間層を使用するアプリケーションでは、中間層に接続するクライアントの識別性を維持できることが重要です。ただし、中間層が持つ利点の 1 つに「接続プーリング」があります。これにより、複数のユーザーが個別に接続しなくても 1 つのデータ・サーバーにアクセスできます。この種の環境では、接続を迅速に確立および切断できる機能が必要です。Oracle は、これらの環境で Oracle Call Interface を介して「軽量セッション」を作成する機能を提供します。これらの軽量セッションにより、各ユーザーをデータベース・パスワードで認証でき、個別のデータベース接続によるオーバーヘッドは生じません。また、中間層を介して実際のユーザーの識別性が保たれます。

パスワードあり、またはパスワードなしで軽量セッションを作成できます。中間層が外部またはファイアウォールにある場合は、軽量ユーザー・セッションごとに、パスワードを伴う軽量セッションを確立する方法が適しています。内部アプリケーション・サーバーの場合は、パスワードを必要としない軽量セッションを作成する方法が適しています。

クライアント、アプリケーション・サーバーおよびデータベース・サーバー

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 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。

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 Web ブラウザなどのクライアントの資格証明を検査できます。また、データベース・サーバーは、アプリケーション・サーバーが自身のために実行する操作と、クライアントのためにアプリケーション・サーバーが実行する操作を監査できます。たとえば、前者の操作はデータベース・サーバーへの接続要求などで、後者の操作はクライアント上に表示する情報の要求などです。

複数層環境における認証は、次のようなトラスト・リージョンに基づいています。

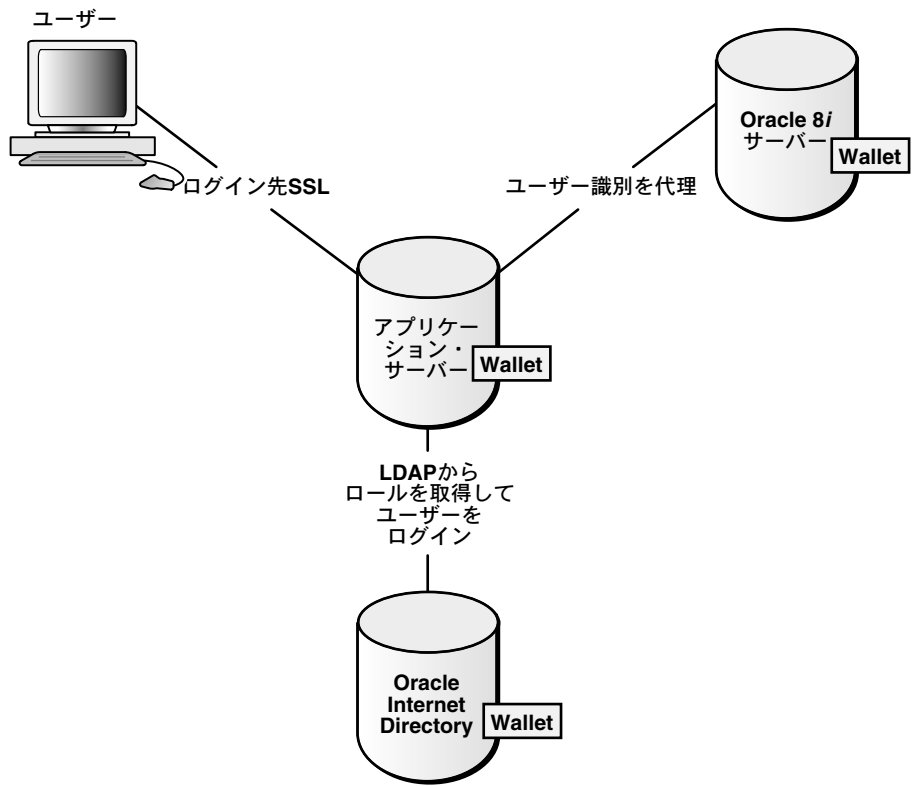
- クライアントは、通常はパスワードまたは X.509 証明書を使用して、アプリケーション・サーバーに認証の証明を提供します。
- アプリケーション・サーバーは、クライアント認証を検査し、自身をデータベース・サーバーに対して認証します。
- データベース・サーバーは、アプリケーション・サーバー認証をチェックし、クライアントが存在するかどうかを検証し、アプリケーション・サーバーがこのクライアントへの接続権限を持っているかどうかを検証します。

アプリケーション・サーバーは、接続中のクライアントのロールを使用可能にすることもできます。また、これらのロールは、認証リポジトリとして機能するディレクトリから取得できます。アプリケーション・サーバーが要求できるのは、これらのロールを使用可能にすることだけです。データベースは、次のことを検証します。

- クライアントがこれらのロールを付与されているかどうか、内部のロール・リポジトリをチェックします。
- アプリケーション・サーバーは、ユーザーのロールを使用して、そのユーザーのために接続する権限を付与されているかどうかをチェックします。

図 26-2 に、複数層認証の例を示します。

図 26-2 複数層の認証



中間層アプリケーションのセキュリティの問題

中間層アプリケーションには、次のようにセキュリティ上の問題が多数あります。

責任	データベース・サーバーは、クライアントのアクションと、アプリケーションがクライアントに代って実行するアクションを区別する必要があります。この両方のアクションを監査できることが必要です。
区別	データベース・サーバーは、Web サーバー・トランザクション、ブラウザ・クライアントのための Web サーバー・トランザクションおよびデータベースに直接アクセスするクライアントを、区別する必要があります。
最小限度の権限	ユーザーと中間層には、そのジョブを実行するための必要最小限度の権限を与える必要があります。

複数層環境における認証の問題

複数層アプリケーションは、すべての接続層を通じてクライアントの認証をメンテナンスします。この操作を必要とするのは、要求元クライアントの認証が失われると、有効な監査レコードをメンテナンスできなくなるためです。また、アプリケーション・サーバーがクライアントのために実行する操作と、アプリケーション自身のための操作を区別できなくなります。

複数層環境における制限付きの権限

複数層環境での権限は、要求された操作の実行に必要な権限に限定されます。

クライアントの権限 クライアントの権限は、複数層環境では最小限度になります。操作は、クライアントにかわってアプリケーション・サーバーが実行します。

アプリケーション・サーバーの権限 複数層環境におけるアプリケーション・サーバーの権限は、クライアント操作の実行中に望ましくない操作や不要な操作を実行できないように制限されます。

関連項目： 複数層認証の詳細は、『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。

Secure Socket Layer プロトコルによる認証

Secure Socket Layer (SSL) プロトコルは、アプリケーション・レイヤー・プロトコルです。SSL は、Oracle Internet Directory でのグローバル・ユーザー管理とは関係なく、データベースに対するユーザー認証に使用できます。つまり、ユーザーは SSL を使用してデータベースを認証でき、そのディレクトリ・アクセスについて何も暗黙的に指定する必要がありません。ただし、エンタープライズ・ユーザー機能を使用してユーザーとその権限をディレクト

リ内で管理する場合、ユーザーは SSL を使用してデータベースを認証する必要があります。SSL の使用は、初期化ファイルのパラメータで指定されます。

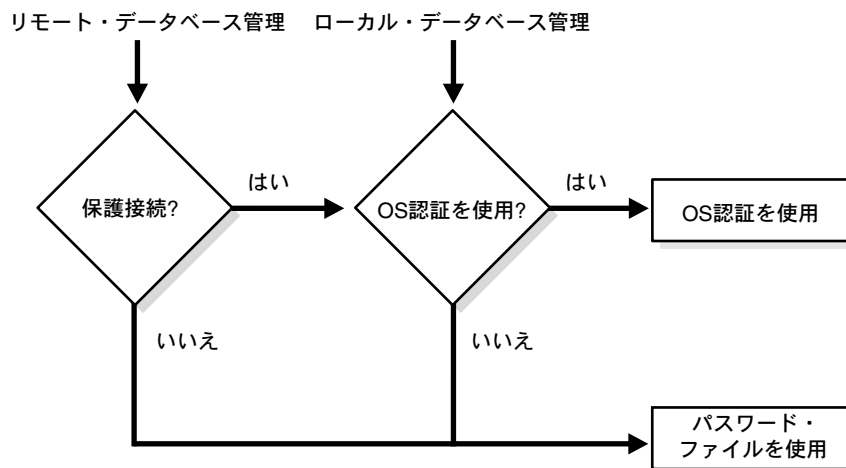
データベース管理者の認証

データベース管理者は、通常のデータベース・ユーザーが実行できない特別な操作（データベースの停止や起動など）を実行します。Oracle では、データベース管理者のユーザー名に対して、さらに安全性の高い認証方式を使用します。

データベース管理者の認証方式として、オペレーティング・システムによる認証とパスワード・ファイルによる認証のどちらを使用するかを選択できます。

図 26-3 に、データベースをローカルに（データベースが存在する同じマシン上で）管理するか、1つのリモート・クライアントから多数の異なるデータベース・マシンを管理するかに応じた、データベース管理者の認証方式の選択肢を示します。

図 26-3 データベース管理者の認証方式



ほとんどのオペレーティング・システムでは、データベース管理者のオペレーティング・システム認証には、データベース管理者のオペレーティング・システム・ユーザー名を特別なグループ（UNIX システムでは **dba** グループ）に入れること、またはそのオペレーティング・システム・ユーザー名に特別な処理を実行する権利を与えることが含まれます。

データベースは、パスワード・ファイルを使用することによって、SYSDBA または SYSOPER 権限を付与されたデータベース・ユーザー名を追跡します。データベース管理者にこれらの権限があると、次のアクションを実行できます。

SYSOPER	STARTUP、SHUTDOWN、ALTER DATABASE OPEN/MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG および RECOVER を実行できます。また、RESTRICTED SESSION 権限もこの権限に含まれます。
SYSDBA	ADMIN OPTION オプションが指定されているすべてのシステム権限、および SYSOPER システム権限が含まれます。CREATE DATABASE と時間ベースのリカバリが許可されます。

関連項目：

- データベース管理者のオペレーティング・システム認証の詳細は、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。
- 『Oracle8i 管理者ガイド』

ユーザー表領域の設定と割当て制限

データベース管理者は、すべてのユーザーのセキュリティ・ドメインの一部として、表領域の使用についていくつかのオプションを設定できます。

- [デフォルト表領域](#)
- [一時表領域](#)
- [表領域のアクセスと割当て制限](#)

デフォルト表領域

ユーザーがオブジェクトを入れる表領域を指定しないでスキーマ・オブジェクトを作成すると、そのオブジェクトはそのユーザーのデフォルト表領域に格納されます。ユーザーのデフォルト表領域は、ユーザー作成時に設定し、ユーザー作成後に変更できます。

一時表領域

一時セグメントの作成を必要とする SQL 文をユーザーが実行すると、ユーザーの一時表領域にそのセグメントが割り当てられます。

表領域のアクセスと割当て制限

データベースのどの表領域についても、ユーザーごとに「表領域の割当て制限」を設定できます。これにより、次の 2 つのことが可能になります。

- ユーザーが適切な権限を付与されている場合は、そのユーザーが指定された表領域を使用してスキーマ・オブジェクトを作成することを許可できます。
- 指定された表領域にあるユーザーのスキーマ・オブジェクトの記憶域に割り当てられたスペースの量を制限できます。

デフォルトでは、各ユーザーはデータベースの表領域での割当て制限を課されていません。このため、ユーザーになんらかのタイプのスキーマ・オブジェクトを作成する権限がある場合、そのユーザーには、オブジェクトを作成するときの表領域割当て制限が設定されているか、十分な表領域割当て制限を設定されている別のユーザーのスキーマにそのオブジェクトを作成する権限も必要になります。

ユーザーには、2 種類の表領域割当て制限を設定できます。1 つは表領域内のディスク領域を特定の量に制限する割当て制限（バイト、KB または MB 単位）、もう 1 つは表領域内のディスク領域を無制限とする割当て制限です。ユーザーのオブジェクトが表領域内の領域を使用しすぎないように、特定量の割当て制限を設定する必要があります。

表領域の割当て制限および一時セグメントは、相互に影響し合いません。

- 一時セグメントが作成されても、それはユーザーに課されている割当て制限の領域には加算されません。Oracle によって一時セグメントに自動的に作成されるスキーマ・オブジェクトは、SYS の所有となるため、割当て制限は適用されません。
- 一時セグメントは、ユーザーが割当て制限を課されていない表領域内に作成できます。

ユーザーの表領域の割当て制限は、そのユーザーの作成時に設定できます。その割当て制限を変更したり、後で別の割当て制限を追加できます。

ユーザーの表領域アクセスを取り消すには、そのユーザーの現行の割当て制限をゼロに変更します。割当て制限をゼロにすると、そのユーザーのオブジェクトは取り消したその表領域内に残りますが、それらのオブジェクトに新しい領域を割り当てることはできません。

ユーザー・グループ PUBLIC

各データベースには、PUBLIC というユーザー・グループがあります。PUBLIC ユーザー・グループは、表やビューなど、特定のスキーマ・オブジェクトへのパブリック・アクセスを提供し、すべてのユーザーに特定のシステム権限を提供します。すべてのユーザーは、自動的に PUBLIC ユーザー・グループに所属します。

PUBLIC のメンバーとして、ユーザーは接頭辞が USER および ALL であるすべてのデータ・ディクショナリ表を参照（選択）できます。さらに、ユーザーは PUBLIC に対して権限やロールを付与できます。すべてのユーザーは、PUBLIC に付与されている権限を使用できます。

PUBLIC には、任意のシステム権限、オブジェクト権限またはロールを付与または取消しできます。ただし、アクセス権に対する厳密なセキュリティを維持するため、PUBLIC に付与する権限とロールは、すべてのユーザーに関係のあるもののみにしてください。

PUBLIC に対してなんらかのシステム権限やオブジェクト権限を付与するか取り消すと、データベース中のすべてのビュー、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が再コンパイルされます。

PUBLIC には次の制限があります。

- PUBLIC に対して表領域割当て制限は設定できませんが、UNLIMITED TABLESPACE システム権限は設定できます。
- データベース・リンクおよびシノニムは、(CREATE PUBLIC DATABASE LINK/ SYNONYM を使用して) PUBLIC として作成できますが、その他のスキーマ・オブジェクトを PUBLIC の所有とすることはできません。たとえば、次の文は無効です。

```
CREATE TABLE public.emp . . . ;
```

注意： キーワード PUBLIC を指定してロールバック・セグメントを作成することはできますが、そのセグメントは PUBLIC ユーザー・グループが所有することにはなりません。すべてのロールバック・セグメントは SYS が所有します。

関連項目：

- [第 4 章「データ・ブロック、エクステンツおよびセグメント」](#)
- [第 27 章「権限、ロールおよびセキュリティ・ルール」](#)

ユーザー・リソースの制限とプロファイル

ユーザーのセキュリティ・ドメインの一部として、各ユーザーが使用できる各種のシステム・リソースの容量に制限を設定できます。これにより、CPU 時間などの貴重なシステム・リソースが無制限に浪費されるのを防止できます。

システム・リソースに多額の費用がかかる大規模なマルチ・ユーザー・システムにおいて、このリソース制限機能はたいへん有効です。1 人以上のユーザーが過度にリソースを消費すると、データベースの他のユーザーに有害な影響を及ぼす可能性があります。シングル・ユーザー・データベースまたは小規模のマルチ・ユーザー・データベースの場合、ユーザーがシステム・リソースを消費してもそれほど有害な影響はないため、システム・リソース機能はそれほど重要ではありません。

ユーザーのリソース制限とパスワード管理の作業環境は、そのユーザーのプロファイルを使用して管理します。プロファイルとは、そのユーザーに割り当てることができる一連のリソース制限に名前を付けたものです。それぞれの Oracle データベースに指定できるプロファイルの数に、制限はありません。Oracle において、セキュリティ管理者は、プロファイルによるリソース制限の規定を全体的に使用可能または使用禁止に設定できます。

リソース制限を設定した場合、ユーザーがセッションを作成すると、パフォーマンスがわずかに低下します。これは、ユーザーがデータベースに接続した時点で、そのユーザーのすべてのリソース制限データがロードされるためです。

関連項目： セキュリティ管理者の詳細は、『Oracle8i 管理者ガイド』を参照してください。

システム・リソースのタイプと制限

Oracle では、CPU 時間と論理読込みを含め、いくつかのタイプのシステム・リソースの使用を制限できます。一般に、それぞれのリソースは、セッション・レベル、コール・レベル、またはその両方で制御できます。

セッション・レベル ユーザーがデータベースに接続するたびに、セッションが作成されます。それぞれのセッションは、Oracle を実行するコンピュータの CPU 時間とメモリーを消費します。複数のリソース制限をセッション・レベルで設定できます。

ユーザーがセッション・レベルのリソース制限を超えると、現行の文は終了（ロールバック）し、セッションの制限に達したことを示すメッセージが戻されます。この時点では、カレント・トランザクション内のそれ以前のすべての文の結果はそのまま残っています。ユーザーが実行できる操作は、COMMIT、ROLLBACK または接続の切断（この場合、カレント・トランザクションはコミットされます）のみです。それ以外の操作を実行すると、エラーが発生します。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後も、カレント・セッションではユーザーはどんな作業も完了できません。

コール・レベル

SQL 文が実行されるたびに、いくつかのステップが実行されて文が処理されます。この処理では、データベースに対して複数のコールが異なる実行フェーズの一部として発行されます。1 回のコールで過度にシステムが使用されないように、複数のリソース制限をコール・レベルで設定できます。

ユーザーがコール・レベルのリソース制限を超えると、Oracle は文の処理を停止し、その文をロールバックし、エラーを戻します。ただし、カレント・トランザクションのそれ以前の文の結果はそのまま残り、そのユーザーのセッションは接続されたままになります。

CPU 時間

SQL 文やその他のタイプのコールが Oracle に発行されると、そのコールを処理するために一定量の CPU 時間が必要になります。平均的なコールであれば、わずかな CPU 時間で済みます。ただし、大量のデータや冗長な問合せを伴う SQL 文は CPU 時間を大量に消費することがあるため、他の処理に使用できる CPU 時間が少なくなります。

CPU 時間が無制限に消費されないようにするため、1 回のコール当りの CPU 時間と、1 つのセッション中に Oracle コールに使用される CPU 時間の合計を制限できます。これらの制限は、コールやセッションに使用される 1/100 秒 (0.01 秒) 単位の CPU 時間で設定し、測定されます。

論理読み込み

入出力 (I/O) は、データベース・システムで最もリソースの使用量が多い操作の 1 つです。I/O を集中的に実行する SQL 文は、メモリーとディスクの使用を独占することがあるため、他のデータベース操作がこれらのリソースをめぐって競合する原因になる可能性があります。

単一の原因による過度の I/O が発生しないようにするため、Oracle では、1 コール当り、および 1 セッション当りの論理データ・ブロック読み込み数を制限できます。論理データ・ブロック読み込みには、メモリーとディスクの両方からの論理データ・ブロック読み込みが含まれます。これらの制限は、1 コールまたは 1 セッション中に実行されるブロック読み込みの数として設定し、測定されます。

その他のリソース

Oracle では、さらに、その他のいくつかのセッション・レベルのリソース制限が提供されています。

- ユーザー当りの「同時実行セッション」数の制限。各ユーザーは、事前に定義された数まで同時実行セッションを作成できます。
- セッションの「アイドル時間」の制限。1 つのセッションでの Oracle コールと Oracle コールの間隔がアイドル時間制限に達すると、カレント・トランザクションはロールバックされ、セッションは異常終了し、そのセッションのリソースはシステムに戻されます。次のコールは、ユーザーがインスタンスから切断されたことを示すエラーを受け取ります。この制限は、分単位の経過時間として設定します。

注意： セッションがアイドル時間の制限を超えたために異常終了すると、その少し後に、異常終了したセッションの後処理としてプロセス・モニター (PMON) バックグラウンド・プロセスがクリーン・アップを実行します。PMON がこのプロセスを完了するまでは、異常終了したセッションも、セッションまたはユーザー・レベルのリソース制限に加算されます。

- セッション当りの経過接続時間の制限。セッションの持続時間が経過時間制限を超えると、カレント・トランザクションがロールバックされ、セッションが削除され、そのセッションのリソースがシステムに戻されます。この制限は、分単位の経過時間として設定します。

注意： Oracle は、経過アイドル時間や経過接続時間を絶えず監視しているわけではありません。絶えず監視するとすれば、システム・パフォーマンスが低下します。そのかわり数分ごとにチェックします。このため、Oracle がこの制限を規定してからセッションを異常終了させるまでの間に、セッションはこの制限をわずかに（5 分など）超える可能性があります。

- セッションのプライベート SGA 領域（プライベート SQL 領域に使用）の容量の制限。この制限が重要になるのは、マルチスレッド・サーバーの構成を使用するシステムの場合のみです。それ以外のシステムの場合、プライベート SQL 領域は PGA 内にあります。この制限は、インスタンスの SGA に使用するメモリーのバイト数として設定します。KB または MB で指定するには、"K" または "M" の文字を使用します。

関連項目： リソース制限を使用可能および使用禁止にする手順は、『Oracle8i 管理者ガイド』を参照してください。

プロファイル

プロファイルとは、Oracle データベースの有効なユーザー名に対して割り当てることができるリソース制限に名前を付けた集合です。プロファイルを使用すると、リソース制限を簡単に管理できます。また、パスワード方針を管理する手段としても使用できます。

プロファイルを使用する場合

ユーザー・プロファイルを作成して管理する必要があるのは、リソース制限がデータベース・セキュリティ・ルールの要件になっている場合のみです。プロファイルを使用するには、まずデータベース内の関連のあるユーザーを、いくつかのタイプに分類します。関連するユーザーの権限を管理するためにロールを使用するのと同様に、関連するユーザーのリソース制限を管理するためにプロファイルを使用します。データベースのすべてのタイプのユーザーを網羅するのに必要なプロファイルの数を決定し、それぞれのプロファイルに適切なリソース制限を決定します。

プロファイルのリソース制限の値の決定

プロファイルを作成し、そのプロファイルに含めるリソース制限を決定する前に、各リソース制限について適切な値を決定する必要があります。これらの値は、典型的なユーザーが実行する操作のタイプを基準として決定できます。たとえば、あるクラスのユーザーが通常は大量の論理データ・ブロック読み込みを実行しない場合、LOGICAL_READS_PER_SESSION および LOGICAL_READS_PER_CALL の制限は控えめに設定してください。

通常、ユーザー・プロファイルの適切なリソース制限値を決定するには、それぞれのタイプのリソースの使用状況について履歴情報を収集するのが最善です。たとえば、データベース管理者やセキュリティ管理者は、AUDIT SESSION 句を使用すると、CONNECT_TIME、LOGICAL_READS_PER_SESSION および LOGICAL_READS_PER_CALL の制限値についての情報を収集できます。

その他の制限値の統計情報は、Oracle Enterprise Manager（または SQL*Plus）のモニター機能、特に統計モニターを使用して収集できます。

関連項目： [第 28 章「監査」](#)

ライセンス

通常、Oracle のライセンスには、名前付きユーザーの最大数、または同時接続ユーザーの最大数が決められています。データベース管理者（DBA）は、そのサイトでライセンス契約を守ることに責任があります。Oracle のライセンス機能は、あるインスタンスに同時に接続されているセッションの数を追跡して制限したり、データベース内に作成されるユーザーの数を制限することによって、DBA モニター・システムの使用を支援するものです。

ライセンスされたセッション数より多くのセッションを接続したり、ライセンスされたユーザーより多くのユーザーを作成することが DBA にとって必要になった場合は、Oracle ライセンスをアップグレードして、制限を適切に変更できます。(Oracle ライセンスをアップグレードする場合は、必ずソフトウェアの販売元にご確認ください。)

注意： Oracle が Oracle アプリケーション (Oracle Office など) に組み込まれている場合、一部の古いオペレーティング・システムで実行されている場合または一部の国で使用するために購入した場合には、セッションの集合数にもユーザーの集合グループにもライセンスはありません。そのような場合に限り、Oracle ライセンス・メカニズムは適用されないため、使用禁止のままにしておいてください。

次に、Oracle で使用可能な 2 つの主要なライセンスのタイプについて説明します。

関連項目： ライセンスの詳細は、『Oracle8i 管理者ガイド』を参照してください。

同時使用ライセンス

「同時使用ライセンス」では、「同時実行ユーザー」の数が指定されます。同時実行ユーザーとは、ある時点で、指定されたコンピュータのデータベースに同時に接続できるセッションのことです。この数には、すべてのバッチ・プロセスとオンライン・ユーザーが含まれます。単一のユーザーが複数の同時実行セッションに関係している場合、その各セッションはセッションの合計数に別々に加算されます。データベースに直接接続されるセッションの数を少なくするために多重化ソフトウェア (TP モニターなど) を使用している場合、同時実行ユーザーの数は多重化フロントエンドへの個々の入力の数になります。

同時使用ライセンス・メカニズムによって、DBA は次のことを実行できます。

- `LICENSE_MAX_SESSIONS` パラメータを設定することにより、あるインスタンスに接続できる同時実行セッションの数の制限を設定できます。この制限に到達すると、`RESTRICTED SESSION` システム権限が付与されているユーザー以外はそのインスタンスに接続できなくなります。この制限により、DBA は不要なセッションをキル (停止) し、他のセッションの接続を可能にできます。
- `LICENSE_SESSIONS_WARNING` パラメータを設定することにより、あるインスタンスに接続できる同時実行セッションの数の警告制限を設定できます。警告制限に達しても、Oracle はそれ以上のセッションの接続を許可します (前述の最大数に達するまで) が、`RESTRICTED SESSION` 権限を付与されて接続しているユーザーには警告メッセージを送信し、データベースの `ALERT` ファイルにその警告メッセージを記録します。

DBA は、これらの制限をデータベースのパラメータ・ファイルに設定することにより、インスタンス起動時にそれが有効になるようにしたり、インスタンスの実行中に（ALTER SYSTEM 文を使用して）変更できます。後者の操作は、オフライン化できないデータベースの場合に便利です。

セッション・ライセンス・メカニズムによって DBA は、接続されているセッションの現行数と、インスタンスを起動して以降の同時実行セッションの最大数を調べることができます。V\$LICENSE ビューには、現行のライセンス制限の設定値、現行のセッション数およびインスタンスが起動して以降の同時実行セッションの最大数（セッションの「高水位標」）が表示されます。DBA は、この情報を使用してシステムのライセンス・ニーズを評価し、システムのアップグレードを計画できます。

Oracle Parallel Server で実行しているインスタンスの場合は、それぞれのインスタンスに独自の同時使用制限と警告制限を設定できます。各インスタンスの制限の合計が、サイトの同時使用ライセンスを超えないようにしてください。

着信データベース・リンク用に作成されるセッションも含め、同時使用制限はすべてのユーザー・セッションに適用されます。Oracle によって作成されたセッションや再帰的なセッションには適用されません。外部の多重化ソフトウェアを介して接続されるセッションは、Oracle ライセンス・メカニズムによって別個にカウントされることはありませんが、それぞれのセッションは Oracle ライセンスの合計に個別に加算されます。これらのセッションを考慮する責任は、DBA にあります。

名前付きユーザー・ライセンス

「名前付きユーザー・ライセンス」では、名前付きユーザーの数がライセンスで指定されます。「名前付きユーザー」とは、指定されたコンピュータで Oracle の使用を認められている個人のことです。それぞれのユーザーが同時に実行できるセッションの数や、データベースの同時実行セッションの数に、制限は設定されません。

名前付きユーザー・ライセンスによって DBA は、データベース・リンクを介して接続するユーザーを含め、データベースの中で定義されるユーザーの数に制限を設定できます。この制限に達すると、新しいユーザーは作成できなくなります。このメカニズムでは、データベースにアクセスする個々の人のユーザー名はデータベース内で一意であり、1 つのユーザー名が複数の人の間で共有されることはないことが前提となっています。

DBA は、これらの制限をデータベースのパラメータ・ファイルに設定することにより、インスタンス起動時にそれが有効になるようにしたり、インスタンスの実行中に（ALTER SYSTEM 文を使用して）変更できます。後者の操作は、オフライン化できないデータベースの場合に便利です。

Oracle Parallel Server で同じデータベースに複数のインスタンスが接続する場合は、同じデータベースに接続されるすべてのインスタンスの名前付きユーザー制限は同じになっている必要があります。

関連項目：

- 『Oracle8i Parallel Server 概要』
- 『Oracle8i Parallel Server 管理、配置およびパフォーマンス』

権限、ロールおよびセキュリティ・ルール

この章では、ユーザーが実行できるシステム操作とスキーマ・オブジェクトに対して実行できるアクセスを、権限、ロールおよびセキュリティ・ルールによって制御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。

- 権限の概要
 - システム権限
 - スキーマ・オブジェクト権限
 - 表のセキュリティに関するトピック
 - ビューのセキュリティに関するトピック
 - プロシージャのセキュリティに関するトピック
 - 型のセキュリティに関するトピック
- ロール
- ファイングレイン・アクセス・コントロール
- アプリケーション・コンテキスト

権限の概要

「権限」は、特定のタイプの SQL 文を実行するため、または別のユーザーのオブジェクトにアクセスするための権利です。たとえば、次のことをする権利が権限です。

- 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）
- 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

権限をユーザーに付与すると、それらのユーザーが業務に必要な作業を実行できるようになります。なお、権限は、必要な作業を実行する上で本当にその権限を必要としているユーザーにのみ付与します。必要でない権限まで付与すると、セキュリティを維持できなくなる可能性があります。ユーザーは次の 2 つの方法で権限を受け取ることができます。

- 権限を明示的にユーザーに付与します。たとえば、EMP 表にレコードを挿入する権限を、ユーザー SCOTT に明示的に付与できます。
- 権限をロール（名前付きの権限グループ）に付与した上で、そのロールを 1 人以上のユーザーに付与します。たとえば、EMP 表からレコードを選択、挿入、更新および削除する権限を、CLERK という名前のロールに付与し、このロール CLERK をユーザー SCOTT や BRIAN に付与できます。

ロールを使用することによって権限の管理が容易になり、改善されるため、通常、権限は個々のユーザーではなくロールに付与します。

権限は次の 2 つに分類できます。

- システム権限
- スキーマ・オブジェクト権限

関連項目： すべてのシステム権限およびスキーマ・オブジェクト権限の詳細リストと、権限管理の手順は、『Oracle8i 管理者ガイド』を参照してください。

システム権限

システム権限とは、特定のアクションを実行する権限、または特定のタイプのスキーマ・オブジェクトに対してアクションを実行する権限のことです。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から行を削除する権限などがシステム権限です。システム権限には 60 以上の種類があります。

システム権限の付与と取消し

システム権限は、ユーザーやロールに対して付与したり取り消すことができます。システム権限をロールに付与すると、そのロールを使用してシステム権限を管理できます。たとえば、ロールを使用すると権限を選択的に使用できるようになります。

注意： 通常、エンド・ユーザーはシステム権限に関連する機能を必要としないため、システム権限は、管理者やアプリケーション開発者にのみ付与するようにしてください。

次のどちらかを使用して、ユーザーやロールにシステム権限を付与したり、その権限を取り消します。

- Oracle Enterprise Manager の「Grant System Privileges/Roles」ダイアログ・ボックス および「Revoke System Privileges/Roles」ダイアログ・ボックス
- SQL 文 GRANT および REVOKE

システム権限を付与または取消しできるユーザー

他のユーザーにシステム権限を付与したり、他のユーザーのシステム権限を取り消すことができるのは、ADMIN OPTION によって特定のシステム権限を付与されているユーザー、または GRANT ANY PRIVILEGE システム権限を付与されているユーザーのみです。

スキーマ・オブジェクト権限

スキーマ「オブジェクト権限」とは、次のように特定のスキーマ・オブジェクトに対して特定のアクションを実行する権限です。

- 表
- ビュー
- 順序
- プロシージャ
- ファンクション
- パッケージ

各タイプのスキーマ・オブジェクトごとに、異なるオブジェクト権限があります。たとえば、DEPT 表から行を削除する権限は、1つのオブジェクト権限です。

クラスタ、索引、トリガーおよびデータベース・リンクなど、一部のスキーマ・オブジェクトには、対応付けられたオブジェクト権限はありません。これらのオブジェクトの使用は、システム権限によって決定されます。たとえば、クラスタを変更するには、ユーザーはそのクラスタを所有しているか、または ALTER ANY CLUSTER システム権限が必要です。

スキーマ・オブジェクトとそのシノニムは、権限に関しては等価です。つまり、表、ビュー、順序、プロシージャ、ファンクションまたはパッケージについて付与されるオブジェクト権限は、その基礎となるオブジェクトを名前で参照するかシノニムで参照するかにかかわらず、適用されます。

たとえば、JWARD.EMP という表があり、それに JWARD.EMPLOYEE というシノニムがある場合に、ユーザー JWARD が次の文を発行するとします。

```
GRANT SELECT ON emp TO swilliams;
```

ユーザー SWILLIAMS は、名前またはシノニム JWARD.EMPLOYEE を使用して表を参照することによって、JWARD.EMP を問い合わせることができます。

```
SELECT * FROM jward.emp;  
SELECT * FROM jward.employee;
```

表、ビュー、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オブジェクト権限を、そのオブジェクトの「シノニム」に付与した場合、結果はシノニムを使用しない場合と同じです。たとえば、JWARD が EMP 表に対する SELECT 権限を SWILLIAMS に付与する場合、JWARD は次のどちらの文でも使用できます。

```
GRANT SELECT ON emp TO swilliams;  
GRANT SELECT ON employee TO swilliams;
```

シノニムが削除された場合、そのシノニムを指定して権限が付与されていた場合でも、基礎になるスキーマ・オブジェクトに対して付与された権限はすべて有効なままです。

スキーマ・オブジェクト権限の付与と取消し

スキーマ・オブジェクト権限は、ユーザーやロールに対して付与したり取り消すことができます。オブジェクト権限をロールに付与する場合、その権限を選択的に使用可能にできます。ユーザーやロールのオブジェクト権限を付与したり取り消すには、それぞれ SQL 文の GRANT と REVOKE を使用するか、Oracle Enterprise Manger の「Add Privilege to Role/User」ダイアログ・ボックスと「Revoke Privilege from Role/User」ダイアログ・ボックスを使用します。

スキーマ・オブジェクト権限を付与できるユーザー

ユーザーは、自分のスキーマに含まれているスキーマ・オブジェクトに関しては、自動的にすべてのオブジェクト権限が付与されています。ユーザーは、自分が所有するスキーマ・オブジェクトに対するオブジェクト権限を、他のユーザーまたはロールに付与できます。GRANT 文の GRANT OPTION を指定して付与した場合、その権限受領者は、さらに別のユーザーにオブジェクト権限を付与できます。このオプションを指定しなければ、権限受領者はその権限を使用できますが、別のユーザーには権限を付与できません。

関連項目：『Oracle8i SQL リファレンス』

表のセキュリティに関するトピック

表に対するスキーマ・オブジェクト権限は、DML および DDL 操作のレベルで表のセキュリティ機能を実現します。

データ操作言語（DML）操作

表またはビューに対する DELETE、INSERT、SELECT および UPDATE DML 操作を使用する権限を付与できます。これらの権限は、表のデータを問い合わせたり変更する必要があるユーザーとロールにのみ付与してください。

表に対する INSERT と UPDATE の各権限は、表の特定の列に制限できます。選択的な INSERT 権限を付与されたユーザーは、選択された列の値を行に挿入できます。その他の列には、NULL またはその列のデフォルト値が挿入されます。選択的な UPDATE 権限によって、ユーザーは行の特定の列に限ってその値を更新できます。機密データに対するユーザー・アクセスを制限するには、INSERT 権限と UPDATE 権限を選択的に使用します。

たとえば、データ入力者が EMP 表の SAL 列を変更しないようにするには、SAL 列を含めずに、選択的な INSERT 権限と UPDATE 権限を付与します。また、SAL 列を含まないようにしたビューを使用すると、同じことをさらに高いセキュリティ・レベルで実現できます。

関連項目： これらの DML 操作の詳細は、『Oracle8i SQL リファレンス』を参照してください。

データ定義言語（DDL）操作

ALTER、INDEX および REFERENCES の各権限は、表に対する DDL 操作の実行を許可します。これらの権限によって、他のユーザーは表の依存性を変更または作成できるようになるため、この権限は慎重に付与する必要があります。表に対して DDL 操作を実行するユーザーには、さらにその他のシステム権限やオブジェクト権限が必要です。たとえば、表にトリガーを作成するには、その表に対する ALTER TABLE オブジェクト権限と CREATE TRIGGER システム権限の両方が必要です。

INSERT 権限および UPDATE 権限と同じように、表の特定の列に REFERENCES 権限を付与できます。REFERENCES 権限を付与されたユーザーは、付与の対象となった表を、自分の表の中に作成する外部キーの親キーとして使用できます。外部キーが存在すると、親キーに対して実行される可能性のあるデータ操作や表の変更が制限されるため、この動作は特殊な権限によって制御されます。列固有の REFERENCES 権限によって、権限受領者が使用できるのは、指定された列（当然、親表の主キーまたは一意キーを最低 1 つ含む）に制限されます。

関連項目： 主キー、一意キーおよび整合性制約の詳細は、[第 25 章「データの整合性」](#)を参照してください。

ビューのセキュリティに関するトピック

ビューに対するスキーマ・オブジェクト権限は、ビューの導出元のベース表に実際に影響する様々な DML 操作を許可します。表に対する DML オブジェクト権限は、ビューに対しても同じように適用されます。

ビューの作成に必要な権限

ビューを作成するには、次の要件を満たしている必要があります。

- 次のどちらかのシステム権限が、明示的に、またはロールを介して付与されている必要があります。
 - CREATE VIEW システム権限（自分のスキーマ内にビューを作成するため）
 - CREATE ANY VIEW システム権限（別のユーザーのスキーマ内にビューを作成するため）
- 次のいずれかの権限が明示的に付与されている必要があります。
 - ビューの基礎となるすべてのベース・オブジェクトに対する SELECT、INSERT、UPDATE または DELETE オブジェクト権限
 - SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE または DELETE ANY TABLE システム権限
- さらに、ビューに対するアクセス権を他のユーザーに付与する場合は、ベース・オブジェクトに対するオブジェクト権限が GRANT OPTION 句付きで付与されているか、または ADMIN OPTION 句によって適切なシステム権限が付与されている必要があります。これらの権限がなければ、権限受領者はビューにアクセスできません。

関連項目：『Oracle8i SQL リファレンス』

ビューによる表セキュリティの強化

ビューを使用するのに必要な権限は、そのビュー自体に対する適切な権限のみです。ビューの基礎となるベース・オブジェクトに対する権限は必要ありません。

ビューを使用することにより、表に対してさらに 2 つのセキュリティ・レベル（列レベルのセキュリティと値ベースのセキュリティ）が追加されます。

- ビューは、ベース表の中から選択した列のアクセスを提供できます。たとえば、EMP 表の中から EMPNO、ENAME および MGR の 3 列のみを表示するようにビューを定義できます。

```
CREATE VIEW emp_mgr AS
  SELECT ename, empno, mgr FROM emp;
```

- ビューにより、表の情報に対して、値ベースのセキュリティを実現できます。ビュー定義の中に WHERE 句を使用すると、ベース表の中から選択した行のみが表示されます。次の 2 つの例を考えてみます。

```
CREATE VIEW lowsal AS
  SELECT * FROM emp
  WHERE sal < 10000;
```

LOWSAL ビューでは、EMP 表のうち給与値が 10000 未満のすべての行にアクセスできます。LOWSAL ビューでは、EMP 表のすべての列にアクセスできることに注目してください。

```
CREATE VIEW own_salary AS
  SELECT ename, sal
  FROM emp
  WHERE ename = USER;
```

OWN_SALARY ビューでは、ENAME がビューのカレント・ユーザーに一致している行にのみアクセスできます。OWN_SALARY ビューは USER 関数を使用します。この関数の値は、常にカレント・ユーザーを参照します。なお、このビューでは、列レベルのセキュリティと値ベースのセキュリティを併用しています。

プロシージャのセキュリティに関するトピック

スタンドアロン・プロシージャ、ファクションおよびパッケージを含め、プロシージャに対する「スキーマ・オブジェクト権限」は、EXECUTE のみです。この権限は、プロシージャの実行、またはそのプロシージャをコールする他のプロシージャのコンパイルを必要とするユーザーにのみ付与してください。

プロシージャの実行とセキュリティ・ドメイン

特定のプロシージャに対する EXECUTE オブジェクト権限を持っているユーザーは、そのプロシージャを実行したり、それを参照するプログラム・ユニットをコンパイルできます。このプロシージャがコールされるときには、実行時権限チェックは実行されません。EXECUTE ANY PROCEDURE システム権限を付与されているユーザーは、データベース内の任意のプロシージャを実行できます。

プロシージャの実行権限は、ロールを通してユーザーに付与できます。

実行者権限プロシージャの場合は、参照オブジェクトに対する追加の権限が必要ですが、定義者権限プロシージャの場合は不要です。

関連項目：

- 27-20 ページの「PL/SQL ブロックとロール」
- 17-11 ページの「ストアド・プロシージャの依存性の追跡」

定義者権限 定義者権限プロシージャのユーザーに必要なのは、そのプロシージャを実行する権限のみで、そのプロシージャがアクセスする基礎となるオブジェクトの権限は不要です。これは、定義者権限プロシージャは、その実行者に関係なく、所有者のセキュリティ・ドメインで動作するからです。プロシージャの所有者は、参照オブジェクトに対する必要なオブジェクト権限をすべて持つ必要があります。定義者権限プロシージャのユーザーに付与する権限が少ないほど、データベース・アクセスを厳密に制御できます。

定義者権限プロシージャを使用して、プライベート・データベース・オブジェクトへのアクセスを制御し、データベースのセキュリティ・レベルを強化できます。定義者権限プロシージャを記述し、ユーザーには EXECUTE 権限のみを付与することにより、ユーザーがそのプロシージャを介さなければ参照オブジェクトにアクセスできないようにできます。

実行時には、定義者権限ストアド・プロシージャの所有者の権限が、常にそのプロシージャを実行する前にチェックされます。参照オブジェクトに対して必要な権限が、定義者権限プロシージャの所有者から取り消されていると、所有者もその他のユーザーも、そのプロシージャを実行できません。

注意： トリガーの実行には、定義者権限プロシージャと同じパターンが適用されます。ユーザーは、実行する権限を付与されている SQL 文を実行できます。SQL 文の実行結果として、トリガーが起動されます。トリガー・アクション内の文は、トリガーを所有するユーザーのセキュリティ・ドメインで実行されます。

関連項目： 第 19 章「トリガー」

実行者権限 実行者権限プロシージャは、実行者のすべての権限で実行されます。実行者権限プロシージャが定義者権限プロシージャによって直接または間接的にコールされた場合でなければ、ロールが使用可能になります。実行者権限プロシージャのユーザーには、そのプロシージャが実行者のスキーマ内で解決される外部参照を介してアクセスする、オブジェクトに対する権限が（直接、またはロールを介して）必要です。

実行者には、DML 文または動的 SQL 文に埋め込まれているプログラム参照にアクセスする権限が実行時に必要です。これは、この種のプログラム参照は実質的に実行時に再コンパイルされるためです。

PL/SQL ファンクション・コールなど、その他のすべての外部参照の場合、所有者の権限はコンパイル時にチェックされ、実行時チェックは行われません。したがって、実行者権限プロシージャのユーザーには、DML 文や動的 SQL 文の外側にある外部参照に対する権限は不要です。また、実行者権限プロシージャの開発者による権限の付与は、実行者権限プロシージャによって直接参照されるすべてのオブジェクトに対してではなく、プロシージャ自体に対する権限を自分に付与するだけでよいことになります。

ほとんどの DBMS_* パッケージなど、Oracle が提供する多数のパッケージは、実行者権限で実行されます。つまり、所有者 (SYS) ではなくカレント・ユーザーとして実行されます。ただし、DBMS_RLS パッケージなど、いくつか例外があります。

複数のプログラム・ユニットからなるソフトウェア・バンドルを作成し、一部のプログラム・ユニットには定義者権限を付与し、残りのプログラム・ユニットには実行者権限を付与して、プログラムのエントリ・ポイントを限定できます (「コントロール・ステップイン」)。エントリ・ポイント・プロシージャを実行する権限を持つユーザーは、内部プログラム・ユニットを間接的に実行できますが、内部プログラムを直接コールすることはできません。

関連項目：

- 17-8 ページの「外部参照の解決」
- 27-23 ページの「ファイングレイン・アクセス・コントロール」
- オラクル社が提供するパッケージの詳細は、『Oracle8i PL/SQL 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。

プロシージャの作成または変更に必要なシステム権限

プロシージャを作成するには、ユーザーには CREATE PROCEDURE または CREATE ANY PROCEDURE システム権限が必要です。プロシージャを変更する、つまりプロシージャを手動で再コンパイルするには、そのプロシージャを所有しているか、ALTER ANY PROCEDURE システム権限を持っている必要があります。

プロシージャを所有するユーザーは、プロシージャ本体で参照されるスキーマ・オブジェクトに対する権限も持っている必要があります。プロシージャを作成するには、プロシージャによって参照されるすべてのオブジェクトに対して、必要な権限 (システム権限やオブジェクト権限) を明示的に付与されている必要があります。ロールを介してはそれらの権限を取得できません。これには、作成するプロシージャ内からコールするプロシージャに対する EXECUTE 権限も含まれます。

また、トリガーでは、参照オブジェクトに対する権限をトリガーの所有者に対して明示的に付与することが必要です。権限が明示的に、またはロールを介して付与されていても、無名 PL/SQL ブロックは任意の権限を使用できます。

パッケージとパッケージ・オブジェクト

パッケージに対する EXECUTE オブジェクト権限を付与されているユーザーは、パッケージ内の任意のパブリック・プロシージャまたはファンクションを実行したり、任意のパブリック・パッケージ変数の値をアクセスまたは変更できます。パッケージの構成体に対して特定の EXECUTE 権限は付与できません。したがって、データベース・アプリケーションのプロシージャ、ファンクションおよびパッケージを開発する場合は、セキュリティの確立に関して2つの選択肢を考慮してください。これらの選択肢について、次に示す例で具体的に説明します。

例 1 この例では、2つのパッケージの本体の中に4つのプロシージャを作成します。

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO emp . . .
    END hire;
  PROCEDURE fire(...) IS
    BEGIN
      DELETE FROM emp . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE EMP SET sal = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE EMP SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

このプロシージャを実行するためのアクセス権限を付与するには、次の文を使用して、パッケージに対する EXECUTE 権限を付与します。

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

つまり、EXECUTE 権限はパッケージに対して付与されるため、これによってパッケージ・オブジェクト全体にわたる一様なアクセスが提供されます。

例 2 この例では、単一のパッケージ本体にある 4 つのプロシージャ定義を示します。さらに 2 つのスタンドアロン・プロシージャと 1 つのパッケージを作成し、メイン・パッケージ内で定義したプロシージャへのアクセスを提供します。

```
CREATE PACKAGE BODY employee_changes AS
    PROCEDURE change_salary(...) IS BEGIN ... END;
    PROCEDURE change_bonus(...) IS BEGIN ... END;
    PROCEDURE insert_employee(...) IS BEGIN ... END;
    PROCEDURE delete_employee(...) IS BEGIN ... END;
END employee_changes;
```

```
CREATE PROCEDURE hire
BEGIN
    employee_changes.insert_employee(...)
END hire;
```

```
CREATE PROCEDURE fire
BEGIN
    employee_changes.delete_employee(...)
END fire;
```

```
PACKAGE raise_bonus IS
    PROCEDURE give_raise(...) AS
    BEGIN
        employee_changes.change_salary(...)
    END give_raise;
```

```
    PROCEDURE give_bonus(...)
    BEGIN
        employee_changes.change_bonus(...)
    END give_bonus;
```

この方法を使用すると、実際に作業を実行するプロシージャ（EMPLOYEE_CHANGES パッケージ内のプロシージャ）は 1 つのパッケージ内で定義され、宣言されたグローバル変数やカーソルなどを共有できます。トップ・レベルのプロシージャの HIRE と FIRE、および追加のパッケージ RAISE_BONUS を宣言することにより、選択的な EXECUTE 権限をメイン・パッケージ内のプロシージャに対して付与できます。

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

型のセキュリティに関するトピック

この項では、型、メソッドおよびオブジェクトに対する権限について説明します。

システム権限

Oracle8i では、名前付きの型（オブジェクト型、VARRAY および NESTED TABLE）について、[表 27-1](#) のシステム権限が定義されています。

表 27-1 名前付きの型に対するシステム権限

権限	許可される操作
CREATE TYPE	自分のスキーマ内で名前付きの型を作成します。
CREATE ANY TYPE	任意のスキーマ内で名前付きの型を作成します。
ALTER ANY TYPE	任意のスキーマ内で名前付きの型を変更します。
DROP ANY TYPE	任意のスキーマ内で名前付きの型を削除します。
EXECUTE ANY TYPE	任意のスキーマ内で名前付きの型を使用および参照します。

CONNECT ロールと 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、これらの権限すべてが含まれています。

オブジェクト権限

名前付きの型に適用されるオブジェクト権限は、EXECUTE のみです。名前付きの型に対する EXECUTE 権限があれば、ユーザーはその型を使用して次の操作を実行できます。

- 表を定義します。
- リレーショナル表に列を定義します。
- 名前付きの変数またはパラメータを宣言します。

EXECUTE 権限により、ユーザーは、型コンストラクタを含め、その型のメソッドを起動できます。これは、ストアド PL/SQL プロシージャに対する EXECUTE 権限と同じです。

メソッド実行モデル

メソッドの実行は、他のストアド PL/SQL プロシージャと同じです。

関連項目： [27-7 ページの「プロシージャのセキュリティに関するトピック」](#)

型の作成と型を使用した表の作成に必要な権限

型を作成するには、次の要件を満たしている必要があります。

- 自分のスキーマ内で型を作成するには CREATE TYPE システム権限が、他のユーザーのスキーマ内で型を作成するには CREATE ANY TYPE システム権限が必要です。この2つの権限は、明示的に、またはロールを介して取得できます。
- 型の所有者には、その型の定義で参照されている他のすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要があります。所有者が、ロールを介して必要な権限を取得することはできません。
- 型の所有者が他のユーザーに型へのアクセス権を付与する意図を持っている場合は、GRANT OPTION で参照された型への EXECUTE 権限を付与されるか、ADMIN OPTION で EXECUTE ANY TYPE システム権限を付与される必要があります。どちらかの権限がなければ、権限不足のため、型の所有者は型へのアクセス権を他のユーザーに付与できません。

型を使用して表を作成するには、表の作成要件のみでなく、次の要件を満たす必要があります。

- 表の所有者には、その表で参照されているすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要があります。所有者が、ロールを介して必要な権限を取得することはできません。
- 表の所有者が他のユーザーに表へのアクセス権を付与する意図を持っている場合は、GRANT OPTION で参照された型への EXECUTE 権限を付与されるか、ADMIN OPTION で EXECUTE ANY TYPE システム権限を付与される必要があります。どちらかの権限がなければ、権限不足のため、表の所有者は型へのアクセス権を他のユーザーに付与できません。

関連項目： 表の作成要件については、27-5 ページの「[表のセキュリティに関するトピック](#)」を参照してください。

例

CONNECT および RESOURCE ロールを持つユーザーが存在するとします。

- USER1
- USER2
- USER3

USER1 は、自分のスキーマで次の 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、自分のスキーマで次の 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 OPTION 付きの EXECUTE 権限を持っているためです。

```
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 の TYPE1 に対して GRANT OPTION 付きの EXECUTE を持っていないためです。

```
GRANT SELECT ON tab1 TO user3;
```

USER3 は、次の文を正常に実行できます。

```
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

DML コマンドについて列レベルと表レベルの権限が存在する場合は、列オブジェクトと行オブジェクトの両方に適用されます。Oracle8i では、オブジェクト表について、表 27-2 の権限が定義されています。

表 27-2 オブジェクト表に対する権限

権限	許可される操作
SELECT	表からオブジェクトとその属性にアクセスします。
UPDATE	表の行を構成するオブジェクトの属性を変更します。
INSERT	表に新規オブジェクトを作成します。
DELETE	行を削除します。

同様に、列オブジェクトには表権限と列権限が適用されます。インスタンスを検索するだけでは、型情報は明らかになりません。ただし、クライアントは、型インスタンスのイメージを解釈するために、名前付きの型の情報にアクセスする必要があります。クライアントからこの種の型情報を要求されると、Oracle は型の EXECUTE 権限をチェックします。

次のスキーマを考えてみます。

```
CREATE TYPE emp_type (  
    eno NUMBER, ename CHAR(31), eaddr addr_t);  
CREATE TABLE emp OF emp_t;
```

さらに、次の 2 つの問合せについて考えます。

```
SELECT VALUE(emp) FROM emp;  
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。

ただし、2 番目の問合せを実行しても、名前付きの型が含まれていないため、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 attr4.attr3.attr2 FROM tab3;
```

USER3 は基礎となる型に対する明示的な権限を持っていませんが、USER3 によるどちらの SELECT 文も成功するので注意してください。型および表の所有者が、GRANT OPTION 付きの必要な権限を持っているためです。

Oracle は、次の要求に対する権限をチェックし、クライアントがそのアクションに必要な権限を持っていない場合はエラーを返します。

- 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する時には、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 権限をチェックします。
- 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する時には、Oracle は目的のオブジェクト表に対する UPDATE 権限をチェックします。
- 新しいオブジェクトをフラッシュする時には、Oracle は目的のオブジェクト表に対する INSERT 権限をチェックします。
- オブジェクトを削除する時には、Oracle は目的の表に対する DELETE 権限をチェックします。
- 名前付きの型のオブジェクトをピンする時には、Oracle はそのオブジェクトに対する EXECUTE 権限をチェックします。

クライアントの 3GL アプリケーション内でオブジェクトの属性を変更する時には、Oracle はオブジェクト全体を更新します。したがって、ユーザーにはオブジェクト表に対する UPDATE 権限が必要です。アプリケーションがオブジェクト表の特定の列に対応する属性を変更する場合でも、その列のみに対する UPDATE 権限では不十分です。したがって、Oracle は、オブジェクト表に対する列レベルの権限はサポートしていません。

型の依存性

プロシージャや表などのストアド・オブジェクトと同様に、他のオブジェクトから参照される型を、依存性があると呼びます。表が型に依存する特殊な問題点はいくつかあります。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は、その型を変更すると、その表に格納されているすべてのデータにアクセスできなくなります。変更によってこのような結果になるのは、型に必要な権限が取り消される場合や、型または依存型が削除される場合です。このどちらかのアクションが発生すると、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。依存型が削除されていたために無効になった表は、アクセス不能のままで、実行できるアクションは削除のみです。

型に対する権限を取り消したり型を削除すると重大な影響が生じるため、デフォルトでは SQL 文 REVOKE および DROP TYPE の実装は意図的に限定されています。つまり、どちらの文でも、名前付きの型に表または型が依存していると、エラーが戻されて文は異常終了します。ただし、どちらの文も、FORCE 句を使用すると常に正常終了し、表が依存している場合はその表は無効になります。

関連項目： REVOKE、DROP TYPE および FORCE 句の使用の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

ロール

Oracle では、ロールを使用することにより、簡単かつ制御された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します。ロールは、エンド・ユーザーのシステム権限とスキーマ・オブジェクト権限を容易に管理できるように設計されています。ただし、ロールは、アプリケーション開発者が使用することを目的としたものではありません。ストアド・プログラム構成体の中からスキーマ・オブジェクトにアクセスする権限は、直接付与する必要があるからです。

ロールの次のような特性によって、データベース内での権限管理がさらに容易になります。

権限管理に要する労力の削減	複数のユーザーに対して同一の権限セットを明示的に付与するかわりに、関連するユーザー・グループのための権限をまとめて1つのロールに付与しておき、そのグループの各メンバーにはそのロールを付与するだけですみます。
動的な権限管理	あるグループの権限の変更が必要な場合、そのロールの権限を修正するだけですみます。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・ドメインには、そのロールに対して加えられる変更が自動的に反映されます。
権限の選択的な可用性	ある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	ロールの存在はデータ・ディクショナリに記録されます。したがって、ユーザーが特定のユーザー名でアプリケーションを実行したときに、アプリケーションがディクショナリに問い合わせ、自動的に特定のロールを使用可能（または使用禁止）にするようにアプリケーションを設計できます。
アプリケーション固有のセキュリティ	ロールの使用はパスワードを使用して保護できます。正しいパスワードを入力するとロールが使用可能になるようなアプリケーションを作成できます。パスワードを知らないユーザーは、ロールを使用可能にできません。

関連項目：

- プロシージャに関する制限の詳細は、27-21 ページの「[データ定義言語の文とロール](#)」を参照してください。
- アプリケーションからロールを使用可能にする手順は、『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。

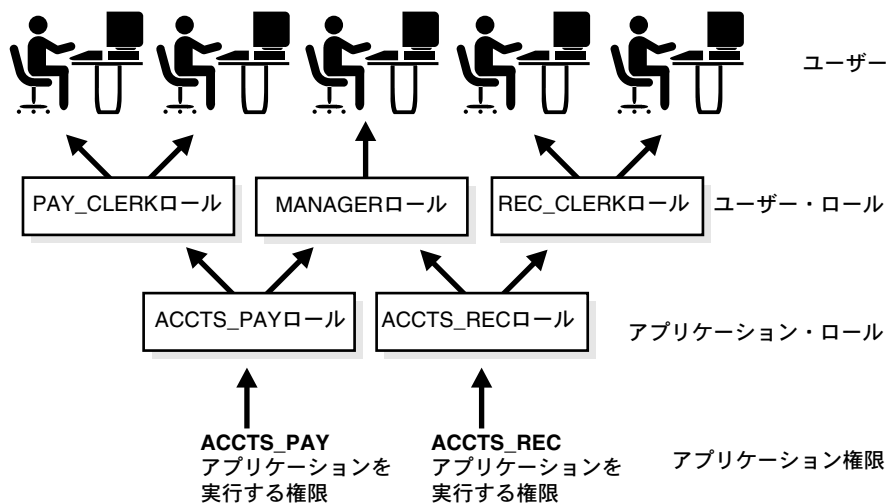
ロールの一般的な使用方法

通常は、次のどちらかの目的でロールを作成します。

- データベース・アプリケーションに対する権限の管理
- ユーザー・グループに対する権限の管理

図 27-1 とその後の項で、ロールの 2通りの使用方法について説明します。

図 27-1 ロールの一般的な使用方法



アプリケーション・ロール

アプリケーション・ロールには、特定のデータベース・アプリケーションを実行するために必要な権限をすべて付与します。そして、そのアプリケーション・ロールを、他のロールや特定のユーザーに対して付与します。1つのアプリケーションに対して複数の異なるロールを設定し、アプリケーション使用時のデータ・アクセスの量や範囲にあわせて異なる権限セットを各ロールに割り当てることができます。

ユーザー・ロール

ユーザー・ロールは、共通の権限要件を持つデータベース・ユーザーのグループのために作成されるロールです。ユーザーの権限は、アプリケーションのロールと権限をユーザー・ロールに付与し、そのユーザー・ロールを適切なユーザーに付与することによって管理します。

ロールのメカニズム

データベース・ロールには次の機能があります。

- ロールには、システム権限またはスキーマ・オブジェクト権限を付与できます。
- ロールには、別のロールを付与できます。ただし、ロールをそのロール自体に付与したり、循環的に付与することはできません。たとえば、ロール B があらかじめロール A に付与されている場合、ロール A をロール B には付与できません。
- 任意のロールを、任意のデータベース・ユーザーに付与できます。
- ユーザーに付与した各ロールは、任意の時点で使用可能または使用禁止にできます。ユーザーのセキュリティ・ドメインには、そのユーザーに対して現在使用可能になっているすべてのロールの権限が含まれており、ユーザーに対して現在使用禁止になっているロールの権限は除外されています。Oracle では、データベース・アプリケーションとユーザーがロールを使用可能または使用禁止にできるため、権限を選択的に使用できます。
- 間接的に付与されたロールとは、ロールに対して付与されたロールです。この種のロールは、ユーザーに対して明示的に使用可能または使用禁止にすることができます。ただし、別のロールを含んだロールを使用可能にすることによって、直接的に付与されたロールに含まれる間接的に付与された全ロールは、すべて暗黙のうちに使用可能になります。

ロールの付与と取消し

ユーザーや別のロールに対してロールを付与したり取り消すには、次の方法があります。

- Oracle Enterprise Manager の「Grant System Privileges/Roles」ダイアログ・ボックスおよび「Revoke System Privileges/Roles」ダイアログ・ボックス
- SQL 文 GRANT および REVOKE

ロールに対して権限を付与または取り消す場合にも同じオプションを使用します。また、ロールは、Oracle を実行しているオペレーティング・システムや、ネットワーク・サービスを使用することによっても、ユーザーに対して付与したり取り消したりできます。

関連項目： ロール管理の詳細は、『Oracle8i 管理者ガイド』を参照してください。

ロールの付与と取消しを実行できるユーザー

GRANT ANY ROLE システム権限を付与されたユーザーは、グローバル・ロールを除き、他のユーザーの任意のロールまたはデータベースのロールの付与または取消しを実行できます。このシステム権限は非常に強力であるため、付与するときは注意が必要です。

ADMIN OPTION 付きでロールを付与されたユーザーは、データベースの他のユーザーやロールに対してロールを付与したりそのロールを取り消すことができます。つまり、このオプションにより、選択的なロールの管理が可能になります。

関連項目： グローバル・ロールについては、『Oracle8i 分散システム』を参照してください。

ロールの命名

各ロール名はデータベース内で一意にする必要があり、ユーザー名とロール名を同一にすることはできません。スキーマ・オブジェクトとは異なり、ロールはスキーマに含まれているわけではありません。そのため、ロールを作成したユーザーを削除しても、そのロールに影響はありません。

ロールとユーザーのセキュリティ・ドメイン

各ロールと各ユーザーは、それぞれ独自のセキュリティ・ドメインを持っています。ロールのセキュリティ・ドメインには、ロールそのものに付与されている権限と、そのロールに付与されたロールに対して付与されている権限が含まれます。

ユーザーのセキュリティ・ドメインには、対応するスキーマ内のすべてのスキーマ・オブジェクトの権限、そのユーザーに対して付与されている権限、およびそのユーザーに対して付与されていて「現在使用可能」になっているロールの権限が含まれます。（1つのロールをあるユーザーに対して使用可能にし、別のユーザーに対しては使用禁止にすることもできます。）さらに、ユーザーのセキュリティ・ドメインには、ユーザー・グループ PUBLIC に対して付与されている権限とロールも含まれます。

PL/SQL ブロックとロール

PL/SQL ブロック内でのロールの使用方法は、それが無名ブロックであるか名前付きブロック（ストアド・プロシージャ、ファンクションまたはトリガー）であるか、および定義者権限と実行者権限のどちらで実行されるかに応じて異なります。

定義者権限を持つ名前付きブロック

定義者権限で実行される名前付き PL/SQL ブロック（ストアド・プロシージャ、ファンクションまたはトリガー）では、すべてのロールは使用禁止になっています。ロールは権限チェックに使用されず、定義者権限プロシージャ内ではロールを設定できません。

SESSION_ROLES ビューは、現在使用可能になっているすべてのロールを示します。定義者権限で実行される名前付き PL/SQL ブロックが SESSION_ROLES を問い合わせると、問合せは行を戻しません。

関連項目：『Oracle8i リファレンス・マニュアル』

実行者権限と無名ブロック

実行者権限で実行される名前付き PL/SQL ブロックと、無名 PL/SQL ブロックは、使用可能になっているロールを通じて付与された権限に基づいて実行されます。実行者権限を持つ PL/SQL ブロック内での権限チェックにはカレント・ロールが使用され、動的 SQL を使用してセッション中にロールを設定できます。

関連項目：

- 実行者権限と定義者権限の説明は、17-11 ページの「[ストアド・プロシージャの依存性の追跡](#)」を参照してください。
- 15-20 ページの「[PL/SQL の動的 SQL](#)」

データ定義言語の文とロール

ユーザーがデータ定義言語（DDL）文を正常に実行するには、その文に応じて1つ以上の権限が必要になります。たとえば、表を作成するには、CREATE TABLE または CREATE ANY TABLE システム権限が必要です。別のユーザーの表のビューを作成するには、CREATE VIEW または CREATE ANY VIEW システム権限のみでなく、その表に対する SELECT オブジェクト権限または SELECT ANY TABLE システム権限も必要です。

Oracle では、特定の DDL 文での特定の権限の使用を制限することによって、ロールを介して受け取った権限への依存性を回避します。次の規則は、DDL 文に対する権限の制限を示しています。

- DDL 操作の実行をユーザーに許可するすべてのシステム権限およびスキーマ・オブジェクト権限は、ロールを介して受け取った場合でも使用可能。

例：

- － システム権限：CREATE TABLE、CREATE VIEW および CREATE PROCEDURE の各権限。
- － スキーマ・オブジェクト権限：表に対する ALTER および INDEX の各権限。

例外：表に対する REFERENCES オブジェクト権限は、それがロールを介して付与された場合には、表の外部キー定義には使用できません。

- DDL 文の発行に必要な DML 操作の実行をユーザーに許可するすべてのシステム権限およびオブジェクト権限は、ロールを介して受け取った場合は使用不可。

例：

- － 表に対する SELECT ANY TABLE システム権限や SELECT オブジェクト権限がロールを介して付与されている場合、それらの権限を使用して別のユーザーの表に基づくビューを作成することはできません。

ロールを介して受け取った権限の使用許可と使用制限について、次の例で具体的に説明します。

例： 次のようなユーザーを想定します。

- CREATE VIEW システム権限を持つロールが付与されています。
- EMP 表の SELECT オブジェクト権限を含むロールが付与されていますが、この EMP 表の SELECT オブジェクト権限は間接的に付与されています。

- DEPT 表に対する SELECT オブジェクト権限が直接付与されています。

前述の権限がこのユーザーに直接的および間接的に付与されているとすると、

- このユーザーは EMP 表と DEPT 表の両方に対して SELECT 文を発行できます。
- このユーザーには EMP 表の CREATE VIEW 権限と SELECT 権限がロールを介して付与されていますが、EMP 表の SELECT オブジェクト権限がロールを介して付与されているため、EMP 表に基づく使用可能なビューは作成できません。作成されたビューにアクセスすると、エラーになります。
- CREATE VIEW 権限がロールを介して付与され、DEPT 表の SELECT 権限が直接付与されているため、DEPT 表のビューは作成できます。

事前定義済みのロール

次のロールは、Oracle データベースに対して自動的に定義されます。

- CONNECT
- RESOURCE
- DBA
- EXP_FULL_DATABASE
- IMP_FULL_DATABASE

これらのロールは、旧バージョンの Oracle との下位互換のために提供されており、Oracle データベース内の他のロールと同じ方法で変更できます。

オペレーティング・システムとロール

環境によっては、オペレーティング・システムでデータベース・セキュリティを管理できる場合もあります。オペレーティング・システムを使用して、データベース・ロールの付与と取消しや、パスワードの認証を管理できます。この機能は、すべてのオペレーティング・システムで利用できるとは限りません。

関連項目： オペレーティング・システムによるロールの管理方法の詳細は、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。

分散環境におけるロール

分散データベース環境でロールを使用する場合は、必要なすべてのロールを分散（リモート）セッションのデフォルト・ロールとして設定する必要があります。ローカル・データベース・セッション内からリモート・データベースに接続しているときは、ロールを使用可能にできません。たとえば、リモート・サイトでロールを使用可能にしようとするリモート・プロシージャは実行できません。

関連項目： 『Oracle8i 分散システム』

ファイングレイン・アクセス・コントロール

ファイングレイン・アクセス・コントロールにより、ファンクションにセキュリティ・ルールを適用し、セキュリティ・ルールを表やビューに対応付けることができます。これらのセキュリティ・ルールは、データがアクセスされるかどうか（非定型問合せなど）に関係なく、データベース・サーバーによって自動的に規定されます。

次のことができます。

- SELECT、INSERT、UPDATE および DELETE に異なるポリシーを使用します。
- 必要な場合（給与情報など）にのみセキュリティ・ルールを使用します。
- パッケージ化されたアプリケーションの最上位に基本ポリシーを作成するなど、各表に複数のポリシーを使用します。

PL/SQL パッケージ DBMS_RLS を使用すると、セキュリティ・ルールを管理できます。このパッケージを使用して、作成したポリシーを追加、削除、使用可能と使用禁止の切替えおよびリフレッシュを行います。

関連項目：

- PL/SQL パッケージの使用方法は、17-12 ページの「[パッケージ](#)」を参照してください。
- パッケージのインプリメンテーションの詳細は、『Oracle8i PL/SQL 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。

動的な述語

作成したセキュリティ・ルールを実現するファンクションまたはパッケージは、述語（WHERE 条件）を戻します。この述語によって、ポリシーで設定されたとおりにアクセスが制御されます。再書込みされた問合せは、完全に最適化され、共有可能になります。

セキュリティ・ルールの例

次のセキュリティ・ルールの例を考えます。

HR という人事管理アプリケーションでは、EMPLOYEES は ALL_EMPLOYEES 表のビューであり、どちらのオブジェクトも APPS スキーマに属しています。この表とビューを作成する文は、次のとおりです。

```
CREATE TABLE all_employees
(employee_id NUMBER(15),
 emp_name   VARCHAR2(30),
 mgr_id     NUMBER(15),
 user_name  VARCHAR2(8), .... );
CREATE VIEW employees AS SELECT * FROM all_employees;
```

社内のユーザーのロールに基づいて EMPLOYEES ビューへのアクセスを制限するために、セキュリティ・ルール・ファンクションを作成する必要があります。このポリシーの述語は、HR_ACCESS パッケージ内の SECURE_PERSON ファンクションで生成できます。このパッケージはスキーマ APPS に属し、HR アプリケーションに関連するすべてのセキュリティ・ルールをサポートするファンクションが含まれています。また、すべてのアプリケーション・コンテキストは、APPS_SEC 名前空間にあります。この例のアプリケーション・コンテキストを作成するには、次の文を使用します。

```
CREATE CONTEXT hr_role USING apps_sec.hr_role
```

セキュリティ・ルール・ファンクションを作成する文は、次のとおりです。

```
CREATE PACKAGE BODY hr_access IS
    FUNCTION secure_person(obj_schema VARCHAR2, obj_name VARCHAR2)
        RETURN VARCHAR2 IS
        d_predicate VARCHAR2(2000);
```

```

BEGIN
  IF SYS_CONTEXT ('apps_sec', 'hr_role') = 'EMP' THEN
    d_predicate = 'emp_name = sys_context(''userenv'', ''user'')';
  IF SYS_CONTEXT ('apps_sec', 'hr_role') = 'MGR' THEN
    d_predicate = 'mgr_id = sys_context(''userenv'', ''uid'')';
  ELSE
    d_predicate = '1=2'; -- deny access to other users,
                        -- may use something like 'keycol=null'
  RETURN d_predicate;
END secure_person;
END hr_access;

```

次のステップでは、EMPLOYEES ビューのポリシー PER_PEOPLE_SEC を、動的な述語を生成する HR_ACCESS.SECURE_PERSON ファンクションに対応付けます。

```

DBMS_RLS.ADD_POLICY('apps', 'employees', 'per_people_sec', 'apps'
                    'hr_access.secure_person', 'select, update, delete');

```

これで、EMPLOYEES ビューに関係する SELECT、UPDATE および DELETE 文では、アプリケーション・コンテキスト HR_ROLE の値に基づいて、3つの述語の1つが選択されます。

また、ALL_EMPLOYEES 表を保護していたのと同じセキュリティ・ルール・ファンクションを使用して、ADDRESSES 表を保護する動的な述語を生成できるので注意してください。この2つの表には、データへのアクセスを制限するために同じポリシーが適用されているためです。

関連項目： セキュリティ・ルールの設定方法の詳細は、『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。

アプリケーション・コンテキスト

アプリケーション・コンテキストにより、ファイングレイン・アクセス・コントロールの実現が容易になります。また、関数にセキュリティ・ルールを実装し、そのセキュリティ・ルールをアプリケーションに対応付けることができます。各アプリケーションには、固有のコンテキストを対応付けることができます。ユーザーは、そのコンテキストを任意に (SQL*Plus などを通じて) 変更することは許されません。

アプリケーション・コンテキストにより、アプリケーションに関する属性に基づき、パラメータベースの柔軟なアクセス制御が可能になります。たとえば、人事管理アプリケーションのコンテキスト属性には「職位」、「部門」および「国」などを含め、受注管理アプリケーションの属性には「顧客番号」や「営業地区」などを含めることができます。

次のことができます。

- コンテキスト値の基礎に述語を使用します。
- 述語内でコンテキスト値をバインド変数として使用します。
- ユーザー属性を設定します。
- ユーザー属性にアクセスします。

アプリケーション・コンテキストを定義する手順

1. アプリケーションのコンテキストを検査して設定する関数を使用して、PL/SQL パッケージを作成します。ログイン時にイベント・トリガーを使用して、ログイン・ユーザーの初期コンテキストを設定できます。
2. CREATE CONTEXT を使用して一意のコンテキスト名を指定し、作成した PL/SQL パッケージに対応付けます。
3. 次のどちらかの操作を行います。
 - ファイングレイン・アクセス・コントロールを実装するポリシー関数内で、アプリケーション・コンテキストを参照します。
 - ログイン時にイベント・トリガーを使用して、ユーザーの初期コンテキストを設定します。たとえば、ユーザーの従業員番号を問い合せて、「従業員番号」コンテキスト値として設定できます。
4. アプリケーション・コンテキストを参照します。

関連項目：

- 『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』

この章では、Oracle の監査機能について説明します。この章の内容は次のとおりです。

- [監査の概要](#)
- [文監査](#)
- [権限監査](#)
- [スキーマ・オブジェクト監査](#)
- [文、権限およびスキーマ・オブジェクトの監査対象の限定](#)

監査の概要

「監査」とは、選択したユーザー・データベース・アクションを監視して記録する処理のことです。監査は、通常次のような目的で使用されます。

- 動作が不審なアクティビティの調査。たとえば、無許可ユーザーが表からデータを削除しようとした場合、セキュリティ管理者は、そのデータベースへのすべての接続と、そのデータベースにあるすべての表からの行の削除（成功および失敗）をすべて監査できます。
- 特定のデータベース・アクティビティに関するデータの監視と収集。たとえば、データベース管理者は、更新された表、実行された論理 I/O の回数、またはピーク時に接続していた同時実行ユーザーの数などに関する統計を収集できます。

監査機能

ここでは、Oracle の監査メカニズムの概要について説明します。

監査のタイプ

Oracle は次の 3 つのタイプの一般監査機能をサポートします。

文監査	SQL 文が処理する特定のスキーマ・オブジェクトではなく、文のタイプに関してのみ実行する SQL 文の選択的な監査。文監査のオプションは通常、適用範囲が広く、オプションごとに何種類かの関連したアクションの使用方法を監査します。たとえば、AUDIT TABLE は、それがどの表に対して発行されたかには関係なく、いくつかの DDL 文を追跡します。文監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
権限監査	AUDIT CREATE TABLE など、システム・アクションを実行する強力なシステム権限の使用方法を監査する選択的な監査。権限監査は、対象となる権限の使用方法のみを監査するため、監査対象が文監査より限定されています。権限監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
スキーマ・オブジェクト監査	AUDIT SELECT ON EMP など、特定のスキーマ・オブジェクトの特定の文に対する選択的な監査。スキーマ・オブジェクト監査は対象が非常に限定されており、特定のスキーマ・オブジェクトに対する特定の文のみを監査します。スキーマ・オブジェクト監査は、データベースのすべてのユーザーに常に適用されます。

監査の対象

Oracle では、監査オプションの対象範囲を広くしたり狭くできます。次の監査ができます。

- 文の正常な実行、失敗した実行、またはその両方
- ユーザー・セッションごと、または文が実行されるごとの文の実行
- すべてのユーザーまたは特定のユーザーのアクティビティ

監査レコードと監査証跡

監査レコードには、監査された操作、操作を実行したユーザーおよび操作の日時などの情報が記録されます。監査レコードは、「データベース監査証跡」と呼ばれるデータ・ディクショナリ表か、オペレーティング・システムの監査証跡のどちらかに格納できます。

データベースの監査証跡は、各 Oracle データベースのデータ・ディクショナリの SYS スキーマにある AUD\$ という名前の単一の表です。この表の情報を使用しやすくするため、複数の事前定義済みのビューが提供されています。

監査対象のイベントと設定されている監査オプションに応じて、監査証跡には様々な種類の情報が記録されます。ただし、次の情報は、特定の監査アクションにとって意味がある限り、それぞれの監査証跡レコードに常に記録されます。

- ユーザー名
- セッション識別子
- 端末識別子
- アクセスされたスキーマ・オブジェクトの名前
- 実行または試行された操作
- 操作の完了コード
- 日時のタイム・スタンプ
- 使用されたシステム権限

オペレーティング・システム監査証跡はコード化されていて読むことはできませんが、次のようなデータ・ディクショナリ・ファイルとエラー・メッセージにデコードできます。

アクション・コード	実行または試行された操作の記述。これらのコードとその記述のリストは、AUDIT_ACTIONS データ・ディクショナリ表にあります。
使用された権限	操作の実行に使用されたシステム権限の記述。これらのコードとその記述は、SYSTEM_PRIVILEGE_MAP 表にあります。
完了コード	試行された操作の結果の記述。成功した操作からは、値 0 が戻されます。失敗した操作からは、操作が異常終了した理由を説明する Oracle エラー・コードが戻されます。

関連項目：

- 事前定義済のビューの作成方法と使用方法は、『Oracle8i 管理者ガイド』を参照してください。
- 完了コードのリストは、『Oracle8i エラー・メッセージ』を参照してください。

監査のメカニズム

この項では、Oracle の監査機能で使用されているメカニズムについて説明します。

監査レコードが生成される場合

監査情報の記録機能は、使用可能または使用禁止にできます。この機能により、許可されたデータベース・ユーザーは監査オプションをいつでも設定できますが、監査情報の記録を制御する操作はセキュリティ管理者のために確保されています。

データベースの監査機能が使用可能になっている場合、監査レコードは文実行の実行フェーズで生成されます。

PL/SQL プログラム・ユニット内の SQL 文は、プログラム・ユニットの実行時に必要に応じて監査されます。

監査証跡レコードの生成と挿入は、ユーザーのトランザクションからは独立して実行されるため、ユーザーのトランザクションがロールバックされても、監査証跡レコードはコミットされたままになります。

注意： 監査レコードは、ユーザー SYS によって確立されたセッションや管理者権限での接続では生成されません。このようなユーザーによる接続では、Oracle の特定の内部機能をバイパスして、特定の管理操作（データベースの起動、停止、リカバリなど）を実行できます。

関連項目：

- 監査を使用可能および使用禁止にする手順は、『Oracle8i 管理者ガイド』を参照してください。
- SQL 文処理の様々なフェーズや共有 SQL については、[第 15 章「SQL と PL/SQL」](#)を参照してください。

オペレーティング・システムの監査証跡に必ず記録されるイベント

データベース監査が使用可能であるかどうかに関係なく、Oracle はある種のデータベース関連アクションをオペレーティング・システムの監査証跡に常に記録します。

インスタンス起動

インスタンスを起動したオペレーティング・システム・ユーザー、そのユーザーの端末識別子、日時のタイム・スタンプ、およびデータベース監査が使用可能になっていたかどうかを記述した監査レコードが生成されます。データベース監査証跡は起動が正常に完了しないと使用可能にならないため、この情報はオペレーティング・システム監査証跡に記録されます。起動時のデータベース監査の状態も記録されるため、管理者が、データベース監査使用禁止の状態のままデータベースを再起動して、監査されないアクションを実行できないようになっています。

インスタンス停止

インスタンスを停止したオペレーティング・システム・ユーザー、そのユーザーの端末識別子および日時のタイム・スタンプを記述した監査レコードが生成されます。

管理者権限によるデータベースへの接続

Oracle に管理者権限によって接続するオペレーティング・システム・ユーザーの詳細を記述した監査レコードが生成されます。この監査レコードにより、管理者権限によって接続したユーザーに関する情報が提供されます。

監査証跡が Oracle からはアクセスできるようになっていないオペレーティング・システムの場合、これらの監査証跡レコードは、バックグラウンド・プロセスのトレース・ファイルと同じディレクトリにある Oracle 監査証跡ファイルに格納されます。

関連項目： オペレーティング・システムの監査証跡の詳細は、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。

監査オプションが有効になる時期

データベース・ユーザーがデータベースに接続した時点で有効になっていた文監査オプションと権限監査オプションは、そのセッションの持続期間中は有効です。文監査または権限監査のオプションの設定または変更の結果は、セッション中には有効になりません。修正した文監査オプションまたは権限監査オプションは、カレント・セッションを終了し、新しいセッションを作成する時点で有効になります。一方、オブジェクト監査オプションについて変更した内容は、カレント・セッションでただちに有効になります。

分散データベースの監査

監査機能はサイト自律性を持っています。つまり、インスタンスは、直接接続しているユーザーが発行する文のみを対象に監査します。ローカル Oracle ノードは、リモート・データベースで発生するアクションを監査できません。リモート接続はデータベース・リンクのユーザー・アカウントを介して確立されるため、リモート Oracle ノードはデータベース・リンクの接続を介して発行された文を監査します。

関連項目： 第 30 章「分散データベースの概念」

オペレーティング・システム監査証跡に対する監査

オペレーティング・システムの監査証跡が使用可能になっている場合に、Oracle は監査証跡レコードをオペレーティング・システムの監査証跡に送ることができます。その他のオペレーティング・システムの場合、これらの監査レコードが、他の Oracle トレース・ファイルと似た形式でデータベース外のファイルに書き込まれることもあります。

注意： この機能がオペレーティング・システムで実装されているかどうかを調べるには、プラットフォーム固有の Oracle マニュアルを参照してください。

Oracle では、オペレーティング・システムの監査証跡（または監査レコードを含むオペレーティング・システム・ファイル）に監査レコードを記録できない場合にも、常に特定のアクションを引き続き監査できます。オペレーティング・システムの監査証跡に記録できないのは、多くの場合、オペレーティング・システムの監査証跡またはファイル・システムがいっぱいになっており、新しいレコードが入らないことが原因です。

オペレーティング・システム監査を構成するシステム管理者は、監査証跡またはファイル・システムがいっぱいにならないようにしてください。ほとんどのオペレーティング・システムでは、このような状況を回避できるように十分な情報と警告が管理者に提供されます。ただし、データベースの監査証跡を使用するように監査を構成すれば、その危険性を回避することに注意してください。監査証跡が文に関するデータベース監査レコードを受け入れられない場合には、監査されているイベントの発生が Oracle Server によって防止されるためです。

文監査

文監査とは、関連する文のグループに対する選択的な監査のことです。文は、次の2つに分類できます。

- 特定のタイプのデータベース構造体やスキーマ・オブジェクトに関する DDL 文。ただし、監査の対象として特定の構造体やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT TABLE はすべての CREATE TABLE 文と DROP TABLE 文を監査します）。
- 特定のタイプのデータベース構造体やスキーマ・オブジェクトに関する DML 文。ただし、監査の対象として特定の構造体やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT SELECT TABLE は、表、ビューまたはスナップショットのどれであるかに関係なく、すべての SELECT ... FROM TABLE/VIEW/SNAPSHOT 文を監査します）。

文監査では、監査の対象範囲を広げてすべてのデータベース・ユーザーのアクティビティを監査したり、範囲を限定して特定のデータベース・ユーザーのアクティビティのみを監査できます。

権限監査

権限監査は、システム権限の使用を許可されている文を選択的に監査します。たとえば、SELECT ANY TABLE システム権限の監査は、SELECT ANY TABLE システム権限を使用して実行されるユーザーの文を監査します。任意のシステム権限の使用を監査できます。

権限監査では、システム権限の監査の前に、所有者権限とスキーマ・オブジェクト権限が必ず検査されます。所有者権限とスキーマ・オブジェクト権限がアクションを許可するのに十分である場合、そのアクションは監査されません。

類似の文監査オプションと権限監査オプションを両方設定しても、監査レコードは1つしか生成されません。たとえば、文の句 TABLE とシステム権限の CREATE TABLE の両方を監査すると、表が作成されるたびに監査レコードが1つのみ生成されます。

権限監査のそれぞれのオプションでは、特定のタイプの文のみが監査され、関連する一連の文が監査されるわけではないため、権限監査の対象は文監査よりも限定されます。たとえば、文監査句 TABLE では CREATE TABLE、ALTER TABLE および DROP TABLE 文が監査されますが、権限監査オプション CREATE TABLE では CREATE TABLE 文のみが監査されます。これは、CREATE TABLE 権限を必要とするのが CREATE TABLE 文のみであるためです。

文監査と同様に、権限監査では、すべてのデータベース・ユーザーのアクティビティを監査したり、特定のデータベース・ユーザーのアクティビティのみを監査できます。

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、特定のスキーマ・オブジェクトの特定の DML 文（問合せを含む）、および GRANT 文と REVOKE 文の選択的な監査です。スキーマ・オブジェクト監査では、特定の表に対する SELECT 文や DELETE 文など、スキーマ・オブジェクト権限によって許可される操作と、それらの権限を制御する GRANT 文と REVOKE 文が監査されます。

表、ビュー、順序、「スタンドアロン」のストアド・プロシージャとストアド・ファンクションおよびパッケージを参照する文を監査できます。パッケージ内のプロシージャは、個別には監査できません。

クラスタ、データベース・リンク、索引またはシノニムを参照する文は、直接監査できません。ただし、ベース表に影響を与える操作を監査することにより、これらのスキーマ・オブジェクトへのアクセスを間接的に監査できます。

スキーマ・オブジェクト監査オプションは、常にすべてのデータベース・ユーザーに対して設定されます。これらのオプションは、特定のユーザーに対しては設定できません。監査可能なすべてのスキーマ・オブジェクトに対して、デフォルトのスキーマ・オブジェクト監査オプションを設定できます。

関連項目： 監査できるスキーマ・オブジェクトの詳細は、『Oracle8i SQL リファレンス』を参照してください。

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

ビューとプロシージャ（ストアド・ファンクション、パッケージおよびトリガーを含む）は、その定義の基礎を形成するスキーマ・オブジェクトを参照します。そのため、ビューとプロシージャに関する監査には、いくつかの固有の特性があります。ビューやプロシージャを使用すると、その結果として複数の監査レコードが生成される可能性があります。ビューやプロシージャの使用は、使用可能になっている監査オプションに依存します。ビューやプロシージャを使用した結果として発行される SQL 文は、ベース・スキーマ・オブジェクトの使用可能監査オプション（デフォルトの監査オプションも含む）に依存します。

次の一連の SQL 文について考えます。

```
AUDIT SELECT ON emp;  
  
CREATE VIEW emp_dept AS  
  SELECT empno, ename, dname  
     FROM emp, dept  
    WHERE emp.deptno = dept.deptno;  
  
AUDIT SELECT ON emp_dept;  
  
SELECT * FROM emp_dept;
```

この EMP_DEPT に対する問合せの結果、2つの監査レコードが生成されます。1つは EMP_DEPT ビューの問合せについての監査レコード、もう1つはベース表 EMP の問合せ（EMP_DEPT ビューを介した間接的な問合せ）についての監査レコードです。ベース表の SELECT 監査オプションが使用可能になっていないため、ベース表 DEPT に対して問合せを実行しても監査レコードは生成されません。すべての監査レコードは、EMP_DEPT ビューの問合せを発行したユーザーに属します。

ビューやプロシージャの監査オプションは、そのビューまたはプロシージャを最初に使用して共有プールに配置する時点で、判別されます。そのビューまたはプロシージャをいったんフラッシュして共有プールに再配置するまで、これらの監査オプションは設定されたままになります。スキーマ・オブジェクトを監査すると、キャッシュ内のスキーマ・オブジェクトが無効になるため、スキーマ・オブジェクトを再ロードすることになります。ベース・スキーマ・オブジェクトの監査オプションに対する変更は、共有プール内のビューとプロシージャには認識されません。

前述の例の続きとして、EMP 表に対する SELECT 文の監査をオフにすると、EMP_DEPT ビューを使用しても EMP 表の監査レコードは生成されなくなります。

文、権限およびスキーマ・オブジェクトの監査対象の限定

Oracle では、文、権限およびスキーマ・オブジェクトのそれぞれの監査の対象を、次の3つの分野に限定できます。

- 監査される SQL 文の正常な実行と失敗した実行
- BY SESSION 監査と BY ACCESS 監査
- データベースの特定のユーザーまたはすべてのユーザー（文監査と権限監査のみ）

文の正常な実行と失敗した実行の監査

文、権限およびスキーマ・オブジェクトの監査では、文の正常な実行、失敗した実行、またはその両方の文実行を選択的に監査できます。このため、監査する文が成功しなかった場合にも、アクションを監査できます。

失敗した文の実行を監査できるのは、有効な SQL 文を実行しても適切な認可がないために失敗した場合や、存在しないスキーマ・オブジェクトを参照したために失敗した場合のみです。有効でなかったために失敗した文は監査できません。たとえば、失敗した文の実行を監査するように権限監査オプションが設定されている場合は、対象のシステム権限を使用しても他の原因で失敗した文が監査されます（たとえば CREATE TABLE を設定したが、指定した表領域に対する割当て制限を設定していなかったために CREATE TABLE 文が失敗した場合など）。

どちらの形式の AUDIT 文にも、次の句を指定できます。

- WHENEVER SUCCESSFUL 句。監査対象の文の正常な実行のみを監査する場合に使用します。
- WHENEVER NOT SUCCESSFUL 句。監査対象の文の失敗した実行のみを監査する場合に使用します。
- 前述のどちらの句も使用しません。監査対象の文の正常な実行と失敗した実行の両方を監査する場合に使用します。

BY SESSION 監査と BY ACCESS 監査

ほとんどの監査オプションでは、1つのユーザー・セッションで監査対象の文が複数回発行された場合の監査レコードの生成方法を指定できます。ここでは、AUDIT 文の BY SESSION 句と BY ACCESS 句の相違点について説明します。

関連項目：『Oracle8i SQL リファレンス』

BY SESSION

どのタイプの監査（スキーマ・オブジェクト、文、権限）の場合も、BY SESSION は、監査されるアクションが含まれているセッション中に、ユーザーおよびスキーマ・オブジェクト当たり1つの監査レコードのみを監査証跡に挿入します。

セッションとは、ユーザーが Oracle データベースに接続してから切断するまでの期間です。

例 1 この例では、次のような状況を想定します。

- SELECT TABLE 文監査オプションを BY SESSION に設定します。
- JWARD でデータベースに接続し、DEPT という表に対して5つの SELECT 文を発行した後、そのデータベースとの接続を切断します。
- SWILLIAMS でデータベースに接続し、表 EMP に対して3つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

この場合、監査証跡には8つの SELECT 文に対して2件（SELECT 文を発行したセッションごとに1件）の監査レコードが記録されます。

例 2 別の例として、次の条件を想定します。

- SELECT TABLE 文監査オプションを BY SESSION に設定します。
- JWARD でデータベースに接続し、DEPT という表に対して 5 つの SELECT 文、表 EMP に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

この場合、監査証跡には、ユーザーがセッション中に SELECT 文を発行した各スキーマ・オブジェクトごとに 1 件ずつ、2 件のレコードが記録されます。

注意： オペレーティング・システムの監査証跡に監査レコードを記録する際に BY SESSION 句を使用すると、Oracle ではアクセスするたびに監査レコードが生成され格納されます。したがって、この監査構成の場合、BY SESSION は BY ACCESS と等価です。

BY ACCESS

監査を BY ACCESS に設定すると、カーソル内で監査対象の操作が実行されるたびに、監査証跡に監査レコードが 1 つ挿入されます。カーソルを再使用させるイベントは、次のとおりです。

- カーソルを再使用するためにオープンしておく、Oracle Forms などのアプリケーション
- 新しいバインド変数を使用してカーソルを再実行すること
- PL/SQL ループ内で実行される文で、1 つのカーソルを再使用するために PL/SQL エンジンにより文が最適化される場合

監査は、カーソルが共有されているかどうかの影響を受けないため注意してください。それぞれのユーザーは、カーソルの最初の実行時に自分の監査証跡レコードを作成します。

例 次のような場合を想定します。

- SELECT TABLE 文監査オプションを BY ACCESS に設定します。
- JWARD でデータベースに接続し、DEPT という表に対して 5 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。
- SWILLIAMS でデータベースに接続し、表 DEPT に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

1 つの監査証跡に、8 つの SELECT 文に対応する 8 件のレコードが記録されます。

デフォルトと除外される操作

AUDIT 文には、BY SESSION と BY ACCESS のどちらかを指定できます。ただし、次のような一部の監査オプションでは、BY ACCESS しか設定できません。

- DDL 文を監査するすべての文監査オプション
- DDL 文を監査するすべての権限監査オプション

他のすべての監査オプションでは、デフォルトで BY SESSION が設定されています。

ユーザー別の監査

文監査オプションと権限監査オプションでは、任意のユーザーが発行した文を監査するか、特定のユーザー・リストに含まれるユーザーが発行する文を監査するかを選択できます。特定のユーザーに限定すると、生成される監査レコードの数は最小限に抑えられます。

例 表やビューを問い合わせたり更新するためにユーザー SCOTT および BLAKE によって発行される文を監査するには、次の文を発行します。

```
AUDIT SELECT TABLE, UPDATE TABLE  
  BY scott, blake;
```

関連項目： ユーザー別の監査の詳細は、『Oracle8i SQL リファレンス』を参照してください。

データベースのリカバリ

この章では、データベースのリカバリ時に使用される構造、バックアップおよびリカバリの操作を簡単にできるようにする Recovery Manager ユーティリティ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・リカバリの概要
- データベースのリカバリで使用される構造
- ロールフォワードとロールバック
- リカバリ・パフォーマンスの改善
- Recovery Manager
- データベースのアーカイブ・モード
- 制御ファイル
- データベースのバックアップ
- 耐障害性

関連項目：

- バックアップおよびリカバリ構造の作成とメンテナンスに必要な手順については、『Oracle8i Recovery Manager 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。
- スタンバイ・データベースの詳細は、『Oracle8i スタンバイ・データベース 概要および管理』を参照してください。

データベース・リカバリの概要

データベース管理者の主な責任の1つは、ハードウェア、ソフトウェア、ネットワーク、プロセスまたは発生する可能性のある障害に対処する準備をしておくことです。そのような障害がデータベース・システムの操作に影響する場合、通常は速やかにデータベースをリカバリし、通常の操作に戻る必要があります。リカバリ作業では、データベースとその関連ユーザーを不要な問題から保護し、同じ作業を手動で繰り返さずにすむようにする必要があります。

リカバリ・プロセスは、発生した障害のタイプ、その障害の影響を受けた構造および実行するリカバリのタイプに応じて異なります。消失または破損したファイルがなければ、インスタンスを再起動するだけでリカバリできます。データが失われている場合は、リカバリには追加ステップが必要になります。

注意： Recovery Manager は、バックアップとリカバリの操作を容易にするユーティリティです。

関連項目：

- 29-16 ページの「[Recovery Manager](#)」
- Recovery Manager の詳細とデータ消失からのリカバリ方法は、『Oracle8i Recovery Manager 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。

エラーと障害

いくつかの問題が原因で、Oracle データベースの通常の操作が停止したり、ディスクへのデータベース I/O が影響を受けることがあります。この後の項では、最も一般的なタイプの障害について説明します。障害によっては、リカバリが自動的に実行される場合や、データベース・ユーザーやデータベース管理者による処置がほとんど、またはまったく必要ない場合もあります。

ユーザー・エラー

データベース管理者は、表を誤って削除してしまうなどのユーザー・エラーを防ぐことはできません。通常は、ユーザーがデータベースとアプリケーションの原理を理解すれば、ユーザー・エラーは減少します。また、管理者が事前に効果的なリカバリ方法を準備しておく、様々なタイプのユーザー・エラーのリカバリに必要な作業を軽減できます。

文障害

Oracle プログラムで文を処理するときに論理的な障害があると、文障害が発生します。たとえば、表のすべてのエクステント（CREATE TABLE 文の MAXEXTENTS パラメータに指定されている数のエクステント）がすでに割り当てられており、データでいっぱいになっている、つまり表が完全にいっぱいであるとしめます。このような場合に有効な INSERT 文を実行しても、使用可能な領域がないため行は挿入できません。このため、そのような文を発行するとエラーになります。

文障害が発生すると、Oracle ソフトウェアまたはオペレーティング・システムからエラー・コードまたはエラー・メッセージが戻されます。文障害では、通常、アクションやリカバリ・ステップは必要ありません。つまり、文の結果をロールバックし（結果がある場合）、アプリケーションに制御を戻すことによって、Oracle が自動的に文障害を訂正します。ユーザーは、エラー・メッセージが示している問題を訂正してから文を再実行するだけです。

プロセス障害

プロセス障害は、接続の切断やプロセスの異常終了など、データベース・インスタンスのユーザー、サーバーまたはバックグラウンド・プロセスの障害です。プロセス障害が発生した場合、そのデータベース・インスタンスの他のプロセスは続行できますが、エラーとなった従属プロセスは続行できません。

Oracle バックグラウンド・プロセス PMON は、異常終了した Oracle プロセスを検出します。ユーザー・プロセスまたはサーバー・プロセスが異常終了した場合、PMON は、異常終了したプロセスのカレント・トランザクションをロールバックし、そのプロセスが使用していたリソースを解放して、この障害を解決します。エラーになったユーザー・プロセスまたはサーバー・プロセスは、自動的にリカバリされます。異常終了したプロセスがバックグラウンド・プロセスである場合、通常、インスタンスは正常に機能できなくなります。このため、インスタンスを停止して再起動する必要があります。

ネットワーク障害

システムでネットワーク（ローカル・エリア・ネットワークや電話回線など）を使用してクライアント・ワークステーションをデータベース・サーバーに接続したり、複数のデータベース・サーバーを接続して分散データベース・システムを形成している場合に、ネットワーク障害（電話接続の異常終了やネットワーク通信ソフトウェアの障害など）が発生すると、データベース・システムの通常の操作に割込みが発生することがあります。たとえば、次のとおりです。

- クライアント・アプリケーションの正常な実行に割り込み、プロセス障害を引き起こす。前の項で説明したとおり、この場合は Oracle バックグラウンド・プロセス PMON が、接続を切断されたユーザー・プロセスに対応する、異常終了したサーバー・プロセスを検出し、そのサーバー・プロセスを解決します。
- 分散トランザクションの 2 フェーズ・コミットに割り込む。ネットワークの問題が訂正されると、この問題に関連していたそれぞれのデータベース・サーバーの Oracle バックグラウンド・プロセス RECO が、分散データベース・システムのすべてのノードにある未解決の分散トランザクションを、自動的に解決します。

関連項目： 第 30 章「分散データベースの概念」

データベース・インスタンス障害

Oracle データベース・インスタンスの作業の続行を妨げる問題が発生すると、データベース・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上の問題や、オペレーティング・システム・クラッシュなどのソフトウェア上の問題によるものです。また、SHUTDOWN ABORT 文や STARTUP FORCE 文を発行した場合にも、インスタンス障害が発生します。

インスタンス障害からのリカバリ クラッシュ・リカバリまたはインスタンス・リカバリでは、データベースは、インスタンス障害が発生する直前のトランザクション一貫性状態にリカバリされます。「クラッシュ・リカバリ」では単一インスタンス構成のデータベースをリカバリし、「インスタンス・リカバリ」では Oracle Parallel Server 構成でのデータベースをリカバリします。Oracle Parallel Server データベースのすべてのインスタンスがクラッシュすると、Oracle はクラッシュ・リカバリを実行します。

インスタンス障害からのリカバリは自動的に実行され、DBA が介入する必要はありません。たとえば、Oracle Parallel Server を使用している場合、障害インスタンスのインスタンス・リカバリは、他のインスタンスによって実行されます。単一インスタンス構成では、データベースの再起動時にデータベースのクラッシュ・リカバリが実行されます。つまり、新しいインスタンスにマウントされ、オープンされます。必要であれば、マウント状態からオープン状態に推移する時点で、クラッシュ・リカバリが自動的にトリガーされます。

クラッシュまたはインスタンス・リカバリは、次のようなステップで構成されています。

1. データ・ファイルには記録されていませんが、オンライン REDO ログには記録されているデータをロールフォワードし、ロールバック・セグメントの内容も適用して、データ・ファイルをリカバリします。これを「キャッシュ・リカバリ」と呼びます。
2. データベースをオープンします。Oracle は、すべてのトランザクションがロールバックされるのを待ってからデータベースを使用可能にするのではなく、キャッシュ・リカバリが完了した時点でデータベースをオープンできるようにします。まだリカバリされていないトランザクションによってロックされているデータ以外は、すぐに使用可能になります。
3. 障害発生時にアクティブであったシステム全体のトランザクションすべてを DEAD とマークし、これらのトランザクションを含むロールバック・セグメントを PARTLY AVAILABLE とマークします。
4. SMON によるリカバリの一部として、デッド・トランザクションをロールバックします。これを「トランザクション・リカバリ」と呼びます。

5. インスタンス障害の発生時に 2 フェーズ・コミットが行われていたために保留になっている分散トランザクションを解決します。
6. 新規トランザクションでは、デッド・トランザクションによってロックされている行を検出すると、デッド・トランザクションを自動的にロールバックしてロックを解除できます。ファースト・スタート・リカバリを使用している場合は、トランザクション全体ではなく、データ・ブロックのみが即時にロールバックされます。

関連項目：

- インスタンス・リカバリの詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』を参照してください。
- インスタンス・リカバリのチューニングの説明は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

メディア（ディスク）障害

Oracle データベースの操作に必要なファイルの書き込みまたは読み込み時にエラーが発生することがあります。記憶域メディア上の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、このような障害は「メディア障害」と呼ばれます。

一般的なメディア障害に、ディスク・ドライブのすべてのファイルが失われるディスク・ヘッド・クラッシュがあります。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など、データベースの関連ファイルはすべてディスク・クラッシュの影響を受けます。

メディア障害からのリカバリの方法は、関係しているファイルによって異なります。

関連項目：

- リカバリ方法の説明は、『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。
- 『Oracle8i Recovery Manager ユーザーズ・ガイドおよびリファレンス』

メディア障害がデータベース操作に及ぼす影響 メディア障害はデータ・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など、Oracle データベースの操作に必要な様々なタイプのファイルに影響します。

オンライン REDO ログ・ファイルや制御ファイルでメディア障害が発生した場合、オンライン REDO ログ・ファイルや制御ファイルが「多重化」されているかどうかに応じて、それ以降のデータベース操作が異なります（多重化することをお勧めします）。オンライン REDO ログ・ファイルや制御ファイルの多重化とは、そのファイルの 2 次コピーを保持することを意味します。メディア障害によって 1 つのディスクが破損した場合、多重化されたオンライン REDO ログがあれば、データベースの操作は通常、割込みなしで続行されます。多重化されていないオンライン REDO ログが破損すると、データベース操作が停止し、データが永久に失われる可能性があります。制御ファイルが破損すると、ファイルが多重化

されているかどうかにかかわらず、Oracle が破損した制御ファイルに対して読み込みまたは書き込みを試行した時点でデータベース操作は停止します。この状況は、頻繁に起きるすべてのチェックポイントやログ・スイッチなどで発生します。

データ・ファイルに影響するメディア障害は、読み込みエラーと書き込みエラーの 2 つに分類されます。読み込みエラーの場合、Oracle がデータ・ファイルを読み込めないことを発見すると、オペレーティング・システムのエラーと、ファイルが見つからない、オープンできない、または読み込めないことを示す Oracle のエラーがアプリケーションに戻されます。Oracle は続行されますが、このエラーは読み込みエラーが発生するたびに戻されます。次のチェックポイントでは、Oracle が標準的なチェックポイント・プロセスの一部としてファイル・ヘッダーを書き込もうとすると、書き込みエラーが発生します。

Oracle がデータ・ファイルに書き込めなかった場合に ARCHIVELOG モードで動作していると、エラーが DBW_n トレース・ファイルに戻され、そのデータ・ファイルが自動的にオフライン化されます。自動的にオフライン化されるのは、書き込み不能のデータ・ファイルのみです。そのファイルが入っている表領域はオンラインのままになります。

書き込み不能のデータ・ファイルが SYSTEM 表領域に入っている場合、そのデータ・ファイルはオフライン化されません。そのかわりにエラーが戻され、Oracle によりインスタンスが停止されます。Oracle が正常に機能するには、SYSTEM 表領域のすべてのファイルがオンラインになっている必要があるため、この例外が適用されます。同様の理由で、アクティブ・ロールバック・セグメントを含む表領域のデータ・ファイルはオンラインである必要があります。

Oracle がデータ・ファイルに書き込めず、データがいっぱいになったオンライン REDO ログ・ファイルをアーカイブしていない場合は、DBW_n バックグラウンド・プロセスとカレント・インスタンスがエラーになります。問題が一時的なもの（ディスク・コントローラの電源が切れたなど）であれば、通常はオンライン REDO ログ・ファイルを使用したクラッシュまたはインスタンス・リカバリが実行可能で、この場合はインスタンスを再起動できます。ただし、データ・ファイルが修復不可能に破損し、アーカイブされていない場合は、最新の一貫性バックアップからデータベース全体をリストアする必要があります。

読取り専用表領域のリカバリ クラッシュまたはインスタンス・リカバリ時には、読取り専用データ・ファイルのリカバリは必要ありません。起動時のリカバリでは、オンラインの読取り専用ファイルそれぞれに関してメディア・リカバリの必要がないことが確認されます。つまり、そのファイルは、読取り専用になる前に作成されたバックアップからはリストアされなかったということです。読取り専用の表領域を、その表領域が読取り専用になる前に作成されたバックアップからリストアした場合は、メディア・リカバリが完了するまでその表領域にはアクセスできません。

データベースのリカバリで使用される構造

Oracle データベースのいくつかの構造により、データは障害から保護されます。この項では、データベース・リカバリの構造と役割について簡単に説明します。

データベースのバックアップ

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物理ファイル（すべてのデータ・ファイルと制御ファイル）のバックアップによって構成されます。メディア障害からのメディア・リカバリを開始するために、Oracle はバックアップ・ファイルを使用して、破損したデータ・ファイルまたは制御ファイルをリストアします。破損したと思われるデータ・ファイル、表領域またはデータベースのカレント・コピーをバックアップ・コピーと置き換えることを、データベースのその部分を「リストアする」と呼びます。

Oracle では、次のように、データベース・バックアップを実行するためのオプションがいくつか提供されています。

- Recovery Manager
- オペレーティング・システム・ユーティリティ
- Export ユーティリティ
- Enterprise Backup Utility

関連項目： 『Oracle8i バックアップおよびリカバリ・ガイド』

REDO ログ

それぞれの Oracle データベース・インスタンスに存在する REDO ログには、Oracle データベースでのすべての変更の内容が記録されます。データベースの REDO ログは、実際にデータベースのデータが格納されるデータ・ファイルとは別個の、最低 2 つの REDO ログ・ファイルによって構成されます。インスタンス障害やメディア障害が発生した場合、データベース・リカバリの一環として、REDO ログに記録されている変更がデータ・ファイルに適用されて、データベースのデータは障害が発生した時点の状態に更新されます。

データベースの REDO ログは、オンライン REDO ログとアーカイブ REDO ログの 2 つの部分から構成されている場合があります。

オンライン REDO ログ

それぞれの Oracle データベースには、対応付けられたオンライン REDO ログがあります。Oracle バックグラウンド・プロセス LGWR は、オンライン REDO ログを使用して、対応付けられているインスタンスによって実行されたすべての変更を即時に記録します。各 REDO レコードには、新旧両方の値が含まれています。また、元の値はロールバック・ブロックにも記録されます。オンライン REDO ログは 2 つ以上の事前割当て済ファイルで構成され、これらのファイルが循環方式で再使用され、進行中のデータベース変更が記録されます。

アーカイブ REDO ログ

必要に応じて、オンライン REDO ログがいっぱいになった時点でそのファイルをアーカイブするための、Oracle データベースを構成できます。アーカイブ対象のオンライン REDO ログ・ファイルは、ログ順序番号によって一意に識別され、アーカイブ REDO ログを構成

します。事前割当て済のオンライン REDO ログ・ファイルを繰り返し再使用して最新のデータベース変更を格納する一方で、いっぱいになったオンライン REDO ログ・ファイルをアーカイブすることにより、メディア・リカバリなどの操作用に過去の REDO ログ情報を保存します。

データ・ファイルは、バックアップからリストアされた場合や、クリーン・データベース停止でクローズされていない場合は、最新のものでない可能性があります。これらのデータ・ファイルには、アーカイブ REDO ログまたはオンライン REDO ログ（あるいはその両方）に記録された変更を適用して更新する必要があります。このプロセスを「リカバリ」と呼びます。

関連項目： 29-18 ページの「データベースのアーカイブ・モード」

ロールバック・セグメント

ロールバック・セグメントは、Oracle データベースの操作で実行される多くの機能で使用されます。一般に、データベースのロールバック・セグメントには、コミットされていないトランザクションについて、進行中のトランザクションによって変更された古いデータ値が格納されます。

特に、ロールバック・セグメントの情報は、データベースのリカバリ時に REDO ログからデータ・ファイルに適用された、コミットされていない変更をすべて取り消すために使用されます。このため、データベース・リカバリが必要な場合に、データが一貫した状態に戻るのには、ロールバック・セグメントを使用して、すべてのコミットされていないデータがデータ・ファイルから削除された後です。

制御ファイル

一般に、データベースの制御ファイルには、データベースの物理構造の状態が格納されます。Oracle は、現行のオンライン REDO ログ・ファイル、データ・ファイルの名前など、制御ファイルの状態情報に従って、インスタンス・リカバリまたはメディア・リカバリを実行します。

関連項目： 29-22 ページの「制御ファイル」

ロールフォワードとロールバック

SGA のバッファ・キャッシュ内のデータベース・バッファは、LRU アルゴリズムを使用して、必要な場合にのみディスクに書き込まれます。DBW_n プロセスはこのアルゴリズムを使用してデータベース・バッファをデータ・ファイルに書き込むため、データ・ファイルには、まだコミットされていないトランザクションによって変更されたデータ・ブロックが含まれていたり、コミットされたトランザクションによる変更内容を失ったデータ・ブロックが含まれている可能性があります。

インスタンス障害が発生すると、次の 2 つの問題が起きる可能性があります。

- トランザクションによって変更されたデータ・ブロックがコミット時にデータ・ファイルに書き込まれず、REDO ログ・ファイルにのみ記録されています。このため、REDO ログには、リカバリ時にデータベースに再適用する必要がある変更が含まれています。
- ロールフォワード・フェーズの完了後に、データ・ファイルには障害発生時にコミットされていなかった変更が含まれている場合があります。このようにコミットされていない変更は、トランザクションの一貫性を保つためにロールバックする必要があります。このような変更は、障害発生前にデータベースに保存されていたものの、ロールフォワード・フェーズで取り込まれたものです。

この矛盾を解決するには、通常、2つの別個のステップを実行してリカバリを実行します。つまり、REDO ログを使用してロールフォワードし（キャッシュ・リカバリ）、ロールバック・セグメントを使用してロールバックします（トランザクション・リカバリ）。

REDO ログとロールフォワード

REDO ログとは、データ、索引およびロールバック・セグメントなど、データベース・バッファに加えられたすべての変更を、コミット済かどうかに関係なく記録するオペレーティング・システム・ファイルの集合です。各 REDO ログ・エントリは、データベースに対する1つ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 ログは、メモリー内のデータベース・バッファに対する変更内容で、データ・ファイルにまだ書き込まれていない情報を保護します。

インスタンスまたはディスク障害からリカバリする際の最初のステップは、「ロールフォワード」です。つまり、REDO ログに記録されているすべての変更をデータ・ファイルに再適用します。ロールバック・データは REDO ログにも記録されるため、ロールフォワードを実行すると、対応するロールバック・セグメントも再生成されます。これを「キャッシュ・リカバリ」と呼びます。

ロールフォワードでは、必要な時点までデータベースの状態を戻すのに必要な数の REDO ログ・ファイルが処理されます。ロールフォワードでは通常、REDO ログ・ファイルが使用され、さらにアーカイブ REDO ログ・ファイルが使用されることもあります。

ロールフォワード後のデータ・ブロックには、コミットされたデータがすべて含まれています。また、障害発生前にデータ・ファイルに保存された変更や、REDO ログに記録され、ロールフォワード中に取り込まれた変更も含まれている場合があります。

ロールバック・セグメントとロールバック

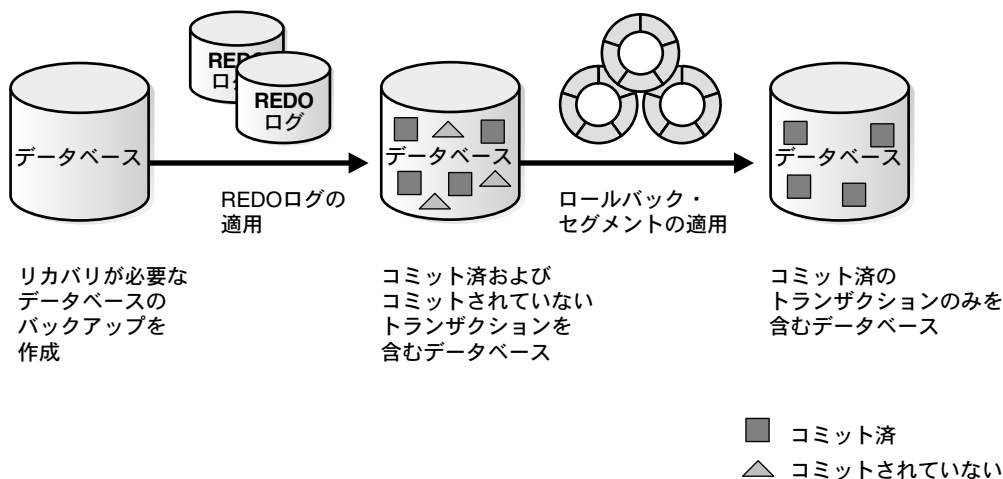
ロールバック・セグメントは、特定のデータベース操作時に取り消す必要があるデータベース・アクションを記録します。ロールバック・セグメントは、データベースのリカバリ時にロールフォワード・フェーズでそれ以前に適用されたコミットされていないトランザクションの効果を取り消します。

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・ファイルをデータベースに再適用してすべての変更を反映させた後、対応するロールバック・セグメントを使用することになります。ロールバック・セグメントは、コ

ミットされていなかったのに障害発生前にデータ・ファイルに保存されたか、ロールフォワード時にデータベースに適用されたトランザクションを識別して取り消すために使用します。このプロセスを「ロールバック」または「トランザクション・リカバリ」と呼びます。

図 29-1 に、あらゆるタイプのシステム障害からのリカバリに必要な 2 つのステップ、つまりロールフォワードとロールバックを示します。

図 29-1 基本的なリカバリ手順：ロールフォワードとロールバック



Oracle では、必要に応じて、複数のトランザクションを同時にロールバックできます。障害発生時にアクティブであったシステム全体のすべてのトランザクションは、DEAD とマークされます。SMON がデッド・トランザクションをロールバックするのを待つのではなく、ブロックしているトランザクションそのものを新しいトランザクションによってリカバリし、トランザクションに必要な行ロックを取得できます。

リカバリ・パフォーマンスの改善

ほとんどの状況では、データベース障害の発生時には迅速にリカバリすることがきわめて重要です。Oracle は、できるだけ短時間でリカバリするために、次のように様々な方法を提供します。

- パラレル・リカバリ
- ファースト・スタート・リカバリ
- 透過的アプリケーション・フェイルオーバー

リカバリのパラレル実行

リカバリでは複数の同時実行プロセスが生成した変更を再適用するため、インスタンス・リカバリまたはメディア・リカバリでは、最初にデータベースを変更したときよりも長い時間がかかることがあります。シリアル・リカバリの場合は、1つのプロセスが REDO ログ・ファイルにある変更を順次適用していきます。パラレル・リカバリを使用すると、複数のプロセスが REDO ログ・ファイルにある変更を同時に適用します。

注意： Oracle8i では、Recovery Manager でのパラレル化が制限されます。Oracle8i Enterprise Edition を使用すると、パラレル化が無制限になります。

パラレル・リカバリを実行するには、次の方法があります。

- パラレル・リカバリは、複数の Oracle Enterprise Manager セッションを生成し、各セッションで別々のデータ・ファイル・セットに対し RECOVER DATAFILE コマンドを発行して手動で実行できます。ただし、この方法を使用すると、各 Oracle Enterprise Manager セッションが REDO ログ・ファイル全体を読み込むことになります。
- Recovery Manager の **RESTORE** および **RECOVER** 文を使用すると、リカバリ処理のすべての段階を自動的にパラレル化できます。Oracle は、1つのプロセスを使用してログ・ファイルを順次読み込み、REDO 情報を複数のリカバリ・プロセスに送ります。リカバリ・プロセスは、ログ・ファイルから取り出した変更内容をデータ・ファイルに適用します。これらのリカバリ・プロセスは、Oracle によって自動的に起動されるため、リカバリを実行するために複数のセッションを使用する必要はありません。また、自動パラレル・リカバリ用に設定する初期化パラメータもあります。
- SQL*Plus の RECOVER 文を使用してパラレル・リカバリを実行できます。詳細は、『Oracle8i 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。
- SQL 文 ALTER DATABASE RECOVER を使用して、パラレル・リカバリを実行できます。ただし、この方法はお勧めしません。

関連項目： 自動パラレル・リカバリ用に初期化パラメータを設定する方法の詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』を参照してください。

パラレル・リカバリを活用できる状況

一般に、パラレル・リカバリは、複数の異なるディスク上にあるいくつかのデータ・ファイルを同時実行でリカバリする際のリカバリ時間を短縮するうえで最も効果的です。多数の異なるディスク・ドライブにある大量のデータ・ファイルのクラッシュ・リカバリ（インスタンス障害が発生した後に実行するリカバリ）とメディア・リカバリを行うのは、パラレル・リカバリのよい候補です。

パラレル・リカバリによってパフォーマンスが向上するかどうかは、オペレーティング・システムで非同期 I/O がサポートされているかどうかにも依存します。非同期 I/O がサポートされていない場合にパラレル・リカバリを使用すると、リカバリ時間が大幅に短縮されます。非同期 I/O がサポートされている場合には、パラレル・リカバリを使用してもリカバリ時間はわずかに短縮されるだけです。

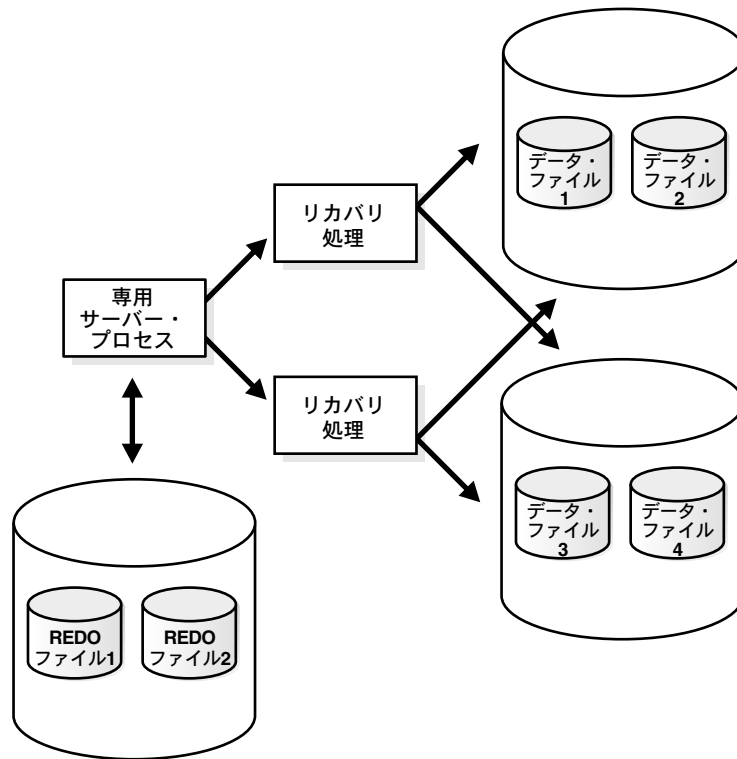
関連項目： システムで非同期 I/O がサポートされているかどうかを判別するには、オペレーティング・システムのマニュアルを参照してください。

リカバリ・プロセス

典型的なパラレル・リカバリの状況では、1つのプロセスが REDO ログ・ファイルから REDO エントリを読み込み、それらのエントリを分配します。これは、リカバリ・セッションを開始する専用サーバー・プロセスです。REDO ログ・ファイルを読み込むこのサーバー・プロセスは、2つ以上のリカバリ・プロセスの支援を受けて REDO エントリにある変更をデータ・ファイルに適用します。

図 29-2 に、典型的なパラレル・リカバリ・セッションを示します。

図 29-2 典型的なパラレル・リカバリ・セッション



ほとんどの状況では、リカバリが必要なデータ・ファイルが入っているディスク・ドライブ 1 つに対して、1 つのリカバリ・セッションと 1 つか 2 つのリカバリ・プロセスがあれば十分です。リカバリは、CPU 集中型のアクティビティとは対照的な、ディスク集中型のアクティビティであるため、必要なリカバリ・プロセスの数は、リカバリで使用するディスク・ドライブの数によって異なります。一般に、パラレル・リカバリのパフォーマンスをシリアル・リカバリよりも高くするには、8 個のリカバリ・プロセスが必要です。

ファースト・スタート・リカバリ

ファースト・スタート・リカバリは、ロールフォワードの所要時間を短縮し、リカバリを境界付きで予測可能なものにするアーキテクチャです。また、システム障害のために異常終了したトランザクションの場合は、リカバリ時のロールバック時間が不要になります。ファースト・スタート・リカバリの内容は、次のとおりです。

- ファースト・スタート・チェックポイント
- ファースト・スタート・オン・デマンド・ロールバック
- ファースト・スタート・パラレル・ロールバック

ファースト・スタート・チェックポイント

ファースト・スタート・チェックポイントは、REDO スレッド（ログ）内でクラッシュ・リカバリまたはインスタンス・リカバリを開始する必要がある位置を記録します。この位置は、バッファ・キャッシュ内の最も古い使用済バッファによって決まります。各 DBW n プロセスは、絶えずバッファをディスクに書き込んでチェックポイント位置を先に進めます。このため、通常の処理中には、オーバーヘッドは生じないか、最小限度ですみます。ファースト・スタート・チェックポイント機能により、クラッシュ・リカバリとインスタンス・リカバリのパフォーマンスは向上しますが、メディア・リカバリのパフォーマンスは向上しません。

クラッシュ・リカバリやインスタンス・リカバリの所要時間が厳しく制限される状況では、リカバリ処理のパフォーマンスを改善できます。クラッシュ・リカバリやインスタンス・リカバリの所要時間は、ロールフォワード・フェーズで読み込みまたは書き込みを必要とするデータ・ブロック数にほぼ比例します。ロールフォワード中に処理を必要とするデータ・ブロック数について、制限つまり上限を指定できます。Oracle サーバーは、指定されたロールフォワード上限にあわせてチェックポイント書き込み率を自動的に調整し、書き込み数を最小限度に抑えます。

動的初期化パラメータ FAST_START_IO_TARGET を設定すると、クラッシュ・リカバリまたはインスタンス・リカバリ時に読み込みを必要とするブロック数を制限できます。このパラメータの値が小さいほど、多数のバッファが書き込まれる必要があるため、通常の処理時のオーバーヘッドが大きくなります。一方、このパラメータの値が小さいほど、リカバリする必要があるブロック数が少なくなるため、リカバリのパフォーマンスは向上します。また、動的初期化パラメータ LOG_CHECKPOINT_INTERVAL および LOG_CHECKPOINT_TIMEOUT も、ファースト・スタート・チェックポイントに影響します。

関連項目： FAST_START_IO_TARGET 値の設定方法の詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

ファースト・スタート・オン・デマンド・ロールバック

デッド・トランザクションが他のトランザクションに必要な行ロックを保持している場合、「ファースト・スタート・オン・デマンド・ロールバック」は、問題のデータ・ブロックのみを即時にリカバリし、デッド・トランザクションの残りの部分についてはバックグラウンドでリカバリ処理が実行されます。これにより、大きいデッド・トランザクションによってロックされているデータにアクセスするユーザーにとっては、データベースの可用性が改善されます。ファースト・スタート・オン・デマンド・ロールバックが使用可能になっていない場合、ユーザーはデッド・トランザクション全体がリカバリされるまで待たなければ、行ロックを取得できません。

ファースト・スタート・パラレル・ロールバック

ファースト・スタート・パラレル・ロールバックにより、サーバー・プロセスのグループを使用してトランザクションの集合をパラレルにリカバリできます。この手法が使用されるのは、パラレル・リカバリの所要時間が、シリアルなリカバリの所要時間より短いと SMON が判断した場合です。

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

迅速なリカバリ処理では、データの使用不能期間は最短になりますが、ユーザー・セッションの障害による中断は処理されません。ユーザーは、データベースへの接続を再確立する必要があり、進行中だった作業は失われる可能性があります。Oracle8i の透過的アプリケーション・フェイルオーバー（TAF）により、アプリケーションの状態を保ち、障害発生時に進行中だった問合せを再開して、多数の障害をユーザーから隠すことができます。開発者は、TAF を活用するアプリケーションを作成し、トランザクションに影響するものを含めて全障害をユーザーに透過的なものにして、これらの機能を拡張できます。

関連項目： 透過的アプリケーション・フェイルオーバーの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。

Recovery Manager

Recovery Manager は、すべてのデータベース・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 REDO ログ・ファイル）のバックアップを作成するプロセスと、バックアップからファイルをリストアまたはリカバリするプロセスを管理する Oracle ユーティリティです。

関連項目： Recovery Manager の詳細は、『Oracle8i Recovery Manager 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。

リカバリ・カタログ

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、リストア操作とメディア・リカバリ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 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

リカバリ・カタログをメンテナンスしているのは、Recovery Manager のみです。バックアップ済データベースのデータベース・サーバーが、リカバリ・カタログに直接アクセスすることはありません。Recovery Manager は、バックアップ・データ・ファイルの集合、アーカイブ REDO ログ、バックアップ制御ファイルおよびデータ・ファイルのコピーに関する情報を、長期保存のためにリカバリ・カタログに伝播します。

リカバリの実行時に、Recovery Manager はリカバリ・カタログから適切な情報を抽出して、データベース・サーバーに渡します。サーバーは、リカバ리를指定された入力ファイルに対して様々な整合性チェックを実行します。Recovery Manager の動作が不正確でも、データベースが破壊されることはありません。

リカバリ・カタログ・データベース

リカバリ・カタログは、Oracle データベースに格納されます。Recovery Manager を格納するためのデータベースは、データベース管理者の責任で準備します。リカバリ・カタログのバックアップも、データベース管理者の責任で作成します。リカバリ・カタログは Oracle データベースに格納されるため、Recovery Manager を使用してリカバリ・カタログのバックアップを作成できます。

リカバリ・カタログが破棄されて使用可能なバックアップがない場合は、カレントの制御ファイルまたは制御ファイルのバックアップから部分的にリカバリ・カタログを再構築できます。

リカバリ・カタログを使用しない操作

リカバリ・カタログの使用は必須ではありませんが、使用することをお勧めします。リカバリ・カタログのほとんどの情報は制御ファイルからも入手できるため、Recovery Manager では制御ファイルのみを使用する操作モードをサポートしています。この操作モードは、リカバリ・データベースとして別のデータベースをインストールして管理することが煩わしく思える、小規模なデータベースで有効なモードです。

Recovery Manager には、リカバリ・カタログを使用する場合にのみ使用可能な機能があります。

関連項目： リカバリ・カタログの作成方法と、リカバリ・カタログの使用を必要とする Recovery Manager の機能の詳細は、『Oracle8i Recovery Manager 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。

パラレル化

Recovery Manager では、非ブロック化 UPI を使用して複数のログオン・セッションを確立し、複数の操作を同時に実行することにより、操作をパラレル化できます。同時実行される操作は、データ・ファイルの別々のセットを処理するものである必要があります。

注意： Oracle8i Enterprise Edition を使用すると、パラレル化が無制限になります。Oracle8i では、Recovery Manager チャネルを一度に 1 つしか割当てできないため、1 つのストリームに対するパラレル化は制限されます。

backup、**copy** および **restore** の各コマンドのパラレル化は、Recovery Manager で内部処理されます。必要なのは、次の情報を指定することのみです。

- ディスクまたはテープ・ドライブなど、1 つ以上のシーケンシャル I/O デバイスのリスト
- バックアップ、コピーまたはリストアするオブジェクト

Recovery Manager はコマンドを順次実行します。つまり、前のコマンドを完了してから次のコマンドを開始します。パラレル化が活用されるのは、1 つのコマンドのコンテキスト内に限られます。したがって、10 個のデータ・ファイル・コピーが必要な場合は、**copy** コマンドを 10 回発行するのではなく、10 回のコピーをすべて指定した **copy** コマンドを 1 回発行してください。

レポートの生成

report および **list** コマンドを使用すると、バックアップとイメージ・コピーに関する情報が得られます。これらのコマンドの出力は、メッセージ・ログ・ファイルに書き込まれます。

report コマンドによって生成されるレポートは、次のような質問に対する答えを提供します。

- バックアップが必要なファイル
- しばらくバックアップを作成していなかったファイル
- 削除できるバックアップ・ファイル

report need backup コマンドと **report unrecoverable** コマンドを定期的に使用することによって、リカバリの実行に必要なバックアップを使用可能にし、リカバリが妥当な時間内で実行されるよう指定できます。

データ・ファイルに格納されているスキーマ・オブジェクトに対して、ログに記録されていない操作が実行された場合、そのデータ・ファイルは「リカバリ不能」と見なされます。

(バックアップがないデータ・ファイルは、リカバリ不能とは見なされません。そのファイルを作成した時点から記録が開始されたログがまだ残っていれば、CREATE DATAFILE 文を使用して、そのようなデータ・ファイルをリカバリできます。)

list コマンドは、リカバリ・カタログを問い合せて、その内容を示すリストを生成します。このコマンドを使用すると、どのバックアップまたはコピーが使用可能であるかがわかります。

- 指定したデータ・ファイル（複数可）のバックアップまたはコピー
- 指定した表領域（複数可）のメンバーであるデータ・ファイルのバックアップまたはコピー
- 指定した名前または指定した範囲内（あるいはその両方）のアーカイブ・ログのバックアップまたはコピー
- 指定したデータベースそのもの

データベースのアーカイブ・モード

データベースは、NOARCHIVELOG モード（メディア・リカバリが使用禁止）と ARCHIVELOG モード（メディア・リカバリが使用可能）の2つのモードで操作できます。

NOARCHIVELOG モード（メディア・リカバリが使用禁止）

データベースを NOARCHIVELOG モードで使用する場合、オンライン REDO ログのアーカイブは使用禁止になります。データベースの制御ファイルの情報は、いっばいのグループをアーカイブする必要はないことを示します。したがって、いっばいのグループが非アクティブになると、そのグループは即時に LGWR プロセスが再使用できるようになります。

NOARCHIVELOG モードでデータベースを保護できるのは、インスタンス障害が発生した場合のみで、ディスク障害の場合は保護できません。クラッシュ・リカバリまたはインスタンス・リカバリに使用できるのは、データベースへの最新の変更（オンライン REDO ログのグループに格納されている）のみで、それ以外の情報は使用できません。Oracle は、変更がデータ・ファイルに安全に記録されるまで、必要かもしれないオンライン REDO ログ・ファイルを上書きしないため、クラッシュ・リカバリとインスタンス・リカバリのニーズを満たすにはこの情報があれば十分です。ただし、メディア・リカバリは実行できません。

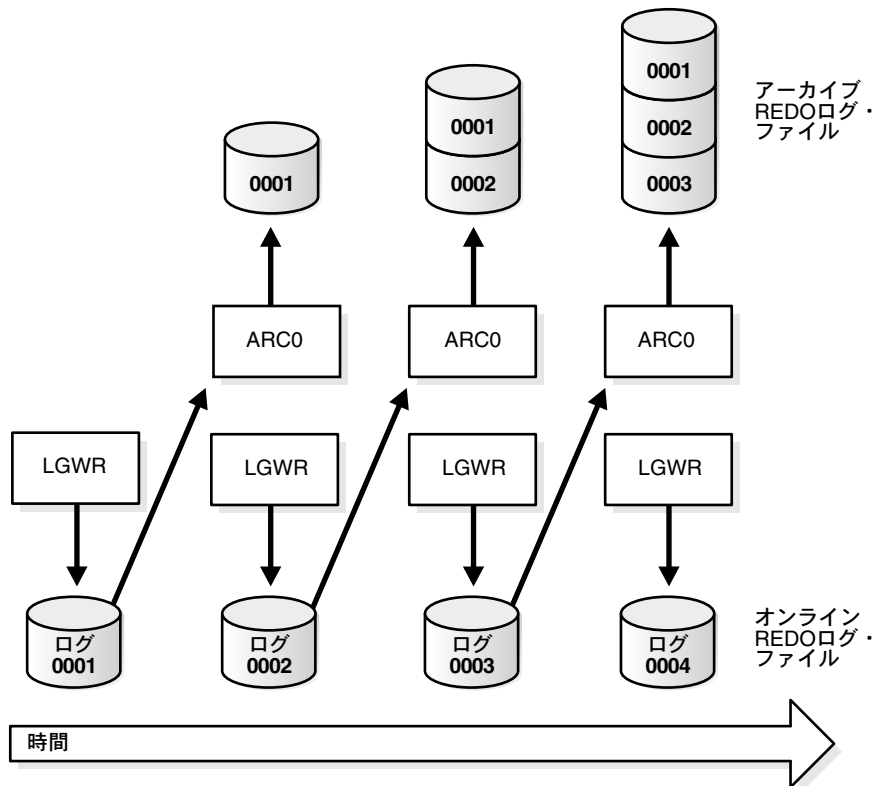
ARCHIVELOG モード（メディア・リカバリが使用可能）

データベースを ARCHIVELOG モードで使用する場合、オンライン REDO ログのアーカイブは使用可能になります。データベースの制御ファイルの情報は、いっぱいのオンライン REDO ログ・ファイルのグループのアーカイブが完了するまで、LGWR はそのグループを再使用できないことを示します。

図 29-3 に、データベースのオンライン REDO ログ・ファイルが ARCHIVELOG モードでどのように使用されるか、また、いっぱいของกลุ่มをアーカイブするプロセス（この図では ARC0）によってアーカイブ REDO ログがどのように生成されるかを示します。

ARCHIVELOG モードでは、データベースに対する変更がすべてアーカイブ REDO ログに永続的に保存されるため、ディスク障害とインスタンス障害から完全にリカバリできます。

図 29-3 ARCHIVELOG モードでのオンライン REDO ログ・ファイルの使用



自動アーカイブと ARCn（アーカイバ）バックグラウンド・プロセス

インスタンスの構成にバックグラウンド・プロセス ARC0（アーカイバ）を追加すると、オンライン REDO ログ・ファイルのグループがアクティブでなくなった場合に、そのプロセスが自動的にこれらのグループをアーカイブできます。自動アーカイブを使用すると、データベース管理者は、いっばいのグループを追跡して記録し、アーカイブする作業を手動で実行する必要がなくなります。ARCHIVELOG モードで稼働するほとんどのデータベース・システムで自動アーカイブを使用しているのは、このような便宜上の理由のためです。データのバルクロードなど、作業負荷が大きい場合は、初期化パラメータ LOG_ARCHIVE_MAX_PROCESSES を設定して、複数のアーカイバ・プロセス（ARC9 まで）を構成できます。

LOG_ARCHIVE_START 初期化パラメータを設定して、インスタンス起動時に自動アーカイブを要求すると、Oracle は LOG_ARCHIVE_MAX_PROCESSES で指定された数の ARCn プロセスをインスタンス起動時に開始します。それ以外の場合、インスタンス起動時には、ARCn プロセスは開始されません。

ただし、データベース管理者は、自動アーカイブをいつでも対話形式で開始したり停止できます。インスタンス起動時に自動アーカイブを指定しないで、後からデータベース管理者が自動アーカイブを開始すると、ARCn バックグラウンド・プロセスが作成されます。ARCn は、インスタンスの存続期間中は、自動アーカイブが一時的にオフになり、再度オンになっても残っています。ただし、ALTER SYSTEM 文で LOG_ARCHIVE_MAX_PROCESSES を設定すると、ARCn プロセスの数を動的に変更できます。

ARCn は、常に最小の順序番号から順にグループをアーカイブします。また、ARCn は、いったいのグループがアクティブでなくなると、そのグループを自動的にアーカイブします。すべての自動アーカイブの記録が、ARCn プロセスによって ARCn トレース・ファイルに書き込まれます。それぞれのエントリには、アーカイブの開始時刻と終了時刻が記録されます。

ARCn があるグループをアーカイブするときにエラーを検出しても（アーカイブ先が無効またはいったいの場合など）、ARCn は引き続きそのグループをアーカイブします。また、エラーは ARCn トレース・ファイルと ALERT ファイルにも書き込まれます。問題が解決されない場合、最終的にはすべてのオンライン REDO ログ・グループがいったいになってもアーカイブされていないと、LGWR が使用できるグループがなくなるため、システムは停止します。したがって、問題が検出された場合は、問題を解決して ARCn がアーカイブを続行できるようにするか（アーカイブ先の変更など）、または問題が解決されるまで手動でグループをアーカイブしてください。

手動アーカイブ

データベースが ARCHIVELOG モードで動作している場合、自動アーカイブが使用可能かどうかには関係なく、必要に応じてアクティブでないオンライン REDO ログ・ファイルのグループのうちいったいになったものは、手動でアーカイブできます。自動アーカイブが使用禁止の場合、データベース管理者は、すべてのいったいのグループを手動でアーカイブする必要があります。

ほとんどのシステムでは、グループが非アクティブになってアーカイブ可能になったかどうかをデータベース管理者が監視する必要はないようにするため、自動アーカイブを使用します。さらに、自動アーカイブを使用禁止にした場合に、手動アーカイブを十分な速度で実行しなければ、アクティブでないグループを再使用できるようになるまで LGWR が強制的に待機状態に入り、データベース操作が一時的に停止することがあります。

手動アーカイブ・オプションは、データベース管理者が次の操作を実行するために提供されています。

- 問題が発生して（アーカイブ REDO ログのアーカイブ先として指定されているオフラインの記憶デバイスで障害が発生した、またはいっぱいになったなど）、自動アーカイブが停止したときにグループをアーカイブします。
- 標準以外の方法でグループをアーカイブします（たとえば、あるグループを1つのオフライン記憶デバイスにアーカイブし、次のグループを別のオフライン記憶デバイスにアーカイブするなど）。
- オリジナルのアーカイブ済バージョンが消失したり、破損した場合にグループを再アーカイブします。

グループを手動でアーカイブする場合、グループをアーカイブする文を発行したユーザー・プロセスが、そのグループの実際のアーカイブ処理を実行します。そのインスタンスの ARCn バックグラウンド・プロセスが存在していても、オンライン REDO ログ・ファイル・グループのアーカイブは、ユーザー・プロセスが実行します。

制御ファイル

データベースの制御ファイルは、データベースの正常な起動と操作に必要な小さいバイナリ・ファイルです。制御ファイルはデータベースの使用時に Oracle によって絶えず更新されるため、データベースがオープンされている間は書込み可能になっている必要があります。なんらかの理由で制御ファイルにアクセスできない場合、データベースは正常に機能しなくなります。

それぞれの制御ファイルは、1 つの Oracle データベースにのみ対応付けられます。

制御ファイルの内容

制御ファイルには、インスタンスからアクセスするデータベースが起動時と通常の操作時に必要とする、関連データベースの情報が格納されています。制御ファイルの情報を変更できるのは、Oracle のみです。データベース管理者やエンド・ユーザーは、データベースの制御ファイルを編集できません。

制御ファイルには、次のような情報が格納されています。

- データベース名
- データベースを作成したときのタイムスタンプ
- 対応するデータ・ファイルとオンライン REDO ログ・ファイルの名前と位置
- 表領域情報
- データ・ファイルのオフライン範囲
- ログ履歴
- アーカイブ・ログ情報

- バックアップ・セットとバックアップ部分の情報
- バックアップ・データ・ファイルと REDO ログ情報
- データ・ファイル・コピー情報
- カレントのログ順序番号
- チェックポイント情報

データベース名とタイムスタンプは、データベース作成時に作成されます。データベース名には、DB_NAME 初期化パラメータに指定した名前、または CREATE DATABASE 文で使
用した名前が使用されます。

データベースのデータ・ファイルやオンライン REDO ログ・ファイルが追加、改名または
削除されるたびに、制御ファイルが更新され、この物理構造の変化が反映されます。これら
の変更は、次の目的のために記録されます。

- データベースの起動時に、オープンするデータ・ファイルとオンライン REDO ログ・
ファイルを Oracle が識別できるようにします。
- データベースのリカバリが必要な場合に必要なファイルや使用可能なファイルを Oracle
が識別できるようにします。

したがって、データベースの物理構造を変更した後は、ただちに制御ファイルのバックアッ
プを作成してください。

制御ファイルには、チェックポイントに関する情報も記録されます。チェックポイント・プ
ロセス（CKPT）は、オンライン REDO ログ内のチェックポイント位置に関する情報を、制
御ファイルに 3 秒ごとに記録します。この情報は、データベースのリカバリ時に使用され、
オンライン REDO ログ・グループの特定のポイントより前に記録されている REDO エント
リはデータベースのリカバリには不要である（すでにデータ・ファイルに書き込まれてい
る）ことを Oracle に伝えます。

関連項目：

データベースの制御ファイルのバックアップを作成する方法は、次のマ
ニュアルを参照してください。

- 『Oracle8i Recovery Manager ユーザーズ・ガイドおよびリファレン
ス』
- 『Oracle8i バックアップおよびリカバリ・ガイド』

多重制御ファイル

オンライン REDO ログ・ファイルと同様に、同一の制御ファイルを同時に複数オープンし
て、同じデータベースの情報を書き込みます。

1つのデータベースについての複数の制御ファイルを異なるディスクに格納することによって、単一地点の障害から制御ファイルを保護できます。制御ファイルが格納されているディスクがクラッシュした場合に、Oracleがこの破損したファイルにアクセスしようとすると、カレント・インスタンスがエラーになります。ただし、制御ファイルのコピーを他のディスクに保存してあれば、データベースをリカバリしなくてもインスタンスを簡単に再起動できます。

データベースの制御ファイルのすべてのコピーが永続的に失われると、重大な問題になるため、その事態を回避するための保護対策が必要です。操作時にデータベースのすべての制御ファイルが永続的に消失した（複数のディスクで障害が発生した）場合は、インスタンスが異常終了し、メディア・リカバリが必要になります。ただし、現行の制御ファイルを使用できず、制御ファイルの古いバックアップを使用するのであれば、メディア・リカバリは容易ではありません。したがって、次の原則を遵守することをお勧めします。

- 各データベースで多重制御ファイルを使用する
- 各コピーを異なる物理ディスクに格納する
- オペレーティング・システムのミラー化機能を使用する

データベースのバックアップ

Oracle ミラー化ログ、Oracle ミラー化制御ファイルおよびアーカイブ・ログを使用すると、メディア障害からリカバリできますが、リカバリ処理中は一部またはすべてのデータが使用不能になることがあります。リカバリ・レベルを上げるために、少なくともデータ・ファイルと制御ファイルには、オペレーティング・システムまたはハードウェアのデータ冗長性を使用することをお勧めします。これにより、システムは完全に使用可能な状態で、メディア障害のいずれか1つがリカバリ可能になることが保証されます。

Oracle データベースについてどのようなバックアップおよびリカバリの計画を考案するとしても、データベースのデータ・ファイルと制御ファイルのバックアップを作成しておくことは、これらのファイルを破損しかねないメディア障害の可能性に備えた保護方針の一環として絶対に必要です。以降の項では、使用可能な様々なタイプのバックアップの概念と、異なるリカバリ方式におけるバックアップの有効性について説明します。

関連項目： データベース・バックアップを実行するためのガイドラインは、『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。

データベース全体のバックアップ

「データベース全体のバックアップ」は、Oracle データベースを構成する、すべてのデータ・ファイルと制御ファイルのバックアップです。このバックアップを実行するには、Recovery Manager を使用方法と、オペレーティング・システム・コマンドを使用する方法があります。データベース全体のバックアップは、データベースが停止している場合、またはデータベースがオープンしている間に実行できます。通常、インスタンス障害やその他の異常な状況が発生した後は、全体バックアップを実行しないでください。

一貫性のある全体バックアップと一貫性のない全体バックアップ

「正しく停止」するとは、クラッシュや SHUTDOWN ABORT 以外の状況で停止することです。データベースが正しく停止すると、データベースを構成するすべてのファイルがクローズされ、現時点での一貫性がとられます。したがって、停止後に実行した全体バックアップを使用すると、そのバックアップの時点までリカバリできます。データベースがオープンしている間に全体バックアップを実行すると、特定の時点での一貫性がないため、オンラインおよびアーカイブ REDO ログ・ファイルを使用してバックアップをリカバリしてからでなければ、データベースは使用可能になりません。

バックアップとアーカイブ・モード

全体バックアップによって取得されたデータ・ファイルは、どのタイプのメディア・リカバリの方式にも有効です。

- データベースが NOARCHIVELOG モードで動作している場合に、ディスク障害が発生してデータベースを構成するファイルの一部またはすべてが破損した場合は、一貫性のある最新の全体バックアップを使用してデータベースを「リストア」できます（リカバリではありません）。

データベースを現時点の状態に戻すためのアーカイブ REDO ログを使用できないため、データベース全体のバックアップ以後に実行したすべてのデータベース作業をやりなおす必要があります。特定の状況下では、NOARCHIVELOG モードでディスク障害が発生しても完全にリカバリできますが、この方法には依存しないでください。

- データベースが ARCHIVELOG モードで動作している場合に、ディスク障害が発生してデータベースを構成するファイルの一部またはすべてが破損した場合は、最新の全体バックアップで収集したデータ・ファイルをデータベース・リカバリの一部として使用できます。

必要なデータ・ファイルを全体バックアップからリストアした後は、アーカイブ済のオンライン REDO ログ・ファイルとカレントのオンライン REDO ログ・ファイルを適用して、リストアしたデータ・ファイルを現時点の状態に戻し、データベース・リカバリ作業を続行できます。

まとめると、データベースが NOARCHIVELOG モードで動作している場合、ディスク障害からデータベースを部分的にでも保護するための方法は、一貫性のあるデータベース全体のバックアップしかありません。データベースが ARCHIVELOG モードで動作しており、ログが使用可能な場合、一貫性のあるデータベース全体のバックアップが一貫性のないデータベース全体のバックアップのいずれかを使用し、ディスク障害からのデータベース・リカバリ作業の一部として、破損したファイルをリストアできます。

部分データベース・バックアップ

「部分データベース・バックアップ」とは、全体バックアップより小規模なバックアップのことで、データベースがオープンまたは停止しているときに実行します。部分データベース・バックアップには、次のものがあります。

- 個々の表領域のすべてのデータ・ファイルのバックアップ
- 1つのデータ・ファイルのバックアップ
- 制御ファイルのバックアップ

部分バックアップが有効なのは、ARCHIVELOG モードでデータベースを実行している場合のみです。アーカイブ REDO ログがあるため、部分バックアップからリストアされたデータ・ファイルは、リカバリ手順の中でデータベースの残りの部分との一貫性を確保できます。

データ・ファイルのバックアップ

部分バックアップには、データベースの一部のデータ・ファイルのみを含めることができます。個々の特定のデータ・ファイルまたはその集合のバックアップは、データベースの他のデータ・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バックアップとは別個に作成できます。データ・ファイルのバックアップは、オフラインでもオンラインでも作成できます。

オンラインとオフラインのどちらのデータ・ファイル・バックアップを作成するかは、データ可用性の要件のみに依存しています。バックアップの対象となるデータを常に使用可能な状態にしておく必要がある場合は、オンラインのデータ・ファイル・バックアップが唯一の選択肢です。

制御ファイルのバックアップ

部分バックアップの別の形態は、制御ファイルのバックアップです。制御ファイルは、対応付けられたデータベースの物理ファイル構造を追跡し記録するため、データベースの制御ファイルのバックアップは、データベースに対して構造上の変更がなされるたびに作成する必要があります。物理的なバックアップとトレース・ファイルへのバックアップを作成してください。

注意： Recovery Manager は、データ・ファイル 1 を含むバックアップでは自動的に制御ファイルのバックアップを作成します。データ・ファイル 1 にはデータ・ディクショナリが含まれています。

制御ファイルを多重化しておくと、単一の制御ファイルが失われた場合の保護対策になります。ただし、ディスク障害によってデータ・ファイルが破損し、部分的なりカバリまたは特定の時点までのリカバリが必要な場合は、必ずしも現在の制御ファイルではなく、目的とするデータベース構造に対応する制御ファイルのバックアップを使用する必要があります。こ

のため、制御ファイルを多重化してあるとしても、データベースの構造が変更されるたびに作成する制御ファイル・バックアップの代替策にはなりません。

不完全なりカバリまたは Point-in-Time リカバリの前に Recovery Manager を使用して制御ファイルをリカバリする場合、Recovery Manager は最適なバックアップ制御ファイルを自動的にリストアします。

Export および Import ユーティリティ

Export および Import は、Oracle のデータを Oracle データベースから外部へ、または外部から Oracle データベース内部へ移動するために使用するユーティリティです。Export は、Oracle データベースのデータをオペレーティング・システムのファイルに、Oracle データベース・フォーマットで書き出すユーティリティです。エクスポートされるファイルには、データベースに対して作成されたスキーマ・オブジェクトに関する情報が格納されます。Import は、エクスポートされたファイルを読み込んで、対応する情報を既存のデータベースに復旧するユーティリティです。Export と Import は、Oracle データの移動を目的として設計されていますが、Oracle データベースのデータを保護する補助的な方法としても利用できます。

関連項目：『Oracle8i ユーティリティ・ガイド』

読取り専用表領域とバックアップ

データベースがオープンしているときでも、読取り専用表領域についてはバックアップを作成できます。表領域を読取り専用にした後は、その表領域のバックアップをただちに作成してください。表領域が読取り専用のままになっている限り、それ以後、この表領域のバックアップを作成する必要はありません。

読取り専用表領域を読み書き可能表領域に変更した場合、その表領域の通常のバックアップを再開する必要があります。これはオフラインの読み書き可能表領域をオンラインに戻した時点でバックアップを再開するのと同様です。

読取り専用表領域のデータ・ファイルをオンライン化しても、これらのファイルは書込み可能にはならず、ファイル・ヘッダーも更新されません。このため、書込み可能データ・ファイルをオンラインに戻したときとは違って、これらのファイルのバックアップを実行する必要はありません。

耐障害性

電源障害、ハードウェア障害またはシステムを中断させるその他の障害に備えて、Oracle には「管理されたスタンバイ・データベース」機能が用意されています。スタンバイ・データベースは、耐障害性と災害リカバリが特に重大な意味を持つサイトを対象にした機能です。もう 1 つの選択肢は、データベース・レプリケーションを使用することです。

関連項目：

- [第 31 章「レプリケーション」](#)
- 『Oracle8i スタンバイ・データベース 概要および管理』

障害リカバリの計画

システム障害やその他の災害から迅速かつ確実にリカバリするには、綿密な計画を練る以外に方法はありません。詳しい手順を記述した定型的な計画を作成しておく必要があります。スタンバイ・データベース・システムを実装するにしても、単一データベース・システムを使用するにしても、突然の障害に備えてどのように対処するかを計画しておく必要があります。

管理されたスタンバイ・データベース

Oracle では、迅速な障害リカバリを容易にするために、管理されたスタンバイ・データベース機能が提供されています。プライマリ・サイトにアーカイブされているログ・ファイルの自動出荷依頼を通じて、最大 4 つのスタンバイ・システムを一貫したメディア・リカバリ状態に保つことができます。1 次システムで障害が発生した場合は、スタンバイ・システムの 1 つをアクティブ化できるため、システムはすぐに使用可能になります。Oracle には、スタンバイ・システムの作成とメンテナンスに伴う操作のためのコマンドと内部確認機能が用意されており、それによって障害リカバリ・スキーマの信頼性が改善されています。

スタンバイ・データベースでは、いつでもリカバリ操作を実行してオンライン状態に入れるように、プライマリ・データベースからのアーカイブ済ログ情報が使用されます。プライマリ・データベースで REDO ログがアーカイブされると、必ずそのログがリモート・サイトに転送され、スタンバイ・データベースに適用されます。

管理されたスタンバイ・データベースによって、電源障害などによる長期停止や、火災、洪水または地震などの自然災害からデータが保護されます。スタンバイ・データベースは障害リカバリを意図して設計されているため、プライマリ・データベースとは別の物理位置に配置するのが理想的です。

スタンバイ・データベースは、読取り専用でオープンできます。これにより、データベースをレポート作成に使用できます。スタンバイ・データベースを読取り専用でオープンすると、REDO ログはキューに入れられ、適用されません。データベースがスタンバイ・モードに戻ると、キューに入れられたログと新しいアーカイブ・ログがただちに適用されます。

関連項目： スタンバイ・データベースの作成とメンテナンスの詳細は、『Oracle8i スタンバイ・データベース 概要および管理』を参照してください。

第 IX 部

分散データベースとレプリケーション

第 IX では、分散データベース・アーキテクチャとネットワーク上でのデータ・レプリケーションについて説明します。

第 IX 部には、次の章が含まれています。

- [第 30 章「分散データベースの概念」](#)
- [第 31 章「レプリケーション」](#)

分散データベースの概念

この章では、Oracle の分散データベース・アーキテクチャの基本的な概念と用語を説明します。この章の内容は、次のとおりです。

- [分散データベース・アーキテクチャの概要](#)
- [データベース・リンク](#)
- [分散データベースの管理](#)
- [分散システムでのトランザクション処理](#)
- [分散データベース・アプリケーションの開発](#)
- [各国語サポート](#)

分散データベース・アーキテクチャの概要

「分散データベース・システム」により、アプリケーションはローカル・データベースとリモート・データベースのデータにアクセスできます。「同種分散システム」では、各データベースは Oracle データベースです。「異種分散データベース・システム」では、データベースのうち最低 1 つは非 Oracle データベースです。分散データベースでは、「クライアント / サーバー」アーキテクチャを使用して情報要求が処理されます。

ここでは、次の内容を取り上げます。

- 同種分散データベース・システム
- 異種分散データベース・システム
- クライアント / サーバー・データベース・アーキテクチャ

同種分散データベース・システム

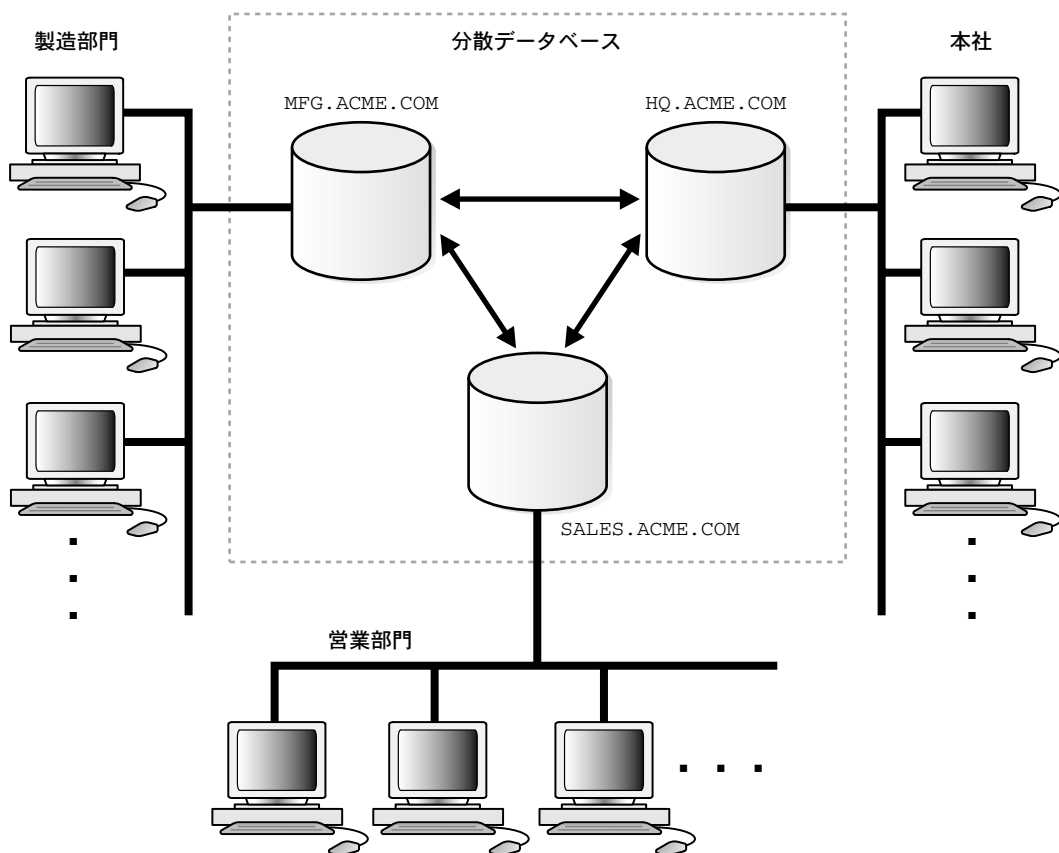
「同種分散データベース・システム」は、1 台以上のマシンに格納された複数の Oracle データベースからなるネットワークです。図 30-1 に、HQ、MFG および SALES という 3 つのデータベースを接続する分散システムを示します。アプリケーションは、単一の分散環境で複数のデータベースにあるデータに同時にアクセスしたり変更できます。たとえば、ローカル・データベース MFG に対する 1 度の問合せで、ローカル・データベースの PRODUCTS 表とリモート HQ データベースの DEPT 表からデータを結合したものを取り出すことができます。

クライアント・アプリケーションにとって、データベースの位置とプラットフォームは透過的です。また、ユーザーがローカル・オブジェクトの場合と同じ構文でアクセスできるように、分散システム内のリモート・オブジェクトに「シノニム」を作成できます。たとえば、データベース MFG に接続していて、データベース HQ のデータにアクセスする必要がある場合、リモート DEPT 表のシノニムを MFG 上で作成すると、次の問合せを発行できます。

```
SELECT * FROM dept;
```

このように、分散システムでは見かけ上その場のデータ・アクセスをしているようになります。MFG 上のユーザーは、アクセスするデータがリモート・データベースに格納されていることを知る必要がありません。

図 30-1 同種分散データベース



Oracle 分散データベース・システムでは、異なるバージョンの Oracle データベースを統合できます。サポートされているすべてのリリースの Oracle を、分散データベース・システムに加えることができます。ただし、分散データベースを処理するアプリケーションは、システムの各ノードで使用可能な機能を認識する必要があります。たとえば、分散データベース・アプリケーションでは、Oracle8i でしか使用できないオブジェクト SQL 拡張機能が Oracle7 データベースに認識されることは期待できません。

分散データベースと分散処理

「分散データベース」と「分散処理」という用語には密接な関係がありますが、意味には明確な違いがあります。

分散データベース	分散システムにおいて、アプリケーションには単一のデータ・ソースに見えるようになっているデータベースの集合。
分散処理	アプリケーションがネットワーク内の異なるコンピュータに作業を分配する場合に生じる操作。たとえば、多くのデータベース・アプリケーションでは、フロントエンドの表示作業を複数のクライアント・コンピュータに分散し、バックエンドのデータベース・サーバーがデータベースへの共有アクセスを管理するようにします。そのため、分散データベース・アプリケーション処理システムは、「クライアント / サーバー」データベース・アプリケーション・システムとも呼ばれます。

Oracle 分散データベース・システムでは、分散処理アーキテクチャを採用しています。たとえば、Oracle Server は、他の Oracle Server が管理するデータを要求するときにはクライアントとして動作します。

分散データベースとレプリケート・データベース

「分散データベース・システム」と「データベース・レプリケーション」という用語には関係がありますが、違いがあります。「純粹」な（レプリケートされていない）分散データベースでは、システムはすべてのデータおよびサポートするデータベース・オブジェクトの1つのコピーを管理します。多くの分散データベース・アプリケーションは、分散トランザクションを使用してローカル・データとリモート・データの両方にアクセスし、グローバル・データベースをリアルタイムに修正します。

注意： このマニュアルでは、純粹な分散データベースについてのみ説明します。

「レプリケーション」という用語は、分散システムに属する複数のデータベース内のデータベース・オブジェクトをコピーおよびメンテナンスする操作を指します。レプリケーションは分散データベース・テクノロジーに依存していますが、データベース・レプリケーションを使用するアプリケーションには、純粹な分散データベース環境では得られない利点があります。

最も一般的な点として、レプリケーションには代替データ・アクセスのオプションがあることによるローカル・データベースのパフォーマンス改善やアプリケーションの可用性の保持に使用されます。たとえばアプリケーションは、普段はリモート・サーバーではなくローカル・データベースにアクセスすることにより、ネットワーク通信量を最小化して最大のパフォーマンスを達成できる可能性があります。しかもそのアプリケーションは、ローカル・サーバーに障害が発生した場合にも、レプリケートされたデータが存在する他のサーバーがアクセス可能である限り継続できます。

関連項目： Oracle のレプリケーション機能の詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

異種分散データベース・システム

「異種分散データベース・システム」では、データベース・システムのうち最低 1 つは非 Oracle システムです。アプリケーションからは、異種分散データベース・システムは、1 つのローカル Oracle データベースに見えます。ローカル Oracle サーバーは、データの分散と異種性を隠します。

Oracle サーバーは、非 Oracle システムにアクセスするために、Oracle8i 異種サービスとシステム固有の Transparent Gateway を使用します。たとえば、Oracle 分散システムに DB2 データベースが含まれている場合は、システム内の Oracle データベースが通信できるように、DB2 固有の Transparent Gateway を取得する必要があります。

関連項目： Oracle のレプリケーション機能の詳細は、『Oracle8i 分散システム』を参照してください。

異種サービス

異種サービスは、Oracle8i サーバー内の統合されたコンポーネントであり、Open Gateway 製品の最新パッケージを使用可能にするテクノロジーです。異種サービスは、Oracle Gateway 製品や他の異種アクセス機能のための共通アーキテクチャと管理メカニズムを提供するだけでなく、Oracle Open Gateway のほとんどの旧リリースのユーザー向けに上位互換性機能を提供します。

関連項目： 異種サービスの概要は、『Oracle8i 分散システム』を参照してください。

Transparent Gateway エージェント

アクセス先となる非 Oracle システムごとに、異種サービスでは、非 Oracle システムにアクセスするための「Transparent Gateway エージェント」が必要です。Transparent Gateway エージェントは、Oracle データベースと非 Oracle データベースとの通信を容易にし、Oracle サーバー内の異種サービス・コンポーネントを使用します。エージェントは、Oracle サーバーにかわって SQL、プロシージャおよびトランザクション要求を非 Oracle システム側で実行します。

関連項目： インストールと構成の詳細は、Oracle Transparent Gateway 関連のインストレーション・ガイドおよびユーザーズ・ガイドを参照してください。

機能

異種サービスの機能は、次のとおりです。

機能	用途
分散トランザクション	トランザクションの一貫性を保証しながら、1つのトランザクションで Oracle システムと非 Oracle システムの両方にまたがることを可能にします。
SQL 変換	非 Oracle システムからのデータを、1つのローカル・データベースに格納されているかのようにして Oracle 環境に統合します。SQL 文は、非 Oracle システムが認識できる SQL 文に透過的に変換されます。
プロシージャ型アクセス	PL/SQL リモート・プロシージャ・コールを使用した、Oracle8i サーバーからのメッセージングなど、プロシージャ型システムへのアクセスを可能にします。
データ・ディクショナリ変換	非 Oracle システムを Oracle サーバーのように見せかけます。Oracle のデータ・ディクショナリ表への参照を含む SQL 文は、非 Oracle システムのデータ・ディクショナリ表への参照を含む SQL 文に変換されます。
パススルー SQL	アプリケーション・プログラマは、非 Oracle システムの SQL 言語を使用して、Oracle アプリケーションから非 Oracle システムに直接アクセスできます。
ストアド・プロシージャへのアクセス	SQL ベースの非 Oracle システムに含まれているストアド・プロシージャには、PL/SQL リモート・プロシージャであるかのようにアクセスできます。
NLS サポート	マルチバイト・キャラクタセットをサポートしており、非 Oracle システムと Oracle8i サーバー間でキャラクタ・セットを変換します。
マルチスレッド・エージェント	必要なプロセス数を減少させて、オペレーティング・システムのスレッド機能を活用します。
エージェント自動登録	リモート・ホスト上で異種サービスの構成データを更新するプロセスを自動化し、異種データベース・リンクを介した正常動作を保証します。

注意： 前述の機能のすべてが異種サービス・エージェントや Oracle Gateway でサポートされているとは限りません。サポートしている機能については、システム固有のマニュアルを参照してください。

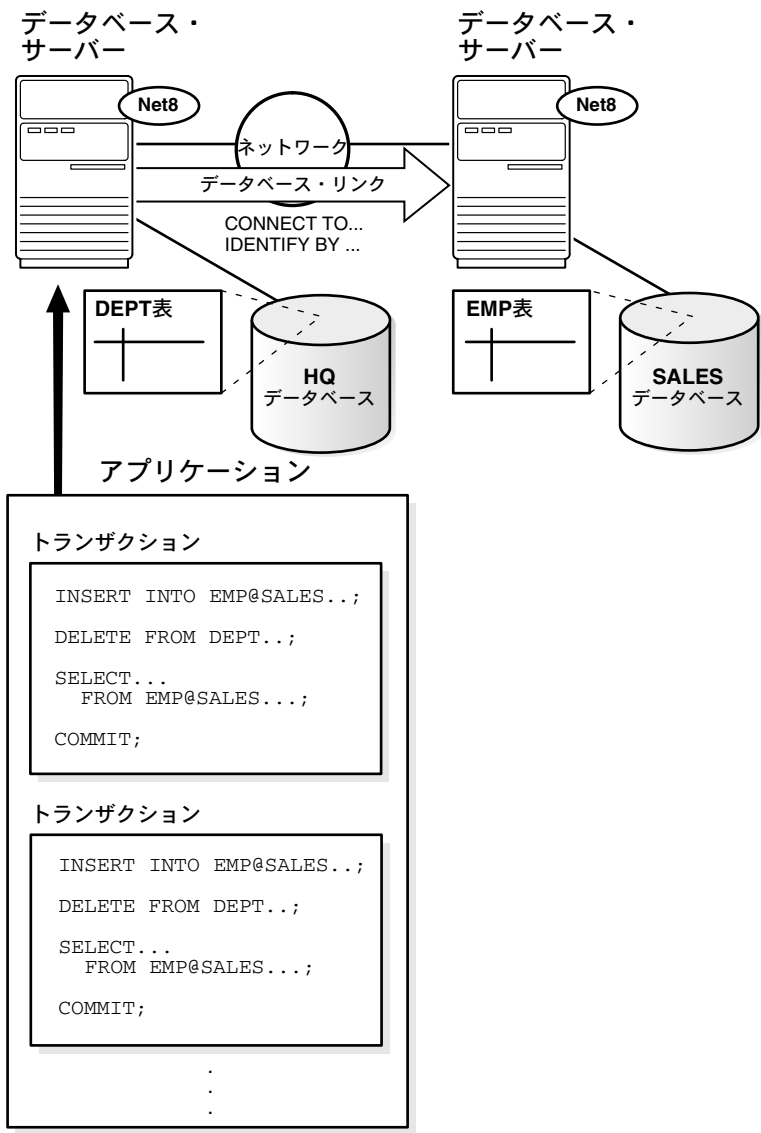
クライアント / サーバー・データベース・アーキテクチャ

「データベース・サーバー」とはデータベースを管理している Oracle ソフトウェアで、「クライアント」とはサーバーに対して情報を要求するアプリケーションのことです。ネットワーク上の各コンピュータは、1つ以上のデータベースのホストとなる「ノード」です。分

分散データベース・システム内の各ノードは、状況に応じてクライアントまたはサーバー、あるいはその両方として動作します。

たとえば、図 30-2 では、HQ データベースのホストは、そのローカル・データに対して文が発行されるときにはデータベース・サーバーとして動作します（たとえば、各トランザクション内の 2 番目の文はローカル DEPT 表に対して文を発行します）。また、文がリモート・データに対して発行されるときにはクライアントとして動作します（たとえば、各トランザクション内の最初の文は SALES データベース内のリモート表 EMP に対して発行されます）。

図 30-2 Oracle 分散データベース・システム



直接接続と間接接続

クライアントは、データベース・サーバーに「直接的」または「間接的」に接続できます。直接接続が発生するのは、クライアントがサーバーに接続し、そのサーバーに格納されているデータベースからの情報にアクセスする場合です。たとえば、図 30-2 のように、HQ データベースに接続してこのデータベース上の DEPT 表にアクセスする場合は、次の文を発行できます。

```
SELECT * FROM dept;
```

ここではリモート・データベース上のオブジェクトにアクセスしていないため、これは直接的な問合せです。

これに対して、間接接続が発生するのは、クライアントがサーバーに接続し、異なるサーバー上のデータベースに格納された情報にアクセスする場合です。たとえば、図 30-2 のように、HQ データベースに接続してから、リモート SALES データベース上の EMP 表にアクセスする場合は、次の文を発行できます。

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

ここでは、アクセス先のオブジェクトは直接接続しているデータベース上にないため、これは間接的な問合せです。

データベース・リンク

分散データベース・システムの中心となる概念は、「データベース・リンク」です。データベース・リンクは、2つの物理データベース・サーバー間の接続のことで、クライアントから1つの論理データベースとしてアクセスできるようにします。

ここでは、次の内容を取り上げます。

- データベース・リンク
- データベース・リンクを使用する理由
- データベース・リンク内のグローバル・データベース名
- データベース・リンク名
- データベース・リンクのタイプ
- データベース・リンクのユーザー
- データベース・リンクの作成：例
- スキーマ・オブジェクトとデータベース・リンク
- データベース・リンクの制限事項

データベース・リンク

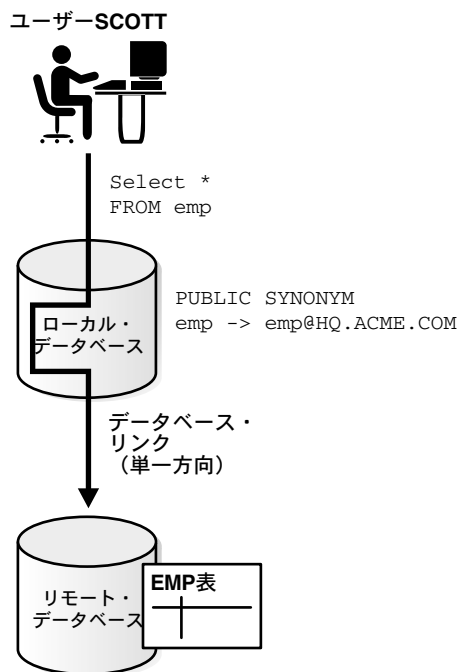
データベース・リンクは、Oracle データベース・サーバーから他のデータベース・サーバーまでの一方向の通信経路を定義するポインタです。このリンク・ポインタは、実際にはデータ・ディクショナリ表のエントリとして定義されます。リンクにアクセスするには、データ・ディクショナリ・エントリを含むローカル・データベースに接続されている必要があります。

データベース・リンク接続は一方向であるため、ローカル・データベース A に接続されているクライアントは、データベース A に格納されたリンクを使用してリモート・データベース B の情報にアクセスできますが、データベース B に接続されているユーザーが同じリンクを使用してデータベース A のデータにアクセスすることはできません。データベース B のローカル・ユーザーがデータベース A のデータにアクセスする場合は、データベース B のデータ・ディクショナリに格納されたリンクを定義する必要があります。

データベース・リンク接続により、ローカル・ユーザーはリモート・データベース上のデータにアクセスできます。このように接続を確立するには、分散システム内の各データベースが、ネットワーク・ドメイン内に一意の「グローバル・データベース名」を持つ必要があります。グローバル・データベース名により、分散システム内の特定のデータベース・サーバーが識別されます。

図 30-3 に、ユーザー SCOTT がグローバル名 HQ.ACME.COM を持つリモート・データベースの EMP 表にアクセスする場合の例を示します。

図 30-3 データベース・リンク



データベース・リンクには、「プライベート」リンクと「パブリック」リンクがあります。プライベート・リンクの場合は、そのリンクを作成したユーザーのみがアクセス権を持ち、パブリック・リンクの場合は、すべてのデータベース・ユーザーがアクセス権をもちます。

両者の基本的な相違点は、リモート・データベースへの接続方法です。リモート・データベースにアクセスするユーザーは、次のいずれかの方法を使用します。

- 「接続ユーザー・リンク」は、それ自体で接続します。つまり、リモート・データベースにも、ローカル・データベース上のアカウントと同じユーザー名でアカウントを持っている必要があります。
- 「固定ユーザー・リンク」は、そのリンクで参照されるユーザー名とパスワードで接続します。たとえば、JANE がユーザー名およびパスワード SCOTT/TIGER で HR データベースに接続する固定ユーザー・リンクを使用する場合は、SCOTT で接続します。JANE には、SCOTT に直接付与されている HR 内でのすべての権限と、SCOTT が HR データベース内で付与されているすべてのデフォルト・ロールが付与されます。
- 「カレント・ユーザー・リンク」は、グローバル・ユーザーで接続します。ローカル・ユーザーは、リンク定義にグローバル・ユーザーのパスワードを格納しなくても、ストアード・プロシージャのコンテキスト内でグローバル・ユーザーで接続できます。たとえ

ば、JANE は SCOTT が記述したプロシージャにアクセスし、SCOTT のアカウントと HR データベース上の SCOTT のスキーマにアクセスできます。カレント・ユーザー・リンクは、Oracle Advanced Security オプションの一部です。

データベース・リンクを作成するには、CREATE DATABASE LINK 文を使用します。リンクの作成後は、それを使用して SQL 文にスキーマ・オブジェクトを指定できます。

共有データベース・リンク

共有データベース・リンクとは、ローカル・サーバー・プロセスとリモート・データベースの間のリンクです。このリンクは、複数のクライアント・プロセスが同じリンクを同時に使用できるため共有されます。

ローカル・データベースがデータベース・リンクを介してリモート・データベースに接続されている場合は、どちらのデータベースも専用サーバー・モードまたはマルチスレッド・サーバー（MTS）モードで動作できます。次の表は、それぞれのモードを示しています。

ローカル・データベースのモード	リモート・データベースのモード
専用	専用
専用	マルチスレッド
マルチスレッド	専用
マルチスレッド	マルチスレッド

共有データベース・リンクは、この 4 つの構成のどれにでも使用できます。共有リンクと標準データベース・リンクには、次のような違いがあります。

- データベース・リンクを介して同じスキーマ・オブジェクトにアクセスする複数のユーザーが、1 つのネットワーク接続を共有できます。
- ユーザーが特定のサーバー・プロセスからリモート・サーバーへの接続を確立する必要がある場合、そのプロセスではすでに確立されているリモート・サーバー接続を再使用できます。接続を再使用できるのは、同じデータベース・リンクにより同じサーバー・プロセス上で接続が確立されている場合で、異なるセッションの場合もあります。非共有データベース・リンクでは、接続が複数セッション間で共有されることはありません。
- MTS 構成で共有データベース・リンクを使用すると、ネットワーク接続はローカル・サーバー内の共有サーバー・プロセスから直接確立されます。ローカル・マルチスレッド・サーバー上の非共有データベース・リンクの場合、この接続はローカル・ディスクパッチャを介して確立されているため、ローカル・ディスクパッチャのコンテキストを切り替え、ディスクパッチャを介してデータを送る必要があります。

関連項目： マルチスレッド・サーバー・オプションの詳細は、『Oracle8i Net8 管理者ガイド』を参照してください。

データベース・リンクを使用する理由

データベース・リンクの大きな利点は、オブジェクト所有者の権限セットによって境界を設け、ユーザーがリモート・データベース内の他のユーザーのオブジェクトにアクセスできるようになることです。つまり、ローカル・ユーザーは、リモート・データベース上のユーザーにならなくても、リモート・データベースへのリンクにアクセスできます。

たとえば、従業員が買掛管理 (A/P) アプリケーションに経費精算書を発行し、A/P アプリケーションを使用するユーザーが HR データベースから従業員情報を取り出す必要があるとします。A/P ユーザーは、HR データベースに接続し、必要な情報を取り出すリモート HR データベース内でストアド・プロシージャを実行する必要があります。この A/P ユーザーは、HR データベースのユーザーにならなくても作業を実行できますが、プロシージャによって制限される制御された方法でしか HR 情報にアクセスできないようにするべきです。

データベース・リンクにより、ローカル・ユーザーにリモート・データベースへの制限付きアクセス権を付与できます。カレント・ユーザー・リンクを使用すると、集中管理されるグローバル・ユーザーを作成できます。この種のユーザーのパスワード情報は、管理者や他の従業員から隠されます。たとえば、A/P ユーザーは HR データベースに SCOTT でアクセスできますが、固定ユーザー・リンクとは異なり、SCOTT の資格証明はデータベース・ユーザーから見えない場所に格納されます。

固定ユーザー・リンクを使用すると、パスワード情報が LINK\$ データ・ディクショナリ表に暗号化されないままに格納される、非グローバル・ユーザーを作成できます。固定ユーザー・リンクの作成は容易で、SSL またはディレクトリ要件がないためオーバーヘッドもあまり必要としません。ただし、パスワード情報がデータ・ディクショナリに格納されるため、セキュリティ上のリスクが生じます。

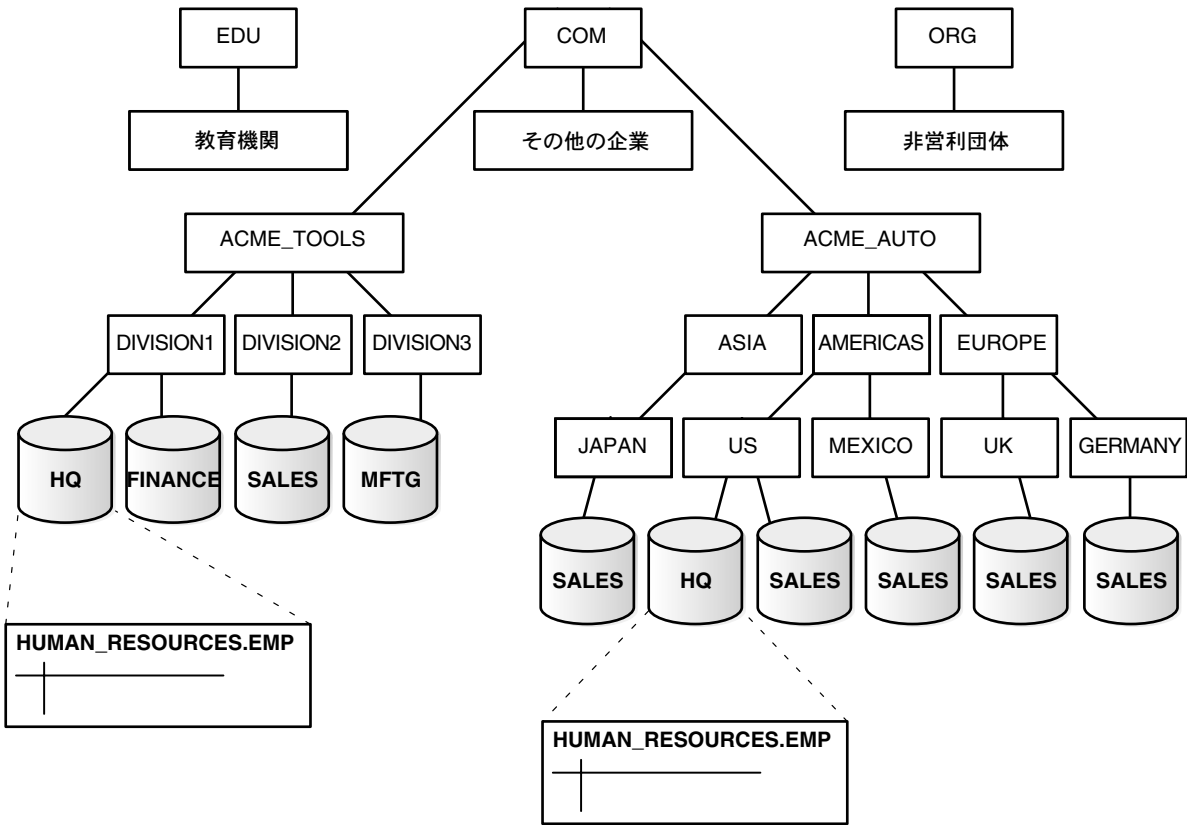
関連項目： データベース・リンク・ユーザーの説明と、パスワードを管理者以外の従業員から隠す方法の詳細は、『Oracle8i 分散システム』を参照してください。

データベース・リンク内のグローバル・データベース名

データベース・リンクの機能を理解するには、最初にグローバル・データベース名の役割を理解する必要があります。分散データベース内の各データベースは、そのグローバル・データベース名によって一意に識別されます。データベースのグローバル・データベース名は、データベース作成時に DB_DOMAIN 初期化パラメータで指定されたデータベースのネットワーク・ドメインに、DB_NAME 初期化パラメータで指定された個々のデータベース名を接頭辞として追加することによって作成されます。

一例として、図 30-4 に、ネットワークにおけるデータベースの代表的な階層型配置を示します。

図 30-4 ネットワーク・ディレクトリとグローバル・データベース名



データベース名には、ツリーのリーフに続いてルートへのパスが指定されます。たとえば、MFTG データベースは、COM ドメインの ACME_TOOLS ブランチの DIVISION3 にあります。MFTG のグローバル・データベース名は、次のようにツリーの各ノードを連結して作成されます。

MFTG.DIVISION3.ACME_TOOLS.COM

複数のデータベースの名前を同じにすることは可能ですが、各データベースのグローバル・データベース名は一意にする必要があります。たとえば、ネットワーク・ドメイン US.AMERICAS.ACME_AUTO.COM と UK.EUROPE.ACME_AUTO.COM には、それぞれ SALES データベースが存在します。グローバル・データベースの命名体系では、

AMERICAS 部門の SALES データベースと EUROPE 部門の SALES データベースは次のように区別されます。

```
SALES.US.AMERICAS.ACME_AUTO.COM  
SALES.UK.EUROPE.ACME_AUTO.COM
```

関連項目： グローバル・データベース名の指定方法と変更方法は、『Oracle8i 分散システム』を参照してください。

データベース・リンク名

通常、データベース・リンク名は、その参照先となるリモート・データベースのグローバル・データベース名と同じです。たとえば、データベースのグローバル・データベース名が SALES.US.ORACLE.COM であれば、データベース・リンク名も SALES.US.ORACLE.COM となります。

初期化パラメータ GLOBAL_NAMES を TRUE に設定すると、データベース・リンク名がリモート・データベースのグローバル・データベース名と同一かどうかを確認されます。たとえば、HQ のグローバル・データベース名が HQ.ACME.COM のときに、GLOBAL_NAMES が TRUE であれば、リンク名は HQ.ACME.COM でなければなりません。Oracle は、init.ora ファイル内の DB_DOMAIN の設定ではなく、データ・ディクショナリに格納されているグローバル・データベース名のドメイン部分をチェックするので注意してください（『Oracle8i 分散システム』を参照）。

初期化パラメータ GLOBAL_NAMES を FALSE に設定した場合、グローバル・ネーミングを使用する必要はありません。希望どおりのデータベース・リンク名を使用できます。たとえば、HQ.ACME.COM へのデータベース・リンクに FOO という名前を与えることができます。

注意： Oracle Advanced Replication など、多数の役立つ機能ではグローバル・ネーミングを規定する必要があるため、グローバル・ネーミングを使用することをお勧めします。

グローバル・ネーミングを使用可能にすると、データベース・リンク名はリンクが指すデータベースのグローバル名と同じであるため、データベース・リンクは分散データベースのユーザーにとって実質上透過的になります。たとえば、次の文は、ローカル・データベース内でリモート・データベース SALES へのデータベース・リンクを作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';
```

関連項目： 初期化パラメータ GLOBAL_NAMES を指定する方法の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

データベース・リンクのタイプ

Oracle では、「プライベート」、「パブリック」および「グローバル」の各データベース・リンクを作成できます。これらの基本的なリンク・タイプには、どのユーザーがリモート・データベースへのアクセスを許可されるかに応じて、次のような違いがあります。

タイプ	所有者	説明
プライベート	リンクを作成したユーザー。 DBA_DB_LINKS または ALL_DB_LINKS を介して 所有権データにアクセス。	ローカル・データベースの特定のスキーマ内でリンクを作成します。プライベート・データベース・リンクの所有者またはスキーマ内の PL/SQL サブプログラムのみが、プライベート・データベース・リンクを使用して、対応するリモート・データベース内のデータベース・オブジェクトにアクセスできます。
パブリック	PUBLIC というユーザー。 DBA_DB_LINKS または ALL_DB_LINKS を介して 所有権データにアクセス。	データベース単位のリンクを作成します。すべてのユーザーとデータベース内の PL/SQL サブプログラムは、このリンクを使用して、対応するリモート・データベース内のデータベース・オブジェクトにアクセスできます。
グローバル	PUBLIC というユーザー。 DBA_DB_LINKS または ALL_DB_LINKS を介して 所有権データにアクセス。	ネットワーク単位のリンクを作成します。Oracle ネットワークで Oracle Names が使用されている場合、システム内のネーム・サーバーにより、ネットワーク内のすべての Oracle データベース用にグローバル・データベース・リンクが自動的に作成され、管理されます。任意のデータベース内のすべてのユーザーと PL/SQL サブプログラムは、グローバル・リンクを使用して、対応するリモート・データベース内のデータベース・オブジェクトにアクセスできます。

分散データベースで使用するデータベース・リンクのタイプは、システムを使用するアプリケーションの特定の要件によって決まります。次のような利点と不利な点を考慮してください。

プライベート・データベース・リンク	このリンクでは、このリンクを使用してリモート・データベースにアクセスできるのは、プライベート・リンクの所有者、または同じスキーマ内のサブプログラムのみであるため、パブリック・リンクやグローバル・リンクよりも安全です。
パブリック・データベース・リンク	多数のユーザーがリモート Oracle データベースへのアクセス・パスを必要とする場合は、データベース内のユーザー全員のために単一のパブリック・データベース・リンクを作成できます。

グローバル・データベース・リンク
Oracle ネットワークで Oracle Names が使用される場合、管理者はシステム内のすべてのデータベースのグローバル・データベース・リンクを容易に管理できます。データベース・リンク管理は一元化され、簡素化されます。

関連項目： 各種データベース・リンクの作成方法と、リンク情報へのアクセス方法の詳細は、『Oracle8i 分散システム』を参照してください。

データベース・リンクのユーザー

リンクの作成時に、どのユーザーがデータにアクセスするためにリモート・データベースに接続する必要があるかを判断します。次の表に、データベース・リンクに関与するユーザーの各カテゴリの違いを示します。

ユーザー・タイプ	意味	リンク作成構文のサンプル
接続ユーザー	固定ユーザー名とパスワードが指定されていないデータベース・リンクにアクセスするローカル・ユーザー。SYSTEM が問合せでパブリック・リンクにアクセスする場合、接続ユーザーは SYSTEM で、Oracle はリモート・データベース内の SYSTEM スキーマに接続します。 注意： 接続ユーザーは、リンクにアクセスするユーザーですが、リンクの作成者でなくてもかまいません。	CREATE PUBLIC DATABASE LINK hq USING 'hq';
カレント・ユーザー	CURRENT USER データベース・リンク内のグローバル・ユーザー。グローバル・ユーザーは、X.509 資格証明とプライベート・キーで認証され、リンクに関与する両方のデータベース上でユーザーであることが必要です。カレント・ユーザー・リンクは、Oracle Advanced Security (OAS) オプションの一部です。 関連項目： グローバル・セキュリティの詳細は、『Oracle8i Advanced Security 管理者ガイド』を参照してください。	CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER USING 'hq';
固定ユーザー	ユーザー名 / パスワードがリンク定義に含まれているユーザー。リンクに固定ユーザーが含まれている場合、リモート・データベースへの接続には、その固定ユーザーのユーザー名とパスワードが使用されます。	CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY doe USING 'hq';

関連項目： リンク作成時にユーザーを指定する方法の詳細は、『Oracle8i 分散システム』を参照してください。

接続ユーザーのデータベース・リンク

接続ユーザー・リンクには、接続文字列は対応付けられていません。接続ユーザー・リンクの利点は、そのリンクを参照するユーザーがリモート・データベースに同じユーザーとして

接続することです。また、このリンクには接続文字列は対応付けられていないため、パスワードがデータ・ディクショナリにクリアテキスト形式で格納されることはありません。

接続ユーザー・リンクには、いくつか不利な点もあります。このリンクでは、ユーザーは接続先となるリモート・データベース上にアカウントと権限を持つ必要があるため、管理者にとっては権限管理作業が増えます。また、ユーザーに必要以上の権限を付与すると、最小限度の権限、つまり、ユーザーにはジョブの実行に必要最小限の権限のみを付与するという基本的なセキュリティの概念に違反することになります。

接続ユーザー・データベース・リンクを使用できるかどうかは、いくつかの要因によって決まります。特に、ユーザーがパスワードを使用して Oracle に認証されるか、オペレーティング・システムまたはネットワーク認証サービスによって外部で認証されるかが重要です。ユーザーが外部で認証される場合に、接続ユーザー・リンクを使用できるかどうかは、リモート・データベースがユーザーのリモート認証を受け入れるかどうかによっても異なります。これは、REMOTE_OS_AUTHENT 初期化パラメータで設定されます。

REMOTE_OS_AUTHENT パラメータの動作は次のとおりです。

REMOTE_OS_AUTHENT の設定	動作
リモート・データベースに対して TRUE	外部認証されるユーザーは、接続ユーザー・データベース・リンクを使用してリモート・データベースに接続できます。
リモート・データベースに対して FALSE	外部認証されるユーザーは、安全なプロトコルが使用されるか、Oracle Advanced Security オプションがサポートされているネットワーク認証サービスが使用されない限り、接続ユーザー・データベース・リンクを使用してリモート・データベースに接続できません。

固定ユーザーのデータベース・リンク

名前付きリンクには、プライマリ・データベース内のユーザーが、接続文字列内のそのユーザーのセキュリティ・コンテキストでリモート・データベースに接続されるという利点があります。たとえば、ローカル・ユーザー JOE は、固定ユーザー SCOTT とパスワード TIGER を指定して、自分のスキーマ内でパブリック・データベース・リンクを作成できます。JANE がこの固定ユーザー・リンクを問合せに使用する場合、JANE はローカル・データベース上のユーザーですが、リモート・データベースには SCOTT/TIGER で接続します。

固定ユーザー・リンクの場合は、接続文字列にユーザー名とパスワードが対応付けられています。このユーザー名とパスワードは、データ・ディクショナリの LINK\$ 表に暗号化されない形式で格納されます。このため、固定ユーザー・データベース・リンクではセキュリティが低下するおそれがあります。たとえば、SELECT ANY TABLE 権限を持つユーザーは、O7_DICTIONARY_ACCESSIBILITY 初期化パラメータが TRUE に設定されていればデータ・ディクショナリにアクセスできるため、固定ユーザーによる認証の信頼性が低下します。

注意： O7_DICTIONARY_ACCESSIBILITY のデフォルト値は FALSE です。

このセキュリティ問題の一例として、JANE には HR データベースに SCOTT/TIGER で接続するプライベート・リンクを使用するための権限が付与されていないが、O7_DICTIONARY_ACCESSIBILITY 初期化パラメータが TRUE に設定されているデータベースに対する SELECT ANY TABLE 権限が付与されているとします。この場合、JANE は LINK\$ から選択し、HR への接続文字列が SCOTT/TIGER であることを読み取ることができます。JANE は、HR が格納されているホストにアカウントを持っていれば、このホストに接続し、パスワード TIGER を使用して HR に SCOTT で接続できます。また、JANE がローカルに接続すると、SCOTT のすべての権限が付与され、監査レコードは JANE が SCOTT であるかのように記録されます。

カレント・ユーザー・データベース・リンク

カレント・ユーザー・データベース・リンクでは、グローバル・ユーザーが使用されます。グローバル・ユーザーは、X.509 資格証明とプライベート・キーで認証され、リンクに関与する両方のデータベース上でユーザーであることが必要です。

CURRENT_USER リンクを起動するユーザーは、グローバル・ユーザーでなくてもかまいません。たとえば、JANE がパスワードによって買掛管理データベースに認証される場合は、HR データベースからデータを取り出すストアド・プロシージャにアクセスできます。このプロシージャでは、JANE を HR にグローバル・ユーザー SCOTT で接続する、カレント・ユーザー・データベース・リンクが使用されます。SCOTT はグローバル・ユーザーであるため、SSL を介して資格証明とプライベート・キーによって認証されますが、JANE はそうではありません。

カレント・ユーザー・データベース・リンクでは、次の結果が生じるので注意してください。

- カレント・ユーザー・データベース・リンクがストアド・オブジェクト内からアクセスされない場合、カレント・ユーザーはそのリンクにアクセスしている接続ユーザーと同じです。たとえば、SCOTT がカレント・ユーザー・リンクを介して SELECT 文を発行する場合、カレント・ユーザーは SCOTT です。
- プロシージャ、ビューまたはトリガーなど、データベース・リンクにアクセスするストアド・オブジェクトを実行する場合、カレント・ユーザーは、そのストアド・オブジェクトを「コール」するユーザーではなく、それを「所有」するユーザーです。たとえば、JANE がプロシージャ SCOTT.P (SCOTT が作成) をコールする場合に、カレント・ユーザー・リンクがコール先プロシージャ内に含まれていれば、SCOTT はそのリンクのカレント・ユーザーです。

- そのストアド・オブジェクトが実行者権限ファンクション、プロシージャまたはパッケージであれば、実行者の認可 ID を使用してリモート・ユーザーで接続されます。たとえば、ユーザー JANE がプロシージャ SCOTT.P（SCOTT が作成した実行者権限プロシージャ）をコールする場合に、そのリンクがプロシージャ SCOTT.P に含まれていれば、JANE がカレント・ユーザーです。

関連項目： データベース・リンクに関連したセキュリティ上の問題点の詳細は、30-24 ページの「分散データベースのセキュリティ」を参照してください。

データベース・リンクの作成：例

データベース・リンクを作成するには、CREATE DATABASE LINK 文を使用します。次の表に、「ローカル」データベースから「リモート」の SALES.US.AMERICAS.ACME_AUTO.COM データベースへのデータベース・リンクを作成する SQL 文の例を示します。

SQL 文	接続先データベース	接続するユーザー	リンク・タイプ
CREATE DATABASE LINK sales.us.americas.acme_auto.com USING 'sales_us';	ネット・サービス名 SALES_US を使用して SALES に	接続ユーザー	プライベート接続ユーザー
CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';	サービス名 AM_SLS を使 用して SALES に	カレント・グローバル・ ユーザー	プライベート・ カレント・ユー ザー
CREATE DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger USING 'sales_us';	ネット・サービス名 SALES_US を使用して SALES に	パスワード TIGER を使 用して SCOTT で	プライベート固 定ユーザー
CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'rev';	ネット・サービス名 REV を使用して SALES に	パスワード TIGER を使 用して SCOTT で	パブリック固定 ユーザー
CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING 'sales';	ネット・サービス名 SALES を使用して SALES に	パスワード Bhide を使 用して ANUPAM とし て認証された、パスワード TIGER を持つ SCOTT で	共有パブリック 固定ユーザー

関連項目：

- リンクの作成方法は、『Oracle8i 分散システム』を参照してください。
- CREATE DATABASE LINK の構文は、『Oracle8i SQL リファレンス』を参照してください。

スキーマ・オブジェクトとデータベース・リンク

データベース・リンクを作成すると、リモート・データベース上のオブジェクトにアクセスする SQL 文を実行できます。たとえば、データベース・リンク FOO を使用してリモート・オブジェクト EMP にアクセスするには、次の文を発行します。

```
SELECT * FROM emp@foo;
```

分散システムでのデータ操作では、データベース・リンクを使用した正しい書式のオブジェクト名を使用することが重要です。

データベース・リンクを使用したスキーマ・オブジェクトのネーミング

Oracle は、グローバル・データベース名と次のスキーマを使用して、スキーマ・オブジェクトにグローバル名を与えます。

```
schema.schema_object@global_database_name
```

各値の意味は次のとおりです。

schema	データの論理構造の集合、すなわちスキーマ・オブジェクト。スキーマはデータベース・ユーザーによって所有され、そのユーザーと同じ名前を持ちます。各ユーザーが単一のスキーマを所有します。
schema_object	表、索引、ビュー、シノニム、プロシージャ、パッケージなどの論理データ構造、またはデータベース・リンク。
global_database_name	特定のリモート・データベースを識別する名前。この名前は、リモート・データベースの初期化パラメータ DB_NAME および DB_DOMAIN を連結したものと同等であることが必要です。ただし、パラメータ GLOBAL_NAMES が FALSE に設定されている場合は、任意の名前を使用できます。

たとえば、データベース SALES.DIVISION3.ACME.COM へのデータベース・リンクを使用すると、ユーザーまたはアプリケーションは次のようにしてリモート・データを参照できます。

```
SELECT * FROM scott.emp@sales.division3.acme.com; # emp table in scott's schema
SELECT loc FROM scott.dept@sales.division3.acme.com;
```

GLOBAL_NAMES が FALSE に設定されていれば、SALES.DIVISION3.ACME.COM へのリンクには任意の名前を使用できます。たとえば、このリンクに FOO という名前を付けることができます。この場合、データベースへのアクセスには次の文を使用します。

```
SELECT name FROM scott.emp@foo; # link name different from global name
```

スキーマ・オブジェクトのシノニム

Oracle では、「シノニム」を作成してデータベース・リンク名をユーザーから隠すことができます。シノニムにより、ローカル・データベース上の表にアクセスするときと同じ構文を使用して、リモート・データベース上の表にアクセスできます。たとえば、リモート・データベースの表に対して次の問合せを発行するとします。

```
SELECT * FROM emp@hq.acme.com;
```

同じデータにアクセスするかわりに次の問合せを発行できるように、EMP@HQ.ACME.COM のシノニム EMP を作成できます。

```
SELECT * FROM emp;
```

関連項目： データベース・リンクを使用して指定したオブジェクトのシノニムを作成する方法は、『Oracle8i 分散システム』を参照してください。

スキーマ・オブジェクトの名前解決

スキーマ・オブジェクトへのアプリケーション参照を解決する（名前解決プロセス）ために、Oracle はオブジェクト名を階層形式で生成します。たとえば、Oracle では、1つのデータベース内で各スキーマの名前が一意であり、1つのスキーマ内では各オブジェクトの名前が一意であることが保証されています。結果として、スキーマ・オブジェクトの名前はデータベース内で常に一意です。また、Oracle は、オブジェクトのローカル名へのアプリケーション参照を解決します。

分散データベースでは、表などのスキーマ・オブジェクトはシステム内のすべてのアプリケーションからアクセスできます。Oracle は、階層型命名モデルをグローバル・データベース名で拡張して、効率的に「グローバル・オブジェクト名」を作成し、分散データベース・システム内のスキーマ・オブジェクトへの参照を解決します。たとえば、問合せでは、表が存在するデータベースを含めた完全修飾名を指定して、リモート表を参照できます。

たとえば、ローカル・データベースにユーザー SYSTEM で接続するとします。

```
CONNECT system/manager@sales1
```

データベース・リンク HQ.ACME.COM を使用して次の文を発行し、リモート・データベース HQ 上の SCOTT および JANE スキーマ内のオブジェクトにアクセスします。

```
SELECT * FROM scott.emp@hq.acme.com;
INSERT INTO jane.accounts@hq.acme.com (acc_no, acc_name, balance)
VALUES (5001, 'BOWER', 2000);
```

```
UPDATE jane.accounts@hq.acme.com  
SET balance = balance + 500;  
DELETE FROM jane.accounts@hq.acme.com  
WHERE acc_name = 'BOWER';
```

データベース・リンクの制限事項

次の操作は、データベース・リンクを介して実行できません。

- リモート・オブジェクトに対する権限の付与。
- 一部のリモート・オブジェクトに対する DESCRIBE 操作の実行。ただし、次のリモート・オブジェクトは DESCRIBE 操作をサポートしています。
 - 表
 - ビュー
 - プロシージャ
 - ファンクション
- リモート・オブジェクトの ANALYZE。
- 参照整合性の定義または規定。
- リモート・データベース内のユーザーに対するロール権限の付与。
- リモート・データベースに対するデフォルト以外のロールの取得。たとえば、JANE がローカル・データベースに接続し、SCOTT で接続して固定ユーザー・リンクを使用するストアド・プロシージャを実行すると、JANE はリモート・データベースに対する SCOTT のデフォルト・ロールを付与されます。JANE が SET ROLE を発行してデフォルト以外のロールを取得することはできません。
- MTS 接続を使用するハッシュ問合せ結合の実行。
- SSL または NT システム固有な認証を介さない、認証なしのカレント・ユーザー・リンクの使用。

分散データベースの管理

ここでは、Oracle 分散データベース・システムでのデータベース管理に関連するトピックについて説明します。

- [サイト自律性](#)
- [分散データベースのセキュリティ](#)
- [データベース・リンクの監査](#)
- [管理ツール](#)

関連項目： 同種システムの管理方法と異種システムの管理方法は、『Oracle8i 分散システム』を参照してください。

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サーバーが、他のすべてのデータベースから独立して管理されることを意味します。複数のデータベースの協同作業は可能ですが、それぞれのデータベースは、別々に管理される個々のデータ・リポジトリです。Oracle 分散データベースでのサイト自律性には、次のような利点があります。

- システムのノードは、互いに対等な関係を維持するために必要な、企業やグループの論理的な組織体系を反映できます。
- ローカル管理者は、対応するローカルなデータを制御します。そのため、各データベース管理者の担当ドメインが小さくなり、管理しやすくなります。
- 1つのノードの障害によって分散データベースの他のノードを中断させる可能性が低くなります。1つのデータベースで障害が発生しても、すべての分散操作を停止する必要はなく、パフォーマンスのボトルネックになることもありません。
- 管理者は、孤立したシステム障害からのリカバリ作業を、システム内の他のノードとは独立して実行できます。
- データ・ディクショナリがローカル・データベースごとに存在します。ローカル・データへのアクセスには、グローバル・カタログは不要です。
- 各ノードで単独でソフトウェアをアップグレードできます。

Oracle では分散データベース・システムの各データベースを独立して管理できますが、システムのグローバルな要件を無視してよいわけではありません。たとえば、次の操作が必要になることがあります。

- サーバー間接続用に作成するリンクをサポートするために、各データベースでユーザー・アカウントを追加作成します。
- DISTRIBUTED_LOCK_TIMEOUT、DISTRIBUTED_TRANSACTIONS、COMMIT_POINT_STRENGTH など、追加の初期化パラメータを設定します。

分散データベースのセキュリティ

Oracle では、分散データベース・システムのために、非分散データベース環境で使用可能なすべてのセキュリティ機能がサポートされます。それには次のものが含まれます。

- ユーザーとロールに対するパスワード認証
- ユーザーとロールに対する次のようなタイプの外部認証
 - 接続ユーザー・リンク用の Kerberos パージョン 5
 - 接続ユーザー・リンク用の DCE

- クライアント / サーバー間接続とサーバー間の相互接続でのログイン・パケットの暗号化

この後の項では、Oracle 分散データベース・システムを構成する場合の追加の考慮事項について説明します。

- データベース・リンクを介した認証
- ユーザー・アカウントとロールのサポート
- ユーザーと権限の集中管理
- データの暗号化

関連項目： 外部認証の詳細は、『Oracle8i Advanced Security 管理者ガイド』を参照してください。

データベース・リンクを介した認証

データベース・リンクには、「プライベート」リンクと「パブリック」リンクがあり、それぞれ「認証」と「無認証」に分かれています。パブリック・リンクを作成するには、リンク作成文に PUBLIC キーワードを指定します。たとえば、次の文を発行できます。

```
CREATE PUBLIC DATABASE LINK foo USING 'sales';
```

認証リンクを作成するには、データベース・リンク作成文に CONNECT TO 句、AUTHENTICATED BY 句またはその両方を指定します。たとえば、次の文を発行できます。

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'sales';
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO mick IDENTIFIED BY jagger
    AUTHENTICATED BY david IDENTIFIED BY bowie USING 'sales';
```

次の表に、リンクを介してリモート・データベースにアクセスする方法を示します。

リンク・タイプ	認証の有無	セキュリティ・アクセス
プライベート	なし	リモート・データベースへの接続時に、Oracle はローカル・セッションから取得したセキュリティ情報（ユーザー ID/ パスワード）を使用します。したがって、このリンクは接続ユーザー・データベース・リンクです。2つのデータベース間では、パスワードの同期をとる必要があります。

リンク・タイプ	認証の有無	セキュリティ・アクセス
プライベート	あり	<p>ユーザー ID/ パスワードは、ローカル・セッションのコンテキストではなくリンク定義から取得されます。したがって、このリンクは固定ユーザー・データベース・リンクです。</p> <p>この構成では、2つのデータベースのパスワードが異なってもかまいませんが、ローカル・データベース・リンクのパスワードはリモート・データベースのパスワードと同じでなければなりません。パスワードはローカル・システム・カタログにクリアテキスト形式で格納されるため、セキュリティ上のリスクが大きくなります。</p>
パブリック	なし	<p>動作はプライベートな無認証リンクと同じですが、だれでもリモート・データベースへのこのポイントを参照できます。</p>
パブリック	あり	<p>ローカル・データベース上のユーザーは、だれでもリモート・データベースにアクセスでき、全員が同じユーザー ID/ パスワードを使用して接続を確立します。また、パスワードはローカル・カタログにクリアテキスト形式で格納されるため、ローカル・データベースに対して十分な権限を持っている場合は、パスワードを見ることができます。</p>

パスワードなしの認証

接続ユーザーまたはカレント・ユーザー・データベース・リンクを使用する場合は、Kerberos などの外部認証ソースを使用して「end-to-end セキュリティ」を取得できます。end-to-end 認証では、資格証明はサーバー間で渡され、同じドメインに属するデータベース・サーバーの認証を受けることができます。たとえば、JANE がローカル・データベース上で外部で認証され、接続ユーザー・リンクを使用してリモート・データベースに JANE で接続する場合、ローカル・サーバーはセキュリティ・チケットをリモート・データベースに渡します。

ユーザー・アカウントとロールのサポート

分散データベース・システムでは、システムを使用するアプリケーションをサポートするのに必要なユーザー・アカウントとロールを慎重に計画する必要があります。次の点に注意してください。

- サーバー間の相互接続を確立するために必要なユーザー・アカウントは、分散データベース・システム内のすべてのデータベースで使用可能であることが必要です。
- アプリケーション権限を分散データベース・アプリケーション・ユーザーから使用できるようにするために必要なロールは、分散データベース・システム内のすべてのデータベースに存在していることが必要です。

分散データベース・システム内のノードのためにデータベース・リンクを作成する場合、各サイトでそのリンクを使用するサーバー間の相互接続をサポートするには、どんなユーザー・アカウントとロールが必要かを決定してください。

通常、分散環境では、ユーザーは多数のネットワーク・サービスにアクセスする必要があります。各ユーザーが各ネットワーク・サービスにアクセスできるようにするための認証を個別に構成する必要があると（特に大規模システムの場合）、セキュリティ管理が煩雑になります。

関連項目： システムで各種データベース・リンクをサポートするために必要なユーザー・アカウントの詳細は、『Oracle8i 分散システム』を参照してください。

ユーザーと権限の集中管理

Oracle では、分散システムに関与するユーザーと権限を様々な方法で管理できます。たとえば、次の選択肢があります。

- エンタープライズ・ユーザー管理。SSL を介して認証されるグローバル・ユーザーを作成し、独立したエンタープライズ・ディレクトリ・サービスを介して、これらのユーザーとその権限をディレクトリ内で管理できます。
- ネットワーク認証サービス。この一般的なテクニックでは、分散環境のセキュリティ管理作業が簡素化されます。Net8 Oracle Advanced Security オプションを使用して、Net8 と Oracle 分散データベース・システムのセキュリティを拡張できます。Windows NT システム固有な認証は、Oracle 以外の認証ソリューションの一例です。

関連項目：

グローバル・ユーザーのセキュリティの詳細は、次のマニュアルを参照してください。

- 『Oracle8i Net8 管理者ガイド』
- 『Oracle8i Advanced Security 管理者ガイド』

スキーマ依存のグローバル・ユーザー ユーザーおよび権限管理を集中化するための選択肢の 1 つは、次を作成することです。

- 集中化されたディレクトリ内でグローバル・ユーザーを作成
- グローバル・ユーザーの接続先となるすべてのデータベース内でユーザーを作成

たとえば、次の SQL 文でグローバル・ユーザー FRED を作成できます。

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England';
```

この方法では、単一のグローバル・ユーザーを一元化されたディレクトリで認証できます。

スキーマ依存のグローバル・ユーザーによるソリューションを使用する場合は、ユーザー FRED がアクセスする必要のあるすべてのデータベースに、このユーザーを作成する必要があります。ほとんどのユーザーは、アプリケーション・スキーマにアクセスするための許可が必要ですが、自分のスキーマは必要でないため、すべてのグローバル・ユーザーについて各データベース内で別個のアカウントを作成すると、大きなオーバーヘッドが生じます。このため、Oracle はスキーマ独立ユーザーもサポートしています。これは、すべてのデータベース内で単一の汎用スキーマにアクセスするグローバル・ユーザーです。

グローバル・ユーザーとデータベース・スキーマの分離 Oracle8i は、グローバル・ユーザーをエンタープライズ・ディレクトリ・サービスで集中管理する機能をサポートしています。ディレクトリ内で管理されるユーザーを、「エンタープライズ・ユーザー」と呼びます。このディレクトリには、次の情報が含まれています。

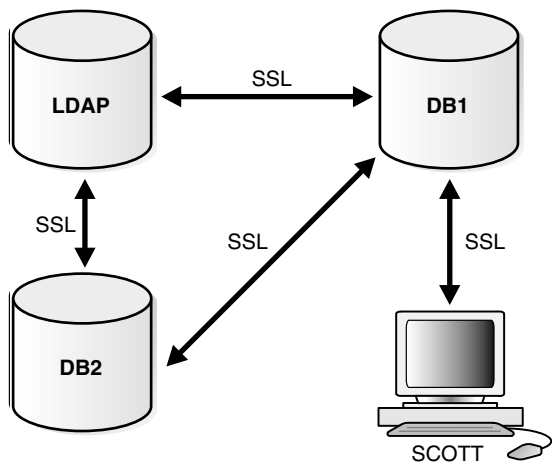
- エンタープライズ・ユーザーが分散システム内でアクセスできるデータベース
- エンタープライズ・ユーザーが各データベース上で使用できるロール
- エンタープライズ・ユーザーが各データベース上で接続できるスキーマ

各データベースの管理者は、エンタープライズ・ユーザーが接続を必要とする各データベース上で、それらのユーザーごとにグローバル・ユーザー・アカウントを作成する必要はありません。そのかわりに、複数のエンタープライズ・ユーザーが同じデータベース・スキーマに接続できます。このスキーマを「共有スキーマ」と呼びます。

たとえば、JANE、BILL および SCOTT が、いずれも人事管理アプリケーションを使用するものとし、HR アプリケーション・オブジェクトは、すべて HR データベース上の GUEST スキーマに含まれています。この場合は、共有スキーマとして使用されるローカル・グローバル・ユーザー・アカウントを作成できます。このグローバル・ユーザー名、つまり共有スキーマ名は GUEST です。JANE、BILL および SCOTT は、いずれもディレクトリ・サービス内でエンタープライズ・ユーザーとして作成されます。また、この 3 人のユーザーはディレクトリ内の HRAPP スキーマにマップされ、HR アプリケーション内で異なる認可を割り当てることができます。

図 30-5 に、エンタープライズ・ディレクトリ・サービスを使用するグローバル・ユーザー・セキュリティの例を示します。

図 30-5 グローバル・ユーザー・セキュリティ



エンタープライズ・ディレクトリ・サービスに、DB1 および DB2 のエンタープライズ・ユーザーに関する次の情報が含まれているとします。

データベース	ロール	スキーマ	エンタープライズ・ユーザー
DB1	clerk1	guest	bill
			scott
DB2	clerk2	guest	jane
			scott

また、DB1 と DB2 のローカル管理者が次の文を発行したとします。

データベース CREATE 文	
DB1	<pre>CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk1 GRANT select ON emp; CREATE PUBLIC DATABASE LINK db2_link CONNECT AS CURRENT_USER USING 'db2';</pre>
DB2	<pre>CREATE USER guest IDENTIFIED GLOBALLY; CREATE ROLE clerk2 GRANT select ON dept;</pre>

エンタープライズ・ユーザー SCOTT が、DB2 を伴う分散トランザクションを実行するために、ローカル・データベース DB1 への接続を要求したとします。ここでは次のステップが発生します（正確な順序は異なる場合があります）。

1. エンタープライズ・ユーザー SCOTT とデータベース DB1 が、SSL を使用して相互を認証します。
2. SCOTT が次の文を発行します。

```
SELECT e.ename, d.loc  
FROM emp e, dept@db2 link d  
WHERE e.deptno=d.deptno
```

3. データベース DB1 および DB2 が、SSL を使用して相互を認証します。
4. DB1 はエンタープライズ・ディレクトリ・サービスに問い合せて、エンタープライズ・ユーザー SCOTT が DB1 へのアクセス権を付与されているかどうかを判別し、SCOTT がロール CLERK1 を使用してローカル・スキーマ GUEST にアクセスできることが検出されます。
5. データベース DB2 は、エンタープライズ・ディレクトリ・サービスに問い合せて、エンタープライズ・ユーザー SCOTT が DB2 へのアクセス権を付与されているかどうかを判別し、SCOTT がロール CLERK2 を使用してローカル・スキーマ GUEST にアクセスできることが検出されます。
6. エンタープライズ・ユーザー SCOTT が、ロール CLERK2 で DB2 のスキーマ GUEST にログインし、SELECT を発行して必要な情報を取得し、それを DB1 に転送します。
7. DB1 は、要求されたデータを DB2 から受け取って、それをクライアント SCOTT に戻します。

関連項目：

エンタープライズ・ユーザーのセキュリティの詳細は、次のマニュアルを参照してください。

- 『Oracle8i Net8 管理者ガイド』
- 『Oracle8i Advanced Security 管理者ガイド』

データの暗号化

Oracle Advanced Security は、ネットワーク・データの暗号化とチェックサムを使用してデータの読み込みや変更を禁止するという点でも、Net8 とその関連製品を拡張します。このオプションは、RSA Data Security RC4 またはデータ暗号化規格（DES）暗号化アルゴリズムを使用することによって、許可なしでは表示できないようにデータを保護します。

データが修正、削除または再実行されていないことを確認するため、Oracle Advanced Security のセキュリティ・サービスでは、暗号保護メッセージ・ダイジェストを生成し、それをネットワーク内で送信される各パケットに含めることができます。

関連項目：

これらの機能を含め、Oracle Advanced Security の各機能の詳細は、次のマニュアルを参照してください。

- 『Oracle8i Net8 管理者ガイド』
- 『Oracle8i Advanced Security 管理者ガイド』

データベース・リンクの監査

監査操作は、常にローカルに実行する必要があります。つまり、ユーザーがローカル・データベースで作業し、データベース・リンクを介してリモート・データベースにアクセスする場合に、それぞれのデータベース内で適切な監査オプションが設定されていれば、ローカル・アクションはローカル・データベース内で監査され、リモート・アクションはリモート・データベース内で監査されます。

リモート・データベースでは、正常な接続要求と後続の SQL 文が他のサーバーからきたものか、ローカル接続されたクライアントからきたものかを判別できません。たとえば、次のような場合です。

- 固定ユーザー・リンク HR.ACME.COM がローカル・ユーザー JANE を、リモート・ユーザー SCOTT でリモートの HR データベースに接続します。
- SCOTT はリモート・データベース上で監査されます。

リモート・データベース・セッション中に実行される監査は、SCOTT がローカルに接続してそこで同じアクションを実行したかのように監査されます。JANE がリモート・データベース内で実行する操作を監査することが目的であれば、リンクに埋め込まれているユーザー名（この場合は SCOTT）によるアクションを獲得するために、リモート・データベース内で監査オプションを設定する必要があります。

注意： グローバル・ユーザーの場合は、グローバル・ユーザー名を監査できます。

リモート・オブジェクトには、ローカル監査オプションを設定できません。したがって、データベース・リンクの使用は監査できませんが、リモート・データベース上でリモート・オブジェクトへのアクセスを監査することはできます。

管理ツール

データベース管理者は、Oracle 分散データベース・システムを管理するときに、複数のツールの中から選択して使用できます。

- [Enterprise Manager](#)

- サードパーティの管理ツール
- SNMP のサポート

Enterprise Manager

Enterprise Manager は、Oracle のデータベース管理ツールです。Oracle Enterprise Manager のグラフィカル・コンポーネント（Enterprise Manager/GUI）を使用すると、グラフィカル・ユーザー・インタフェース（GUI）を利用してデータベース管理作業を実行できます。Oracle Enterprise Manager の非グラフィカル・コンポーネントは、コマンドライン・インタフェースを提供されています。

Oracle Enterprise Manager には、管理機能のために使用しやすいインタフェースが用意されています。Oracle Enterprise Manager を使用すると、次の作業を実行できます。

- データベースの起動、シャットダウン、バックアップおよびリカバリなど、従来の管理作業を実行します。これらの作業を実行する場合に、Oracle Enterprise Manager のグラフィカル・インタフェースを使用すると、SQL 文を手動で入力するかわりに、マウスのポイント・アンド・クリックでコマンドをすばやく簡単に実行できます。
- 複数の作業を同時に実行します。Oracle Enterprise Manager では、複数のウィンドウを同時にオープンできるため、複数の管理作業や管理以外の作業を同時に実行できます。
- 複数のデータベースを管理します。Oracle Enterprise Manager を使用すると、単一のデータベースを管理したり、複数のデータベースを同時に管理できます。
- データベース管理作業を集中化します。ローカル・データベースと、世界中の Oracle プラットフォームで実行されているリモート・データベースの両方を管理できます。さらに、これらの Oracle プラットフォームを、Net8 がサポートする任意のネットワーク・プロトコルによって接続できます。
- SQL、PL/SQL および Oracle Enterprise Manager の文を動的に実行します。Oracle Enterprise Manager を使用して、文を入力、編集および実行できます。Oracle Enterprise Manager は、実行した文の履歴のメンテナンスもします。
このため、再び入力しなくても文を再実行できます。これは、分散データベース・システムで一連の長い文を繰り返して実行する必要がある場合に特に役立つ機能です。
- グラフィカル・ユーザー・インタフェースが使用できない場合や、使用しないほうがよい場合には、Oracle Enterprise Manager の行モード・インタフェースによって管理作業を実行します。
- グローバル・ユーザー、グローバル・ロールおよびエンタープライズ・ディレクトリ・サービスなどのセキュリティ機能を管理します。

サードパーティの管理ツール

現在、60 を超える会社から Oracle データベース管理およびネットワーク管理に役立つ 150 を超える製品が出されており、本当の意味でオープンな環境が整っています。

SNMP のサポート

ネットワーク管理機能に加え、Oracle の「シンプル・ネットワーク管理プロトコル (SNMP) サポート」により、任意の SNMP ベースのネットワーク管理システムから Oracle Server の検索や問合せができます。SNMP は、広く使用されている次のような多数のネットワーク管理システムの基礎として受け入れられている標準です。

- HP の OpenView
- Digital の POLYCENTER Manager (NetView 上)
- IBM の NetView/6000
- Novell の NetWare Management System
- SunSoft の SunNet Manager

関連項目： SNMP の詳細は、『Oracle SNMP サポート・リファレンス・ガイド』を参照してください。

分散システムでのトランザクション処理

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。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、そのユーザーによってコミットまたはロールバックされた時点で終了します。

「リモート・トランザクション」には、単一のリモート・ノードにアクセスする文のみが含まれます。これに対して、「分散トランザクション」には、複数のノードにアクセスする文が含まれます。

この後の各項では、トランザクション処理の重要な概念と、トランザクションが分散データベース内のデータにアクセスする方法について説明します。

- [リモート SQL 文](#)
- [分散 SQL 文](#)
- [リモート文と分散文のための共有 SQL](#)
- [リモート・トランザクション](#)
- [分散トランザクション](#)
- [2 フェーズ・コミット・メカニズム](#)

- データベース・リンクの解決
- スキーマ・オブジェクトの名前解決

リモート SQL 文

「リモート問合せ」文とは、同一のリモート・ノードに存在している、1つ以上のリモート表から情報を選択する問合せのことです。たとえば、次の問合せは、リモート・データベース SALES の SCOTT スキーマにある DEPT 表からのデータにアクセスします。

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

「リモート更新」文とは、同一のリモート・ノードに存在している1つ以上の表すべてのデータを変更する更新のことです。たとえば、次の問合せは、リモート・データベース SALES の SCOTT スキーマにある DEPT 表を更新します。

```
UPDATE scott.dept@mktng.us.americas.acme_auto.com  
SET loc = 'NEW YORK'  
WHERE deptno = 10;
```

注意： リモート更新には1つ以上のリモート・ノードからデータを取り出す副問合せが含まれることがありますが、更新操作は1つのリモート・ノードのみで発生するため、その文はリモート更新に分類されます。

分散 SQL 文

「分散問合せ」文は、複数のノードから情報を取り出します。たとえば、次の問合せは、ローカル・データベースとリモート・データベース SALES からのデータにアクセスします。

```
SELECT ename, dname  
FROM scott.emp e, scott.dept@sales.us.americas.acme_auto.com d  
WHERE e.deptno = d.deptno;
```

「分散更新」文は、複数のノードのデータを変更します。分散更新は、複数の異なるノード上のデータにアクセスする複数のリモート更新を含む、プロシージャまたはトリガーなどの PL/SQL サブプログラム・ユニットを使用して実行できます。たとえば、次の PL/SQL プログラム・ユニットは、ローカル・データベースとリモート・データベース SALES の表を更新します。

```
BEGIN
  UPDATE scott.dept@sales.us.americas.acme_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
    WHERE deptno = 10;
END;
COMMIT;
```

このプログラムの中の文はリモート・ノードに送られ、その実行は1単位として成功または失敗します。

リモート文と分散文のための共有 SQL

共有 SQL を使用するリモート文または分散文の仕組みは、実際にはローカル文の場合と同じです。SQL テキストが合致することと、参照オブジェクトが合致することが必要です。使用可能な場合は、共有 SQL 領域を任意の文や分解された問合せのローカル処理とリモート処理に使用できます。

リモート・トランザクション

「リモート・トランザクション」とは、単一のリモート・ノードを参照する1つ以上のリモート文を含むトランザクションのことです。たとえば、次のトランザクションには、それぞれリモート・データベース SALES にアクセスする2つの文が含まれています。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.acme_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

分散トランザクション

「分散トランザクション」とは、分散データベースにある複数の異なるノード上で、個別にまたはグループとしてデータを更新する、1つ以上の文を含むトランザクションのことです。たとえば、次のトランザクションは、ローカル・データベースとリモート・データベース SALES を更新します。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

```
UPDATE scott.emp  
  SET deptno = 11  
  WHERE deptno = 10;  
COMMIT;
```

注意： トランザクションのすべての文が単一のリモート・ノードのみを参照する場合、そのトランザクションはリモート・トランザクションであって、分散トランザクションではありません。

2 フェーズ・コミット・メカニズム

データベースでは、1つのトランザクション（分散または非分散）内のすべての文が1単位としてコミットされるかロールバックされるかのいずれかであることが保証される必要があります。進行中のトランザクションの影響は、全ノードにある他のすべてのトランザクションからは参照できないようにする必要があります。この透過性は、問合せ、更新またはリモート・プロシージャ・コールなど、あらゆるタイプの操作が含まれるトランザクションに当てはまります。

非分散データベースにおけるトランザクション制御の一般的なメカニズムの詳細は、『Oracle8i 概要』を参照してください。分散データベースでは、ネットワーク全体で同じ特性によってトランザクション制御を調整し、ネットワーク障害やシステム障害が発生した場合にもデータの一貫性が保たれる必要があります。

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に係するすべてのデータベース・サーバーが、そのトランザクション内の文のすべてをコミットするか、そうでなければすべてをロールバックするかのどちらか一方のみになるように保証するものです。また、2 フェーズ・コミット・メカニズムは、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙の DML 操作も保護します。

関連項目： Oracle の 2 フェーズ・コミット・メカニズムの詳細は、『Oracle8i 分散システム』を参照してください。

データベース・リンクの解決

「グローバル・オブジェクト名」とは、データベース・リンクを使用して指定されるオブジェクトです。グローバル・オブジェクト名の重要なコンポーネントは次のとおりです。

- オブジェクト名
- データベース名
- ドメイン

次の表に、明示的に指定されたグローバル・データベース・オブジェクト名のコンポーネントを示します。

文	オブジェクト	データベース	ドメイン
SELECT * FROM joan.dept@sales.acme.com	dept	sales	acme.com
SELECT * FROM emp@mktg.us.acme.com	emp	mktg	us.acme.com

SQL 文にグローバル・オブジェクト名の参照が含まれていると、Oracle はそのグローバル・オブジェクト名に指定されているデータベース名と合致する名前を、データベース・リンク内で検索します。たとえば、次の文を発行するとします。

```
SELECT * FROM scott.emp@orders.us.acme.com;
```

Oracle は、データベース・リンク ORDERS.US.ACME.COM を検索します。この操作によって、指定されたりモート・データベースへのパスが判別されます。

合致するデータベース・リンクは、常に次の順序で検索されます。

- 1. SQL 文を発行したユーザーのスキーマにあるプライベート・データベース・リンク
- 2. ローカル・データベース内のパブリック・データベース・リンク
- 3. グローバル・データベース・リンク（Oracle Names Server が使用可能な場合のみ）

グローバル・データベース名が完全な場合の解決

完全なグローバル・データベース名を指定する次の SQL 文を発行したとします。

```
SELECT * FROM emp@prod1.us.oracle.com
```

この場合、データベース名（PROD1）とドメイン・コンポーネント（US.ORACLE.COM）の両方が指定されているため、プライベート、パブリックおよびグローバルの各データベース・リンクが検索されます。検索されるのは、指定したグローバル・データベース名と合致するリンクのみです。

グローバル・データベース名が部分的な場合の解決

ドメインのどの部分を指定しても、完全なグローバル・データベース名を指定したものと見なされます。SQL 文で部分的なグローバル・データベース名（つまり、データベース・コンポーネントのみ）を指定すると、DB_DOMAIN パラメータ内の値が DB_NAME パラメータ内の値に付加され、完全な名前が構築されます。たとえば、次の文を発行するとします。

```
CONNECT scott/tiger@locdb
SELECT * FROM scott.emp@orders;
```

LOCDB のネットワーク・ドメインが US.ACME.COM であれば、このドメインが ORDERS に付加され、完全なグローバル・データベース名 ORDERS.US.ACME.COM が構築されます。構築されたグローバル名のみと合致するデータベース・リンクが検索されます。合致するリンクが見つからない場合はエラーが戻され、SQL 文は実行できません。

グローバル・データベース名を指定しない場合の解決

グローバル・オブジェクト名がローカル・データベース内のオブジェクトを参照する場合に、データベース・リンク名が @ 記号を使用して指定されていないければ、そのオブジェクトがローカルであることが自動的に検出され、データベース・リンクは検索されず、オブジェクト参照の解決には使用されません。たとえば、次の文を発行するとします。

```
CONNECT scott/tiger@locdb
SELECT * from scott.emp;
```

2 番目の文では、データベース・リンクの接続文字列を使用してグローバル・データベース名を指定していないため、データベース・リンクは検索されません。

解決のための検索の終了

Oracle は、最初に合致するデータベース・リンクが見つかっていても、検索を停止しない場合があります。リモート・データベースへの完全パス（リモート・アカウントとサービス名の両方）が判別されるまで、Oracle は合致するプライベート、パブリックおよびネットワークの各データベース・リンクを検索する必要があります。

最初に合致した名前によって、次の表のようにリモート・スキーマが判別されます。

ユーザー指定	Oracle の動作	例
CONNECT 句を指定しない場合	「接続ユーザー」データベース・リンクを使用します。	CREATE DATABASE LINK k1 USING 'prod'
CONNECT TO ... IDENTIFIED BY 句を指定する場合	「固定ユーザー」データベース・リンクを使用します。	CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY tiger USING 'prod'
CONNECT TO CURRENT_USER 句を指定する場合	「カレント・ユーザー」データベース・リンクを使用します。	CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'

ユーザー指定	Oracle の動作	例
USING 句を指定しない場合	データベース文字列を指定するリンクが見つかるまで検索します。合致するデータベース・リンクが見つかって、文字列が識別されていなければ、Oracle はエラーを戻します。	CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER

Oracle は、完全パスを判別すると、同じローカル・セッションのための同一接続がまだオープンされていないければ、リモート・セッションを作成します。セッションがすでに存在する場合は、それが再使用されます。

スキーマ・オブジェクトの名前解決

ローカルの Oracle データベースが、SQL 文を発行したローカル・ユーザーのために指定されたリモート・データベースに接続すると、オブジェクト解決はリモート・ユーザーがその SQL 文を発行した場合と同様に続行されます。最初に合致した名前により、次のルールに従ってリモート・スキーマが判別されます。

使用するデータベース・リンク	オブジェクト解決が進行するスキーマ
固定ユーザー・データベース・リンク	リンク作成文に指定したスキーマ
接続ユーザー・データベース・リンク	接続ユーザーのリモート・スキーマ
カレント・ユーザー・データベース・リンク	カレント・ユーザーのスキーマ

そのオブジェクトが見つからない場合は、リモート・データベースのパブリック・オブジェクトがチェックされます。オブジェクトを解決できなくても、確立されたリモート・セッションはそのままですが、SQL 文は実行できず、エラーが戻されます。

名前解決の例

次の例は、分散データベース・システムにおけるグローバル・オブジェクトの名前解決を示しています。後述のすべての例で、次のことを前提としています。

- リモート・データベース名は、SALES.DIVISION3.ACME.COM です。
- ローカル・データベース名は、HQ.DIVISION3.ACME.COM です。
- Oracle Names Server は（したがって、グローバル・データベース・リンクも）使用できません。

例：完全オブジェクト名の解決 次の例は、Oracle が完全なグローバル・オブジェクト名を解決し、プライベートとパブリックのデータベース・リンクの両方を使用して、リモート・データベースへの適切なパスを判別する方法を示しています。ここでは、リモート表 EMP がスキーマ TSMITH に含まれているとします。

SCOTT がローカル・データベースで次の文を発行した場合を考えます。

```
CONNECT scott/tiger@hq
```

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

後で、JWARD が接続して次の文を発行します。

```
CONNECT jward/bronco@hq
```

```
CREATE DATABASE LINK sales.division3.acme.com
CONNECT TO tsmith IDENTIFIED BY radio;
```

```
UPDATE tsmith.emp@sales.division3.acme.com
SET deptno = 40
WHERE deptno = 10;
```

最後の文は次のように処理されます。

1. 完全なグローバル・データベース名は、JWARD の更新文で参照されているものと判別されます。したがって、名前が合致するデータベース・リンクは、ローカル・データベース内で検索されます。
2. 合致するプライベート・データベース・リンクが、スキーマ JWARD 内で見つかります。ただし、プライベート・データベース・リンク JWARD.SALES.DIVISION3.ACME.COM は、リモート・データベース SALES への完全パスではなく、リモート・アカウントのみを示しています。したがって、今度は合致するパブリック・データベース・リンクが検索されます。
3. パブリック・データベース・リンクが SCOTT のスキーマ内で見つかります。このパブリック・データベース・リンクから、Oracle はサービス名 DBSTRING を使用します。
4. Oracle は、合致するプライベートの固定ユーザー・データベース・リンクから取り出したリモート・アカウントと組み合わせて完全パスを判別し、リモート・データベース SALES への接続をユーザー TSMITH/RADIO で確立する操作に進みます。
5. これで、リモート・データベースは EMP 表へのオブジェクト参照を解決できます。Oracle は TSMITH スキーマ内で検索して、参照される EMP 表を検出します。
6. リモート・データベースは文の実行を完了し、結果をローカル・データベースに戻します。

例：部分的なオブジェクト名の解決 次の例は、Oracle が部分的なグローバル・オブジェクト名を解決し、プライベートとパブリックのデータベース・リンクの両方を使用して、リモート・データベースへの適切なパスを判別する方法を示しています。

この例の前提は次のとおりです。

- リモート・データベース SALES の表 EMP は、スキーマ SCOTT ではなくスキーマ TSMITH に含まれています。
- パブリック・シノニム EMP はローカル・データベース HQ に格納されており、リモート・データベース SALES にある TSMITH.EMP を指します。
- 30-40 ページの「例：完全オブジェクト名の解決」に示すパブリック・データベース・リンクは、すでにローカル・データベース HQ に作成されています。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

ローカル・データベース HQ で、次の文を発行したとします。

```
CONNECT scott/tiger@hq
```

```
CREATE DATABASE LINK sales.division3.acme.com;
```

```
DELETE FROM emp@sales
WHERE empno = 4299;
```

最後の DELETE 文は次のように処理されます。

1. Oracle は、部分的なグローバル・オブジェクト名が SCOTT の DELETE 文中で参照されていることに注目し、ローカル・データベースのドメインを使用して、それを次のように完全なグローバル・オブジェクト名に展開します。

```
DELETE FROM emp@sales.division3.acme.com
WHERE empno = 4299;
```

2. ローカル・データベース内で、合致する名前を持つデータベース・リンクが検索されます。
3. 合致する「プライベート」な接続ユーザー・リンクがスキーマ SCOTT 内で見つかりますが、このプライベート・データベース・リンクではパスがまったく指定されていません。Oracle は、パスのリモート・アカウント部に接続ユーザー名 / パスワードを使用し、合致する「パブリック」データベース・リンクを検索して検出します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

- 4. Oracle は、パブリック・データベース・リンクから、データベースのネット・サービス名 DBSTRING を取得します。この時点で、完全パスが判別されています。
- 5. Oracle はリモート・データベースに SCOTT/TIGER で接続し、スキーマ SCOTT 内でオブジェクト EMP を検索しますが、見つかりません。
- 6. リモート・データベースは、パブリック・シノニム EMP を検索して検出します。
- 7. リモート・データベースは文を実行し、結果をローカル・データベースに戻します。

ビュー、シノニムおよびプロシージャでのグローバル・ネーム解決

ビュー、シノニムまたは PL/SQL プログラム・ユニット（プロシージャ、ファンクションまたはトリガーなど）は、リモート・スキーマ・オブジェクトを、そのグローバル・オブジェクト名で参照できます。グローバル・オブジェクト名が完全であれば、それが展開されないままオブジェクトの定義が格納されます。ただし、部分的な名前の場合は、ローカル・データベース名のドメインを使用して名前が展開されます。

次の表に、Oracle がビュー、シノニムおよびプログラム・ユニットの部分的なグローバル・オブジェクト名の展開を完了する場合を示します。

ユーザーの操作	Oracle の動作
ビューの作成	部分的なグローバル名は展開されず、それを定義する問合せの正確なテキストがデータ・ディクショナリに格納されます。かわりに、そのビューを使用する文が解析されるたびに、部分的なグローバル・オブジェクト名が展開されます。
シノニムの作成	部分的なグローバル名が展開されます。データ・ディクショナリに格納されるシノニムの定義には、展開後のグローバル・オブジェクト名が含まれます。
プログラム・ユニットのコンパイル	部分的なグローバル名が展開されます。

グローバル名に変更があった場合の結果

グローバル名の変更は、部分的なグローバル・オブジェクト名を使用してリモート・データを参照するビュー、シノニムおよびプロシージャに影響することがあります。参照先データベースのグローバル名が変化すると、ビューおよびプロシージャは存在しないデータベースや間違ったデータベースを参照しようとするおそれがあります。これに対して、シノニムでは実行時にデータベース・リンク名が展開されないため、変更はありません。

たとえば、SALES.UK.ACME.COM および HR.UK.ACME.COM という 2 つのデータベースを考えます。また、SALES データベースに次のビューとシノニムが含まれているとします。

```
CREATE VIEW employee_names AS
    SELECT ename FROM scott.emp@hr;

CREATE SYNONYM employee FOR scott.emp@hr;
```

Oracle は、EMPLOYEE シノニムの定義を展開して次のように格納します。

```
scott.emp@hr.uk.acme.com
```

例 1: 両方のデータベース名に変更がある場合 最初に、営業部門と人事管理部門の両方がアメリカに移転される状況を考えます。その結果、対応するグローバル・データベース名はどちらも次のように変更されます。

- SALES.US.ACME.COM
- HR.US.ACME.COM

SALES への問合せ	変更前の展開	変更後の展開
SELECT * FROM employee_names	SELECT * from scott.emp@hr.uk.acme.com	SELECT * from scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

例 2: 一方のデータベース名に変更がある場合 ここでは、営業部門のみがアメリカに移転され、人事管理部門はイギリスに残る場合を考えます。その結果、グローバル・データベース名は次のようになります。

- SALES.US.ACME.COM
- HR.UK.ACME.COM

SALES への問合せ	変更前の展開	変更後の展開
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

EMPLOYEE_NAMES ビューを定義している問合せは、存在しないグローバル・データベース名に展開されます。これに対して、EMPLOYEE シノニムは、引き続き正しいデータベース HR.UK.ACME.COM を参照します。

分散データベース・アプリケーションの開発

分散システムにおけるアプリケーション開発では、非分散システムには見られない問題が発生します。ここでは、分散アプリケーションの開発に関連した次のトピックについて説明します。

- 分散データベース・システムにおける透過性
- リモート・プロシージャ・コール (RPC)
- 分散問合せの最適化

関連項目： 分散システム用のアプリケーションの開発方法は、『Oracle8i 分散システム』を参照してください。

分散データベース・システムにおける透過性

システムで作業するユーザーに対して、Oracle 分散データベース・システムを透過的にするアプリケーションを最小限の労力で開発できます。このような透過性の目標は、分散データベース・システムを単一の Oracle データベースであるかのように見せることです。その結果、システムの開発者およびユーザーは、分散データベース・アプリケーションの開発の難航やユーザーの生産性低下を招く原因になりかねない複雑な作業から解放されます。

この後の各項では、分散データベース・システムでの透過性についてさらに説明します。

位置の透過性

Oracle 分散データベース・システムには、アプリケーションの開発者や管理者がデータベース・オブジェクトの物理的な位置をアプリケーションやユーザーから隠せるようにする機能があります。「位置の透過性」は、アプリケーションが接続しているノードに関係なく、表などのデータベース・オブジェクトをユーザーが広く参照できる場合に存在します。位置の透過性には、次のような利点があります。

- リモート・データへのアクセスが単純。データベース・ユーザーはデータベース・オブジェクトの物理的な位置を知る必要がないためです。
- 管理者は、エンドユーザーや既存のデータベース・アプリケーションに影響を与えることなくデータベース・オブジェクトを移動できます。

多くの場合、管理者と開発者は、シノニムを使用することによって、アプリケーション・スキーマ内の表およびサポートするオブジェクトについての位置の透過性を確立します。たとえば、次の文は、別のリモート・データベースにある表のシノニムをデータベース内に作成します。

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.acme_auto.com
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.acme_auto.com
```

こうすれば、次のような問合せによってリモート表にアクセスするかわりに、

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.acme_auto.com e,
       scott.dept@sales.us.americas.acme_auto.com d
 WHERE e.deptno = d.deptno;
```

アプリケーションは、次のように、リモート表の位置を指定する必要のない単純な問合せを発行できます。

```
SELECT ename, dname
  FROM emp e, dept d
 WHERE e.deptno = d.deptno;
```

シノニムに加えて、開発者はビューやストアド・プロシージャを使用して、分散データベース・システムで作業するアプリケーションの位置の透過性を確立することもできます。

SQL と COMMIT の透過性

Oracle の分散データベース・アーキテクチャでは、問合せ、更新およびトランザクションの透過性も提供されます。たとえば、SELECT、INSERT、UPDATE および DELETE などの標準的な SQL 文は、非分散データベース環境の場合と同じように動作します。さらに、アプリケーションは、標準的な SQL 文 COMMIT、SAVEPOINT および ROLLBACK によってトランザクションを制御します。分散トランザクションを制御するために複雑なプログラミングや他の特殊な操作を実行する必要はありません。

- 1 つのトランザクション内の文から、任意の数のローカル表またはリモート表を参照できます。
- Oracle により、分散トランザクションにかかわるすべてのノードが同じアクションを実行することが保証されます。トランザクションはすべてコミットされるか、またはすべてロールバックされるかのどちらか一方になります。
- 分散トランザクションのコミット時にネットワーク障害またはシステム障害が発生すると、トランザクションは自動的、透過的かつグローバルに解決されます。つまり、ネットワークまたはシステムがリストアされると、ノードはトランザクションをすべてコミットするか、またはすべてをロールバックするかのどちらか一方にします。

内部操作 コミットされた各トランザクションには、トランザクション内の文によって加えられた変更を一意に識別するシステム変更番号 (SCN) が関連付けられます。分散データベースでは、次の場合に通信ノードの SCN が調整されます。

- 1 つ以上のデータベース・リンクによって記述されるパスを使用して接続が確立される場合
- 分散 SQL 文が実行される場合
- 分散トランザクションがコミットされる場合

なによりも大きな利点は、分散データベース・システムのノード間の SCN の調整により、文レベルとトランザクション・レベルの両方で、グローバルな分散読み込みの一貫性が得られることです。必要であれば、グローバルな時間ベースの分散リカバリも完了できます。

レプリケーション透過性

さらに Oracle では、システム内の複数のノード間で透過的にデータをレプリケートするためのたくさんの機能が用意されています。Oracle のレプリケーション機能の詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

リモート・プロシージャ・コール (RPC)

開発者は、PL/SQL のパッケージとプロシージャを、分散データベースで動作するアプリケーションをサポートするようにコーディングできます。アプリケーションでは、ローカル・データベースでの作業を実行するローカル・プロシージャ・コールや、リモート・データベースでの作業を実行するリモート・プロシージャ・コール (RPC) を実行できます。

プログラムがリモート・プロシージャをコールすると、ローカル・サーバーは、コールされているリモート・サーバーにすべてのプロシージャ・パラメータを渡します。たとえば、次の PL/SQL プログラム・ユニットは、リモート・データベース SALES にあるパッケージ・プロシージャ DEL_EMP をコールして、パラメータ 1257 を渡します。

```
BEGIN
  emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;
```

RPC が成功するには、コール先プロシージャがリモート・サイトに存在する必要があります。

分散データベース・システム用のパッケージやプロシージャを開発する開発者は、リモート位置でプログラム・ユニットが実行する必要がある作業や、結果をコール元のアプリケーションに戻す方法を理解した上でコーディングする必要があります。

分散問合せの最適化

「分散問合せの最適化」は、Oracle8iのデフォルト機能であり、分散 SQL 文で参照されるリモート表からトランザクションがデータを取り出す場合に、サイト間で必要となるデータ転送の量を低減します。

分散問合せの最適化では、Oracle のコストベース最適化を使用して、リモート表から必要なデータのみを抽出する SQL 式を検索または生成し、そのデータをリモート・サイト（または、場合によってはローカル・サイト）で処理し、結果を最終処理のためにローカル・サイトに送信します。この操作により、すべての表データを処理のためにローカル・サイトに転送する所要時間に比べて、必要なデータ転送量が減少します。

DRIVING_SITE、NO_MERGE および INDEX など、コストベースの各種のオプティマイザ・ヒントを使用すると、Oracle でデータが処理される場所とデータへのアクセス方法を制御できます。

関連項目： コストベース最適化の詳細は、『Oracle8i 分散システム』を参照してください。

各国語サポート

Oracle では、クライアント、Oracle サーバーおよび Oracle 以外のサーバーで異なるキャラクタ・セットを使用する環境がサポートされています。Oracle8i では、異種環境のための NCHAR サポートが提供されます。様々な NLS および HS パラメータを設定し、異なるキャラクタ・セット間のデータ変換を制御できます。

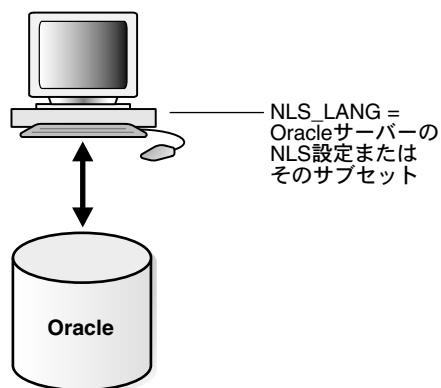
キャラクタ・セットは、次の NLS パラメータと HS パラメータで定義されます。

パラメータ	環境	定義対象
NLS_LANG	クライアント / サーバー	クライアント
NLS_LANGUAGE	クライアント / サーバー 同種分散 異種分散	Oracle Server
NLS_CHARACTERSET		
NLS_TERRITORY		
HS_LANGUAGE	異種分散	Oracle 以外のサーバー Transparent Gateway
NLS_NCHAR	異種分散	Oracle Server Transparent Gateway
HS_NLS_NCHAR		

クライアント / サーバー環境

クライアント / サーバー環境では、[図 30-6](#) のように、クライアントのキャラクタ・セットを、Oracle サーバーと同じキャラクタ・セット、またはそのサブセットに設定します。

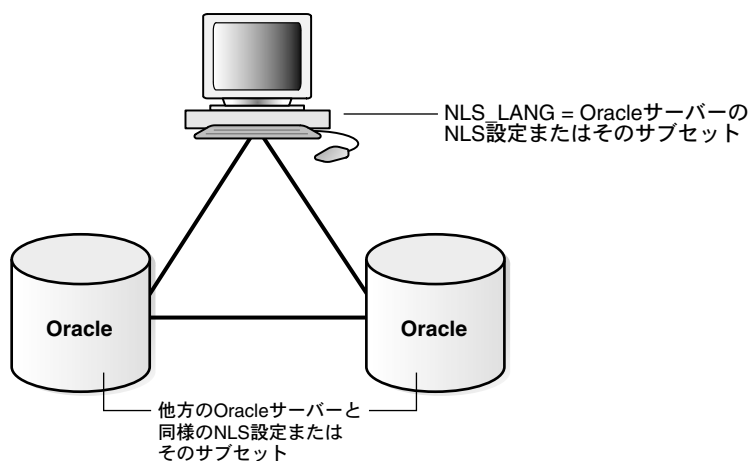
図 30-6 クライアント / サーバー環境での NLS 設定



同種分散環境

同種環境では、[図 30-7](#) のように、クライアントとサーバーのキャラクタ・セットを、メイン・サーバーと同じキャラクタ・セット、またはそのサブセットに設定する必要があります。

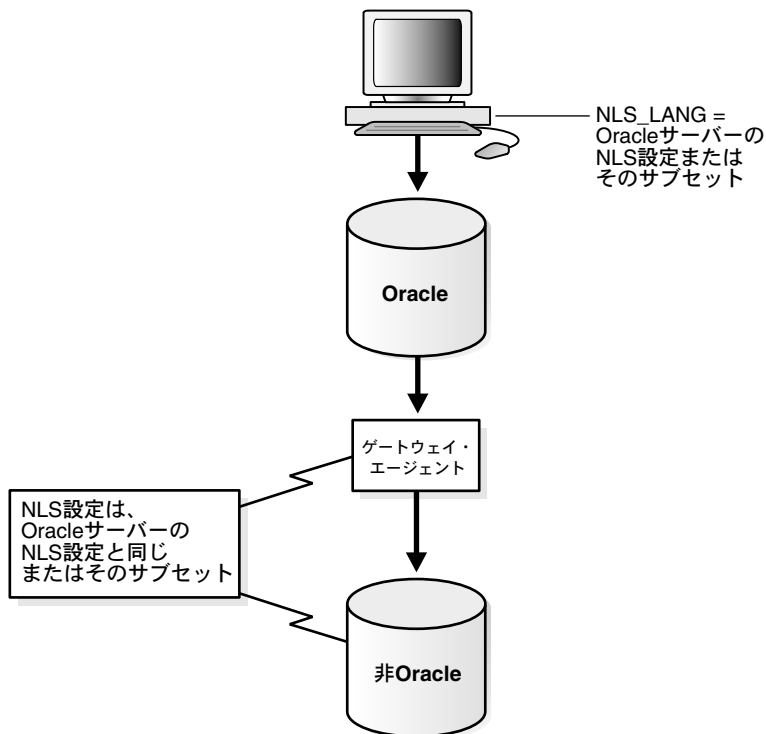
図 30-7 同種環境での NLS 設定



異種分散環境

異種環境では、クライアント、Transparent Gateway および Oracle 以外のデータ・ソースの NLS 設定は、[図 30-8](#) のように Oracle サーバーと同じ NLS キャラクタ・セット、またはそのサブセットにする必要があります。Transparent Gateway には、全面的な NLS サポート機能があります。

図 30-8 異種環境での NLS 設定



異種環境では、HS テクノロジが組み込まれている Transparent Gateway のみが、完全な NCHAR 機能をサポートします。特定の Transparent Gateway で NCHAR がサポートされるかどうかは、そのターゲットとなる Oracle 以外のデータ・ソースによって決まります。特定の Transparent Gateway で NCHAR サポートが処理される方法の詳細は、該当する Transparent Gateway のマニュアルを参照してください。

関連項目： 各国語サポート機能の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

レプリケーション

この章では、Oracle のレプリケーション機能に関連した基本的な概念と用語について説明します。この章の内容は、次のとおりです。

- [レプリケーションの概要](#)
- [レプリケーションを使用するアプリケーション](#)
- [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](#)
- [レプリケーション環境のタイプ](#)
- [レプリケーション環境の管理ツール](#)
- [レプリケーションの競合](#)
- [マルチマスター・レプリケーションに関するその他のオプション](#)

注意： Trusted Oracle を使用している場合に、Trusted Oracle 環境でレプリケーション機能を使用する方法の詳細は、Oracle セキュリティ関連製品のマニュアルを参照してください。

レプリケーションの概要

「レプリケーション」とは、分散データベース・システムを構成する複数のデータベースの中に、表などのデータベース・オブジェクトをコピーして維持するプロセスのことです。あるサイトで適用された変更は、ローカルに獲得されて格納されてから、各リモート・ロケーションに転送され、適用されます。Oracle レプリケーションは、Oracle Server に完全に統合された機能であり、別個のサーバーではありません。

レプリケーションでは、分散データベース・テクノロジーを使用して複数のサイト間でデータが共有されますが、レプリケート・データベースと分散データベースは同じものではありません。分散データベースでは、データを多数の場所で使用できますが、特定の表は1箇所に格納されます。たとえば、EMP 表を、DB2 および DB3 データベースも格納されている分散データベース・システムの DB1 データベースに格納できます。レプリケーションは、同じデータを複数の場所で使用可能にすることを意味します。たとえば、EMP 表を DB1、DB2 および DB3 で使用可能にすることができます。

一般的に、レプリケーションは次のような目的で使用されます。

可用性

レプリケーションにより、データ・アクセスの代替手段が提供されるため、アプリケーションの可用性が改善されます。あるサイトが使用不能になっても、ユーザーは引き続き残りのサイトを問い合わせで更新できます。つまり、レプリケーションは優れたフェイルオーバー保護機能を提供します。

パフォーマンス

レプリケーションにより、複数サイト間でアクティビティが均衡化されるため、共有データへの高速なローカル・アクセスが可能になります。ユーザーが特定のサーバーにアクセスしている間に、他のユーザーは他のサーバーにアクセスできるため、すべてのサーバーの負荷が減少します。また、ユーザーはアクセス・コストが最も低いレプリケーション・サイトからデータにアクセスできます。通常、これは、地理的に最も近距離にあるサイトです。

モバイル・コンピューティング

「スナップショット」には、特定の時点からのターゲット・マスター表の完全または部分的なコピー（レプリカ）が含まれています。スナップショットにより、ユーザーは中央のデータベース・サーバーに接続していない間も、そのデータベースのサブセットで作業できます。後で接続が確立された時点で、必要に応じてスナップショットを同期化（リフレッシュ）できます。ユーザーがスナップショットをリフレッシュすると、そのユーザーが行ったすべての変更で中央のデータベースが更新され、切断中に発生した変更内容を取得できます。

ネットワーク負荷の低減

レプリケーションを使用すると、データを複数の地域に分散できます。これにより、アプリケーションは単一の中央サーバーにアクセスするかわりに、各地のサーバーにアクセスできます。この構成では、ネットワーク負荷を大幅に低減できます。

大量配布

企業にとって、データを使用および操作する機能が必要な多数のアプリケーションを配布する必要性は高まる一方です。Oracle レプリケーションでは、配置テンプレートを使用して複数のスナップショット環境を迅速に作成できます。変数を使用して、各スナップショット環境を個々のニーズにあわせてカスタマイズできます。たとえば、営業活動の自動化に配置テンプレートを使用できます。この場合、テンプレートには、各種の営業地域と営業担当に関する変数が含まれます。

レプリケーションを使用するアプリケーション

レプリケーションは、異なる要件を持つことの多い多様なアプリケーションをサポートしています。アプリケーションによっては、比較的自律型の個々のスナップショット・サイトに対応できます。たとえば、営業自動化、フィールド・サービス、小売および他の大量配布アプリケーションでは、通常、中央のデータベース・システムと多数の小規模なリモート・サイトの間で、データを定期的に同期化する必要がありますが、これらのリモート・サイトは中央のデータベースから切断されることがよくあります。営業担当は、中央のデータベースに接続されているかどうかを問わず、トランザクションを完了できることが必要です。この場合は、リモート・サイトを自律型にする必要があります。

これに対して、コール・センターやインターネット・システムなどのアプリケーションでは、提供するサービスを常に使用可能で等価の状態に保つために、複数のサーバー上にあるデータを絶えず、ほぼ瞬時に同期化する必要があります。たとえば、インターネット上の小売業の web サイトでは、どの顧客も各サイトのオンライン・カタログで同じ情報を目にすることを保証する必要があります。この場合は、サイトの自律性よりもデータ整合性が重要です。

Oracle レプリケーションは、この 2 つのタイプのアプリケーションのみでなく、両方をあわせ持つアプリケーションにも使用できます。つまり、1 つの Oracle レプリケーション環境で、大量配布とサーバー間レプリケーションをサポートできるため、1 つの一貫した環境への統合が可能になります。たとえば、このような環境では、営業自動化とカスタマ・サービスのコール・センターがデータを共有できます。

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

この後の項では、レプリケーション・オブジェクト、レプリケーション・グループおよびレプリケーション・サイトなど、レプリケーション・システムの基本的なコンポーネントについて説明します。

レプリケーション・オブジェクト

「レプリケーション・オブジェクト」は、分散データベース・システムの中で複数のサーバー上に存在するデータベース・オブジェクトです。レプリケーション環境では、あるサイトでレプリケーション・オブジェクトを更新すると、その更新内容は他のすべてのサイトにあるコピーに適用されます。Oracle レプリケーションにより、次のタイプのオブジェクトをレプリケートできます。

- 表
- 索引
- ビュー
- パッケージとパッケージ本体
- プロシージャとファンクション
- トリガー
- 順序
- シノニム

レプリケーション・グループ

レプリケーション環境の場合、Oracle は、「レプリケーション・グループ」を使用してレプリケーション・オブジェクトを管理します。レプリケーション・グループとは、論理的に関連したレプリケーション・オブジェクトのコレクションです。レプリケーション・グループ内のオブジェクトは、まとめて管理されます。

レプリケーション・グループ内で相互に関連したデータベース・オブジェクトを編成すると、多数のオブジェクトを管理しやすくなります。多くの場合、特定のデータベース・アプリケーションのサポートに必要なスキーマ・オブジェクトを編成するために、レプリケーション・グループを作成して使用します。ただし、レプリケーション・グループとスキーマが相互に対応付けられている必要はありません。レプリケーション・グループに複数のスキーマからのオブジェクトを含めたり、1つのスキーマが複数のレプリケーション・グループ内のオブジェクトを持つことができます。ただし、レプリケーション・オブジェクトは1つのレプリケーション・グループのメンバーにしかありません。

レプリケーション・サイト

レプリケーション・グループは、複数の「レプリケーション・サイト」に存在できます。レプリケーション環境では、「マスター・サイト」と「スナップショット・サイト」という2

つの基本的な型のサイトがサポートされます。1つのサイトが、マスター・サイトとスナップショット・サイトの両方を同時に兼ねることができます。

マスター・サイトとスナップショット・サイトの相違点は、次のとおりです。

- マスター・サイトにあるレプリケーション・グループを、特に「マスター・グループ」と呼びます。スナップショット・サイトにあるレプリケーション・グループを、特に「スナップショット・グループ」と呼びます。また、各マスター・グループには「マスター定義サイト」が1つずつあります。レプリケーション・グループのマスター定義サイトは、レプリケーション・グループおよびそのグループ内のオブジェクトを管理する制御センターとなるマスター・サイトです。
- マスター・サイトでは、レプリケーション・グループ内のすべてのオブジェクトの完全コピーがメンテナンスされますが、スナップショット・サイトにあるスナップショットには、マスター・グループ内のすべての表データまたはそのサブセットを含めることができます。たとえば、SCOTT_MG マスター・グループに表 EMP および DEPT が含まれている場合、すべてのマスター・サイトでは EMP と DEPT の完全コピーをメンテナンスする必要があります。ただし、あるスナップショット・サイトには EMP 表のスナップショットのみを格納し、別のスナップショット・サイトには EMP 表と DEPT 表のスナップショットを格納できます。
- マルチマスター・レプリケーション環境にあるすべてのマスター・サイトは、相互に直接通信して、レプリケーション・グループ内のデータとスキーマの変更を継続的に伝播させます。スナップショット・サイトには、特定時点からの表データのイメージ、つまりスナップショットが含まれます。通常、スナップショットはマスター・サイトとの同期をとるために定期的にリフレッシュされます。複数のスナップショットを編成して「リフレッシュ・グループ」にまとめることができます。リフレッシュ・グループ内のスナップショットは、1つ以上のスナップショット・グループに属することができます。また、そのグループに含まれるすべてのスナップショットのデータが、トランザクション内で一貫した同一時点に対応するように、同時にリフレッシュされます。

レプリケーション環境のタイプ

Oracle レプリケーションは、次のタイプのレプリケーション環境をサポートしています。

- [マルチマスター・レプリケーション](#)
- [スナップショット・レプリケーション](#)
- [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](#)

マルチマスター・レプリケーション

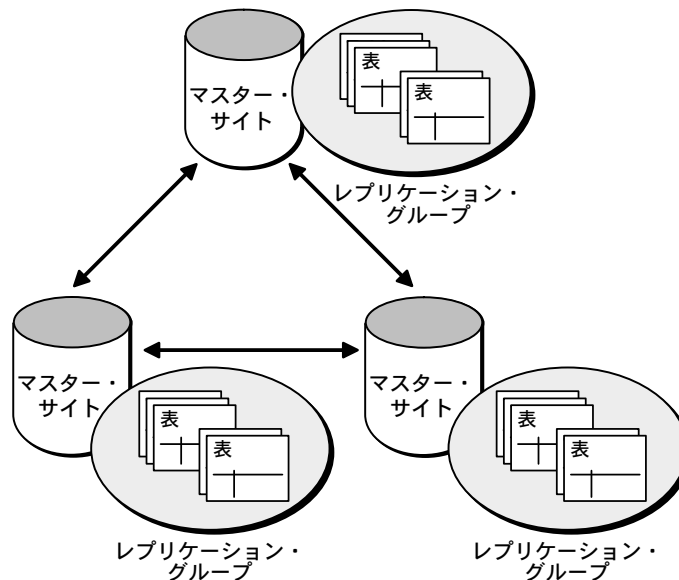
「マルチマスター・レプリケーション」(peer-to-peer または *n*-way レプリケーションとも呼びます) では、同等のピアとして機能する複数のサイトが、レプリケートされたデータベース・オブジェクトのグループを管理できます。マルチマスター・レプリケーション環境では、各サイトがマスター・サイトです。

マルチマスター構成では、アプリケーションは任意のサイトにあるレプリケート表を更新できます。マルチマスター環境でマスター・サイトとして稼働する Oracle データベース・サーバーは、すべての表レプリカのデータを自動的に集中させることによって、グローバルなトランザクションの一貫性とデータの整合性を保証します。

非同期レプリケーションは、マルチマスター・レプリケーションを実装するための最も一般的な方法です。もう 1 つの方法は同期レプリケーションとプロシージャ型レプリケーションを伴いますが、これについては後述します。非同期レプリケーションを使用すると、表の更新内容は変更が発生したマスター・サイトの遅延トランザクション・キューに格納されます。これらの変更を「遅延トランザクション」と呼びます。遅延トランザクションは、他に参加しているマスター・サイトに定期的にプッシュ（伝播）されます。この時間間隔は制御できます。

非同期レプリケーションを使用すると、同じ行の値がほぼ同時に 2 つの異なるマスター・サイトで更新されることがあるため、競合を生じる可能性があります。ただし、その場合は競合を回避する手法を使用でき、競合が発生しても、Oracle には解消のためのメカニズムが組み込まれています。

図 31-1 マルチマスター・レプリケーション



マスター・グループの休止

マスター・グループに対して特定の管理作業を実施できるように、ときにはマスター・グループに対するすべてのレプリケーション・アクティビティを停止する必要があります。たとえば、マスター・グループ内の任意の表に対してデータ定義言語（DDL）文を発行するには、そのグループに対するすべてのレプリケーション・アクティビティを停止する必要があります。このように、マスター・グループに対するすべてのレプリケーション・アクティビティを停止することを、グループの「休止」と呼びます。マスター・グループが休止されると、ユーザーはそのグループ内のオブジェクトにはデータ操作言語（DML）文を実行できなくなります。

スナップショット・レプリケーション

スナップショットには、特定の時点からのターゲット・マスター表の完全または部分的なコピーが含まれています。スナップショットは、読取り専用でも更新可能でもかまいません。

すべてのスナップショットに、次の利点があります。

- ローカル・アクセスを提供して応答時間と可用性を改善します。
- ユーザーはかわりにローカル・スナップショットを問合せできるため、マスター・サイトからの問合せをオフロードします。
- ターゲット・マスター表のデータ・セットのうち、選択したサブセットのみをレプリケートできるため、データ・セキュリティが向上します。

読取り専用スナップショット

基本構成では、スナップショットは、マスター・サイトから発行される表データへの読取り専用アクセスを提供します。アプリケーションは、ネットワークが使用できるかどうかには関係なく、読取り専用スナップショットのデータを問い合わせることができ、ネットワーク・アクセスを回避します。ただし、更新操作を実行するには、システム全体のどのアプリケーションも、マスター・サイトのデータにアクセスする必要があります。[図 31-2](#) に、基本的な読取り専用レプリケーションを示します。読取り専用スナップショットのマスター表は、マスター・グループに属す必要はありません。

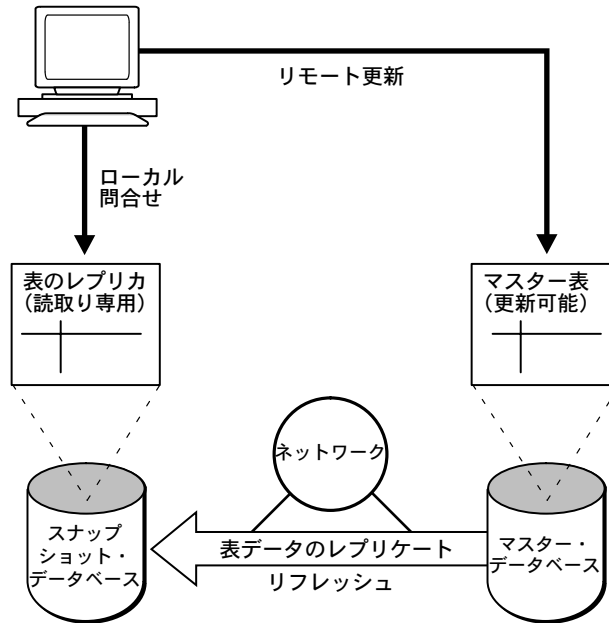
読取り専用スナップショットには、次の利点があります。

- 更新できないため、競合を生じる可能性がありません。
- 複雑なスナップショットをサポートします。たとえば、集合演算や CONNECT BY 句を含むスナップショットを使用できます。

関連項目： 複雑なスナップショットの詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

図 31-2 読取り専用スナップショットのレプリケーション

クライアント・アプリケーション



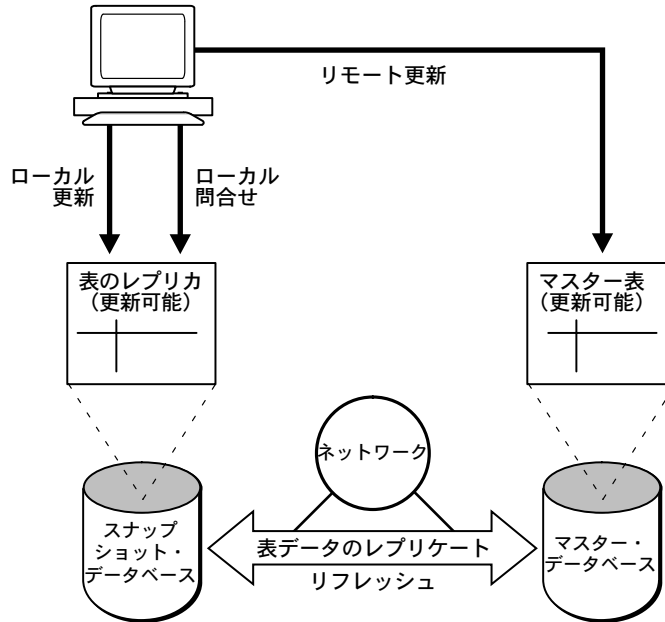
更新可能スナップショット

より高度な構成では、ユーザーがスナップショット上で行の挿入、更新および削除操作を実行して、これらの操作をターゲット・マスター表に対して実行できるように、「更新可能スナップショット」を作成できます。更新可能スナップショットには、ターゲット・マスター表のデータ・セットのサブセットのみを含めることもできます。図 31-3 に、更新可能スナップショットを使用するレプリケーション環境を示します。

更新可能スナップショットは、レプリケーションのサポート用にセットアップされたマスター・サイトにある表に基づいています。実際には、更新可能スナップショットは、マスター・サイトにあるマスター・グループに基づくスナップショット・グループの一部である必要があります。

図 31-3 更新可能スナップショットのレプリケーション

クライアント・アプリケーション



更新可能スナップショットには次のプロパティがあります。

- 常に単一の表に基づいています。
- 増分（すなわち「高速な」）リフレッシュができます。
- Oracle は、更新可能スナップショットに加えられた変更を、そのスナップショットのリモート・マスター表に伝播させます。必要であれば、更新は他のすべてのマスター・サイトにあるマスター表にもカスケードします。
- Oracle は、読取り専用スナップショットをリフレッシュする場合と同様に、更新可能スナップショットをリフレッシュ・グループの一部としてリフレッシュできます。

更新可能スナップショットには、次の利点があります。

- ユーザーは、マスター・サイトから切断されているときにも、ローカルにレプリケートされたデータセットを問い合わせて更新できます。
- データの更新がサポートされるだけでなく、マルチマスター・レプリケーションに比べて必要なリソースが減少します。たとえば、スナップショットは Oracle8i Lite のデータベースに格納できるため、スナップショット・クライアントに必要なディスク領域およびメモリー量は、Oracle8i サーバーに比べて大幅に少なくなります。

スナップショットのリフレッシュ

スナップショットとそのマスター表との一貫性を保つために、スナップショットを定期的のリフレッシュする必要があります。Oracle では、次の 3 つの方法でスナップショットをリフレッシュできます。

- 「高速リフレッシュ」では、スナップショット・ログを使用して、前回のリフレッシュ以降に変更があった行のみが更新されます。
- 「COMPLETE リフレッシュ」では、スナップショット全体が更新されます。
- 「強制リフレッシュ」では、可能な場合は高速リフレッシュが実行され、高速リフレッシュを実行できない場合は COMPLETE リフレッシュが実行されます。

スナップショット相互がトランザクションで一貫していることが重要な場合は、「リフレッシュ・グループ」として編成できます。リフレッシュ・グループをリフレッシュすると、そのグループ内のすべてのスナップショットに含まれるデータが、トランザクションの一貫した同一時点に対応していることを保証できます。リフレッシュ・グループ内のスナップショットを個別にリフレッシュすることもできますが、その場合、そのグループ内の他のスナップショットはリフレッシュされないため、リフレッシュ・グループの利点は得られません。

スナップショット・ログ

「スナップショット・ログ」は、マスター表に加えられたすべての DML 変更が記録される表です。スナップショット・ログはマスター表と 1 対 1 の関係で対応付けられ、マスターからリフレッシュされるスナップショット数に関係なく、各マスター表はスナップショット・ログを 1 つのみ持ちます。スナップショットの高速リフレッシュを実行できるのは、そのスナップショットのマスター表のスナップショット・ログが記録されている場合のみです。スナップショットを高速リフレッシュすると、そのスナップショットには、対応付けられたスナップショット・ログ内で、前回のリフレッシュ以降に表示されたエントリが適用されます。

配置テンプレート

「配置テンプレート」により、多数のリモート・スナップショット・サイトの配置およびメンテナンス作業が簡素化されます。配置テンプレートを使用すると、マスター・サイトでスナップショット定義のコレクションを定義し、定義内のパラメータを使用して、個々のユーザーまたはユーザーのタイプにあわせてスナップショットをカスタマイズできます。

たとえば、あるテンプレートを営業用に作成し、別のテンプレートをフィールド・サービス担当用に作成するとします。この場合、パラメータ値には営業地域やカスタマ・サポート・レベルを指定できます。リモート・ユーザーがマスター・サイトに接続すると、使用可能なテンプレートのリストが表示されます。ユーザーがテンプレートをインスタンスiertすると、適切なスナップショットが作成され、リモート・サイトに移入されます。適切なパラメータ値は、リモート・ユーザーが指定するか、マスター・サイトでメンテナンスされている表から取り込むことができます。

オンライン表領域とオフライン表領域 ユーザーがスナップショット・サイトでテンプレートをインスタンスiertすると、そこでオブジェクトの DDL (CREATE SNAPSHOT... や CREATE TABLE... など) が実行され、適切なスキーマ・オブジェクトが作成され、適切なデータとともにオブジェクトが移入されます。

ネットワークを介してマスター・サイトに接続中 (オンライン・インスタンスiエーション) でも、マスター・サイトから切断されている間 (オフライン・インスタンスiエーション) でも、テンプレートはインスタンスiertできます。

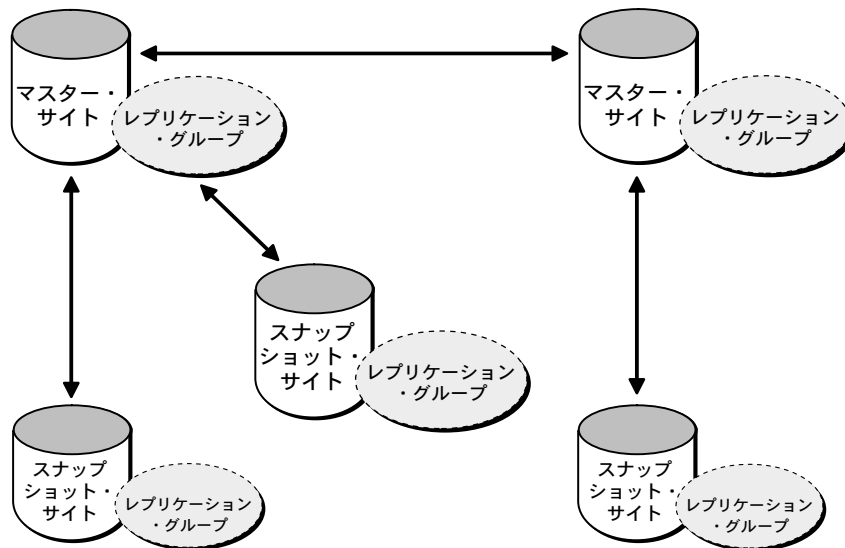
通常、オフライン・インスタンスiエーションは、ピーク使用時間中のサーバーの負荷を低減し、リモート接続時間を短縮するために使用されます。テンプレートをオフラインでインスタンスiertするには、そのテンプレートと必要なデータを、テープや CD-ROM など、なんらかのタイプの記憶メディアにパッケージ化します。これにより、マスター・サイトからデータをプルするかわりに、テンプレートとデータが格納されている記憶メディアからプルすることになります。

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

マルチマスター・レプリケーションとスナップショットをハイブリッド (複合) 構成で組み合わせると、様々なアプリケーションの要件を満たすことができます。複合構成では、任意の数のマスター・サイトと、各マスターごとに複数のスナップショット・サイトが可能です。

たとえば、[図 31-4](#) のように、2 つのマスター間のマルチマスター (または *n*-way) レプリケーションでは、2 つの地域をサポートするデータベース間の全表レプリケーションをサポートできます。表全体または表のサブセットを各地域のサイトにレプリケートするために、マスターに対してスナップショットを定義できます。

図 31-4 ハイブリッド構成



スナップショットとレプリケートされたマスターの主な違いは、次のとおりです。

- レプリケートされたマスターはレプリケートされる表全体のデータを含む必要がありますが、スナップショットではマスター表のサブセットをレプリケートできます。
- マルチマスター・レプリケーションを使用すると、各トランザクションごとの変更を、その発生時にレプリケートできます。スナップショットのリフレッシュは集合指向であり、複数のトランザクションの変更は、より効率的なバッチ指向の操作で、より少ない頻度で伝播します。
- 同じデータの複数のコピーに変更が加えられたために競合が発生する場合、マスター・サイトはその競合を検出して解消します。

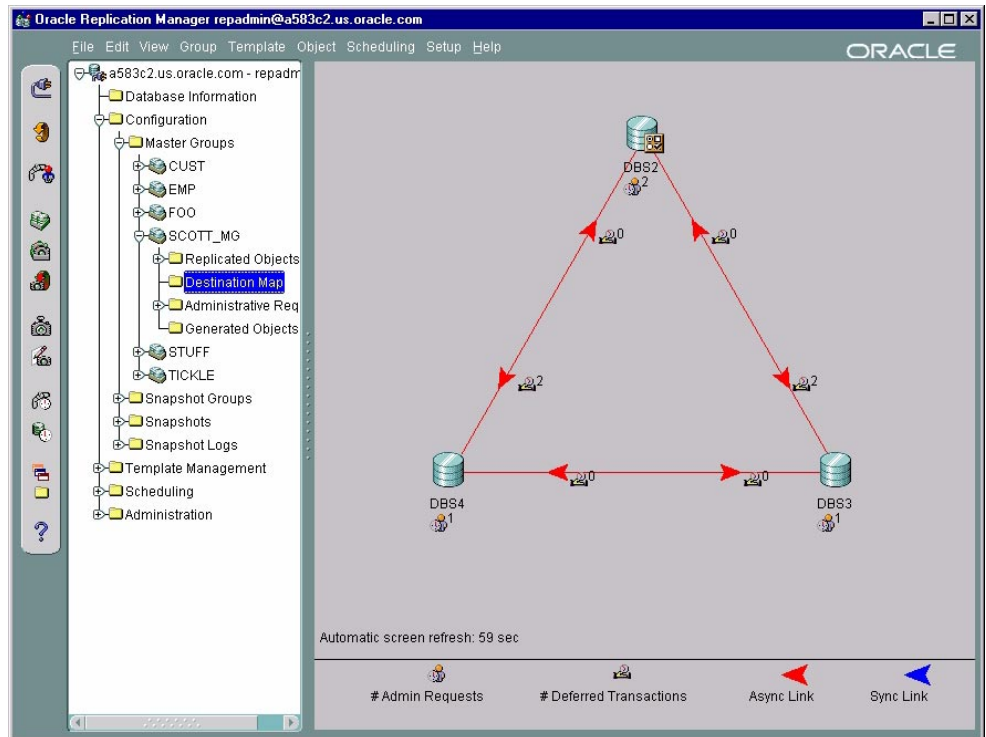
レプリケーション環境の管理ツール

レプリケーション環境を管理し、監視するために、複数のツールを使用できます。Oracle Replication Manager は、環境を容易に管理できるように強力な GUI インタフェースを提供し、レプリケーション・マネージメント API は、レプリケーション管理用にカスタマイズされたスクリプトを作成できるように、標準的なアプリケーション・プログラミング・インタフェース (API) を提供します。また、レプリケーション・カタログにより、レプリケート環境に関する情報を常に把握できます。

Oracle Replication Manager

マルチマスター・レプリケーションとスナップショット・レプリケーションの両方をサポートしているレプリケーション環境の場合、構成および管理作業が煩雑になる可能性があります。このようなレプリケーション環境の管理を支援するため、Oracle には、洗練された管理ツールである Oracle Replication Manager が用意されています。Oracle Replication Manager の使用方法と使用例は、他の項を参照してください。ただし、Replication Manager に関するマニュアルは、主として Replication Manager オンライン・ヘルプの形式で提供されます。

図 31-5 Replication Manager



関連項目： Replication Manager の概要は『Oracle8i レプリケーション・ガイド』、Replication Manager の使用手順の詳細は Replication Manager オンライン・ヘルプを参照してください。

レプリケーション・マネージメント API

レプリケーション・マネージメント・アプリケーション・プログラミング・インタフェース (API) は、Oracle レプリケーション環境の構成に使用できる、プロシージャとファンクションをカプセル化した PL/SQL パッケージの集合です。この API は、Replication Manager のコマンドライン代替機能であり、Replication Manager では、作業を実行するためにレプリケーション・マネージメント API のプロシージャとファンクションを使用します。たとえば、Replication Manager を使用して新しいマスター・グループを作成する場合、作業を完了させるため Replication Manager は、DBMS_REPCAT.CREATE_MASTER_REPGROUP プロシージャをコールします。レプリケーション・マネージメント API により、カスタム・スクリプトを作成してレプリケーション環境を管理する作業が容易になります。

関連項目： レプリケーション・マネージメント API の使用方法の詳細は、『Oracle8i レプリケーション・マネージメント API リファレンス』を参照してください。

レプリケーション・カタログ

レプリケーション・レプリケーション環境にあるすべてのマスター・サイトとスナップショット・サイトには、「レプリケーション・カタログ」があります。サイトのレプリケーション・カタログは、そのサイトにあるレプリケーション・オブジェクトとレプリケーション・グループに関する管理情報をメンテナンスする、データ・ディクショナリ表およびビューの固有の集合です。レプリケーション環境に関係するすべてのサーバーは、そのレプリケーション・カタログ内の情報を使用して、レプリケーション・グループ内のオブジェクトのレプリケーションを自動化できます。

関連項目： レプリケーション・カタログの詳細は、『Oracle8i レプリケーション・マネージメント API リファレンス』を参照してください。

分散スキーマ管理

レプリケーション環境では、すべての DDL 文は Replication Manager または DBMS_REPCAT パッケージを使用して発行する必要があります。これらのインタフェースを使用すると、すべての DDL 文はレプリケーション環境に参加するすべてのサイトにレプリケートされます。

注意： SQL*Plus などのツールを使用して直接発行した DDL 文は、他のサイトにレプリケートされません。

レプリケーションの競合

非同期のマルチマスター環境と更新可能スナップショット・レプリケーション環境は、異なるサイトから発行された2つのトランザクションが、ほぼ同時に同じ行を更新するときなどに発生する可能性があるレプリケーションの競合を取り扱う必要があります。データ競合が発生した場合は、それがビジネス・ルールに従って解消されることと、すべてのサイトでデータが収束されることを保証するメカニズムが必要です。

Oracle レプリケーションは、レプリケート環境で発生する競合をログイングするだけでなく、多様な競合を解消する方法が組み込まれています。これにより、ビジネス・ルールに従って競合を解消するシステムを、データベースにあわせて定義できます。Oracle の組込みの競合を解消する方法では解消できない固有の状況が発生した場合は、独自の競合ルーチンを作成して使用することもできます。

関連項目：

- データの競合を回避するデータベースの設計方法と、この種の競合が発生した場合にそれを解消する競合解消ルーチンの作成方法は、『Oracle8i レプリケーション・ガイド』を参照してください。
- Replication Manager を使用して競合解消方法を構成する手順は、Replication Manager オンライン・ヘルプを参照してください。
- レプリケーション・マネージメント API を使用して競合解消ルーチンを作成する方法の詳細は、『Oracle8i レプリケーション・マネージメント API リファレンス』を参照してください。

マルチマスター・レプリケーションに関するその他のオプション

非同期レプリケーションは、マルチマスター・レプリケーションを実装するための最も一般的な方法です。ただし、その他にも同期レプリケーションおよびプロシージャ型レプリケーションという2つのオプションがあります。

同期レプリケーション

マルチマスター・レプリケーション環境では、非同期または同期レプリケーションを使用してデータをコピーできます。非同期レプリケーションでは、あるマスター・サイトで行われた変更は、参加している他のすべてのマスター・サイトでも後から発生します。同期レプリケーションでは、あるマスター・サイトで行われた変更は、参加している他のすべてのマスター・サイトで即時に発生します。

同期レプリケーションを使用している場合、表を更新すると、参加しているすべてのマスター・サイトでも、その更新が即時にレプリケートされます。実際には、各トランザクションにすべてのマスター・サイトが含まれることになります。したがって、あるマスター・サイトがなんらかの理由でトランザクションを処理できないと、そのトランザクションはすべてのマスター・サイトでロールバックされます。

同期レプリケーションの使用時に競合が発生する可能性は回避されますが、操作を円滑に進行させるにはきわめて安定的な環境が必要です。あるマスター・サイトへの通信がネットワーク障害などによって不可能になると、通信が再び確立されるまでトランザクションを完了できません。

プロシージャ型レプリケーション

バッチ処理アプリケーションでは、1つのトランザクションの中で大量のデータを変更する場合があります。その場合、典型的な行レベルのレプリケーションでは、ネットワークに多数のデータ変更による負荷がかかる可能性があります。このような問題を回避するため、レプリケーション環境で動作するバッチ処理アプリケーションでは、Oracle の「プロシージャ型レプリケーション」を使用して、データ・レプリカを収集するための簡単なストアド・プロシージャ・コールをレプリケートできます。プロシージャ型レプリケーションでは、アプリケーションが表の更新に使用するストアド・プロシージャのコールのみがレプリケートされます。データ変更自体はレプリケートされません。

プロシージャ型レプリケーションを使用するには、システムのデータを修正するパッケージをすべてのサイトにレプリケートする必要があります。パッケージをレプリケートした場合は、各サイトごとに、そのパッケージのための「ラッパー」を生成する必要があります。アプリケーションがローカル・サイトにあるパッケージ・プロシージャをコールしてデータを修正すると、そのラッパーによって、レプリケート環境内の他のすべてのサイトでも最終的に同じパッケージ・プロシージャがコールされることが保証されます。プロシージャ型レプリケーションは、同期または非同期のいずれでも実行できます。

競合の検出とプロシージャ型レプリケーション

レプリケーション・システムがプロシージャ型レプリケーションを使用してデータをレプリケートする場合、データをレプリケートするプロシージャは、レプリケートされたデータの整合性を保証する必要があります。つまり、そのようなプロシージャは、レプリケーションの競合を回避するか、あるいはそれを検出して解消するように設計されている必要があります。そのため、プロシージャ型レプリケーションが使用されるのは、通常、データベースが大規模なバッチ処理のみで変更される場合です。そのような状況では、多数のトランザクションが同じデータを要求することがないため、レプリケーションの競合はほとんど発生しません。

関連項目：『Oracle8i レプリケーション・ガイド』

第 X 部

付録

第 X 部に含まれる付録は、次のとおりです。

- 付録 A「オペレーティング・システム固有の情報」

オペレーティング・システム固有の情報

このマニュアルでは、特定のオペレーティング・システムで Oracle を使用する際の詳細情報について、他の Oracle マニュアルに言及していることがあります。正式名称は実際のオペレーティング・システムによって異なりますが、これらの Oracle マニュアルを『インストールおよび構成ガイド』と呼びます。

この付録では、このマニュアル内でオペレーティング・システム別の Oracle のマニュアルを参照しているすべての箇所と、オペレーティング・システム（OS）によって異なる初期化パラメータのリストを記載します。Oracle を複数のオペレーティング・システムで使用している場合は、それらのオペレーティング・システム間でアプリケーションの移植性を確保するために、この付録の情報が参考になります。

次のリストは、このマニュアルのオペレーティング・システム別のトピックを、ページ順に示したものです。

- データ・ファイル、ファイル・ヘッダーのサイズ： 3-16 ページの「[データ・ファイル](#)」
- データ・ブロック、サイズ： 4-3 ページの「[データ・ブロック](#)」
- ロールバック・セグメント、トランザクションあたりの数： 4-20 ページの「[トランザクションとロールバック・セグメント](#)」
- 管理者権限、前提条件： 5-3 ページの「[管理者権限での接続](#)」
- DBA の認証： 5-3 ページの「[管理者権限での接続](#)」 および 26-13 ページの「[データベース管理者の認証](#)」
- Net8、Net8 ソフトウェアに含まれるドライバ： 6-5 ページの「[Net8 の機能](#)」
- プログラム・グローバル領域 (PGA)： 7-16 ページの「[PGA のサイズ](#)」
- ソフトウェア・コード領域、共有または非共有： 7-17 ページの「[ソフトウェア・コード領域](#)」
- Oracle の構成： 8-2 ページの「[プロセスのタイプ](#)」
- バックグラウンド・プロセス、作成： 8-5 ページの「[バックグラウンド・プロセス](#)」
- バックグラウンド・プロセス、DBW n プロセス： 8-8 ページの「[データベース・ライター \(DBW \$n\$ \)](#)」
- バックグラウンド・プロセス、ARC n ： 8-12 ページの「[アーカイバ \(ARC \$n\$ \) プロセス](#)」
- 専用サーバー、管理操作のための要求： 8-20 ページの「[マルチスレッド・サーバーの限定的運用](#)」
- クライアント / サーバー通信： 8-22 ページの「[専用サーバー構成](#)」
- Net8、ネットワーク・ドライバの選択とインストール： 8-26 ページの「[プログラム・インタフェース・ドライバ](#)」
- 通信ソフトウェア： 8-26 ページの「[オペレーティング・システムの通信ソフトウェア](#)」
- 索引、索引ブロックのオーバーヘッド： 10-29 ページの「[索引ブロックの形式](#)」
- ユーザーの認証： 26-4 ページの「[オペレーティング・システムによる認証](#)」
- オペレーティング・システムによるロール管理： 27-23 ページの「[オペレーティング・システムとロール](#)」
- 監査： 28-5 ページの「[オペレーティング・システムの監査証跡に必ず記録されるイベント](#)」 および 28-6 ページの「[オペレーティング・システム監査証跡に対する監査](#)」
- パラレル・リカバリと非同期 I/O： 29-11 ページの「[パラレル・リカバリを活用できる状況](#)」

索引

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, 8-15
- 2 フェーズ・コミット
 - 手動による問題解決, 1-34
 - 説明, 1-33, 30-36
 - トランザクションの管理, 16-8
 - トリガー, 19-22
 - パラレル DML, 23-41
- 3 値論理 (TRUE、FALSE、UNKNOWN)
 - NULL のために生じる, 10-8

A

- ADMIN OPTION
 - システム権限, 27-3
 - ロール, 27-20
- AFTER トリガー, 19-10
 - 起動されるタイミング, 19-23
 - 定義, 19-10
- ALL_UPDATABLE_COLUMNS ビュー, 10-16
- ALL_ ビュー, 2-6
- ALTER DATABASE 文, 5-7
- ALTER INDEX 文
 - REBUILD PARTITION 句, 11-59
 - SPLIT PARTITION のためのロギングなしモード, 11-57, 22-7
 - パーティション属性, 11-36
- ALTER SESSION 文, 15-6
 - ENABLE PARALLEL DML 句, 23-39
 - FORCE PARALLEL DDL 句, 23-25, 23-27
 - CREATE TABLE AS SELECT, 23-26, 23-27
 - 索引の作成または再作成, 23-25, 23-28
 - パーティションの移動または分割, 23-25, 23-28
- FORCE PARALLEL DML 句
 - 更新と削除, 23-22, 23-27
 - 挿入, 23-24, 23-27
- SET CONSTRAINTS DEFERRED 句, 25-25
- 動的パラメータ, 5-5
- トランザクション分離レベル, 24-8, 24-32
- ALTER SYSTEM 文, 15-6
- SWITCH LOGFILE 句, 8-12
- 動的パラメータ, 5-5
 - LOG_ARCHIVE_MAX_PROCESSES, 8-13, 29-21
- ALTER TABLESPACE 文
 - READ ONLY 句, 3-11
 - READ WRITE 句, 3-11
 - TEMPORARY または PERMANENT, 3-13
- ALTER TABLE 文
 - CACHE 句, 7-4
 - DEALLOCATE UNUSED 句, 4-14
 - DROP COLUMN 句, 10-7
 - EXCHANGE PARTITION, 11-11
 - MERGE PARTITIONS 句, 11-14
 - MODIFY CONSTRAINT 句, 25-27
 - SPLIT PARTITION のためのロギングなしモード, 11-57, 22-7
 - UNUSED 列, 10-7
 - VALIDATE または NOVALIDATE 制約, 25-26
 - 監査, 28-7
 - 制約を使用禁止または使用可能にする, 25-26
 - トリガー, 19-7
 - パーティション属性, 11-26
 - ハッシュ・パーティションの追加または結合, 11-16

ALTER USER 文
一時セグメント, 4-18
ALTER 文, 15-4
パーティションの監査, 11-61
ANALYZE 文, 15-5
共有プール, 7-11
推定による統計, 21-15
パーティションの統計, 11-12
ヒストグラムの作成, 21-12
ANSI SQL 規格
Oracle が準拠, 1-3
データ型, 12-21
ANSI/ISO SQL 規格, 1-3
コンポジット外部キー, 25-16
データ同時実行性, 24-2
分離レベル, 24-11
AQ_ADMINISTRATOR ロール, 18-9
AQ_TM_PROCESS パラメータ, 18-8, 18-9
ARCHIVELOG モード
アーカイバ (ARCn) プロセス, 1-18, 8-12, 29-19
概要, 1-46
定義, 29-19
データベース全体のバックアップ, 29-25
部分データベース・バックアップ, 1-47, 29-26
ARCn バックグラウンド・プロセス, 1-17, 8-12
「アーカイバ・プロセス」も参照
AUDIT 文, 15-5
ロック, 24-30

B

BEFORE トリガー, 19-10
起動されるタイミング, 19-23
定義, 19-10
BFILE データ型, 12-13
BLOB, 12-12
BOOLEAN データ型, 12-2
BSP, 「ブロック・サーバー・プロセス」を参照
B-tree 索引, 10-29
索引構成表, 10-39
ビットマップ索引と対比, 10-34, 10-35
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, 11-29

C

CACHE 句, 7-4
CASCADE アクション
DELETE 文, 25-17
CHARTOROWID 関数, 12-22
CHAR データ型, 12-5
空白埋め比較方法, 12-5
CHECK 制約, 25-21
1つの列に対する複数の制約, 25-21
一部が NULL の外部キー, 25-16
チェックのメカニズム, 25-23
定義, 25-21
副問合せは指定禁止, 25-21
CKPT バックグラウンド・プロセス, 1-17, 8-11
CLOB データ型, 12-13
COMMENT 文, 15-5
COMMIT 文, 15-5
2 フェーズ・コミット, 16-8, 30-36
DDL による暗黙, 16-2, 16-5
高速コミット, 8-10
トランザクションの終了, 16-2, 16-5
パラレル DML での 2 フェーズ・コミット, 23-41
COMPATIBLE パラメータ
読取り専用表領域, 3-11
CONNECT INTERNAL, 5-3
CONNECT ロール, 27-22
CPU 時間の制限, 26-18
CREATE CLUSTER 文
HASHKEYS 句, 10-56, 10-60
SINGLE TABLE HASHKEYS 句, 10-60
記憶域パラメータ, 4-16
CREATE FUNCTION 文, 17-18
CREATE INDEX 文
一時セグメント, 4-17
記憶域パラメータ, 4-17
パーティション属性, 11-36
パラレル化のルール, 23-25
ロギングなしモード, 11-57, 22-7
CREATE OUTLINE 文, 21-7
CREATE PACKAGE BODY 文, 17-12, 17-18
CREATE PACKAGE 文
パッケージ名, 17-18
例, 17-12, 19-12
ロック, 24-30
CREATE PROCEDURE 文
プロシージャ名, 17-18

- 例, 17-6
- ロック, 24-30
- CREATE SYNONYM 文
 - ロック, 24-30
- CREATE TABLE AS SELECT
 - パラレル化のルール
 - 索引構成表, 23-31
- CREATE TABLESPACE 文
 - TEMPORARY 句, 3-13
- CREATE TABLE 文
 - AS SELECT
 - 意思決定支援システム, 23-32
 - 一時記憶域, 23-34
 - ダイレクト・ロード・インサートと対比, 22-2
 - パラレル化のルール, 23-25
 - 領域の断片化, 23-34
 - ロギングなしモード, 11-57, 22-7
- CACHE 句, 7-4
- 監査, 28-7, 28-9
- 記憶域パラメータ, 4-16
- 制約を使用可能または使用禁止にする, 25-26
- トリガー, 19-7
- パーティション属性, 11-26
- パラレル化, 23-32
 - 索引構成表, 23-31
- 例
 - NESTED TABLE, 13-12
 - オブジェクト表, 13-7, 13-12
 - 列オブジェクト, 13-5
- ロック, 24-30
- CREATE TEMPORARY TABLESPACE 文, 3-13
- CREATE TEMPORARY TABLE 文, 10-11
- CREATE TRIGGER 文
 - コンパイルと格納, 19-26
 - 例, 19-12, 19-15, 19-25
 - ロック, 24-30
- CREATE TYPE 文
 - NESTED TABLE, 13-4, 13-11
 - VARRAY, 13-11
 - オブジェクト型, 13-4
 - オブジェクト・ビュー, 14-3
- CREATE USER 文
 - 一時セグメント, 4-18
- CREATE VIEW 文
 - 例, 19-15
 - オブジェクト・ビュー, 14-3
 - ロック, 24-30

- CREATE_STORED_OUTLINES セッション・パラメータ, 21-7
- CREATE 文, 15-4

D

- DATE データ型, 12-9
 - 深夜, 12-10
 - デフォルト書式の変更, 12-10
 - パーティション化, 11-12, 11-21
 - パーティション・プルーニング, 11-21
 - 日付算術, 12-11
 - 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, 7-13
 - バッファ・キャッシュ, 7-5
- DB_BLOCK_SIZE パラメータ
 - システム・グローバル領域のサイズ, 7-13
 - バッファ・キャッシュ, 7-5
- DB_FILES パラメータ, 7-16
- DB_NAME パラメータ, 29-23
- DB_WRITER_PROCESSES パラメータ, 1-16
- DBA_QUEUE_SCHEDULES ビュー, 18-12
- DBA_SYNONYMS.SQL スクリプト
 - 使用, 2-6
- DBA_UPDATABLE_COLUMNS ビュー, 10-16
- DBA_ ビュー, 2-6
- DBA ロール, 27-22
- DBMS, 1-2
 - 一般要件, 1-49
 - オブジェクト・リレーショナル DBMS, 13-2
- DBMS_AQADM パッケージ, 18-6, 18-9
- DBMS_AQ パッケージ, 18-6
- DBMS_JOB パッケージ, 8-14
 - オラクル社が提供するパッケージ, 17-17
- DBMS_LOCK パッケージ, 24-40
 - オラクル社が提供するパッケージ, 17-17
- DBMS_PCLXUTIL パッケージ, 11-29
- DBMS_RLS パッケージ
 - セキュリティ・ルール, 27-23
 - 定義者権限を使用, 27-9
- DBMS_SQL パッケージ, 15-20
 - DDL 文の解析, 15-20
 - オラクル社が提供するパッケージ, 17-17
- DBMS_STATS パッケージ, 21-13
 - 推定による統計, 21-15
 - パーティションの統計, 11-12

ヒストグラムの作成, 21-12
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, 15-4
「データ定義言語」も参照
DELETE CASCADE 制約, 25-17
DELETE 文, 15-4
外部キー参照, 25-16
データ・ブロック内の領域の解放, 4-9
トリガー, 19-2, 19-7
パラレル DELETE, 23-22
ロギングなしモード, 22-8
LOB, 22-7
DEQUEUE 要求, 18-10
DETERMINISTIC 関数
ファンクション索引, 20-8
Digital の POLYCENTER Manager (NetView 上),
30-33
DISABLED 索引, 20-8
DISABLE 制約, 25-26
DISTRIBUTED_TRANSACTIONS パラメータ, 8-12
DML, 15-4
DML サブパーティション・ロック, 11-45
DML 文
分散トランザクションに許される, 30-34
Dnnn バックグラウンド・プロセス, 1-18, 8-14
「ディスパッチャ・プロセス」も参照
DROP COLUMN 句, 10-7
DROP TABLE 文
監査, 28-7
トリガー, 19-7
DROP 文, 15-4
DSS データベース
索引のパーティション化, 11-36
スコア表, 23-38
ディスクのストライプ化, 23-49
パーティション, 11-6
パフォーマンス, 11-8
パラレル DML, 23-38
DUAL 表, 2-7

E

ENABLE 制約, 25-26
EXCHANGE PARTITION, 11-11
EXECUTE 権限
ユーザー・アクセスの検査, 17-19

EXP_FULL_DATABASE ロール, 27-22
EXPLAIN PLAN 文, 15-4
パーティション・プルーニング, 11-21
Export ユーティリティ, 1-10
統計のコピー, 21-9
パーティションのメンテナンス操作, 11-46
バックアップでの使用方法, 29-27

F

FAST_START_IO_TARGET パラメータ, 29-14
FIPS 規格, 15-6
FOREIGN KEY 制約
親キー値の変更, 25-16
親キー表の更新, 25-16
親表の行の削除, 25-17
制約チェック, 25-23
表の更新, 25-18

G

GLOBAL_NAMES 初期化パラメータ, 30-15
GRANT ANY PRIVILEGE システム権限, 27-3
GRANT 文, 15-5
ロック, 24-30
GROUP BY 句
一時表領域, 3-12

H

HASHKEYS パラメータ, 10-56, 10-60
HEXTORAW 関数, 12-22
HI_SHARED_MEMORY_ADDRESS パラメータ, 7-14
HP の OpenView, 30-33

I

IBM の NetView/6000, 30-33
ILMS, 15-21
IMP_FULL_DATABASE ロール, 27-22
Import ユーティリティ, 1-11
統計のコピー, 21-9
パーティションのメンテナンス操作, 11-46
リカバリでの使用方法, 29-27
INIT.ORA, 「初期化パラメータ・ファイル」を参照
INSERT 文, 15-4
INSERT ... SELECT のパラレル化, 23-23

- 空きリスト, 4-9
- ダイレクト・ロード・インサート, 22-2
 - ロギングなしモード, 11-57, 22-5, 22-7
- トリガー, 19-2, 19-7
 - BEFORE トリガー, 19-10
- パラレル INSERT の記憶域, 22-9
- INSTEAD OF トリガー, 19-13
- NESTED TABLE, 14-5
- オブジェクト・ビュー, 14-5
- Inter-Language Method Services (ILMS), 15-21
- INTERNAL 接続, 5-3
 - 監査されない文の実行, 28-4
- INVALID 状態, 20-3
- IS NULL 述語, 10-8
- ISO SQL 規格, 1-3, 12-21
 - コンボジット外部キー, 25-16

J

- Java
 - トリガー, 19-1, 19-8
- Java Messaging Service, 18-4, 18-14
- JOB_QUEUE_PROCESSES パラメータ, 18-12

L

- LCK0 バックグラウンド・プロセス, 1-18, 8-13
- LGWR バックグラウンド・プロセス, 1-17, 8-9
 - 「ログ・ライター・プロセス」も参照
- LICENSE_MAX_SESSIONS パラメータ, 26-21
- LICENSE_SESSIONS_WARNING パラメータ, 26-21
- 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, 12-13
 - NOLOGGING モード, 22-7
 - 制限
 - パラレル DDL, 23-32
 - パラレル DML, 23-45
 - デフォルトのロギング・モード, 22-8
- LOCK TABLE 文, 15-4
- LOCK_SGA パラメータ, 7-13, 7-17
- LOG_ARCHIVE_MAX_PROCESSES パラメータ,
 - 1-18, 8-13
 - 自動アーカイブ, 29-20

- LOG_ARCHIVE_START パラメータ, 29-21
- LOG_BUFFER パラメータ, 7-6
 - システム・グローバル領域のサイズ, 7-13
- LOG_CHECKPOINT_INTERVAL パラメータ, 29-14
- LOG_CHECKPOINT_TIMEOUT パラメータ, 29-14
- LONG RAW データ型, 12-14
 - LONG データ型との類似点, 12-14
 - 索引の設定は禁止, 12-14
 - パーティション化の制限事項, 11-12
- LONG データ型
 - 記憶域, 10-8
 - 自動的に最後の列になる, 10-8
 - 定義, 12-7
 - パーティション化の制限事項, 11-12
- 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
- LRU アルゴリズム
 - 共有 SQL プール, 7-8, 7-10
 - 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
 - フル・テーブル・スキャン, 7-4
 - ラッチ, 8-8

M

- MAXEXTENTS UNLIMITED 記憶域パラメータ, 23-41
- MAXVALUE
 - パーティション表とパーティション索引, 11-20
- MINIMUM EXTENT パラメータ, 23-34
- MOVE PARTITION 文
 - パラレル化のルール, 23-25
 - ロギングなしモード, 11-57, 22-7
- MPP, 「超並列処理」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
- MTS, 「マルチスレッド・サーバー」を参照

N

- NCHAR データ型, 12-6
- NCLOB データ型, 12-13
- NESTED TABLE, 10-10, 13-11
 - INSTEAD OF トリガー, 14-5

- 索引構成表, 10-41
 - キー圧縮, 10-32
- 制限, 23-30
- ビュー内で更新, 14-5
- Net8, 1-18, 1-34, 6-4
 - 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
- NET81, 30-12
- NLS
 - 「各国語サポート」を参照
- NLS_DATE_FORMAT パラメータ, 12-10
- NLS_LANGUAGE パラメータ, 11-19
- NLS_LANG 環境変数, 11-19
- NLS_NUMERIC_CHARACTERS パラメータ, 12-9
- NLS_SORT パラメータ
 - パーティション化キーに影響を与えない, 11-19
- NOARCHIVELOG モード, 29-18
 - LOGGING モードとの関係, 22-6
 - 概要, 1-46
 - 定義, 29-18
 - リカバリ用のデータベース・バックアップ, 29-25
- NOAUDIT 文, 15-5
 - ロック, 24-30
- NOLOGGING モード
 - 影響を受ける SQL 操作, 22-7
 - ダイレクト・ロード・インサート, 22-5
 - パーティション, 11-57
 - パラレル DDL, 23-31, 23-33
- NOT NULL 制約
 - PRIMARY KEY による暗黙, 25-12
 - UNIQUE キー, 25-11
 - 制約チェック, 25-23
 - 定義, 25-7
- NOVALIDATE 制約, 25-26
- Novell の NetWare Management System, 30-33
- NULL
 - NULL 以外の値, 10-8
 - UNIQUE キー制約, 25-11
 - UNIQUE キーでの不一致, 25-11
 - 値に変換, 10-8
 - 外部キー, 25-15, 25-16
 - 格納方法, 10-8
 - 禁止, 25-7
 - 索引, 10-9, 10-25, 10-38
 - 主キーでは使用禁止, 25-11

- 定義, 10-8
- デフォルト値, 10-9
- パーティション表とパーティション索引, 11-20
- 比較では UNKNOWN 扱い, 10-8
- 列の順序, 10-8
- NUMBER データ型, 12-7
 - 内部形式, 12-9
 - ラウンド, 12-8
- NVARCHAR2 データ型, 12-6
- NVL 関数, 10-8
- n-way* レプリケーション, 「マルチマスター・レプリケーション」を参照, 31-5

O

- Object Type Translator (OTT), 13-15
- OCI, 8-25
 - OCIObjectFlush, 14-4
 - OCIObjectPin, 14-4
 - オブジェクト・キャッシュ, 13-14
 - ストアド・プロシージャ, 15-19
 - バインド変数, 15-14
 - 無名ブロック, 15-18
- ODCIIndex, 10-46
- OLTP データベース, 11-6
 - 索引のパーティション化, 11-35
 - パーティション, 11-7
 - バッチ・ジョブ, 23-39
 - パラレル DML, 23-38
- OPEN_CURSORS パラメータ, 15-7
 - プライベート SQL 領域の管理, 7-9
- OPEN_LINKS パラメータ, 7-16
- OPS, 「Oracle Parallel Server」を参照
- OPTIMAL 記憶域パラメータ, 4-24
- OPTIMIZER_PERCENT_PARALLEL パラメータ, 21-9
- Oracle
 - Oracle Server, 1-4
 - Parallel Server オプション, 1-5
 - 「Parallel Server」も参照
 - SQL 処理, 15-9
 - アーキテクチャ, 1-11
 - 移植性, 1-3
 - インスタンス, 1-4, 1-14, 5-2
 - 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, 25-5
 - スケーラビリティ, 6-4
 - 接続性, 1-2
 - データ・アクセス, 1-49
 - 動作例, 1-19
 - 専用サーバー, 8-24
 - 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, 1-34
 - プロセス, 1-15, 8-5
 - ライセンス, 26-20
 - Oracle Advanced Queuing, 「アドバンスド・キューイング」を参照
 - Oracle Data Cartridge Interface, 10-46
 - Oracle Enterprise Login Assistant, 26-6
 - Oracle Enterprise Manager
 - ALERT ファイル, 8-15
 - PL/SQL, 15-18, 15-19
 - SGA のサイズの表示, 7-13
 - SQL 文, 15-2
 - 起動, 5-5
 - システム権限の付与, 27-3
 - スキーマ・オブジェクト権限, 27-4
 - チェックポイント統計, 8-11
 - 停止, 5-10, 5-11
 - 統計モニター, 26-20
 - パッケージの実行, 17-6
 - パラレル・リカバリ, 29-11
 - プロシージャの実行, 17-4
 - ロールの付与, 27-19
 - ロックとラッチのモニター, 24-30
 - Oracle Enterprise Security Manager, 26-5
 - Oracle Forms
 - PL/SQL, 15-17
 - オブジェクト依存性, 20-13
 - Oracle Internet Directory, 26-5
 - Oracle Parallel Server, 1-5
 - DML ロックとパフォーマンス, 11-46
 - PCM ロック, 24-20
 - 一時表領域, 3-12
 - インスタンス・グループ, 23-19
 - 逆キー索引, 10-33
 - 共有モード
 - ロールバック・セグメント, 4-27
 - システム変更番号, 8-10
 - システム・モニター・プロセス, 8-11, 23-42
 - ディスクの親和性, 23-49
 - 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, 5-7
 - 同時実行の制限, 26-22
 - 名前付きユーザー・ライセンス, 26-23
 - 排他モード
 - ロールバック・セグメント, 4-27
 - パラレル SQL, 23-1
 - ファイルとログの管理ロック, 24-31
 - 分散ロック, 24-20
 - 分離レベル, 24-12
 - 読取り一貫性, 24-7
 - ロック・プロセス, 1-18, 8-13
 - Oracle Replication Manager, 31-13
 - Oracle Server, 1-4
 - 「Oracle」も参照
 - Oracle Wallet Manager, 26-5
 - Oracle XA
 - 大規模プール内のセッション・メモリー, 7-12
 - Oracle Wallet, 26-5
 - Oracle コード, 8-2, 8-25
 - Oracle コール・インタフェース, 「OCI」を参照
 - Oracle 認証局, 26-5
 - Oracle プログラム・インタフェース (OPI), 8-26
 - Oracle ブロック, 1-7, 4-2
 - 「データ・ブロック」も参照
 - ORDBMS, 1-21, 13-2
 - OTT, 「Object Type Translator (OTT)」を参照
 - OUTLN スキーマ
 - DBA 権限, 21-7
- ## P
-
- PARALLEL SERVER パラメータ, 5-7
 - Parallel Server, 「Oracle Parallel Server」を参照
 - PARALLEL_INDEX ヒント, 23-17
 - PARALLEL_MAX_SERVERS パラメータ, 23-8
 - PARALLEL_MIN_PERCENT パラメータ, 23-19
 - PARALLEL_MIN_SERVERS パラメータ, 23-7, 23-8
 - PARALLEL 句
 - パラレル化ルール, 23-20

PARALLEL ヒント, 23-17
 UPDATE と DELETE, 23-22
 パラレル化ルール, 23-20
PARTITION 句, 11-62
PCTFREE 記憶域パラメータ
 PCTUSED, 4-7
 仕組み, 4-6
PCTUSED 記憶域パラメータ
 PCTFREE, 4-7
 仕組み, 4-7
peer-to-peer レプリケーション, 「マルチマスター・レ
 プリケーション」を参照
PGA, 1-15, 7-14
 マルチスレッド・サーバー, 8-19
PKI, 26-5
PL/SQL, 15-16
 DDL 文の解析, 15-20
 PL/SQL エンジン, 15-16, 17-2
 コンパイラ, 17-17
 製品, 15-17
 プロシージャの実行, 17-20
 オブジェクト・ビュー, 14-4
 解析ロック, 24-30
 外部プロシージャ, 15-21, 17-12
 概要, 1-53, 15-16
 拡張パーティション表名, 11-63
 ゲートウェイ, 15-21
 言語構文, 15-18
 実行, 15-16, 17-19, 17-20
 ストアド・プロシージャ, 1-23, 15-16, 17-2, 17-6
 データ型, 12-2
 データベース・トリガー, 19-1
 動的 SQL, 15-20
 バインド変数
 ユーザー定義型, 13-13
 パッケージ, 17-4, 17-12
 プログラム・ユニット, 1-23, 7-10, 15-16, 17-2
 共有 SQL 領域, 7-10
 コンパイル済, 15-17, 17-11, 17-17
 プロシージャ内のロール, 27-20
 文の監査, 28-4
 無名ブロック, 15-16, 17-11
 ユーザー定義データ型, 13-13
 ユーザー・ロック, 24-40
 例外処理, 15-19
PL/SQL Server Pages, 15-21

PMON バックグラウンド・プロセス, 1-17, 8-12
 「プロセス・モニター・プロセス」も参照
Point-in-Time リカバリ
 クローン・データベース, 5-7
Point-to-Point モデル, 18-4
PRIMARY KEY 制約, 25-11
 暗黙の NOT NULL 制約, 25-12
 規定に索引が使用される, 25-12
 名前, 25-12
 制約チェック, 25-23
 説明, 25-11
 列の最大数, 25-12
Pro*C/C++
 SQL 文の処理, 15-11
 ユーザー定義データ型, 13-13
PSP, 「PL/SQL Server Pages」を参照
PUBLIC ユーザー・グループ, 26-16, 27-20
 プロシージャの妥当性, 17-19
P コード, 17-18

Q

QMN_n バックグラウンド・プロセス, 1-18, 8-14,
 18-8
 実行の時間枠, 18-9
 定期的な統計, 18-14

R

RADIUS, 26-7
RAWTOHEX 関数, 12-23
RAW データ型, 12-14
RDBMS, 1-20
 「Oracle」も参照
 オブジェクト・リレーショナル DBMS, 1-21, 13-2
READ ONLY 句
 ALTER TABLESPACE, 3-11
READ WRITE 句
 ALTER TABLESPACE, 3-11
REBUILD INDEX PARTITION 文, 11-59
 パラレル化のルール, 23-25
 ロギングなしモード, 22-7
REBUILD INDEX 文
 パラレル化のルール, 23-25
 ロギングなしモード, 11-57, 22-7

Recovery Manager, 1-49, 29-16
 カタログを使用しない操作, 29-17
 パラレル操作, 29-17
 パラレル・リカバリ, 29-11
 リカバリ・カタログ, 29-16
 レポートの生成, 29-18
REDO ログ, 1-9, 29-7, 29-9
 アーカイバ (ARCn) プロセス, 1-17, 8-12
 アーカイブ, 1-46, 29-19
 アーカイブ時のエラー, 29-21
 自動, 29-20
 手動, 29-21
 アーカイブ・モード, 29-18
 一時セグメントが関係する場合, 4-18
 エントリ, 1-9, 29-9
 オンラインまたはオフライン, 1-45, 1-46, 29-7
 概要, 1-9, 1-45
 コミットされたデータ, 29-9
 コミットされていないデータ, 29-9
 循環バッファ, 8-9
 制御ファイルに名前がある, 29-22
 多重化, 1-45
 用途, 1-9
 トランザクションのコミット, 8-10
 トランザクションのコミット前の書込み, 8-10
 バッファ, 1-14, 7-6
 バッファ管理, 8-9
 バッファの書込み, 8-9
 バッファのサイズ, 7-6
 パラレル・リカバリ, 29-11
 モード, 1-46
 リカバリ, 29-7
 ロールフォワード, 29-9
 インスタンス障害, 29-4
 ログ順序番号, 1-45
 制御ファイルに記録される, 29-23
 ログ・スイッチ
 ALTER SYSTEM SWITCH LOGFILE, 8-12
 アーカイバ・プロセス, 1-17, 8-12
 ログ・ライター・プロセス, 7-6, 8-9
REF
 暗黙の参照解除, 13-10
 オブジェクト・ビューの行, 14-3
 参照解除, 13-9
 参照先がない, 13-9
 ピン, 14-4
 有効範囲付き, 13-9

REFERENCES 権限
 ロールを介して付与された場合, 27-21
REFTOHEX 関数, 12-23
REMOTE_DEPENDENCIES_MODE パラメータ,
 20-11
RENAME 文, 15-4
RESOURCE ロール, 27-22
RESTRICTED SESSION 権限, 26-21
REVOKE 文, 15-5
 ロック, 24-30
ROLLBACK 文, 15-5
ROWID, 10-8
 Oracle 以外のデータベース, 12-21
 アクセス, 12-15
 行の移行, 4-10
 クラスタ化された行, 10-8
 索引のソートでの使用, 10-30
 内部使用, 12-15, 12-19
 汎用, 12-15
 物理, 12-15
 変更, 12-16
 論理, 12-15
 論理 ROWID, 12-19
 索引構成表の索引, 10-42
 推測の陳腐化, 12-20
 推測の統計, 12-20
 物理推測, 10-42, 12-19
ROWIDTOCHAR 関数, 12-22
ROWID データ型, 12-15
 拡張 ROWID 形式, 12-16
 制限付き ROWID 形式, 12-17
ROW サンプルング, 21-16

S

SAMPLE 句
 コストベース最適化, 21-17
SAVEPOINT 文, 15-5
SCN, 「システム変更番号」を参照
SELECT 文, 15-4
 SAMPLE 句
 コストベース最適化, 21-17
 「問合せ」も参照
 コンボジット索引, 10-24
 副問合せ, 15-13

- Server Manager
 - PL/SQL, 15-18, 15-19
 - SQL 文, 15-2
- 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, 27-21
- SET CONSTRAINTS 文
 - DEFERRABLE または IMMEDIATE, 25-25
- SET ROLE 文, 15-6
- SET TRANSACTION 文, 15-5
 - ISOLATION LEVEL, 24-8, 24-32
 - READ ONLY 句, 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, 7-14
- 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, 7-13
- SHUTDOWN ABORT 文, 5-11
 - 必要なクラッシュ・リカバリ, 29-4
- SINGLE TABLE HASHKEYS, 10-60
- SKIP_UNUSABLE_INDEXES パラメータ, 20-8
- SMON バックグラウンド・プロセス, 1-17, 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, 23-50
- Snnn バックグラウンド・プロセス, 8-15
- SNPn バックグラウンド・プロセス, 1-18, 8-13
 - メッセージの伝播, 18-12
- SORT_AREA_RETAINED_SIZE パラメータ, 7-16
- SORT_AREA_SIZE パラメータ, 4-18, 7-16
- SPLIT PARTITION 文
 - パラレル化のルール, 23-25
 - ロギングなしモード, 11-57, 22-7
- SQL, 15-2
 - PL/SQL, 1-53, 15-16
 - 埋込み, 1-51, 15-6
 - ユーザー定義データ型, 13-14
 - カーソル, 15-7
 - 解析, 15-8
 - 概要, 1-49, 15-2
 - 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, 15-2
 - CHECK 制約, 25-21
 - COUNT, 10-38
 - NVL, 10-8
 - 列のデフォルト値, 10-9
- 共有 SQL, 15-8
- 再帰, 15-7
 - カーソル, 15-7
- システム制御文, 15-6
- セッション制御文, 15-6
- データ操作言語 (DML), 15-4
- データ定義言語 (DDL), 15-4
- 動的 SQL, 15-20
- トランザクション, 1-51, 16-2, 16-5
- トランザクション制御文, 15-5
- パラレル実行, 23-2
- 文のタイプ, 1-50, 15-3
- 文レベルのロールバック, 16-4
- メモリー割当て, 7-11
- ユーザー定義データ型, 13-13
 - OCI, 13-14
 - 埋込み SQL, 13-14
- 予約語, 15-3
- SQL*Loader, 1-11
 - ダイレクト・ロード
 - NOLOGGING モード, 11-57, 22-7
 - ダイレクト・ロード・インサートに類似, 22-2
 - パラレル・ダイレクト・ロード, 22-3
 - パーティション操作, 11-46, 11-48
- SQL*Menu
 - PL/SQL, 15-17
- SQL*Module
 - FIPS フラガー, 15-6
 - ストアド・プロシージャ, 15-19
- SQL*Net
 - 「Net8」を参照
- SQL*Plus
 - ALERT ファイル, 8-15
 - SGA のサイズの表示, 7-13
 - SQL 文, 15-2
 - ストアド・プロシージャ, 15-19
 - セッション変数, 15-18
 - 接続, 26-4
 - 統計モニター, 26-20
 - パッケージの実行, 17-6
 - パラレル・リカバリ, 29-11
 - プロシージャの実行, 17-4
 - 無名ブロック, 15-18
 - ロックとラッチのモニター, 24-30
- SQL_TRACE パラメータ, 8-15
- SQL92, 24-2

- SQL 文, 1-50, 15-3, 15-9
 - 1 つの SQL 文で起動されるトリガーの数, 19-23
 - 依存オブジェクトの参照, 20-4
 - 埋込み, 15-6
 - カーソルの作成, 15-12
 - 解析, 15-12
 - 解析ロック, 24-30
 - 概要, 1-50
 - 監査, 28-7, 28-9
 - 概要, 1-42
 - レコードが生成される場合, 28-4
 - 実行, 15-9, 15-15
 - 実行計画, 21-2
 - 障害, 29-3
 - 正常な実行, 16-3
 - タイプ, 1-50, 15-3
 - ディクショナリ・キャッシュ・ロック, 24-31
 - トランザクション, 15-15
 - トリガー, 19-2, 19-9
 - トリガー・イベント, 19-7
 - 配列処理, 15-15
 - パラレル化, 23-2, 23-10
 - パラレル実行, 23-2
 - ハンドル, 1-15
 - 必要な権限, 27-3
 - 分散
 - ノードへのルーティング, 15-12
 - 分散データベース, 30-34
 - リソース制限, 26-18
- SQL 文のハンドル, 1-15, 7-9
- SQL 領域
 - 共有, 1-14, 7-8, 15-8
 - プライベート, 7-8
 - 持続, 7-8
 - ランタイム, 7-8
- STARTUP FORCE 文
 - 必要なクラッシュ・リカバリ, 29-4
- STORAGE 句
 - 使用, 4-11
 - パラレル実行, 23-34
- SUBPARTITION 句, 11-62
- SunSoft の SunNet Manager, 30-33
- SYS.AUD\$ ビュー
 - 削除, 2-5
- SYSDBA 権限, 5-3
- SYSOPER 権限, 5-3

- SYSTEM 表領域, 3-6
 - 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, 3-6
 - 格納されるプロシージャ, 3-6, 17-18
 - メディア障害, 29-6
- SYSTEM ユーザー名
 - セキュリティ・ドメイン, 26-3
- 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, 28-4
 - 所有する一時スキーマ・オブジェクト, 26-15
 - セキュリティ・ドメイン, 26-3
 - データ・ディクショナリ表の所有, 2-3

T

- TAF, 29-15
- TO_CHAR 関数
 - CHECK 制約のデフォルト NLS, 25-21
 - データ変換, 12-22
 - ビューのデフォルト NLS, 10-15
 - ユリウス暦, 12-10
- TO_DATE 関数, 12-10
 - CHECK 制約のデフォルト NLS, 25-21
 - データ変換, 12-22
 - パーティション, 11-12, 11-21
 - ビューのデフォルト NLS, 10-15
 - ユリウス暦, 12-10
- TO_NUMBER 関数, 12-9
 - CHECK 制約のデフォルト NLS, 25-21
 - データ変換, 12-22
 - ビューのデフォルト NLS, 10-15
 - 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 文, 15-4

U

- UNIQUE キー制約, 25-8
 - NOT NULL 制約, 25-11
 - NULL, 25-11
 - 規定に索引が使用される, 25-10
 - コンポジット・キー, 25-9, 25-11

サイズの制限, 25-10
制約チェック, 25-23
列の最大数, 25-10
UNLIMITED エクステンツ, 23-41
UNUSED 索引
 ファンクション, 20-8
UNUSED 列, 10-7
UPDATE No Action 制約, 25-16
UPDATE 文, 15-4
 外部キー参照, 25-16
 データ・ブロック内の領域の解放, 4-9
 トリガー, 19-2, 19-7
 BEFORE トリガー, 19-10
 パラレル UPDATE, 23-22
 ロギングなしモード, 22-8
 LOB, 22-7
UROWID データ型, 12-15
USE_INDIRECT_DATA_BUFFERS パラメータ, 7-14
USE_STORED_OUTLINES セッション・パラメータ,
 21-7
USER_UPDATABLE_COLUMNS ビュー, 10-16
USER_ ビュー, 2-6
USER 疑似列, 27-7

V

V_\$ ビューと V\$ ビュー, 2-7
 V\$LICENSE, 26-22
VALIDATE 制約, 25-26
VALUES LESS THAN 句, 11-19
 DATE データ型, 11-21
 MAXVALUE, 11-20, 11-22
 複数列からなるキー, 11-22
 例, 11-14, 11-17
VARCHAR2 データ型, 12-5
 RAW データ型との類似点, 12-14
 非空白埋め比較方法, 12-6
VARCHAR データ型, 12-6
VARRAY, 13-11
 索引構成表, 10-41
 キー圧縮, 10-32
VLDB
 パーティション, 11-5
 パラレル SQL, 23-2

W

Wallet, 26-5
Wallet Manager, 26-5
web ページのスクリプト作成, 15-21
WITH OBJECT OID 句, 14-3, 14-4

X

X.509 証明書, 26-5
XA
 大規模プール内のセッション・メモリー, 7-12

あ

アーカイバ (ARC*n*) プロセス
 自動アーカイブ, 29-20
 手動アーカイブに使用しない, 29-22
 説明, 1-17, 8-12
 トレース・ファイル, 29-21
 マルチ・プロセス, 1-18, 8-13
 例, 29-19
アーカイブ REDO ログ, 1-46
 自動アーカイブ, 29-20
 手動アーカイブ, 29-21
 使用可能にする, 29-19
アーキテクチャ
 MPP, 23-50
 Oracle, 1-11
 SMP, 23-50
 クライアント / サーバー, 1-31
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, 1-17, 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
 エクステンツ, 4-13
 SMON プロセス, 1-17, 8-11
 データ・ブロック内, 4-9
アクセス制御, 27-2
 権限, 27-2
 任意, 1-37
 パスワード暗号化, 26-7

- ファイングレイイン・アクセス・コントロール, 27-23
- ロール, 27-17
- アクセス・パス
 - 定義, 21-5
- アクセス方法
 - 実行計画, 21-2
- 「アクセスをシリアル化できません。», 24-11
- 圧縮、索引キー, 10-31
- アドバンスト・キューイング, 18-1
 - DEQUEUE 要求, 18-10
 - Point-to-Point モデル, 18-4
 - イベントの発行, 19-19
 - エージェント, 18-6
 - 機能, 18-9
 - キュー表, 18-6
 - キュー表のエクスポート, 18-15
 - キュー・モニター, 18-8
 - キュー・モニター・プロセス, 1-18, 8-14, 18-8
 - 実行の時間枠, 18-9
 - 定期的な統計, 18-14
 - サブスクライバ, 18-7
 - パブリッシュ / サブスクライプのサポート, 18-13, 19-19
 - パブリッシュ / サブスクライプ・モデル, 18-5
 - メッセージ, 18-6
 - メッセージ・キューイング, 18-2
 - ユーザー・キュー, 18-6
 - リモート・データベース, 18-12
 - ルール, 18-8
 - 例外キュー, 18-6
 - 例外処理, 18-14
 - レシビエント, 18-7
 - サブスクライバ・リスト, 18-7
 - ルールベースのサブスクリプション, 18-8
- 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, 19-3
 - 意思決定支援システム (DSS), 10-35
 - パラレル SQL, 23-2, 23-32
 - 依存性, 20-11
 - オブジェクト依存性, 20-13
 - オンライン・トランザクション処理 (OLTP)
 - 逆キー索引, 10-33
 - オンライン分析処理 (OLAP), 10-45
 - 空間データ・アプリケーション, 10-45
 - コードの共有, 7-18

- コンテキスト, 27-25
- 索引構成表, 10-43
- 情報検索 (IR), 10-44
- 制約違反を検出可能, 25-6
- セキュリティ
 - アプリケーション・コンテキスト, 27-25
- セキュリティの強化, 1-40, 25-6
- ダイレクト・ロード・インサート, 23-39
- ディスクリット・トランザクション, 16-9
- データ・ウェアハウス, 10-34
- 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, 16-5
- ネットワーク通信, 6-5
- パラレル DML, 23-38
- プログラム・インタフェース, 8-25
- プロセス, 8-4
 - ロール, 27-18
- アラート・ファイル, 8-15
 - ARCn プロセス, 8-13
 - REDO ログ, 8-10
- 暗黙の参照解除, 13-10

い

- 意思決定支援システム (DSS), 11-6
 - スコア表, 23-38
 - ディスクのストライプ化, 23-49
 - パーティション, 11-6
 - パフォーマンス, 11-8, 23-38
 - パラレル DML, 23-38
 - パラレル SQL, 23-2, 23-32, 23-38
 - ビットマップ索引, 10-35
 - マテリアライズド・ビュー, 10-18
- 異種サービス, 30-5
- 移植性, 1-3
- 依存性, 20-1
 - Oracle Forms トリガー, 20-13
 - 管理, 20-1
 - 共有ルール, 20-10
 - 権限, 20-6
 - スキーマ・オブジェクト間, 20-2
 - 存在しない参照オブジェクト, 20-8
 - 存在しない他のオブジェクト, 20-9
 - ファンクション索引, 10-28, 20-7
 - リモート・オブジェクト, 20-10
 - ローカル, 20-10

- 一意キー, 1-56, 25-9
 - コンポジット, 25-9
- 一意索引, 10-24
- 一時セグメント, 4-15, 4-18, 10-11
 - REDO ログに記録されない場合, 4-18
 - エクステントの割当て解除, 4-15
 - 削除, 4-15
 - 問合せ用の割当て, 4-18
 - パラレル DDL, 23-34
 - パラレル INSERT, 22-9
 - 必要な操作, 4-17
 - 表領域, 4-15, 4-18
 - 割当て, 4-18
 - 割当て制限は無視される, 26-15
- 一時表, 10-11
- 一時表領域, 3-12
- 位置の透過性, 1-33
- インスタンス化, 31-11
- インスタンス, 1-4
 - 異常終了, 5-10, 29-4
 - インスタンス・グループ, 23-19
 - 概要, 1-4
 - 仮想メモリー, 7-17
 - 起動, 5-5
 - 監査レコード, 28-5
 - サービス名, 6-6
 - システム識別子 (SID), 6-6
 - 障害, 1-44, 29-4
 - 図, 8-6
 - 制限モード, 5-6
 - 説明, 5-2
 - 定義, 1-14
 - 停止, 5-10, 5-11
 - 監査レコード, 28-5
 - データベースに対応付け, 5-2, 5-6
 - データベースの共有, 1-5
 - 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
 - リカバリ, 5-10, 29-4
 - 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, 29-14
 - ロールバック・セグメントの取得, 4-26
- インスタンスの異常終了, 5-10, 29-4

- インスタンスのリカバリ, 29-4
 - SMON プロセス, 1-17, 8-11, 23-42
 - 「クラッシュ・リカバリ」も参照
 - インスタンス障害, 1-44, 29-4
 - 読取り専用表領域, 29-6
- インターオペレータ並行性, 23-13
- インダウト・トランザクション, 4-24, 5-9
- イントラオペレータ並行性, 23-13
- インライン・ビュー, 10-17
 - 例, 10-17

う

- ウェアハウス
 - 「データ・ウェアハウス」も参照
 - 表データのリフレッシュ, 23-38
 - マテリアライズド・ビュー, 10-18
- 埋込み SQL, 1-51, 15-6
 - PL/SQL の動的 SQL, 15-20

え

- 永続キューイング, 18-3
- エクステント
 - 概要, 4-10
 - 管理, 4-11
 - 結合, 4-14
 - 増分, 4-10
 - 定義, 4-3
 - ディクショナリ管理, 3-7
 - 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, 23-34
 - パラレル INSERT
 - 記憶域パラメータ, 22-9
 - マテリアライズド・ビュー, 4-15
 - ローカルに管理される, 3-8
 - ロールバック・セグメント内
 - カレントの変更, 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, 15-6
 - トレース・ファイルに記録, 8-15
- エンタープライズ・ユーザー, 26-2

お

- 応答キュー, 8-17
- 応答時間, 21-9
- オーダー・メソッド, 1-55, 13-7
- オブジェクト
 - 権限, 27-12
- オブジェクト型, 1-21, 13-2, 13-3
 - Object Type Translator, 13-15
 - オブジェクト・ビュー, 10-17
 - 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, 1-55, 13-6
 - 制限
 - パラレル DDL, 23-32
 - パラレル DML, 23-45
 - 属性, 13-2, 13-4
 - 発注の例, 13-2, 13-4
 - パラレル問合せ, 23-30
 - 制限, 23-30
 - 比較メソッド, 13-6
 - メソッド, 1-55, 13-4
 - PL/SQL, 13-13
 - 発注の例, 13-2, 13-5
 - メッセージ・キューイング, 18-9
 - 列オブジェクト, 13-8
- オブジェクト型の属性, 13-4
- オブジェクト型のメソッド, 1-55, 13-4
 - PL/SQL, 13-13
 - オーダー・メソッド, 1-55, 13-7
 - コンストラクタ・メソッド, 1-55
 - 自己参照的なスタイルによる起動, 13-6
 - 発注の例, 13-2, 13-5
 - マップ・メソッド, 1-55, 13-7
- オブジェクト・キャッシュ
 - OCI, 13-14

- Pro*C, 13-14
- オブジェクト・ビュー, 14-4
- オブジェクト権限, 27-3
 - 「スキーマ・オブジェクト権限」も参照
- オブジェクト識別子, 14-3, 14-4
 - WITH OBJECT OID 句, 14-3, 14-4
 - オブジェクト・ビュー, 14-3, 14-4
 - コレクション
 - キー圧縮, 10-32, 10-41
- オブジェクト・ビュー, 10-17, 14-1
 - INSTEAD OF トリガーの使用方法, 14-5
 - NESTED TABLE, 14-5
 - オブジェクト識別子, 14-3, 14-4
 - 更新, 14-5
 - 定義, 14-3
 - 変更の問題, 19-13
 - 利点, 14-2
- オブジェクト表, 13-2, 13-7
 - 仮想的なオブジェクト表, 14-2
 - 行オブジェクト, 13-8
- オブジェクト・リレーショナル DBMS (ORDBMS), 1-21, 13-2
- オフライン REDO ログ・ファイル, 1-46, 29-7
- オフライン・バックアップ
 - データベース全体のバックアップ, 29-24
- 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, 26-4
 - ブロック・サイズ, 4-3
 - ロール, 27-23
- オンライン REDO ログ, 1-45, 29-7
 - アーカイブ, 29-19, 29-20
 - 制御ファイルに記録される, 29-22
 - 多重化, 29-6
 - チェックポイント, 29-23
 - メディア障害, 29-6
- オンライン・トランザクション処理 (OLTP), 11-6
 - 逆キー索引, 10-33

か

- カーソル
 - 埋込み SQL, 15-6
 - オープン, 7-9, 15-7
 - オブジェクト依存性, 20-10
 - 概要, 1-15

- 再帰, 15-7
- 再帰 SQL, 15-7
- 最大数, 15-7
- 作成, 15-12
- ストアド・プロシージャ, 15-19
- 定義, 15-7
- プライベート SQL 領域, 7-9, 15-7
- カーディナリティ, 10-35
- 解析, 15-12
 - DBMS_SQL パッケージ, 15-20
 - SQL 文, 15-12, 15-20
 - 埋込み SQL, 15-6
 - 解析コール, 15-8
 - 解析ロック, 15-12, 24-30
 - 実行, 15-8
- 解析ツリー, 17-18
 - 共有 SQL 領域, 7-8
 - 構築, 15-8
 - データベースへの格納, 17-18
- 階層, 1-27, 10-20
 - 結合キー, 1-27, 10-20
 - レベル, 1-27, 10-20
- 外部キー, 1-56
 - 一部が NULL, 25-16
 - 親キーを使用するための権限, 27-5
 - 定義, 1-56
- FOREIGN KEY 制約
 - 表の更新, 25-19
- 外部キー制約
 - NULL, 25-15
 - 列の最大数, 25-13
- 外部キーのマッチング
 - 完全一致、部分一致または不一致, 25-16
- 外部参照, 17-8
 - 名前解決, 17-20
- 外部プロシージャ, 15-21, 17-12
- 書込みが読込みを阻止するか, 24-11
- 拡張 ROWID 形式, 12-16
- 拡張可能な最適化, 21-18
 - ユーザー定義コスト, 21-19
 - ユーザー定義選択性, 21-19
 - ユーザー定義統計, 21-18
- 仮想表, 1-22
- 仮想メモリー, 7-17
- 型
 - 権限, 27-12
 - 「データ型」、「オブジェクト型」を参照
- 各国語サポート (NLS)
 - CHECK 制約, 25-21
 - DATE データ型とパーティション, 11-12, 11-21
 - NCHAR および NVARCHAR2 データ型, 12-6
 - NCLOB データ型, 12-13
 - 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, 30-47
 - パラメータ, 5-5
 - ビュー, 10-15
- 仮読込み, 24-3, 24-11
- 監査, 1-41, 28-1
 - DDL 文, 28-7
 - DML 文, 28-7
 - アクセス別, 28-11
 - 必須, 28-12
 - オプションが有効になる時期, 28-5
 - 監査オプション, 28-3
 - 監査証跡, 28-3
 - オペレーティング・システム, 28-5, 28-6
 - データベース, 28-3
 - 監査レコード, 28-3
 - 管理者権限での接続, 28-5
 - 起動と停止, 28-5
 - 権限の使用, 28-2, 28-7
 - 失敗した実行, 28-9
 - 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, 28-2, 28-8
 - 正常な実行, 28-9
 - セキュリティ, 28-6
 - セッション別, 28-10
 - 禁止, 28-12
 - 説明, 1-41, 28-2
 - 対象範囲, 28-3, 28-9
 - タイプ, 28-2
 - ディクショナリ内のデータの削除, 2-5
 - データベースと OS のユーザー名, 26-4
 - トランザクションに依存しない, 28-4
 - パーティション表とパーティション索引, 11-61
 - 文, 28-2, 28-7
 - 分散データベース, 28-6
 - ユーザー, 28-12
- 関数
 - Java
 - パラレル実行, 23-48
 - PL/SQL, 17-2, 17-6
 - DETERMINISTIC, 20-8
 - 「プロシージャ」も参照

- 権限, 27-7
- パラレル実行, 23-48
- プロシージャと対比, 17-2
- ロール, 27-20
- SQL, 15-3
 - CHECK 制約, 25-21
 - COUNT, 10-38
 - NVL, 10-8
 - デフォルトの列値, 10-9
 - ビューでの使用, 10-15
- ハッシュ関数, 10-57
- ファンクション索引, 10-26
- ユーザー定義
 - 拡張可能な最適化, 21-18
- 管理されたスタンバイ・データベース, 29-28
- 管理者権限, 5-3
 - OUTLN スキーマ, 21-7
 - 監査されない文の実行, 28-4
 - 監査される接続, 28-5

き

- キー
 - 値の最大記憶域, 10-25
 - 一意, 25-8
 - コンポジット, 25-9, 25-11
 - 親, 25-13, 25-15
 - 外部, 25-13
 - キー値, 1-56
 - 逆キー索引, 10-33
 - クラスタ, 1-25, 10-51
 - 索引, 10-25
 - PRIMARY KEY 制約, 25-12
 - UNIQUE 制約, 25-10
 - 圧縮, 10-31
 - 逆キー, 10-33
 - 参照, 1-56, 25-13
 - 主, 25-11
 - 制約における, 1-56
 - 定義, 25-9
 - ハッシュ, 10-56, 10-60
- 一意キー
 - コンポジット, 25-11
- キー圧縮, 10-31
- 記憶域
 - NULL, 10-8
 - クラスタ, 10-52

- 索引, 10-28
- 索引パーティション, 11-36
- データ・ファイル, 3-16
- トリガー, 19-2, 19-26
- ハッシュ・クラスタ, 10-54
- パラレル DDL の断片化, 23-34
- パラレル INSERT, 22-9
- ビュー定義, 10-15
- 表パーティション, 11-26
- 表領域の取消し, 26-15
- 表領域割当て制限, 26-15
- ユーザーごとの割当て制限, 1-40
- ユーザーに対する制限, 26-15
- 論理構造, 3-5, 10-2
- 記憶域パラメータ
 - MAXEXTENTS UNLIMITED, 23-41
 - NEXT, 22-9
 - OPTIMAL (ロールバック・セグメント), 4-24, 23-41
 - PCTINCREASE, 22-9
 - 設定, 4-11
 - パラレル・ダイレクト・ロード・インサート, 22-9
- 規格, 1-3
 - ANSI/ISO, 1-3, 25-5, 25-16
 - 分離レベル, 24-2, 24-11
 - FIPS, 15-6
 - Oracle が準拠, 1-3
 - 整合性制約, 25-5, 25-16
- 規格化されていない機能のフラグ付け, 15-6
- 疑似コード, 17-18
 - トリガー, 19-26
- 疑似列
 - CHECK 制約が禁止する
 - LEVEL および ROWNUM, 25-21
 - ROWID, 12-15
 - USER, 27-7
 - ビューの変更, 19-14
- 起動, 5-2, 5-5
 - SGA の割当て, 7-2
 - 開始アドレス, 7-14
 - 監査レコード, 28-5
 - 強制実行, 5-6
 - ステップ, 5-5
 - 制限モード, 5-6
 - ディスパッチャ・プロセスでは禁止, 8-20
 - リカバリ, 29-4
- 逆キー索引, 10-33

- キャッシュ
 - オブジェクト・キャッシュ, 13-14
 - オブジェクト・ビュー, 14-4
 - 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, 1-14
 - 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, 24-7
- キャラクタ・セット
 - CLOB および NCLOB データ型, 12-13
 - NCHAR および NVARCHAR2, 12-6
 - 各国語, 5-5
 - 列の長さ, 12-6
- キュー
 - シングル・コンシューマ, 18-6
 - マルチ・コンシューマ, 18-6
- キューイング, 18-2
 - インスタンス親和性, 18-13
 - キュー表, 18-6, 18-15
 - キュー表のエクスポート, 18-15
 - キュー・モニター・プロセス, 1-18, 8-14, 18-8
 - 実行の時間枠, 18-9
 - 定期的な統計, 18-14
 - キュー・レベルのアクセス制御, 18-12
 - パブリッシュ / サブスクライプのサポート, 18-13
 - イベントの発行, 19-19
 - リモート・データベース, 18-12
 - 例外処理, 18-14
 - レシビエント, 18-7
 - サブスクライバ・リスト, 18-7
 - ルールベースのサブスクリプション, 18-8
- キューイングのエージェント, 18-6
- キュー・モニター (QMN n) プロセス, 1-18, 8-14, 18-8
 - 実行の時間枠, 18-9
 - 定期的な統計, 18-14
- 行, 1-21, 10-3
 - ROWID が変更される場合, 12-16
 - ROWID での表示, 12-16, 12-17
 - アドレス, 10-8
 - 格納形式, 10-5
 - 行オブジェクト, 13-8
 - 行ソース, 21-4
 - 行レベルのセキュリティ, 27-23
 - クラスタ化, 10-7
 - ROWID, 10-8
 - サイズ, 10-5
 - 索引構成表での行オーバーフロー, 10-41
 - 新規ブロックへの移行, 4-10
 - 説明, 10-3
 - 断片, 10-5
 - 定義, 1-21
 - データ・ブロック内での形式, 4-4
 - トリガー, 19-9
 - フェッチ, 15-13
 - ブロックにまたがる連鎖, 4-9, 10-5
 - ヘッダー, 10-5
 - ロック, 11-44, 24-11, 24-21, 24-24
 - 論理 ROWID, 12-19
 - 索引構成表, 10-42
- 行オブジェクト, 13-8
- 行キャッシュ, 7-10
- 競合
 - データ
 - デッドロック, 8-19, 24-17
 - ロックの段階的拡大が発生しない, 24-17
 - プロシージャ型レプリケーション, 31-17
 - ロールバック・セグメント, 4-21
- 行ソース, 21-4
- 行断片, 10-5
 - 識別方法, 10-8
 - ヘッダー, 10-6
- 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, 19-9
 - 「トリガー」も参照
 - 起動されるタイミング, 19-23
- 行の連鎖, 4-9, 10-5
- 共有 SQL 領域, 7-8, 15-8
 - ANALYZE 文, 7-11
 - SQL のロード, 15-12
 - 依存性管理, 7-11
 - 解析ロック, 24-30
 - 概要, 1-14, 15-8
 - サイズ, 7-8
 - 説明, 7-8
 - プロシージャ、パッケージ、トリガー, 7-10

- 共有グローバル領域 (SGA), 7-2
- 共有サーバー, 1-16
- 共有サーバー (Snnn) プロセス, 8-15, 8-19
 - 説明, 8-19
- 共有プール, 7-6
 - ANALYZE 文, 7-11
 - 依存性管理, 7-11
 - オブジェクト依存性, 20-10
 - 概要, 1-14
 - 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, 17-18
 - 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), 24-25
- 行レベル・ロック, 24-11, 24-21
- 行ロック, 24-11, 24-21
 - シリアル化が可能なトランザクション, 24-9
 - ブロック・レベルのリカバリ, 24-21, 29-15

く

- 空間データ・アプリケーション
 - 索引構成表, 10-45
- クエリー・リライト, 10-18
 - セキュリティ・ルール内の動的な述語, 27-24
- クライアント / サーバー・アーキテクチャ, 6-2
 - 概要, 1-31, 6-2
 - クライアント, 1-31
 - 図, 6-2
 - 直接接続と間接接続, 30-9
 - プログラム・インタフェース, 8-25
 - 分散処理, 6-2
 - 分散データベース, 30-7
- クラスタ
 - ROWID, 10-8
 - 概要, 10-49
 - 格納形式, 10-52
 - キー, 1-25, 10-51, 10-52
 - NULL の索引付けへの影響, 10-9
 - 記憶域パラメータ, 10-5
 - クラスタ化するデータの選択, 10-51
 - 結合, 10-51

- 索引, 10-23, 10-53
 - パーティション化できない, 11-2
 - ハッシュと対比, 10-54
- スキャン, 7-4
- 定義, 1-25
- ディクショナリ・ロック, 24-30
- パーティション化できない, 11-2
- ハッシュ, 10-53
 - 記憶域, 10-54
 - 索引と対比, 10-54
 - 衝突の解決, 10-56
 - 単一表, 10-60
 - 領域の割当て, 10-58
 - ルート・ブロック, 10-58
- パフォーマンスの考慮事項, 10-51
- パラメータの設定, 10-52
- 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, 10-51
- クラッシュ・リカバリ, 29-4, 29-14
 - SMON プロセス, 1-17, 8-11
 - インスタンス障害, 1-44, 5-10, 29-4
 - インスタンスの異常終了後に必要, 5-10
 - データベースのオープン, 5-8
 - 読取り専用表領域, 29-6
- グループ、インスタンス, 23-19
- 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, 11-31
 - 索引タイプのまとめ, 11-33
 - パーティションの管理, 11-32, 11-59
- グローバル・スキーマ・オブジェクト名, 1-27
- グローバル・データベース名
 - 共有プール, 7-11
- グローバル・ユーザー
 - カレント・ユーザーのリンク, 17-22
- グローバル・ネーミング, 30-15
- クローン・データベース
 - マウント, 5-7

け

- 計画
 - SQL の実行, 15-4, 15-12
- 軽量セッション, 26-9
- 結合
 - クラスタ, 10-51

- 結合順序
 - 実行計画, 21-2
 - 述語の選択性, 21-9, 21-10, 21-19
- パーティション・ワイズ, 11-5
- ビュー, 1-22, 10-16
- ビューにカプセル化, 1-22, 10-14
- 結合ビュー, 10-16
- 権限
 - RESTRICTED SESSION, 26-21
 - 解析時にチェックされる, 15-12
 - 概要, 1-39, 27-2
 - 監査の使用方法, 1-42, 28-7
 - 管理者, 5-3
 - OUTLN スキーマ, 21-7
 - 監査されない文の実行, 28-4
 - 監査される接続, 28-5
- システム, 27-2
 - 概要, 1-39
 - 付与と取消し, 27-3
- スキーマ・オブジェクト, 27-3
 - DML 操作と DDL 操作, 27-5
 - 概要, 1-39
 - パッケージ, 27-10
 - 付与と取消し, 27-4
 - プロシージャ, 27-7
- データベースの起動または停止, 5-3
- トリガー権限, 27-8
- 取消し, 27-3, 27-4
 - オブジェクト依存性, 20-6
- パーティション表とパーティション索引, 11-61
- ビュー, 27-6
 - 作成, 27-6
 - 使用, 27-6
- ファンクション索引, 10-28, 20-8
- 付与, 1-39, 27-3, 27-4
 - 例, 27-10, 27-11
- プロシージャ, 27-7
 - 作成と変更, 27-9
 - 実行, 17-19, 27-7
 - パッケージ内, 27-10
- ロール, 27-17
 - 制限, 27-21
- ロールとしてグループ化, 1-39

こ

- 公開鍵インフラストラクチャ, 26-5
- 更新
 - 位置の透過性, 30-45
 - オブジェクト・ビュー, 14-5
 - オブジェクト・ビューの更新可能性, 14-5
 - 更新可能な結合ビュー, 10-16
 - ビューの更新可能性, 10-16, 19-13, 19-14
 - 頻繁に更新が行われる環境, 24-9
- 更新可能スナップショット, 31-8
- 高水位標
 - ダイレクト・ロード・インサート, 22-3
- 構成
 - パラメータ・ファイル, 5-4
 - プロセスの構造, 8-2
- 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, 12-17
 - データ・ブロック
 - ROWID での表示, 12-17
 - 物理, 1-8
 - REDO ログ・ファイル, 1-9, 29-7
 - 制御ファイル, 1-9, 29-22
 - データ・ファイル, 1-8, 3-1, 3-16
 - プロセス, 1-11, 1-15, 8-1
 - メモリー, 1-12, 7-1
 - ロック, 24-29
 - 論理, 1-5, 4-1
 - エクステンツ, 1-6, 4-2, 4-10
 - スキーマ・オブジェクト, 1-6, 10-2
 - セグメント, 1-6, 4-2, 4-16
 - データ・ブロック, 1-6, 4-2, 4-3
 - 表領域, 1-5, 3-1, 3-5
- 構造化問合せ言語 (SQL), 1-49, 15-2
 - 「SQL」も参照
- 高速コミット, 8-10
- 高速リフレッシュ, 10-19
- 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, 30-46
- 互換性, 1-3
- 互換性レベル
 - トランスポートابل表領域, 3-15
- コストベース最適化, 21-8, 30-47
 - 拡張可能な最適化, 21-18

- クエリー・リライト, 10-18
- 述語の選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - ユーザー定義, 21-19
- 統計, 21-9
 - ユーザー定義, 21-18
- ヒストグラム, 21-10
- ユーザー定義コスト, 21-19
- 固定ビュー, 2-7
- コミット読み込み分離, 24-7, 24-8
- コレクション, 13-10
 - NESTED TABLE, 13-11
 - 可変配列 (VARRAY), 13-11
 - 索引構成表, 10-41
 - キー圧縮, 10-32
- コレクションのメソッド
 - コンストラクタ・メソッド, 1-55
- コンストラクタ・メソッド, 1-55, 13-6
- コンパイル済 PL/SQL, 17-17
 - 疑似コード, 17-18, 19-26
 - 共有プール, 15-17
 - 再コンパイル, 17-20
 - トリガー, 19-26
 - プロシージャ, 17-11
 - 利点, 17-9
- コンポジット索引, 10-24

さ

- サーバー, 1-31
 - 共有, 1-16
 - クライアント / サーバー・アーキテクチャ, 6-2
 - 専用, 1-16, 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, 1-31
 - プロセス, 1-16
 - マルチスレッド, 1-16
 - アーキテクチャ, 8-3, 8-16
 - 専用と対比, 8-16
 - プロセス, 8-14, 8-15, 8-16, 8-19
- サーバー側スクリプト, 15-21
- サーバー・プロセス, 1-16, 8-5
 - リスナー・プロセス, 6-6
- サービス名, 6-6
- 再帰 SQL
 - カーソル, 15-7

- 最適化, 21-2
 - 拡張可能オプティマイザ, 21-18
 - クエリー・リライト, 10-18
 - セキュリティ・ルール内, 27-24
 - コストベース, 21-8
 - ヒストグラム, 21-10
 - ユーザー定義コスト, 21-19
 - 索引作成, 10-24
 - 述語の選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - ユーザー定義, 21-19
 - 説明, 21-2
 - 統計, 21-9
 - ユーザー定義, 21-18
 - パーティション索引, 11-34
 - パーティションの実行計画, 11-12
 - パーティション・プルーニング, 11-4
 - 索引, 11-35
 - パーティション・プルーニング (絞込み), 11-4
 - パーティション・ワイズ・ジョイン, 11-5
 - パラレル SQL, 23-10
 - ファンクション索引, 10-27
 - プラン・スタビリティ, 21-7
 - ルールベース, 21-20
- 最適化のプラン・スタビリティ, 21-7
- サイト自律性, 1-33, 30-24
- 作業負荷の不整, 23-19
- 索引, 1-24, 10-23
 - B-tree 構造, 10-29
 - LONG RAW データ型では禁止, 12-14
 - NULL, 10-9, 10-25, 10-38
 - ROWID, 10-30
 - 位置, 10-28
 - 一意, 10-24
 - カーディナリティ, 10-35
 - 概要, 1-24, 10-23
 - 拡張可能, 10-45
 - 格納形式, 10-29
 - キー, 10-25
 - UNIQUE キー制約, 25-10
 - 主キー制約, 25-12
 - キー圧縮, 10-31
 - 逆キー索引, 10-33
 - クラスタ, 10-53
 - 削除, 10-53
 - パーティション化できない, 11-2
 - 表と対比, 10-53

- グローバル・パーティション索引, 11-31
 - パーティションの管理, 11-32, 11-59
- コンボジット, 10-24
- 索引構成表, 10-39
 - 2次索引, 10-42
 - 論理 ROWID, 10-42, 12-19
- 索引使用禁止状態 (IU), 11-59
- 作成
 - 既存の索引を使用, 10-24
- 整合性制約の規定, 25-10, 25-12
- 説明, 1-24, 10-23
- ダイレクト・ロード・インサート後の再作成, 22-8
- ドメイン, 10-45
- ドメイン索引
 - 拡張可能な最適化, 21-18
 - ユーザー定義統計, 21-18
- 内部構造, 10-29
- 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, 11-35
- 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, 11-57
- パーティションの再作成, 11-59
- パーティション表, 10-39
- パーティション・プルーニング, 11-4
- パフォーマンス, 10-24
- パラレル DDL 記憶域, 23-34
- パラレルの索引スキャン, 23-5
- 非一意, 10-24
- ビットマップ索引, 10-34, 10-39
 - NULL, 10-9
 - パラレル問合せおよび DML, 10-35
- ビューでの使用, 10-15
- ファンクション, 10-26
 - DETERMINISTIC 関数, 20-8
 - DISABLED, 20-8
 - 依存性, 10-28, 20-7, 20-9
 - 権限, 10-28, 20-8
 - 最適化, 10-27
- 複合データ型, 10-45
- ブランチ・ブロック, 10-30
- リーフ・ブロック, 10-30
- 連結, 10-24
- ローカル索引, 11-28, 11-57
 - パーティションを並列に作成, 11-29
- ロギングなしモード, 22-7

- 索引構成表, 10-39
 - 2次索引, 10-42
 - 2次パーティション索引, 11-43
 - アプリケーション, 10-43
 - キー圧縮, 10-32, 10-41
 - キュー表, 18-15
 - 行オーバーフロー領域, 10-41
 - 再作成, 10-42
 - パーティション, 11-40
 - パラレル CREATE, 23-31
 - パラレル問合せ, 23-29
 - 利点, 10-41
 - 論理 ROWID, 10-42, 12-19
 - オンライン分析処理 (OLAP), 10-45
- 索引セグメント, 1-7, 4-17
- 索引タイプ, 10-46
- 索引の NOREVERSE 句, 10-33
- 索引の REVERSE 句, 10-33
- サブスクリプション
 - ルールベース, 18-8
- サブパーティション
 - 統計, 21-12
- サブパーティション・ロック
 - DML, 11-45
- サマリー, 10-18
- 参照
 - オブジェクト
 - 依存性, 20-2
 - 外部参照, 17-8, 17-20
 - キー, 1-56, 25-13
 - パーティション, 11-18
- 参照解除, 13-9
 - 暗黙の, 13-10
- 参照先がない REF, 13-9
- 参照整合性, 24-12, 25-13
 - CASCADE 規則, 25-3
 - PRIMARY KEY 制約, 25-11
 - RESTRICT 規則, 25-3
 - SET TO DEFAULT 規則, 25-3
 - SET TO NULL 規則, 25-3
 - 一部が NULL の外部キー, 25-16
 - 自己参照型制約, 25-15, 25-21
 - 例, 25-21

し

- シグネチャのチェック, 20-11
- 自己参照的なスタイルのメソッドの起動, 13-6
- システム・グローバル領域 (SGA), 7-2
 - REDO ログ・バッファ, 7-6, 16-6
 - いつ割り当てられるか, 7-2
 - 概要, 1-14, 7-2
 - 共有および書込み可能, 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, 5-2
 - 大規模プール, 7-12
 - 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, 26-19
 - ロールバック・セグメント, 16-5
 - 割当て, 5-5
- システム権限, 27-2
 - ADMIN OPTION, 27-3
 - 説明, 27-2
 - 付与と取消し, 27-3
- システム制御文, 1-51, 15-6
- システム変更番号 (SCN)
 - REDO ログ, 8-10
 - 決定される時期, 24-5
 - 定義, 16-6
 - トランザクションのコミット, 16-6
 - 読取り一貫性, 24-5, 24-6
- システム・モニター (SMON) プロセス, 8-11
 - Parallel Server, 8-11, 23-42
 - 一時セグメントのクリーン・アップ, 8-11
 - インスタンスのリカバリ, 29-4
 - 定義, 1-17, 8-11
 - ロールバック・トランザクション, 29-10
 - パラレル DML インスタンス・リカバリ, 23-42
 - パラレル DML システム・リカバリ, 23-42
- 事前書込み, 8-9
- 事前生成済み専用プロセス, 8-25
- 持続領域, 7-8
- 実行計画
 - EXPLAIN PLAN, 15-4
 - SQL の解析, 15-12
 - 位置, 7-8
- 概要, 21-2
- 実行順序, 21-6
- パーティションとパーティション・ビュー, 11-12
- 表示, 21-5
- プラン・スタビリティ, 21-7
- 実行者権限, 17-7
 - 提供されるパッケージ, 27-9
 - 名前解決, 17-20
 - プロシージャのセキュリティ, 27-8
- シノニム, 20-8
 - オブジェクトからの権限の継承, 27-4
 - 概要, 1-24
 - 拡張パーティション表名, 11-63
 - 使用方法, 10-22
 - 制約が間接的に影響する, 25-5
 - 説明, 10-22
 - データ・ディクショナリ・ビュー, 2-4
 - 名前解決, 30-42
 - パブリック, 10-22
 - プライベート, 10-22
- シャドウ・プロセス, 8-22
- 主キー, 1-56, 25-11
 - 定義, 25-3
 - 利点, 25-11
- 述語
 - 選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - ユーザー定義, 21-19
 - 動的
 - セキュリティ・ルール内, 27-24
 - パーティション・プルーニング, 11-4
 - 索引, 11-35
- 述語の選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - ユーザー定義選択性, 21-19
- 手動ロック, 1-31, 24-32
- 順序, 1-23, 10-21
 - CHECK 制約が禁止する, 25-21
 - 監査, 28-8
 - 数値の生成, 10-21
 - 数値の長さ, 10-21
 - 表からの独立, 10-21
- 障害, 29-2
 - REDO ログファイルのアーカイブ, 29-21
 - 「リカバリ」も参照
 - インスタンス, 1-44, 29-4
 - リカバリ, 5-8, 5-10, 29-4

- 説明, 1-43, 29-2
- 耐障害性, 29-27
- 提供されている保護対策, 29-6
- データベース・バッファ, 29-8
- 内部エラー
 - トレース・ファイルに記録, 8-15
- ネットワーク, 29-3
- 文障害とプロセス障害, 1-43, 29-3
- 文とプロセス, 8-12
- メディア, 1-44, 29-5
- ユーザー・エラー, 1-43, 29-2
- 障害リカバリ, 29-27, 29-28
- 使用済バッファ, 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, 29-14
- 情報検索 (IR) アプリケーション
 - 索引構成表, 10-44
- 初期化パラメータ
 - AQ_TM_PROCESS, 18-8, 18-9
 - BUFFER_POOL_KEEP, 7-5
 - BUFFER_POOL_RECYCLE, 7-5
 - COMPATIBLE, 3-11
 - DB_BLOCK_BUFFERS, 7-5, 7-13
 - DB_BLOCK_SIZE, 7-5, 7-13
 - DB_FILES, 7-16
 - DB_NAME, 29-23
 - DB_WRITER_PROCESSES, 1-16
 - DISTRIBUTED_TRANSACTIONS, 8-12
 - FAST_START_IO_TARGET, 29-14
 - HI_SHARED_MEMORY_ADDRESS, 7-14
 - JOB_QUEUE_PROCESSES, 18-12
 - LICENSE_MAX_SESSIONS, 26-21
 - LICENSE_SESSIONS_WARNING, 26-21
 - LOCK_SGA, 7-13, 7-17
 - LOG_ARCHIVE_MAX_PROCESSES, 1-18, 8-13, 29-20
 - LOG_ARCHIVE_START, 29-21
 - LOG_BUFFER, 7-6, 7-13
 - LOG_CHECKPOINT_INTERVAL, 29-14
 - LOG_CHECKPOINT_TIMEOUT, 29-14
 - MTS_MAX_SERVERS, 8-19, 8-20
 - MTS_SERVERS, 8-19
 - NLS_LANGUAGE, 11-19
 - NLS_NUMERIC_CHARACTERS, 12-9
 - NLS_SORT, 11-19
 - OPEN_CURSORS, 7-9, 15-7
 - OPEN_LINKS, 7-16

- OPTIMIZER_PERCENT_PARALLEL, 21-9
- PARALLEL_MAX_SERVERS, 23-8
- PARALLEL_MIN_PERCENT, 23-19
- PARALLEL_MIN_SERVERS, 23-7, 23-8
- PARALLEL_SERVER, 5-7
- REMOTE_DEPENDENCIES_MODE, 20-11
- ROLLBACK_SEGMENTS, 4-26
- SERVICE_NAMES, 6-6
- SHARED_MEMORY_ADDRESS, 7-14
- SHARED_POOL_SIZE, 7-6, 7-13
- SKIP_UNUSABLE_INDEXES, 20-8
- SORT_AREA_RETAINED_SIZE, 7-16
- SORT_AREA_SIZE, 4-18, 7-16
- SQL_TRACE, 8-15
- TRANSACTIONS, 4-26
- TRANSACTIONS_PER_ROLLBACK_SEGMENT, 4-26
- USE_INDIRECT_DATA_BUFFERS, 7-14
- 初期化パラメータ・ファイル, 5-4, 5-5
 - 起動, 5-5
 - 例, 5-4
- 初期即時制約, 25-24
- 初期遅延制約, 25-24
- ジョブ, 8-2
- ジョブ・キュー (SNP_n) プロセス, 1-18, 8-13
 - メッセージの伝播, 18-12
- 処理
 - DDL 文, 15-15
 - DML 文, 15-11
 - 概要, 15-9
 - 問合せ, 15-13
 - パラレル SQL, 23-2
 - 分散, 1-31
- 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, 30-33
- 親和性
 - パーティション, 23-49
 - パラレル DML, 23-50

す

- スキーマ, 26-2
 - OUTLN, 21-7
 - オブジェクト, 10-2
 - 定義, 26-2
 - 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, 1-36, 10-2
- スキーマ・オブジェクト, 10-1
 - INVALID 状態, 20-3
 - 依存性, 20-2
 - 存在しない他のオブジェクト, 20-9
 - トリガー管理, 19-22
 - ビュー, 10-16
 - 分散データベース, 20-12
 - 失われた権限に依存, 20-6
 - 概要, 1-6, 1-21, 10-2
 - 監査, 1-42, 28-8
 - グローバル名, 30-22
 - 権限, 27-3
 - 索引タイプ, 10-46
 - 作成
 - 表領域割当て制限が必要, 26-15
 - ディメンション, 10-20
 - 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, 26-14
 - ドメイン索引, 10-47
 - トリガーの依存性, 19-26
 - 取り消した表領域内, 26-15
 - 分散データベースの命名規則, 30-22
 - マテリアライズド・ビュー, 10-18
 - ユーザー定義オペレータ, 10-48
 - ユーザー定義型, 13-3
- スキーマ・オブジェクト権限, 27-3
 - DML 操作と DDL 操作, 27-5
 - 概要, 1-39
 - ビュー, 27-6
 - 付与と取消し, 27-4
- スキャン
 - 全表
 - LRU アルゴリズム, 7-4
 - パラレル問合せ, 23-5
 - 表スキャンと CACHE 句, 7-4
- スケラビリティ
 - クライアント / サーバー・アーキテクチャ, 6-4
 - バッチ・ジョブ, 23-39
 - パラレル DML, 23-38
 - パラレル SQL 実行, 23-2
- スタック領域, 7-15

- スタンバイ・データベース
 - 耐障害性, 29-27
 - マウント, 5-7
- ストアド・ファンクション, 17-2, 17-6
- ストアド・プロシージャ, 15-16, 17-2, 17-6
 - 「プロシージャ」も参照
 - コール, 15-19
 - トリガーと対比, 19-2
 - 変数と定数, 15-18
 - 無名ブロックと対比, 17-11
- スナップショット, 31-7
 - 更新可能, 31-8
 - スナップショット・ログ, 31-10
 - 配置テンプレート, 31-11
 - マテリアライズド・ビューと同義, 1-23
 - 読取り専用, 31-7
 - リフレッシュ, 8-13, 31-10
 - ジョブ・キュー・プロセス, 1-18, 8-13
- スナップショット・グループ, 31-5
- スナップショット読込み時間, 24-11
- スナップショット・ログ, 31-10
- スループット, 21-8
- 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, 1-27, 10-20
- 正規化されていない表, 1-27, 10-20
- 制御ファイル, 1-9, 29-22
 - 概要, 1-9, 29-22
 - 記録される変更内容, 29-23
 - 指定方法, 5-4
 - 多重化, 1-47, 29-23
 - チェックポイント, 29-23
 - データベースのマウントでの使用, 5-6
 - 内容, 29-22
 - バックアップ, 29-26
 - リカバリ, 1-47
- 制限
 - NESTED TABLE, 23-30
 - 関数のパラレル実行, 23-48
 - ダイレクト・ロード・インサート, 22-10, 23-45
 - パーティション
 - 拡張パーティション表名, 11-63

- データ型, 11-12, 11-21
- ビットマップ索引, 11-12
- パラレル DDL, 23-32
 - リモート・トランザクション, 23-28
- パラレル DML, 23-45
 - リモート・トランザクション, 23-28, 23-47
- 制限付き ROWID 形式, 12-17
- 制限モード
 - インスタンスの起動, 5-6
- 整合性制約, 25-2
 - 「制約」も参照
 - デフォルトの列値, 10-9
- 整合性ルール, 1-20
 - パラレル DML の制限事項, 23-46
- 制約, 1-55
 - CHECK, 25-21
 - ENABLE または DISABLE, 25-26
 - FOREIGN KEY, 1-56, 25-13
 - NOT NULL, 25-7, 25-11
 - PRIMARY KEY, 1-56, 25-11
 - UNIQUE キー, 1-56, 25-8
 - 一部が NULL, 25-11
 - VALIDATE または NOVALIDATE, 25-26
 - アプリケーションが違反を検出可能, 25-6
 - 一時的な使用禁止, 25-7
 - 違反した場合の処理, 25-5
 - 概要, 1-55
 - 規定のメカニズム, 25-21
 - 索引による規定, 10-25
 - PRIMARY KEY, 25-12
 - UNIQUE, 25-10
- 参照
 - 更新の効果, 25-16
 - 自己参照型, 25-15
- 代替処置, 25-5
- タイプのリスト, 1-56, 25-1
- 定義, 10-4
- デフォルト値, 25-24
- トリガーと対比, 19-5
- トリガーは違反できない, 19-22
- パフォーマンスに対する効果, 25-7
- パラレル CREATE TABLE, 23-25
- ビューでは定義不可, 10-13
- 評価されるタイミング, 10-9
- 変更, 25-27
- 西暦 2000 年, 12-11

- セーブポイント, 1-53, 16-7
 - 暗黙的, 16-4
 - 概要, 1-53
 - 説明, 16-7
 - ロールバック, 16-8
- セキュリティ, 1-39, 26-2
 - アプリケーションを使用して規定, 1-40
 - 監査, 28-2, 28-6
 - 監査データの削除, 2-5
 - 管理者権限, 5-3
 - 規定のメカニズム, 1-37
 - システム, 1-36, 2-3
 - 実行者権限, 17-7, 17-20
 - セキュリティ・ルール, 27-23
 - 説明, 1-36
 - 定義者権限, 17-7, 17-20
 - データ, 1-37
 - 動的な述語, 27-24
 - ドメイン, 1-38, 26-2
 - 任意アクセス制御, 1-37, 26-2
 - パスワード, 26-7
 - ビュー, 10-14
 - ビューによる強化, 27-6
 - ファイングレイン・アクセス・コントロール, 27-23
 - プログラム・インタフェースでの規定, 8-25
 - プロシージャによる強化, 27-8
 - メッセージ・キュー, 18-9
 - ユーザー・アクションの監査, 1-41
 - ルール
 - 実装, 27-25
- セキュリティ・ドメイン, 1-38, 26-2
 - 使用可能なロール, 27-19
 - 表領域の割当て制限, 26-14
- セグメント, 1-7, 4-16
 - 一時, 1-7, 4-17, 10-11
 - SMON によるクリーン・アップ, 8-11
 - 削除, 4-15
 - パラレル INSERT, 22-9
 - 必要な操作, 4-17
 - 表領域, 4-15, 4-18
 - 割当て, 4-17
 - 割当て制限は無視される, 26-15
- エクステントの割当て解除, 4-13
- 概要, 1-7, 4-16
- 索引, 4-17
- 定義, 4-3

- データ, 4-16
- 表
 - 高水位標, 22-3
- 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, 23-39
 - PGA 内のスタック領域, 7-15
 - 監査オプションが有効になる時期, 28-5
 - 軽量, 26-9
 - 時間制限, 26-19
 - 情報の格納場所, 7-15
 - 接続と対比, 8-4
 - 大規模プール内のメモリー割当て, 7-12
 - 定義, 8-4, 28-10
 - 同時セッションに対する制限, 1-41
 - ライセンスによる, 26-21
 - トランザクション分離レベル, 24-32
 - パッケージの状態, 20-6
 - 別監査, 28-10
 - ユーザー当りの制限, 26-19
 - リソース制限, 26-17
- セッション制御文, 1-51, 15-6
- 接続
 - 埋込み SQL, 15-6
 - 管理者権限での, 5-3
 - 監査レコード, 28-5
 - 制限, 5-6
 - セッションと対比, 8-4
 - 定義, 8-4
 - リスナー・プロセス, 6-6, 8-15
- 接続性, 1-2
- 接続プーリング, 26-9
- 専用サーバー, 8-22
 - 使用例, 8-24
 - 定義, 1-16
 - マルチスレッド・サーバーと対比, 8-16

そ

- 関連名
 - インライン・ビュー, 10-17
- 増分チェックポイント, 8-8
- 増分リフレッシュ, 10-19
- ソート・セグメント, 3-12
- ソート操作, 3-12
- ソート領域, 7-16

- 即時制約, 25-24
- 属性
 - オブジェクト型, 13-2, 13-4
- 阻止しているトランザクション, 24-11
- 阻止しているトランザクションを待機するか, 24-11
- 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, 7-18

た

- 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, 23-2
- 大規模プール, 7-12
 - 概要, 1-15
- 耐障害性, 29-27
- タイムスタンプのチェック, 20-11
- ダイレクト・ロード・インサート, 22-2
 - シリアル INSERT, 22-3
 - 制限, 22-10, 23-45
 - パラレル INSERT, 22-3
 - パラレル・ロードとパラレル INSERT の比較, 22-3
 - 領域管理, 22-9
 - ロギング・モード, 22-5
- 多重化
 - REDO ログ・ファイル, 1-45
 - 制御ファイル, 1-47, 29-23
 - リカバリ, 29-6
- タスク, 8-2
- 単一表ハッシュ・クラスタ, 10-60
- 断片化
 - パラレル DDL, 23-34

ち

- チェックポイント
 - DBWn プロセス, 8-8, 8-11
 - 制御ファイル, 29-23
 - 増分, 8-8
 - チェックポイント (CKPT) プロセス, 1-17, 8-11
 - 統計, 8-11
 - ファースト・スタート・チェックポイント, 29-14
- チェックポイント (CKPT) プロセス, 1-17, 8-11
- 遅延制約
 - 初期遅延または初期即時, 25-24
 - 遅延可能または遅延不可, 25-24

超並列処理 (MPP)

親和性, 23-6, 23-49, 23-50

パラレル SQL 実行, 23-2

複数の Oracle インスタンス, 5-3

つ

通信プロトコル, 6-5

て

定義者権限, 17-7

名前解決, 17-20

プロシージャのセキュリティ, 27-8

提供されるパッケージ, 17-17

実行者権限または定義者権限, 27-9

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, 3-7

ディクショナリ・キャッシュ・ロック, 24-31

停止, 5-10, 5-11

SGA の割当て解除, 7-2

異常, 5-6, 5-11

監査レコード, 28-5

ステップ, 5-10

ディスパッチャ・プロセスでは禁止, 8-20

定数

ストアド・プロシージャ内, 15-18

ディスク障害, 1-44, 29-5

ディスクの親和性

パーティション, 23-49

パラレル DML, 23-50

ディスクのストライプ化

親和性, 23-49

パーティション, 11-9

ディスクリット・トランザクションの管理

要約, 16-9

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 (Dnm) プロセス

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, 26-19

説明, 8-14

定義, 1-18

ネットワーク・プロトコル, 8-14

リスナー・プロセス, 8-15

ディメンション, 1-27, 10-20

階層, 1-27, 10-20

結合キー, 1-27, 10-20

正規化された表と正規化されていない表, 1-27, 10-20

属性, 1-27, 10-20

データ

アクセス, 1-49

制御, 26-2

セキュリティ・ドメイン, 26-2

同時, 24-2

ファイングレイン・アクセス・コントロール, 27-23

メッセージ・キュー, 18-9

一貫性

基礎となる原理, 24-15

手動ロック, 24-32

定義, 1-53

トランザクション・レベル, 24-6

読取り一貫性, 1-29

リピータブル・リード, 24-6

ロック, 24-3

ロック動作の例, 24-33

整合性, 1-28, 10-4, 25-2

2 フェーズ・コミット, 1-33

CHECK 制約, 25-21

概要, 1-55

型, 25-3

規定, 25-4, 25-5

参照, 25-3

パラレル DML の制限事項, 23-46

表への格納方法, 10-4

分散操作, 1-33

レプリケート, 1-34

ロック, 24-20

データ・ウェアハウス

階層, 1-27, 10-20

サマリー, 10-18

ディメンション・スキーマ・オブジェクト, 1-27, 10-20

ビットマップ索引, 10-34

表データのリフレッシュ, 23-38

マテリアライズド・ビュー, 10-18

- データ・オブジェクト番号
 - 拡張 ROWID, 12-16
- データ型, 12-2, 12-3
 - ANSI, 12-21
 - BOOLEAN, 12-2
 - CHAR, 12-5
 - DATE, 12-9
 - DB2, 12-21
 - LOB データ型, 12-11
 - BFILE, 12-13
 - BLOB, 12-12
 - CLOB および NCLOB, 12-13
 - デフォルトのロギング・モード, 22-8
 - LONG, 12-7
 - 記憶域, 10-8
 - NCHAR および NVARCHAR2, 12-6
 - NESTED TABLE, 10-10, 13-11
 - NUMBER, 12-7
 - PL/SQL, 12-2
 - RAW および LONG RAW, 12-14
 - ROWID, 12-15
 - SQL/DS, 12-21
 - VARCHAR, 12-6
 - 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
 - 配列型, 13-11
 - 表との関連, 10-3
 - 変換
 - Oracle 以外の型, 12-21
 - Oracle データ型から別の Oracle データ型へ, 12-22
 - プログラム・インタフェース, 8-25
 - マルチメディア, 13-3
 - 文字, 12-5, 12-13
 - ユーザー定義, 13-1, 13-3
 - 統計, 21-18
 - 要約, 12-3
 - 列, 1-21
- データ・セグメント, 1-7, 4-16, 10-4
- データ操作言語
 - 「データ操作言語」も参照
 - 監査, 28-7
 - 権限制御, 27-5
 - 取得されるロック, 24-26
 - 説明, 15-4
 - 定義, 1-50
 - トリガー, 19-3, 19-24
 - パーティション・ロック, 11-44
 - パラレル DML, 23-3, 23-36
 - パラレル DML のためのトランザクション・モデル, 23-40
 - 副問合せのシリアル化が可能な分離, 24-14
 - 分散トランザクションに許される文, 30-34
 - 文の処理, 15-11
- データ定義言語, 1-50
 - DBMS_SQL での解析, 15-20
 - PL/SQL への埋込み, 15-20
 - 「データ定義言語」も参照
 - 暗黙のコミット, 16-5
 - 監査, 28-7
 - 説明, 15-4
 - 定義, 1-50
 - パラレル DDL, 23-3
 - 文の処理, 15-15
 - ロールと権限, 27-21
 - ロック, 24-29
- データ・ディクショナリ
 - DUAL 表, 2-7
 - SYSTEM 表領域, 2-2, 2-5, 3-6
 - アクセス, 2-2
 - 依存性の追跡, 20-3
 - 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, 21-16
 - 使用方法, 2-3
 - 表と列の定義, 15-12
 - 所有者, 2-3
 - 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, 3-7
 - データ・ファイル, 3-6, 29-26
 - 統計, 21-16
 - パーティションの統計, 11-12
 - 動的パフォーマンス表, 2-7

- 内容, 2-2, 7-10
 - プロシージャ, 17-18
- バックアップ, 29-26
- パブリック・シノニム, 2-4
- ビューの接頭辞, 2-5
- プロシージャの妥当性, 17-19
- レプリケーション, 31-14
- ロック, 24-29
- 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, 1-53
 - 「読取り一貫性」も参照
 - マルチバージョンの一貫性モデル, 1-29
- データの変換
 - ANSI データ型, 12-21
 - SQL/DS および DB2 データ型, 12-22
 - プログラム・インタフェース, 8-25
- データ・ファイル
 - ROWID での表示, 12-16, 12-17
 - 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
 - 概要, 1-5, 1-8, 3-16
 - 制御ファイルに名前がある, 29-22
 - データ・ファイル 1, 3-6
 - SYSTEM 表領域, 3-6
 - バックアップ, 29-26
 - 内容, 3-16
 - バックアップ, 29-26
 - パラレル・リカバリ, 29-11
 - 表領域との関連, 3-2
 - 読取り専用, 3-11
 - リカバリ, 29-6
 - 読取り専用表領域, 3-12
 - リカバリ不能, 29-18
- データ・ブロック, 1-7, 4-2
 - ROWID での表示, 12-16, 12-17
 - 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, 10-7
 - 行の格納方法, 10-5
 - 行を挿入できる領域, 4-8
 - クラスタ化, 10-52
 - クラスタによる共有, 10-49
 - 形式, 4-3
 - ディスクへの書込み, 8-8
 - ハッシュ・キー, 10-58
 - 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・レベルのリカバリ, 29-15
 - メモリーにキャッシュ, 8-8
 - 読取り専用トランザクション, 24-33
- データ・ブロック内の空き領域の圧縮, 4-9
- データベース
 - アーカイブのモード, 29-18
 - アクセス制御
 - 概要, 1-49
 - セキュリティ・ドメイン, 26-2
 - パスワード暗号化, 26-7
 - オープン, 5-8
 - 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
 - 起動, 5-2
 - 強制実行, 5-11
 - クローズ, 5-10
 - インスタンスの異常終了, 5-10, 29-4
 - クローン・データベース, 5-7
 - 構成, 5-4
 - 構造
 - REDO ログ・ファイル, 1-9, 29-7
 - ROWID を使用して明確化, 12-17
 - エクステント, 1-6, 4-2, 4-10
 - スキーマ・オブジェクト, 1-6, 10-2
 - 制御ファイル, 1-9, 29-22
 - セグメント, 1-6, 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, 1-8, 3-1, 3-16
 - データ・ブロック, 1-6, 4-2, 4-3
 - 表領域, 1-5, 3-1, 3-5
 - 物理, 1-8
 - プロセス, 1-11, 1-15, 8-1
 - メモリー, 1-12, 7-1
 - 論理, 1-5, 4-1
 - 使用の制限, 26-17
 - スキーマが組み込まれている, 26-2
 - スケーラビリティ, 6-4, 23-2, 23-38
 - スタンバイ, 5-7, 29-27
 - 制御ファイルに格納される名前, 29-22
 - 定義, 1-5
 - 停止, 5-10
 - ディスマウント, 5-11
 - バックアップ, 1-47, 29-24

- 分散, 1-32
 - 2 フェーズ・コミット, 1-33
 - 概要, 1-31, 1-32
 - グローバル・データベース名の変更, 7-11
 - サイト自律性, 30-24
 - ノード, 1-32
 - 表複製, 1-34
- マウント, 5-6
- 読取り専用のオープン, 5-9
- リカバリ, 1-42, 29-2
- 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, 1-20
- データベース管理者 (DBA)
 - DBA ロール, 27-22
 - データ・ディクショナリ・ビュー, 2-6
 - 認証, 26-13
 - パスワード・ファイル, 26-14
 - バックアップおよびリカバリに対する責任, 29-2
- データベース構造
 - REDO ログ・ファイル, 1-9, 29-7
 - ROWID を使用して明確化, 12-17
 - エクステンツ, 1-6, 4-2, 4-10
 - スキーマ・オブジェクト, 1-6, 10-2
 - 制御ファイル, 1-9, 29-22
 - セグメント, 1-6, 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, 1-8, 3-1, 3-16
 - データ・ブロック, 1-6, 4-2, 4-3
 - 表領域, 1-5, 3-1, 3-5
 - プロセス, 1-11, 1-15, 8-1
 - メモリー, 1-12, 7-1
 - 論理, 1-5
- データベース全体のバックアップ, 1-47, 29-24
- データベース・トリガー, 19-1
 - 「トリガー」も参照
- データベースの構成
 - プロセスの構造, 8-2
- データベース・バッファ
 - 書込み, 8-8
 - キャッシュのサイズ, 7-5
 - クリーン, 8-8
 - 使用可能, 7-3
 - 使用済, 7-3, 8-8
 - 使用中, 7-3
 - 定義, 1-14, 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, 16-6
 - 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 (DBW_n) プロセス, 8-8
 - LRU アルゴリズム, 8-8
 - アクティブになるとき, 8-8
 - 概要, 1-16
 - 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
 - トレース・ファイル, 29-6
 - 複数の DBW_n プロセス, 8-8
 - メディア障害, 29-6
- データベース・リンク, 1-27
 - 解決, 30-36
 - 拡張パーティション表名, 11-63
 - 定義, 1-27
 - リモート・データベースに対するロール, 30-23
- データ変換
 - ANSI データ型, 12-21
 - SQL/DS および DB2 データ型, 12-22
 - プログラム・インタフェース, 8-25
- データ・モデル, 1-20
- データ・ロック
 - 存続期間, 24-16
 - 段階的な拡大, 24-17
 - 変換, 24-17
- データ・ファイル
 - SYSTEM 表領域, 3-6
 - データ・ディクショナリ, 3-6, 29-26
 - バックアップ, 29-26
- デッド・トランザクション, 29-4
 - ブロック・レベルのリカバリ, 29-15
- デッドロック
 - 回避, 24-19
 - 検出, 24-18
 - 人工, 8-19
 - 定義, 24-17
 - 分散トランザクション, 24-19
- デフォルト値, 10-9
 - 制約が与える影響, 10-9, 25-24
- 伝播スケジューリング機能, 18-12
- テンポラリ・ファイル, 3-17

と

問合せ

- DML, 15-4
- ROWID による表スキャンの平行化, 23-4
- SAMPLE 句
 - コストベース最適化, 21-17
- 一時セグメント, 4-18, 15-13
- 位置の透過性, 30-45
- インライン・ビュー, 10-17
- 記述フェーズ, 15-13
- 行のフェッチ, 15-13
- コンボジット索引, 10-24
- 処理, 15-13
- 定義フェーズ, 15-13
- デフォルト・ロック, 24-27
- トリガーの使用法, 19-24
- パーティションによって平行化される索引スキャン, 23-5
- 平行処理, 23-2
- 非定型, 23-32
- ビュー問合せとのマージ, 10-15
- ビューとして格納, 1-22, 10-12
- フェーズ, 24-5
- 分散, 30-34
- 分散またはリモート, 30-34
- 読取り一貫性, 1-30, 24-6
- 問合せ処理の記述フェーズ, 15-13
- 問合せの処理定義フェーズ, 15-13
- 問合せの行のフェッチ, 15-15
 - 埋込み SQL, 15-6
- 同一キー索引, 11-29, 11-33
- 同一行の書き込みが書き込みを阻止するか, 24-11
- 同一レベル・パーティション化, 11-23
 - 1 ディメンション, 11-23
 - LOB 列, 11-37
 - 索引構成表のオーバーフロー, 11-41, 11-42
 - 例, 11-24, 11-30, 11-31
 - レンジ・パーティション化, 11-23
 - ローカル索引, 11-29
- 透過的アプリケーション・フェイルオーバー, 29-15
- 同期レプリケーション, 31-16
- 統計
 - ANALYZE による, 21-15
 - B-tree 索引またはビットマップ索引から, 21-14
 - DBMS_STATS を使用した生成と管理, 21-13
 - エクスポートとインポート, 21-9

- オブティマイザでの使用, 21-8, 21-9
- 拡張可能な最適化, 21-18
- キューイング, 18-13
- 述語の選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - ユーザー定義, 21-19
- 推定, 21-15
 - Block サンプリング, 21-16
 - ROW サンプリング, 21-16
- 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, 21-12
- パーティション表とパーティション索引, 11-12
- ユーザー定義統計, 21-18

同時実行性

- 制限, 1-41, 22-10
 - データベース当り, 26-21
 - ユーザー当りの, 26-19
- 説明, 24-2
- ダイレクト・ロード・インサート, 22-10
- 定義, 1-28
- トランザクション, 24-16
- ロックを使用して規定, 1-30
- パーティションのメンテナンス, 11-48

動的 SQL

- DBMS_SQL パッケージ, 15-20
- 埋込み, 15-20
- 名前解決, 17-20

動的な述語

- セキュリティ・ルール内, 27-24

動的パーティション化, 23-6

動的パフォーマンス表 (V\$ 表), 2-7

ドメイン索引, 10-47

- 拡張可能な最適化, 21-18
- ユーザー定義統計, 21-18

ドライバ, 8-26

トランザクション, 1-51, 16-1

- アドバンスド・キューイング, 18-3
- アプリケーションの終了, 16-5
- インダウト

- 自動解決, 1-34, 5-9

- 自動的に解決, 16-8

- 手動による解決, 1-34

- 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, 16-5

- 概要, 1-51
- コミット, 1-52, 8-10, 16-4, 16-5
 - 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, 16-6
- 終了, 16-5
 - 一貫したデータ, 15-15
- 手動ロック, 24-32
- シリアル化が可能, 24-7
- 自律型, 16-10
 - PL/SQL ブロック内, 16-10
- セーブポイント, 1-53, 16-7
- 説明, 16-2
- 定義と制御, 15-15
- ディスクリット・トランザクション, 15-16, 16-9
- データ・ブロック内で使用される領域, 4-5
- デッド, 29-4
- デッドロック, 16-4, 24-17
- 同時実行性, 24-16
- トランザクション制御文, 15-5
- トランザクションの制御, 15-15
- トリガー, 19-24
- パラレル DML での 2 フェーズ・コミット, 23-41
- 非同期処理, 18-2
- ブロック・レベルのリカバリ, 24-21, 29-15
- 分散, 1-30
 - 2 フェーズ・コミット, 1-33, 16-8, 30-36
 - 自動解決, 8-12
 - デッドロック, 24-19
 - パラレル DDL の制限事項, 23-28
 - パラレル DML の制限事項, 23-28, 23-47
- 文レベルのロールバック, 16-4
- 読取り一貫性, 1-29, 24-6
- 読取り専用, 1-30, 24-7
 - ロールバック・セグメントには割り当てられない, 4-20
- リカバリ, 29-4
- ロールバック, 1-52, 16-6
 - オフライン表領域, 4-30
 - 部分的な, 16-8
 - 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, 24-10, 24-11
- トランザクション制御文, 1-50, 15-5
 - 自律型 PL/SQL ブロック内, 16-11
- トランザクションのコミット
 - 概要, 1-52
 - グループ・コミット, 8-10
 - 高速コミット, 8-10
 - 実現, 8-10
 - 定義, 16-2
 - パラレル DML, 23-41
- トランザクションのロールバック, 29-4
- トランザクション表, 4-20
 - リカバリ時にリセット, 8-12
- トランスポートابل表領域, 3-13
- トリガー, 19-1, 20-8
 - AFTER トリガー, 19-10
 - BEFORE トリガー, 19-10
 - INSTEAD OF トリガー, 19-13
 - オブジェクト・ビュー, 14-5
 - INVALID 状態, 20-3, 20-6
- Java, 19-8
- Oracle Forms トリガーと対比, 19-3
- UNKNOWN では起動されない, 19-8
- アクション, 19-8
 - タイミング, 19-10
- 依存性の管理, 19-26, 20-6
 - 使用可能なトリガー, 19-22
- イベント, 19-7
- 概要, 19-2
- 各部分, 19-6
- 監査, 28-8
- 記憶域, 19-26
- 起動 (実行), 19-2, 19-26
 - 実行されるステップ, 19-22
 - タイミング, 19-23
 - 必要な権限, 19-26
- 行, 19-9
- 共有 SQL 領域, 7-10
- 実行する権限, 27-8
 - ロール, 27-20
- 使用可能または使用禁止, 19-22
- 使用方法, 19-3
- スキーマ・オブジェクトの依存性, 19-22, 19-26
- 制限, 19-8, 23-47
 - ダイレクト・ロード・インサート, 22-10
 - パラレル DML, 23-45

制約が適用される, 19-22
制約と対比, 19-5
タイプ, 19-9
データ・アクセス, 19-24
データ整合性の規定, 25-5
パブリッシュ / サブスクライブのサポート, 19-19
ビューでは定義不可, 10-13
複数トリガーの起動順序, 19-23
プログラム・ユニット, 1-54
プロシージャと対比, 19-2
文, 19-9
例, 19-12, 19-14, 19-25
連鎖, 19-4
取消し, 1-7
「ロールバック」も参照
トレース・ファイル, 8-15
 ARC*n* トレース・ファイル, 29-21
 DBW*n* トレース・ファイル, 29-6
 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

な

内容を保証しない書込み, 24-11
内容を保証しない読込み, 24-3, 24-11
名前解決
 分散データベース, 30-22
名前付きユーザー・ライセンス, 26-22

に

任意アクセス制御, 1-37, 26-2
認証
 Oracle, 26-7
 オペレーティング・システム, 26-4
 公開鍵インフラストラクチャ, 26-5
 説明, 26-3
 データベース管理者, 26-13
 ネットワーク, 26-4
 複数層, 26-9
 リモート, 26-7
認証局, 26-5

ね

ネットワーク
 2タスク・モード, 8-23

Net8, 6-4
Oracle の使用, 1-18, 1-34
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, 29-3
通信プロトコル, 6-5, 8-26
ディスパッチャ・プロセス, 8-14, 8-16
ドライバ, 8-26
ネットワーク認証サービス, 26-4
分散データベースの使用, 30-2
リスナー・プロセス, 6-6, 8-15
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, 8-15, 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, 8-21

の

ノード
 パラレル・サーバーでのディスク親和性, 23-49
 分散データベース, 1-32

は

パーティション, 11-2, 11-11
 DATE データ型, 11-12, 11-21
 DML パーティション・ロック, 11-44
 EXCHANGE PARTITION, 11-11
 LONG および LONG RAW の制限事項, 11-12
 OLTP データベース, 11-7
 VLDB, 11-5
拡張パーティション表名, 11-62
グローバル索引, 11-31, 11-59
索引のパーティション化, 11-28, 11-35
実行計画, 11-12
親和性, 23-49
制限
 拡張パーティション表名, 11-63
 データ型, 11-12, 11-21
 ビットマップ索引, 11-12
セグメント, 4-16, 4-17
同一キー索引, 11-29
同一レベル・パーティション化, 11-23
 1 デイメンション, 11-23
 LOB 列, 11-37
索引構成表のオーバーフロー, 11-41, 11-42

- 例, 11-24, 11-30, 11-31
- レンジ・パーティション化, 11-23
- ローカル索引, 11-29
- 統計, 11-12, 21-12
- 同時実行のメンテナンス操作, 11-48
- 動的パーティション化, 23-6
- パーティション化キー, 11-13, 11-18
- パーティション化の基本的なモデル, 11-11
- 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, 11-18
- パーティションの透過性, 11-10
- パーティション・バウンド, 11-19
- パーティション・プルーニング, 11-4
 - DATE データ型, 11-21
 - 索引, 11-35
 - ディスクのストライプ化, 23-49
 - ブロック範囲によるパラレル化, 23-4
- パーティション名, 11-17
- パーティション・ワイズ・ジョイン, 11-5
- ハッシュ・パーティション化, 11-15
- パラレル DDL, 23-31
- パラレル化のルール, 23-25, 23-27
- パラレル問合せ, 23-4
- ビットマップ索引, 10-39
- 非同一キー索引, 11-30, 11-34
- 表のパーティション化, 11-26
- 複数列からなるキー, 11-21
- 物理属性, 11-26, 11-36
- マージ, 11-14
- マテリアライズド・ビュー, 10-19, 11-2
- メンテナンス操作, 11-46
- 利点, 11-5, 11-7
- レンジ・パーティション化, 11-13
 - ディスクのストライプ化, 23-49
- ローカル索引, 11-28, 11-57
 - 並列に作成, 11-29
- ロギングなしモード, 22-7
- パーティション化
 - LOB
 - DML ロック, 11-45
 - メンテナンス操作, 11-55
 - LOB 列を持つ表, 11-37
 - 列, 11-13
- パーティションのマージ, 11-14
- パーティション・ビュー, 11-11
- パーティション・プルーニング, 11-4, 23-4, 23-49
 - DATE データ型, 11-21
 - EXPLAIN PLAN, 11-21
 - 索引, 11-35
 - 索引パーティション, 11-4
- 排他モード, 4-27
- 排他ロック
 - RX ロック, 24-24
 - 行ロック (TX), 24-21
 - 表ロック (TM), 24-22
- 配置テンプレート, 31-11
 - インスタンス化, 31-11
- バイナリ・データ
 - BFILE, 12-13
 - BLOB, 12-12
 - RAW および LONG RAW, 12-14
- 配列
 - VARRAY のサイズ, 13-11
 - 可変 (VARRAY), 13-11
- 配列処理, 15-15
- バインド変数
 - ユーザー定義型, 13-13
- パスワード
 - アカウントのロック, 26-7
 - 暗号化, 26-7
 - 管理者権限, 5-3
 - 時間切れ, 26-8
 - 接続, 8-4
 - データベース・ユーザーの認証, 26-7
 - パスワードの再使用, 26-8
 - パスワード・ファイル, 26-14
 - パスワードを指定しない接続, 26-4
 - 複雑度の検証, 26-8
 - ロールでの使用, 1-40
- バックアップ
 - Export を使用した補助, 29-27
 - Recovery Manager, 1-49, 29-16
 - 概要, 1-42, 29-24
 - 制御ファイル, 29-26
 - タイプ, 1-47
 - データ・ファイル, 29-26
 - データベース全体のバックアップ, 1-47, 29-24
 - パラレル, 29-17
 - 部分, 1-47, 29-26
 - 読取り専用表領域, 29-27
- バックエンド, 6-2

バックグラウンド・プロセス, 1-16, 8-5

「プロセス」も参照

概要, 1-16

図, 8-6

説明, 8-5

トレース・ファイル, 8-15

パッケージ, 17-4, 17-12

OUTLN_PKG, 21-7

格納, 17-17

監査, 28-8

キューイング, 18-6

共有 SQL 領域, 7-10

権限

構成体ごとに分割, 27-10

実行, 27-7, 27-10

実行, 15-17, 17-19

セッションの状態, 20-6

妥当性, 17-19

提供されるパッケージ, 17-17

実行者権限または定義者権限, 27-9

動的 SQL, 15-20

パブリック, 17-16

プライベート, 17-16

プログラム・ユニット, 1-54

利点, 17-16

例, 17-12, 27-10, 27-11

ロック用, 24-40

発行

DDL 文, 19-21

DML 文, 19-21

システム・イベント

起動と停止, 19-20

サーバー・エラー, 19-20

トリガーの使用, 19-19

ログオンおよびログオフ・イベント, 19-20

ハッシュ・クラスタ, 1-27, 10-53

概要, 1-27

記憶域, 10-54

索引と対比, 10-54

衝突の解決, 10-56

単一表ハッシュ・クラスタ, 10-60

領域の割当て, 10-58

ルート・ブロック, 10-58

発注の例

オブジェクト型, 13-2, 13-4

バッファ

REDO ログ, 1-14, 7-6

データベース・バッファ・キャッシュ

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,
29-14

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), 7-14

データベース, 1-14, 7-3, 8-8

複数のバッファ・プール, 7-5

バッファ・プール, 7-5

パフォーマンス

DSS データベース, 11-8, 23-38

I/O, 11-9

Oracle Parallel Server と DML ロック, 11-46

SGA のサイズ, 7-12

クラスタ, 10-51

グループ・コミット, 8-10

向上させる構造体, 1-24, 1-25

索引作成, 10-24

実行計画の表示, 21-5

制約が与える影響, 25-7

ソート操作, 3-12

同一キー索引と非同ーキー索引, 11-34

動的パフォーマンス表 (V\$), 2-7

パーティション, 11-8

パッケージ, 17-16

パラレル・リカバリ, 29-11

リカバリ, 29-14

リソース制限, 26-17

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, 18-13

イベントの発行, 19-19

トリガー, 19-19

リスニング機能, 18-14

ルールベースのサブスクライバ, 18-8

パブリッシュ / サブスクライブ・モデル, 18-5

パブリッシュ / サブスクライブのサポート

メッセージの伝播, 18-12

パラメータ

各国語サポート, 5-5

記憶域, 4-5, 4-11

初期化, 5-4

「初期化パラメータ」も参照

ロック動作, 24-19

パラレル DDL, 23-31

エクステンツの割当て, 23-34

関数, 23-48

制限

- LOB, 23-32
 - オブジェクト型, 23-30, 23-32
- パーティション表とパーティション索引, 23-31
 - ローカル索引の作成, 11-29
- パラレル化のタイプ, 23-3
- パラレル化ルール, 23-20
- パラレル DELETE, 23-21, 23-22
- パラレル DML, 23-36
 - PARALLEL DML の使用可能化, 23-39
 - アプリケーション, 23-38
 - 関数, 23-48
 - 制限, 23-45
 - オブジェクト型, 23-30, 23-45
 - リモート・トランザクション, 23-47
 - トランザクション・モデル, 23-40
- パラレル化のタイプ, 23-3
- パラレル化ルール, 23-20
- ビットマップ索引, 10-35
- 並列度, 23-20, 23-23
- リカバリ, 23-41
- リソースのロックとエンキュー, 23-43
- ロールバック・セグメント, 23-41

パラレル SQL, 23-2

- Parallel Server, 23-1
- 「パラレル実行」も参照
- インスタンス・グループ, 23-19
- オブティマイザ, 23-10
- コーディネータ・プロセス, 23-6
 - ダイレクト・ロード・インサート, 22-3
- サーバー・プロセス, 23-6
 - ダイレクト・ロード・インサート, 22-3, 22-8, 22-9
- サマリー表またはロールアップ表, 23-32
- パラレル化ルール, 23-20
- パラレル実行サーバーの数, 23-7
- パラレル実行サーバーへの行の割当て, 23-11
- 並列度, 23-16
- マルチスレッド・サーバー, 23-8

パラレル UPDATE, 23-21, 23-22

パラレル化

- 程度, 23-15

パラレル実行, 23-2

- 「パラレル SQL」も参照
- インターオペレータ並行性, 23-13
- イントラオペレータ並行性, 23-13
- コーディネータ, 22-3, 23-6
 - ダイレクト・ロード・インサート, 22-3

- サーバー, 22-3, 23-6
 - 一時セグメント, 22-9
- 索引メンテナンス, 22-8
 - ダイレクト・ロード・インサート
 - 一時セグメント, 22-9
 - 索引メンテナンス, 22-8
- パーティション表とパーティション索引, 23-4
 - フル・テーブル・スキャン, 23-5

パラレル実行サーバー

- ダイレクト・ロード・インサート, 22-3

パラレル操作のインスタンス・グループ, 23-19

パラレル問合せ, 23-28

- オブジェクト型, 23-30
 - 制限, 23-30
- 関数, 23-48
- 索引構成表, 23-29
- パラレル化ルール, 23-20
- ビットマップ索引, 10-35

パラレル・バックアップ操作, 29-17

パラレル・リカバリ, 29-11, 29-17

ひ

非一意索引, 10-24

非永続キュー, 18-12

比較メソッド, 13-6

非コミット読み込み, 24-3

ビジネス・ルール

- アプリケーション・コードで規定, 25-5
- ストアド・プロシージャを使用して規定, 25-5
- 制約を使用して規定, 1-55, 25-1
- 利点, 25-5

ヒストグラム, 21-10

ビットマップ索引, 10-34

- NULL, 10-9, 10-38
- カーディナリティ, 10-35
- パーティション表, 11-12
- パラレル問合せおよび DML, 10-35

ビットマップによる表領域管理, 3-8

- 一時表領域, 3-13

非同一キー索引, 11-30, 11-34

- グローバル・パーティション索引, 11-32

非同期 I/O

- パラレル・リカバリ, 29-12

非同期処理, 18-2

ビュー, 1-22, 10-12

- INSTEAD OF トリガー, 19-13

- INVALID 状態, 20-3
 - NLS パラメータ, 10-15
 - SQL 関数, 10-15
 - 依存性の状態, 20-5
 - インライン・ビュー, 10-17
 - オブジェクト・ビュー, 10-17, 14-1
 - 更新可能性, 14-5
 - 概要, 1-22, 10-12
 - 格納方法, 10-13
 - 監査, 28-8
 - 疑似列, 19-14
 - 権限, 27-6
 - 更新可能性, 10-16, 14-5, 19-14
 - 固定ビュー, 2-7
 - コンパイルの前提条件, 20-5
 - 索引, 10-15
 - 式を含む, 19-14
 - 使用方法, 10-14
 - スキーマ・オブジェクトの依存性, 10-16, 20-4, 20-8
 - 制約が間接的に影響する, 25-5
 - 制約とトリガーは定義不可, 10-13
 - セキュリティ・アプリケーション, 27-6
 - 定義の展開, 20-5
 - データ・ディクショナリ
 - 更新可能な列, 10-16
 - ユーザー・アクセス可能ビュー, 2-3
 - 統計, 21-16
 - 名前解決, 30-42
 - パーティションの統計, 11-12
 - パーティション・ビュー, 11-11
 - ヒストグラム, 21-12
 - ベース表, 1-22
 - ベース表の変更, 20-5
 - 変更, 19-13
 - 変更可能, 19-14
 - 本質的に変更可能, 19-14
 - マテリアライズド・ビュー, 1-23, 10-18
 - スナップショットと同義, 1-23
 - 列の最大数, 10-13
- 表
- DUAL, 2-7
 - LOB 列
 - パーティション化, 11-37
 - NESTED TABLE, 10-10, 13-11
 - PARTITION 句, 11-62
 - SUBPARTITION 句, 11-62
 - VALIDATE または NOVALIDATE 制約, 25-26
 - 依存ビューに対する影響, 20-5
 - 一時, 10-11
 - セグメント, 4-18
 - オブジェクト表, 13-2, 13-7
 - 仮想, 14-2
 - 概要, 1-21, 10-3
 - 拡張パーティション表名, 11-62
 - 仮想またはビュー, 1-22
 - 監査, 11-61, 28-8
 - キュー表, 18-6, 18-15
 - クラスタ化, 10-49
 - 権限, 27-5
 - 索引, 10-23
 - 索引構成
 - キー圧縮, 10-32, 10-41
 - 索引構成表, 10-39
 - 論理 ROWID, 10-42, 12-19
 - サマリー表またはロールアップ表, 23-32
 - 使用するトリガー, 19-2
 - 正規化された表と正規化されていない表, 1-27, 10-20
 - 整合性制約, 25-2, 25-5
 - 整合性制約を含む, 1-56
 - 制約を使用可能または使用禁止にする, 25-26
 - 単一表ハッシュ・クラスタ, 10-60
 - ディレクトリ, 4-4
 - データ・ウェアハウスでのリフレッシュ, 23-38
 - データの格納方法, 10-4
 - 動的パーティション化, 23-6
 - 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
 - ハッシュ, 10-58
 - パラレル DDL 記憶域, 23-34
 - パラレル作成, 23-32
 - パラレル実行での STORAGE 句, 23-34
 - パラレルの表スキャン, 23-4
 - ビューでの提示, 10-12
 - 表領域に含まれる, 10-5
 - 表領域の指定, 10-5
 - フル・テーブル・スキャンとバッファ・キャッシュ, 7-4
 - ベース表, 1-22
 - データ・ディクショナリでの使用, 2-2
 - ビューとの関連, 10-13
 - 領域割当ての制御, 10-4, 22-9
 - 履歴データ, 23-39

- 列の最大数, 10-13
- レプリケート, 1-34
- ロギングなしモード, 22-7
- ロック, 11-44, 24-21, 24-24, 24-25
- 表の更新
 - 親キー, 25-18, 25-19
- 表領域, 3-5
 - 「SYSTEM 表領域」を参照
 - 一時, 1-40, 3-12
 - ユーザーのデフォルト, 26-15
 - 一時セグメントに使用, 4-15, 4-18
 - オブジェクト作成のデフォルト, 1-40, 26-14
 - オフライン, 1-6, 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, 3-10
 - 読取り専用にできない, 3-11
 - オンライン, 1-6, 3-9, 3-17
 - 概要, 1-5, 3-5
 - サイズ, 3-4
 - スキーマと対比, 10-2
 - 説明, 3-5
 - 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, 3-7
 - データ・ファイルとの関連, 3-2
 - トランспортаブル, 3-13
 - 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, 26-15
 - 読取り専用, 3-11
 - オブジェクトの削除, 3-12
 - 推移モード, 3-11
 - 読取り専用推移モード, 3-11
 - 領域割当て, 3-7
 - ローカルに管理される, 3-8
 - 一時表領域, 3-13
 - ロギングなしモード, 22-7
 - ロック, 24-31
 - 割当て制限, 1-40, 1-41, 26-14, 26-15
 - 制限ありと制限なし, 26-15
 - デフォルトなし, 26-15
- 表領域の Point-in-Time リカバリ
 - クローン・データベース, 5-7
- 非リピータブル・リード, 24-3, 24-11
- ヒント
 - PARALLEL, 23-17
 - PARALLEL_INDEX, 23-17
 - 拡張可能な最適化, 21-18

ふ

- ファースト・スタート
 - オン・デマンド・ロールバック, 29-10
 - チェックポイント, 29-14
 - パラレル・ロールバック, 29-15
 - リカバリ, 29-14
- 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, 1-5, 1-8, 29-6
 - 「制御ファイル」, 「データ・ファイル」, 「REDO ログ・ファイル」も参照
 - 初期化パラメータ, 5-4, 5-5
 - パスワード, 26-14
 - 管理者権限, 5-3
 - ファイル管理ロック, 24-31
 - ファイングレイン・アクセス・コントロール, 27-23
 - ファジー読込み, 24-3
 - ファンクション
 - PL/SQL
 - プロシージャと対比, 1-54
- ファンクション索引, 10-26
 - DISABLED, 20-8
 - UNUSABLE, 20-8
 - 依存性, 10-28, 20-7
 - 権限, 10-28, 20-8
- 副問合せ, 15-13
 - CHECK 制約が禁止する, 25-21
 - DDL 文, 23-32
 - DML 文
 - シリアル化が可能な分離, 24-14
 - 「問合せ」も参照
 - インライン・ビュー, 10-17
 - 問合せ処理, 15-13
 - リモート更新, 30-34
- 不整なパラレル DML 作業負荷, 23-19
- 物理データベース構造, 1-8
 - REDO ログ・ファイル, 1-9, 29-7
 - 制御ファイル, 1-9, 29-22
 - データ・ファイル, 1-8, 3-16
- 部分バックアップ, 29-26
- 付与
 - 権限とロール, 27-3
- プライベート SQL 領域
 - カーソル, 7-9
 - 持続領域, 7-8

- 説明, 7-8
 - どのように管理されるか, 7-9
 - ランタイム領域, 7-8
- プライベート・ロールバック・セグメント, 4-25
- プランチ・ブロック, 10-30
- プリコンパイラ
 - FIPS フラガー, 15-6
 - 埋込み SQL, 15-6
 - カーソル, 15-12
 - ストアド・プロシージャ, 15-19
 - バインド変数, 15-14
 - 無名ブロック, 15-18
- フル・テーブル・スキャン
 - LRU アルゴリズム, 7-4
 - パラレル実行, 23-5, 23-6
- プロキシ, 26-9
- プログラム・インタフェース, 8-25
 - 2 タスク・モード, 8-23
 - Oracle 側 (OPI), 8-26
 - 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), 1-15, 7-14
 - サイズ, 7-16
 - 内容, 7-14
 - 非共有と書込み可能, 7-14
 - 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, 1-23, 15-16, 17-2
 - 共有プール, 7-10
 - コンパイルの前提条件, 20-5
- プロシージャ, 15-16, 17-1, 17-6, 20-8
 - INVALID 状態, 20-3, 20-6
 - 依存性の追跡, 20-6
 - カーソル, 15-19
 - 外部参照, 17-8, 17-20
 - 外部プロシージャ, 15-21, 17-12
 - 格納, 17-17
 - 監査, 28-8
 - 共有 SQL 領域, 7-10
 - 権限
 - 作成または変更, 27-9
 - 実行, 27-7
 - パッケージ内での実行, 27-10
 - コンパイルの前提条件, 20-5
 - 実行, 15-17, 17-19
 - 実行者権限, 17-7, 27-8
 - 使用されるロール, 27-21
 - 提供されるパッケージ, 27-9
 - ストアド・プロシージャ, 15-16, 15-19, 17-2
 - セキュリティの強化, 17-9, 27-8
 - 妥当性, 17-19
 - 定義者権限, 17-7, 27-8
 - ロールは使用禁止, 27-20
 - 提供されるパッケージ, 17-17
 - 実行者権限または定義者権限, 27-9
 - トリガー, 19-2
 - ファンクションと対比, 1-54, 17-2
 - 無名ブロックと対比, 17-11
 - 利点, 17-8
 - リモート・コール, 30-46
 - 例, 17-6, 27-10, 27-11
- プロシージャ型レプリケーション, 31-16
 - 競合の検出, 31-17
 - ラッパー, 31-16
- プロシージャの名前解決, 17-20
- プロセス, 8-2
 - Oracle, 1-15, 8-5
 - アーカイバ (ARC*n*), 1-17, 8-12, 29-20
 - 概要, 1-15
 - キュー・モニター (QMN*n*), 1-18, 8-14, 18-8
 - 構造, 8-2
 - サーバー, 1-16, 1-31, 8-5
 - 共有, 8-14, 8-15, 8-19
 - 専用, 8-22
 - システム・モニター (SMON), 1-17, 8-11
 - 事前生成済み, 8-25
 - シャドウ, 8-22
 - 障害, 29-3
 - ジョブ・キュー (SNP*n*), 1-18, 8-13
 - メッセージの伝播, 18-12
 - 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), 1-17, 8-11
 - デイスパッチャ (Dnn*m*), 1-18, 8-14
 - データベース・ライター (DBW*n*), 1-16, 8-8
 - トレース・ファイル, 8-15
 - バックグラウンド, 1-16, 8-5
 - 図, 8-6
 - パラレル実行コーディネータ, 23-6
 - ダイレクト・ロード・インサート, 22-3
 - パラレル実行サーバー, 23-6
 - ダイレクト・ロード・インサート, 22-3, 22-8, 22-9

プロセス・モニター (PMON), 1-17, 8-12
 ブロック・サーバー (BSP), 24-7
 分散トランザクションの解決, 8-12
 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, 1-15, 8-4
 PGA の割当て, 7-14
 サーバー・プロセスの共有, 8-14, 8-15
 手動アーカイブ, 29-22
 障害からのリカバリ, 8-12
 リカバラ (RECO), 1-18, 8-12
 インダウト・トランザクション, 1-34
 リカバリ時, 29-12
 リスナー, 6-6, 8-15
 共有サーバー, 8-16
 ログ・ライター (LGWR), 1-17, 8-9
 ロック (LCK0), 1-18, 8-13
 プロセス・グローバル領域 (PGA), 7-14
 「プログラム・グローバル領域」も参照
 プロセス・モニター (PMON) プロセス
 説明, 1-17, 8-12
 タイムアウト・セッションのクリーン・アップ,
 26-19
 ネットワーク障害, 29-3
 パラレル DML プロセス・リカバリ, 23-42
 プロセス障害, 29-3
 ブロック
 サンプリング, 21-16
 データベース, 4-3
 ブロック・レベルのリカバリ, 29-15
 無名, 15-16, 17-11
 ブロック・サーバー (BSP) プロセス, 24-7
 ブロック・レベルのリカバリ, 24-21, 29-15
 プロファイル
 概要, 1-41
 使用する場合, 26-20
 パスワード管理, 26-8
 フロントエンド, 6-2
 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, 1-31,
 6-2
 説明, 1-31, 6-2
 データ操作言語, 15-11
 マテリアライズド・ビュー (スナップショット),
 10-18
 分散スキーマ管理
 レプリケーション
 分散スキーマ管理, 31-14
 分散データベース
 2 フェーズ・コミット, 1-33
 依存スキーマ・オブジェクト, 20-10
 概要, 1-32, 30-2
 監査, 28-6
 管理ツール, 30-31
 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, 30-22
 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, 30-24
 ジョブ・キュー (SNPn) プロセス, 1-18, 8-13
 デッドロック, 24-19
 透過性, 30-44
 ノード, 30-7
 表複製, 1-34
 分散更新, 30-34
 分散問合せ, 30-34
 メッセージの伝播, 18-12
 リカバラ (RECO) プロセス, 8-12
 リモート依存性, 20-11
 リモート問合せと更新, 30-34
 分散問合せの最適化, 30-47
 分散トランザクション
 2 フェーズ・コミット, 1-33, 16-8
 定義, 30-35
 ノードへの文のルーティング, 15-12
 パラレル DDL の制限事項, 23-28
 パラレル DML の制限事項, 23-28, 23-47
 文トリガー, 19-9
 「トリガー」も参照
 起動されるタイミング, 19-23
 説明, 19-9
 分離レベル
 コミット読込み, 24-8
 設定, 24-8, 24-32
 選択, 24-13
 文レベルの読取り一貫性, 24-6

へ

- 並列度, 23-15, 23-20, 23-23
 - 問合せ操作間, 23-13
 - パラレル SQL, 23-7, 23-16
- ページ, 4-2
- ベース表, 1-22
 - 「ビュー」も参照
 - データ・ディクショナリ, 2-2
- ヘッダー
 - 行断片, 10-5
 - データ・ブロック, 4-4
- 別の行の書き込みが書き込みを阻止するか, 24-11
- 別名
 - 副問合せを修飾 (インライン・ビュー), 10-17
- 変更エラーとトリガー, 19-24
- 変数
 - 埋込み SQL, 15-6
 - オブジェクト変数, 14-4
 - ストアド・プロシージャ内, 15-18
 - バインド変数
 - ユーザー定義型, 13-13

ま

- マスター・グループ, 31-5
- マスター・サイト, 31-5
- マスター定義サイト, 31-5
- マップ・メソッド, 1-55, 13-7
- マテリアライズド・ビュー, 10-18
 - エクステンツの割当て解除, 4-15
 - 概要, 1-23
 - スナップショットと同義, 1-23
 - パーティション化, 10-19, 11-2
 - マテリアライズド・ビュー・ログ, 10-19
 - リフレッシュ, 10-19
- マテリアライズド・ビュー・ログ, 10-19
- 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, 8-15, 8-19
 - 限定的運用, 8-20
 - サーバー・プロセス, 1-16, 8-19
 - 使用例, 8-20
 - 人工デッドロック, 8-19
 - セッション情報, 7-15
 - 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, 7-12
- ディスパッチャ・プロセス, 1-18, 8-14
- パラレル SQL 実行, 23-8
- 必要なプロセス, 8-16
- プライベート SQL 領域, 7-9
 - ソート領域, 7-16
- プライベート SQL 領域の制限, 26-19
- マルチバージョン同時実行性制御, 24-6
- マルチバージョンの一貫性モデル, 1-29
- マルチ・プロセス・システム (マルチユーザー・システム), 8-2
- マルチブロック WRITE, 8-8
- マルチマスター・レプリケーション, 31-5
- 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, 15-16, 17-11
 - アプリケーション, 15-18
 - ストアド・プロシージャと対比, 17-11
 - ストアド・プロシージャのコール, 15-20
 - 動的 SQL, 15-20
 - パフォーマンス, 17-11

め

- 明示的ロック, 24-32
- メソッド
 - 権限, 27-12
 - コンストラクタ・メソッド, 13-6
 - 比較メソッド, 13-6
- メッセージ・キューイング, 18-2
 - キュー表, 18-6
 - キュー表のエクスポート, 18-15
 - キュー・モニター・プロセス, 1-18, 8-14, 18-8
 - 実行の時間枠, 18-9
 - 定期的な統計, 18-14
 - パブリッシュ / サブスクライブのサポート, 18-13
 - イベントの発行, 19-19
 - メッセージ, 18-6
 - リモート・データベース, 18-12
 - レシビエント, 18-7
 - サブスクライバ・リスト, 18-7
 - ルールベースのサブスクリプション, 18-8
- メッセージ「スナップショットが古すぎます」, 24-5
- メディア障害, 1-44, 29-5

メモリー

- SQL 文のための割当て, 7-11
 - 「システム・グローバル領域」も参照
- カーソル (文ハンドル), 1-15
- 拡張バッファ・キャッシュ (32 ビット), 7-14
- 仮想, 7-17
- 共有 SQL 領域, 7-8
- 構造, 7-2
- 構造の概要, 1-11
- 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, 7-14
 - 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, 17-9, 17-18
- 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, 29-18
- 表ロック, 24-22

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, 10-18

ゆ

有効範囲付き REF, 13-9

ユーザー, 26-2

- PUBLIC ユーザー・グループ, 26-16, 27-20
- アクセス権, 26-2
- 一時表領域, 1-40, 4-18, 26-15
- エンタープライズ, 26-2
- 監査, 28-12
- 権限, 1-39
- スキーマ, 1-36, 26-2
- スキーマに対応付け, 10-2
- セキュリティ・ドメイン, 1-38, 26-2, 27-20
- 専用サーバー, 8-22
- データ・ディクショナリにリスト, 2-2
- デフォルトの表領域, 26-14
- 同時実行動作の調整, 1-28

認証, 26-3

パスワード暗号化, 26-7

表領域, 1-40

表領域割当て制限, 1-41, 26-14

プロセス, 1-15, 8-4

プロファイル, 1-41, 26-20

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, 26-22

ユーザー名, 1-38, 26-2

セッションと接続, 8-4

ライセンス, 26-20

リソース使用に対する制限, 1-40

リソース制限, 26-17

ロール, 27-17

ユーザーのタイプ, 27-19

ロック, 24-40

ユーザー・アクションの監視, 1-41, 28-2

ユーザー定義オペレータ, 10-48

ユーザー定義コスト, 21-19

ユーザー定義データ型, 13-1, 13-3

オブジェクト型, 13-2, 13-3

オブジェクト・リレーショナル・モデル, 1-21

コレクション, 13-10

NESTED TABLE, 13-11

可変配列 (VARRAY), 13-11

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, 29-22

セッション, 8-4

接続, 8-4

専用サーバー・プロセス, 8-22

よ

読込み

データ・ブロック

制限, 26-18

内容を保証しない, 24-3

リピータブル, 24-6

読込みが書込みを阻止するか, 24-11

読取り一貫性, 24-2, 24-4

DML の副問合せ, 24-14

Oracle Parallel Server, 24-7

仮読込み, 24-3, 24-11

キャッシュ・フュージョン, 24-7

- 定義, 1-29
- 問合せ, 15-13, 24-4
- トランザクション, 1-29, 24-4, 24-6
- トリガー, 19-22, 19-24
- 内容を保証しない読込み, 24-3, 24-11
- 非リピータブル・リード, 24-3, 24-11
- 文レベル, 24-6
- マルチバージョンの一貫性モデル, 1-29, 24-4
- メッセージ「スナップショットが古すぎます」, 24-5
- ロールバック・セグメント, 4-20
- 読取り専用
 - データベース
 - オープン, 5-9
 - トランザクション, 1-30
 - 表領域, 3-11
 - 制限, 3-12
 - バックアップ, 29-27
 - 読取り専用推移モード, 3-11
- 読取り専用推移表領域, 3-11
- 読取り専用スナップショット, 31-7
- 予約語, 15-3

ら

- ライセンス
 - 現行の制限の表示, 26-22
 - 同時使用, 26-21
 - 名前付きユーザー, 26-22
- ライブラリ・キャッシュ, 7-6, 7-7, 7-10
- ラッチ
 - 説明, 24-31
- ラッパー
 - プロシージャ型レプリケーション, 31-16
- ランタイム領域, 7-8

り

- リーフ・ブロック, 10-30
- リカバラ (RECO) プロセス, 1-18, 8-12
 - インダウト・トランザクション, 1-34, 5-9, 16-8
- リカバリ
 - Point-in-Time
 - クローン・データベース, 5-7
 - Recovery Manager, 1-49, 29-16
 - インスタンスのリカバリ, 29-4
 - SMON プロセス, 1-17, 8-11, 23-42

- インスタンス障害, 1-44, 29-4
- パラレル DML, 23-42
- ファースト・スタート・チェックポイント, 29-14
- 読取り専用表領域, 29-6
- 概要, 1-42, 29-8
- 基本的な手順, 1-48, 29-10
- クラッシュ・リカバリ, 1-44, 29-4, 29-14
 - SMON プロセス, 1-17, 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, 5-10
 - データベースのオープン, 5-8
 - 読取り専用表領域, 29-6
- 障害リカバリ, 29-28
- 使用される構造, 1-45, 29-6
- 図, 29-13
- 推奨事項, 29-13
- スタンバイ・データベース, 29-28
- データベース全体のバックアップ, 29-25
- データベース・バッファ, 29-8
- デッド・トランザクション, 29-4
- トランザクションのロールバック, 29-10
- パラレル DML, 23-41
- パラレル・リカバリ, 29-11
- パラレル・リストア, 29-17
- プロセスのリカバリ, 8-12, 29-3
- ブロック・レベルのリカバリ, 24-21, 29-15
- 分散処理, 8-12
- 分散トランザクション, 5-9
- 文障害, 29-3
- メディア・リカバリ
 - 使用可能または使用禁止, 29-18
 - ディスパッチャ・プロセス, 8-20
- ロールフォワード, 29-9
- リカバリ時のロールフォワード, 1-48, 29-9
- リスナー, 6-6, 8-15
 - サービス名, 6-6
- リソース制限
 - CPU 時間の制限, 26-18
 - 値の決定, 26-20
 - 概要, 1-41
 - コール・レベル, 26-18
 - セッション当りのアイドル時間, 26-19
 - セッション当りの接続時間, 26-19
 - セッション当りのプライベート SGA 領域, 26-19
 - セッション・レベル, 26-17

- ユーザー当りのセッション数, 26-19
- 論理読込みの制限, 26-18
- リピータブル・リード, 24-3
- リフレッシュ
 - ジョブ・キュー (SNP_n) プロセス, 1-18, 8-13
 - スナップショット, 31-10
 - 増分, 10-19
 - マテリアライズド・ビュー, 10-19
- リモート依存性, 20-11
- リモート・データベース, 1-32
- リモート・トランザクション, 30-35
 - パラレル DML および DDL の制限事項, 23-28
- リモート・プロシージャ・コール, 30-46
- リモート文と分散文のための共有 SQL, 30-35
- 領域管理
 - MINIMUM EXTENT パラメータ, 23-34
 - 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, 22-9
 - データ・ブロック, 4-5
 - パラレル DDL, 23-34
 - 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・ルール内の述語, 27-24
 - マテリアライズド・ビューを使用, 10-18
- リレーショナル DBMS (RDBMS)
 - SQL, 15-2
 - 「Oracle」も参照
 - オブジェクト・リレーショナル DBMS, 13-2
 - 原理, 1-20
- リレーショナル・データベースの操作, 1-20
- リレーション, 1-21
- 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, 11-47

る

- ルート・ブロック, 10-58
- ルールベース最適化, 21-20
- ルールベースのサブスクリプション, 18-8

れ

- 例外
 - ストアド・プロシージャ, 15-19
 - トリガー実行中, 19-24
 - 発生, 15-19
- レシピエント, 18-7
 - サブスクリバ・リスト, 18-7
- 列
 - NESTED TABLE, 10-10
 - NULL の使用禁止, 25-7
 - カーディナリティ, 10-35
 - 疑似列
 - ROWID, 12-15
 - USER, 27-7
 - 削除, 10-7
 - 順序, 10-8
 - 整合性制約, 10-4, 10-9, 25-4, 25-7
 - 説明, 10-3
 - 選択性, 21-9
 - ヒストグラム, 21-10, 21-11
 - 定義, 1-21
 - デフォルト値, 10-9
 - ビューまたは表での最大数, 10-13
 - 未使用, 10-7
 - 列オブジェクト, 13-8
 - 連結索引での最大数, 10-25
- 列の SET UNUSED 句, 10-7
- レプリケーション
 - 概要, 31-2
 - 可用性, 31-2
 - 管理, 31-12
 - 競合
 - プロシージャ型レプリケーション, 31-17
 - 競合の解消, 31-15
 - グループ, 31-4
 - サイト, 31-4
 - 使用するアプリケーション, 31-3
 - 使用方法, 31-2
 - スナップショット, 31-7
 - スナップショット・グループ, 31-5
- 制限
 - ダイレクト・ロード・インサート, 22-10
 - パラレル DML, 23-45
 - 大量配布, 31-3
 - 定義, 31-2
 - 同期, 31-16

- ネットワーク負荷の低減, 31-2
- 配置テンプレート, 31-11
- ハイブリッド構成, 31-11
- パフォーマンス, 31-2
- プロシージャ, 31-16
- マスター・グループ, 31-5
- マスター・サイト, 31-5
- マスター定義サイト, 31-5
- マテリアライズド・ビュー (スナップショット), 10-18
- マルチマスター, 31-5
- モバイル・コンピューティング, 31-2
- レプリケーション・マネージメント API, 31-14
- レプリケーション・オブジェクト, 31-4
- レプリケーション・カタログ, 31-14
- レプリケーション・マネージメント API, 31-14
- 連結索引, 10-24
- レンジ・パーティション化, 11-13
 - キーの比較, 11-19, 11-21
 - 主キー列, 11-41
 - 同一レベル・パーティション化, 11-23
 - パーティション・バウンド, 11-19

ろ

- ローカル索引, 11-28, 11-33
 - 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, 11-57
 - パーティションを並列に作成, 11-29
 - ビットマップ索引
 - パーティション表, 10-39
 - パラレル問合せおよび DML, 10-35
- ローカル・データベース, 1-32
- ローカルに管理される表領域, 3-8
 - 一時表領域, 3-13
- ロール, 1-39, 27-17
 - CONNECT ロール, 27-22
 - DBA ロール, 27-22
 - DDL 文, 27-21
 - EXP_FULL_DATABASE ロール, 27-22
 - IMP_FULL_DATABASE ロール, 27-22
 - PL/SQL ブロック内で設定, 27-21
 - RESOURCE ロール, 27-22
 - アプリケーション, 27-18
 - アプリケーションにおける, 1-40
 - 依存性管理, 27-21
 - オペレーティング・システムを介した管理, 27-23

- 概要, 1-39
- 機能性, 27-2
- キュー管理者, 18-9
- 権限に関する制限, 27-21
- 事前定義済, 27-22
- 実行者権限プロシージャで使用, 27-21
- 使用可能または使用禁止, 27-19
- 使用方法, 27-18
- スキーマには含まれない, 27-20
- セキュリティ・ドメイン, 27-20
- 定義者権限プロシージャでは使用禁止, 27-20
- データベース・リンクを介して取得, 30-23
- 取消し, 27-19
- パスワードの使用, 1-40
- 付与, 27-3, 27-19
- 付与できるユーザー, 27-20
- 命名, 27-20
- ユーザー, 27-19
- ロールバック, 4-19, 16-6
 - セーブポイントまで, 16-8
 - 説明, 16-6
 - 定義, 1-51
 - トランザクションの終了, 16-2, 16-5, 16-6
 - 文レベル, 16-4
 - リカバリ時, 1-48, 29-9
- ロールバック・エントリ, 4-19
- ロールバック・セグメント, 1-7, 4-19
 - 1 つ当りのトランザクション数, 4-21
 - MAXEXTENTS UNLIMITED, 23-41
 - OPTIMAL, 23-41
 - 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
 - エクステンツの割当て, 4-21
 - 新しいエクステンツ, 4-24
 - エクステンツの割当て解除, 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
 - 概要, 4-19, 29-8
 - 起動時の取得, 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, 1-7
- 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, 23-41
- パラレル・リカバリ, 29-10
- 部分的に使用可能な状態, 4-27, 29-4
- プライベート, 4-25
- 無効な状態, 4-27
- 読取り一貫性, 1-29, 4-20, 24-4
- リカバリが必要な状態, 4-27
- リカバリでの使用, 1-47, 29-10
- ロック, 24-31
- ロールバック・トランザクション, 1-52, 16-2, 16-6
- ロギング・モード
 - NOARCHIVELOG モードとの関係, 22-6
 - 影響を受ける SQL 操作, 22-7
 - ダイレクト・ロード・インサート, 22-5
 - パーティション, 11-57
 - パラレル DDL, 23-31, 23-33
- ログ・エントリ, 1-9, 29-9
 - 「REDO ログ・ファイル」も参照, 1-9
- ログ管理ロック, 24-31
- ログ順序番号, 1-45
- ログ・スイッチ
 - ALTER SYSTEM SWITCH LOGFILE, 8-12
 - アーカイバ・プロセス, 1-17, 8-12
- ログ・ライター (LGWR) プロセス, 1-17, 8-9
 - REDO ログ・バッファ, 7-6
 - アーカイブ・モード, 29-18
 - 新しい ARC_n プロセスの起動, 8-13
 - グループ・コミット, 8-10
 - システム変更番号, 16-6
 - 事前書込み, 8-9
 - 手動アーカイブ, 29-21
- ロック, 1-30, 24-3
 - DML が取得, 24-28
 - 図, 24-26
 - DML パーティション・ロック, 11-44
 - Oracle での使用方法, 24-15
- Oracle のロック・マネージメント・サービス, 24-40
 - オブジェクト・レベルのロック, 13-14
 - 解析, 15-12, 24-30
 - 概要, 1-30, 24-3
 - 行 (TX), 24-21
 - ブロック・レベルのリカバリ, 29-15
 - 行共有表ロック (RS), 24-24
 - 行排他ロック (RX), 24-24
 - 共有行排他ロック (SRX), 24-25
 - 共有表ロック (S), 24-25
 - 共有副排他ロック (SSX), 24-25
 - 索引付きの外部キー, 25-19
 - 索引なしの外部キー, 25-18
 - 自動, 1-30, 24-16, 24-19
 - 手動, 1-31, 24-32
 - 動作の例, 24-33
 - タイプ, 24-19
 - 段階的拡大が発生しない, 24-17
- ディクショナリ, 24-29
 - クラスタ, 24-30
 - 存続期間, 24-30
- ディクショナリ・キャッシュ, 24-31
- データ, 24-20
 - 存続期間, 24-16
- デッドロック, 24-17, 24-18
 - 回避, 24-19
- トランザクションをコミットした後, 16-6
- 内部, 24-30
- 排他表ロック (X), 24-26
- パラレル DML, 23-43
- パラレル・キャッシュ管理 (PCM), 24-20
- 表 (TM), 24-22
- 表領域, 24-31
- 表ロックのモード, 24-22
- ファイル管理ロック, 24-31
- 副共有表ロック SS, 24-24
- 副排他表ロック (SX), 24-24
- 変換, 24-17
- ラッチ, 24-31
- ロールバック・セグメント, 24-31
- ログ管理ロック, 24-31
- ロック (LCK0) プロセス, 1-18, 8-13
- 論理 ROWID, 12-19
 - 索引構成表の索引, 10-42
 - 推測の陳腐化, 12-20

- 推測の統計, 12-20
- 物理推測, 10-42, 12-19
- 論理 ROWID での推測, 12-19
 - 陳腐化, 12-20
 - 統計, 12-20
- 論理 ROWID での物理推測, 12-19
 - 陳腐化, 12-20
 - 統計, 12-20
- 論理データベース構造, 1-5
 - 表領域, 3-5
- 論理ブロック, 4-2
- 論理読み込みの制限, 26-18

わ

- 割当て制限
 - SYS ユーザーには適用されない, 26-15
 - ゼロに設定, 26-15
 - 表領域, 1-41, 26-14
 - 一時セグメントでは無視される, 26-15
 - 表領域アクセスの取消し, 26-15