

Oracle8 Server

概要

Vol.2

リリース 8.0

1998 年 2 月

部品番号 A56809-1

ORACLE®

Enabling the Information Age™

Oracle8 Server 概要 : Vol.2

部品番号 A56809-1

リリース 8.0

第 1 版 : 1998 年 2 月

原本名 : Oracle8 Concepts, Volume 2

原本部品番号 : A58398-01

Copyright© Oracle Corporation, 1997, 1998.

All rights reserved.

Printed in Japan.

原著者 : Lefty Leverenz

制限付権利の説明

プログラムの使用、複製、または開示は、オラクル社との契約に記された制約条件に従うものとします。

本書の情報は、予告なしに変更されることがあります。本書に問題を見つけたら、当社にコメントをお送りください。オラクル社は、本書の無謬性を保証しません。

危険な用途への使用について

当社製品は、原子力、航空産業、大量輸送、又は医療の分野など、本質的に危険が伴うアプリケーションを用途として特に開発されておりません。当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って行っていただきたく、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、当社および開発元である米国 Oracle Corporation（その関連会社も含みます）は一切責任を負いかねます。

ORACLE は、Oracle Corporation の登録商標です。

本文中の他社の商品名は、それぞれ各社の商標または登録商標です。

目 次

はじめに	xxv
------------	-----

Vol.1

第 I 部 Oracle の概要

1 Oracle Server の基礎知識

データベースと情報管理	1-2
Oracle Server.....	1-4
Oracle データベース	1-7
データベースの構造と領域管理	1-8
論理データベース構造	1-8
物理データベース構造	1-10
メモリー構造とプロセス	1-12
メモリー構造	1-13
プロセスのアーキテクチャ	1-16
プログラム・インタフェース	1-19
Oracle の動作例	1-19
データの同時実行性と一貫性	1-20
同時実行性	1-20
読み込み一貫性	1-21
ロックのメカニズム	1-22
分散処理と分散データベース	1-23
クライアント/サーバー・アーキテクチャ：分散処理	1-23

分散データベース	1-24
表のレプリケーション	1-25
Oracle と Net8.....	1-26
起動操作と停止操作	1-26
データベース・セキュリティ	1-27
セキュリティのメカニズム	1-28
Trusted Oracle.....	1-33
データベースのバックアップとリカバリ	1-33
なぜ回復は重要か	1-34
障害のタイプ	1-34
回復のために使われる構造	1-36
基本的な回復手順	1-38
Recovery Manager.....	1-39
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	1-40
リレーショナル・モデル	1-40
オブジェクト・リレーショナル・モデル	1-40
スキーマとスキーマ・オブジェクト	1-41
データ・ディクショナリ	1-46
データ・アクセス.....	1-47
SQL - 構造化問合せ言語.....	1-47
トランザクション	1-48
PL/SQL	1-50
データの整合性	1-52

第 II 部 データベースの構造

2 データ・ブロック、エクステンツ、セグメント

データ・ブロック、エクステンツ、セグメントの関係	2-2
データ・ブロック.....	2-3
データ・ブロックの形式	2-3
PCTFREE、PCTUSED、行連鎖の基礎知識	2-5
エクステンツ.....	2-10
エクステンツはいつ割り当てられるか	2-10

エクステントの数とサイズの決定	2-11
エクステントはどのように割り当てられるか	2-11
エクステントはいつ割当てを解除されるか	2-12
セグメント	2-14
データ・セグメント	2-14
索引セグメント	2-15
一時セグメント	2-15
ロールバック・セグメント	2-16
3 表領域とデータ・ファイル	
表領域とデータ・ファイルの基礎知識	3-2
表領域	3-3
SYSTEM 表領域	3-4
データベースにより多くの領域を割り当てる	3-4
表領域をオンライン / オフラインにする	3-7
読み込み専用表領域	3-9
一時表領域	3-10
データ・ファイル	3-11
データ・ファイルの内容	3-11
データ・ファイルのサイズ	3-12
オフライン・データ・ファイル	3-12
4 データ・ディクショナリ	
データ・ディクショナリの基礎知識	4-2
データ・ディクショナリの構造	4-2
SYS、データ・ディクショナリの所有者	4-3
データ・ディクショナリが使用される方法	4-3
Oracle はどのようにデータ・ディクショナリを使うか	4-3
Oracle ユーザーはどのようにデータ・ディクショナリを利用できるか	4-5
動的パフォーマンス表	4-6

第 III 部 Oracle インスタンス

5 データベースとインスタンスの起動と停止

Oracle インスタンスの概要.....	5-2
インスタンスとデータベース	5-2
管理者権限での接続	5-3
パラメータ・ファイル	5-3
インスタンスとデータベースの起動.....	5-5
インスタンスを起動する	5-5
データベースをマウントする	5-6
データベースをオープンする	5-6
データベースとインスタンスの停止.....	5-7
データベースをクローズする	5-8
データベースをディスマウントする	5-8
インスタンスを停止する	5-8

6 メモリー構造

Oracle メモリー構造の概要.....	6-2
システム・グローバル領域 (SGA)	6-2
データベース・バッファ・キャッシュ	6-3
REDO ログ・バッファ	6-5
共有プール	6-6
SGA のサイズ	6-11
SGA のメモリーの使用方法の制御	6-12
プログラム・グローバル領域 (PGA)	6-12
PGA の内容	6-13
PGA のサイズ	6-14
ソート領域.....	6-15
ソート・ダイレクト書込み	6-15
仮想メモリー	6-16
ソフトウェア・コード領域	6-16

7 プロセスの構造

プロセスの概要	7-2
シングル・プロセスの Oracle	7-2
マルチ・プロセスの Oracle	7-3
ユーザー・プロセス	7-4
Oracle プロセス	7-5
トレース・ファイルと ALERT ファイル	7-14
Oracle の構成におけるバリエーション	7-15
シングル・タスク構成	7-16
専用サーバー（2 タスク）構成	7-17
マルチスレッド・サーバー	7-19
Oracle の動作例	7-24
専用サーバー・プロセスを使う Oracle の例	7-24
マルチスレッド・サーバーを使う Oracle の例	7-25
プログラム・インタフェース	7-26
プログラム・インタフェースの構造	7-27
プログラム・インタフェース・ドライバ	7-27
オペレーティング・システムの通信ソフトウェア	7-27

第 IV 部 オブジェクト・リレーショナル DBMS

8 スキーマ・オブジェクト

スキーマ・オブジェクトの概要	8-2
表	8-3
表データはどのように格納されるか	8-4
NULL	8-7
列のデフォルト値	8-8
ネストした表	8-9
ビュー	8-10
ビューの記憶領域	8-11
ビューはどのように使用されるか	8-11
ビューのメカニズム	8-12

依存性とビュー	8-13
更新可能な結合ビュー	8-13
オブジェクト・ビュー	8-14
順序生成機能	8-14
シノニム	8-15
索引	8-16
一意索引と一意でない索引	8-16
複合索引	8-17
索引とキー	8-17
索引はどのように格納されるか	8-18
逆キー索引	8-20
ビットマップ索引	8-21
索引構成表	8-25
索引構成表の利点	8-26
行オーバーフロー領域付きの索引構成表	8-26
索引構成表に適したアプリケーション	8-27
クラスタ	8-28
パフォーマンスの考慮事項	8-31
クラスタ化されたデータ・ブロックの形式	8-31
クラスタ・キー	8-32
クラスタ索引	8-32
ハッシュ・クラスタ	8-33
データはどのようにハッシュ・クラスタに格納されるか	8-33
ハッシュ・キー値	8-35
ハッシュ関数	8-36
ハッシュ・クラスタに対する領域の割当て	8-37
9 パーティション表とパーティション索引	
パーティション化の基礎知識	9-2
パーティション化とは何か	9-2
パーティション化の利点	9-4
大規模データベース (VLDB)	9-4

定期メンテナンス操作による休止時間の短縮	9-5
データ障害による休止時間の短縮	9-6
DSS のパフォーマンス	9-7
I/O のパフォーマンス	9-7
ディスクのストライプ化：パフォーマンスと可用性	9-7
パーティションの透過性	9-9
パーティション・ビューを使った手動のパーティション化	9-9
パーティション化の基本的なモデル	9-10
レンジ・パーティション化	9-11
パーティション名	9-13
パーティション・バウンドとパーティション化キー	9-13
同一レベル・パーティション化	9-17
表および索引をパーティション化するときのルール	9-19
表のパーティション化	9-19
索引のパーティション化	9-21
DML パーティション・ロック	9-28
Oracle Parallel Server のパフォーマンスの考慮事項	9-29
メンテナンス操作	9-29
パーティションのメンテナンス操作	9-30
索引の管理	9-35
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	9-38
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	9-39
SQL の拡張：拡張パーティション表名	9-39
拡張パーティション表名の例	9-40

10 ビルトイン・データ型

Oracle データ型	10-2
文字データ型	10-2
NUMBER データ型	10-5
DATE データ型	10-7
LOB データ型	10-9
RAW および LONG RAW データ型	10-10

ROWID データ型	10-11
MLSLABEL データ型	10-15
Oracle データ型についての情報のまとめ	10-15
ANSI データ型、DB2 データ型、SQL/DS データ型	10-17
データ変換	10-18
 11 ユーザー定義データ型 (オブジェクト・オプション)	
基礎知識	11-2
複合データ・モデル	11-2
マルチメディア・データ型	11-3
ユーザー定義データ型	11-3
オブジェクト型	11-3
コレクション型	11-9
アプリケーション・インタフェース	11-11
SQL	11-11
PL/SQL	11-12
Pro*C/C++	11-12
OCI	11-12
OTT	11-13
 12 ユーザー定義データ型の使用方法	
参照とネーム変換	12-2
表の別名	12-2
引数なしのメソッド・コール	12-3
ユーザー定義型の記憶領域	12-4
リーフ・レベル属性	12-4
行オブジェクト	12-4
列オブジェクト	12-4
参照 (REF)	12-5
ネストした表	12-5
VARRAY	12-5
オブジェクト属性のプロパティ	12-5
NULL	12-6

デフォルト	12-7
制約	12-7
索引	12-8
トリガー	12-9
ユーザー定義型の権限とそのメソッド	12-9
システム権限	12-10
スキーマ・オブジェクト権限	12-10
新しい型または表で型を使う	12-10
例	12-11
型アクセスとオブジェクト・アクセスについての権限	12-11
依存性と不完全な型	12-13
不完全な型を完全にする	12-14
表の型の依存性	12-14
ユーザー定義型の Import/Export	12-15

13 オブジェクト・ビュー

基礎知識	13-2
オブジェクト・ビューの利点	13-2
オブジェクト・ビューの定義	13-2
オブジェクト・ビューの使用法	13-4
オブジェクト・ビューの更新	13-4

Vol.2

第 V 部 データ・アクセス

14 SQL と PL/SQL

構造化問合せ言語 (SQL)	14-2
SQL 文	14-2
非標準 SQL の識別	14-6
再帰 SQL	14-6
カーソル	14-6
共有 SQL	14-7

解析	14-7
SQL の処理.....	14-8
SQL 文の実行の概要.....	14-8
DML 文の処理	14-10
DDL 文の処理	14-13
トランザクションの制御	14-13
PL/SQL.....	14-14
PL/SQL が実行される方法	14-14
PL/SQL の言語要素	14-16
ストアド・プロシージャ	14-17
外部プロシージャ	14-18
 15 トランザクションの管理	
トランザクションの基礎知識.....	15-2
文の実行とトランザクションの制御	15-3
文レベルのロールバック	15-4
Oracle とトランザクションの管理.....	15-4
トランザクションをコミットする	15-5
トランザクションをロールバックする	15-6
セーブポイント	15-7
2 フェーズ・コミット・メカニズム	15-7
離散トランザクションの管理.....	15-8
 16 アドバンスド・キューイング	
メッセージ・キューイングの基礎知識	16-2
同期通信	16-2
非同期通信	16-2
Oracle Advanced Queuing	16-2
キューイング・エンティティ	16-3
アドバンスド・キューイングの機能	16-4

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基礎知識	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
プロシージャのガイドライン	17-7
プロシージャの利点	17-7
無名 PL/SQL ブロックとストアド・プロシージャ	17-8
スタンドアロン・プロシージャ	17-9
ストアド・プロシージャの依存性の追跡	17-9
外部プロシージャ	17-9
パッケージ	17-9
パッケージの利点	17-14
パッケージの依存性の追跡	17-15
Oracle がプロシージャとパッケージを格納する方法	17-15
プロシージャとパッケージのコンパイル	17-15
コンパイル済みコードのメモリーへの格納	17-15
プロシージャとパッケージのデータベースへの格納	17-15
Oracle がプロシージャとパッケージを実行する方法	17-16
ユーザー・アクセスの検査	17-16
プロシージャの有効性の検査	17-16
プロシージャの実行	17-17

18 データベース・トリガー

トリガーの基礎知識	18-2
トリガーの使用方法	18-3
トリガーの使用上の注意	18-3
トリガーと宣言整合性制約	18-5
トリガーの各部分	18-5
トリガー・イベントまたはトリガー文	18-6
トリガー条件	18-7
トリガー・アクション	18-7

トリガーのタイプ.....	18-7
行トリガーと文トリガー	18-7
BEFORE トリガーと AFTER トリガー	18-8
トリガーの組合せ	18-8
INSTEAD OF トリガー	18-11
トリガーの実行	18-13
トリガーの実行モデルと整合性制約のチェック	18-14
トリガーのデータ・アクセス	18-15
トリガーの記憶領域	18-17
トリガーの実行	18-17
トリガーの依存性のメンテナンス	18-17
 19 Oracle の依存性の管理	
依存性の問題の基礎知識.....	19-2
スキーマ・オブジェクトの依存性の解決.....	19-4
ビューと PL/SQL プログラム・ユニットのコンパイル.....	19-5
依存性の管理と存在しないスキーマ・オブジェクト.....	19-7
共有 SQL の依存性管理.....	19-8
ローカルとリモートの依存性の管理.....	19-8
ローカル依存性の管理	19-9
リモート依存性の管理	19-9
 20 オプティマイザ	
最適化とは？.....	20-2
実行計画	20-2
実行の順序	20-5
コストベース最適化とルールベース最適化.....	20-6
コストベースのアプローチ	20-6
オプティマイザの操作の概要.....	20-11
オプティマイザの操作	20-11
SQL 文のタイプ.....	20-11
式と条件の評価	20-12

定数	20-13
LIKE 演算子	20-13
IN 演算子	20-13
ANY または SOME 演算子	20-14
ALL 演算子	20-14
BETWEEN 演算子	20-15
NOT 演算子	20-15
推移律	20-15
文の変換と最適化	20-17
OR を複合問合せに変換する	20-17
複合文を結合文に変換する	20-20
ビューにアクセスする文の最適化	20-22
複合問合せの最適化	20-34
分散型の文の最適化	20-37
最適化アプローチと目標の選択	20-37
OPTIMIZER_MODE 初期化パラメータ	20-38
データ・ディクショナリ内の統計データ	20-38
ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータ	20-39
FIRST_ROWS、ALL_ROWS、CHOOSE、RULE の各ヒント	20-39
PL/SQL とオブティマイザの目標	20-39
アクセス・パスの選択	20-40
アクセス方法	20-40
アクセス・パス	20-42
アクセス・パスの選択	20-55
結合文の最適化	20-59
結合操作	20-60
結合文の実行計画の選択	20-65
外部結合のビュー	20-68
非結合と半結合の最適化	20-70
「スター」問合せの最適化	20-70
スター問合せの例	20-71
スター問合せのチューニング	20-71

スター型変換	20-72
第 VI 部 パラレル SQL とダイレクト・ロード・インサート	
21 ダイレクト・ロード・インサート	
ダイレクト・ロード・インサートの基礎知識.....	21-2
ダイレクト・ロード・インサートの利点	21-2
INSERT ... SELECT 文.....	21-3
ダイレクト・ロード・インサート文の種類	21-3
シリアル INSERT とパラレル INSERT	21-3
ロギング・モード	21-5
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.....	21-8
索引のメンテナンス	21-8
領域についての考慮事項	21-8
ロックについての考慮事項	21-9
ダイレクト・ロード・インサートの制限事項.....	21-9
22 パラレル実行	
パラレル実行の概要	22-2
パラレル化できる操作	22-2
Oracle が操作をパラレル化する方法.....	22-3
パラレル実行のプロセス・アーキテクチャ	22-5
パラレル・サーバー・プール	22-6
SQL 文のパラレル化.....	22-8
並行度の設定	22-12
操作の並行度の決定	22-12
作業負荷のバランス調整	22-14
SQL 文のパラレル化ルール.....	22-15
パラレル DDL	22-21
パラレル化できる DDL 文	22-22
パラレルの CREATE TABLE ... AS SELECT	22-22
回復可能性とパラレル DDL	22-23
パラレル DDL の領域管理	22-24

パラレル DML	22-26
手動のパラレル化と比べたときのパラレル DML の利点	22-26
いつパラレル DML を使うか	22-27
パラレル DML を有効にする	22-28
パラレル DML のためのトランザクション・モデル	22-29
パラレル DML の回復	22-30
パラレル DML の領域に関する考慮事項	22-31
パラレル DML のためのリソースのロックとエンキュー	22-31
パラレル DML の制限事項	22-33
親和性	22-36
その他の並行性	22-37

第 VII 部 データの保護

23 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	23-2
回避可能な現象とトランザクション分離レベル	23-2
ロックのメカニズム	23-3
データの同時実行性と一貫性の管理方法	23-3
マルチバージョン一貫性制御	23-4
文レベルの読み取り一貫性	23-5
トランザクション・レベルの読み取り一貫性	23-6
Oracle の分離レベル	23-6
分離レベルの設定	23-6
コミット読み取りと直列可能分離の比較	23-9
分離レベルの選択	23-11
データをロックする方法	23-14
トランザクションとデータ同時実行性	23-14
デッドロック	23-16
ロックの種類	23-18
DML (データ) ロック	23-19
DDL ロック (ディクショナリ・ロック)	23-26

ラッチと内部ロック	23-28
明示的（手動）データ・ロック	23-29
Oracle のロック管理サービス.....	23-38
24 データの整合性	
データ整合性の定義	24-2
データ整合性のタイプ	24-2
Oracle がデータの整合性を施行する方法.....	24-3
整合性制約の基礎知識.....	24-4
整合性制約の利点	24-5
整合性制約のパフォーマンス・コスト	24-6
整合性制約のタイプ	24-6
NOT NULL 整合性制約.....	24-6
UNIQUE キー整合性制約	24-7
PRIMARY KEY 整合性制約.....	24-10
FOREIGN KEY（参照）整合性制約.....	24-11
CHECK 整合性制約.....	24-16
制約チェックのメカニズム	24-17
デフォルト列値と整合性制約チェック	24-18
遅延制約チェック	24-19
制約の属性	24-19
SET CONSTRAINTS モード.....	24-19
一意制約と一意索引	24-20
制約の使用可能および使用禁止、施行	24-20
25 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	25-2
スキーマおよびデータベース・ユーザー、セキュリティ・ドメイン.....	25-2
ユーザーの認証	25-3
オペレーティング・システムによる認証	25-3
ネットワークによる認証	25-4
Oracle データベースによる認証.....	25-4

データベース管理者の認証	25-6
ユーザー表領域の設定と割当て制限	25-7
デフォルト表領域	25-7
一時表領域	25-7
表領域のアクセスと割当て制限	25-7
ユーザー・グループ PUBLIC	25-8
ユーザー・リソースの制限とプロファイル	25-9
システム・リソースのタイプと制限	25-9
プロファイル	25-11
ライセンス	25-12
同時使用ライセンス	25-13
名前付きユーザー・ライセンス	25-14
 26 権限とロール	
権限	26-2
システム権限	26-2
スキーマ・オブジェクト権限	26-3
ロール	26-10
ロールの一般的な使用方法	26-11
ロールのメカニズム	26-11
ロールの付与と取消し	26-12
ロールの付与と取消しを実行できるユーザー	26-12
ロールの命名	26-13
ロールとユーザーのセキュリティ・ドメイン	26-13
名前付き PL/SQL ブロックとロール	26-13
データ定義言語の文とロール	26-13
事前定義済みのロール	26-14
オペレーティング・システムとロール	26-14
分散環境におけるロール	26-15
 27 監査	
監査の基礎知識	27-2

監査機能	27-2
監査のメカニズム	27-4
文監査	27-6
権限監査	27-7
スキーマ・オブジェクト監査	27-7
ビューとプロシージャのスキーマ・オブジェクト監査オプション	27-8
文および権限、スキーマ・オブジェクトの監査の対象を限定する	27-8
成功した文の実行と失敗した文の実行の監査	27-9
BY SESSION 監査と BY ACCESS 監査	27-9
ユーザー別の監査	27-11
28 データベースの回復	
データベース回復の基礎知識	28-2
エラーと障害	28-2
データベースの回復で使われる構造	28-6
データベースのバックアップ	28-7
REDO ログ	28-7
ロールバック・セグメント	28-7
制御ファイル	28-8
ロールフォワードとロールバック	28-8
REDO ログとロールフォワード	28-8
ロールバック・セグメントとロールバック	28-9
Recovery Manager	28-10
リカバリ・カタログ	28-10
パラレル化	28-11
レポートの生成	28-12
回復のパラレル実行	28-12
パラレル回復を活用できる状況	28-13
回復プロセス	28-13
データベースのアーカイブ・モード	28-14
NOARCHIVELOG モード（メディア回復が使用不可能）	28-15
ARCHIVELOG モード（メディア回復が使用可能）	28-15

制御ファイル.....	28-18
制御ファイルの内容	28-18
制御ファイルの多重化	28-19
データベースのバックアップ.....	28-19
全体データベース・バックアップ	28-20
部分データベース・バックアップ	28-21
Export および Import ユーティリティ	28-22
読み込み専用表領域とバックアップ	28-22
耐障害性.....	28-22
災害時回復の計画	28-22
スタンバイ・データベース	28-23

第 VIII 部 分散処理と分散データベース

29 分散処理

Oracle クライアント / サーバー・アーキテクチャ	29-2
分散処理.....	29-2
Net8.....	29-4
Net8 の仕組み.....	29-5

30 分散データベース

Oracle の分散データベース・アーキテクチャ.....	30-2
クライアントとサーバー	30-2
ネットワーク	30-4
データベースとデータベース・リンク	30-4
データベース・リンク	30-6
スキーマ・オブジェクトのネーム変換	30-6
複数のバージョンの Oracle Server の間の接続	30-7
分散データベースと分散処理	30-7
分散データベースとデータベース・レプリケーション	30-7
異種間分散データベース.....	30-8
透過的な SQL アクセス.....	30-8
プロシージャ・アクセス	30-8

ゲートウェイの機能	30-9
バージョン 8 ゲートウェイ	30-9
バージョン 4 ゲートウェイ	30-10
分散データベース・アプリケーションの開発.....	30-10
リモート SQL 文と分散 SQL 文	30-10
リモート・プロシージャ・コール (RPC)	30-11
リモート・トランザクションと分散トランザクション	30-11
分散データベース・システムにおける透過性	30-13
Oracle 分散データベース・システムの管理	30-15
サイト自律性	30-15
分散データベースのセキュリティ	30-15
Oracle 分散データベースの管理ツール.....	30-17
Oracle Enterprise Manager.....	30-17
サードパーティの管理ツール	30-18
SNMP のサポート	30-18
各国語サポート	30-18

31 データベースのレプリケーション

レプリケーションとは?.....	31-2
基本レプリケーションの概念	31-2
アドバンスト (対称型) レプリケーション	31-3
基本レプリケーションの概念.....	31-4
基本レプリケーションの使用方法	31-4
読み込み専用表スナップショット	31-6
スナップショットのリフレッシュ	31-8
その他の基本レプリケーション・オプション	31-10
アドバンスト・レプリケーションの概念.....	31-11
アドバンスト・レプリケーションの使用方法	31-12
アドバンスト・レプリケーションの構成	31-13
アドバンスト・レプリケーションと Oracle Replication Manager	31-17
レプリケーションのオブジェクトおよびグループ、 サイト、カタログ	31-17
Oracle のアドバンスト・レプリケーション・アーキテクチャ.....	31-18

レプリケーションの管理者および伝播担当者、受信者	31-22
レプリケーションの競合	31-22
固有のアドバンスト・レプリケーション・オプション	31-26

第 IX 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。

Oracle8 および Oracle8 Enterprise Edition

『Oracle8 Server概要』では、Oracle8およびOracle8 Enterprise Edition製品の機能を説明しています。Oracle8およびOracle8 Enterprise Editionには、同じ基本機能がいくつかあります。ただし、Enterprise Edition だけ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、オブジェクト機能を使うには、Enterprise Edition とオブジェクト・オプションが必要です。

Oracle8とOracle8 Enterprise Editionの違いと、使用可能な機能とオプションの詳細は、『Oracle8とOracle8 Enterprise Editionの解説』を参照してください。

対象読者

このマニュアルは、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・システムの環境についての知識が必要です。最初に、第 1 章「Oracle Server の基礎知識」を必ず読んでください。第 1 章では、このマニュアル全体で使われている概念と用語について包括的に説明されています。

インストールや移行について

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、主としてインストールに関心のある方は、使用するオペレーティング・システムを対象にした Oracle のマニュアルを参照してください。また、主としてデータベースおよびアプリケーションの移行に関心のある方は、『Oracle8 Server 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。Oracle Server を管理する方法の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

アプリケーションの設計について

このマニュアルには、データベース管理者だけでなく、Oracle に精通したユーザーや、高度なデータベース・アプリケーションの設計者にも役立つ情報が記載されています。ただし、データベース・アプリケーションの開発者は、『Oracle8 Server アプリケーション開発者ガイド』と、Oracle データベース・アプリケーションの開発に使うツールまたは言語製品のマニュアルも参照する必要があります。

このマニュアルの構成

このマニュアルは 2 巻構成で、次のような部に分かれています。

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

- 第 VI 部: パラレル SQL とダイレクト・ロード・インサート
- 第 VII 部: データの保護
- 第 VIII 部: 分散処理と分散データベース
- 第 IX 部: 付録

Vol.1

第 I 部: Oracle の概要

第 1 章: Oracle Server の基礎知識

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

第 II 部: データベースの構造

第 2 章: データ・ブロック、エクステンツ、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 8 章「スキーマ・オブジェクト」の内容を補足するものです。

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・ディクショナリ

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

第 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: メモリー構造

Oracle データベース・システムで使うメモリー構造について説明します。

第 7 章: プロセスの構造

Oracle インスタンスのプロセス構造と、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Ⅳ部：オブジェクト・リレーショナル DBMS

第8章：スキーマ・オブジェクト

表、ビュー、名前付き順序、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。さらに、データを効率的に検索するための索引やクラスタなどのオプション構造についても説明します。

第9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10章：ビルトイン・データ型

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

第11章：ユーザー定義データ型（オブジェクト・オプション）

Oracle 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）によって提供されるオブジェクト拡張機能について概説します。

第12章：ユーザー定義データ型の使用方法

Oracle ORDBMS で利用可能なユーザー定義のオブジェクト型について説明します。

第13章：オブジェクト・ビュー

Oracle ORDBMS 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14章：SQL と PL/SQL

Oracle と対話するために使う 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15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う SQL 文について説明します。トランザクションとは、1 つの単位としてまとめて実行される論理作業単位です。

第16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。この機能を使うと、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成体について説明します。

第 18 章：データベース・トリガー

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

第 19 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 20 章：オプティマイザ

オプティマイザの仕組みについて説明します。オプティマイザは、各 SQL 文を最も効率的に実行する方法を選択する Oracle の機能です。

第 VI 部：パラレル SQL とダイレクト・ロード・インサート

第 21 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・パスについて説明します。

第 22 章：パラレル実行

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

第 VII 部：データの保護

第 23 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 24 章：データの整合性

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

第 25 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 26 章：権限とロール

システム・レベルとオブジェクト・レベルでのセキュリティについて説明します。

第 27 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 28 章：データベースの回復

データベースの回復に使わ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 VIII 部：分散処理と分散データベース

第 29 章：分散処理

Oracle Server を実行できる分散処理環境について説明します。

第 30 章：分散データベース

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

第 31 章：データベースのレプリケーション

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

第 IX 部：付録

付録 A: オペレーティング・システム固有の情報

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

このマニュアルの使用法

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

このマニュアルの各部分は、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、第 1 章を読んだ後、主にセキュリティの管理について関心のある管理者は、第 VII 部「データの保護」の中の、特に第 25 章「データベース・アクセスの制御」、第 26 章「権限とロール」、第 27 章「監査」の部分をよく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います。

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われています。

英大文字

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われます。

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。

かぎカッコ

マニュアルのタイトルは『 』で囲んであります。強調する語句は「 」で囲んであります。

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、モノスペース・フォントで記載します。たとえば、次のとおりです。

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

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

コード例の中の英大文字

例文の中の大文字の語は、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、単なる例として使わ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V部

データ・アクセス

第 V 部では、Oracle データベースのデータにアクセスするための SQL 文で構成されるトランザクションの使用方法を説明し、また、付加的なデータ・アクセス機能を提供するプロシージャ型言語の構成体について説明します。さらに、各 SQL 文を実行するための最も効率的な方法を選ぶ、オブティマイザ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 14 章「SQL と PL/SQL」
- 第 15 章「トランザクションの管理」
- 第 16 章「アドバンスト・キューイング」
- 第 17 章「プロシージャとパッケージ」
- 第 18 章「データベース・トリガー」
- 第 19 章「Oracle の依存性の管理」
- 第 20 章「オブティマイザ」

High thoughts must have high language.

Aristophanes: *Frogs*

この章では、SQL（構造化問合せ言語）と、PL/SQL（SQL に対する Oracle のプロシージャ型拡張機能）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（SQL）
- SQL の処理
- PL/SQL

追加情報：PL/SQL の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

構造化問合せ言語 (SQL)

SQL は、非常に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、ユーザーが実行したい処理内容を 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。オラクル社が Oracle 用にインプリメントした SQL は、ANSI/ISO 1992 規格 SQL データ言語のエントリ・レベルに 100% 準拠しています。

Oracle SQL には、ANSI/ISO 標準 SQL 言語に対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は追加のコマンドを提供します。SQL*Plus、Oracle Enterprise Manager、Server Manager などの Oracle Tools を使うと、Oracle データベースに対して ANSI/ISO 標準の SQL 文、またはこれらのツールで使用可能な追加のコマンド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 を使って実行されます。その他のデータ・アクセス方法を使うと、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。

追加情報：SQL コマンドおよび SQL のその他の部分（演算子、関数、書式モデルなど）の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

Oracle Enterprise Manager コマンドと Server Manager コマンドの詳細（SQL コマンドとの相違など）は、『Oracle Enterprise Manager 管理者ガイド』を参照してください。

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

SQL 文

Oracle データベースの情報に対して実行されるすべての操作は、SQL 「文」を使って実行します。SQL 文は、有効な「SQL コマンド」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、他の目的には使えない語です。たとえば、SELECT と UPDATE は予約語なので、表名には使えません。

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。次に示すように、文 (ステートメント) は、SQL の「センテンス」に相当するものでなければなりません。

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

実行できるのは SQL 文として認められるものだけです。次のような「センテンスの断片」を実行しようとする、SQL 文を実行するにはテキストが不足している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

注意 : Oracle では、PL/SQL プログラム・ユニットの中で SQL 文を使うこともサポートされています。この機能の詳細は、第 17 章「プロシージャとパッケージ」および第 18 章「データベース・トリガー」を参照してください。

データ操作言語 (DML) 文

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、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す (SELECT)
- 表またはビューに新しいデータ行を追加する (INSERT)
- 表またはビューの既存の行の列値を変更する (UPDATE)
- 表またはビューから行を削除する (DELETE)
- SQL 文の実行計画を表示する (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する (LOCK TABLE)

DML 文は、最も頻繁に使う 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 (DDL) 文

DDL 文は、スキーマ・オブジェクトを定義したり、その構造を変更したり、それらのオブジェクトを削除したりする操作を実行します。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) を作成および変更、削除する (CREATE、ALTER、DROP)
- スキーマ・オブジェクトの名前を変更する (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する (TRUNCATE)
- スキーマ・オブジェクトに関する統計情報を収集したり、オブジェクト構造の妥当性を検査したり、オブジェクト内の連鎖行のリストを表示したりする (ANALYZE)
- 権限とロールを付与したり取り消したりする (GRANT、REVOKE)
- 監査オプションのオン / オフを切り換える (AUDIT、NOAUDIT)
- データ・ディクショナリにコメントを追加する (COMMENT)

DDL 文は、先行するトランザクションを暗黙のうちにコミットし、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;
```

データベースおよびデータへのアクセスに対応する DDL 文の詳細は、第 25 章「データベース・アクセスの制御」および第 26 章「権限とロール」、第 27 章「監査」を参照してください。

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する (COMMIT)
- トランザクションの中での変更のうち、その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なされたものの内容を取り消す (ROLLBACK)
- ロールバックできるポイントを設定する (SAVEPOINT)
- トランザクションのプロパティを設定する (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 / 使用禁止の切換えなど、特殊化された機能を実行することによって現行のセッションを変更する (ALTER SESSION)
- 現行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する (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。

システム制御コマンドは、ALTER SYSTEM だけです。このコマンドは、共有サーバーの最小数など設定値の変更、セッションのキル (停止)、その他の作業のために使います。

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われます。

埋込み SQL 文によって、次のことを実行できます。

- カーソルを定義したり、割り当てたり、解放したりする (DECLARE CURSOR、OPEN、CLOSE)
- データベースを指定し、Oracle に接続する (DECLARE DATABASE、CONNECT)
- 変数名を割り当てる (DECLARE STATEMENT)
- 記述子を初期化する (DESCRIBE)
- エラー条件と警告の処理方法を指定する (WHENEVER)
- SQL 文を解析して実行する (PREPARE、EXECUTE、EXECUTE IMMEDIATE)
- データベースからデータを取り出す (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う 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 フラガーを使って「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 フラガーを使います。

追加情報 : FIPS フラガーの使用の詳細は、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』または『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』、『SQL*Module for Ada Programmer's Guide』を参照してください。

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。Oracle が内部的に実行する再帰 SQL にユーザーが関心を払う必要はありません。

カーソル

カーソルとは、解析済みの文とその文の処理に使うその他の情報が入れら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。

多くの Oracle ユーザーは Oracle ユーティリティの自動カーソル処理を利用しますが、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え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えます。

ユーザー・セッションごとに、初期化パラメータ OPEN_CURSORS で設定された制限を上限として、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 は暗黙のうちに再帰 SQL 文を発行する必要があり、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 文を使うと、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たくさんの更新が加えられます。それらの再帰カーソルのために再帰コールが実行されます。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 領域も使います。

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われる 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われます。したがって、同じ文に対しては 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なので、任意の Oracle プロセスが共有 SQL 領域を使えます。SQL 領域を共有することで、データベース・サーバー上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際、Oracle は、ユーザーとアプリケーションが直接発行する SQL 文と、DDL 文によって内部的に発行される再帰 SQL 文を評価します。

追加情報：共有 SQL の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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する。
- 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別する。
- 文にプライベート SQL 領域を割り当てる。

また、Oracle は、ライブラリ・キャッシュにその文の解析済み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みの表現を使って、すぐにその文を実行します。存在しない場合、Oracle はその文の解析済みの表現を生成し、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、そこに解析済みの表現をそこに格納します。

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 領域と関連付けられます。文がプライベート SQL 領域と関連付けられたなら、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なら、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、できるだけ実行しないようにしてください。

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。(14-14 ページの「PL/SQL」を参照。) 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。

SQL の処理

この項では、SQL 処理の基本について説明します。トピックは次のとおりです。

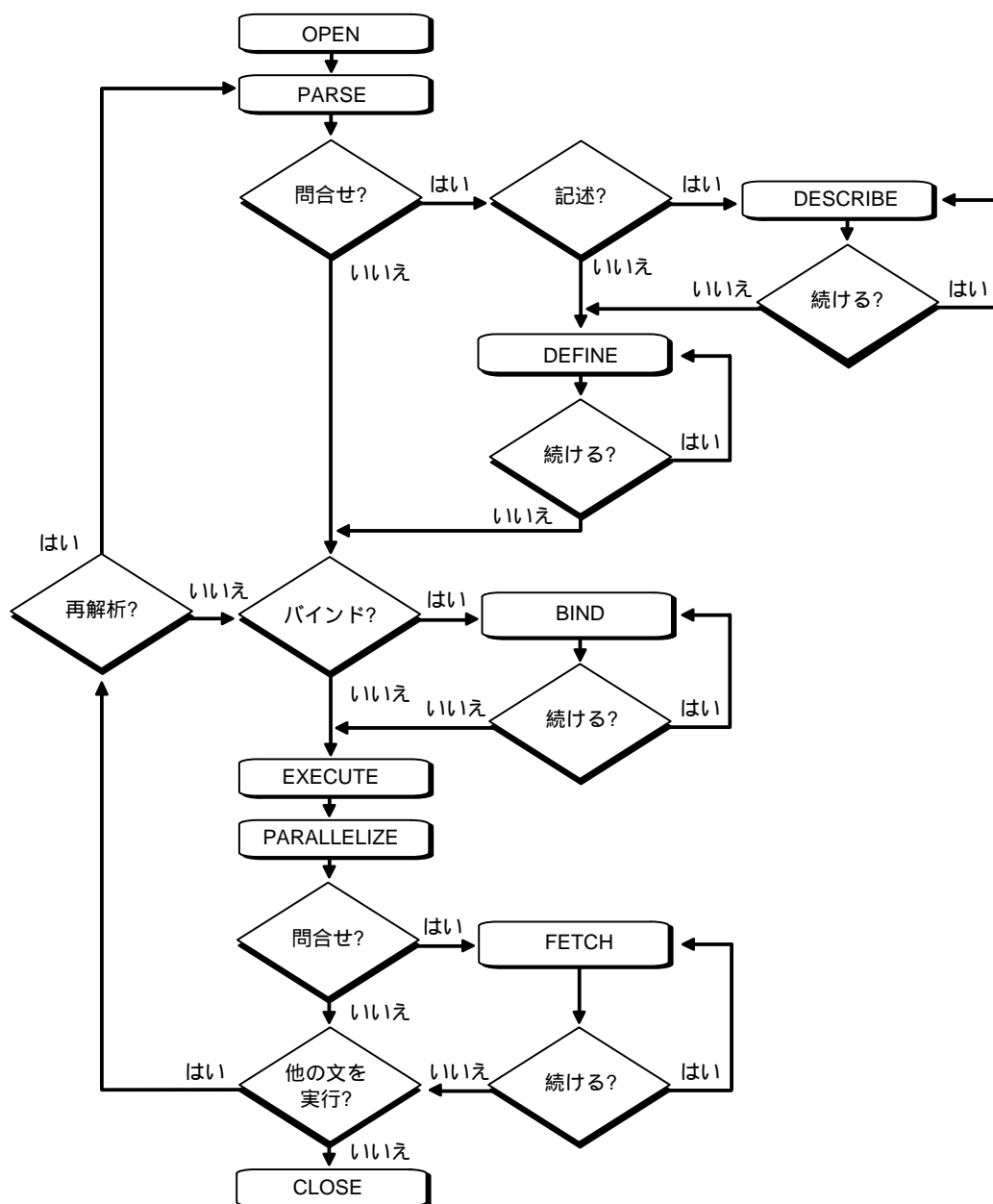
- SQL 文の実行の概要
- DML 文の処理
- DDL 文の処理
- トランザクションの制御

SQL 文の実行の概要

図 14-1 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。しかし、Oracle アプリケーションを作る場合は、この情報が参考になるかもしれません。

図 14-1 SQL 文の処理の段階



DML 文の処理

ここでは、SQL 文の実行中に何が起きるかを簡単に説明します。特に、DML 文処理の各段階で何が起きるかを説明します。

Pro*C プログラムを使って、ある部門の従業員全員の給与を増やす処理を実行するとします。現在使っ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、部門番号の値が入っている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 値が使われます。

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 14-1 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

詳細は、14-11 ページの「問合せの処理」を参照してください。

段階 1: カーソルの作成

カーソルは、プログラム・インタフェース・コールによって作られます。カーソルは SQL 文からは独立して作られます。また任意の SQL 文を想定して作られます。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られます。しかしプリコンパイラ・プログラムでは、暗黙のうちにカーソルが作られたり、カーソル作成が明示的に宣言されたりすることもあります。

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、解析済みの表現が共有 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、その妥当性を検証する。
- 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する。
- 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する。
- 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する。
- 文の最適な実行計画を決定する。
- その実行計画を共有 SQL 領域にロードする。
- 分散型の文の場合は、そ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する。

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だけです。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。(14-7 ページの「共有 SQL」を参照してください。)

解析段階には、文の実行回数に関係なく 1 回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 回だけ変換し、後でその文が参照されると、解析済み文を再実行します。

SQL 文を解析するとその文の妥当性が検査されますが、解析するだけ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・報告されません。

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だけですが、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。

問合せ処理だけに関連する問題がいくつかあります。明示的な SELECT 文だけでなく、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。

- 「読み込み一貫性」が必要。
- 中間処理に一時セグメントを使える。
- SQL 文処理の記述、定義、フェッチ段階が必要になることがある。

段階 3: 問合せの結果の記述

記述段階が必要なのは、問合せ結果の特性がわかっていない場合、たとえば問合せがユーザーにより対話式に入力されたなどの場合だけです。

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

段階 4: 問合せの出力の定義

問合せ定義の段階では、フェッチされた各値を受け取るよう定義された変数の位置およびサイズ、データ型を指定します。必要に応じて、Oracle は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」といいます。

プログラムでは、値を見つけることのできる位置（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を入力するためのプロンプトを表示するだけなので、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かもしれません。

位置を指定した（参照によるバインド）ので、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・アドレスを使っ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、暗黙のうち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。

追加情報：値のデータ型と長さを指定する場合の詳細は、次の資料を参照してください。

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

段階 6: 文の平行化

Oracle では、問合せ (SELECT)、INSERT、UPDATE、DELETE、および (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の) 特定の DDL 操作を平行化できます。平行化すると、複数のサーバー・プロセスが SQL 文の処理を実行するので、処理を高速に完了できます。

平行 SQL の詳細は、第 22 章「平行実行」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いました。それで、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ので、行をロックする必要はありません。しかし、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え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ます。このような処理を「配列処理」といいます。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 の配列の開始点と想定されます。

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なので、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、実際には解析およびデータ・ディクショナリの参照、実行が含まれます。

トランザクション管理およびセッション管理、システム管理の SQL 文は、解析および実行段階を使って処理されます。それらを再実行するには、実行段階をもう一度実行するだけです。

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注意を払うのは、Oracle へのプログラム・インタフェースを使うアプリケーション・デザイナーだけです。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するため、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すべて含める必要があります。余分も不足もあってはなりません。

- 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 文だけを含めてください。

たとえば、口座間の振替操作（トランザクションまたは作業論理単位）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つの作業論理単位として一緒に失敗または一緒に成功するようであればなりません。引き出されていないのに預け入れられることがあってはいけません。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だけでなく、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うべきかも判断しなければなりません。詳細は、15-8 ページの「分散トランザクションの管理」を参照してください。

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 を使うと、SQL 文をプロシージャ型の言語要素と混合して使えます。さらに PL/SQL では、プロシージャ、ファンクション、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る、名前が付いていないが、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 文を記述できるの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・プロシージャを作ると、Oracle はそのプロシージャを解析し、その解析済み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）を作って保管することもできます。

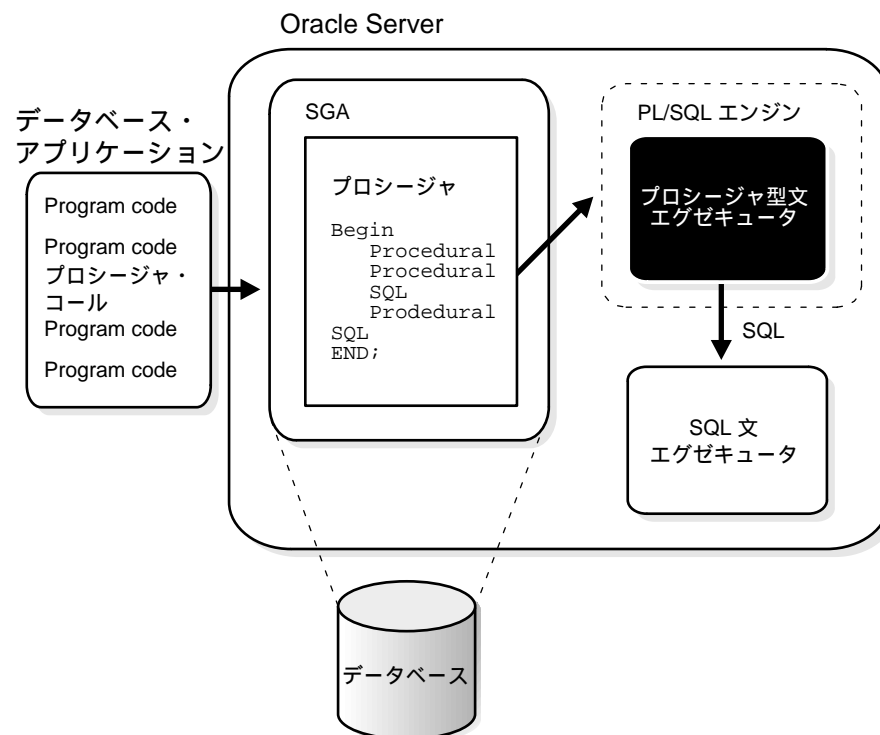
ストアド・プロシージャおよびファンクション、データベース・トリガーの詳細は、第 17 章「プロシージャとパッケージ」および第 18 章「データベース・トリガー」を参照してください。

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 を含む多くの Oracle 製品からなる特別なコンポーネントです。

図 14-2 に、Oracle Server に含まれている PL/SQL エンジンを図示します。

図 14-2 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 はコンパイル済みのプロシージャ（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協働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 の言語要素

PL/SQL ブロックには、次の PL/SQL 言語要素を組み込みます。

- 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。

追加情報：『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。

変数と定数

プロシージャまたはファンクション、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、必要なときに値を受け渡しするのに SQL 文や PL/SQL 文で使えます。

注意：Server Manager などの対話形式のツールを使うと、現行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えます。

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、プロシージャまたはファンクション、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・プロシージャを作っ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み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 エンジンを使って処理できるプロシージャとパッケージだけを対象にしています。

追加情報：それぞれの Oracle Tool の PL/SQL 機能の詳細は、Oracle Tool の該当するユーザーズ・ガイドを参照してください。

ストアド・プロシージャは、次のツールを使っ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。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。

追加情報：それぞれのタイプのアプリケーションからストアド・プロシージャをコールする方法は、『Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』または『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。

PL/SQL の動的 SQL

DBMS_SQL パッケージを使うことによって、動的 SQL が含まれるストアド・プロシージャや無名 PL/SQL ブロックを記述できます。動的 SQL 文は、ソース・プログラムに埋め込まれるのではなく、実行時にプログラムに入力される文字列、またはプログラムによって作られる文字列に格納されます。

これにより、汎用性の高いプロシージャを作れます。たとえば、動的 SQL を使うと、実行時までは名前がわからない表を操作するプロシージャを作れます。

また、DBMS_SQL パッケージを使って、データ操作言語 (DML) またはデータ定義言語 (DDL) の文を解析できます。この方法は、PL/SQL を使って DDL 文を直接解析できないという問題を解決するのに役立ちます。たとえば、DBMS_SQL パッケージによって提供される PARSE プロシージャを使って、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。

追加情報：動的 SQL の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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っ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。

追加情報：外部プロシージャの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

トランザクションの管理

The pigs did not actually work, but directed and supervised the others.

George Orwell: *Animal Farm*

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っ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基礎知識
- Oracle とトランザクションの管理
- 分散トランザクションの管理

トランザクションの基礎知識

「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）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。

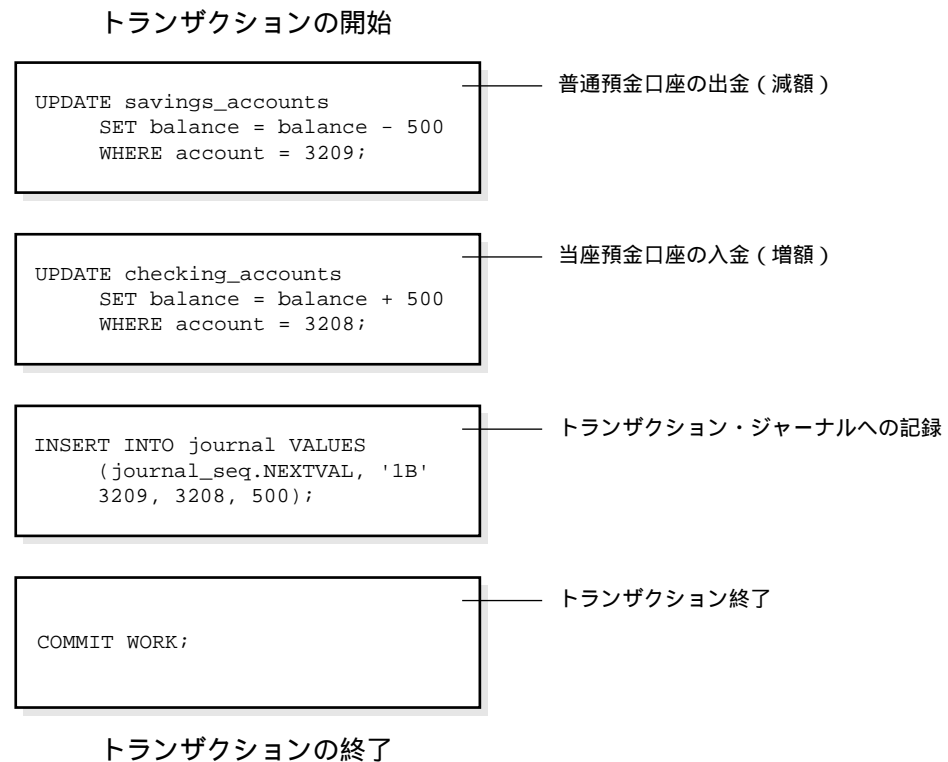
トランザクションは、最初の実行可能な SQL 文から開始します。トランザクションは、明示的に（COMMIT または ROLLBACK 文を使う）または暗黙のうちに（DDL 文を発行）コミットまたはロールバックされると、そこで終了します。

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しょう。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際、トランザクションは、普通預金口座の減額および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という 3 つの別々の操作で構成されます。

Oracle は、2 つの状況を考慮に入れます。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効果をデータベースに適用できます。しかし、なんらかの問題（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）が原因でトランザクション内の 1 つか 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 15-1 に銀行のトランザクションの例を示します。

図 15-1 銀行業務トランザクション



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、単一の文が解析されて有効な SQL 構造であることが確認され、文全体がアトミック単位としてエラーなしで実行されることを意味します（たとえば、複数行の更新ですべての行が更新された場合）。しかし、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り消せます。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、文に対して使います。

「コミット」とは、ユーザーが明示的または暗黙のうちに「このトランザクションの更新内容を確定しなさい」と指示することを意味します。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だけが、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かのような状態にすることです。これが「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た場合、文レベルのロールバックが発生します。（そのような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。）SQL 文の解析中にエラーが検出された場合（構文エラーなど）は、まだ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。23-16 ページの「デッドロック」を参照してください。

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、「現行の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失われません」。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。

注意：ユーザーは、ロールバック文の中の暗黙のセーブポイントの直接参照はできません。

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。詳細は、2-17 ページの「トランザクションとロールバック・セグメント」を参照してください。

トランザクションは、次のどれかの状況が発生すると終了します。

- COMMIT または ROLLBACK（SAVEPOINT 句なし）文が発行される。
- DDL 文（CREATE、DROP、RENAME、ALTER など）が実行される。現行のトランザクションに DML 文が含まれている場合、Oracle はまず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。（現行の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。(現行のトランザクションはロールバックされません。)

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。

トランザクションをコミットする

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る。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る。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もあります。
- SGA のデータベース・バッファに変更が加えられる。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もあります。

注意：コミットされたトランザクションによるデータ変更は、SGA のデータベース・バッファに格納されており、データベース・ライター (DBWn) のバックグラウンド・プロセスによって即座にデータ・ファイルに書き込まれるとは限りません。その書込みは、そのための最も効率的な時点に実行されます。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たりします。

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。

- 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、その表に記録される。
- ログ・ライター・プロセス (LGWR) 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る。また、LGWR は、

トランザクションの SCN もオンライン REDO ログ・ファイルに書き込みます。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。

- 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る。(ロックの詳細は、23-3 ページの「ロックのメカニズム」を参照してください。)
- Oracle により、トランザクションに「完了」のマークが付けられる。

バックグラウンド・プロセス LGWR および DBWR の詳細は、7-5 ページの「Oracle プロセス」を参照してください。

トランザクションをロールバックする

「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 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、トランザクションのうちコミットされていない後半部分をセーブポイントと呼ばれるマークーまでロールバックすることもできます。詳細は、15-7 ページの「セーブポイント」を参照してください。

次のすべてのタイプのロールバックで、同じ手順が使われます。

- 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
- 異常なプロセス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
- 回復のときの、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。

- 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・セグメントを使って取り消される。
- Oracle により、そのトランザクションのすべてのデータ・ロックが解放される (ロックの詳細は、23-3 ページの「ロックのメカニズム」を参照してください)。
- トランザクションが終了する。

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。

- Oracle により、セーブポイントの後に実行された文だけがロールバックされる。
- 指定されたセーブポイントは保持されるが、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る。

- Oracle により、そのセーブポイントの後に獲得された表と行のすべてのロックが解放されるが、そのセーブポイントより前に獲得されたすべてのデータ・ロックは保持される（ロックの詳細は、23-3 ページの「ロックのメカニズム」を参照してください）。
- トランザクションはアクティブなまま残る。つまり、継続できる。

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けます。

セーブポイントを使うと、大規模な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。そのようにするなら、後で、そのトランザクションの中で宣言されたセーブポイント以来、トランザクション内の現時点（トランザクションの最後）までの間に実行されたすべての作業を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うと、エラーになってもすべての文を再送信する必要がなくなります。

また、セーブポイントは、アプリケーション・プログラムの中でも同じように便利です。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を開始する前にセーブポイントを作れます。その後、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そのファンクションを再実行したり、回復作業を実行したり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・セグメントが取得したデータ・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。

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でもデータの一貫性が保たれなければなりません。

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、そうでなければすべてをロールバックするか、のどちらか一方だけ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。実際、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しているとは限りません。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文の構文は必要ありません。

リカバラ (RECO) バックグラウンド・プロセスによって、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) の結果は自動的に解決されます。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。

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継続する可能性のあるロックされた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回復しなければならない場合、Oracle の回復機能を使うことによって、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せます。これによって、グローバル・データベースは一貫した状態を保つことができます。

離散トランザクションの管理

アプリケーションの開発者は、プロシージャ `BEGIN_DISCRETE_TRANSACTION` を使うことによって、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ので、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。

離散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、離散トランザクションであってもそうでなくても、コミットされていない変更の参照はできません。

Oracle は REDO 情報を生成しますが、それをメモリーの別々の位置に格納します。トランザクションがコミット要求を発行すると、Oracle は REDO 情報を (他のグループ・コミットと一緒に) REDO ログ・ファイルに書き込み、データベース・ブロックに対する変更をそのブロックに直接適用します。Oracle は、コミットが完了すると、アプリケーションに制御を戻します。このことにより、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ので、取消し情報を生成する必要がなくなります。

追加情報：離散トランザクションの詳細は、『Oracle8 Server チューニング』を参照してください。

アドバンスト・キューイング

Many that are first shall be last; and the last shall be first.

Matthew 19:30: *The Bible*

この章では、Oracle アドバンスト・キューイング (Oracle AQ) 機能を説明します。この章の内容は、次のとおりです。

- メッセージ・キューイングの基礎知識
- Oracle Advanced Queuing
 - 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：

- Oracle8 製品を購入した場合は、Oracle AQ を使えません。
- 「オブジェクト・オプション」の付いていない Oracle8 Enterprise Edition 製品を購入した場合は、RAW データ型のキューでだけ Oracle AQ を使えます。
- 「オブジェクト・オプション」付きの Oracle8 Enterprise Edition 製品を購入した場合は、Oracle AQ のすべての機能を使えます。

Oracle8 Enterprise Edition で使用可能な機能とオプションの詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

追加情報: Oracle AQ の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

メッセージ・キューイングの基礎知識

プログラム間の通信は、次の 2 つのタイプのどちらかに分類できます。

- 同期通信（オンライン・モデルまたは接続モデル）
- 非同期通信（切断モデルまたは遅延モデル）

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」ともいう）は、作業を進めるために応答を受信しなければならない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、アプリケーションが動作するために、多くのプログラムが使用可能かつ実行中の状態でなければならないという点です。ネットワークまたはマシンの障害が発生したなら、プログラムは機能を停止してしまいます。

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、アプリケーションに対してデータを入力したり、操作を実行したりしなければならないことがあります。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は次の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、応答を待機の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、ちょうど 1 度だけ処理されなければなりません。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。

クライアント/サーバーの要求を「1 度だけ」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ために重要になることがよくあります。たとえば、特定の価格での株の買い注文が要求される場合、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、要求を実行しなかったり、2 回実行したりすることは許されません。

Oracle Advanced Queuing

Oracle Advanced Queuing (Oracle AQ) により、メッセージ・キューイング・システムと Oracle データベースが統合されます。これによって、Oracle Server による遅延取出しおよび遅延処理のために、メッセージをキューに格納できるようになります。

アプリケーションは、PL/SQL インタフェースを介してキューイング機能にアクセスできます。これにより、トランザクション処理 (TP) 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 AQ では、次の機能が提供されます。

- キュー要素の優先順位と順序
- 各キュー要素の実行の時間枠を指定する機能
- 標準 SQL を使っ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
- 複数のキュー要素をまとめてデキューする機能
- 複数の受信側を指定する機能
- 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波及させる機能
- 永続キューイング
- キュー内のメッセージに関する統計

Oracle AQ キューはデータベースの表内でインプリメントされるため、高い可用性、高い拡張性、そして高い信頼性といった操作上のすべての利点がキュー・データに適用されます。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えます。

キューイング・エンティティ

Oracle AQ には、メッセージ、キュー、キュー表、エージェント、キュー・モニターという 5 つの基本的なエンティティがあります。

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たりする情報の最小単位です。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。制御情報には、メッセージを管理するために Oracle AQ が使うメッセージ・プロパティが示されます。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 AQ 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・コールによって作られ、デキュー・コールによって消費されます。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（通常）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のキューです。キュー内のすべてのメッセージは同じデータ型でなければなりません。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できない場合、そのメッセージは例外キューに転送されます。

キューを作成および変更、開始、停止、削除するには、DBMS_AQADMパッケージを使います。

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。

キュー表を作ると、約25列を含むデータベース表が作られます。それらの列には、Oracle AQ 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つの索引が作られます。ビューを使うことにより、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います。

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、そして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@d-blink` 形式のアドレスを使う Oracle データベース・リンクだけです。

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および再試行、遅延を管理するメカニズムを提供し（16-5ページの「実行の時間枠」を参照）、これによって、間隔の統計を収集できます（16-7ページの「キューイングの統計」を参照）。

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_PROCESS` 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。

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。

構造化されたペイロード

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えます。(RAW データ型は、構造化されていないペイロードに使えます。)

統合されたデータベース・レベルの操作サポート

Oracle AQ では、メッセージは表の中に格納されます。回復、再起動、 Oracle Enterprise Manager などの標準データベース機能は、すべてサポートされています。

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 を使うことにより、メッセージのプロパティ、メッセージの履歴、そして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 テクノロジーを使うことにより、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。

実行の時間枠

特定の時間枠内にメッセージを消費してしまわなければならないことを指定できます。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、そして指定した時間制限が切れる前に消費しなければならないことをマークできます。

初期化パラメータ AQ_TM_PROCESS 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われます。間隔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(16-7 ページの「キューイングの統計」を参照)。

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 つ作ります。このパラメータを 2 ~ 10 に設定すると、Oracle はその数の QMNn プロセスを作ります。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・プロセスは作られません。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だけ有効になります。

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えます。

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してしまったら、現在の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獲得されます。

さらに、メッセージの相関識別子を使って特定のメッセージを取り出すこともできます。

メッセージの順序

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のに使うプロパティを指定する）、優先順位（各メッセージに割り当てられる）、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する）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、あるコンシューマはすぐに使え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おうとしているメッセージは飛ばします。

デキューのモード

DEQUEUE 要求により、メッセージをブラウズしたり取り消したりできます。メッセージをブラウズすると、それは将来の処理のために使える状態になります。メッセージを取り消すと、DEQUEUE 要求には使えなくなります。キューのプロパティに応じて、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、そして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

メッセージは、ちょうど1度だけ使われなければなりません。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る場合、メッセージは、ユーザー指定の遅延時間経過後に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、つまり実行時間枠あるいは再試行の制限内では、メッセージを使えないことがあります。そのような条件が生じたら、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。しかし、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ことによって、特定の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人のユーザーだけが使え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で作られなければならない、1つのトランザクションで作ら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。

保持

メッセージは、使用後もそれを保持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持できます。履歴は、追跡操作およびデータ・ウェアハウスの操作、データ探索操作のために使えます。

メッセージの履歴

Oracle AQ には、各メッセージの履歴の情報が入れられます。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、しばしばアプリケーションによって組み立てられる「イベント・ジャーナル」を表すものです。Oracle AQでは、アプリケーションが自動的にイベント・ジャーナルを作るように設計されています。

他のデータベースへのメッセージの波及

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波及できます。ソース・キューと宛先キューのデータ型は一致していなければなりません。

メッセージの波及により、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。波及では、ローカル・データベースとリモート・データベース間のデータベース・リンクと Net8 を使います。どちらでも Oracle AQ が有効化されていなければなりません。

メッセージ波及をスケジューリング（またはスケジュール解除）し、開始時間と波及の時間枠、そして後で定期的なスケジュールで行う波及の時間枠に対して日付機能を指定することができます。データ・ディクショナリ・ビュー DBA_QUEUE_SCHEDULES は、メッセージ波及の現行のスケジュールを示します。

ジョブ・キュー・バックグラウンド・プロセス（SNPn）は、メッセージの波及を処理します。波及を使用可能にするには、初期化パラメータ JOB_QUEUE_PROCESSES を使って少なくとも 1 つのジョブ・キュー・プロセスを起動しなければなりません。

キューイングの統計

Oracle AQ は、キューイング・システムの現行の状態に関する統計と、動的表 GV\$AQ における時間間隔統計を保持します。

キューイング・システムの現行の状態に関する統計には、準備完了および待ち状態、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（16-5 ページの「実行の時間枠」を参照）次のような間隔統計を保持する必要があります。

- 各状態（準備完了および待ち状態、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

インポート / エクスポート

キューをインポート/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/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。

複数の受取り側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、キュー表だけ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。しかし、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使い果たしたなど）が生じる場合、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。この識別子を使うことによって、特定のメッセージを取り出せます。

Oracle Enterprise Manager

Oracle Enterprise Managerは、キューの起動と停止、波及のスケジューリングとスケジュール解除、スキーマ・マネージャの一部としてのキュー・プロパティの表示といったキュー管理機能の一部を行うためのグラフィカル・ユーザー・インタフェース (GUI) を提供しています。

追加情報: Oracle AQの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

プロシージャとパッケージ

We're dealing here with science, but it is science which has not yet been fully codified by scientific minds. What we have are the memoirs of poets and occult adventurers...

Anne Rice: *The Tale of the Body Thief*

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基礎知識
- 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

プロシージャおよびファンクション、パッケージの間の依存性、Oracle がそれらの依存性を管理する方法の詳細は、第 19 章「Oracle の依存性の管理」を参照してください。

追加情報: Trusted Oracle を使っている場合は、Trusted Oracle のマニュアル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基礎知識

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・オブジェクトを使ってデータベース情報をアクセスしたり処理したりできます。プロシージャおよびファンクション、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、実行に使うツールです。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することができて、Oracle Server の PL/SQL エンジンを使っ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。詳細は、14-14 ページの「PL/SQL」を参照してください。

ストアド・プロシージャとファンクション

プロシージャおよびファンクションは、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まりを、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られ、繰り返し使うためにデータベースに格納されます。SQL*Plus などの Oracle Tool を使ってプロシージャやファンクションを対話的に実行したり、Oracle Forms や プリコンパイラのアプリケーションなどのデータベース・アプリケーションのコード、または別のプロシージャやトリガーのコードの中で明示的にコールしたりすることができます。

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、という点で違います。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

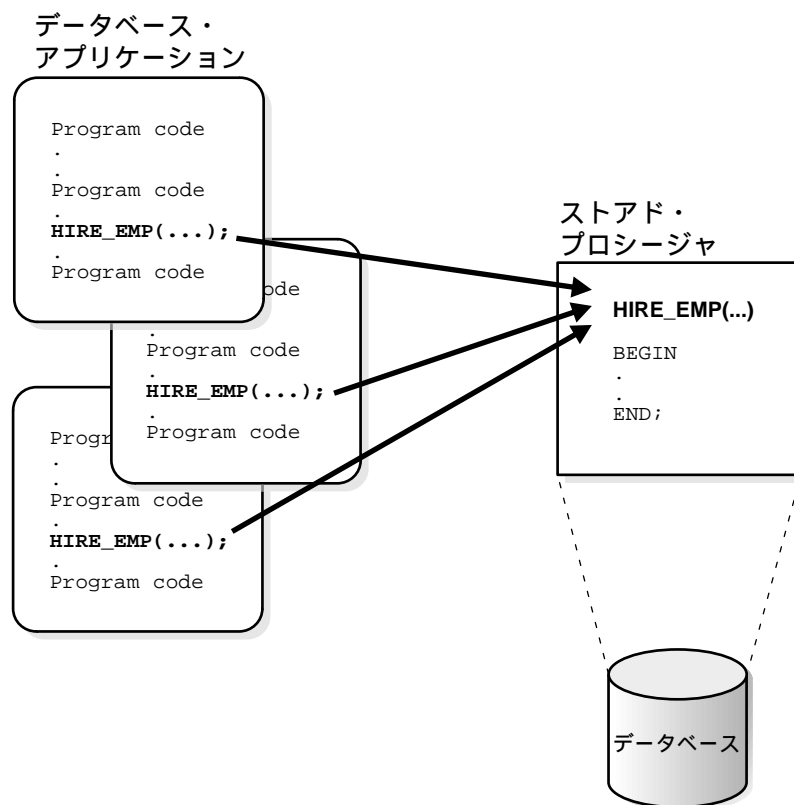


図 17-1のストアド・プロシージャは、従業員レコードをEMP表に挿入するものです。このプロシージャを図 17-2 に示します。

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Server Managerを使って、次の文によって HIRE_EMP プロシージャを実行することもできます。

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

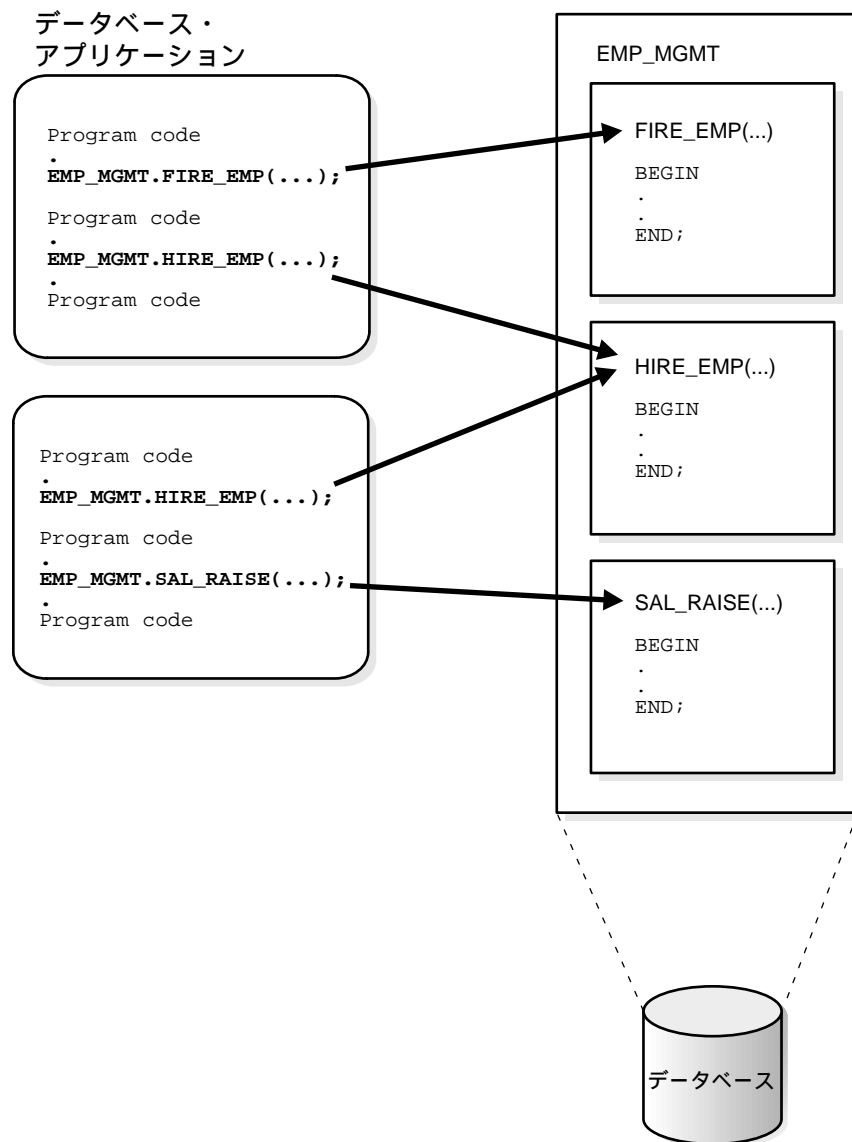
この文により、EMP 表に TSMITH のための新しい従業員レコードが挿入されます。

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、そしてこれらのプロシージャとファンクションが使うカーソルと変数をグループとしてまとめたもので、1つの単位として継続的に使え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・プロシージャおよびファンクションと同様、パッケージ・プロシージャおよびファンクションは、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を管理するために使う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、その中に入っ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 Server Manager を使って次の文を発行し、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。(17-9 ページの「パッケージ」を参照)

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」とは、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したりするために、1 つの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うと、SQL が持つ平易さや柔軟性と、構造型プログラミング言語が持つプロシージャ的な機能性を合体できます。

たとえば、次の文は、預金口座に入金する CREDIT_ACCOUNT プロシージャを作ります。

```
CREATE PROCEDURE credit_account  
    (acct NUMBER, credit NUMBER) AS  
/* このプロシージャの引数は 2 個： 口座番号 ( acct ) と  
   その口座番号に入金する金額 ( credit )  
   指定された口座が存在しないなら、新しくその口座が  
   作られる。 */  
  
    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。

プロシージャのガイドライン

どのストアド・プロシージャの設計と使用でも、次のガイドラインを守るようにしてください。

- 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する。いくつかの個別のサブタスクを含むような長いプロシージャを定義することは避けてください。多く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してしまうからです。
- 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。たとえば、宣言整合性制約を使えば簡単に施行できる単純なデータ整合性規則を施行するプロシージャは定義しないでください。

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。

セキュリティ

ストアド・プロシージャは、データ・セキュリティの強化に役立ちます。プロシージャとファンクションを介してのみデータにアクセスできるようにすることにより、ユーザーが実行できるデータベース操作を制限することができます。

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、表データの問合せまたは更新、削除の権限のないユーザーは、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。

パフォーマンス

ストアド・プロシージャを使うと、次の方法でデータベースのパフォーマンスを向上します。

- 情報は 1 回しか送信されず、それ以降は使うときに呼び出されるので、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 に送信したりする場合に比べて、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。

- データベースの中でプロシージャのコンパイル済み形式をすぐ使えるので、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ので、すぐ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ので、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つだけメモリーにロードすれば済みます。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の周り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 表の行を挿入、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、これらのタスクを実行するのに必要な 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。データ管理の方法が変わった場合、修正する必要があるのはプロシージャだけで、プロシージャを使う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の周りにすべ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ことをテストし、そのことの検証が終わった後は、いくつものアプリケーションでそのプロシージャとファンクションを再利用することができます。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が何かの点で変更された場合は、そのプロシージャだけ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。

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て、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ので、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、名前がない PL/SQL ブロックを Oracle Tools やアプリケーションから Oracle Server に送信することにより、無名 PL/SQL ブロックを作れます。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みバージョンが入れられます。しかし、そのソース・コードやコンパイル済みバージョンは、現行のインスタンスの後も再利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 を使うことによって、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、PL/SQL ブロックをデータベース・アプリケーションから、データベース内かメモリー内に格納されているデータベース・プロシージャに移すことにより、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。

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」といいます。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。(パッケージの利点の詳細は、17-9 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、プロシージャから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しなければなりません。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。

依存性の追跡の詳細は、第 19 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っ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。

追加情報: 外部プロシージャの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

パッケージ

パッケージは、互いに関連するいくつかのプロシージャおよびファンクション、それに関係するカーソルや変数を 1 つの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。

パッケージは本体と仕様部の 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」部分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る。パッケージ本体を実際には作らずに、仕様部を作り、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、パッケージ本体に入っているプロシージャ本体を変更できる。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が付けられることはありません。つまり、プロシージャの再コンパイルが必要であるとのマークが付けられることはありません。(依存性の詳細は、第 19 章「Oracle の依存性の管理」を参照してください。

次の例では、銀行業務の取引きを処理するいくつか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ります。

```
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

/* 銀行の取引きを入力するパッケージ */

    new_status CHAR(20); /* 適用される取引きの状態を記録する
                           グローバル変数。APPLY_TRANSACTIONS での更新に
                           使われる。 */

    PROCEDURE do_journal_entry (acct NUMBER,
                                kind CHAR) IS

/* APPLY_TRANSACTIONS プロシージャによって適用された
   各銀行取引きのジャーナル・エントリを記録 */

    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

/* 銀行口座に指定した金額を入金する。その銀行口座が存在しない場合は、
   このプロシージャで新しい口座を作る。 */

    old_balance NUMBER;
    new_balance NUMBER;

BEGIN
    SELECT balance INTO old_balance FROM accounts
    WHERE acct_id = acct
    FOR UPDATE OF balance; /* 入金処理のため口座をロック */

    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 /* 口座が存在しなければ、新しく作成 */
        INSERT INTO accounts (acct_id, balance)
        VALUES(acct, credit);
        do_journal_entry(acct, 'N');
    WHEN OTHERS THEN /* その他のエラーをアプリケーションに戻す */
        new_status := 'Error: ' || SQLERRM(SQLCODE);
END credit_account;

PROCEDURE debit_account (acct NUMBER, debit NUMBER) IS

/* 結果が、許容される最低残高より大きい場合は、
   既存の口座を負債口座にする */
```

```
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 /* その他のエラーをアプリケーションに戻す */
        new_status := 'Error: ' || SQLERRM(SQLCODE);
END debit_account;

PROCEDURE apply_transactions IS

/* TRANSACTIONS 表にある未処理の取引きを、
ACCOUNTS 表に適用。これを使って定期的に銀行口座を
更新し、新しい取引きの入力を妨げないようにする。 */

/* カーソルは、TRANSACTIONS 表のうち状態が 'Pending' の行をすべて
フェッチおよびロック。未処理の取引きをすべて
適用した後、ロック解放。 */

    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 /* 暗黙のオープン、フェッチ */
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;
/* TRANSACTIONS 表を更新し、
この取引を適用した結果を戻す。 */
UPDATE transactions SET status = new_status
WHERE CURRENT OF trans_cursor;
END LOOP;
COMMIT; /* TRANSACTIONS 表の行ロックを解除 */
END apply_transactions;

PROCEDURE enter_transaction (acct  NUMBER,
                             kind   CHAR,
                             amount NUMBER) IS

/* 銀行取引を TRANSACTIONS 表に入力。新しい取引は、
いったんこのキューに入れられてから、
APPLY_TRANSACTIONS プロシージャによって指定の口座に
適用される。したがって、多くの取引を、相互に妨げることなく、
同時に入力できる。 */

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の開発者は、パッケージを使うことによって、たくさん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。

パッケージの利点

パッケージは、互いに関連するいくつかのプロシージャおよび変数、カーソルを定義するために使います。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・プロシージャおよびプライベート・プロシージャ、変数、定数、カーソルの宣言
- すぐれたパフォーマンス

カプセル化

ストアド・プロシージャを使うと、関連するいくつかのストアド・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 つの単位としてデータベースに格納できます。このようにすると、開発プロセスでの効率が向上します。

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。パッケージを使う権限を付与すると、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

パッケージを定義する仕方により、変数およびカーソル、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る。

プライベート パッケージのユーザーから隠される。

たとえば、パッケージに 10 個のプロシージャが入っているとします。このうち、3 つだけをパブリックにしてパッケージのユーザーがそれらのプロシージャを実行できるようにし、残りのプロシージャはプライベートにしてパッケージ内のプロシージャだけがアクセスできるようするという形で、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と、PUBLIC に権限を付与することを混同しないようにしてください。詳細は、第 25 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たなら、すでにメモリーにあるコンパイル済み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したりできます。その結果、パッケージ仕様部も置き換えるのでない限り、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する）を再コンパイルする必要はありません。パッケージを使うと、不必要な再コンパイルが最小限に抑えられるので、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。依存性の追跡の詳細は、第 19 章「Oracle の依存性の管理」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する。
- コンパイル済みコードをメモリーに格納する。
- プロシージャまたはパッケージをデータベースに格納する。

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。

追加情報：コンパイル・エラーの識別の詳細は、『Oracle8 Server アプリケーション開発者ガイド』に説明があります。

コンパイル済みコードのメモリーへの格納

Oracle は、コンパイル済み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。このことにより、コードをすぐに実行でき、たくさん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みバージョンは、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させても、修正最低使用頻度アルゴリズムに従って共有プールに残ります。共有プール・バッファに固有の情報は、6-6 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 この名前を使って、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 文または CREATE FUNCTION 文、CREATE PACKAGE 文、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・コードの解析済み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、解析ツリーとオブジェクトの 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起動されたとき、現在それが SGA 内に存在しない場合は、そのプロシージャまたはパッケージのコンパイル済みのバージョンが PL/SQL エンジンによって SGA の共有ブル・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われます。

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データ・ディクショナリ (SYSTEM 表領域にある) 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、ユーザー・アクセスの有効性およびプロシージャの有効性が検査されてから、プロシージャが実行されます。

ユーザー・アクセスの検査

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だけです。

プロシージャの有効性の検査

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・ディクショナリによって調べます。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て以降に、次のいずれかの状況が発生した場合に無効になります。

- 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 (表、ビュー、他のプロシージャなど) が、変更または削除された (ユーザーが表に列を追加した場合など)

- 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、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。
- 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、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。

上記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、コンパイル済み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。

プロシージャとパッケージの有効と無効、およびプロシージャの再コンパイル、依存性の問題の詳細は、第 19 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る場合、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い場合、PL/SQL エンジン はコンパイル済みの 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 つの単位としてロードされます。

14-15 ページの図 14-2 「PL/SQL エンジンと Oracle Server」で示すとおり、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・トリガー

You may fire when you are ready, Gridley.

George Dewey: at the battle of Manila Bay

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、データベースに格納されているプロシージャであり、表を修正すると暗黙に実行（起動）されます。この章の内容は、次のとおりです。

- トリガーの基礎知識
- トリガーの各部分
- トリガーのタイプ
- トリガーの実行

追加情報: Trusted Oracleを使っている場合は、Trusted Oracleのマニュアルを参照してください。

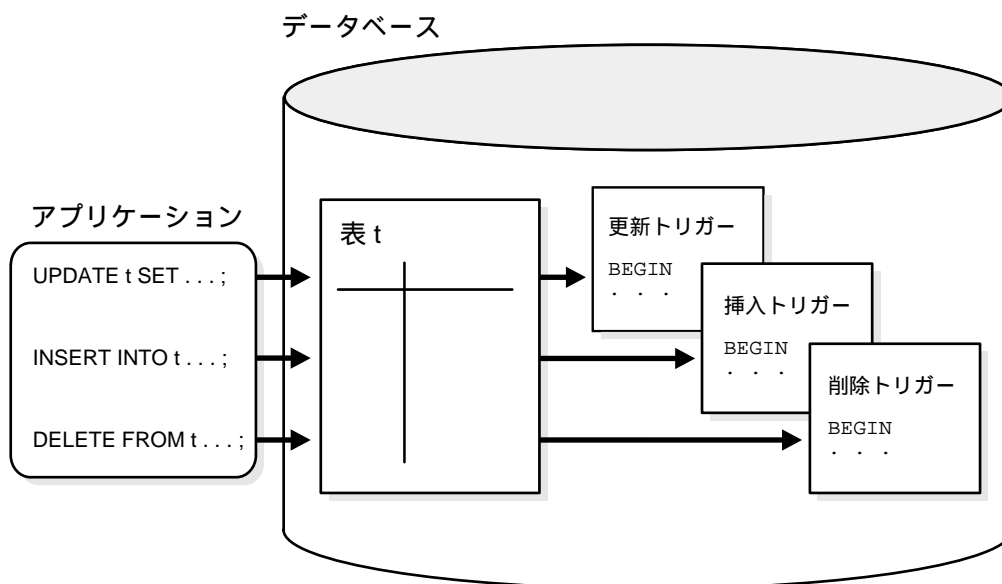
トリガーの基礎知識

Oracle では、表に対して INSERT、UPDATE、DELETE 文を発行すると暗黙のうちに実行されるプロシージャを定義できます。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。

トリガーは、第 17 章「プロシージャとパッケージ」で説明されているストアド・プロシージャと似ています。トリガーは、1 つの単位として実行可能な SQL や PL/SQL 文を収めたもので、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、ユーザーまたはアプリケーション、トリガーによって明示的に実行されます。トリガー（1 つまたは複数）は、接続しているユーザーや使われているアプリケーションには関係なく、トリガーの元となる INSERT または UPDATE、DELETE 文が発行された時点で暗黙のうちに起動（実行）されます。

図 18-1 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いくつか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。

図 18-1 トリガー



トリガーは、データベースの中に、それに対応付けられている表とは別に保管されます。

トリガーを定義できるのは表だけです。ビューには定義できません。ただし、ビューに対して INSERT、UPDATE、DELETE 文を発行すれば、そのビューの実表のトリガーが起動されます。

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だけに制限できます。また、トリガーを使うことによって、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施行
- 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施行
- 複雑なビジネス・ルールの施行
- 透明なイベント・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

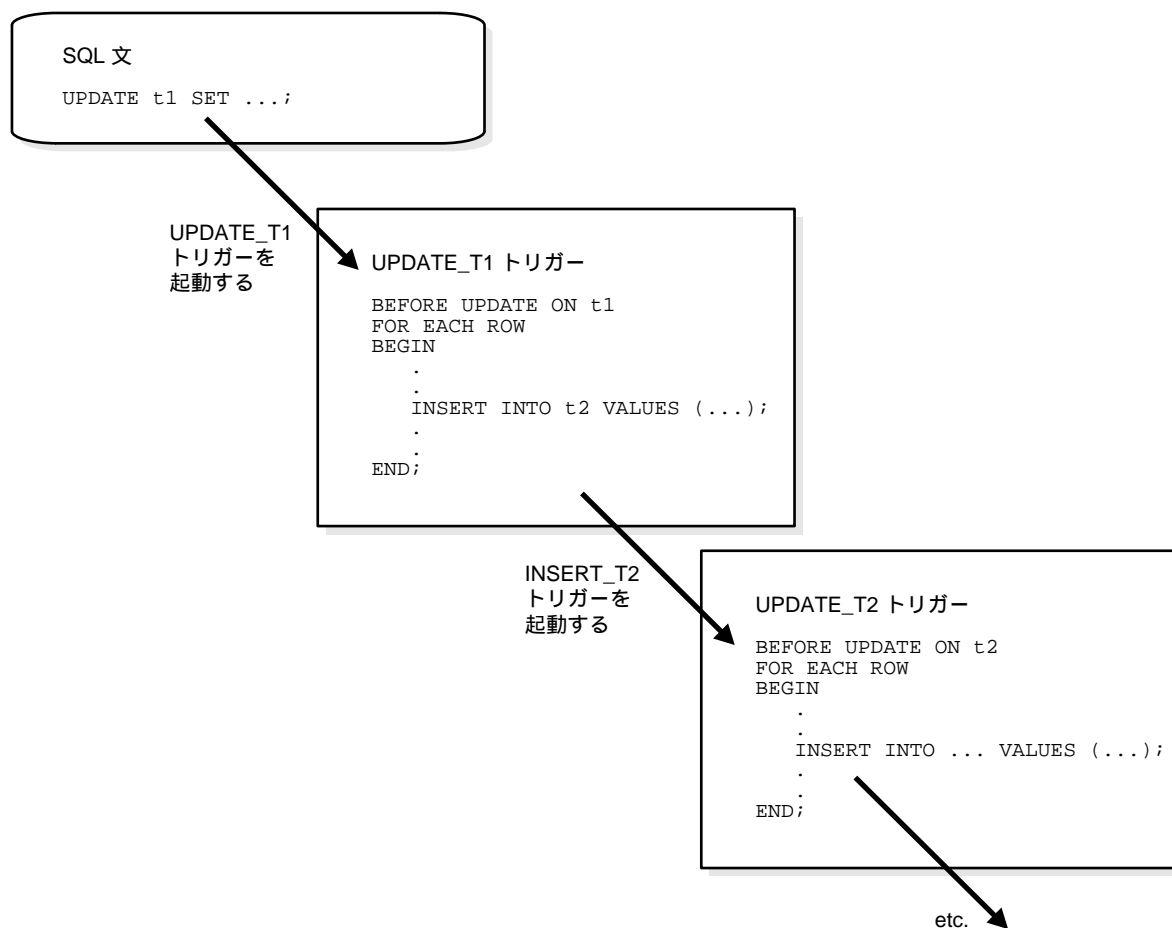
追加情報：これらのトリガーの多くの使用例が、『Oracle8 Server アプリケーション開発者ガイド』に記載されています。

トリガーの使用上の注意

トリガーはデータベースをカスタマイズするのに便利です。しかし、必要な場合以外は使わないようにしてください。トリガーを使いすぎると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、図 18-2 に示すように、トリガーを起動すると、そのトリガー・アクション内の SQL 文が他のトリガーを起動する場合があります。

トリガー本体の文が別のトリガーを起動するとき、これらのトリガーは「カスケード状態にある」といいます。

図 18-2 トリガーのカスケード



注意: Oracle Forms でも、少し種類の違うトリガーを定義および格納、実行できます。しかし、この章で説明するデータベース・トリガーと Oracle Forms トリガーを混同しないでください。

トリガーと宣言整合性制約

データベース・トリガーと整合性制約の両方を使うことによって、どのような整合性規則でも定義および施行できます。しかし、データベース・トリガーを使ってデータの入力を制約するのは、次の場合だけにしてください。

- 必要な参照整合性規則を、次の整合性制約を使って施行できない場合。
 - NOT NULL、UNIQUE キー
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK
 - 更新 CASCADE
 - 更新および削除 SET NULL
 - 更新および削除 SET DEFAULT
- 子表と親表が分散データベースの別々のノードにある場合に、整合性制約を施行する。
- 整合性制約では定義できない複雑な業務ルールを施行する。

整合性制約の詳細は、24-3 ページの「Oracle がデータの整合性を施行する方法」を参照してください。

トリガーの各部分

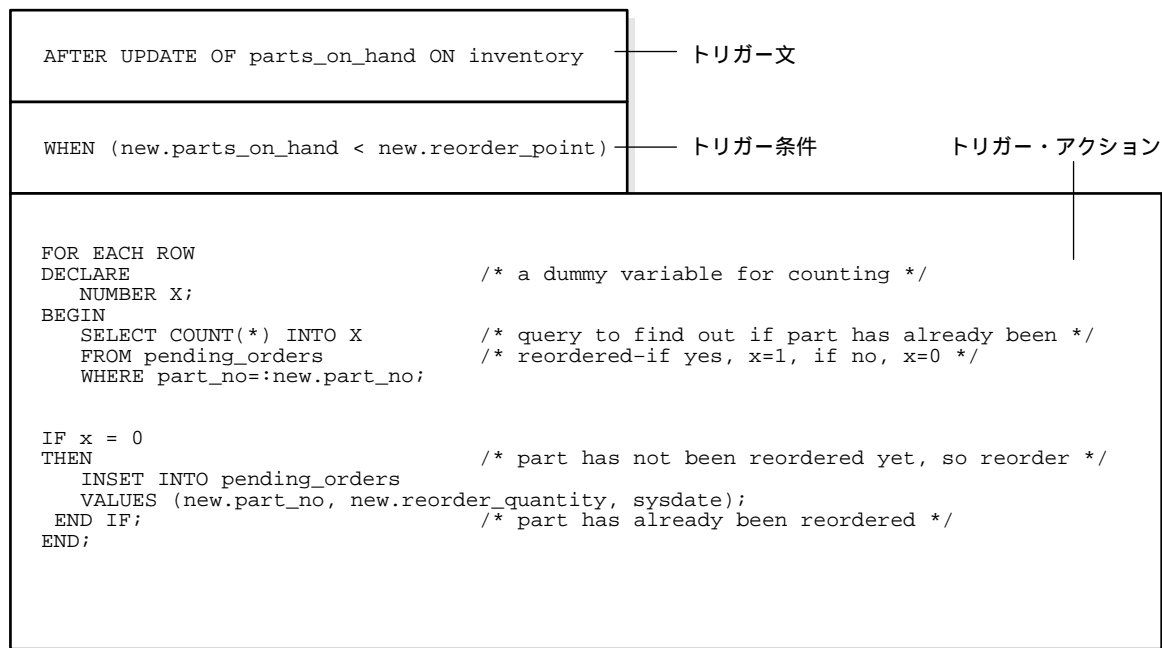
トリガーには次の3つの基本部分があります。

- トリガー・イベントまたはトリガー文
- トリガー条件
- トリガー・アクション

図 18-3 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 18-3 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文

トリガー・イベントまたはトリガー文とは、トリガーを起動させる SQL 文のことです。トリガー・イベントには、表に対する INSERT 文、UPDATE 文、DELETE 文があります。たとえば、図 18-3 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、トリガーが起動され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 DML 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文またはUPDATE文、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DML文でトリガーが起動される場合は、条件述語を使ってトリガー文のタイプを検出できます。それで、トリガーを起動させ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れます。

トリガー条件

トリガー条件にはブール（論理）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にならなければなりません。トリガーの評価が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
```

トリガー・アクション

トリガー・アクションは、SQL文とPL/SQLコードを含むプロシージャ（PL/SQLブロック）です。これらのSQL文やPL/SQLコードは、トリガー文が発行され、トリガー条件がTRUEと判断されたときに実行されます。

ストアド・プロシージャと同じように、トリガー・アクションでもSQL文とPL/SQL文を使ったり、PL/SQLの言語要素（変数、定数、カーソル、例外など）を定義したり、ストアド・プロシージャをコールしたりできます。さらに、行トリガーでは、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現在行の列値（新しい値と古い値）にアクセスできます。各列の古い値と新しい値には、2つの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、さまざま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

行トリガーと文トリガー

トリガーを定義する際、トリガー・アクションの実行回数を指定できます。つまり、トリガー文によって処理されるすべての行ごとに1回ずつ実行するのか（複数行を更新するUPDATE文によって起動された場合など）または、トリガーに関係する行の数とは無関係にトリガー文ごとに1回だけ実行するのかを指定します。

行トリガー

行トリガーは、表がトリガー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 18-3 は、トリガーを実行する文によって影響を受ける各行の値を使う行トリガーの一例です。

文トリガー

文トリガーは、トリガー文によって影響を受ける表の行数とは関係なく（影響を受ける行がない場合でも）トリガー文によって1回だけ実行されます。たとえば、DELETE 文が表の複数の行を削除した場合、表から削除された行数とは無関係に、文レベルの DELETE トリガーが1回だけ起動されます。

トリガー・アクションのコードが、トリガー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しない場合には、文トリガーが便利です。たとえば、トリガーが現在の時刻や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、あるいはトリガーがトリガー文のタイプに基づいて単一の監査レコードを生成する場合に、文トリガーを使います。

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。（18-11 ページの「INSTEAD OF トリガー」では、また別の種類のトリガーについて説明しています。）

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います。

- 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 トリガーを使うことにより、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ロールバックするということを防げます。
- INSERT トリガー文や UPDATE トリガー文を実行する前に、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。AFTER トリガーは、次のような場合に使います。

- トリガー・アクションを実行する前に、トリガー文の実行を完了したい場合。
- BEFORE トリガーがすでに存在する場合、AFTER トリガーは、同じトリガー文に対して異なるアクションを実行できます。

トリガーの組合せ

これまでに説明したオプションを使って、次の4つのタイプのトリガーを作れます。

- 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 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。

- 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現在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せん。

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 行トリガーを使うため、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、DELETE) に対して、上記のトリガーを必要な数だけ作れます。

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したい場合を考えます。下記の例は、表 SAL に対するアクション (UPDATE、DELETE、INSERT など) の時間とタイプごと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分されます。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;
```

トリガーのタイプ

```
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

INSTEAD OF トリガーを使うと、SQL DML 文 (INSERT および UPDATE、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 トリガーといいます。このトリガーは、基礎となる表に対して、更新操作または挿入操作、削除操作を直接に実行します。

ビューに対して通常の INSERT および DELETE、UPDATE 文を作成できます。INSTEAD OF トリガーが表面に現れることなくバックグラウンドで動作することにより、正しいアクションがとられます。デフォルトでは、INSTEAD OF トリガーは行ごとにアクティブにされます。

ビューの変更

ビューを変更する操作には、曖昧さの問題があります。

- ビューで行を削除するという操作は、実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対して選択されないようにする操作のどちらかを意味しています。
- ビューに行を挿入するという操作は、実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ています。
- 結合が含まれるビューで列を更新すると、ビューに表示されて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 (第 13 章「オブジェクト・ビュー」を参照)。たとえば、オブジェクト・ビューの主な使用法は、マスター/ディテールの関連を表すことです。ここ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 (7 章を参照)。INSTEAD OF トリガーでは、それ以外の手段では変更できないリレーショナル・ビューだけではなく、オブジェクト・ビューに対しても使えます。

INSTEAD OF トリガーのメカニズムにより、OCI を使ってクライアント側からオブジェクト・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・ビューによって具体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。しかし、オブジェクトが読み専用であれば、オブジェクトを確保するためのトリガーを定義する必要はありません。

追加情報: 詳細は、『Oracle コール・インタフェース・プログラマーズ・ガイド』を参照してください。

変更不可能なビュー

INSTEAD OF トリガーを使わずにビューを挿入または更新、削除でき、かつビューが下に示した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要素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または更新、削除を実行できません。

- 集合演算子
- グループ・ファンクション
- GROUP BY、CONNECT BY、または START WITH 句
- DISTINCT 演算子
- 結合（ただし、結合ビューのサブセットは更新可能です。8-13 ページの「更新可能な結合ビュー」を参照してください。）

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 文を使う方法しかありません。

INSTEAD OF トリガーの例

次の例は、行を MANAGER_INFO ビューに挿入するための INSTEAD OF トリガーを示しています。

```
CREATE VIEW manager_info AS
  SELECT e.name, e.empno, d.dept_type, d.deptno, p.level,
         p.projno
  FROM emp e, dept d, project p
  WHERE e.empno = d.mgr_no
  AND d.deptno = p.resp_dept;

CREATE TRIGGER manager_info_insert
  INSTEAD OF INSERT ON manager_info
  REFERENCING NEW AS n          -- new manager information

  FOR EACH ROW
  BEGIN
    IF NOT EXISTS SELECT * FROM emp
      WHERE emp.empno = :n.empno
    THEN
      INSERT INTO emp
        VALUES(:n.empno, :n.name);
    ELSE
      UPDATE emp SET emp.name = :n.name
        WHERE emp.empno = :n.empno;
    END IF;

    IF NOT EXISTS SELECT * FROM dept
      WHERE dept.deptno = :n.deptno
```

```
THEN
    INSERT INTO dept
        VALUES(:n.deptno, :n.dept_type);
ELSE
    UPDATE dept SET dept.dept_type = :n.dept_type
        WHERE dept.deptno = :n.deptno;
END IF;

IF NOT EXISTS SELECT * FROM project
    WHERE project.projno = :n.projno
THEN
    INSERT INTO project
        VALUES(:n.projno, :n.project_level);
ELSE
    UPDATE project SET project.level = :n.level
        WHERE project.projno = :n.projno;
END IF;
END;
```

MANAGER_INFO ビューに挿入される行のアクションでは、最初に MANAGER_INFO の基礎になっている実表の中に該当する行がすでに存在しているかどうかを確認しています。次にこのアクションは、必要に応じて新しい行を挿入するか、既存の行を更新します。UPDATE および DELETE についても、類似のトリガーを使って適切なアクションを指定できます。

追加情報: INSTEAD OF トリガーの詳細は、『Oracle8 Server SQLリファレンス』の CREATE TRIGGER コマンド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。

使用可能トリガーの場合、Oracle は次の処理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する。

- 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する。
- 問合せと制約に対して、読み込み一貫性を保証する。
- 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する。
- 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・コミットを使います。
- 特定の1つの文について同じタイプのトリガーが1つ以上存在する場合、順序不定で各トリガーが起動される。

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、BEFORE 行トリガーおよび BEFORE 文トリガー、AFTER 行トリガー、AFTER 文トリガーという、最大 4 つのトリガーを起動する可能性があります。トリガー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 (カスケード・トリガー)。

Oracle では、次の実行モデルを使って、複数のトリガーと制約チェックの順序を維持します。

1. その文に適用されるすべての BEFORE 文トリガーを実行する。
2. その SQL 文の影響を受ける各行をループする。
 - a. その文に適用されるすべての BEFORE 行トリガーを実行する。
 - b. 行をロックして変更し、整合性制約チェックを実施する。(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ない)。
 - c. その文に適用されるすべての AFTER 行トリガーを実行する。
3. 遅延整合性制約チェックを完了する。
4. その文に適用されるすべての AFTER 文トリガーを実行する。

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 文が発行される。
2. BEFORE 行トリガーが起動される。
3. BEFORE 行トリガー内の UPDATE 文により、AFTER 文トリガーが起動する。
4. AFTER 文トリガーの文が実行される。
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る。

6. BEFORE 行トリガーの文が実行される。
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る。
8. SQL 文が実行される。
9. SQL 文から整合性制約がチェックされる。

実行モデルの重要な特性は、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しなければならないということです。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、トリガーによって起動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、トリガーによって整合性制約が破られることはあり得ません。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）、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：各種タイプのトリガーは、特定の順序で起動することに注意してください。ただし、同一の文に対して同じタイプのトリガーが複数ある場合は、確実に特定の順序で起動されることは保証できません。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、参照先の表の読み込み一貫性のある現行の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：

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 列を更新する操作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 に由来する) 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。それで、そのトリガーの問合せは、トランザクション T2 の読み込み一貫性ポイントの値として、MAXSAL の古い値を戻します。

例：

TOTAL_SALARY トリガーが次のように定義されているとします。このトリガーは、ある部門のメンバー全員の給与総額を保存するための導出列をメンテナンスします。

```
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 列を更新したとします。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操作が含まれている（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。

トリガーの記憶領域

トリガーは、ストアド・プロシージャと同じように、コンパイル済み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み PL/SQL 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。

PL/SQL コードのコンパイルと格納の詳細は、17-15 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ってトリガーを内部的に実行します。両者の唯一のわずかな相違点は、ユーザーにトリガーを起動する権限が付与されるのは、トリガー文を起動する権限のあるユーザーである場合だということです。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。

詳細は、17-16 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」を参照してください。

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。

詳細は、第 19 章「Oracle の依存性の管理」を参照してください。

Oracle の依存性の管理

Whoever you are - I have always depended on the kindness of strangers.

Tennessee Williams: *A Streetcar Named Desire*

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（たとえば、表）が参照されます。したがって、定義するオブジェクトは、その定義で参照してい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・管理する方法を説明します。この章の内容は次のとおりです。

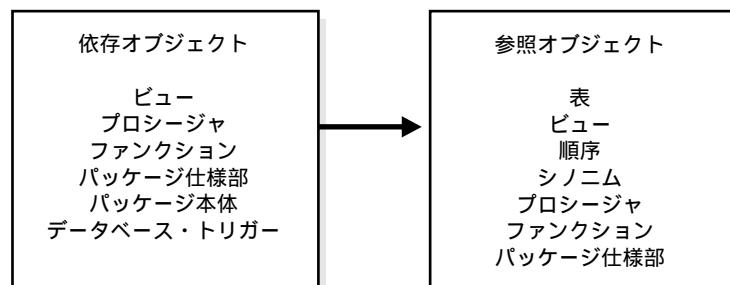
- 依存性の問題の基礎知識
- スキーマ・オブジェクトの依存性の解決
- 依存性の管理と存在しないスキーマ・オブジェクト
- 共有 SQL の依存性管理
- ローカルとリモートの依存性の管理

追加情報: Trusted Oracle を使っている環境でのスキーマ・オブジェクトの依存性の詳細は、Trusted Oracle のマニュアルを参照してください。

依存性の問題の基礎知識

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」、参照されるオブジェクトを「参照オブジェクト」といいます。図 19-1 に、さまざまなタイプの依存オブジェクトと参照オブジェクトを示します。

図 19-1 さまざまな依存スキーマ・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えません。

Oracle ではオブジェクト間の依存性が自動的に記録されるので、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（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	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えます。
-------	----------------------------------

INVALID

オブジェクトを使うには、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、パッケージについては、オブジェクトをコンパイルする必要があるということです。また、ビューの場合は、データ・ディクショナリ内の現在の定義を使ってビューを再解析する必要があることを意味します。依存オブジェクトだけがINVALIDになり得ます。つまり、表、順序、シノニムは常にVALIDです。

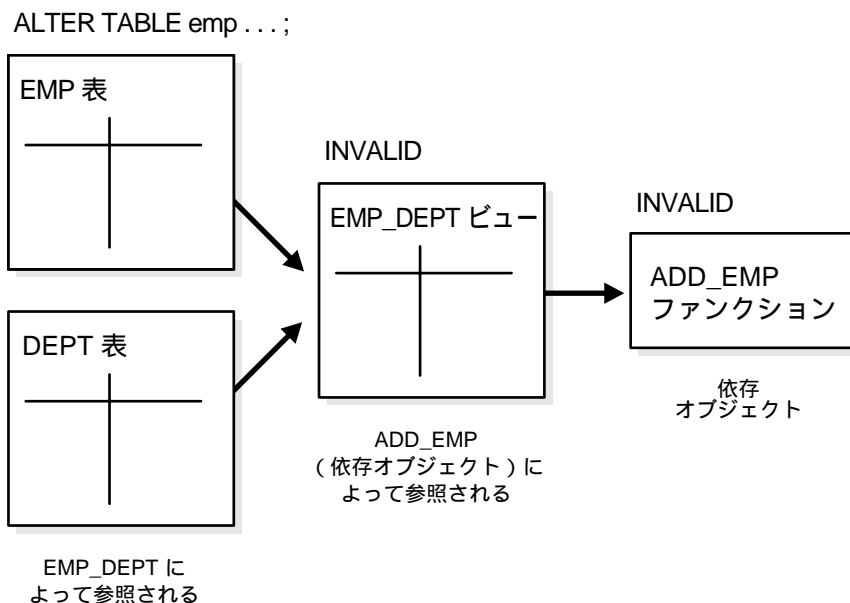
ビューまたはプロシージャ、ファンクション、パッケージがINVALIDである場合、Oracleはそのオブジェクトをコンパイルしようとしたが、そのオブジェクトに関連したエラーが発生した、ということです。たとえば、ビューのコンパイル時に、実表の1つが存在しなかったり、実表の正当な権限がなかったりする場合があります。パッケージのコンパイル時には、PL/SQLまたはSQLの構文エラーが発生したり、参照されているオブジェクトについての適切な権限が存在しなかったりする場合があります。このような問題があるオブジェクトはINVALIDのままです。

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の変更の場合は、依存オブジェクトに対する直接的な変更だけでなく、間接的な変更も記録されます。

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てみましょう。ストアド・プロシージャは、このビューの実表を間接的に参照します。このため、実表を変更すればビューが無効となり、さらにストアド・プロシージャも無効となります。図 19-2 に、そのことを示します。

図 19-2 間接的な依存性



スキーマ・オブジェクトの依存性の解決

(SQL 文で直接的に、または依存オブジェクトの参照によって間接的に) スキーマ・オブジェクトを参照する際に、SQL 文と参照オブジェクトに明示的に指定されているオブジェクトの状態は、必要に応じて Oracle によってチェックされます。Oracle のアクションは、SQL 文で直接または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、付加的な処理なしで SQL 文が即時に実行されます。
- 参照される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せん。
 - 無効な状態の参照オブジェクトがすべて正常にコンパイルされた場合、そのオブジェクトは問題なくコンパイルされ、SQL 文は正常に終了します。
 - 無効な状態の参照オブジェクトを正常にコンパイルできなかった場合、そのオブジェクトは無効な状態のままで、エラーが発生し、SQL 文を含んでいるトランザクションはロールバックされます。

注意: Oracle は、無効であると検出されて以来置き換えられていないオブジェクトだけ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は実行されません。

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なら、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 文が適切に構成されていなければなりません。
- 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、その列が実表に存在していなければなりません。
- 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 権限がなければなりません。

ビューと実表

ビューは、そのビューを定義している問合せで参照されている実表（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場合、定義の問合せはデータ・ディクショナリへの格納時に展開され、参照先の実表内の現行のすべての列に含まれるようになります。

ビューの実表（またはビュー）を変更、改名、削除すると、そのビューは無効になり、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、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）なビューを使おうとすると、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。

- ビューの問合せの定義が参照しているすべての実表が存在していなければなりません。ビューの実表を改名または削除すると、そのビューは無効になり、使えなくなります。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、実表を元の名前に改名した場合、または実表を作り直した場合だけです。
- 実表を変更したり、同じ列を含む実表を作りなおしたりした場合に、その実表の 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。
- ビューの実表が、少なくとも同じ列セットを含むような仕方に変更された場合、またはそのように作りなおされた場合、そのビューは有効にすることができます。実表が新しい列を含むように作りなおされ、その結果の表の中にビュー定義で参照していた列が存在しない場合、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。つまり、そのような問合せ

の定義はビューを作るときにデータ・ディクショナリ内で展開され、そのまま永久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る。
- 参照ビューの実表を変更すると、ビューと依存プロシージャが無効になる。
- 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る。
- 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影響はない。しかし、参照スタンドアロン・プロシージャ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、パッケージを使う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明らかにしています。

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、そのパッケージのそれぞれ独自のインスタンスが存在します。それには、パブリック変数とプライベート変数およびカーソル、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（仕様部または本体）のど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（状態も含む）が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、セキュリティ認可だけを検証し、有効にする必要があります。このように最適化すれば、不要な再コンパイルは実行されず、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。

追加情報：無効なビューまたはプログラム・ユニットの強制的な再コンパイルの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。Trusted Oracle を使っている場合は、Trusted Oracle のマニュアルを参照してください。

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られると、Oracle は最初に現行スキーマを検索してすべての参照を解決しようとします。現行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。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。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。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。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。このような現象は、依存オブジェクトが使っている参照が、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る。
- COMPANY.EMP のために EMP という名前の PUBLIC シノニムが作られており、COMPANY.EMP に対する SELECT 権限が PUBLIC ロールに付与されている。
- 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ない。
- ユーザー JWARD は、次の文を使って自分のスキーマ内にビューを作る。

```
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 ビューを作る際、EMP への参照を解決するために、表、ビュー、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。それは見つからないため、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、それは見つかります。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。

ここで JWARD が、次の文を使って EMP という名前のビューを自分のスキーマ内に作るとします。

```
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、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、仮にそのオブジェクトが存在したとすると、新しく作ろうとしているオブジェクトの定義の解釈が変わってしまうようなスキーマ・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ら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覚えておく必要があります。存在しないはずのオブジェクトが実際に作られた場合、その依存オブジェクトを再コンパイルして有効にするためには、すべての依存オブジェクトを無効にする必要があります。

したがって、上記の例で、JWARD.EMP が作られると、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 が使われた時点で、Oracle はこの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）。しかし、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られた場合に備えてそのオブジェクトに対する依存性を管理しなければならない、ということです。

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、ビュー、シノニム、順序を生成または変更、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したりすると、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。

ローカルと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。最も単純な場合、Oracle は、単一データベース内のオブジェクトの間の依存性を管理します（ローカル依存性管理）。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。もっと複雑なシステムの場合、Oracle は、ネットワーク全体にわたる分散環境の中での依存性を管理する必要があります（リモート依存性管理）。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・ビューを定義する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」表を使っ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、この依存表を使っ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 を参照するとします。この表の定義を変更すると、JWARD.EMP を参照するすべてのオブジェクトの状態は、プロシージャ UPDATE_SAL を含めて、無効に変更されます。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。しかし、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

アプリケーションからデータベースへの依存性、またアプリケーションから分散データベースへの依存性も管理しなければなりません。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ることがあり、また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われます。

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

分散データベース・システム内のストアド・プロシージャ（ファンクション、パッケージ、トリガーなど）の相互間の依存性は、「タイム・スタンプのチェック」または「署名のチェック」を使って管理します。

動的初期化パラメータ REMOTE_DEPENDENCIES_MODE は、タイム・スタンプと署名のどちらでリモート依存性を管理するかを決定します。

追加情報：タイム・スタンプまたは署名を使ったリモート依存性の管理の詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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、データ・ディクショナリ内にその「タイムスタンプ」（作成、変更、置換の時刻）が記録されます。さらに、コンパイル済みプロシージャには、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われると、Oracle は、コンパイル時に記録されたりリモート・スタンプと、リモートで参照されたプロシージャの現在の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れば、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い場合、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・プロシー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だけが実行されます。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）。しかし、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。

署名のチェック

Oracle には、署名を使ったりリモート依存性の付加的な機能が用意されています。この署名機能は、リモート依存性にしか関係しません。ローカル依存性（同じサーバー）は、その環境でいつでも再コンパイルできるため、関係しません。

プロシージャの署名には、次の情報が含まれます。

- パッケージまたはプロシージャ、ファンクションの名前
- パラメータの基礎型
- パラメータのモード（IN、OUT、IN OUT）

注意：重要なのはパラメータの型とモードだけです。パラメータの名前は署名には影響しません。

署名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いて、かつそのプロシージャの署名が互換性のない方法で変更されていると、リモート・プログラム・ユニット（パッケージまたはストアド・プロシージャ、ストアド・ファンクション、トリガー）への依存性により、その依存単位が無効になる場合があります。

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以外のリモート・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・ビューを作り、定義とします。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われ、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しなければなりません）。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ほうがよいでしょう。

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 およびプリコンパイラ、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別およびオペレーティング・システム別の該当するマニュアルを参照してください。

*I do the very best I know how - the very best I can;
and I mean to keep doing so until the end.*

Abraham Lincoln

この章では、SQL文の実行方法をどのように Oracle オブティマイザが選択するのかについて説明します。この章の内容は次のとおりです。

- 最適化とは？
 - 実行計画
 - 実行の順序
- コストベース最適化とルールベース最適化
- オブティマイザの操作の概要
 - 式と条件の評価
 - 文の変換と最適化
 - 最適化アプローチと目標の選択
 - アクセス・パスの選択
 - 結合文の最適化
 - 非結合と半結合の最適化
 - 「スター」問合せの最適化

追加情報: Oracleオブティマイザの詳細は、『Oracle8 Serverチューニング』を参照してください。

最適化とは？

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 または INSERT、UPDATE、DELETE などのデータ操作言語（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、実行方法がさまざまになることがあります。文を実行するときに使う手順は、文の実行速度に大きく影響します。

Oracle の「オブティマイザ」と呼ばれる部分は、最も効率がよいと考えられる方法を選択します。オブティマイザは、代替アクセス・パスの中から選択する要因の数を調べます。特定のアプリケーションのデータについて、オブティマイザよりもさらに詳細な点を知っているアプリケーション・デザイナーが、SQL 文を実行するためのさらに効率的な方法を選択することもよくあります。アプリケーション・デザイナーは、SQL 文内にヒントを使って文の実行方法を指定できます。

注意：オブティマイザは、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が決定されるようになってきています。

追加情報：SQL 文でのヒントの使用方法は、『Oracle8 Server チューニング』を参照してください。

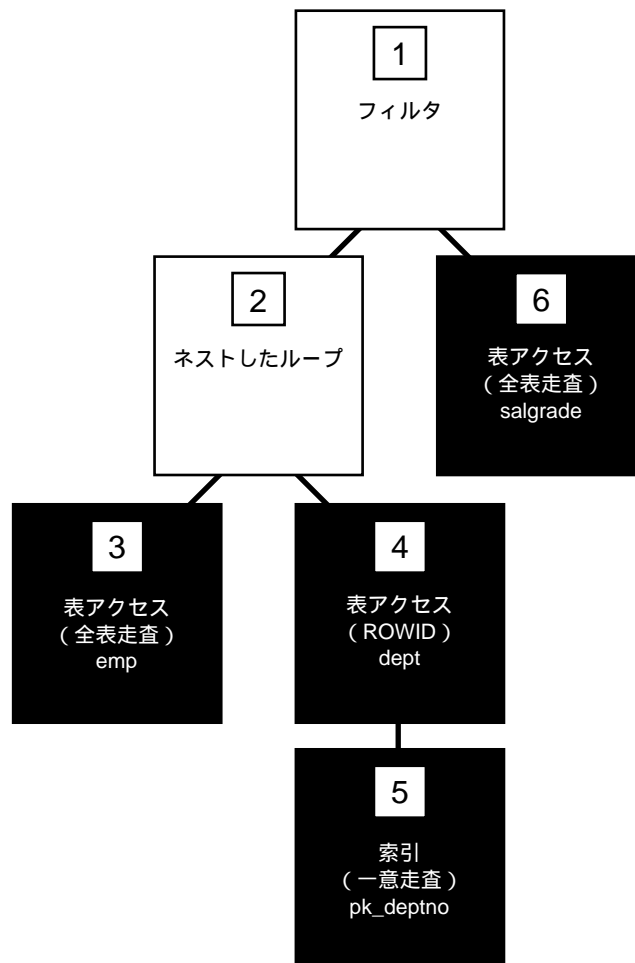
実行計画

DML 文を実行するため、Oracle 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 が文を実行するのに使うステップの組合せを、「実行計画」といいます。

図 20-1 に、次の SQL 文の実行計画を図示します。この文は、給与が基準範囲内にないすべての従業員の名前（ename）、職種（job）、給与（sal）、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 20-1 実行計画



実行計画のステップ

実行計画の各ステップは、次のステップで使われ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」といいます。

図 20-1 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 コマンド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違っています（20-5 ページの「実行の順序」を参照）。

実行計画の各ステップは、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」といいます。
 - ステップ 3 とステップ 6 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 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 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は、行ソースを操作します。
 - ステップ 2 は、ネストした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 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。

アクセス・パスの詳細は、20-40 ページの「アクセス・パスの選択」を参照してください。Oracle が行ソースを結合するのに使う方法の詳細は、20-60 ページの「結合操作」を参照してください。

EXPLAIN PLAN コマンド

EXPLAIN PLAN コマンドを使うと、オブ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。このコマンドは、オブ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、前の項で考え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 20-1 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 20-1 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 コマンドを使った後、その出力表に問い合わせることによって取得できます。

追加情報：EXPLAIN PLAN の使用方法と、その出力の生成方法および解釈方法の詳細は、『Oracle8 Server 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 20-1 のステップ 3、5、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、Oracle は親ステップを実行します。

たとえば、図 20-1 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。

- Oracle はステップ 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、Oracle は次のようなステップを実行します。
 - Oracle はステップ 5 を実行し、結果の ROWID をステップ 4 に戻します。
 - Oracle はステップ 4 を実行し、結果の行をステップ 2 に戻します。
 - Oracle はステップ 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。
 - Oracle はステップ 6 を実行し、結果の行があればステップ 1 に戻します。
 - Oracle はステップ 1 を実行します。ステップ 6 から行が戻されない場合、Oracle は 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 5、4、2、6、および 1 は、ステップ 3 から 1 行戻されるたびに 1 回実行されることに注意してください。1 つの親ステップが実行されるために、その前に子ステップから行が 1 つ戻される必要がある場合には、Oracle は子ステップから 1 行戻されるとすぐに、親ステップを（そしておそらく残りの実行計画も）実行します。行が 1 つ戻されれば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。

このように、ツリーを順次さかのぼることにより、おそらく実行計画の残りの部分すべてを実行していけます。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。子ステップからそれぞれの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、ネスト・ループ結合、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、Oracle は親ステップを実行できません。そのような親ステップとしては、ソートおよびソート・マージ結合、グループ・ファンクション、集約操作があります。

コストベース最適化とルールベース最適化

オブティマイザが SQL 文の実行計画を選択する際、コストベースのアプローチかルールベースのアプローチのどちらかを使います。

コストベースのアプローチ

オブティマイザがコストベースのアプローチを使う場合、使用可能なアクセス・パスと、その文がアクセスするスキーマ・オブジェクト（表またはクラスタ、索引）に関するデータ・ディクショナリに含まれている統計情報の要素を考慮して、効率が最高の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. オブティマイザは、その使用可能なアクセス・パスとヒントに基づいて、文の実行計画のセットを生成します。
2. オブティマイザは、データ・ディクショナリにある表およびクラスタ、索引のデータ分布と記憶特性の統計に基づいて、それぞれの実行計画のコストを推定します。

「コスト」は、実行計画を使っ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。オブティマイザは、計画を使って文を実行するために必要な、I/O、CPU 時間、メモリーなどのコンピュータ・リソースの推定に基づいて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。しかしパラレル実行計画を使うのであれば、リソースの使用状況と経過時間の間に直接の関係はありません。

3. オブティマイザは、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。オブティマイザが最適化アプローチと目標を選択する方法の詳細は、20-37 ページの「最適化アプローチと目標の選択」を参照してください。

注意：パラレル実行の場合、オプティマイザはリソースを消費することと引換えに経過時間を最小にするよう選択できます。オプティマイザでどの程度のパラレル化を試行するかを指定するには、初期化パラメータ OPTIMIZER_PERCENT_PARALLEL を使います。

追加情報：OPTIMIZER_PERCENT_PARALLEL パラメータの使用方法は、『Oracle8 Server チューニング』を参照してください。

コストベースのアプローチで使われる統計情報

コストベースのアプローチは、それぞれの実行計画のコストを推定するときに、統計データを使います。それらの統計データは、表および列、索引、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。そのような統計情報を生成するには、ANALYZE コマンドを使います。オプティマイザは、特定の実行計画を使って 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、この統計情報を使って推定します。

統計情報は、次のデータ・ディクショナリ・ビューによって参照できます。

- USER_TABLES および ALL_TABLES、DBA_TABLES
- USER_TAB_COLUMNS および ALL_TAB_COLUMNS、DBA_TAB_COLUMNS
- USER_INDEXES および ALL_INDEXES、DBA_INDEXES
- USER_CLUSTERS および DBA_CLUSTERS
- USER_TAB_PARTITIONS および ALL_TAB_PARTITIONS、DBA_TAB_PARTITIONS
- USER_IND_PARTITIONS および ALL_IND_PARTITIONS、DBA_IND_PARTITIONS
- USER_PART_COL_STATISTICS および ALL_PART_COL_STATISTICS、DBA_PART_COL_STATISTICS

追加情報：これらの統計情報の詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

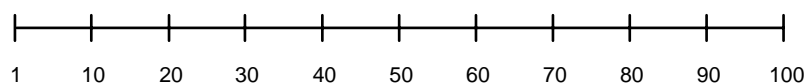
ヒストグラム

Oracle のコストベース・オプティマイザは、データ値ヒストグラムを使って列データの分布を正確に推定します。ヒストグラムを使うと、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られます。ヒストグラムは、ANALYZE コマンドを使って生成します。

コストベース・オプティマイザの基本的な機能の 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います。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。Oracle のコストベース・オプティマイザでは、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のバランスを調整したヒストグラムを使います。

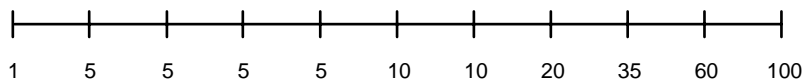
ヒストグラムの例

値が 1 ~ 100 までの列 C と、10 個の枠（バケット）で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。数値は終点の値です。



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。

高さのバランスを調整したヒストグラム

Oracle では、「幅のバランス」ではなく「高さのバランスを調整した」ヒストグラムが使われます

- 幅のバランスを調整したヒストグラムでは、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数値をカウントします。
- 高さのバランスを調整したヒストグラムでは、各範囲に同じ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終端が決められます。

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（ヒストグラムでは範囲のことを「バケット」といいます）のヒストグラムを作るとします。幅のバランスを調整したヒストグラムでは、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。高さのバランスを調整したヒストグラムでは、各バケットは同じ高さであり（この場合は 100 行）、列内の値の種類の密度に基づいて各バケットの終端が決まります。

高さのバランスを調整したヒストグラムの利点

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 行の表の 800 行の値がすべて 5 であり、残りの 200 行は 1 ~ 100 の間で均等に分布しているとします。幅のバランスをとったヒストグラムでは、1 ~ 10 のバケットに 820 行が入り、その他のバケットにはほぼ 20 行が入ります。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 ~ 5 のバケットが 7 つ、5 ~ 50 のバケットが 1 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、高さ調整したヒストグラムを見ると、約 80% もの行がその値であることがわかります。しかし、幅のバランスを調整した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% の行だけになります。したがって、列値の選択性を判断するには、高さベースのヒストグラムのほうが適しています。

いつヒストグラムを使うか

一般に、WHERE 句では索引付きの列がよく使われるので、ヒストグラムを作るときには、ほとんどの場合、ANALYZE コマンドの FOR ALL INDEXED COLUMNS オプションを使うのが適切です。

ヒストグラムは、次のビューを使って表示できます。

- USER_HISTOGRAMS および ALL_HISTOGRAMS、DBA_HISTOGRAMS
- USER_PART_HISTOGRAMS および ALL_PART_HISTOGRAMS、DBA_PART_HISTOGRAMS
- TAB_COLUMNS

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だけです。データの分布が静的でない場合は、ヒストグラムを頻繁に更新する必要があります。(データの「分布」が一定であれば、データが静的である必要はありません。)

パフォーマンスに影響を与える可能性があるため、ヒストグラムを使うのは、問合せ計画を実際に改善できる場合だけにしてください。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っている。
- その列のデータが均一に分布している。
- 問合せの WHERE 句でその列が使われていない。
- その列が一意で、等価を表す述語だけで使われている。

追加情報: ヒストグラムの詳細は、『Oracle8 Server チューニング』を参照してください。

いつコストベースのアプローチを使うか

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ってください。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えるようになる前に作ら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えます。

次の機能に対しては、コストベースの最適化しか使えません。

- 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
- 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

追加情報：コストベース・アプローチをどのようなときに使うかの詳細は、『Oracle8 Server チューニング』を参照してください。

ルールベースのアプローチ

ルールベースのアプローチを使うと、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（20-43ページの表 20-1「アクセス・パス」を参照）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えます。

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います。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。

詳細は、20-58ページの「ルールベースのアプローチでのアクセス・パスの選択」を参照してください。

オブティマイザの操作の概要

この項では、Oracle のオブティマイザが実行する操作について要約し、最適化できる SQL 文のタイプを説明します。

オブティマイザの操作

Oracle が処理する SQL 文に対して、オブティマイザは次のことを実行します。

式と条件の評価	オブティマイザは、最初に、定数が入っている式と条件をできるだけ十分に評価します。(20-12 ページの「式と条件の評価」を参照。)
文の変換	複雑な文(たとえば相関副問合せが入っている文)の場合、オブティマイザは元の文を等価な結合文に変換することがあります。(20-17 ページの「文の変換と最適化」を参照。)
ビューのマージ	ビューにアクセスする SQL 文の場合、オブティマイザは文にある問合せをビューの問合せにマージした上で、その結果を最適化することがあります。(20-22 ページの「ビューにアクセスする文の最適化」を参照。)
最適化アプローチの選択	オブティマイザは、最適化アプローチとしてコストベースかルールベースを選択し、最適化の目標を決定します。(20-37 ページの「最適化アプローチと目標の選択」を参照。)
アクセス・パスの選択	文がアクセスするそれぞれの表について、オブティマイザは、表のデータを取得するための1つ以上の使用可能なアクセス・パスを選択します。(20-40 ページの「アクセス・パスの選択」を参照。)
結合順序の選択	3 つ以上の表を結合する表の場合、オブティマイザは最初に結合する表の対を選択し、次にその結果に結合する表を選択する、という要領で選択処理を繰り返します。(20-59 ページの「結合文の最適化」を参照。)
結合操作の選択	結合文については、オブティマイザは結合を実行するのに使う操作を選択します。(20-59 ページの「結合文の最適化」を参照。)

SQL 文のタイプ

Oracle は、次のようなさまざまなタイプの SQL 文を最適化します。

単純な文	単一の表だけが関係する INSERT または UPDATE、DELETE、SELECT 文。
単純な問合せ	SELECT 文のこと。

結合	複数の表からデータを選択する問合せ。結合には、FROM 句に複数の表が指定されます。Oracle は、WHERE 句に指定されている条件を使ってこれらの表の行を組み合わせ、結果の行を戻します。この条件のことを結合条件といい、通常は、結合されたすべての表の列がこの条件と比較されます。
等価結合	等号演算子を含む結合条件。
非同レベル結合	等号演算子以外の演算子を含む結合条件。
外部結合	1 つの表の 1 つ以上の列で外部結合演算子 (+) が使われている結合条件。Oracle は、結合条件に合致するすべての行を戻します。また、外部結合演算子が指定された表に一致する行が存在しない行を含め、外部結合演算子が指定されていない表のすべての行を戻します。
直積	<p>結合条件が指定されていない結合は、直積またはクロス積になります。直積とは、各表から取り出した各行の可能なすべての組合せの集合です。つまり、2 つの表を結合する場合は、一方の表の各行が、もう一方の表の各行と 1 つずつ対応付けられます。3 つ以上の表の直積演算は、どれか 1 つの表の各行と、残りの表の直積の各行を組み合わせた結果になります。</p> <p>他の種類の結合はすべて、直積演算のサブセットです。つまり、実質的には、最初に直積演算を導出し、次に結合条件に合わない行を排除することによって作ったものです。</p>
複合文	副問合せを含む INSERT または UPDATE、DELETE、SELECT 文。ある文で処理する値の集合を生成するための SELECT 文がその文の中に含まれている場合、その SELECT 文のことを副問合せといいます。副問合せを含む複合文のうち、その副問合せの外側の部分を「親問合せ」といいます。
複合問合せ	集合演算子 (UNION または UNION ALL、INTERSECT、MINUS) を使って 2 つ以上の単純文または複合文を組み合わせる問合せ。複合問合せのそれぞれの単純文または複合文のことを「コンポーネント問合せ」といいます。
ビューにアクセスする文	表だけでなく 1 つ以上のビューにアクセスする単純文、結合文、複合文。
分散型の文	リモート・データベースのデータにアクセスする文。

式と条件の評価

オプティマイザは、可能なときはいつでも式を十分に評価し、特定の構文上の構成体を等価構成体に変換します。これは、元の式より変換後の式のほうが Oracle で高速に評価できるため、または元の式が変換後の式と構文上まったく等価であるためです。同一の処理を実行するのにさまざまな SQL 構成体があり得るため（たとえば、= ANY (副問合せ) と IN (副問合せ) ）、Oracle はそれらを単一の構成体に変換します。

定数

定数の計算は、文が実行されるたびにではなく、文が最適化される時点で 1 回だけ実行されます。

たとえば、月給が 2000 を超えているかどうかテストする次の条件を考えてみましょう。

```
sal > 24000/12
```

```
sal > 2000
```

```
sal*12 > 24000
```

SQL 文に最初の条件が含まれている場合、オブティマイザはその条件を第 2 の条件に単純化します。

オブティマイザは、比較演算子を越えて式を単純化することはありません。オブティマイザは、第 3 の式を第 2 の式に単純化することはありません。この理由で、アプリケーションの開発者は、列が含まれる式を比較する条件ではなく、可能な限り、定数と列を比較する条件を作るようにしてください。

LIKE 演算子

オブティマイザは、LIKE 比較演算子を使ってもワイルド・カード文字が入っていない式を比較する条件を、等号演算子を使う等価の条件に単純化します。たとえば、オブティマイザは、次の最初の条件を第 2 の条件に単純化します。

```
ename LIKE 'SMITH'
```

```
ename = 'SMITH'
```

オブティマイザが上記の式を単純化するのは、可変長のデータ型を比較する場合だけです。たとえば、ENAME のデータ型が CHAR(10) の場合、等価 (=) 演算子は必要な空白を末尾に埋めての比較方法になりますが、LIKE は空白埋めを行わないので、オブティマイザは LIKE 演算子を等価演算に変換できません。

IN 演算子

オブティマイザは、IN 比較演算子を使う条件を、等価比較演算子と OR 論理演算子を使う等価の条件に展開します。たとえば、オブティマイザは、次の最初の条件を第 2 の条件に展開します。

```
ename IN ('SMITH', 'KING', 'JONES')
```

```
ename = 'SMITH' OR ename = 'KING' OR ename = 'JONES'
```

詳細は、20-24 ページの「例 2: IN 副問合せ」を参照してください。

ANY または SOME 演算子

オブティマイザは、ANY または SOME 比較演算子の後にかっこで囲まれた値のリストが続く条件を、等価比較演算子とOR論理演算子を使う条件に展開します。たとえば、オブティマイザは、次の最初の条件を第2の条件に展開します。

```
sal > ANY (:first_sal, :second_sal)
```

```
sal > :first_sal OR sal > :second_sal
```

オブティマイザは、ANY または SOME 比較演算子の後に副問合せが続く条件を、EXISTS 演算子と相関副問合せが含まれる条件に変換します。たとえば、オブティマイザは、次の最初の条件を第2の条件に変換します。

```
x > ANY (SELECT sal
        FROM emp
        WHERE job = 'ANALYST')
```

```
EXISTS (SELECT sal
        FROM emp
        WHERE job = 'ANALYST'
        AND x > sal)
```

ALL 演算子

オブティマイザは、ALL 比較演算子の後にかっこで囲まれた値のリストが続く条件を、等価比較演算子と AND 論理演算子を使う等価の条件に展開します。たとえば、オブティマイザは、次の最初の条件を第2の条件に展開します。

```
sal > ALL (:first_sal, :second_sal)
```

```
sal > :first_sal AND sal > :second_sal
```

オブティマイザは、ALL 比較演算子の後に副問合せが続く条件を、ANY 比較演算子と補足的な比較演算子を使う等価の条件に変換します。たとえば、オブティマイザは、次の最初の条件を第2の条件に変換します。

```
x > ALL (SELECT sal
        FROM emp
        WHERE deptno = 10)
```

```
NOT (x <= ANY (SELECT sal
              FROM emp
              WHERE deptno = 10) )
```

この後、オブティマイザは、ANY 比較演算子の後に相関副問合せが続く条件を変換するときのルールを使って、第 2 の問合せを次の問合せに変換します。

```
NOT EXISTS (SELECT sal
             FROM emp
             WHERE deptno = 10
             AND x <= sal)
```

BETWEEN 演算子

オブティマイザは、BETWEEN 比較演算子を使う条件を、>= と <= を使う等価の条件に置き換えます。たとえば、オブティマイザは、次の最初の条件を第 2 の条件に置き換えます。

```
sal BETWEEN 2000 AND 3000
```

```
sal >= 2000 AND sal <= 3000
```

NOT 演算子

オブティマイザは、条件を単純化して NOT 論理演算子を排除します。この単純化には、NOT 論理演算子の削除と、比較演算子を反対の比較演算子に置き換える操作が関係しています。たとえば、オブティマイザは、次の最初の条件を第 2 の条件に単純化します。

```
NOT deptno = (SELECT deptno FROM emp WHERE ename = 'TAYLOR')
```

```
deptno <> (SELECT deptno FROM emp WHERE ename = 'TAYLOR')
```

NOT 論理演算子を含む条件は、いろいろな仕方で記述できます。オブティマイザは、そのような条件を変換する際に、NOT によって否定される副条件ができる限り単純になるようにします。その結果、変換後の条件に含まれる NOT の数が元の数より多くなることもあります。たとえば、オブティマイザは、次の第 1 の条件を第 2 の条件、さらに第 3 の条件に単純化します。

```
NOT (sal < 1000 OR comm IS NULL)
```

```
NOT sal < 1000 AND comm IS NOT NULL
```

```
sal >= 1000 AND comm IS NOT NULL
```

推移律

WHERE 句の 2 つの条件に共通の列が含まれている場合、オブティマイザは移行の原理（推移律）を使って第 3 の条件を推論することがあります。この後、オブティマイザは、推論した条件を使って文を最適化します。推論した条件により、元の条件では使用可能でなかった索引アクセス・パスが使用可能になることがあります。

注意：推移律を使うのは、コストベースのアプローチの場合だけです。

たとえば、WHERE 句に次の形式の 2 つの条件が含まれているとします。

```
WHERE column1 comp_oper constant
      AND column1 = column2
```

この場合、オプティマイザは次のような条件を推論します。

```
column2 comp_oper constant
```

各値の意味は次のとおりです。

comp_oper	比較演算子 =、!= または ^=、<、<>、>、<=、>= のいずれか。
constant	演算子および SQL ファンクション、リテラル、バインド変数、相関変数が含まれている定数式。

例：

それぞれ EMP.DEPTNO 列を使う 2 つの条件が WHERE 句に含まれている、次の問合せを考えてみます。

```
SELECT *
  FROM emp, dept
 WHERE emp.deptno = 20
      AND emp.deptno = dept.deptno;
```

オプティマイザは、推移律を使ってこの条件を次のように推論します。

```
dept.deptno = 20
```

DEPT.DEPTNO 列に索引がある場合、この条件により、その索引を使うアクセス・パスが使用可能になります。

注意：オプティマイザは、列を他の列と比較する条件ではなく、列を定数式と比較する条件についてだけ推論を実行します。たとえば、WHERE 句に次の形式の 2 つの条件が含まれているとします。

```
WHERE column1 comp_oper column3
      AND column1 = column2
```

この場合、オプティマイザが次のような条件を推論することはありません。

```
column2 comp_oper column3
```

文の変換と最適化

SQL は柔軟な問合せ言語なので、特定の目標を達成するための候補となる文がたくさんあります。他に同じ目標を達成でき、しかも実行効率がより高い文がある場合、オプティマイザは、1 つの文をその別の文に変換することがあります。

ここでは、次の内容を取り上げます。

- OR を複合問合せに変換する
- 複合文を結合文に変換する
- ビューにアクセスする文の最適化
- 複合問合せの最適化
- 分散型の文の最適化

文の最適化の詳細は、20-59 ページの「結合文の最適化」と 20-70 ページの「「スター」問合せの最適化」を参照してください。

OR を複合問合せに変換する

問合せの WHERE 句に OR 演算子で結ばれた複数の条件があり、その句を UNION ALL 集合演算子を使う等価の複合問合せに変換すれば実行の効率が上がるという場合に、オプティマイザはそのような変換を実行します。

- それぞれの条件により索引アクセス・パスが使用可能になる場合、オプティマイザはこの変換を実行できる。その場合、オプティマイザは、異なる索引を使って何回か表にアクセスし、その結果をまとめる、という変換後の文の実行計画を選択します。
- 索引が使用可能でないため全表走査が必要になる条件がある場合、オプティマイザは文を変換しない。オプティマイザは、その文を実行するのに全表走査を選択し、Oracle は、表の各行をテストして、条件を満たしているかどうか判断します。

- コストベースのアプローチを使う文の場合、オブティマイザは、元の文と変換後の文を実行するときのコストを統計情報を使って推定して比較し、その変換を実施するかどうかを決める。
- コストベースのオブティマイザでは、同じ列での IN リストまたは OR のための OR 変換は実施されません。そのかわりに IN リスト反復演算子を使います。

追加情報：詳細は、『Oracle8 Server チューニング』を参照してください。

アクセス・パスの詳細、および索引がアクセス・パスを使用可能にする方法の詳細は、20-43 ページの表 20-1 「アクセス・パス」と、その後の項を参照してください。

例：

2つの条件を OR 演算子で組み合わせて指定している WHERE 句が含まれている次の問合せを考えてみます。

```
SELECT *
  FROM emp
 WHERE job = 'CLERK'
        OR deptno = 10;
```

JOB 列と DEPTNO 列の両方に索引がある場合、オブティマイザはこの問合せを、次のように等価の問合せに変換することがあります。

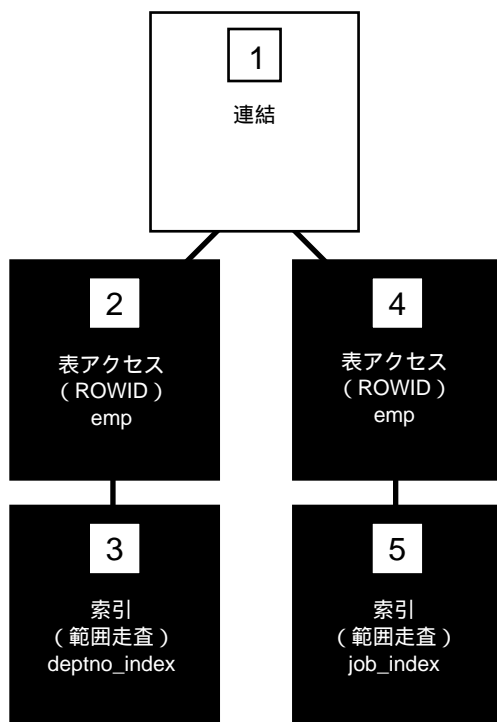
```
SELECT *
  FROM emp
 WHERE job = 'CLERK'
UNION ALL
SELECT *
  FROM emp
 WHERE deptno = 10
        AND job <> 'CLERK';
```

コストベースのアプローチを使っている場合、オブティマイザは、元の問合せを全表走査を使って実行するコストと、変換後の問合せを実行するコストを比較して、変換を実施するかどうかを決定します。

ルールベースのアプローチを使っている場合は、変換後の複合問合せの各コンポーネント問合せは索引を使って実行できるので、オブティマイザはこの UNION ALL 変換を実行します。ルールベースのアプローチでは、元の問合せを全表走査を使って実行するよりも、複合問合せを 2 つの索引走査を使って実行するほうが高速であると判断されます。

変換された文の実行計画を、図 20-2 に示します。

図 20-2 OR を含む問合せを変換した場合の実行計画



変換後の問合せを Oracle が実行する場合のステップは、次のとおりです。

- ステップ 3 と 5 は、コンポーネント問合せの条件を使って JOB 列と DEPTNO 列の索引を走査する。これらのステップで、コンポーネント問合せの条件を満たす行の ROWID が取得されます。
- ステップ 2 と 4 は、ステップ 3 と 5 で取得された ROWID を使って、各コンポーネント問合せの条件を満たしている行を探す。
- ステップ 1 は、ステップ 2 と 4 から戻された行ソースをまとめる。

JOB 列か DEPTNO 列のどちらかに索引がない場合は、結果の複合問合せに含まれるコンポーネント問合せを実行するために全表走査が必要になるため、オブティマイザはその変換を考慮することさえしません。全表走査と索引走査を使って複合問合せを実行したほうが、全表走査を使って元の問合せを実行するよりも高速になるということは、おそらくあり得ません。

例：

ENAME 列だけに索引があると仮定して、次の問合せを考えます。

```
SELECT *
  FROM emp
 WHERE ename = 'SMITH'
        OR sal > comm;
```

上記の問合せを変換すると、次のような複合問合せになります。

```
SELECT *
  FROM emp
 WHERE ename = 'SMITH'
UNION ALL
SELECT *
  FROM emp
 WHERE sal > comm;
```

第2のコンポーネント問合せの WHERE 句の条件 (SAL > COMM) では索引が使用可能にならないので、複合問合せには全表走査が必要になります。この理由で、オブティマイザは変換を実行せず、全表走査によって元の文を実行することになります。

複合文を結合文に変換する

複合文を最適化する場合、オブティマイザは次の方法のいずれかを選択します。

- 複合文を等価の結合文に変換し、その結合文を最適化する。
- 複合文をそのまま最適化する。

複合文を結合文に変換した後、その結合文が必ず複合文とまったく同じ行を戻すとわかっている場合、オブティマイザは複合文を結合文に変換します。この変換により、Oracle は、20-59 ページの「結合文の最適化」で説明する結合最適化手法の利点を活用して文を実行できるようになります。

所有者が CUSTOMERS 表に含まれているすべての行を ACCOUNTS 表から選択する、次のような複合文を考慮します。

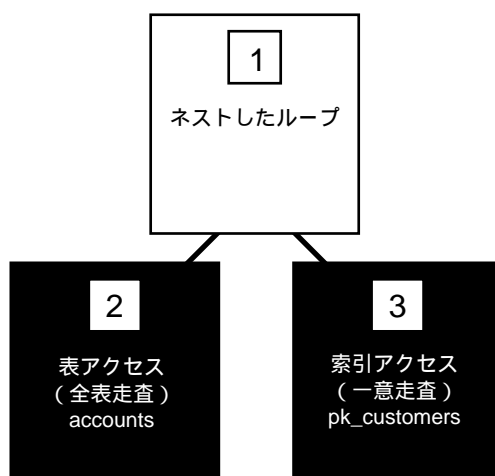
```
SELECT *
  FROM accounts
 WHERE custno IN
        (SELECT custno FROM customers);
```

CUSTOMERS 表の CUSTNO 列が主キーであるかその列に UNIQUE 制約がある場合、オブティマイザはこの複合問合せを、確実にそれと同じデータを戻す次の結合文に変換できます。


```
SELECT accounts.*  
  FROM accounts, customers  
 WHERE accounts.custno = customers.custno;
```

この文の実行計画は、図 20-3 のようになります。

図 20-3 ネストされたループ結合の実行計画



この文を実行する場合、Oracle はネストされたループ結合操作を実行します。ネストされたループ結合の詳細は、20-60 ページの「結合操作」を参照してください。

オブティマイザが複合文を結合文に変換できない場合、オブティマイザは親文の実行計画と副問合せの実行計画を、あたかもそれぞれが別々の文であるかのようにして選択します。その後、Oracle は、副問合せを実行し、その問合せから戻される行を使って親問合せを実行します。

残高が平均残高を上回るすべての行をACCOUNTS表から戻す、次の複合文を考えてみましょう。

```
SELECT *  
  FROM accounts  
 WHERE accounts.balance >  
        (SELECT AVG(balance) FROM accounts);
```

この文の機能を実行できる結合文はないので、オブティマイザはこの文を変換しません。副問合せに AVG などのグループ・ファンクションが入っている複合問合せは、結合文に変換できないことに注意してください。

ビューにアクセスする文の最適化

ビューにアクセスする文を最適化する場合、オブティマイザは次の方法のいずれかを選択します。

- 文を、ビューの実表にアクセスする等価の文に変換し、さらにその結果の文を最適化する。オブティマイザは、次のいずれかの手法を使って文を変換できます。
 - ビューの問合せを、アクセスする文の参照問合せブロックにマージする。
 - 参照問合せブロックの述語を、ビューの内部に入れる（マージ不可能なビューの場合）。
- ビューの問合せを発行し、すべての戻り行を収集してから、その行セットを、表にアクセスする場合と同様に、元の文を使ってアクセスする。（20-32 ページの「元の文を使ったビューの行へのアクセス」を参照。）

文へのビューの問合せのマージ

ビューの問合せをアクセスする文の参照問合せブロックにマージする場合、オブティマイザは、問合せブロックにあるビューの名前をそのビューの実表の名前に置き換え、ビューの問合せの WHERE 句の条件をアクセスする問合せブロックの WHERE 句に追加します。

この最適化は「選択表示結合」ビューに適用されます。選択表示結合ビューとは、選択および表示、結合だけを含むビューです。つまり、集合演算子、グループ関数、DISTINCT、GROUP BY、CONNECT BYなどは含まないビューです（20-23 ページの「マージ可能なビューとマージ不可能なビュー」に説明があります）。

例：

deptno の 10 で勤務するすべての従業員を表示する、次のようなビューを考えます。

```
CREATE VIEW emp_10
AS SELECT empno,ename, job, mgr, hiredate, sal, comm, deptno
FROM emp
WHERE deptno = 10;
```

そのビューにアクセスする、次の問合せを考えます。この問合せは、deptno が 10 に勤務する従業員の ID のうち、7800 より大きい ID を選択するものです。

```
SELECT empno
FROM emp_10
WHERE empno > 7800;
```

オブティマイザは、この問合せを、ビューの実表にアクセスする次の問合せに変換します。

```
SELECT empno
FROM emp
WHERE deptno = 10
AND empno > 7800;
```

DEPTNO 列と EMPNO 列に索引がある場合、変換後の WHERE 句によってそれらの索引が使用可能になります。

マージ可能なビューとマージ不可能なビュー

ビューが 1 つ以上の実表を持っている場合、ビューに次の要素が含まれていなければ、オブティマイザはそのビューを参照問合せブロックにマージできます。

- 集合演算子 (UNION、UNION ALL、INTERSECT、MINUS)
- CONNECT BY 句
- ROWNUM 疑似列
- 選択リスト内のグループ関数 (AVG、COUNT、MAX、MIN、SUM)

ビューに次の構造の 1 つが含まれているときは、次に説明する「複合ビューのマージ」が使用可能にされている場合にだけ、そのビューを参照問合せブロックにマージできます。

- GROUP BY 句
- 選択リストの DISTINCT 演算子

ビューが外部結合の右側にある場合、複数の実表を持つビューはマージできません。ただし、外部結合の右側にあるビューが実表を 1 つしか持っていない場合は、ビュー内の式が NULL 以外の値を NULL に対して返す可能性があっても、オブティマイザは複合ビューのマージを使うことができます。詳細は、20-68 ページの「外部結合のビュー」を参照してください。

複合ビューのマージ

ビューの問合せで、GROUP BY 句または選択リストの DISTINCT 演算子が含まれている場合、オブティマイザは、複合ビューのマージが使用可能にされている場合にだけビューの問合せをアクセスする文にマージできます。複合マージを使うと、副問合せが非相関であれば、アクセスする文に IN 副問合せをマージすることもできます。(20-24 ページの「例 2: IN 副問合せ」を参照)

複合マージはコストベースではありません。複合マージは、初期化パラメータ COMPLEX_VIEW_MERGING または MERGE ヒントを使って使用可能にしなければなりません。つまり、COMPLEX_VIEW_MERGING パラメータを TRUE に設定するか、問合せブロックのアクセスに MERGE ヒントを含める必要があります。このヒントまたはパラメータが設定されていないと、オブティマイザは別のアプローチを使います (20-25 ページの「ビューへの述語の追加」を参照)

追加情報: MERGE ヒントと NO_MERGE ヒントの詳細は、『Oracle8 Server チューニング』を参照してください。

例 1: GROUP BY 句を使ったビュー

各部門の給与平均額を含むビュー AVG_SALARY_VIEW について考えます。

```
CREATE VIEW avg_salary_view AS
  SELECT deptno, AVG(sal) AS avg_sal_dept,
     FROM emp
     GROUP BY deptno;
```

複合ビューのマージが使用可能にされていれば、オブティマイザはロンドンの部門の給与平均額を検索する次の問合せを、

```
SELECT dept.deptloc, avg_sal_dept
  FROM dept, avg_salary_view
 WHERE dept.deptno = avg_salary_view.deptno
    AND dept.deptloc = 'London';
```

次の問合せに変換できます。

```
SELECT dept.deptloc, AVG(sal)
  FROM dept, emp
 WHERE dept.deptno = emp.deptno
    AND dept.deptloc = 'London'
 GROUP BY dept.rowid, dept.deptloc;
```

変換された問合せはビューの実表にアクセスし、ロンドンに勤務している従業員の行だけを選択して、それらを部門別にグループ化します。

例 2: IN 副問合せ

複合マージは、ビューだけでなく、非相関副問合せを持つ IN 句でも使えます。各部門の給与最低額を含むビュー MIN_SALARY_VIEW について考えます。

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

複合マージが使用可能にされていれば、オブティマイザは、給与額がロンドンの部門の給与最低額であるすべての従業員を検索する次の問合せを、

```
SELECT emp.ename, emp.sal
  FROM emp, dept
 WHERE (emp.deptno, emp.sal) IN min_salary_view
    AND emp.deptno = dept.deptno
    AND dept.deptloc = 'London';
```

次の問合せに変換できます (E1 と E2 は、それぞれアクセスする問合せブロックとビューの問合せブロックで参照される EMP 表を表します)。

```
SELECT e1.ename, e1.sal
  FROM emp e1, dept, emp e2
 WHERE e1.deptno = dept.deptno
       AND dept.deptloc = 'London'
       AND e1.deptno = e2.deptno
 GROUP BY e1.rowid, dept.rowid, e1.ename, e1.sal
 HAVING e1.sal = MIN(e2.sal);
```

ビューへの述語の追加

オブティマイザは、ビューの問合せの内部に問合せブロックの述語を追加することによって、マージ不可能なビューにアクセスする問合せブロックを変換できます。

例 1:

2 つの従業員表のユニオンである TWO_EMP_TABLES ビューを考えます。このビューは、UNION 集合演算子を使う複合問合せを使って定義されます。

```
CREATE VIEW two_emp_tables
  (empno, ename, job, mgr, hiredate, sal, comm, deptno) AS
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
  FROM emp1
 UNION
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
  FROM emp2;
```

そのビューにアクセスする、次の問合せを考えます。この問合せは、どちらかの表に記述されている、deptno が 20 に勤務する従業員の ID と名前を選択するものです。

```
SELECT empno, ename
  FROM two_emp_tables
 WHERE deptno = 20;
```

ビューは複合問合せとして定義されているので、オブティマイザは、アクセスする問合せブロックにビューの問合せをマージできません。そのかわりに、オブティマイザはこの問合せの述語である WHERE 句条件 (DEPTNO = 20) をビューの複合問合せに追加することにより、アクセスする文を変換できます。

この結果の文は、次のようになります。

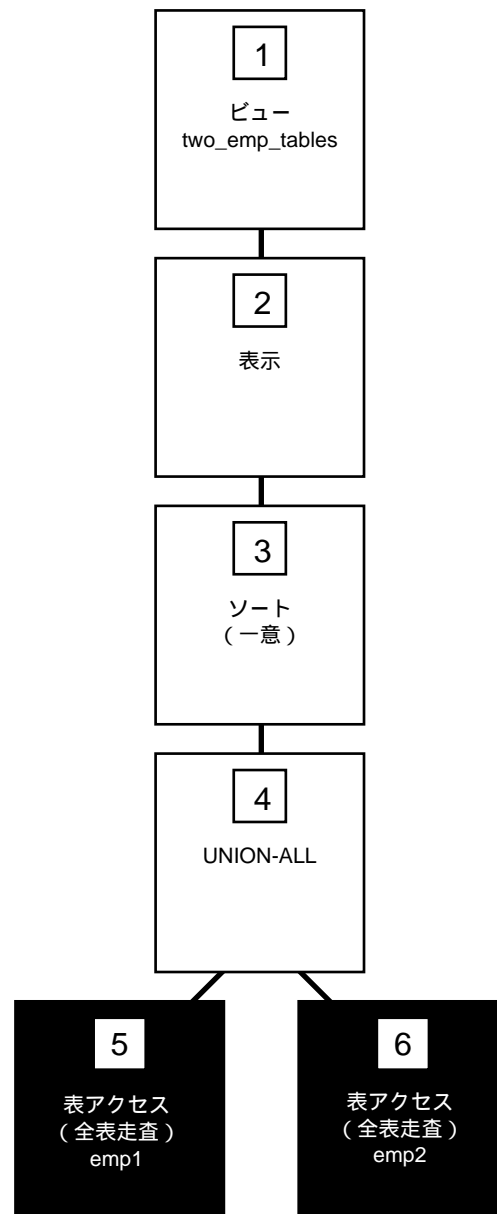
```
SELECT empno, ename
  FROM ( SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
```

```
        FROM emp1
        WHERE deptno = 20
UNION
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
FROM emp2
WHERE deptno = 20 );
```

DEPTNO 列に索引がある場合、変換後の WHERE 句によりその索引が使用可能になります。

図 20-4 「UNION 集合演算子を使って定義されたビューへのアクセス」に、結果の文の実行計画を示します。

図 20-4 UNION 集合演算子を使って定義されたビューへのアクセス



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 5 と 6 で、EMP1 表と EMP2 表の全走査を実行する。
- ステップ 4 で、重複行のすべてのコピーも含め、ステップ 5 またはステップ 6 から戻されるすべての行を戻す UNION-ALL 操作を実行する。
- ステップ 3 で、ステップ 4 の結果をソートし、重複行を排除する。
- ステップ 2 で、ステップ 3 の結果から希望する列を抽出する。
- ステップ 1 は、ビューの問合せはアクセスする問合せにマージされなかったことを示す。

例 2:

従業員がいるすべての部門の部門番号および給与平均額、給与最低額、給与最高額を示す、次のビュー EMP_GROUP_BY_DEPTNO を考えます。

```
CREATE VIEW emp_group_by_deptno
AS SELECT deptno,
          AVG(sal) avg_sal,
          MIN(sal) min_sal,
          MAX(sal) max_sal
FROM emp
GROUP BY deptno;
```

EMP_GROUP_BY_DEPTNO ビューから deptno が 10 の給与平均額および最低額、最高額を選択する、次の問合せを考えます。

```
SELECT *
FROM emp_group_by_deptno
WHERE deptno = 10;
```

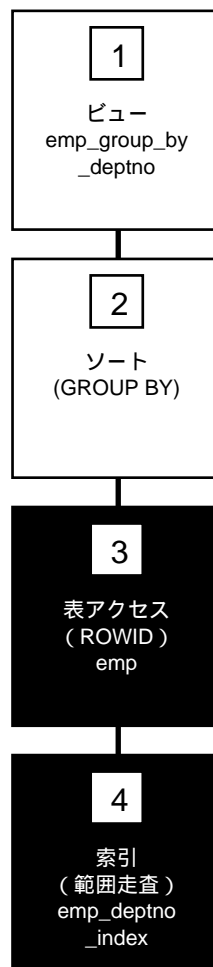
オブティマイザは、文の述語（WHERE 句条件）をビューの問合せに追加することにより、この文を変換します。この結果の文は、次のようになります。

```
SELECT deptno,
          AVG(sal) avg_sal,
          MIN(sal) min_sal,
          MAX(sal) max_sal,
FROM emp
WHERE deptno = 10
GROUP BY deptno;
```

DEPTNO 列に索引がある場合、変換後の WHERE 句によりその索引が使用可能になります。

図 20-5 「GROUP BY 句を使って定義されたビューへのアクセス」に、結果の文の実行計画を示します。この実行計画では、DEPTNO 列の索引を使います。

図 20-5 GROUP BY 句を使って定義されたビューへのアクセス



この文を実行する場合、Oracle は次の操作を実行します。

- ステップ4で、索引 EMP_DEPTNO_INDEX (EMP 表の DEPTNO 列の索引) に対して範囲走査を実行し、EMP 表から DEPTNO 値が 10 であるすべての行の ROWID を検索する。
- ステップ3で、ステップ4で検索した ROWID を使って EMP 表にアクセスする。
- ステップ2で、ステップ3で戻された行をソートし、SAL の平均値および最小値、最大値を計算する。
- ステップ1は、ビューの問合せはアクセスする問合せにマージされなかったことを示す。

ビューへのグループ関数の適用

オプティマイザは、ビューの問合せにグループ関数 (AVG、COUNT、MAX、MIN、SUM) を適用することによって、これらの関数を含む問合せを変換できます。

例：

直前の例で定義した EMP_GROUP_BY_DEPTNO ビューにアクセスする、次の問合せを考えます。この問合せは、部門給与の平均額および最低額、最高額の平均を、従業員表から導出するものです。

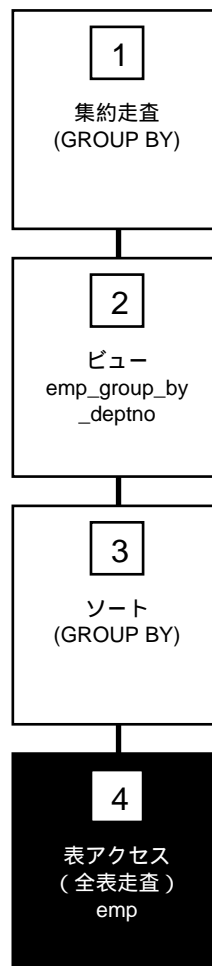
```
SELECT AVG(avg_sal), AVG(min_sal), AVG(max_sal)
FROM emp_group_by_deptno;
```

オプティマイザは、ビューの問合せの選択リストに AVG グループ・ファンクションを適用することにより、この文を変換します。

```
SELECT AVG(AVG(sal)), AVG(MIN(sal)), AVG(MAX(sal))
FROM emp
GROUP BY deptno;
```

図 20-6 に、結果の文の実行計画を示します。

図 20-6 GROUP BY 句を使って定義されたビューにグループ・ファンクションを適用する



この文を実行する場合、Oracle は次の操作を実行します。

- ステップ 4 で、EMP 表の全走査を実行する。
- ステップ 3 で、ステップ 4 で戻された行を DEPTNO 値に基づいてグループ別にソートし、それぞれのグループの SAL の平均値および最小値、最大値を計算する。
- ステップ 2 は、ビューの問合せはアクセスする問合せにマージされなかったことを示す。

- ステップ1で、ステップ2で戻された値の平均値を計算する。

元の文を使ったビューの行へのアクセス

オブティマイザは、ビューにアクセスする文すべてを、実表にアクセスする等価の文に変換できるとは限りません。たとえば、問合せがビュー内の ROWNUM 疑似列にアクセスする場合に、ビューを問合せにマージすることや、問合せの述語をビューに追加することはできません。

実表にアクセスする文に変換できない文を実行する場合、Oracle はビューの問合せを発行し、結果の行セットを収集し、表にアクセスする場合と同様に元の文を使ってその行セットにアクセスします。

例：

前の項で定義した、次の EMP_GROUP_BY_DEPTNO ビューを考えます。

```
CREATE VIEW emp_group_by_deptno
AS SELECT deptno,
          AVG(sal) avg_sal,
          MIN(sal) min_sal,
          MAX(sal) max_sal
FROM emp
GROUP BY deptno;
```

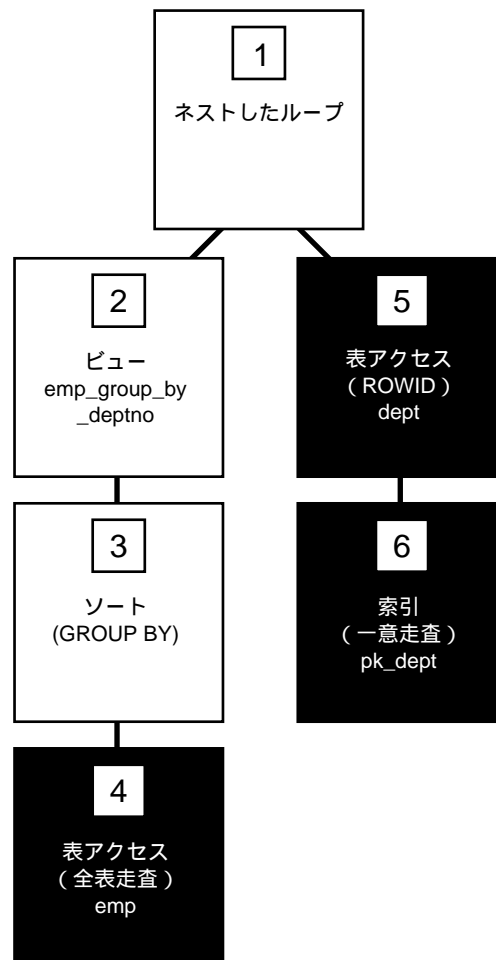
このビューにアクセスする、次の問合せを考えます。この問合せは、DEPT 表にある部門の名前と場所に、このビューに表されているそれぞれの部門の給与平均額および最低額、最高額を結合するものです。

```
SELECT emp_group_by_deptno.deptno, avg_sal, min_sal,
       max_sal, dname, loc
FROM emp_group_by_deptno, dept
WHERE emp_group_by_deptno.deptno = dept.deptno;
```

実表だけをアクセスする等価の文はないので、オブティマイザはこの文を変換できません。そのかわりに、オブティマイザは、このビューの問合せを発行する実行計画を選択し、その結果の行セットを、表にアクセスした結果の行を使う場合と同様の方法で使います。

図 20-7 「GROUP BY 句を使って定義したビューを表に結合する」に、この文の実行計画を示します。ネストされたループの結合操作の実行方法の詳細は、20-60ページの「結合操作」を参照してください。

図 20-7 GROUP BY 句を使って定義したビューを表に結合する



この文を実行する場合、Oracle は次の操作を実行します。

- ステップ 4 で、EMP 表の全走査を実行する。
- ステップ 3 で、ステップ 4 の結果をソートし、EMP_GROUP_BY_DEPTNO ビューの問合せで選択された SAL 値の平均値および最小値、最大値を計算する。
- ステップ 2 で、前の 2 つのステップで得られたデータをビューに使う。

- ステップ 6 で、ステップ 2 で戻されるそれぞれの行について、DEPTNO 値を使って PK_DEPT 索引の一貫走査を実行する。
- ステップ 5 で、ステップ 6 で戻されるそれぞれの ROWID を使って、DEPTNO 値と一致する DEPTNO 表の行を探す。
- Oracle は、ステップ 2 で戻されたそれぞれの行を、ステップ 5 で戻された一致した行と組み合わせて、結果を戻す。

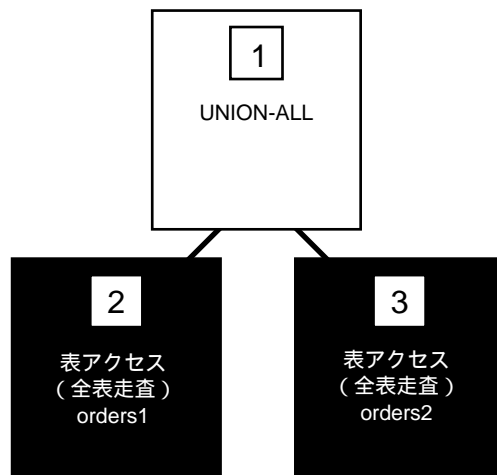
複合問合せの最適化

複合問合せの実行計画を選択する場合、オプティマイザは各コンポーネント問合せの実行計画を選択してから、複合問合せで使われている集合演算子に応じて、結果の行ソースを UNION、INTERSECTION、または MINUS 操作で結合します。

図 20-8 「UNION ALL 集合演算子を使った複合問合せ」に、この文の実行計画を示します。ここでは、UNION ALL 演算子を使って、ORDERS1 表または ORDERS2 表のどちらかに含まれているすべての部品を選択しています。

```
SELECT part FROM orders1
UNION ALL
SELECT part FROM orders2;
```

図 20-8 UNION ALL 集合演算子を使った複合問合せ



この文を実行する場合、Oracle は次のステップを実行します。

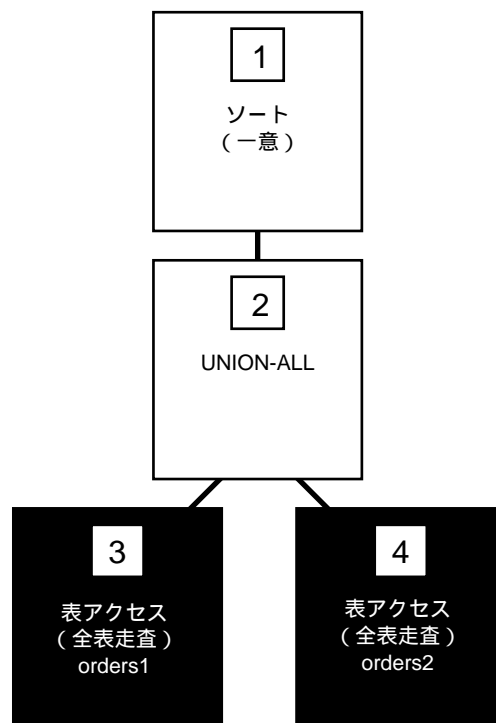
- ステップ 2 と 3 で、ORDERS1 表と ORDERS2 表に対して全表走査を実行する。

- ステップ 1 で、重複行のすべてのコピーも含め、ステップ 2 またはステップ 3 から戻されるすべての行を戻す UNION-ALL 操作を実行する。

図 20-9 「UNION 集合演算子を使った複合問合せ」に、UNION 演算子を使って ORDERS1 表または ORDERS2 表のどちらかに含まれているすべての部品を選択する文の実行計画を示します。

```
SELECT part FROM orders1
UNION
SELECT part FROM orders2;
```

図 20-9 UNION 集合演算子を使った複合問合せ

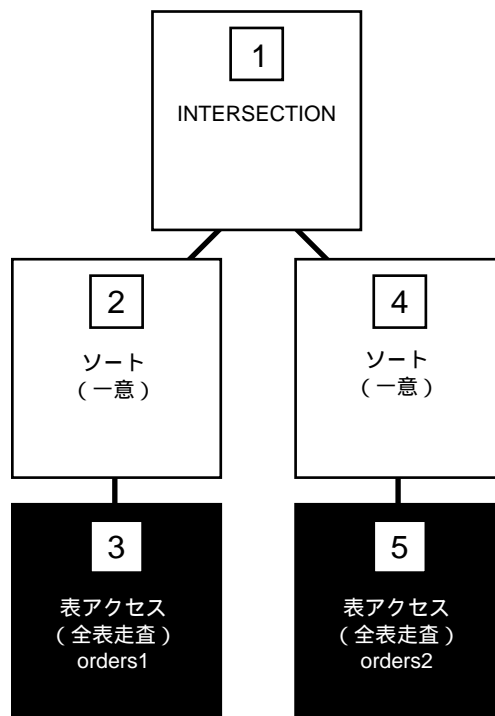


この実行計画は、図 20-8 「UNION ALL 集合演算子を使った複合問合せ」に示されている UNION-ALL 演算子の場合の実行計画とほとんど同じですが、UNION-ALL 操作で戻された重複値を SORT 操作を使って排除している点が違います。

図 20-10 「INTERSECT 集合演算子を使った複合問合せ」に、INTERSECT 演算子を使って ORDERS1 表と ORDERS2 表の両方に含まれている部品だけを選択する文の実行計画を示します。

```
SELECT part FROM orders1  
INTERSECT  
SELECT part FROM orders2;
```

図 20-10 INTERSECT 集合演算子を使った複合問合せ



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 3 と 5 で、ORDERS1 表と ORDERS2 表に対して全表走査を実行する。
- ステップ 2 と 4 で、ステップ 3 と 5 の結果をソートし、それぞれの行ソースにある重複値を排除する。

- ステップ 1 で、ステップ 2 と 4 の両方から戻された行だけを戻す INTERSECTION 操作を実行する。

分散型の文の最適化

オブティマイザは、ローカル・データにだけアクセスする文の実行計画を選択するのと同様に、リモート・データベースのデータにアクセスする SQL 文の実行計画を選択します。

- SQL 文がアクセスするすべての表が同じリモート・データベースに配置されている場合、Oracle はそのリモート・データベースに SQL 文を送信します。リモート Oracle インスタンスは、その文を実行し、その結果だけをローカル・データベースに送り返します。
- SQL 文が別々のデータベースに配置されている表にアクセスする場合、Oracle はその文を、それぞれが単一のデータベースの表にアクセスする個々の断片部分に分解します。その後、Oracle はそれぞれの断片部分を、それがアクセスするデータベースに送ります。それぞれのデータベースのリモート Oracle インスタンスがその断片部分を実行し、ローカル・データベースに結果を戻します。ローカル・データベースでは、ローカル Oracle インスタンスが、元の文が求めるなんらかの追加処理を実行することがあります。

分散型の文のコストベースの実行計画を選択する場合、オブティマイザは、ローカル・データベースで索引を考慮するのと同様に、リモート・データベースで使用可能な索引を考慮します。コストベースで最適化する場合、オブティマイザはリモート・データベースに関する統計情報も考慮します。さらにオブティマイザは、データにアクセスする場合のコストを推定するときに、そのデータの位置を考慮します。たとえば、リモート表を全走査する場合のコストは、同一のローカル表を全走査する場合のコストより高くなります。

ルールベースの実行計画の場合、オブティマイザはリモート表にある索引は考慮に入れません。

最適化アプローチと目標の選択

SQL 文の最適化アプローチと目標を選択するときのオブティマイザの動作は、次の要因の影響を受けます。

- OPTIMIZER_MODE 初期化パラメータ
- データ・ディクショナリにある統計データ
- ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータ
- SQL 文の中のヒント（コメント）
- PL/SQL ブロック内で実行されている文

OPTIMIZER_MODE 初期化パラメータ

OPTIMIZER_MODE 初期化パラメータは、インスタンスの最適化アプローチを選択する場合のデフォルトの動作を設定します。このパラメータの有効な値は、次のとおりです。

CHOOSE	オブティマイザは、コストベースのアプローチのための統計情報が使用可能かどうかに基づいて、コストベースとルールベースのどちらかのアプローチを選択します。アクセスする表のうちどれか1つについてでも統計がデータ・ディクショナリに入っている場合、オブティマイザはコストベースのアプローチを使い、最高のスループットを目標にして最適化を実行します。アクセスする表のうちどの表の統計もデータ・ディクショナリに入っていない場合、オブティマイザはルールベースのアプローチを使います。この設定値が、このパラメータのデフォルト値です。
ALL_ROWS	オブティマイザは、統計があるかどうかに関係なくそのセッション内のすべての SQL 文にコストベースのアプローチを使い、最高のスループット（文全体を完了するのに使うリソースを最小限にする）を目標にして最適化を実行します。
FIRST_ROWS	オブティマイザは、統計があるかどうかに関係なくそのセッション内のすべての SQL 文にコストベースのアプローチを使い、最短の応答時間（結果セットの先頭の行を戻すのに使うリソースを最小限にする）を目標にして最適化を実行します。
RULE	オブティマイザは、統計情報があるかどうかに関係なく、インスタンスに対して発行されるすべての SQL 文にルールベースのアプローチを選択します。

オブティマイザが SQL 文にコストベースのアプローチを使っており、その文がアクセスする表のいくつかに統計情報がない場合、オブティマイザは、内部情報（それらの表に割り当てられているデータ・ブロックの数など）を使って、それらの表に関するその他の統計情報を推定します。

データ・ディクショナリ内の統計データ

Oracle は、コストベースのオブティマイザで使う列および表、クラスタ、索引、パーティションについての統計情報をデータ・ディクショナリに格納します（20-7 ページの「コストベースのアプローチで使われる統計情報」を参照）。

ANALYZE コマンドの次の 2 つのオプションにより、統計情報が生成されます。

- COMPUTE STATISTICS（正確な統計情報を生成）
- ESTIMATE STATISTICS（データ抽出により推定値を生成）

追加情報：詳細は、『Oracle8 Server チューニング』を参照してください。

ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータ

ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータを使うと、個々のセッションに対して OPTIMIZER_MODE パラメータによって設定された最適化アプローチと目標を上書きできます。

このパラメータの値は、セッション中にコールされたストアド・プロシージャとファンクションが発行する SQL 文の最適化に影響を与えますが、セッション中に Oracle が発行する再帰 SQL 文の最適化には影響を与えません。再帰 SQL 文の最適化アプローチに影響を与えるのは、OPTIMIZER_MODE 初期化パラメータの値だけです。

OPTIMIZER_GOAL パラメータの有効値は次のとおりです。

CHOOSE	オブティマイザは、コストベースのアプローチのための統計情報が使用可能かどうかに基づいて、コストベースとルールベースのどちらかのアプローチを選択します。アクセスする表のうちどれか1つについてでも統計がデータ・ディクショナリに入っている場合、オブティマイザはコストベースのアプローチを使い、最高のスループットを目標にして最適化を実行します。アクセスする表のうちどの表の統計もデータ・ディクショナリに入っていない場合、オブティマイザはルールベースのアプローチを使います。
ALL_ROWS	オブティマイザは、統計があるかどうかに関係なくそのセッション内のすべての SQL 文にコストベースのアプローチを使い、最高のスループット（文全体を完了するのに使うリソースを最小限にする）を目標にして最適化を実行します。
FIRST_ROWS	オブティマイザは、統計があるかどうかに関係なくそのセッション内のすべての SQL 文にコストベースのアプローチを使い、最短の応答時間（結果セットの先頭の行を戻すのに使うリソースを最小限にする）を目標にして最適化を実行します。
RULE	オブティマイザは、統計情報があるかどうかに関係なく、Oracle インスタンスに対して発行されるすべての SQL 文にルールベースのアプローチを選択します。

FIRST_ROWS、ALL_ROWS、CHOOSE、RULE の各ヒント

個々の SQL 文の中にある FIRST_ROWS または ALL_ROWS、CHOOSE、RULE ヒントは、ALTER SESSION コマンドの OPTIMIZER_MODE 初期化パラメータと OPTIMIZER_GOAL パラメータの両方の効果を上書きできます。

追加情報：ヒントの使用方法の詳細は、『Oracle8 Server チューニング』を参照してください。

PL/SQL とオブティマイザの目標

オブティマイザの目標は、PL/SQL 内から送られた問合せではなく、直接に送られた問合せにだけ適用されます。

- ALTER SESSION OPTIMIZER_GOAL 文は、PL/SQL から実行されている SQL に影響しない。
 - PL/SQL は、初期化パラメータ OPTIMIZER_MODE = FIRST_ROWS を無視する。
- ヒントを使うと、PL/SQL 内から送られた SQL 文のアクセス・パスを判断できます。

アクセス・パスの選択

実行計画を組み立てるとき、オプティマイザによる最も重要な選択の 1 つは、データベースからデータを取り出す方法です。SQL 文がアクセスする表の行には、その行を検索して取り出すことのできるアクセス・パスが数多く存在する可能性があります。オプティマイザは、そのうちの 1 つを選択します。

ここでは、次のことについて説明します。

- Oracle がデータをアクセスする基本的な方法
- 各アクセス・パスとそのパスをオプティマイザがいつ利用できるか
- オプティマイザが使用可能なアクセス・パスの中から選択する方法

アクセス方法

ここでは、Oracle がデータにアクセスする基本的な方法について説明します。

全表走査

全表走査は、表から行を取り出します。全表走査を実行する場合、Oracle はその表のすべての行を読み込み、それぞれの行が文の WHERE 句の条件を満たしているかどうかを検査します。Oracle は、表に割り当てられているすべてのデータ・ブロックを順次読み込み、マルチブロック読み込みを使って全表走査をなるべく効率的に実行できるようにします。Oracle は、各データ・ブロックを 1 回だけ読み込みます。

ROWID による表アクセス

ROWID による表アクセスも、表から行を取り出します。行の ROWID は、データ・ファイルおよびその行が入っているデータ・ブロック、ブロック内でのその行の位置を指定します。Oracle で 1 つの行を検索する場合、ROWID で行を探すのが最も高速な方法です。

ROWID によって表にアクセスする場合、Oracle はまず、その文の WHERE 句から、またはその表の 1 つ以上の索引の索引走査を実行して、選択する行の ROWID を取得します。その後、Oracle は ROWID に基づいて表中のそれぞれの選択行を探します。

クラスタ走査

索引クラスタに格納されている表からクラスタ走査を実行すると、クラスタ・キー値が同じである行が取り出されます。索引クラスタでは、同じクラスタ・キー値を持つすべての行が、同じデータ・ブロックに格納されます。クラスタ走査を実行する場合、Oracle はまず、クラスタ索引を走査して選択する行のどれか 1 つから ROWID を取得します。その後、Oracle はこの ROWID に基づいてそれらの行を探します。

ハッシュ走査

Oracle では、ハッシュ値に基づいてハッシュ・クラスタにある行を探す場合にハッシュ走査を使えます。ハッシュ・クラスタでは、同じハッシュ値を持つすべての行が同じデータ・ブロックに格納されます。ハッシュ走査を実行する場合、Oracle はまず、文に指定されたクラスタ・キー値にハッシュ関数を適用して、ハッシュ値を取得します。その後、Oracle は、そのハッシュ値が入っているデータ・ブロックを走査します。

索引走査

索引走査は、索引の 1 つ以上の列の値に基づいて、索引からデータを検索します。索引走査を実行する場合、Oracle は、その文がアクセスする索引付き列の値を索引から検索します。その文が索引の列だけをアクセスしている場合、Oracle はその索引付き列の値を、表からではなく、索引から直接読み込みます。

索引には、索引の対象になる値だけでなく、表の中のその値を持つ行の ROWID も入っています。したがって、その文が索引付きの列以外の列にもアクセスする場合、Oracle は ROWID による表アクセスまたはクラスタ走査を使って、表の行を検索できます。

索引走査には、次のタイプがあります。

一意走査	索引の一意走査は、ROWID を 1 つだけ戻します。多くの ROWID ではなく 1 つの ROWID だけが必要な場合、Oracle は一意走査を実行できます。たとえば、文が確実に 1 つの行だけをアクセスすることを保証する UNIQUE 制約または PRIMARY KEY 制約がある場合、Oracle は一意走査を実行します。
範囲走査	索引の範囲走査は、文がアクセスする行の数に応じて、ゼロ個以上の ROWID を戻します。
全走査	全索引走査は、述語が索引に含まれる列の 1 つを参照している場合に使えます。述語は、索引ドライバである必要はありません。さらに、全走査は、問合せで参照される表の中のすべての列が索引に含まれており、少なくとも 1 つの索引列が NULL 禁止になっている場合にも使用可能です。全走査は、ソート操作を排除するために使えます。この走査では、ブロックが単一で読み込まれます。
高速全走査	<p>高速全索引走査は、問合せに必要なすべての列が索引に含まれていて、索引キー内の少なくとも 1 つの列に NOT NULL 制約がある場合に、全表走査のかわりに使えます。高速全走査は、表にアクセスしないで、索引自体の中にあるデータにアクセスします。これは、ソート操作を排除するためには使えません。高速全走査は、(全索引走査と違って) マルチブロック読み込みを使って索引全体を読み込み、パラレル化することができます。</p> <p>この走査が使用可能なのは、コストベースで最適化する場合だけです。この走査は、初期化パラメータ FAST_FULL_SCAN_ENABLED または INDEX_FFS ヒントを使って指定できます。</p>

ビットマップ

ビットマップ索引は、キー値のビットマップと、各ビット位置を ROWID に変換するマッピング機能を使います。ビットマップは、ブール演算子を使って AND または OR 条件を解決し、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。ビットマップ・アクセスが使用可能なのは、コストベースで最適化する場合だけです。8-21 ページの「ビットマップ索引」を参照してください。

注意: ビットマップ索引は、Oracle8 Enterprise Editionを購入した場合にのみ利用できます。詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

アクセス・パス

表 20-1 に、データ・アクセス・パスのリストを示します。文に、特定のアクセス・パスを使用可能にする WHERE 句条件または他の構成要素が指定されている場合、オプティマイザはそのアクセス・パスを使って表にアクセスする以外に選択の余地はありません。

- コストベースのアプローチは、リソースの使用量に基づいてパスを選択します (20-55 ページの「コストベースのアプローチでのアクセス・パスの選択」を参照)。
- ルールベースのアプローチでは、複数のパスが使用可能な場合、それぞれのパスのランクを使います (20-58 ページの「ルールベースのアプローチでのアクセス・パスの選択」を参照)。

表 20-1 アクセス・パス

ランク	アクセス・パス
1	ROWID による単一行アクセス
2	クラスタ結合による単一行アクセス
3	一意キーまたは主キーが指定されているハッシュ・クラスタ・キーによる単一行アクセス
4	一意キーまたは主キーによる単一行アクセス
5	クラスタ結合
6	ハッシュ・クラスタ・キー
7	索引クラスタ・キー
8	複合キー
9	単一系列の索引
10	索引列に対する有界範囲検索
11	索引列に対する非有界範囲検索
12	ソート / マージ結合
13	索引列の MAX または MIN
14	索引列の ORDER BY
15	全表走査
ランクなしのアクセス・パス	
-	高速全索引走査（ルールベースのオプティマイザでは使用不可）。『Oracle8 Server チューニング』を参照
-	ビットマップ索引走査（ルールベースのオプティマイザでは使用不可）。8-23 ページの「ビットマップ索引」を参照

これ以降の各項では、これらのアクセス・パスと、それに関する次のことを説明します。

- そのアクセス・パスをいつ使えるか
- Oracle がそのアクセス・パスでデータをアクセスするときに使う方法
- そのアクセス・パスについて EXPLAIN PLAN コマンドで生成される出力

パス 1: ROWID による単一行アクセス

このアクセス・パスが使用可能なのは、文の WHERE 句により、ROWID または Oracle プリコンパイラでサポートされている CURRENT OF CURSOR 組込み SQL 構文によって選択された行が識別される場合だけです。この文を実行する場合、Oracle は ROWID によって表にアクセスします。

例：

このアクセス・パスは、次の文で使えます。

```
SELECT * FROM emp WHERE ROWID = 'AAAA7bAA5AAAA1UAAA';
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWI	EMP

パス 2: クラスタ結合による単一行アクセス

このアクセス・パスは、次の条件が両方とも真である場合、同じクラスタに格納されている表を結合する文に使えます。

- 文の WHERE 句に、一方の表のクラスタ・キーの各列を、もう一方の表の対応する列と等号で結ぶ条件が含まれている。
- 文の WHERE 句に、結合した結果として確実に 1 行だけを戻すことを保証する条件も含まれている。このような条件は、一意キーまたは主キーの列に対する等号による条件である場合がほとんどです。

これらの条件が、AND 演算子を使って結ばれている必要があります。この文を実行する場合、Oracle はネストされたループ操作を実行します。(ネストされたループ操作の詳細は、20-60 ページの「結合操作」を参照してください。)

例：

このアクセス・パスは、EMP 表と DEPT 表が DEPTNO 列に基づいてクラスタ化されており、EMPNO 列が EMP 表の主キーになっている場合に、次のような文に使えます。

```
SELECT *
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND emp.empno = 7900;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
NESTED LOOPS		
TABLE ACCESS	BY ROWID	EMP
INDEX	UNIQUE SCAN	PK_EMP
TABLE ACCESS	CLUSTER	DEPT

PK_EMP は、主キーを施行する索引の名前です。

パス3: 一意キーまたは主キーが指定されているハッシュ・クラスタによる単一行アクセス

このアクセス・パスは、次の条件が両方とも真である場合に使用可能です。

- 文の WHERE 句が、ハッシュ・クラスタ・キーのすべての列を等価条件で使っている。複合クラスタ・キーの場合は、すべての等価条件を AND 演算子を使って組み合わせる必要があります。
- ハッシュ・クラスタ・キーを構成する列が一意キーまたは主キーをも構成しているの
で、文が 1 行だけを戻すことが保証されている。

この文を実行する場合、Oracle は、文に指定されているハッシュ・クラスタ・キー値にクラスタのハッシュ関数を適用して、ハッシュ値を取得します。その後、Oracle は、そのハッシュ値を使ってその表にハッシュ走査を実行します。

例：

このアクセス・パスは、ORDERS 表と LINE_ITEMS 表がハッシュ・クラスタに格納されており、ORDERNO 列が ORDERS 表のクラスタ・キーと主キーになっている場合に、次のような文に使えます。

```
SELECT *
  FROM orders
 WHERE orderno = 65118968;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	HASH	ORDERS

パス 4: 一意キーまたは主キーによる単一行アクセス

このアクセス・パスは、文の WHERE 句が一意キーまたは主キーのすべての列を等価条件で使っている場合に使用可能です。複合クラスタ・キーの場合は、すべての等価条件は AND 演算子を使って組み合わせる必要があります。この文を実行する場合、Oracle は一意キーまたは主キーの索引に対して一意走査を実行し、単一の ROWID を検索してから、その ROWID によってその表をアクセスします。

例：

このアクセス・パスは、EMPNO 列が EMP 表の主キーになっている場合に、次のような文に使えます。

```
SELECT *
  FROM emp
 WHERE empno = 7900;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX	UNIQUE SCAN	PK_EMP

PK_EMP は、主キーを施行する索引の名前です。

パス 5: クラスタ結合

このアクセス・パスは、文の WHERE 句に、一方の表のクラスタ・キーの各列を、もう一方の表の対応する列と等号で結ぶ条件が含まれている場合に、同じクラスタに格納されている表を結合する文で使用可能です。複合クラスタ・キーの場合は、すべての等価条件を AND 演算子を使って組み合わせる必要があります。この文を実行する場合、Oracle はネストされたループ操作を実行します。(ネストされたループ操作の詳細は、20-60 ページの「結合操作」を参照してください。)

例：

このアクセス・パスは、EMP 表と DEPT 表が DEPTNO 列に基づいてクラスタ化されている場合に、次のような文に使えます。

```
SELECT *
  FROM emp, dept
 WHERE emp.deptno = dept.deptno;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
NESTED LOOPS		
TABLE ACCESS	FULL	DEPT
TABLE ACCESS	CLUSTER	EMP

パス 6: ハッシュ・クラスタ・キー

このアクセス・パスは、文の WHERE 句がハッシュ・クラスタ・キーのすべての列を等価条件の中で使っている文に使用可能です。複合クラスタ・キーの場合は、すべての等価条件を AND 演算子を使って組み合わせる必要があります。この文を実行する場合、Oracle は、文に指定されているハッシュ・クラスタ・キー値にクラスタのハッシュ関数を適用して、ハッシュ値を取得します。その後、Oracle は、そのハッシュ値を使ってその表にハッシュ走査を実行します。

例：

このアクセス・パスは、ORDERS 表と LINE_ITEMS 表がハッシュ・クラスタに格納されており、ORDERNO 列がクラスタ・キーである場合に、次のような文に使えます。

```
SELECT *
  FROM line_items
 WHERE orderno = 65118968;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	HASH	LINE_ITEMS

パス 7: 索引クラスタ・キー

このアクセス・パスは、文の WHERE 句が索引クラスタ・キーのすべての列を等価条件の中で使っている文に使用可能です。複合クラスタ・キーの場合は、すべての等価条件を AND 演算子を使って組み合わせる必要があります。この文を実行する場合、Oracle はクラスタ索引に対して一意走査を実行し、指定されたクラスタ・キー値が含まれている 1 つの行の ROWID を取得します。その後、Oracle は、その ROWID を使ってクラスタ走査で表にアクセスします。同じクラスタ・キー値が指定されているすべての行が一緒に格納されているので、クラスタ走査でそれらすべての行を検索するのに 1 つの ROWID だけですみます。

例：

このアクセス・パスは、EMP 表が索引クラスタに格納されており、DEPTNO 列がクラスタ・キーになっている場合に、次のような文に使えます。

```
SELECT * FROM emp
 WHERE deptno = 10;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	CLUSTER	EMP
INDEX	UNIQUE SCAN	PERS_INDEX

PERS_INDEX は、クラスタ索引の名前です。

パス 8: 複合索引

このアクセス・パスは、文の WHERE 句が複合キーのすべての列を、AND 演算子で組み合わせた等価条件で使っている場合に使用可能です。この文を実行する場合、Oracle は索引に対して範囲走査を実行し、選択する行の ROWID を取得してから、それらの ROWID を使って表にアクセスします。

例：

このアクセス・パスは、JOB 列と DEPTNO 列に対する複合索引がある場合に、次のような文で使えます。

```
SELECT *
  FROM emp
 WHERE job = 'CLERK'
    AND deptno = 30;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX	RANGE SCAN	JOB_DEPTNO_INDEX

JOB_DEPTNO_INDEX は、JOB 列と DEPTNO 列に対する複合索引の名前です。

パス 9: 単一系列の索引

このアクセス・パスは、文の WHERE 句が 1 つ以上の単一系列の索引の列を等価条件の中で使っている文に使用可能です。複数の単一系列索引の場合、これらの条件をすべて AND 演算子を使って組み合わせる必要があります。

WHERE 句が 1 つの索引の列だけを使っている場合、Oracle は文を実行するときに、その索引に対して範囲走査を実行し、選択する行の ROWID を取得してから、それらの ROWID によって表にアクセスします。

例：

このアクセス・パスは、EMP 表の JOB 列に対する索引がある場合に、次のような文で使えます。

```
SELECT *
  FROM emp
 WHERE job = 'ANALYST';
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX RANGE SCAN		JOB_INDEX

JOB_INDEX は、EMP.JOB に対する索引です。

WHERE 句に数多くの単一列の索引の列が使われている場合、Oracle は、文を実行するときに、それぞれの索引に対して範囲走査を実行し、それぞれの条件を満たす行の ROWID を検索します。その後、Oracle は、ROWID のセットをマージして、すべての条件を満たす行の ROWID のセットを取得します。最後に、Oracle は、それらの ROWID を使って表にアクセスします。

Oracle では、最大 5 つまで索引をマージできます。WHERE 句で 6 個以上の単一列の索引の列が使われている場合、Oracle はそのうち 5 つをマージして ROWID によって表にアクセスし、結果の行がその他の条件を満たしているかどうかを判断するテストを実行してから、それらの行を戻します。

例：

このアクセス・パスは、EMP 表の JOB 列と DEPTNO 列の両方に対する索引がある場合に、次のような文で使えます。

```
SELECT *
  FROM emp
 WHERE job = 'ANALYST'
        AND deptno = 20;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP

アクセス・パスの選択

AND-EQUAL		
INDEX	RANGE SCAN	JOB_INDEX
INDEX	RANGE SCAN	DEPTNO_INDEX

AND-EQUAL 操作は、JOB_INDEX と DEPTNO_INDEX の走査で取得した ROWID をマージし、その結果として、問合せの条件を満たす行の ROWID のセットを生成します。

パス 10: 索引列に対する有界範囲検索

このアクセス・パスは、文の WHERE 句に、単一列の索引か、複合索引の先頭部分を構成する 1 つ以上の列のどちらかを使った条件が含まれている場合に使用可能です。

column = expr

column >[=] expr AND column <[=] expr

column BETWEEN expr AND expr

column LIKE 'c%'

これらのそれぞれの条件は、文がアクセスする索引値の有界範囲を指定します。これらの条件は最小値と最大値を両方とも指定しているので、この範囲は有界です。この文を実行する場合、Oracle は索引に対して範囲走査を実行し、ROWID によって表にアクセスします。

このアクセス・パスは、式 expr が索引列を参照している場合には使えません。

例：

このアクセス・パスは、EMP 表の SAL 列に対する索引がある場合に、次のような文で使えます。

```
SELECT *
  FROM emp
 WHERE sal BETWEEN 2000 AND 3000;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX	RANGE SCAN	SAL_INDEX

SAL_INDEX は、EMP.SAL に対する索引の名前です。

例：

このアクセス・パスは、EMP 表の ENAME 列に対する索引がある場合に、次のような文でも使えます。

```
SELECT *
      FROM emp
     WHERE ename LIKE 'S%';
```

パス 11: 索引列に対する非有界範囲検索

このアクセス・パスは、文の WHERE 句に、単一系列の索引の列か、複合索引の先頭部分の 1 つ以上の列のどちらかを使った条件がどれか 1 つ含まれている場合に使用可能です。

```
WHERE column >[=] expr
```

```
WHERE column <[=] expr
```

これらのそれぞれの条件は、文がアクセスする索引値の非有界範囲を指定します。これらの条件は最小値か最大値（両方ではなくどちらか一方）を指定しているので、この範囲は非有界です。この文を実行する場合、Oracle は索引に対して範囲走査を実行し、ROWID によって表にアクセスします。

例：

このアクセス・パスは、EMP 表の SAL 列に対する索引がある場合に、次のような文で使えます。

```
SELECT *
      FROM emp
     WHERE sal > 2000;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX	RANGE SCAN	SAL_INDEX

例：

このアクセス・パスは、LINE_ITEMS 表の ORDER 列と LINE 列に対する複合索引がある場合に、次のような文で使えます。

```
SELECT *
      FROM line_items
     WHERE order > 65118968;
```

WHERE 句で ORDER 列、つまり索引の先頭部分を使っているので、このアクセス・パスを使えます。

例：

このアクセス・パスは、ORDER 列と LINE 列に対する索引がある場合に、次のような文では使えません。

```
SELECT *
  FROM line_items
 WHERE line < 4;
```

WHERE 句で、索引の先頭部分ではない LINE 列しか使われていないので、このアクセス・パスは使えません。

パス 12: ソート / マージ結合

このアクセス・パスは、文の WHERE 句でそれぞれの表の列が等価条件で使われている場合に、クラスタに一緒に格納されていない表を結合する文で使えます。そのような文を実行する場合、Oracle はソート / マージ操作を使います。また、Oracle は、結合文を実行するのにネストされたループ操作も使えます。(これらの操作の詳細は、20-59 ページの「結合文の最適化」を参照。)

例：

このアクセス・パスは、EMP 表と DEPT 表が同じクラスタに格納されていない場合に、次のような文に使えます。

```
SELECT *
  FROM emp, dept
 WHERE emp.deptno = dept.deptno;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
MERGE JOIN		
SORT	JOIN	
TABLE ACCESS	FULL	EMP
SORT	JOIN	
TABLE ACCESS	FULL	DEPT

パス 13: 索引列の MAX または MIN

このアクセス・パスは、次の条件がすべて真である場合、SELECT 文に使えます。

- 問合せが、MAX または MIN 関数を使って、単一系列の索引の列か複合索引の先頭列の最大値または最小値を選択している。この索引は、クラスタ索引であってはなりません。MAX または MIN 関数への引数としては、列、定数、または加算演算子 (+)、連結操作 (||)、または CONCAT 関数を含む任意の式を使えます。
- 選択リストに他の式がない。
- 文に WHERE 句や GROUP BY 句がない。

この問合せを実行する場合、Oracle は索引の範囲走査を実行し、最大または最小の索引値を検索します。この値だけを選択すればよいので、Oracle は索引を走査した後で表をアクセスする必要はありません。

例：

このアクセス・パスは、EMP 表の SAL 列に対する索引がある場合に、次のような文で使えます。

```
SELECT MAX(sal) FROM emp;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
AGGREGATE	GROUP BY	
INDEX	RANGE SCAN	SAL_INDEX

パス 14: 索引列の ORDER BY

このアクセス・パスは、次の条件がすべて真である場合、SELECT 文に使えます。

- 問合せに、単一系列索引の列か複合索引の先頭部分のどちらかを使う ORDER BY 句が入っている。この索引は、クラスタ索引であってはなりません。
- ORDER BY 句のリストに指定されている索引列のうち少なくとも 1 つは NULL 禁止の列であることを保証する、PRIMARY KEY 整合性制約または NOT NULL 整合性制約が必要。
- NLS_SORT パラメータが BINARY に設定されている。

この問合せを実行する場合、Oracle は索引の範囲走査を実行し、選択する行の ROWID をソート順に取得します。その後、Oracle は、それらの ROWID によって表にアクセスします。

例：

このアクセス・パスは、EMP 表の EMPNO 列に対する主キーがある場合に、次のような文で使えます。

```
SELECT *
FROM emp
```

```
ORDER BY empno;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	BY ROWID	EMP
INDEX	RANGE SCAN	PK_EMP

PK_EMP は、主キーを施行する索引の名前です。主キーの列には NULL 値が入らないことが保証されます。

パス 15: 全表走査

パス 15: 全表走査このアクセス・パスは、WHERE 句の条件には関係なく、任意の SQL 文で使えます。

次の条件により、索引アクセス・パスが使えなくなることに注意してください。

- column1 > column2
- column1 < column2
- column1 >= column2
- column1 <= column2

上記の column1 と column2 は同じ表にあるとします。

- column IS NULL
- column IS NOT NULL
- column NOT IN
- column != expr
- column LIKE '%pattern'

上記の column に索引があるかどうかは関係ありません。

- expr = expr2

上記の expr は、列に索引があるかどうかに関係なく、演算子または関数によって列を操作する式です。

- NOT EXISTS 副問合せ
- ビュー内の ROWNUM 疑似列
- 索引がない列を含む任意の条件

上記の構成体だけが含まれており、索引アクセス・パスが使えるようになる他の要素が含まれていない SQL 文では、全表走査を使う必要があります。

例：

この文は、全表走査を使って EMP 表にアクセスします。

```
SELECT *
  FROM emp;
```

この文の EXPLAIN PLAN 出力は、次のようになります。

OPERATION	OPTIONS	OBJECT_NAME

SELECT STATEMENT		
TABLE ACCESS	FULL	EMP

アクセス・パスの選択

ここでは、使用可能ないくつかのアクセス・パスの中からオブティマイザが選択する方法について説明します。

- ・ コストベースのアプローチを使っている場合
- ・ ルールベースのアプローチを使っている場合

コストベースのアプローチでのアクセス・パスの選択

コストベースのアプローチでは、オブティマイザは次の要因に基づいてアクセス・パスを選択します。

- ・ その文で使えるアクセス・パス
- ・ それぞれのアクセス・パスまたはパスの組合せを使って文を実行する場合の推定コスト

アクセス・パスを選択するために、Oracle はまず文の WHERE 句の条件を検査して、使用可能なアクセス・パスを判断します。その後、オブティマイザは、使用可能なアクセス・パスを使って考えられる一式の実行計画を生成し、文からアクセスできる索引および列、表の統計を使ってそれぞれの計画のコストを推定します。最後に、オブティマイザは、推定コストが最も低い実行計画を選択します。

オブティマイザによる使用可能なアクセス・パスの選択は、ヒントによって上書きが可能です。

追加情報：SQL 文のヒントの詳細は、『Oracle8 Server チューニング』を参照してください。

使用可能なアクセス・パスを選択するために、オブティマイザは次の要因を考慮します。

- 選択性: 「選択性」とは、表のうち、問合せが選択する行の割合（パーセント）のことです。表から選択する行の割合が小さい問合せの選択性は高く、選択する行の割合が大きい問合せの選択性は低いということになります。

オプティマイザは、選択性が低い問合せよりも選択性が高い問合せの場合に、全表走査ではなく索引走査を選択することが多くなります。通常、表の中でアクセスする行数の割合が低い問合せほど、索引走査のほうが全表走査よりも効率的です。他方、この割合が高い問合せの場合は、全表走査の処理のほうが高速です。

問合せの選択性を判断するために、オプティマイザは次の情報を検討します。

- WHERE 句で使われている演算子
- WHERE 句で使われている一意キー列と主キー列
- 表の統計情報

後述の例で、オプティマイザが選択性を利用する方法について説明します。

- DB_FILE_MULTIBLOCK_READ_COUNT: 全表走査ではマルチブロック読み込みを使うので、全表走査のコストは、表全体を読み込むのに必要なマルチブロック読み込みの回数によって決まります。このマルチブロック読み込みの回数は、1 回のマルチブロック読み込みによって読み込まれるブロック数によって決まります。そのブロック数は、初期化パラメータ DB_FILE_MULTIBLOCK_READ_COUNT によって指定されます。この理由で、オプティマイザは、このパラメータの値が高い場合に全表走査を選択することが多くなります。

例:

WHERE 句で等価条件を使って、JACKSON という名前のすべての従業員を選択するための次の問合せを考えます。

```
SELECT *  
  FROM emp  
 WHERE ename = 'JACKSON';
```

ENAME 列が一意キーまたは主キーの場合、オプティマイザは JACKSON という名前の従業員は 1 人しかいないと判断するので、問合せからは 1 行しか戻されません。この場合、問合せの選択性は非常に高いといえます。このため、オプティマイザは、一意キーまたは主キーを施行する索引に対する一意走査（アクセス・パス 4）を使って表にアクセスする可能性が高くなります。

例:

直前の例の問合せを再び考えます。ENAME 列が一意キーまたは主キーでない場合、オプティマイザは次のような統計情報を使って、この問合せの選択性を推定します。

- USER_TAB_COLUMNS.NUM_DISTINCT（表の各列の値の数）
- USER_TABLES.NUM_ROWS（各表の行数）

オブティマイザは、EMP 表の行数を ENAME 列の一意の値の数で割って、同じ名前の従業員が占める割合を推定します。オブティマイザは、ENAME 値が均一に分布していると仮定し、問合せの選択性の推定値としてこの割合を使います。

例：

従業員 ID 番号が 7500 より小さいすべての従業員を選択するための次の問合せを考えます。

```
SELECT *  
  FROM emp  
 WHERE empno < 7500;
```

この問合せの選択性を推定するために、オブティマイザは、WHERE 句条件にある範囲境界値 7500 と、EMPNO 列についての HIGH_VALUE 統計および LOW_VALUE 統計値（使用可能な場合）を使います。これらの統計は、USER_TAB_COLUMNS ビューにあります。オブティマイザは、EMPNO 値が最低値から最高値までの範囲内で均等に分布していると仮定します。次に、オブティマイザは、この範囲内で 7500 未満の値の割合を判断し、その値を問合せの選択性の推定値として使います。

例：

WHERE 句条件の範囲境界値にリテラル値ではなくバインド変数が使われている次の問合せを考えます。

```
SELECT *  
  FROM emp  
 WHERE empno < :e1;
```

オブティマイザには、バインド変数 E1 の値がわかりません。実際、E1 の値は、問合せを実行するたびに異なる可能性があります。この理由で、オブティマイザは、直前の例で説明した方法を使ってこの問合せの選択性を判断できません。この場合、オブティマイザは、列の選択性として小さい値をヒューリスティックに予測します（列に索引があるので）。オブティマイザは、演算子 <、または >、<=、>= のどれかが指定されている条件にバインド変数が範囲境界値として使われている場合に、必ずそのような仮定をします。

オブティマイザのバインド変数処理では、定数ではなくバインド変数を使っているという点だけが異なる SQL 文について、異なる実行計画を選択することがあります。この相違が特にはっきりする例として、Oracle プリコンパイラ・プログラムにおいて埋込み SQL 文にバインド変数が使われている場合と、SQL*Plus において同じ SQL 文に定数が指定される場合とでは、オブティマイザは異なる実行計画を選択する可能性があります。

例：

BETWEEN 演算子を使う条件で範囲境界値として 2 つのバインド変数を使っている次の問合せを考えます。

```
SELECT *  
  FROM emp
```

```
WHERE empno BETWEEN :low_e AND :high_e;
```

オプティマイザは、BETWEEN 条件を次の 2 つの条件に分解します。

```
empno >= :low_e  
empno <= :high_e
```

オプティマイザは、索引の使用を優先するために、索引列に対しては低い選択性をヒューリスティックに推定します。

例：

従業員 ID 番号が 7500 ~ 7800 のすべての従業員を BETWEEN 演算子を使って選択するための次の問合せを考えます。

```
SELECT *  
  FROM emp  
 WHERE empno BETWEEN 7500 AND 7800;
```

問合せの選択性を判断するために、オプティマイザは WHERE 句の条件を次の 2 つの条件に分解します。

```
empno >= 7500  
empno <= 7800
```

オプティマイザは、前の例で説明した方法を使って、それぞれの条件ごとの選択性を推定します。オプティマイザは、その後、選択性 (S1 および S2) と絶対値ファンクション (ABS) を使って、BETWEEN 条件の選択性 (S) を推定します。

$$S = \text{ABS}(S1 + S2 - 1)$$

ルールベースのアプローチでのアクセス・パスの選択

ルールベースのアプローチでは、オプティマイザは次の要因に基づいてアクセス・パスを選択します。

- その文で使えるアクセス・パス
- 20-43 ページの表 20-1 「アクセス・パス」に示されているアクセス・パスのランク

アクセス・パスを選択するために、Oracle はまず文の WHERE 句の条件を調べて、使用可能なアクセス・パスを判断します。その後、オプティマイザは、その中で最も上位にランクされている使用可能なアクセス・パスを選択します。

このリストの中では、全表走査が最下位にランクされています。したがって、ルールベースのアプローチでは、たとえ処理自体は全表走査のほうが高速であっても、索引が使用可能であれば、索引を使うアクセス・パスが必ず選択されます。

WHERE 句の条件の順序は、通常、オブティマイザによるアクセス・パスの選択には影響を与えません。

例：

EMP 表に含まれている従業員の中から、ENAME 値が 'CHUNG' で、SAL 値が 2000 より大きいすべての従業員の従業員番号を選択するための次の SQL 文を考えます。

```
SELECT empno
FROM emp
WHERE ename = 'CHUNG'
AND sal > 2000;
```

また、EMP 表には次の整合性制約と索引があるとします。

- EMPNO 列には、索引 PK_EMPNO によって施行される PRIMARY KEY 制約がある。
- ENAME 列には、ENAME_IND という名前の索引がある。
- SAL 列には、SAL_IND という名前の索引がある。

このとき、SQL 文の WHERE 句にある条件および整合性制約、索引に基づいて、次のアクセス・パスが使用可能です。

- ENAME = 'CHUNG' という条件により、ENAME_IND 索引を使う単一列索引アクセス・パスが使用可能になる。このアクセス・パスのランクは 9 です。
- SAL > 2000 という条件により、SAL_IND 索引を使う境界なしの範囲走査が使用可能になる。このアクセス・パスのランクは 11 です。
- すべての SQL 文について、全表走査は自動的に使用可能になる。このアクセス・パスのランクは 15 です。

WHERE 句の条件に索引列 EMPNO が含まれていないので、PK_EMPNO 索引により、主キーによる単一列アクセス・パスが使用可能になることはないので注意してください。

ルールベースのアプローチを使っている場合、オブティマイザは、ENAME_IND 索引を使うアクセス・パスを選択してこの文を実行します。オブティマイザは、使用可能な中で最も高いランクなので、このパスを選択します。

結合文の最適化

結合文の実行計画を選択するために、オブティマイザは、相互に関連する次のものを作成する必要があります。

アクセス・パス	単純な文の場合、オブティマイザは、結合文のそれぞれの表からデータを検索するためのアクセス・パスを選択する必要があります。 (20-40 ページの「アクセス・パスの選択」を参照。)
---------	---

結合操作	行ソースのそれぞれの対を結合するために、Oracleは次のいずれかの操作を実行する必要があります。 <ul style="list-style-type: none">• ネストされたループ• ソート / マージ• クラスタ• ハッシュ結合（ルールベース最適化では使用不可）
結合順序	3 つ以上の表を結合する文を実行する場合、Oracle はそのうちの 2 つの表を結合してから、その結果の行ソースを次の表に結合します。すべての表を結合した結果が完成するまで、この処理が続行されます。

結合操作

オブティマイザは、次の操作によって 2 つの行ソースを結合できます。

- ネストされたループ結合
- ソート / マージ結合
- クラスタ結合
- ハッシュ結合

ネストされたループ結合

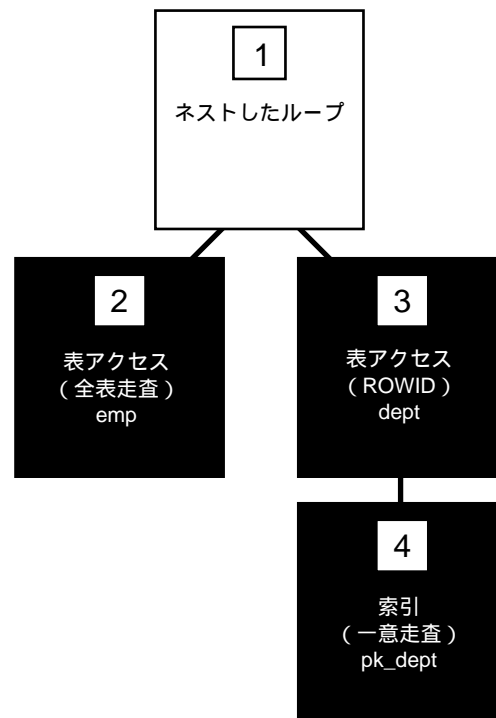
ネストされたループ結合を実行する場合、Oracle は次のステップに従います。

1. オブティマイザが、どれか 1 つの表を「外部表」（「駆動表」）として選択する。もう一方の表を「内部表」といいます。
2. 外部表の各行について、Oracle が、結合条件を満たす内部表の行を検出する。
3. Oracle は、結合条件を満たす行のそれぞれの対のデータを結合し、結果の行を戻す。

図 20-11 に、ネストされたループ結合を使った次の文の実行計画を示します。

```
SELECT *  
  FROM emp, dept  
 WHERE emp.deptno = dept.deptno;
```


図 20-11 ネストされたループ結合



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 2 で、全表走査を使って外部表 (EMP) にアクセスする。
- ステップ 2 から戻される各行について、ステップ 4 で、EMP.DEPTNO 値を使って PK_DEPT 索引の一意走査を実行する。
- ステップ 3 で、ステップ 4 で取得した ROWID を使って内部表 (DEPT) の一致する行を探す。
- Oracle は、ステップ 2 で戻された各行と、ステップ 4 で戻された一致する行を結合して、結果を戻す。

ソート / マージ結合

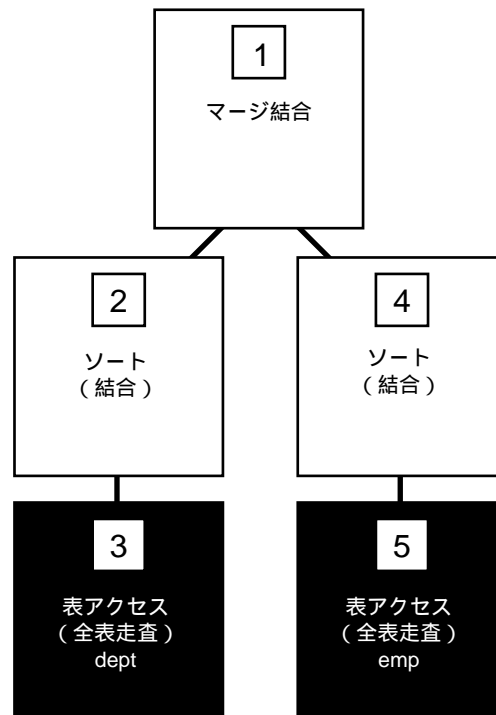
Oracle がソート / マージ結合を実行できるのは、同一レベル結合の場合だけです。ソート / マージ結合を実行する場合、Oracle は次のステップに従います。

1. 結合するそれぞれの行ソースが直前の操作でソートされていない場合、Oracle はこれらの行ソースをソートする。これらの行は、結合操作で使う列の値を基準にしてソートされます。
2. Oracle は、結合条件で使われた列に一致する値が入っている行の対を組み合わせて 2 つのソースをマージし、結果の行ソースとして戻す。

図 20-12 に、ソート / マージ結合を使った次の文の実行計画を示します。

```
SELECT *  
FROM emp, dept  
WHERE emp.deptno = dept.deptno;
```

図 20-12 ソート / マージ結合



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 3 と 5 で、EMP 表と DEPT 表に対して全表走査を実行する。
- ステップ 2 と 4 で、それぞれの行ソースを別々にソートする。

- ステップ 1 で、ステップ 2 のそれぞれの行をステップ 4 のそれぞれの一致する行と結合することによってステップ 2 とステップ 4 のソースをマージし、結果の行ソースを戻す。

クラスタ結合

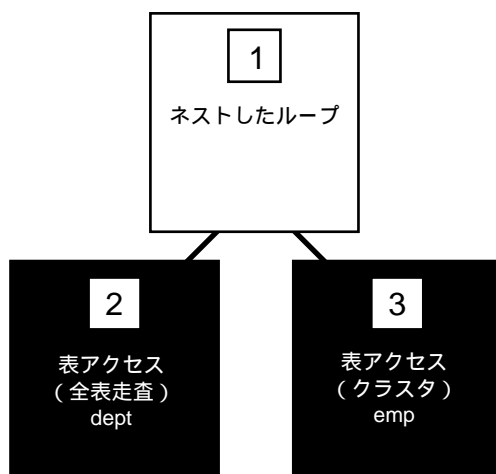
Oracle がクラスタ結合を実行できるのは、同じクラスタに入っている 2 つの表のクラスタ・キー列を等号で結ぶ同一レベル結合の場合だけです。クラスタでは、同じクラスタ・キー値を持つ両方の表の行が同じブロックに格納されるので、Oracle はそれらのブロックにアクセスするだけで済みます。

追加情報：『Oracle8 Server チューニング』には、最高のパフォーマンスを得るためにどの表をクラスタ化すればよいかを決定するためのガイドラインが記載されています。

図 20-13 に、同じクラスタに EMP 表と DEPT 表が一緒に格納されている場合の、次の文の実行計画を示します。

```
SELECT *
  FROM emp, dept
 WHERE emp.deptno = dept.deptno;
```

図 20-13 クラスタ結合



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 2 で、全表走査を使って外部表 (DEPT) にアクセスする。
- ステップ 2 で戻される各行について、ステップ 3 で、DEPT.DEPTNO 値を使ってクラスタ走査で内部表 (EMP) の一致する行を検出する。

クラスタ結合は、同じクラスタに格納された 2 つの表に対するネストされたループ結合にほかなりません。DEPT 表の各行は EMP 表の一致する行と同じデータ・ブロックに格納されているので、Oracle は一致する行を最も効率的にアクセスできます。

ハッシュ結合

Oracle がハッシュ結合を実行できるのは、同一レベル結合の場合だけです。ハッシュ結合は、ルールベース最適化では使えません。初期化パラメータ HASH_JOIN_ENABLED (ALTER SESSION コマンドで設定可能) または USE_HASH ヒントを使って、ハッシュ結合の最適化を使用可能にする必要があります。

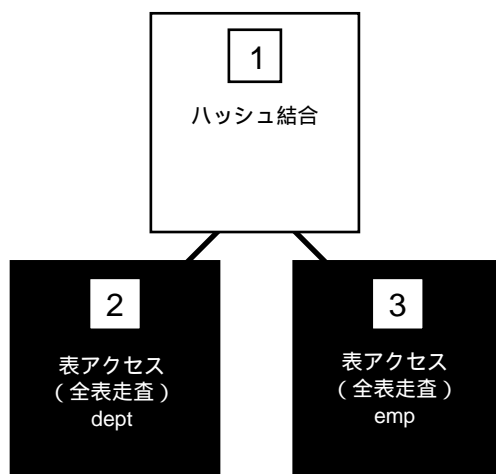
ハッシュ結合を実行する場合、Oracle は次のステップに従います。

1. Oracle は、それぞれの表に対して全表走査を実行し、それぞれの表を、どれだけのメモリーを使えるかに基づいて可能な限り多くの数のパーティションに分ける。
2. Oracle は、どれか 1 つのパーティションからハッシュ表を作る (可能なら、Oracle は、使用可能なメモリーに収まるサイズのパーティションを選択)。その後、Oracle は、外部表の対応するパーティションを使ってハッシュ表を調べます。メモリーに入りきらないサイズのパーティションの対は、すべてディスクに置かれます。
3. Oracle は、パーティション (各表から 1 つずつ) の対ごとに、小さい方のパーティションを使ってハッシュ表を作り、大きい方のパーティションを使ってハッシュ表を調べます。

図 20-14 に、ハッシュ結合を使った次の文の実行計画を示します。

```
SELECT *  
  FROM emp, dept  
 WHERE emp.deptno = dept.deptno;
```

図 20-14 ハッシュ結合



この文を実行する場合、Oracle は次のステップを実行します。

- ステップ 2 と 3 で、EMP 表と DEPT 表に対して全表走査を実行する。
- ステップ 1 で、ステップ 2 の結果の行からハッシュ表を作り、ステップ 3 の結果の各行を使ってハッシュ表を調べる。

ハッシュ結合操作に使うメモリー量を制御するには初期化パラメータ `HASH_AREA_SIZE` を使い、ハッシュ結合操作が同時に読み書きする必要のあるブロック数を制御するには初期化パラメータ `HASH_MULTIBLOCK_IO_COUNT` を使います。

追加情報：これらの初期化パラメータと `USE_HASH` ヒントの詳細は、『Oracle8 Server チューニング』を参照してください。

結合文の実行計画の選択

ここでは、オブティマイザが結合文の実行計画を選択する方法を説明します。

- コストベースのアプローチを使っている場合
- ルールベースのアプローチを使っている場合

コストベースおよびルールベースのアプローチに当てはまる、次の考慮事項に注意してください。

- オブティマイザは、まず、2 つ以上の表を結合したときに、多くても 1 行しか含まれない行ソースが生成されるかどうかを判断します。オブティマイザは、それらの表の `UNIQUE` 制約および `PRIMARY KEY` 制約に基づいてそうした状況を認識します。そのよ

うな状況がある場合、オブティマイザはそれらの表を結合順序の先頭に配置します。その後、オブティマイザは、その他の表セットの結合を最適化します。

- 外部結合条件が使われている結合文の場合、外部結合演算子が使われている表は、結合順序ではもう一方の表の後に置かれます。オブティマイザは、このルールに違反する結合順序は考慮しません。

コストベースのアプローチで結合の実行計画を選択する

コストベースのアプローチの場合、オブティマイザは、有効な結合順序および結合操作、使用可能なアクセス・パスに基づいて実行計画のセットを生成します。その後、オブティマイザは、それぞれの計画のコストを推定し、コストが最も低い計画を選択します。オブティマイザは、次の方法でコストを推定します。

- ネストされたループ操作のコストは、外部表から選択される各行と、内部表から取得される一致する各行をメモリーに読み込むコストに基づいて推定します。オブティマイザは、これらのコストをデータ・ディクショナリの統計を使って推定します。
- ソート/マージ結合のコストは、すべてのソースをメモリーに読み込み、それらのソースをソートする場合のコストに基づいてほとんど決まります。
- オブティマイザは、それぞれの操作のコストを決定する際、その他の要因も考慮します。たとえば、次のとおりです。
 - ソート領域が小さいほどソート処理により多くの CPU 時間と I/O を消費するので、ソート領域サイズが小さいほどソート/マージ結合のコストが高くなる傾向があります。ソート領域サイズは、初期化パラメータ SORT_AREA_SIZE で指定します。
 - マルチブロックの読み込み数を多くすると、ネストされたループ結合よりも、ソート/マージ結合のほうがコストが低くなる傾向があります。1 回のディスク I/O で読み込める順次ブロックの数が増えると、ネストされたループ走査のパフォーマンスが内部表にある索引によって全表走査よりも改善される程度が低くなるからです。マルチブロック読み込み数は、初期化パラメータ DB_FILE_MULTIBLOCK_READ_COUNT で指定します。
 - 外部結合条件が使われている結合文の場合、外部結合演算子が使われている表は、結合順序ではもう一方の表の後に置かれます。オブティマイザは、このルールに違反する結合順序は考慮しません。

コストベースのアプローチの場合、オブティマイザによる結合順序の選択は、ORDERED ヒントを使って上書きできます。ORDERED ヒントで外部結合のルールに違反する結合順序を指定すると、オブティマイザはそのヒントを無視して順序を選択します。また、オブティマイザによる結合操作の選択を、このヒントを使って上書きすることもできます。

追加情報：ヒントの使用の詳細は、『Oracle8 Server チューニング』を参照してください。

ルールベースのアプローチで結合の実行計画を選択する

ルールベースのアプローチの場合、オブティマイザは、次のステップに従って、R 個の表を結合する文の実行計画を選択します。

1. オプティマイザは、異なる表が先頭になっている、R 個の結合順序のセットを生成します。オプティマイザは、次のアルゴリズムを使って、考えられるそれぞれの結合順序を生成します。
 - a. 結合順序の各位置を埋めるために、オプティマイザは 20-43 ページの表 20-1 「アクセス・パス」で示したアクセス・パスのランクに従って、最も上にランクされている使用可能なアクセス・パスのある表を選択します。オプティマイザは、このステップを繰り返して、結合順序でその後続くそれぞれの位置を埋めていきます。
 - b. オプティマイザは、さらに、結合順序内の各表について、その表を結合順序内の直前の表または行ソースに結合するための操作も選択します。その際に、オプティマイザは、ソート / マージ操作をアクセス・パス 12 として「ランキング」した上で、次のルールを適用します。
 - 選択した表のアクセス・パスが 11 かそれより高いランクになっている場合、オプティマイザは、結合順序内の直前の表または行ソースを外部表として使うネストされたループ操作を選択します。
 - その表のアクセス・パスのランクが 12 より低く、選択した表と結合順序内の直前の表または行ソースとの間に同一レベル結合条件が存在する場合、オプティマイザはソート / マージ操作を選択します。
 - 選択した表のアクセス・パスのランクが 12 より低く、同一レベル結合条件がない場合、オプティマイザは、結合順序内の直前の表または行ソースを外部表として使うネストされたループ操作を選択します。
2. その後、オプティマイザは、生成された実行計画の中から選択します。オプティマイザの選択目標は、内部表を索引走査を使ってアクセスするネストされたループ結合操作の数を最大化することです。ネストされたループ結合には内部表を何回もアクセスする操作が含まれるので、内部表に索引があると、ネストされたループ結合のパフォーマンスが大幅に向上します。

通常、実行計画を選択する際に、オプティマイザは FROM 句に表が記述される順序を考慮しません。オプティマイザは、次のルールを順番に適用することによって実行計画を選択します。

 - a. オプティマイザは、内部表を全表走査でアクセスする、ネストされたループ操作の数が最も少ない実行計画を選択します。
 - b. 同順位の実行計画がいくつかある場合、オプティマイザは、ソート / マージ操作の数が最も少ない実行計画を選択します。
 - c. それでもまだ同順位の実行計画がある場合、オプティマイザは、結合順序内の先頭の表が最上位のアクセス・パスにランクされている実行計画を選択します。
 - 先頭の表が単一列索引アクセス・パスでアクセスされる複数の実行計画が同順位である場合、オプティマイザは、マージ率が最も高い索引を使って先頭の表がアクセスされる計画を選択します。

- 先頭の表が境界付き範囲走査を使ってアクセスされる複数の実行計画が同順位である場合、オブティマイザは、複合索引の先頭部分にある列のうち最も多くを使って先頭の表にアクセスする計画を選択します。
- d. それでもまだ同順位の実行計画がある場合、オブティマイザは、先頭の表が問合せの FROM 句の後ろのほうに出てくる実行計画を選択します。

外部結合のビュー

外部結合の右側にあるビューの場合、オブティマイザは、ビューがアクセスする実表の数に応じて、次の 2 つの方法のどちらかを使うことができます。

- ビューに実表が 1 つしかない場合、オブティマイザは「ビューのマージ」を行うことができる。
- ビューに複数の実表がある場合、オブティマイザはビューに「結合述語を追加」できる。

単一の実表を持つビューのマージ

1 つの実表を持ち、外部結合の右側にあるビューは、アクセスする文の問合せブロックにマージできます。(20-22 ページの「文へのビューの問合せのマージ」を参照。) ビュー内の式が NULL に対して NULL 以外の値を返す可能性があっても、ビューのマージは可能です。

例：

EMP 表の最初の名前と最後の名前を連結するビュー NAME_VIEW について考えます。

```
CREATE VIEW name_view
  AS SELECT emp.firstname || emp.lastname AS emp_fullname, emp.deptno
  FROM emp;
```

また、ロンドンのすべての従業員の名前とその部門、および従業員がいない部門を検索する次の外部結合文を考えます。

```
SELECT dept.deptno, name_view.emp_fullname
  FROM emp_fullname, dept
 WHERE dept.deptno = name_view.deptno(+)
        AND dept.deptloc = 'London';
```

オブティマイザは、ビューの問合せを外部結合文にマージします。この結果の文は、次のようになります。

```
SELECT dept.deptno, DECODE(emp.rowid, NULL, NULL, emp.firstname || emp.lastname)
  FROM emp, dept
 WHERE dept.deptno = emp.deptno(+)
        AND dept.deptloc = 'London';
```

変換された文は、ロンドンに勤務する従業員だけを選択します。

複数の実表を持つビューへの結合述語の追加

外部結合の右側に複数の実表を持つビューの場合、初期化パラメータ PUSH_JOIN_PREDICATE が TRUE に設定されているか、アクセスする問合せに PUSH_JOIN_PRED ヒントが含まれていれば、オブティマイザは結合述語をビューに追加できます（20-25ページの「ビューへの述語の追加」を参照）。

結合述語の追加はコストベースの変換であり、ネストされたループ結合へのハッシュ結合の変換や、索引走査への全表走査の変換など、より効率のよいアクセス・パスと結合方法を使用可能にできます。

追加情報：オブティマイザ・ヒントの詳細は、『Oracle8 Server チューニング』を参照してください。

例：

ロンドンに勤務する従業員を選択するビュー LONDON_EMP について考えます。

```
CREATE VIEW london_emp
AS SELECT emp.ename
   FROM emp, dept
   WHERE emp.deptno = dept.deptno
      AND dept.deptloc = 'London';
```

また、ロンドンに勤務する技術者および会計担当者のうち、ボーナスを受け取った者を検索する次の外部結合文を考えます。

```
SELECT bonus.job, london_emp.ename
   FROM bonus, london_emp
  WHERE bonus.job IN ('engineer', 'accountant')
     AND bonus.ename = london_emp.ename(+);
```

オブティマイザは、外部結合述語をビューに追加します。結果の文（標準の SQL 構文には準拠しない）は次のようになります。

```
SELECT bonus.job, london_emp.ename
   FROM bonus, (SELECT emp.ename FROM emp, dept
                 WHERE bonus.ename = london_emp.ename(+)
                   AND emp.deptno = dept.deptno
                   AND dept.deptloc = 'London')
  WHERE bonus.job IN ('engineer', 'accountant');
```

非結合と半結合の最適化

「非結合」は、述語の左側の行のうち、述語の右側に対応する行を持たない行を戻します。つまり、右側の副問合せに合致しない (NOT IN) 行を戻します。たとえば、非結合では、特定の部門の集合に属さない従業員のリストを選択できます。

```
SELECT * FROM emp
WHERE deptno NOT IN
  (SELECT deptno FROM dept
   WHERE loc = 'HEADQUARTERS');
```

オブティマイザは、デフォルトではNOT IN副問合せにネストされたループのアルゴリズムを使います。ただし、初期化パラメータ ALWAYS_ANTI_JOIN が MERGE または HASH に設定されていて、NOT IN 副問合せをソート/マージまたはハッシュ非結合に変換するために必要な各種条件が満たされている場合は例外です。MERGE_AJ または HASH_AJ ヒントを NOT IN 副問合せに入れて、オブティマイザが使うアルゴリズムを指定できます。

「半結合」は、右側にある複数の行が副問合せの基準を満たすときに、述語の左側から行を複製することなく、EXISTS副問合せに合致する行を戻します。たとえば、次のとおりです。

```
SELECT * FROM dept
WHERE EXISTS
  (SELECT * FROM emp
   WHERE dept.ename = emp.ename
   AND emp.bonus > 5000);
```

問合せでは、EMP 内の多数の行が副問合せに合致したとしても、DEPT から戻される行は 1 行でなければなりません。EMP 内の BONUS 列に索引がない場合は、半結合を使って問合せのパフォーマンスを改善できます。

オブティマイザは、デフォルトではEXISTS副問合せにネストされたループのアルゴリズムを使います。ただし、初期化パラメータ ALWAYS_SEMI_JOIN が MERGE または HASH に設定されていて、必要な各種条件が満たされている場合は例外です。MERGE_SJ または HASH_SJ ヒントを EXISTS 副問合せに入れて、オブティマイザが使うアルゴリズムを指定できます。

追加情報：オブティマイザ・ヒントの詳細は、『Oracle8 Server チューニング』を参照してください。

「スター」問合せの最適化

データ・ウェアハウス設計のタイプの 1 つとして、「スター型」スキーマが注目されています。それは、データ・ウェアハウスのうち主要な情報が含まれている 1 つ以上の非常に大規模な「事実」表と、その事実表内の特定の属性のエントリに関する情報が含まれているずっと小規模なたくさんの「ディメンション」表（「参照」表）とによって特徴付けられます。

「スター問合せ」とは、1つの事実表とたくさんの参照表とを結合する問合せです。それぞれの参照表は、主キーから外部キーへの結合によって事実表に結合されますが、参照表どうしは結合されません。

Oracle のコストベースのオプティマイザは、スター問合せを認識し、そのような問合せの効率的な実行計画を生成します。(スター問合せは、ルールベースのオプティマイザでは認識されません。)

典型的な事実表には、「キー」と「量目」が入っています。たとえば、単純な事実表として、「売上げ」という量目と、「日時」、「製品」、「市場」というキーが入っている表が考えられます。この場合は、「日時」、「製品」、および「市場」に対応するディメンション表が存在することになります。たとえば、典型的な「製品」ディメンション表には、事実表に記載されているそれぞれの製品番号に関する情報が入れます。

「スター型結合」とは、ディメンション表から事実表への、主キー対外部キーによる結合のことです。通常、事実表にはキー列に対する連結索引があり、このタイプの結合を容易に行できるようにになっています。

スター問合せの例

ここでは、次の例を使って、スター問合せを説明します。

```
SELECT SUM(dollars)
FROM facts, time, product, market
WHERE market.stat = 'New York'
      AND product.brand = 'MyBrand'
      AND time.year = 1995
      AND time.month = 'March'
/* Joins*/
      AND time.key = facts.tkey
      AND product.pkey = facts.pkey
      AND market.mkey = facts.mkey;
```

スター問合せのチューニング

スター問合せを効率的に実行するには、コストベースのオプティマイザを使う必要があります。まず ANALYZE コマンドを使って、問合せによってアクセスされるそれぞれの表の統計情報を収集することから作業を始めます。

索引

上記の例では、tkey および pkey、mkey 列に対して連結索引を組み立てることができます。この索引での列の順序は、パフォーマンスに決定的な影響を与えます。索引の列は、データの順序付けを活用するべきです。大規模な表に日時の順で行が追加されている場合は、日時の列を索引の最初のキーにしてください。データが別のデータベースから抽出した統計である場合は、そのデータをキー列に基づいてソートしてからロードしてください。

すべての問合せが小規模な表のそれぞれに対する述語を指定している場合は、単一の連結索引があれば十分です。連結索引の先頭列をいくつか省略している問合せがある場合は、索引を追加すると便利です。ここで考えている例で、日時表を省略した問合せが頻繁に発行される場合は、pkey と mkey に対する索引を追加できます。

ヒント

通常、表を分析する際に、オプティマイザは効率的なスター計画を選択します。また、ヒントを使って、その計画を改善することもできます。最も正確な方法は、FROM 句に指定する表の順序を索引のキーの順序にし、大規模な表を最後に指定するようにすることです。その後、次のようなヒントを使います。

```
/*+ ORDERED USE_NL(facts) INDEX(facts fact_concat) */
```

もっと一般的なのは、STAR ヒント「/*+ STAR */」を使う方法です。

拡張スター・スキーマ

それぞれの小規模な表は、いくつかの小規模な表を結合することによって置き換えることができます。たとえば、製品表は、ブランド表と製造業者表に正規化できます。小規模な表をすべて正規化すると、パフォーマンスの問題が発生する可能性があります。オプティマイザが考慮しなければならない順列の数が多くなりすぎると、そのような問題が発生します。他にも、小規模な表の結合を何回も実行することから発生する問題もあります。どちらの問題も、非正規化ビューを使えば解決できます。たとえば、次のとおりです。

```
CREATE VIEW prodview AS SELECT /*+ NO_MERGE */ *  
FROM brands, mfgrs WHERE brands.mfkey = mfgrs.mfkey;
```

このヒントにより、オプティマイザの検索範囲が狭められ、ビューの結果がキャッシュされるようになります。

スター型変換

スター型変換とは、スター問合せを効率的に実行することを目ざした、コストベースの問合せ変換のことです。次元表の数が少なく事実表の密度が高いスキーマでは、スター最適化が優れた機能を果たしますが、次のどれかが真である場合は、別の方法としてスター型変換を考慮できます。

- 次元表の数が多い。
- 事実表の密度が低い。
- いくつかの次元表に対する制約述語が指定されていない問合せがある。

スター型変換は次元表の直積演算には依存していないので、事実表の密度が低かったり、次元表の数が多すぎたりするために、大規模な直積演算をしなければならないのに事実表で実際に一致する行はごくわずかである、という状況に適しています。さらに、スター型変換は、連結索引ではなく、個々の事実表の列に対するビットマップ索引を組み合わせる方法を基礎としています。

このように、この変換方法では、制約次元に正確に対応する索引の組合せを選択できます。いろいろな列順序がいろいろな問合せのいろいろな制約次元のパターンと一致する、いくつもの索引連結を作る必要はありません。

注意: ビットマップ索引は、Oracle8 Enterprise Editionを購入した場合にのみ利用できます。Oracle8ではビットマップ索引が利用不能であり、スター問合せ処理はB* ツリー索引を使います。Oracle8 Enterprise Editionでは、スター問合せ処理に、パラレル・ビットマップ索引結合のアルゴリズムを使うこともできます。

Oracle8とOracle8 Enterprise Editionで使用可能な機能の詳細は、『Oracle8とOracle8 Enterprise Editionの解説』を参照してください。

スター型変換は、事実表に対するビットマップ索引アクセス・パスを駆動するのに使える、新しい副問合せを生成することによって機能します。

ディメンション表が"d1"、"d2"、"d3"の3つ、事実表が"fact"1つ、という単純な構成の場合を考えてみます。次のような問合せがあるとします。

```
EXPLAIN PLAN FOR
SELECT * FROM fact, d1, d2, d3
WHERE fact.c1 = d1.c1 AND fact.c2 = d2.c1 AND fact.c3 = d3.c1
AND d1.c2 IN (1, 2, 3, 4)
AND d2.c2 < 100
AND d3.c2 = 35
```

この問合せは、次の3つの副問合せを追加することにより変換されます。

```
SELECT * FROM fact, d1, d2
WHERE fact.c1 = d1.c1 AND fact.c2 = d2.c1
AND d1.c2 IN (1, 2, 3, 4)
AND d2.c2 < 100
AND fact.c1 IN (SELECT d1.c1 FROM d1 WHERE d1.c2 IN (1, 2, 3, 4))
AND fact.c2 IN (select d2.c1 FROM d2 WHERE d2.c2 < 100)
AND fact.c3 IN (SELECT d3.c1 FROM d3 WHERE d3.c2 = 35)
```

fact.c1、fact.c2、および fact.c3 にビットマップ索引があるとすれば、新しく生成したこれらの問合せを使って、次のような方法でビットマップ索引アクセス・パスを駆動できます。最初の副問合せから取得される d1.c1 のそれぞれの値について、その値のビットマップが d1.c1 の索引から取得され、それらのビットマップがマージされます。結果として、副問合せの WHERE 句にある d1 の条件と正確に一致する、fact 表の行のビットマップが生成されます。

同様に、2つ目の副問合せの値を fact.c2 のビットマップ索引と組み合わせることにより、2つ目の副問合せの d2 の条件と一致する fact 表の行に対応する、マージされたビットマップを生成できます。3つ目の副問合せにも、同じ操作を適用します。これら3つのマージされたビットマップを AND で結合すれば、3つの副問合せの条件を同時に満たす fact 表の行に対応するビットマップが生成されます。

このビットマップを使って fact 表にアクセスし、関連する行を取り出せます。これらの行を d1、d2、d3 に結合すれば、問合せに対する回答が生成されます。直積演算は必要ありません。

実行計画

上記の問合せは、次のような実行計画になります。

```
SELECT STATEMENT
  HASH JOIN
    HASH JOIN
      HASH JOIN
        TABLE ACCESS                FACT                BY ROWID
        BITMAP CONVERSION              TO ROWIDS
        BITMAP AND
          BITMAP MERGE
            BITMAP KEY ITERATION
              TABLE ACCESS            D3                FULL
              BITMAP INDEX             FACT_C3            RANGE SCAN
            BITMAP MERGE
              BITMAP KEY ITERATION
                TABLE ACCESS D1 FULL
                BITMAP INDEX          FACT_C1            RANGE SCAN
              BITMAP MERGE
                BITMAP KEY ITERATION
                  TABLE ACCESS        D2                FULL
                  BITMAP INDEX          FACT_C2            RANGE SCAN
                TABLE ACCESS          D1                FULL
              TABLE ACCESS            D2                FULL
            TABLE ACCESS              D3                FULL
```

この計画では、3つのマージされたビットマップのビットマップ AND に基づくビットマップ・アクセス・パスによって、fact 表にアクセスしています。3つのビットマップは、自分より下の行ソース・ツリーからビットマップを提供してもらって、BITMAP MERGE 行ソースによって生成されます。そのような行ソース・ツリーはそれぞれ、副問合せの行ソース・ツリーから値をフェッチする BITMAP KEY ITERATION 行ソース（この例では、単に全表アクセス）で構成されており、ツリーからフェッチされる値のビットマップをビットマップ索引から取り出します。このアクセス・パスを使って、関連する fact 表の行が取り出された後、そ

これらの行が次元表と結合されて、問合せへの回答が生成されます。

スター型変換は、次のような意味でコストベースの変換といえます。オブティマイザは、変換なしで生成できる最良の計画を生成し、保存します。変換が使用可能になると、オブティマイザはその変換を問合せに適用しようとし、適用が可能な場合は、変換後の問合せを使って最良の計画を生成します。変換前と変換後の2つのバージョンの最良の計画の推定コストを比較した結果に基づいて、オブティマイザはこの2つのバージョンのうちどちらの計画を使うかを決定します。

問合せで、事実表にある行のかなりの割合をアクセスする必要がある場合は、変換を使うよりも全表走査を使うほうがよいでしょう。しかし、次元表の制約述語の選択性が十分に高く、事実表のわずかな部分だけを検索すればよい場合は、おそらく変換に基づいた計画のほうが優れています。

オブティマイザが次元表に対する副問合せを生成するのは、基準の数に基づいてそうするのが妥当と判断した場合だけです。すべての次元表について副問合せが必ず生成されるという保証はありません。また、オブティマイザは、表と問合せの特性に基づいて、特定の問合せには変換を適用しても利点はないと判断することがあります。この場合は、標準的な計画の中で最良のものを使います。

スター型変換の使用方法

スター型変換を使用可能にするには、初期化パラメータ `STAR_TRANSFORMATION_ENABLED` を `TRUE` に設定します。 `STAR_TRANSFORMATION` ヒントを使うと、オブティマイザは、変換が適用された中で最良の計画を使うようになります。

スター型変換の制限事項

次のいずれかの特性がある表では、スター型変換はサポートされません。

- ビットマップ・アクセス・パスとの互換性がない表ヒントが指定されている表
- ビットマップ索引が少なすぎる表（事実表の列への副問合せの生成をオブティマイザの候補にするには、その列に対するビットマップ索引が存在していなければならない）
- リモート表（ただし、生成される副問合せの中でのリモート・ディメンション表は許される）
- 反結合表
- 副問合せの中ですでにディメンション表として使われている表
- 実際にはマージされていないビューであり、ビュー・パーティションではない表
- 優れた単一表アクセス・パスがある表
- 変換が効果を発揮するには小さすぎる表

第 VI 部

パラレルSQLとダイレクト・ロード・インサート

第VI部では、SQL文のパラレル実行とダイレクト・ロード・インサート機能を説明します。
含まれる章は、次のとおりです。

- 第21章「ダイレクト・ロード・インサート」
- 第22章「パラレル実行」

ダイレクト・ロード・インサート

The translator of Homer should above all be penetrated by a sense of four qualities of his author: that he is eminently rapid; that he is eminently plain and direct ... both in his syntax and in his words; that he is eminently plain and direct in the substance of his thought, ...; and, finally, that he is eminently noble.

Matthew Arnold: *On Translating Homer*

この章では、シリアル・インサートまたはパラレル・インサートのための 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・インサートの基礎知識
- 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
 - ログング・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

パラレル固有の情報は、第 22 章「パラレル実行」を参照してください。

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、Oracle8 Enterprise Editionを購入した場合にのみ利用できます。詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

追加情報：パラレル・ダイレクト・ロード・インサートの調整方法の詳細は、『Oracle8 Server チューニング』を参照してください。

ダイレクト・ロード・インサートの基礎知識

「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・キャッシュを使わ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の中へ、パラレルかシリアルのどちらかで挿入できます。

並行性、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。これらの機能については、21-3 ページの「ダイレクト・ロード・インサート文の種類」に記載されています。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パーティション化オプションの詳細は、第 22 章「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・インサートを使うことの主な利点は、REDO エントリまたは UNDO エントリのログをとらずにデータをロードできるということです。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においても、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 を使った場合は、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・インサートを使うと、新しい表を作ることなく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るだけです。詳細は、22-22 ページの「パラレルの CREATE TABLE ... AS SELECT」を参照してください。

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 を使った場合、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・ロードを使う場合には、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は "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、表と索引が自動的に更新されます (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)。

追加情報: パラレル・ロードの詳細は、『Oracle8 Server ユーティリティ』を参照してください。

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 構文だけであり、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・ヒントまたは表定義のパラレル句のどちらかによって決定されます。

追加情報: INSERT ... SELECT 文の構文の詳細は、『Oracle8 Server 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・インサートには、次のような種類があります。

- シリアルまたはパラレル
- 非パーティション表または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。

- 非パーティション表または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、表セグメントまたは各パーティション・セグメントの現在の最後の位置の後に挿入されます。コミットを実行すると、最後の位置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・サーバー・プロセスは、新しい一時セグメントを割り当てて、データをその一時セグメントに挿入します。コミットを実行すると、コーディネータ・プロセスは新しい一時セグメントをマージして 1 次表セグメントへマージします。（コーディネータ・プロセスとパラレル・サーバー・プロセスの詳細は、22-5 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。）
- 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・サーバー・プロセスは、1 つ以上のパーティションに割り当てられ、どのパーティションについても作動しているのは 1 つのプロセスだけです。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最後の位置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・セグメントの最後の位置がコーディネータ・プロセス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、最後の位置の更新または一時セグメントのマージは、コミットが発行されるまで待ってから実行されます。それが実行されると、すぐに他のプロセスからもデータが見えるようになるからです（つまり、挿入操作をコミットします）。

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・インサートの指定

シリアル・ダイレクト・ロード・インサートを使うには、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、文中では PARALLEL ヒントが、あるいは表定義では 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 PARALLEL DML文を使ってオンにする必要もあります。

表21-1に、ダイレクト・ロード・インサートと従来型INSERTを比較しつつ、これらの要件をまとめます。

表 21-1 シリアル INSERT ... SELECT 文とパラレル INSERT ... SELECT 文のまとめ

挿入の種類	シリアル	パラレル
ダイレクト・ロード・インサート	可能。次のものが必要。 <ul style="list-style-type: none"> SQL 文での APPEND ヒント 	可能。次のものが必要。 <ul style="list-style-type: none"> ALTER SESSION ENABLE 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)	不可能

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 属性を設定することによって指定できます。次に例を示します。

```
ALTER TABLE emp PARALLEL (DEGREE 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 属性を設定することにより、次のように、SELECT 操作の並行性も指定できます。

```
ALTER TABLE emp PARALLEL (DEGREE 10);
ALTER TABLE t_emp PARALLEL (DEGREE 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 文の並行度は、EMP 表および 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;
```

パラレルINSERT文の詳細は、22-18ページの「INSERT ... SELECTのパラレル化のルール」を参照してください。

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。データ挿入先の表またはパーティション、索引にロギングなしのモードを指定するには、ALTER TABLEコマンドまたはALTER INDEXコマンド、ALTER TABLESPACEコマンドを使います。

- 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、インスタンス回復とメディア回復のための完全なREDOロギングが実行されます。「ロギングあり」がデフォルトのモードです。
- 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REDOロギングまたはUNDOロギングなしで挿入されます。(ただし、新しいエクステン트가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完全にログが取られます。) メディア回復時にこれが適用されると、REDOデータのログがとられていないので、「エクステン트가無効」レコードは特定のブロック範囲を論理的に壊れ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。メディア回復を可能にするには、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をバックアップしなければなりません。

注意：ロギングありモード/ロギングなしモードは、表またはパーティション、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が挿入し、バックアップを取った後は、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取られるように設定できます。

表 21-2 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 21-2 LOGGING オプションと NOLOGGING オプションのまとめ

挿入の種類	LOGGING	NOLOGGING
ダイレクト・ロード・インサート	可能。回復には次のものが 必要。 <ul style="list-style-type: none"> ARCHIVELOG データベース・モード 	可能。次のものが 必要。 <ul style="list-style-type: none"> 表領域または表、パーティション、索引の NOLOGGING 属性
従来型 INSERT	可能（デフォルト）。回復には次のものが 必要。 <ul style="list-style-type: none"> ARCHIVELOG データベース・モード 	不可能

ロギングなしモードの例

行の挿入先の表の NOLOGGING 属性を設定することにより、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。たとえば、

```
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、索引にも NOLOGGING 属性を設定できます。たとえば、

```
ALTER TABLE emp MODIFY PARTITION emp_lmnop NOLOGGING;

ALTER TABLESPACE personnel NOLOGGING;

ALTER INDEX emp_ix NOLOGGING;

ALTER INDEX emp_ix MODIFY PARTITION eix_lmnop NOLOGGING;
```


ロギングなしモードを使える SQL 文

表またはパーティション、索引、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えるのは、次の操作だけです。

- 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

これらの SQL 文はすべてパラレル化できます (第 22 章「パラレル実行」を参照)。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 (UPDATE および DELETE、従来型パス INSERT、上記に含まれない種々の DDL 文など) は、スキーマ・オブジェクトの NOLOGGING 属性による影響を受けません。

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、表またはパーティション、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。

- CACHE が指定されている場合は、LOGGING が使われる (LOB は CACHE NOLOGGING を持てないため)。
- 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る。

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

ここでは、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、そしてデータのロックを説明します。

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、インサート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・サーバー・プロセスによって行われ、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、後で再作成することにより、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してしまうため、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、表セグメントの最後の位置より前にある空きブロックも無視されます。ダイレクト・ロード・インサートを使う場合は、さらにいくつかの領域要件を考慮しておかなければなりません。

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・インサートを実行すると、一時セグメント（並行度ごとに1つのセグメント）が作られます。たとえば、並行度を4に設定して非パーティション表に対してパラレル INSERT を使うと、一時セグメントが4つ作られます。

各パラレル・サーバー・プロセスは、まずデータを一時セグメントに挿入してから、最後にすべての一時セグメントに入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。）

必要以上に大きなセグメントを使って領域を無駄にしまうことなく、一時セグメント用に十分な記憶領域を確保するために、パラレル INSERT を実行する非パーティション表の記憶領域パラメータ NEXT と PCTINCREASE には適切な値を指定する必要があります。それらのパラメータ値を変更するには、ALTER TABLE文でSTORAGEオプションを指定します。パラレル DML 文を実行した後、NEXT および PCTINCREASE 記憶領域パラメータを、非パラレル操作に適した設定に戻せます。

注意: PCTINCREASE 記憶領域パラメータが0に設定されていない場合、非常に大きな一時セグメントが作られることがあります。パラレル DML の実行中に領域を使い果たしてしまわないように、PCTINCREASE は必ず0に設定してください。

しかし、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られません。各パラレル・サーバー・プロセスは、データをパーティションの最後の位置より後に挿入するだけです。

追加情報：領域管理の詳細は、『Oracle8 Server チューニング』のパラレル実行についての章を参照してください。

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または更新、削除を同時に実行できないようにします。しかし、問合せを同時に実行することはサポートされているので、インサートを開始する前の表中のデータだけは見えます。またそ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・インサートを使う前に考慮しておく必要があります。詳細は、22-31 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」を参照してください。

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を使ったダイレクト・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からです。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るが、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 文で同じ表をアクセスすることはできない。
 - 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るが、後に行うことはできない。
 - 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）によって変更済み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る。
- 初期化パラメータが ROW_LOCKING = INTENT となっていれば、ダイレクト・ロード・パスによって挿入を実行できない。
- 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ない。
- 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ない。
- 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ない。

- 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ない。
- ダイレクト・ロード・インサート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したりはできない。
- クラスタ化された表はサポートされない。

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っ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）を使おうとすると、PARALLELヒントまたは句（存在する場合）だけではなく APPEND ヒントも無視されます。

パラレルDML（パラレルINSERTを含む）の一般的な制限事項の詳細は、22-33ページの「パラレルDMLの制限事項」を参照してください。

パラレル実行

Civilization advances by extending the number of important operations which we can perform without thinking about them.

Alfred North Whitehead: *An Introduction to Mathematics*

この章では、SQL文のパラレル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。

- パラレル実行の概要
- パラレル実行のプロセス・アーキテクチャ
- 並行度の設定
- パラレル DDL
- パラレル DML
- 親和性

注意: この章で説明するパラレル実行機能は、Oracle8 Enterprise Editionを購入した場合にのみ利用できます。Oracle8 Enterprise Editionの詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

注意: パラレル実行は、Oracle Parallel Server (「パラレル・サーバー・オプション」) と同じではありません。パラレル・サーバー・オプションは、SQL文をパラレル実行する際には必要ありません。しかし、パラレル実行のいくつかの面は、Oracle Parallel Server にしか適用されません。

パラレル実行の概要

Oracle が SQL 文をパラレル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、パラレル実行では、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 はその文をもっと速く実行できます。

パラレル実行により、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。対称型マルチプロセッシング（SMP）、クラスタ化されたシステム、大規模パラレル処理システム（MPP）においては、パラレル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からです。

パラレル実行によって、システムはハードウェア・リソースの使用が最適化され、パフォーマンスの点で一段と向上します。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、パラレル実行を使う前にシステムの負荷を軽減するか、またはそれらのハードウェア・リソースを増やす必要があります。

追加情報：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、パラレル実行を活用できるようにすることの詳細は、『Oracle8 Server チューニング』を参照してください。

パラレル化できる操作

Oracle Server では、次の操作にパラレル実行を利用できます。

- 表の走査
- ネストされたループ結合
- ソート・マージ結合
- ハッシュ結合
- "NOT IN"
- GROUP BY
- SELECT DISTINCT
- UNION と UNION ALL
- 集約操作
- 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

- 索引の再構築
- 索引パーティションの再構築
- パーティションの移動
- パーティションの分割
- UPDATE
- DELETE
- INSERT ... SELECT
- 制約を使用可能にする（表の走査がパラレル化される）
- スター型変換

Oracle が操作をパラレル化する方法

SELECT 文は、問合せだけで構成されています。多くの場合、DML 文または DDL 文は、問合せ部分と DML または DDL 部分で構成されています。Oracle では、前の章に示されている SQL 文の問合せ部分、および DML または DDL 部分の両方をパラレル化できます。

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」は挿入および更新、削除だけを指します。

Oracle は基本的に次の方法で SQL 文をパラレル化します。

1. 走査操作（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パラレル化する。
2. 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ため、パーティションによってパラレル化する。
3. 非パーティション表だけに挿入するため、パラレル・サーバー・プロセスによってパラレル化する。

ブロック範囲によるパラレル化

Oracle は、実行時に問合せを動的にパラレル化します。「動的並行性」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」）に分け、異なる範囲に対して操作をパラレルに実行します。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パラレル化が自動的に最適化されます。

ブロック範囲によるパラレル走査では、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えます。

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1つのパーティションをまたがることは不可能です。OracleはROWID範囲を使っ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・マップ参照を避けます。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走査されないようにします（「パーティションの切詰め」）。

したがって、表走査によってパーティション表にアクセスするパラレル問合せの作業量は、全体として、非パーティション表に対する問合せの作業量以下になります。ディスクのアクセス回数の合計はパーティション切詰め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のパラレル化を達成して実行されます。

表および索引に対する操作のうち、ブロック範囲（ROWID範囲）によってパラレル化できるものは、次のとおりです。

- 表走査を使った問合せ（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構築
- 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パラレル化

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パラレル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。パラレル化の最小単位はパーティションです。パーティション内をさらにパラレル化することはありません。

パーティション表と索引に対する操作は、表または索引の異なるパーティションに異なるパラレル・サーバー・プロセス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、問合せ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（パーティションの切詰め）。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、この操作は順次実行されます。

パラレル操作では、アクセスされるパーティションの数より少ないパラレル・サーバー・プロセスが使われることもあります（リソース制限またはヒント、表属性による）。各パーティションは1つのパラレル・サーバー・プロセスによってアクセスされます。しかし、1つのパラレル・サーバー・プロセスから、複数のパーティションにアクセスできます。

パーティション表および索引に対する操作は、2つ以上のパーティションにアクセスする場合、そしてアクセスされる表または索引ページの最小数として事前に決定してあった数より表または索引の選択肢が多い場合に限り、パラレルで走査されます。

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範囲走査を使う問合せ

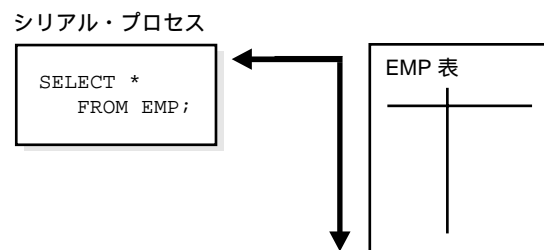
パラレル・サーバー・プロセスによるパラレル化

非パーティション表だけの場合、Oracle は複数のパラレル・サーバー・プロセス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がないので、行は複数のパラレル・サーバー・プロセスの間で分配されて、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

パラレル実行を使わない場合は、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、全表走査 (SELECT * FROM EMP など) を実行するには、図 22-1 に示すようにして 1 つのプロセスで操作全体を実行します。

図 22-1 順次全表走査



パラレル実行では、複数の「パラレル・プロセス」を使って操作がパラレルで実行されます。「パラレル・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・サーバー・プロセス」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。

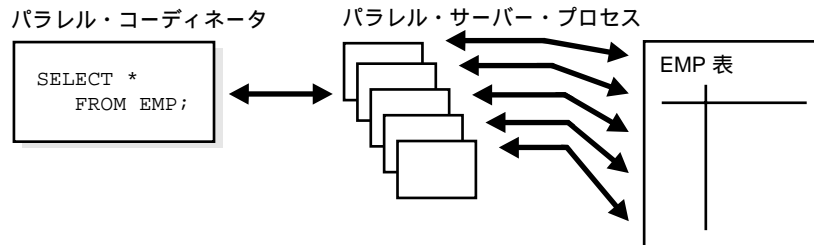
注意：ここでの「パラレル・サーバー・プロセス」という用語は、Oracle Parallel Server のプロセスを指すのではなく、操作をパラレルで実行するプロセスを指します。（しかし、Oracle Parallel Server では、パラレル・サーバー・プロセスが複数インスタンスにわたる場合があります。）パラレル・サーバー・プロセスは「スレーブ・プロセス」とも呼ばれます。

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けられる場合、Oracle はその操作に使う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・サーバー・プロセスに割り当てます。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・サーバー・プロセスに作業を割り当てるのに使われる親和性に影響を与えます。

詳細は、22-36 ページの「親和性」を参照してください。

図 22-2 に、EMP 表がブロック範囲によって動的に分けられ（「動的パーティション化」）、その表の部分的な走査を複数のパラレル・サーバー・プロセスが同時に実行する様子を示します。パラレル・サーバー・プロセスは結果をパラレル・コーディネータ・プロセスに送り返し、それぞれの結果が組み立てられて、最終的な全表走査になります。

図 22-2 パラレル全表走査



パラレル・コーディネータは実行する機能をパラレルな断片に分け、その後、パラレル・サーバー・プロセス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・サーバー・プロセスの数が、その操作の「並行度」となります。同じ SQL 文内の複数の操作の並行度は、すべて同じです（22-12 ページの「操作の並行度の決定」を参照）。

パラレル・サーバー・プール

インスタンスを起動すると、パラレル操作に使えるパラレル・サーバー・プロセスのプールが作られます。初期化パラメータ PARALLEL_MIN_SERVERS は、インスタンスの起動時に作られるパラレル・サーバー・プロセスの数を指定します。

パラレル操作の実行中、パラレル・コーディネータはプールからパラレル・サーバー・プロセス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・サーバー・プロセスを作れます。それらのパラレル・サーバー・プロセスは、ジョブの実行中はずっと 1 つの操作に対応付けられ、その後、他の操作に使えるようになります。文が完全に処理された後、パラレル・サーバー・プロセスはプールに戻ります。

注意：パラレル・コーディネータとパラレル・サーバー・プロセスは、一度に 1 つの文にしか作用しません。パラレル・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。

パラレルな複数の文を同時に実行する必要があるなら、PARALLEL_MIN_SERVERS の値を大きく設定できます。

ユーザーが SQL 文を発行すると、オブ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、各操作の並行度を決定します。操作に必要なパラレル・サーバー・プロセスの数は、さまざまな方法で指定できます（22-12 ページの「並行度の設定」を参照）。

オブ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。

- SQL 文のフォアグラウンド・プロセスがパラレル・コーディネータになります。
- パラレル・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・サーバー・プロセス（並行度によって判断される）を取得します。
- 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。
- 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戻し、さらにパラレル・サーバー・プロセスをサーバー・プールに戻します。

パラレル・コーディネータは、SQL 文の実行時（文の解析時ではない）にパラレル・サーバー・プロセスをコールします。それで、マルチスレッド・サーバーでパラレル実行を使う場合には、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コーディネータ・プロセスになります。

パラレル・サーバー・プロセスの数の変動

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・サーバー・プロセス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・サーバー・プロセスが作られます。しかし、Oracle は、初期化パラメータ PARALLEL_MAX_SERVERS が指定するより多くのパラレル・サーバー・プロセスをインスタンス用に作ることはありません。

パラレル操作の数が減少した場合は、初期化パラメータ PARALLEL_SERVER_IDLE_TIME によって指定された期間だけアイドル状態が続いているパラレル・サーバー・プロセスが Oracle によって終了されます。問合せサーバー・プロセスがアイドル状態となっていた時間には関係なく、Oracle がプール・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・サーバー・プロセスがない場合の処理

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。初期化パラメータ PARALLEL_MIN_PERCENT で最小値を指定することの詳細は、22-14 ページの「パラレル・サーバー・プロセスの最小数」を参照してください。

プール中のすべてのパラレル・サーバー・プロセスが占有されており、最大数のパラレル・サーバー・プロセスがすでに開始されている場合、パラレル・コーディネータはシリアル処理に切り替わります。

追加情報：パラレル・サーバー・プロセス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、『Oracle8 Server チューニング』を参照してください。

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、もっと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。（最適化の詳細は、第 20 章「オブティマイザ」を参照。）

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・コーディネータは、実行計画の中の各操作のパラレル化の方法を決定します（たとえば、ブロック範囲により全表走査をパラレル化したり、パーティションによって索引範囲走査をパラレル化したりします）。コーディネータは、操作がパラレルで実行できるかどうか、またパラレルの場合にはパラレル・サーバー・プロセスの数（つまり「並行度」）を決定する必要があります。

詳細は、22-12 ページの「並行度の設定」および 22-15 ページの「SQL 文のパラレル化ルール」を参照してください。

複数のパラレル・サーバー・プロセスの間で作業を分ける

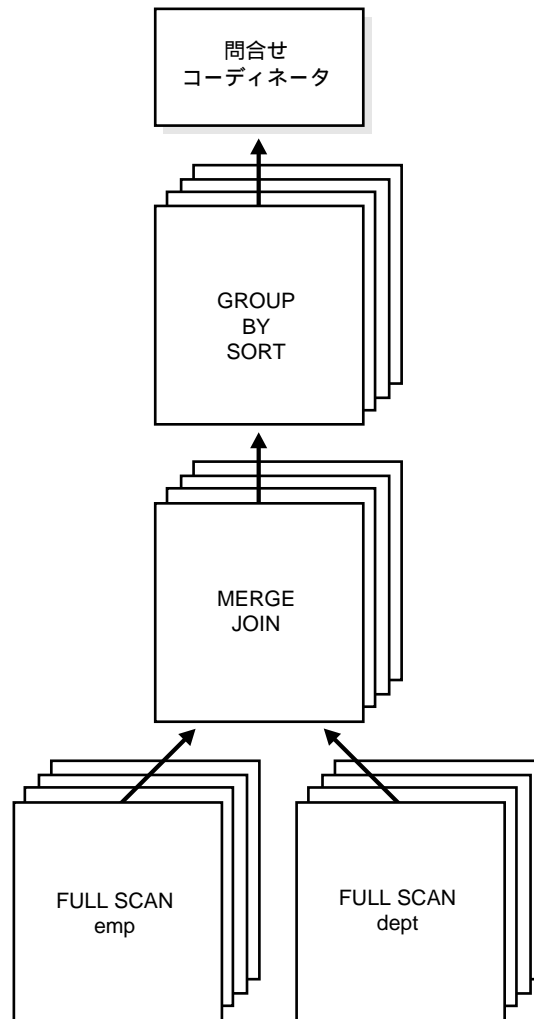
パラレル・コーディネータは、各操作の再分散化要件を調べます。操作の「再分散化要件」とは、操作によって処理する行を複数のパラレル・サーバー・プロセスの間で分けるための方法です。

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 22-3 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 22-3 EMP 表と DEPT 表の結合のデータ・フロー・ダイアグラム



操作間の並行性

他の操作の出力を必要とする操作のことを、「親操作」といいます。図 22-3 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・サーバー・プロセスが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・サーバー・プロセスのセットがMERGE JOIN 操作の実行を開始してその行を使います。

同時に実行される上記の2つの操作には、それぞれ専用のパラレル・サーバー・プロセスのセットが用意されます。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並行性が指定されていることになります。個々の操作の並行性のことを「イントラオペレーション（操作内）並行性」といい、データ・フロー・ツリーの各操作間での並行性のことを「インターオペレーション（操作間）並行性」といいます。

Oracle Server の操作には作成者/使用者という特性があるので、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のは2つの操作だけです。

イントラオペレーション並行性とインターオペレータ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;
```

実行計画は、EMP 表の全走査と、その後に実行される、取得した行をENAME 列の値に基づいてソートする操作とで構成されています。この例では、ENAME 列には索引が作られていないものとします。また、この問合せの並行度は4に設定されているとします。つまり、それぞれの1つの操作について4つのパラレル・サーバー・プロセスをアクティブにできることになります。

図 22-4 に、この例の問合せのパラレル実行を示します。

図 22-4 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

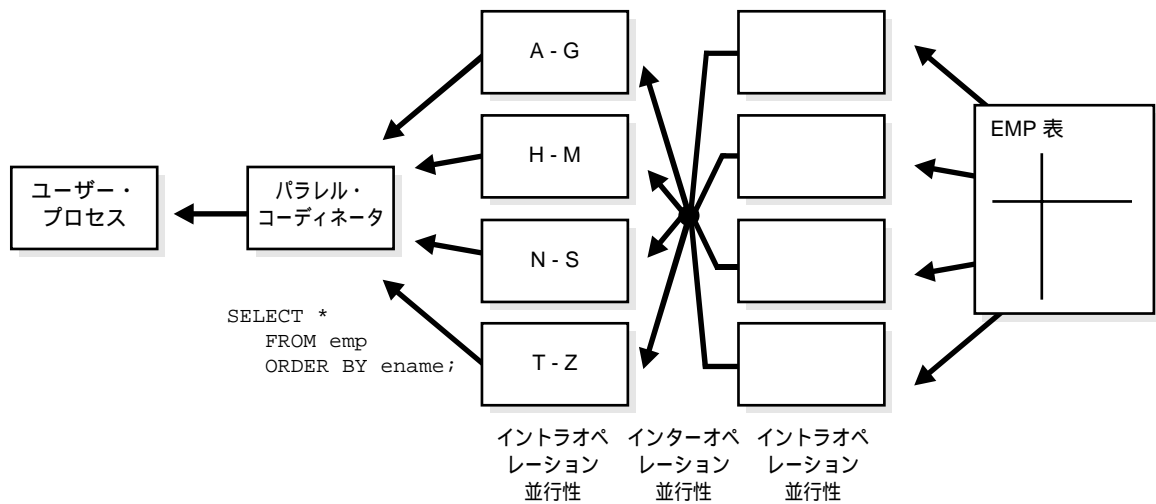


図 22-4 からわかるとおり、並行度は 4 ですが、問合せに実際に関係しているのは 8 つのパラレル・サーバー・プロセスです。これは、親オペレータと子オペレータを同時に実行できるためです（インターオペレーション並行性）。

走査操作に含まれるすべてのパラレル・サーバー・プロセスが、ソート操作を実行するパラレル・サーバー・プロセス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・サーバー・プロセスによって走査された行の ENAME 列の内容が A ~ G の間の値であるなら、その行は最初の ORDER BY パラレル・サーバー・プロセスへ送られます。走査操作が完了すると、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：一連のパラレル・サーバー・プロセスが完了すると、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 操作が続いているとしたなら、表走査を実行しているパラレル・サーバー・プロセスが、表走査完了後に 2 番目の ORDER BY 操作を実行します。

並行度の設定

パラレル・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パラレル・サーバー・プロセスを2つ以上登録することがあります。1つの操作に対応するパラレル・サーバー・プロセスの数を「並行度」といいます。

並行度は、問合せレベルで（ヒントまたはPARALLEL句を使って）、表または索引レベルで（表または索引の定義を使って）、またはディスクかCPUの数に基づくデフォルトによって指定します。

この並行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だけです。インターオペレーション並行性が可能であれば、パラレル・サーバー・プロセスの合計数は、指定した並行度の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。

操作の並行度の決定

パラレル・コーディネータは、いくつかの仕様を考慮して並行度を決定します。コーディネータは、最初にSQL文自体で指定されているヒントまたはPARALLEL句を調べ、次に表または索引の定義を調べ、最後にデフォルトの並行度を調べます（22-13ページの「デフォルトの並行度」を参照）。これらの指定のうちのどれかで並行度が決まると、その並行度がその操作の並行度になります。

並行度の詳細は、22-15ページの「SQL文のパラレル化ルール」を参照してください。

ヒントおよびPARALLEL句、表または索引定義、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・サーバー・プロセス数を決定するだけです。実際に使われるパラレル・サーバー・プロセスの数は、パラレル・サーバー・プール中使用可能なプロセス数（22-6ページの「パラレル・サーバー・プール」を参照）、およびインターオペレーション並行性が可能かどうか（22-9ページの「操作間の並行性」を参照）によって異なります。

ヒント

SQL文でヒントを指定することによって、表または索引に対して並行度を設定したり、操作のキャッシュ動作を設定したりできます。

- PARALLEL ヒントは、表に対する操作だけに関して使われます。このヒントを使うことによって、問合せおよびDML文（INSERT および UPDATE、DELETE）をパラレル化できます。
- PARALLEL_INDEX ヒントは、パーティション索引の索引範囲の走査をパラレル化します。（索引操作では、PARALLEL ヒントは無効であり、無視されます。）

追加情報：SQL文でのヒントの使用法の概要、またPARALLEL、NOPARALLEL、PARALLEL_INDEX、CACHE、NOCACHEの各ヒントの明確な構文は、『Oracle8 Server 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

並行度は、表定義または索引定義の中にも指定できます。表または索引の並行度を設定するには、SQL文CREATE TABLEまたはALTER TABLE、CREATE INDEX、ALTER INDEXのうちの1つを使います。

追加情報: SQL文の完全な構文は、『Oracle8 Server SQLリファレンス』を参照してください。

デフォルトの並行度

ヒントの中にも、また表または索引の定義にも並行度を指定しない場合、デフォルトの並行度が使われます。ほとんどのアプリケーションには、デフォルトの並行度が適しています。

追加情報: 並行度の調整の詳細は、『Oracle8 Server チューニング』を参照してください。

SQL 文のデフォルトの並行度は、次の要因によって決められます。

1. システム中の CPU 数。
2. Oracle Parallel Server のインスタンスの数。
3. 表または索引が格納されているディスク（または親和性情報が使用不可能な場合はファイル）の数。
4. パーティションによるパラレル化の場合、パーティションの切詰め（隣接しているかどうか）に基づく、アクセスされるパーティションの数。
5. 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の中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・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。

たとえば、システムに 20 個の CPU があり、15 個のディスク・ドライブに格納されている表に対してパラレル問合せを発行する場合、問合せのデフォルトの並行度は 15 の問合せサーバーです。

注意: Oracle は、ディスクおよび CPU についての情報をオペレーティング・システムから取得します。

上記の要素により、使われるパラレル・サーバー・プロセスのデフォルト数が決定されますが、実際に使われるパラレル・サーバー・プロセスの数は、要求したインスタンスが実行時にどの程度利用可能かによって制限されます。1 つのインスタンスに対して可能なパラレル・サーバー・プロセスの合計数の上限は、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・サーバー・プロセスの最小数（初期化パラメータ PARALLEL_MIN_PERCENTで指定されるもの）が使えない場合、ユーザー・エラーが発生します。その場合、ユーザーは並行性を低くして問合せを再試行できます。

注意: PARALLEL_DEFAULT_SCANSIZE 初期化パラメータと PARALLEL_DEFAULT_MAX_SCANS 初期化パラメータは廃止されました。

パラレル・サーバー・プロセスの最小数

最低 2 つのパラレル・サーバー・プロセスが使用可能なら、パラレル操作を実行できます。使用可能なパラレル・サーバー・プロセスが少なすぎる場合、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・サーバー・プロセスのうちある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できない、というように指定できます。これにより、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・サーバーのうちの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・サーバー・プロセス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、問合せだけでなく 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・サーバー・プロセスの少なくとも 50% が使用可能でなければなりません。20 個のパラレル・サーバー・プロセスが要求された場合、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 に設定した場合、少なくとも 2 つのパラレル・サーバー・プロセスが処理のために使える限り、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・グループを使うことによって、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構成されるインスタンス・グループを好きな数だけ作れます。その後、パラレル操作のいくつかまたはすべてに対して、どのインスタンス・グループを使うかを指定できます。パラレル・サーバー・プロセスは、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対してしか使われません。

追加情報: インスタンス・グループの詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

作業負荷のバランス調整

パフォーマンスを最適化するには、すべてのパラレル・サーバー・プロセスに同等の作業負荷がかかるようにするべきです。ブロック範囲またはパラレル・サーバー・プロセス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・サーバー・プロセスの間で動的に分けられます。これにより、パラレル・サーバー・プロセスの中に、他のプロセスよりかなり多くの作業を実行するものがある場合に発生する「作業負荷の不整」を最小化することができます。

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るなら、パラレル・サーバー・プロセスの数とパーティションの数を一致させることにより、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行度を選択することにより、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・サーバー・プロセスを使う（並行度 = 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 の時間で作業を終えられます。5 つのプロセスを使えば 5 分の 1 の時間、2 つのプロセスを使えば 2 分の 1 の時間で作業を終了できます。

しかし、9 個のプロセスを使って 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の作業を始めます。その他のプロセスは、作業を終了した後、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すれば、パフォーマンスはあまりよくなりません。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違います。

同じように、4 個のプロセスを使って 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・サーバー・プロセスを使って 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。他のプロセスが最後のパーティションの作業を終えるのを待機しなければならない状況のプロセスがあるかもしれないからです。しかし、並行度を適切に選択すれば、作業負荷の不整を最小限にでき、パフォーマンスを最適化できます。

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ことの詳細は、22-37 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 と宣言されていたりすると、SQL 文はパラレル化できます。さらに、データ定義言語（DDL）文は、PARALLEL 句を使ってパラレル化できます。しかし、これらの方法すべてがすべてのタイプの SQL 文に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と、並行度の 2 つの要素が関係しています。これらの要素は、問合せおよび DDL 操作、DML 操作のそれぞれで異なる方法で判断されます。

並行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表と索引のそれぞれを調べて、どれが参照表かを判断する。並行度が最大の表または索引を選択する、というのが基本的なルールです。

- パラレルDML(挿入および更新、削除) の場合、並行度を判断する参照オブジェクトは、挿入または更新、削除操作によって変更される表になる。パラレルDMLでは、デッドロックを防ぐため、並行度にいくらか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、その副問合せの並行度はDML 操作と同じです。
- パラレル DDL の場合、並行度を決定する参照オブジェクトは、作成または再構築、分割、移動の対象となる表または索引、パーティションである。パラレルDDL 文に副問合せが含まれる場合、その副問合せの並行度はDDL 操作と同じです。

問合せのパラレル化のルール

パラレル化するかどうかの決定

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、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。
 - 全表走査
 - 複数のパーティションにわたる索引範囲走査

並行度

問合せの並行度は、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である潜在的なすべての索引 (「参照オブジェクト」) から取得した並行度の最大値を使う。つまり、並行度が最大である表または索引によって、その問合せの並行度が決まります (「最大問合せ命令」) 。
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、ヒント指定のほうが PARALLEL 宣言仕様部より優先される。

UPDATE と DELETE のパラレル化のルール

更新操作と削除操作は、パーティションによりパラレル化されます。更新と削除のパラレル化は、パーティション表に対してしかできません。1つのパーティションの中で、あるいは非パーティション表に対して、更新 / 削除のパラレル化はできません。

UPDATE 操作と DELETE 操作にパラレル命令を指定するには、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 句を指定する (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・ヒントを文に指定する。

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走査にも適用されません。

CREATE TABLE コマンドと ALTER TABLE コマンドの PARALLEL 句によって、表の平行化が指定されます。表定義に PARALLEL 句が含まれる場合には、問合せの並行性だけでなく DML 文の並行性もそれによって決定されます。しかし、DML 文に表のための明示的な PARALLEL・ヒントが存在する場合には、その表の PARALLEL 句による指定は PARALLEL・ヒントによって上書きされます。

平行化するかどうかの決定

UPDATE/DELETE 文において更新 / 削除操作を平行化するかどうかは、次の規則によって決まります。

- 更新 / 削除をする表に PARALLEL が指定されているか、DML 文中に PARALLEL ヒントが指定されている場合にだけ、更新または削除操作が平行化される。

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が、それらの平行化命令は更新または削除を平行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 / 削除部分の両方について平行化するかどうかの決定に使われます。しかし、更新 / 削除部分を平行化するかどうかは、問合せ部分には依存せずに決定され、その逆の依存関係もありません。

並行度

並行度は、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、修正の対象となる 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 / 削除操作の並行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行度は、表のパーティション数と同じです。1 つの PARALLEL・サーバー・プロセスから複数のパーティションのデータを更新したり削除したりできますが、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・サーバー・プロセスからだけです。

並行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。操作に関係するパーティションの数より並行度のほうが大きい場合は、超過分の PARALLEL・サーバー・プロセス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、表の走査が順次走査である（たとえば索引走査）としても、更新操作は平行化されます（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 に対する走査操作と更新操作が、並行度 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（並行度は除く）。

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違うため、ヒントでは、特に挿入操作のためのパラレル命令を指定できます。

INSERT... SELECT 文にパラレル命令を指定するには、次の 4 つの方法があります（PARALLEL DML モードが使用可能であるとします）。

1. SELECT パラレル・ヒント（複数可）をその文に指定する。
2. 選択対象の表の定義に PARALLEL 句を指定する。
3. INSERT パラレル・ヒントをその文に指定する。
4. 挿入先の表の定義に PARALLEL 句を指定する。

パラレル化するかどうかの決定

INSERT... SELECT 文において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。

- 挿入先の表（参照オブジェクト）に PARALLEL 宣言が指定されているか、DML 文中の INSERT の後に PARALLEL ヒントが指定されている場合にだけ、挿入操作がパラレル化される。

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、その逆の依存関係もありません。

並行度

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ったら、次の優先順位ルールを使って、文全体の「並行度」を決定するためのパラレル命令が 1 つ選ばれます。

挿入ヒント命令 > 挿入先の表のパラレル宣言仕様部 > 最大問合せ命令

ここで、「最大問合せ命令」とは、複数の表と索引の中で並行度が最大の表または索引が、問合せ操作の並行性を決定することを意味します。

選択したパラレル命令は、選択操作と挿入操作の両方に適用されます。

例：

次の例で使われている並行度は 2 です。これは、INSERT のヒントに指定されている並行度です。

```
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

平行化するかどうかの決定

DDL 操作は、構文の中に 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。

並行度

並行度は、PARALLEL 句の中の仕様部により決定されます。パーティション索引の再構築は決して平行化されません。

索引の作成、索引の再構築、パーティションのマージ/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 句によってしか平行化できません。

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 の走査操作の並行性は、REBUILD または CREATE 操作と同じであり、同じ並行度です。REBUILD または CREATE に並行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 句によってしか平行化できません。これらの文の走査操作の並行性は、対応する MOVE/SPLIT 操作と同じです。並行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、2 つの部分が含まれています。

- CREATE 部 (DDL)
- SELECT 部 (問合せ)

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。

平行化するかどうかの決定 (問合せ部)

CREATE TABLE ... AS SELECT 文の問合せ部は、次の条件が満たされている場合に限り平行化できます。

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。

- 全表走査
- 複数のパーティションにわたる索引範囲走査

並行度（問合せ部）

CREATE TABLE ... AS SELECT文の問合せ部の並行度は、次のルールの1つによって決定されます。

1. 問合せ部では、CREATE 部の PARALLEL 句で指定された値を使う。
2. PARALLEL 句が指定されていない場合のデフォルトは CPU 数。

並行性のヒントに指定する値は、すべて無視されます。

パラレル化するかどうかの決定（CREATE 部）

CREATE TABLE ... AS SELECTのCREATE操作は、PARALLEL句によってしかパラレル化できません。

CREATE TABLE ... AS SELECTのCREATE操作がパラレル化されると、可能なら走査操作もパラレル化されます。走査操作は、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走査する場合

CREATE 操作がパラレル化されない場合は、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と、SELECT 操作をパラレル化できます。

並行度（CREATE 部）

CREATE 操作の並行度、および SELECT 操作の並行度（パラレル化される場合）は、CREATE 文の PARALLEL 句によって指定します。CREATE 文に並行度を指定しない場合のデフォルト値は CPU 数です。SELECT 句のヒントに並行度を指定しても、それは無視されます。

パラレル化ルールのまとめ

表 22-1 に、さまざまなタイプの SQL 文をパラレル化する方法を示します。また、並行性を指定するときにどの方法が優先されるかも示します。

- 優先順位（1）の指定は、優先順位（2）と優先順位（3）の指定に優先する。
- 優先順位（2）の指定は、優先順位（3）の指定に優先する。

追加情報：SQL 文の PARALLEL 句とパラレル・ヒントの詳細は、
『Oracle8 Server SQL リファレンス』を参照してください。

表 22-1 パラレル化ルール

パラレル操作	句、ヒント、または基礎を形成する表 / 索引宣言によるパラレル化（優先順位： 1、2、3）		
	PARALLEL 句	パラレル・ヒント	パラレル宣言
パラレル問合せ表走査（パーティション表または非パーティション表）		(1) PARALLEL	(2) 表の宣言
パラレル問合せ索引範囲走査（パーティション索引）		(1) PARALLEL_INDEX	(2) 索引の宣言
パラレル UPDATE/DELETE（パーティション表だけ）		(1) PARALLEL	(2) 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) 挿入先の表の宣言
パラレル INSERT... SELECT の SELECT 操作（パーティション表または非パーティション表）		(1) SELECT 部の PARALLEL	(2) 選択する表の宣言
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）	(1)	（注意：SELECT 句のヒントは CREATE 操作には影響しない。）	
パラレル CREATE TABLE... AS SELECT の SELECT 操作（パーティション表または非パーティション表）	(2)	(1) PARALLEL/PARALLEL_INDEX	(3) 問い合わせる表 / 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）	(1)		
パラレル REBUILD INDEX（非パーティション索引）	(1)		
REBUILD INDEX（パーティション索引）		決してパラレル化されない	
パラレル REBUILD INDEX パーティション	(1)		
パーティションのパラレル MOVE/SPLIT	(1)		

パラレル DDL

この項では、データ定義言語（DDL）文の並行性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

- 回復可能性とパラレル DDL
- パラレル DDL の領域管理

パラレル化できる DDL 文

非パーティション表・索引またはパーティション表・索引の DDL 文は、パラレル化できます。DDL 文でパラレル化できる操作は、22-21 ページの表 22-1 にまとめられています。

非パーティション表・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・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 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 Usable である場合だけ

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（21-5 ページの「ロギング・モード」を参照）で実行できます。

操作ごとに異なる並行性が使われます（22-21 ページの表 22-1 を参照）。パーティション表のパラレル CREATE TABLE AS SELECT、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行度で実行されます。

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・セッションを使って構築できるので、ANALYZE {TABLE, INDEX} PARTITION コマンドがあるため、パーティションのパラレル分析表を作る必要はほとんどありません。

追加情報: パラレル DDL 文の構文と使用方法の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

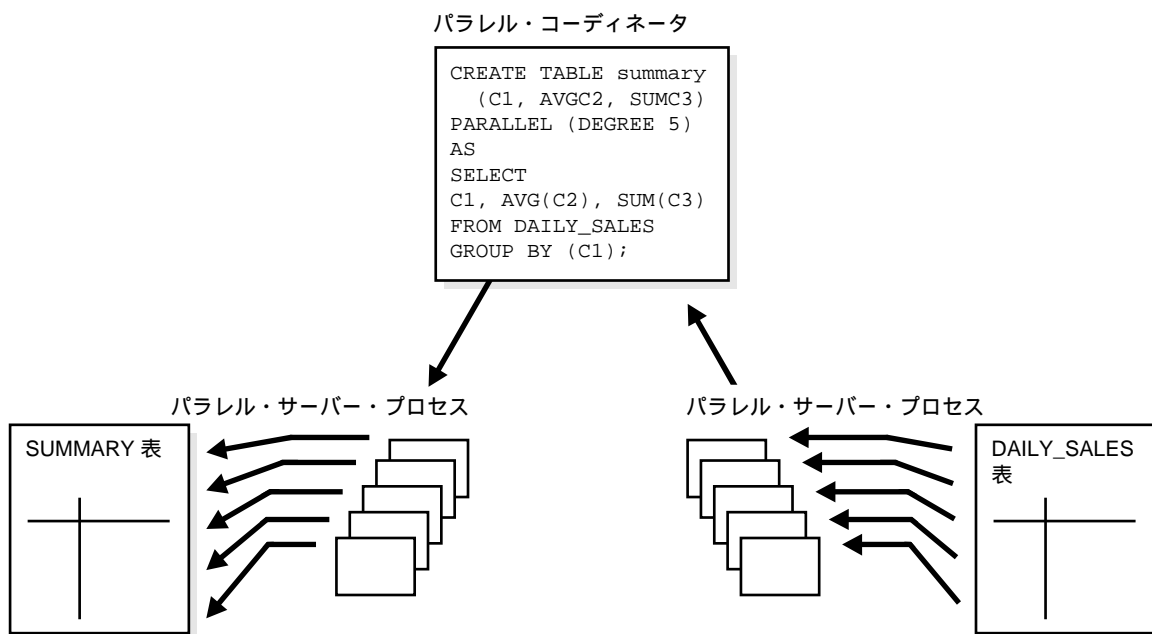
パラレルの CREATE TABLE ... AS SELECT

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うために、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しなければならないことがよくあります。そのような作業は、定期的に（毎晩または毎週など）システムが非アクティブである短い期間に実行されます。

パラレル実行を使うと、別の表または表の集合からの副問合せとしての表を作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 22-5 に、副問合せからパラレルで表を作る様子を示します。

図 22-5 パラレルでサマリー表を作る



注意：クラスタ化表は、パラレルで作ったり挿入したりできません。

回復可能性とパラレル 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なので、メディア障害からの回復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回復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）の間のロ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、表が作られた後で、その表を含む表領域のバックアップを取っておいてください。

取消しログと REDO ログの生成をオフにするには、CREATE/ALTER TABLE/INDEX 文の NOLOGGING 句を使います。詳細は、21-5 ページの「ロギング・モード」を参照してください。

追加情報：パラレルで作られた表の回復可能性の詳細は、『Oracle8 Server 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な記憶領域と、表または索引が作られた後で使用可能な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ると、各パラレル・サーバー・プロセスは CREATE 文の STORAGE 句の値を使って、行を格納するための一時的なセグメントを作ります。それで、INITIAL として 5M を、そして PARALLEL DEGREE として 12 を指定して作った表の場合、各プロセスごとに 5MB のエクステントで開始されるため、表を作るには少なくとも 60MB の記憶領域が使われます。パラレル・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ので、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。

追加情報：CREATE TABLE コマンドの構文の説明は、『Oracle8 Server 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ると、各パラレル・サーバー・プロセスは新しいエクステントを割り当て、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、並行度 3 で索引を作ると、その索引には最初から 3 つ以上のエクステントが割り当てられます。（この説明は、索引のパラレル再構築、およびパーティションの移動または分割、再構築のパラレル化にも適用されます。）

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・オブジェクトを作るパラレル・サーバー・プロセスにあるのと同じ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ると、空き領域の「飛び地」、つまり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・サーバー・プロセスが使う一時セグメントが、行を格納するのに必要な大きさ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入れると、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振ることができますが、これは連続する領域ではないので（「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大きい場合、一時セグメントの中の行をマージして表または索引に入れるときに未使用領域を

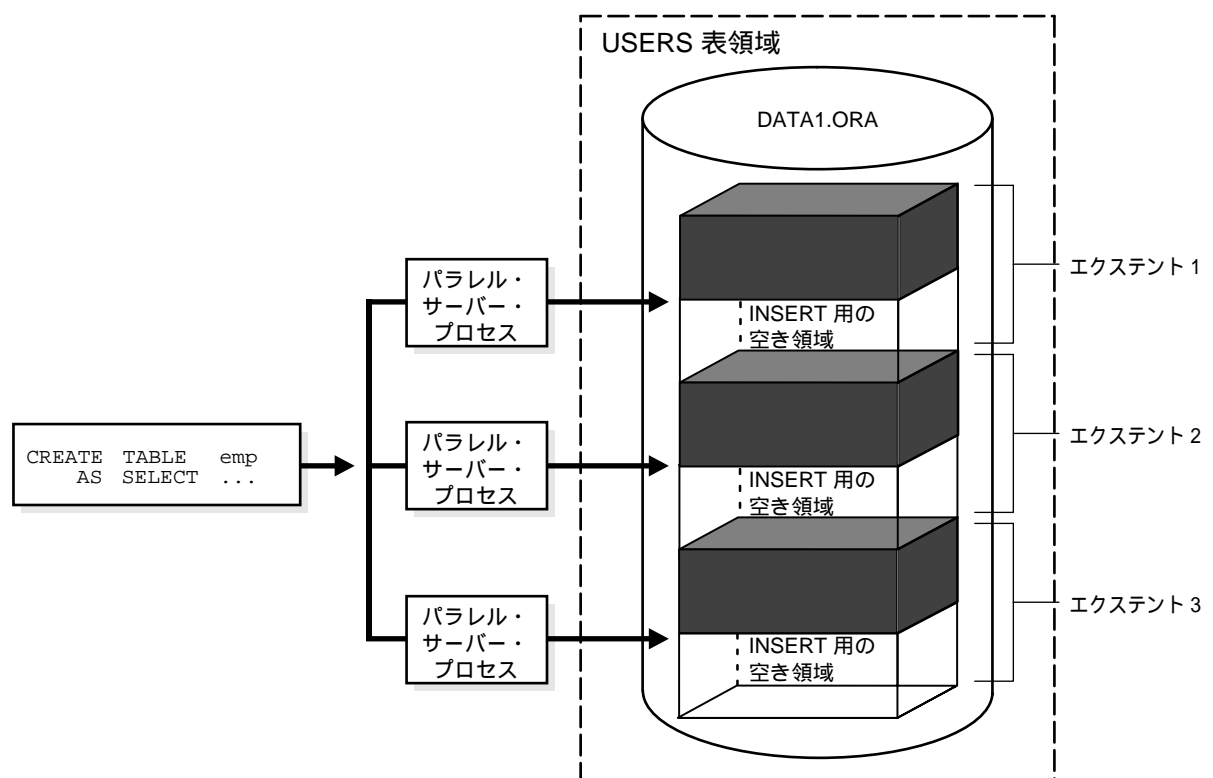
切り捨てることはできません。この未使用領域はシステムの空き領域には戻されず、表または索引の一部になり（「内部断片化」）追加の領域を必要とする後続の挿入または更新でしか使えません。

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つしかないなら、図 22-6 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。

データ・ファイルと表領域の詳細は、第3章「表領域とデータ・ファイル」を参照してください。

追加情報：パラレルでの表と索引の作成の詳細は、『Oracle8 Server チューニング』を参照してください。

図 22-6 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、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したりするために、パラレル実行メカニズムが使われます。

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」という用語は挿入および更新、削除だけを指します。

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。

- 手動のパラレル化と比べたときのパラレル DML の利点
- いつパラレル DML を使うか
- パラレル DML を有効にする
- 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 の回復
- 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

パラレル INSERT 文の詳細は、第 21 章「ダイレクト・ロード・インサート」を参照してください。

手動のパラレル化と比べたときのパラレル DML の利点

異なるデータ集合に対して複数の DML コマンドを同時に発行することにより、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 文を発行する。
- キーの範囲または ROWID の範囲が異なる、複数の UPDATE 文および DELETE 文を発行する。

しかし、手動によるパラレル化には次のような不利な点もあります。

- 使いにくい。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。
- トランザクショナルな特性がない。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、さまざまな文のコミットまたはロールバックを手動により調整しなければなりません (おそらくインスタンス間で)。

- 作業分割の複雑さ。作業を正確に分けるには、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。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、現在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、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。

いつパラレル DML を使うか

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われます。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。しかし、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、実動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)することが必要です。これは、更新可能な結合ビューと結合されるパラレル DML を使う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 を使う更新可能な結合ビューを使うことによって、更新された行をリフレッシュしたり、パラレル INSERT を使う非ハッシュ結合を使って新しい行をリフレッシュしたりできます。

追加情報：詳細は、『Oracle8 Server チューニング』を参照してください。

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しなければならず、この計算にはたくさん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 を使うと、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。1つの利点は、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なければならないので、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、マーケティング・リストへの追加など）に使われます。

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わせて更新します。パラレル DML を使うと、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、そしてパラレルの DELETE 操作により、この入れ替え作業の速度が向上します。

外部から大量のデータを挿入するのにパラレル・ダイレクト・ローダー（SQL*Loader）を使うこともできますが、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 のほうが速くなります。

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なければなりません。

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・ジョブには、ジョブを完了しなければならない一定の時間帯があります。ジョブを時間内に確実に終わらせるためのよい方法は、その操作をパラレル化することです。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 を有効にする

DML 文をパラレル化できるのは、ALTER SESSION に ENABLE PARALLEL DML オプションを指定することによってセッションでパラレル DML を明示的に有効にした場合だけです。パラレル DML とシリアル DML ではロック要件およびトランザクション要件、ディスク領域要件が異なるので、このモードが必要になります。（22-31 ページの「パラレル DML の領域に関する考慮事項」および 22-31 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」を参照してください。）

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 句を使っても、DML はパラレル実行されません。

セッション内で PARALLEL DML を有効にすると、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。しかし、PARALLEL DML を有効にしても、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（22-33 ページの「パラレル DML の制限事項」を参照）。

セッションを PARALLEL DML モードにしても、SELECT 文および DDL 文、DML 文の問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 文中の走査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。

PARALLEL DML を有効にしたトランザクション

PARALLEL DML を有効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問合せや DML 文がそのトランザクションで同じ表にもう一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 を有効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、コーディネータ・プロセスはパラレル・サーバー・プロセスを獲得または生成し、各パラレル・サーバー・プロセス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・サーバー・プロセスごとに、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る。
- 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する（次の部分を参照）。

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の速度を上げるには、十分なロールバック・セグメントを作ってそれをオンラインにすることによって、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の中にロールバック・セグメントを作り、それらのロールバック・セグメントの MAXEXTENTS 記憶領域パラメータを UNLIMITED に設定してください。

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・プロトコルを使うことにより、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回復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 の回復

パラレル DML 操作をロールバックする時間は、先送り操作を実行する時間とほぼ同じです。

Oracle では、トランザクションおよびプロセスの障害が発生した場合のパラレル・トランザクション回復（「取消し」回復）がサポートされています。インスタンスおよびシステムの障害の場合、そこまではサポートされていません。

トランザクション回復の速度を上げるには、初期化パラメータ `CLEANUP_ROLLBACK_ENTRIES` が、先送り操作のために生成されるロールバック・エントリの数にほぼ等しい高い値に設定されていることが必要です。

トランザクション回復

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・コーディネータおよびパラレル・サーバー・プロセス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、先送りトランザクションを処理した場合とほとんど同じです。

プロセス回復

パラレル DML コーディネータまたはパラレル・サーバー・プロセスの障害からの回復は、PMON プロセスによって実行されます。

- 1 つのパラレル・サーバー・プロセス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・サーバー・プロセスはそれぞれ自分の作業をロールバックする。
- 複数のパラレル・サーバー・プロセス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する。
- コーディネータ・プロセスで障害が発生すると、PMON はコーディネータを回復し、すべてのパラレル・サーバー・プロセスはそれぞれ自分の作業をパラレルでロールバックする。

したがって、プロセス障害からの回復にかかる時間は、元の（先送り）作業の実行時間以上になる場合があります。

システム回復

システム障害から回復するには、新しく起動することが必要です。回復は、SMON プロセスによって実行されます。パラレル DML 文はシリアルに回復され、回復が完了するまですべてのリソースはロックされたままになります。したがって、先送りトランザクションで高い並行度を使っており、大量の作業を実行していた場合、回復には元の（先送り）トランザクションよりもはるかに長い時間がかかることがあります。

トランザクション回復の速度を上げる 1 つの方法は、パラレル DML 文を再実行することです。新しいコーディネータとパラレル・サーバー・プロセスがロック・リソースを見つけると、それらはトランザクション回復を並行して起動します。新しいパラレル・プロセスがリソースの回復を完了したなら、そのトランザクションをコミットまたはロールバックできます。

インスタンス回復（ Oracle Parallel Server ）

Oracle Parallel Server におけるインスタンス障害からの回復は、有効な他のインスタンスの SMON 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 SMON プロセスは、障害の発生したパラレル・コーディネータまたはインスタンスのパラレル・サーバー・プロセス・トランザクション（またはその両方）を独立して回復します。障害の発生したインスタンスのパラレル・サーバー・プロセスが有効なインスタンスより多い場合、回復時間は、障害の発生したインスタンスによる先送り作業にかかる時間より長くなります。

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われます。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領域使用の特性が異なる場合があります。

ダイレクト・ロード・インサートの領域の詳細は、21-8 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っています。パラレル DML はたくさん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 または DELETE、INSERT 文のプロセスで獲得するロックは、次のとおりです。

- ・ コーディネータ・プロセスは次のものを獲得します。
 - 表ロック SX を 1 つ
 - 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対するパーティション・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なら、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・サーバー・プロセスごとに、次のものを獲得します。
 - 表ロック SX を 1 つ
 - 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・サーバー・プロセスは 1 つ以上のパーティションに対する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・サーバー・プロセスによってしか処理できません。

たとえば、600 個のパーティションがある表を並行度 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 個取得する。
- パラレル・サーバー・プロセス全体として、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 個取得する。

表 22-2 に、異なるタイプのパラレル DML 文ごとに、コーディネータおよびパラレル・サーバー・プロセスが取得するロックのタイプをまとめます。

表 22-2 パラレル DML 文が取得するロック

文のタイプ	コーディネータ・プロセスが取得するロック	各パラレル・サーバー・プロセスごとに獲得するロック
パーティション表へのパラレル UPDATE または DELETE。 WHERE 句によってパーティションが指定される	表ロック SX を 1 つ 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 パーティションごとにパーティション待機ロック X を 1 つ
パーティション表へのパラレル UPDATE または DELETE、INSERT	表ロック SX を 1 つ すべてのパーティションに対するパーティション・ロック X	表ロック SX を 1 つ パーティションごとにパーティション・ロック NULL を 1 つ パーティションごとにパーティション待機ロック X 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ない。
- パラレル更新操作の場合、グローバルな「UNIQUE 索引」はサポートされていない。その他すべての索引は、パラレル操作によって完全にメンテナンスされます。
- 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るが、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をもう一度アクセスすることはできない。
 - この制限は、シリアル・ダイレクト・ロード INSERT 文の後にも存在する。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をアクセスできない。
 - 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るが、後に行うことはできない。
 - 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る。
- 初期化パラメータが ROW_LOCKING = INTENT である場合、INSERT、UPDATE、DELETE はパラレル化されない（直列可能モードとは無関係）。
- パラレル DML 操作ではトリガーはサポートされない。
- パラレル DML ではレプリケーション機能はサポートされない。
- 特定の制約（自己参照型の整合性、削除カスケード、遅延整合性）があると、パラレル DML は実行されない。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ない。
- 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 を実行できない。
- 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したりはできない。
- クラスタ化された表はサポートされない。

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・メッセージが出されます）。たとえば、グローバル UNIQUE 索引のメンテナンスが必要な更新操作では、その更新はシリアル化されます。

この後、制限事項についてさらに詳しく説明します。

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、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だけです。これは、パーティション表についての一般的な制限です。

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 および 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施行されるものなので、パラレル DML にとって問題とはなりません。

UNIQUE および PRIMARY KEY

UNIQUE 索引では、これらの 2 つの制約が施行されます。UNIQUE キー索引または主キー索引を変更する UPDATE コマンドは、その索引がローカルな場合に限りパラレル化されます。

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 22-3 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 22-3 参照整合性の制限事項

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 カスケード	パラレル化されない	（適用されない）	パラレル化されない

削除カスケード

パラレル・サーバー・プロセス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ので、削除カスケードを使う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。その他のすべての列に対する DML では、パラレル化が可能です。

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/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。

ファンクションの制限事項

パラレル DML 文の中では、データベースやパッケージの状態を読み書きしないファンクションだけが許可されています。データベース状態やパッケージ状態の読み込みか書き込みを実行するファンクションが DML 文に埋め込まれている場合、その DML 操作はパラレル化されません。

追加情報： プラグマ RESTRICT_REFERENCES の説明は『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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:

分散トランザクション

```
select * from t1@dblink; /* 分散トランザクションを開始 */
delete /*+ parallel (t2,2) */ from t2; /* パラレル化されない */
commit;
```

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;
/* リモート・オブジェクトからのパラレル DELETE はできない */
```

例 3:

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;
/* リモート・オブジェクトを参照しているためパラレル化なし */
```

親和性

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」があるといえます。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。

注意：この項で説明している機能は、パラレル・サーバー・オプション付きのOracle8 Enterprise Editionを購入した場合にのみ利用できます。Oracle8 Enterprise Edition で使用可能な機能とオプションの詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

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、そのインスタンスにはその表領域（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）に対する親和性があるという場合があります。

Oracle では、パラレル・サーバー・プロセスに作業を割り当てる場合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われることは、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うと、データに「近い」プロセッサで走査を実行するので、ディスクからデータを走査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。この構成を使うと、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走査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のがよいでしょう。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ってアクセスするデータの合計量を切り詰めることができ、しかもROWID範囲によるパラレル表（パーティション）走査によって並行性も実現できます。デバイスがRAIDとして構成される場合は、可用性もたいへんよくなります。DSS用に使う場合であっても、索引は個々のデバイス上でパーティション化するべきです。

その他の構成（たとえば、複数のデバイスにわた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）を使っても正しい問合せ結果が得られますが、正しい並行度を選択するにはヒントを使う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 および UPDATE、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・サーバー・プロセス (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。

1. 特定の MPP アーキテクチャでは、Oracle は、パラレル・サーバー・プロセス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単位 (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います。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・ヒット率がよくなるようにし、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。
2. SMP 共有ディスク・クラスタの場合、Oracle は、ラウンドロビン・メカニズムを使っ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われます。
3. SMP およびクラスタ、MPP の各アーキテクチャでは、プロセスとデバイス間の親和性を使っ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・サーバー・プロセス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インプリメントするのにも使われます。

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る際に、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だけです。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 ping を少なくします。

追加情報: Oracle Parallel Server の詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

その他の並行性

Oracle では、パラレル SQL 実行以外に、次のタイプの操作にも並行性を使えます。

- パラレル回復
- 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)

パラレル回復とパラレル伝播は、パラレル SQL と同じように、パラレル・コーディネータ・プロセスと複数のパラレル・サーバー・プロセスによって実行されます。しかし、パラレル・ロードは、異なるメカニズムを使います。

追加情報：パラレル・ロードの詳細と SQL*Loader の概要は、『Oracle8 Server ユーティリティ』を参照してください。パラレル・ロードの使用方法についてのアドバイスは、『Oracle8 Server チューニング』も参照してください。

パラレル・コーディネータとパラレル・サーバー・プロセスの動作は、どんな種類の操作（パラレル SQL またはパラレル回復、パラレル伝播）を実行するかによって違う場合があります。たとえば、プール中のすべてのパラレル・サーバー・プロセスが使用中で、すでに最大数のパラレル・サーバー・プロセスが開始されている場合、

- パラレル SQL ロールでは、パラレル・コーディネータはシリアル処理に切り替える。
- パラレル伝播ロールでは、パラレル・コーディネータはエラーを戻す。

1 つの特定のセッションごとに、パラレル・コーディネータは 1 種類の操作だけを調整します。パラレル・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回復を同時には調整できません。

パラレル回復の概要は、28-12 ページの「回復のパラレル実行」を参照してください。

追加情報：パラレル回復の詳細は『Oracle8 Server バックアップおよびリカバリ』を参照してください。また、パラレル伝播の詳細は、『Oracle8 Server レプリケーション』を参照してください。

第 VII 部

データの保護

第 VII 部では、Oracle のデータベースの中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できることについて説明します。

第 VII 部に含まれる章は、次のとおりです。

- 第 23 章「データの同時実行性と一貫性」
- 第 24 章「データの整合性」
- 第 25 章「データベース・アクセスの制御」
- 第 26 章「権限とロール」
- 第 27 章「監査」
- 第 28 章「データベースの回復」

データの同時実行性と一貫性

A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines.

Ralph Waldo Emerson

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
- 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

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修正するということがないので、ユーザーは何も心配せずにデータベース内のデータを修正できます。しかし、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りません。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。

- データの同時実行性とは、たくさんのユーザーが同時にデータにアクセスできることです。
- データ整合性とは、各ユーザーにデータの一貫したビューが表示され、その中にそのユーザー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ています。

「データの整合性」とは、データベースと対応付けられる業務ルールを施行するもので、第24章「データの整合性」で説明されています。

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「直列可能性」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直列可能モードでは、複数のトランザクションの実行は、並行にではなく1つずつ（直列に）な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、このモードでたくさん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。並行して実行される各トランザクションが完全に分離されているというのは、あるトランザクションが問合せが実行している表に対して、他のトランザクションは挿入を実行できないという状態を指します。つまり、現実の世界では、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えます。

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並行して実行する場合に防がなければならない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	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んでしまう現象です。
反復不能読み込み（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です。

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です。

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。

分離レベル	内容を保証しない読み込み	反復不能読み込み	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
反復可能読み込み	なし	なし	あり
直列可能	なし	なし	なし

Oracleで使える分離レベルは、SQL92の一部ではない読み込み専用モードと、「コミット読み込み」および「直列可能」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だけが提供されています。コミット読み込みと直列可能の分離レベルの詳細は、23-3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・ロックを使っ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。

ロックのさまざまなタイプ（データ・ロックおよびDDLロック、内部ロック）の詳細は、23-14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

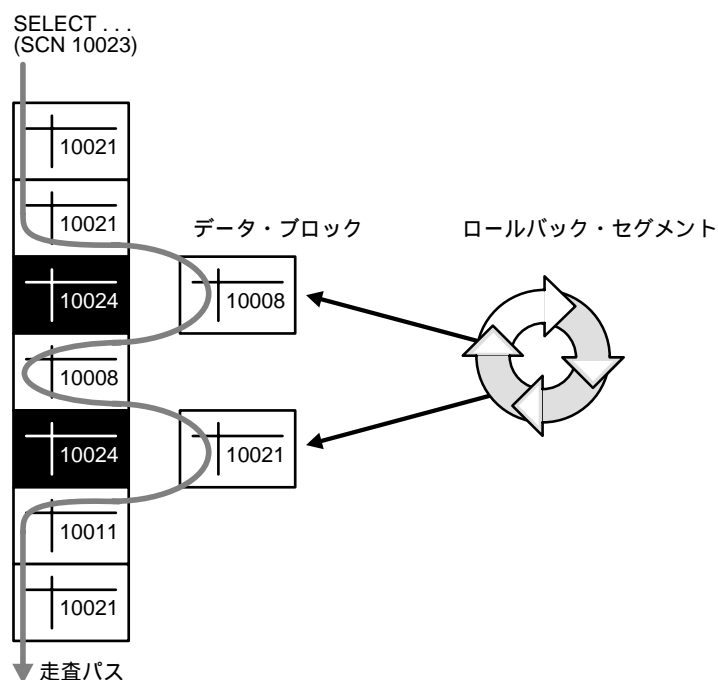
Oracleは、マルチバージョン一貫性モデルや、さまざまなタイプのロックおよびトランザクションを使うことによって、マルチユーザー環境でのデータの一貫性を維持します。

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み一貫性が提供されます（文レベルの読み一貫性）。読み一貫性は、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・レベルの読み一貫性）。

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います。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、以前の値が格納されています。図 23-1 に、Oracle がロールバック・セグメント内のデータを使って文レベルの読み一貫性を実現する方法を示します。

図 23-1 トランザクションと読み一貫性



問合せが実行段階に入った時点で現行のシステム変更番号（SCN）が決定されます。図 23-1 では、このシステム変更番号は 10023 です。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）の書き込まれたブロックだけが使われます。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み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ため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 1555 が発生します。

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。

文レベルの読み込み一貫性

Oracle では、常に文レベルの読み込み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だけです。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、DELETE はすべて、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、問合せを使って、処理（SELECT、INSERT、UPDATE、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えます。UPDATE 文と DELETE 文では、WHERE 句や副問合せを使い、表のすべての行ではなく、一部の行を処理の対象にできます。

INSERT 文および UPDATE 文、DELETE 文で使われる問合せでは、一貫した結果セットの取得が保証されます。しかし、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・レベルの読み一貫性

Oracle では、トランザクション・レベルの読み一貫性を保つこともできます。直列可能モード（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、直列可能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・レベルの読み一貫性によって反復可能読みが実現され、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 の分離レベル

Oracle には 3 つのトランザクション分離レベルがあります。

コミット読み	デフォルトのトランザクション分離レベルです。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い）が開始される前にコミットされたデータだけです。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ので、問合せを 2 回実行する間に最初の問合せ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、反復不能読みと仮読み（実体のない読み）の現象の両方が起こり得ます。

直列可能トランザクション	直列可能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みの変更と、そのトランザクション自身が INSERT 文および UPDATE 文、DELETE 文で実行した変更だけが参照されます。直列可能トランザクションでは、反復不能読みや仮読みの現象は起きません。
--------------	---

読み専用	読み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みの変更だけが参照され、INSERT 文および UPDATE 文、DELETE 文は使えません。
------	--

分離レベルの設定

アプリケーション・デザイナーおよびアプリケーションの開発者、データベース管理者は、アプリケーションとワークロードに応じて、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コマンドのどれかを使って設定できます。

```
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 コマンドで開始する場合のネットワーキングおよび処理のコストを節約するために、ALTER SESSION コマンドを使って、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;
```

追加情報: これらのSQL コマンドの詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

コミット読み込みによる分離

Oracle のデフォルト分離レベルは、コミット読み込みです。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反復不能読み込みと仮読み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。コミット読み込みによる分離は、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。

直列可能の分離

直列可能の分離は、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い、短いトランザクションでごく少数の行しか更新しない。
- 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。
- 比較的長時間にわたって実行されるトランザクションは主に読み込み専用である。

直列可能の分離の場合に、同時実行トランザクションが実行できる変更は、仮にトランザクションが順に 1 つずつ実行されたとした場合に実行できるようなデータベース変更だけです。特に、直列可能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、直列可能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だけです。

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いない変更が含まれているかを示す情報が使われます。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、たくさん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。たくさん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。そのようにすることで、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。

直列可能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と、次のようなエラーが生成されます。

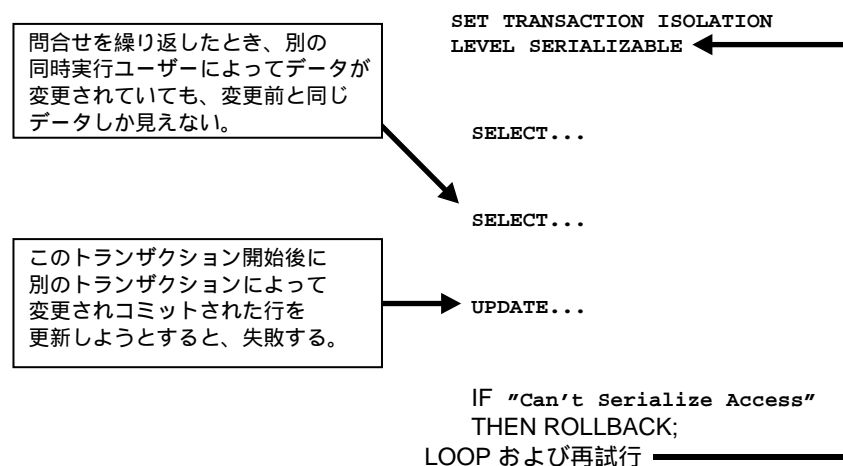
ORA-08177: このトランザクションのアクセスを逐次化できません。

直列可能トランザクションが失敗し、「アクセスを逐次化できません」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とることができます。

- 実行した作業をそのポイントまでコミットする。
- その他の(異なる)文を実行する(おそらく、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で)。
- トランザクション全体をロールバックする。

図 23-2 に、「アクセスを逐次化できません」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 23-2 直列可能トランザクションの障害



コミット読み込みと直列可能分離の比較

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。コミット読み込みと直列可能のどちらの分離レベルでも、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「読み込み一貫性」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キングがインプリメントされることによって競合が削減されます。また、これらの分離レベルは、実社会のアプリケーション展開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込みと直列可能の分離レベルを理解するため、次のようなシナリオを考えてみましょう。データベースの表の集まり（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、また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、コミット読み込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。直列可能モードでは、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 23-1 に、Oracle でのコミット読みトランザクションと直列可能トランザクションの主な違いをまとめます。

表 23-1 コミット読みトランザクションと直列可能トランザクション

	コミット読み	直列可能
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
反復不能読み	あり	なし
仮読み	あり	なし
ANSI/ISO SQL 92 に準拠しているか	はい	はい
スナップショット読み時期	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
「アクセスを逐次化できません」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直列可能トランザクションは、どちらも行レベルのロックを使います。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、待ち状態のトランザクションは（どの分離モードでも）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。

しかし、最初の（阻止する側の）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。ただし、直列可能トランザクションの場合には、その直列可能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、直列可能トランザクションはエラー「アクセスを逐次化できません」を出して失敗します。

参照整合性

Oracle は、読み込み一貫性トランザクションでも直列可能トランザクションでも読み込みロックを使わないので、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がないということを仮定するべきではありません。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なら、直列可能トランザクションを使っても、データベースの一貫性が損なわれることがあります。

追加情報：参照整合性および直列可能トランザクションの詳細は、
『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

Oracle Parallel Server

Oracle Parallel Server（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る）では、コミット読み込みトランザクションと直列可能トランザクションの両方の分離レベルを使えます。

分散トランザクション

分散データベース環境では、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（一部のノードだけがコミットされることがないように2フェーズ・コミットによって保護されます）。そのような環境では、直列可能トランザクションに関与するすべてのサーバーで（Oracleもそうでないものも）直列可能分離モードがサポートされていなければなりません。

直列可能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直列可能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・サーバーが直列可能トランザクションをサポートしている場合だけです。

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、直列可能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。

分離レベルの選択

アプリケーションのデザイナーおよび開発者は、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（トランザクションのスループット）とのバランスを考慮しなければならないことがよくあります。

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 つの分離モードはどちらも、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、高水準の一貫性と同時実行性（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ので、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直列可能の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う分離レベルです。

コミット読み込みによる分離では、同時実行性がかなり向上する一方で、一部のトランザクションにおいて（仮読み込みや反復不能読み込みのために）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の高い、高いパフォーマンスが求められる多くの環境では、直列可能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わずかで、トランザクション到着率が非常に低い環境では、仮読み込みおよび反復不能読み込み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わ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「アクセスを逐次化できません」エラーを検出したり、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なので、仮読み込みおよび反復不能読み込み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必要を避けるために、多くの開発者はコミット読み込みを選択します。

直列可能分離モードの選択

Oracle の直列可能分離は、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み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い、短いトランザクションでごく少数の行だけが更新されるような環境です。

直列可能分離モードは、仮読みと反復不能読み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、読み/書き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。

他の処理系の直列可能分離では書きみだけでなく読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、それらによって書きみ/読みの競合が少なくなります。読み/書きの競合が多く発生するアプリケーションでは、Oracle の直列可能分離は、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 上では直列可能分離が適していても、他のシステムでは直列可能分離に適さないアプリケーションもあります。

Oracle の直列可能トランザクション内のすべての問合せは、単一の時点でのデータベースを参照するので、読み/書き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。直列可能モードでは、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 および UPDATE、DELETE トランザクションも実行できるので、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、直列可能モードを使うとよいでしょう。

注意：副問合せを持つ DML 文を含むトランザクションは、コミット読みを保証するために直列可能分離を使う必要があります。

直列可能トランザクションをコーディングする場合には、追加の作業（「アクセスを逐次化できません」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）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る場合には、直列可能モードに対応したトランザクションを設計しなければならないことがあります。直列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 のコミット読みモード（直列可能性エラーを生成しないモード）で使えます。

直列可能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ので、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。（従来の読みロックによる直列可能モードの「消極的な」実現方式も、この環境には適していないことに注意してください。そのような実現方式では、長時間実行のトランザクションは、それが読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したりするからです。）

アプリケーション開発者は、直列可能モードの使用時に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み込みロックのシステムと同様に、直列可能モードを使うときには、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、再実行する必要が生じます。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、「アクセスを逐次化できません」のエラーを受け取った後に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ほとんどありません。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、直列可能トランザクション内のできるだけ前の部分で実行するとよい場合があります。しかし、そのトランザクションが正常に完了する保証はないので、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 の直列可能モードには SQL92 との互換性があり、読み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 での読み込みは、他のシステムとは違って書き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わなければならないことがあります。直列可能モードを使っ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、この問題を検討しなければなりません。

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（メモリー中の共有データ構造およびデータ・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の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。

Oracle が使う内部ロックの詳細は、23-18 ページの「ロックの種類」を参照してください。

トランザクションとデータ同時実行性

Oracle ではロック・メカニズムを使って、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・デザイナーに必要なのはトランザクションを適切に定義するだけであり、ロックは Oracle が自動的に管理します。

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。

ロックを手動で取得したり、Oracle のデフォルトのロック動作を変更したりする必要がある状況や、それらの操作方法の詳細は、後の項で説明します。23-29ページの「明示的（手動）データ・ロック」を参照してください。

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・モードを使います。

- | | |
|-----------|---|
| 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だけが参照できます。

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が、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だけです。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、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 文を使う場合を考えます。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。後ほど、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。表ロックの詳細は、23-20ページの「表ロック（TM）」を参照してください。

ロックの段階的拡大が発生するのは、あるレベル（たとえば行レベル）でたくさんのロックが保持されている場合に、それらのロックをデータベースがもっと高いレベル（たとえば表レベル）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たくさん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、ロックされている対象の制限は大きくなります。

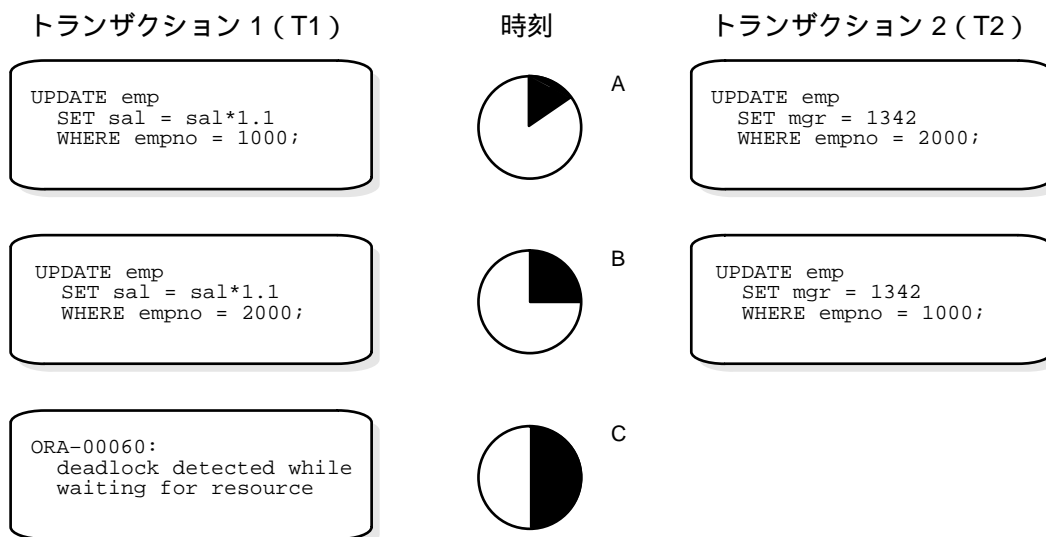
Oracle では、ロックの段階的拡大は発生しません。ロックの段階的拡大があると、デッドロックの発生する可能性はかなり高くなります（下記参照）。たとえば、システムがトランザクション T1 のためにロックを拡大しようとしますが、トランザクション T2 によって保持されているロックのために拡大できないものとします。ここで、トランザクション T2 の処理を進めるのに同じデータの段階的ロック拡大が必要な場合、デッドロックが発生します。

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、互い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、いくつかのトランザクションは作業を継続できなくなります。図 23-3 に、デッドロックの状態にある2つのトランザクションを示します。

図 23-3 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ので、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。しかし各トランザクションは、次に、もう一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。そのため、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ので、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 23-3 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、衝突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・ロックを明示的に置き換える場合です。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わず、また行レベルのロック（ページ・レベルのロックではない）を使うため、Oracleではデッドロックはまれにしか起こりません。手動でのロックの取得の詳細とデッドロック状況の例は、23-29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、暗黙のロックか明示的ロックのどちらかによりそれらの表を同じ順序でロックするなら回避できます。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うとよいでしょう。そのような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分かっているときは、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するとよいでしょう。

ロックの種類

Oracle は、データへの並行アクセスを制御し、各ユーザー間の有害な干渉を防止するためにさまざまなタイプのロックを自動的に使います。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（表ブロックか索引ブロック）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 ロックおよび DDL ロック、内部ロックのそれぞれについて説明します。

追加情報: 分散ロックと PCM ロックの詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

DML（データ）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だけ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、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で使われる略称です。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、行レベルのロックだけです。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上位のレベル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ので、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だけになります。特に、次のような場合です。

- データの読み込みは、同じデータ行の書き込みを待機しない。
- データの書き込みは、同じデータ行の読み込みを待機しない（読み込みのロックを要求するSELECT... FOR UPDATEを使っていない場合に限る）。
- 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する。

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を待機しなければならないこともあります。

INSERT文およびUPDATE文、DELETE文、FOR UPDATE句を含むSELECT文のいずれかで変更されたそれぞれの行ごとに、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。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。表ロックも必要なのは、衝突する DDL 操作によって現行トランザクションにおけるデータ変更が上書きされることを防ぐためです。次の項では、表ロックについて説明します。また、DDL 操作で必要なロックの詳細は、23-26 ページの「DDL ロック (ディクショナリ・ロック)」を参照してください。

表ロック (TM)

INSERT 文、UPDATE 文、DELETE 文、FOR UPDATE 句付きの SELECT 文、LOCK TABLE 文の各 DML 文で表が変更された場合、トランザクションでは表ロックが取得されます。これらの DML 操作が表ロックを必要とするのは、表への DML アクセスをトランザクション用に保存するのと、トランザクションと競合する可能性のある 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 ロックを防止するので、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。(排他 DDL ロックの詳細は、23-27 ページの「排他 DDL ロック」を参照。)

表ロックは、行共有 (RS) および行排他 (RX)、共有 (S)、共有行排他 (SRX)、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 23-2 に、文が取得する表ロックのモードと、それらの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 23-2 表ロックのまとめ

SQL 文	表ロックのモード	ロック・モードの許可				
		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 SHARE 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。手動ロックの詳細は、23-29 ページの「明示的（手動）データ・ロック」を参照してください。

行共有表ロック（RS）

行共有表ロック（「副共有表ロック、SS」ともいう）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」について行共有表ロックが自動的に取得されます。

データをロックする方法

```
SELECT . . . FROM table . . . FOR UPDATE OF . . . ;
```

```
LOCK TABLE table IN ROW SHARE MODE;
```

行共有表ロックは、最も制限の弱い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、同じ表にある行に対して問合せ、挿入、更新、削除、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、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だけを使って同じ表に排他書き込みアクセスを実行することを防止します。

```
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」ともいう) 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、同じ表にある行に対して問合せ、挿入、更新、削除、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、次の文を使っ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;
```

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、LOCK TABLE...IN SHARE MODE 文の実行だけが許可されます。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だけです。

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ことを禁止します。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」ともいう）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だけが、特定の表について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および共有行排他ロック、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。

データをロックする方法

```
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き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だけが、表について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だけを実行できます。

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施行を禁止します。

DML 文について自動的に取得される DML ロック

ここまでで、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。この後の部分では、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ロックするかをまとめます。

表 23-3 に、この後の項で説明する情報をまとめておきます。

表 23-3 DML 文によって取得されるロック

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だけなので、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、DELETE 文には、文の一部として暗黙の問合せが入っ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 句を使わないすべての問合せに当てはまります。

- 問合せはデータ・ロックを取得しない。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ず、そのため Oracle ではこのような問合せを「非ブロッキング問合せ」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る。（待ち状態の分散トランザクションがある場合は、ごくまれに問合せがデータ・ロックの解除を待機しなければならないことがあります。）

INSERT および UPDATE、DELETE、SELECT ... FOR UPDATE のデフォルト・ロック

INSERT 文、UPDATE 文、DELETE 文、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する。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から、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る。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るが、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ない。
- 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する。トランザクションがその表について、共有ロックまたは共有行排他ロック、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（ディクショナリ・ロック）

スキーマ・オブジェクト（表など）が処理中の DDL 操作によって影響を受けたり参照されたりする間、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 文は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）。たとえば、ユーザーがプロシージャを作る場合を考えます。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が完了する前に、プロシージャで参照されるオブジェクトが変更されたり削除されたりするのが防止されます。

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・オブジェクトだけがロックされます。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、ブレーク可能解析ロックの 3 つに分類されます。

排他 DDL ロック

ほとんどの DDL 操作（次の項「共有 DDL ロック」に示されている DDL 操作以外）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 操作では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 DDL ロックが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保持されている場合、取得は古い DDL ロックが解除されるまで待機し、その後処理されます。

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

DDL 操作によっては、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れるため、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。しかし、参照中の任意の表について排他 DDL ロックを取得できるトランザクションはありません。参照中の表を変更したり削除したり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。

共有 DDL ロックは、コマンド AUDIT、NOAUDIT、COMMENT、CREATE [OR REPLACE] VIEW/ PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/TRIGGER、CREATE SYNONYM、CREATE TABLE（CLUSTER パラメータが含まれていない場合）を含む DDL 文についてスキーマ・オブジェクトに対して取得されます。

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。依存性の管理の詳細は、第 19 章「Oracle の依存性の管理」を参照してください。解析ロックは、どのような DDL 操作も拒否せず、矛盾する DDL 操作を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。

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も排他 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 ロック操作は、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、そのクラスタに対して共有ロックを取得します。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構造を保護します。ユーザーは、それらの構造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Server Manager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直列化メカニズムです。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したりする間、非常に短い間だけ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、いろいろな目的で機能します。

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われたりしている間、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

これらのロックはさまざま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つのプロセスだけ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ので、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 を使っている場合は特に重要です。

追加情報：ロックの詳細は、『Oracle8 Parallel Server 概要および管理』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しなければなりません。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、文レベルの読み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。しかし、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・レベルの読み一貫性または「反復可能読み」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られるデータが、他のトランザクションによる変更を反映せず、トランザクションの存続期間にわたって一貫性を維持しなければならない場合。トランザクション・レベルの読み一貫性は、明示的ロック、読み専用トランザクション、直列可能トランザクションを使うか、デフォルトのロックを変更することで実現できます。
- 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	<p>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。</p> <ul style="list-style-type: none">• SET TRANSACTION ISOLATION LEVEL コマンド• LOCK TABLE コマンド (表をロックするコマンド、またはビューを使うときにその基礎となる実表をロックするコマンド)• SELECT... FOR UPDATE コマンド <p>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。</p>
セッション	<p>セッションでは、ALTER SESSION コマンド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。</p>

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かまたは適切に処理される、といった基準を満たすものでなければなりません。

追加情報: SQL 文 LOCK TABLE および SELECT ... FOR UPDATE の詳細は、『Oracle8 Server 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 文が使われるときに、Oracle がデータの同時実行性、整合性、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
<code>LOCK TABLE scott.dept</code>	1	
<code>IN ROW SHARE MODE;</code>		
文が処理される。		

トランザクション 1	時点	トランザクション 2
	2	DROP TABLE scott.dept; DROP TABLE scott.dept * ORA-00054 (T1 の表ロックのため、排他 DDL ロックは不可)
	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 行選択
UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (T2 が同じ行をロックしているため待機)	5	
	6	ROLLBACK; (行ロックを解除)
1 行処理。 ROLLBACK;	7	
LOCK TABLE scott.dept IN ROW EXCLUSIVE MODE; 文が処理される。	8	

データをロックする方法

トランザクション 1	時点	トランザクション 2
<pre> SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 行選択。 ROLLBACK; </pre>	9	<pre> LOCK TABLE scott.dept IN EXCLUSIVE MODE NOWAIT; ORA-00054 </pre>
	10	<pre> LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054 </pre>
	11	<pre> LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054 </pre>
	12	<pre> UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; 1 行処理。 </pre>
	13	<pre> ROLLBACK; </pre>
	14	
	15	<pre> UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (T1 が同じ行をロックしているため待機) </pre>
	16	

トランザクション 1	時点	トランザクション 2
	17	1 行処理。 (競合するロックが解除された) ROLLBACK;
LOCK TABLE scott.dept IN ROW SHARE MODE 文が処理される。	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; 文が処理される。
	22	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 行選択。
	23	SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 行選択。

データをロックする方法

トランザクション 1	時点	トランザクション 2
	24	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (T1 が競合する表ロックを保持しているため待機)
ROLLBACK;	25	
	26	1 行処理。 (競合する表ロックは解除) ROLLBACK;
LOCK TABLE scott.dept IN SHARE ROW EXCLUSIVE MODE; 文が処理される。	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

トランザクション 1	時点	トランザクション 2
	33	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 行選択。
	34	SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; LOC - - - - - DALLAS 1 行選択。
	35	UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (T1 が競合する表ロックを保持しているため待機)
UPDATE scott.dept SET loc = 'NEW YORK' WHERE deptno = 20; (T2 が同じ行をロックしているため待機)	36	(デッドロック)
操作取消し ROLLBACK;	37	
	38	1 行処理。
LOCK TABLE scott.dept IN EXCLUSIVE MODE;	39	

データをロックする方法

トランザクション 1	時点	トランザクション 2
	40	LOCK TABLE scott.dept IN EXCLUSIVE MODE; ORA-00054
	41	LOCK TABLE scott.dept IN ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	42	LOCK TABLE scott.dept IN SHARE MODE; ORA-00054
	43	LOCK TABLE scott.dept IN ROW EXCLUSIVE MODE NOWAIT; ORA-00054
	44	LOCK TABLE scott.dept IN ROW SHARE MODE NOWAIT; ORA-00054
	45	SELECT loc FROM scott.dept WHERE deptno = 20; LOC - - - - - DALLAS 1 行選択。
	46	SELECT loc FROM scott.dept WHERE deptno = 20 FOR UPDATE OF loc; (T1 が競合する表ロックを保持しているため待機)

トランザクション 1	時点	トランザクション 2
UPDATE scott.dept	47	
SET deptno = 30		
WHERE deptno = 20;		
1 行処理。		
COMMIT;	48	
	49	0 行選択。
		(T1 が競合するロックを解除)
SET TRANSACTION READ ONLY;	50	
SELECT loc	51	
FROM scott.dept		
WHERE deptno = 10;		
LOC		
- - - - -		
BOSTON		
	52	UPDATE scott.dept
		SET loc = 'NEW YORK'
		WHERE deptno = 10;
		1 行処理。
SELECT loc	53	
FROM scott.dept		
WHERE deptno = 10;		
LOC		
- - - - -		
BOSTON		
(T1 から未コミット・データは見えない)		
	54	COMMIT;

データをロックする方法

トランザクション 1	時点	トランザクション 2
<code>SELECT loc</code>	55	
<code>FROM scott.dept</code>		
<code>WHERE deptno = 10;</code>		
LOC		
- - - - -		
(T2 のコミット後も同じ結果が参照される)		
<code>COMMIT;</code>	56	
<code>SELECT loc</code>	57	
<code>FROM scott.dept</code>		
<code>WHERE deptno = 10;</code>		
LOC		
- - - - -		
NEW YORK		
(コミットされたデータが参照される)		

Oracle のロック管理サービス

アプリケーションの開発者は、Oracle のロック管理サービスを使っ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する。
- 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、一意の名前を指定する。
- ロック・タイプを変更する。
- ロックを解除する。

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 のロック管理サービスは、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。

追加情報 : Oracle のロック管理サービスの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

データの整合性

Does one's integrity ever lie in what he is not able to do?

Flannery O'Connor: *Wise Blood*

この章では、データベースに関連する業務ルール（業務規則）を施行し、無効な情報を表に入力しないように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。

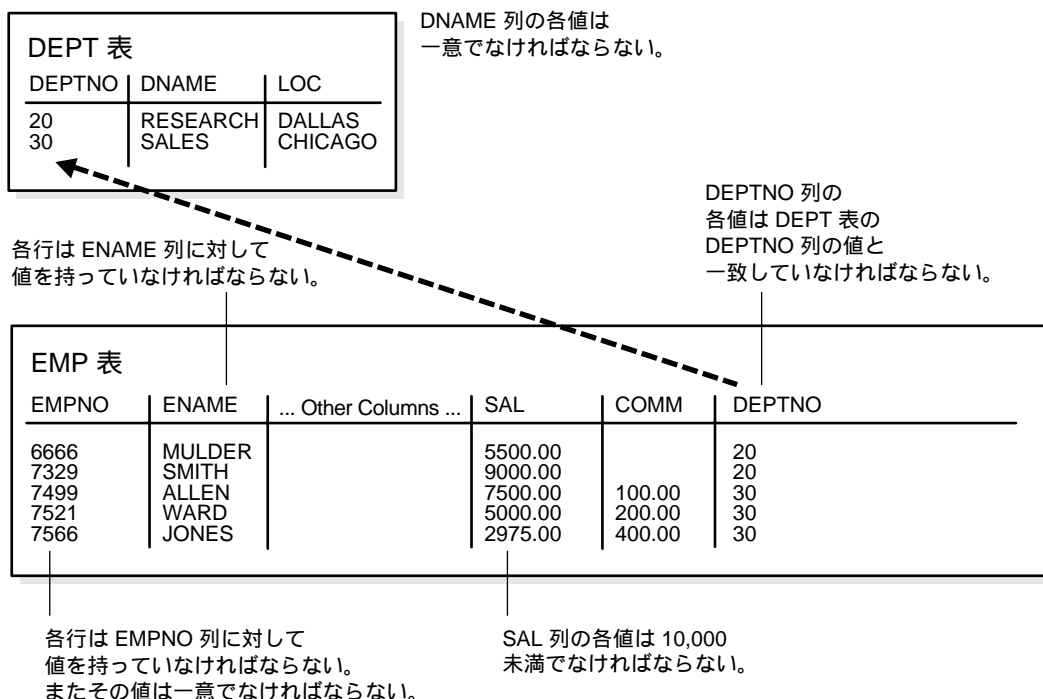
- データ整合性の定義
- 整合性制約の基礎知識
- 整合性制約のタイプ
- 制約チェックのメカニズム
- 遅延制約チェック
- 制約の使用可能および使用禁止、施行

追加情報: Trusted Oracleを使っている環境での整合性制約の詳細は、Trusted Oracle のマニュアルを参照してください。

データ整合性の定義

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 24-1 に示す EMP 表と DEPT 表、およびこれらの各表の情報に関する業務ルールについて考えてみましょう。

図 24-1 データの整合性の例



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、さまざまな種類のデータの整合性を施行するために表の列に適用できるルール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ルールであり、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、そして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する。
SET 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に設定される。
SET DEFAULT 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る。
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る。
No Action	参照先のデータの更新または削除を禁止する。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を使います。）

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施行する方法

Oracle では、前述のデータ整合性規則のそれぞれを定義し、施行できます。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・トリガーを使っ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 FOREIGNKEY 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
- CHECK 制約。複雑な整合性規則の場合に使います。

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って参照整合性の施行はできません。しかし、データベース・トリガーを使って、分散データベースで参照整合性を施行することはできます（下記参照）。

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、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）を使うことによって、宣言以外の形の整合性制約を施行できます。データの整合性の施行で使われるデータベース・トリガーの詳細と例は、第 18 章「データベース・トリガー」を参照してください。

整合性制約の基礎知識

Oracle では、データベースの実表に無効なデータが入力されないようにするため、整合性制約を使います。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・ルールを施行できます。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、基礎となる実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。この表のどの行のこの列にも 10,000 より大きい数値を入れてはならないという規則を、整合性制約として施行します。INSERT 文か UPDATE 文がこの整合性制約に違反しそうになると、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 でインプリメント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・ルールを施行する。
- ストアド・プロシージャを使ってデータへのアクセスを完全に制御する。
- トリガー起動されるストアド・データベース・プロシージャを使ってビジネス・ルールを施行する（第 18 章「データベース・トリガー」を参照）。

宣言の容易さ

整合性制約は、SQL コマンドを使って定義します。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・プロシージャを使ってデータ・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うほうがストアド・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、表に対して定義され（アプリケーションに対してではない）、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、整合性制約で使わ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を最大化できる

整合性制約を変更することでビジネス・ルールを施行する場合、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がその修正された制約を自動的に遵守できるようになります。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・ルールを施行するとすれば、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、テストしなければなりません。

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さえ、データベース・アプリケーションがその情報を使っ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、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って、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

整合性制約宣言は意味の上で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 問合せオブティマイザは、宣言を使って、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施行することの利点は、パフォーマンスを多少犠牲にしてはじめて獲得できます。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
- FOREIGN KEY (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 (値なし) を使えます。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 表のすべての行にわたって ENAME 列に必ず値を入力するよう求める NOT NULL 制約を定義できます。

図 24-2 に、NOT NULL 整合性制約を示します。

図 24-2 NOT NULL 整合性制約

EMP 表							
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 制約
(この列に対しては
NULL 値を持つことができない)

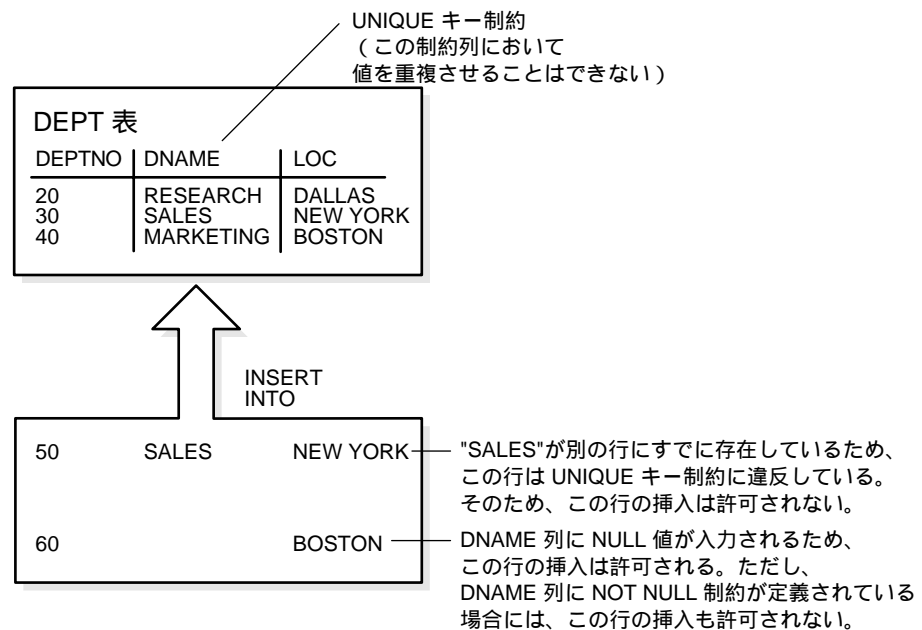
NOT NULL 制約が
定義されていない
(この列に対して
行は NULL 値を
持つことができる)

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なければなりません。つまり、指定した列または列の集合について、表の 2 つの行の値が重複していません。

たとえば、図 24-3 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 24-3 UNIQUE キー制約

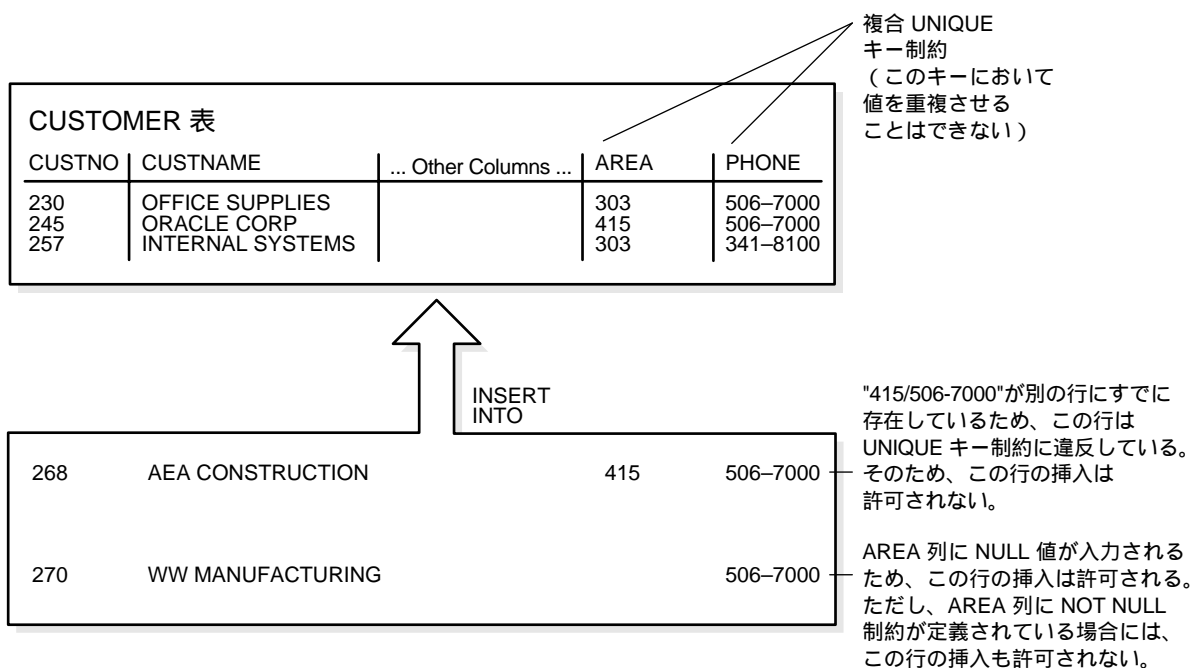


一意キー

UNIQUEキー制約の定義に含まれている列（または列の集合）を、「一意キー」といいます。「一意キー」という用語は、誤って「UNIQUE キー制約」や「UNIQUE 索引」の同義語として使わ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われている列または列の集合だけを指して使われるので、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「複合一意キー」といいます。たとえば、図 24-4 の例では、CUSTOMER 表の複合一意キー（AREA 列と PHONE 列）に対して UNIQUE キー制約が定義されています。

図 24-4 複合 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」を同じ表の中で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してしまうのを避けられます。

UNIQUE キー制約と索引

Oracle は、索引を使って一意の整合性制約を施行します。(図 24-4 で、Oracle は、複合一意キーに対する一意索引を暗黙的に作ることにより、UNIQUE キー制約を施行しています。)したがって、複合 UNIQUE キー制約は複合索引に対して課されているのと同じ制限を受けます。複合一意キーを構成する最大列数は 32 個で、キー値の合計サイズ(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てはなりません。UNIQUE キー制約が作成されたときに使える索引がある場合、制約は新しい索引を暗黙的に作るかわりにその索引を使います。

UNIQUE キー整合性制約と NOT NULL 整合性制約を組み合わせる

図 24-3 と図 24-4 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 はどんなものとも等しいとみなされることがないので、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（または複合 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、よく使われます。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しなければならなくなり、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 の複合 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 つまたは複数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 にインプリメントされている 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ない。
- 主キー列では、NULL は許可されない（つまり、それぞれの行の主キー列には値が必ず存在しなければならない）。

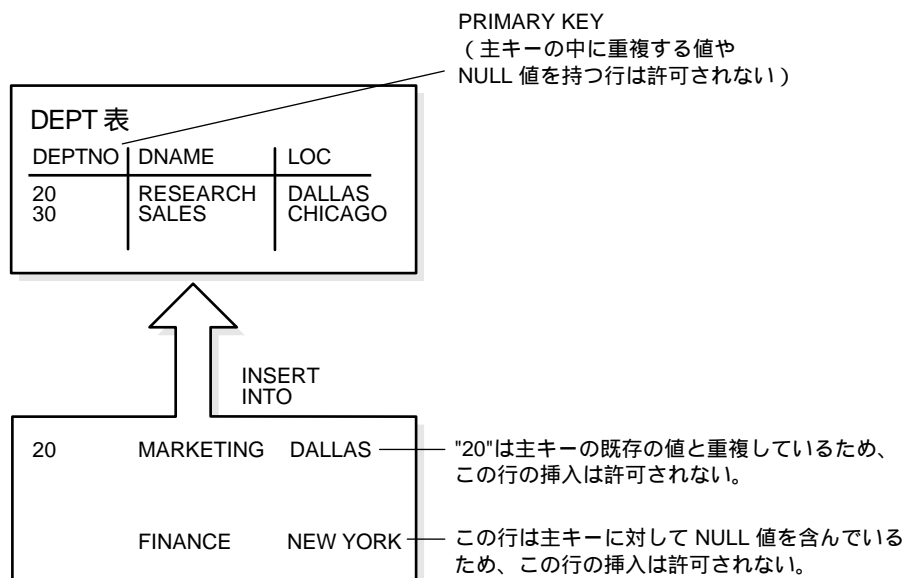
主キー

表の PRIMARY KEY 整合性制約の定義に含まれている列（または列の集合）を、主キーといいます。主キーは必須ではありませんが、次の利点を考慮すると、すべての表に主キーを指定するようにしてください。

- 表のそれぞれの行を一意に識別できる。
- 重複する行が表に存在しない。

図 24-5 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 24-5 PRIMARY KEY 制約



PRIMARY KEY 制約と索引

Oracle では、すべての PRIMARY KEY 制約が索引を使って施行されます。図 24-5 では、DEPTNO 列に対して作られた主キー制約は、次の方法で施行されます。

- その列に対して一意の索引を暗黙のうちに作る。
- その列に対して NOT NULL 制約を暗黙のうちに作る。

Oracle は、索引を使って主キー制約を施行し、複合主キー制約は、複合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、制約を作るのに使う 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。主キー制約が作成されたときに使える索引がある場合、主キー制約は新しい索引を暗黙的に作るかわりにその索引を使います。

FOREIGN KEY (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なければなりません。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。

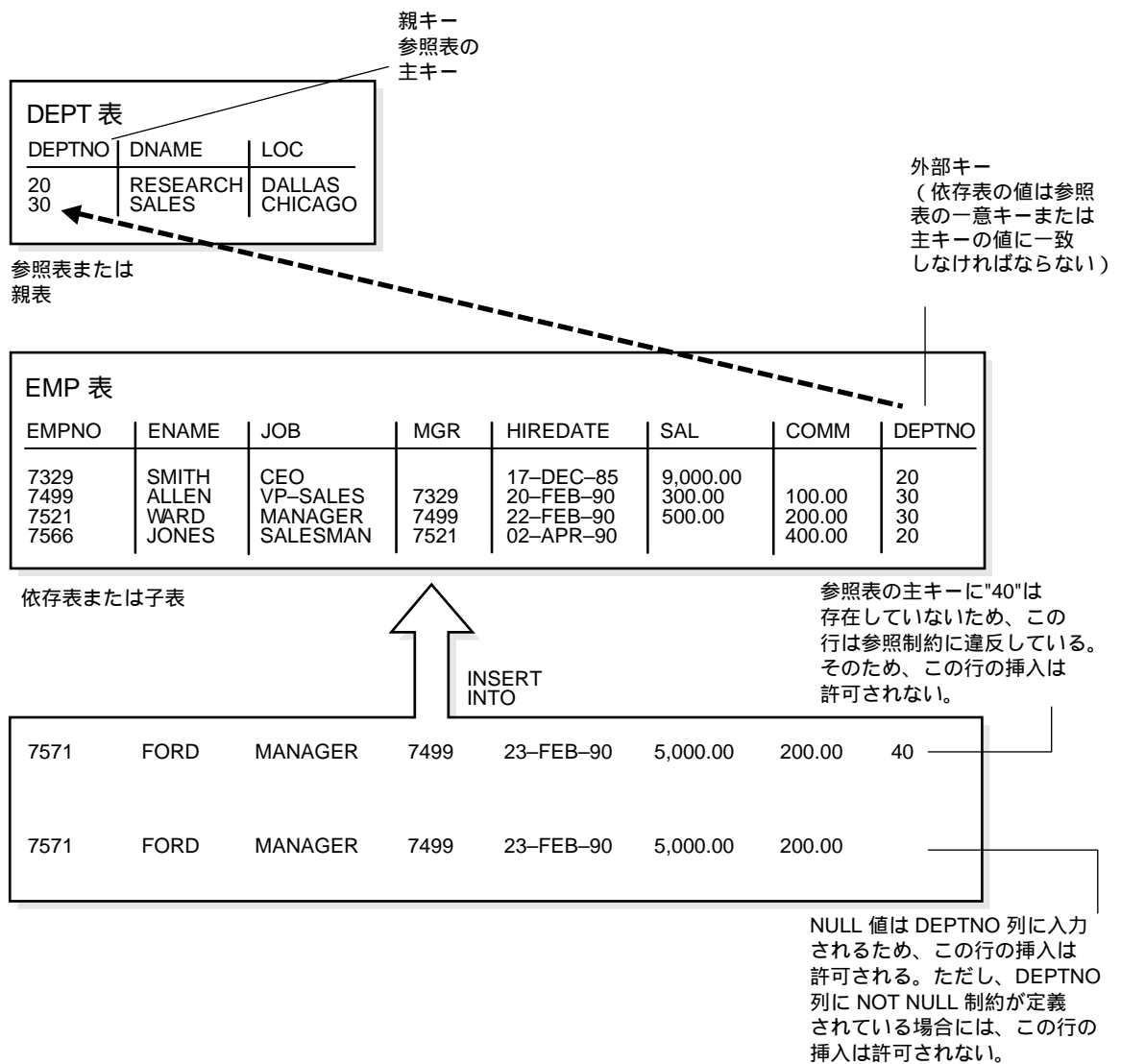
外部キー	参照整合性制約の定義に含まれている列または列の集合のうち、参照キー（次を参照）を参照するもの。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。

参照整合性制約では、表のそれぞれの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 24-6 に、EMP 表の DEPTNO 列に対して定義された外部キーを示します。それにより、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、複数の列で構成できます。ただし、複合外部キーの列数とデータ型は、参照先の複合主キーまたは一意キーと同じでなければなりません。複合主キーおよび一意キーは 32 列までに制限されているので、複合外部キーも 32 列までに制限されます。

図 24-6 参照整合性制約

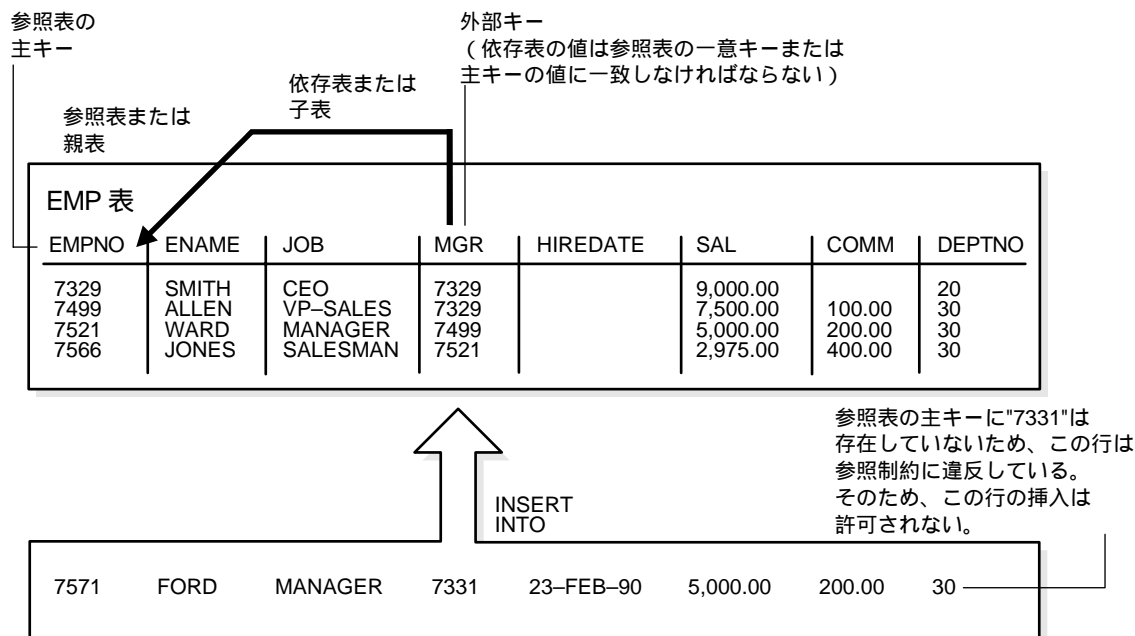


自己参照型整合性制約

図 24-7 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いいます。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 24-7 の例では、参照整合性制約によって、EMP 表の MGR 列のすべての値が、同じ表の EMPNO 列に現在存在する値（必ずしも同じ行ではない）と一致する（つまり、すべての管理職は従業員でもなければならない）ことが保証されます。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 24-7 単一の表の参照制約



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。複合（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）標準では、複合外部キーの非 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構いません。一部が NULL になっている外部キー処理は、他の制約（NOT NULL 制約、CHECK 制約など）を使ってデフォルトの処理から変更できます。

複合外部キーの値としては、NULL、すべて NULL 以外、または一部 NULL という値があります。次の用語は、複合外部キーに関する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 の複合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 の複合外部キーが許可されます。複合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 および DELETE No Action、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・トリガーを使って施行できます。詳細は、第 18 章「データベース・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、依存データなので、参照先的主キー値を削除できません。

DELETE CASCADE

DELETE CASCADE アクションは、参照キー値が入っている行が削除された場合に、子表のうち依存外部キー値を含むすべての行も削除すること (削除カスケード) を指定します。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。

参照アクションに関する DML 制限

表24-1に、親表の主/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 24-1 UPDATE No Action と DELETE No Action で許される DML 文

DML 文	親表に対して発行する場合	子表に対して発行する場合
INSERT	親キー値が一意であれば常に発行できる。	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の場合だけ発行できる。
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る。	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る。
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る。	常に発行できる。
DELETE CASCADE	常に発行できる。	常に発行できる。

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、指定した条件が真または不明でなければなりません。DML 文の結果が、CHECK 制約の条件が偽に評価されることになる場合、その文はロールバックされます。

チェック条件

CHECK制約を使うと、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施行できます。CHECK 制約の条件には次のような制限があります。

- 挿入または更新しようとしている行の値を使って評価されるブール式でなければならない。
- 副問合せ、順序、SQL ファンクションの SYSDATE または UID、USER、USERENV、あるいは疑似列 LEVEL または ROWNUM が含まれていてはならない。

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 設定を使います。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。

追加情報 : NLS 機能の詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

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。

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、24-14 ページの図 24-7 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る。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 列だけを対象にします。

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 を入力できる。外部キーには NULL が許されるので、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る。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を入力できるということは、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ているはずです。
- 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る。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、その後で、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することもできます。24-19 ページの「遅延制約チェック」を参照。）

同じ自己参照型の整合性制約に関して、次のシナリオを考慮してみましょう。

- 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なので、これらの値にも 5000 を加算する必要があります。

今のところ、この表は図 24-8 に示すような状態になっています。

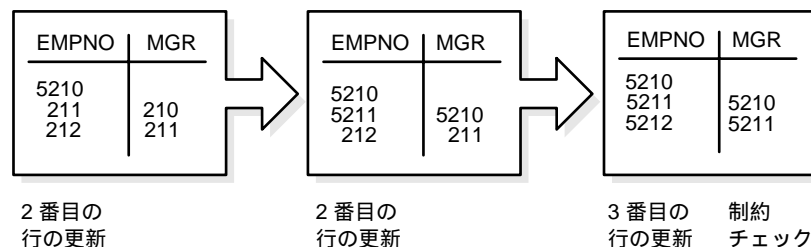
図 24-8 更新前の EMP 表

EMPNO	MGR
210	
211	210
212	211

```
UPDATE emp
SET empno = empno + 5000,
    mgr = mgr + 5000;
```

それぞれの MGR 列が EMPNO 列と一致しているかどうか検証する制約が定義されてはいても、Oracle はその制約チェックを実際には文が完了した後で実行するので、この文は有効です。図 24-9 に、Oracle が、制約チェックの前に SQL 文全体のアクションを実行する様子を示します。

図 24-9 制約チェック



この部分の例は、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すものでした。UPDATE および INSERT、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っています。

また、これらの例は、自己参照型整合性制約を使ってチェックのメカニズムを示すものでした。NOT NULL および UNIQUE キー、PRIMARY KEY、すべてのタイプの FOREIGN KEY、CHECK 制約でも、同じメカニズムが使われています。

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」とは、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ない場合のこと。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る。制約違反があると、その文はすぐ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は関係なく、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。それらの属性は、制約ごとに違う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って指定します。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE

追加情報: これらの制約属性とそのデフォルト値の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

制約は追加したり、削除したり、使用可能や使用禁止にしたり、妥当性を検査したりできますが、変更はできません。特に、遅延不可の制約を遅延可能に変更することはできません。

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 標準に準拠）。この文を使うと、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ずっと、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、各制約文を実行したらすぐに、指定されている制約をチェックするように指示します。Oracle は、トランザクション内の遅延制約を最初にチェックし、次にそのトランザクション内のそれ以降の文の制約は即時にチェックします（チェック制約が一貫しており、他にSET CONSTRAINTS文が発行されない限り）。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のオプションがあります。これらのオプションでは、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ない）。これらのオプションを指定することは、現行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います。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ので、これは便利です。）遅延可能でない一意制約と主キーは、制約が施行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います。

制約の使用可能および使用禁止、施行

CREATE TABLE 文または ALTER TABLE 文を使うと、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。ALTER TABLE 文の ENABLE NOVALIDATE オプションを使うと、表の中のすべてのデータの妥当性を検査することなく、使用禁止にした制約に対する制約チェックを再開できます。

- ENABLE CONSTRAINT は、表のすべての行が有効であること、つまりすべての行が制約に適合していることを確認する。
- DISABLE CONSTRAINT は、制約に違反する行が表に含まれることを許す。
- ENABLE NOVALIDATE CONSTRAINT は、既存の行が制約に違反することは許すが、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を確認する。

追加情報: ENABLE および DISABLE、ENABLE NOVALIDATE CONSTRAINT オプションの使用の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

データベース・アクセスの制御

Allow me to congratulate you, sir. You have the most totally closedmind that I've ever encountered!

Jon Pertwee (as the Doctor): *Frontier in Space*

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。この章の内容は次のとおりです。

- データベース・セキュリティ
- スキーマおよびデータベース・ユーザー、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

追加情報: Trusted Oracleを使っている環境でのデータベース・アクセスの詳細は、Trusted Oracle のマニュアルを参照してください。

データベース・セキュリティ

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したり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・ドメインを使って、データへのアクセスを制御したり、さまざまなデータベース・リソースの使用を制限したりします。

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、他のユーザーの任意で付与されるものなので、「任意セキュリティ機能」という語を使っています。権限の詳細は、第26章「権限とロール」を参照してください。

スキーマおよびデータベース・ユーザー、セキュリティ・ドメイン

「ユーザー」(「ユーザー名」ともいう)は、データベースの中で定義されている名前であり、この名前を使ってオブジェクトに接続したりアクセスしたりできます。「スキーマ」は、特定のユーザーに対応付けられている表、ビュー、クラスタ、プロシージャ、パッケージなどのオブジェクトの集まり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。

データベースにアクセスするには、データベース・アプリケーション (Oracle Forms、SQL*Plus、プリコンパイラ・プログラムなど) を実行し、データベースに定義されているユーザー名を使って接続する必要があります。

データベース・ユーザーを作ると、同じ名前の対応するスキーマがこのユーザー用に作られます。デフォルトでは、ユーザーがいったんデータベースに接続すると、そのユーザーは対応するスキーマに含まれているすべてのオブジェクトへのアクセス権を所有することになります。ユーザーは、同じ名前のスキーマにしか対応付けられていません。したがって、ユーザーという用語とスキーマという用語は、よく交換可能な語として使われます。

ユーザーのアクセス権は、そのユーザーのセキュリティ・ドメインのさまざまな設定値によって制御されます。新たにデータベースを作ったり、既存のデータベースを変更したりする場合、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、データベースまたはオペレーティング・システム、ネットワークの認証サービスのどれを使って維持管理するか。
- 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。ユーザーが使えるシステム・リソースの量を制限します。

- データベース操作を実行するのに必要な、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とロール。

この章では、上記の最初の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。権限とロールについては、第 26 章「権限とロール」で説明しています。

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。

追加情報：特殊なユーザー SYS および SYSTEM の詳細は、『Oracle8 Server 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われないようにするため、Oracle では次の 3 つの方法で標準データベース・ユーザーの妥当性を検査します。

- オペレーティング・システムによる認証
- ネットワーク・サービスによる認証
- 対応する Oracle データベースによる認証

通常は、検査を簡素化するため、上記のどれか 1 つの方法を使ってデータベースのすべてのユーザーを認証します。しかし、Oracle では、同じデータベース・インスタンス内で上記の方法をすべて使えます。

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

オペレーティング・システムの中には、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えるようになっているものがあります。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 に接続できる（ユーザー名やパスワードを指定しなくてよい）。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る。Oracle がユーザー・パスワードを格納したり管理したり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項目と、オペレーティング・システムの監査証跡が一致する。

データベース・ユーザーの認証にオペレーティング・システムを使う場合は、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。詳細は、第 30 章「分散データベース」を参照してください。

追加情報：オペレーティング・システムによる認証の詳細は、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ください。

ネットワークによる認証

ネットワーク認証サービス（DCE、Kerberos、SESAME など）を使える場合、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うには、Oracle Secure Network Services 製品も必要になります。

追加情報：ネットワーク認証サービスを使う場合は、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。ネットワーク認証の詳細は、『Oracle8 Server 分散システム』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って、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。データベース・ユーザーの妥当性チェックにオペレーティング・システムを使えないときは、この方法を採用する必要があります。

Oracle でデータベースによる認証を使う場合は、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ります。ユーザーは、接続を確立する時に正しいパスワードを入力します。この方法により、データベースが無許可で使われることが防止されます。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）アルゴリズムを使って暗号化され、その後ネットワーク経由で送信されます。

追加情報：ネットワーク・システムにおけるパスワードの暗号化の詳細は、『Oracle8 Server 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によって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の時間を構成するのに使います。プロファイルの詳細は、25-11 ページの「プロファイル」を参照してください。

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。

パスワードの存続期間と期限切れ

パスワードの存続期間と時間切れのオプションを使うと、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しなければなりません。パスワードの期限が切れた後にデータベース・アカウントにログインしようとする最初の試みで、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しようと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いなら、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期限切れに設定することもできます。その場合、ユーザー・アカウント状態は期限切れに変更され、ユーザーがログインすると、アカウントは猶予期間に入ります。

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、指定された期間に、または指定されたパスワード変更回数の中に、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 文を使ってパスワードの再使用のルールを構成できます。

パスワードの複雑さの検証

複雑さの検証は、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できるだけ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、パスワードの複雑さのデフォルトの検証ルーチンでは、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
- 少なくとも 1 文字のアルファベット、1 文字の数字、1 文字の句読点が含まれている

- welcome、account、database、user などの簡単な語からなる内部リストにある単語に一致しない
- 前のパスワードと、3 文字以上異なっている

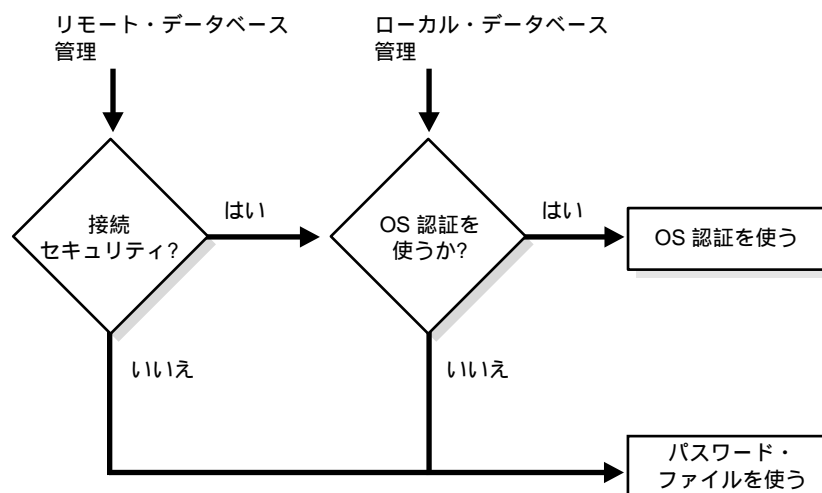
データベース管理者の認証

データベース管理者は、通常のデータベース・ユーザーの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います。

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うかを選択できます。

図 25-1 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て、データベース管理者の認証方式の選択肢を示します。

図 25-1 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 OS 認証には、データベース管理者の OS 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 OS 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。

追加情報：データベース管理者の OS 認証の詳細は、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ください。

データベースは、パスワード・ファイルを使うことによって、SYSDBA または SYSOPER 権限を付与されたデータベース・ユーザー名の記録を取ります。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 および SHUTDOWN、ALTER DATABASE OPEN/MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG、RECOVERを実行できます。そして、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 と時間ベースの回復が許可されます。

追加情報：『Oracle8 Server 管理者ガイド』を参照してください。

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、表領域の使用方法についていくつかのオプションを設定できます。

- ユーザーのデフォルト表領域
- ユーザーの一時表領域
- ユーザーに対するデータベースの表領域での領域使用の割当て制限

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入れられます。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たり、ユーザーの作成後に変更したりできます。

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。そのようにするなら、次の2つのことが達成されます。

- ユーザーに適切な権限を付与されているなら、指定された表領域を使ってオブジェクトを作ることをそのユーザーに対して許可できる。
- 指定された表領域の中にある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る。

デフォルトでは、それぞれの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・オブジェクトを作る権限がある場合、そのユーザーには、オブジェクトを作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る権限が付与されていなければなりません。

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（バイトまたはKB、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いすぎないようにするため、特定量の割当て制限を設定するべきです。

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られても、それはユーザーが課せられている割当て制限の領域には加算されない。
- 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れる。

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したりできます。

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、特定のスキーマ・オブジェクト（表、ビューなど）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えます。

ユーザーは、PUBLICに対して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。権限とロールの詳細は、第26章「権限とロール」を参照してください。しかし、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とって関係のあるものだけにしてください。

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したり取り消したりすると、データベース中のすべてのビューおよびプロシージャ、ファンクション、パッケージ、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ないが、UNLIMITED TABLESPACE システム権限は設定できる。

- データベース・リンクおよびシノニムは、(CREATE PUBLIC DATABASE LINK/SYN-ONYM を使って) PUBLIC として作れるが、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ない。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・セグメントを作ることにはできません。すべてのロールバック・セグメントは SYS が所有します。ロールバック・セグメントの詳細は、第2章「データ・ブロック、エクステンツ、セグメント」を参照してください。

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、各ユーザーが使える各種のシステム・リソースの容量に制限を設定できます。そのようにするなら、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ないようにできます。

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 人がそれ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は、そのユーザーのプロファイルを使って管理します。プロファイルとは、そのユーザーに割り当てることの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施行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。

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、セッションが作られます。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。いくつか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、現行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、現行のトランザクションはコミットされる）だけです。こ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でも、現行のセッションにおいて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。この処理の際に、データベースに対していくつかのコールが、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われないようにするため、いくつか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。しかし、現行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。しかし、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ので、他の処理に使える CPU 時間が少なくなってしまうます。

CPU 時間が無制限に消費されないようにするため、1 回のコールあたりの CPU 時間と、1 つのセッション中に Oracle コールに使われる 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われる 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ので、他のデータベース操作がこれらのリソースをめぐる競争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 コール当たり、また 1 セッション当たりの論理データ・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。

- ユーザーあたりの同時実行セッション数の制限。それぞれのユーザーは、事前に定義された数まで同時実行セッションを作れます。
- 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、現行の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、ユーザーがもはやインスタンスに接続されていない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/ユーザー・リソース制限の中に含まれます。

- セッション当た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、現行のトランザクションはロールバックされ、セッションは削除され、そのセッションのリソースは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てしまいます。そうではなく数分ごとにチェックします。このため、Oracle がこの制限を実施して、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（たとえば5分）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 領域に使われる）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うシステムの場合だけです。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 に使う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” の文字を使います。

追加情報：リソース制限を使用可能または使用禁止にする方法の詳細は、『Oracle8 Server 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のできる一連のリソース制限に名前を付けたものです。プロファイルを使うと、リソース制限を簡単に管理できます。

いつプロファイルを使うか

ユーザー・プロファイルを作っ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・セキュリティ方針の要件になっている場合だけです。プロファイルを使うには、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うのと同じようにして、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います。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

プロファイルを作り、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、それぞれの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。それらの値は、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 オプションを使うことによって、CONNECT_TIME および LOGICAL_READS_PER_SESSION、LOGICAL_READS_PER_CALL の制限値についての情報を収集できます。詳細は、第 27 章「監査」を参照してください。

その他の制限値の統計情報は、Oracle Enterprise Manager（または Server Manager）のモニター機能、特に統計モニターを使って収集できます。

ライセンス

通常、Oracle のライセンスには、名前の付けられるユーザーの最大数、または同時に接続する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）には、そのサイトでライセンス契約を守ることに關する責任があります。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られるユーザーの数を制限したりすることによって、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ったりすることが DBA にとって必要になったなら、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、必ずオラクル社カスタマー・サポートに連絡してください。）

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、または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、一部の国で使う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で、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。

追加情報：ライセンスの詳細は、『Oracle8 Server 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）を使っている場合、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に制限を設定できる。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に警告制限を設定できる。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って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 コマンドを使って）それらを変更したりできます。後者の操作は、オフラインには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来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、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、この情報を使っ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。それらの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・リンク用に作ら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 によって作ら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に入れる責任は、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れなくなります。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って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 コマンドを使って) それらを変更したりできます。後者の操作は、オフラインには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。

追加情報: Oracle Parallel Server の詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

権限とロール

My right and my privilege to stand here before you has been won - won in my lifetime - by the blood and the sweat of the innocent.

Jesse Jackson: *Speech at the Democratic National Convention, 1988*

この章では、ユーザーがどんなシステム操作を実行できるか、およびスキーマ・オブジェクトに対してどんなアクセスを実行できるかを、権限とロ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
 - システム権限
 - スキーマ・オブジェクト権限
- ロール

追加情報：Trusted Oracle を使っている環境でのロールと権限の詳細は、Trusted Oracle のマニュアルを参照してください。

権限

「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（セッションを作る）
- 表を作る
- 他のユーザーの表から行を選択する
- 他のユーザーのストアド・プロシージャを実行する

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だけに付与しなければなりません。必要でない権限を過剰に付与すると、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れます。

- 権限を明示的にユーザーに付与する。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する。たとえば、EMP 表からレコードを選択、挿入、更新、削除する権限を、CLERK という名前のロールに付与できます。さらに、今度はこのロール CLERK をユーザー SCOTT や BRIAN に付与できます。

ロールを使う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

追加情報：すべてのシステム権限とスキーマ・オブジェクト権限のリスト、および権限を管理する方法の詳細は、『Oracle8 Server 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、あるい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、表領域を作る権限、データベース内の任意の表から行を削除する権限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したりできます。システム権限をロールに付与すると、そのロールを使ってシステム権限を管理できます（たとえば、ロールを使うと権限を選択的に利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ので、システム権限は、管理者やアプリケーション開発者だけに付与するようにしてください。

次のどちらかを使って、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たりします。

- Oracle Enterprise Managerの「システム権限/ロールの許可」ダイアログ・ボックスおよび「システム権限 / ロールの取消し」ダイアログ・ボックス
- SQL コマンド GRANT および REVOKE

だれがシステム権限を付与したり取り消したりできるか？

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したりできるのは、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（通常はデータベース管理者またはセキュリティ管理者）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（「オブジェクト権限」）とは、特定の表、ビュー、順序、プロシージャ、ファンクション、パッケージに対して特定のアクションを実行する権限（または権利）です。さまざまなタイプのスキーマ・オブジェクトごとに、さまざまな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。

一部のスキーマ・オブジェクト（クラスタおよび索引、トリガー、データベース・リンク）には、関連付けられる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制御されます。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 システム権限を持っていないければなりません。

スキーマ・オブジェクトとそのシノニムは、権限に関しては同等です。つまり、表またはビュー、順序、プロシージャ、ファンクション、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 を使っ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、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わ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えます。

```
GRANT SELECT ON emp TO swilliams;  
GRANT SELECT ON employee TO swilliams;
```

シノニムが削除された場合、その削除されたシノニムの指定によっ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、付与したり取り消したりできます。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したりするには、それぞれ SQL コマンドの GRANT と REVOKE を使うか、Oracle Enterprise Manager の「ロール/ユーザーへの権限追加」ダイアログ・ボックスと「ロール/ユーザーからの権限取消」ダイアログ・ボックスを使います。

だれがスキーマ・オブジェクト権限を付与できるか？

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 コマンドの 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。このオブジェクトを指定しない場合、権限受領者はその権限を使えますが、別のユーザーには権限を付与できません。

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) の操作

DELETE および INSERT、SELECT、UPDATE の各権限は、それぞれ表またはビューに対する DELETE および INSERT、SELECT、UPDATE の各 DML 操作を可能にします。これらの権限は、表のデータを問い合わせたり変更したりする必要があるユーザーとロール以外には付与しないでください。

追加情報：これらの DML 操作の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

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 権限を選択的に使います。

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 権限を UPDATE 権限を付与するとよいでしょう。(また、SAL 列を含まないようにしたビューを使うなら、同じことをさらに高いセキュリティで実現できます。)

データ定義言語 (DDL) の操作

ALTER、INDEX、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したり作ったりできるようになるため、この権限は慎重に付与しなければなりません。表に対して DDL 操作を実行しようとするユーザーは、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、自分の表の中に作りたい外部キーの親キーとして使えます。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、この動作は特別な権限によって制御されます。列固有の REFERENCES 権限によって、権限受領者が使えるのは、指定された列(当然ながら、親表の主キーまたは一意キーを最低1つ含んでいる)に制限されます。主キーおよび一意キー、整合性制約の詳細は、第 24 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、ビューの導出元の実表に実際に影響を与えるさまざまな 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。

ビューを作るのに必要な権限

ビューを作るには、次の要件を満たし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る) または 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る)が、明示的にまたはロールによって付与されていなければならない。
- さらに、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、DELETE の各オブジェクト権限、または SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE の各システム権限が明示的に付与されていなければならない。これらの権限は、ロールによっては取得できません。
- 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 オプション付きで付与されている

か、またはADMIN OPTIONオプションによって適切なシステム権限が付与されていなければならない。これらの権限がない場合、権限受領者はビューにアクセスできません。

ビューを使って表のセキュリティを強化する

ビューを使うのに必要な権限は、そのビュー自体に対する適切な権限だけです。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。

ビューを使うことにより、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、実表の中から選択した列のアクセスを提供する。たとえば、EMP表の中からEMPNO および ENAME、MGR の3列だけ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る。ビュー定義の中に WHERE 句を使うと、実表の中から選択した行だけが表示されます。次の2つの例を考えてみます。

```
CREATE VIEW lowsal AS
  SELECT * FROM emp
  WHERE sal < 10000;
```

LOWSAL ビューによって、EMP 表のうち給与が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点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 がビューの現ユーザーに一致している行だけにアクセスできます。OWN_SALARY ビューは USER 関数を使います。この関数の値は、常に現ユーザーを返します。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一体化しています。

プロシージャのセキュリティに関するトピック

プロシージャ（スタンドアロン・プロシージャ、ファクション、パッケージを含む）に対する1つの「スキーマ・オブジェクト権限」は、EXECUTE です。この権限は、プロシージャを実際に行う必要があるユーザー以外には付与しないでください。

プロシージャを使って、データベースのセキュリティ・レベルを強化できます。ユーザーに必要なのはプロシージャを実行する権限だけであり、プロシージャがアクセスする基礎オブジェクトに対する権限は必要ありません。プロシージャを記述し、ユーザーにはEXECUTE権限だけ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（つまり、ユーザーはそのデータベースに対して非定型のSQL文を発行できません）。

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してEXECUTE オブジェクト権限を付与されているユーザーは、そのプロシージャを実行できます。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。

プロシージャの実行時には、プロシージャは、実際に実行しているユーザーに関係なく、その所有ユーザーのセキュリティ・ドメインで動作します。したがって、ユーザーはプロシージャを実行する場合、参照オブジェクトに対する権限を必要としません。プロシージャの所有者には参照オブジェクトに対して必要なオブジェクト権限がなければならないため、プロシージャのユーザーに付与される権限の数は少なくなり、データベース・アクセスの制御が強化されます。

ストアド・プロシージャの所有者の現行の権限は、そのプロシージャを実行する前に必ずチェックされます。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：トリガーの実行についても同じことが適用されます。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。起動されたトリガー・アクションの文は、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。

プロシージャを作ったり変更したりするのに必要なシステム権限

プロシージャを作るに、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なければなりません。ロールを介してはそれらの権限を取得できません。それには、作ろうとしてい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。権限が明示的に付与されていても、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えます。

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）プロシージャおよび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修正したりできます。パッケージの構成体に対して個別には EXECUTE 権限を付与できません。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およびファンクション、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 つの選択肢を考慮するとよいでしょう。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ります。

```
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;
```

上記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、次の文を使って、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、1つのパッケージ本体の中に4つのプロシージャを定義します。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って、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;
```

この方法を使うと、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、宣言される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;
```

ロール

Oracle では、ロールを使うことにより、簡単かつ制御された権限管理を実現できます。「ロール」は、関連する権限の名前付きグループであり、ユーザーまたは他のロールに対して付与されます。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。しかし、ロールは、アプリケーション開発者が使う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。プロシージャに関する制限の詳細は、26-13 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。

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簡単になります。

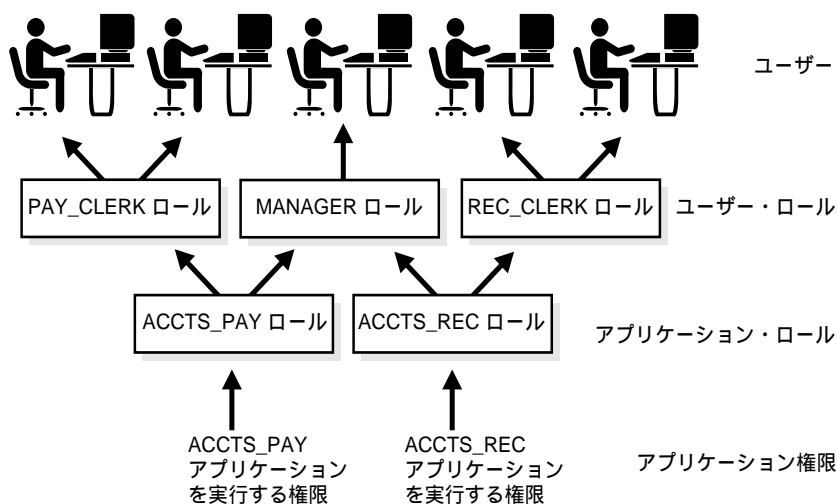
権限管理に要する労力の削減。	複数のユーザーに対して同一の権限セットを明示的に付与するかわりに、関連するユーザー・グループのための権限をまとめて1つのロールに付与しておき、そのグループの各メンバーにはそのロールを付与するだけで済みます。
動的な権限管理	あるグループの権限を変更しなければならない場合、そのロールの権限を修正するだけで済みます。グループのロールを付与した全ユーザーのセキュリティ・ドメインは、そのロールに対して加えられる変更を自動的に反映します。
権限の選択的な可用性	あるユーザーに付与したロールは、選択的に使用可能または使用禁止にできます。この機能によって、どのような状況でもユー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ようとしたときに、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って保護できます。正しいパスワードを指定するとロールが使用可能になる、というアプリケーションを作れます。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。

追加情報：アプリケーションからロールを使用可能にする方法は、『Oracle8 Server アプリケーション開発者ガイド』に記載されています。

ロールの一般的な使用方法

一般に、ロールは、データベース・アプリケーションに対する権限の管理、またはユーザー・グループに対する権限の管理のどちらかの目的で作られます。図 26-1 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 26-1 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することになります。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て、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合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られるロールです。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る。
- ロールには、別のロールを付与できる。ただし、ロールをそのロール自体に付与したり、循環的に付与したりはできません（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る。
- 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る。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、権限を選択的に使えます。
- 間接的に付与されたロール（ロールに対して付与されたロール）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できる。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入っている、間接的に付与された全ロールは暗黙のうちに使用可能にされます。

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したりするには、次の方法があります。

- Oracle Enterprise Manager の「システム権限/ロールの許可」ダイアログ・ボックスおよび「システム権限/ロールの取消し」ダイアログ・ボックス
- SQL コマンド 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上のオプションを使います。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・サービスを使うことによって、ユーザーに対して付与したり取り消したりできます。

追加情報：ロールを管理する方法の詳細は、『Oracle8 Server 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、他のユーザーの任意のロール（グローバル・ロールを除く）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なので、付与するときは注意が必要です。

追加情報：グローバル・ロールの詳細は、『Oracle8 Server 分散システム』を参照してください。

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したりできます。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。

ロールの命名

各ロール名はデータベース内で一意でなければならないため、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、ロールはどれかのスキーマに「含まれている」わけではありません。そのため、ロールを作っ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。

ある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、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は使用可能にし、別のユーザーに対しては使用禁止にする、ということもできます。)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。

名前付き PL/SQL ブロックとロール

名前付きPL/SQLブロック（ストアド・プロシージャまたはファンクション、トリガー）で使用禁止にされたすべてのロールには、次のことが当てはまります。

- PL/SQLブロックで参照されているオブジェクトを所有しないユーザー・スキーマ内で作成されている。
- PL/SQLブロックの所有者以外のユーザーとして実行できる。

しかし、無名PL/SQLブロックは、使用可能なロールを介して付与された権限に基づいて実行されます。

SESSION_ROLES ビューは、現在使用可能にされているすべてのロールを示します。名前付き PL/SQL ブロックが SESSION_ROLES を問い合わせる場合、問合せは行を戻しません。

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るには、CREATE VIEW または CREATE ANY VIEW システム権限、それに加えてその表に対する SELECT オブジェクト権限または SELECT ANY TABLE システム権限が必要です。

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に対する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、CREATE PROCEDURE の各権限）
- － スキーマ・オブジェクト権限（表に対する ALTER、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えません。

- DDL 文を発行するために必要な 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、それらの権限を使って別のユーザーの表に基づくビューを作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る。
- 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るが、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る。
- DEPT 表に対する SELECT オブジェクト権限が直接付与されている。

以上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 文を発行できる。
- このユーザーには EMP 表の CREATE VIEW 権限と SELECT 権限（いずれもロールを介して付与）があるが、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れない。ビューを作ろうとすると、エラーになる。
- CREATE VIEW 権限（ロールを介して付与）と DEPT 表の SELECT 権限（直接付与）があるため、DEPT 表のビューは作れる。

事前定義済みのロール

ロール CONNECT、RESOURCE、DBA、EXP_FULL_DATABASE、IMP_FULL_DATABASE は、Oracle データベースに対して自動的に定義されます。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用意されており、Oracle データベース内の他のロールと同じ方法で修正できます。

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・システムを使って、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。

追加情報：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・システム別の Oracle の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う場合は、必要なすべてのロールが分散（リモート）セッションのデフォルト・ロールとして設定されているように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。

追加情報：分散データベース環境の詳細は、『Oracle8 Server 分散システム』を参照してください。

ロール

You can observe a lot by watching.

Yogi Berra

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。

- 監査の基礎知識
- 文監査
- 権限監査
- スキーマ・オブジェクト監査
- 文および権限、スキーマ・オブジェクトの監査の対象を限定する

追加情報: Trusted Oracleを使っている環境での監査の詳細は、Trusted Oracleのマニュアルを参照してください。

監査の基礎知識

「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われます。

- 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、そのデータベースに対するすべての接続と、そのデータベースにあるすべての表からの行の削除（成功したものと失敗したもの）をすべて監視できます。
- 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、更新された表または実行された論理I/Oの回数、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

一般に、Oracle は次の 3 つのタイプの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、文のタイプだけに関して実行する 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、対象にする権限の使用方法だけを監査するので、文監査よりも対象が限定されています。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だけを監査します。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ています。

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したりできます。次のことができます。

- 成功した文の実行の監査、失敗した文の実行の監査、またはその両方
- ユーザー・セッションごとに、または文が実行されるたびに文監査を実行

- 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の監査

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。この表の情報を使いやすくするため、複数の事前定義済みのビューが提供されています。

追加情報: これらのビューの作成方法と使用方法の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

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、監査証跡にはさまざま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いつも記録されます。

- 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
- 使われたシステム権限 (Trusted Oracle の MAC 権限を含む)
- ユーザー・セッションのラベル (Trusted Oracle の場合だけ)
- アクセスされたスキーマ・オブジェクトのラベル (Trusted Oracle の場合だけ)

オペレーティング・システム監査証跡はコード化されており読めませんが、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。

使われた権限 この操作の実行に使わ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。これらのコードのリストは、『Oracle8 Server エラー・メッセージ』に記載されています。

監査のメカニズム

ここでは、Oracle の監査機能で使われているメカニズムについて説明します。

監査レコードはどのような場合に生成されるか

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。

追加情報：監査を使用可能または使用禁止にする方法の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

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。

注意：SQL 文処理のさまざまなフェーズや共有 SQL に精通していない場合は、これ以降の説明に対する基礎知識として、第 14 章「SQL と PL/SQL」を参照してください。

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ので、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、回復など）を実行できます。

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必ず記録します。

インスタンスの起動	インスタンスを起動した OS 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、データベース監査が使用可能だったか使用禁止だった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 OS 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ので、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起動して監査されないアクションを実行する、ということとはできないようになっています。
インスタンスの停止	インスタンスを停止した OS 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 OS 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 に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入れられます。

追加情報：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・システム別の Oracle のマニュアルを参照してください。

監査オプションはいつ有効になるか

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、現行のセッションを終了し、新しいセッションを作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、現行セッションですぐに有効になります。

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だけ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ので、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。分散データベースとデータベース・リンクの詳細は、第 30 章「分散データベース」を参照してください。

OS 監査証跡に対する監査

Oracle と Trusted Oracle では、オペレーティング・システムによって Oracle がオペレーティング・システムの監査証跡が使えるようにされている場合に、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れます。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は、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ます。

追加情報：この機能が使用しているオペレーティング・システムでインプリメントされているかどうか調べるには、プラットフォーム別の Oracle のマニュアルを参照してください。

Oracle と Trusted 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が入っているオペレーティング・システム・ファイル）に監査レコードを記録できない場合でも、常に監査されている特定のアクションは継続できます。引き続き実行できます。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。

OS 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ってい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、このような状況が絶対に発生しないようにするのに十分な情報と警告が管理者に提示されます。しかし、データベースの監査証跡を使うように監査を構成すれば、その危険性をなくせ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からです。

文監査

文監査とは、文の関連グループに対する選択的な監査のことです。文は、次の 2 つの項目に分類できます。

- 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 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、表またはビュー、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/SHOT 文を監査します）。

文監査では、監査の対象範囲を広くし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だけを監査したりできます。

権限監査

権限監査では、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システム権限を使っ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、いつでも、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のための十分な許可である場合、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つだけが生成されます。たとえば、文オプションのTABLEとシステム権限のCREATE TABLEを両方監査すると、表が作られるたびに監査レコードが1つだけ生成されます。

権限監査のそれぞれのオプションでは、特定のタイプの文だけが監査され、関連する一連の文が監査されるわけではないので、権限監査の対象は文監査よりも限定されます。たとえば、文監査オプションのTABLEではCREATE TABLEおよびALTER TABLE、DROP TABLE文が監査されますが、権限監査オプションのCREATE TABLEではCREATE TABLE権限だけが監査されます。これは、CREATE TABLE権限を必要とするのがCREATE TABLE文だけであるためです。

文監査と同様、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だけを監査したりできます。

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・プロシージャおよびファンクション、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、シノニムを参照する文は、しかし、実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。

追加情報：『Oracle8 Server SQLリファレンス』の「AUDIT（スキーマ・オブジェクト）」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・ファンクションおよびパッケージ、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うと、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従属します。ビューやプロシージャを使っ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)に従属します。

次の一連の SQL 文について考慮してみましょう。

```
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つは実表 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。実表の SELECT 監査オプションが使用可能になっていないので、実表 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っ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、キャッシュの中のスキーマ・オブジェクトが無効になるので、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。

上記の例の続きとして、EMP 表に対する SELECT 文の監査をオフにすると、EMP_DEPT ビューを使っても EMP 表の監査レコードは生成されなくなります。

文および権限、スキーマ・オブジェクトの監査の対象を限定する

Oracle では、文、権限、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 文の成功した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（文監査と権限監査だけ）

成功した文の実行と失敗した文の実行の監査

文および権限、スキーマ・オブジェクトの監査では、成功した文の実行、失敗した文の実行、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、監査する文が完全に成功しなかったとしても、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 文を実行したにもかかわらず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だけです。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ったにもかかわらず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）。

AUDIT コマンドには、次のどちらかのオプションを指定できます。

- WHENEVER SUCCESSFUL オプション。監査対象の文の成功した文だけを監査する場合。
- WHENEVER NOT SUCCESSFUL オプション。監査対象の文の失敗した文の実行だけを監査する場合。
- 上記のどちらのオプションも使わない。監査対象の文の成功した文と失敗した文の両方の実行を監査する場合。

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 コマンドの BY SESSION オプションと BY ACCESS オプションの相違点について説明します。

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の中で、ユーザーおよびスキーマ・オブジェクト当たり 1 つの監査レコードだけ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その接続を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 に設定する。

- JWARD から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する。
- SWILLIAMS から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する。

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 に設定する。
- JWARD から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する。

この場合、監査証跡には 2 件 (ユーザーが 1 つのセッション中に SELECT 文を発行した各オブジェクトごとに 1 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 オプションを使うと、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。ですから、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には、次のようなものがあります。

- 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っ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によって影響を受けないことに注意してください。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生成します。

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 に設定する。
- JWARD から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する。

- SWILLIAMS から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する。

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 コマンドには、BY SESSION か BY ACCESS のどちらかを指定できます。しかし、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のどちらかを実行できます。特定のユーザーに限定することで、生成される監査レコードの数は最小限に抑えられます。

追加情報：ユーザー別の監査の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

例

表やビューを問い合わせたり更新したり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;
```

文および権限、スキーマ・オブジェクトの監査の対象を限定する

データベースの回復

These unhappy times call for the building of plans...

Franklin Delano Roosevelt

この章では、データベースの回復時に使われる構造、およびバックアップおよびリカバリ（回復）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。

- データベース回復の基礎知識
- データベースの回復で使われる構造
- ロールフォワードとロールバック
- Recovery Manager
- 回復の平行実行
- 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

追加情報：バックアップとリカバリの構造を作ってメンテナンスするのに必要な手順の詳細は、『Oracle8 Server バックアップおよびリカバリ』を参照してください。

データベース回復の基礎知識

データベース管理者のおもな責任の1つは、ハードウェア、ソフトウェア、ネットワーク、またはプロセスで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を与える場合、通常は速やかにデータベースを回復し、通常の操作に戻る必要があります。回復作業では、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済むようにしなければなりません。

回復プロセスは、発生した障害のタイプおよびその障害の影響を受けた構造、実行する回復のタイプに応じて異なります。消失したり破損したり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回復できます。データが失われている場合は、回復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 (回復) の操作を簡単にできるようにするユーティリティです。28-10 ページの「Recovery Manager」を参照してください。

追加情報: Recovery Manager の詳細とデータ消失からの回復方法は、『Oracle8 Server バックアップおよびリカバリ』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 が影響を受けたりすることがあります。この後、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、回復が自動的に実行されることがあります。また、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、ユーザー・エラー (表を誤って削除してしまうなど) を防止できません。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・エラーが少なくなります。また、管理者が事前に効果的な回復計画を準備しておけば、さまざまなタイプのユーザー・エラーの回復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、データで満杯になっている、つまり表が完全に満杯であるとします。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、アクションや回復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、ユーザー、サーバー、またはデータベース・インスタンスのバックグラウンド・プロセスの障害です（接続の切断やプロセス終了の異常など）。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トランザクションをロールバックし、そのプロセスが使っ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、自動的に回復されます。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、インスタンスは正しく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、電話回線など）を使ってクライアント・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・システムを形成したりしている場合に、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が割り込まれることがあります。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まだ解決されていない分散トランザクションを、自動的に解決します。分散データベース・システムの詳細は、第 30 章「分散データベース」を参照してください。

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（SGA とバックグラウンド・プロセス）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回復

インスタンス回復は、インスタンス障害が発生する前のトランザクション一貫性のある状態にデータベースを復旧する作業です。オンライン・バックアップ時にインスタンス障害が起きた場合は、メディア回復が必要になります。その他のすべての場合のインスタンス障害は、ほとんどが自動的に回復されます。たとえば、Oracle Parallel Serverを使っている場合、他のインスタンスによってインスタンス回復が実行されます。シングル・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インスタンス回復が実行されます（新しいインスタンスにマウントされ、オープンされる）。必要なら、マウント状態からオープン状態に移する時点で、自動的にインスタンス回復がトリガー起動されます。

インスタンス回復は、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ないが、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・ファイルを回復する。
2. データベースをオープンする。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、キャッシュ回復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。まだ回復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。この機能を「高速ウォームスタート」といいます。
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が入っているロールバック・セグメントを PARTLY AVAILABLE とマークする。
4. SMON による回復の一部として、デッド・トランザクションを回復する。
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する。

増分チェックポイント機能

増分チェックポイント機能により、クラッシュ回復とインスタンス回復のパフォーマンスが向上します（ただしメディア回復のパフォーマンスは向上しません）。増分チェックポイントは、REDO スレッド（ログ）内のクラッシュまたはインスタンス回復を開始する必要がある位置を記録します。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みバッファによって決まります。増分チェックポイント情報は、通常の処理中に、最小限のオーバーヘッドまたはオーバーヘッドなしで定期的にメンテナンスされます。

回復のパフォーマンスは、クラッシュの前にデータベースに書き込まれていなかったバッファの数に大まかに比例します。クラッシュまたはインスタンス回復のパフォーマンスは、初期化パラメータ `DB_BLOCK_MAX_DIRTY_TARGET` を設定することで変えることができます。この初期化パラメータは、任意の時点でインスタンスのバッファ・キャッシュ内に存在できる使用済みバッファの数の上限を指定します。したがって、バッファ・キャッシュが非常に大きい場合や、クラッシュ/インスタンス回復の持続時間に厳しい制限がある場合に、回復時間に影響を与えることができます。このパラメータの値が小さいほど、より多く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、回復する必要があるブロック数が少なくなるので、回復のパフォーマンスは向上します。

増分チェックポイント情報は、他のチェックポイント（ログ・スイッチ・チェックポイントやユーザー指定チェックポイントなど）に影響を与えずに、Oracle Serverによって自動的にメンテナンスされます。つまり、増分チェックポイント機能は、インスタンスで発生する他のチェックポイントからは独立しています。

読み込み専用表領域とインスタンス回復

インスタンス回復時には、読み込み専用ファイルの回復は必要ありません。起動時の回復では、オンラインの読み込み専用ファイルそれぞれに関してメディア回復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、読み込み専用になる前に作られたバックアップからは復旧されなかったということです。読み込み専用の表領域を、その表領域が読み込み専用になる前に作られたバックアップから復旧した場合は、メディア回復が完了するまでその表領域にはアクセスできません。

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、REDO ログ・ファイル、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回復の方法は、関係しているファイルによって異なります。

追加情報：メディア回復の詳細は、『Oracle8 Server バックアップおよびリカバリ』を参照してください。

メディア障害がデータベース操作に与える影響

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、制御ファイルなど、Oracle データベースの操作に必要なさまざまなタイプのファイルに影響を与えます。

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、単にそのファイルの 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 がその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。

データ・ファイルに影響を与えるメディア障害は、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、または読み取れないことを示す 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とき、満杯になったオンライン REDO ログ・ファイルがアーカイブされている場合は、Oracle はエラーを DBWn トレース・ファイルに戻し、そのデータ・ファイルを自動的にオフラインにします。自動的にオフラインにされるのは、書き込み不能のデータ・ファイルだけです。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にされません。そのかわりにエラーが戻され、データベースが Oracle により停止されます。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・セグメントが入っている表領域のデータ・ファイルはオンラインでなければなりません。

Oracle がデータ・ファイルに書き込めず、データが満杯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、DBWn バックグラウンド・プロセスと現行の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・ファイルを使ったインスタンス回復が実行可能で、この場合はインスタンスを再起動できます。しかし、データ・ファイルが永続的に破壊され、アーカイブされていない場合は、最新のバックアップからデータベース全体を復旧する必要があります。

データベースの回復で使われる構造

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、データベース回復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データベース回復を開始する際、Oracle はこのバックアップ・ファイルを使って破損したデータ・ファイルまたは制御ファイルを復旧します。

Oracle では、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。

追加情報：『Oracle8 Server バックアップおよびリカバリ』を参照してください。

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、データベース回復の一環として、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 ログとアーカイブ済み 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。オンライン REDO ログは、Oracle バックグラウンド・プロセス LGWR とともに機能し、対応付けられているインスタンスによってなされたすべての変更をすぐに記録します。オンライン REDO ログは 2 つ以上の事前割当て済みファイルで構成され、これらのファイルが循環方式で再利用されながら、進行中のデータベース変更が記録されます。

アーカイブ済み（オフライン）REDO ログ

必要に応じて、オンライン REDO ログが満杯になったらそのファイルをアーカイブするように、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、一意に識別され、アーカイブ済み REDO ログ・ファイルを構成します。事前割当て済みのオンライン REDO ログ・ファイルを繰り返し再利用して最新のデータベース変更を格納する一方で、満杯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、より広範囲なデータベース回復操作で使うための過去の REDO ログ情報を保存します。

詳細は、28-14 ページの「データベースのアーカイブ・モード」を参照してください。

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われます。一般に、データベースのロールバック・セグメントには、実行中のトランザクション（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、元のデータ値が格納されます。

ロールバック・セグメントの情報は、特に、データベースの回復時に 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」ために使われます。このため、データベース回復が必要な場合に、データが一貫した状態に戻るのには、ロールバック・セグメントを使って、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、制御ファイルの所定の状態情報（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）に従って、インスタンス回復またはメディア回復を実行します。

詳細は、28-18 ページの「制御ファイル」を参照してください。

ロールフォワードとロールバック

SGA 中のデータベース・バッファは、最低使用頻度アルゴリズムを使って、必要に応じてディスクに書き込まれます。DBWn プロセスはこのアルゴリズムを使ってデータベース・バッファをデータ・ファイルに書き込むので、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たりする可能性があります。

インスタンス障害が発生すると、次のような 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・ファイルにだけ記録されている。このため、REDO ログには、回復時にデータベースに再適用する必要のある変更が含まれています。
- REDO ログには、コミットされなかったデータも入っている場合があるので、REDO ログによって適用されたコミットされていないトランザクションの変更（以前の REDO ログによって適用されたコミットされていない変更も含む）はデータベースから消去する必要がある。

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回復を実行します。つまり、REDO ログを使ってロールフォワードし、ロールバック・セグメントを使ってロールバックします。

REDO ログとロールフォワード

REDO ログは、データ、索引、ロールバック・セグメントなどのデータベース・バッファに対するすべての変更内容が、コミットされているかどうかに関係なく記録される、オペレーティング・システム・ファイルのセットです。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回復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ので、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。

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だけの数の 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・ファイルが使われ、さらにアーカイブ済み REDO ログ・ファイルが使われることもあります。

ロールフォワードが終わると、データ・ブロックには、REDO ログに記録されていたすべてのコミットされた変更とコミットされなかった変更が復元されます。

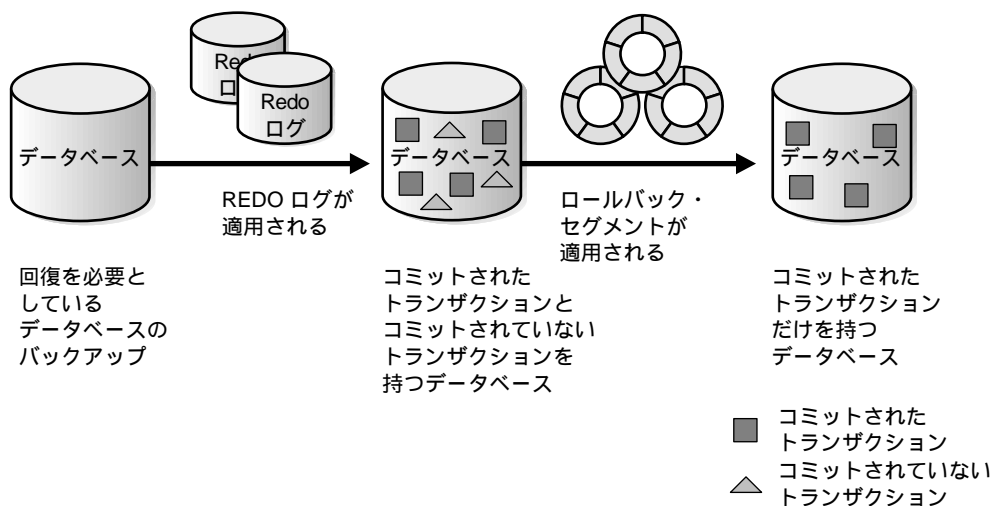
ロールバック・セグメントとロールバック

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のあるデータベース・アクションを記録します。ロールバック・セグメントは、データベースの回復時にロールフォワード・フェーズで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・セグメントを使うことになります。ロールバック・セグメントは、コミットされていないのに REDO ログに記録され、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います。このプロセスを「ロールバック」といいます。

図 28-1 に、あらゆるタイプのシステム障害からの回復に必要な 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 28-1 基本的な回復ステップ： 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回復し、トランザクションに必要な行ロックを取得できます。この機能を高速トランザクション・ロールバックといいます。

Recovery Manager

「Recovery Manager」は、データベース・ファイルのバックアップを作り、バックアップからファイルをリストア（復旧）またはリカバリ（回復）するプロセスを管理するユーティリティです。

追加情報: Recovery Managerの詳細は、『Oracle8 Serverバックアップおよびリカバリ』を参照してください。

リカバリ・カタログ

RecoveryManager は、「リカバリ・カタログ」というリポジトリを維持しています。リカバリ・カタログには、バックアップ・ファイルとアーカイブ済みログ・ファイルに関する情報が入ります。Recovery Managerは、このリカバリ・カタログを使って、復元操作と媒体回復の両方を自動化します。

リカバリ・カタログには、次の情報が入っています。

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

リカバリ・カタログを保持しているのは、Recovery Manager だけです。データベース・サーバーがリカバリ・カタログを直接にアクセスすることは絶対にありません。Recovery Manager は、バックアップ・データ・ファイルのセット、アーカイブ済み REDO ログ、およびデータ・ファイルのコピーに関する情報をリカバリ・カタログに波及して、長期間保存します。

回復の実行時に、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、回復を指定された入力ファイルに対してさまざま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ります。リカバリ・カタログは Oracle データベースに格納されるので、Recovery Manager を使ってリカバリ・カタログのバックアップを作れます。

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、現行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・カタログを使わない操作

リカバリ・カタログを使うことは、必須ではありません。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ので、Recovery Manager では制御ファイルだけを使う操作モードをサポートしています。

この操作モードは、回復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。

この操作モードでは、表領域の特定の時点への回復はサポートされていません。

パラレル化

Recovery Manager では、非ブロック化UPIを使うことによって、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、データベース・ファイルの別々のセットを処理するものでなければなりません。

注意：Oracle8 では、Recovery Manager チャンネルを一度に 1 つしか割り当てることができないので、1 つのストリームに対する並行性は制限されます。Oracle8 Enterprise Edition を使うと、並行性が無制限になります。Oracle8 と Oracle8 Enterprise Edition で使用可能な機能の詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

backup および copy、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だけです。

- 1 つ以上の順次 I/O デバイスのリスト
- バックアップまたはコピー、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。並列性が活用されるのは、単一のコマンドのコンテキスト内に限られます。それで、10 個のデータ・ファイル・コピーが必要な場合は、個々の 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 回発行するほうがよいでしょう。

レポートの生成

report および list コマンドを使うと、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のはどのファイルか？
- しばらくバックアップを作っていないのはどのファイルか？
- どのバックアップ・ファイルを削除できるか？

定期的に report need backup コマンドと report unrecoverable コマンドを使うことによって、回復を実行するのに必要なバックアップを入手できるようにし、回復が妥当な時間内で実行されるようにできます。report deletable コマンドは、冗長であるため、または recover コマンドでは使えなくなったため、削除可能なバックアップ・セットとデータ・ファイル・コピーのリストを表示します。

データ・ファイルは、次の場合に「回復不能」とみなされます。

- データ・ファイルの唯一の既存のバックアップは、そのデータ・ファイルを回復するのに必要なアーカイブ・ログがリカバリ・カタログにないか、必要なログをリカバリ・カタログが認識していない場合に、使えなくなる。
- データ・ファイル内のスキーマ・オブジェクトに対して、ログに記録されない操作が行われた場合。

(バックアップがないデータ・ファイルは、回復不能とはみなされません。そのファイルを作っ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 コマンドを使って、そのようなデータ・ファイルを回復できます。)

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。このコマンドを使うと、どのバックアップまたはコピーが使用可能かがわかります。

- 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

回復の平行実行

回復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ので、インスタンス回復またはメディア回復では、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。シリアル回復の場合は、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。平行回復を使うと、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: Oracle8 では、Recovery Manager での並行性が制限されます。Oracle8 Enterprise Edition を使うと、並行性が無制限になります。Oracle8 と Oracle8 Enterprise Edition で使用可能な機能の詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

平行回復は、複数の Oracle Enterprise Manager セッションを生成し、それぞれのセッションで別々のデータ・ファイル・セットに対する RECOVERDATAFILE コマンドを発行することによって手動で実行できます。しかしこの方法を使うと、それぞれの 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。

インスタンス回復とメディア回復は、RECOVER コマンドに対する初期化パラメータまたはオプションを指定することにより、自動的に平行化できます。Oracle は、1 つのプロセスを使ってログ・ファイルを順次読み込み、REDO 情報を複数の回復プロセスに送ります。回復プロセスは、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。これらの回復プロセスは、Oracle によって自動的に起動されるので、回復を実行するために 2 つ以上のセッションを使う必要はありません。

平行回復を活用できる状況

一般に、平行回復は、いくつか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回復する際の回復時間を短縮するのに最も効果的です。多くの異なるディスク・ドライブにある多くのデータ・ファイルのクラッシュ回復（インスタンス障害が発生した後に行う回復）とメディア回復を行うのは、平行回復のよい例です。

平行回復で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平行回復を使うと、回復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、平行回復を使っても回復時間はわずかに短縮されるだけです。

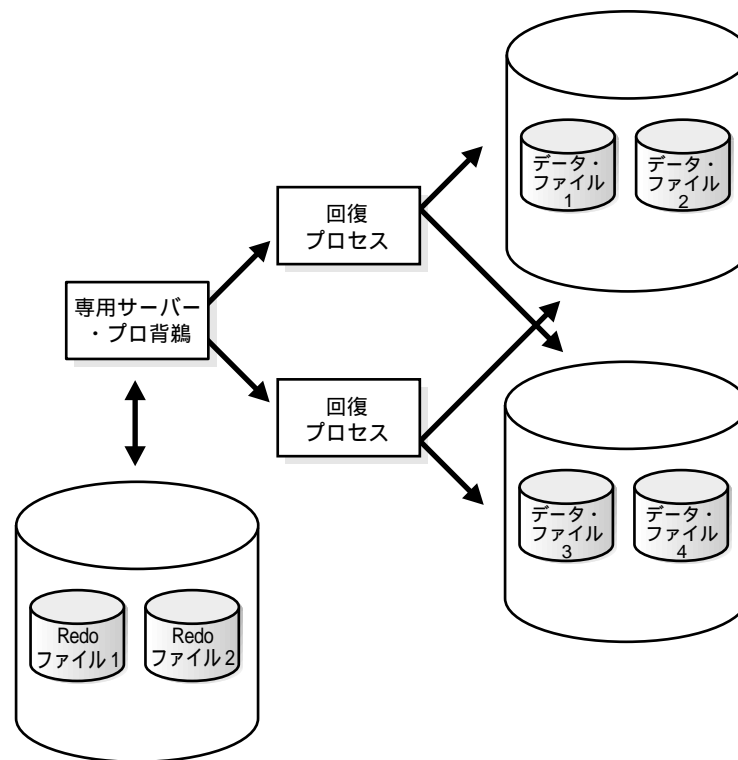
追加情報: システムで非同期 I/O がサポートされているかどうかを判別するには、使っているオペレーティング・システムのマニュアルを参照してください。

回復プロセス

典型的な平行回復の状況では、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、回復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 つ以上の回復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 28-2 に、典型的な平行回復セッションを示します。

図 28-2 典型的なパラレル回復セッション



ほとんどの状況では、回復が必要なデータ・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つの回復セッションと1つか2つの回復プロセスがあれば十分です。回復は、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なので、必要な回復プロセスの数は、回復で使うディスク・ドライブの数に依存します。一般に、パラレル回復のパフォーマンスをシリアル回復よりも高くするには、8個の回復プロセスが必要です。

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（メディア回復が使用不可能）と ARCHIVELOG モード（メディア回復が使用可能）の2つのモードで操作できます。

NOARCHIVELOG モード（メディア回復が使用不可能）

データベースを NOARCHIVELOG モードで使う場合、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には、満杯のグループをアーカイブする必要はないことが示されます。このため、満杯のグループが非アクティブになり、ログ・スイッチ時のチェックポイントが完了すると、LGWR プロセスはこのグループを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だけで、ディスク（メディア）障害の場合は保護できません。インスタンス回復で使え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）だけで、それ以外の情報は使えません。

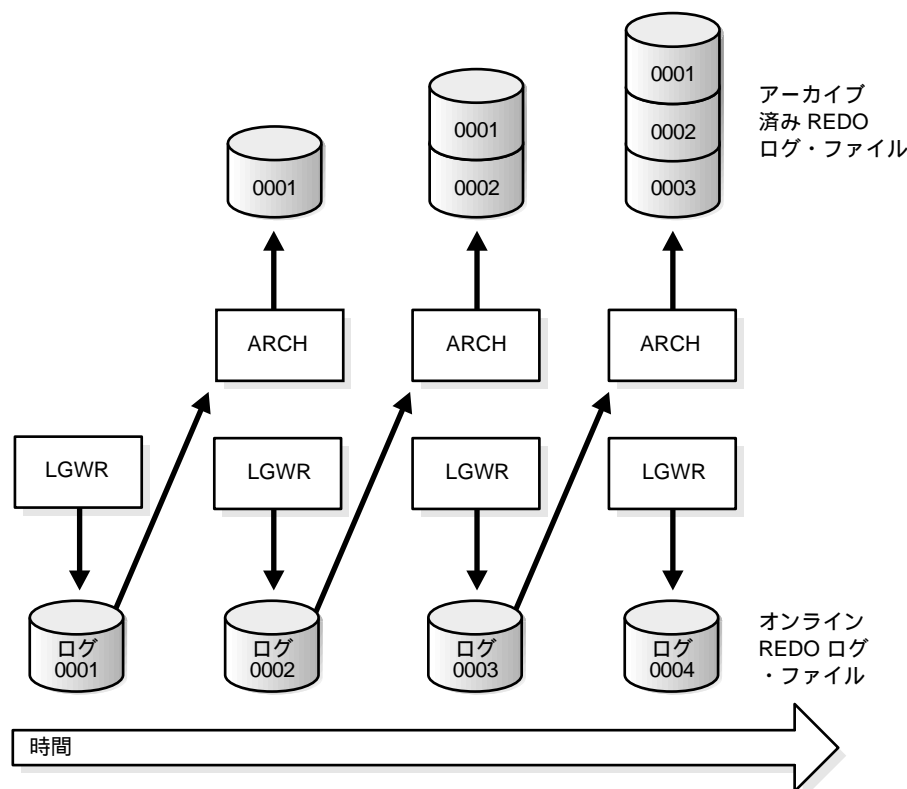
ARCHIVELOG モード（メディア回復が使用可能）

データベースを ARCHIVELOG モードで使う場合、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には、満杯のオンライン 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が示されます。満杯になったグループは、ログ・スイッチが発生する（グループが非アクティブになる）と、アーカイブを実行するプロセスに対して、すぐに使えるようになります。アーカイブを実行するプロセスは、ログ・スイッチのチェックポイントが完了するのを待たなくても、アクティブでないグループにアクセスして、アーカイブできます。

図 28-3 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われるか、また、満杯のグループをアーカイブするプロセス（たとえば、この図では ARCH）によってアーカイブ済み 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済み REDO ログに永続的に保存されるので、ディスク障害とインスタンス障害から完全に回復できます。

図 28-3 ARCHIVELOG モードでオンライン REDO ログ・ファイルを使う



自動アーカイブと ARCH (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 ARCH (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とき、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。したがって、自動アーカイブを使うと、データベース管理者が満杯のグループを追跡して記録し、アーカイブするという作業を手動で実行する必要がなくなります。アーカイブ済み REDO ログがあ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っているのは、このような便宜上の理由のためです。

インスタンスの起動時に、LOG_ARCHIVE_START 初期化パラメータを設定して自動アーカイブを要求すると、インスタンスの起動中に ARCH が起動されます。このように設定しないと、インスタンスの起動中に ARCH は起動されません。

しかし、データベース管理者は、自動アーカイブをいつでも対話的に開始したり停止したりできます。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、ARCH バックグラウンド・プロセスが作られます。自動アーカイブを一時的にオフにした後でオンに切り替えても、インスタンスの持続期間中は ARCH は残留します。

ARCH は必ず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、ARCH は、満杯のグループがアクティブでなくなると、それらのグループを自動的にアーカイブします。自動アーカイブの毎回の記録が、ARCH プロセスによって ARCH 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。

ARCH があるグループをアーカイブしようとしてエラーを検出しても（アーカイブ先が無効または満杯であったなど）、ARCH は引き続きそのグループをアーカイブしようとします。また、エラーは ARCH 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、アーカイブされていなくても最終的にはすべてのオンライン REDO ログ・グループが満杯になり、LGWR が使えるグループがなくなるため、システムは停止します。それで、問題が検出されたら、問題を解決（アーカイブ先を変更するなど）して ARCH がアーカイブを続行できるようにするか、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満杯になったものは、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、すべての満杯のグループをアーカイブする責任があります。

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います。さらに、自動アーカイブを使用禁止にし、手動アーカイブを十分な速さで実行しないとすると、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（アーカイブ済み 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、または満杯になったなど）、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する。
- 標準以外の方法でグループをアーカイブする（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みファイルを失ったり、破損したりした場合にグループを再アーカイブする。

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 ARCH 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 によって頻繁に更新されるので、データベースがオープンされている間は書き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 つのデータベースにだけ対応付けられます。

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、データベースに関する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 だけです。データベース管理者やエンド・ユーザーがデータベースの制御ファイルを編集することはできません。

制御ファイルには、たとえば次の情報が格納されています。

- データベース名
- データベースを作っ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
- 現行のログ順序番号
- 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られます。データベース名は、DB_NAME 初期化パラメータに指定した名前か、CREATE DATABASE 文で使った名前のどちらから取られます。

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、削除されるたびに、制御ファイルが更新され、この物理的な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、オープンするべきデータ・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する。

- データベースの回復が必要な場合に必要となる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する。

したがって、データベースの物理構造を変更したら、すぐに制御ファイルのバックアップを作ってください。

追加情報：データベースの制御ファイルをバックアップする方法の詳細は、『Oracle8 Server バックアップおよびリカバリ』を参照してください。

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。チェックポイントが開始すると、制御ファイルはオンライン REDO ログに入力する必要のある次のエントリに関する情報を記録します。この情報は、データベースの回復時に使われ、オンライン REDO ログ・グループのこのポイントより前に記録されている REDO エントリはどれもデータベースの回復には必要でない（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。

制御ファイルの多重化

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めます。

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、現行のインスタンスがエラーになります。しかし、制御ファイルのコピーが他のディスクに保存してあれば、データベースを回復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、重大な問題になるので、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を永続的に失った（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、メディア回復が必要になります。しかし、現行の制御ファイルを使わず、制御ファイルの古いバックアップを使うことになるのであれば、メディア回復は容易ではありません。したがって、可能な限り、データベースごとに多重化した制御ファイルを作り、それぞれのコピーを異なる物理ディスクに格納するようにしてください。

データベースのバックアップ

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っておくことは、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。

以降の部分では、使用可能なさまざまなタイプのバックアップの概念と、異なるリカバリ（回復）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。

追加情報：『Oracle8 Server バックアップおよびリカバリ』には、データベース・バックアップ実行のガイドラインとともに詳細が記述されています。

全体データベース・バックアップ

「全体データベース・バックアップ」は、Oracle データベースを構成する、すべてのデータ・ファイルと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。全体データベース・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

データベースが正常に停止すると、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うと、直前の全体バックアップの時点でのデータを回復できます。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、バックアップを回復（オンラインとアーカイブ済みの REDO ログ・ファイルを使う）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、どんなタイプのメディア回復の方式の場合にも有効です。

- 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ってデータベースを「復旧」できる（回復ではない）。

データベースを現時点の状態に戻すためのアーカイブ済み REDO ログを使えないので、全体データベース・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回復できますが、これはあてにしないようにしてください。

- 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「データベース回復」の一部として使える。

必要なデータ・ファイルを全体バックアップから復旧した後は、アーカイブ済みのオンライン REDO ログ・ファイルと現在のオンライン REDO ログ・ファイルを適用して、復旧したデータ・ファイルを現時点の状態に戻すことにより、データベース回復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、一貫性のある全体バックアップしかありません。データベースが ARCHIVELOG モードで動作している場合、一貫性のある全体データベース・バックアップか一貫性のない全体データベース・バックアップのいずれかを使うことによって、ディスク障害からのデータベース回復作業の一部として、破損したファイルを復旧できます。

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も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だけです。アーカイブ済み REDO ログがあるので、部分バックアップから復旧されたデータ・ファイルは、回復手順の中でデータベースの残りの部分との一貫性をとれます。

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・ファイルだけを含めることができます。特定の個々のデータ・ファイルまたはひとまとまりのデータ・ファイルのバックアップは、データベースの他のデータ・ファイルおよびオンライン REDO ログ・ファイル、制御ファイルのバックアップとは別個に取れます。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れます。

オンラインとオフラインのどちらのデータ・ファイル・バックアップを作るかは、データ可用性の要件だけに依存しています。バックアップの対象となるデータが常に使用可能な状態でなければならない場合は、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ので、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る必要があります。

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をバックアップします。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、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。しかし、ディスク障害によってデータ・ファイルが破損し、部分的な回復、または特定の時点までの回復が必要な場合は、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う必要があります。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。

不完全な回復または Point-in-Time 回復の前に Recovery Manager を使って制御ファイルを回復する場合、Recovery Manager は最適なバックアップ制御ファイルを自動的に復旧します。

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う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られたスキーマ・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。

追加情報：『Oracle8 Server ユーティリティ』を参照してください。

読み込み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、読み込み専用表領域についてはバックアップを作れます。表領域を読み込み専用にしたらすぐに、その表領域のバックアップを作ってください。表領域が読み込み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る必要はありません。

読み込み専用表領域を読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。

読み込み専用表領域のデータ・ファイルをオンラインにしても、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、システムを中断させるその他の災害時に備えて、Oracle には「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、耐障害性と災害回復が特に重大な意味をもつサイトを対象にした機能です。

災害時回復の計画

システム障害やその他の災害から迅速かつ確実に回復できるようにするには、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っておく必要があります。スタンバイ・データベース・システムをインプリメントするにしても、単一データベース・システムを使うにしても、突然の障害に備えてどのように対処すべきかを計画しておかなければなりません。

スタンバイ・データベース

Oracle では、スタンバイ・データベース・システムをインプリメントして迅速な災害時回復を容易にする、信頼性の高い支援メカニズムが提供されています。この方式では、同じハードウェア構成の 2 次システムを利用します。2 次システムでは、アプリケーションにより、プライマリ・サイトでアーカイブされたログ・ファイルの一定のメディア回復状態が保持されています。万一、1 次システムで障害が発生した場合には、最小限の回復によってスタンバイ側（待機側）をアクティブにできるので、システムはすぐに使えるようになります。Oracle には、スタンバイ・システムを作ってメンテナンスすること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、それによって災害時回復機構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、いつでも回復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みログ情報が使われます。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。このため、スタンバイ・データベースは時間やトランザクション履歴の面で、常にプライマリ・データベースを後追いする形になります。

スタンバイ・データベースを置く物理的なハードウェアは、災害時回復システム専用にする必要があります。他のアプリケーションをこのハードウェアでは実行しないようにしてください。スタンバイ・データベースは災害時回復を意図して設計されているので、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・データベースには、電源障害やハードウェア障害に対する防護策としての役割だけでなく、火災や竜巻、地震などの物理的災害が発生した場合にデータを保護する役割もあります。

追加情報：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、『Oracle8 Server バックアップおよびリカバリ』を参照してください。

第VIII部

分散処理と分散データベース

第VIII部では、Oracle Server とデータベース・アプリケーションのための分散処理環境について、また分散データベース・アーキテクチャと複数ネットワークでのデータ・レプリケーション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29章「分散処理」
- 第30章「分散データベース」
- 第31章「データベースのレプリケーション」

We must try to trust one another. Stay and cooperate.

Jomo Kenyatta

この章では、分散処理について定義し、分散処理環境でOracle Serverとデータベースのアプリケーションがどのように機能するかを説明します。この資料の内容は、ほとんどすべてのタイプの Oracle データベース・システム環境に適用されます。

この章の内容は、次のとおりです。

- Oracle クライアント / サーバー・アーキテクチャ
- 分散処理

Oracle クライアント / サーバー ・ アーキテクチャ

Oracle データベース ・ システム環境は、データベース ・ アプリケーションとデータベースが、フロントエンド（クライアント）部とバックエンド（サーバー）部の 2 つの部分にわかれているクライアント / 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、マウスなどのポインティング ・ デバイスを使っ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したほうが効率的です。以降の部分では、Oracle クライアント / サーバー ・ アーキテクチャのさまざまな形態について説明します。

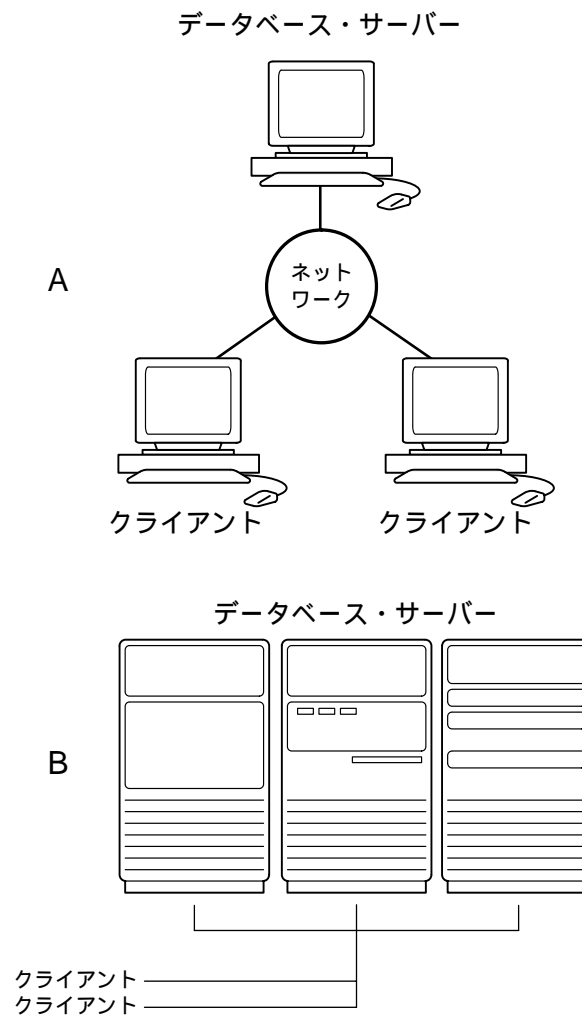
分散処理

「分散処理」とは、複数のプロセッサを使って個々の作業を処理することです。Oracle データベース ・ システムでの分散処理の例を、図 29-1 に示します。

- 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、それらのコンピュータをネットワークで接続しています。
- 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）が別のサーバーにあるデータベースにアクセスすることが必要となる場合があります。分散データベースでのクライアントとサーバーの詳細は、第 30 章「分散データベース」を参照してください。

図 29-1 クライアント / サーバー・アーキテクチャと分散処理



ネットワークの例（A）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 を使って通信しています。詳細は、29-4 ページの「Net8」を参照してください。

分散処理環境での Oracle クライアント / サーバー・アーキテクチャには、次のような利点があります。

- クライアント・アプリケーションではデータ処理は実行されない。そのかわり、クライアント・アプリケーションは、ユーザーからの入力を要求し、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）を使って表示します。
- クライアント・アプリケーションはデータの物理的な位置に依存しない。データを他のデータベース・サーバーに移動したり分散したりしても、アプリケーションはほとんど、またはまったく変更なしで機能を続行できます。
- Oracle は、基礎にあるオペレーティング・システムのマルチタスク機能と共有メモリー機能を活用できる。その結果、クライアント・アプリケーションに最高の同時実行性およびデータの整合性、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（たとえば、グラフィックスやマウスのサポートを提供する）、サーバーはデータの処理と格納用に最適化できる（たとえば大容量のメモリーとディスク領域を装着する）。
- ネットワーク化された環境では、費用のかからないクライアント・ワークステーションを使って、サーバーのリモート・データに効率的にアクセスできる。
- Oracle は、システムの拡大とともに必要に応じて「拡張」できる。複数のサーバーを追加して、データベース処理の負荷をネットワーク全体に分散させたり（「水平拡張」）、Oracle をミニコンピュータやメインフレームなどに移動して、より大規模なシステムのパフォーマンス上の利点を活用したり（「垂直拡張」）できます。どちらの場合も、Oracle にはシステム間での移植性があるので、すべてのデータとアプリケーションをほとんど、またはまったく変更なしで維持できます。
- ネットワーク化された環境の場合、共有データは、システムのすべてのコンピュータに格納されるのではなく、サーバーに格納される。このため、同時アクセスの管理が簡単で効率的になります。
- ネットワーク化された環境では、クライアント・アプリケーションからサーバーにデータベース要求を送るときに SQL 文を使える。サーバーが受け取った SQL 文はそのサーバーで処理され、その結果がクライアント・アプリケーションに戻されます。ネットワーク通信でやり取りされるのは要求と結果だけなので、ネットワーク通信量は最小限で済みます。

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および変更、共有、格納できるようにするための Oracle ネットワーク・インタフェースです。Net8 は、ネットワーク通信におけるプログラム・インタフェースの一部とみなされます。プログラム・インタフェースの詳細は、第 7 章「プロセスの構造」を参照してください。

Net8 は、幅広い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) を使って、 Oracle のための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式の規格のことです。ソフトウェアでインプリメントされます。
- ネットワークの場合、API は、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を使って、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では、PC LANでサポートされているものから最大規模のメインフレーム・コンピュータ・システムで使われるものに至るまで、主要なネットワーク・プロトコルはすべてサポートされています。

Net8 を使わないとすれば、アプリケーションの開発者は、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、手動でコーディングしなければなりません。ネットワークのハードウェアまたはトポロジ、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 を使えば、アプリケーションの開発者は、データベース・アプリケーション内でネットワーク通信をサポートする作業に携わらなくて済みます。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、アプリケーションのほうは変更なしで機能を続行できます。

Net8 の仕組み

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上位レベル・プロトコルまたはプログラム・インタフェースを介して Oracle に送信します。また、Oracle からの回答を取り込んでパッケージ化し、同じ上位レベルの通信メカニズムを介して Oracle Tools に送り返します。これらの機能はすべて、ネットワーク・オペレーティング・システムからは独立して実行されます。

追加情報：Oracle を実行するオペレーティング・システムによっては、データベース・サーバーの Net8 ソフトウェアにドライバ・ソフトウェアが含まれており、Net8 ソフトウェアによって Oracle バックグラウンド・プロセスが起動される場合があります。詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

Net8 の詳細は、『Oracle Net8 管理者ガイド』を参照してください。

分散データベース

Good sense is of all things in the world the most equally distributed, for everybody thinks he is so well supplied with it, that even the most difficult to please in all other matters never desire more of it than they already possess.

René Descartes: *Le Discours de la Methode*

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。

- Oracle の分散データベース・アーキテクチャ
- 異種間分散データベース
- 分散データベース・アプリケーションの開発
- Oracle 分散データベース・システムの管理
- 各国語サポート

Oracle の分散データベース・アーキテクチャ

「分散データベース」は、複数のコンピュータに格納されているデータベースの集合体で、通常、アプリケーションからは1つのデータベースとして見えるようになっています。そのため、アプリケーションはネットワーク内の複数のデータベースにあるデータに同時にアクセスして修正できます。システム内の各 Oracle データベースは、それぞれのローカル Oracle Server によって制御されますが、グローバルな分散データベースの一貫性を維持するために協働します。

図 30-1 に、代表的な Oracle 分散データベース・システムを示します。

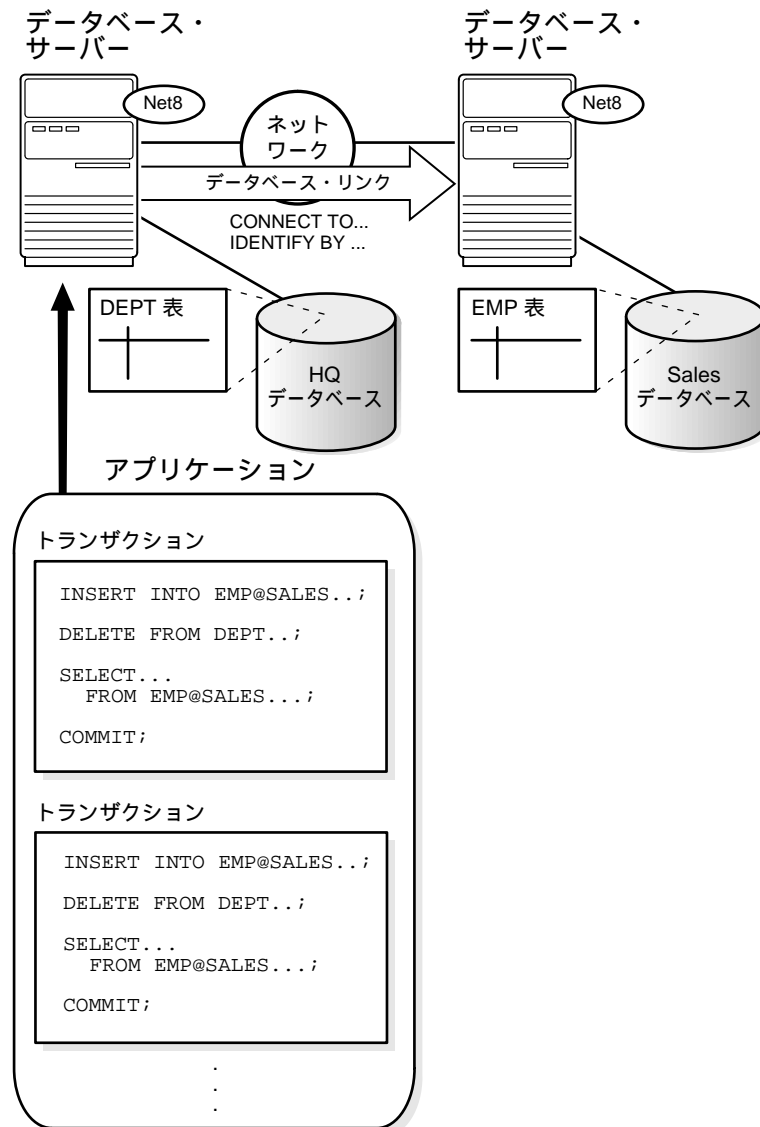
クライアントとサーバー

「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」とはサーバーにある情報を要求するアプリケーションのことです。システム内の各コンピュータが、「ノード」です。分散データベース・システム内のノードは、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作動します。たとえば、図 30-1 では、HQ データベースを管理するコンピュータは、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 表に対して問合せを発行しています）。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 に対して発行されています）。

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。図 30-1 では、クライアント・アプリケーションから各トランザクションの第 1 の文と第 3 の文が発行されると、クライアントは中間の HQ データベースには直接的に接続し、リモート・データを保有している SALES データベースには間接的に接続します。

図 30-1 Oracle 分散データベース・システム



ネットワーク

分散データベース・システムの個々のデータベースをリンクするには、ネットワークが必要です。以降の部分では、Oracle 分散データベース・システムのうちネットワークに関することをさらに説明します。

Net8

分散データベース・システムにあるすべての Oracle データベースは、ネットワークでのデータベース間通信のために Oracle のネットワーキング・ソフトウェアである Net8 を使います。Net8 は、1 つのネットワーク内の複数の異なるコンピュータ上で動作するクライアントとサーバーを接続しますが、またそれとちょうど同じようにして、複数のネットワークにわたる各データベース・サーバーが互いに通信できるようにすることにより、分散データベース内のリモート・トランザクションおよび分散トランザクションをサポートできるようにします。

Net8 によって、システムを使うアプリケーションでは、SQL 要求を送信したりデータを受信したりするのに必要な接続性を意識する必要がありません。Net8 は、クライアントからの SQL 文を取得してパッケージ化し、サポートする業界標準の通信プロトコルやプログラム・インタフェースによって Oracle Server に送信します。また、Net8 はサーバーからの応答を取得してパッケージ化し、適切なクライアントに送信します。Net8 は、基礎を形成するオペレーティング・システムに依存することなくすべての処理を実行します。

追加情報：Net8 とその機能の詳細は、『Oracle Net8 管理者ガイド』を参照してください。

Oracle Names

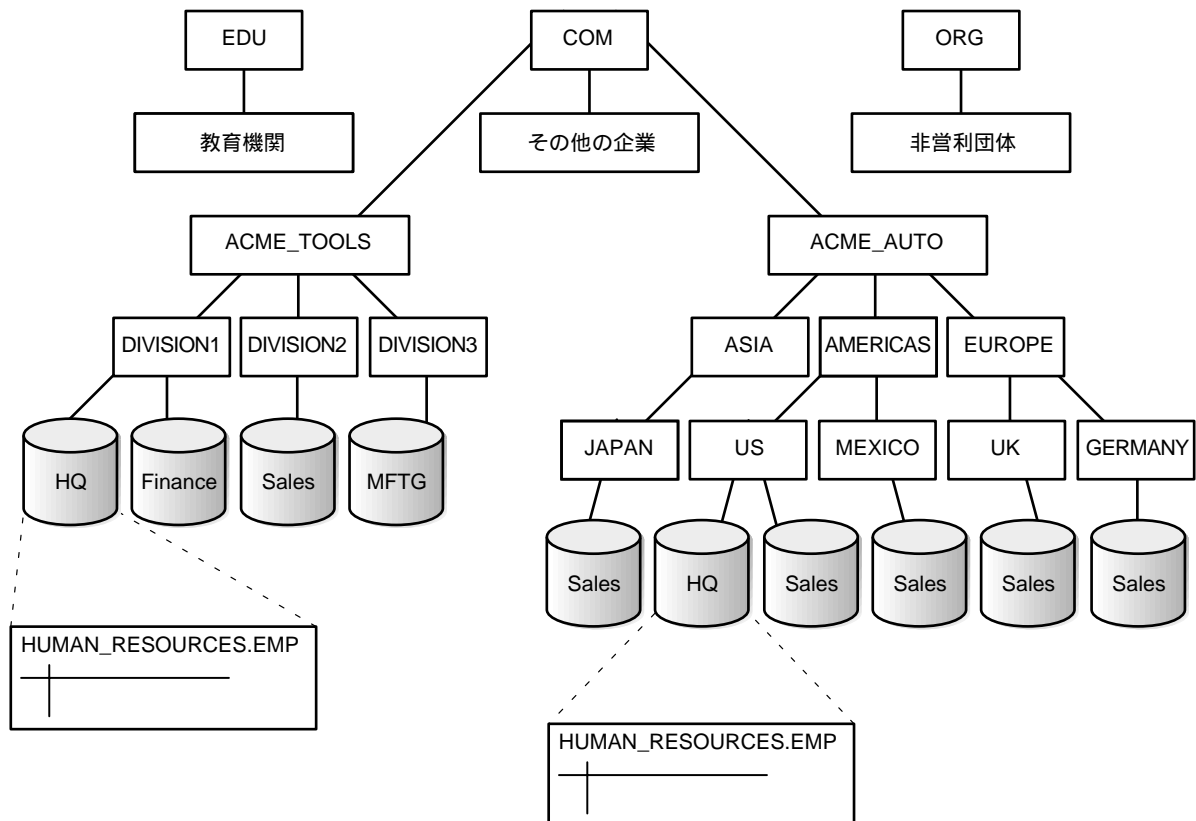
Oracle ネットワークでは、オプションとして Oracle Names を使うことによって、グローバルなディレクトリ・サービスをシステムに提供できます。Oracle ネットワークが分散データベース・システムをサポートしているなら、システム内の各データベースに関する情報が集中するリポジトリとして Oracle Names サーバーを使うことによって、

データベースとデータベース・リンク

分散データベース内の各データベースは、システム内の他のすべてのデータベースから区別されており、それぞれ独自のグローバル・データベース名が付けられています。Oracle が作るデータベースのグローバル・データベース名は、データベースのネットワーク・ドメインの最初に個々のデータベース名を付けたものです。

一例として、図 30-2 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 30-2 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



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でなければなりません。

たとえば、図 30-2 のネットワーク・ドメイン `US.AMERICAS.ACME_AUTO.COM` と `UK.EUROPE.ACME_AUTO.COM` のそれぞれに、`SALES` データベースが存在するという状態は可能です。

`SALES.US.AMERICAS.ACME_AUTO.COM`

`SALES.UK.EUROPE.ACME_AUTO.COM`

データベース・リンク

分散データベース・システムでのアプリケーション要求を処理するために、Oracle は「データベース・リンク」を使います。データベース・リンクは、Oracle データベースから他のデータベースまでの一方向の通信パスを定義するものです。

データベース・リンクの名前はそのリンクが指し示すデータベースのグローバル名と同じなので、Oracle 分散データベース・システムのユーザーにとってデータベース・リンクは本質的に意識しないで作業できます。たとえば、次の SQL 文では、ローカル・データベース内に、SALES.US.AMERICAS.ACME_AUTO.COM データベースへのリモート・パスを記述するデータベース・リンクが作られます。

```
CREATE DATABASE LINK sales.us.americas.acme_auto.com ... ;
```

データベース・リンクを作ったなら、ローカル・データベースと接続しているアプリケーションは、リモート・データベース SALES.US.AMERICAS.ACME_AUTO.COM のデータにアクセスできます。次の部分では、分散データベース内のリモート・スキーマ・オブジェクトをアプリケーションが参照する方法を説明し、SQL 文でデータベース・リンクを使う例を示します。

追加情報: Oracle では、さまざまなタイプのデータベース・リンクがサポートされています。詳細は、『Oracle8 Server 分散システム』を参照してください。

スキーマ・オブジェクトのネーム変換

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（「ネーム変換」プロセス）ために、Oracle は、階層型アプローチを使ってオブジェクト名を生成します。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。さらに、Oracle では、スキーマ・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簡単に解決できます。

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、単に階層型命名モデルをグローバル・データベース名で拡張することによって、効率的に「グローバル・オブジェクト名」を作り、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することによってリモート表を参照できます。

```
SELECT * FROM scott.emp@sales.us.americas.acme_auto.com;
```

要求を完了させるためローカル・データベース・サーバーは、リモート・データベース SALES と接続しているデータベース・リンクを暗黙のうちに使います。

複数のバージョンの Oracle Server の間の接続

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、そのような分散データベース・システムに加えることができます。しかし、分散データベース内で動作するアプリケーションでは、システム内の各ノードで利用できる機能について認識していなければなりません。たとえば、分散データベース・アプリケーションでは、Oracle8 で使用可能となったオブジェクト SQL 拡張を Oracle7 データベースが理解す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、意味ははっきりと異なっています。

分散データベース	分散データベースは、複数のコンピュータに格納されているデータベースの集合体で、アプリケーションからは1つのデータベースに見えるようになっているデータベースです。
分散処理	分散処理は、アプリケーション・システムがネットワーク内の異なるコンピュータの間に作業を分配する場合に生じます。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 PC または NC に分散させ、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・システムと呼ばれることがよくあります。

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場合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データベース・レプリケーション

「分散データベース」と「データベース・レプリケーション」という用語にも密接な関係がありますが、それらは異なっています。純粋な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・オブジェクトの単一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うことによってローカル・データとリモート・データの両方にアクセスし、グローバルなデータベースをリアルタイムに修正します。

注意：この章では、純粋な分散データベースについて説明します。レプリケーションの説明は、第 31 章「データベースのレプリケーション」を参照してください。

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。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うアプリケーションには、純粋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ため、パフォーマンスを改善したりアプリケーションの可用性を保護したりするのに役立ちます。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・トラフィックを最小化して最大のパフォーマンスを達成するようにするかもしれません。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でも、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利用できます。

追加情報: Oracleのレプリケーション機能の詳細は、『Oracle8 Serverレプリケーション』を参照してください。

異種間分散データベース

異種間分散データベースとは、少なくとも1つのデータベースが非 Oracle システムである分散データベースのことです。Oracle Serverから非Oracleシステムにアクセスするには、「Oracle オープン・ゲートウェイ」を使います。このゲートウェイを使うことによって、Oracle Serverのアプリケーションには、完全な異機種間透過性が提供されます。つまり、アプリケーションは非 Oracle システムにアクセスしていることを意識せずにすみずみます。

Oracle Server と非 Oracle システムの間では、データベース・リンクを使うことによって、非 Oracle システムにあるデータをアクセスしたり、非 Oracle システムでリモート・プロシージャを実行したりします。非 Oracle システムのトランザクション・システムを Oracle Server と統合することにより、データの整合性が保証されます。

透過的な SQL アクセス

アプリケーションは単にOracle SQL文をローカルOracle Serverに対して発行するだけです。そのSQL文では、非 Oracle リモート・システムにあるデータを、あたかもリモート Oracle システムであるかのように参照できます。Oracle Serverは、ゲートウェイを使うことにより、独自のSQL方言で非 Oracle システムにアクセスするために必要な変換を実行します。

プロシージャ・アクセス

一部の非 Oracle システムでは、プロシージャ・アクセスが必要です。アプリケーションは単にPL/SQL リモート・プロシージャ・コールを発行し、Oracle Serverは、ゲートウェイを使うことによって非 Oracle リモート・システムでプロシージャやファクションを実行するのに必要な変換を実行します。たとえば、リモート・プロシージャはメッセージ・システムと通信し、非 Oracle のメッセージ・システムにメッセージを入力できます。そのメッセージ・システムがトランザクションをサポートしている場合、Oracle Serverはゲートウェイとともに、さらに規模の大きいOracle 分散トランザクションの中のメッセージ・システムでの操作を実行し、すべての変更（メッセージ・システムでの変更とOracle Serverでの変更）がコミットされるかロールバックされるかのどちらか一方になるように保証することができます。

ゲートウェイの機能

まとめると、Oracle オープン・ゲートウェイには次のような機能があります。ただし、これがすべてではありません。

- 分散トランザクション。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も可能です。その場合でも、Oracle の 2 フェーズ・コミット・メカニズムによって、変更がすべてコミットされるかすべてロールバックされるかのどちらか一方になるように保証されます。
- 透過的な SQL アクセス。非 Oracle システムからのデータを、あたかも 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。アプリケーションが発行する SQL 文は、非 Oracle システムが認識できる SQL 文に暗黙のうちに変換されます。
- プロシージャ・アクセス。メッセージ・システムやキューイング・システムと同じく、プロシージャ・システムには、PL/SQL リモート・プロシージャ・コールを使って Oracle Server からアクセスします。
- データ・ディクショナリの変換。非 Oracle システムが Oracle Server とみなされるようにするため、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。オプションとして、アプリケーション・プログラマは、非 Oracle システムの SQL 方言を使って、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・プロシージャは、あたかも PL/SQL リモート・プロシージャであるかのようにアクセスされます。
- 各国語サポート。ゲートウェイは、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 Oracle Server との間でキャラクタ・セットを変換します。
- グローバルな問合せ最適化。Oracle の問合せオブティマイザでは、非 Oracle システムでの表の濃度と索引について考慮し、非 Oracle システムで効率的に実行される SQL 文になるよう分解します。

追加情報：上記の機能すべてがすべてのゲートウェイに適用されるとは限りません。どの機能がサポートされるかは、ゲートウェイのマニュアルを参照してください。

バージョン 8 ゲートウェイ

バージョン 8 ゲートウェイは、Oracle の異種間サービス機能によって、Oracle Server と密接に統合されています。異種間サービスは Oracle Server に統合されているので、異種間アクセスの管理作業は異なる複数のゲートウェイにわたって共通であり、Oracle Server に統合されています。

追加情報：異種間サービスの詳細は、『Oracle8 Server レプリケーション』を参照してください。

バージョン 4 ゲートウェイ

バージョン 4 ゲートウェイは、Oracle7 Server と Oracle8 Server に対してサポートされます。

追加情報：ゲートウェイの詳細は、『Oracle Open Gateway Technology, Concepts and Administration Guide Version 4』を参照してください。

分散データベース・アプリケーションの開発

分散データベース・システムの上でアプリケーションを作成する場合、いくつかの考慮点があります。この後の部分では、分散データベースの中のデータにアプリケーションがアクセスする方法について説明します。

リモート SQL 文と分散 SQL 文

「リモート問合せ」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、次のとおりです。

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

「リモート更新」とは、同一のリモート・ノードに存在している 1 つ以上の表のデータを変更する更新のことです。たとえば、次のとおりです。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com  
SET loc = 'NEW YORK'  
WHERE deptno = 10;
```

追加情報：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・ノードだけで発生するので、その文はリモート更新に分類されます。

「分散問合せ」は、複数のノードから情報を取り出します。たとえば、次のとおりです。

```
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」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・ユニットを使って実行できます。たとえば、次のとおりです。

```
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;

```

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・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、次のとおりです。

```

BEGIN
  emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;

```

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・ユニットが実行しなければならない作業や、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しなければなりません。

リモート・トランザクションと分散トランザクション

「リモート・トランザクション」とは、同一のリモート・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、次のとおりです。

```

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」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、次のとおりです。

```
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・ノードだけを参照する場合、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

DBMS 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 つの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一方でなことが保証されなければなりません。このため、トランザクションが適切に設計されていれば、論理データベース内のデータは常に一貫しています。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。このことは、問合せまたは更新、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、第 15 章「トランザクション管理」に記載されています。分散データベースでは、ネットワーク全体で同じ特性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でもデータの一貫性が保たれなければなりません。

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、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だけ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、整合性制約およびリモート・プロシージャ・コール、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。

追加情報：Oracle の 2 フェーズ・コミット・メカニズムの詳細は、『Oracle8 Server 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

最小限の努力により、システムで作業するユーザーにとっては Oracle 分散データベース・システムの機能が意識されないようにできます。そのような透過性の目標は、分散データベース・システムがあたかも単一の Oracle データベースであるかのようにすることです。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が困難なものとなる原因やユーザーの生産性が減少する原因になりかねない複雑な作業から解放されます。

この後の部分では、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る。

多くの場合、管理者と開発者は、シノニムを使うことによって、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ります。

```
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;
```

次のようにして、アプリケーションは、リモート表の位置を指定する必要のない、もっと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、開発者はビューやストアド・プロシージャによっても、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できます。

文とトランザクションの透過性

Oracle の分散データベース・アーキテクチャでは、問合せおよび更新、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、DELETE などの標準的な SQL コマンドは、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、SQL コマンド COMMIT および SAVEPOINT、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る。
- 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る。トランザクションはすべてコミットされるか、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る。つまり、ネットワークまたはシステムが復元されると、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っ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回復も完了できます。

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。

追加情報: Oracle のレプリケーション機能の詳細は、『Oracle8 Server レプリケーション』を参照してください。

Oracle 分散データベース・システムの管理

Oracle 分散データベース・システム用のアプリケーションを開発する場合に考慮すべき固有の問題点があるのと同じく、分散データベースの管理においても理解しておくべき特殊な問題点があります。以降の部分では、Oracle 分散データベース・システムのうちデータベース管理に関する特殊な事情を説明します。

サイト自律性

「サイトの自律性」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、データベースが非分散データベースとして稼働しているときと同じように、他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、別々に管理される固有で別個のデータ・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、企業の論理的な組織体系、または「互いに対等の立場に立った」関係を維持するのに必要な協調組織を反映できる。
- ローカルなデータベース管理者は、対応するローカルなデータを制御する。そのため、各データベース管理者の管轄領域が小さくなり、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少ない。1つのデータベースとネットワークが使える限りは、グローバルなOracleデータベースを少なくとも部分的に利用できます。1つのデータベースで障害が発生しても、すべてのグローバルな操作を停止する必要はなく、全体の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回復作業を、システム内の他のノードとは独立して実行できる。
- 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する。ローカル・データをアクセスするのにグローバル・カタログは不要です。
- 各ノードで単独にソフトウェアをアップグレードできる。

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もよいというわけではありません。たとえば、サーバー間接続のために作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・アカウントを追加しなければならないことがあります。この後の部分では、そのような特定のトピックについて説明し、システム内の個々のノードを管理する場合に、分散データベース環境の全体をグローバルにとらえる必要性を示します。

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または外部サービスの認証
- クライアント-サーバー間およびサーバー相互間の接続でのログイン・パケットの暗号化

この後の部分では、Oracle 分散データベース・システムを構成する場合に考慮すべき付加的な注意事項について説明します。

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、システムを使う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注意深く計画する必要があります。

- サーバー相互間接続を確立するのに必要なユーザー・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なければならない。
- 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・ユーザーから使えるようにするのに必要なロールは、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なければならない。

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・リンクを作る場合、各サイトでそのリンクを使うサーバー相互間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。

追加情報：システムでさまざまなタイプのデータベース・リンクをサポートするのに必要なユーザー・アカウントの詳細は、『Oracle8 Server 分散システム』を参照してください。

グローバルなユーザーとロール

多くの場合、分散環境では、ユーザーはたくさんのネットワーク・サービスにアクセスする必要があります。各ユーザーがネットワーク・サービスのそれぞれにアクセスできるようにするための認証を個別に構成しなければならないとすれば、特に大規模システムの場合、セキュリティ管理がわずらわしいものになってしまいます。グローバル認証サービスを使うことは、分散環境のセキュリティ管理を単純にするための一般的なテクニックです。

Oracle のクライアント/サーバー環境または分散データベース環境では、ユーザーとロールのグローバルな認証をサポートする 2 つのオプションがあります。

- Oracle Security Server。これは、Oracle ネットワークでの集中認証と分散認証をサポートする製品です。Oracle Security Server は Oracle の標準オプションです。
- グローバル・データベースのユーザーとロールの認証を非 Oracle の認証サービス（たとえば DCE）の枠内で機能させなければならない場合、Oracle 分散データベース環境では Net8 の Advanced Networking Option を使える。Net8 Advanced Networking Option は、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するための複数の機能を 1 つにまとめたオプション製品です。

注意: Advanced Networking Option は、Oracle8 Enterprise Edition を購入した場合にのみ利用できます。

データの暗号化

Net8 Advanced Networking Option は、ネットワーク・データの暗号化とチェックサムを使うことによってデータを読み込んだり変更したりできないようにするという点でも、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うことによって、許可なしでは表示できないようデータを保護します。データが修正、削除、再実行されていないことを確認するため、Advanced Networking Option 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。

追加情報: Advanced Networking Optionの機能の詳細は、『Oracle Net8管理者ガイド』を参照してください。Oracle8 Enterprise Editionで利用できる機能とオプションの詳細は、『Oracle8とOracle8 Enterprise Editionの解説』を参照してください。

Oracle 分散データベースの管理ツール

Oracle 分散データベース・システムを管理する際、データベース管理者は、次の複数の選択肢の中から使うツールを選べます。

- Oracle Enterprise Manager
- サードパーティの管理ツール
- SNMP サポート

Oracle Enterprise Manager

Oracle 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・コンポーネントを使うと、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の行モード・コンポーネントにより、行モードのインタフェースも提供されています。

Oracle Enterprise Managerには、管理機能のために使いやすいインタフェースが用意されています。Oracle Enterprise Manager を使うと、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する。そのような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うと、SQL コマンド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する。Oracle Enterprise Manager では、複数のウィンドウを同時にオープンできるので、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する。Oracle Enterprise Manager を使うと、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したりできます。
- データベース管理作業を集中化する。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 および PL/SQL、Oracle Enterprise Manager のコマンドを動的に実行する。Oracle Enterprise Manager を使うことによって、文を入力および編集、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実行しなければならない場合に特に役立つ機能です。
- グラフィカル・ユーザー・インターフェースが使えない場合や、使わないほうがよい場合には、Oracle Enterprise Manager の行モード・インターフェースによって管理作業を実行する。

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 を検索したり問い合わせたりできます。SNMP は、広く使われている次のようなたくさん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 on NetView
- IBM の NetView/6000
- Novell の NetWare Management System
- SunSoft の SunNet Manager

追加情報：『Oracle SNMP サポート・リファレンス・ガイド』を参照してください。

各国語サポート

Oracle では、クライアントとサーバーとで使うキャラクタ・セットの違うクライアント/サーバー環境がサポートされています。クライアントで使われるキャラクタ・セットは、クライアント・セッションの NLS_LANG パラメータの値によって定義されます。サーバーで使われるキャラクタ・セットは、そのサーバーのデータベースのキャラクタ・セットです。これらのキャラクタ・セットが異なる場合は、キャラクタ・セット間で自動的にデータ変換が実行されます。

追加情報：各国語サポート機能の詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

データベースのレプリケーション

*Lady, you are the cruel'st she alive,
If you will lead these grace to the grave
And leave the world no copy.*

Shakespeare: *Twelfth-Night*

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。

- レプリケーションとは？
- 基本レプリケーションの概念
- アドバンスド・レプリケーションの概念

注意：この章で説明しているアドバンスド・レプリケーション機能は、Oracle8 Enterprise Edition を購入した場合にのみ利用できます。

追加情報：『Oracle8 Server レプリケーション』には、データベースのレプリケーションに関する詳細情報が記載されています。

レプリケーションとは？

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。。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ため、パフォーマンスを改善したりアプリケーションの可用性を保護したりするのに役立ちます。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・トラフィックを最小化して最大のパフォーマンスを達成するようにするかもしれませんが。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でも、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。

Oracle では、基本レプリケーションとアドバンスド・レプリケーションという、2つの異なる形式のレプリケーションがサポートされています。

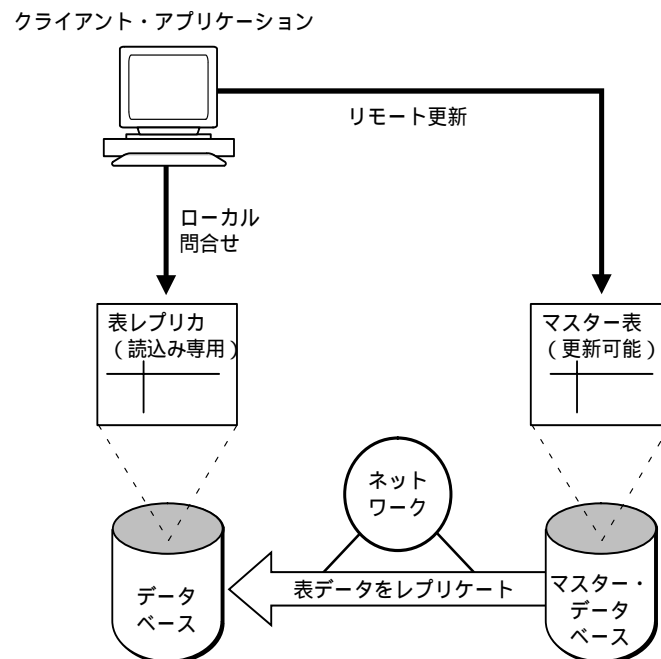
基本レプリケーションの概念

基本レプリケーションでは、データのレプリカによって、プライマリ（マスター）サイトに由来する表データへの読み専用アクセスが提供されます。アプリケーションは、ネットワークが使えるかどうかには関係なく、ローカルなデータ・レプリカからデータを問い合わせることができ、ネットワーク・アクセスを回避できます。しかし、更新操作が必要な場合は、システム全体のどのアプリケーションも、プライマリ・サイトのデータにアクセスしなければなりません。

図 31-1 に、基本レプリケーションを示します。

Oracle は、「読み専用表スナップショット」を使うことによって、読み専用の基本レプリケーション環境をサポートします。基本レプリケーションと読み専用スナップショットの詳細は、31-4 ページの「基本レプリケーションの概念」を参照してください。

図 31-1 読み込み専用の基本レプリケーション



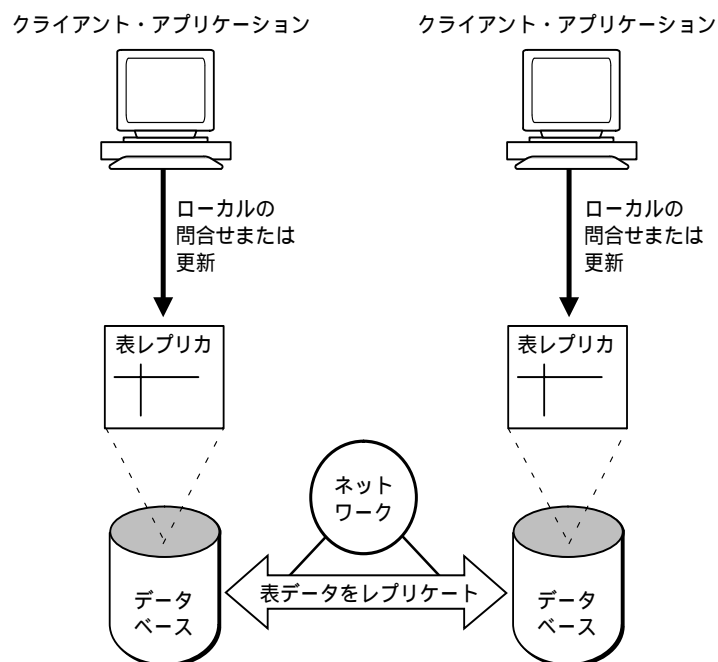
アドバンスト（対称型）レプリケーション

Oracle のアドバンスト・レプリケーション機能は、レプリケートされたデータベース全体でアプリケーションが表レプリカを更新できるように、読み込み専用の基本レプリケーションの機能を拡張したものです。アドバンスト・レプリケーションでは、システム内のどこにあるデータ・レプリカでも、表データへの読み込みおよび更新アクセスが提供されます。関係する Oracle データベース・サーバーは、自動的にすべての表レプリカのデータを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。

図 31-2 に、アドバンスト・レプリケーションを示します。

Oracle では、複数の構成を使うアドバンスト・レプリケーション環境の要件がサポートされます。アドバンスト・レプリケーション・システムの詳細は、31-11 ページの「アドバンスト・レプリケーションの概念」を参照してください。

図 31-2 アドバンスド・レプリケーション



基本レプリケーションの概念

基本レプリケーション環境では、データのレプリカによって、プライマリ・サイトに由来する表データへの読み専用アクセスが必要なアプリケーションがサポートされます。ここでは、基本レプリケーション環境の基本概念について説明します。

- 基本レプリケーションの使用方法
- 読み専用表スナップショット
- スナップショットのリフレッシュ

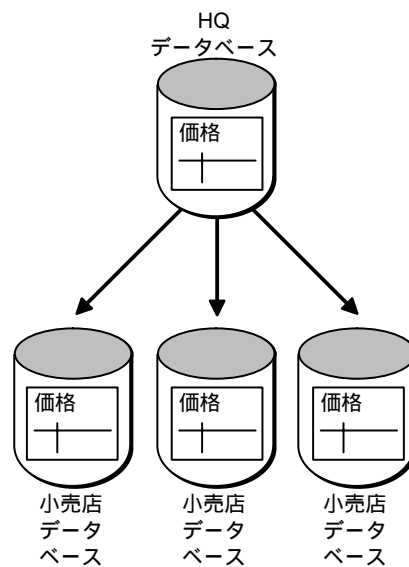
基本レプリケーションの使用方法

読み専用の基本データ・レプリケーションは、さまざまなアプリケーションで役立ちます。

情報の配布

基本レプリケーションは、情報の配布に役立ちます。たとえば、大手デパートのチェーン店の業務を考えてみましょう。この種の業務では、製品の価格情報が常に入手可能になっていて、しかも比較的最近の情報であり、すべての小売店で一貫していることが保証されていなければなりません。この目標を達成するため、各小売店ごとに独自の価格表を用意し、本店にある1次価格表に基づいて毎晩リフレッシュするように設定できます。

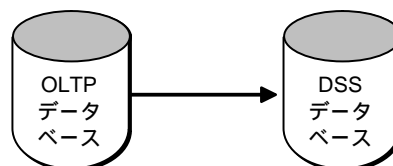
図 31-3 情報の配布



情報のオフロード

基本レプリケーションは、データベース全体をレプリケートしたり情報をオフロードしたりする方法として役立ちます。たとえば、大規模なトランザクション処理システムのパフォーマンスが非常に重要である場合、複製データベースを維持して、意思決定支援アプリケーションの要求問合せだけを特別扱いにするとよいでしょう。

図 31-4 情報のオフロード



情報輸送

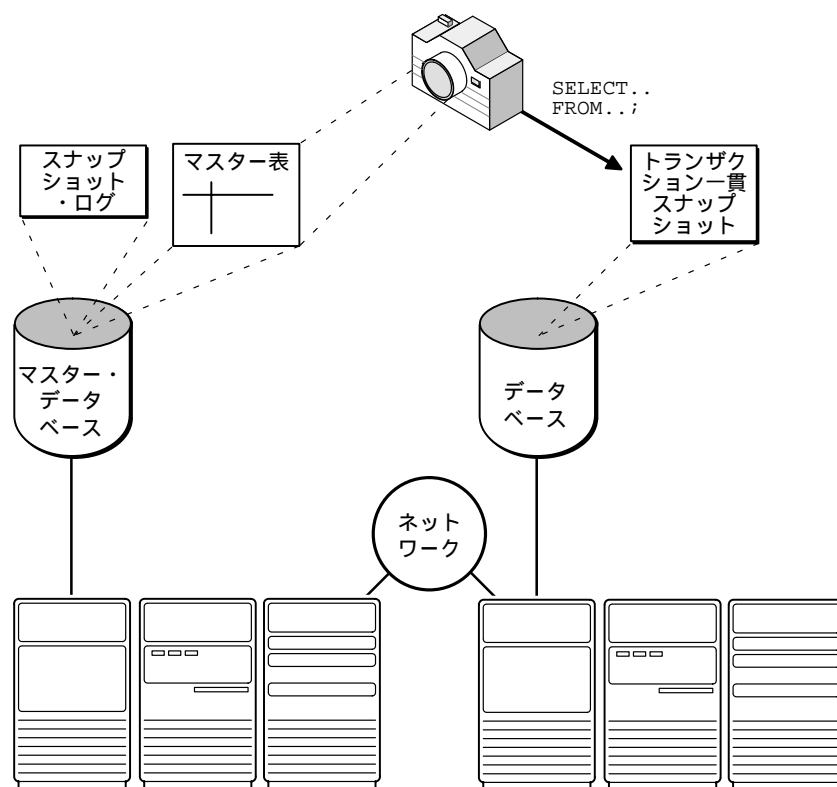
基本レプリケーションは、情報輸送メカニズムとして役立ちます。たとえば、基本レプリケーションを使って、実動トランザクション処理データベースにあるデータをデータ・ウェアハウスに定期的に移動できます。

読み専用表スナップショット

読み専用表スナップショットは、1つ以上のリモート・マスター表に基づいて作られた表データをローカルにコピーしたものです。アプリケーションは、読み専用表スナップショット内のデータの問合せはできますが、スナップショット内の行の挿入、更新、削除はできません。

図 31-5 とその後の項で、読み専用表スナップショットと基本レプリケーションについてさらに説明します。

図 31-5 読み専用スナップショットおよびマスター表、スナップショット・ログ



読み込み専用スナップショットのアーキテクチャ

Oracle は、表スナップショット・メカニズムによって基本データ・レプリケーションをサポートします。ここでは、単純な読み込み専用表スナップショットのアーキテクチャについて説明します。

注意：この他にも Oracle では、個々のアプリケーションの要件に応じて、複合スナップショットや ROWID スナップショットなどの基本レプリケーション機能が提供されています。これらの特殊な構成の詳細は、31-10ページの「その他の基本レプリケーション・オプション」を参照してください。

スナップショットの定義問合せ

表スナップショットの論理データ構造は、1 つ以上のリモート・マスター表にあるデータを参照する問合せによって定義されます。スナップショットの内容は、スナップショットの定義問合せによって決まります。

スナップショットの定義問合せでは、スナップショット内の各行が 1 つのマスター表の 1 つの行または 1 つの部分に直接対応していなければなりません。特に、スナップショットの定義問合せには、DISTINCT ファンクションや集約ファンクション、GROUP BY 句やCONNECT BY 句、結合、限定型の副問合せ、あるいは集合演算を含めることはできません。次の例は、単純な表スナップショットの定義を示すものです。

```
CREATE SNAPSHOT sales.customers AS
  SELECT * FROM sales.customers@hq.acme.com
```

注意：どの場合でも、スナップショットの定義問合せはマスター表にあるすべての主キー列を参照する必要があります。

スナップショットの定義問合せには、複数の表を参照してスナップショットのマスター表の行にフィルタをかける限定型の副問合せを含めることができます。「副問合せスナップショット」を使うと、複数のレベルが関係するかもしれない子表から親表への多対 1 参照を「さかのぼる」スナップショットを作れます。次の例は、単純な副問合せスナップショットを作る例です。

```
CREATE SNAPSHOT sales.orders AS
  SELECT * FROM sales.orders@hq.acme.com o
  WHERE EXISTS
    ( SELECT c_id FROM sales.customers@hq.acme.com c
      WHERE o.c_id = c.c_id AND zip = 19555);
```

スナップショットのリフレッシュ

スナップショットで示されるデータは、必ずしもそのマスター表の現行データと合致するとはかぎりません。表スナップショットは、特定の時点で存在していたデータとして、トランザクションの点で一貫した状態のマスター・データを反映したものです。スナップショットのデータをマスターのデータに対して比較的最近のものにしておくため、Oracle は定期的なスナップショットをリフレッシュする必要があります。スナップショットのリフレッシュは、スナップショットがマスターの現行の状態を反映するようにするための効率的なバッチ操作です。

より現行の状態に近いマスター・データが反映されるようにするには、いつ、またどのように各スナップショットをリフレッシュするかを決定する必要があります。たとえば、アプリケーションが頻繁に更新するマスター表に基づくスナップショットは、多くの場合、頻繁にリフレッシュしなければなりません。一方、比較的变化の少ないマスター表に依存するスナップショットでは、たまにリフレッシュすれば十分です。要約すると、アプリケーションの特性と要件を分析し、スナップショットの適切なリフレッシュ間隔を決定してください。

スナップショットのリフレッシュのために、Oracle では、さまざまなリフレッシュ（完全リフレッシュと高速リフレッシュ）、スナップショット・リフレッシュ・グループ、手動リフレッシュと自動リフレッシュをサポートしています。

完全リフレッシュと高速リフレッシュ

Oracle は、完全リフレッシュまたは高速リフレッシュのいずれかを使って、個々のスナップショットをリフレッシュします。

完全リフレッシュ

スナップショットの完全リフレッシュを実行する場合、スナップショットを管理するサーバーは、スナップショットの定義問合せを実行します。問合せの結果セットによって既存のスナップショット・データは置換され、それによってスナップショットがリフレッシュされます。完全リフレッシュは、どのスナップショットに対しても実行できます。

高速リフレッシュ

高速リフレッシュを実行する場合、スナップショットを管理するサーバーは、最後に実行されたスナップショットのリフレッシュ以来にマスターで生じた変更を識別し、その変更をスナップショットに適用します。関係するサーバーとネットワークがレプリケートするデータが少ないために、マスターに加えられる変更が少ない場合には、高速リフレッシュのほうが完全リフレッシュより効率的です。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、マスター表にスナップショット・ログがある場合だけです。

スナップショット・ログ

マスター表が1つ以上のスナップショットに対応する場合、その表のスナップショット・ログを作ることによって、スナップショットの高速リフレッシュが使えるようにしてください。マスター表のスナップショット・ログには、対応するすべてのスナップショットの高速リフレッシュ・データが記録されます。スナップショット・ログは、マスター表ごとに1つだけです。スナップショットの高速リフレッシュを実行する場合、サーバーは、マスター表のスナップショット・ログのデータを使うことにより、スナップショットを効率的にリフレッシュします。すべてのスナップショットがリフレッシュを実行し、ログ・データの必要がなくなると、特定のリフレッシュ・データがスナップショット・ログから自動的に削除されます。

スナップショット・リフレッシュ・グループ

互いに関連する複数のマスター表の表スナップショットの間で参照整合性とトランザクション一貫性を保つため、Oracleは各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編成およびリフレッシュします。グループ内のすべてのスナップショットは、1回の操作でリフレッシュされます。リフレッシュ・グループ内のすべてのスナップショットがリフレッシュされたなら、そのグループ内のすべてのスナップショットのデータは、トランザクションの点で一貫性のある同じ時点に対応したデータです。

自動スナップショット・リフレッシュ

スナップショット・リフレッシュ・グループを作る場合、普通、管理者は、Oracleが自動的にスナップショットをリフレッシュするようにグループを構成します。そうしないなら、管理者は、必要な場合に手動でグループをリフレッシュしなければなりません。

リフレッシュ・グループが自動的にリフレッシュされるように構成するには、次のことが必要です。

- グループのリフレッシュ間隔を指定する
- リフレッシュ期限に達しているスナップショットをリフレッシュするために、定期的に起動される1つ以上のSNPnバックグラウンド・プロセスによってスナップショットを管理するようサーバーを構成する

自動リフレッシュ間隔

スナップショット・リフレッシュ・グループを作る際、そのグループの自動リフレッシュ間隔を指定できます。グループのリフレッシュ間隔を設定する場合は、次のような動作について理解しておいてください。

- リフレッシュ間隔を指定する日付や日付式の評価結果は、将来の時点でなければならない。
- リフレッシュ間隔は、リフレッシュを実行するのに必要な時間より長くなければならない。
- 相対日付式は、最後のリフレッシュ日付を基準とした相対値として評価される。スケジュールに従ったグループ・リフレッシュがネットワーク障害やシステム障害によって妨げられた場合でも、それに応じて相対日付式の評価を変更できます。

- 明示日付式は、最近のリフレッシュ日付に関係なく、特定の時点に評価される。

リフレッシュの種類

デフォルトでは、Oracle はリフレッシュ・グループ内の各スナップショットごとに高速リフレッシュの実行を試みます。なにかの理由で（たとえばマスター表にスナップショット・ログがないなど）個々のスナップショットについて Oracle が高速リフレッシュを実行できない場合、サーバーは、そのスナップショットについては完全リフレッシュを実行します。

SNPn バックグラウンド・プロセス

Oracle の自動スナップショット・リフレッシュ機能は、定期的に行われる内部システム・プロシージャをジョブ・キューがスケジューリングすることによって機能します。ジョブ・キューを使うには、少なくとも1つのSNPnバックグラウンド・プロセスが実行されていなければなりません。「SNPn バックグラウンド・プロセス」は、定期的起動されて、ジョブ・キューをチェックし、未解決のジョブを実行します。

手動スナップショット・リフレッシュ

スケジューリングされた自動スナップショット・リフレッシュは、常に適切とはかぎりません。たとえば、大量のデータをマスター表にロードした直後には、依存スナップショットにマスター表のデータが反映されていません。スケジューリングされた次のグループ・リフレッシュを待つのではなく、依存スナップショット・グループを手動でリフレッシュすることにより、関連したスナップショットにマスター表の新しい行を即時に伝えることができます。

その他の基本レプリケーション・オプション

Oracle では、特定の状況で役立つその他の基本レプリケーション機能がサポートされています。

- 複合スナップショット
- ROWID スナップショット

複合スナップショット

スナップショットの定義問合せの中に DISTINCT ファンクションや集約ファンクション、GROUP BY句やCONNECT BY句、結合、限定型の副問合せ、または集合演算が含まれている場合、このスナップショットは複合スナップショットです。

次の例は、複合表スナップショットの定義の例です。

```
CREATE SNAPSHOT scott.emp AS
  SELECT ename, dname
  FROM scott.emp@hq.acme.com a, scott.dept@hq.acme.com b
  WHERE a.deptno = b.deptno
  SORT BY dname
```

複合スナップショットの主な弱点は、高速リフレッシュができないということです。Oracle では、複合スナップショットに対して完全リフレッシュしかできません。そのため、複合スナップショットを使った場合、完全スナップショット・リフレッシュの際にネットワーク・パフォーマンスに影響が出ます。

ROWID スナップショット

主キー・スナップショット（この章で前述）は、Oracle のデフォルトです。Oracle の主キー・スナップショットは、マスター表の主キーに基づいています。この構造のため、次のことができます。

- スナップショットの完全なリフレッシュをすることなく、スナップショットのマスター表を再編成する
- 限定型の副問合せを含む定義問合せによってスナップショットを作る

あくまで下位互換性のためですが、Oracle では、マスター表内の行の物理行識別子（ROWID）に基づく ROWID スナップショットもサポートされています。ROWID スナップショットは、Oracle リリース 7.3 データベースでのマスター表のスナップショットだけを対象としているものなので、Oracle8 データベースのマスター表のスナップショットを新しく作る場合は使わないでください。

注意：ROWID スナップショットをサポートするために、Oracle はスナップショットの実表に対して `L_SNAP$_snapshotname` という名前の付加的な索引を作ります。

アドバンスト・レプリケーションの概念

アドバンスト・レプリケーション環境では、システム内のどこにあるデータ・レプリカでも、表データへの読み込みおよび更新アクセスが提供されます。

注意：この項の説明は、Oracle8 Enterprise Edition のアドバンスト・レプリケーション機能だけに適用されます。Oracle8 Enterprise Edition で利用できる機能の詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

この項では、次のトピックを含むアドバンスト・レプリケーション・システムの主な概念を説明します。

- アドバンスト・レプリケーションの使用方法
- アドバンスト・レプリケーションの構成
- レプリケーションのオブジェクトおよびグループ、 サイト、 カタログ
- Oracle のアドバンスト・レプリケーション・アーキテクチャ

- レプリケーションの管理者および伝播担当者、受信者
- レプリケーションの競合
- 固有のアドバンスド・レプリケーション・オプション

アドバンスド・レプリケーションの使用方法

対称型のアドバンスド・データ・レプリケーションは、特殊な要件のあるさまざまなアプリケーション・システムで役立ちます。

切断環境

アドバンスド・レプリケーションは、切断コンポーネントを使って作動するトランザクション処理アプリケーションの開発に役立ちます。たとえば、典型的な生命保険会社の営業自動化システムを考えてみましょう。各販売員はラップトップ・コンピュータを携えて顧客を定期的に訪問し、個人データベースに注文を記録します。この間、会社のコンピュータ・ネットワークおよび中央のデータベース・システムからは切断されています。オフィスに戻ると、各販売員はすべての注文を会社の中央データベースに転送する必要があります。

フェールオーバー・サイト

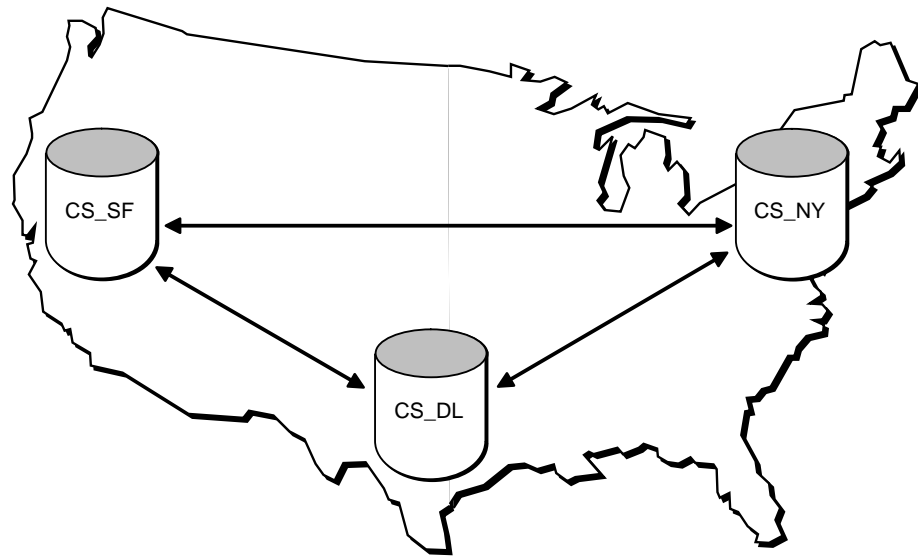
アドバンスド・レプリケーションは、非常に重要な任務を帯びたデータベースの可用性を保護するのに役立ちます。たとえば、対称型レプリケーション・システムによって、データベース全体をレプリケートし、プライマリ・サイトがシステムやネットワーク停止のために使用不能になった場合のフェールオーバー・サイトを確立できます。Oracle のスタンバイ・データベース機能とは対照的に、このようなフェールオーバー・サイトは、完全に機能するデータベースとして動作し、プライマリ・サイトが同時に並行して動作しているときもアプリケーション・アクセスをサポートします。

アプリケーション負荷の分散

アドバンスド・レプリケーションは、アプリケーションの大きな負荷を分散させたり、継続的な可用性を保証したり、データ・アクセスをさらにローカル化したりするために、データベース情報への複数のアクセス・ポイントを必要とするトランザクション処理アプリケーションに役立ちます。

図 31-6 で示すように、このような要件を持つアプリケーションとして広く使われるのは、顧客サービス向けのアプリケーションです。

図31-6 複数の更新アクセス・ポイントをサポートするアドバンスド・レプリケーション・システム



情報輸送

アドバンスド・レプリケーションは、情報輸送メカニズムとして役立ちます。たとえば、アドバンスド・レプリケーション・システムによって、大量の更新操作をするデータベースのデータを、定期的にデータ・ウェアハウスまたはデータ・マートにオフロードできます。

アドバンスド・レプリケーションの構成

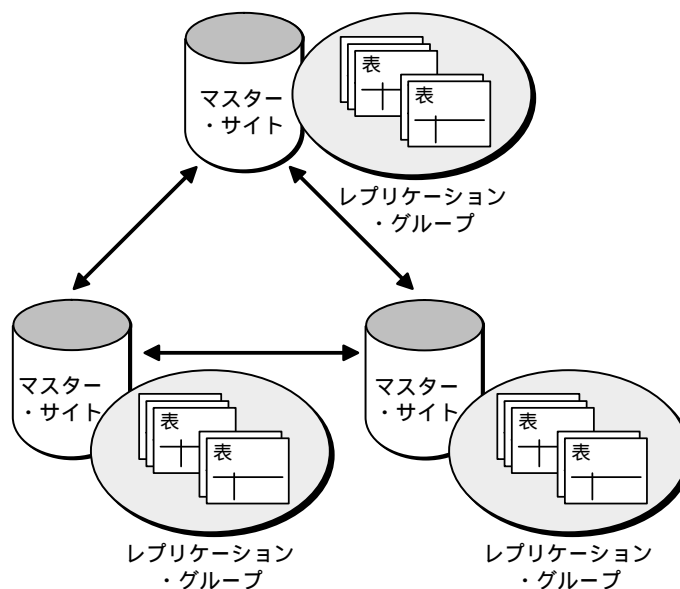
Oracle では、スナップショットが複数あるだけでなく、マスターが複数ある（マルチマスター）レプリケーションを使ったアドバンスド・レプリケーション環境の要件がサポートされています。

マルチマスター・レプリケーション

Oracleのマルチマスター・レプリケーションを使うと、互いに対等の複数のサイトによって、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。マルチマスター構成の場合、アプリケーションは、任意のサイトにあるレプリケートされた表を更新できます。

図 31-7 に、マルチマスター対称型レプリケーション・システムを示します。

図 31-7 マルチマスター・レプリケーション・システム

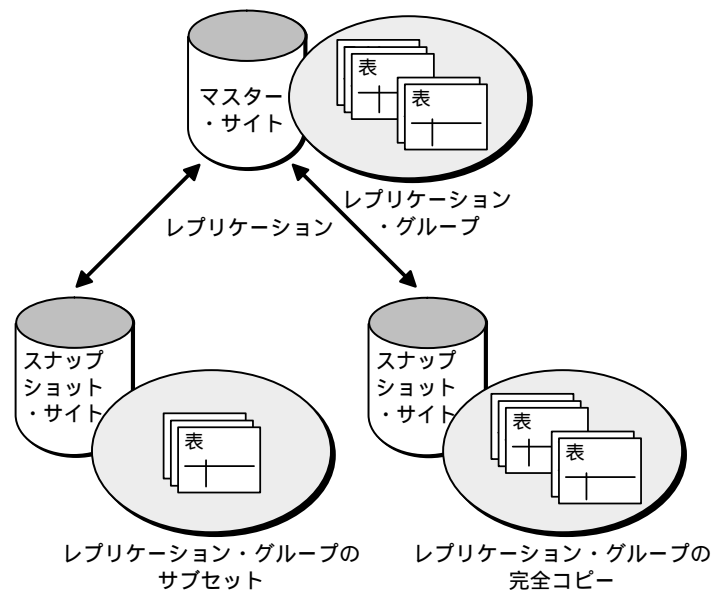


スナップショット・サイトと更新可能スナップショット

アドバンスド・レプリケーション・システムのマスター・サイトでは、アプリケーションがリモート・スナップショット・サイトで更新する情報を整理統合できます。Oracle の対称型レプリケーション機能を使うと、アプリケーションは更新可能スナップショットによって表の行を挿入したり、更新したり、削除したりできます。

図 31-8 に、更新可能スナップショットを使ったアドバンスド・レプリケーション環境を示します。

図 31-8 更新可能スナップショットを使ったアドバンスド・レプリケーション・システム



更新可能スナップショットには次の特性があります。

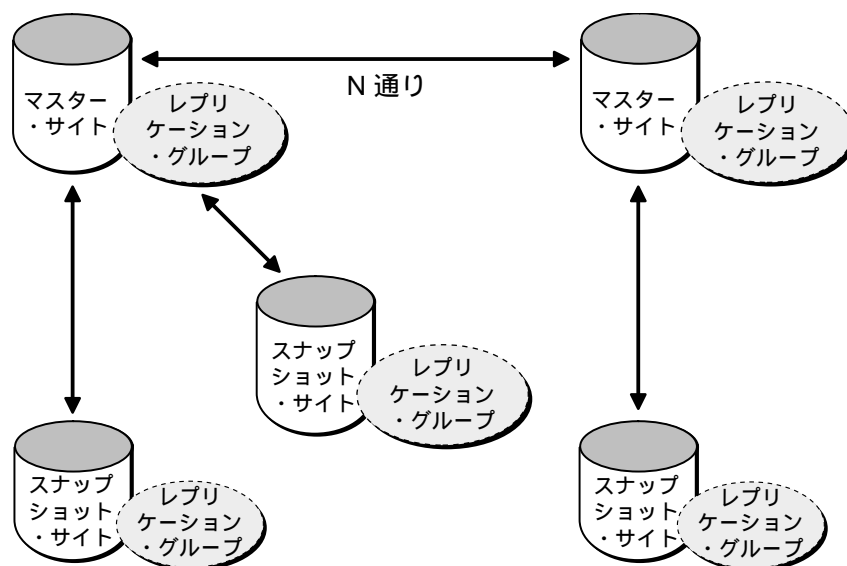
- 更新可能スナップショットは、常に単純であり、高速リフレッシュ可能表スナップショットである。
- Oracle は、更新可能スナップショットによって加えられた変更をスナップショットのリモート・マスター表に伝える。必要であれば、更新は他のすべてのマスター・サイトにも波及します。
- Oracle は、更新可能スナップショットをリフレッシュ・グループの一部としてリフレッシュする。読み専用スナップショットの場合と同じです。
- 更新可能スナップショットの基礎となるオブジェクト（実表、索引、ビュー）は、読み専用スナップショットの場合と同じである。さらに、Oracle は、更新可能スナップショットをサポートするため、表 `USLOG$_snapshotname` を作ります。

混合構成

マルチマスター・レプリケーションと更新可能スナップショットを混合構成で組み合わせると、さまざまなアプリケーションの要件を満たせます。混合構成では、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 31-9 に示すように、2つのマスターの間の n 重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の間の表全体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 31-9 混合構成



更新可能スナップショットとレプリケートされたマスターの間には、次のような重要な違いがあります。

- レプリケートされたマスターにはレプリケートされる表全体のデータが入っていないが、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る。
- マルチマスター・レプリケーションを使えば、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る。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、より少ない頻度で波及します。
- 同じデータの複数のコピーに変更が加えられた結果として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決する。

アドバンスト・レプリケーションと Oracle Replication Manager

どこでも更新可能なデータ・モデルをサポートするアドバンスト・レプリケーション環境は、構成や管理が困難です。アドバンスト・レプリケーション環境の管理を支援するため、Oracle では、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。

追加情報: 『Oracle8 Serverレプリケーション』には、Replication Managerの使用方法に関する情報と例が記載されています。

レプリケーションのオブジェクトおよびグループ、 サイト、 カタログ

ここでは、アドバンスト・レプリケーション・システムの基本的なコンポーネント(レプリケーションのオブジェクトおよびグループ、 サイト、 カタログを含む)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。Oracle のアドバンスト・レプリケーション機能を使えば、表と、サポートするオブジェクト(ビュー、データベース・トリガー、パッケージ、索引、シノニムなど)をレプリケートできます。

レプリケーション・グループ

アドバンスト・レプリケーション環境の場合、Oracle は、レプリケーション・グループを使っ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・グループ内で互いに関連したデータベース・オブジェクトを編成すると、たくさん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・アプリケーションをサポートするのに必要なスキーマ・オブジェクトを編成するために、レプリケーション・グループを作り、それを使います。これは、レプリケーション・グループとスキーマが互いに対応していなければならない、ということではありません。レプリケーション・グループ内のオブジェクトを複数のデータベース・スキーマのものとしたり、スキーマにさまざまなレプリケーション・グループのメンバーであるオブジェクトを含めたりできます。基本的な制限として、レプリケーション・オブジェクトは1つの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。アドバンスト・レプリケーション環境では、2つの基本的な型のサイト(マスター・サイトとスナップショット・サイト)がサポートされます。

- ・ 「マスター・サイト」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なコピーをメンテナンスします。マルチマスター対称型レプリケーション環境内のすべてのマスター・サイトは、互いに直接通信して、レプリケーション・グループ内のデータおよびスキーマの変更を波及させます。マスター・サイトにあるレプリケーション・グループは、細かくいえば「マスター・グループ」です。
- ・ さらに、すべてのレプリケーション・グループには、それぞれ「マスター定義サイト」が1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点となるマスター・サイトです。

- 「スナップショット・サイト」は、関連するマスター・サイトにある表データの単純な読み込み専用スナップショットおよび更新可能スナップショットをサポートします。スナップショット・サイトの表スナップショットには、レプリケーション・グループ内の表データのすべてまたはサブセットを入れることができますが、このスナップショットは、マスター・サイトの表と1対1対応している単純スナップショットでなければなりません。たとえば、スナップショット・サイトに、レプリケーション・グループのうち選択した表だけのスナップショットを入れることができます。また、特定のスナップショットには、レプリケートされた表の選択部分しか入れないようにもできます。スナップショット・サイトにあるレプリケーション・グループは、細かくいえば「スナップショット・グループ」です。スナップショット・グループには、他のレプリケーション・オブジェクトを入れることもできます。

レプリケーション・カタログ

アドバンスト・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。アドバンスト・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って、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。

レプリケーション管理 API と管理要求

アドバンスト・レプリケーション環境を構成して管理するため、関係する各サーバーは、Oracle のレプリケーション・アプリケーション・プログラム・インタフェース (API) を使います。サーバーの「レプリケーション管理 API 」は、管理者が Oracle のアドバンスト・レプリケーション機能を構成するために使うプロシージャとファンクションをカプセル化した PL/SQL パッケージの集まりです。Oracle Replication Manager では、作業を実行するのに、各サイトのレプリケーション管理 API のプロシージャとファンクションも使います。

「管理要求」は、Oracle のレプリケーション管理 API のプロシージャまたはファンクションのコールのことです。たとえば、Replication Manager を使って新しいマスター・グループを作る場合、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。一部の管理要求では、要求を完了するために、さらにその他のレプリケーション管理 API コールが生成されます。

Oracle のアドバンスト・レプリケーション・アーキテクチャ

Oracle は、通常のアドバンスト・レプリケーション構成のデータを収集する際に、行レベルのレプリケーションと非同期データ伝播を使います。ここでは、これらのメカニズムがどのように機能するかを説明します。

注意：この他にも Oracle では、個々のアプリケーションの要件に応じて、プロシージャ・レプリケーションや同期データ伝播などのアドバンスド・レプリケーション機能が提供されています。これらの特殊な構成の詳細は、31-26 ページの「固有のアドバンスド・レプリケーション・オプション」を参照してください。

行レベルのレプリケーション

多くのトランザクション処理アプリケーションは、トランザクションごとに少しの行を修正します。普通、このようなアプリケーションは、アドバンスド・レプリケーション環境で作業する際、Oracle の行レベル・レプリケーション・メカニズムに依存しています。「行レベルのレプリケーション」の場合、アプリケーションは、標準的な DML 文を使って、ローカル・データ・レプリカのデータを修正します。トランザクションがローカル・データを変更すると、サーバーは自動的にその修正に関する情報を獲得し、対応する遅延トランザクションをキューに入れてローカルな変更をリモート・サイトに送ります。

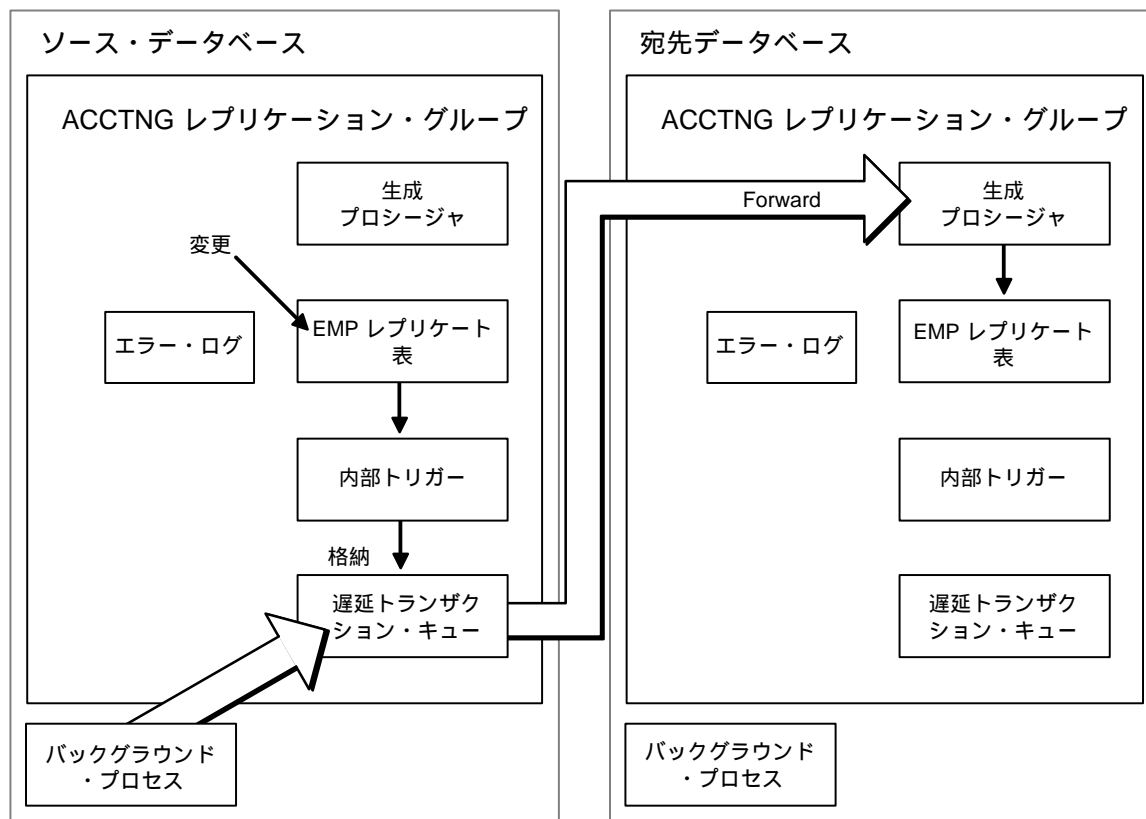
非同期（保管後転送）データ伝播

行レベルのレプリケーションに依存する多くのアドバンスド・レプリケーション構成では、「非同期データ・レプリケーション」によってデータ・レベルの変更を伝播します。非同期データ・レプリケーションは、アプリケーションが表のローカル・レプリカを更新し、レプリケーション情報をローカル・キューに格納し、そのレプリケーション情報を後で他のレプリケーション・サイトに送る場合に発生します。このため、非同期データ・レプリケーションは、「保管後転送データ・レプリケーション」とも呼ばれます。

図 31-10 に示すように、Oracle では、トリガーおよび遅延トランザクション、遅延トランザクション・キュー、ジョブ・キューの内部システムを使って、データレベルの変更がアドバンスド・レプリケーション・システム中のマスター・サイトの間で非同期に伝播します。これは、更新可能スナップショットがそのマスター表に伝播するのと同じです。

- アプリケーションがアドバンスド・レプリケーション環境で動作する場合、Oracle は、内部トリガーを使うことによって、レプリケートされたデータの更新情報を獲得および格納する。内部トリガーは、ローカル・サイトで実行されたデータ変更をリモート・レプリケーション・サイトで再実行するためのリモート・プロシージャ・コール（RPC）をビルドします。データ・レプリケーションをサポートする内部トリガーは、基本的には Oracle Server 実行ファイル内のコンポーネントです。したがって、Oracle は、システム・リソースの使用を最小限に抑えながら、レプリケートされたデータに加えられた更新を非常に高速に獲得および格納できます。

図 31-10 非同期データ・レプリケーションのメカニズム



- Oracle は、内部トリガーが生成した RPC を、後で伝播できるようにサイトの「遅延トランザクション・キュー」に格納する。Oracle はまた、トランザクションの初期化に関する情報を記録するので、トランザクションを構成するすべての RPC は、1 つのトランザクションとしてリモートに伝播および適用できます。Oracle のアドバンスド・レプリケーション機能は、Oracle のアドバンスド・キューイング・メカニズムによって遅延トランザクションをインプリメントします。
- Oracle は、「ジョブ・キュー・メカニズム」と「遅延トランザクション」によって、伝播プロセスを管理する。アドバンスド・レプリケーション・システムに係る各サーバーにはローカル・ジョブ・キューがあります。サーバーのジョブ・キューは、ジョブで実行する PL/SQL コールや、ジョブを実行する時刻など、ローカル・ジョブに関する情報が格納される 1 つのデータベース表です。アドバンスド・レプリケーション環境にあるジョブとして典型的なものには、リモート・マスター・サイトに遅延トランザクションを送るジョブ、適用されたトランザクションを遅延トランザクション・キューから消

去するジョブ、スナップショット・リフレッシュ・グループをリフレッシュするジョブなどが含まれます。

- Oracle は、RPC を遅延トランザクションの一部として実行することにより、データ・レプリケーション情報を送ります。また、分散トランザクション・プロトコルを使うことによって、グローバルなデータベースの整合性が自動的に保護され、データの耐障害性が保証されます。

シリアル伝播

「シリアル伝播」の場合、Oracle はレプリケートされたトランザクションを、1 つずつ、元のサイトでコミットされたのと同じ順序で非同期に伝播します。

パラレル伝播

「パラレル伝播」の場合、スループットを高くするために、複数のパラレル伝送ストリームを使って、レプリケートされたトランザクションが非同期に伝播されます。必要であれば、グローバルなデータベースの整合性を保証するために、依存トランザクションの実行の順序が Oracle によって決められます。

パラレル伝播は、パラレル問合せ、ロード、回復、その他のパラレル操作で使われるのと同じ実行メカニズムが使われます。各サーバー・プロセスは、1 つのストリームによってトランザクションを伝播します。パラレル調整プロセスによって、これらのサーバー・プロセスが制御されます。この調整プロセスはトランザクションの依存性を追跡し、サーバー・プロセスに作業を割り振り、その進行状況を記録します。

遅延トランザクション・キューの消去

サイトが遅延トランザクションを宛先に送った後も、そのトランザクションは、別のジョブによって消去されるまで、遅延トランザクション・キュー内に残っています。

スナップショットの伝播のメカニズム

アドバンスド・レプリケーション環境にある更新可能スナップショットは、マスター表から、またマスター表へとデータをやりとりできます。

マスター表の更新

更新可能スナップショットへの更新は、Oracle の行レベル非同期データ伝播メカニズム（RPC、遅延トランザクション、ジョブ・キュー）を使って、マスター表に非同期に送られます。

スナップショット・リフレッシュ

基本レプリケーション環境と同じく、アドバンスド・レプリケーション・システムでは、Oracle のスナップショット・リフレッシュ・メカニズムを使って、マスター表の変更を、関連する更新可能（および読み専用）スナップショットに非同期に送ります。

その他の考慮事項

更新可能スナップショットによるデータのやりとりは独立した操作であり、関連付けて、または別個に構成できます。

- スナップショット・サイトは、メンバー・スナップショットに加えられたすべての変更を自動的にマスター・サイトに送ってから、スナップショットをリフレッシュするように、リフレッシュ・グループを構成できる。
- スナップショット・サイトでは、変更をマスター・サイトに送る作業とスナップショットをリフレッシュする作業を別々の時刻および間隔で実行するように、更新可能スナップショットを構成できる。

たとえば、情報をマスター・サイトに整理統合するアドバンスド・レプリケーション環境では、変更内容は1時間ごとにマスター・サイトに送り、更新可能スナップショットのリフレッシュはそれより低い頻度にするように、更新可能スナップショットを構成できます。

レプリケーションの管理者および伝播担当者、受信者

Oracle 対称型レプリケーション環境では、いくつかの固有のデータベース・ユーザー・アカウントが正しく機能している必要があります。そのアカウントには、レプリケーションの管理者、伝播担当者、および受信者が含まれます。

- Oracle 対称型レプリケーション・システムのどのサイトにも、少なくとも1人の「レプリケーション管理者」が必要です。レプリケーション管理者は、レプリケートされたデータベース・オブジェクトの構成と管理を担当するユーザーです。
- Oracle 対称型レプリケーション・システムの各レプリケーション・サイトには、レプリケートされたデータへの変更内容を伝播および適用するための特殊なユーザー・アカウントが必要です。

構成オプション

ほとんどのアドバンスド・レプリケーション構成では、すべての目的（レプリケーション管理者、レプリケーション伝播担当者、レプリケーション受信者）のためにただ1つのアカウントを使います。しかし、Oracle では、固有の構成に合わせた固有のアカウントもサポートされています。

レプリケーションの競合

どこでも更新可能なデータ・レプリカのモデルをサポートするアドバンスド・レプリケーション・システムでは、レプリケーションが競合する可能性に対処する必要があります。ここでは、レプリケーション競合のさまざまな種類、それがいつ発生するか、Oracle ではどのようにレプリケーション競合を検出して解決するかを説明します。

レプリケーション競合の種類

アドバンスド・レプリケーション環境では、発生する可能性のある「競合」として、一意性競合および更新競合、削除競合の3種類があります。

一意性競合

「一意性競合」は、行のレプリケーションによってエンティティの整合性（PRIMARY KEY 制約や UNIQUE 制約）に違反しそうになったときに発生します。たとえば、2つの異なるサイトから来る2つのトランザクションが、同じ主キー値の行をそれぞれの表レプリカに挿入する場合に何が起きるかを考えてみましょう。トランザクションのレプリケーションによって、一意性競合が発生します。

更新競合

「更新競合」は、ある行に対する更新のレプリケーションが、同じ行に対する別の更新と競合する場合に発生します。更新の競合は、別々のサイトから来る2つの異なるトランザクションが、ほとんど同時に同じ行を更新する場合に生じる可能性があります。

削除競合

「削除競合」は、別々のサイトから来る2つのトランザクションのうち、1つのトランザクションが更新または削除している行を、別のトランザクションが削除しようとする場合に発生します。

レプリケート・データ・モデルと競合

アドバンスド・レプリケーションを使うデータベース・システムの上で動作するアプリケーションを設計する場合、レプリケーション競合の可能性を考慮する必要があります。この場合、アプリケーションは、レプリケーション競合を回避または解決することによってグローバルなデータベース整合性を保証するための、レプリケートされた複数の異なる「データ所有権モデル」の中から1つ選択しなければなりません。

プライマリ・サイトの静的所有権

「プライマリ所有権」(「静的所有権」ともいう)は、読み込み専用の基本レプリケーション環境でサポートされるレプリケート・データ・モデルです。一連のレプリケートされたデータへの更新アクセスを許可するサーバーは1つだけになるため、プライマリ所有権によってすべてのレプリケーション競合が発生しないようになります。

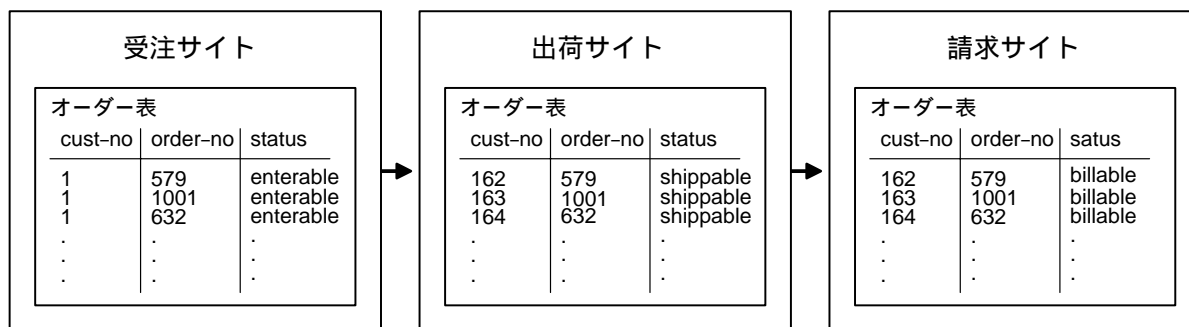
データの所有権を表レベルで制御するかわりに、アプリケーションは、水平パーティション化や垂直パーティション化によって、データの静的所有権をよりきめの細かいものにできます。たとえば、アプリケーションは、レプリケートされた表の中の特定の列や行に対する更新アクセス権限をサイト別に持てます。

動的所有権

「動的所有権」レプリケート・データ・モデルは、プライマリ・サイト所有権データ・モデルより制限の少ないものです。動的所有権の場合、データ・レプリカを更新する許可はサイトからサイトに移動します。なおかつ、特定の時点に特定のデータへの更新アクセスを提供するのは1つのサイトだけであることが保証されます。ワークフロー・システムは、動的所有権の概念を明確に示しています。たとえば、互いに関連する各部門アプリケーションにおいて、商品受注の状態コードを読み、受注を更新できるかどうかを判断できます。

図 31-11 に、動的所有権モデルを使うアプリケーションを示します。

図 31-11 受注処理システムにおける動的所有権



共有所有権

競合の回避を推進するプライマリ・サイト所有権および動的所有権のレプリケーション・データ・モデルの場合、一部のデータベース・アプリケーションでは、制限が多すぎインプリメントできなかったりすることがよくあります。一部のアプリケーションは、「共有所有権」レプリケート・データ・モデルを使って動作する必要があります。このモデルの場合、アプリケーションは、任意の時刻に任意の表レプリカのデータを更新できます。

共有データ所有権システムが非同期に変更をレプリケートする場合（保管後転送レプリケーション）対応するアプリケーションは、レプリケーション競合を常に回避するか、または競合が発生した場合にそれを検出して解決しなければなりません。たとえば、次のとおりです。

- 削除競合を非同期の共有所有権モデルで解決することは、トランザクションがデータを削除すると同時にレプリケーション・システムが履歴情報を記録していないかぎり、困難である。そのため、非同期の共有所有権データ・モデルで動作するアプリケーションは、DELETE 文を使って行を削除しないようにすることによって、削除の競合を回避しなければなりません。そのかわりに、アプリケーションでは、行に削除マークを付け、プロシージャ・レプリケーションを使って定期的に削除行を消去するようにシステムを構成できます。
- 「調整順序生成」は、共有データ所有権システムで一貫性競合を回避するためにアプリケーションで使える 1 つの方法です。たとえば、多くのアプリケーションは、Oracle 順序を使って数値主キーを生成します。共有データ所有権システム内の各サイトでは、各順序が相互に排他的な順序番号の集合を生成するようレプリカの順序を作るようにします。

競合の検出

アプリケーションが非同期の行レベル・レプリケーションとともに共有所有権データ・モデルを使う場合、Oracle は自動的に一意性競合、更新競合、および削除競合を検出します。レプリケーション時に競合を検出するため、元のサイトにある最小量の行データが、受信サイトにある対応する行情報と比較されます。違いが存在すると、競合が検出されたこととなります。

レプリケーション競合を正確に検出するため、データ・レプリケート時に異なるサイトにあって対応している行を一意に識別して一致させることができればなりません。多くの場合、Oracle のアドバンスド・レプリケーション機能では、表の主キーを使って、表の中の行を一意に識別します。表に主キーがない場合は、「代替キー」(データ・レプリケート時に表内の行を識別するのに Oracle が使う列または列集合)を指定する必要があります。どちらの場合でも、アプリケーションで表の ID 列を更新することはできません。それは、Oracle が行を識別し、レプリケートされたデータの整合性を保持できるようにするためです。

競合の解決

アドバンスド・レプリケーション・システムにおける受信サイトが、非同期の行レベル・レプリケーションを使っていて、トランザクションで競合を検出した場合のデフォルトの動作は、その競合とトランザクション全体をログ記録して、変更前のデータのローカル・バージョンをそのままにしておくことです。ほとんどの場合、Oracle のアドバンスド・レプリケーション機能を使って、レプリケーション競合の解決を自動化するとよいでしょう。また、競合を起こしたトランザクションについて各サーバーのDEFERRORデータ・ディクショナリ・ビューを調べて、必要であれば手動で解決することもできます。

列グループ

Oracle では、「列グループ」を使うことによって、非同期の行レベル対称型レプリケートで競合を検出し解決します。列グループとは、表内の 1 つ以上の列を論理的にグループ化したものです。レプリケートされる表の各列は、1 つの列グループの一部です。レプリケート表を構成する際には、列グループを作ってから、その各グループに列および対応する競合解決メソッドを割り当てることができます。

レプリケートされた表の中の各列グループには、1 つ以上の競合解決メソッドのリストを指定できます。1 つのグループに複数の競合解決メソッドを指定した場合、1 つのメソッドが競合の解決に失敗した場合に、別のメソッドで競合が解決されるようにできます。あるグループについての競合を解決する場合、そのグループの解決メソッドは、リストに指定された順序で実行されます。

デフォルトでは、レプリケートされたどの表にも「シャドウ列グループ」があります。表のシャドウ列グループには、特定の列グループに含まれないすべての列が含まれています。表のシャドウ・グループに、競合解決メソッドを割り当てることはできません。

競合解決メソッド

列グループを設計する場合、たくさんのビルトイン「競合解決メソッド」の中から選択できます。たとえば、更新競合を解決するため、宛先サイトの列値を元のサイトの列値で上書きさせることを選択できます。Oracle には、その他多数の競合解決メソッドが用意されています。

固有のアドバンスド・レプリケーション・オプション

一部のアプリケーションには、アドバンスド・レプリケーション・システムに関する特殊な要件があります。ここでは、Oracle 固有のアドバンスド・レプリケーション・オプションについて説明します。これには、次のオプションが含まれます。

- プロシージャ・レプリケーション
- 同期（リアルタイム）データ伝播

プロシージャ・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。そのような場合、典型的な行レベルのレプリケーションでは、ネットワークが大量のデータ変更で飽和してしまう可能性があります。このような問題を回避するため、アドバンスド・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・レプリケーション」を使うことによって、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・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うストアド・プロシージャのコールだけがレプリケートされます。プロシージャ・レプリケーションでは、データの修正はレプリケートされません。

プロシージャ・レプリケーションを使うには、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・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・レプリケーション

アドバンスド・レプリケーション・システムがプロシージャ・レプリケーションを使っ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しなければなりません。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決するように設計されていなければなりません。そのため、プロシージャ・レプリケーションが使われるのは、たいてい、データベースが大規模なバッチ操作でしか使えない場合です。そのような状況では、たくさんのトランザクションが同じデータを要求するということがないため、レプリケーションの競合はほとんど発生しません。

追加情報：『Oracle8 Server レプリケーション』を参照してください。

同期（リアルタイム）データ伝播

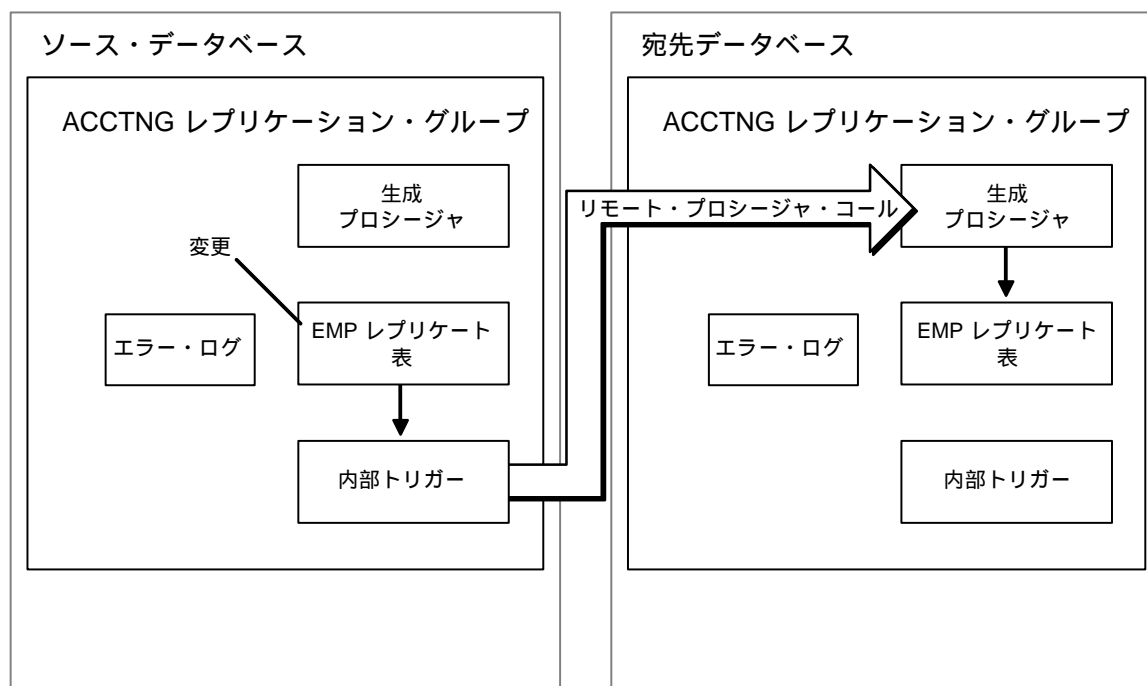
非同期データ伝播は、アドバンスド・レプリケーション環境で標準の構成です。しかし、Oracle では、特殊な要件のあるアプリケーションのために、同期データ伝播もサポートされています。「同期データ伝播」は、アプリケーションが表のローカル・レプリカを更新し、その同じトランザクションで同じ表の他のすべてのレプリカも更新する場合に発生します。そのため、同期データ・レプリケーションは、「リアルタイム・データ・レプリケーション」とも呼ばれます。同期レプリケーションは、アプリケーションの要件のために、レプリケートされたサイトの同期を継続的に取らなければならない場合だけに使ってください。

注意：レプリケーション・データのリアルタイム伝播を使うレプリケーション・システムは、システム内のすべてのサイトが同時に使用可能でなければ機能しないので、システムとネットワークの可用性に大きく依存することになります。

図 31-12 に示されているように、Oracle は、内部データベース・トリガーの同じシステムを使って、データレベルの変更を他のレプリケーション・サイトにレプリケートすることによって行レベルの同期データ・レプリケーションをサポートする RPC を生成します。しかし、Oracle はそのような RPC の実行を延期しません。そのかわり、データ・レプリケーション RPC は、ローカル・レプリカを修正したのと同じトランザクションの境界内で実行されます。このため、レプリケート表を管理するすべてのサイトでデータレベルの変更が可能でなければなりません。そうでない場合は、トランザクション・ロールバックが発生します。

一部のサイトは変更を同期に伝播し、その他のサイトは非同期の伝播（遅延トランザクション）を使うようなレプリケート環境を作ることができます。

図 31-12 同期データ・レプリケーションのメカニズム



レプリケーションの競合と同期データ・レプリケーション

共有所有権システムがすべての変更を同期的にレプリケートする場合（リアルタイム・レプリケーション）、レプリケーションの競合は発生しません。リアルタイム・レプリケーションの場合、アプリケーションは、分散トランザクションによって表のすべてのレプリカを同時に更新します。非分散データベース環境の場合と同じように、Oracle は、各分散トランザクションのために自動的に行をロックすることによって、トランザクション相互間でのあらゆる破壊的な干渉を防止します。リアルタイム・レプリケーション・システムではレプリケーションの競合は防止できますが、この型のシステムは、システム内のすべてのサイトが使用可能でなければ機能しないため、システムとネットワークの可用性に大きく依存することになります。

追加情報：基本データベース・レプリケーションとアドバンスト・データベース・レプリケーションの詳細は、『Oracle8 Server レプリケーション』を参照してください。

第IX部

付録

第IX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 を使う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。実際のオペレーティング・システムによって正式な名前は異なりますが、これらの Oracle マニュアルを『インストール・ガイド』と呼びます。このマニュアルでは、このようなマニュアルに言及する場合に、左余白にアイコンを記して読者の注意を引いています。

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・システムで使っている場合は、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。

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のリストは、それぞれのトピックを説明している項のページ番号を付記して、アルファベット順に記載してあります。

- 管理者権限、前提条件 :5-3 ページの「管理者権限での接続」
- 監査 :27-6 ページの「OS 監査証跡に対する監査」
- ユーザーの認証 :25-3 ページの「オペレーティング・システムによる認証」
- データベース管理者の認証 :5-3 ページの「管理者権限での接続」および 25-6 ページの「データベース管理者の認証」
- バックグラウンド・プロセス、作成 :7-5 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、複数 DBWR プロセス :7-8 ページの「データベース・ライター (DBWn)」
- バックグラウンド・プロセス、ARCHの使用法 :7-13 ページの「アーカイバ・プロセス (ARCH)」
- クライアント / サーバー通信 :7-18 ページの「専用サーバー (2 タスク) 構成」
- 通信ソフトウェア :7-27 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」
- Oracle の構成 :7-15 ページの「Oracle の構成におけるバリエーション」
- データ・ファイル、ファイル・ヘッダーのサイズ :3-12 ページの「データ・ファイルのサイズ」
- 専用サーバー、管理操作のための要求 :7-24 ページの「マルチスレッド・サーバーでの操作の制限」
- 索引、索引ブロックのオーバーヘッド :8-18 ページの「索引ブロックの形式」
- パラレル回復と非同期 I/O :28-13 ページの「パラレル回復を活用できる状況」
- オペレーティング・システムによるロール管理 :26-14 ページの「オペレーティング・システムとロール」
- ロールバック・セグメント、トランザクションあたりの数 :2-17 ページの「トランザクションとロールバック・セグメント」
- ソフトウェア・コード領域、共有または非共有 :6-16 ページの「ソフトウェア・コード領域」
- Net8、ネットワーク・ドライバの選択とインストール :7-27 ページの「プログラム・インタフェース・ドライバ」
- Net8、Net8 ソフトウェアに含まれるドライバ :29-5 ページの「Net8 の仕組み」

索引

数字

- 2 タスク・モード 7-16, 7-17
 - 説明 7-17
 - ネットワーク通信 7-19
 - プログラム・インタフェース 7-19
 - リスナー・プロセス 7-14
- 2 フェーズ・コミット
 - 手動による問題解決 1-25
 - 説明 1-25, 30-12
 - トランザクションの管理 15-7
 - トリガー 18-14
 - パラレル DML 22-30
- 3 値論理 (TRUE、FALSE、不明)
 - NULL のために生じる 8-7

A

- ADMIN OPTION
 - EXECUTE ANY TYPE 12-10
 - システム権限 26-3
 - ロール 26-12
- ADT、オブジェクト型を参照
- Advanced Networking Option 30-16
 - データの暗号化 30-17
- AFTER トリガー 18-8
 - 起動されるタイミング 18-14
 - 定義 18-8
- ALERT ファイル 7-14
 - REDO ログ 7-10
- ALL 20-14
- ALL_ROWS ヒント 20-39
- ALL_UPDATABLE_COLUMNS ビュー 8-13

- ALL_ ビュー 4-6
- ALTER ANY TYPE 権限 12-10
 - 権限も参照
- ALTER INDEX コマンド
 - REBUILD PARTITION 9-36
- ALTER SESSION コマンド 14-5
 - ENABLE PARALLEL DML 22-28
 - HASH_JOIN_ENABLED 20-64
 - OPTIMIZER_GOAL 20-39
 - SET CONSTRAINTS DEFERRED 24-20
 - 動的パラメータ 5-4
 - トランザクション分離レベル 23-7, 23-30
- ALTER SYSTEM コマンド 14-5
 - 動的パラメータ 5-4
- ALTER TABLE コマンド
 - CACHE 句 6-4
 - DEALLOCATE UNUSED 2-12
 - EXCHANGE PARTITION 9-9
 - SPLITPARTITIONのためのロギングなしモード 21-7
 - 監査 27-7
 - 制約を施行する 24-20
 - 制約を使用禁止または使用可能にする 24-20
- ALTER USER コマンド
 - 一時セグメント 2-16
- ALTER コマンド 14-4
 - パーティションの監査 9-39
- ALWAYS_ANTI_JOIN パラメータ 20-70
- ALWAYS_SEMI_JOIN パラメータ 20-70
- ANALYZE コマンド 14-4
 - COMPUTE STATISTICS 句 20-38

ESTIMATE STATISTICS 句 20-38
共有プール 6-10
パーティションの統計 9-11
ヒストグラムの作成 20-7
ANSI SQL 規格
Oracle が準拠 1-3
データ型 10-17
ANSI/ISO SQL 規格 1-3
データ同時実行性 23-2
複合外部キー 24-14
分離レベル 23-10
ANY 20-14
AQ
キュー・モニター・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キュー表 16-4
キュー表のエクスポート 16-8
メッセージ・キューイング 16-2
リモート・データベース 16-7
AQ_ADMINISTRATOR ロール 16-5
AQ_TM_PROCESS パラメータ 16-4, 16-5
ARCHIVELOG モード
アーカイバ・プロセス (ARCH) 7-13, 28-15
概要 1-37
全体データベース・バックアップ 28-20
定義 28-15
部分データベース・バックアップ 1-38, 28-21
ARCH バックグラウンド・プロセス 1-18, 7-13
アーカイバ・プロセスも参照
AUDIT コマンド 14-4
ロック 23-27

B
B* ツリー索引 8-19
索引構成表 8-25
ビットマップ索引との対比 8-21, 8-22
BEFORE トリガー 18-8
起動されるタイミング 18-14
定義 18-8
BETWEEN 20-15
BFILE データ型 10-10
BLOB 10-9
BOOLEAN データ型 10-2
BUFFER_POOL_KEEP パラメータ 6-5

BUFFER_POOL_RECYCLE パラメータ 6-5

C

CACHE 句 6-4
CASCADE アクション
DELETE 文 24-15
CHAR データ型 10-3
空白埋め比較方法 10-3
CHECK 制約 24-16
1 つの列に対する複数の制約 24-17
一部が NULL の外部キー 24-14
チェックのメカニズム 24-18
定義 24-16
パーティション・ビュー 9-9
副問合せは指定禁止 24-16
CHOOSE ヒント 20-39
CKPT バックグラウンド・プロセス 1-18, 7-11
CLEANUP_ROLLBACK_ENTRIES パラメータ 22-30
CLOB データ型 10-9
COMMENT コマンド 14-4
COMMIT コマンド 14-5
2 フェーズ・コミット 15-7, 30-12
DDL による暗黙 15-2, 15-4
高速コミット 7-10
トランザクションの終了 15-2, 15-4
パラレル DML での 2 フェーズ・コミット 22-30
COMPLEX_VIEW_MERGING パラメータ 20-23
COMPUTE STATISTICS 句 20-38
CONNECT BY 句
ビュー問合せの最適化 20-23
CONNECT INTERNAL 5-3
CONNECT ロール 26-14
ユーザー定義型 12-10, 12-11
CPU 時間の制限 25-10
CREATE ANY TYPE 権限 12-10
権限も参照
CREATE FUNCTION コマンド 17-15
CREATE INDEX コマンド
一時セグメント 2-15
オブジェクト型 12-9
パラレル化のルール 22-19
ロギングなしモード 21-7
CREATE PACKAGE BODY コマンド 17-10, 17-15
CREATE PACKAGE コマンド
パッケージ名 17-15

- 例 17-10, 18-9
- ロック 23-27
- CREATE PROCEDURE コマンド
 - プロシージャ名 17-15
 - 例 17-6
 - ロック 23-27
- CREATE SYNONYM コマンド
 - ロック 23-27
- CREATE TABLE AS SELECT
 - ダイレクト・ロード・インサートとの比較 21-2
 - パラレル化のルール 22-20
 - 領域の断片化 22-24
 - ロギングなしモード 21-7
- CREATE TABLE コマンド
 - CACHE 句 6-4
 - 監査 27-6, 27-7, 27-9
 - 制約を使用可能または使用禁止にする 24-20
 - パラレル化 22-22
 - 例
 - オブジェクト表 11-7, 11-11, 12-2, 12-8
 - ネストした表 11-11
 - 列オブジェクト 11-5, 12-2
 - ロック 23-27
- CREATE TRIGGER コマンド
 - コンパイルと格納 18-17
 - 例 18-9, 18-12, 18-16
 - オブジェクト表 12-9
 - ロック 23-27
- CREATE TYPE 権限 12-10
 - 権限も参照
- CREATE TYPE コマンド
 - VARRAY 11-10
 - オブジェクト・ビュー 13-3
 - オブジェクト型 11-4, 12-2, 12-6, 12-7
 - ネストした表 11-4, 11-10, 12-7
 - 不完全な型 12-13
- CREATE USER コマンド
 - 一時セグメント 2-16
- CREATE VIEW コマンド
 - 例 18-12
 - オブジェクト・ビュー 13-3
 - ロック 23-27
- CREATE コマンド 14-4

D

- DATE データ型 10-7
 - デフォルト書式の変更 10-7
 - パーティション化 9-11, 9-15
 - 日付算術 10-8
 - ユリウス暦 10-8
- DB_BLOCK_BUFFERS パラメータ
 - システム・グローバル領域のサイズ 6-11
 - バッファ・キャッシュ 6-4
- DB_BLOCK_LRU_LATCHES パラメータ 7-8
- DB_BLOCK_MAX_DIRTY_TARGET パラメータ 7-9, 28-5
- DB_BLOCK_SIZE パラメータ
 - システム・グローバル領域のサイズ 6-11
 - バッファ・キャッシュ 6-4
- DB_FILE_MULTIBLOCK_READ_COUNT パラメータ 20-56
 - コストベース最適化 20-66
- DB_FILES パラメータ 6-14
- DB_NAME パラメータ 28-18
- DB_WRITER_PROCESSES パラメータ 1-17, 7-8
- DBA_QUEUE_SCHEDULES ビュー 16-7
- DBA_SYNONYMS.SQL スクリプト
 - 使用 4-6
- DBA_UPDATABLE_COLUMNS ビュー 8-13
- DBA_ ビュー 4-6
- DBA ロール 26-14
 - ユーザー定義型 12-10
- DBMS 1-2
 - 一般要件 1-47
 - オブジェクト・リレーショナル DBMS 11-2
- DBMS_AQADM パッケージ 16-4, 16-5
- DBMS_AQ パッケージ 16-3
- DBMS_JOB パッケージ 7-13
- DBMS_LOCK パッケージ 23-38
- DBMS_SQL パッケージ 14-18
 - DDL 文の解析 14-18
- DBW_n バックグラウンド・プロセス 7-8
 - データベース・ライター・プロセスも参照
- DBWR_IO_SLAVES パラメータ 7-9
- DDL 1-48, 14-4
 - データ定義言語も参照
- DEFERROR ビュー
 - 競合 31-25
- DELETE 権限、オブジェクト表 12-11, 12-13

DELETE コマンド 14-3
 外部キー参照 24-15
 データ・ブロック内の領域の解放 2-9
 トリガー 18-2, 18-6
 INSTEAD OF トリガー 18-11
 パラレル DELETE 22-16
Digital POLYCENTER Manager on NetView 30-18
DISTINCT 演算子
 ビューの最適化 20-23
DISTRIBUTED_TRANSACTIONS パラメータ 7-12
DML 1-48, 14-3
 データ操作言語も参照
Dnnn バックグラウンド・プロセス 1-18, 7-14
 ディスパッチャ・プロセスも参照
DROP ANY TYPE 権限 12-10
 権限も参照
DROP TABLE コマンド
 監査 27-6, 27-7
DROP TYPE コマンド
 FORCE オプション 12-14
 依存性 12-14
DROP コマンド 14-4
DSS データベース
 索引のパーティション化 9-27
 スコア表 22-28
 ディスクのストライプ化 22-36
 パーティション 9-4
 パフォーマンス 9-7
 パラレル DML 22-27
DUAL 表 4-6

E

Enterprise Manager
 ALERT ファイル 7-15
 PL/SQL 14-16, 14-17
 SGA のサイズの表示 6-12
 SQL 文 14-2
 アドバンスド・キューイング 16-8
 起動 5-5
 システム権限の付与 26-3
 スキーマ・オブジェクト権限 26-4
 チェックポイント統計 7-11
 停止 5-7, 5-8
 統計モニター 25-12
 パッケージの実行 17-6

 パラレル回復 28-13
 プロシージャの実行 17-4
 分散データベース 30-17
 ロールの付与 26-12
 ロックとラッチのモニター 23-28
ESTIMATE STATISTICS 句 20-38
EXCHANGE PARTITION 9-9
EXECUTE ANY TYPE 権限 12-10
 権限も参照
EXECUTE 権限
 権限も参照
 ユーザー・アクセスの検査 17-16
 ユーザー定義型 12-10, 12-12, 12-13
EXECUTE 権限の GRANT オプション 12-10
EXECUTE ユーザー定義型 12-10
EXP_FULL_DATABASE ロール 26-14
EXPLAIN PLAN コマンド 14-3
 アクセス・パス 20-44, 20-45, 20-46, 20-47, 20-48,
 20-49, 20-50, 20-51, 20-52, 20-53, 20-54,
 20-55
 スター型変換 20-74
 スター問合せ 20-73
Export ユーティリティ
 パーティションのメンテナンス操作 9-29
 バックアップでの使用方法 28-22
 ユーザー定義型 12-15

F

FAST_FULL_SCAN_ENABLED パラメータ 20-41
FIPS 規格 1-3, 14-6
FIRST_ROWS ヒント 20-39
FORCE オプション
 オブジェクト型の依存性 12-14
FOREIGN KEY 制約
 NULL 24-14
 親キー値の変更 24-15
 親キー表の更新 24-15
 親表の行の削除 24-15
 制約チェック 24-18
 列の最大数 24-12

G

GRANT ANY PRIVILEGE システム権限 26-3
GRANT コマンド 14-4
 ロック 23-27

GROUP BY 句
ビューの最適化 20-23

H

HASH_AJ ヒント 20-70
HASH_AREA_SIZE パラメータ 20-65
HASH_JOIN_ENABLED パラメータ 20-64
HASH_MULTIBLOCK_IO_COUNT パラメータ 20-65
HASH_SJ ヒント 20-70
HASHKEYS パラメータ 8-35
HI_SHARED_MEMORY_ADDRESS パラメータ 6-12
HIGH_VALUE 列
 USER_TAB_COLUMNS ビュー 20-57
HP OpenView 30-18

I

IBM NetView/6000 30-18
ID 列
 競合の検出 31-25
IMP_FULL_DATABASE ロール 26-14
Import コーティリティ
 回復での使用方法 28-22
 パーティションのメンテナンス操作 9-29
 ユーザー定義型 12-15
INDEX_FFS ヒント 20-41
INIT.ORA ファイル 5-3, 5-5
INSERT 権限、オブジェクト表の 12-11, 12-12
INSERT コマンド 14-3
 INSERT ... SELECT のパラレル化 22-18
 空きリスト 2-9
 ダイレクト・ロード・インサート 21-2
 トリガー 18-2, 18-6
 BEFORE トリガー 18-8
 INSTEAD OF トリガー 18-11, 18-13
 パラレル INSERT の記憶領域 21-8
INSTEAD OF トリガー 18-11
 オブジェクト・ビュー 13-5
INTERNAL 接続 5-3
 監査レコードを生成しない 27-4
INTERSECT 演算子
 ビュー問合せの最適化 20-23
 複合問合せ 20-12
 例 20-36
INVALID 状態 19-3
IN 演算子 20-13

ビューのマージ 20-24

IN 副問合せ 20-23
IS NULL 述語 8-7
ISO SQL 規格 1-3, 10-17
 複合外部キー 24-14

J

JOB_QUEUE_PROCESSES パラメータ 16-7

L

LCK_n バックグラウンド・プロセス 1-19, 7-13
 ロック・プロセスも参照
LGWR_IO_SLAVES パラメータ 7-11
LGWR バックグラウンド・プロセス 1-17, 7-9
 ログ・ライター・プロセスも参照
LICENSE_MAX_SESSIONS パラメータ 25-13
LICENSE_SESSIONS_WARNING パラメータ 25-13
LIKE 20-13
LOB データ型 10-9
 BFILE 10-10
 BLOB 10-9
 CLOB および NCLOB 10-9
 デフォルトのロギング・モード 21-7
LOCK TABLE コマンド 14-3
LOCK_SGA_AREAS パラメータ 6-12, 6-16
LOCK_SGA パラメータ 6-12, 6-16
LOG_ARCHIVE_START パラメータ 28-16
LOG_BUFFER パラメータ 6-6
 システム・グローバル領域のサイズ 6-11
LOG_CHECKPOINT_INTERVAL パラメータ 7-8
LOG_CHECKPOINT_TIMEOUT パラメータ 7-8
LOG_FILES パラメータ 6-14
LONG RAW データ型 10-10
 LONG データ型との類似点 10-10
 索引の設定は禁止 10-11
 パーティション化の制限事項 9-11
LONG データ型
 記憶領域 8-7
 自動的に最後の列になる 8-7
 定義 10-4
 パーティション化の制限事項 9-11
LOW_VALUE 列
 USER_TAB_COLUMNS ビュー 20-57
LRU 6-3, 6-4, 7-8
 共有 SQL プール 6-8, 6-10

ディクショナリ・キャッシュ 4-4
ラッチ 7-8

M

MAC 1-33
MAXVALUE
パーティション表とパーティション索引 9-14
MERGE_AJ ヒント 20-70
MERGE_SJ ヒント 20-70
MERGE ヒント 20-23
MINIMUM EXTENT パラメータ 22-24
MINUS 演算子
ビュー問合せの最適化 20-23
複合問合せ 20-12
MLSLABEL データ型 10-15
MOVE PARTITION コマンド
パラレル化のルール 22-19
ロギングなしモード 21-7
MPP
大規模並列処理を参照
MTS_MAX_SERVERS パラメータ 7-23
人工デッドロック 7-24
MTS_SERVERS パラメータ 7-23

N

NCHAR データ型 10-4
NCLOB データ型 10-9
29-5, 29-4
Net8 29-4, 30-4
Advanced Networking Option 30-16
概要 29-4
マルチスレッド・サーバーの要件 7-14, 7-20
NEXT 記憶領域パラメータ
パラレル DML 21-8
NLS
各国語サポートを参照
NLS_DATE_FORMAT パラメータ 10-7
NLS_LANGUAGE パラメータ 9-14
NLS_LANG 環境変数 9-14
NLS_NUMERIC_CHARACTERS パラメータ 10-6
NLS_SORT パラメータ
ORDER BY アクセス・パス 20-53
パーティション化キーに影響を与えない 9-14
NOARCHIVELOG モード 28-15
LOGGING モードとの関係 21-5

回復用のデータベース・バックアップ 28-20
概要 1-37
定義 28-15
NOAUDIT コマンド 14-4
ロック 23-27
NOLOGGING モード
影響を受ける SQL 操作 21-7
ダイレクト・ロード・インサート 21-5
パーティション 9-34
パラレル DDL 22-22, 22-23
NOREVERSE オプション、索引 8-20
NOT 20-15
NOT IN 副問合せ 20-70
NOT NULL 制約
PRIMARY KEY による暗黙 24-11
UNIQUE キー 24-10
制約チェック 24-18
定義 24-6
Novell NetWare Management System 30-18
NULL
NULL 以外の値 8-7, 20-68
UNIQUE キー制約 24-10
UNIQUE キーでの不一致 24-10
アトミック 12-6
オブジェクト型 12-6
外部キー 24-14, 24-15
格納方法 8-7
禁止 24-6
索引 8-7, 8-24
主キーでは使用禁止 24-10
値に変換する 8-7
最適化 20-68
定義 8-7
デフォルト値 8-8
パーティション表とパーティション索引 9-14
比較では不明扱い 8-7
列の順序 8-7
NUM_DISTINCT 列
USER_TAB_COLUMNS ビュー 20-56
NUM_ROWS 列
USER_TABLES ビュー 20-56
NUMBER データ型 10-5
内部形式 10-6
丸め 10-6
NVARCHAR2 データ型 10-4

NVL 関数 8-7

O

OCI 7-27

OCIObjectFlush 13-4

OCIObjectPin 13-4

オブジェクト・キャッシュ 11-12

ストアド・プロシージャ 14-17

バインド変数 14-12

無名ブロック 14-16

OID 11-8, 12-4, 13-3, 13-4

WITH OBJECT OID 句 13-3, 13-4

OLTP データベース 9-4

索引のパーティション化 9-27

パーティション 9-5

バッチ・ジョブ 22-28

パラレル DML 22-27

OPEN_CURSORS パラメータ 14-6

プライベート SQL 領域の管理 6-9

OPEN_LINKS パラメータ 6-14

OPTIMAL 記憶領域パラメータ 2-22

OPTIMIZER_GOAL オプション 20-39

OPTIMIZER_MODE 20-38

影響を与えるヒント 20-39

Oracle

Oracle Server 1-4

Parallel Server オプション 1-7

Parallel Server も参照

SQL 処理 14-8

Trusted Oracle 1-33

アーキテクチャ 1-8, 1-12

移植性 1-4

インスタンス 1-6, 1-15, 5-2

拡張性 29-4

機能 1-2

クライアント / サーバー・アーキテクチャ 29-2

構成 7-2, 7-15

シングル・プロセス Oracle 7-2

マルチ・プロセス Oracle 7-3, 7-15

互換性 1-4

準拠する規格 1-3

整合性制約 24-4

接続性 1-4

異なる Oracle バージョン 30-7

データ・アクセス 1-47

動作例 1-19, 7-24

ネットワークでの使用 1-4, 1-26

プロセス 1-16, 7-5

ライセンス 25-12

Oracle AQ 16-1

キュー・モニター・プロセス 1-19, 7-13, 16-4

間隔統計 16-7

実行の時間枠 16-5

キュー表 16-4

キュー表のエクスポート 16-8

メッセージ・キューイング 16-2

リモート・データベース 16-7

Oracle Enterprise Manager

Enterprise Manager を参照

Oracle Forms

PL/SQL 14-15

オブジェクト依存性 19-11

Oracle Names

グローバルなディレクトリ・サービス 30-4

Oracle Parallel Server 1-7

Parallel Server も参照

Oracle Replication Manager 31-17

Oracle Security Server 30-16

Oracle Server 1-4

Oracle も参照

Oracle Type Translator (OTT) 11-13

Oracle オープン・ゲートウェイ 30-8

Oracle コード 7-2, 7-26

Oracle コール・インタフェース (OCI) 7-27

OCIObjectFlush 13-4

OCIObjectPin 13-4

オブジェクト・キャッシュ 11-12

ストアド・プロシージャ 14-17

バインド変数 14-12

無名ブロック 14-16

Oracle ブリコンパイラ

FIPS フラガー 14-6

埋込み SQL 14-5

カーソル 14-10

ストアド・プロシージャ 14-17

バインド変数 14-12

無名ブロック 14-16

Oracle プログラム・インタフェース (OPI) 7-27

Oracle ブロック 1-9, 2-2

データ・ブロックも参照

ORDBMS 1-40, 11-2
ORDERED ヒント 20-66
OTT 11-13

P

Parallel Server 1-7

- DML ロックとパフォーマンス 9-29
- PCM ロック 23-18
- インスタンス・グループ 22-14
- 逆キー索引 8-20
- 共有モード 5-6
 - ロールバック・セグメント 2-24
- システム・モニター・プロセス 7-12
- システム変更番号 7-10
- ディスクの親和性 22-36
- データベースとインスタンス 5-2
- データベースのマウント 5-6
- 同時実行の制限 25-13
- 名前付きユーザー・ライセンス 25-14
- 排他モード 5-6
 - ロールバック・セグメント 2-24
- パラレル SQL 22-1
- ファイルとログの管理ロック 23-29
- 分散ロック 23-18
- 分離レベル 23-11
- ロック・プロセス 1-19, 7-13

PARALLEL_DEFAULT_MAX_SCANS パラメータ (廃止) 22-14

PARALLEL_DEFAULT_SCANSIZE パラメータ (廃止) 22-14

PARALLEL_INDEX ヒント 22-12

PARALLEL_MAX_SERVERS パラメータ 22-7

PARALLEL_MIN_PERCENT パラメータ 22-14

PARALLEL_MIN_SERVERS パラメータ 22-6, 22-8

PARALLEL_SERVER_IDLE_TIME パラメータ 22-8

PARALLEL 句

- パラレル化ルール 22-15

PARALLEL ヒント 22-12

- UPDATE と DELETE 22-16

- パラレル化ルール 22-15

PCTFREE 記憶領域パラメータ

- PCTUSED 2-7

- その仕組み 2-5

PCTINCREASE 記憶領域パラメータ

- パラレル DML 21-8

PCTUSED 記憶領域パラメータ

- PCTFREE 2-7

- その仕組み 2-6

PGA 1-16, 6-12

- マルチスレッド・サーバー 7-23

PL/SQL 14-14

- DDL 文の解析 14-18

- PL/SQL エンジン 14-14, 17-2

 - コンパイラ 17-15

 - 製品 14-15

 - プロシージャの実行 17-17

- オブジェクト・ビュー 13-4

- オブティマイザの目標 20-39

- 解析ロック 23-27

- 外部プロシージャ 14-18, 17-9

- 概要 1-50, 14-14

- 言語要素 14-16

- 実行 14-14, 17-16, 17-17

- ストアド・プロシージャ 1-42, 14-14, 17-2, 17-6

- データ型 10-2

- データベース・トリガー 1-53, 18-1

- 動的 SQL 14-18

- 名前付き PL/SQL ブロックで使用禁止にされた

 - ルール 26-13

- バインド変数

 - ユーザー定義型 11-12

- パッケージ 17-4, 17-9

- プログラム・ユニット 1-42, 6-9, 14-14, 17-2

 - 共有 SQL 領域 6-9

 - コンパイル済み 14-15, 17-8, 17-15

- 文の監査 27-4

- 無名ブロック 14-14, 17-9

- ユーザー・ロック 23-38

- ユーザー定義データ型 11-12

- 例外処理 14-17

PMON バックグラウンド・プロセス 1-18, 7-12

- プロセス・モニター・プロセスも参照

PRIMARY KEY 制約 24-10

- 暗黙の NOT NULL 制約 24-11

- 施行に索引が使われる 24-11

 - 名前 24-11

- 制約チェック 24-18

- 説明 24-10

- 列の最大数 24-11

Pro*C/C++

SQL 文の処理 14-10
ユーザー定義データ型 11-12
PUBLIC ユーザー・グループ 25-8, 26-13
 プロセスの有効性 17-17
PUSH_JOIN_PREDICATE パラメータ 20-69
PUSH_JOIN_PRED ヒント 20-69
P コード 17-16

Q

QMN_n バックグラウンド・プロセス 1-19, 7-13, 16-4
 間隔統計 16-7
 実行の時間枠 16-5

R

RAW データ型 10-10
RDBMS 1-40
 Oracle も参照
 オブジェクト・リレーショナル DBMS 1-40, 11-2
REBUILD INDEX PARTITION コマンド 9-36
 パラレル化のルール 22-19
 ロギングなしモード 21-7
REBUILD INDEX コマンド
 パラレル化のルール 22-19
 ロギングなしモード 21-7
Recovery Manager 1-39, 28-10
 カタログを使わない操作 28-11
 パラレル操作 28-11
 リカバリ・カタログ 28-10
 レポートの生成 28-12
REDO ログ・バッファ 1-15, 6-5
 書込み 7-9
 サイズ 6-6
 循環 7-9
 トランザクションのコミット 7-10
 ログ・ライター・プロセス 6-6
REDO ログ・ファイル 1-11, 28-7
 アーカイバ・プロセス (ARCH) 7-13
 アーカイブ済み 1-37, 28-15
 アーカイブ時のエラー 28-17
 自動 28-16
 手動 28-17
 一時セグメントが関係する場合 2-16
 オンラインまたはオフライン 1-36, 1-37, 28-7
 回復 28-7
 概要 1-11, 1-36

制御ファイルに名前がある 28-18
多重化 1-37
 用途 1-11
トランザクションのコミット前の書込み 7-10
バッファ管理 7-9
パラレル回復 28-12
物理データベース構造 1-5
モード 1-37
ロールフォワード 28-8
ログ・ライター・プロセス 7-9
ログ順序番号 1-36
 制御ファイルに記録される 28-18

REF 11-8

REF ターゲット 12-14
REF のターゲット候補 12-14
暗黙的な解除 11-8
オブジェクト・ビューの行 13-3
オブジェクト識別子から組み立てる 12-4, 12-5
サイズ 12-5
索引 12-8
参照解除 11-8
相互に依存する型 12-13
表の別名の使用 12-2
ピン 12-12, 13-4
ぶら下がり 11-8
有効範囲付き 11-8, 12-5
REFERENCES 権限
 ロールを介して付与された場合 26-14
REF ターゲット 12-14
 REF も参照
REMOTE_DEPENDENCIES_MODE パラメータ 19-9
RENAME コマンド 14-4
Replication Manager 31-17
RESOURCE ロール 26-14
 ユーザー定義型 12-10, 12-11
RESTRICTED SESSION 権限 25-13
REVERSE オプション、索引 8-20
REVOKE コマンド 14-4
 FORCE オプション 12-14
 オブジェクト型と依存性 12-14
 ロック 23-27
ROLLBACK コマンド 14-5
ROWID 8-7
 Oracle 以外のデータベース 10-14
 REF 12-5

- アクセス 10-11
- クラスタ化された行 8-7
- 索引のソートでの使用 8-20
- 内部使用 10-14
- 表アクセス 20-40
- 変更 10-11
- ROWID スナップショット 31-11
- ROWID データ型 10-11
 - 拡張 ROWID 形式 10-11
 - 制限付き ROWID 形式 10-12
- ROWLABEL 列 10-15
- ROWNUM 疑似列
 - 索引を使えない 20-54
 - ビュー問合せの最適化 20-23, 20-32
- RPC 30-11, 31-19
- RULE ヒント
 - OPTIMIZER_MODE 20-39
- S**
- SAVEPOINT コマンド 14-5
- SCN 15-5
 - システム変更番号も参照
- SELECT 権限、オブジェクト表の 12-11, 12-12
- SELECT コマンド 14-3
 - 問合せも参照
 - 副問合せ 14-11
- Server Manager
 - ALERT ファイル 7-15
 - PL/SQL 14-16, 14-17
 - SGA のサイズの表示 6-12
 - SQL 文 14-2
 - セッション変数 14-16
 - 統計モニター 25-12
 - パッケージの実行 17-6
 - プロシージャの実行 17-4
 - ロックとラッチのモニター 23-28
- SESSION_ROLES ビュー
 - PL/SQL ブロックからの問合せ 26-13
- SET CONSTRAINTS コマンド
 - DEFERRABLE または IMMEDIATE 24-19
- SET ROLE コマンド 14-5
- SET TRANSACTION コマンド 14-5
 - ISOLATION LEVEL 23-7, 23-30
 - READ ONLY 2-17
- SGA

- システム・グローバル領域を参照
- SHARED_MEMORY_ADDRESS パラメータ 6-12
- SHARED_POOL_SIZE パラメータ 6-6
 - システム・グローバル領域のサイズ 6-11
- SHUTDOWN ABORT コマンド 5-8
- SMON バックグラウンド・プロセス 1-18, 7-12
 - 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 22-37
- Snnn* バックグラウンド・プロセス 7-14
- SNPn バックグラウンド・プロセス 1-19, 7-13
 - 自動スナップショット・リフレッシュ 31-9, 31-10
 - メッセージの波及 16-7
- SOME 20-14
- SORT_AREA_RETAINED_SIZE パラメータ 6-15
- SORT_AREA_SIZE パラメータ 2-15, 6-15
 - コストベース最適化 20-66
- SORT_DIRECT_WRITES パラメータ 6-15
- Spatial Data アプリケーション
 - 索引構成表 8-28
- SPLIT PARTITION コマンド
 - パラレル化のルール 22-19
 - ロギングなしモード 21-7
- SQL 14-2
 - PL/SQL 1-50, 14-14
 - 埋込み 1-48, 14-5
 - ユーザー定義データ型 11-12
 - カーソル 14-6
 - 解析 14-7
 - 概要 1-47, 14-2
 - 関数
 - COUNT 8-24
 - NVL 8-7
 - ビュー問合せの最適化 20-23, 20-30
 - 列のデフォルト値 8-8
 - 共有 SQL 14-7
 - 再帰 14-6
 - カーソル 14-6
 - システム制御文 14-5
 - セッション制御文 14-5
 - データ操作言語 (DML) 14-3
 - データ定義言語 (DDL) 14-4
 - 動的 SQL 14-18
 - トランザクション 1-48, 15-2, 15-5
 - トランザクション制御文 14-5

- パラレル実行 22-2
 - ファンクション 14-2
 - CHECK 制約 24-16
 - 文のタイプ 1-47, 14-3
 - 最適化 20-11
 - 文レベルのロールバック 15-4
 - メモリー割当て 6-10
 - ユーザー定義データ型 11-11, 12-2
 - OCI 11-13
 - 埋込み SQL 11-12
 - 予約語 14-2
- SQL*Loader
 - ダイレクト・ローダー 21-2
 - パーティション操作 9-29, 9-31
- SQL*Menu
 - PL/SQL 14-15
- SQL*Module
 - FIPS フラガー 14-6
 - ストアド・プロシージャ 14-17
- SQL*Net
- SQL*Plus
 - SQL 文 14-2
 - ストアド・プロシージャ 14-17
 - 接続 25-3
 - 無名ブロック 14-16
- SQL_TRACE パラメータ 7-15
- SQL92 23-2
- SQL 文 1-47, 14-2, 14-8
 - 1 つの SQL 文で起動されるトリガーの数 18-14
 - 依存オブジェクトの参照 19-4
 - 埋込み 14-5
 - カーソルの作成 14-10
 - 解析 14-11
 - 解析ロック 23-27
 - 概要 1-47
 - 監査 27-6, 27-9
 - 概要 1-32
 - レコードが生成される場合 27-4
 - 再帰
 - OPTIMIZER_GOAL は影響を与えない 20-39
 - 最適化
 - 複合文 20-20
 - 文のタイプ 20-11
 - 実行 14-8, 14-13
 - 実行計画 20-2
 - 障害 28-2
 - 正常な実行 15-3
 - タイプ 1-47, 14-3, 20-11
 - 単純 20-11
 - ディクショナリ・キャッシュ・ロック 23-28
 - トランザクション 14-13
 - トリガー 18-2, 18-8
 - トリガー・イベント 18-6
 - 配列処理 14-13
 - パラレル化 22-2, 22-8
 - パラレル問合せ 22-2
 - ハンドル 1-16
 - 必要な権限 26-3
 - 複合 20-12, 20-20
 - 最適化 20-20
 - 分散
 - 最適化 20-37
 - 定義 20-12
 - ノードへのルーティング 14-11
 - 分散データベース 30-10
 - 変換
 - 例 20-17
 - リソース制限 25-10
 - SQL 文のハンドル 1-16, 6-9
 - SQL 領域
 - 共有 1-15, 6-8, 14-7
 - プライベート 6-8
 - 持続 6-8
 - 実行時 6-8
 - STAR_TRANSFORMATION_ENABLED パラメータ 20-75
 - STAR_TRANSFORMATION ヒント 20-75
 - STAR ヒント 20-72
 - STORAGE 句
 - 使用 2-11
 - パラレル問合せ 22-24
 - SunSoft
 - SunNet Manager 30-18
 - SYS.AUD\$ ビュー
 - 削除 4-5
 - SYSDBA 権限 5-3
 - SYSOPER 権限 5-3
 - SYSTEM 表領域 3-4
 - いつもオンラインでなければならない 3-7

- 格納されているデータ・ディクショナリ 4-5, 4-2
- 格納されるプロシージャ 17-16
- メディア障害 28-6
- SYSTEM ユーザー名
 - セキュリティ・ドメイン 25-3
- SYSTEM ロールバック・セグメント 2-22
- SYS ユーザー名
 - 監査レコードを生成しない 27-4
 - セキュリティ・ドメイン 25-3
 - データ・ディクショナリ表を所有する 4-3

T

- TO_DATE 関数 10-7
 - パーティションの切詰め 9-11, 9-15
- TRANSACTIONS_PER_ROLLBACK_SEGMENT パラメータ 2-23
- TRANSACTIONS パラメータ 2-23
- TRUNCATE コマンド 14-4
 - 5-3
- Trusted Oracle
 - MLSLABEL データ型 10-15
 - 説明 1-33
 - 必須アクセス制御 1-33

U

- UNION ALL 演算子
 - OR からの変換 20-17
 - ビュー問合せの最適化 20-23
 - 例 20-18, 20-20, 20-34
- UNION ALL ビュー 9-9
- UNION 演算子
 - ビュー問合せの最適化 20-23
 - 複合問合せ 20-12
 - 例 20-25, 20-35
- UNIQUE キー制約 24-7
 - NOT NULL 制約 24-10
 - NULL 24-10
 - サイズの制限 24-9
 - 施行に索引が使われる 24-9
 - 制約チェック 24-18
 - 複合キー 24-8, 24-10
 - 列の最大数 24-9
- UPDATE No Action 制約 24-15
- UPDATE 権限、オブジェクト表の 12-11, 12-12
- UPDATE コマンド 14-3

- 外部キー参照 24-15
- データ・ブロック内の領域の解放 2-9
- トリガー 18-2, 18-6
 - BEFORE トリガー 18-8
 - INSTEAD OF トリガー 18-11
- パラレル UPDATE 22-16
- USE_INDIRECT_DATA_BUFFERS パラメータ 6-12
- USER_TAB_COLUMNS ビュー 20-56, 20-57
- USER_TABLES ビュー 20-56
- USER_UPDATABLE_COLUMNS ビュー 8-13
- USER_ ビュー 4-5
- USER 疑似列 26-6

V

- V_\$ ビューと V\$ ビュー 4-6
 - V\$LICENSE 25-13
- VARCHAR2 データ型 10-3
 - RAW データ型との類似点 10-10
 - 非空白埋め比較方法 10-3
- VARCHAR データ型 10-3
- VARRAY 11-9
- VLDB
 - パーティション 9-4
 - パラレル SQL 22-2

W

- WITH OBJECT OID 句 13-3, 13-4

あ

- アーカイバ・プロセス (ARCH)
 - 自動アーカイブ 28-16
 - 説明 7-13
 - 定義 1-18
 - 例 28-15
- アーカイブ済み REDO ログ 1-37
 - 自動アーカイブ 28-16
 - 手動アーカイブ 28-17
 - 使用可能にする 28-15
- アーキテクチャ
 - MPP 22-37
 - Oracle 1-12
 - SMP 22-37
 - クライアント / サーバー 1-23
- 空きリスト 2-9

空き領域 (データ・ブロックのセクション) 2-5

アクセス・パス

ROWID による単一行アクセス 20-44

一意キーまたは主キーによる単一行アクセス 20-46

クラスタ結合 20-46

クラスタ結合による単一行アクセス 20-44

最適化 20-40

索引クラスタ・キー 20-47

定義 20-4

ハッシュ・クラスタ・キー 20-47

ハッシュ・クラスタ・キー (一意キーの指定付き) による単一行アクセス 20-45

複合索引 20-48

リスト 20-42

アクセス制御 26-2

権限 26-2

任意 1-28

パスワード暗号化 25-4

必須 1-33

ルール 26-10

アクセスを逐次化できません 23-10

アドバンスト・レプリケーション 31-3

RPC 31-19

概要 31-11

行レベルのレプリケーション 31-19

更新可能スナップショット 31-14

混合構成 31-15

順序 31-24

使用方法 31-12

ジョブ・キュー 31-20

遅延トランザクション 31-20

同期伝播 31-27

非同期伝播 31-19

プロシージャ・レプリケーション 31-26

マルチマスタ構成 31-13

アドバンスト・キューイング (Oracle AQ) 16-1

キュー・モニター・プロセス 1-19, 7-13, 16-4

間隔統計 16-7

実行の時間枠 16-5

キュー表 16-4

キュー表のエクスポート 16-8

メッセージ・キューイング 16-2

リモート・データベース 16-7

アトミック NULL 12-6

アプリケーション

Spatial Data アプリケーション 8-28

アプリケーション・トリガーとデータベース・トリガー 18-4

意思決定支援システム (DSS) 8-22

パラレル SQL 22-2, 22-22

依存性 19-9

オブジェクト依存性 19-11

オンライン・トランザクション処理 (OLTP)

逆キー索引 8-20

オンライン分析処理 (OLAP) 8-28

コードの共有 6-16

索引構成表 8-27

情報検索 (IR) 8-27

制約の違反を検出できる 24-6

セキュリティの強化 1-31, 24-5

ダイレクト・ロード・インサート 22-28

データ・ウェアハウス 8-21, 20-70

データ・ディクショナリの参照 4-4

データベース・アクセス 7-2

トランザクションの終了 15-5

ネットワーク通信 29-5

パラレル DML 22-27

プログラム・インタフェース 7-26

プロセス 7-2, 7-4

離散トランザクション 15-8

ルール 26-11

暗号化 30-17

暗黙的な参照解除 11-8

い

意思決定支援システム (DSS) 9-4

スコア表 22-28

ディスクのストライプ化 22-36

パーティション 9-4

パフォーマンス 9-7, 22-27

パラレル DML 22-27

パラレル SQL 22-2, 22-22, 22-27

ビットマップ索引 8-22

異種間サービス 30-9

異種間分散データベース 30-8

移植性 1-4

依存性

Oracle Forms トリガー 19-11

オブジェクト型の定義 12-13, 12-14

- 共有プール 19-8
- 権限 19-6
- スキーマ・オブジェクト 19-2
- 存在しない参照オブジェクト 19-7
- 存在しない他のオブジェクト 19-7
- リモート・オブジェクト 19-8
- ローカル 19-9
- 一意キー 1-52, 1-53, 24-8
 - 検索 20-46
 - 最適化 20-20
 - 複合 24-8, 24-10
- 一意索引 8-16
- 一意性競合 31-22
- 一意でない索引 8-16
- 一時セグメント 2-14, 2-16
 - REDO ログに記録されない場合 2-16
 - エクステンツの割当て解除 2-14
 - 削除 2-14
 - 操作 2-15
 - パラレル DDL 22-24
 - パラレル INSERT 21-8
 - 表領域 2-14, 2-16
 - 割当て 2-16
 - 割当て制限は無視される 25-8
- 一時表領域 3-10
- 位置の透過性 1-24
- 一貫性、データ
 - マルチバージョンの一貫性モデル 1-21, 1-50
 - 読み込み一貫性も参照
- インスタンス 1-6
 - インスタンス・グループ 22-14
 - 回復 28-4
 - SMON プロセス 7-12
 - 増分チェックポイント 28-4
 - データベースのオープン 5-7
 - 概要 1-6
 - 仮想メモリー 6-16
 - 起動 5-5
 - 障害 1-35, 28-3
 - シングル・プロセス 7-2
 - 図 7-6
 - 制限モード 5-5
 - 説明 5-2
 - 定義 1-15
 - 停止 5-7, 5-8

- データベースに対応付ける 5-2, 5-6
- データベースの共有 1-7
- プロセスの構造 7-2
- マルチ・プロセス 7-3, 7-15
- メモリー構造 6-2
- ロールバック・セグメントの取得 2-23
- インスタンス・グループ、パラレル操作 22-14
- インターオペレータ並行性 22-10
- インダウト・トランザクション 2-21, 5-7
- イントラオペレータ並行性 22-10

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 22-27
- 内側獲得 12-2
- 埋込み SQL 文 1-48, 14-5

え

- 永続キューイング 16-2
- エクステンツ
 - 概要 2-10
 - 増分 2-10
 - 定義 2-2
 - データ・ブロックの集合 2-10
 - データ・ブロックの割当て 2-11
 - パラレル DDL 22-24
 - ロールバック・セグメント内
 - 現行の変更 2-19
 - ロールバック・セグメントの削除 2-22
 - ロールバック・セグメントへの割当て
 - セグメント作成後 2-21
 - セグメント作成時 2-18
 - 割当て解除
 - いつ実行されるか 2-12
 - ロールバック・セグメント 2-22
 - 割当てが行われる方法 2-11
- エクステンツの割当て解除 2-12
- エラー
 - 埋込み SQL 14-5
 - トレース・ファイルに記録される 7-14

お

- 応答キュー 7-20

- 応答時間 20-6
 - コストベースのアプローチ 20-38
 - オブジェクト・キャッシュ
 - OCI 11-13
 - Pro*C 11-12
 - オブジェクト・ビュー 13-4
 - 権限 12-12
 - オブジェクト・ビュー 8-14, 13-1
 - INSTEAD OF トリガーの使用方法 13-5
 - オブジェクト識別子 13-3, 13-4
 - 行オブジェクト 11-8
 - 更新 13-4
 - 定義 13-2
 - 変更の問題 18-11
 - 利点 13-2
 - オブジェクト・リレーショナル DBMS (ORDBMS)
 - 1-40, 11-2
 - オブジェクト型 1-40, 11-2, 11-3
 - Oracle Type Translator 11-13
 - オブジェクト・ビュー 8-14
 - キャッシュでのロック 11-13
 - 行オブジェクト 11-7
 - コンストラクタ・メソッド 1-52, 11-5, 12-4
 - 相互に依存する 12-13
 - 属性 11-2, 11-3, 11-4
 - 発注の例 11-2, 11-4
 - 比較メソッド 11-6
 - 表の別名の使用 12-2
 - 不完全 12-14
 - メソッド 1-51, 11-4
 - PL/SQL 11-12
 - 発注の例 11-2, 11-5
 - メソッド・コール 12-3
 - メッセージ・キューイング 16-5
 - 列オブジェクト 11-7
 - 索引 12-8
 - オブジェクト型のメソッド 1-51, 11-4
 - PL/SQL 11-12
 - 空のカッコの使用 12-3
 - コンストラクタ・メソッド 1-52, 12-4
 - 自己参照的なスタイルのメソッドの起動 11-5
 - 実行権限 12-10
 - 順序メソッド 1-52, 11-6
 - 発注の例 11-2, 11-5
 - マップ・メソッド 1-52, 11-6
 - オブジェクト権限 26-3
 - スキーマ・オブジェクトも参照
 - オブジェクト識別子 11-8, 13-3
 - WITH OBJECT OID 句 13-3, 13-4
 - オブジェクト・ビュー 13-3, 13-4
 - オブジェクト型 12-4
 - 索引 12-5
 - 行オブジェクト 11-8
 - オブジェクト表 11-3, 11-7
 - 仮想的なオブジェクト表 13-2
 - 行オブジェクト 11-7
 - 索引 12-8
 - 制約 12-7
 - トリガー 12-9
 - オフライン・バックアップ
 - 全体データベース・バックアップ 28-20
 - オフライン REDO ログ・ファイル 1-37, 28-7
 - 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 7-27
 - 認証 25-3
 - ブロック・サイズ 2-3
 - ロール 26-14
 - オンライン・トランザクション処理 (OLTP) 9-4
 - 逆キー索引 8-20
 - オンライン REDO ログ 1-36, 28-7
 - アーカイブ 28-15, 28-16
 - 制御ファイルに記録される 28-18
 - 多重化 28-6
 - チェックポイント 28-19
 - メディア障害 28-6
 - オンライン分析処理 (OLAP)
 - 索引構成表 8-28
- ## か
- カーソル
 - 埋込み SQL 14-5
 - オープン 6-9, 14-6
 - オブジェクト依存性 19-8
 - 概要 1-16
 - 再帰 14-6
 - 再帰 SQL 14-6
 - 最大数 14-6
 - 作成 14-10
 - ストアド・プロシージャ 14-16

- 定義 14-6
- プライベート SQL 領域 6-9, 14-6
- 解析 14-11
 - DBMS_SQL パッケージ 14-18
 - SQL 文 14-11, 14-18
 - 埋込み SQL 14-5
 - 解析コール 14-7
 - 解析ロック 14-11, 23-27
 - 実行 14-7
- 解析ツリー 17-16
 - 共有 SQL 領域 6-8
 - 構築 14-7
 - データベースへの格納 17-16
- 外部キー 1-52
 - 一部が NULL 24-15
 - 親キーを使うための権限 26-5
 - 定義 1-53
- 外部キーのマッチング
 - 完全一致、部分一致、不一致 24-15
- 回復
 - Recovery Manager 1-39, 28-10
 - インスタンス
 - SMON プロセス 7-12
 - インスタンスの回復 28-4
 - 異常終了後に必要な回復 5-8
 - 増分チェックポイント 28-5
 - データベースのオープン 5-7
 - パラレル DML 22-31
 - 概要 1-33, 28-8
 - 基本的な手順 1-38, 28-9
 - 災害時回復 28-22
 - 図 28-14
 - 推奨事項 28-14
 - スタンバイ・データベース 28-23
 - 全体データベース・バックアップ 28-20
 - 使われる構造 1-36, 28-6
 - データベース・バッファ 28-8
 - パラレル DML 22-30
 - パラレル回復 28-12
 - パラレル復旧 28-11
 - プロセスの回復 7-12, 28-3
 - 分散処理 7-12
 - 分散トランザクション 5-7
 - 文障害 28-3
 - メディア回復

- 使用可能または使用禁止 28-14
- ディスパッチャ・プロセス 7-24
- ロールバック 28-9
- ロールフォワード 28-8
- 外部結合
 - NULL に対する NULL 以外の値 20-68
 - 定義 20-12
- 外部プロシージャ 14-18, 17-9
- 書き込みが読み込みを阻止するか 23-10
- 拡張 ROWID 形式 10-11
- 拡張性
 - クライアント / サーバー・アーキテクチャ 29-4
 - バッチ・ジョブ 22-28
 - パラレル DML 22-27
 - パラレル SQL 実行 22-2
- 獲得回避規則 12-2
- 仮想表 1-41
- 仮想メモリー 6-16
- カタログ、レプリケーション 31-18
- カック、メソッド・コールでの使用 12-3
- 各国語サポート (NLS)
 - CHECK 制約 24-16
 - NCHAR および NVARCHAR2 データ型 10-4
 - NCLOB データ型 10-9
 - キャラクタ・セット 10-3
 - クライアントとサーバーで個別に選択可能 30-18
 - パラメータ 5-4
 - ビュー 8-12
- 仮読み込み 23-3, 23-10
- 監査 1-32, 27-1
 - DDL 文 27-6
 - DML 文 27-6
 - アクセス別 27-10
 - 必須 27-11
 - オプションはいつ有効になるか 27-5
 - 監査オプション 27-2
 - 監査証跡 27-3
 - オペレーティング・システム 27-4, 27-6
 - データベース 27-3
 - 監査レコード 27-3
 - 権限 27-2, 27-7
 - 失敗した実行 27-9
 - スキーマ・オブジェクト 27-2, 27-7
 - 成功した実行 27-9
 - セキュリティ 27-6

- セッション別 27-9
 - 禁止 27-11
- 説明 1-32, 27-2
- 対象範囲 27-2, 27-8
- タイプ 27-2
- 使うデータ・ディクショナリ 4-5
- データベースと OS のユーザー名 25-4
- トランザクションに依存しない 27-4
- パーティション表とパーティション索引 9-39
- 文 27-2, 27-6
- 分散データベース 27-5
- ユーザー 27-11
- 監査証跡
 - ディクショナリ内のデータの削除 4-5
- 監視、ユーザー・アクション 1-32, 27-2
- 関数
 - SQL 14-2
 - COUNT 8-24
 - NVL 8-7
 - デフォルトの列値 8-8
 - ビューでの使用 8-12
 - ビュー問合せの最適化 20-23, 20-30
 - ハッシュ関数 8-36
- 完全リフレッシュ 31-8
- 管理者権限
 - 監査されない 27-4
- 管理要求 31-18

き

キー

- 一意 24-7
 - 複合 24-8, 24-10
- 親 24-12, 24-14
- 外部キー 24-11, 24-12
- キー値 1-53
- 逆キー索引 8-20
- クラスタ 1-44, 8-29
- 検索 20-45
- 索引 8-17, 8-20, 24-9, 24-11
- 参照 1-53, 24-12
- 主キー 24-10
- 制約における 1-53
- 値の最大記憶領域 8-17
- 定義 24-8
- ハッシュ 8-35

記憶域

- NULL 8-7
- REF 12-5
- オブジェクト表 12-4
- 索引 8-18
- 索引クラスタ 8-31
- 索引パーティション 9-27
- データ・ファイル 3-11
- トリガー 18-2, 18-17
- ネストした表 12-5
- ハッシュ・クラスタ 8-33
- パラレル DDL の断片化 22-24
- パラレル INSERT 21-8
- ビュー 8-12
- 表パーティション 9-20
- 表領域の取消し 25-8
- 表領域の割当て制限 25-8
- ユーザーごとの割当て制限 1-31
- ユーザーに対する制限 25-7
- 論理構造 3-3, 8-2

記憶領域パラメータ

- NEXT 21-8
- OPTIMAL (ロールバック・セグメント) 2-22
- PCTINCREASE 21-8
- 設定 2-11

規格

- ANSI/ISO 1-3, 24-4, 24-14
 - 分離レベル 23-2, 23-10
- FIPS 1-3, 14-6
- Oracle が準拠する 1-3
- 整合性制約 24-4, 24-14

疑似コード

- トリガー 18-17

記述フェーズ、問合せの処理

疑似列

- CHECK 制約が禁止する
 - LEVEL および ROWNUM 24-16
- ROWID 10-11
- ROWNUM
 - 索引を使えない 20-54
 - ビュー問合せの最適化 20-23, 20-32
- USER 26-6
- ビューの変更 18-12

起動

- SGA の割当て 6-2

- 開始アドレス 6-12
- 回復 28-4
- 強制実行 5-5
- 共有モード 5-6
- 高速ウォームスタート 28-4
- ステップ 5-5
- 制限モード 5-5
- ディスパッチャ・プロセスでは禁止 7-24
- 排他モード 5-6
- 基本レプリケーション 31-2, 31-4
- 使用方法 31-4
- 逆キー索引 8-20
- キャッシュ
 - オブジェクト・キャッシュ 11-12, 11-13, 12-12
 - オブジェクト・ビュー 13-4
 - キャッシュ・ヒット 6-4
 - キャッシュ・ミス 6-4
 - 共有 SQL 領域 6-6, 6-8
 - データ・ディクショナリ 4-4, 6-10
 - 位置 6-6
 - データベース・バッファ 1-15
 - バッファ・キャッシュ 6-3
 - 複数のバッファ・プール 6-5
 - バッファの書込み 7-8
 - プライベート SQL 領域 6-8
 - ライブラリ・キャッシュ 6-6
- キャラクタ・セット
 - CLOB および NCLOB データ型 10-9
 - NCHAR および NVARCHAR2 10-4
 - 各国語 5-4
 - 列の長さ 10-3
- キュー・モニター・プロセス (QMNn) 1-19, 7-13, 16-4
 - 間隔統計 16-7
 - 実行の時間枠 16-5
- キューイング 16-2
 - キュー・モニター・プロセス 1-19, 7-13, 16-4
 - 間隔統計 16-7
 - 実行の時間枠 16-5
 - キュー表 16-4, 16-8
 - キュー表のエクスポート 16-8
 - リモート・データベース 16-7
- キューイングのエージェント 16-4
- 行 1-41, 8-3
 - ROWID での表示 10-12, 10-13

- ROWID はいつ変更されるか 10-11
- ROWID を使う検索 20-40, 20-44
- アドレス 8-7
- 格納形式 8-5
- 行オブジェクト 11-7
- 行ソース 20-3
- クラスタ化 8-6
 - ROWID 8-7
- サイズ 8-5
- 索引構成表での行オーバーフロー 8-26
- 説明 8-3
- 断片 8-5
- 定義 1-41
- データ・ブロック内での形式 2-4
- トリガー 18-7
- フェッチ 14-11
- ブロックにまたがる連鎖 2-9, 8-5
- ヘッダー 8-5
 - ロック 9-28, 23-10, 23-19, 23-21
- 行オブジェクト 11-7
- 行キャッシュ 6-10
- 競合 31-22
 - 一意性 31-22
 - 解決 31-25
 - 行レベルのレプリケーション 31-25
 - 検出 31-25
 - 更新 31-23
 - 削除 31-23
 - データ
 - デッドロック 7-23, 23-16
 - ロックの段階的拡大が発生しない 23-16
 - データ・モデル 31-23
 - プロシージャ・レプリケーション 31-26
 - 列グループ 31-25
 - レプリケーション 31-22
 - ロールバック・セグメント 2-18
- 競合の解決 31-25
- 行ソース 20-3
- 行断片 8-5
 - 識別方法 8-7
 - ヘッダー 8-5
- 行ディレクトリ 2-4
- 行データ (データ・ブロックのセクション) 2-5
- 行トリガー 18-7
 - トリガーも参照

- 起動されるタイミング 18-14
- 行のロック 23-10, 23-19
 - 直列可能トランザクション 23-7
- 業務ルール
 - 制約を使って施行する 24-1
- 共有 SQL 領域 6-8, 14-7
 - ANALYZE コマンド 6-10
 - SQL のロード 14-11
 - 依存性の管理 6-10
 - 解析ロック 23-27
 - 概要 1-15, 14-7
 - サイズ 6-8
 - 説明 6-8
 - プロシージャ、パッケージ、トリガー 6-9
- 共有グローバル領域 (SGA) 6-2
 - システム・グローバル領域も参照
- 共有サーバー 1-17
 - 管理者権限でセツゾクできない 5-3
- 共有サーバー・プロセス (Snnn) 7-14, 7-23
 - 説明 7-23
- 共有所有権 31-24
- 共有プール 6-6
 - ANALYZE コマンド 6-10
 - 依存性の管理 6-10
 - オブジェクト依存性 19-8
 - 概要 1-15
 - 行キャッシュ 6-10
 - サイズ 6-6
 - 説明 6-6
 - フラッシュ 6-11
 - プロシージャとパッケージ 17-15
 - 割当て 6-10
- 共有モード 5-6
 - ロールバック・セグメント 2-24
- 共有ロック
 - 共有表ロック (S) 23-23
- 行レベル・ロック 23-10, 23-19
- 行レベルのレプリケーション 31-19
 - 競合の検出 31-25
- 切詰め、パーティション 9-3, 22-4, 22-36
 - TO_DATE 書式マスク 9-11, 9-15
 - 索引 9-26
 - 索引パーティション 9-3

く

- クライアント / サーバー・アーキテクチャ 29-2
 - 概要 1-23, 29-2
 - クライアント 1-23
 - 図 29-3
 - 直接接続と間接接続 30-2
 - プログラム・インタフェース 7-26
 - 分散処理 29-3
 - 分散データベース 30-2
- クラスタ
 - ROWID 8-7
 - 概要 8-28
 - 格納形式 8-31
 - キー 1-44, 8-29, 8-32
 - NULL の索引付けに対する影響 8-7
 - 記憶領域パラメータ 8-4
 - クラスタ化するデータの選択 8-31
 - 結合 8-31, 20-44, 20-63, 20-46
 - 索引 8-16, 8-32
 - 走査 20-47
 - ハッシュとの対比 8-33
 - 走査 6-4, 20-40, 20-44
 - 結合 20-46
 - ハッシュ 20-45, 20-47
 - 定義 1-44
 - ディクショナリ・ロック 23-28
 - ハッシュ 8-33
 - 記憶領域 8-33
 - 索引との対比 8-33
 - 衝突の解決 8-35
 - 走査 20-41, 20-45, 20-47
 - 領域の割当て 8-37
 - ルート・ブロック 8-37
 - パフォーマンスの考慮事項 8-31
 - パラメータの設定 8-31
- クラスタ・キー 1-44, 8-29
- クラスタ化コンピュータ・システム
 - Oracle Parallel Server 5-2
- クラスタ結合 20-63
- グループ、インスタンス 22-14
- グループ・コミット 7-11
- グローバル・スキーマ・オブジェクト名 1-46, 30-6
- グローバル・データベース名
 - 共有プール 6-11
- グローバル索引 9-23, 9-25

パーティションの管理 9-36
クロス結合 20-12

け 計画

OR 演算子 20-18
SQL の実行 14-3, 14-11
結合 20-59, 20-65
スター型変換 20-74
ビューのアクセス 20-26, 20-29, 20-30
ビューの結合 20-32
複合問合せ 20-34, 20-35, 20-36
複合文 20-21

ゲートウェイ 30-8

結合

外部 20-12
NULL に対する NULL 以外の値 20-68
クラスタ 8-31, 20-44, 20-63
検索 20-46
クロス 20-12
最適化 20-66
実行計画 20-59
選択表示結合ビュー 20-22
ソート / マージ 20-61
コストベース最適化 20-66
例 20-52
直積 20-12
定義 20-12
等価結合 20-12
ネストされたループ 20-60
コストベース最適化 20-66
ハッシュ結合 20-64
半結合 20-70
非結合 20-70
非同レベル結合 20-12
ビュー 1-42, 8-13
ビューにカプセル化 1-42, 8-11
副問合せに変換する 20-20

結合ビュー 8-13

権限

RESTRICTED SESSION 25-13
解析時にチェックされる 14-11
概要 1-29, 26-2
監査の使用方法 1-33, 27-7
管理者

監査されない 27-4

システム 26-2

概要 1-30

付与と取消し 26-2

ユーザー定義型 12-10

スキーマ・オブジェクト 26-3

DML 操作と DDL 操作 26-4

概要 1-30

パッケージ 26-8

付与と取消し 26-4

プロシージャ 26-6

データベースの起動または停止 5-3

トリガー権限 26-7

取消し 26-2, 26-4

オブジェクト依存性 19-6

パーティション表とパーティション索引 9-38

ビュー 26-5

作成 26-5

使用 26-6

付与 1-30, 26-2, 26-4

例 26-8, 26-9

プロシージャ 26-6

作成と変更 26-7

実行 17-16, 26-6

パッケージ内 26-8

ユーザー定義型

ADMIN OPTION付きのEXECUTE ANY TYPE
12-10

ALTER ANY TYPE 12-10

CREATE ANY TYPE 12-10

CREATE TYPE 12-10

DELETE 12-11, 12-13

DROP ANY TYPE 12-10

EXECUTE 12-10, 12-12, 12-13

EXECUTE ANY TYPE 12-10

GRANT オプション付きの EXECUTE 12-10

INSERT 12-11, 12-12

SELECT 12-11, 12-12

UPDATE 12-11, 12-12

システム権限 12-10

使用 12-10, 12-14

ピン時のチェック 12-12

列レベル、オブジェクト表に対する 12-13

ロールによって獲得される 12-10

ロール 26-10

- 制限 26-13
- ロールとしてグループ化 1-30
- こ
- 更新
 - 更新可能性、ビューの 8-13, 18-11
 - 位置の透過性 30-14
 - オブジェクト・ビュー 13-4
 - オブジェクト・ビューの更新可能性 13-5
 - 更新可能な結合ビュー 8-13
 - 頻繁に更新が行われる環境 23-8
 - 分散 30-10
 - リモート 30-10
 - レプリケートされたデータでの競合 31-23
- 更新可能スナップショット 31-14
- 構造化問合せ言語 (SQL) 1-47, 14-2
 - SQL も参照
- 構造体
 - スキーマ・オブジェクト 8-2
 - 物理 1-5, 1-10
 - データ・ファイル 3-11
 - ロック 23-26
 - 論理 1-5, 1-8
 - 表領域 3-3
- 高速ウォームスタート 28-4
- 高速コミット 7-10
- 高速全索引走査 20-41
- 高速トランザクション・ロールバック 28-10
- 高速リフレッシュ 31-8
- コール
 - Oracle コール・インタフェース 7-27
 - リモート・プロシージャ 30-11
- 互換性 1-4
- コストベース最適化 20-6
 - 統計 20-38
 - ヒストグラム 20-7
- コミット、トランザクション
 - 概要 1-49
 - グループ・コミット 7-11
 - 高速コミット 7-10
 - 実現 7-10
 - 定義 15-2
 - パラレル DML 22-30
- コミット読み込みの分離 23-7
- コミット読み込み分離 23-6

- コレクション 11-9
 - 可変配列 (VARRAY) 11-9
 - ネストした表 11-10
- コレクションのメソッド
 - コンストラクタ・メソッド 1-52
- 混合構成
 - アドバンスド・レプリケーション 31-15
- コンストラクタ・メソッド 1-52, 11-5, 12-4
 - リテラル起動 12-7
- コンパイル、オブジェクト型 12-13
- コンパイル済み PL/SQL 17-15
 - 疑似コード 17-16, 18-17
 - 共有プール 14-15
 - 再コンパイル 17-17
 - トリガー 18-17
 - プロシージャ 17-8
 - 利点 17-8
- コンパイル済みトリガー 18-17
- さ
- サーバー 1-23
 - 共有 1-17
 - クライアント / サーバー・アーキテクチャ 29-2
 - 専用 1-17, 7-17
 - マルチスレッド・サーバーとの比較 7-19
 - 専用サーバー・アーキテクチャ 7-16
 - 定義 1-23
 - プロセス 1-16
 - マルチスレッド 1-17
 - アーキテクチャ 7-16, 7-19
 - 専用サーバーとの比較 7-19
 - プロセス 7-14, 7-19, 7-23
- サーバー・プロセス 1-16, 7-5
- 災害時回復 28-22
- 再帰 SQL
 - カーソル 14-6
- 最低使用頻度アルゴリズム (LRU)
 - 共有 SQL プール 6-8, 6-10
 - 全表走査 6-4
 - ディクショナリ・キャッシュ 4-4
 - データベース・バッファ 6-3
 - ラッチ 7-8
- 最適化 20-2
 - DISTINCT 20-23
 - GROUP BY ビュー 20-23

NULL に対する NULL 以外の値 20-68
 PL/SQL 20-39
 SQL 文のタイプ 20-11
 アプローチの選択 20-38
 コストベース 20-6, 20-66
 アクセス・パスの選択 20-55
 ヒストグラム 20-7
 リモート・データベース 20-37
 例 20-56
 索引作成 8-16
 式と述語の変換 20-12
 実行される操作 20-11
 手動 20-39
 推移律 20-15
 説明 20-2
 選択表示結合ビュー 20-22
 問合せの選択性 20-56
 統計 20-38
 パーティション索引 9-26
 パーティションの切詰め 9-3
 索引 9-26
 パーティションの実行計画 9-9, 9-11
 パラレル SQL 22-8
 半結合 20-70
 ヒント 20-39, 20-41
 複合ビューのマージ 20-23
 分散型の SQL 文 20-37
 文へのビューのマージ 20-22
 マージなし 20-32
 ルールベース 20-10, 20-66
 アクセス・パスの選択 20-58
 例 20-58
 サイト自律性 1-24, 30-15
 作業負荷の不整 22-14
 索引 1-43, 8-16
 B* ツリー構造 8-19
 Index Unusable (IU) 9-37
 LONG RAW データ型では禁止 10-11
 NULL 8-7, 8-24
 REF 12-8
 ROWID 8-20
 位置 8-18
 一意 8-16
 一意走査 20-41
 一意でない索引 8-16
 オブジェクト識別子 12-5
 オブジェクト列の属性 12-8
 概要 1-43, 8-16
 格納形式 8-18
 キー 8-17
 UNIQUE キー制約 24-9
 主キー制約 24-11
 逆キー索引 8-20
 クラスタ 8-32
 削除 8-33
 走査 20-47
 表との対比 8-32
 グローバル索引 9-23, 9-36
 高速全走査 20-41
 最適化 20-17
 索引構成表 8-25
 作成
 既存の索引を使った 8-16
 整合性制約の施行 24-9, 24-11
 説明 1-43, 8-16
 走査 20-41
 MAX または MIN 20-52
 ORDER BY 20-53
 クラスタ・キー 20-47
 制限 20-54
 単一系列 20-48
 非有界範囲 20-51
 複合 20-48
 有界範囲 20-50
 ダイレクト・ロード・インサートの後の再作成 21-8
 内部構造 8-19
 パーティション 9-2, 9-21
 パーティション化のガイドライン 9-27
 パーティションについての権限 9-38
 パーティションの監査 9-39
 パーティションの管理 9-35
 パーティションの切詰め 9-3
 パーティションの再作成 9-36
 パーティション表 8-25
 パフォーマンス 8-16
 パラレル DDL 記憶領域 22-24
 パラレルの索引走査 22-5
 範囲走査 20-41
 ビットマップ索引 8-21, 8-25

- NULL 8-7
- パラレル問合せおよび DML 8-22
- ビューでの使用 8-12
- 複合 8-17
 - 走査 20-48
- ブランチ・ブロック 8-19
- 文の変換 20-17
- ユーザー定義型 12-8
- リーフ・ブロック 8-19
- 連結 8-17
- ローカル索引 9-21, 9-35
- ロギングなしモード 21-7
- 索引構成表 8-25
 - アプリケーション 8-27
 - キュー表 16-8
 - 行オーバーフロー領域 8-26
 - 利点 8-26
- 索引セグメント 1-10, 2-15
- 削除 No Action 制約 24-15
- 削除カスケード制約 24-15
- 削除競合 31-23
- 参照
 - オブジェクト 19-2
 - キー 1-53, 24-12
 - パーティション 9-13
- 参照解除 11-8
 - 暗黙的 11-8
- 参照整合性 23-11, 24-11, 24-12
 - CASCADE 規則 24-3
 - PRIMARY KEY 制約 24-10
 - RESTRICT 規則 24-3
 - SET DEFAULT 規則 24-3
 - SET NULL 規則 24-3
 - 一部が NULL の外部キー 24-15
 - 自己参照型制約 24-14, 24-17
 - 例 24-17
- し
- 自己参照的なスタイルのメソッドの起動 11-5
- システム・グローバル領域 (SGA) 6-2
 - REDO ログ・バッファ 6-5, 15-5
 - いつ割り当てられるか 6-2
 - 概要 1-15, 6-2
 - 共有および書込み可能 6-2
 - 共有プール 6-6

- 固定 6-3
- サイズ 6-11
 - 可変パラメータ 5-4
- 図解 5-2
- データ・ディクショナリのキャッシュ 4-4
- データベース・バッファ・キャッシュ 6-3
- 内容 6-3
- マルチスレッド・サーバーでの使用を制限する 25-11
- ロールバック・セグメント 15-5
- 割当て 5-5
- システム・モニター・プロセス (SMON) 7-12
 - Parallel Server 7-12, 22-31
 - 一時セグメントのクリーン・アップ 7-12
 - インスタンスの回復 28-4
 - 定義 1-18, 7-12
 - パラレル DML インスタンス回復 22-31
 - パラレル DML システム回復 22-31
 - ロールバック、トランザクション 28-10
- システム権限 26-2
 - ADMIN OPTION 12-10, 26-3
 - 権限も参照
 - 説明 26-2
 - 付与と取消し 26-2
 - ユーザー定義型 12-10
- システム制御文 1-48, 14-5
- システム変更番号 (SCN)
 - REDO ログ 7-10
 - いつ決定されるか 23-5
 - 定義 15-5
 - トランザクションのコミット 15-5
 - 読み込み一貫性 23-5
- 事前書込み 7-10
- 持続領域 6-8
- 実行計画
 - EXPLAIN PLAN 14-3
 - OR 演算子 20-18
 - SQL の解析 14-11
 - 位置 6-8
 - 概要 20-2
 - 結合 20-59, 20-65
 - 参照する 20-4
 - 実行順序 20-5
 - スター型変換 20-74
 - パーティションとパーティション・ビュー 9-9, 9-

- 11
- ビューのアクセス 20-26, 20-29, 20-30
- ビューの結合 20-32
- 複合問合せ 20-34, 20-35, 20-36
- 複合文 20-21
- 例 20-21
- 実行時領域 6-8
- 実表 1-42
 - ビューも参照
 - データ・ディクショナリ 4-2
- 自動リフレッシュ
 - リフレッシュ・グループ 31-9
 - リフレッシュ間隔 31-9
- シノニム 19-7
 - オブジェクトから権限を継承する 26-3
 - 概要 1-43
 - 使用方法 8-15
 - 制約が間接的に影響を与える 24-4
 - 説明 8-15
 - データ・ディクショナリ・ビュー 4-4
 - パブリック 8-15
 - プライベート 8-15
- シャドウ・プロセス 7-17
- シャドウ列グループ 31-25
- 主キー 1-53, 24-10
 - 検索 20-46
 - 最適化 20-20
 - 定義 24-3
 - 利点 24-10
- 主キー・スナップショット 31-11
- 述語
 - パーティションの切詰め 9-3
 - 索引 9-26
 - ビュー問合せの最適化 20-22
 - ビューへの追加 20-25, 20-30
 - 例 20-25, 20-28
- 手動リフレッシュ 31-10
- 手動ロック 1-23, 23-29
- 順序 1-42, 8-14
 - CHECK 制約が禁止する 24-16
 - 監査 27-7
 - 数値の生成 8-14
 - 数値の長さ 8-14
 - 調整生成 31-24
 - 表からの独立 8-14

- 順序メソッド 1-52, 11-6
- 障害 28-2
 - REDO ログファイルのアーカイブ 28-17
 - 回復も参照
 - インスタンス 1-35, 28-3
 - 回復 28-4
 - 説明 1-34, 28-2
 - 耐障害性 28-22
 - 提供されている保護対策 28-6
 - データベース・バッファ 28-8
 - 内部エラー
 - トレース・ファイルに記録される 7-14
 - ネットワーク 28-3
 - 文障害とプロセス障害 1-34, 7-12, 28-2
 - メディア 1-35, 28-5
 - ユーザー・エラー 1-34, 28-2
- 使用済みバッファ 6-3
 - 増分チェックポイント 7-9, 28-4
- 情報検索 (IR) アプリケーション
 - 索引構成表 8-27
- 情報の整理統合
 - アドバンスト・レプリケーション 31-14
- 初期化パラメータ
 - ALWAYS_ANTI_JOIN 20-70
 - ALWAYS_SEMI_JOIN 20-70
 - AQ_TM_PROCESS 16-4, 16-5
 - BUFFER_POOL_KEEP 6-5
 - BUFFER_POOL_RECYCLE 6-5
 - CLEANUP_ROLLBACK_ENTRIES 22-30
 - COMPLEX_VIEW_MERGING 20-23
 - DB_BLOCK_BUFFERS 6-4, 6-11
 - DB_BLOCK_LRU_LATCHES 7-8
 - DB_BLOCK_MAX_DIRTY_TARGET 7-9, 28-5
 - DB_BLOCK_SIZE 6-4, 6-11
 - DB_FILE_MULTIBLOCK_READ_COUNT 20-56, 20-66
 - DB_FILES 6-14
 - DB_NAME 28-18
 - DB_WRITER_PROCESSES 1-17, 7-8
 - DBWR_IO_SLAVES 7-9
 - DISTRIBUTED_TRANSACTIONS 7-12
 - FAST_FULL_SCAN_ENABLED 20-41
 - HASH_AREA_SIZE 20-65
 - HASH_JOIN_ENABLED 20-64
 - HASH_MULTIBLOCK_IO_COUNT 20-65

HI_SHARED_MEMORY_ADDRESS 6-12
 JOB_QUEUE_PROCESSES 16-7
 LGWR_IO_SLAVES 7-11
 LICENSE_MAX_SESSIONS 25-13
 LICENSE_SESSIONS_WARNING 25-13
 LOCK_SGA 6-12, 6-16
 LOCK_SGA_AREAS 6-12, 6-16
 LOG_ARCHIVE_START 28-16
 LOG_BUFFER 6-6, 6-11
 LOG_CHECKPOINT_INTERVAL 7-8
 LOG_CHECKPOINT_TIMEOUT 7-8
 LOG_FILES 6-14
 MTS_MAX_SERVERS 7-23, 7-24
 MTS_SERVERS 7-23
 NLS_LANGUAGE 9-14
 NLS_NUMERIC_CHARACTERS 10-6
 NLS_SORT 9-14
 OPEN_CURSORS 6-9, 14-6
 OPEN_LINKS 6-14
 OPTIMIZER_MODE 20-38
 PARALLEL_DEFAULT_MAX_SCANS 22-14
 PARALLEL_DEFAULT_SCANSIZE 22-14
 PARALLEL_MAX_SERVERS 22-7
 PARALLEL_MIN_PERCENT 22-14
 PARALLEL_MIN_SERVERS 22-6, 22-8
 PARALLEL_SERVERS_IDLE_TIME 22-8
 PUSH_JOIN_PREDICATE 20-69
 REMOTE_DEPENDENCIES_MODE 19-9
 ROLLBACK_SEGMENTS 2-23
 SHARED_MEMORY_ADDRESS 6-12
 SHARED_POOL_SIZE 6-6, 6-11
 SORT_AREA_RETAINED_SIZE 6-15
 SORT_AREA_SIZE 2-15, 6-15, 20-66
 SORT_DIRECT_WRITES 6-15
 SQL_TRACE 7-15
 STAR_TRANSFORMATION_ENABLED 20-75
 TRANSACTIONS 2-23
 TRANSACTIONS_PER_ROLLBACK_SEGMENT 2-23
 USE_INDIRECT_DATA_BUFFERS 6-12
 初期即時制約 24-19
 初期遅延制約 24-19
 ジョブ 7-2
 ジョブ・キュー 31-20
 スナップショットのリフレッシュ 31-10
 ジョブ・キュー・プロセス (SNP_n)
 自動スナップショット・リフレッシュ 31-9, 1-19, 31-10
 メッセージの波及 16-7, 7-13
 署名のチェック 19-9
 処理
 DDL 文 14-13
 DML 文 14-10
 概要 14-8
 問合せ 14-11
 パラレル SQL 22-2
 分散 1-23
 シリアル伝播 31-21
 シングル・タスク・モード 7-15
 シングル・プロセスのシステム (シングル・ユーザーのシステム) 7-2
 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 30-18
 親和性
 パーティション 22-36
 パラレル DML 22-37
 す
 スキーマ 25-2
 オブジェクト 8-2
 定義 25-2
 内容 8-2
 表領域との対比 8-2
 ユーザー定義データ型 11-12
 ユーザーとの対応付け 1-27, 8-2
 スキーマ・オブジェクト 8-1
 INVALID 状態 19-3
 依存性 19-2
 存在しない他のオブジェクト 19-7
 トリガー管理 18-14
 ビュー 8-13
 分散データベース 19-10
 失われた権限に依存する 19-6
 概要 1-9, 1-41, 8-2
 監査 1-33, 27-7
 グローバルな名前 30-6
 権限 26-3
 作成
 表領域の割当て制限が必要 25-8

- 情報 4-2
- 定義 1-5
- データ・ファイルとの関連 3-12, 8-2
- デフォルトの表領域 25-7
- トリガーの依存性 18-17
- 取り消した表領域内 25-8
- 分散データベースにおける名前 30-6
- ユーザー定義型 11-3
- スキーマ・オブジェクト権限 26-3
 - DML 操作と DDL 操作 26-4
 - 概要 1-30
 - ビュー 26-5
 - 付与と取消し 26-4
- スキーマ名
 - データベース内で一意の 30-6
 - 分散データベースにおける 30-6
 - 列名の修飾 12-3
- スター型変換 20-72
 - 制限 20-75
 - 例 20-73
- スター問合せ 20-72
 - 拡張スター・スキーマ 20-72
 - 索引 20-71
 - チューニング 20-71
 - ヒント 20-72
- スタック領域 6-13
- スタンバイ・データベース 28-22
- ストアド・ファンクション 1-43, 17-2, 17-6
- ストアド・プロシージャ 1-43, 14-14, 17-2, 17-6
 - プロシージャも参照
 - コール 14-17
 - トリガーとの対比 18-2
 - 変数と定数 14-16
 - 無名ブロックと対比 17-8
- スナップショット
 - ROWID 31-11
 - グループ 31-18
 - 更新 31-8
 - 更新可能 31-14
 - サイト 31-18
 - 主キー 31-11
 - 単純
 - 構造 31-7
 - 定義問合せ 31-7
 - 複合 31-10

- 副問合せ 31-7
- 読み込み専用 31-6
- リフレッシュ 7-13, 31-8, 31-10
- ログ 31-9
- スナップショットの定義問合せ 31-7
- スナップショットのリフレッシュ
 - 完全メディア回復 31-8
 - グループ 31-9
 - リフレッシュ間隔 31-9
 - 高速リフレッシュ 31-8
 - ジョブ・キュー・プロセス (SNPn) 1-19, 7-13
 - 自動スナップショット・リフレッシュ 31-9
 - スナップショット・ログ 31-9
- スナップショット読み込み時期 23-10
- スループット 20-6
 - コストベースのアプローチ 20-38
- スレッド
 - マルチスレッド・サーバー 7-14, 7-19

せ

- 世紀 10-8
- 制御ファイル 1-12, 28-18
 - 回復 1-37
 - 概要 1-12, 28-18
 - 記録される変更内容 28-18
 - 指定方法 5-4
 - 多重化 1-37, 28-19
 - チェックポイント 28-19
 - データベースのマウントでの使用 5-6
 - 内容 28-18
 - バックアップ 28-21
 - 物理データベース構造 1-5
- 制限
 - ダイレクト・ロード・インサート 21-9
 - パーティション
 - 拡張パーティション表名 9-40
 - データ型 9-11, 9-15
 - ビットマップ索引 9-11
 - パーティション・ビュー 9-10
 - パラレル DML とダイレクト・ロード・インサート 22-33
- 制限付き ROWID 形式 10-12
- 制限モード
 - インスタンスの起動 5-5
- 整合性規則 1-40

- パラレル DML の制限事項 22-34
- 整合性制約 24-2
 - 制約も参照
 - デフォルトの列値 8-8
- 静的所有権 31-23
- 制約 1-52
 - CHECK 24-16
 - FOREIGN KEY 1-53, 24-11
 - NOT NULL 24-6, 24-10
 - PRIMARY KEY 1-53, 24-10
 - UNIQUE キー 1-53, 24-7
 - 一部が NULL 24-10
 - アプリケーションが違反を検出できる 24-6
 - 一時的に使用禁止にする 24-6
 - 違反した場合どうなるか 24-4
 - オブジェクト表 12-7
 - 概要 1-52
 - 索引による施行 8-18
 - PRIMARY KEY 24-11
 - UNIQUE 24-9
 - 参照制約
 - 更新の効果 24-15
 - 自己参照型 24-14
 - 施行のメカニズム 24-17
 - 制約を施行する 24-20
 - 制約を使用可能または使用禁止にする 24-20
 - 代替処置 24-5
 - タイプのリスト 1-53, 24-1
 - 定義 8-4
 - デフォルト値 24-18
 - トリガーとの対比 18-5
 - トリガーは違反できない 18-14
 - パフォーマンスに対する効果 24-6
 - パラレル CREATE TABLE 22-19
 - ビューでは定義できない 8-10
 - 評価されるタイミング 8-8
- 制約を施行する 24-20
- 制約を使用可能にする 24-20
- 制約を使用禁止にする 24-20
- 西暦 2000 年 10-8
- セーブポイント 1-50, 15-7
 - 暗黙的 15-4
 - 概要 1-50
 - 説明 15-7
 - ロールバック 15-6

- セキュリティ 1-30, 25-2
 - アプリケーションを使った施行 1-31
 - 監査 27-2, 27-6
 - 監査データの削除 4-5
 - 管理者権限 5-3
 - 施行のメカニズム 1-28
 - システム 1-27, 4-3
 - 説明 1-27
 - データ 1-27
 - データの暗号化 30-17
 - ドメイン 1-29, 25-2
 - 任意アクセス制御 1-28, 25-2
 - パスワード 25-4
 - ビュー 8-11
 - ビューによる強化 26-6
 - プログラム・インタフェースでの実施 7-26
 - プロシージャによる強化 26-7
 - 分散データベース 30-15
 - メッセージ・キュー 16-5
 - ユーザー・アクションの監査 1-32
- セキュリティ・ドメイン 1-29, 25-2
 - 使用可能なロール 26-12
 - 表領域の割当て制限 25-7
- セグメント 1-10, 2-14
 - 一時 1-10, 2-15
 - SMON によるクリーン・アップ 7-12
 - 削除 2-14
 - 操作 2-15
 - パラレル INSERT 21-8
 - 表領域 2-14, 2-16
 - 割当て 2-15
 - 割当て制限は無視される 25-8
 - エクステンツの割当て解除 2-12
 - 概要 1-10, 2-14
 - 索引 2-15
 - 定義 2-3
 - データ 2-14
 - ヘッダー・ブロック 2-10
 - ロールバック 2-16
- セッション
 - PARALLEL DML を有効にする 22-28
 - PGA 内のスタック領域 6-13
 - 監査オプションはいつ有効になるか 27-5
 - 時間制限 25-11
 - 情報が格納される場所 6-13

- セッション別監査 27-9
- 接続との比較 7-4
- 定義 7-5, 27-9
- 同時セッションに対する制限 1-32
 - ライセンスによる 25-13
- トランザクション分離レベル 23-30
- パッケージの状態 19-6
- ユーザー当たりの制限 25-11
- リソース制限 25-9
- セッション制御文 1-48, 14-5
- 接続
 - 埋込み SQL 14-5
 - 管理者権限での 5-3
 - 制限 5-5
 - セッションとの比較 7-4
 - 定義 7-4
 - ユーザー名 25-2
 - リスナー・プロセス 7-14
- 接続性 1-4
- 全索引走査 20-41
- 全体データベース・バックアップ 1-38, 28-20
- 選択性、問合せの 20-56
- 選択表示結合ビュー 20-22
- 全表走査 20-40, 20-54
 - LRU アルゴリズム 6-4
 - 選択性 20-56
 - パラレル問合せ 22-5
 - マルチブロック読み込み 20-56
 - ルールベースのオプティマイザ 20-58
- 専用サーバー 7-17
 - 使用例 7-24
 - 定義 1-17
 - マルチスレッド・サーバー 7-19

そ

- 走査 20-40
 - 一意 20-41, 20-46, 20-47
 - クラスタ 20-44, 20-45, 20-46, 20-47
 - 索引 20-47
 - 高速全索引走査 20-41
 - 索引 20-41
 - MAX または MIN 20-52
 - ORDER BY 20-53
 - クラスタ・キー 20-47
 - 制限 20-54

- 選択性 20-56
- 単一列 20-48
- ビットマップ 20-42
- 非有界範囲 20-51
- 複合 20-48
- 有界範囲 20-50
- 全表 20-40, 20-54
 - LRU アルゴリズム 6-4
 - パラレル問合せ 22-5
 - マルチブロック読み込み 20-56
 - ルールベースのオプティマイザ 20-58
- ハッシュ・クラスタ 20-45, 20-47
- 範囲 20-41, 20-48
 - MAX または MIN 20-52
 - ORDER BY 20-53
 - 非有界 20-51
 - 有界 20-50
- 表走査と CACHE 句 6-4
- 増分チェックポイント 7-9, 28-4
- ソート / マージ結合 20-61
 - アクセス・パス 20-52
 - コストベース最適化 20-66
 - 例 20-52
- ソート・ダイレクト書き込み機能 6-15
- ソート領域 6-15
- 即時制約 24-19
- 属性、オブジェクト型の 11-2, 11-3, 11-4
- 阻止しているトランザクション 23-10
- 阻止しているトランザクションを待機するか 23-10
- ソフトウェア・コード領域 6-16
 - プログラムとユーティリティによる共有 6-16

た

- 大規模データベース (VLDB) 9-4
 - パーティション 9-4
 - パラレル SQL 22-2
- 大規模並列処理 (MPP)
 - 複数の Oracle インスタンス 5-2
- 耐障害性 28-22
- 対称型レプリケーション 31-3, 31-12
 - 概要 31-11
 - 使用方法 31-12
- 代替キー
 - 競合の検出 31-25
- タイプ

- データ型、オブジェクト型を参照
- タイムスタンプのチェック 19-9
- 大量パラレル処理 (MPP)
 - 親和性 22-6, 22-36, 22-37
 - パラレル SQL 実行 22-2
- ダイレクト・ロード・インサート 21-2
 - シリアル INSERT 21-3
 - 制限 21-9, 22-33
 - パラレル・ロードとパラレル INSERT の比較 21-2
 - パラレル INSERT 21-3
 - 領域管理 21-8
 - ロギング・モード 21-5
- 多重化
 - REDO ログ・ファイル 1-37
 - 回復 28-6
 - 制御ファイル 1-37, 28-19
- タスク 7-2
- 単純スナップショット
 - 構造 31-7
- ダンプ・ファイル
 - Export と Import 12-15
- 断片化
 - パラレル DDL 22-24

ち

- チェックポイント
 - DBW_n プロセス 7-11
 - DBW_n プロセスへの信号 7-8
 - 制御ファイル 28-19
 - 増分 7-9, 28-4
 - チェックポイント・プロセス (CKPT) 1-18, 7-11
 - 統計 7-11
- チェックポイント・プロセス (CKPT) 1-18, 7-11
- 遅延トランザクション 31-20
- 遅延制約
 - 初期遅延または初期即時 24-19
 - 遅延可能または遅延不可 24-19
- 調整順序生成 31-24
- 直積 20-12

つ

- 通信プロトコル 29-5

て

- 定義フェーズ、問合せの処理の 14-12
- 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・キャッシュ・ロック 23-28
- 停止 5-7, 5-8
 - SGA の割当て解除 6-2
 - 異常 5-5, 5-8
 - ステップ 5-7
 - ディスパッチャ・プロセスでは禁止 7-24
- 定数
 - 計算されるとき 20-13
 - ストアド・プロシージャ内 14-16
 - 比較 20-13
- ディスク障害 1-35, 28-5
- ディスクの親和性
 - パーティション 22-36
 - パラレル DML 22-37
- ディスクのストライブ化
 - 親和性 22-36
 - パーティション 9-7
- ディスク領域
 - データ・ファイルへの割当て 3-11
 - 表への割当ての制御 8-4
- ディスパッチャ・プロセス (Dnnn)
 - 7-20
 - Net8 を介したユーザー・プロセスの接続 7-14
 - 応答キュー 7-20
 - 起動と停止の禁止 7-24
 - セッション当たりの SGA 領域の制限 25-11
 - 説明 7-14
 - 定義 1-18
 - ネットワーク・プロトコル 7-14
 - リスナー・プロセス 7-14
- データ
 - アクセス 1-47
 - 制御 25-2
 - セキュリティ・ドメイン 25-2
 - メッセージ・キュー 16-5
 - 一貫性
 - 基礎となる原理 23-14
 - 手動ロック 23-29
 - 定義 1-50
 - トランザクション・レベル 23-6
 - 反復可能読み込み 23-6

読み込み一貫性 1-21
 ロック 23-3
 ロック動作の例 23-30
 整合性 1-20, 8-4, 24-2
 2 フェーズ・コミット 1-25
 CHECK 制約 24-16
 概要 1-52
 参照制約 24-3
 施行 24-3, 24-5
 タイプ 24-2
 パラレル DML の制限事項 22-34
 同時アクセス 23-2
 表への格納方法 8-4
 分散操作 1-24
 レプリケーション 1-25, 31-2
 ロック 23-19
 データ・ウェアハウス 20-70
 スター問合せ 20-70
 ビットマップ索引 8-21
 表データのリフレッシュ 22-27
 データ・オブジェクト番号
 拡張 ROWID 10-12
 データ・セグメント 1-10, 2-14, 8-4
 データ・ディクショナリ
 DUAL 表 4-6
 アクセス 4-2
 依存性の追跡 19-3
 オブジェクトを追加する 4-4
 監査証跡 (SYS.AUD\$) 4-5
 キャッシュ 6-10
 位置 6-6
 行キャッシュ 6-10
 更新 4-4
 構造 4-2
 最適化で使われるビュー 20-7
 使用方法 4-3
 表と列の定義 14-11
 所有者 4-3
 接頭辞が ALL のビュー 4-6
 接頭辞が DBA のビュー 4-6
 接頭辞が USER のビュー 4-5
 定義 1-46, 4-2
 統計データ 20-38
 パーティションの統計 9-11
 動的パフォーマンス表 4-6
 内容 4-2, 6-10
 プロシージャ 17-16
 バックアップ 28-21
 パブリック・シノニム 4-4
 ビューの接頭辞 4-5
 プロシージャの有効性 17-16
 ロック 23-26
 データ・ディクショナリ・ビューの接頭辞 4-5
 データ・ファイル
 ROWID での表示 10-12, 10-13
 オフラインにする 3-12
 オンライン表領域またはオフライン表領域にある
 3-12
 回復不能 28-12
 概要 1-8, 1-10, 3-11
 制御ファイルに名前がある 28-18
 データ・ファイル 1 内のディクショナリ 28-21
 内容 3-11
 バックアップ 28-21
 パラレル回復 28-13
 表領域との関連 3-2
 物理データベース構造 1-5
 読み込み専用 3-9
 回復 28-5
 読み込み専用表領域 3-10
 データ・ブロック 1-9, 2-2
 ROWID での表示 10-12, 10-13
 空きデータ・ブロックの結合 2-12
 空きリスト 2-9
 空き領域の制御 2-5
 エクステンツへの割当て 2-11
 概要 2-2
 行ディレクトリ 8-6
 行の格納方法 8-5
 行を挿入できる領域 2-8
 クラスタ化 8-31
 クラスタによる共有 8-28
 形式 2-3
 ディスクへ書き込む 7-8
 ハッシュ・キー 8-37
 バッファ・キャッシュに格納される 6-3
 メモリーにキャッシュされる 7-8
 読み込み専用トランザクション 23-30
 データ・ブロック内の空き領域の圧縮 2-9
 データ・モデル 1-40

- データ・ロック
 - 存続期間 23-14
 - 段階的な拡大 23-15
 - 変換 23-15
- データ型 10-2, 10-15
 - ANSI 10-17
 - BOOLEAN 10-2
 - CHAR 10-3
 - DATE 10-7
 - DB2 10-17
 - LOB データ型 10-9
 - BFILE 10-10
 - BLOB 10-9
 - CLOB および NCLOB 10-9
 - デフォルトのロギング・モード 21-7
 - LONG 10-4
 - 記憶領域 8-7
 - MLSLABEL 10-15
 - NCHAR および NVARCHAR2 10-4
 - NUMBER 10-5
 - PL/SQL 10-2
 - RAW および LONG RAW 10-10
 - REF 11-8
 - ROWID 10-11
 - SQL/DS 10-17
 - VARCHAR 10-3
 - VARCHAR2 10-3
 - オブジェクト型 1-40, 11-3
 - コレクション 11-9
 - 使用可能なデータ型のリスト 10-2
 - ネストした表 8-9, 11-10
 - 配列型 11-9
 - 表との関連 8-3
 - 変換
 - Oracle 以外の型 10-17
 - Oracle データ型から別の Oracle データ型へ 10-18
 - プログラム・インタフェース 7-26
 - マルチメディア 11-3
 - 文字 10-2, 10-9
 - ユーザー定義 11-1, 11-3
 - 要約 10-15
 - 列 1-41
- データ所有権モデル 31-23
 - 共有所有権 31-24
 - 静的所有権 31-23
 - 動的所有権 31-23
 - プライマリ所有権 31-23
- データ操作言語 (DML)
 - 監査 27-6
 - 権限制御 26-4
 - 取得されるロック 23-24
 - 説明 14-3
 - 定義 1-48
 - トリガー 18-3, 18-15
 - パーティション・ロック 9-28
 - パラレル DML 22-3, 22-26
 - パラレル DML のためのトランザクション・モデル 22-29
 - 副問合せの直列可能分離 23-13
 - 分散トランザクション 30-10
 - 文の処理 14-10
- データ定義言語 (DDL)
 - DBMS_SQL での解析 14-18
 - 暗黙のコミット 15-4
 - 監査 27-6
 - 説明 14-4
 - 定義 1-48
 - パラレル DDL 22-3
 - 文の処理 14-13
 - ロールと権限 26-13
 - ロック 23-26
- データベース
 - アーカイブのモード 28-14
 - アクセス制御
 - 概要 1-47
 - セキュリティ・ドメイン 25-2
 - パスワード暗号化 25-4
 - オープン 5-6
 - ロールバック・セグメントの取得 2-23
 - オープンとクローズ 5-2
 - 回復 1-33, 28-2
 - 拡張性 22-2, 22-27, 29-4
 - 起動 5-2
 - 強制実行 5-8
 - クローズ 5-8
 - インスタンスの異常終了 5-8
 - グローバル・データベース名 30-4
 - 構成 5-3
 - サイズ

- 判別する方法 3-6
- 使用の制限 25-9
- スキーマが組み込まれている 25-2
- スタンバイ 28-22
- 制御ファイルに格納される名前 28-18
- 定義 1-7, 1-8
- 停止 5-7
- ディスマウント 5-8
- バックアップ 1-38, 28-19
- 物理構造 1-5, 1-10, 2-2
 - ROWID を使って明らかにする 10-13
- 分散 1-24, 30-1
 - 2 フェーズ・コミット 1-25
 - 概要 1-23, 1-24, 30-1
 - グローバル・データベース名の変更 6-11
 - サイト自律性 30-15
 - ノード 1-24, 30-2
 - 表のレプリケーション 1-25
 - 文の最適化 20-37
- マウント 5-6
- 論理構造 1-5
- 論理構造 (オブジェクト) 1-8
- データベース・スキーマのオブジェクト 1-5
 - スキーマ・オブジェクトも参照
- データベース・トリガー 1-53, 18-1
 - トリガーも参照
- データベース・バッファ
 - 空き 6-3
 - 書込み 7-8
 - キャッシュのサイズ 6-4
 - クリーン 7-8
 - コミット、トランザクションの 7-10
 - 使用済み 6-3, 7-8
 - 使用中 6-3
 - 定義 1-15, 6-3
 - トランザクションをコミットした後 15-6
 - バッファ・キャッシュ 6-3, 7-8
 - 複数のバッファ・プール 6-5
- データベース・ライター・プロセス (DBW_n) 7-8
 - アクティブになる時 7-8
 - 概要 1-17
 - 最低使用頻度アルゴリズム (LRU) 7-8
 - 事前書込み 7-10
 - チェックポイント信号 7-8
 - チェックポイント時のディスクへの書込み 7-11

- 定義 7-8
- トレース・ファイル 28-6
- 複数の DBW_n プロセス 7-8
- 複数の I/O プロセス 7-9
- メディア障害 28-6
- データベース・リンク 1-46
 - 概要 30-6
 - 定義 1-46
- 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 11-2
 - 原理 1-40
- データベース管理者 (DBA)
 - DBA ロール 12-10, 26-14
 - データ・ディクショナリ・ビュー 4-6
 - 認証 25-6
 - パスワード・ファイル 25-7
 - バックアップおよびリカバリに対する責任 28-2
- データベースの構成
 - パラメータ・ファイル 5-3
 - プロセスの構成 7-2, 7-15
- データ変換
 - ANSI データ型 10-17
 - SQL/DS および DB2 データ型 10-17
 - プログラム・インタフェース 7-26
- デッドロック
 - 回避 23-18
 - 検出 23-17
 - 人工 7-23
 - 定義 23-16
 - 分散トランザクション 23-17
- デフォルト値 8-8
 - 制約が与える影響 8-8, 24-18
 - ユーザー定義型 12-7
- 伝播
 - シリアル 31-21
 - パラレル 31-21
- と
- 問合せ
 - DML 14-3
 - IN 副問合せの最適化 20-23
 - ROWID による表走査の平行化 22-3
 - 一時セグメント 2-15, 14-12
 - 位置の透過性 30-14

- 記述フェーズ 14-12
- 行のフェッチ 14-11
- 処理 14-11
- スター問合せ 20-70
- 選択性 20-56
- 定義 20-11
- 定義フェーズ 14-12
- デフォルト・ロック 23-25
- トリガーの使用方法 18-15
- パーティションによってパラレル化される索引走査 22-5
- パラレル処理 22-2
- 非定型 22-22
- ビュー問合せとのマージ 8-12
- ビュー問合せの最適化 20-22
- ビューとして格納 1-41, 8-10
- フェーズ 23-5
- 複合
 - OR を複合問合せに変換する 20-17
 - 最適化 20-34
 - 定義 20-12
 - 分散またはリモート 30-10
 - 読み込み一貫性 1-22, 23-5
- 同一キー索引 9-22, 9-25
- 同一行の書き込みが書き込みを阻止するか 23-10
- 同一レベル・パーティション化 9-17
- 等価結合
 - クラスタ結合 20-63
 - ソート / マージ 20-61
 - 定義 20-12
 - ハッシュ結合 20-64
- 同期データ伝播 31-27
- 統計
 - ANALYZE コマンド 20-38
 - オプティマイザでの使用 20-6, 20-7
 - キューイング 16-7
 - チェックポイント 7-11
 - パーティション表とパーティション索引 9-11
- 同時実行性
 - 制限 1-32, 21-9
 - データベース当たりの 25-13
 - ユーザー当たりの 25-11
 - 説明 23-2
 - ダイレクト・ロード・インサート 21-9
 - 定義 1-20
 - トランザクション 23-14
 - パーティションのメンテナンス 9-31
 - ロックを使って施行する 1-22
- 動的 SQL
 - DBMS_SQL パッケージ 14-18
- 動的所有権 31-23
- 動的パーティション化 22-6
- 動的パフォーマンス表 (V\$ 表) 4-6
- ドライバ 7-27
- トランザクション 1-48, 15-1
 - アドバンスト・キューイング 16-2
 - アプリケーションの終了 15-5
 - インダウト
 - 自動解決 1-25, 5-7, 15-8
 - 手動による解決 1-25
 - 部分的に使用可能なセグメントの使用 2-26
 - ロールバック・セグメント 2-21
 - ロールバック・セグメントのアクセスの制限 2-26
- 開始 15-4
- 概要 1-48
- コミット 1-49, 7-10, 15-4, 15-5
 - グループ・コミット 7-11
 - ロールバック・セグメントの使用 2-18
- コミット前に書き込まれる REDO ログ・ファイル 7-10
- システム変更番号 7-10
- システム変更番号を割り当てる 15-5
- 終了 15-4
 - 一貫したデータ 14-13
- 手動ロック 23-30
- セーブポイント 1-50, 15-7
- 説明 15-2
- 直列可能 23-6
- 定義と制御 14-13
- データ・ブロック内で使われる領域 2-5
- デッドロック 15-4, 23-16
- 同時実行性 23-14
- トランザクション制御文 14-5
- トランザクションの制御 14-13
- トリガー 18-15
- パラレル DML での 2 フェーズ・コミット 22-30
- 非同期処理 16-2
- 分散 1-22
 - 2 フェーズ・コミット 1-25, 15-7, 30-12

- 自動解決 7-12
- デッドロック 23-17
- パラレル DML の制限事項 22-35
- 文レベルのロールバック 15-4
- 読み込み一貫性 1-21, 23-6
- 読み込み専用 1-22, 23-6
 - ロールバック・セグメントには割り当てられない 2-17
- 離散トランザクション 14-14, 15-8
- ロールバック 1-49, 15-6
 - オフライン表領域 2-27
 - 部分的な 15-6
 - ロールバック・セグメントの使用 2-17
 - ロールバック・セグメント 2-17
 - ロールバック・セグメントへの書き込み 2-18
 - ロールバック・セグメントへの配分 2-18
 - ロールバック・セグメントへの割当て 2-17
- トランザクション集合の一貫性 23-9, 23-10
- トランザクション制御文 1-48, 14-5
- トランザクション表 2-17
 - 回復時にリセット 7-12
- トリガー 1-53, 18-1, 19-7
 - AFTER トリガー 18-8
 - BEFORE トリガー 18-8
 - INSTEAD OF トリガー 18-11
 - オブジェクト・ビュー 13-5
 - INVALID 状態 19-3, 19-6
 - Oracle Forms トリガー 18-4
 - UNKNOWN では起動されない 18-7
 - アクション 18-7
 - タイミング 18-8
 - 依存性の管理 18-17, 19-6
 - 使用可能なトリガー 18-14
 - イベント 18-6
 - 概要 1-53, 18-2
 - 各部分 18-5
 - カスケード 18-3
 - 監査 27-8
 - 記憶領域 18-17
 - 起動（実行） 18-2, 18-17
 - 実行されるステップ 18-13
 - タイミング 18-14
 - 必要な権限 18-17
 - 行 18-7
 - 共有 SQL 領域 6-9

- 実行する権限 26-7
- 使用可能または使用禁止 18-13
- 使用禁止にされたロール 26-13
- 使用方法 18-3
- スキーマ・オブジェクトの依存性 18-14, 18-17
- 制限 18-7, 22-35
 - ダイレクト・ロード・インサート 21-9
 - パラレル DML 22-33
- 制約が適用される 18-14
- 制約との対比 18-5
- タイプ 18-7
- データ・アクセス 18-15
- データ整合性の維持 1-53
- データ整合性の施行 24-4
- ビューでは定義できない 8-10
- 複数トリガーの起動順序 18-14
- プログラム・ユニット 1-51
- プロシージャとの対比 18-2
- 文 18-8
 - ユーザー定義型 12-9
 - 例 18-9, 18-12, 18-15
- 取消し 1-10
 - ロールバックも参照
- トレース・ファイル 7-14
 - DBW n 28-6
 - LGWR トレース・ファイル 7-10
- トレース・ファイルに記録される内部エラー 7-14

な

- 内容を保証しない書き込み 23-10
- 内容を保証しない読み込み 23-2, 23-10
- 名前付きユーザー・ライセンス 25-14

に

- 任意アクセス制御 1-28, 25-2
- 認証
 - Oracle 25-4
 - オペレーティング・システム 25-3
 - 説明 25-3
 - ネットワーク 25-4

ね

- ネストされたループ結合 20-60
- コストベース最適化 20-66

- ネストした表 8-9, 11-10
 - 索引 12-8
- ネットワーク
 - 2 タスク・モード 7-19
 - Net8 29-4, 30-4
 - Oracle Names 30-4
 - Oracle の使用 1-7, 1-26
 - クライアント / サーバー・アーキテクチャでの使用 29-2
 - 障害 28-3
 - 通信プロトコル 7-27, 29-5
 - ディスクパッチャ・プロセス 7-14, 7-20
 - ドライバ 7-27
 - ネットワーク認証サービス 25-4
 - 分散データベース 30-2, 30-4
 - リスナー・プロセス 7-14
- の
- ノード
 - パラレル・サーバーでのディスク親和性 22-36
 - 分散データベース 1-24
- は
- パーティション 9-2, 9-10
 - DML パーティション・ロック 9-28
 - EXCHANGE PARTITION 9-9
 - LONG および LONG RAW の制限事項 9-11
 - OLTP データベース 9-5
 - VLDB 9-4
 - 拡張パーティション表名 9-39
 - グローバル索引 9-23, 9-36
 - 索引のパーティション化 9-21, 9-27
 - 実行計画 9-9, 9-11
 - 親和性 22-36
 - 制限
 - 拡張パーティション表名 9-40
 - データ型 9-11, 9-15
 - ビットマップ索引 9-11
 - 同一キー索引 9-22
 - 同一レベル・パーティション化 9-17
 - 統計 9-11
 - 同時実行のメンテナンス操作 9-31
 - 動的パーティション化 22-6
 - パーティション・バウンド 9-13
 - パーティション化キー 9-12, 9-14
 - パーティション化の基本的なモデル 9-10
 - パーティションの切詰め 9-3
 - TO_DATE 書式マスク 9-11, 9-15
 - 索引 9-26
 - ディスクのストライプ化 22-36
 - ブロック範囲によるパラレル化 22-4
 - パーティションの再作成 9-36
 - パーティションの参照 9-13
 - パーティションの透過性 9-9
 - パーティション名 9-13
 - パラレル DDL 22-22
 - パラレル化のルール 22-19, 22-20
 - パラレル問合せ 22-3
 - ビットマップ索引 8-25
 - 表のパーティション化 9-19
 - 物理属性 9-20, 9-27
 - メンテナンス操作 9-29
 - 利点 9-4, 9-5
 - レンジ・パーティション化 9-11
 - ディスクのストライプ化 22-36
 - ローカル索引 9-21, 9-35
 - ロギングなしモード 21-7
- パーティション・ビュー 9-9
- パーティション化キー 9-12, 9-14
 - 複数列からなるキー 9-15
- 排他モード 2-24, 5-6
- 排他ロック
 - RX ロック 23-22
 - 行ロック (TX) 23-19
 - 表ロック (TM) 23-20
- バイナリ・データ
 - BFILE 10-10
 - BLOB 10-9
 - RAW および LONG RAW 10-10
- 配列
 - VARRAY のサイズ 11-10
 - 可変 (VARRAY) 11-9
- 配列処理 14-13
- バインド変数
 - 最適化 20-57
 - ユーザー定義型 11-12
- パスワード
 - アカウントのロック 25-5
 - 暗号化 25-4
 - 管理者権限 5-3

- 時間切れ 25-5
- データベース・ユーザーの認証 25-4
- パスワード・ファイル 25-7
- パスワードの再使用 25-5
- パスワードを指定した接続 7-5
- パスワードを指定しない接続 25-3
- 複雑さの検証 25-5
- ロールでの使用 1-31
- バックアップ
 - Export を使った補助 28-22
 - Recovery Manager 1-39, 28-10
 - 概要 1-33, 28-19
 - 制御ファイル 28-21
 - 全体データベース・バックアップ 1-38, 28-20
 - タイプ 1-38
 - データ・ファイル 28-21
 - パラレル 28-11
 - 部分 1-38, 28-21
 - 読み込み専用表領域 28-22
- バックエンド 29-2
- バックグラウンド・プロセス 1-17, 7-5, 7-6
 - プロセスも参照
 - 概要 1-17
 - 図 7-6
 - 説明 7-5
 - トレース・ファイル 7-14
- パッケージ 17-4, 17-9
 - 概要 1-43
 - 格納 17-15
 - 監査 27-7
 - キューイング 16-3
 - 共有 SQL 領域 6-9
 - 権限
 - 構成体ごとに分割 26-8
 - 実行 26-6, 26-8
 - 実行 14-15, 17-16
 - セッションの状態 19-6
 - 動的 SQL 14-18
 - パブリック 17-14
 - プライベート 17-14
 - プログラム・ユニット 1-51
 - 有効性 17-16
 - 利点 17-14
 - 例 17-10, 26-8, 26-9
 - ロック用 23-38
- ハッシュ・クラスタ 1-46, 8-33
 - 概要 1-46
 - 走査 20-41, 20-45, 20-47
- ハッシュ結合 20-64
 - HASH_AREA_SIZE パラメータ 20-65
 - HASH_MULTIBLOCK_IO_COUNT パラメータ 20-65
- 発注の例
 - オブジェクト型 11-2, 11-4
- バッファ
 - REDO ログ・バッファ 1-15, 6-5
 - データベース・バッファ・キャッシュ 1-15, 6-3, 7-8
 - 増分チェックポイント 7-9, 28-4
- バッファ・キャッシュ 6-3, 7-8
 - 拡張バッファ・キャッシュ (32 ビット) 6-12
 - 複数のバッファ・プール 6-5
- バッファ・プール 6-5
- パフォーマンス
 - DSS データベース 9-7, 22-27
 - I/O 9-7
 - Oracle Parallel Server と DML ロック 9-29
 - SGA のサイズ 6-11
 - 回復 28-5
 - クラスタ 8-31
 - グループ・コミット 7-11
 - 向上させる構造体 1-43
 - 索引作成 8-16
 - 実行計画の表示 20-4
 - 制約が与える影響 24-6
 - 同一キー索引と非同一次元索引 9-26
 - 動的パフォーマンス表 (V\$) 4-6
 - パーティション 9-7
 - パッケージ 17-14
 - パラレル回復 28-13
 - リソース制限 25-9
- パブリック・ロールバック・セグメント 2-22
- パラメータ
 - 各国語サポート 5-4
 - 記憶域 2-5, 2-11
 - 初期化 5-3
 - 初期化パラメータを参照
 - ロック動作 23-18
- パラメータ・ファイル 5-3
 - 起動時の使用 5-5

- 例 5-4
- パラレル・コーディネータ・プロセス 22-5
- パラレル・サーバー・プロセス 22-5
- パラレル・バックアップ操作 28-11
- パラレル・モード 5-6
- パラレル DDL
 - エクステンツの割当て 22-24
 - パーティション表とパーティション索引 22-22
 - パラレル化ルール 22-15
 - 並行性のタイプ 22-3
- パラレル DELETE 22-16
- パラレル DML 22-26
 - PARALLEL DML を有効にする 22-28
 - アプリケーション 22-27
 - 回復 22-30
 - 制限 22-33
 - トランザクション・モデル 22-29
 - パラレル化ルール 22-15
 - ビットマップ索引 8-22
 - 並行性のタイプ 22-3
 - 並行度 22-15, 22-17
 - リソースのロックとエンキュー 22-31
- パラレル SQL
 - Parallel Server 22-1
 - インスタンス・グループ 22-14
 - 最適化 22-8
 - 行をパラレル・サーバーに割り当てる 22-8
 - コーディネータ・プロセス 22-5
 - サマリー表またはロールアップ表 22-22
 - 実行計画にある操作 22-8
 - パラレル・サーバー・プロセスの数 22-6
 - パラレル化ルール 22-15
 - 並行度 22-12
 - マルチスレッド・サーバー 22-7
- パラレル UPDATE 22-16
- パラレル回復 28-11, 28-12
- パラレル伝播 31-21
- パラレル問合せ 22-2
 - インターオペレータ並行性 22-10
 - イントラオペレータ並行性 22-10
 - 全表走査 22-5
 - パーティション表とパーティション索引 22-3
 - パラレル化ルール 22-15
 - ビットマップ索引 8-22
- 半結合 20-70

- 反復可能読み込み 23-3
- 反復不能読み込み 23-2, 23-10

ひ

- 比較メソッド 11-6
- 非結合 20-70
- 非コミット読み込み 23-3
- ビジネス・ルール
 - アプリケーション・コードで施行する 24-5
 - ストアド・プロシージャを使って施行する 24-5
 - 制約を使って施行する 1-52
 - 利点 24-5
 - トリガーを使って施行 1-53
- ヒストグラム 20-7
- 必須アクセス制御 1-33
- ビットマップ索引 8-21
 - NULL 8-7, 8-24
 - スター型変換 20-73
 - 走査 20-42
 - パーティション表 9-11
 - パラレル問合せおよび DML 8-22
- 非同一キー索引 9-26
- 非同一レベル結合
 - 定義 20-12
- 非同期 I/O
 - パラレル回復 28-13
- 非同期処理 16-2
- 非同期レプリケーション 31-19
- ビュー 1-41, 8-10
 - INVALID 状態 19-3
 - NLS パラメータ 8-12
 - NULL に対する NULL 以外の値 20-68
 - SQL ファンクション 8-12
 - 依存性の状態 19-5
 - オブジェクト・ビュー 8-14, 13-1
 - 行オブジェクト 11-8
 - 更新可能性 13-5
 - 概要 1-41, 8-10
 - 格納方法 8-11
 - 監査 27-7, 27-8
 - 疑似列 18-12
 - 権限 26-5
 - 更新可能性 8-13, 13-5, 18-11
 - コンパイルの前提条件 19-5
 - 最適化 20-22

索引 8-12
式を含む 18-12
実表 1-42
実表の変更 19-5
使用方法 8-11
スキーマ・オブジェクトの依存性 8-13, 19-4, 19-7
制約が間接的に影響を与える 24-4
制約とトリガーは定義できない 8-10
セキュリティ・アプリケーション 26-6
選択表示結合ビュー 20-22
定義の展開 19-5
データ・ディクショナリ
 更新可能な列 8-13
 ユーザー・アクセス可能ビュー 4-3
パーティション・ビュー 9-9
パーティションの統計 9-11
ヒストグラム 20-9
複合ビューのマージ 20-23
変更 18-11
変更可能 18-11
本質的に変更可能 18-11
列の最大数 8-10

表

DUAL 4-6
依存ビューに対する影響 19-5
オブジェクト表 11-3, 11-7
 仮想 13-2
 索引 12-8
 制約 12-7
 トリガー 12-9
概要 1-41, 8-3
拡張パーティション表名 9-39
仮想またはビュー 1-42
監査 9-39, 27-7
キュー表 16-4, 16-8
クラスタ化 8-28
権限 26-4
索引 8-16
索引構成表 8-25
サマリー表またはロールアップ表 22-22
実表 1-42
 データ・ディクショナリでの使用 4-2
 ビューとの関連 8-11
整合性制約 24-2, 24-4
整合性制約を含む 1-52

制約を施行する 24-20
制約を使用可能または使用禁止にする 24-20
全表走査とバッファ・キャッシュ 6-4
使うトリガー 18-2
データ・ウェアハウスでのリフレッシュ 22-27
データの格納方法 8-4
動的パーティション化 22-6
ネストした表 8-9, 11-10
 索引 12-8
パーティション 9-2, 9-19
パーティションについての権限 9-38
ハッシュ 8-37
パラレル DDL 記憶領域 22-24
パラレル作成 22-22
パラレル問合せでの STORAGE 句 22-24
パラレルの表走査 22-3
ビューでの提示 8-10
表の別名 12-2, 12-3
表名
 列名の修飾 12-2, 12-3
表領域に含まれる 8-4
表領域の指定 8-4
領域割当ての制御 8-4, 21-8
履歴データ 22-28
列の最大数 8-10
レプリケーション 1-25, 31-2, 31-3
ロギングなしモード 21-7
ロック 9-28, 23-20, 23-21, 23-23
表ディレクトリ 2-4
表領域 3-3
 一時 1-31, 3-10
 ユーザーのデフォルト 25-7
 一時セグメントに使われる 2-14, 2-16
オブジェクト作成のデフォルト 1-31, 25-7
オフライン 1-9, 3-7, 3-12
 再マウント時にオフラインのまま 3-8
 索引データとの関係 3-9
 読み込み専用できない 3-10
オンライン 1-9, 3-7, 3-12
概要 1-8, 3-3
サイズ 3-6
スキーマとの対比 8-2
説明 3-3
データ・ファイルとの関連 3-2
表に対して指定する方法 8-4

- ユーザーからのアクセスを取り消す 25-8
- 読み込み専用 3-9
 - オブジェクトの削除 3-10
- ロギングなしモード 21-7
- ロック 23-29
- 割当て制限 1-31, 25-7
 - 制限ありと制限なし 25-8
 - デフォルトはない 25-8
- ヒント
 - INDEX 20-72
 - INDEX_FFS 20-41
 - MERGE 20-23
 - MERGE_AJ および HASH_AJ 20-70
 - MERGE_SJ および HASH_SJ 20-70
 - OPTIMIZER_MODE と OPTIMIZER_GOAL を上書き 20-39
 - ORDERED 20-66, 20-72
 - PARALLEL 22-12
 - PARALLEL_INDEX 22-12
 - PUSH_JOIN_PRED 20-69
 - STAR 20-72
 - USE_HASH 20-64
- ふ
- ファイル
 - ALERT とトレース・ファイル 7-10, 7-14
 - Oracle データベース 1-8, 1-10, 28-6
 - 制御ファイル、データ・ファイル、REDO ログ・ファイルも参照
 - オペレーティング・システム 1-5
 - 初期化パラメータ 5-3, 5-5
 - ダンプ・ファイルの Export と Import 12-15
 - パスワード 25-7
 - 管理者権限 5-3
- ファイル管理ロック 23-29
- ファジー読み込み 23-2
- ファンクション
 - PL/SQL
 - 権限 26-6, 17-2, 17-6
 - プロシージャも参照
 - 使用禁止にされたロール 26-13
 - パラレル DML の制限事項 22-35
 - プロシージャと対比 1-51, 17-2
 - SQL
 - CHECK 制約 24-16
- フェールオーバー・データベース 31-12
- フェッチする、問合せの行を 14-13
 - 埋込み SQL 14-5
- 不完全なオブジェクト型 12-14
- 複合索引 8-17
- 複合スナップショット 31-10
- 複合ビューのマージ 20-23
- 副問合せ 14-11
 - CHECK 制約が禁止する 24-16
 - DML 文
 - 直列可能分離 23-13
 - IN 副問合せの最適化 20-23
 - NOT IN 20-70
 - 問合せも参照
 - 結合に変換する 20-20
 - リモート更新 30-10
- 副問合せスナップショット 31-7
- 不整なパラレル DML 作業負荷 22-14
- 物理データベース構造 1-5
- 部分バックアップ 28-21
- 付与
 - EXECUTE ユーザー定義型 12-10
 - 権限とロール 26-2
- プライベート・ロールバック・セグメント 2-22
- プライベート SQL 領域
 - カーソル 6-9
 - 持続領域 6-8
 - 実行時領域 6-8
 - 説明 6-8
 - どのように管理されるか 6-9
- プライマリ所有権 31-23
- ぶら下がり REF 11-8
- フラッグ、規格化されていない機能 1-3, 14-6
- ブランチ・ブロック 8-19
- プリコンパイラ
 - FIPS フラガー 14-6
 - 埋込み SQL 14-5
 - カーソル 14-10
 - ストアド・プロシージャ 14-17
 - バインド変数 14-12
 - 無名ブロック 14-16
- プログラム・インタフェース 7-26
 - 2 タスク・モード 7-19
 - Oracle 側 (OPI) 7-27
 - 概要 1-19

- 構造 7-27
- シングル・タスク・モード 7-17
- ユーザー側 (UPI) 7-27
- プログラム・グローバル領域 (PGA) 1-16, 6-12
 - サイズ 6-14
 - 内容 6-13
 - 非共有と書込み可能 6-12
 - マルチスレッド・サーバー 7-23
 - 割当て 6-13
- プログラム・ユニット 1-42, 14-14, 17-2
 - 共有ブル 6-9
 - コンパイルの前提条件 19-5
- プロシージャ 14-14, 17-1, 17-6, 19-7
 - INVALID 状態 19-3, 19-6
 - 依存性の追跡 19-6
 - カーソル 14-16
 - 外部プロシージャ 14-18, 17-9
 - 格納 17-15
 - 監査 27-7, 27-8
 - 共有 SQL 領域 6-9
 - 権限
 - 作成または変更 26-7
 - 実行 26-6
 - バッケージ内での実行 26-8
 - コンパイルの前提条件 19-5
 - 実行 14-15, 17-16
 - 使用禁止にされたロール 26-13
 - ストアド・プロシージャ 14-14, 14-17, 17-2
 - セキュリティの強化 17-7, 26-7
 - トリガー 18-2
 - ファンクションと対比 1-51, 17-2
 - 無名ブロックと対比 17-8
 - 有効性 17-16
 - 利点 17-7
 - リモート・プロシージャ・コール 30-11
 - 例 17-6, 26-8, 26-9
- プロシージャ・レプリケーション 31-26
 - 競合の検出 31-26
 - ラッパー 31-26
- プロセス 7-2
 - Oracle 1-16, 7-5
 - シングル・プロセス Oracle 7-2
 - アーカイバ (ARCH) 1-18, 7-13
 - 回復時 28-13
 - 概要 1-16

- キュー・モニター (QMN_n) 1-19, 7-13, 16-4
- 構造 7-2
- サーバー 1-16, 1-23, 7-5
 - 共有 7-14, 7-23
 - 専用 7-17
- システム・モニター (SMON) 1-18, 7-12
- シャドウ 7-17
- 障害 28-3
- ジョブ・キュー (SNP_n)
 - スナップショットのリフレッシュ 31-10, 1-19, 7-13, 31-9
 - メッセージの波及 16-7
- 専用サーバー 7-23
- チェックポイント 7-8
- チェックポイント (CKPT) 1-18, 7-11
- ディスパッチャ (Dnm_n) 1-18, 7-14
- データベース・ライター (DBW_n) 1-17, 7-8
- トレース・ファイル 7-14
- バックグラウンド 1-17, 7-5
 - 図 7-6
- パラレル・コーディネータ 22-5
- パラレル・サーバー・プロセス 22-5
- プロセス・モニター (PMON) 1-18, 7-12
- 分散トランザクションの解決 7-12
- マルチ・プロセス Oracle 7-3
- マルチスレッド・サーバー 7-19
 - クライアントの要求 7-20
 - 人工デッドロック 7-23
- ユーザー 1-16, 7-4
 - PGA の割当て 6-13
 - サーバー・プロセスの共有 7-14
 - 障害からの回復 7-12
- リカバラ (RECO) 1-18, 7-12
 - インダウト・トランザクション 1-25
- リスナー 7-14
 - 共有サーバー 7-20
- ログ・ライター (LGWR) 1-17, 7-9
- ロック (LCK_n) 1-19, 7-13
- プロセス・グローバル領域 (PGA) 6-12
 - プログラム・グローバル領域を参照
- プロセス・モニター・プロセス (PMON)
 - 説明 1-18, 7-12
 - タイムアウト・セッションのクリーン・アップ 25-11
- ネットワーク障害 28-3

パラレル DML プロセス回復 22-30
 プロセス障害 28-3
 ブロック
 データベース 2-3
 データ・ブロックも参照
 無名 14-14, 17-8
 プロファイル
 いつ使うか 25-12
 概要 1-32
 パスワード管理 25-5
 フロントエンド 29-2
 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 1-23, 29-3
 説明 1-23, 29-2
 データ操作言語 14-10
 分散データベース 30-7
 分散データベース 30-1
 2 フェーズ・コミット 1-25, 30-12
 異種間 30-8
 依存スキーマ・オブジェクト 19-8
 概要 1-24, 30-2
 監査 27-5
 管理ツール 30-17
 クライアント/サーバー・アーキテクチャ 29-2
 グローバル・スキーマ・オブジェクト名 30-6
 異なる Oracle バージョン 30-7
 サーバーをクライアントとしても使える 29-2
 サイト自律性 30-15
 ジョブ・キュー・プロセス (SNP_n) 1-19, 7-13
 図 30-3
 データベース・リンク 30-6
 デッドロック 23-17
 透過性 30-13
 ノード 30-2
 表のレプリケーション 1-25, 31-2, 31-3
 読み専用 31-2, 31-4
 読みと更新 31-11
 分散更新 30-10
 分散問合せ 30-10
 文の最適化 20-37
 メッセージの波及 16-7
 リカバラ・プロセス (RECO) と 7-12
 リモート依存性 19-9
 リモート問合せと更新 30-10
 分散データベースでのネーム変換 30-6
 分散トランザクション
 2 フェーズ・コミット 1-25, 15-7
 最適化 20-37
 定義 30-11
 ノードへの文のルーティング 14-11
 パラレル DML の制限事項 22-35
 文トリガー 18-7
 トリガーも参照
 起動されるタイミング 18-14
 説明 18-8
 文へのビューのマージ 20-22
 分離レベル
 コミット読み 23-7
 設定 23-6, 23-29
 選択 23-11
 文レベルの読み一貫性 23-5
 へ
 並行度 22-15, 22-17
 問合せ操作間 22-10
 パラレル SQL 22-6, 22-12
 ページ 2-2
 ヘッダー
 行断片 8-5
 データ・ブロック 2-4
 別の行の書込みが書込みを阻止するか 23-10
 別名
 列名の修飾 12-2, 12-3
 変換、データの
 ANSI データ型 10-17
 SQL/DS および DB2 データ型 10-17
 プログラム・インタフェース 7-26
 変数
 埋込み SQL 14-5
 オブジェクト変数 13-4
 ストアド・プロシージャ内 14-16
 バインド変数
 最適化 20-57
 ユーザー定義型 11-12
 ほ
 保管後転送レプリケーション 31-19

ま

- マスター・グループ 31-17
- マスター・サイト 31-17
- マスター定義サイト 31-17
- マスター表
 - スナップショット・ログ 31-9
- マップ・メソッド 1-52, 11-6
- マルチ・プロセス・システム (マルチ・ユーザー・システム) 7-3
- マルチ・ブロック書込み 7-8
- マルチ・ユーザー環境 7-3
- マルチスレッド・サーバー 7-19
 - 7-20, 7-14
 - 共有サーバー・プロセス 7-14, 7-23
 - サーバー・プロセス 1-17, 7-14, 7-23
 - 使用例 7-25
 - 人工デッドロック 7-23
 - 制限された操作 7-24
 - 説明 7-16, 7-19
 - 専用サーバーとの比較 7-19
 - ディスパッチャ・プロセス 1-18, 7-14
 - 必要なプロセス 7-19
- マルチバージョン同時実行性制御 23-5
- マルチバージョンの一貫性モデル 1-21
- マルチマスター・レプリケーション 31-13
- マルチメディア・データ型 11-3
- マルチユーザー環境 1-2

む

- 無名 PL/SQL ブロック 14-14, 17-8
 - アプリケーション 14-16
 - ストアド・プロシージャと対比 17-8
 - ストアド・プロシージャのコール 14-17
 - 動的 SQL 14-18
 - パフォーマンス 17-9

め

- 明示的ロック 23-29
- メソッド
 - コンストラクタ・メソッド 11-5
 - リテラル起動 12-7
 - 比較メソッド 11-6
- メッセージ・キューイング 16-2
 - キュー・モニター・プロセス 1-19, 7-13, 16-4

- 間隔統計 16-7

- 実行の時間枠 16-5

- キュー表 16-4

- キュー表のエクスポート 16-8

- メッセージ 16-3

- リモート・データベース 16-7

- メッセージ「スナップショットが古すぎます」 23-5

- メディア障害 1-35, 28-5

- メモリー

- SQL 文のための割当て 6-10

- カーソル (文ハンドル) 1-16

- 拡張バッファ・キャッシュ (32 ビット) 6-12

- 仮想 6-16

- 共有 SQL 領域 6-8

- 構造 6-2

- 構造の概要 1-12

- システム・グローバル領域 (SGA)

- SGA のサイズ 6-11

- 開始アドレス 6-12

- 初期化パラメータ 6-11, 6-12

- 物理メモリーへのロック 6-12, 6-16

- 割当て 6-2

- ストアド・プロシージャ 17-8, 17-15

- ソート領域 6-15

- ソフトウェア・コード領域 6-16

- 内容 6-2

- プロセスでの使用 7-2

も

- モード

- 2 タスク 7-16, 7-17

- アーカイブ・ログ 28-14

- 共有 5-6

- シングル・タスク 7-15

- 排他 5-6

- 表ロック 23-20

ゆ

- 有効範囲付き REF 11-8, 12-5

- ユーザー 25-2

- PUBLIC ユーザー・グループ 25-8, 26-13

- アクセス権 25-2

- 一時表領域 1-31, 2-16, 25-7

- 監査 27-11

- 権限 1-29

- シングル・ユーザー Oracle 7-2
 - スキーマ 1-27, 25-2
 - スキーマに対応付けられる 8-2
 - セキュリティ・ドメイン 1-29, 25-2, 26-13
 - 専用サーバー 7-17
 - データ・ディクショナリにリストされている 4-2
 - デフォルトの表領域 25-7
 - 同時実行動作の調整 1-20
 - 認証 25-3
 - パスワード暗号化 25-4
 - 表領域 1-31
 - 表領域の割当て制限 1-31, 25-7
 - プロセス 1-16, 7-4
 - プロファイル 1-32, 25-11
 - 分散データベース 30-16
 - マルチ・ユーザー環境 7-3
 - マルチユーザー環境 1-2
 - ユーザーの数によるライセンス 25-14
 - ユーザー名 1-29, 25-2
 - セッションと接続 7-5
 - ライセンス 25-12
 - リソース使用に対する制限 1-31
 - リソース制限 25-9
 - ロール 26-10
 - ユーザーのタイプ 26-11
 - ユーザー・プログラム・インタフェース (UPI) 7-27
 - ユーザー・プロセス
 - PGA の割当て 6-13
 - 共有サーバー・プロセス 7-23
 - セッション 7-5
 - 接続 7-4
 - 専用サーバー・プロセス 7-17
 - ユーザー・ロック 23-38
 - ユーザー定義データ型 11-1, 11-3, 12-1
 - Export と Import 12-15
 - オブジェクト・リレーショナル・モデル 1-40
 - オブジェクト型 11-2, 11-3
 - 表の別名の使用 12-2
 - 記憶域 12-4
 - 権限 12-9
 - コレクション 11-9
 - 可変配列 (VARRAY) 11-9
 - ネストした表 11-10
 - 不完全な型 12-13
- よ
- 読み込み
 - 使用済み 23-2
 - データ・ブロック
 - 制限 25-10
 - 反復可能 23-6
 - 読み込み一貫性
 - DML の副問合せ 23-13
 - 定義 1-21
 - 問合せ 14-12
 - トランザクション 1-21, 23-6
 - トリガー 18-14, 18-15
 - マルチバージョンの一貫性モデル 1-21
 - メッセージ「スナップショットが古すぎます」
 - 23-5
 - ロールバック・セグメント 2-17
 - 読み込みが書き込みを阻止するか 23-10
 - 読み込み専用スナップショット 31-6
 - リフレッシュの種類 31-10
 - 読み込み専用トランザクション 1-22
 - 読み込み専用表領域
 - 制限 3-10
 - 説明 3-9
 - バックアップ 28-22
 - 読み込み専用レプリケーション 31-2, 31-4
 - 使用方法 31-4
 - 予約語 14-2
- ら
- ライセンス
 - 現在の制限を表示する 25-13
 - 同時使用 25-13
 - 名前付きユーザー 25-14
 - ラッチ
 - LRU 7-8
 - 説明 23-28
 - ラッパー
 - プロシージャ・レプリケーション 31-26
 - ラベル
 - MLSLABEL データ型 10-15
- り
- リアルタイム・レプリケーション 31-27
 - リーフ・ブロック 8-19

- リカバラ・プロセス (RECO) 1-18, 7-12
 - インダウト・トランザクション 1-25, 5-7, 15-8
- 離散トランザクションの管理
 - 要約 15-8
- リスナー・プロセス 7-14
- リソース制限
 - CPU 時間の制限 25-10
 - 値の決定 25-12
 - 概要 1-32
 - コール・レベル 25-10
 - セッション・レベル 25-9
 - セッション当たりのアイドル時間 25-11
 - セッション当たりの接続時間 25-11
 - セッション当たりのプライベート SGA 領域 25-11
 - ユーザー当たりのセッション数 25-11
 - 論理読み込みの制限 25-10
- リテラル起動
 - コンストラクタ・メソッド 12-7
- リフレッシュ
 - ジョブ・キュー・プロセス (SNPr) 1-19, 7-13
- リフレッシュ・グループ 31-9
 - 自動リフレッシュ 31-9
 - 手動リフレッシュ 31-10
- リフレッシュ間隔
 - スナップショット・リフレッシュ・グループ 31-9
- リフレッシュの種類
 - 読み込み専用スナップショット 31-10
- リモート・データベース 1-24, 30-2
 - 分散データベースも参照
 - データベース・リンク 30-6
- リモート・トランザクション 30-11
- リモート・プロシージャ・コール 30-11, 31-19
- リモート依存性 19-9
- 領域管理
 - MINIMUM EXTENT パラメータ 22-24
 - PCTFREE 2-5
 - PCTUSED 2-6
 - 空き領域の圧縮 2-9
 - 行連鎖 2-9
 - セグメント 2-14
 - ダイレクト・ロード・インサート 21-8
 - パラレル DDL 22-24
- リレーショナル・データベースの操作 1-40
- リレーショナル DBMS (RDBMS)

- SQL 14-2
- Oracle も参照
- オブジェクト・リレーショナル DBMS 11-2
- 原理 1-40
- リレーション 1-41
- 履歴データベース
 - パーティション 9-4
 - メンテナンス操作 9-30
- リンク 30-6

る

- ルート・ブロック 8-37
- ルールベース最適化 20-10

れ

- 例外
 - ストアド・プロシージャ 14-17
 - トリガー実行中の 18-15
 - 呼び出す 14-17

列

- NULL の使用禁止 24-6
- 疑似列
 - ROWID 10-11
 - ROWNUM 20-23, 20-32, 20-54
 - USER 26-6
- 順序 8-7
- 整合性制約 8-4, 8-8, 24-3, 24-6
- 説明 8-3
- 定義 1-41
- デフォルト値 8-8
- ネストした表 8-9
- ビューまたは表での最大数 8-10
- 列オブジェクト 11-7
 - 索引 12-8
- 列名
 - 問合せでの修飾 12-2
 - 問合せの修飾 12-3
 - 連結索引での最大数 8-17

列グループ 31-25

- シャドウ 31-25

列のパーティション化 9-12

レプリケーション 31-1

- オブジェクト 31-17
- カタログ 31-18
- 管理者 31-22

- 基本 31-2, 31-4
- 競合 31-22
 - 解決 31-25
 - 解決メソッド 31-25
 - 行レベルのレプリケーション 31-25
 - 検出 31-25
 - データ・モデル 31-23
 - プロシージャ・レプリケーション 31-26
 - 列グループ 31-25
- グループ 31-17
- サイト 31-17
- 受信者 31-22
- 制限
 - ダイレクト・ロード・インサート 21-9
 - パラレル DML 22-33
- 対称型レプリケーション 31-11, 31-12
- 対称型レプリケーションの使用方法 31-12
- 定義 31-2
 - 分散データベース 30-7
- 伝播担当者 31-22
- プロシージャ 31-26
- 読み専用レプリケーションの使用方法 31-4
- リアルタイム 31-27
- レプリケーション管理 API 31-18
- レプリケートされたデータの受信者 31-22
- レプリケートされたデータの伝播担当者 31-22
- 連結索引 8-17
- 連鎖、行の 2-9, 8-5
- レンジ・パーティション化 9-11

ろ

- ローカル・データベース 1-24
- ローカル索引 9-21, 9-25
 - パーティションの管理 9-35
 - ビットマップ索引
 - パーティション表 8-25
 - パラレル問合せおよび DML 8-22
- ロール 1-30, 26-10
 - CONNECT ロール 12-10, 12-11, 26-14
 - DBA ロール 12-10, 26-14
 - DDL 文 26-13
 - EXP_FULL_DATABASE ロール 26-14
 - IMP_FULL_DATABASE ロール 26-14
 - RESOURCE ロール 12-10, 12-11, 26-14
 - アプリケーション 26-11

- アプリケーションにおける 1-31
- 依存性管理 26-13
- オペレーティング・システムを介した管理 26-14
- 概要 1-30
- 機能性 26-2
- キュー管理者 16-5
- グローバル認証サービス 30-16
- 権限に関する制限 26-13
- 事前定義済み 26-14
- 使用可能または使用禁止 26-12
- 使用方法 26-11
- スキーマには含まれない 26-13
- セキュリティ・ドメイン 26-13
- 取消し 26-12
- 名前付き PL/SQL ブロックで使用禁止にされた 26-13
- パスワードの使用 1-31
- 付与 26-2, 26-12
- 付与できるユーザー 26-12
- 分散データベース・アプリケーション 30-16
- 命名 26-13
- ユーザー 26-11
- ロールバック 2-16, 15-6
 - 回復時 1-39, 28-9
 - 高速トランザクション・ロールバック 28-10
 - セーブポイントまで 15-6
 - 説明 15-6
 - 定義 1-49
 - トランザクションの終了 15-2, 15-4, 15-6
 - 文レベル 15-4
- ロールバック、回復時の 28-9
- ロールバック、トランザクションの 1-49, 15-2, 15-6
 - 高速ウォームスタート 28-4
- ロールバック・エントリ 2-16
- ロールバック・セグメント 1-10, 2-16
 - 1 つあたりのトランザクション数 2-18
 - SYSTEM ロールバック・セグメント 2-22
 - アクセス 2-17
 - いつ取得されるか 2-23
 - いつ使われるか 2-17
 - インダウト分散トランザクション 2-21
 - エクステンツの割当て 2-18
 - 新しいエクステンツ 2-21
 - エクステンツの割当て解除 2-22
 - オフライン 2-24, 2-26

- オフライン表領域 2-27
- オンライン 2-24, 2-26
- 回復が必要な状態 2-24
- 回復での使用 1-38, 28-9
- 概要 2-16, 28-7
- 起動時に取得する 5-7
- 競合 2-18
- 削除 2-22
 - 制限 2-26
- 取得時のクラッシュ 2-23
- 状態 2-24
- 遅延 2-27
- 次のエクステンツへの移動 2-19
- 定義 1-10
- トランザクション 2-17
- トランザクションからどのように書き込まれるか 2-18
- トランザクションのコミット 2-18
- バブリック 2-22
- パラレル回復 28-10
- 部分的に使用可能な状態 2-24, 28-4
- プライベート 2-22
- 無効な状態 2-24
- 読み込み一貫性 1-21, 2-17, 23-4
- ロールバック・セグメント 2-18
- ロック 23-29
- ロールフォワード、回復時の 1-39, 28-8
- ロギング・モード
 - NOARCHIVELOG モードとの関係 21-5
 - 影響を受ける SQL 操作 21-7
 - ダイレクト・ロード・インサート 21-5
 - パーティション 9-34
 - パラレル DDL 22-22, 22-23
- ログ・ライター・プロセス (LGWR) 1-17, 7-9
 - REDO ログ・バッファ 6-6
 - アーカイブ・モード 28-15
 - グループ・コミット 7-11
 - システム変更番号 15-5
 - 事前書込み 7-10
 - 手動アーカイブ 28-17
 - 複数の I/O プロセス 7-11
- ログ管理ロック 23-29
- ログ順序番号 1-36
- ロック 1-22, 23-3
 - DML が取得する 23-26
 - 図 23-24
 - DML パーティション・ロック 9-28
 - Oracle での手順 23-14
 - Oracle のロック管理サービス 23-38
 - オブジェクト・レベルのロック 11-13
 - 解析 14-11, 23-27
 - 概要 1-22, 23-3
 - 行 (TX) 23-19
 - 行共有表ロック (RS) 23-21
 - 行排他ロック (RX) 23-22
 - 共有行排他ロック (SRX) 23-23
 - 共有表ロック (S) 23-23
 - 共有副排他ロック (SSX) 23-23
 - 自動 1-22, 23-14, 23-18
 - 手動 1-23, 23-29
 - 動作の例 23-30
 - タイプ 23-18
 - 段階的拡大が発生しない 23-16
 - ディクショナリ 23-26
 - クラスタ 23-28
 - 存続期間 23-28
 - ディクショナリ・キャッシュ 23-28
 - データ 23-19
 - 存続期間 23-14
 - デッドロック 23-16, 23-17
 - 回避 23-18
 - トランザクションをコミットした後 15-6
 - 内部 23-28
 - 排他表ロック (X) 23-24
 - パラレル・キャッシュ管理 (PCM) 23-18
 - パラレル DML 22-31
 - 表 (TM) 23-20
 - 表領域 23-29
 - 表ロックのモード 23-20
 - ファイル管理ロック 23-29
 - 副共有表ロック (SS) 23-21
 - 副排他表ロック (SX) 23-22
 - 変換 23-15
 - ラッチ 23-28
 - ロールバック・セグメント 23-29
 - ログ管理ロック 23-29
- ロック・プロセス (LCKn) 1-19, 7-13
- 論理構造 1-8
- 論理データベース構造 1-5
- 論理ブロック 2-2

論理読み込みの制限 25-10

わ

割当て制限

ゼロに設定 25-8

表領域 1-31, 25-7

一時セグメントでは無視される 25-8

表領域アクセスの取消し 25-8

