Oracle8*i*

概要 Vol. 2

リリース 8.1

ORACLE

Oracle8*i* 概要 Vol.2 リリース 8.1

部品番号：A62757-1

第 1 版 1999 年 5 月

原本名：Oracle8*i* Concepts, Volume 2, Release 8.1.5

原本部品番号： A67783-01

原本著者：Lefty Leverenz, Diana Rehfield

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

目次

Vol.1

第 II 部　データベースの構造

2　データ・ディクショナリ

3　表領域とデータ・ファイル

4 データ・ブロック、エクステントおよびセグメント

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

6 分散処理

7 メモリー・アーキテクチャ

8 プロセスのアーキテクチャ

9 データベース・リソースの管理

第 IV 部　オブジェクト・リレーショナル DBMS

10　スキーマ・オブジェクト

11 パーティション表とパーティション索引

12 ビルトイン・データ型

13 ユーザー定義データ型

14 ユーザー定義データ型の使用方法

15 オブジェクト・ビュー

索引

Vol. 2

第 V 部　データ・アクセス

16　SQL と PL/SQL

17　トランザクションの管理

18 プロシージャとパッケージ

19 アドバンスト・キューイング

20 トリガー

21 Oracle の依存性の管理

第 VI 部　SQL 文の最適化

22　オプティマイザ

23　オプティマイザの操作

24 結合の最適化

第 VII 部 パラレル SQL とダイレクト・ロード・インサート

25 ダイレクト・ロード・インサート

26 パラレル実行

第 VIII 部　データの保護

27　データの同時実行性と一貫性

28　データの整合性

29 データベース・アクセスの制御

30 権限、ロールおよびセキュリティ・ポリシー

31 監査

32 データベースの回復

第 IX 部　分散データベースとレプリケーション

33　分散データベース

34　データベースのレプリケーション

第 X 部　付録

A　オペレーティング・システム固有の情報

索引

はじめに

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

Oracle8*i* および Oracle8*i* Enterprise Edition

では、Oracle8*i* および Oracle8*i* Enterprise Edition 製品の機能の詳細は、『Oracle8*i* 概要』を参照してください。Oracle8*i* および Oracle8*i* Enterprise Edition は、同じ機能を持っています。ただし、Enterprise Edition のみで使用可能な 拡張機能もいくつかあり、これらの一部はオプションです。たとえば、表をパーティション化するには、Enterprise Edition と Partitioning Option が必要です。

対象読者

このマニュアルは、データベース管理者、システム管理者、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用されている概念と用語について包括的に説明されています。

インストレーションや移行について

このマニュアルは、インストールや移行の手引き書ではありません。したがって、主としてインストレーションに関心のある方は、使用するオペレーティング・システムを対象にした Oracle マニュアルを参照してください。また、主としてデータベースおよびアプリケーションの移行に関心のある方は、『Oracle8*i* 移行ガイド』を参照してください。

データベース管理について

このマニュアルは、Oracle Server のアーキテクチャ、プロセス、構造、その他の概念について説明しています。Oracle Server を管理する方法については説明していません。詳細は、『Oracle8*i* 管理者ガイド』参照してください。

アプリケーション設計について

このマニュアルには、管理者だけでなく、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*i* アプリケーション開発者ガイド 基礎編』と、Oracle データベース・アプリケーションの開発に使用するツールまたは言語製品のマニュアルも参照する必要があります。

このマニュアルの構成

このマニュアルは 2 巻構成で、次のような部に分かれています。

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

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

Vol.1

第 I 部 : Oracle の概要

第 1 章「Oracle Server の基礎知識」

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

第 II 部 : データベースの構造

第 2 章「データ・ディクショナリ」

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

第 3 章「表領域とデータ・ファイル」

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

第 4 章「データ・ブロック、エクステントおよびセグメント」

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

第 III 部 : Oracle インスタンス

第 5 章「データベースとインスタンスの起動と停止」

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

第 6 章「分散処理」

Oracle データ・サーバーを実行できる分散処理環境について説明します。

第 7 章「メモリー・アーキテクチャ」

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

第 8 章「プロセスのアーキテクチャ」

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

第 9 章「データベース・リソースの管理」

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

第 IV 部 : オブジェクト・リレーショナル DBMS

第 10 章「スキーマ・オブジェクト」

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

第 11 章「パーティション表とパーティション索引」

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

第 12 章「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 13 章「ユーザー定義データ型」

Oracle が提供するオブジェクト拡張機能の概要について説明します。

第 14 章「ユーザー定義データ型の使用方法」

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

第 15 章「オブジェクト・ビュー」

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

Vol.2

第 V 部 : データ・アクセス

第 16 章「SQL と PL/SQL」

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

第 17 章「トランザクションの管理」

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

第 18 章「プロシージャとパッケージ」

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

第 19 章「アドバンスト・キューイング」

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

第 20 章「トリガー」

トリガーと呼ばれるプロシージャ型言語の構成体について説明します。トリガーは、データベース表に行が挿入されたり、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 21 章「Oracle の依存性の管理」

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

第 VI 部 : SQL 文の最適化

第 22 章「オプティマイザ」

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

第 23 章「オプティマイザの操作」

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

第 24 章「結合の最適化」

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

第 VII 部 : パラレル SQL とダイレクト・ロード・インサート

第 25 章「ダイレクト・ロード・インサート」

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

第 26 章「パラレル実行」

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

第 VIII 部 : データの保護

第 27 章「データの同時実行性と一貫性」

マルチ・ユーザー環境において、Oracle が共有情報への同時アクセスを提供し、その情報の正確さを維持する仕組みについて説明します。そして、複数のユーザーが同時に操作を実行しても相互に干渉し合わないようにするための、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 28 章「データの整合性」

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

第 29 章「データベース・アクセスの制御」

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

第 30 章「権限、ロールおよびセキュリティ・ポリシー」

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

第 31 章「監査」

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

第 32 章「データベースの回復」

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

第 IX 部 : 分散データベースとレプリケーション

第 33 章「分散データベース」

分散データベース・アーキテクチャ、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 34 章「データベースのレプリケーション」

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

第 X 部 : 付録

付録 A「オペレーティング・システム固有の情報」

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

このマニュアルの使用方法

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

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

このマニュアルで使用する表記規則

このマニュアルでは、情報の種類に応じていくつかの表記を使用します。

マニュアルの本文

このマニュアルの本文では、次のような表記規則が使用されています。

大文字

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

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

コード例

SQL および Oracle Enterprise Manager の行モード（Server Manager）、SQL*Plus の コマンドや文は、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、単なる例として使用されている語を示します。たとえば、小文字の語は、表、列、またはファイルの名前を示します。

第 V 部

データ・アクセス

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

16

SQL と PL/SQL

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 および Server Manager などの Oracle Tools を使用すると、Oracle データベースに対して ANSI/ISO 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 を使用して実行されます。その他のデータ・アクセス方法を使用すると、Oracle に組み込まれているセキュリティが活用されず、データのセキュリティと整合性が損なわれる可能性があります。

> **追加情報：** SQL コマンドおよび SQL のその他の部分（演算子、関数および書式モデルなど）の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。
>
> Oracle Enterprise Manager の詳細は『Oracle Enterprise Manager 管理者ガイド』、SQL コマンドとの違いなど、SQL*Plus の詳細は『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

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するすべての操作は、SQL「文」を使用して実行します。SQL 文は、有効な「SQL 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、他の目的には使用できません。たとえば、SELECT と UPDATE は予約語なので、表名には使用できません。

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「不完全文」を実行すると、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・ユニット内でも SQL 文を使用できます。この機能の詳細は、第 18 章「プロシージャとパッケージ」および第 20 章「トリガー」を参照してください。

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や操作を実行します。次のことを実行できます。

- 1 つ以上の表またはビューからデータを取り出す（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、最も頻繁に使用する SQL 文です。次に、DML 文の例をいくつか示します。

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

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

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

データ定義言語（DDL）文

DDL 文は、スキーマ・オブジェクトの定義、その構造の変更およびスキーマ・オブジェクトの削除を実行します。DDL 文によって、次のことを実行できます。

- スキーマ・オブジェクトとデータベース構造（データベースそのものとデータベース・ユーザーを含む）を作成、変更および削除する（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の収集、オブジェクト構造の妥当性検査、オブジェクト内の連鎖行のリスト表示を行う（ANALYZE）
- 権限とロールの付与および取消しを行う（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的にコミットし、新しいトランザクションを開始します。

次に、DDL 文の例をいくつか示します。

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

DROP TABLE plants;

GRANT SELECT ON emp TO scott;

REVOKE DELETE ON emp FROM scott;
```

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 29 章「データベース・アクセスの制御」、第 30 章「権限、ロールおよびセキュリティ・ポリシー」および第 31 章「監査」を参照してください。

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）
- トランザクション内での変更のうち、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実行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および使用禁止の切換えなど、特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 のみです。このコマンドは、設定値（共有サーバーの最小数など）の変更、セッションのキル（停止）およびその他の作業のために使用します。

埋込み SQL 文

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

- カーソルを定義し、割り当て、解放する（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用する SQL 文を識別する手段を提供する必要があります。Oracle 拡張機能は、対話型 SQL、Oracle プリコンパイラまたは SQL*Module で FIPS フラガーを使用して「フラグを立てる」ことによって識別できます。

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

> **追加情報：** FIPS フラガーの詳細は、『Pro*C / C++ プリコンパイラ・プログラマーズ・ガイド』、『Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』または『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用する、その他の情報が格納されるメモリー内の領域（プライベート SQL 領域）のハンドルまたは名前のことです。

ほとんどの Oracle ユーザーは Oracle ユーティリティの自動カーソル処理を使用しますが、アプリケーションの設計者はプログラム・インタフェースによってカーソルを自由に制御できます。アプリケーション開発の場合、カーソルはプログラムが使用できる名前付きのリソースです。特にアプリケーションに埋め込まれた SQL 文を解析するために使用できます。

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

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

共有 SQL

複数のアプリケーションがデータベースに対して同じ SQL 文を送信すると、Oracle はそのことを自動的に認識します。その文が最初に出現したときの処理に使用された SQL 領域は「共有」されます。つまり、その後に同じ文が出現すると、それを処理するためにこの領域が使用されます。したがって、一意の文に対しては 1 つの共有 SQL 領域しか存在しません。共有 SQL 領域は共有メモリー領域なので、どの Oracle プロセスも共有 SQL 領域を使用できます。SQL 領域を共有することで、データベース・サーバーのメモリー使用量が節約され、システムのスループットが向上します。

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

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

解析

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

- 文の構文上および意味上の妥当性をチェックする。
- 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用して、すぐにその文を実行します。存在しない場合、Oracle はその文の解析済みの表現を生成し、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格納します。

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

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

これらの説明は、PL/SQL ブロックの解析と PL/SQL 領域の割当てにも当てはまります。（16-14 ページの「PL/SQL」を参照）。ストアド・プロシージャ、ファンクションおよびパッケージとトリガーには、PL/SQL 領域が割り当てられます。Oracle は、PL/SQL ブロック内のそれぞれの SQL 文にも、共有 SQL 領域とプライベート SQL 領域を割り当てます。

SQL の処理

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

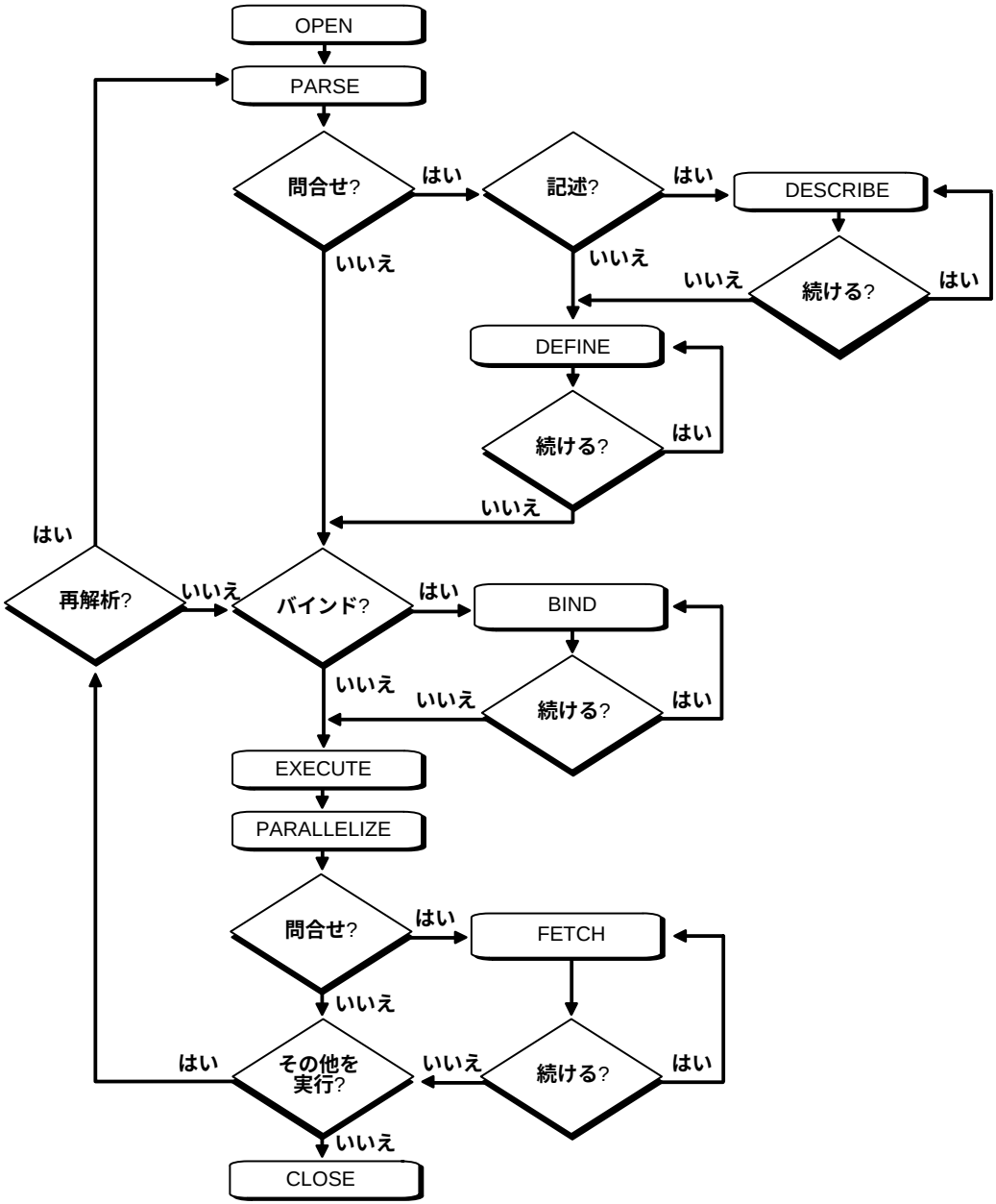
- SQL 文の実行の概要
- DML 文の処理
- DDL 文の処理
- トランザクションの制御

SQL 文の実行の概要

図 16-1 に、SQL 文を処理して実行するための一般的な段階を示します。場合によっては、順序が少し異なることもあります。たとえば、「定義」の段階は、コーディングの仕方によっては「フェッチ」の直前にくることがあります。

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

図 16-1 SQL 文の処理の段階



DML 文の処理

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

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

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

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

各タイプの文の処理では、次の段階が必要です。

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

オプションとして、次の段階を含めることもできます。

- 段階 6: 文のパラレル化

図 16-1 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、16-11 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

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

段階 2: 文の解析

解析段階では、SQL 文がユーザー・プロセスから Oracle に渡され、解析済みの表現が共有 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のみです。その場合は、新しい共有 SQL 領域が割り当てられて、文が解析されます。（16-7 ページの「共有 SQL」を参照）。

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

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。

問合せ処理のみ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。

- 「読込み一貫性」が必要。
- 中間処理に一時セグメントを使用できる。
- SQL 文処理の記述、定義およびフェッチ段階が必要になることがある。

段階 3: 問合せの結果の記述

記述段階が必要なのは、問合せがユーザーにより対話式に入力された場合など、問合せ結果の特性がわかっていない場合のみです。

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

段階 4: 問合せの出力の定義

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

段階 5: 変数のバインド

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

プログラムでは、値を検出できる位置（メモリー・アドレス）を指定する必要があります。Oracle ユーティリティはアプリケーションのエンド・ユーザーに新しい値の入力を要求するプロンプトを表示するだけなので、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可能性があります。

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

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

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。

- 『Oracle8*i* 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 メソッド 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 メソッド 4」を参照）

段階 6: 文のパラレル化

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

パラレル SQL の詳細は、第 26 章「パラレル実行」を参照してください。

段階 7: 文の実行

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

文によっては、実行回数を指定できる場合があります。このような処理を配列処理といいます。*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* の配列の開始点とみなされます。

段階 8: 問合せの行のフェッチ

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

段階 9: カーソルのクローズ

SQL 文の処理の最後の段階は、カーソルのクローズです。

DDL 文の処理

DDL 文を正常実行するにはデータ・ディクショナリへの書込みアクセスが必要なので、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、データ・ディクショナリの参照および実行が含まれます。

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

トランザクションの制御

一般に、1 つのトランザクションとしてまとめるアクションのタイプに配慮するのは、Oracle へのプログラム・インタフェースを使用するアプリケーション設計者のみです。作業が論理的な単位として完遂され、データの一貫性が保たれるように、トランザクションは適切に定義する必要があります。トランザクションには、1 つの論理作業単位に必要な部分を過不足なくすべて含める必要があります。

- トランザクションの開始前と終了後に、すべての参照表のデータは必ず一貫した状態になっている必要がある。
- 各トランザクションには、データに対して首尾一貫した 1 つの変更を加える SQL 文のみを含める。

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

アプリケーションを設計するときは、トランザクションにどのタイプのアクションを含めるかだけでなく、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用するべきかも判断する必要があります。詳細は、17-8 ページの「離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 を使用すると、SQL 文をプロシージャ型の言語要素と混合して使用できます。さらに PL/SQL では、プロシージャ、ファンクションおよびパッケージなどの PL/SQL プログラム・ユニットを定義して実行できます。

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

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

「ストアド・プロシージャ」とは、Oracle によってデータベースに格納され、アプリケーションから名前でコールできる PL/SQL ブロックのことです。ストアド・プロシージャを作成すると、Oracle はそのプロシージャを解析し、その解析済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）を作成して保管することもできます。

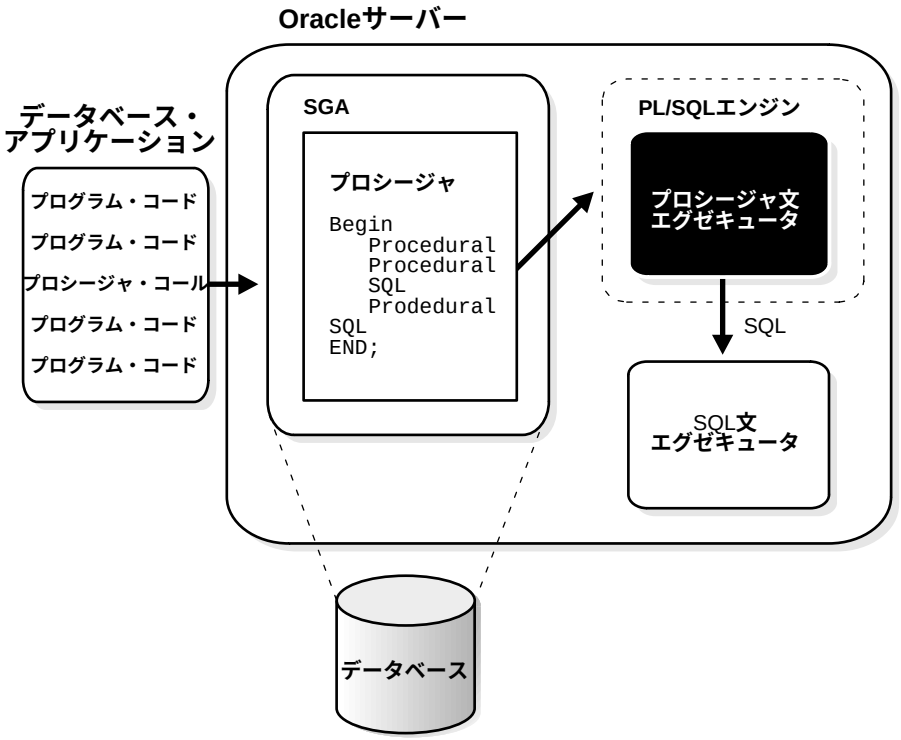
ストアド・プロシージャ、ファンクションおよびデータベース・トリガーの詳細は、第 18 章「プロシージャとパッケージ」および第 20 章「トリガー」を参照してください。

PL/SQL が実行される方法

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

図 16-2 に、Oracle Server に含まれている PL/SQL エンジンを示します。

図 16-2 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連動してプロシージャ内の文を処理します。

PL/SQL エンジンは、次の Oracle 製品に組み込まれています。

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

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

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

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

PL/SQL の言語要素

PL/SQL ブロックには、次の PL/SQL 言語要素を組み込むことができます。

- 変数と定数
- カーソル
- 例外

この項では、それぞれの要素について概説します。

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

注意： SQL*Plus などの対話形式のツールを使用すると、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用できます。

カーソル

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

例外

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

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

ストアド・プロシージャ

Oracle では、ストアド・プロシージャを作成してコールすることもできます。アプリケーションがストアド・プロシージャをコールすると、そのプロシージャの解析済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 エンジンを使用して処理できるプロシージャとパッケージのみを対象にしています。

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

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

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

別の PL/SQL ブロック（無名ブロックまたは別のストアド・プロシージャ）からストアド・プロシージャをコールすることもできます。詳細は、第 18 章「プロシージャとパッケージ」を参照してください。

追加情報： 各種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』など、個々のアプリケーション・ツールのマニュアルを参照してください。

PL/SQL の動的 SQL

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

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

- 動的 SQL 文を PL / SQL ブロックに埋め込む。
- DBMS_SQL パッケージを使用する。

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

追加情報： 動的 SQL のための 2 つのアプローチの比較は『Oracle8*i* アプリケーション開発者ガイド 基礎編』、動的 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。

外部プロシージャ

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

追加情報： 外部プロシージャと Inter-Language Method Services（ILMS）の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

17

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用して作業を管理する方法について説明します。この章の内容は次のとおりです。

- トランザクションの基礎知識
- Oracle とトランザクションの管理
- 離散トランザクションの管理
- 自律型トランザクション

トランザクションの基礎知識

「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。

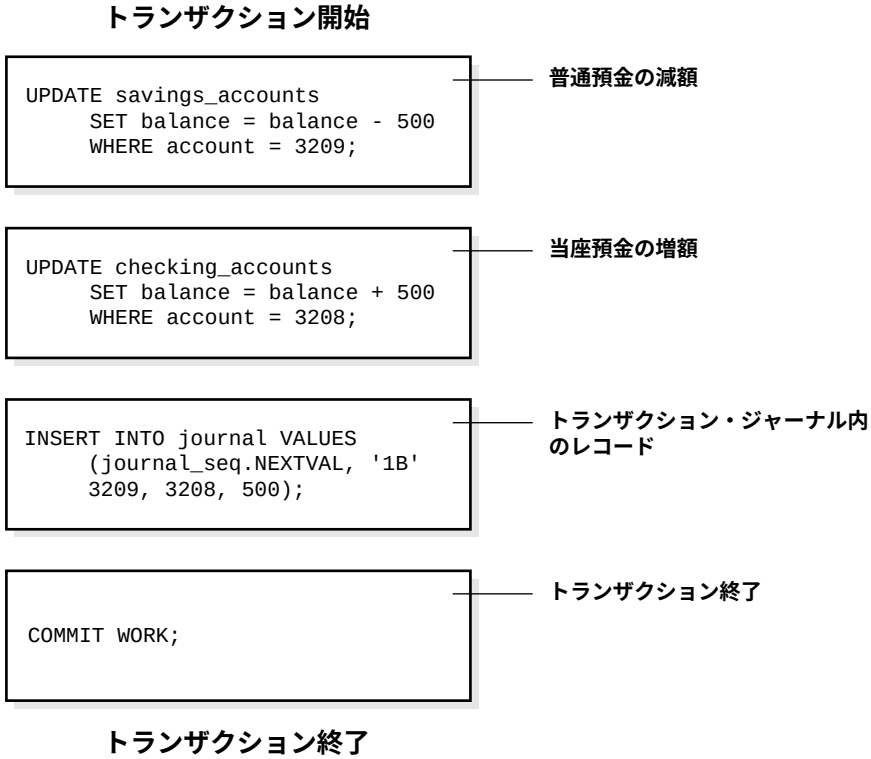
トランザクションは、最初の実行可能な SQL 文から開始します。トランザクションは、明示的に（COMMIT 文または ROLLBACK 文を使用する）、または暗黙的に（DDL 文を発行）コミットまたはロールバックされると、そこで終了します。

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す。銀行の顧客が普通預金口座から当座預金口座へ預金を振り替えるときに、トランザクションは、普通預金口座の減額、当座預金口座の増額およびトランザクション・ジャーナルへのトランザクションの記録という 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結果をデータベースに適用できます。ただし、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 つまたは 2 つの文が完了しない場合は、すべての口座の差引残高が正しく維持されるようにトランザクション全体をロールバックする必要があります。

図 17-1 に銀行のトランザクションの例を示します。

図 17-1 バンキング・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（複数行の更新ですべての行が更新された場合など）。ただし、その文が含まれているトランザクションがコミットされるまでは、トランザクションをロールバックしてその文のすべての変更を取消しできます。正常に実行される（成功する）という表現は、トランザクションではなく、文に対して使用します。

「コミット」とは、ユーザーが明示的または暗黙的に「このトランザクションの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のみが、コミットされた変更を参照できます。

文レベルのロールバック

実行中に SQL 文のエラーが発生すると、その文のすべての効果はロールバックされます。ロールバックの効果は、その文がまったく実行されなかった場合と同じ状態にすることです。これが「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ると、文レベルのロールバックが発生します。（この種のエラーの一例として、主キーに重複値を挿入しようとした場合があります。）解析中にエラーが検出された場合（構文エラーなど）、まだ SQL 文は実行されていないため、文レベルのロールバックは発生しません。1 つのデッドロック（同じデータに関する競合）に単一 SQL 文が複数関係している場合も、文レベルのロールバックが発生します。27-17 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無駄になりません。文が DDL 文の場合、その文の直前に実行された暗黙のコミットは取り消されません。

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

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、4-20 ページの「トランザクションとロールバック・セグメント」を参照してください。

トランザクションは、次のいずれかの状況が発生すると終了します。

- COMMIT 文または ROLLBACK（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 は最初にそのトランザクションをコミットし、新しい単一文トランザクションとしてその DDL 文を実行し、コミットします。
- ユーザーが Oracle の接続を切断する。（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す。）

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

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

トランザクションのコミット

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

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

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

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

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、トランザクションの SCN も、LGWR によってオンライン REDO ログ・ファイルに書き込まれます。これが、トランザクションのコミットを構成するアトミック・イベントです。
- 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、27-3 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 DBW*n* の詳細は、8-5 ページの「Oracle プロセス」を参照してください。

トランザクションのロールバック

「ロールバックする」とは、コミットされていないトランザクション内の SQL 文によって実行されたデータ変更を取り消すことを意味します。

Oracle では、コミットされていないトランザクション全体をロールバックできます。また、コミットされていないトランザクションの後半部分をセーブポイントと呼ばれるマーカーまでロールバックすることもできます。詳細は、17-7 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、同じ手順が使用されます。

- 文レベルのロールバック（文またはデッドロックの実行エラーによる）
- セーブポイントへのロールバック
- ユーザー要求によるトランザクションのロールバック
- プロセスの異常終了によるトランザクションのロールバック
- インスタンスが異常終了したときの、すべての保留中のトランザクションのロールバック
- 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・セグメントを使用して取り消される。
- 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、27-3 ページの「ロックのメカニズム」を参照）。
- 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のみ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取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取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、27-3 ページの「ロックのメカニズム」を参照）。
- トランザクションはアクティブであり、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割します。

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

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

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

2 フェーズ・コミット・メカニズム

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

「2 フェーズ・コミット・メカニズム」は、分散トランザクションに関係するすべてのデータベース・サーバーが、そのトランザクション内の文のすべてをコミットするか、またはすべてをロールバックするかのどちらか一方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。事実、ユーザーは、トランザクションが分散していることも認識する必要はありません。トランザクションの終了を示す COMMIT 文があると、トランザクションをコミットするための 2 フェーズ・コミット・メカニズムが自動的に起動されます。データベース・アプリケーションの本体に、分散トランザクションを含めるための特別なコーディングや複雑な構文は必要ありません。

リカバラ（RECO）バックグラウンド・プロセスは、インダウト分散トランザクション（システム障害やネットワーク障害によってコミットが中断された分散トランザクション）の結果を自動的に解決します。障害が修復され、通信が再確立された後、各ローカル Oracle Server の RECO 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、ローカルのデータベース管理者は、長時間にわたる障害の結果として無期限にロックされる可能性のあるリソースを解放できます。

あるデータベースを過去のある時点まで回復する必要がある場合、Oracle の回復機能を使用すると、他のサイトのデータベース管理者は、自分のデータベースも過去のその同じ時点に戻すことができます。これによって、グローバル・データベースは一貫した状態に保たれます。

離散トランザクションの管理

アプリケーション開発者は、プロシージャ BEGIN_DISCRETE_TRANSACTION を使用して、小規模な非分散型トランザクションのパフォーマンスを向上させることができます。このプロシージャにより、トランザクション処理の効率が改善されるので、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るかどうかにかかわらず、コミットされていない変更の参照はできません。

Oracle は REDO 情報を生成しますが、それをメモリー内の別個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。これにより、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。

常に REDO が生成される離散トランザクションと、ダイレクト・パス操作にのみ適用される NOLOGGING モードとの間には、相互作用はありません。（25-5 ページの「ロギング・モード」を参照）。したがって、離散トランザクションは、NOLOGGING 属性が設定されている表に対して発行できます。

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

自律型トランザクション

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

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

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

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

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

自律型 PL/SQL ブロック

PL/SQL ブロックから自律型トランザクションをコールできます。仕様 PRAGMA AUTONOMOUS_TRANSACTION では、次の種類の PL/SQL ブロックを自律型として宣言できます。

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

自律型ブロックの "BEGIN .. END" セクションで実行されるトランザクション操作は、自律型トランザクションの一部として、つまり、コール側ブロックのトランザクション・コンテキストから独立した状態で実行されます。自律型 PL/SQL ブロックに入ると、コール側のトランザクション・コンテキストは中断されます。これにより、このブロック（または、そこからコールされる他のブロック）で実行される SQL 操作は、コール側のトランザクション・コンテキストの状態に依存せず、影響しないことが保証されます。

自律型 PL/SQL ブロックは、SQL 操作がそのブロック内で実行されるか、またはそのブロックからコールされる他のブロック内で実行されるかを問わず、RNDS（データベースの状態を読み込まない）および WNDS（データベースの状態を書き込まない）の純粋さレベルを持っていると見なされます。したがって、この種のブロックを SQL のコンテキストからコールできます。

追加情報： 純粋さレベルの詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

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

自律型ブロック内のトランザクション制御文

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

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

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

追加情報： 自律型トランザクションの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

18

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、ファンクションおよびパッケージの間の依存性と、Oracle がそれらの依存性を管理する方法の詳細は、第 21 章「Oracle の依存性の管理」を参照してください。

ストアド・プロシージャとパッケージの基礎知識

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・オブジェクトを使用し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、ファンクションおよびパッケージは、すべて PL/SQL プログラム・ユニットの例です。

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

多くの Oracle 製品に PL/SQL コンポーネントが含まれていますが、この章では、Oracle データベースに格納でき、Oracle Server の PL/SQL エンジンを使用して処理できるプロシージャとパッケージを対象にして説明します。それぞれの Oracle Tool の PL/SQL 機能については、該当する Oracle Tool のマニュアルに説明があります。詳細は、16-14 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

プロシージャとファクションは、特定の作業を実行するための SQL および PL/SQL プログラミング言語文の集合を論理的にグループ化したスキーマ・オブジェクトです。プロシージャとファンクションは、ユーザーのスキーマ内に作成され、繰り返し使用するためにデータベースに格納されます。SQL*Plus などの Oracle Tool を使用し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できます。

図 18-1 に、データベースに格納され、いくつかの異なるデータベース・アプリケーションによってコールされる単純なプロシージャを示します。

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

図 18-1 ストアド・プロシージャ

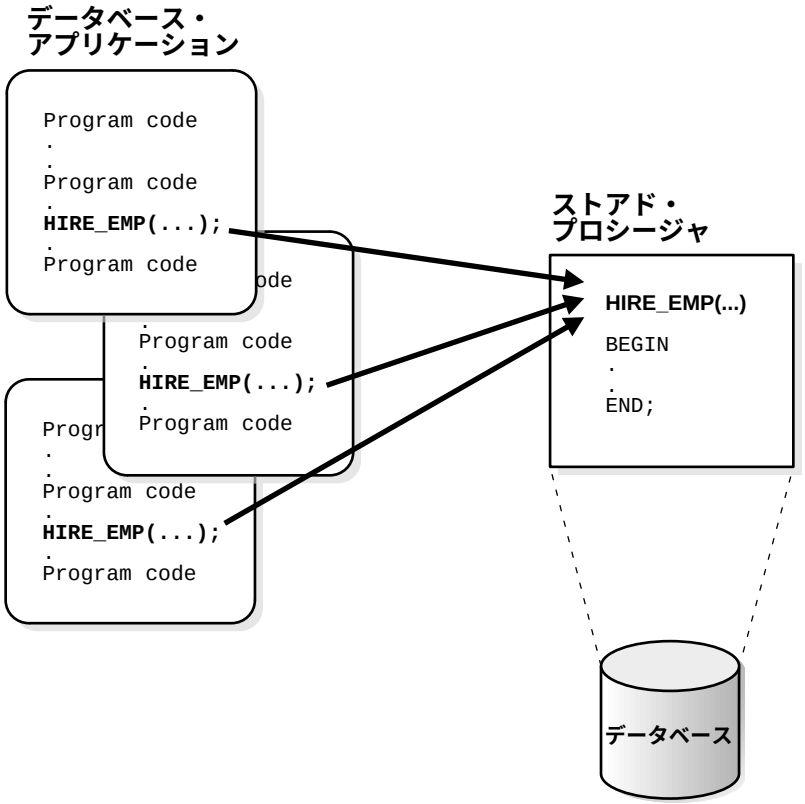


図 18-1 のストアド・プロシージャは、従業員レコードを EMP 表に挿入します。このプロシージャを図 18-2 に示します。

図 18-2 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 18-1 のデータベース・アプリケーションはすべて、HIRE_EMP プロシージャをコールしています。また、権限を付与されたユーザーは、Oracle Enterprise Manager または SQL*Plus を使用して、次の文で HIRE_EMP プロシージャを実行できます。

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

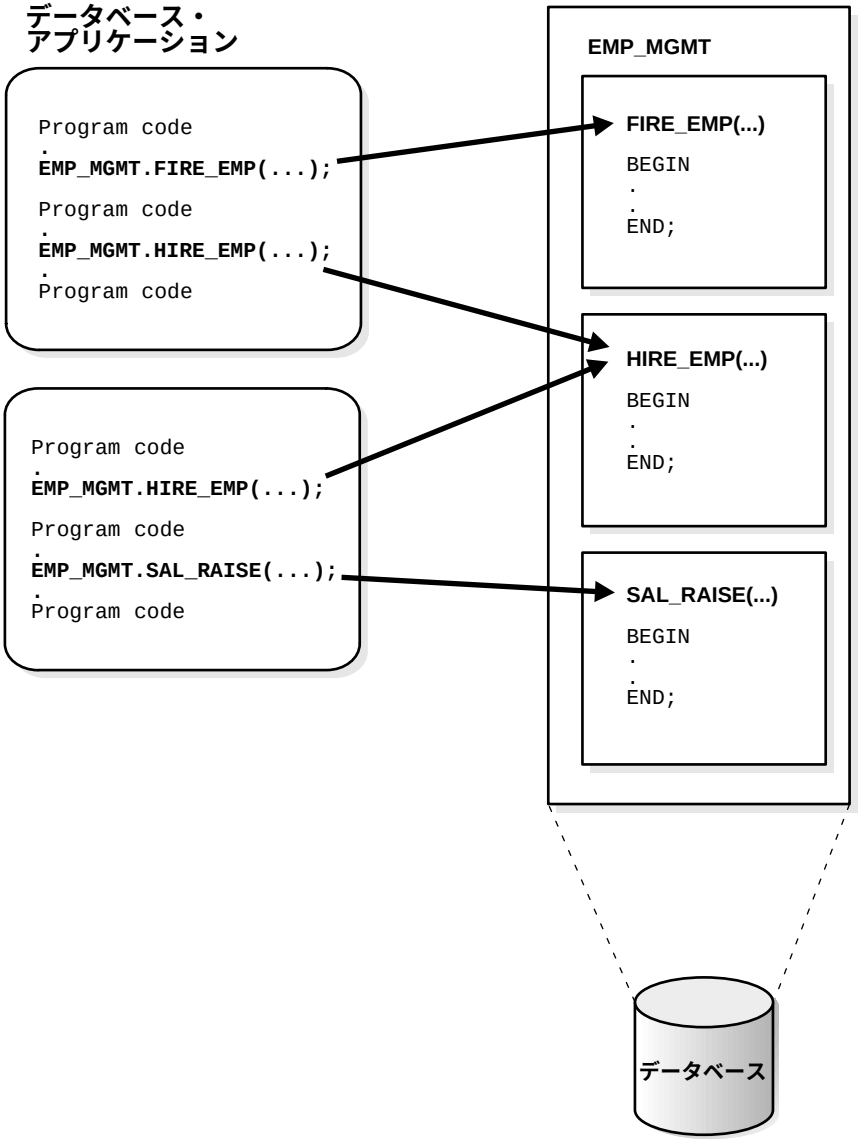
この文により、EMP 表に TSMITH のための新しい従業員レコードが挿入されます。

パッケージ

パッケージとは、関連するプロシージャとファンクション、およびこれらのプロシージャとファンクションが使用するカーソルと変数をグループとしてまとめたもので、1 単位として継続的に使用できるようにデータベースに格納されています。スタンドアロン・プロシージャやファンクションと同様に、パッケージ・プロシージャとファンクションは、アプリケーションやユーザーが明示的にコールできます。

図 18-3 に、従業員データベースの管理に使用するいくつかのプロシージャをカプセル化するパッケージを示します。

図 18-3 ストアド・パッケージ



データベース・アプリケーションは、必要に応じて明示的にパッケージ・プロシージャをコールします。EMP_MGMT パッケージに対する権限を付与されたユーザーは、そこに含まれている任意のプロシージャを明示的に実行できます。たとえば、Oracle Enterprise Manager または SQL*Plus では、次の文を発行して HIRE_EMP パッケージ・プロシージャを実行できます。

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

スタンドアロンのストアド・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。（18-11 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」は、SQL 文やその他の PL/SQL 構成体のセットで構成され、グループとしてまとめてデータベースに格納されるスキーマ・オブジェクトです。特定の問題を解決したり、関連する一連のタスクを実行するために、1 単位として実行されます。プロシージャとファンクションには、コール側から、入力のみ、出力のみ、または入出力の値が入るパラメータを指定できます。

プロシージャとファンクションを使用すると、SQL が持つ平易さや柔軟性を、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と合体できます。たとえば、次の文は、預金口座に入金する CREDIT_ACCOUNT プロシージャを作成します。

```
CREATE PROCEDURE credit_account
      (acct NUMBER, credit NUMBER) AS
/* This procedure accepts two arguments: an account number and an
   amount of money to credit to the specified account. If the
   specified account does not exist, a new account is created. */

        old_balance  NUMBER;
        new_balance  NUMBER;
   BEGIN
        SELECT balance INTO old_balance FROM accounts
          WHERE acct_id = acct
          FOR UPDATE OF balance;

        new_balance := old_balance + credit;
        UPDATE accounts SET balance = new_balance
          WHERE acct_id = acct;
        COMMIT;

        EXCEPTION
          WHEN NO_DATA_FOUND THEN
            INSERT INTO accounts (acct_id, balance)
                VALUES(acct, credit);
          WHEN OTHERS THEN
        ROLLBACK;
```

```
END credit_account;
```

このサンプル・プロシージャには、SQL 文と PL/SQL 文の両方が含まれていることに注意してください。

プロシージャのガイドライン

ストアド・プロシージャの設計時には、次のガイドラインに従ってください。

- プロシージャは、単一の限定的なタスクを実行するように定義する。複数の個別のサブタスクを含む長いプロシージャは定義しないでください。多数のプロシージャに共通のサブタスクが存在すると、複数のプロシージャ・コードに不必要な重複が発生するからです。
- 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、整合性制約を宣言すれば簡単に施行できる単純なデータ整合性規則の場合、それ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・セキュリティの施行に役立ちます。定義者の権限で実行するプロシージャとファンクションを介してのみユーザーがデータにアクセスできるようにすると、ユーザーが実行できるデータベース操作を制限できます（18-9 ページの「定義者権限と起動者権限」を参照）。たとえば、表を更新するプロシージャへのアクセス権は付与しても、その表自体へのアクセス権は付与しないこともできます。ユーザーがプロシージャを呼び出すと、そのプロシージャがプロシージャの所有者の権限で操作を実行します。プロシージャの実行権限しかなく、表データの問合せ、更新または削除の権限を持たないユーザーは、プロシージャを呼び出すことはできても、表のデータをそれ以外の方法では操作できません。

パフォーマンス

ストアド・プロシージャを使用すると、次の方法でデータベースのパフォーマンスが向上します。

- 情報は 1 回しか送信されず、それ以降は使用するときに呼び出されるため、個々の SQL 文を発行したり、PL/SQL ブロック全体のテキストを Oracle に送信する場合に比べて、ネットワークを介して送信する必要のある情報量が少なくなる。
- データベース内でプロシージャのコンパイル済み形式をそのまま使用できるため、実行時にコンパイルする必要がない。

- プロシージャがすでに SGA の共有プール内にある場合は、ディスクから検索する必要がないため、ただちに実行を開始できる。

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 つメモリーにロードするだけですみます。同じコードを何人ものユーザーが共有すれば、アプリケーションに必要な Oracle メモリーを実質的に削減できます。

生産性

ストアド・プロシージャにより、開発の生産性が向上します。共通のプロシージャを中心にして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、各タスクの実行に必要な SQL 文を書き換えることなく、どんなアプリケーションからでもコールできます。データ管理の方法に変更があった場合も、修正する必要があるのはプロシージャのみで、プロシージャを使用するすべてのアプリケーションを修正する必要はありません。

整合性

ストアド・プロシージャにより、アプリケーションの整合性と一貫性が向上します。共通のプロシージャ群を中心としてのアプリケーションを開発すれば、コーディング・エラーの発生回数を少なくすることができます。

たとえば、プロシージャやファンクションが正確な結果を戻すかどうかをテストし、検証後は、そのプロシージャとファンクションを必要な数のアプリケーションで再使用できます。再びテストする必要はありません。そのプロシージャが参照するデータの構造になんらかの変更があった場合は、そのプロシージャのみを再コンパイルすれば十分です。そのプロシージャをコールする側のアプリケーションを修正する必要はありません。

無名 PL/SQL ブロックとストアド・プロシージャ

ストアド・プロシージャを作成し、それをスキーマ・オブジェクトとしてデータベースに格納できます。いったん作成してコンパイルしたプロシージャは、再コンパイルしなくても実行可能な名前付きのオブジェクトになります。さらに、依存性情報がデータ・ディクショナリに格納されるので、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、無名 PL/SQL ブロックを Oracle Tools やアプリケーションから Oracle Server に送信し、無名 PL/SQL ブロックを作成できます。Oracle によって PL/SQL ブロックがコンパイルされ、SGA の共有プールにそのコンパイル済みバージョンが入れられます。ただし、そのソース・コードやコンパイル済みバージョンは、現行のインスタンスの後も再使用できるようにデータベースに格納されることはありません。共有 SQL を使用すると、共有プールに入っている無名 PL/SQL ブロックがその共有プールからフラッシュされるまでの間は、そのブロックを再使用および共有できます。

どちらの場合でも、データベース・アプリケーションからデータベースまたはメモリーに格納されているデータベース・プロシージャに、PL/SQL ブロックを移すことにより、Oracle が実行時に不必要な再コンパイルを実行することがなくなり、アプリケーションと Oracle の全体的なパフォーマンスが向上します。

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、「パッケージ」18-11 ページを参照してください。）

定義者権限と起動者権限

「PL/SQL プロシージャ」は、プロシージャ定義に応じて、その所有者の権限（定義者権限）または現ユーザーの権限（起動者権限）で実行できます。

- 定義者権限プロシージャは、その定義者の権限で実行される。定義者権限プロシージャでは、ロールは使用禁止になります。
- 起動者権限プロシージャは、使用可能になっているロールも含め、すべての起動者の権限で実行される。

権限の詳細は 30-7 ページの「プロシージャのセキュリティに関するトピック」、ロールの詳細は 30-18 ページの「PL/SQL ブロックとロール」を参照してください。

現ユーザー

起動者権限プロシージャがソフトウェア・バンドル内で最初にコールされるプログラムであれば、起動者、つまり「現ユーザー」がセッション・ユーザーです。これは、ログインしているユーザーの場合と、リモート・プロシージャ・コール・セッションに対応付けられているユーザーの場合があります。起動者権限プロシージャが他の起動者権限プロシージャをコールしても、現ユーザーは変わりません。

ただし、定義者権限プロシージャをコールすると、そのプロシージャの所有者が現ユーザーになります。定義者権限プロシージャが起動者権限プロシージャをコールしても、現ユーザーは引き続き定義者権限プロシージャの所有者です。

定義者権限プロシージャを終了すると、現ユーザーはプロシージャの所有者から前の現ユーザー、つまり、定義者権限プロシージャをコールしたプロシージャの現ユーザーに戻ります。

外部参照の変換

PL/SQL プロシージャ内の「外部参照」とは、プログラム・ユニットの外側にあるオブジェクトを参照するものです。

- 定義者権限プロシージャの場合、すべての外部参照は、そのプロシージャを含むスキーマ内で変換される。
- 起動者権限プロシージャの場合、外部参照の変換方法は、それを含む文の種類に応じて異なる。次の名前は、起動者に対応付けられたスキーマ内で変換されます。
  - 表、ビューおよび順序など、DML 文中の名前
  - カーソル内の名前
  - 動的 SQL 文中と DBMS_SQL 文中の名前

  起動者権限プロシージャがコールするプログラム・ユニットの名前は、そのプロシージャを含むスキーマ内で変換されます。

起動者のスキーマ内で名前が変換されるため、アプリケーションはスキーマを指定しなくても、ユーザー固有の表にアクセスできます。詳細は、18-19 ページの「データベース・オブジェクトとプログラム・ユニットの名前変換」を参照してください。

ストアド・プロシージャの依存性の追跡

ストアド・プロシージャは、その本体で参照するオブジェクトに依存します。Oracle は、そのような依存性を自動的に追跡し、管理します。たとえば、プロシージャが参照している表の定義を変更した場合は、そのプロシージャを再コンパイルして、引き続き設計どおりに機能することを確認する必要があります。通常、Oracle はこのような依存性の管理を自動的に処理します。

依存性の追跡の詳細は、第 21 章「Oracle の依存性の管理」を参照してください。

外部プロシージャ

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

追加情報： 外部プロシージャと Inter-Language Method Services（ILMS）の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

パッケージ

パッケージは、相互に関連するプロシージャ、ファンクションおよびそれに関係するカーソルや変数を 1 単位としてカプセル化してデータベースに入れたものです。

パッケージは、仕様部および本体という 2 つの部分で構成されています。パッケージの「仕様部」ではパッケージのすべてのパブリックな構成体を宣言し、「本体」ではパッケージのすべての構成体（パブリックとプライベート）を定義します。パッケージを 2 つに分けることには、次のような利点があります。

- 開発者は開発サイクルを柔軟なものにすることができる。パッケージ本体を実際には作成せずに、仕様部を作成し、パブリック・プロシージャを参照するようにできます。
- パッケージ仕様部にパブリックに宣言されている仕様部とは別個に、パッケージ本体に含まれているプロシージャ本体を変更できる。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、つまり、プロシージャの再コンパイルが必要であることを示すマーク、が付けられることはありません。（依存性の詳細は、第 21 章「Oracle の依存性の管理」を参照してください。）

次の例では、銀行業務の取引きを処理する複数のプロシージャとファンクションを含むパッケージの仕様部と本体を作成します。

```
CREATE PACKAGE bank_transactions (null) AS
  minimum_balance  CONSTANT NUMBER := 100.00;
  PROCEDURE apply_transactions;
  PROCEDURE enter_transaction (acct   NUMBER,
                               kind   CHAR,
                               amount NUMBER);
END bank_transactions;

CREATE PACKAGE BODY bank_transactions AS

/*  Package to input bank transactions */

   new_status  CHAR(20);  /* Global variable to record status
                             of transaction being applied. Used
                             for update in APPLY_TRANSACTIONS. */
```

```
   PROCEDURE do_journal_entry (acct NUMBER,
                               kind CHAR) IS

/*  Records a journal entry for each bank transaction applied
    by the APPLY_TRANSACTIONS procedure. */

  BEGIN
        INSERT INTO journal
          VALUES (acct, kind, sysdate);
        IF kind = 'D' THEN
          new_status := 'Debit applied';
        ELSIF kind = 'C' THEN
          new_status := 'Credit applied';
        ELSE
          new_status := 'New account';
        END IF;
  END do_journal_entry;

  PROCEDURE credit_account (acct NUMBER, credit NUMBER) IS

/* Credits a bank account the specified amount. If the account
   does not exist, the procedure creates a new account first. */

        old_balance  NUMBER;
        new_balance  NUMBER;

  BEGIN
        SELECT balance INTO old_balance FROM accounts
          WHERE acct_id = acct
          FOR UPDATE OF balance; /* Locks account for credit update */

        new_balance := old_balance + credit;
        UPDATE accounts SET balance = new_balance
          WHERE acct_id = acct;
        do_journal_entry(acct, 'C');

  EXCEPTION
        WHEN NO_DATA_FOUND THEN  /* Create new account if not found */
          INSERT INTO accounts (acct_id, balance)
            VALUES(acct, credit);
          do_journal_entry(acct, 'N');
        WHEN OTHERS THEN  /* Return other errors to application */
          new_status := 'Error: ' || SQLERRM(SQLCODE);
  END credit_account;

  PROCEDURE debit_account (acct  NUMBER, debit NUMBER) IS
```

```
/* Debits an existing account if result is greater than the
   allowed minimum balance. */

        old_balance         NUMBER;
        new_balance         NUMBER;
        insufficient_funds  EXCEPTION;

  BEGIN
        SELECT balance INTO old_balance FROM accounts
          WHERE acct_id = acct
          FOR UPDATE OF balance;
        new_balance := old_balance - debit;
        IF new_balance >= minimum_balance THEN
          UPDATE accounts SET balance = new_balance

            WHERE acct_id = acct;
        do_journal_entry(acct, 'D');
        ELSE
          RAISE insufficient_funds;
        END IF;

  EXCEPTION
        WHEN NO_DATA_FOUND THEN
          new_status := 'Nonexistent account';
        WHEN insufficient_funds THEN
          new_status := 'Insufficient funds';
        WHEN OTHERS THEN  /* Returns other errors to application */
          new_status := 'Error: ' || SQLERRM(SQLCODE);
  END debit_account;

  PROCEDURE apply_transactions IS

/* Applies pending transactions in the table TRANSACTIONS to the
   ACCOUNTS table. Used at regular intervals to update bank
   accounts without interfering with input of new transactions. */

/* Cursor fetches and locks all rows from the TRANSACTIONS
   table with a status of 'Pending'. Locks released after all
         pending transactions have been applied. */

        CURSOR trans_cursor IS
          SELECT acct_id, kind, amount FROM transactions
            WHERE status = 'Pending'
            ORDER BY time_tag
            FOR UPDATE OF status;
```

```
  BEGIN
       FOR trans IN trans_cursor LOOP   /* implicit open and fetch */
         IF trans.kind = 'D' THEN
           debit_account(trans.acct_id, trans.amount);
         ELSIF trans.kind = 'C' THEN
           credit_account(trans.acct_id, trans.amount);
         ELSE
           new_status := 'Rejected';
         END IF;
         /* Update TRANSACTIONS table to return result of applying
            this transaction. */
         UPDATE transactions SET status = new_status
           WHERE CURRENT OF trans_cursor;
       END LOOP;
       COMMIT;  /* Release row locks in TRANSACTIONS table. */
  END apply_transactions;
  PROCEDURE enter_transaction (acct   NUMBER,
                               kind   CHAR,
                               amount NUMBER) IS

/* Enters a bank transaction into the TRANSACTIONS table. A new
   transaction is always put into this 'queue' before being
   applied to the specified account by the APPLY_TRANSACTIONS
   procedure. Therefore, many transactions can be simultaneously
   input without interference. */

  BEGIN
       INSERT INTO transactions
         VALUES (acct, kind, amount, 'Pending', sysdate);
       COMMIT;
  END enter_transaction;

END bank_transactions;
```

データベース管理者やアプリケーション開発者は、パッケージを使用することによって、多数の類似したルーチンを編成できます。さらに、機能性とデータベースのパフォーマンスが向上します。

パッケージの利点

パッケージは、互いに関連するプロシージャ、変数およびカーソルの定義に使用します。次のような分野で、その長所を発揮します。

- 関連するプロシージャと変数のカプセル化
- パブリック・プロシージャ、プライベート・プロシージャ、変数、定数およびカーソルの宣言

- 優れたパフォーマンス

カプセル化

ストアド・パッケージを使用すると、関連するストアド・プロシージャ、変数、データ型などをカプセル化（グループ化）し、名前を付けて 1 単位としてデータベースに格納できます。これにより、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。パッケージを使用する権限を付与すると、権限を付与されたユーザーは、そのパッケージのすべての構成体にアクセスできるようになります。

パブリックおよびプライベートなデータとプロシージャ

パッケージの定義方法により、変数、カーソルおよびプロシージャを次のどちらかに指定できます。

パブリック	パッケージのユーザーが直接アクセスできる。
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含まれているとします。このうち、3 つだけはユーザーが実行できるようにパブリックにし、残りのプロシージャはパッケージ内のプロシージャしかアクセスできないようにプライベートにするという形で、パッケージを定義できます。

パブリックおよびプライベートなパッケージ変数を、PUBLIC への権限付与と混同しないでください。詳細は、第 29 章「データベース・アクセスの制御」を参照してください。

パフォーマンスの向上

パッケージ内のプロシージャが初めてコールされた時点で、そのパッケージ全体がメモリーにロードされます。スタンドアロン・プロシージャは個別にロードする必要があるのと対照的に、このロードは 1 回の操作で完了します。そのため、関連するパッケージ・プロシージャがコールされても、すでにメモリーにあるコンパイル済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ければ、パッケージの構成体を参照するスキーマ・オブジェクト（常に仕様部を介してアクセスする）を再コンパイルする必要はありません。パッケージを使用すると、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 21 章「Oracle の依存性の管理」を参照してください。

Oracle が提供するパッケージ

Oracle は、データベースや PL/SQL の機能を拡張する機能を含んだ多数の PL/SQL パッケージを提供します。これらのほとんどのパッケージは、DBMS_SQL、DBMS_LOCK および DBMS_JOB のように、名前に "DBMS_" 接頭辞が付いています。また、UTL_HTTP や UTL_FILE のように "UTL_" 接頭辞が付いているパッケージや、"DEBUG_" または "OUTLN_" など、他の接頭辞が付いているパッケージもあります。

> **追加情報：** Oracle が提供するパッケージの詳細は、『Oracle8*i* パッケージ・プロシージャ・リファレンス』を参照してください。

Oracle がプロシージャとパッケージを格納する方法

プロシージャまたはパッケージを作成すると、次のステップが実行されます。

- プロシージャまたはパッケージをコンパイルする。
- 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・エラーの識別方法は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

コンパイル済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。これにより、コードをただちに実行し、多数のユーザーの間で共有できます。プロシージャまたはパッケージのコンパイル済みバージョンは、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しても、共有プールが使用した、修正された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、7-6 ページの「共有プール」を参照してください。

プロシージャとパッケージのデータベースへの格納

作成時とコンパイル時に、Oracle はプロシージャまたはパッケージに関する次の情報を自動的にデータベースに格納します。

スキーマ・オブジェクト名	この名前を使用して、プロシージャまたはパッケージを識別します。この名前は、CREATE PROCEDURE 文、CREATE FUNCTION 文、CREATE PACKAGE 文または CREATE PACKAGE BODY 文に指定します。
ソース・コードと解析ツリー	PL/SQL コンパイラはソース・コードを解析して、「解析ツリー」と呼ばれるソース・コードの解析済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、解析ツリーおよびオブジェクトの P コードがデータベースに格納されます。そのため、プロシージャまたはパッケージがその起動時に SGA 内に存在しなければ、そのプロシージャまたはパッケージのコンパイル済みのバージョンが PL/SQL エンジンによって SGA の共有プール・バッファに読み込まれます。解析ツリーは、そのプロシージャをコールするコードをコンパイルするときに使用されます。

データベース・プロシージャのすべての部分は、対応するデータベースのデータ・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。定義者権限プロシージャと起動者権限プロシージャでは、検査方法と実行方法が異なります（18-9 ページの「定義者権限と起動者権限」を参照）。

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のみです。

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・ディクショナリで調べます。プロシージャまたはパッケージは、最後にコンパイルされた後に次のいずれかの状況が発生すると無効になります。

- 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、ビューまたは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 から、またはプロシージャかパッケージの所有者から取り消された。
- プロシージャまたはパッケージが参照する 1 つ以上のスキーマ・オブジェクトに対する必要なスキーマ・オブジェクト権限が、PUBLIC から、またはプロシージャかパッケージの所有者から取り消された。

前述のいずれかの操作によって無効にされていない場合、そのプロシージャは「有効」です。有効なスタンドアロン・プロシージャまたはパッケージ・プロシージャがコールされると、コンパイル済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、プロシージャの再コンパイルおよび依存性の問題の詳細は、第 21 章「Oracle の依存性の管理」を参照してください。

プロシージャの実行

PL / SQL エンジンは、状況に応じて異なるステップでプロシージャまたはパッケージを実行します。

- 有効なプロシージャがメモリーにあれば、PL / SQL エンジンは単に P コードを実行する。
- 有効なプロシージャがメモリーになければ、PL / SQL エンジンはコンパイル済みの P コードをディスクからメモリーにロードして実行する。パッケージの場合、パッケージのすべての構成体（すべてのプロシージャ、変数など、実行可能なコード断片としてコンパイルされるもの）は 1 単位としてロードされます。

16-16 ページの図 16-2 のように、PL/SQL エンジンは、プロシージャ型の文はすべて自分で処理し、SQL 文は SQL 文エグゼキュータに渡すことによって、プロシージャの文を文単位で処理します。

データベース・オブジェクトとプログラム・ユニットの名前変換 定義者権限プロシージャの場合、すべての外部参照は、定義者のスキーマ内で変換されます。起動者権限プロシージャの場合、外部参照の変換方法は、それが含まれる文の種類に応じて異なります。

- DML 文中と動的 SQL 文中の外部参照は起動者のスキーマ内で変換され、Oracle は実行時に起動者権限を使用してアクセス権限をチェックする。このルールは、次の各文に含まれるデータベース・オブジェクト（表、ビューおよび順序など）の名前に適用されます。
  - SELECT、UPDATE、INSERT および DELETE 文
  - OPEN カーソル（カーソル宣言内の SELECT 文は、OPEN 時に変換される）
  - LOCK TABLE 文
  - 動的 SQL 文 EXECUTE IMMEDIATE および PREPARE
  - DBMS_SQL 文、つまり DBMS_SQL.PARSE() を使用して解析される文

  スキーマ・オブジェクトの名前は実行時に変換されますが、コンパイラは名前を定義者のスキーマにある一時オブジェクトに一時的に変換して、この種の各参照のタイプをコンパイル時に識別します。
- DML または動的 SQL 以外のすべての文中の外部参照は、定義者のスキーマ内で変換され、Oracle コンパイル時に定義者の権限を使用してアクセス権限をチェックする。このルールは、プロシージャがコールする他のプログラム・ユニット（パッケージ、プロシージャ、ファンクションおよびタイプなど）の名前に適用されます。

  たとえば、外部ファンクション・コール "x := func(1)" を伴う代入文の場合、ファンクション名はプロシージャの定義者のスキーマ内で変換され、コンパイル時には定義者の権限で "func" へのアクセスがチェックされます。

データベース・リンクの名前変換 PL/SQL プロシージャ内のデータベース・リンク名は、前項のルールに従って変換されます。リモート・データベースへの接続には、次のいずれかの認可 ID が使用されます。

1. 名前付きリンクの場合、Oracle はリンク内で指定されたユーザー名を使用して、リモート・データベースに接続します。この動作は、定義者権限プロシージャおよび起動者権限プロシージャと同じです。

```
CREATE DATABASE LINK link1
    CONNECT TO scott IDENTIFIED BY tiger
    USING connect_string;
```

JOE が所有するプロシージャで LINK1 が使用される場合、このリンクで指定されている名前は SCOTT なので、そのプロシージャの起動者に関係なく SCOTT として接続されます。

2. 無名リンクの場合、Oracle はセッション・ユーザー名を使用してリモート・データベースに接続します。この動作は、定義者権限プロシージャおよび起動者権限プロシージャと同じです。

```
CREATE DATABASE LINK link2
    USING connect_string;
```

JOE が所有するプロシージャで無名リンク LINK2 が使用される場合に、ユーザー SCOTT がこのプロシージャを起動すると、リモート・データベースには SCOTT として接続されます。

3. 現ユーザーのリンクの場合、定義者権限プロシージャと起動者権限プロシージャでは動作が異なります。

   – 起動者権限プロシージャの場合、Oracle は起動者の許可 ID を使用してリモート・ユーザーで接続する。

   ```
   CREATE DATABASE LINK link3
       CONNECT TO CURRENT_USER
       USING connect_string;
   ```

   JOE が所有する起動者権限プロシージャをグローバル・ユーザー SCOTT が起動する場合、現ユーザーは SCOTT なので、LINK3 はリモート・データベースにユーザー SCOTT で接続します。

   – 定義者権限プロシージャの場合、Oracle は所有者の許可 ID を使用してリモート・ユーザーで接続する。JOE が定義者権限プロシージャを所有している場合は、JOE が現ユーザーになるため、LINK3 はリモート・データベースにグローバル・ユーザー JOE で接続します。

19

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）機能について説明します。この章の内容は、次のとおりです。

- メッセージ・キューイングの基礎知識
- 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： この章で説明している機能は、Oracle8*i* Enterprise Edition を購入した場合にのみ使用できます。

追加情報： Oracle AQ の詳細は、『Oracle8*i* アプリケーション開発者ガイド アドバンスト・キューイング』を参照してください。

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する必要があるプログラムに適しています。従来のクライアント / サーバーのアーキテクチャは、このモデルに基づいています。

同期通信モデルの大きな欠点は、アプリケーションが動作するために、多くのプログラムを使用可能かつ実行中の状態にしておく必要があるという点です。ネットワークまたはマシンの障害が発生すると、プログラムは機能を停止します。

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、アプリケーションへのデータ入力や、操作の実行を必要とする場合があります。要求を受けるプログラムは、キューから要求を取り出し、その要求に従って動作します。このモデルは、要求をキューに入れた後でも作業を続けられるアプリケーションに適しています。そのようなアプリケーションでは、応答を待機するために動作が止まることはありません。

ネットワーク、マシンおよびアプリケーションに障害が発生しても正しく動作する遅延実行の場合、要求は永続的に格納されて、1 度だけ処理される必要があります。このことは、永続キューイングとトランザクションの保護を組み合せることによって実現できます。

通常、クライアント / サーバーの要求を「1 度だけ」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うえでに重要になります。たとえば、特定の価格で株の買い注文を要求する場合、その要求の伝送時、受信時または実行時にネットワーク障害あるいはシステム障害が発生したとしても、要求をゼロ回または 2 回実行することは認められません。

Oracle Advanced Queuing

Oracle Advanced Queuing（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、Java、C/C++ および Visual Basic で定義されたインタフェースを通じて、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。

- メッセージ用に構造化されたペイロード
- キュー内のメッセージの優先順位と順序付け
- 各メッセージための実行時間枠を指定する機能
- 標準 SQL を使用してキューを問い合せる機能
- アプリケーションの開発と管理を単純化する統合されたトランザクション
- 複数のメッセージをまとめてデキューする機能
- 複数のレシピエントを指定する機能
- ローカルまたはリモートの Oracle データベース内のキューにメッセージを伝播させる機能
- コンテント・ベースのフィルタリングによるルールベースのパブリッシュ / サブスクライブ
- 複数キューでメッセージを待機する機能
- 永続キューイングと非永続キューイング
- キューに格納されたメッセージと他のキューに伝播したメッセージに関する統計
- 分析のためのメッセージとメッセージ履歴の保存
- キュー・レベルのアクセス制御
- 例外処理のサポート
- パフォーマンス向上を目的とした Oracle Parallel Server 環境のサポート
- コールバック関数を使用した非同期通知

Oracle 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、高い拡張性および高い信頼性など、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用できます。

キューイング・エンティティ

Oracle AQ の基本エンティティは、メッセージ、キュー、キュー表、エージェント、レシピエント、レシピエントとサブスクライバのリスト、ルール、ルールベースのサブスクライバおよびキューモニターです。

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 が使用するメッセージ・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・コールによって作成され、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にする必要があります。メッセージを取り出して処理することがなんらかの理由でできない場合、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、変更、開始、停止および削除するには、DBMS_AQADM パッケージを使用します。

キュー表

キューは「キュー表」に格納されます。各キュー表はデータベース表であり、1 つ以上のキューが含まれています。各キュー表にはデフォルトの例外キューが含まれます。

キュー表を作成すると、約 25 列を含むデータベース表が作成されます。それらの列には、Oracle 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 つの索引が作成されます。ビューを使用することにより、メッセージ・データを問い合せることができます。索引は、メッセージ・データへのアクセスを速めるために使用します。

エージェント

エージェントは、キューの使用者です。エージェントには、キューにメッセージを入れる（エンキューする）プロデューサと、キューからメッセージを取り出す（デキューする）コンシューマの 2 種類があります。任意の数のプロデューサとコンシューマが同時に 1 つのキューにアクセスできます。

エージェントは、名前、アドレスおよびプロトコルによって識別されます。リモート・データベース上のエージェントに対して現在サポートされているプロトコルは、*queue_name@dblink* 形式のアドレスを使用する Oracle データベース・リンクのみです。

レシピエント

メッセージの「レシピエント」は、その名前のみで指定することができます。その場合、レシピエントはメッセージがエンキューされたキューから、そのメッセージをデキューする必要があります。レシピエントは、名前とプロトコル値 0 のアドレスで指定できます。このアドレスは、同じデータベース内、または別の Oracle8 データベース内（データベース・リンクで識別する）の別のキューの名前にする必要があります。別の Oracle8 データベースの場合は、メッセージが指定のキューに伝播され、指定の名前を持つコンシューマがデキューできます。レシピエント名が NULL であれば、そのメッセージはアドレスで指定したキューに伝播し、アドレスで指定したキューのサブスクライバがデキューできます。プロトコル・フィールドがゼロ以外の値であれば、名前フィールドとアドレス・フィールドはシステムで解釈されず、メッセージは特殊なコンシューマがデキューできます（伝播の項のサードパーティ・サポートを参照）。

レシピエント・リストおよびサブスクライバ・リスト

1 つのメッセージを、複数のコンシューマで使用するように設計できます。これには、次の 2 つの方法があります。

- エンキュー元は、メッセージのレシピエントとしてメッセージを受信するコンシューマを明示的に指定できる。レシピエントは、名前、アドレスおよびプロトコルで指定するエージェントです。
- キュー管理者は、キューからメッセージを受信するレシピエントのデフォルト・リストを指定できる。デフォルト・リストで指定されたレシピエントを、サブスクライバといいます。レシピエントを指定しないでメッセージをエンキューすると、そのメッセージはすべてのサブスクライバに暗黙的に送信されます。

キューごとに異なるサブスクライバを指定し、同じレシピエントが複数のキューのサブスクライバになることができます。また、サブスクライバ・リストを上書きすることによって、キュー内の特定のメッセージを、そのキューのサブスクライバかどうかに関係なく、特定のレシピエントに送信できます。

ルール

「ルール」は、その規則に準拠するメッセージへのサブスクライブに関係のある 1 人以上のサブスクライバを定義するものです。この基準を満たしているメッセージは、関係のあるサブスクライバに配信されます。もう 1 つの方法では、ルールによって、サブスクライバが関係のある主題について、キュー内のメッセージにフィルタがかけられます。

ルールは、SQL 問合せの WHERE 句に似た構文を使用して、ブール式（TRUE または FALSE に評価される式）として指定します。このブールには、次の条件を含めることができます。

- メッセージ・プロパティ（現行の priority と corrid）
- ユーザー・データのプロパティ（オブジェクト・ペイロードのみ）
- 関数（SQL 問合せの WHERE 句で指定）

ルールベースのサブスクライバ

ルールベースのサブスクライバとは、デフォルトのレシピエント・リストに対応付けられたルールを持つサブスクライバです。ルールベース・サブスクライバには、そのメッセージについて対応するルールが TRUE に評価された場合に、レシピエントが明示的に指定されていないメッセージが送信されます。

キュー・モニター

キュー・モニターは、キュー内のメッセージを監視するオプションのバックグラウンド・プロセスです。キュー・モニターは、メッセージの時間切れ、再試行および遅延を管理するメカニズムを提供し（19-7 ページの「実行の時間枠」を参照）、これによって、間隔統計を収集できます（19-10 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_PROCESSES では、インスタンスの起動時に、1 つ以上のキュー・モニター・プロセスの作成を指定します。

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用できます。（RAW データ型は、構造化されていないペイロードに使用できます。）

統合されたデータベース・レベルの操作サポート

Oracle 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 を使用することにより、メッセージのプロパティ、メッセージの履歴およびペイロードにアクセスできます。索引などの使用可能なすべての SQL テクノロジを使用することにより、メッセージへのアクセスを最適化できます。

AQ_ADMINISTRATOR ロールにより、キューに関する情報にアクセスできます。

実行の時間枠

特定の時間枠内にメッセージを消費するように指定できます。メッセージには、指定した時間（遅延時間）の経過後に限って処理できること、および指定した制限時間が切れる前に消費する必要があることを示すマークを設定できます。

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用されます。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（19-10 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 つ作成します。このパラメータを 2 ～ 10 に設定すると、Oracle はその数の QMN*n* プロセスを作成します。このパラメータを指定しないか 0 に設定すると、キュー・モニター・プロセスは作成されません。
キュー・モニター操作を開始および停止する DBMS_AQADM パッケージ内のプロシージャは、このパラメータによって少なくとも 1 つのキュー・モニター・プロセスがインスタンス起動の一部として起動された場合に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用できます。

ナビゲーション

キューからメッセージを選択するには、いくつかの方法があります。最初のメッセージを選択できますが、メッセージを選択して一貫読込みスナップショットを確定すると、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。新しい一貫読込みスナップショットは、キューから最初のメッセージを選択するたびに取得されます。

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

メッセージの優先順位と順序付け

メッセージの消費順序を指定するには、ソート順（キュー内のすべてのメッセージの順序を決めるのに使用する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）および順序逸脱（あるメッセージを他のメッセージとの関係で配置する）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、ひとりのコンシューマはすぐに使用できる最初のメッセージを取り出します。別のコンシューマが使用中のメッセージはスキップします。

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消したりできます。メッセージをブラウズしても、それはその後の処理に引き続き使用できます。メッセージを取り消すと、DEQUEUE 要求には使用できなくなります。キューのプロパティによっては、取り消されたメッセージがキュー表内に保持されることもあります。

メッセージ着信の待機

空のキューに対しても DEQUEUE を発行できます。要求がメッセージの着信を待機するかどうかと、待機する場合にはその待機時間を指定できます。

遅延による再試行

メッセージの使用は、1 度かぎりであることが必要です。メッセージのデキューに失敗し、トランザクションがロールバックされた場合、メッセージは、ユーザー指定の遅延時間経過後に再処理できるようになります。指定した限度まで再処理が試行されます。

例外キュー

特定の制約内、つまり実行時間枠または再試行の制限内に、メッセージを使用できないことがあります。このような条件が生じると、メッセージはユーザー指定の例外キューに移されます。

可視性

ENQUEUE / DEQUEUE 要求は、多くの場合、その要求を含むトランザクションの一部となっています。これによりトランザクションは適切に動作します。ただし、要求の結果を他のトランザクションからすぐに参照できるようにするために、要求自体をトランザクションとして指定することもできます。

メッセージのグループ化

特定のキューに入っているメッセージを、一度に 1 人のユーザーのみが使用できる集合となるようグループ化できます。そのためには、メッセージのグループ化の可能なキュー表内にキューを作成する必要があります。

1 つのグループに属するメッセージはすべて同じトランザクション内に作成される必要があり、1 つのトランザクション内に作成されたメッセージはすべて同じグループに属します。この機能により、複雑なメッセージを単純なメッセージにセグメント化できるようになります。たとえば、送付状が入っているキューに送信されるメッセージの場合、ヘッダー・メッセージで開始し、その後に明細を示すメッセージ、その後に後書きメッセージが続くメッセージ・グループとして構成できます。

保持

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、データ・ウェアハウスの操作およびデータ探索操作のために使用できます。

メッセージの履歴

Oracle 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、アプリケーションによって組み立てられることの多い「イベント・ジャーナル」を表すものです。Oracle AQ では、アプリケーションが自動的にイベント・ジャーナルを作成するように設計されています。

キュー・レベルのアクセス制御

Oracle 8*i* では、8.1 形式のキューの所有者が、キューに対するキュー・レベルの権限を付与または取消しできます。DBA は、任意のデータベース・ユーザーに、新しい AQ システム・レベルの権限を付与したり、取り消したりできます。また、任意のデータベース・ユーザーを AQ 管理者にすることもできます。

他のデータベースへのメッセージの伝播

あるデータベースにエンキューされたメッセージは、別のデータベースのキューに伝播できます。ソース・キューと宛先キューのデータ型は一致する必要があります。

メッセージの伝播により、アプリケーションは、同じデータベースや同じキューに接続していなくても、相互に通信できます。伝播では、ローカル・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 を使用します。どちらの場合も Oracle AQ が使用可能にされている必要があります。

メッセージ伝播をスケジューリング（またはスケジュール解除）して、開始時間、伝播の時間枠、および後で定期的なスケジュールで行う伝播の時間枠に対する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、メッセージ伝播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（SNP*n*）は、メッセージの伝播を処理します。伝播を使用可能にするには、初期化パラメータ JOB_QUEUE_PROCESSES を使用して少なくとも 1 つのジョブ・キュー・プロセスを起動する必要があります。

伝播に関する統計

伝播統計は、スケジュールの範囲内で伝播された合計メッセージ数 / 平均メッセージ数 / バイト数の形式で取得できます。この情報を使用して、メッセージ伝播のパフォーマンスをチューニングできます。

非永続キュー

AQ は、非永続メッセージをサブスクライバに非同期に送信できます。これらのメッセージはイベント・ドリブン方式であり、システム（またはインスタンス）障害の発生後は存続しません。AQ では、共通 API を通じて永続メッセージと非永続メッセージがサポートされます。

パブリッシュ / サブスクライブのサポート

アプリケーション間でパブリッシュ / サブスクライブ方式のメッセージングを行えるように、一連の機能の組合せが導入されています。これらの機能には、ルールベース・サブスクライバ、メッセージ伝播、リスニング機能および通知機能があります。トリガーを使用すると、システム・イベントとユーザー・イベントを発行できます（20-17 ページの「システム・イベントとユーザー・イベントのトリガー」を参照）。

Oracle Parallel Server 環境のサポート

アプリケーションは、キュー表に対するインスタンス親和性を指定できます。AQ をパラレル・サーバーおよび複数インスタンスと併用する場合は、キュー・モニタースケジューリング用にインスタンス間でキュー表をパーティション化するために、この情報が使用されます。キュー表は、ユーザーが指定したインスタンスのキュー・モニターによって監視されます。インスタンスの親和性を指定しなければ、キュー表は使用可能なインスタンス間で任意に分割されます。キュー表にアクセスするアプリケーションと、それを監視するキュー・モニターの間で「ping」できます。このインスタンス親和性を指定すると、アプリケーションは他のインスタンスからのキュー表とそのキューへのアクセスが可能となります。

この機能により、異なるインスタンスで実行中のキュー・モニターと AQ 伝播ジョブの間での「ping」が防止されます。Oracle8*i* リリース 8.1.5 では、キュー表についてインスタンス親和性（1 次および 2 次）を指定できます。AQ をパラレル・サーバーおよび複数インスタンスと併用する場合は、キュー・モニタースケジューリングおよび伝播用に、インスタンス間でキュー表をパーティション化するために、この情報が使用されます。キュー表は、常に 1 つのインスタンスとの親和性を持ちます。親和性を明示的に指定しなければ、使用可能な任意のインスタンスがキュー表の所有者となります。キュー表の所有者がいなくなると、2 次インスタンスまたは使用可能な他のインスタンスが、そのキュー表の所有権を引き継ぎます。

キューイングの統計

Oracle 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、待ち状態および期限切れのメッセージの個数が含まれます。

1 つ以上のキュー・モニター・プロセスを起動して、次のような間隔統計を保持する必要があります。

- 各状態（準備完了、待ち状態および期限切れ）にあるメッセージの個数
- 待ち状態メッセージの平均待ち時間
- 待ち状態メッセージの合計待ち時間

非同期通知

OCI クライアントは、新しいコールである OCISubscriptionRegister を使用して、メッセージ通知のコールバックを登録できます。クライアントは、サブスクリプション名とコールバックを指定する登録コールを発行します。サブスクリプション・メッセージが受信されると、コールバックが起動されます。コールバックは、メッセージを取り出すために明示的なデキューを発行できます。

リスニング機能（複数キューでのメッセージ待機）

リスニング・コールは、複数のキュー上のメッセージを待機できるようにする、ブロック化コールです。ゲートウェイ・アプリケーションは、このコールを使用して 1 組のキューを監視できます。また、サブスクリプション・リストのメッセージを待機することもできます。リスニングによって正常に値が戻される場合は、デキューを使用してメッセージを取り出す必要があります。

相関識別子

各メッセージには識別子を割り当てることができます。この識別子を使用することによって、特定のメッセージを取り出せます。

インポート / エクスポート

キューをインポートまたはエクスポートすると、その基礎となるキュー表に関連付けられたディクショナリ表がインポートまたはエクスポートされます。キューのインポートおよびエクスポートは、キュー表単位でしか実行できません。

キュー表をエクスポートすると、表定義情報とキュー・データの両方がエクスポートされます。キュー表をインポートすると、エクスポート・アクションの手順によりキュー・ディクショナリが維持されます。キュー表のデータもエクスポートされるため、キュー表のデータのトランスポート時には、アプリケーション・レベルでのデータの整合性が維持されるようユーザー側で注意する必要があります。

複数のレシピエントをサポートするキュー表には、重要なキューのメタデータが含まれる索引構成表があります。このメタデータはキューの操作の基本となるものであるため、インポート後にこの表に含まれるキューが動作するためには、キュー表だけではなく、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。ただし、インポート時に別の問題（ロールバック・セグメントの領域を使用し尽くしたなど）が生じた場合は、その問題を訂正してからインポートを再実行する必要があります。

追加情報： Oracle AQ の詳細は、『Oracle8*i* アプリケーション開発者ガイド アドバンスト・キューイング』を参照してください。

20

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、PL/SQL、Java または C で記述され、データベースに格納されているプロシージャであり、表またはビューが変更された場合、またはなんらかのユーザー・アクションやデータベース・システム・アクションが発生した場合に、暗黙的に実行（起動）されます。トリガーが起動されるのは、特定のスキーマ・オブジェクトでの DML 文、スキーマまたはデータベース内で発行された DDL 文、ユーザーのログオンまたはログオフ・イベント、サーバー・エラー、データベース起動またはインスタンス停止のいずれかが発生した場合です。

この章の内容は、次のとおりです。

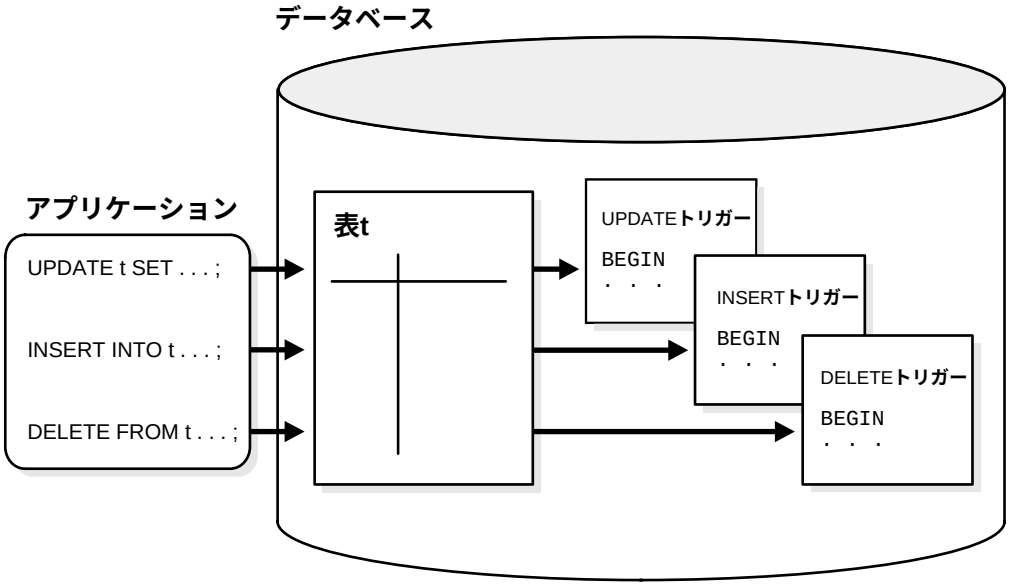
トリガーの基礎知識

Oracle では、表（または、場合によってはビュー）に対して INSERT 文、UPDATE 文または DELETE 文を発行するか、データベース・システム・アクションが発生したときに暗黙的に実行されるプロシージャを定義できます。このようなプロシージャを「トリガー」といいます。これらのプロシージャは、PL/SQL または Java で記述してデータベースに格納するか、C のコールアウトとして記述できます。

トリガーは、第 18 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。データベースに格納されているトリガーには、1 単位として実行可能な SQL や PL/SQL 文を格納することができ、また、トリガーは他のストアド・プロシージャを起動できます。ただし、トリガーとプロシージャとでは、起動する方法が異なります。プロシージャは、ユーザー、アプリケーションまたはトリガーによって明示的に実行されます。トリガー（1 つ以上）は、接続ユーザーや使用中のアプリケーションに関係なく、トリガーの元になるイベントが発生した時点で暗黙的に起動（実行）されます。

図 20-1 に示すデータベース・アプリケーションには、データベースに格納された複数のトリガーを暗黙のうちに起動する SQL 文がいくつか含まれています。データベースには、トリガーとそれに対応付けられている表が別々に格納されていることに注目してください。

図 20-1 トリガー



また、トリガーは C プロシージャをコールアウトし、計算集約型の操作に使用できます。

トリガーを起動するイベントには、表中のデータを変更する DML 文（INSERT、UPDATE または DELETE）、DDL 文、起動、停止、エラー・メッセージなどのシステム・イベント、またはログオンとログオフなどのユーザー・イベントがあります。詳細は、20-6 ページの「トリガー・イベントまたはトリガー文」を参照してください。

注意： Oracle Forms でも、種類の違うトリガーを定義、格納および実行できます。ただし、この章で説明するトリガーと Oracle Forms トリガーを混同しないでください。

トリガーの使用方法

トリガーは Oracle の標準機能を補い、高度にカスタマイズされたデータベース管理システムを提供します。たとえば、トリガーによって、表に対する DML 操作を通常の業務時間内のみに制限できます。また、トリガーを使用すると、平日の特定の時間にしか DML 操作が実行されないようにも制限できます。トリガーには、その他に次のような使用方法があります。

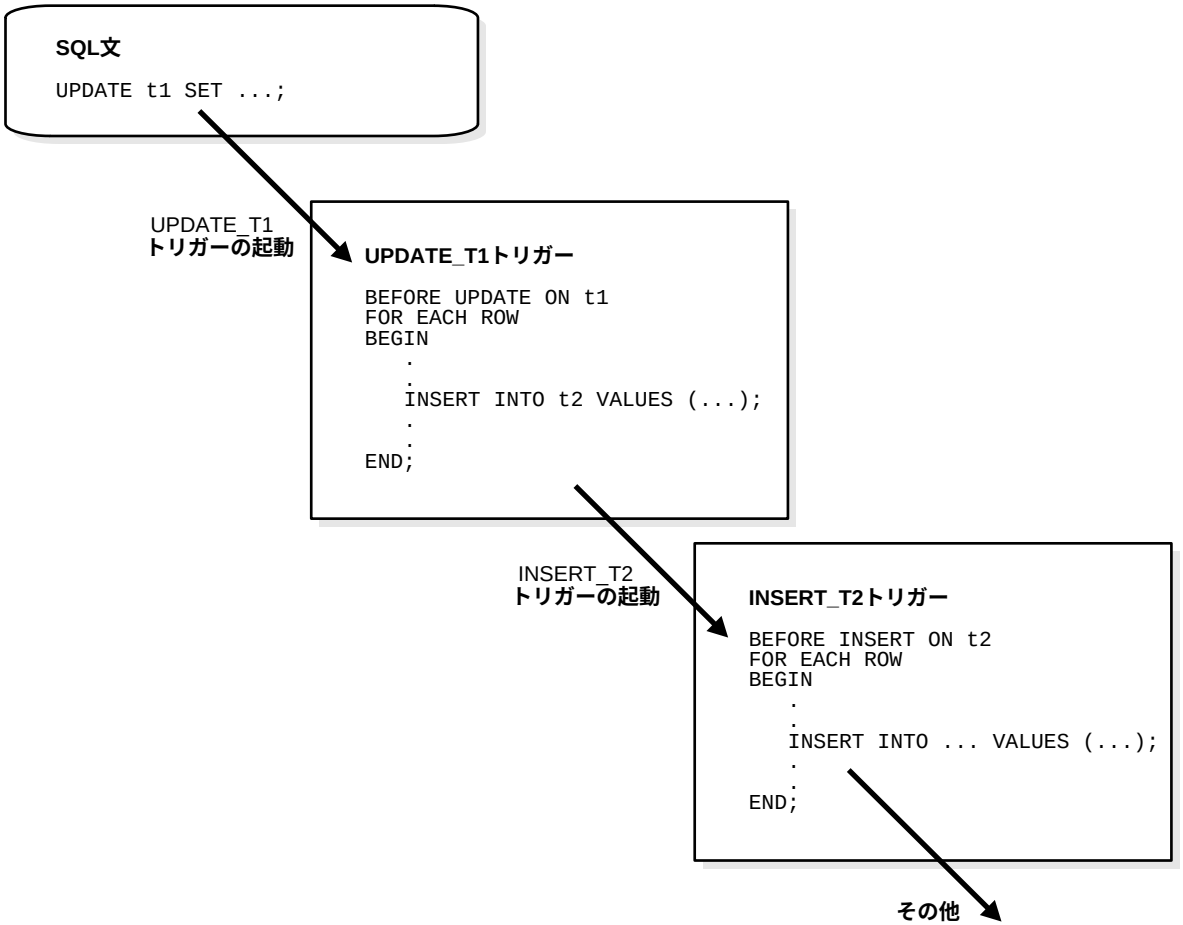
- 導出列の値の自動生成
- 不正なトランザクションの防止
- 複雑なセキュリティ認可の施行
- 分散データベース内の複数ノードにまたがる参照整合性の施行
- 複雑なビジネス・ルールの施行
- 透過的なイベント・ロギングの実現
- 高度な監査の実現
- 表レプリケーションの同期の維持
- 表アクセスについての統計情報の収集
- ビューに対して DML 文が発行された場合の表データの修正
- データベース・イベント、ユーザー・イベントおよび SQL 文に関する情報の、サブスクライバ・アプリケーションに対する発行

追加情報： これらの多数のトリガーの使用例は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

トリガーの使用上の注意

トリガーはデータベースのカスタマイズに使用すると便利ですが、必要な場合にのみ使用してください。トリガーを使用しすぎると相互依存性が複雑になるため、大規模アプリケーションのメンテナンスが困難になります。たとえば、トリガーが起動すると、そのトリガー・アクションに含まれる SQL 文によって他のトリガーが起動され、「トリガーがカスケード状態になる」ことがあります。図 20-2 に、この状態を示します。

図 20-2 トリガーのカスケード



トリガーと宣言整合性制約

トリガーと整合性制約を併用すると、任意の整合性規則を定義して施行できます。ただし、トリガーを使用してデータ入力に制約を適用するのは、次の場合に限定してください。

- 子表と親表が分散データベースの別々のノードにある場合に、参照整合性を施行する。
- 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。
- 必要な参照整合性規則を、次の整合性制約を使用して施行できない場合。
  - NOT NULL、UNIQUE キー
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK
  - DELETE CASCADE
  - DELETE SET NULL

整合性制約の詳細は、28-4 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 20-3 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 20-3 REORDER トリガー

REORDERトリガー

```
AFTER UPDATE OF parts_on_hand ON inventory
```

トリガー文

```
WHEN (new.parts_on_hand < new.reorder_point)
```

トリガー条件

トリガー・アクション

```
FOR EACH ROW
DECLARE                          /* a dummy variable for counting */
   NUMBER X;
BEGIN
   SELECT COUNT(*) INTO X        /* query to find out if part has already been */
   FROM pending_orders           /* reordered–if yes, x=1, if no, x=0 */
   WHERE part_no=:new.part_no;


IF x = 0
THEN                             /* part has not been reordered yet, so reorder */
   INSET INTO pending_orders
   VALUES (new.part_no, new.reorder_quantity, sysdate);
 END IF;                         /* part has already been reordered */
END;
```

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、データベース・イベントまたはユーザー・イベントのことです。トリガー・イベントは、次に示すものの 1 つ以上からなります。

- 特定の表（または、場合によってはビュー）に対する INSERT 文、UPDATE 文または DELETE 文
- スキーマ・オブジェクトに対する CREATE 文、ALTER 文または DROP 文
- データベース起動またはインスタンス停止
- 特定または任意のエラー・メッセージ
- ユーザーのログオンまたはログオフ

たとえば、図 20-3 のトリガーを実行する文は次のとおりです。

```
. . . UPDATE OF parts_on_hand ON inventory . . .
```

この文は、INVENTORY 表の PARTS_ON_HAND 列を更新すると、トリガーが起動することを意味します。なお、トリガー・イベントが UPDATE 文の場合は、どの列が更新されるとトリガーが起動されるかを示す列リストを指定できます。INSERT 文と DELETE 文では情報行全体が処理の対象になるため、それらの文には列のリストを指定できません。

トリガー・イベントには、次のように複数の SQL 文を指定できます。

```
. . . INSERT OR UPDATE OR DELETE OF inventory . . .
```

この場合、INVENTORY 表に対して INSERT 文、UPDATE 文または DELETE 文を発行すると、トリガーが起動されます。複数のタイプの SQL 文でトリガーが起動される場合は、条件述語を使用してトリガー文のタイプを検出できます。これにより、トリガーを起動した文のタイプに応じて異なるコードを実行するトリガーを作成できます。

トリガー条件

トリガー条件にはブール（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 になる必要があります。トリガー条件の評価が FALSE または UNKNOWN になった場合、トリガー・アクションは実行されません。この例では、次のようにしてトリガーを制限しています。

```
new.parts_on_hand < new.reorder_point
```

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、Java プログラムまたは C コールアウト）です。これらの SQL 文やコードは、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・プロシージャと同様に、トリガー・アクションでも SQL 文と PL/SQL 文または Java 文の使用、PL/SQL の言語要素（変数、定数、カーソル、例外など）または Java 言語要素の定義、またはストアド・プロシージャのコールが可能です。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、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
- システム・イベントとユーザー・イベントのトリガー

行トリガーと文トリガー

トリガーを定義するときに、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各行ごとに 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に依存する場合には、行トリガーが便利です。たとえば、図 20-3 は、トリガー文の影響を受ける各行の値を使用する行トリガーの一例です。

文トリガー

「文トリガー」は、トリガー文の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も）、トリガー文のために 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、文トリガーを使用します。

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。（もう 1 種類のトリガーの説明は、20-11 ページの「INSTEAD OF トリガー」を参照。）

DML 文によって起動される BEFORE トリガーと AFTER トリガーは、表にのみ定義でき、ビューには定義できません。ただし、ビューに対して INSERT 文、UPDATE 文または DELETE 文を発行すれば、そのビューの実表のトリガーが起動されます。DDL 文によって起動される BEFORE トリガーと AFTER トリガーは、データベースやスキーマにのみ定義でき、特定の表には定義できません。BEFORE トリガーおよび AFTER トリガーを使用して DML 文および DDL 文の情報をサブスクライバに発行する方法は、20-17 ページの「システム・イベントとユーザー・イベントのトリガー」を参照してください。

BEFORE トリガー

BEFORE トリガーは、トリガー文を実行する前にトリガー・アクションを実行します。このタイプのトリガーは、次のような場合によく使用します。

- トリガー文を実行してよいかどうかを、トリガー・アクションによって決める場合。BEFORE トリガーを使うことにより、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結果的にロールバックすることになるのを避けられます。
- 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、トリガー・アクションを実行する前に、トリガー文を完了させたい場合に使用します。

トリガー・タイプの組合せ

これまでに説明したオプションを使用して、次の 4 タイプの行トリガーと文トリガーを作成できます。

- **BEFORE 文トリガー**

  トリガー文を実行する前にトリガー・アクションが実行されます。

- **BEFORE 行トリガー**

  トリガー文の影響を受ける各行が修正される前、かつ該当する整合性制約の検査の前に、トリガー条件に違反していない限りトリガー・アクションが実行されます。

- **AFTER 行トリガー**

  トリガー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す。

- **AFTER 文トリガー**

  トリガー文を実行し、遅延整合性制約を適用した後に、トリガー・アクションが実行されます。

ある表に対する 1 つの文に、同じタイプのトリガーを複数指定できます。たとえば、EMP 表に対する UPDATE 文に 2 つの BEFORE 文トリガーを指定できます。同じタイプのトリガーを複数指定することにより、同じ表に対してトリガーを持つ複数のアプリケーションをモジュール化してインストールできます。また、Oracle スナップショット・ログは AFTER 行トリガーを使用するため、Oracle によって定義される AFTER 行トリガーに加えて、ユーザー独自の AFTER 行トリガーを設計できます。

各種 DML 文（INSERT、UPDATE または DELETE）に対して、前述のトリガーを必要な数だけ作成できます。

たとえば、表 SAL で、表がアクセスされる時期と発行される問合せのタイプを確認したい場合を考えます。次の例は、表 SAL に対するアクション（UPDATE、DELETE または INSERT など）の時間とタイプ別にその情報を記録するサンプルのパッケージとトリガーを示しています。グローバル・セッション変数 STAT.ROWCNT は、BEFORE 文トリガーによって 0 に初期化されます。その後、行トリガーが実行されるたびに増やされます。最終的に、AFTER 文トリガーによって、統計情報が表 STAT_TAB に保存されます。

SAL 表のサンプル・パッケージおよびトリガー

```
DROP TABLE stat_tab;
CREATE TABLE stat_tab(utype CHAR(8),
      rowcnt INTEGER, uhour INTEGER);

CREATE OR REPLACE PACKAGE stat IS
 rowcnt INTEGER;
END;
/

CREATE TRIGGER bt BEFORE UPDATE OR DELETE OR INSERT ON sal
BEGIN
 stat.rowcnt := 0;
END;
/

CREATE TRIGGER rt BEFORE UPDATE OR DELETE OR INSERT ON sal
FOR EACH ROW BEGIN
 stat.rowcnt := stat.rowcnt + 1;
END;
```

```
/

CREATE TRIGGER at AFTER UPDATE OR DELETE OR INSERT ON sal
DECLARE
 typ CHAR(8);
 hour NUMBER;
BEGIN
 IF updating
 THEN typ := 'update'; END IF;
 IF deleting THEN typ := 'delete'; END IF;
 IF inserting THEN typ := 'insert'; END IF;

 hour := TRUNC((SYSDATE - TRUNC(SYSDATE)) * 24);
 UPDATE stat_tab
    SET rowcnt = rowcnt + stat.rowcnt
   WHERE utype = typ
     AND uhour = hour;
 IF SQL%ROWCOUNT = 0 THEN
   INSERT INTO stat_tab VALUES (typ, stat.rowcnt, hour);
 END IF;

EXCEPTION
 WHEN dup_val_on_index THEN
   UPDATE stat_tab
      SET rowcnt = rowcnt + stat.rowcnt
     WHERE utype = typ
       AND uhour = hour;
END;
/
```

INSTEAD OF トリガー

注意： INSTEAD OF トリガーを使用できるのは、Oracle8*i* Enterprise Edition を購入した場合のみです。これらのトリガーは、リレーショナル・ビューとオブジェクト・ビューで使用できます。

INSTEAD OF トリガーを使用すると、SQL DML 文（INSERT、UPDATE および DELETE）で直接変更できないビューを透過的に変更できるようになります。他のタイプのトリガーとは異なり、Oracle はトリガー文を実行する「かわりに（instead of）」このトリガーを起動するため、このようなトリガーを INSTEAD OF トリガーといいます。

通常の INSERT、UPDATE および DELETE 文をビューに対して記述し、それに応じて基礎となる表が更新されるように INSTEAD OF トリガーを起動させることができます。INSTEAD OF トリガーは、変更があったビューの行ごとにアクティブ化されます。

ビューの変更

ビューを変更する操作には、本質的に曖昧さの問題があります。

- ビューで行を削除する操作は、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選択されないようにする操作のどちらかを意味する。
- ビューに行を挿入する操作は、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する。
- 結合が含まれるビューで列を更新すると、ビューに表示されない別の列の意味が変わってしまうことがある。

オブジェクト・ビューには、その他にも問題があります（第 15 章「オブジェクト・ビュー」を参照）。たとえば、オブジェクト・ビューの主な使用方法は、マスターと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（次の項を参照）。INSTEAD OF トリガーは、それ以外の手段では変更できないリレーショナル・ビューだけでなく、オブジェクト・ビューに対しても使用できます。

ビューが本来は変更可能であっても、挿入、更新または削除する値の妥当性検査を実行する必要があります。この場合にも、INSTEAD OF トリガーを使用できます。変更される行の妥当性検査がトリガー・コードによって実行され、有効であれば、基礎となる表に変更が伝播されます。

INSTEAD OF トリガーにより、OCI を使用してクライアント側のオブジェクト・ビューのインスタンスを変更することもできます。オブジェクト・ビューによって具象化されたオブジェクトをクライアント側のオブジェクト・キャッシュ内で修正し、それを永続的な格納領域にフラッシュ・バックするには、オブジェクト・ビューが変更可能でない限り、INSTEAD OF トリガーを指定する必要があります。ただし、オブジェクト・キャッシュ内のビュー・オブジェクトをピンして読み込むだけであれば、これらのトリガーを定義する必要はありません。

> **追加情報：** 詳細は、『Oracle8*i* 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 トリガーを使用せずにビューを挿入、更新または削除でき、かつビューが後述の条件に合致する場合、このビューは「本質的に変更可能」です。ビュー問合せに次の言語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、更新または削除を実行できません。

- 集合演算子
- 集計関数
- GROUP BY、CONNECT BY または START WITH 句
- DISTINCT 演算子
- 結合（ただし、結合ビューのサブセットは更新可能です。10-15 ページの「更新可能な結合ビュー」を参照してください。）

ビューに疑似列または式が含まれる場合、そのビューを更新するには、その疑似列または式を参照しない UPDATE 文を使用する方法しかありません。

INSTEAD OF トリガーの例

次の例は、すべての部門とそのマネージャをリストする `manager_info` ビューの各行を更新する INSTEAD OF トリガーを示しています。

`dept` は、部門リストを含むリレーショナル表であるとします。

```
CREATE TABLE dept (
   deptno NUMBER PRIMARY KEY,
   deptname VARCHAR2(20),
   manager_num NUMBER
);
```

`emp` は、従業員とその所属部門のリストを含むリレーショナル表であるとします。

```
CREATE TABLE emp (
   empno NUMBER PRIMARY KEY,
   empname VARCHAR2(20),
   deptno NUMBER REFERENCES dept(deptno),
   startdate DATE
);
ALTER TABLE dept ADD (FOREIGN KEY(manager_num) REFERENCES emp(empno));
```

各部門のすべてのマネージャをリストする `manager_info` ビューを作成します。

```
CREATE VIEW manager_info AS
  SELECT d.deptno, d.deptname, e.empno, e.empname
  FROM   emp e, dept d
  WHERE  e.empno =  d.manager_num;
```

次に、このビューへの挿入を処理する INSTEAD OF トリガーを定義します。manager_info ビューへの挿入を、dept 表の manager_num 列の更新に変換できます。

このトリガーでは、だれかを部門のマネージャにするには、その部門に最低 1 人は従業員が所属する必要があるという制約も施行できます。

```
CREATE TRIGGER manager_info_insert
  INSTEAD OF INSERT ON manager_info
    REFERENCING NEW AS n              -- new manager information
  FOR EACH ROW
  DECLARE
    empCount NUMBER;
  BEGIN

     /* First check to make sure that the number of employees
      * in the department is greater than one */
     SELECT COUNT(*) INTO empCount
      FROM emp e
      WHERE e.deptno = :n.deptno;

     /* If there are enough employees then make him or her the manager */
     IF empCount >= 1 THEN

        UPDATE dept d
         SET manager_num = :n.empno
         WHERE d.deptno = :n.deptno;

     END IF;
  END;
  /
```

manager_info ビューへの次のような挿入を考えます。

```
INSERT INTO manager_info VALUES (200,'Sports',1002,'Jack');
```

この文によって manager_info_insert トリガーが起動され、基礎となる表が更新されます。ビューに対する INSERT および DELETE の場合も、類似のトリガーを使用して適切なアクションを指定できます。

使用上の注意事項

CREATE TRIGGER 文の INSTEAD OF オプションを指定できるのは、ビューについて作成するトリガーの場合のみです。ビューについて作成するトリガーには、BEFORE および AFTER オプションは使用できません。

ビューに対する挿入または更新に INSTEAD OF トリガーが使用される場合、ビューの CHECK オプションは施行されません。このチェックは、INSTEAD OF トリガー本体で施行します。

追加情報： INSTEAD OF トリガーの詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』、および『Oracle8*i* SQL リファレンス』の CREATE TRIGGER コマンドを参照してください。

ネストした表に対する INSTEAD OF トリガー

ビュー内のネストした表列の要素は、TABLE 句で直接変更することはできません。ただし、ビューのネストした表列に INSTEAD OF トリガーを定義すれば、要素を変更できます。ネストした表のトリガーは、その表の要素が更新、挿入または削除すると起動し、基礎となる表に対する実際の変更を処理します。

ここでも、部門と従業員の例を考えます。部門ビューに、部門のリストと各部門の従業員の集合が含まれているものとします。次の例は、部門ビュー内で従業員オブジェクトのネストした表の要素を、INSTEAD OF トリガーを使用して変更する方法を示しています。

```
/* Create an employee type */
CREATE TYPE emp_t AS OBJECT
(
  empno NUMBER,
  empname VARCHAR2(20),
  days_worked NUMBER
);
/
/* Create a nested table type of employees */
CREATE TYPE emplist_t AS TABLE OF emp_t;
/
/* Now, create the department type */
CREATE TYPE dept_t AS OBJECT
(
  deptno NUMBER,
  deptname VARCHAR2(20),
  emplist  emplist_t
);
/
/* The dept_view can now be created based on the dept (department) and emp
 * (employee) tables. */
CREATE VIEW dept_view OF dept_t WITH OBJECT OID(deptno)
  AS SELECT d.deptno, d.deptname, -- department number and name
       CAST (MULTISET (
                SELECT e.empno, e.empname, (SYSDATE - e.startdate)
                FROM   emp e
                WHERE  e.deptno = d.deptno)
            AS emplist_t)       -- emplist - nested table of employees
     FROM dept d;
```

TABLE 構文を使用して、ビュー内でネストした表 emplist への挿入を可能にするには、

```
INSERT INTO TABLE
   (SELECT d.emplist FROM dept_view d WHERE d.deptno = 10)
   VALUES (10,'Harry Mart',334);
```

次のように、ネストした表 emplist に対して、「挿入を処理」する INSTEAD OF トリガーを定義できます。

```
CREATE TRIGGER dept_empinstr INSTEAD OF INSERT ON
   NESTED TABLE emplist OF dept_view FOR EACH ROW
BEGIN
    INSERT INTO emp VALUES(:NEW.empno, :NEW.empname,
                           :PARENT.deptno, SYSDATE - :NEW.days_worked);
END;
/
```

同様に、ネストした表の要素に対する更新と削除を処理するトリガーも定義できます。

ネストした表のトリガー内からの親行の属性へのアクセス 通常のトリガーでは、現在行の値には NEW 修飾子および OLD 修飾子を使用してアクセスできます。ビューのネストした表列のトリガーでは、これらの修飾子は変更するネストした表要素の属性を参照します。このネストした表列を含む親行の値にアクセスするには、PARENT 修飾子を使用できます。

この修飾子を使用できるのは、このようにネストした表のトリガー内からのみです。この PARENT 修飾子を使用して取得された親行の値は、変更できません（つまり、読込み専用です）。

ここでは、前述の *dept_empinstr* トリガーの例を考えます。NEW 修飾子は、ネストした表に挿入される行を参照し（つまり、empno、empname および days_worked）、従業員が所属する部門番号（deptno）は含めません。ただし、トリガー内で従業員表に部門番号を挿入する必要があります。この値は、PARENT 修飾子を使用して、従業員リストを含む親行から取得できます。

ネストした表とビューのトリガーの起動 前述のように、ビュー内のネストした表の列に INSTEAD OF トリガーが定義されている場合は、その表の要素が挿入、更新または削除されるときに、トリガーが起動されて実際の変更を行います。

この処理のために、ネストした表の列を含むビューに INSTEAD OF トリガーを定義する必要はありません。ビューに定義されたトリガーは、ネストした表の要素に対する変更時には起動しません。

逆に、ビュー内の行を変更する文では、そのビューのネストした表の列上のトリガーではなく、ビューに定義されたトリガーのみが起動します。たとえば、emplist ネストした表列が、次のように dept_view を通じて更新されるとします。

```
UPDATE dept_view SET emplist = emplist_t(emp_t(1001,'John',234));
```

この場合は、dept_empinstr ネストした表のトリガーではなく、dept_view に定義された INSTEAD OF 更新トリガーが起動します。

> **追加情報：** ネストした表の INSTEAD OF トリガーの詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

システム・イベントとユーザー・イベントのトリガー

トリガーを使用すると、データベース・イベントに関する情報を、サブスクライバに対して発行できます。アプリケーションは、他のアプリケーションからのメッセージにサブスクライブするのと同様に、データベース・イベントにサブスクライブできます。これらのデータベース・イベントは、次のとおりです。

- システム・イベント
  - データベースの起動と停止
  - サーバー・エラー・メッセージ・イベント
- ユーザー・イベント
  - ユーザーのログオンとログオフ
  - DDL 文（CREATE、ALTER および DROP）
  - DML 文（INSERT、DELETE および UPDATE）

システム・イベントのトリガーは、データベース・レベルまたはスキーマ・レベルで定義できます。たとえば、データベース停止トリガーは、データベース・レベルで次のように定義します。

```
CREATE TRIGGER register_shutdown
  ON DATABASE
  SHUTDOWN
    BEGIN
    ...
    DBMS_AQ.ENQUEUE(...);
    ...
    END;
```

DDL 文またはログオン・イベントおよびログオフ・イベントのトリガーも、データベース・レベルまたはスキーマ・レベルで定義できます。DML 文のトリガーは、表またはビューに定義できます。データベース・レベルで定義されたトリガーはすべてのユーザーに対して起動し、スキーマ・レベルまたは表レベルで定義されたトリガーは、トリガー・イベントがそのスキーマまたは表に関連する場合にのみ起動します。

イベントの発行

イベントの発行では、Oracle Advanced Queuing の発行およびサブスクライブ・メカニズムが使用されます（第 19 章「アドバンスト・キューイング」を参照）。「キュー」は、各種サブスクライバに必要な主題を含むメッセージ・リポジトリの役割を果たします。トリガーは、特定のシステム・イベントまたはユーザー・イベントの発生時に、DBMS_AQ パッケージを使用してメッセージをエンキューします。

イベントの属性

各イベントでは、トリガー・テキスト内で属性を使用できます。たとえば、データベース起動および停止トリガーは、インスタンス番号およびデータベース名の属性を持ち、ログオン・トリガーおよびログオフ・トリガーは、ユーザー名の属性を持ちます。属性をイベント発生時に発行する場合は、トリガーの作成時に、その属性と同じ名前のファンクションを指定できます。属性値は、トリガーの起動時にファンクションまたはペイロードに渡されます。DML 文のトリガーの場合、これは :NEW 列および :OLD 列の値を使用して実行されます。

システム・イベント

トリガーを起動できるシステム・イベントは、インスタンスの起動と停止およびエラー・メッセージに関連しています。起動および停止イベントに対して作成するトリガーは、データベースに対応付ける必要があります。エラー・イベントに対して作成するトリガーは、データベースまたはスキーマに対応付けることができます。

- STARTUP トリガーは、インスタンスによってデータベースがオープンされるときに起動する。この属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- SHUTDOWN トリガーは、サーバーがインスタンスの停止操作を開始する直前に起動する。このトリガーを使用して、データベースの停止時にサブスクライバ・アプリケーションを完全に停止できます。（インスタンスが異常停止する場合、このトリガーは起動しません。）SHUTDOWN トリガーの属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- SERVERERROR トリガーは、特定のエラーの発生時、または、エラー番号を指定しなければ任意のエラーの発生時に起動する。このトリガーの属性には、システム・イベントとエラー番号があります。

ユーザー・イベント

トリガーを起動できるユーザー・イベントは、ユーザー・ログオンとログオフ、DDL 文および DML 文に関連しています。

LOGON イベントおよび LOGOFF イベントのトリガー LOGON トリガーおよび LOGOFF トリガーは、データベースまたはスキーマに対応付けることができます。その属性には、システム・イベントとユーザー名があり、USERID と USERNAME について簡単な条件を指定できます。

- LOGON トリガーは、ユーザーのログオン成功後に起動する。
- LOGOFF トリガーは、ユーザー・ログオフの開始時に起動する。

DDL 文のトリガー DDL トリガーは、データベースまたはスキーマに対応付けることができます。その属性には、システム・イベント、スキーマ・オブジェクトのタイプおよび名前があります。ここでは、スキーマ・オブジェクトの型と名前だけでなく、USERID や USERNAME などの関数についても簡単な条件を指定できます。

- BEFORE CREATE トリガーおよび AFTER CREATE トリガーは、データベースまたはスキーマ内でスキーマ・オブジェクトが作成されるときに起動する。
- BEFORE ALTER トリガーおよび AFTER ALTER トリガーは、データベースまたはスキーマ内でスキーマ・オブジェクトが変更されるときに起動する。
- BEFORE DROP トリガーおよび AFTER DROP トリガーは、データベースまたはスキーマからスキーマ・オブジェクトが削除されるときに起動する。

DML 文のトリガー イベント発行用の DML トリガーは、表に対応付けられます。指定した DML 操作が発生する各行に対して起動するように、BEFORE トリガーまたは AFTER トリガーを使用できます（20-8 ページの「行トリガー」および 20-9 ページの「BEFORE トリガーと AFTER トリガー」を参照）。DML 文に関連するイベントの発行には、INSTEAD OF トリガーを使用できません。かわりに、INSTEAD OF トリガーによってビューの基礎となる表に生じる DML 操作について、BEFORE トリガーまたは AFTER トリガーを使用してイベントを発行できます。

イベント発行用の DML トリガーの属性には、システム・イベントと、SELECT リストにユーザーが定義する列があります。これにより、スキーマ・オブジェクトの型と名前だけでなく、関数（UID、USER、USERENV および SYSDATE）、疑似列および列についても、簡単な条件を指定できます。列には、接頭辞として新旧の値を示す :OLD と :NEW を使用できます。

- BEFORE INSERT および AFTER INSERT トリガーは、表に挿入される行ごとに起動する。
- BEFORE UPDATE および AFTER UPDATE トリガーは、表中で更新される行ごとに起動する。
- BEFORE DELETE および AFTER DELETE トリガーは、表から削除される行ごとに起動する。

> **追加情報：** システム・イベントとユーザー・イベントのトリガーを使用したイベント発行の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

トリガーの実行

トリガーには次の 2 つのモードがあります。

使用可能　使用可能にされているトリガーは、トリガー文が発行され、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、読込み一貫性ビューを提供する。
- 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・コミットを使用する。
- 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、BEFORE 文トリガー、AFTER 行トリガーおよび AFTER 文トリガーという、最大 4 つのトリガーを起動できます。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のチェックが実施されます。またトリガーには、他のトリガーを起動する文を含めることもできます（カスケード・トリガー）。

Oracle では、次の実行モデルを使用して、複数のトリガーと制約チェックの適切な起動順序を維持します。

1. その文に適用されるすべての BEFORE 文トリガーを実行します。
2. その SQL 文の影響を受ける各行をループします。
   a. その文に適用されるすべての BEFORE 行トリガーを実行します。
   b. 行をロックして変更し、整合性制約チェックを実施します。（トランザクションがコミットされるまでロックは解除されません）。
   c. その文に適用されるすべての AFTER 行トリガーを実行します。

3. 遅延整合性制約チェックを完了します。
4. その文に適用されるすべての AFTER 文トリガーを実行します。

実行モデルの定義は再帰的です。たとえば、ある SQL 文によって BEFORE 行トリガーを起動し、整合性制約をチェックできます。次に、この BEFORE 行トリガーが実行する更新によって、整合性制約をチェックし、AFTER 文トリガーを起動できます。この AFTER 文トリガーによって、整合性制約がチェックされます。この場合、実行モデルでは、次のステップが再帰的に繰り返し実行されます。

1. 元の SQL 文が発行されます。
2. BEFORE 行トリガーが起動されます。
3. BEFORE 行トリガー内の UPDATE 文により、AFTER 文トリガーが起動します。
4. AFTER 文トリガーの文が実行されます。
5. AFTER 文トリガーによって変更された表の整合性制約がチェックされます。
6. BEFORE 行トリガーの文が実行されます。
7. BEFORE 行トリガーによって変更された表の整合性制約がチェックされます。
8. SQL 文が実行されます。
9. SQL 文から整合性制約がチェックされます。

この再帰には、次の 2 つの例外があります。

- トリガー文が参照制約内で 1 つの表（主キーまたは外部キー表）を変更し、トリガーされた文が他方の表を変更すると、トリガー文のみが整合性制約をチェックする。これにより、行トリガーで参照整合性を強化できます。
- DELETE CASCADE および DELETE SET NULL によって起動されるトリガー文は、個々の施行文の前後ではなく、ユーザーの DELETE 文の前後に起動される。これにより、この種の文トリガーによる変更エラーの発生が防止されます。

実行モデルの重要なプロパティは、SQL 文の結果として実行されるアクションとチェックがすべて成功する必要があることです。トリガー内で発生した例外を明示的に処理できない場合、元の SQL 文によって実行されたすべてのアクションが、起動されたトリガーによって実行されたアクションを含めて、ロールバックされます。したがって、トリガーが整合性制約に違反することはありません。実行モデルは整合性制約を考慮し、宣言整合性制約に違反するトリガーを認めません。

たとえば、前述のステップ 1 ～ 8 を正しく実行し、ステップ 9 で整合性制約に違反があったとします。その違反の結果として、SQL 文によるすべての変更（ステップ 8）、起動された BEFORE 行トリガー（ステップ 6）および起動された AFTER 文トリガー（ステップ 4）がロールバックされます。

注意： 各種トリガーは特定の順序で起動されますが、同じ文に対する同じタイプのトリガーは、特定の順序で起動されるとは限りません。たとえば、1 つの UPDATE 文に対して定義されている BEFORE 行トリガーすべてが、毎回同じ順序で起動されるとは限りません。同じタイプのトリガーが複数ある場合は、その起動順序に依存しないようアプリケーションを設計してください。

トリガーのデータ・アクセス

トリガーが起動されると、トリガー・アクション内で参照される表は、他のユーザーのトランザクション内の SQL 文によって変更されている最中である可能性があります。トリガー内で実行される SQL 文は、常に単独の SQL 文と同じ規則に従います。起動されたトリガーが読込み（問合せ）または書込み（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する。
- 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、処理を続行する。

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 列に対する更新が含まれているとします。その更新の時点で、トランザクション T2 内の文によって SALARY_CHECK トリガーが起動されます。起動されたトリガー内の SELECT 文（T2 に由来する）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。そのトリガーの問合せは、トランザクション T2 の読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、部門内のメンバー全員の給与総額が格納されている導出列をメンテナンスするものとします。

```
CREATE TRIGGER total_salary
AFTER DELETE OR INSERT OR UPDATE OF deptno, sal ON emp
 FOR EACH ROW BEGIN
  /* assume that DEPTNO and SAL are non-null fields */
   IF DELETING OR (UPDATING AND :old.deptno != :new.deptno)
   THEN UPDATE dept
  SET total_sal = total_sal - :old.sal
```

```
  WHERE deptno = :old.deptno;
   END IF;
   IF INSERTING OR (UPDATING AND :old.deptno != :new.deptno)
   THEN UPDATE dept
    SET total_sal = total_sal + :new.sal
    WHERE deptno = :new.deptno;
   END IF;
   IF (UPDATING AND :old.deptno = :new.deptno AND
    :old.sal != :new.sal )
   THEN UPDATE dept
    SET total_sal = total_sal - :old.sal + :new.sal
   WHERE deptno = :new.deptno;
  END IF;
END;
```

この例で、コミットされていない第一のユーザーのトランザクションが、DEPT 表の 1 つの行の TOTAL_SAL 列に対する更新が含まれていたとします。その更新の時点で、第二のユーザーの SQL 文によって TOTAL_SALARY トリガーが起動されます。第一のユーザーの**コミットされていない**トランザクションに、TOTAL_SAL 列に含まれる関連する値の更新が含まれている（つまり行ロックがかけられている）ため、行ロックを保持しているトランザクションがコミットまたはロールバックされるまで、TOTAL_SALARY トリガーによる更新は実行されません。このため、第二のユーザーは、第一のユーザーのコミット・ポイントまたはロールバック・ポイントまで待機します。

PL/SQL トリガーの記憶領域

PL/SQL トリガーは、ストアド・プロシージャと同じように、コンパイル済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 コードがデータベースに格納され、トリガーのソース・コードは共有プールからフラッシュされます。

PL/SQL コードのコンパイルと格納の詳細は、18-16 ページの「Oracle がプロシージャとパッケージを格納する方法」を参照してください。

トリガーの実行

Oracle は、プロシージャの実行と同じステップを使用してトリガーを内部的に実行します。両者の唯一の違いは、ユーザーにトリガーを起動する権限が付与されるのは、ユーザーがトリガー文を実行する権限を持っている場合ということです。この点を除けば、トリガーはストアド・プロシージャと同じ方法でチェックされ、実行されます。

詳細は、18-17 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 21 章「Oracle の依存性の管理」を参照してください。

21

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*

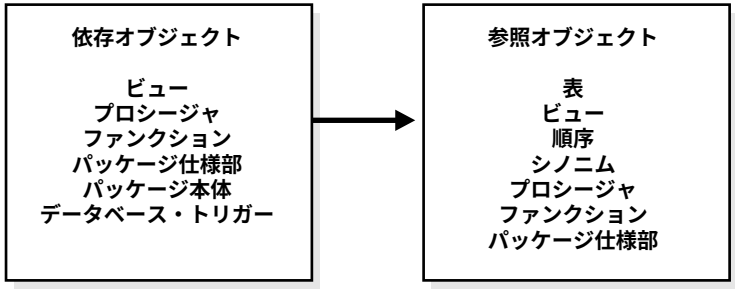
ビューやプロシージャなどのオブジェクトの定義では、他のオブジェクト（表など）が参照されます。したがって、定義するオブジェクトは、その定義で参照する他のオブジェクトに依存しています。この章では、スキーマ・オブジェクトの相互間の依存性と、Oracle がそのような依存性を自動的に追跡し管理する方法について説明します。この章の内容は次のとおりです。

- 依存性の問題の基礎知識
- 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
- ローカルおよびリモートの依存性の管理

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 文を組み込めます。定義の一部として他のオブジェクトを参照するオブジェクトを「依存」オブジェクトといい、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 21-1 に、各種の依存オブジェクトと参照オブジェクトを示します。

図 21-1 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用できません。

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（再度実行またはコンパイルされた）時点で自動的に再コンパイルされます。

スキーマ・オブジェクト間の依存性を管理するために、データベース内のすべてのスキーマ・オブジェクトは次のいずれかの状態になります。

VALID	スキーマ・オブジェクトはコンパイルされており、参照するとただちに使用できます。
INVALID	スキーマ・オブジェクトを使用するには、その前にコンパイルする必要があります。

- プロシージャ、ファンクションおよびパッケージの場合は、スキーマ・オブジェクトをコンパイルする必要があることを意味します。
- また、ビューの場合は、データ・ディクショナリ内の現在の定義を使用して、ビューを再解析する必要があることを意味します。

依存オブジェクトのみが INVALID になり得ます。つまり、表、順序およびシノニムは常に VALID です。

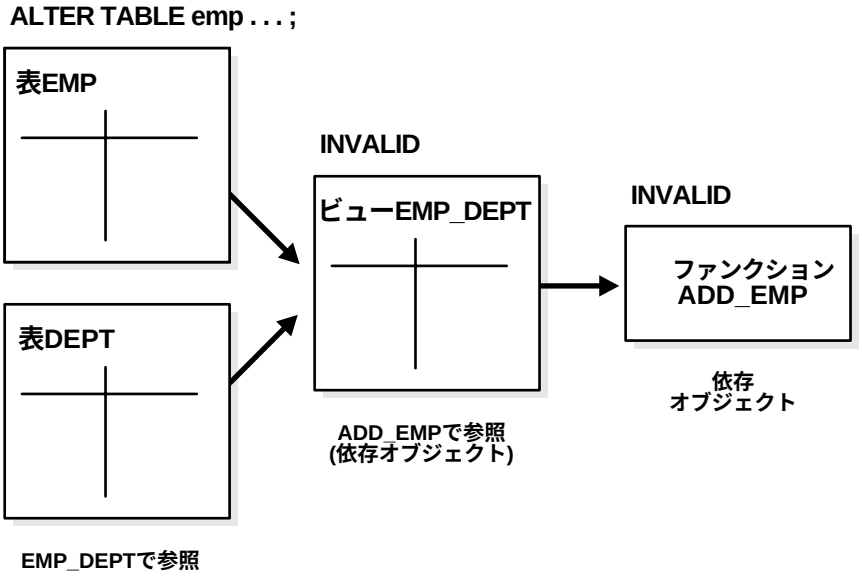
ビュー、プロシージャ、ファンクションまたはパッケージが INVALID である場合、Oracle がそのオブジェクトをコンパイルしようとしても、そのオブジェクトに関連したエラーが発生します。たとえば、ビューのコンパイル時に、実表の 1 つが存在しなかったり、実表の正当な権限がなかったりする場合があります。パッケージのコンパイル時には、PL/SQL または SQL の構文エラーが発生したり、参照されているオブジェクトについての適切な権限が存在しない場合があります。このような問題があるスキーマ・オブジェクトは INVALID のままです。

Oracle はデータベースの変更を自動的に追跡し、関係するオブジェクトの状態をデータ・ディクショナリに記録します。

状態の記録は再帰的な処理です。参照オブジェクトの状態が変更されると、直接的に依存するオブジェクトの状態が変更されるだけでなく、間接的に依存するオブジェクトの状態も変更されます。

たとえば、ビューを直接参照するストアド・プロシージャを考えます。ストアド・プロシージャは、このビューの実表を間接的に参照します。このため、実表を変更するとビューは無効になり、さらにストアド・プロシージャも無効になります。図 21-2 に、この状況を示します。

図 21-2 間接的な依存性



スキーマ・オブジェクトの依存性の解決

スキーマ・オブジェクトを（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）参照するときに、SQL 文に明示的に指定されているオブジェクトと参照オブジェクトの状態が、Oracle によって必要に応じてチェックされます。Oracle のアクションは、SQL 文で直接および間接的に参照されるオブジェクトの状態に応じて異なります。

- 参照オブジェクトがすべて有効であれば、追加的な処理なしで SQL 文が即時に実行される。
- 参照先のビューやプロシージャ（ファンクションやパッケージを含む）が無効な状態になっていると、Oracle はその参照オブジェクトを自動的にコンパイルしようとする。
  - 無効な参照オブジェクトは、そのすべてを正常にコンパイルできる場合には問題なくコンパイルされ、SQL 文は正常に実行される。
  - 無効なオブジェクトを正常にコンパイルできなければ、オブジェクトは無効のままになる。Oracle はエラーを戻し、その SQL 文を含むトランザクションをロールバックします。

注意： Oracle は、無効であると検出された後に置き換えられていないオブジェクトのみを動的に再コンパイルします。これにより、むだな再コンパイルが省略されます。

ビューと PL/SQL プログラム・ユニットのコンパイル

次の条件が満たされている場合は、ビューまたは PL/SQL プログラム・ユニットをコンパイルして、その状態を VALID にすることができます。

- ビューまたはプログラム・ユニットの定義が正しい。すべての SQL 文と PL/SQL 文が適切に構成されている必要があります。
- 参照オブジェクトが存在し、適切な構成である。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る必要があります。
- ビューまたはプログラム・ユニットの**所有者**に、参照オブジェクトに対する必要な権限が付与されている。たとえば、プロシージャ内の SQL 文によって表に行が挿入される場合、プロシージャの所有者にはその参照表に対する INSERT 権限が必要です。

ビューと実表

ビューは、そのビューの定義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ければ、定義問合せはデータ・ディクショナリへの格納時に展開され、参照先の実表内の現行のすべての列が含まれるようになります。

ビューの実表（またはビュー）を変更、改名または削除すると、そのビューは無効になりますが、その定義は無効なビューを参照する権限、シノニム、他のオブジェクトおよび他のビューとともに、データ・ディクショナリ内に残ります。

無効（INVALID）なビューを使用すると、そのビューは動的に再コンパイルされます。再コンパイルされたビューは、次に示す条件に応じて有効または無効になります。

- ビューの定義問合せが参照しているすべての実表が存在している必要がある。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、使用できなくなります。無効なビューを参照すると、その参照文は失敗します。ビューをコンパイルできるのは、実表を元の名前に改名した場合、または実表を再作成した場合だけです。
- 実表を変更したり、同じ列を含む実表を再作成した場合に、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る。
- ビューの実表が、少なくとも同じ列セットを含むように変更または再作成された場合、そのビューは有効にすることができる。実表が新しい列を含むように再作成され、その結果の表中にもはや存在しない列をビューが参照していると、そのビューは有効になりません。後者のケースは、`SELECT * FROM table` の問合せで定義されたビューに特に当てはまります。つまり、このような定義問合せはビューの作成時にデータ・ディクショナリ内で展開され、そのまま永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は影響しない。ただし、参照パッケージの「仕様部」を置き換えると、依存プロシージャは無効になります。

この最後の条件は、パッケージを使用することによってプロシージャおよび参照オブジェクトの間の依存性を最小にするメカニズムを示しています。

セッションの状態と参照パッケージ

パッケージ構成体を参照するセッションごとに、各パッケージの独自のインスタンスが存在します。このインスタンスには、パブリック変数とプライベート変数、カーソルおよび定数の永続的な状態が含まれます。その後、セッションのインスタンス化されたパッケージ（仕様部または本体）のいずれかが無効にされて再コンパイルされると、セッションのパッケージのすべてのインスタンシエーション（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引き続き持っていることを検証します。
Oracle は内部的に、そのようなオブジェクトを「再コンパイル」する必要はないことを確認します。つまり、オブジェクト構造を検査するのではなく、セキュリティ認可のみを検査します。このように最適化すれば、不要な再コンパイルが省略され、依存オブジェクトのタイムスタンプを変更する必要もありません。

> **追加情報：** 無効なビューとプログラム・ユニットを強制的に再コンパイルする方法の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

ファンクションベース索引の依存性

ファンクションベース索引は、その索引を定義している式に使用されたファンクションに依存しています。（10-24 ページの「ファンクションベース索引」を参照）。このようなファンクション（PL/SQL 関数またはパッケージ・ファンクション）に変更があると、索引に使用禁止マークが設定されます。

ここでは、ファンクションベース索引の要件と、削除される時期や使用権限が取り消される時期など、なんらかの方法でファンクションが変更された場合の操作について説明します。

要件

ファンクションベース索引を作成する手順

- 次の初期化パラメータを定義する必要がある。
  - QUERY_REWRITE_INTEGRITY を TRUSTED に設定する。
  - QUERY_REWRITE_ENABLED を TRUE に設定する。
  - COMPATIBLE を 8.1.0.0.0 以上の値に設定する。
- ユーザーには、CREATE INDEX と QUERY REWRITE、または CREATE ANY INDEX と GLOBAL QUERY REWRITE を付与する必要がある。

ファンクションベース索引を使用する手順

- 表は、索引の作成後に分析する必要がある。
- NULL 値は索引に格納されないため、問合せには索引付きの式からの NULL 値を必要としないことを保証する必要がある。

この後の各項では、追加の要件について説明します。

DETERMINISTIC 関数

ファンクションベース索引に使用するためにユーザーが記述する関数は、現在も将来も入力引数値の集合に対して常に同じ出力戻り値を戻すことを示すために、DETERMINISTIC キーワードで宣言する必要があります。詳細は、23-9 ページの「DETERMINISTIC 関数」を参照してください。

定義する関数に対する権限

索引所有者は、ファンクションベース索引の定義に使用する関数に対して EXECUTE 権限を持つ必要があります。EXECUTE 権限が取り消されると、ファンクションベース索引に DISABLED マークが設定されます。索引所有者は、この関数に対する EXECUTE WITH GRANT OPTION 権限がなくても、基礎となる表の SELECT 権限を付与できます。

ファンクションベース索引の依存性の解決

ファンクションベース索引は、それを使用するファンクションに依存します。ファンクション、またはそのファンクションを含むパッケージの仕様部が再定義されると（または、索引所有者の EXECUTE 権限が取り消されると）、その索引には DISABLED マークが設定されます。

- DISABLED 索引の問合せは、オプティマイザが索引を使用するために選択すると、失敗する。
- DISABLED 索引の DML 操作は、索引にさらに UNUSABLE マークが設定され、初期化パラメータ SKIP_UNUSABLE_INDEXES が TRUE に設定されていなければ、失敗する。

ファンクションの変更後に索引を再使用可能にするには、ALTER INDEX ... ENABLE 文を使用します。

依存性の管理と存在しないスキーマ・オブジェクト

依存オブジェクトが作成されると、Oracle は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用している参照が、別のオブジェクトが存在する場合に異なって解釈される可能性がある場合に発生します。たとえば、次のような場合です。

- 現時点では、COMPANY スキーマには EMP という名前の表が含まれている。
- COMPANY.EMP のために EMP という名前の PUBLIC シノニムが作成されており、COMPANY.EMP に対する SELECT 権限が PUBLIC ロールに付与されている。
- 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、次の文を使用して自分のスキーマ内にビューを作成する。

```
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 ビューを作成する際、EMP への参照を解決するために、表、ビューおよびプライベート・シノニムとしての JWARD.EMP が最初に検索されます。いずれも見つからないため、次に EMP という名前のパブリック・シノニムとして検索され、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、次の文を使用して EMP という名前の新しいビューを自分のスキーマ内に作成するとします。

```
CREATE VIEW emp AS
   SELECT empno, ename, mgr, deptno
   FROM company.emp;
```

JWARD.EMP と COMPANY.EMP の列構成が異なることに注意してください。

オブジェクト定義に含まれる参照を解決しようとするときに、新しい依存オブジェクトが「存在しないはず」のオブジェクトに依存していることが Oracle 内部で考慮されます。存在しないはずのオブジェクトとは、そのオブジェクトが存在していると、新しく作成するオブジェクトの定義の解釈が変わってしまうスキーマ・オブジェクトのことです。存在しないはずのオブジェクトが後に作成された場合に備えるため、そのような依存性があるということを覚えておく必要があるので注意してください。存在しないはずのオブジェクトが実際に作成された場合、その依存オブジェクトを再コンパイルして有効にするには、すべての依存オブジェクトを無効にして、すべての依存ファンクションベース索引に UNUSABLE マークを設定する必要があります。

したがって、前述の例では、JWARD.EMP が作成されると、JWARD.DEPT_SALARIES は JWARD.EMP に依存しているため無効になります。JWARD.DEPT_SALARIES が使用されると、Oracle はビューを再コンパイルします。EMP の参照は解決され、JWARD.EMP が見つかります（PUBLIC.EMP は参照オブジェクトではなくなります）。ただし、JWARD.EMP には SAL 列が含まれないため、ビューを置き換える際にエラーが発生し、無効のままになります。

要約すると、オブジェクトを解決する際に存在しないはずのオブジェクトが見つかった場合、存在しないはずのオブジェクトが後で実際に作成される場合に備えて、そのオブジェクトに対する依存性を管理する必要があります。

共有 SQL の依存性管理

スキーマ・オブジェクト相互間の依存性管理に加えて、Oracle は共有プール内の各共有 SQL 領域の依存性も管理します。表またはビュー、シノニム、順序を生成、変更または削除したり、プロシージャまたはパッケージの仕様部を再コンパイルすると、それらに依存する共有 SQL 領域もすべて無効になります。共有 SQL 領域が無効になった後で、その領域に対応するカーソルを実行すると、Oracle によって SQL 文が再解析され、共有 SQL 領域が再生成されます。

ローカルおよびリモートの依存性の管理

依存性の追跡や必要な再コンパイルは、Oracle によって自動的に実行されます。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・ビューの定義問合せがリモート表を参照する場合があります。

ローカル依存性の管理

Oracle は、データベース内部の「依存」表を使用してローカル依存性を管理します。この表は、スキーマ・オブジェクトごとに依存オブジェクトを追跡し、記録するものです。参照オブジェクトが修正されると、この依存表を使用して依存オブジェクトが識別され、それらの依存オブジェクトが無効にされます。たとえば、ストアド・プロシージャ UPDATE_SAL が表 JWARD.EMP を参照している場合を考えます。この表の定義になんらかの変更があると、ストアド・プロシージャ UPDATE_SAL を含めて、JWARD.EMP を参照するすべてのオブジェクトの状態が INVALID に変更されます。そのため、このプロシージャは、再コンパイルされて有効にされるまで実行できません。同様に、ユーザーの DML 権限を取り消すと、そのユーザーのスキーマ内の依存オブジェクトはすべて無効になります。ただし、認可が取り消されたために無効になったオブジェクトは、「再認可」によって有効にできます。その場合は、完全に再コンパイルする必要はありません。

リモート依存性の管理

アプリケーションからデータベースへの依存性、および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する必要があります。たとえば、Oracle Forms アプリケーションには表を参照するトリガーが含まれていたり、ローカルなストアド・プロシージャから分散データベース・システム内のリモート・プロシージャがコールされることがあります。データベース・システムでは、このようなオブジェクト間の依存性を管理する必要があります。関係するオブジェクトの種類によっては、リモート依存性を管理するために別のメカニズムが使用されます。

ローカルおよびリモートのデータベース・プロシージャ間の依存性

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、パッケージおよびトリガーなど）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」を使用して管理します。

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のリモート依存性を管理する方法の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

タイムスタンプのチェック タイムスタンプ・チェック依存性モデルでは、プロシージャがコンパイルまたは再コンパイルされるたびに、その「タイムスタンプ」（作成、変更または置換の時刻）がデータ・ディクショナリに記録されます。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用されると、Oracle は、コンパイル時に記録されたリモート・タイムスタンプと、リモートで参照されたプロシージャの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ると、ローカル・プロシージャとリモート・プロシージャはコンパイルされずに実行される。
- リモートで参照されたプロシージャのいずれかの比較結果が一致しなければ、ローカル・プロシージャは無効となり、コール元の環境にエラーが戻される。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・プロシージャ本体内の文がリモート・プロシージャを実行する時点で実行されます。この時点では、分散データベースの通信リンクを介してタイムスタンプの比較のみが実行されます。したがって、ローカル・プロシージャの文のうち無効なプロシージャ・コールより前にあるすべての文は、正常に実行される可能性があります。無効なプロシージャ・コールより後の文はまったく実行されません（コンパイルが必要）。ただし、無効なプロシージャ・コールより前に実行された DML 文があると、すべてロールバックされます。

署名のチェック Oracle には「署名」を使用する「リモート依存性」の機能もあります。この署名機能の影響を受けるのは、リモート依存性のみです。ローカル（同じサーバー）依存性は、その環境でいつでも再コンパイルできるため影響を受けません。

プロシージャの署名には、次の情報が含まれます。

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

注意： 重要なのはパラメータの型とモードのみです。パラメータの名前は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おり、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・ファンクションまたはトリガー）への依存性により、その依存単位が無効になります。

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を除いては、リモート・スキーマ・オブジェクト間の依存性は管理されません。

たとえば、リモート表を参照する問合せによって、ローカル・ビューを作成して定義するとします。また、ローカル・プロシージャに、同じリモート表を参照する SQL 文が含まれているとします。その後、表の定義が変更されたとします。

表の変更後にビューとプロシージャが使用され、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、プリコンパイラおよび SQL*Module アプリケーションは、無名 PL/SQL ブロックを送信できます。また Oracle Forms アプリケーションのトリガーは、スキーマ・オブジェクトを参照できます。

このようなアプリケーションは、参照するスキーマ・オブジェクトに依存しています。依存性管理の方法は、開発環境によって異なります。データベース・アプリケーション内でリモート依存性を管理する方法の詳細は、アプリケーション開発ツール固有およびオペレーティング・システム固有の該当するマニュアルを参照してください。

第 VI 部

SQL 文の最適化

第 VI 部では、各 SQL 文を実行するための最も効率的な方法を選択するオプティマイザについて説明します。

第 VI 部に含まれる章は、次のとおりです。

22

オプティマイザ

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、Oracle オプティマイザの概要を説明します。この章の内容は次のとおりです。

- 最適化とは何か
- コストベース最適化
- 拡張可能な最適化
- ルールベース最適化

Oracle オプティマイザの機能の詳細は、次の各章を参照してください。

- 第 23 章「オプティマイザの操作」
- 第 24 章「結合の最適化」

追加情報： クエリー・リライトにマテリアライズド・ビューを使用する方法など、オプティマイザの詳細は、『Oracle8*i* チューニング』を参照してください。

最適化とは何か

最適化とは、SQL 文を実行するのに最も効率的な方法を選択する処理です。この処理は、SELECT、INSERT、UPDATE または DELETE など、データ操作言語（DML）文の処理において重要なステップです。SQL 文を実行する場合、表や索引にアクセスする順序などによって、実行方法がさまざまになることがあります。文を実行するときに使用する手順は、文の実行速度に大きく影響します。

Oracle のうち「オプティマイザ」と呼ばれる部分では、SQL 文の最も効率的な実行方法が計算されます。オプティマイザは、多数の要因を評価して代替アクセス・パスを選択します。また、コストベースまたはルールベースのアプローチを使用できます（22-8 ページの「コストベース最適化」および 22-18 ページの「ルールベース最適化」を参照）。

注意： オプティマイザは、Oracle のバージョンが変わると、同じ決定をしない可能性があります。最近のバージョンでは、さらに改善され洗練された情報に基づいて実行方法が決定されるようになってきています。

オプティマイザのアプローチと目標を設定し、コストベース最適化の統計を収集すると、オプティマイズによる選択を調整できます。特定のアプリケーションのデータについて、オプティマイザよりもさらに詳細な知識を持つアプリケーション・デザイナのほうが、SQL 文をより効率的に実行する方法を選択できることもあります。アプリケーション・デザイナは、SQL 文内にヒントを使用して文の実行方法を指定できます。

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

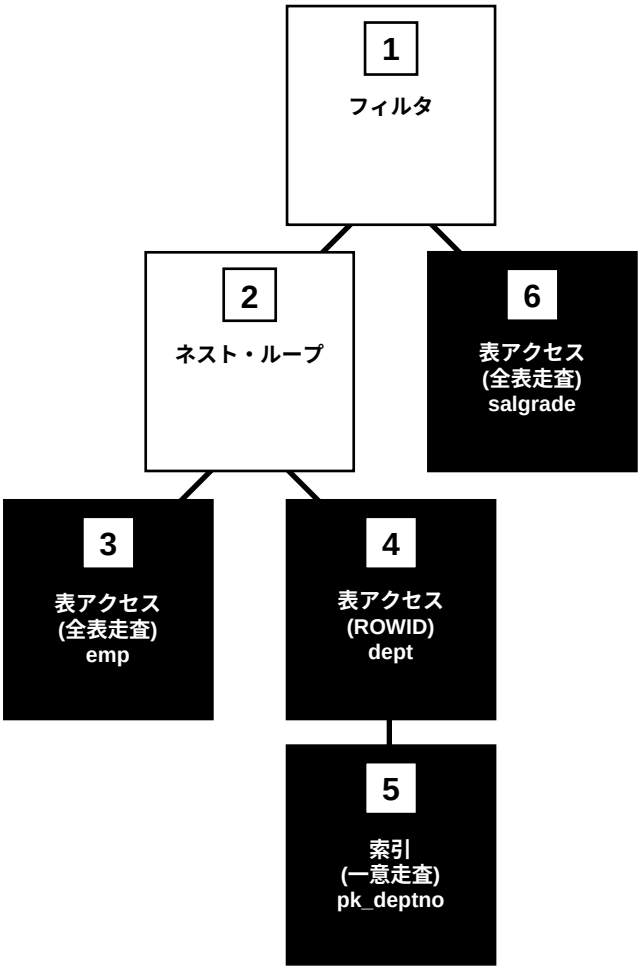
実行計画

DML 文を実行するために、Oracle では数多くのステップを必要とする場合があります。それらの各ステップでは、データベースからデータ行を物理的に検索するか、文を発行したユーザーのためになんらかの方法でデータ行を準備します。Oracle が文の実行に使用するステップの組合せを、「実行計画」といいます。実行計画には、文がアクセスする表ごとの「アクセス方法」と表の順序付け（「結合順序」）が含まれています。索引、ハッシュ・クラスタおよび表走査など、各種アクセス方法の説明は、23-32 ページの「アクセス方法」を参照してください。

図 22-1 に、次の SQL 文の実行計画を示します。この文は、給与が基準範囲内にないすべての従業員の名前（ename）、職種（job）、給与（sal）および部門名（dname）を選択します。

```
SELECT ename, job, sal, dname
    FROM emp, dept
  WHERE emp.deptno = dept.deptno
    AND NOT EXISTS
      (SELECT *
           FROM salgrade
         WHERE emp.sal BETWEEN losal AND hisal);
```

図 22-1 実行計画



実行計画のステップ

実行計画の各ステップでは、次のステップで使用さ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 22-1 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（22-6 ページの「実行の順序」を参照）。実行計画の各ステップでは、入力として、データベースから行を検索するか、1 つ以上の行ソースから行を受け取ります。

- 黒のボックスで示されているステップでは、データベースのオブジェクトから物理的にデータを取り出す。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 では、それぞれ EMP 表と SALGRADE 表のすべての行を読み込む。
  - ステップ 5 では、ステップ 3 で戻された各 DEPTNO 値を、PK_DEPTNO 索引の中から検索する。次に、DEPT 表内の対応する行の ROWID を検索します。
  - ステップ 4 では、ステップ 5 で戻された ROWID が入っている行を DEPT 表から取り出す。
- 白のボックスで示されているステップでは、行ソースを操作する。
  - ステップ 2 では、ネスト・ループ操作を実行する。ステップ 3 と 4 から行ソースを受け取り、ステップ 3 のソースから受け取った各行をステップ 4 の対応する行と結合し、結果の行をステップ 1 に戻します。
  - ステップ 1 では、フィルタ操作を実行する。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、23-32 ページの「アクセス・パスの選択」を参照してください。Oracle で行ソースを結合する方法の詳細は、24-2 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 コマンドを使用すると、オプティマイザが選択する SQL 文の実行計画を調べることができます。このコマンドは、オプティマイザに実行計画を選択させ、その計画を記述したデータをデータベース表に挿入します。

たとえば、前項に示した文の記述の出力表は、次のようになります。

```
ID    OPERATION           OPTIONS        OBJECT_NAME
-----------------------------------------------------------
0     SELECT STATEMENT
1       FILTER
2         NESTED LOOPS
3           TABLE ACCESS  FULL           EMP
4           TABLE ACCESS  BY ROWID       DEPT
5             INDEX       UNIQUE SCAN    PK_DEPTNO
6           TABLE ACCESS  FULL           SALGRADE
```

図 22-1 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 22-1 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 コマンドを使用した後、その出力表に問い合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*i* 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 22-1 のステップ 3、5 および 6）を最初に実行します。それぞれのステップから戻された行が、その親ステップの行ソースになります。この後、親ステップを実行します。

たとえば、図 22-1 の文を実行する場合、Oracle は次の順序でステップを実行します。

- ステップ 3 を実行し、結果の行を 1 つずつステップ 2 に戻す。
- ステップ 3 から戻されるそれぞれの行について、次のステップを実行する。
  - ステップ 5 を実行し、結果の ROWID をステップ 4 に戻す。
  - ステップ 4 を実行し、結果の行をステップ 2 に戻す。
  - ステップ 2 を実行する。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。
  - ステップ 6 を実行し、結果の行があればステップ 1 に戻す。
  - ステップ 1 を実行する。ステップ 6 から行が戻されなければ、SQL 文を発行したユーザーにステップ 2 から受け取った行を戻します。

ステップ 3 で戻された行ごとに、ステップ 5、4、2、6 および 1 が 1 回ずつ実行されるので注意してください。親ステップが、実行前に必要とするのが子ステップからの 1 行のみであれば、Oracle は子ステップから 1 行が戻されると、ただちに親ステップ（および実行計画の残りの部分）を実行します。行が 1 つ戻されると親ステップの親をアクティブにできる場合は、その親も実行されます。

このように、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くことができます。Oracle は、子ステップで順次それぞれの行が取り出されるたびに、親ステップとすべてのカスケード・ステップを 1 回実行します。子ステップから各行が戻されるたびに起動される親ステップには、表アクセス、索引アクセス、ネスト・ループ・ジョインおよびフィルタがあります。

子ステップからの行がすべてないと親ステップを実行できない場合、すべての行が子ステップから戻されるまで、親ステップを実行できません。このような親ステップとしては、ソート、ソート・マージ結合および集計関数があります。

オプティマイザのプラン・スタビリティ

アプリケーションを慎重にチューニングした後で、同じ SQL 文が実行されると常に同じ実行計画がオプティマイザで生成されるかどうかを確認する必要があります。「プラン・スタビリティ」により、表の再分析、データの追加や削除、表の列、定数、索引の変更またはシステム構成の変更、さらにオプティマイザの新バージョンへのアップグレードなど、データベースに対する変更に関係なく、同じ SQL 文について同じ実行計画を維持できます。

CREATE OUTLINE は、オプティマイザで実行計画の作成に使用される属性の集合を含む、「ストアド・アウトライン」を作成します。ストアド・アウトラインは、システム・パラメータ CREATE_STORED_OUTLINES を TRUE に設定して自動作成する方法もあります。

システム・パラメータ USE_STORED_OUTLINES は、TRUE、FALSE またはカテゴリ名に設定し、実行する問合せに既存のストアド・アウトラインを使用するかどうかを指定できます。OUTLN_PKG パッケージは、ストアド・アウトラインの管理に使用するプロシージャを提供します。

プラン・スタビリティのインプリメントによって、DBA 権限を持つ新しいスキーマ OUTLN が作成されます。データベース管理者は、SYS および SYSTEM スキーマの場合と同様に、OUTLN スキーマのパスワードを変更する必要があります。

> **追加情報：** プラン・スタビリティの使用方法は『Oracle8*i* チューニング』、CREATE OUTLINE 文の詳細は『Oracle8*i* SQL リファレンス』および OUTLN_PKG パッケージの詳細は『Oracle8*i* パッケージ・プロシージャ リファレンス』を参照してください。

コストベース最適化

コストベースのアプローチを使用すると、オプティマイザは使用可能なアクセス・パスと、SQL 文がアクセスするスキーマ・オブジェクト（表または索引）に関する統計情報の要素を考慮し、最も効率的な実行計画を判断します。コストベースのアプローチでは、ヒント、つまり文のコメントとして示されている最適化に関する提案も考慮に入れられます。

コストベースのアプローチは、概念的に次のようなステップで構成されています。

1. オプティマイザは、使用可能なアクセス・パスとヒントに基づいて、SQL 文の実行計画の集合を生成します。
2. オプティマイザは、文がアクセスする表、索引およびパーティションのデータ分布および記憶特性に関するデータ・ディクショナリ内の統計に基づいて、各実行プランのコストを見積ります。

   「コスト」は、特定の実行計画を使用して文を実行するために必要な、予想されるリソースの使用量に比例する推定値です。オプティマイザは、計画を使用して文を実行するために必要な I/O、CPU 時間およびメモリーなど、コンピュータ・リソースの推定に基づいて、考えられる各アクセス方法と結合順序のコストを計算します。

   コストが高いシリアル実行計画には、コストが低い実行計画よりも実行に時間がかかります。ただし、パラレル実行計画を使用する場合、リソースの使用状況と経過時間の間に直接的な関係はありません。
3. オプティマイザは、複数の実行計画のコストを比較して、コストが最も低い実行計画を選択します。

コストベース・アプローチの目標

コストベースのアプローチの目標は、デフォルトでは、最大の「スループット」を達成すること、つまり文がアクセスするすべての行を処理するために必要なリソース使用量を最小にすることです。

また、Oracle では、「応答時間」を最短に、つまり SQL 文がアクセスする最初の行を処理するのに必要なリソース使用量を最小に抑えることを目標にして、文を最適化することもできます。オプ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、23-30 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。

SQL 文の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。オプティマイザで実行のパラレル化をどの程度試行するかを指定するには、初期化パラメータ OPTIMIZER_PERCENT_PARALLEL を使用します。

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

コストベース最適化の統計

コストベースのアプローチでは、統計を使用して述語の選択性を計算し、各実行計画のコストを推定します。「選択性」は、SQL 文の述語によって表内で選択される行の割合です。オプティマイザは、述語の選択性を使用して、特定のアクセス方法のコストを推定し、最適の結合順序を判断します。

統計データは、表、列、索引およびパーティションのデータ分布と記憶特性を数量化したものです。オプティマイザは、特定の実行計画を使用して SQL 文を実行する場合に、どの程度の I/O および CPU 時間、メモリーが必要になるかを、この統計情報を使用して推定します。統計はデータ・ディクショナリに格納されており、あるデータベースからエクスポートして別のデータベースにインポートできます（たとえば、テスト・システムにデータのサンプルが少数しかなくても、本番の統計をテスト・システムに転送して現実の環境をシミュレーションできます）。

オプティマイザにスキーマ・オブジェクト関連の情報を提供するには、統計を定期的に収集する必要があります。スキーマ・オブジェクトのデータや構造が変更されたために、前の統計が不正確になる場合は、新しい統計を収集してください。たとえば、きわめて多数の行を表にロードした後は、行数に関して新しい統計を収集します。表内のデータを更新した後は、行数について新しい統計を収集する必要はありませんが、行の平均の長さについての新しい統計が必要になる場合があります。22-12 ページの「統計収集」を参照してください。

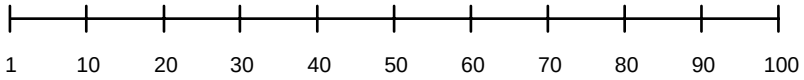
コストベース最適化のヒストグラム

コストベース最適化では、データ値ヒストグラムを使用して列データの分布を正確に推定します。「ヒストグラム」は、帯に含まれるすべての列値が同じ範囲に含まれるように、列値を帯状にパーティション化したものです。ヒストグラムを使用すると、データの偏りが存在する場合の選択性の推定を改善でき、その結果、不均一なデータ分布がある場合でも最適の実行計画が作成されます。

コストベース最適化の基本的な機能の 1 つは、問合せに含まれている述語の選択性を決定することです。選択性の推定は、索引の使用時期を決定するときと、表を結合する順序を決定するときに使用します。ほとんどの属性ドメイン（表の列）は、均一に分布していません。

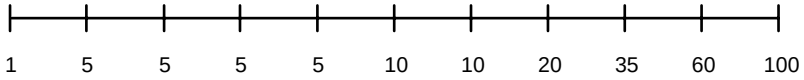
コストベース最適化は、不均一なドメインの分布を示すために、指定した属性に対して高さのバランスを調整したヒストグラムを使用します。高さ調整済ヒストグラムでは、列値は各帯にほぼ同数の値が含まれるように、複数の帯に分割されます。その後で、ヒストグラムから得られた各終点が入る値の範囲を有効な情報として活用します。

値が 1 ～ 100 までの列 C と、10 個の枠（バケット）からなるヒストグラムがあるとします。C のデータが均一に分布している場合、このヒストグラムは次のようになります。数値は終点の値です。



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。

高さ調整済ヒストグラム Oracle では、（「幅調整済み」に対して）「高さ調整済」ヒストグラムが使用されます

- 幅調整済ヒストグラムでは、それぞれの値の範囲が同じ幅になるように、データを特定の数に分けてから、それぞれの範囲内にある値の数をカウントします。
- 高さ調整済ヒストグラムでは、各範囲にほぼ同数の値が入るようにし、その範囲に入っている値の数によって範囲の終点が決められます。

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成するとします。幅調整済ヒストグラムでは、すべてのバケットが同じ幅（1-10、11-20、21-30、など）になり、各バケットはそのバケットの範囲内にある行数をカウントします。高さ調整済ヒストグラムでは、各バケットは同じ高さであり（この場合は 100 行）、列内の個々の値の密度に基づいて各バケットの終点が決まります。

高さのバランスを調整するアプローチは、データが非常に不整であるときに威力を発揮します。1000 行を含む表のうち 800 行に値 5 が入っており、残りの 200 行は 1 ～ 100 に均一に分布しているとします。幅調整済ヒストグラムでは、バケット 1 ～ 10 に 820 行が入り、他の各バケットに約 20 行ずつ入ります。高さベースのヒストグラムの場合、1 ～ 5 のバケットが 1 つ、5 ～ 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、高さ調整済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。ただし、幅調整済ヒストグラムの場合、5 の値と 6 の値を区別するための手段がありません。このヒストグラムで計算した場合、値が 5 の行は全体の 8% の行のみになります。したがって、列値の選択性を判断するには、高さ調整済みヒストグラムのほうが適しています。

ヒストグラムを使用する場合 パフォーマンスに影響を与える可能性があるので、ヒストグラムを使用するのは、問合せ計画を実際に改善できる場合のみにしてください。通常は、問合せの WHERE 句での使用頻度が高く、非常に不整なデータ分布を持つ列について、ヒストグラムを作成する必要があります。多くのアプリケーションの場合、一般に索引列は WHERE 句で最も使用頻度の高い列であるため、適切な方法はすべての索引列についてヒストグラムを作成することです。

ヒストグラムは永続オブジェクトであるため、使用するとメンテナンスと領域に関するコストが生じます。ヒストグラムは、データ分布が非常に不整であることがわかっている列についてのみ計算してください。均一に分布するデータの場合、コストベース最適化では、ヒストグラムを使用しなくても、特定の文の実行コストを正確に推定できます。

ヒストグラムは、他のすべてのオプティマイザ統計と同様に静的です。ヒストグラムが有用なのは、ある列の現在のデータ分布がヒストグラムに反映される場合のみです。（データの「分布」が一定であれば、データを変更できます。）列のデータ分布が頻繁に変化する場合は、そのヒストグラムを頻繁に再計算する必要があります。

列に次のような特性がある場合、ヒストグラムは役立ちません。

- その列のすべての述語でバインド変数を使用している。
- その列のデータが均一に分布している。
- 問合せの WHERE 句でその列が使用されていない。
- その列が一意で、等価述語だけで使用されている。

ヒストグラムを生成するには、DBMS_STATS パッケージまたは ANALYZE コマンドを使用します（22-12 ページの「統計収集」を参照）。表またはパーティションの列のヒストグラムを生成できます。ヒストグラムの統計は、パラレルには収集されません。

ヒストグラム情報は、次のデータ・ディクショナリ・ビューを使用して表示できます。

- USER_HISTOGRAMS、ALL_HISTOGRAMS および DBA_HISTOGRAMS
- USER_PART_HISTOGRAMS、ALL_PART_HISTOGRAMS および DBA_PART_HISTOGRAMS
- USER_SUBPART_HISTOGRAMS、ALL_SUBPART_HISTOGRAMS および DBA_SUBPART_HISTOGRAMS
- USER_TAB_COLUMNS、ALL_TAB_COLUMNS および DBA_TAB_COLUMNS

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

パーティション化されたスキーマ・オブジェクトの統計

パーティション化されたスキーマ・オブジェクトは、統計の集合を複数含むことができます。つまり、スキーマ・オブジェクト全体を参照する統計（グローバル統計）、個々のパーティションを参照する統計、およびコンポジット・パーティション・オブジェクトの個々のサブパーティションを参照する統計を含むことができます。

問合せの述語によって対象を単一パーティションに限定しなければ、オプティマイザはグローバル統計を使用します。ほとんどの問合せはこのように限定的ではないので、正確なグローバル統計を生成することが最も重要になります。パーティション・レベルの統計からグローバル統計を生成するのは簡単なように思われますが、これは一部の統計にしか当てはまりません。たとえば、各パーティション内の各値の数から 1 列の個々の値の数を検出することは、値がオーバーラップしている可能性があるためきわめて困難です。したがって、実際にグローバル統計を収集するときには、ANALYZE コマンドを使用して計算するのではなく（22-13 ページの「ANALYZE コマンド」を参照）、DBMS_STATS パッケージを使用することをお薦めします。

注意： 現在、Oracle はグローバル・ヒストグラム統計を収集しません。

統計収集

この項では、さまざまな統計収集方法について説明します。

DBMS_STATS パッケージ PL/SQL パッケージ DBMS_STATS を使用すると、コストベース最適化のための統計を生成して管理できます。このパッケージでは、統計を収集、変更、表示および削除できます。また、このパッケージを使用して統計の集合を格納することも可能です（22-14 ページの「統計表」を参照）。

DBMS_STATS パッケージでは、索引、表、列およびパーティションの統計や、スキーマまたはオブジェクト内のすべてのスキーマ・オブジェクトの統計を収集できます。クラスタ統計は収集されませんが、DBMS_STATS を使用するとクラスタ全体ではなく個々の表の統計を収集できます。

統計収集操作は、シリアルまたはパラレルに実行できます。可能であれば、DBMS_STATS はパラレル問合せをコールして、指定した並行度で統計を収集します。それ以外の場合は、シリアル問合せまたは ANALYZE 文をコールします。索引統計は、パラレルには収集されません。

この統計は、正確に計算するか、行またはブロックのランダムなサンプリングから推定できます（22-14 ページの「正確な統計と推定による統計」を参照）。

パーティション表および索引の場合、DBMS_STATS ではパーティションごとに別々の統計を収集したり、表または索引全体のグローバル統計を収集できます。同様に、コンポジット・パーティション化の場合、DBMS_STATS ではサブパーティション、パーティションおよび表または索引全体について、別々の統計を収集できます。最適化する SQL 文に応じて、オプティマイザはパーティション（またはサブパーティション）の統計とグローバル統計のどちらを使用するかを選択します。

DBMS_STATS が収集するのはコストベース最適化のための統計のみで、他の統計は収集しません。たとえば、DBMS_STATS が収集する表統計には、行数、その時点でデータが含まれているブロック数および行の長さの平均が含まれますが、連鎖行数、平均空き領域または未使用のデータ・ブロック数は含まれません。

> **追加情報：** DBMS_STATS パッケージを使用した統計の収集方法の例は、『Oracle8*i* チューニング』を参照してください。

索引用の COMPUTE STATISTICS オプション Oracle は、B* ツリー索引またはビットマップ索引の作成または再構築中に、一部の統計を自動的に収集できます。この統計収集を使用可能にするには、CREATE INDEX または ALTER INDEX ... REBUILD の COMPUTE STATISTICS オプションを使用します。

COMPUTE STATISTICS オプションを指定した場合に Oracle によって収集される統計は、索引がパーティション化されているかどうかによって異なります。

- 非パーティション索引の場合、索引の作成または再構築中には、索引、表および列の統計が収集される。連結キー索引の場合、列統計はキーの先行列のみを参照します。
- パーティション索引の場合、索引の作成中またはそのパーティションの再構築中には、表または列の統計は収集されない。
  - パーティション索引の作成時には、各パーティションと索引全体の索引統計が収集される。索引がコンポジット・パーティション化を使用している場合は、各サブパーティションの統計も収集されます。
  - 索引のパーティションまたはサブパーティションの再構築中には、そのパーティションまたはサブパーティションの索引統計のみが収集される。

統計の正確さを保証するために、COMPUTE STATISTICS オプションを指定して索引を作成するときには、索引作成に他の索引が使用可能な場合でも、常に実表が使用されます。

> **追加情報：** CREATE INDEX および ALTER INDEX コマンドの COMPUTE STATISTICS オプションの詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

ANALYZE コマンド ANALYZE コマンドを使用すると、コストベース最適化のための統計も生成できます。次のように各種の制限を伴うため、この目的に ANALYZE コマンドを使用することはお薦めしません。

- ANALYZE は常にシリアルに実行される。
- ANALYZE では、パーティション表とパーティション索引のグローバル統計が、直接収集されないで計算される。このため、個々の値の数など、一部の統計が不正確になる可能性があります。
  - パーティション表とパーティション索引の場合、ANALYZE では個々のパーティションの統計が収集された後に、パーティション統計からグローバル統計が計算される。

- コンポジット・パーティション化の場合、ANALYZE ではサブパーティションの統計が収集された後に、サブパーティション統計からパーティション統計とグローバル統計が計算される。

■ ANALYZE では、DBMS_STATS によって収集された統計値の一部を上書きも削除もできない。

ANALYZE では、連鎖行に関する情報や、索引、表およびクラスタの構造の一貫性に関する情報など、オプティマイザが使用しない追加情報を収集できます。この情報は、DBMS_STATS では収集されません。

> **追加情報：** ANALYZE 文の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

正確な統計と推定による統計 DBMS_STATS パッケージまたは ANALYZE コマンドによって収集される統計には、正確な統計と推定による統計があります。（索引の作成または再構築の COMPUTE STATISTICS オプションを指定すると、常に正確な統計が収集されます。）

正確な統計を計算するには、Oracle は索引、表、パーティションまたはスキーマ内のすべてのデータを読み込む必要があります。その時点で表にデータが含まれているデータ・ブロック数や、ルート・ブロックからリーフ・ブロックまでの索引の深さなど、一部の統計は常に正確に計算されます。

統計を推定するために、Oracle はデータのサンプルをランダムに選択します。サンプリング率と、サンプリングの基礎を行にするかブロックにするかを指定できます。

「ROW サンプリング」では、ディスク上の物理位置に関係なく行が読み込まれます。これにより、推定用に最もランダムなデータが得られますが、必要以上のデータが読み込まれることがあります。たとえば、最悪の場合は、行サンプルによって各ブロックから 1 行ずつ選択され、全表走査や全索引走査が必要になることがあります。

「Block サンプリング」では、ブロックのサンプルがランダムに読み込まれ、そのブロック内のすべての行が推定に使用されます。これにより、特定のサンプル・サイズに対する I/O アクティビティは減少しますが、行がディスク上にランダムに分布していなければ、サンプルのランダム性が低下するおそれがあります。Block サンプリングは、索引統計には使用できません。

統計の管理

この項では、統計表と、データ・ディクショナリに格納された統計情報を表示するビューについて説明します。

統計表 DBMS_STATS パッケージを使用すると、統計を「統計表」に格納できます。列、表、索引またはスキーマに関する統計を統計表に転送し、後からデータ・ディクショナリに復元できます。オプティマイザは、統計表に格納されている統計は使用しません。

統計表により、さまざまな統計の集合を試験的に使用できます。たとえば、統計の集合のバックアップを作成してから、オリジナルの統計を削除または変更したり、新規に生成したりできます。その後、さまざまな統計の集合に合わせて最適化された SQL 文のパフォーマンスを比較し、表に格納されている統計が最高のパフォーマンスを発揮した場合は、それをデータ・ディクショナリに復元できます。

その場合、統計の集合を 1 つの統計表にまとめて保存する方法と、複数の統計表を作成して別々に格納する方法があります。

統計の表示 DBMS_STATS パッケージを使用すると、データ・ディクショナリや統計表に格納されている統計を表示できます。

また、データ・ディクショナリ内の統計を問い合せるときには、次のデータ・ディクショナリ・ビューも使用できます。

- USER_TABLES、ALL_TABLES および DBA_TABLES
- USER_TAB_COLUMNS、ALL_TAB_COLUMNS および DBA_TAB_COLUMNS
- USER_INDEXES、ALL_INDEXES および DBA_INDEXES
- USER_CLUSTERS および DBA_CLUSTERS
- USER_TAB_PARTITIONS、ALL_TAB_PARTITIONS および DBA_TAB_PARTITIONS
- USER_TAB_SUBPARTITIONS、ALL_TAB_SUBPARTITIONS および DBA_TAB_SUBPARTITIONS
- USER_IND_PARTITIONS、ALL_IND_PARTITIONS および DBA_IND_PARTITIONS
- USER_IND_SUBPARTITIONS、ALL_IND_SUBPARTITIONS および DBA_IND_SUBPARTITIONS
- USER_PART_COL_STATISTICS、ALL_PART_COL_STATISTICS および DBA_PART_COL_STATISTICS
- USER_SUBPART_COL_STATISTICS、ALL_SUBPART_COL_STATISTICS および DBA_SUBPART_COL_STATISTICS

追加情報： これらのビュー内の統計の詳細は、『Oracle8*i* リファレンス』を参照してください。

コストベースのアプローチを使用する場合

一般に、新しいアプリケーションには、すべてコストベースのアプローチを使用してください。ルールベースのアプローチは、コストベースの最適化が使用できるようになる前に作成されたアプリケーションのためのものです。コストベースの最適化は、リレーショナル・データとオブジェクト型のどちらにも使用できます。

次の機能に対しては、コストベースの最適化しか使用できません。

- パーティション表
- パーティション・ビュー
- 索引構成表
- 逆キー索引
- ビットマップ索引
- ファンクションベース索引
- SELECT 文の SAMPLE 句
- パラレル問合せおよびパラレル DML
- スター型変換
- スター型結合
- 拡張可能な最適化

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用するかの詳細は、『Oracle8*i* チューニング』を参照してください。

拡張可能な最適化

拡張可能な最適化により、ユーザー定義関数とドメイン索引の作成者は、コストベース最適化で実行プランの選択に使用される、統計、選択性およびコスト評価という 3 つの主要コンポーネントを制御できます。

拡張可能な最適化により、次のことが可能になります。

- コスト・ファンクションとデフォルト・コストを、ドメイン索引、索引タイプ、パッケージおよびスタンドアロン・ファンクションに対応付ける。
- 選択関数とデフォルトの選択性を、オブジェクト型、パッケージ・ファンクションおよびスタンドアロン・ファンクションのメソッドに対応付ける。
- 統計収集関数を表のドメイン索引および列に対応付ける。
- コストに基づいて述語を関数で順序付けする。
- アクセス・コストに基づいて表のユーザー定義アクセス方法（ドメイン索引）を選択する。
- ANALYZE コマンドを使用して、ユーザー定義の統計収集関数および統計削除関数を起動する。

- 新しいデータ・ディクショナリ・ビューを使用して、列、ドメイン索引、索引タイプまたは関数に対応付けた統計収集関数、コスト・ファンクションまたは選択関数に関する情報を含める。
- ヒントを追加して、関数の述語の評価順を保つ。

追加情報： 拡張可能な最適化の詳細は、『Oracle8*i* データ・カートリッジ開発者ガイド』を参照してください。

ユーザー定義統計

ドメイン索引、表の個々の列およびユーザー定義データ型に対して、「統計収集関数」を定義できます。

統計を収集するためにドメイン索引が分析されるたびに、Oracle はそれに対応する統計収集関数をコールします。表列が分析されると、Oracle はその列の標準統計を収集し、対応する統計収集関数をコールします。データ型の統計収集関数が存在する場合、Oracle は分析される表内でそのデータ型を持つ列ごとに、統計収集関数をコールします。

ユーザー定義選択性

SQL 文の述語の選択性は、特定のアクセス方法のコストを推定するため、および最適の結合順序を決定するために使用されます。ユーザー定義オペレータに関する情報はないので、この種のオペレータを含む述語については、正確な選択性を計算できません。

ユーザー定義オペレータ、スタンドアロン・ファンクション、パッケージ・ファンクションまたは型のメソッドを含む述語に対して、「選択関数」を定義できます。オプティマイザは、定数 <、<=、=、>=、> または LIKE のいずれかに関連してオペレータ、関数またはメソッドを含む述語を検出すると、ユーザー定義の選択関数をコールします。

ユーザー定義コスト

オプティマイザは、ドメイン索引の内部記憶構造を認識しないため、この種の索引のコストの正確な推定を計算できません。また、PL/SQL の起動、再帰 SQL の使用および BFILE へのアクセスを行うユーザー定義関数や、CPU 集中型の関数については、コストを過小に推定する場合があります。

ドメイン索引とユーザー定義のスタンドアロン・ファンクション、パッケージ・ファンクションおよび型のメソッドのコストを定義できます。これらのユーザー定義コストは、オプティマイザが単に参照するだけのデフォルト・コスト形式で使用する方法と、オプティマイザがコストを計算するためにコールする完全なコスト・ファンクションとして使用する方法があります。

ルールベース最適化

ルールベースのアプローチを使用すると、オプティマイザは、使用可能なアクセス・パスと、そのアクセス・パスのランク（23-36 ページの表 23-1 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用できます。

Oracle のアクセス・パスのランク付けは、ヒューリスティックな手法です。SQL 文を実行する方法が 2 つ以上ある場合、ルールベースのアプローチでは、常にランクが低いほうの操作を使用します。通常、ランクの低い操作は、高いほうのランクの構成体に対応付けされている操作よりも高速に実行されます。

詳細は、23-52 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。

注意： Oracle8*i* の一部の拡張機能には、ルールベース最適化を使用できません。これらの機能のリストは、22-15 ページの「コストベースのアプローチを使用する場合」を参照してください。

23

オプティマイザの操作

$dy/dx = 0; d^2y/dx^2 > 0.$

Leibniz

この章では、SQL 文の実行方法をどのように Oracle オプ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。

- オプ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

追加情報： オプティマイザの詳細は、『Oracle8*i* チューニング』を参照してください。

オプティマイザの操作の概要

この項では、Oracle のオプ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。

オプティマイザの操作

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

式と条件の評価	オプティマイザは、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。（23-4 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、オプ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。（23-9 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、オプ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。（23-14 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	オプティマイザは、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。（23-30 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、オプティマイザは、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。（23-32 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、オプティマイザは最初に結合する表のペアを選択し、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。（24-2 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、オプティマイザは結合を実行するのに使用する操作を選択します。（24-2 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	単一の表のみが関係する INSERT、UPDATE、DELETE または 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用してこれらの表の行を組み合せて、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。
非同一レベル結合	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（+）が使用されている結合条件。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	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せた結果になります。 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成したものです。
複合文	副問合せを含む INSERT、UPDATE、DELETE または 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に、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外側の部分を、「親文」といいます。
複合問合せ	集合演算子（UNION、UNION ALL、INTERSECT または MINUS）を使用して 2 つ以上の単純文または複合文を組み合せる問合せ。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、結合文または複合文。

分散型の文	分散データベースの複数のノード上にあるデータにアクセスする文。「リモート文」は、分散データベースの 1 つのリモート・ノード上にあるデータにアクセスします。33-10 ページの「リモート SQL 文と分散 SQL 文」を参照してください。

式と条件の評価

オプティマイザは、可能であれば式を完全に評価し、特定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にはさまざまな SQL 構成体があり得るため（= ANY(副問合せ) と IN(副問合せ) など）、Oracle はそれらを単一の構成体に変換します。

この後の各項では、オプティマイザ次の要素を含む式と条件を評価する方法について説明します。

- 定数
- LIKE 演算子
- IN 演算子
- ANY または SOME 演算子
- ALL 演算子
- BETWEEN 演算子
- NOT 演算子
- 推移律
- DETERMINISTIC 関数

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ます。

```
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、オプティマイザはその条件を第 2 の条件に単純化します。

オプティマイザが比較演算子を超えて式を単純化することはありません。前述の例では、第 3 の式を第 2 の式に単純化することはありません。このため、アプリケーション開発者は、列が含まれる式を比較する条件ではなく、できるだけ定数と列を比較する条件を作成してください。

LIKE 演算子

オプティマイザは、LIKE 比較演算子を使用して、ワイルド・カード文字が使用されていない式を比較する条件を、等号演算子を使用する等価の条件に単純化します。たとえば、オプティマイザは、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'
```

オプティマイザがこの式を単純化できるのは、可変長のデータ型を比較する場合のみです。たとえば、ENAME のデータ型が CHAR(10) の場合、等価（=）演算子は空白埋め比較方法になりますが、LIKE は空白埋めを行わないので、オプティマイザは LIKE 演算子を等価演算に変換できません。

IN 演算子

オプティマイザは、IN 比較演算子を使用する条件を、等価比較演算子と OR 論理演算子を使用する等価の条件に展開します。たとえば、オプティマイザは、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、23-16 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

オプティマイザは、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 論理演算子を使用する条件に展開します。たとえば、オプティマイザは、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
```

オプティマイザは、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、オプティマイザは、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

オプティマイザは、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 論理演算子を使用する等価の条件に展開します。たとえば、オプティマイザは、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
```

オプティマイザは、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用する等価の条件に変換します。たとえば、オプティマイザは、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、オプティマイザは、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用して、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

オプティマイザは、BETWEEN 比較演算子を使用する条件を、>= および <= 比較演算子を使用する等価の条件に置き換えます。たとえば、次の最初の条件は第 2 の条件に置き換えられます。

```
sal BETWEEN 2000 AND 3000

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

NOT 演算子

オプティマイザは、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、オプティマイザは、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、いろいろな方法で記述できます。オプティマイザは、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、オプティマイザは、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、オプティマイザは移行の原理（推移律）を使用して第 3 の条件を推論することがあります。この後、オプティマイザは、推論した条件を使用し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： 推移律を使用するのは、コストベースのアプローチの場合のみです。

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、SQL ファンクション、リテラル、バインド変数および相関変数が含まれている定数式。

例 : それぞれ EMP.DEPTNO 列を使用する 2 つの条件が WHERE 句に含まれている、次の問合せを考えます。

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

オプティマイザは、推移律を使用し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、その索引を使用するアクセス・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のみ推論を実行します。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
```

DETERMINISTIC 関数

オプティマイザは、ユーザーが記述した関数を実行するのではなく、前に計算した値を使用できる場合があります。これが安全なのは、限定的な方法で動作する関数の場合のみです。この関数は、特定の入力引数値の集合について、常に同じ出力戻り値を戻す必要があります。

パッケージ変数、データベース、または NLS パラメータなどのセッション・パラメータの内容に違いがあっても、この関数の結果は、常に同一でなければなりません。また、この関数が後で再定義される場合も、その出力戻り値は特定の入力引数値セットに関する前の定義で計算された値と同じでなければなりません。関数を再実行するかわりに前に計算された値を使用するなど、アプリケーションの正確さに影響する副次作用が発生しないようにする必要があります。

関数の作成者は、CREATE FUNCTION 文で関数を宣言するとき、または CREATE PACKAGE 文か CREATE TYPE 文で、キーワード DETERMINISTIC を使用し、Oracle サーバーに対して、この関数がこれらの制限に従って動作することを保証できます。明らかにデータベースやパッケージ変数を操作する関数を DETERMINISTIC と宣言できる場合にも、サーバーはこの宣言を検査しません。(21-7 ページの「DETERMINISTIC 関数」を参照)。プログラマは、このキーワードを適切な場合にのみ使用する必要があります。

DETERMINISTIC 関数のコールは、同じ問合せの中で複数回コールされる場合や、この関数の関連コールを含むファンクションベース索引やマテリアライズド・ビューが定義されている場合は、すでに計算されている値を使用するように置き換えることができます。

> **追加情報：** プラグマ RESTRICT_REFERENCES の説明は『Oracle8*i* アプリケーション開発者ガイド 基礎編』、CREATE FUNCTION、CREATE INDEX および CREATE MATERIALIZED VIEW の説明は『Oracle8*i* SQL リファレンス』を参照してください。また、ファンクションベース索引の説明は 10-24 ページの「ファンクションベース索引」、マテリアライズド・ビューの詳細は『Oracle8*i* チューニング』を参照してください。

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多数あります。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、ある文を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 から複合問合せへの変換
- 複合文から結合文への変換
- 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

結合、セミ・ジョインまたはアンチ・ジョインを含む文の最適化に関する追加情報は、第 24 章「結合の最適化」を参照してください。

OR から複合問合せへの変換

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 集合演算子を使用する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、異なる索引を使用し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用する文の場合、オプティマイザは、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用して推定して比較し、その変換を実施するかどうかを決める。
- コストベースのアプローチでは、同じ列での IN リストまたは OR のための OR 変換は使用されません。そのかわりに IN リスト反復演算子を使用します。

追加情報： 詳細は、『Oracle8*i* 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、23-36 ページの表 23-1 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、オプ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。

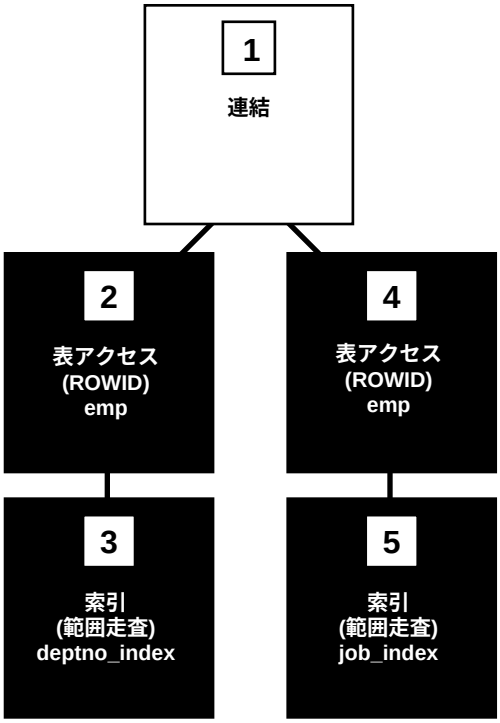
```
SELECT *
  FROM emp
  WHERE job = 'CLERK'
UNION ALL
SELECT *
  FROM emp
  WHERE deptno = 10
    AND job <> 'CLERK';
```

コストベースのアプローチを使用している場合、オプティマイザは、元の問合せを全表走査を使用し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用している場合は、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用して実行できるので、オプティマイザはこの UNION ALL 変換を実行します。ルールベースのアプローチは、元の問合せを全表走査を使用して実行するよりも、複合問合せを 2 つの索引走査を使用して実行するほうが高速であることを前提としています。

図 23-1 に、変換された文の実行計画を示します。

図 23-1 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用して 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 を使用して、各コンポーネント問合せの条件を満たしている行を検出する。
- 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、オプティマイザはその変換を考慮しません。全表走査と索引走査を使用して複合問合せを実行したほうが、全表走査を使用して元の問合せを実行するよりも高速になることは、おそらくあり得ません。

例 : ENAME 列にのみ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;
```

こ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、オプ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します。

複合文から結合文への変換

複合文を最適化する場合、オプ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、その結合文が複合文と正確に同じ行を戻すことが保証されていれば、オプティマイザは常に複合文を結合文に変換します。この変換により、Oracle は、24-2 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検討します。

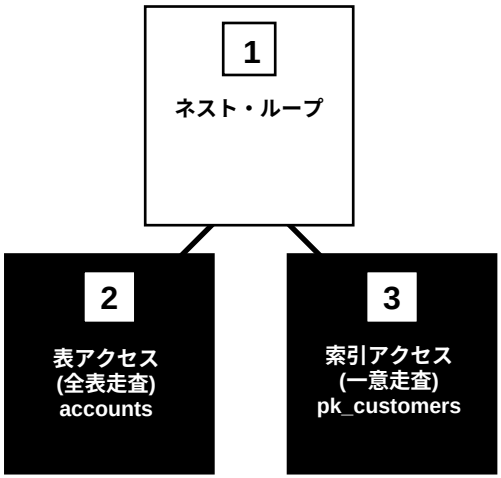
```
SELECT *
  FROM accounts
  WHERE custno IN
    (SELECT custno FROM customers);
```

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、オプティマイザはこの複合問合せを、それと同じデータを戻すことが保証されている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 23-2 のようになります。

図 23-2 ネスト・ループ・ジョインの実行計画



この文を実行する場合、Oracle はネスト・ループ・ジョイン操作を実行します。ネスト・ループ・ジョインの詳細は、24-2 ページの「結合操作」を参照してください。

オプティマイザが複合文を結合文に変換できない場合、オプティマイザは親文の実行計画と副問合せの実行計画を、それぞれを別々の文であるものとして選択します。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用し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、オプティマイザはこの文を変換しません。副問合せに AVG などの集計関数を含む複合問合せは、結合文に変換できないので注意してください。

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、オプ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。オプティマイザは、次のいずれかの手法を使用し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、元の文を使用してアクセスする。（23-24 ページの「元の文を使用し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、オプティマイザは、ビューの名前を問合せブロックにある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、表示および結合のみを含むビューです。つまり、集合演算子、集計関数、DISTINCT、GROUP BY、CONNECT BY などは含まないビューです（23-15 ページの「マージ可能ビューとマージ不可能ビュー」を参照）。

例： 部門（deptno）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、部門（deptno）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;
```

オプティマイザは、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 列または 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。

マージ可能ビューとマージ不可能ビュー 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、オプ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
- 選択リスト内の集計関数（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のみ、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、オプティマイザは複合ビューのマージを使用することができます。詳細は、24-11 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、オプティマイザは、複合ビューのマージが使用可能にされている場合にのみ、ビューの問合せをアクセスする文にマージできます。複合マージを使用すると、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。（23-16 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。複合マージを使用可能にするには、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または MERGE ヒントを使用する必要があります。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ければ、オプティマイザは別のアプローチを使用します（23-17 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*i* チューニング』を参照してください。

例 1: GROUP BY 句を使用したビュー 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、オプ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のみを選択して、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 句でも使用できます。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、オプティマイザは、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

オプティマイザは、ビューの問合せ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 つの従業員表の結合である TWO_EMP_TABLES ビューを考えます。このビューは、UNION 集合演算子を使用する複合問合せを使用して定義されます。

```
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;
```

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、どちらかの表に記述されている、部門（deptno）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、オプティマイザは、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、オプ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。

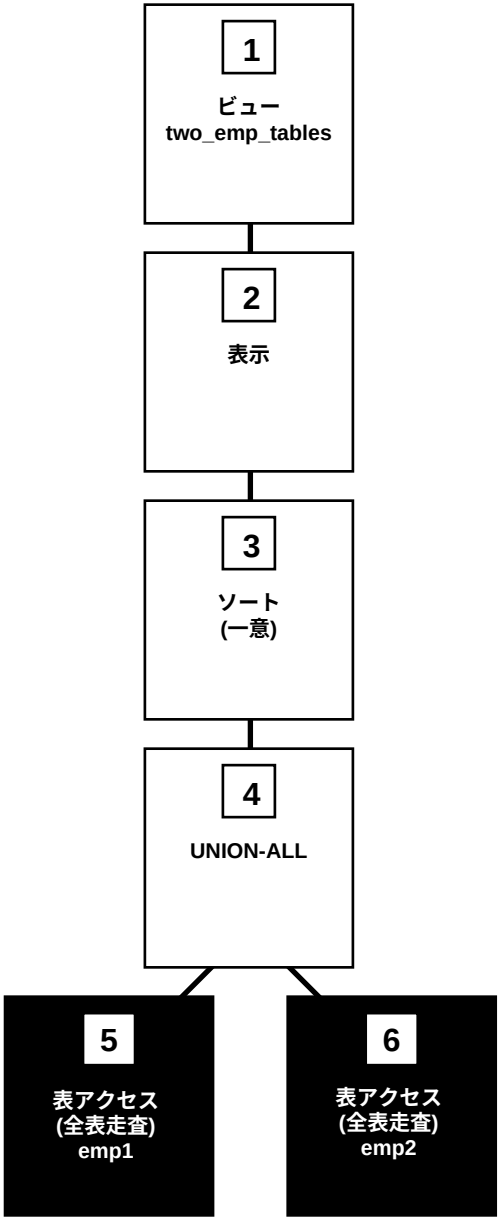
この結果の文は、次のようになります。

```
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 23-3 に、結果の文の実行計画を示します。

図 23-3 UNION 集合演算子を使用し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 の結果から必要な列を抽出する。
- 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、給与平均額、給与最低額および給与最高額を示す、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 ビューから部門（deptno）10 の給与平均額、最低額および最高額を選択する、次の問合せを考えます。

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

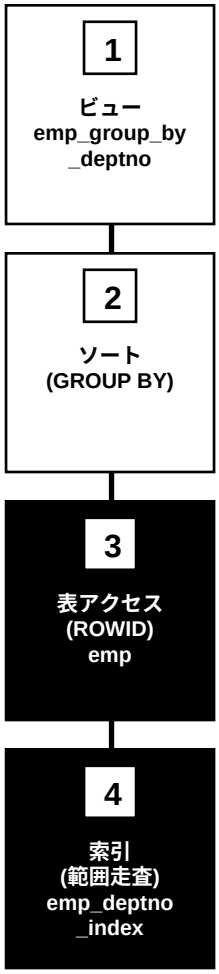
オプティマイザは、文の述語（WHERE 句条件）をビューの問合せに追加することにより、この文を変換します。この結果の文は、次のようになります。

```
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 23-4 に、結果の文の実行計画を示します。この実行計画では、DEPTNO 列の索引を使用します。

図 23-4 GROUP BY 句を使用し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 を取り出す。

- ステップ 3 で、ステップ 4 で取り出した ROWID を使用して 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、最小値および最大値を計算する。
- 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。

ビューへの集計関数の適用 オプティマイザは、ビューの問合せに集計関数（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、最低額および最高額の平均を、従業員表から導出するものです。

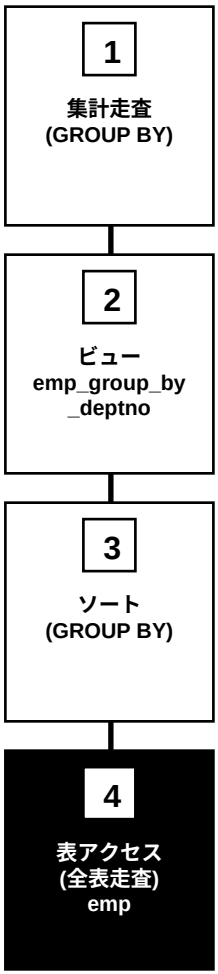
```
SELECT AVG(avg_sal), AVG(min_sal), AVG(max_sal)
    FROM emp_group_by_deptno;
```

オプティマイザは、ビューの問合せの選択リストに AVG 集計関数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 23-5 に、結果の文の実行計画を示します。

図 23-5 GROUP BY 句を使用して定義されたビューに集計関数を適用する



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、最小値および最大値を計算する。
- 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。

元の文を使用したビューの行へのアクセス

オプティマイザは、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したり、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用し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;
```

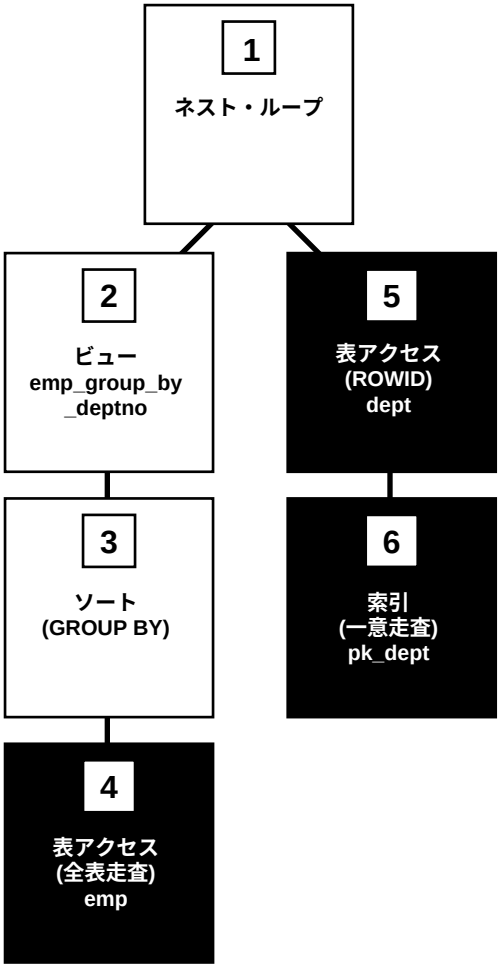
このビューにアクセスする、次の問合せを考えます。この問合せは、DEPT 表にある部門の名前と場所に、このビューに表されているそれぞれの部門の給与平均額、最低額および最高額を結合するものです。

```
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;
```

実表にのみアクセスする等価の文はないので、オプティマイザはこの文を変換できません。そのかわりに、オプティマイザは、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用する場合と同様の方法で使用します。

図 23-6 に、この文の実行計画を示します。ネスト・ループ・ジョイン操作の実行方法の詳細は、24-2 ページの「結合操作」を参照してください。

図 23-6 GROUP BY 句を使用して定義したビューの表への結合



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、最小値および最大値を計算する。
- 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用する。

- 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 値を使用して PK_DEPT 索引の一意走査を実行する。
- ステップ 5 で、ステップ 6 で戻されるそれぞれの ROWID を使用して、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。

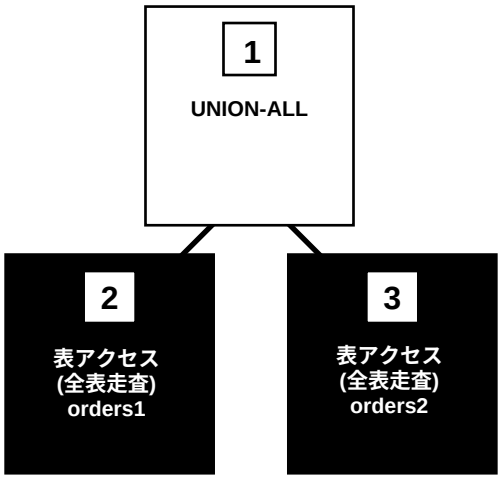
複合問合せの最適化

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、複合問合せで使用さ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 23-7 に、この文の実行計画を示します。ここでは、UNION ALL 演算子を使用して、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 23-7 UNION ALL 集合演算子を使用した複合問合せ



この文を実行する場合、Oracle は次のステップを実行します。

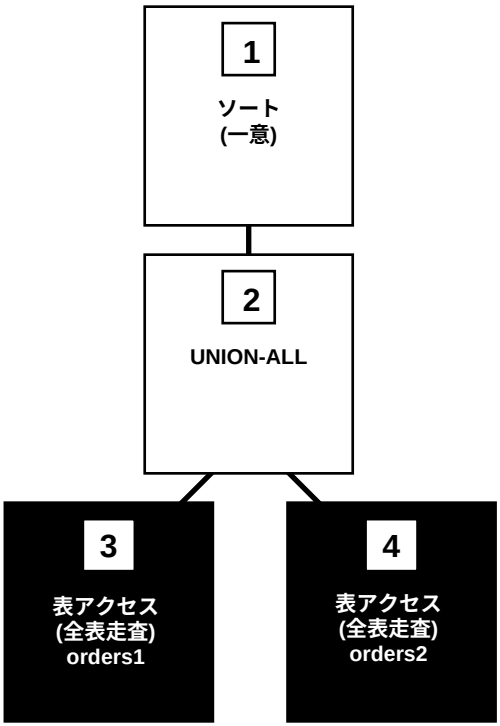
- ステップ 2 と 3 で、ORDERS1 表と ORDERS2 表に対して全表走査を実行する。

- 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 23-8 に、UNION 演算子を使用して、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 23-8 UNION 集合演算子を使用した複合問合せ

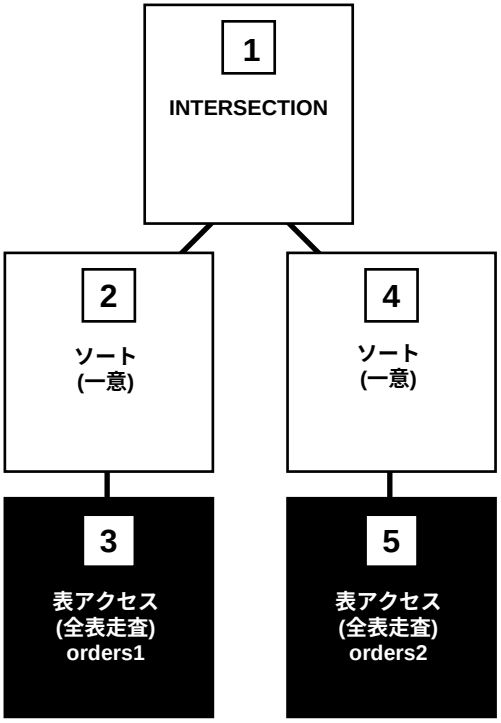


この実行計画は、23-26 ページの図 23-7 に示されている UNION-ALL 演算子の場合の実行計画とほとんど同じですが、UNION-ALL 操作で戻された重複値を SORT 操作を使用して排除している点が違います。

図 23-9 に、INTERSECT 演算子を使用して、ORDERS1 表と ORDERS2 表の両方に含まれている部品のみ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 23-9 INTERSECT 集合演算子を使用した複合問合せ



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のみを戻す INTERSECTION 操作を実行する。

分散型の文の最適化

オプティマイザは、ローカル・データにのみアクセスする文の実行計画を選択するのとほぼ同じ方法で、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 文を送信する。リモート Oracle インスタンスは、その文を実行し、その結果のみをローカル・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する。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、オプティマイザは、ローカル・データベースで索引を考慮するのとまったく同様に、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、オプ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。さらにオプティマイザは、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、オプ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プ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。有効な値は、次のとおりです。

CHOOSE
: オプティマイザは、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少なくともどれか 1 つに関する統計がデータ・ディクショナリに入っている場合、オプティマイザはコストベースのアプローチを使用し、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、オプティマイザはルールベースのアプローチを使用します。この設定値が、このパラメータのデフォルト値です。

ALL_ROWS
: オプティマイザは、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用し、最高のスループット（文全体を完了するのに使用する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
: オプティマイザは、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用し、最短の応答時間（結果セットの先頭の行を戻すのに使用する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
: オプティマイザは、統計情報があるかどうかに関係なく、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。

オプティマイザが SQL 文にコストベースのアプローチを使用しており、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、オプティマイザは、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）を使用して、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、コストベース最適化に使用する列、表、クラスタ、索引および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。DBMS_STATS パッケージ、ANALYZE コマンド、CREATE または ALTER INDEX コマンドの COMPUTE STATISTICS 句を使用すると、これらのスキーマ・オブジェクト内で物理記憶特性やデータ分布に関する正確な統計または見積りを収集できます。

オプティマイザに最新の統計を提供するために、統計に影響する方法でスキーマのデータや構造を変更した後は、新しい統計情報を収集する必要があります。統計の収集方法のの詳細は、22-9 ページの「コストベース最適化の統計」を参照してください。

ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータ

ALTER SESSION コマンドの OPTIMIZER_GOAL パラメータを使用すると、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
: オプティマイザは、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、少なくとも 1 つの表の統計がデータ・ディクショナリに入っていれば、オプティマイザはコストベースのアプローチを使用し、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、オプティマイザはルールベースのアプローチを使用します。

ALL_ROWS
: オプティマイザは、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用し、最高のスループット（文全体を完了するのに使用する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
: オプティマイザは、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用し、最短の応答時間（結果セットの先頭の行を戻すのに使用する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
: オプティマイザは、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 および RULE の各ヒント

個々の SQL 文中にある FIRST_ROWS、ALL_ROWS、CHOOSE または 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*i* チューニング』を参照してください。

PL/SQL とオプティマイザの目標

オプ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、直接送られた問合せにのみ適用されます。

- 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。

ヒントを使用すると、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 がデータに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使用できる時期
- 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、マルチブロック読込みを使用し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。

サンプル表スキャン

サンプル表スキャンでは、表からランダムなデータ・サンプルが取り出されます。このアクセス方法が使用されるのは、文の FROM 句に SAMPLE オプションまたは SAMPLE BLOCK オプションが含まれている場合です。行別にサンプリングするときに（SAMPLE オプション）サンプル表スキャンを実行するために、Oracle は指定されたパーセンテージだけ表の行を読み込み、各行を検査して文の WHERE 句の条件を満たしているかどうかを判断します。ブロック別にサンプリングするときに（SAMPLE BLOCK オプション）サンプル表スキャンを実行するために、Oracle は指定されたパーセンテージだけ表のブロックを読み込み、サンプリングしたブロックの各行を検査して、文の WHERE 句の条件を満たしているかどうかを判断します。

Oracle は、問合せに結合またはリモート表が関係している場合のサンプル表スキャンはサポートしていません。結合と分散型の文の詳細は、23-3 ページの「SQL 文のタイプ」を参照してください。

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・ファイルその行が入っているデータ・ブロックおよび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 で行を検索する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検索します。

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 は最初にクラスタ索引を走査して、選択する行の 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検索します。

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検索する場合にハッシュ走査を使用できます。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、そのハッシュ値を持つ行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、索引からデータを取り出します。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。その文が索引の列にのみ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 も含まれています。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用して、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 のみが必要な場合、Oracle は一意走査を実行します。たとえば、文が確実に 1 行にのみ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用できます。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用できます。この走査では、ブロックが 1 つずつ読み込まれます。

高速全走査
: 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用できます。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、ソート操作の排除のためには使用できません。高速全走査は、（全索引走査と違って）マルチブロック読込みを使用して索引全体を読み込み、パラレル化することができます。

  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のみです。この走査を使用するには、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または INDEX_FFS ヒントで指定します。

索引結合

索引結合は、問合せで参照される表からのすべての列が一緒に含まれている、複数の索引のハッシュ結合です。索引結合を使用する場合、関連するすべての列値を索引から取り出せるので、表アクセスは不要です。これは、ソート操作を排除する目的では使用できません。

索引結合が使用可能なのは、コストベースで最適化する場合のみです。索引結合走査を使用するには、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または INDEX_JOIN ヒントで指定します。

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用します。ビットマップは、ブール演算子を使用して AND または OR 条件を解決し、WHERE 句に指定された複数の条件に対応する索引を効率的にマージします（10-30 ページの「ビットマップ索引」を参照）。

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のみです。

注意： ビットマップ索引は、Oracle8*i* Enterprise Edition を購入した場合にのみ利用できます。

アクセス・パス

表 23-1 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・パスを使用し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では、リソースの使用量に基づいてパスを選択する（23-48 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であれば、それぞれのパスのランクを使用してパスを選択する（23-52 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 23-1 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	
—	サンプル表スキャン（ルールベース最適化では使用不可）。23-47 ページの「サンプル表スキャン（ランクなしのアクセス・パス）」を参照。
—	高速全索引走査（ルールベース最適化では使用不可）。『Oracle8*i* チューニング』を参照。
—	索引結合（ルールベース最適化では使用不可）。23-48 ページの「索引結合（ランクなしのアクセス・パス）」を参照。
—	ビットマップ索引走査（ルールベース最適化では使用不可）。24-16 ページの「スター型変換」を参照。

この後の各項では、アクセス・パスとそのパスが使用可能な場合、Oracle でデータ・アクセスに使用される方法、および EXPLAIN PLAN コマンドによって生成される出力について説明します。

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 構文で選択された行が識別される場合のみです。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、次の文で使用できます。

```
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 ROWID       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用できます。

- 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 行のみ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 演算子を使用し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・ループ操作を実行します。（ネスト・ループ操作の詳細は、24-2 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、次のような文に使用できます。

```
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用している。複合クラスタ・キーの場合は、すべての等価条件を AND 演算子を使用して組み合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 行のみ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用し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、次のような文に使用できます。

```
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用し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 演算子を使用して組み合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 によってその表にアクセスします。

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、次のような文に使用できます。

```
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 演算子を使用して組み合せる必要があります。この文を実行する場合、Oracle はネスト・ループ操作を実行します。（ネスト・ループ操作の詳細は、24-2 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、次のような文に使用できます。

```
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用し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 演算子を使用して組み合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用し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、次のような文に使用できます。

```
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用し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 演算子を使用して組み合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 を使用し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 のみですみます。

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、次のような文に使用できます。

```
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 演算子で組み合せた等価条件で使用し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 を使用し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、次のような文で使用できます。

```
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 つ以上の単一列の索引の列を等価条件の中で使用している文に使用可能です。複数の単一列索引の場合、これらの条件をすべて AND 演算子を使用して組み合せる必要があります。

WHERE 句が 1 つの索引の列のみを使用している場合、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、次のような文で使用できます。

```
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 句に多数の単一列の索引の列を使用されている場合、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 を使用し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用されている場合、Oracle はそのうち 5 つをマージし、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、次のような文で使用できます。

```
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
    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用し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用できません。

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、次のような文で使用できます。

```
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用できます。

```
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用した条件がどれか 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または最大値（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、次のような文で使用できます。

```
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、次のような文で使用できます。

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

WHERE 句で ORDER 列、つまり索引の先頭部分を使用しているので、このアクセス・パスを使用できます。

例: このアクセス・パスは、ORDER 列と LINE 列に対する索引がある、次のような文では使用できません。

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

WHERE 句で、索引の先頭部分ではない LINE 列しか使用されていないので、このアクセス・パスは使用できません。

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用されている場合に、クラスタに一緒には格納されていない表を結合する文で使用できます。そのような文を実行する場合、Oracle はソート / マージ操作を使用します。また、Oracle は、結合文を実行するのにネスト・ループ操作も使用します。（これらの操作の詳細は、24-2 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、次のような文に使用できます。

```
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 文に使用できます。

- 問合せが、MAX または MIN 関数を使用して、単一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用できます。
- 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。この値のみを選択すればよいので、Oracle は索引を走査した後で表に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、次のような文で使用できます。

```
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 文に使用できます。

- 問合せに、単一列索引の列か複合索引の先頭部分のどちらかを使用する 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、次のような文で使用できます。

```
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

このアクセス・パスは、FROM 句に SAMPLE または SAMPLE BLOCK が含まれている場合を除き、WHERE 句の条件に関係なく、任意の SQL 文に使用できます。

次の条件により、索引アクセス・パスが使用できなくなるので注意してください。

- 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'

この場合、*column* に索引があるかどうかは関係ありません。

- expr = expr2

この場合、*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

前述の構成体のみが含まれており、索引アクセス・パスが使用できるようになる他の要素が含まれていない SQL 文では、全表走査を使用する必要があります。

例 : この文は、全表走査を使用して 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
```

サンプル表スキャン（ランクなしのアクセス・パス）

このアクセス・パスは、FROM 句に SAMPLE または SAMPLE BLOCK を含む SELECT 文に使用できます。サンプル表スキャンには、コストベース最適化が必須です。

例: この文は、サンプル表スキャンを使用してブロック単位でサンプリングし、EMP 表の 1% にアクセスします。

```
SELECT *
    FROM emp SAMPLE BLOCK (1);
```

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            SAMPLE         EMP
```

索引結合（ランクなしのアクセス・パス）

このアクセス・パスは、表の複数の索引列で検出されたデータにアクセスする SELECT 文に使用できます。関連するすべての列値は索引から取り出せるので、表アクセスは不要です。索引結合には、コストベース最適化が必須です。

例: この文は、索引結合を使用して、EMP 表の索引列 EMPNO および SAL 列にアクセスします。

```
SELECT empno, sal
    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
  VIEW                                   index$_join$_001
    HASH JOIN
      INDEX             RANGE SCAN       EMP_SAL
      INDEX             FAST FULL SCAN   EMP_EMPNO
```

アクセス・パスの選択

この項では、コストベースまたはルールベースのアプローチを使用する場合に、オプティマイザが使用可能なアクセス・パスからどのように選択するかについて説明します。

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。

- その文で使用できるアクセス・パス

- それぞれのアクセス・パスまたはパスの組合せを使用し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（および SAMPLE または SAMPLE BLOCK オプションの場合は FROM 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、オプティマイザは、使用可能なアクセス・パスを使用して、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、列および表の統計を使用して、それぞれの計画のコストを推定します。最後に、オプティマイザは、推定コストが最も低い実行計画を選択します。

文の FROM 句に SAMPLE または SAMPLE BLOCK が含まれている場合を除き、使用可能な中からオプティマイザによって選択されたアクセス・パスはヒントで上書きできます。

> **追加情報：** SQL 文のヒントの詳細は、『Oracle8*i* 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、オプ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 句で使用されている演算子
  - WHERE 句で使用さ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込みを使用するので、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込みの回数によって決まります。このマルチブロック読込みの回数は、1 回のマルチブロック読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 句で等価条件を使用して、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）を使用し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用して、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（各表の行の数）

オプティマイザは、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。オプティマイザは、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用します。

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、オプティマイザは、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）を使用します。これらの統計は、USER_TAB_COL_STATISTICS ビュー（または USER_TAB_COLUMNS ビュー）で検出できます。オプティマイザは、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、オプティマイザは、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用します。

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用されている、次の問合せを考えます。

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

オプティマイザには、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、オプティマイザは、直前の例で説明した方法を使用してこの問合せの選択性を判断できません。この場合、オプティマイザは、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（列に索引があるため）。オプティマイザは、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用されている場合に、必ずそのように仮定します。

オプ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用しているという点のみが異なる 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用されている場合と、SQL*Plus において同じ SQL 文に定数が指定される場合とでは、オプ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 演算子を使用する条件で範囲境界値として 2 つのバインド変数を使用している、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 演算子を使用して選択するための、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用して、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*）と絶対値関数（ABS）を使用して、BETWEEN 条件の選択性（*S*）を推定します。

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

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。

- その文で使用できるアクセス・パス
- これらのアクセス・パスのランク（23-36 ページの表 23-1 を参照）

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、索引を使用するアクセス・パスが必ず選択されます。

WHERE 句の条件の順序は、通常、オプ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 索引を使用する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 索引を使用する非有界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用している場合、オプティマイザは、ENAME_IND 索引を使用するアクセス・パスを選択してこの文を実行します。オプティマイザは、使用可能なパスの中でこのパスが最も高いランクなので、このパスを選択します。

24

結合の最適化

Will you, won't you, will you, won't you, will you join the dance?

Lewis Carroll

この章では、結合、アンチ・ジョインおよびセミ・ジョインを含む SQL 文が Oracle オプティマイザでどのように実行されるかについて説明します。オプティマイザでビットマップ索引を使用してスター問合せを実行し、事実表を複数のディメンション表に結合する方法についても説明します。この章の内容は、次のとおりです。

- 結合文の最適化
- アンチ・ジョインとセミ・ジョインの最適化
- 「スター」問合せの最適化

追加情報： オプティマイザの詳細は、『Oracle8*i* チューニング』を参照してください。

結合文の最適化

結合文の実行計画を選択するために、オプティマイザは、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、オプティマイザは、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。（23-32 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ペアを結合するために、Oracle は次のいずれかの操作を実行する必要があります。

- ネスト・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

オプティマイザは、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・ループ・ジョイン
- ソート / マージ結合
- クラスタ結合
- ハッシュ結合

ネスト・ループ・ジョイン

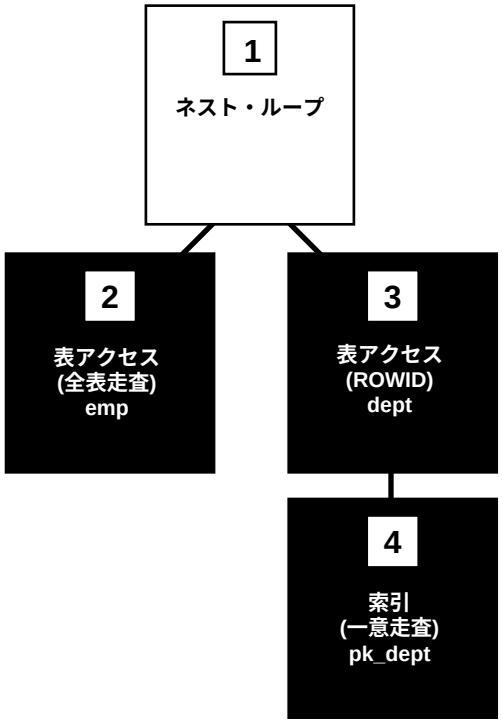
ネスト・ループ・ジョインを実行する場合、Oracle は次のステップに従います。

1. オプティマイザが、どちらか一方の表を「外部表」（「駆動表」）として選択する。他方の表を「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各ペアのデータを結合し、結果の行を戻す。

図 24-1 に、ネスト・ループ・ジョインを使用した次の文の実行計画を示します。

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

図 24-1 ネスト・ループ・ジョイン



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、全表走査を使用して外部表（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 値を使用して 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 を使用して内部表（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

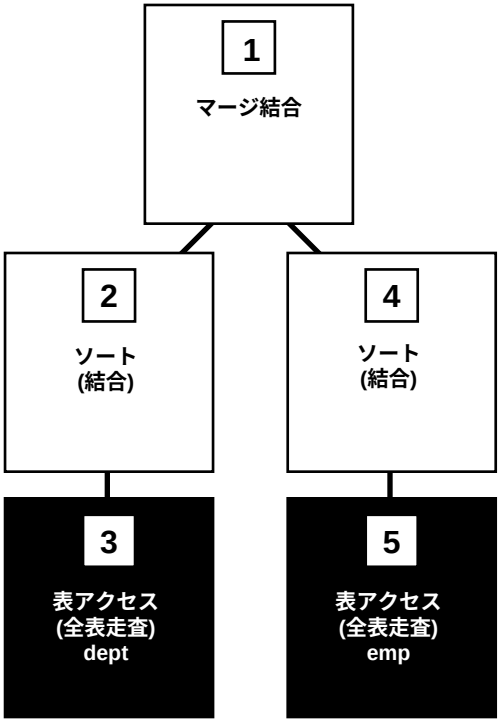
Oracle がソート / マージ結合を実行できるのは、同一レベル結合の場合だけです。ソート / マージ結合を実行する場合、Oracle は次のステップに従います。

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、結合操作で使用する列の値を基準にしてソートされます。
2. Oracle は、結合条件で使用された列に一致する値が入っている行のペアを組み合せて 2 つのソースをマージし、結果の行ソースとして戻します。

図 24-2 に、ソート / マージ結合を使用した次の文の実行計画を示します。

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

図 24-2 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 の各行をステップ 4 の一致する各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

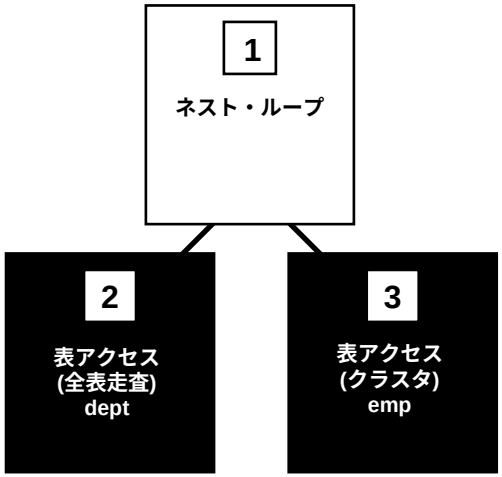
Oracle がクラスタ結合を実行できるのは、同じクラスタに入っている 2 つの表のクラスタ・キー列を等号で結ぶ同一レベル結合の場合のみです。クラスタでは、同じクラスタ・キー値を持つ両方の表の行が同じブロックに格納されるので、Oracle はそれらのブロックにアクセスするだけで済みます。

追加情報：『Oracle8*i* 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 24-3 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 24-3 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、全表走査を使用して外部表（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 値を使用し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・ループ・ジョインにすぎません。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 は一致する行に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のみです。ハッシュ結合は、ルールベース最適化では使用できません。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 ヒントを使用して、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。

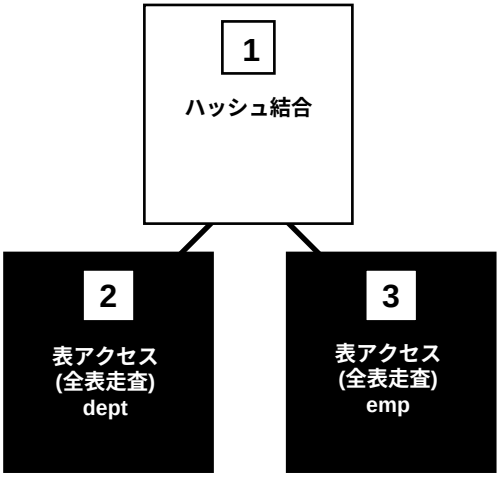
ハッシュ結合を実行する場合、Oracle は次のステップに従います。

1. Oracle は、各表に対して全表走査を実行し、使用可能なメモリーかに基づいて各表をできるだけ多数のパーティションに分割します。
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成します（可能であれば、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用してハッシュ表を調べます。メモリーに収まりきらないサイズのパーティションのペアは、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）のペアごとに、小さい方のパーティションを使用してハッシュ表を作成し、大きい方のパーティションを使用してハッシュ表をプローブします。

図 24-4 に、ハッシュ結合を使用した次の文の実行計画を示します。

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

図 24-4 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成し、ステップ 3 の結果の各行を使用してハッシュ表を調べる。

ハッシュ結合操作に使用するメモリー量を制御するには初期化パラメータ HASH_AREA_SIZE を使用し、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 を使用します。

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*i* 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、オプ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用している場合
- ルールベースのアプローチを使用している場合

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。

- オプティマイザは、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。オプティマイザは、表の UNIQUE 制約および PRIMARY KEY 制約に基づいて、このような状況を認識します。このような状況がある場合、オプ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、オプティマイザは、その他の表セットの結合を最適化します。
- 外部結合条件が使用されている結合文の場合、外部結合演算子が使用されている表は、結合順序ではもう一方の表の後に置かれます。オプティマイザは、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、オプティマイザは、有効な結合順序、結合操作および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、オプティマイザは、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。オプティマイザは、次の方法でコストを推定します。

- ネスト・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いる。オプティマイザは、これらのコストをデータ・ディクショナリの統計を使用して推定します。
- ソート / マージ結合のコストの大部分は、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いる。
- オプティマイザは、それぞれの操作のコストを決定する際に、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る。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込み数を多くすると、ネスト・ループ・ジョインよりも、ソート / マージ結合のほうがコストが低くなる傾向がある。1 回の I/O で読み込める順次ブロックの数が多くなると、全表走査よりもネスト・ループ・ジョインの内部表にある索引のほうがパフォーマンスが改善される程度が低くなる傾向があります。マルチブロック読込み数は、初期化パラメータ DB_FILE_MULTIBLOCK_READ_COUNT で指定します。
  - 外部結合条件が使用されている結合文の場合、外部結合演算子が使用されている表は、結合順序ではもう一方の表の後に置かれます。オプティマイザは、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、オプティマイザによる結合順序の選択は、ORDERED ヒントを使用して上書きできます。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、オプ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、オプティマイザによる結合操作の選択を、このヒントを使用して上書きすることもできます。

> **追加情報：** ヒントの使用方法の詳細は、『Oracle8*i* 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、オプティマイザは、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用して、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 23-36 ページの表 23-1 で示したアクセス・パスのランクに従って、最上位にランクされている使用可能なアクセス・パスを持つ表を選択します。オプティマイザは、このステップを繰り返して、結合順序でその後に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用するネスト・ループ操作を選択する。

   – 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 / マージ操作を選択する。

   – 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用するネスト・ループ操作を選択する。

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用してアクセスするネスト・ループ・ジョイン操作の数を最大化することです。ネスト・ループ・ジョイン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・ループ・ジョイン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・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用して先頭の表がアクセスされる計画を選択する。

– 先頭の表が境界付き範囲走査を使用してアクセスされる複数の実行計画が同順位である場合、オプティマイザは、複合索引の先頭部分にある列のうち最も多くを使用して先頭の表にアクセスする計画を選択する。

d. 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、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、オプティマイザは、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用することができます。

- ビューに実表が 1 つしかない場合、オプティマイザは「ビューのマージ」を使用できる。
- ビューに複数の実表がある場合、オプ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。(23-14 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 name_view, dept
  WHERE dept.deptno = name_view.deptno(+)
    AND dept.deptloc = 'London';
```

オプティマイザは、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のみを選択します。

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 OPTIMIZER_FEATURES_ENABLE 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、オプティマイザは結合述語をビューに追加できます（23-17 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、ハッシュ結合からネスト・ループ・ジョインへの変換や、全表走査から索引走査への変換など、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：** オプティマイザ・ヒントの詳細は、『Oracle8*i* 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(+);
```

オプティマイザは、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');
```

アンチ・ジョインとセミ・ジョインの最適化

「アンチ・ジョイン」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、アンチ・ジョインでは、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');
```

オプティマイザは、デフォルトでは NOT IN 副問合せにネスト・ループのアルゴリズムを使用します。ただし、初期化パラメータ ALWAYS_ANTI_JOIN が MERGE または HASH に設定されており、NOT IN 副問合せをソート / マージまたはハッシュ・アンチ・ジョイン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 副問合せに挿入し、オプティマイザが使用するアルゴリズムを指定できます。

「セミ・ジョイン」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せずに、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 行である必要があります。EMP 内の BONUS 列に索引がない場合は、セミ・ジョインを使用して問合せのパフォーマンスを改善できます。

オプティマイザは、デフォルトでは EXISTS 副問合せにネスト・ループのアルゴリズムを使用します。ただし、初期化パラメータ ALWAYS_SEMI_JOIN が MERGE または HASH に設定されており、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 副問合せに挿入し、オプティマイザが使用するアルゴリズムを指定できます。

追加情報： オプティマイザ・ヒントの詳細は、『Oracle8*i* 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。その特徴は、1 つ以上の非常に大規模な「事実」表にデータ・ウェアハウスのうち主要な情報が含まれ、その事実表内の特定の属性のエントリに関する情報が、小規模な多数の「ディメンション」表（「参照」表）含まれていることです。

「スター問合せ」とは、1 つの事実表と多数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、参照表が相互に結合されることはありません。

コストベース最適化では、スター問合せが認識され、その問合せの効率的な実行計画が生成されます。（スター問合せは、ルールベース最適化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」および「市場」というキーを含む表が考えられます。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、通常は事実表に記載されている各製品番号に関する情報が入れられます。

「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実行できるようになっています。

追加情報： ディメンションとデータ・ウェアハウスの詳細は、『Oracle8*i* チューニング』を参照してください。

スター問合せの例

ここでは、次の例を使用して、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、コストベース最適化を使用する必要があります。最初に、問合せでアクセスされる表ごとに、統計を（DBMS_STATS パッケージまたは ANALYZE コマンドを使用して）収集します。

索引

前述の例では、tkey、pkey および mkey 列に対して連結索引を組み立てることができます。索引内の列の順序はパフォーマンスを左右するので、データの順序付けを使用してください。大規模な表に日時の順で行が追加されている場合は、tkey 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の静的抽出である場合は、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省略した問合せが頻繁に出てくる場合は、索引を追加すると便利です。この例で、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、ヒントを使用して、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用します。

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

もっと一般的なのは、STAR ヒント「/*+ STAR */」を使用する方法です。

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する必要がある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用すれば解決できます。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的とした、コストベースの問合せ変換のことです。ディメンションの数が少なく、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。

- ディメンションの数が多い。
- 事実表の密度が低い。
- いくつかのディメンション表に対する制約述語が指定されていない問合せがある。

スター型変換はディメンション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、ディメンションの数が多すぎたりするために、大規模な直積演算を実行する必要がある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、制約ディメンション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さまざまな問合せの異なる制約ディメンションのパターンと一致する、いくつもの索引連結を作成する必要はありません。

注意： ビットマップ索引は、Oracle8*i* Enterprise Edition を購入した場合にのみ使用可能です。Oracle8*i* では、ビットマップ索引は使用不能であり、スター問合せ処理には B* ツリー索引を使用します。Oracle8*i* 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用することもできます。

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用できる、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" および "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, d3
  WHERE fact.c1 = d1.c1 AND fact.c2 = d2.c1 AND fact.c3 = d3.c3
    AND d1.c2 IN (1, 2, 3, 4)
    AND d2.c2 < 100
    AND d3.c2 = 35
    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)
```

また、コスト面の効率がよければ、1 つ以上の副問合せの結果を一時表に格納してさらに最適化できます。これにより、副問合せは一時表の副問合せに置き換えられます。たとえば、この一時表の変換用に前述の最初の副問合せが選択された場合は、次のように ORA_TEMP_1_123 という一時表が作成され、副問合せの結果で埋められます。

```
SELECT d1.c1 from d1 where d1.c2 in (1, 2, 3, 4)
```

完全に変換された問合せは次のようになります。

```
SELECT * FROM fact, ORA_TEMP_1_123, d2, d3
  WHERE fact.c1 = ORA_TEMP_1_123.c1 AND fact.c2 = d2.c1 and fact.c3 = d3.c1
    AND ORA_TEMP_1_123.c1 IN (1, 2, 3, 4)
    AND d2.c2 < 100
    AND d3.c2 = 35
    AND fact.c1 IN (SELECT ORA_TEMP_1_123.c1 FROM ORA_TEMP_1_123)
    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用して、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 c1 のそれぞれの値について、ビットマップが fact.c1 の索引から取得され、各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用して 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 および 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

前述の問合せは、次のような実行計画になります。

```
SELECT STATEMENT
 TEMP TABLE GENERATION
 TEMP TABLE GENERATION
  HASH JOIN
   HASH JOIN
    HASH JOIN
     TABLE ACCESS              FACT            BY INDEX 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         ORA_TEMP_1_123  FULL
          BITMAP INDEX         FACT_C1         RANGE SCAN
        BITMAP MERGE
         BITMAP KEY ITERATION
          TABLE ACCESS         D2              FULL
          BITMAP INDEX         FACT_C2         RANGE SCAN
     TABLE ACCESS              ORA_TEMP_1_123  FULL
    TABLE ACCESS               D2              FULL
   TABLE ACCESS                D3              FULL
```

この計画では、マージされた 3 つの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。それぞれの行ソース・ツリーは、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・パスを使用して、関連する fact 表の行が取り出された後、それらの行がディメンション表および一時表と結合されて、問合せへの回答が生成されます。"TEMP TABLE GENERATION" というラベルが付いた実行計画の 2 つの行には、一時表の作成と移入に使用された SQL コマンドが含まれています。これらのコマンドは前述の例には表示されていませんが、実行計画の OTHER 列にあります。

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。オプティマイザは、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、オプティマイザはその変換を問合せに適用し、適用が可能な場合は、変換後の問合せを使用し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、オプティマイザはこの 2 つのバージョンのうちどちらの計画を使用するかを決定します。

問合せで、事実表にある行の大部分にアクセスする必要がある場合は、変換ではなく全表走査を使用してください。ただし、ディメンション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のみ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。

オプティマイザがディメンション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妥当であると判断した場合だけです。すべてのディメンション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、オプティマイザは、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用します。

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。一時表なしでスター型変換を使用するには、このパラメータの値を TEMP_DISABLE に設定します。STAR_TRANSFORMATION ヒントを使用すると、オプティマイザは、変換が適用された中で最良の計画を使用するようになります。

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プ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る必要がある）
- 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）
- アンチ・ジョイン表
- 副問合せの中ですでにディメンション表として使用されている表
- 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

また、次の場合には、一時表はスター型変換に使用されません。

- データベースが読込み専用モードになっている。
- スター問合せが直列可能モードのトランザクションの一部になっている。

第VII部

パラレル SQL とダイレクト・ロード・インサート

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

25

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・インサート用の Oracle ダイレクト・ロード・インサート機能について説明します。ダイレクト・ロード・インサートといくつかの DDL 文で使用可能な NOLOGGING 機能についても説明します。この章のトピックは次のとおりです。

- ダイレクト・ロード・インサートの基礎知識
- 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
  - ロギング・モード
- ダイレクト・ロード・インサートについてのその他の考慮事項
- ダイレクト・ロード・インサートの制限事項

パラレル固有の情報は、第 26 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*i* Enterprise Edition を購入した場合にのみ使用可能です。

追加情報： パラレル・ダイレクト・ロード・インサートのチューニング方法の詳細は、『Oracle8*i* 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・キャッシュを使用せずに直接実行することにより、インサート操作のパフォーマンスを改善します。この機能は、ダイレクト・ローダー・ユーティリティ（SQL*Loader）の機能と似ています。

ダイレクト・ロード・インサートは、挿入するデータを、表中の既存のデータの後に追加します。既存のデータ内の空き領域は再利用されません。データは、パーティション表または非パーティション表にパラレルまたはシリアルで挿入できます。

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。これらの機能の詳細は、25-3 ページの「ダイレクト・ロード・インサート文の種類」を参照してください。ダイレクト・ロード・インサートのパラレル化オプションと Partitioning Option の詳細は、第 26 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・インサートを使用することの主な利点は、REDO エントリまたは UNDO エントリのログを記録せずにデータをロードできることです。これにより、挿入時のパフォーマンスが著しく改善されます。シリアルとパラレルのどちらでも、ダイレクト・ロード・インサートには、従来型パス INSERT と比べてパフォーマンス上の利点があります。

対照的に、従来型パス INSERT を使用した場合は、オブジェクト内の空き領域が再利用され、参照整合性が維持されます。挿入のための従来型パスは、パラレル化できません。

CREATE TABLE ... AS SELECT との比較

ダイレクト・ロード・インサートを使用すると、新しい表を作成するかわりに、既存の表にデータを挿入できます。ダイレクト・ロード・インサートは表の索引を更新しますが、CREATE TABLE ... AS SELECT は索引のない新しい表を作成するだけです。26-28 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 を使用する場合は、必ずトランザクションが最小単位になります。複数のパラレル・ロードを使用する場合には、最小単位としての実行が保証されません。さらに、パラレル・ロードでは、索引の更新中にエラーが発生した場合に、ロード終了時に表の索引のいくつかが「UNUSABLE」状態のままになることがあります。対照的に、パラレル INSERT では、表と索引がアトミックに更新されます（つまり、索引の更新中にエラーが発生すると、文はロールバックされます）。

追加情報： パラレル・ロードの詳細は、『Oracle8*i* ユーティリティ・ガイド』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 構文のみで、INSERT... *values* の構文はサポートされていません。INSERT ... SELECT のパラレル化は、パラレル・ヒントまたはパラレルの表定義句のどちらかによって決定されます。

> **追加情報：** INSERT ... SELECT 文の構文の詳細は、『Oracle8*i* 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、表のセグメントまたは各パーティションのセグメントの現行の高水位標（データ受入れ用にブロックが過去にフォーマットされたことのないレベル）の後に挿入されます。コミットを実行すると、高水位標が新しい値に更新され、他のプロセスからもデータが見えるようになります。
- **非パーティション表へのパラレル・ダイレクト・ロード・インサート。**各パラレル実行サーバーが新しい一時セグメントを割り当てて、そこにデータを挿入します。コミットを実行すると、パラレル実行コーディネータは新しい一時セグメントを 1 次表セグメントにマージします。（パラレル実行コーディネータとパラレル実行サーバーの詳細は、26-5 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。**各パラレル実行サーバーに 1 つ以上のパーティションが割り当てられ、どのパーティションでも 1 つのプロセスしか作動していません。パラレル・サーバー・プロセスは、割り当てられたパーティション・セグメントの現行の高水位標の後にデータを挿入します。コミットを実行すると、各パーティション・セグメントの高水位標がパラレル実行コーディネータによって新しい値に更新され、他のプロセスからもデータが見えるようになります。

いずれにしても、高水位標の更新または一時セグメントのマージは、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・インサートを使用するには、APPEND ヒントが必要です。パラレル・ダイレクト・ロード・インサートの場合、文中の PARALLEL ヒント、または表定義の PARALLEL 句が必要です。APPEND ヒントはオプションです。パラレル・ダイレクト・ロード・インサートでは、パラレル DML を ALTER SESSION ENABLE/FORCE PARALLEL DML 文を使用してオンにする必要もあります。

表 25-1 に、ダイレクト・ロード・インサートの要件および従来型 INSERT との比較をまとめます。

表 25-1 シリアルおよびパラレルの INSERT ... SELECT 文のまとめ

挿入のタイプ	シリアル	パラレル
ダイレクト・ロード・インサート	可能。次のものが必要。 ■ SQL 文での APPEND ヒント	可能。次のものが必要。 ■ ALTER SESSION ENABLE/FORCE PARALLEL DML ■ 表の PARALLEL 属性、または文の PARALLEL ヒント（APPEND ヒントはオプション）
従来型 INSERT	可能（デフォルト）	不可

シリアル・ダイレクト・ロード・インサートとパラレル・ダイレクト・ロード・インサートの例

シリアル・ダイレクト・ロード・インサートは、APPEND ヒントで指定できます。次に例を示します。

```
INSERT /*+ APPEND */ INTO emp
   SELECT * FROM t_emp;
COMMIT;
```

パラレル・ダイレクト・ロード・インサートは、行の挿入先の表の PARALLEL 属性を設定することによって指定できます。次に例を示します。

```
ALTER TABLE emp PARALLEL (10);
ALTER SESSION ENABLE PARALLEL DML;
INSERT INTO emp
   SELECT * FROM t_emp;
COMMIT;
```

行を選択する選択元の表の PARALLEL 属性を設定することにより、次のように、SELECT 操作の並行性も指定できます。

```
ALTER TABLE emp PARALLEL (10);
ALTER TABLE t_emp PARALLEL (10);
ALTER SESSION ENABLE PARALLEL DML;
INSERT INTO emp
   SELECT * FROM t_emp;
COMMIT;
```

INSERT 操作または SELECT 操作のための PARALLEL ヒントは、表の PARALLEL 属性より優先されます。たとえば、次の INSERT ... SELECT 文の並行度は、EMP 表と T_EMP 表に PARALLEL 属性が設定されているかどうかには関係なく、12 になります。

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL(emp,12) */ INTO emp
   SELECT /*+ PARALLEL(t_emp,12) */ * FROM t_emp;
COMMIT;
```

パラレル INSERT 文の詳細は、26-21 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、パーティションまたは索引にロギングなしのモードを指定するには、ALTER TABLE コマンド、ALTER INDEX コマンドまたは ALTER TABLESPACE コマンドを使用します。

- **ロギングありのダイレクト・ロード・インサート。**このモードでは、インスタンスおよびメディア回復用に完全な REDO ログが生成されます。「ロギングあり」がデフォルトのモードです。
- **ロギングなしのダイレクト・ロード・インサート。**このモードでは、データは挿入されますが、REDO または UNDO ログは生成されません。（ただし、新しいエクステントが無効であることをマークするために最小限のロギングが実行されます。またディクショナリの変更については常に完全なログが記録されます。）メディア回復時にこれが適用されると、REDO データのログが記録されていないので、「エクステント無効」レコードは特定のブロック範囲を論理的に破損しているものとしてマークします。

ロギングなしモードにすると、生成されるログ・データが非常に少なくなるため、パフォーマンスが向上します。メディア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のバックアップを作成する必要があります。

ロギングなしモードと離散トランザクションの間に相互作用はなく、離散トランザクションでは常に REDO 情報が生成されます。（17-8 ページの「離散トランザクションの管理」を参照）。離散トランザクションは、NOLOGGING 属性を使用する表に対して発行できます。

注意： ロギングありモードとロギングなしモードは、表、パーティションまたは索引の永続的な属性ではありません。挿入先のデータベース・オブジェクトにデータを挿入し、バックアップを作成した後は、オブジェクトの状況をロギングありモードにして、それ以降の変更のログが記録されるように設定できます。

表 25-2 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 25-2 LOGGING オプションと NOLOGGING オプションのまとめ

挿入のタイプ	LOGGING	NOLOGGING
ダイレクト・ロード・インサート	可能。回復には次のものが必要。 ■ ARCHIVELOG データベース・モード	可能。次のものが必要。 ■ 表領域、表、パーティションまたは索引の 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。 ■ ARCHIVELOG データベース・モード	不可

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。次に例を示します

```
ALTER TABLE emp NOLOGGING;
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL(emp,12) */ INTO emp
   SELECT /*+ PARALLEL(t_emp,12) */ * FROM t_emp;
COMMIT;
```

パーティション、表領域または索引にも、NOLOGGING 属性を設定できます。次に例を示します。

```
ALTER TABLE emp MODIFY PARTITION emp_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用できる SQL 文

表、パーティション、索引または表領域に NOLOGGING 属性を設定できますが、ロギングなしモードは、NOLOGGING 属性を設定したスキーマ・オブジェクトに対して実行される操作のすべてに適用されるわけではありません。ロギングなしモードを使用できるのは、次の操作のみです。

- ダイレクト・ロード（SQL*Loader）
- ダイレクト・ロード・インサート
- CREATE TABLE ...AS SELECT
- CREATE INDEX
- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER INDEX ... SPLIT PARTITION
- ALTER INDEX ... REBUILD
- ALTER INDEX ...REBUILD PARTITION
- オフラインで格納される NOCACHE NOLOGGING モードの LOB の INSERT、UPDATE および DELETE

これらの SQL 文はすべてパラレル化できます（第 26 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 文は、スキーマ・オブジェクトの NOLOGGING 属性の影響を受けません。たとえば、このような SQL 文には、UPDATE と DELETE（前述のように一部の LOB を除く）、従来型パスの INSERT および前述のリストにない各種の DDL 文があります。

デフォルトのロギング・モード

LOGGING 句または NOLOGGING 句が指定されていない場合、表、パーティションまたは索引のロギング属性のデフォルトは、それが存在する表領域のロギング属性になります。

LOB の場合に LOGGING 句または NOLOGGING 句が省略されると、次の規則が適用されます。

- CACHE が指定されている場合は、LOGGING が使用される（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

この項では、ダイレクト・ロード・インサートのための索引のメンテナンス、領域の割当ておよびデータのロックについて説明します。

索引のメンテナンス

ローカル索引またはグローバル索引を持つ非パーティション表またはパーティション表へのダイレクト・ロード・インサートの場合、INSERT 操作の終わりに索引のメンテナンスが行われます。この索引のメンテナンスは、パーティション表または非パーティション表に対するパラレル・ダイレクト・ロード・インサートの場合にはパラレル実行サーバーによって行われ、シリアル・ダイレクト・ロード・インサートの場合には単一のプロセスによって行われます。

ダイレクト・ロード・インサートによって表内のデータのほとんどが変更される場合は、インサートの前に索引を削除し、後で再作成することにより、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するため、従来型パス INSERT より多くの領域が必要です。非パーティション表へのパラレル・ダイレクト・ロード・インサートの場合、表セグメントの高水位標より後にある空きブロックも無視されます。ダイレクト・ロード・インサートを使用する場合は、さらにいくつかの領域要件を考慮する必要があります。

非パーティション表に対してパラレル・ダイレクト・ロード・インサートを実行すると、一時セグメント（並行度ごとに 1 つのセグメント）が作成されます。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 を使用すると、一時セグメントが 4 つ作成されます。

各パラレル実行サーバーは、最初にデータを一時セグメントに挿入し、最後に、すべての一時セグメントに入れられているデータを表に追加します。（CREATE TABLE... AS SELECT と同じメカニズムです。）

パーティション表に対するパラレル INSERT の場合は、一時セグメントは作成されません。各パラレル実行サーバーは、データを高水位標の後のパーティションに挿入するだけです。

ローカルに管理されておらず、自動モードでない非パーティション表に対してパラレル INSERT を実行し、次のパラメータの値を変更すると、不要になった領域を無駄にせずに、一時セグメント用の記憶領域を十分に確保できます。

- NEXT（オブジェクトに割り当てられる、そのオブジェクトの次のエクステントのサイズ（単位はバイト））
- PCTINCREASE（第 3 以降のエクステントが直前のエクステントを上回るときの成長率）

- MINIMUM EXTENT（表領域内の各使用済みエクステントや使用可能エクステントのサイズが、最低でも指定した値以上で、かつその値の倍数になるようにして、表領域内の空き領域の断片化を制御）

これらのパラメータには、次のような値を選択してください。

- 各エクステントが小さくなりすぎない程度のサイズ（1MB 以上）。このサイズは、オブジェクト内の合計エクステント数に影響します。
- 各エクステントが、パラレル INSERT によって必要以上に大きいセグメントの領域を無駄にすることのない程度のサイズ。

NEXT および PCTINCREASE パラメータの値を変更するには、ALTER TABLE 文で STORAGE オプションを指定します。MINIMUM EXTENT パラメータの値は、ALTER TABLESPACE 文で変更できます。パラレル DML 文を実行した後、NEXT、PCTINCREASE および MINIMUM EXTENT 記憶領域パラメータを、非パラレル操作に適した設定に戻すことができます。

記憶領域の計算

ここで使用する記号は、次のとおりです。

- オブジェクト用のデータの総量は *V*。
- 並行度は *DOP*。
- 未使用領域のパーセンテージは *P*。

NEXT の計算 パラレル実行サーバーあたりの平均データ量は、V / DOP です。NEXT 記憶領域パラメータの値は、次の範囲内で指定できます。

1 MB < NEXT の値 < V / DOP

NEXT の値をサーバーあたりの平均データ量に近い値に設定すると、エクステント数は減少しますが、オブジェクト内に大量の未使用領域が生じることがあります。未使用領域が最大になるのは、各パラレル実行サーバーにまったく未使用のエクステントがあり、未使用領域の量が DOP * NEXT に等しい場合です。つまり、平均未使用領域は、(DOP * NEXT) / 2 となります。計算式 (P * V) / 100 を使用して、未使用領域の許容量を決定してください。NEXT に適切な値を決定するには、次の計算式を使用します。

NEXT = (2 * P * V) / (100 * DOP)

P の値は、エクステントが 1MB 以上（オブジェクトが大きい場合はできれば 20MB 以上）になるように設定する必要があります。また、P の値は、合計エクステント数が 1000 を超えず、特にローカル管理でない表領域を持つエクステントが適度な大きさになるようにしてください。

PCTINCREASE の計算 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成されることがあります。パラレル DML の実行中に領域をすべて使用してしまわないように、PCTINCREASE は必ず 0 に設定してください。

MINIMUM EXTENT の計算 データベース内のすべてのオブジェクトがほぼ同じエクステント・サイズを使用している場合は、表領域オプション MINIMUM EXTENT の値を NEXT に近い値に設定してください。少数のエクステントを使用可能なオブジェクトがある場合は、未使用領域が減少するように、MINIMUM EXTENT の値を小さくします。

記憶領域の計算例

例 25-1 に、適切な記憶領域パラメータ値を求める計算式を示し、例 25-2 には不適切な記憶領域パラメータ値を求める計算式を示します。

例 25-1 ダイレクト・ロード・インサート用の適切な記憶領域パラメータ値

V = 500GB ($5 * 10^{11}$ バイト)

DOP = 100

P = 5 %

NEXT = $(2 * 5 * 5 * 10^{11}) / (100 * 100) = 5 * 10^{7}$ = 500 MB

- 各エクステントのサイズは 500MB。
- 合計エクステント数は 1,000。
- 未使用領域の平均量は、割当済み領域全体の 5%。

例 25-2 ダイレクト・ロード・インサート用の不適切な記憶領域パラメータ値

V = 50MB ($5 * 10^{7}$ バイト)

DOP = 10

P = 5 %

NEXT = $(2 * 5 * 5 * 10^{7}) / (100 * 10) = 5 * 10^{5}$ = 0.5 MB

この場合、

- 各エクステントのサイズは 0.5MB。
- 合計エクステント数は 100。
- 未使用領域の平均量は、割当済み領域全体の 5%。

サイズが 0.5MB のエクステントは小さすぎるので、P の値を大きくする方が適切です。

追加情報： 領域管理の詳細は、『Oracle8*i* 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、更新または削除を同時に実行できないようにします。ただし、問合せを同時に実行することは許されているので、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、このようなロックがあるため、索引の同時作成または同時再作成はできません。このことは、表の同時実行性に影響してくるため、ダイレクト・ロード・インサートを使用する前に考慮しておく必要があります。詳細は、26-38 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 を使用したダイレクト・パス・パラレル・ロードの場合と同じです。どちらも同じメカニズムが基礎になっているからです。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 の場合、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用してシリアルに実行されます。1 つのトランザクションで同じ表に複数回アクセスする文に関する制限は例外で、この場合はエラー・メッセージが出されます。

たとえば、表にトリガーまたは参照整合性がある場合にダイレクト・ロード・インサート（シリアルまたはパラレル）を使用しようとすると、PARALLEL ヒントまたは句（存在する場合）だけでなく 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、26-40 ページの「パラレル DML の制限事項」を参照してください。

26

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
- パラレル問合せ
- パラレル DDL
- パラレル DML
- 関数のパラレル実行
- 親和性
- その他の並行性

注意： この章で説明するパラレル実行機能は、Oracle8*i* Enterprise Edition を購入した場合にのみ使用できます。また、パラレル実行は Oracle Parallel Server と同じではありません。Parallel Server Option は、SQL 文をパラレル実行する際には必要ありません。ただし、パラレル実行のいくつかの面は、Oracle Parallel Server にのみ適用されます。

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 はさらに高速で実行できます。

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および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、パフォーマンスが一段と向上します。システムの CPU とディスク・コントローラの負荷がすでに大きい場合は、パフォーマンスを改善するため、パラレル実行を使用する前にシステムの負荷を軽減するか、またはハードウェア・リソースを増やす必要があります。

> **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方法の詳細は、『Oracle8*i* 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。

- 表走査
- ネスト・ループ・ジョイン
- ソート・マージ結合
- ハッシュ結合
- NOT IN
- GROUP BY
- SELECT DISTINCT
- UNION と UNION ALL
- 集計操作
- SQL からコールされる PL/SQL ファンクション
- ORDER BY
- CREATE TABLE AS SELECT
- CREATE INDEX

- 索引の再構築
- 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
- 更新
- 削除
- INSERT ... SELECT
- 制約を使用可能にする（表走査がパラレル化される）
- スター型変換
- キューブ
- ロールアップ

Oracle が操作をパラレル化する方法

SELECT 文は、問合せのみで構成されています。通常、DML 文または DDL 文は、問合せ部分と DML または DDL 部分で構成されており、それぞれパラレル化できます。

注意： 通常、データ操作言語 (DML) に問合せも含まれますが、この章の「DML」は挿入、更新および削除のみを指します。

Oracle は基本的に次の方法で SQL 文をパラレル化します。

1. 走査の操作（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します。
2. パーティション表とパーティション索引に対する DDL および DML 操作のため、パーティションによってパラレル化します。
3. 非パーティション表にのみ挿入するため、パラレル実行サーバーによってパラレル化します。

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」）に分割し、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用できます。

パーティション表とパーティション索引の場合、1 つのパーティションに複数の ROWID 範囲を含めることはできますが、ROWID 範囲がパーティションにまたがることは不可能です。Oracle は、ROWID 範囲を使用してパーティション番号を送信することによって、パーティション・マップ参照を回避します。パーティション化列に対するコンパイル時述語および実行時述語は、ROWID 範囲を関係のあるパーティションに限定し、不要なパーティションが走査されないようにします（「パーティション・プルニング」）。

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同じ問合せの作業量と同じか、それ以下になります。ディスクのアクセス回数の合計はパーティション・プルニングによって減少しますが、パーティション表に対する問合せは同等にパラレル化されて実行されます。

表および索引に対する操作のうち、ブロック範囲（ROWID 範囲）によってパラレル化できるものは、次のとおりです。

- 表走査を使用した問合せ（DML および DDL 文の問合せを含む）
- パーティションの移動
- パーティションの分割
- 索引パーティションの再構築
- 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。次の場合を除き、パーティション内でさらにパラレル化することはありません。

- 問合せ。前述のようにブロック範囲によってパラレル化できます。
- コンポジット・パーティション化。パラレル化のグラニュル（最小単位）はサブパーティションです（11-17 ページの「コンポジット・パーティション化」を参照）。

パーティション表と索引に対する操作は、表または索引の各パーティションに異なるパラレル実行サーバーを割り当てることによって、パラレルで実行されます。コンパイル時述語および実行時述語は、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・プルニング）。コンパイル時述語または実行時述語が、操作を 1 つのパーティションに制限している場合、この操作はシリアルで実行されます。

パラレル操作では、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実行サーバーが使用されることもありますが（リソース制限、ヒントまたは表属性による）、各パーティションは 1 つのパラレル実行サーバーによってアクセスされます。ただし、1 つのパラレル実行サーバーから、複数のパーティションにアクセスできます。

パーティション表とパーティション索引に対する操作がパラレルに実行されるのは、複数のパーティションにアクセスする場合だけです。

パーティション表とパーティション索引に対する操作のうち、パーティションによってパラレル化できるものは次のとおりです。

- CREATE INDEX
- CREATE TABLE ... AS SELECT
- UPDATE
- DELETE
- INSERT ... SELECT
- ALTER INDEX ... REBUILD
- パーティション索引での範囲走査を使用する問合せ

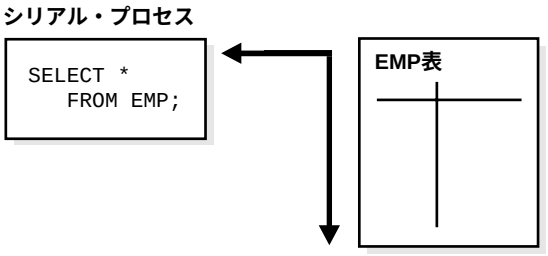
パラレル実行サーバーによるパラレル化

非パーティション表の場合のみ、Oracle は複数のパラレル実行サーバーの間で作業を分けることによって INSERT 操作をパラレル化します。新しい行には ROWID がないので、行は複数のパラレル実行サーバーの間で分配されて、空き領域に挿入されます。

パラレル実行のプロセス・アーキテクチャ

パラレル実行を使用しない場合は、SQL 文の順次実行に必要な処理すべてを 1 つのサーバー・プロセスが実行します。たとえば、全表走査（SELECT * FROM EMP など）を実行するには、図 26-1 に示すように、1 つのプロセスで操作全体を実行します。

図 26-1 シリアル全表走査



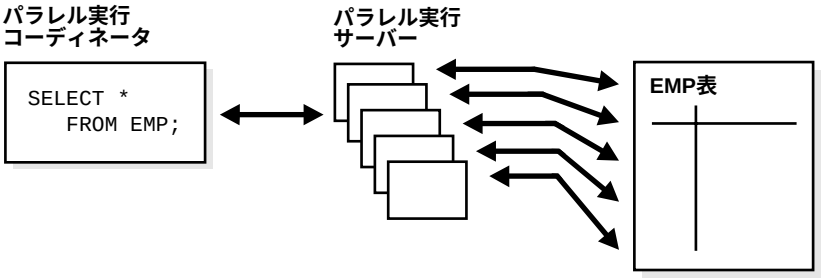
パラレル実行では、複数の「パラレル・プロセス」を使用して操作がパラレルで実行されます。「パラレル実行コーディネータ」という 1 つのプロセスが、文の実行を複数の「パラレル実行サーバー」に分配し、すべてのサーバー・プロセスからの結果を調整してユーザーに送り返します。

注意 : 「パラレル実行サーバー」という用語は、Oracle Parallel Server のプロセスを意味するのではなく、操作を他のプロセスとパラレルで実行するプロセスを意味します。（Oracle Parallel Server では、パラレル実行サーバーが複数インスタンスにまたがる場合があります。）パラレル実行サーバーは「スレーブ・プロセス」とも呼ばれます。

大量パラレル処理（MPP）構成でのパラレル実行のために操作が断片に分割される場合、Oracle はその操作に使用する表または索引の断片に対応するプロセスの「親和性」を考慮することにより、操作の特定の断片をパラレル実行サーバーに割り当てます。パーティション表および索引の物理的なレイアウトは、パラレル実行サーバーに作業を割り当てるのに使用される親和性に影響します。詳細は、26-44 ページの「親和性」を参照してください。

図 26-2 に、EMP 表がブロック範囲によって動的に分割され（「動的パーティション化」）、その表の部分的な走査を複数のパラレル実行サーバーが同時に実行する様子を示します。パラレル実行サーバーは結果をパラレル実行コーディネータに送り返し、それぞれの結果が組み立てられて、最終的な全表走査になります。

図 26-2 パラレル全表走査



パラレル実行コーディネータは、実行する機能をパラレルな断片に分割し、その後、パラレル実行サーバーによって処理された各断片（結果）を再び統合します。1 つの操作に割り当てられたパラレル実行サーバーの数が、その操作の「並行度」（DOP）となります。同じ SQL 文中の複数の操作の並行度は、すべて同じです（26-15 ページの「Oracle による操作の並行度の決定方法」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、どのパラレル操作にも使用可能なパラレル実行サーバーのプールが作成されます。初期化パラメータ PARALLEL_MIN_SERVERS は、インスタンスの起動時に作成されるパラレル実行サーバーの数を指定します。

パラレル操作の実行中、パラレル実行コーディネータはプールからパラレル実行サーバーを取得して操作に割り当てます。必要であれば、Oracle はその操作のために追加のパラレル実行サーバーを作成できます。それらのパラレル実行サーバーは、ジョブの実行中は 1 つの操作に対応付けられたままになり、その後、他の操作に使用できるようになります。文が完全に処理された後、パラレル実行サーバーはプールに戻ります。

注意： パラレル実行コーディネータとパラレル実行サーバーは、一度に 1 つの文にしか作用しません。パラレル実行コーディネータは、たとえばパラレル問合せとパラレル DML 文を同時には調整できません。

ユーザーが SQL 文を発行すると、オプティマイザは操作をパラレルで実行するかどうかを決定し、各操作の並行度を決定します。操作に必要なパラレル実行サーバーの数は、さまざまな方法で指定できます（26-14 ページの「並行度の設定」を参照）。

オプティマイザがパラレル処理のために文を処理する場合、次のことが順に行われます。

- SQL 文のフォアグラウンド・プロセスがパラレル実行コーディネータになる。

- パラレル実行コーディネータが、サーバー・プールから必要な数のパラレル実行サーバー（並行度によって判断される）を取得する。
- Oracle が、一連の操作として文を実行する。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、さらにパラレル実行サーバーをサーバー・プールに戻す。

パラレル実行コーディネータは、SQL 文の（解析時ではなく）実行時にパラレル実行サーバーをコールします。したがって、マルチスレッド・サーバーでパラレル実行を使用する場合には、ユーザーの文の EXECUTE コールを処理するサーバー・プロセスがその文のパラレル実行コーディネータになります。

パラレル実行サーバーの数の変動

1 つのインスタンスによって同時に処理されるパラレル操作の数が大幅に変動する場合、Oracle はプール中のパラレル実行サーバーの数を自動的に変更します。

パラレル操作の数が多くなると、入ってくる要求を処理するために追加のパラレル実行サーバーが作成されます。ただし、Oracle は、初期化パラメータ PARALLEL_MAX_SERVERS で指定されたより多数のパラレル実行サーバーをインスタンス用に作成することはありません。

パラレル操作の数が減少すると、Oracle はしきい値で指定された時間だけアイドル状態になっていたパラレル実行サーバーを終了させます。パラレル実行サーバーがアイドル状態になっていた時間がいくら長くても、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。

十分なパラレル実行サーバーがない場合の処理

Oracle は、要求された数より少ないプロセスでパラレル操作を実行できます。初期化パラメータ PARALLEL_MIN_PERCENT で最小値を指定する方法の詳細は、26-17 ページの「パラレル実行サーバーの最小数」を参照してください。

プール中のすべてのパラレル実行サーバーが占有されており、最大数のパラレル実行サーバーがすでに開始されている場合、パラレル実行コーディネータはシリアル処理に切り替わります。

> **追加情報：** パラレル実行サーバーのインスタンスのプールの監視、および初期化パラメータの適切な値の決定の詳細は、『Oracle8*i* チューニング』を参照してください。

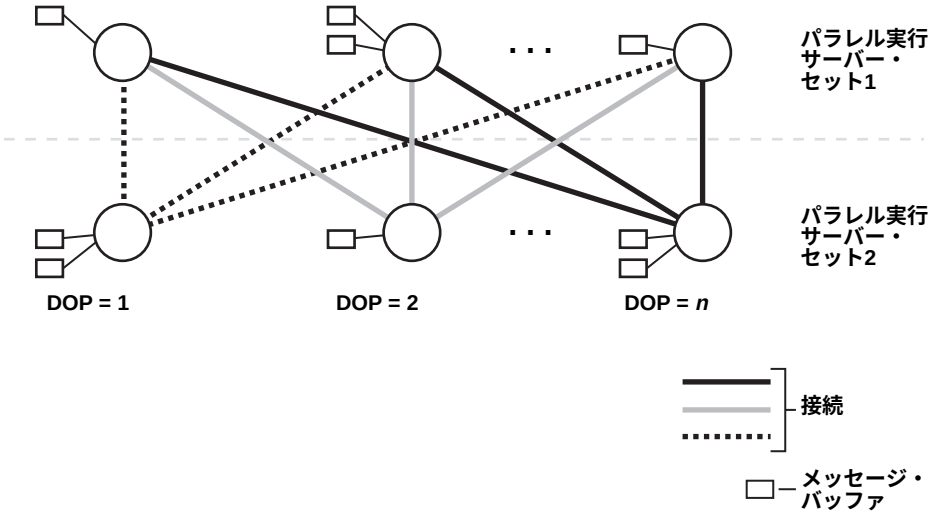
パラレル実行サーバーの通信方法

問合せをパラレルで実行するために、通常、Oracle はプロデューサ・キュー・サーバーとコンシューマ・サーバーを作成します。プロデューサ・キュー・サーバーは表から行を取り出し、コンシューマ・サーバーはこれらの行に対して操作（結合、ソート、DML、DDL など）を実行します。プロデューサ実行プロセス・セット内の各サーバーは、コンシューマ・セット内の各サーバーに接続します。これは、パラレル実行サーバー間の仮想接続数が、並行度の 2 乗の倍率で増加することを意味します。

各通信チャネルには、1 ～ 4 個のメモリー・バッファがあります。複数のメモリー・バッファがあれば、パラレル実行サーバー相互の非同期通信が容易になります。

単一インスタンス環境では、通信チャネルごとに最大 3 個のバッファが使用されます。OPS 環境では、チャネルあたり最大 4 個のバッファが使用されます。図 26-3 に、メッセージ・バッファと、プロデューサ・パラレル実行サーバーがコンシューマ・パラレル実行サーバーに接続する方法を示します。

図 26-3 パラレル実行サーバーの接続とバッファ



同じインスタンス内の 2 つのプロセス間に接続があれば、サーバーはバッファをやりとりして通信します。異なるインスタンス内のプロセス間に接続があれば、外部の高速ネットワーク・プロトコルを使用してメッセージが送信されます。図 26-3 では、DOP はパラレル実行サーバー数（この場合は "n"）と同数です。図 26-3 には、パラレル実行コーディネータは示されていません。各パラレル実行サーバーは、実際にはパラレル実行コーディネータへの追加の接続を持っています。

SQL 文のパラレル化

各 SQL 文ごとに、解析時に最適化プロセスおよびパラレル化プロセスが実行されます。そのため、データが変更されて、さらに最適な実行計画やパラレル化計画が使用可能になると、Oracle はその新しい状況に自動的に対応できます（最適化の詳細は、第 22 章「オプティマイザ」を参照）。

オプティマイザが文の実行計画を決定した後で、パラレル実行コーディネータは、計画に含まれる各操作のパラレル化の方法を決定します（たとえば、ブロック範囲によ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スキャンをパラレル化します）。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実行サーバーの数（つまり「並行度」）を決定する必要があります。

詳細は、26-14 ページの「並行度の設定」および 26-18 ページの「SQL 文のパラレル化ルール」を参照してください。

複数のパラレル実行サーバー間での作業の分割

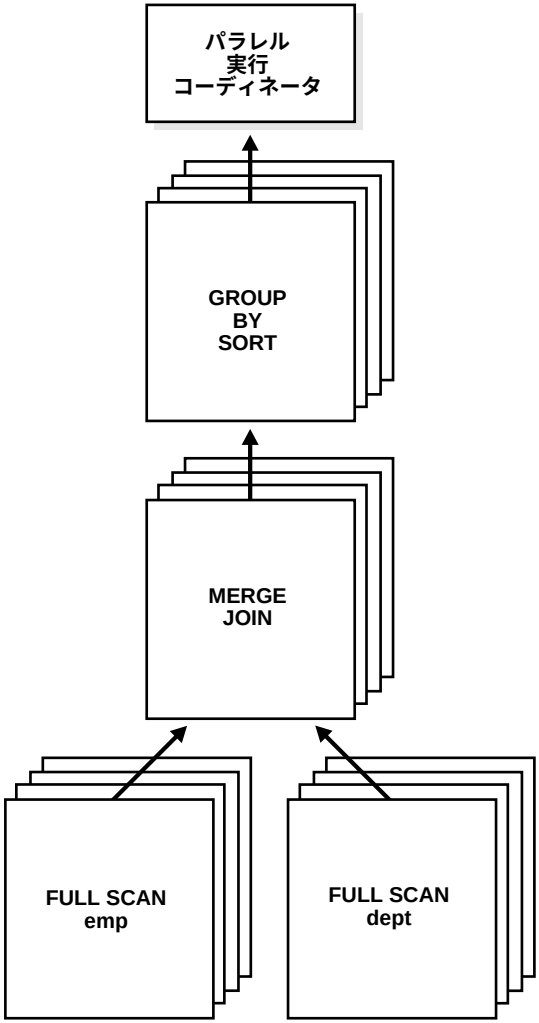
パラレル実行コーディネータは、各操作の再分散化要件を調べます。操作の「再分散化要件」とは、操作によって処理する行を複数のパラレル実行サーバーの間で分配するための方法です。

実行計画の中の各操作の再分散化要件を決定した後、オプ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、オプティマイザは文のデータ・フローを決定できます。

図 26-4 に、次の問合せのデータ・フローを示します。

```
SELECT dname, MAX(sal), AVG(sal)
  FROM emp, dept
    WHERE emp.deptno = dept.deptno
    GROUP BY dname;
```

図 26-4 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 26-4 では、GROUP BY SORT 操作が MERGE JOIN 操作の出力を必要とするため、GROUP BY SORT 操作は MERGE JOIN 操作の親操作です。

子操作によって行が生成されると、親操作はすぐにその行の使用を開始できます。前の例では、パラレル実行サーバーが FULL SCAN DEPT 操作で行を生成している間に、別の一連のパラレル実行サーバーのセットが MERGE JOIN 操作の実行を開始してその行を使用します。

同時に実行される前述の 2 つの操作には、それぞれ専用のパラレル実行サーバーの集合が用意されます。したがって、問合せ操作と、データ・フロー・ツリーそのものの両方に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 の操作にはプロデューサ（作成者）およびコンシューマ（使用者）という特性があるため、特定のツリーに含まれる操作のうち、実行時間を最小にするために同時に実行する必要がある操作は 2 つだけです。

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、その後に取得した行の ENAME 列の値に基づいたソート操作で構成されています。この例では、ENAME 列には索引が作成されていないものとします。また、この問合せの並行度は 4 に設定されているとします。つまり、それぞれの操作について 4 つのパラレル実行サーバーをアクティブにできることになります。

図 26-5 に、この例の問合せのパラレル実行を示します。

図 26-5 インターオペレーション並行性と動的パーティション化

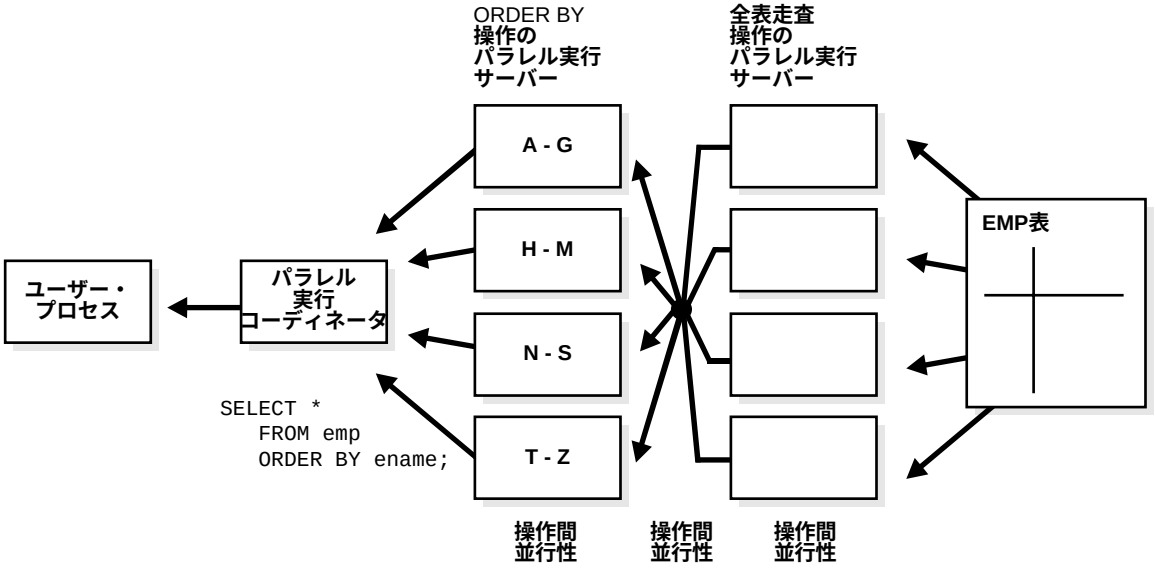


図 26-5 からわかるとおり、並行度は 4 ですが、問合せに実際に関与しているのは 8 つのパラレル実行サーバーです。これは、親オペレータと子オペレータを同時に実行できるためです（インターオペレーション並行性）。

走査操作に含まれるすべてのパラレル実行サーバーが、ソート操作を実行するパラレル実行サーバーのうちの適切なプロセスに行を送っていることにも注目してください。パラレル実行サーバーによって走査された行の ENAME 列の内容が A ～ G の間の値であれば、その行は最初の ORDER BY パラレル実行サーバーへ送られます。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： 一連のパラレル実行サーバーの操作が完了すると、データ・フローの中で次の順位にある操作に移ります。たとえば、前述の図で、ORDER BY の後にもう 1 つ別の ORDER BY 操作が続いている場合は、表走査を実行しているパラレル実行サーバーが、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

パラレル実行コーディネータは、1 つの SQL 文の処理のために、そのインスタンスのパラレル実行サーバーを 2 つ以上登録することがあります。1 つの操作に関連付けられたパラレル実行サーバーの数を「並行度」（DOP）といいます。

並行度は、問合せレベル（ヒントまたは PARALLEL 句を使用して）、表または索引レベル（表または索引の定義を使用して）、または CPU の数に基づくデフォルトによって指定します。

次の例は、表に対する並行度を 4 に設定する文を示しています。

```
ALTER TABLE emp PARALLEL 4;
```

次の例では、索引に対する並行度を設定しています。

```
ALTER INDEX iemp PARALLEL;
```

最後の例では、問合せのヒントを 4 に設定しています。

```
SELECT /*+ PARALLEL(emp,4) */ COUNT(*) FROM emp ;
```

追加情報： これらの文の構文は、『Oracle8*i* リファレンス・マニュアル』および『Oracle8*i* チューニング』を参照してください。

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のみです。インターオペレーション並行性が可能であれば、パラレル実行サーバーの合計数は、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。

パラレル実行は、複数の CPU とディスクを効率的に使用して問合せに迅速に応答するように設計されています。複数のユーザーが同時にパラレル実行を使用すると、使用可能な CPU、メモリーおよびディスク・リソースは急速に使い果たされます。Oracle には、次のように、パラレル実行に伴うリソース使用状況を取り扱うために、複数の方法が用意されています。

- 適応マルチユーザー・アルゴリズム。システムにかかる負荷が大きくなるにつれて、並行度を低下させます。このオプションをオンにするには、初期化ファイル内で ALTER SYSTEM 文の PARALLEL_ADAPTIVE_MULTI_USER パラメータを使用します。
- ユーザーのリソース制限とプロファイル。ユーザーのセキュリティ・ドメインの一部として、各ユーザーが使用できる各種のシステム・リソースの容量に制限を設定できます。詳細は、29-15 ページの「ユーザー・リソースの制限とプロファイル」を参照してください。
- データベース・リソース・マネージャ。さまざまなユーザー・グループにリソースを割当てできます。詳細は、第 9 章「データベース・リソースの管理」を参照してください。

追加情報： ALTER SYSTEM SQL 文の構文は、『Oracle8*i* SQL リファレンス』を参照してください。

Oracle による操作の並行度の決定方法

パラレル実行コーディネータは、いくつかの指定を考慮して並行度を決定します。その手順は、次のとおりです。

1. SQL 文そのもので指定されたヒントまたは PARALLEL 句をチェックします。
2. 表または索引の定義を調べます。
3. デフォルトの並行度をチェックします（26-16 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、26-18 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、PARALLEL 句、表または索引定義およびデフォルト値は、特定の操作に関してコーディネータが要求するパラレル実行サーバーの数を決定するだけです。実際に使用されるパラレル実行サーバーの数は、パラレル・サーバー・プール中で使用可能なプロセス数（26-7 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（26-11 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できます。

- PARALLEL ヒントは、表に対する操作にのみ使用される。このヒントを使用して、問合せおよび DML 文（INSERT、UPDATE および DELETE）を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走査をパラレル化する。（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方法の概要と、PARALLEL、NOPARALLEL、PARALLEL_INDEX、CACHE および NOCACHE の各ヒントの明確な構文は、『Oracle8*i* 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、ALTER TABLE、CREATE INDEX または ALTER INDEX のうちの 1 つを使用します。

追加情報： SQL 文の完全な構文は、『Oracle8*i* 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

操作のパラレル化を要求しながら、ヒントにも、表または索引の定義にも並行度を指定しなければ、デフォルトの並行度が使用されます。ほとんどのアプリケーションには、デフォルトの並行度が適しています。

> **追加情報：** 並行度の調整の詳細は、『Oracle8*i* 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定されます。

1. システム内のすべての Oracle Parallel Server インスタンス用の CPU 数と、パラメータ PARALLEL_THREADS_PER_CPU の値。
2. パーティションによるパラレル化の場合、パーティション・プルニングに基づく、アクセスされるパーティションの数（概算の場合）。
3. グローバルな索引メンテナンスでのパラレル DML 操作の場合、更新するすべてのグローバル索引間でのトランザクション空きリストの最小数。パーティション化されたグローバル索引のためのトランザクション空きリストの最小数は、索引パーティションすべてにわたる最小数です。これは、自己デッドロックを防ぐための要件となります。

> **注意：** Oracle は、CPU 情報をオペレーティング・システムから取得します。

前述の要素により、使用されるパラレル実行サーバーのデフォルト数が決定されます。ただし、実際に使用されるパラレル実行サーバーの数は、要求したインスタンス上で実行時にそれらがどの程度使用可能かによって制限されます。1 つのインスタンスに許されるパラレル実行サーバーの合計数の上限は、初期化パラメータ PARALLEL_MAX_SERVERS によって設定されます。

必要なパラレル実行サーバーの最小数（初期化パラメータ PARALLEL_MIN_PERCENT で指定）が使用できなければ、ユーザー・エラーが発生します。その場合、ユーザーは並行性を低くして問合せを再試行できます。

適応マルチユーザー・アルゴリズム

適応マルチユーザー・アルゴリズムが使用可能になっている場合、パラレル実行コーディネータはシステム負荷に応じて並行度を変更します。負荷は、データベース・リソース・マネージャによって計算された、割当て済みスレッド数を検査することによって決定されます。使用可能な CPU 数について、現在の割当て済みスレッド数が最適なスレッドを超えていれば、このアルゴリズムは並行度を下げます。これにより、リソースの過剰割当てが回避され、スループットが改善されます。

パラレル実行サーバーの最小数

最低 2 つのパラレル実行サーバーが使用可能であれば、パラレル操作を実行できます。使用可能なパラレル実行サーバーが少なすぎると、SQL 文の実行に予想以上の時間がかかる場合があります。要求されたパラレル実行サーバーのうち、一定の割合以上が使用可能になっていなければ操作を実行しないようにも指定できます。これにより、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、使用可能なものの割合が最小割合未満の場合は、SQL 文は実行されず、エラーが戻されます。

要求されたパラレル実行サーバーに必要な最小パーセンテージは、初期化パラメータ PARALLEL_MIN_PERCENT に指定します。このパラメータは、問合せだけでなく 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実行サーバーの少なくとも 50% が使用可能である必要があります。20 個のパラレル実行サーバーが要求されると、少なくとも 10 個のサーバーが使用可能でなければ、ユーザーにエラーが戻されます。PARALLEL_MIN_PERCENT を NULL に設定すると、処理のために使用できるパラレル実行サーバーが少なくとも 2 つあれば、すべてのパラレル操作が続行されます。

使用可能なインスタンス数の制限

Oracle Parallel Server においては、インスタンス・グループを使用して、パラレル操作に関与するインスタンスの数を制限できます。それぞれが 1 つ以上のインスタンスからなるインスタンス・グループを必要な数だけ作成できます。その後、パラレル操作の一部またはすべてに対して、どのインスタンス・グループを使用するかを指定できます。パラレル実行サーバーは、指定されたインスタンス・グループのメンバーであるインスタンスにのみ使用されます。

> **追加情報 :** インスタンス・グループの詳細は、『Oracle8*i* 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、すべてのパラレル実行サーバーに同等の作業負荷がかかるようにする必要があります。ブロック範囲またはパラレル実行サーバーによりパラレル化される SQL 文の場合、作業負荷はパラレル実行サーバーの間で動的に分割されます。これにより、一部のパラレル実行サーバーが他のプロセスよりも大量の作業を実行する場合に発生する「作業負荷の不整」を最小限に抑えることができます。

パーティションによりパラレル化される SQL 文の場合、作業負荷が各パーティションに均等に分散していれば、パラレル実行サーバーの数とパーティションの数を一致させるか、またはパーティションの数がプロセスの数の倍数になるように並行度を選択して、パフォーマンスを最適化できます。

たとえば、1 つの表に 10 個のパーティションがあり、パラレル操作によって作業が各パーティションに均等に分配される場合、10 個のパラレル実行サーバーを使用する（並行度 = 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 の時間で作業を完了できます。5 つのプロセスを使用すれば 5 分の 1 の時間、2 つのプロセスを使用すれば 2 分の 1 の時間で作業を完了できます。

ただし、9 個のプロセスを使用して 10 個のパーティションに対する作業を実行すると、1 つのパーティションで作業を終了した最初のプロセスが、10 番目のパーティションで作業を開始します。その他のプロセスは、作業を終了した後にアイドル状態になります。この場合、各パーティションに均等に作業が分散されていると、パフォーマンスはあまり改善されません。作業の分け方が均等でない場合、最後に残されたパーティションの作業量が他のパーティションと比べて多いか少ないかによってパフォーマンスが異なります。

同じように、4 個のプロセスを使用して 10 個のパーティションに対する作業を実行し、かつその作業が均等に分けられている場合、各プロセスは最初のパーティションの処理を終了した後に 2 番目のパーティションを処理します。そしてプロセスのうち 2 つは 3 番目のパーティションを処理しますが、残りの 2 つのプロセスはアイドル状態になります。

一般に、P 個のパラレル実行サーバーを使用して N 個のパーティションに対してパラレル操作を実行する場合の時間は、必ずしも N / P ではありません。一部のプロセスは、他のプロセスが最後のパーティションの作業を終了するまで待機させられる場合もあるためです。ただし、並行度を適切に選択すれば、作業負荷の不整を最小限に抑えてパフォーマンスを最適化できます。

ディスク親和性がある場合に作業負荷のバランスを調整する方法の詳細は、26-45 ページの「親和性とパラレル DML」を参照してください。

SQL 文のパラレル化ルール

SQL 文にパラレル・ヒントが含まれていたり、操作の対象となる表または索引が CREATE 文か ALTER 文で PARALLEL と宣言されている場合は、SQL 文をパラレル化できます。さらに、データ定義言語（DDL）文は、PARALLEL 句を使用してパラレル化できます。ただし、これらの方法すべてがどのタイプの SQL 文にも適用されるとは限りません。

パラレル化には、パラレル化するかどうかの決定および並行度という 2 つの要素が関係しています。これらの要素は、問合せ、DDL 操作および DML 操作について、それぞれ異なる方法で判断されます。

並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各表および索引を調べて、どれが参照表であるかを判断する。並行度が最大の表または索引を選択するのが基本的なルールです。
- パラレル DML（挿入、更新および削除）の場合、並行度を決定する参照オブジェクトは、挿入、更新または削除操作によって変更される表である。パラレル DML では、デッドロックを防ぐため、並行度にある程度の制限も追加されます。パラレル DML 文に副問合せが含まれる場合、その副問合せの並行度は 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、再構築、分割または移動の対象となる表、索引またはパーティションである。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またがる索引範囲走査

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となるすべての索引（「参照オブジェクト」）から取得した並行度の最大値を使用します。つまり、最大の並行度を持つ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ます。

UPDATE と DELETE のパラレル化のルール

更新操作と削除操作は、パーティション（またはサブパーティション、11-17 ページの「コンポジット・パーティション化」を参照）によってパラレル化されます。パラレル化できるのは、パーティション表に対する更新と削除のみです。1 つのパーティション内で、また非パーティション表上で、更新または削除のパラレル化はできません。

UPDATE 操作と DELETE 操作にパラレル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 句を指定します（参照オブジェクト）。
2. 更新パラレル・ヒントまたは削除パラレル・ヒントを文に指定します。

パラレル・ヒントは、UPDATE 文および DELETE 文の UPDATE または DELETE キーワードのすぐ後に指定します。また、ヒントは、変更する表に対する基礎的な走査にも適用されます。

CREATE TABLE コマンドと ALTER TABLE コマンドの PARALLEL 句によって、表のパラレル化が指定されます。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 文の並行性も決定されます。ただし、DML 文に表のための明示的なパラレル・ヒントが含まれていれば、その表の PARALLEL 句による指定はパラレル・ヒントによって上書きされます。

ALTER SESSION FORCE PARALLEL DML 文を使用すると、セッションの後続の UPDATE および DELETE 文の PARALLEL 句を上書きできます。UPDATE 文や DELETE 文に PARALLEL ヒントが含まれている場合、ALTER SESSION FORCE PARALLEL DML 文が上書きされます。

パラレル化の決定 UPDATE または DELETE 文で更新または削除操作をパラレル化するかどうかは、次の規則によって決まります。

- UPDATE または DELETE 操作は、次の条件の少なくとも 1 つが当てはまる場合にのみパラレル化されます
  - 更新または削除される表に PARALLEL 仕様部がある。
  - DML 文に PARALLEL ヒントが指定されている。
  - このセッションで、前に ALTER SESSION FORCE PARALLEL DML 文が発行されている。

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、それらのパラレル化命令は更新または削除をパラレル化するかどうかの決定には影響を与えません。

表に対する PARALLEL ヒントまたは PARALLEL 句は、問合せ部分と更新または削除部分の両方についてパラレル化するかどうかの決定に使用されます。ただし、更新または削除部分をパラレル化するかどうかは、問合せ部分とは関係なく決定され、その逆も同様です。

並行度 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。

更新または削除操作の並行度を決定する優先順位の規則では、更新または削除のパラレル・ヒント指定が、ターゲット表のパラレル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（または、コンポジット・サブパーティションの場合はサブパーティション数）と同じです。1 つのパラレル実行サーバーから複数のパーティションのデータを更新したり削除できますが、各パーティションを更新または削除できるのは 1 つのパラレル実行サーバーからのみです。

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、超過分のパラレル実行サーバーは何も作業を実行しません。

例 1: `UPDATE tbl_1 SET c1=c1+1 WHERE c1>100;`

TBL_1 がパーティション表で、その表定義に PARALLEL 句が含まれている場合、表の走査が順次走査である（索引走査など）としても、更新操作はパラレル化されます（この表に C1 が 100 を超えるパーティションが 2 つ以上あるものとします）。

例 2: `UPDATE /*+ PARALLEL(tbl_2,4) */ tbl_2 SET c1=c1+1;`

TBL_2 に対する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 モードが使用可能になっている場合）。

1. その文に SELECT パラレル・ヒント（複数可）を指定します。
2. 選択対象の表の定義に PARALLEL 句を指定します。
3. その文に INSERT パラレル・ヒントを指定します。
4. 挿入先の表の定義に PARALLEL 句を指定します。

ALTER SESSION FORCE PARALLEL DML 文を使用すると、セッションの後続の挿入操作の PARALLEL 句を上書きできます。挿入操作にパラレル・ヒントが含まれている場合、ALTER SESSION FORCE PARALLEL DML 文が上書きされます。

パラレル化の決定 INSERT... SELECT 文で挿入操作をパラレル化するかどうかは、次の規則によって決まります。

- INSERT は、次の条件の少なくとも 1 つが当てはまる場合にのみパラレル化されます。
  - DML 文で INSERT の後に PARALLEL ヒントが指定されている場合。
  - 挿入する表（参照オブジェクト）に PARALLEL 宣言仕様部がある場合。
  - このセッションで、前に ALTER SESSION FORCE PARALLEL DML 文が発行されている場合。

このように、挿入操作をパラレル化するかどうかの決定は、選択操作には依存せず、その逆も同様です。

並行度 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ると、次の優先順位ルールを使用して、文全体の「並行度」を決定するためのパラレル命令が 1 つ選択されます。

挿入ヒント命令 > 挿入先となる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、複数の表と索引のうち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 : 次の例で使用されている並行度は 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 文のパラレル化のルール

パラレル化の決定 DDL 操作は、構文中に PARALLEL 句（「宣言」）が指定されている場合にパラレル化できます。CREATE INDEX および ALTER INDEX ...REBUILD または ALTER INDEX ... REBUILD PARTITION の場合、PARALLEL 宣言はデータ・ディクショナリに格納されます。

ALTER SESSION FORCE PARALLEL DDL 文を使用すると、セッション中の後続の DDL 文の PARALLEL 句を上書きできます。

並行度 並行度は、ALTER SESSION FORCE PARALLEL DDL 文で上書きされない限り、PARALLEL 句の仕様部によって決まります。パーティション索引の再構築がパラレル化されることはありません。

索引の作成、索引の再構築およびパーティションのマージ / 分割のパラレル化のルール

CREATE INDEX または ALTER INDEX ... REBUILD のパラレル化 CREATE INDEX および ALTER INDEX ... REBUILD 文は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でのみパラレル化できます。

ALTER INDEX ... REBUILD は、非パーティション索引の場合しかパラレル化できませんが、ALTER INDEX ... REBUILD PARTITION は PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL によってパラレル化できます。

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、並行度も同じです。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 のパラレル化 ALTER INDEX ... MOVE PARTITION および ALTER INDEX ... SPLIT PARTITION 文は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でしかパラレル化できません。これらの文の走査操作の並行性は、対応する MOVE または 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 のパラレル化のルール

CREATE TABLE ... AS SELECT 文には、CREATE 部（DDL）および SELECT 部（問合せ）という 2 つの部分が含まれています。Oracle では、文の中のこれら 2 つの部分を両方ともパラレル化できます。CREATE 部は、その他の DDL 操作と同じルールに従います。

パラレル化するかどうかの決定（問合せ部） CREATE TABLE ... AS SELECT 文の問合せ部をパラレル化できるのは、次の条件が満たされる場合のみです。

1. 問合せに PARALLEL ヒント指定（PARALLEL または PARALLEL_INDEX）が含まれている場合、その文の CREATE 部に PARALLEL 句が指定されている場合、または問合せで参照されているスキーマ・オブジェクトに PARALLEL 宣言が対応付けられている場合。
2. 問合せの中で指定されている表の少なくとも 1 つで、次のどちらかが必要。
   - 全表走査
   - 複数のパーティションにまたがる索引範囲走査

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用します。
2. PARALLEL 句が指定されていない場合のデフォルトの並行度は CPU 数です。
3. CREATE がシリアルの場合、並行度は問合せによって決定されます。26-19 ページの「問合せのパラレル化のルール」を参照してください。

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（作成部） CREATE TABLE ... AS SELECT の CREATE 操作は、PARALLEL 句または ALTER SESSION FORCE PARALLEL DDL 文でのみパラレル化できます。

CREATE TABLE ... AS SELECT の CREATE 操作がパラレル化されると、可能であれば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場合は、SELECT 操作をパラレル化できます。

並行度（作成部）CREATE 操作の並行度と SELECT 操作の並行度（パラレル化される場合）は、ALTER SESSION FORCE PARALLEL DDL 文で上書きされない限り、CREATE 文の PARALLEL 句によって指定します。PARALLEL 句に並行度を指定しない場合のデフォルト値は CPU 数です。

パラレル化ルールのまとめ

表 26-1 に、さまざまなタイプの SQL 文をパラレル化する方法と、並行性を指定するときの各方法の優先順位を示します。

- 優先順位（1）の指定は、優先順位（2）と優先順位（3）の指定に優先する。
- 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*i* SQL リファレンス』を参照してください。

表 26-1 パラレル化ルール

パラレル操作	句、ヒント、または基礎となる表 / 索引宣言によるパラレル化（優先順位 : 1、2、3）			
	パラレル・ヒント	パラレル句	ALTER SESSION	パラレル宣言
パラレル問合せ表走査（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（パーティション表のみ）	(1) PARALLEL		(2) FORCE PARALLEL DML	(3) 更新または削除の対象となる表の宣言
パラレル INSERT... SELECT の **INSERT 操作**（パーティション表または非パーティション表）	(1) INSERT 部の PARALLEL		(2) FORCE PARALLEL DML	(3) 挿入先の表の宣言
INSERT がパラレルのときの INSERT ... SELECT の **SELECT 操作**	INSERT 文からの並行度			
INSERT がシリアルのときの INSERT ... SELECT の **SELECT 操作**	(1) PARALLEL			(2) 選択先の表の宣言

表 26-1 パラレル化ルール （続き）

パラレル操作	句、ヒント、または基礎となる表 / 索引宣言によるパラレル化（優先順位 : 1、2、3）			
	パラレル・ヒント	パラレル句	ALTER SESSION	パラレル宣言
パラレル CREATE TABLE... AS SELECT の **CREATE 操作**（パーティション表または非パーティション表）	(**注意 :** SELECT 句のヒントは CREATE 操作には影響しない。）	(2)	(1) FORCE PARALLEL DDL	
CREATE がパラレルのときの CREATE TABLE ... AS SELECT の **SELECT 操作**	CREATE 文からの並行度			
CREATE がシリアルのときの CREATE TABLE ... AS SELECT の **SELECT 操作**	(1) PARALLEL または PARALLEL_INDEX			(2) 問い合せする表またはパーティション索引の宣言
パラレル CREATE INDEX（パーティション表または非パーティション表）		(2)	(1) FORCE PARALLEL DDL	
パラレル REBUILD INDEX（非パーティション索引）		(2)	(1) FORCE PARALLEL DDL	
REBUILD INDEX（パーティション索引）— パラレル化されない			—	—
パラレル REBUILD INDEX パーティション		(2)	(1) FORCE PARALLEL DDL	
パーティションのパラレル MOVE / SPLIT		(2)	(1) FORCE PARALLEL DDL	

パラレル問合せ

問合せと副問合せは、SELECT 文だけでなく、DDL 文と DML 文（INSERT、UPDATE および DELETE）の問合せ部でパラレル化できます。問合せをパラレル化する方法については、これまでの項で説明しました。

- 26-2 ページの「パラレル化できる操作」。Oracle がパラレル化できる問合せのリストが記載されています。
- 26-3 ページの「Oracle が操作をパラレル化する方法」。ROWID 範囲による動的パラレル化について説明しています。
- 26-5 ページの「パラレル実行のプロセス・アーキテクチャ」。パラレル問合せを実行するプロセスについて説明しています。

- 26-10 ページの「SQL 文のパラレル化」。プロセスによるパラレル問合せの実行方法について説明しています。
- 26-19 ページの「問合せのパラレル化のルール」。問合せのパラレル化に関する条件と、並行度を決定する要因について説明しています。

ただし、リモート・オブジェクトを参照している DDL 文や DML 文の問合せ部は、パラレル化できません。問合せ部でリモート・オブジェクトを参照するパラレルの DML 文または DDL 文を発行すると、操作は通知なしにシリアルで実行されます。例は、26-42 ページの「分散トランザクションの制限事項」を参照してください。

索引構成表のパラレル問合せ

索引構成表の場合は、次のパラレル走査方法がサポートされています。

- 非パーティション索引構成表のパラレル高速全走査
- パーティション索引構成表のパラレル高速全走査
- パーティション索引構成表のパラレル索引範囲走査

これらの走査方法は、オーバーフロー領域を持つ索引構成表と、LOB を含む索引構成表に使用できます。

非パーティション索引構成表

非パーティション索引構成表のパラレル問合せには、パラレル高速全走査を使用します。並行度は、優先順位の低い順から PARALLEL ヒント（存在する場合）または表に対応する並行度（CREATE TABLE または ALTER TABLE コマンドで指定した場合）によって決定されます。

作業の割当ては、索引セグメントを適切な数のブロック範囲に分割し、各ブロック範囲を必要に応じてパラレル実行サーバーに割り当てることによって行われます。いずれかの行に対応するオーバーフロー・ブロックには、その行を所有するプロセスのみが必要に応じてアクセスします。

パーティション索引構成表

索引範囲走査と高速全走査の両方をパラレルで実行できます。パラレル高速全走査のパラレル化は、非パーティション索引構成表の場合とまったく同じです。パーティション索引構成表に対するパラレル索引範囲走査の場合、並行度は（パラレル高速全走査の場合と同様に）前述の優先順位リストから選択された最小並行度と、索引構成表のパーティション数です。並行度に応じて、各パラレル実行サーバーは 1 つ以上の（必要に応じて割り当てられる）パーティションを取得します。各パーティションには、主キー索引セグメントおよび対応するオーバーフロー・セグメント（存在する場合）が含まれています。

オブジェクト型のパラレル問合せ

オブジェクト型の表とオブジェクト型の列を含む表に対して、パラレル問合せを実行できます。オブジェクト型のパラレル問合せでは、次のようにオブジェクト型の順次問合せに使用可能な機能をすべてサポートしています。

- オブジェクト型のメソッド
- オブジェクト型の属性アクセス
- オブジェクト型のインスタンスを作成するコンストラクタ
- オブジェクト・ビュー
- オブジェクト型に対する PL/SQL および OCI 問合せ

パラレル問合せには、オブジェクト型のサイズ制限はありません。

オブジェクト型に対するパラレル問合せの使用には、次の制限が適用されます。

1. 結合とソート（ORDER BY、GROUP BY または集合演算による）を伴う問合せをパラレル化するには、MAP 関数が必要です。MAP 関数がなければ、問合せは自動的にシリアルで実行されます。
2. ネストした表のパラレル問合せはサポートされません。表またはパラレル・ヒントにパラレル属性があっても、問合せはシリアルで実行されます。
3. オブジェクト型では、パラレル DML とパラレル DDL はサポートされません。DML 文と DDL 文は常にシリアルで実行されます。

これらの条件が 1 つでも当てはまるために問合せをパラレルで実行できない場合は、問合せ全体がシリアルで実行され、エラー・メッセージは表示されません。

パラレル DDL

この項では、データ定義言語（DDL）文の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

パーティション化されているかどうかに関係なく、表と索引に対する DDL 文をパラレルかできます。DDL 文でパラレル化できる操作のまとめは、26-24 ページの表 26-1 を参照してください。

非パーティション表および索引のパラレル DDL 文は次のとおりです。

- CREATE INDEX
- CREATE TABLE ...AS SELECT
- ALTER INDEX ... REBUILD

パーティション表とパーティション索引のパラレル DDL 文は次のとおりです。

- CREATE INDEX
- CREATE TABLE ...AS SELECT
- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER TABLE ... COALESCE PARTITION
- ALTER INDEX ...REBUILD PARTITION
- ALTER INDEX ... SPLIT PARTITION— 分割する（グローバル）索引パーティションが使用可能な場合のみ

これらのすべての DDL 操作は、パラレル実行でもシリアル実行でも、非ロギング・モード（25-5 ページの「ロギング・モード」を参照）で実行できます。

索引構成表の CREATE TABLE は、AS SELECT 句を使用してもしなくてもパラレル化できます。

操作ごとに異なる並行性が使用されます（26-24 ページの表 26-1 を参照）。パーティション表のパラレル CREATE TABLE AS SELECT およびパーティション索引のパラレル CREATE INDEX は、パーティション数と同じ並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・セッションを使用して構築できるため、ANALYZE {TABLE, INDEX} PARTITION コマンドでパーティションのパラレル分析表を作成する必要はほとんどありません。

パラレル DDL は、オブジェクト列または LOB 列を含む表には使用できません。

> **追加情報：** パラレル DDL 文の構文と使用方法の詳細は、『Oracle8*i* 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

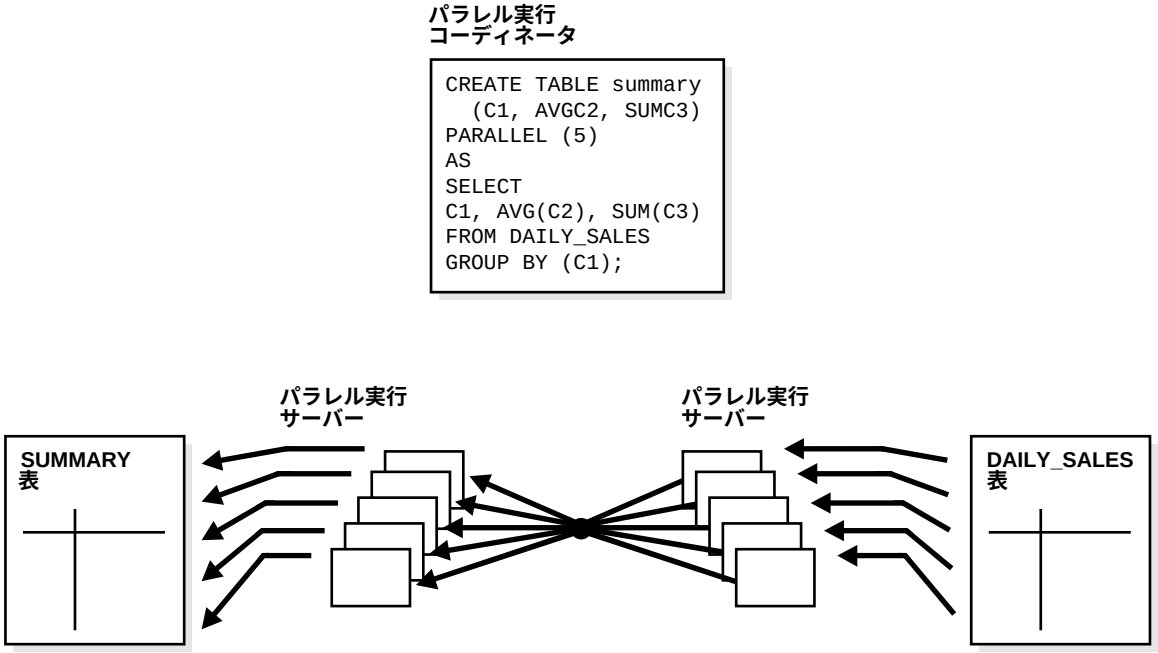
意思決定支援アプリケーションでは、大量のデータを非定型の意思決定支援問合せで使用するために、パフォーマンス上の理由でそれらのデータをより小さな表に「ロールアップ」する必要がある場合があります。このロールアップは、定期的に（毎晩または毎週など）、システムがアクティブでない短い期間に実行されます。

パラレル実行を使用すると、別の表または表の集合からの副問合せとして表を作成するための、問合せ操作と作成操作をパラレル化できます。

図 26-6 に、副問合せからパラレルで表を作成する様子を示します。

注意： クラスタ化表は、パラレルでは作成も挿入もできません。

図 26-6 パラレルでのサマリー表の作成



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、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）中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、表が作成された後で、その表を含む表領域のバックアップを作成してください。

UNDO ログと REDO ログの生成をオフにするには、CREATE / ALTER TABLE / INDEX 文の NOLOGGING 句を使用します。詳細は、25-5 ページの「ロギング・モード」を参照してください。

追加情報： パラレルで作成した表の回復可能性の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

パラレル DDL の領域管理

表または索引のパラレル作成には、パラレル操作中に必要となる記憶領域と、表または索引が作成された後にできる空き領域の両方に影響する、領域管理の問題が関係しています。

CREATE TABLE ... AS SELECT および CREATE INDEX の記憶領域

パラレルで表または索引を作成すると、各パラレル実行サーバーは CREATE 文の STORAGE 句の値を使用して、行を格納するための一時的なセグメントを作成します。したがって、INITIAL として 5MB、PARALLEL DEGREE として 12 を指定して作成した表の場合、各プロセスが 5MB のエクステントで開始されるため、表を作成するには少なくとも 60MB の記憶領域が使用されます。パラレル実行コーディネータがセグメントを結合する時点でセグメントの一部が切り捨てられることがあるので、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*i* SQL リファレンス』を参照してください。

空き領域とパラレル DDL

パラレルで索引および表を作成すると、各パラレル実行サーバーは新しいエクステントを割り当てて、そのエクステントに表または索引のデータを挿入します。したがって、並行度 3 で索引を作成すると、当初、その索引用に少なくとも 3 つのエクステントが割り当てられます。（この説明は、パラレルでの索引の再構築、およびパラレルでのパーティションの移動、分割または再構築にも適用されます。）

シリアル操作では、スキーマ・オブジェクトに少なくとも 1 つのエクステントが含まれている必要があります。パラレル CREATE では、表または索引に、少なくともそのスキーマ・オブジェクトを作成するパラレル実行サーバーにあるのと同数のエクステントが含まれている必要があります。

表または索引をパラレルで作成すると、空き領域の「飛び地」（外部または内部断片化のどちらか）になることがあります。これは、パラレル実行サーバーが使用する一時セグメントが、行を格納するのに必要なサイズより大きい場合に生じます。

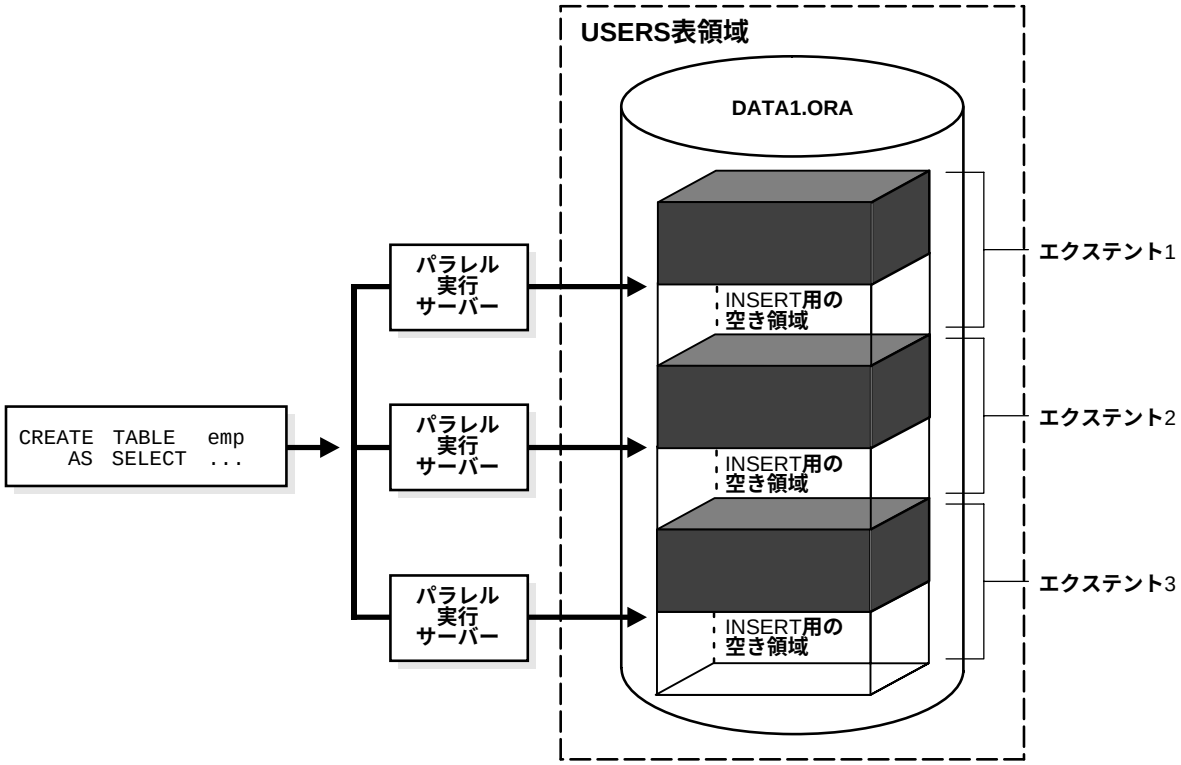
- 各一時セグメントの未使用領域が、表領域レベルで設定された MINIMUM EXTENT パラメータの値より大きい場合、すべての一時セグメントからの行をマージして表または索引にするときに、未使用領域が切り捨てられる。未使用領域はシステムの空き領域に戻され、新しいエクステントに割り当てることができますが、これは連続した領域ではないので（「外部断片化」）、それらを合わせても 1 つの大きなセグメントにはなりません。

- 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小さい場合、一時セグメントの中の行をマージして表または索引にするときに未使用領域を切り捨てることはできない。この未使用領域はシステムの空き領域には戻されず、表または索引の一部になり（「内部断片化」）、追加の領域を必要とする後続の挿入または更新でしか使用できません。

たとえば、CREATE TABLE ... AS SELECT 文で並列度 3 を指定した場合、表領域にデータ・ファイルが 1 つしかないと、図 26-7 に示すような内部断片化が発生する可能性があります。データ・ファイルの内部表エクステント内にある空き領域の「飛び地」を他の空き領域と合わせて、エクステントとして割り当てることはできません。空き領域を合わせる方法の詳細は、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*i* チューニング』を参照してください。

図 26-7 未使用の空き領域（内部断片化）



パラレル DML

「パラレル DML」（パラレル INSERT、パラレル UPDATE およびパラレル DELETE）では、大規模データベース表や索引に対する大規模な DML 操作の速度を上げたり、規模をさらに大きくするために、パラレル実行メカニズムが使用されます。

注意： 一般にはデータ操作言語（DML）に問合せも含まれますが、この章の「DML」という用語は挿入、更新および削除のみを指します。

ここで扱うパラレル DML に関するトピックは、次のとおりです。

- 手動によるパラレル化と比較したときのパラレル DML の利点
- パラレル DML を使用する場合
- パラレル DML の使用可能化
- パラレル DML のためのトランザクション・モデル
- パラレル DML の回復
- 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 25 章「ダイレクト・ロード・インサート」を参照してください。

手動によるパラレル化と比較したときのパラレル DML の利点

異なるデータ集合に対して複数の DML 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。
- キー値の範囲または ROWID の範囲が異なる、複数の UPDATE 文および DELETE 文を発行する。

ただし、手動によるパラレル化には次のような短所もあります。

- 使用しにくい。複数のセッションをオープンし（おそらく異なるインスタンス上で）、複数の文を発行する必要があります。
- トランザクション的なプロパティがない。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する必要があります（おそらくインスタンス間で）。
- 作業分割の複雑さ。作業を正確に分割するには、ROWID の範囲またはキー値の範囲を検索するために表を問い合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 および DELETE をパラレルで自動的に実行することにより、これらの欠点を補います。

パラレル DML を使用する場合

パラレル DML は、主として、大規模なデータベース・オブジェクトに対する規模の大きな DML 操作の速度を向上させるために使用されます。規模が大きいオブジェクトへのアクセスのパフォーマンスや拡張性が重要となっている意思決定支援システム（DSS）環境において、このパラレル DML は有用です。パラレル DML は、DSS データベースに問合せ機能と更新機能の両方を提供するという点で、パラレル問合せを補完するものとなっています。

パラレル化を設定することによるオーバーヘッドのため、パラレル DML 操作は短い OLTP トランザクションには適しません。ただし、OLTP データベース内で実行されるバッチ・ジョブは、パラレル DML 操作によって実行速度を上げることができます。

データ・ウェアハウス・システムの表のリフレッシュ

データ・ウェアハウス・システムの大規模な表は、実働システムの新しいまたは変更されたデータに基づいて定期的に「リフレッシュ」（更新）する必要があります。これは、更新可能な結合ビューと結合されるパラレル DML を使用することによって効果的に実行できます。

リフレッシュが必要なデータは、一般に、リフレッシュ・プロセスが開始する前に一時表にロードされます。この表には、新しい行またはデータ・ウェアハウスの最後のリフレッシュより後に更新された行が入れられます。パラレル UPDATE を使用する更新可能な結合ビューを使用することによって、更新された行をリフレッシュしたり、パラレル INSERT を使用する非ハッシュ結合を使用して新しい行をリフレッシュできます。

追加情報： 詳細は、『Oracle8*i* 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する必要があり、この計算には多数の大規模な中間的なサマリー表の組立てや処理が含まれます。これらのサマリー表は一時的な表であることが多いため、ログをとる必要がない場合がほとんどです。パラレル DML を使用すると、これらの規模の大きい中間表に対する操作の速度が向上します。利点の 1 つは、中間表に少しずつ結果を入れることができ、パラレル UPDATE を実行できることです。

さらに、サマリー表には累積情報または比較情報が入れられることもあります。そのような情報は複数のアプリケーション・セッション間で持続させる必要があるので、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、メーリング・リストへの追加など）に使用されます。

スコアを付ける作業では、大規模な表の多くの行を問い合せて更新します。パラレル DML を使用すると、これら規模の大きい表に対する操作の速度が向上します。

履歴データの表

履歴データの表には、最近までの特定の期間における会社全体の業務取引きが記録されます。DBA は、定期的に表の中の一連の古い行を削除し、一連の新しい行を挿入します。パラレルの INSERT... SELECT 操作とパラレルの DELETE 操作により、この入れ替え作業の速度が向上します。

外部ソースから大量のデータを挿入する場合、パラレル・ダイレクト・ローダー（SQL*Loader）を使用することもできますが、データベース内の別の表にすでに存在しているデータを挿入する場合はパラレル INSERT... SELECT のほうが高速です。

古い行を削除するためにパーティションを削除することもできますが、そのためには表が日付に基づいて適切な期間でパーティション化されている必要があります。

バッチ・ジョブ

終業後に OLTP データベースで実行されるバッチ・ジョブは、一定の時間枠の範囲内で完了させる必要があります。ジョブを時間内に確実に完了させるには、操作をパラレル化するのが適切な方法です。作業量が増加するにつれて、さらに多くのマシン・リソースを追加できます。パラレル操作の拡張性という特性を活用すれば、時間的な制約を満たせます。

パラレル DML の使用可能化

DML 文をパラレル化できるのは、ALTER SESSION に ENABLE PARALLEL DML オプションを指定して、セッションでパラレル DML を明示的に使用可能にした場合のみです。パラレル DML とシリアル DML ではロック要件、トランザクション要件およびディスク領域要件が異なるので、このモードが必要になります。（26-38 ページの「パラレル DML の領域に関する考慮事項」および 26-38 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 句を使用しても、DML はパラレル実行されません。

セッション内で PARALLEL DML を使用可能にすると、そのセッション内のすべての DML 文をパラレル実行するものとみなされます。ただし、PARALLEL DML を使用可能にしても、パラレル・ヒントや PARALLEL 句が指定されていない場合やパラレル操作の制限事項に違反する場合は、DML 操作がシリアルに実行されることがあります（26-40 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、DDL 文および 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 を使用可能にしたトランザクション

PARALLEL DML を使用可能にしたセッションでは、セッション内のトランザクションが次のような特別なモードになることがあります。つまり、トランザクションの DML 文が表をパラレルで変更する場合は、後続のシリアルまたはパラレルの問合せや DML 文がそのトランザクションで同じ表に再度アクセスすることはできません。したがって、トランザクション中にパラレル変更の結果を見ることはできません。

同じトランザクション内でパラレルで変更された表にアクセスしようとするシリアル文またはパラレル文は、エラー・メッセージ付きで拒否されます。

PARALLEL DML を使用可能にしたセッションで PL/SQL プロシージャまたはブロックが実行される場合、そのプロシージャまたはブロック内の文に、ここで説明したすべてのルールが適用されます。

パラレル DML のためのトランザクション・モデル

DML 操作をパラレルで実行するために、パラレル実行コーディネータはパラレル実行サーバーを取得または生成し、各パラレル実行サーバーは作業の一部分をそれ自身のパラレル・プロセス・トランザクションとして実行します。

- 各パラレル実行サーバーごとに、それぞれ異なるパラレル・プロセス・トランザクションを作成する。
- 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（次の項を参照）。

コーディネータにもそれ自身のコーディネータ・トランザクションがあり、コーディネータ・トランザクションにはそれ自身のロールバック・セグメントがあります。

ロールバック・セグメント

Oracle は、アクティブ・トランザクションが最少のロールバック・セグメントにトランザクションを割り当てます。先送り操作と取消し操作の両方をスピードアップするには、十分なロールバック・セグメントを作成し、それをオンラインにして、1 つのロールバック・セグメントに割り当てられるパラレル・プロセス・トランザクションが 2 個以下になるようにします。

必要に応じて拡張できるように、十分な領域を含む表領域内にロールバック・セグメントを作成し、それらのロールバック・セグメントの MAXEXTENTS 記憶領域パラメータを UNLIMITED に設定します。また、パラレル DML トランザクションのコミット後に、ロールバック・セグメントが OPTIMAL サイズに縮小されるように、ロールバック・セグメントの OPTIMAL 値を設定します。

2 フェーズ・コミット

1 つのパラレル DML 操作は、2 つ以上の独立したパラレル・プロセス・トランザクションによって実行されます。ユーザー・レベルのトランザクションがアトミックであるようにするため、コーディネータは 2 フェーズ・コミット・プロトコルを使用することにより、パラレル・プロセス・トランザクションによって実行される変更をコミットします。

この 2 フェーズ・コミット・プロトコルは、簡略化されたバージョンであり、共有ディスク・アーキテクチャを最大限に利用して、特にトランザクションの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、フォワード操作を実行する時間とほぼ同じです。

Oracle は、トランザクション障害とプロセス障害の発生後、およびインスタンス障害とシステム障害の発生後の、パラレル・ロールバックをサポートしています。Oracle では、トランザクション回復のロールフォワード段階とロールバック段階をパラレル化できます。

> **追加情報 :** パラレル・ロールバックの詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

ユーザーが発行したロールバックのトランザクション回復

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、パラレル実行コーディネータとパラレル実行サーバーによってパラレルで実行されます。そのロールバックにかかる時間は、フォワード・トランザクションの場合とほとんど同じです。

プロセス回復

パラレル DML コーディネータまたはパラレル実行サーバーの障害からの回復は、PMON プロセスによって実行されます。

- 1 つのパラレル実行サーバーで障害が発生すると、PMON はそのプロセスの作業をロールバックし、その他のすべてのパラレル実行サーバーはそれぞれ自分の作業をロールバックする。
- 複数のパラレル実行サーバーで障害が発生すると、PMON はそれらのプロセスの作業すべてをシリアルにロールバックする。
- パラレル実行コーディネータで障害が発生すると、PMON はコーディネータを回復し、すべてのパラレル実行サーバーはそれぞれ自分の作業をパラレルでロールバックする。

システム回復

システム障害から回復するには、新しく起動する必要があります。回復は、SMON プロセスと SMON によって作成された回復サーバー・プロセスによって実行されます。パラレル DML 文は、パラレル・ロールバックを使用してパラレルで回復できます。初期化パラメータ COMPATIBLE を 8.1.3 以上に設定していると、「ファースト・スタート・オン・デマンド・ロールバック」により、必要に応じてデッド・トランザクションを 1 度に 1 ブロックずつ回復できます（32-14 ページの「ファースト・スタート・オン・デマンド・ロールバック」を参照）。

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、他の有効なインスタンスの回復プロセス（つまり、SMON プロセスと、SMON によって作成された回復サーバー・プロセス）によって実行されます。有効なインスタンスの各回復プロセスは、障害が発生したパラレル実行コーディネータまたはインスタンスのパラレル実行サーバー・トランザクション（あるいはその両方）を独立して回復します。

パラレル DML の領域に関する考慮事項

パラレル UPDATE では、既存のオブジェクト内の領域が使用されます。この点で、データ用に新しいセグメントを取得するダイレクト・ロード・インサートとは対照的です。

パラレル操作ではオブジェクトを修正する同時実行の子トランザクションが複数あるため、文が順番に実行される場合とパラレル環境とでは領域使用の特性が異なる場合があります。

ダイレクト・ロード・インサートの領域の詳細は、25-8 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ります。パラレル DML は多数のロックを保持するため、ENQUEUE_RESOURCES パラメータと DML_LOCKS パラメータの値を大きくする必要があります。

パラレル UPDATE、DELETE または INSERT 文のプロセスで取得するロックは、次のとおりです。

- パラレル実行コーディネータが取得するロック
  - 表ロック SX を 1 つ
  - パーティションまたはサブパーティションごとにパーティション・ロック X を 1 つ

  パーティション表へのパラレル INSERT の場合、コーディネータはすべてのパーティションについてパーティション・ロックを取得します。パラレル UPDATE または DELETE の場合、関係するパーティションが WHERE 句で制限されていない限り、コーディネータはすべてのパーティションについてパーティション・ロックを取得します。

- 各パラレル実行サーバーが取得するロック
  - 表ロック SX を 1 つ
  - パーティションまたはサブパーティションごとにパーティション・ロック NULL を 1 つ
  - パーティションまたはサブパーティションごとにパーティション待機ロック X を 1 つ

1 つのパラレル実行サーバーは 1 つ以上のパーティションに対して処理を実行できますが、1 つのパーティションは 1 つのパラレル実行サーバーによってしか処理されません。

たとえば、600 個のパーティションがある表を並行度 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。
- パラレル実行サーバー全体として、表ロック SX を 100 個、パーティション・ロック NULL を 600 個、パーティション待機ロック X を 600 個取得する。

「ROW 移行パラレルの更新」と呼ばれる特殊なタイプのパラレル UPDATE が存在します。このパラレル更新方法を使用するのは、表移動句を使用可能にして表が定義されており、行を別のパーティションまたはサブパーティションに移動できる場合のみです。

表 26-2 に、異なるタイプのパラレル DML 文ごとに、コーディネータとパラレル実行サーバーが取得するロックのタイプをまとめます。

表 26-2 パラレル DML 文が取得するロック

文のタイプ	パラレル実行コーディネータが取得するロック	各パラレル実行サーバーが取得するロック
パーティション表のパラレル UPDATE または DELETE。WHERE 句はパーティションまたはサブパーティションのサブセットにプルニングされる。	表ロック SX を 1 つ プルニングされた（サブ）パーティションごとにパーティション・ロック X を 1 つ	表ロック SX を 1 つ パラレル実行サーバーが所有する、プルニングされた（サブ）パーティションごとにパーティション・ロック NULL を 1 つ パラレル実行サーバーが所有する、プルニングされた（サブ）パーティションごとにパーティション待機ロック S を 1 つ

表 26-2 パラレル DML 文が取得するロック

文のタイプ	パラレル実行コーディネータが取得するロック	各パラレル実行サーバーが取得するロック
パーティション表のパラレル行移行 UPDATE。WHERE 句は（サブ）パーティションのサブセットにプルニングされる。	表ロック SX を 1 つ	表ロック SX を 1 つ
	プルニングされた（サブ）パーティションごとにパーティション・ロック X を 1 つ	パラレル実行サーバーが所有する、プルニングされた（サブ）パーティションごとにパーティション・ロック NULL を 1 つ
		パラレル実行サーバーが所有する、プルニングされたパーティションごとにパーティション待機ロック S を 1 つ
	他のすべての（サブ）パーティション用にパーティション・ロック SX を 1 つ	他のすべての（サブ）パーティション用にパーティション・ロック SX を 1 つ
パーティション表のパラレル UPDATE、DELETE または INSERT	表ロック SX を 1 つ	表ロック SX を 1 つ
	すべての（サブ）パーティション用にパーティション・ロック X を 1 つ	パラレル実行サーバーが所有する（サブ）パーティションごとにパーティション・ロック NULL を 1 つ
		パラレル実行サーバーが所有する（サブ）パーティションごとにパーティション待機ロック S を 1 つ
非パーティション表へのパラレル INSERT	表ロック X を 1 つ	なし

パラレル DML の制限事項

パラレル DML（ダイレクト・ロード・インサートを含む）には次の制限が適用されます。

- 非パーティション表に対する更新操作と削除操作はパラレル化されない。
- 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に再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 文の後も存続する。後続の SQL 文（DML または問合せ）は、そのトランザクション中は変更された表にアクセスできない。
  - 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 および 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、削除カスケードおよび遅延整合性）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することはできない。
- 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・メッセージが出ます）。たとえば、非パーティション表にある UPDATE はシリアル化されます。

この後の項では、制限事項についてさらに詳しく説明します。

パーティション化キーについての制限事項

パーティション表のパーティション化キーを新しい値に更新できるのは、行移動句を使用可能にして表が定義されていない限り、更新を実行しても行が新しいパーティションへ移動しない場合のみです。

ファンクションの制限事項

パラレル DML に関するファンクションの制限事項は、パラレル DDL およびパラレル問合せの場合と同じです。26-43 ページの「関数のパラレル実行」を参照してください。

データの整合性についての制限事項

ここでは、整合性制約とパラレル DML 文の相互関係について説明します。

NOT NULL と CHECK このタイプの整合性制約は許可されています。これらは列および行レベルで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 と PRIMARY KEY このタイプの整合性制約は許可されています。

FOREIGN KEY（参照整合性） ある表に対する DML 操作によって別の表で再帰的 DML 操作が実行される場合、または、整合性検査を実行するために、修正対象のオブジェクトに加えられたすべての変更を同時に見る必要がある場合には、参照整合性についての制限があります。

表 26-3 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 26-3 参照整合性制約

DML 文	親に対して発行	子に対して発行	自己参照型
INSERT	（適用されない）	パラレル化されない	パラレル化されない
UPDATE No Action	サポートされる	サポートされる	パラレル化されない
DELETE No Action	サポートされる	サポートされる	パラレル化されない
DELETE CASCADE	パラレル化されない	（適用されない）	パラレル化されない

DELETE Cascade パラレル実行サーバーは複数のパーティション（親表および子表）から行を削除しようとするので、DELETE カスケードを使用する外部キーを持つ表に対する削除はパラレル化されません。

自己参照整合性 参照キー（主キー）が関係する場合、自己参照整合性制約がある表に対する DML はパラレル化されません。他のすべての列に対する DML は、パラレル化できます。

遅延可能整合性制約 操作対象の表に遅延可能制約がある場合、DML 操作はパラレル化されません。

トリガーの制限事項

DML 操作の影響を受ける表に、その DML 文によって起動される可能性のある有効なトリガーがある場合、その DML 操作はパラレル化されません。これは、レプリケートされる表に対する DML 文はパラレル化されないということにもなります。

そのような表に対する DML をパラレル化するには、関連するトリガーを使用禁止にする必要があります。トリガーを使用可能または使用禁止にすると、依存する共有カーソルは無効になることに注意してください。

分散トランザクションの制限事項

DML 操作が分散トランザクション内にある場合、または DML 操作または問合せ操作の対象がリモート・オブジェクトである場合、その DML 操作はパラレル化できません。

例 1: リモート・オブジェクトを問い合せる DML 文

```
INSERT /* APPEND PARALLEL (t3,2) */ INTO t3 SELECT * FROM t4@dblink;
```

この問合せ操作はリモート・オブジェクトを参照しているため、シリアルで実行され、通知はありません。

例 2: リモート・オブジェクトに対する DML 操作

```
DELETE /*+ PARALLEL (t1, 2) */ FROM t1@dblink;
```

この DELETE 操作はリモート・オブジェクトを参照しているため、パラレル化されません。

例 3: 分散トランザクション

```
SELECT * FROM t1@dblink;
DELETE /*+ PARALLEL (t2,2) */ FROM t2;
COMMIT;
```

この DELETE 操作は、分散トランザクション（SELECT 文で開始）で発生しているため、パラレル化されません。

関数のパラレル実行

PL/SQL や Java で記述されたか、または C で外部プロシージャとして記述されたユーザー記述関数の実行は、パラレル化できます。関数に使用される PL/SQL のパッケージ変数または Java の静的属性全体を、個々のパラレル実行プロセスに対してプライベートにすることができます。ただし、これらは、元のセッションからコピーされるのではなく、各パラレル実行プロセスの開始時に新しく初期化されます。このため、パラレルで実行した場合に、すべての関数で正しい結果が生成されるとは限りません。

ユーザー記述関数をパラレルで実行できるようにするには、その関数を CREATE FUNCTION または CREATE PACKAGE 文で宣言するときに、PARALLEL_ENABLE キーワードを使用します。

パラレル問合せにおける関数

SELECT 文や、DML または DDL 文の副問合せでユーザー記述関数をパラレルで実行できるのは、PARALLEL_ENABLE キーワードで宣言されている場合、パッケージまたは型で宣言されており、すべての WNDS、RNPS および WNPS を示す PRAGMA RESTRICT_REFERENCES がある場合、または、CREATE FUNCTION で宣言されており、システムが PL/SQL コードの本体を分析し、そのコードがデータベースへの書込みもパッケージ変数の読込みや変更もしないことを判断できる場合です。

問合せまたは副問合せの他の部分は、その関数はそのままシリアルで実行する必要がある場合にも、パラレルで実行できる場合があります。

> **追加情報：** プラグマ RESTRICT_REFERENCES の説明は『Oracle8*i* アプリケーション開発者ガイド 基礎編』、CREATE FUNCTION の説明は『Oracle8*i* SQL リファレンス』を参照してください。

パラレルの DML 文と DDL 文の関数

パラレルの DML 文または DDL 文で、パラレル問合せのようにユーザー記述関数をパラレルで実行できるのは、PARALLEL_ENABLE キーワードで宣言されている場合、パッケージまたは型で宣言されており、すべての RNDS、WNDS、RNPS および WNPS を示す PRAGMA RESTRICT_REFERENCES がある場合、または、CREATE FUNCTION で宣言されており、システムが PL/SQL コードの本体を分析し、そのコードがデータベースの読書きもパッケージ変数の読込みや変更もしないことを判断できる場合です。

パラレル DML 文の場合は、パラレルで実行できないファンクション・コールがあると、DML 文全体がシリアルで実行されます。

INSERT ... SELECT または CREATE TABLE ... AS SELECT 文の場合、問合せ部のファンクション・コールは、前述のパラレル問合せルールに従ってパラレル化されます。問合せは、文の残りの部分をシリアルに実行する必要がある場合にもパラレル化でき、その逆も可能です。

親和性

注意： この項で説明している機能は、Parallel Server Option 付きの Oracle8*i* Enterprise Edition を購入した場合にのみ利用できます。

共有ディスク・クラスタまたは大量パラレル処理（MPP）構成では、Oracle Parallel Server のインスタンスを実行している 1 つまたは複数のプロセッサが直接にデバイスにアクセスする場合、そのインスタンスはそのデバイスに対して「親和性」があるといいます。同様に、ファイルが格納されているデバイスに対して「親和性」のあるインスタンスには、そのファイルに対する親和性があります。

親和性の決定には、複数のデバイスにわたってストライプ化されているファイルについての任意の決定が含まれることがあります。さらに言えば、インスタンスが表領域内の最初のファイルに対して親和性を持っているなら、そのインスタンスはその表領域（またはその表領域内の表あるいは索引のパーティション）に対して親和性があるといえます。

Oracle では、パラレル実行サーバーに作業を割り当てるときに親和性が考慮されます。SQL 文のパラレル実行に親和性が使用されることは、ユーザーからはわかりません。

親和性とパラレル問合せ

パラレル問合せで親和性の機能を使用すると、データに「近い」プロセッサで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方法は、表パーティションまたは索引パーティションを 1 つのデバイス上の 1 つのファイルに格納することです。この構成を使用すると、デバイス障害による損傷を限定することによる高い可用性が約束され、パーティションによるパラレルの索引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用してアクセスするデータの合計量をプルニングすることができ、しかも ROWID 範囲によるパラレル表（パーティション）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 用に使用する場合であっても、索引は個々のデバイス上でパーティション化します。

その他の構成（たとえば、複数のデバイスにまたがってストライプ化されたファイル内に複数のパーティションがある構成）を使用しても正しい問合せ結果が得られますが、正しい並行度を選択するにはヒントを使用するかオブジェクト属性を明示的に設定する必要があります。

親和性とパラレル DML

パラレル DML（INSERT、UPDATE および DELETE）の場合、親和性の拡張機能により、DML 操作をそのパーティションに対する親和性があるノードにルーティングできるため、キャッシュのパフォーマンスが向上します。

DML 操作をパラレルで実行するために、一連のインスタンスまたはパラレル実行サーバー（あるいはその両方）の間で作業を分配する方法は、親和性によって決められます。親和性を利用すると、次のように問合せのパフォーマンスが改善されます。

1. 特定の MPP アーキテクチャでは、Oracle は、パラレル実行サーバーをどのノードで生成するか（「パラレル・プロセスの割当て」）、そして特定のノードにどの程度の作業グラニュル（ROWID の範囲またはパーティション）を送るか（「作業の割当て」）を決定するのに、デバイスとノード間の親和性情報を使用します。パフォーマンスを改善するには、各ノードが主としてローカル・デバイスをアクセスするようにし、各ノードのバッファ・キャッシュ・ヒット率を高め、ネットワークのオーバヘッドと 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・メカニズムを使用してデバイスをノードに割り当てます。項目 1 と同じように、パラレル・プロセスの割当てや作業の割当てを決定するときに、このデバイスとノード間の親和性が使用されます。
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 - デバイス間の親和性を使用してデバイスの分離が実現されます。これにより、複数のパラレル実行サーバーが同じデバイスに同時にアクセスする機会が少なくなります。このプロセス - デバイス間の親和性情報は、プロセス間でのスチール処理をインプリメントするのにも使用されます。

パーティション表とパーティション表索引では、パーティションとノード間の親和性情報により、プロセスの割当てと作業の割当てが決定されます。非共有 MPP システムの場合、Oracle Parallel Server はパーティションをインスタンスに割り当てる際に、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のみです。複数の文にわたって持続する親和性情報は、バッファ・キャッシュ・ヒット率を改善し、インスタンス間でのブロックの ping を少なくします。

> **追加情報：** Oracle Parallel Server の詳細は、『Oracle8*i* 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用できます。

- 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、パラレル実行コーディネータと複数のパラレル実行サーバーによって実行されます。ただし、パラレル・ロードは、異なるメカニズムを使用します。

> **追加情報：** パラレル・ロードの詳細と SQL*Loader の概要は、『Oracle8*i* ユーティリティ・ガイド』を参照してください。パラレル・ロードの使用方法は、『Oracle8*i* チューニング』を参照してください。

パラレル実行コーディネータとパラレル実行サーバーの動作は、どんな種類の操作（SQL、回復または伝播）を実行するかによって異なる場合があります。たとえば、プール中のすべてのパラレル実行サーバーが使用中で、すでに最大数のパラレル実行サーバーが開始されている場合の動作は、次のとおりです。

- パラレル SQL ロールでは、パラレル実行コーディネータはシリアル処理に切り替える。
- パラレル伝播ロールでは、パラレル実行コーディネータはエラーを戻す。

特定のセッションごとに、パラレル実行コーディネータは 1 種類の操作のみを調整します。パラレル実行コーディネータは、たとえばパラレル SQL とパラレル伝播またはパラレル回復を同時には調整できません。

パラレル回復の概要は、32-10 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*i* バックアップおよびリカバリ・ガイド』、パラレル伝播の詳細は『Oracle8*i* レプリケーション・ガイド』を参照してください。

第 VIII 部

データの保護

第 VIII 部では、Oracle のデータベース内のデータを保護する方法、およびデータベース管理者がデータ保護をさらに強化するために行えることについて説明します。

第 VIII 部に含まれる章は、次のとおりです。

27

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。ただし、マルチユーザー・データベースでは、複数のトランザクション内の文によって、同じデータが同時に更新される可能性があります。同時に実行される複数のトランザクションで、意味のある一貫した結果が得られなければなりません。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。

- 「データ同時実行性」とは、多数のユーザーが同時にデータにアクセスできることを意味する。
- 「データ一貫性」は、各ユーザーにデータの一貫したビューが表示され、その中にユーザー自身のトランザクションや他のユーザーのトランザクションによる参照可能な変更も含まれることを意味する。

「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規則を施行するもので、第 28 章「データの整合性」で説明しています。

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は、同時にではなく 1 つずつ（直列に）実行されるように見えます。

一般に、この程度までのトランザクションの分離が望ましいとされますが、このモードで多数のアプリケーションを実行すると、アプリケーションのスループットがかなり低下する可能性があります。同時に実行される各トランザクションの完全な分離とは、あるトランザクションが問合せ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用できます。

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同時に実行する場合に防ぐ必要がある 3 つの現象という観点から定義されています。

3 つの防止可能な現象とは、次のものです。

内容を保証しない読込み	まだコミットされていないトランザクションが書き込んだデータを、別のトランザクションが読み込む現象

反復不能読込み（ファジー読込み）	あるトランザクションが以前に読み込んだデータをもう一度読み込んだときに、コミットされた別のトランザクションによってそのデータが変更または削除されたことが明らかになる現象
仮読込み	あるトランザクションが検索条件を満たす一連の行を戻す問合せを 2 度実行する間に、コミットされた別のトランザクションによってその条件を満たす新しい行が挿入されたことが明らかになる現象

SQL92 では、それぞれの分離レベルで実行されるトランザクションで起こり得る現象という観点で 4 つの分離レベルを定義しています。

分離レベル	内容を保証しない読込み	反復不能読込み	仮読込み
非コミット読込み	あり	あり	あり
コミット読込み	なし	あり	あり
反復可能読込み	なし	なし	あり
直列可能	なし	なし	なし

Oracle で使用できる分離レベルは、SQL92 の一部ではない読込み専用モードと、「コミット読込み」および「直列可能」です。デフォルトはコミット読込みであり、Oracle リリース 7.3 より前では自動分離レベル としてコミット読込みのみが提供されていました。コミット読込みと直列可能の分離レベルの詳細は、27-4 ページの「データの同時実行性と一貫性の管理方法」を参照してください。

ロックのメカニズム

一般に、マルチユーザー・データベースでは、なんらかのデータ・ロックを使用してデータの同時実行性、一貫性、および整合性の問題を解決しています。「ロック」は、同じリソースにアクセスしている複数のトランザクションの間で破壊的な相互作用が起きないようにするメカニズムです。

リソースには、次の 2 つの一般的なタイプのオブジェクトが含まれます。

- ユーザー・オブジェクト。たとえば、表と行（構造体とデータ）。
- ユーザーには見えないシステム・オブジェクト。たとえば、メモリー内の共有データ構造やデータ・ディクショナリ行。

DDL ロックや内部ロックなど、各種ロックの詳細は、27-15 ページの「データをロックする方法」を参照してください。

データの同時実行性と一貫性の管理方法

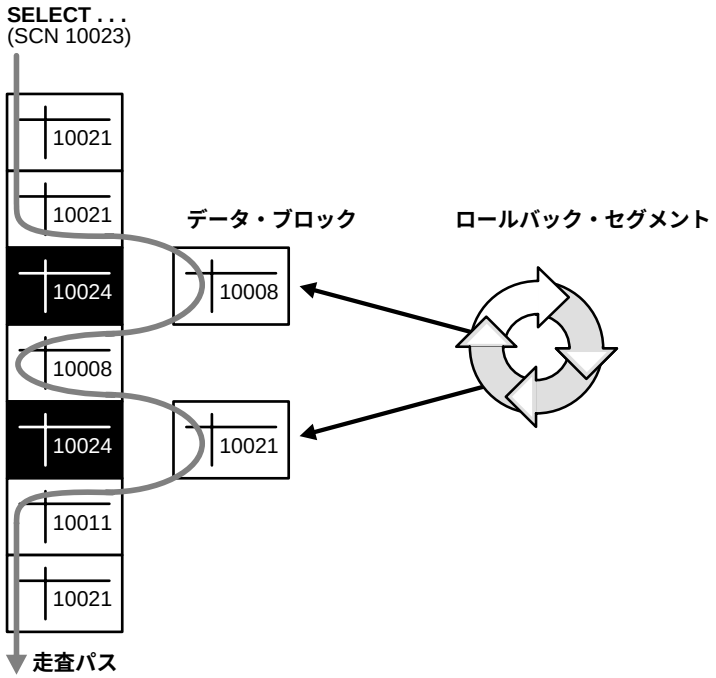
Oracle は、マルチバージョン一貫性モデルや、さまざまなタイプのロックおよびトランザクションを使用することによって、マルチユーザー環境でのデータの一貫性を維持します。

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込み一貫性が提供されます（文レベルの読込み一貫性）。読込み一貫性は、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・レベルの読込み一貫性）。

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用します。ロールバック・セグメントは、コミットされていないトランザクション、または最近コミットされたトランザクションによって変更されたデータの古い値が含まれています。図 27-1 に、Oracle がロールバック・セグメント内のデータを使用して、文レベルの読込み一貫性をどのように提供するかを示します。

図 27-1 トランザクションと読込み一貫性



問合せが実行段階に入ると現行のシステム変更番号（SCN）が判別されます。図 27-1 では、このシステム変更番号が 10023 になっています。問合せのためにデータ・ブロックを読み込むと、観察されたシステム変更番号（SCN）が書き込まれたブロックのみが使用されます。変更されたデータを含むブロック（もっと後の SCN）があると、それらはロールバック・セグメント内のデータに基づいて再構成され、問合せには再構成されたデータが戻されます。そのため、問合せを実行するたびに、問合せの実行開始時点で記録された SCN に関してコミット済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のに十分な情報がロールバック・セグメント内に残っていないためです。通常、このエラーが発生するのは、多くの更新アクティビティによってロールバック・セグメントが折り返され、長時間実行する問合せが必要とするデータの再構築に必要な変更が上書きされてしまう場合です。このイベントでは、次のようなエラー 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・セグメント番号： %s、名前： %S が小さすぎます）。

このエラー状態は、より多くの、またはより大きなロールバック・セグメントを作成することによって回避できます。また、同時実行トランザクションの数が少ないときに長時間実行の問合せを発行するという方法や、問合せの対象となる表について共有ロックを取得してトランザクションの実行中には他の排他ロックを禁止するという方法でも、このエラーを回避できます。

文レベルの読込み一貫性

Oracle では、常に「文レベル」の読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のみです。問合せは、文の実行の開始後にコミットされた変更は参照しません。

あらゆる問合せに対して一貫した一連の結果が保証され、これによって、データの一貫性が保証されます。この際、ユーザーは何もする必要がありません。SQL 文の SELECT、副問合せを伴う INSERT、UPDATE および DELETE は、いずれも明示的または暗黙的にデータの問合せを実行し、一貫性のあるデータを戻します。これらの文は、問合せを使用して、処理（SELECT、INSERT、UPDATE または DELETE）の対象となるデータを判別します。

SELECT 文は明示的な問合せであり、ネストした問合せや結合操作を持つこともあります。INSERT 文では、ネストした問合せを使用できます。UPDATE 文と DELETE 文では、WHERE 句や副問合せを使用し、表のすべての行ではなく、一部の行を処理の対象にすることができます。

INSERT 文、UPDATE 文および DELETE 文で使用される問合せでは、一貫した結果セットの取得が保証されます。ただし、その DML 文そのものによって加えられる変更は見えません。つまり、そのような操作での問合せは、その操作によって変更される前のデータの状態を参照します。

トランザクション・レベルの読込み一貫性

Oracle では、「トランザクション・レベルの読込み一貫性」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・レベルの読込み一貫性によって反復可能読込みが実現され、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 Parallel Server における読込み一貫性

Oracle Parallel Server では、単一データベースにアクセスする複数インスタンス間でデータ整合性を保証するために、「キャッシュ・フュージョン」というパラレル・キャッシュ管理テクニックが採用されています。読込み一貫性を備えたブロックに対するインスタンス要求によって、読込み側と書込み側にキャッシュ一貫性矛盾が生じると、キャッシュ・フュージョンはブロック・サーバー・プロセス（BSP）を使用して、ブロックを保持しているインスタンスのメモリー・キャッシュから要求側インスタンスのメモリー・キャッシュに、ブロックを直接コピーします。ブロックを保持しているインスタンスは、コミットされていないトランザクションをロールバックし、ブロックをディスクに書き込まないで要求側に直接送ります。このブロックの状態は、要求側ノードで要求が発行された時点で一貫性を備えています。

> **追加情報：** キャッシュ・フュージョンおよびブロック・サーバー・プロセスのプロセスの詳細は、『Oracle8*i* Parallel Server 概要および管理』を参照してください。

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。

コミット読込み	デフォルトのトランザクション分離レベル。あるトランザクションによって実行されるそれぞれの問合せで参照されるのは、その問合せ（トランザクションではなく）が開始される前にコミットされたデータのみです。Oracle の問合せでは、内容が保証されない（コミットされていない）データが読み込まれることはありません。 Oracle では、問合せで読み込まれたデータを他のトランザクションで変更できるので、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、反復不能読込みと仮読込み（実体のない読込み）の現象の両方が起こり得ます。
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、UPDATE 文および DELETE 文で実行した変更のみが参照されます。直列可能トランザクションでは、反復不能読込みや仮読込みの現象は起きません。
読込み専用	読込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のみが参照され、INSERT 文、UPDATE 文および DELETE 文は使用できません。

分離レベルの設定

アプリケーション・デザイナ、アプリケーション開発者およびデータベース管理者は、アプリケーションと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いずれかを使用して設定できます。

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION ISOLATION LEVEL READ ONLY;
```

各トランザクションを SET TRANSACTION 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 コマンドを使用して、後続のすべてのトランザクションのトランザクション分離レベルを設定することもできます。

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;

ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;
```

追加情報： これらの SQL コマンドの詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

コミット読込みによる分離

Oracle のデフォルト分離レベルは、コミット読込みです。このレベルの分離は、競合するトランザクションの数が少ない環境に適しています。問合せごとに、それぞれ独自のスナップショット時間を基準として実行されます。このため、ある問合せを複数回実行する間に反復不能読込みと仮読込みが発生することもありますが、高いスループットが得られます。コミット読込みによる分離は、競合するトランザクションの数が少ない環境に適した分離レベルです。

直列可能分離

直列可能分離は、次のような環境に適しています。

- 大規模データベースを使用し、短いトランザクションでごく少数の行しか更新しない。
- 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込み専用である。

直列可能分離では、同時実行トランザクションが実行できる変更は、トランザクションが順に 1 つずつ実行される場合に実行できるデータベース変更のみです。特に、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のみです。

この判別の効率をよくするために、データ・ブロック内に格納されている制御情報、つまりブロック内の行のうち、どの行にコミットされた変更が含まれていて、どの行にコミットされてない変更が含まれているかを示す情報が使用されます。ある意味で、ブロックには、ブロック内の各行に影響を与えたトランザクションの最新の履歴が含まれていると考えられます。ここに保存される履歴の量は、CREATE TABLE および ALTER TABLE の INITRANS パラメータによって制御されます。

場合によっては、「ごく最近の」トランザクションで行が更新されたかどうかを判別するには履歴情報が不十分である場合もあります。これは、多数のトランザクションが同時に同じデータ・ブロックを変更している場合や、非常に短い期間にそのような操作が実行されている場合に起きることがあります。多数のトランザクションが同一のブロックを変更するような表については、INITRANS の値を大きく設定しておくことにより、この状況を回避できます。それにより、ブロックにアクセスした最新のトランザクションの履歴を記録するのに十分な記憶域が各ブロックに割り当てられます。

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。

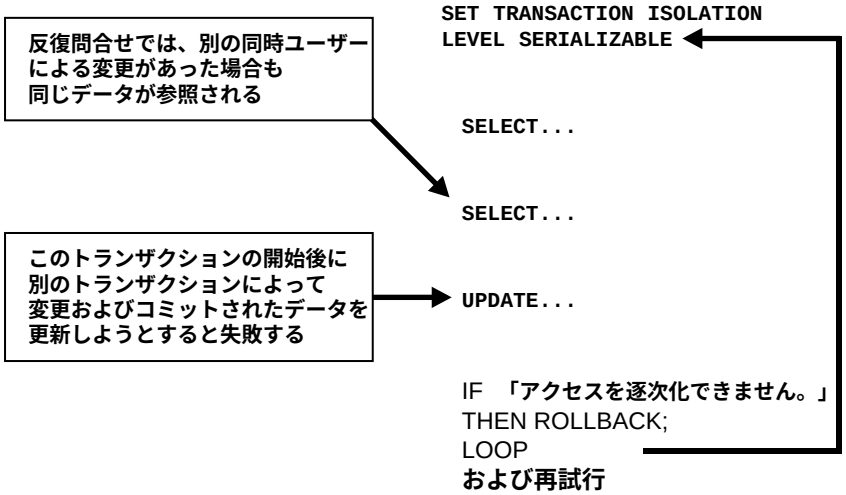
```
ORA-08177: このトランザクションのアクセスを逐次化できません。
```

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実行できます。

- 実行した作業をそのポイントまでコミットする。
- その他の（異なる）文を実行する（トランザクション内で以前に設定したセーブポイントまでロールバックした後など）。
- トランザクション全体をロールバックする。

図 27-2 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 27-2 直列可能トランザクションの失敗



コミット読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、次のような使用例を考えます。データベースの表の集合（またはデータの任意の集合）があり、それらの表内の行を特定の順序で何度か読み込み、一連のトランザクションをある特定の時点でコミットするとします。ある操作（問合せまたはトランザクション）のどの読込みでも、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 27-1 に、Oracle での コミット読込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 27-1 コミット読込みと直列可能トランザクション

	コミット読込み	直列可能
内容を保証しない書込み	なし	なし
内容を保証しない読込み	なし	なし
反復不能読込み	あり	なし
仮読込み	あり	なし
ANSI/ISO SQL 92 への準拠	はい	はい
スナップショット読込み時期	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
読込みが書込みを阻止するか	いいえ	いいえ
書込みが読込みを阻止するか	いいえ	いいえ
別の行の書込みが書込みを阻止するか	いいえ	いいえ
同一行の書込みが書込みを阻止するか	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

コミット読込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用します。また、コミットされていない同時実行のトランザクションによって更新された行を変更しようとすると、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。

ただし、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ので、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込みロックを使用しないので、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はないとは想定しないでください。このことを考慮してアプリケーション・レベルの一貫性チェックをコーディングしない限り、直列可能トランザクションを使用しても、データベースの一貫性が損なわれることがあります。

追加情報： 参照整合性と直列可能トランザクションの詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

Oracle Parallel Server

Oracle Parallel Server（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用できます。

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（一部のノードのみ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 も非 Oracle も）、直列可能分離モードがサポートされている必要があります。

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と、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のみです。

これとは対照的に、コミット読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・デザイナおよびアプリケーション開発者は、アプリケーションのコーディング以外にアプリケーションのパフォーマンスと一貫性のニーズに基づいて分離レベルを選択する必要があります。

多数の同時実行ユーザーが次々にトランザクションを送る環境では、トランザクションの予想到着率および応答時間の要求の面から、トランザクションのパフォーマンス要件を評価する必要があります。高いパフォーマンスが必要な環境では、分離レベルを選択する際に一貫性と同時実行性（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込みによっては書込みが阻止されないという点を考慮する必要があります。

Oracle の 2 つの分離モードは、行レベル・ロックと Oracle のマルチバージョン同時実行性制御システムとの組合せによって、高水準の一貫性と同時実行性（およびパフォーマンス）を提供します。Oracle では、読込み側と書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、コミット読込みが最適な分離レベルです。これは、リリース 7.3 より前の Oracle 上で実行されるアプリケーションが使用する分離レベルです。

コミット読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（仮読込みや反復不能読込みのために）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が高く、高いパフォーマンスが求められる多くの環境では、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少数で、トランザクション到着率が非常に低い環境では、仮読込みおよび反復不能読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 のコミット読込み分離では、すべての問合せについてトランザクション集合の一貫性が提供されます（つまり、どの問合せでも一貫性のある状態のデータが参照されます）。したがって、マルチバージョン同時実行性制御を使用しない他のデータベース管理システム上で実行される場合にはさらに高水準の分離を必要とするようなアプリケーションでも、多くの場合、コミット読込み分離で十分に対応できます。

コミット読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、仮読込みおよび反復不能読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込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用し、短いトランザクションでごく少数の行のみが更新される環境です。

直列可能分離モードは、仮読込みと反復不能読込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、読込み / 書込みトランザクションが 1 つの問合せを複数回実行する場合に重要になることがあります。

他の処理系の直列可能分離では書込みだけでなく読込みについてもブロックがロックされるのに対し、Oracle では非ブロッキング問合せときめ細かい行レベル・ロックが提供され、それらによって書込み / 読込みの競合が少なくなります。読込み / 書込みの競合が多く発生するアプリケーションでは、Oracle の直列可能分離は、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に適していても、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、一時点でのデータベースを参照するので、読込み / 書込みトランザクションで複数の一貫した問合せを発行する必要がある場合は、この分離レベルが適しています。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、UPDATE および 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用してください。

注意 : 副問合せを持つ DML 文を含むトランザクションは、コミット読込みを保証するために直列可能分離を使用する必要があります。

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成する場合には、直列可能モードに対応したトランザクションの設計が必要なことがあります。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 のコミット読込みモード（直列可能性エラーを生成しないモード）で使用できます。

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。（従来型の読込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ので注意してください。この実現方式では、長時間実行されるトランザクションは、読込みトランザクションであっても、短い更新トランザクションの処理を阻止したり、逆に短いトランザクションが長時間実行のトランザクションの処理を阻止するからです。）

アプリケーション開発者は、直列可能モードを使用するときはトランザクションのロールバックと再試行に要するコストを考慮する必要があります。デッドロックが頻繁に発生する読込みロックのシステムと同様に、直列可能モードを使用するときには、異常終了したトランザクションによって実行された処理をロールバックし、再実行する必要が生じます。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で再起動されたトランザクションと別のトランザクションの間で、2 度目の競合が発生することは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。ただし、そのトランザクションが正常に完了する保証はないので、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・デザイナは、Oracle での読込みは、他のシステムとは違って、書込みを阻止しないという点を考慮する必要があります。データベースの一貫性をアプリケーション・レベルでチェックするトランザクションでは、SELECT FOR UPDATE などのコーディング技法を使用しなければならないことがあります。直列可能モードを使用しているアプリケーションを他の環境から Oracle に移植する場合には、この問題を検討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やデータ・ディクショナリ行など）のどちらかにアクセスする複数のトランザクションの間で、破壊的な相互作用を回避するためのメカニズムです。

どのような状況でも、SQL 文が実行されると、Oracle によって必要なロックが自動的に取得されます。そのため、ユーザーがそのような詳細事項にかかわる必要はありません。Oracle は、最も高度なデータ同時実行性とフェイルセーフなデータの整合性を同時に実現するために、最も低いレベルの制限でデータを自動的にロックします。また、必要に応じて、手動でもデータをロックできます。

Oracle が使用する内部ロックの詳細は、27-19 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・メカニズムを使用して、トランザクション間のデータの同時実行性と一貫性を実現します。Oracle のロック・メカニズムはトランザクション制御と密接に結び付けられているため、アプリケーション・デザイナがトランザクションを適切に定義するだけで、ロックは Oracle によって自動的に管理されます。

Oracle のロックは完全に自動化されていて、ユーザー・アクションを必要としません。すべての SQL 文について暗黙のロックが発生するため、データベース・ユーザーがリソースを明示的にロックする必要はありません。Oracle のデフォルトのロック・メカニズムは、最高度のデータ同時実行性を実現しながら、データの整合性を保証するために、最も低い制限レベルでデータをロックします。

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する必要がある状況や、その操作方法の詳細は、後の項で説明します。27-30 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・モードを使用します。

排他ロック・モード	関連リソースが共有されないようにします。このロック・モードはデータを変更するために取得します。リソースを排他的にロックした最初のトランザクションは、その排他ロックが解除されるまで、そのリソースを変更できる唯一のトランザクションになります。
共有ロック・モード	操作の種類に応じて、関連するリソースの共有が可能です。データの読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込み、更新内容の消失、有害な DDL 操作など）が防止されます。あるトランザクションの SQL 文によって加えられた変更を参照できるのは、そのトランザクションがコミットされた後に開始される別のトランザクションのみです。

トランザクションをコミットまたはロールバックすると、そのトランザクション内の文によって取得されたすべてのロックが解除されます。また、セーブポイントまでロールバックした場合には、そのセーブポイントより後で取得されたロックが解除されます。ただし、ロックを解除されて使用可能となったリソースに対してロックを取得できるのは、ロック中だったリソースを待機していなかったトランザクションのみです。ロック中のリソースを待機していたトランザクションは、元のトランザクションが完全にコミットされるかロールバックされるまで待機し続けます。

データ・ロック変換とロックの段階的拡大の比較

トランザクションは、トランザクション内で挿入、更新または削除の対象になる行すべてに対して排他ロックを保持します。行ロックは、制限の最も高い程度で取得されるため、ロック変換はまったく必要なく、実行されません。

Oracle は、低い制限の表ロックを、より高い制限の適切な表ロックに自動的に変換します。たとえば、トランザクションが表の行をロックするために、FOR UPDATE 句を指定した SELECT 文を使用する場合を考えます。その結果、排他行ロックとその表に対する行共有表ロックが取得されます。後ほど、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、27-21 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（行レベルなど）で多数のロックが保持されている場合に、各ロックをデータベースが上位レベル（表レベルなど）の別のロックに変更した場合です。たとえば、あるユーザーが表の中の多数の行をロックした場合、データベースによっては、そのユーザーが保持する複数の行ロックを単一の表ロックに自動的に拡大する場合があります。ロックの数は少なくなりますが、ロックされている対象の制限は大きくなります。

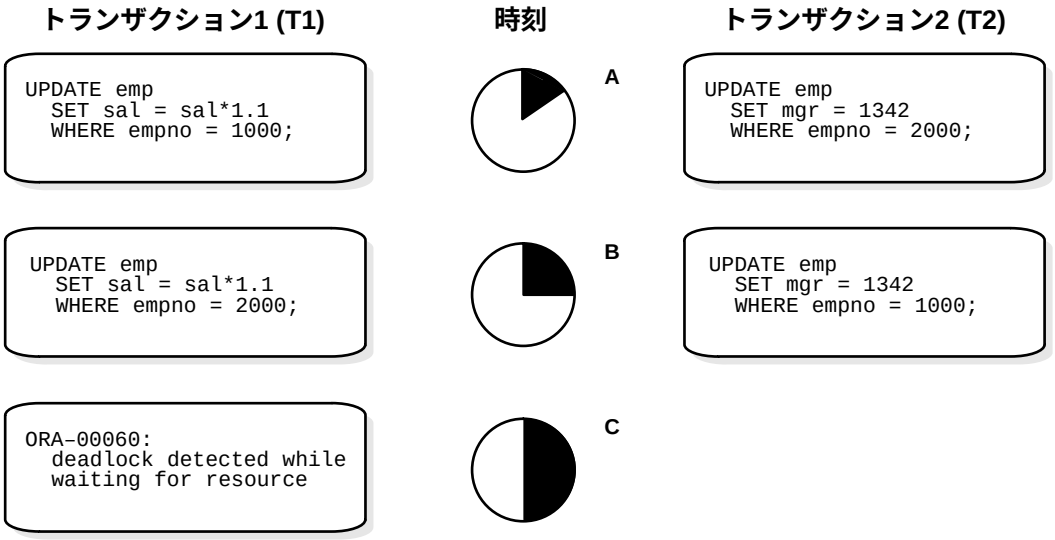
Oracle では、ロックの段階的拡大が発生することはありません。発生すると、デッドロックの可能性が大幅に増大します（後述）。たとえば、システムがトランザクション T1 のためにロックを拡大しようとしますが、トランザクション T2 によって保持されているロックのために拡大できないものとします。ここで、トランザクション T2 の処理を進めるのに同じデータのロックの段階的拡大が必要な場合、デッドロックが発生します。

デッドロック

「デッドロック」が発生するのは、2 人以上のユーザーが、相手がロックしているデータを待機している場合です。デッドロックが発生すると、一部のトランザクションは処理を継続できなくなります。図 27-3 に、デッドロック状態になっている 2 つのトランザクションを示します。

図 27-3 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。ただし、各トランザクションは、次に、他方のトランザクションが現在保持している行を更新しようとします。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となる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 27-3　デッドロック状態の 2 つのトランザクション



デッドロックの検出

Oracle は、デッドロック状況を自動的に検出し、そのデッドロックに含まれる文の 1 つをロールバックして、競合する一連の行ロックを解放することにより、デッドロックを解決します。また、対応するメッセージが、文レベルのロールバックの対象となったトランザクションに戻されます。ロールバックされる文は、デッドロックが検出されたトランザクションに属しています。通常、通知されたトランザクションは明示的にロールバックする必要がありますが、待機した後でロールバック文を再試行できます。

注意： 分散トランザクションの場合、ローカル・デッドロックは「待機」グラフを分析することによって検出され、グローバル・デッドロックはタイムアウトによって検出されます。いったん検出されると、非分散デッドロックも分散デッドロックも、データベースとアプリケーションによって同じように処理されます。

デッドロックが最も頻繁に発生するのは、トランザクションが Oracle のデフォルト・ロックを明示的に上書きする場合です。Oracle 自体はロックの段階的拡大を実行せず、また、問合せに読込みロックを使用せず、また行レベルのロック（ページ・レベルのロックではない）を使用するため、Oracle ではデッドロックはまれにしか起こりません。手動によるロックの取得の詳細とデッドロック状況の例は、27-30 ページの「明示的（手動）データ・ロック」を参照してください。

デッドロックの回避

多くの場合、複数表のデッドロックは、複数の同じ表にアクセスする複数のトランザクションが、暗黙のロックまたは明示的ロックを通じて各表を同じ順序でロックすれば回避できます。たとえば、すべてのアプリケーション開発者は、マスター表とディテール表の両方を更新するときに、まずマスター表をロックしてからディテール表をロックするという規則に従ってください。この規則を適切に設計し、すべてのアプリケーションがそれに従えば、デッドロックが起こる可能性はかなり低くなります。

あるトランザクションについて一連のロックが必要となることがわかっているときは、最も排他的な（最も共存不能な）ロックが最初に取得されるように考慮してください。

ロックの種類

Oracle は、データへの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用します。Oracle は、トランザクションのためにリソースを自動的にロックし、他のトランザクションが同じリソースの排他アクセスを要求する処理を行うことを防止します。なんらかのイベントが発生し、トランザクションがそのリソースを必要としなくなると、ロックは自動的に解除されます。

全操作を通じて、Oracle はロックされるリソースと実行される操作に応じて、様々な制限レベルの様々なタイプのロックを自動的に取得します。

Oracle のロックは次のいずれか 1 つに分類されます。

DML ロック（データ・ロック）	DML ロックはデータを保護します。たとえば、表ロックは表全体をロックし、行ロックは選択された行をロックします。
DDL ロック（ディクショナリ・ロック）	DDL ロックは、スキーマ・オブジェクトの構造、たとえば表とビューの定義を保護します。

内部ロックとラッチ	内部ロックとラッチは、データ・ファイルなどの内部データベース構造を保護します。内部ロックとラッチは完全に自動的です。
分散ロック	分散ロックは、Oracle Parallel Server の複数のインスタンス間に分散されたデータや他のリソースが一貫性を維持していることを保証します。分散ロックは、トランザクションではなくインスタンスによって保持され、Oracle Parallel Server のインスタンス間でリソースの現在の状態を伝達します。
パラレル・キャッシュ管理（PCM）ロック	パラレル・キャッシュ管理ロックは、バッファ・キャッシュ内の 1 つ以上のデータ・ブロック（表ブロックまたは索引ブロック）を対象とする分散ロックです。PCM ロックは、トランザクションのためには行をロックしません。

この章では、DML ロック、DDL ロックおよび内部ロックについて個別に説明します。

追加情報： 分散ロックと PCM ロックの詳細は、『Oracle8*i* 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のみが表の中の特定の行を更新すること、また、コミットされていないトランザクションに表への挿入操作が含まれている場合はその表が削除されないことが保証されます。

DML 操作では、2 つの異なるレベルでデータ・ロックを取得できます。1 つは特定の行に対するロック、もう 1 つは表全体に対するロックです。この後の各項では、行ロックと表ロックについて説明します。

注意： これ以降、それぞれのタイプのロックまたはロック・モードの後のカッコ内にある頭字語は、Oracle Enterprise Manager の Locks Monitor で使用される略称です。Oracle Enterprise Manager では、表ロックのモード（RS や SRX など）が示されるかわりに、すべての表ロックに TM と表示される可能性があります。

行ロック（TX）

Oracle が自動的に取得する DML ロックは、行レベルのロックのみです。1 つの文またはトランザクションで保持できる行ロックの数に制限はありません。また、Oracle が行レベルのロックをさらに粗い単位のロックに拡大することはありません。行ロックでは、最もきめの細かいロックが実現されるので、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・ロックを組み合せると、同じデータに関して複数のユーザーが競合するのは、同じ行にアクセスする場合のみになります。特に、次のような場合です。

- データの読込みは、同じデータ行の書込みを待機しない。
- データの書込みは、同じデータ行の読込みを待機しない（読込みのロックを要求する SELECT... FOR UPDATE を使用していない場合のみ）。
- 他の書込みが同時に同じ行を更新しようとする場合に限り、書込みは他の書込みを待機する。

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・ブロックへの書込みの待機が必要になることもあります。

INSERT 文、UPDATE 文、DELETE 文および FOR UPDATE 句を含む SELECT 文のいずれかで変更された行ごとに、トランザクションは排他 DML ロックを取得します。

変更された行は、ロックを保持しているトランザクションがコミットされるかロールバックされるまで、他のユーザーがその行を変更できないように**常に**排他的にロックされます。（ただし、インスタンス障害のためにトランザクションが終了すると、トランザクション全体が回復される前に、ブロック・レベルの回復によって行が使用可能になります。32-3 ページの「データベース・インスタンス障害」を参照してください。）前述の文の結果として、行ロックは、常に Oracle によって自動的に取得されます。

ある行について行ロックを取得したトランザクションは、その行に対応する表についての表ロックも取得します。表ロックがあると、現行トランザクション内のデータ変更を上書きする DDL 操作の競合が回避されます。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、27-27 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 文および LOCK TABLE 文の各 DML 文で表が変更される場合、トランザクションは表ロックを取得します。これらの DML 操作が表ロックを必要とするのは、トランザクションのために表への DML アクセスを確保する、およびトランザクションと競合する可能性がある DDL 操作を防止するという 2 つの目的のためです。すべての表ロックは同じ表に対する排他 DDL ロックを防止するので、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、27-28 ページの「排他 DDL ロック」を参照。）

表ロックは、行共有（RS）、行排他（RX）、共有（S）、共有行排他（SRX）および排他（X）のいずれかのモードで保持できます。表ロックのモードの制限は、他の表ロックが同じ表について取得し、保持できるモードを決定します。

表 27-2 に、文が取得する表ロックのモードと、各ロックで許可されている操作と禁止されている操作を示します。

表 27-2 表ロックのまとめ

SQL 文	表ロックのモード	許可されるロック・モード				
		RS	RX	S	SRX	X
SELECT...FROM table...	なし	可	可	可	可	可
INSERT INTO table ...	RX	可	可	不可	不可	不可
UPDATE table ...	RX	可 *	可 *	不可	不可	不可
DELETE FROM table ...	RX	可 *	可 *	不可	不可	不可
SELECT ... FROM table FOR UPDATE OF ...	RS	可 *	可 *	可 *	可 *	不可
LOCK TABLE table IN ROW SHARE MODE	RS	可	可	可	可	不可
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	可	可	不可	不可	不可
LOCK TABLE table IN SHARE MODE	S	可	不可	可	不可	不可
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	可	不可	不可	不可	不可
LOCK TABLE table IN EXCLUSIVE MODE	X	不可	不可	不可	不可	不可
	RS: 行共有 RX: 行排他 S: 共有 SRX: 共有行排他 X: 排他		* 別のトランザクションによって、競合する行ロックが保持されていない場合。そうでない場合は待機が発生。			

この後、制限レベルの低いものから高いものという順序で、表ロックの各モードについて説明します。それぞれの項では、表ロックのモード、トランザクションがそのモードで表ロックを取得する原因となるアクション、そのモードのロックで他のトランザクションに許可されるアクションと禁止されるアクションについて説明します。手動ロックの詳細は、27-30 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」の行共有表ロックが自動的に取得されます。

```
SELECT . . . FROM table . . . FOR UPDATE OF . . . ;

LOCK TABLE table IN ROW SHARE MODE;
```

行共有表ロックは、最も制限の緩やかなモードの表ロックであり、最高度の同時実行性を表に提供します。

許可される操作 : トランザクションによって保持される行共有表ロックでは、他のトランザクションは、同じ表の中の行に対して問合せ、挿入、更新、削除またはロックを同時に実行できます。そのため他のトランザクションは、同じ表について同時に行共有ロック、行排他ロック、共有ロックおよび共有行排他ロックを取得できます。

禁止される操作 : トランザクションによって保持される行共有表ロックは、他のトランザクションが次の文のみを使用して同じ表に排他書込みアクセスを実行するのを防止します。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

行排他表ロック（RX） 行排他表ロック（「副排他表ロック、SX」ともいう）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、同じ表の中の行に対して問合せ、挿入、更新、削除またはロックを同時に実行できます。そのため、行排他表ロックによって、複数のトランザクションが同時に、同じ表について行排他ロックと行共有表ロックを取得できます。

禁止される操作 : トランザクションによって保持される行排他表ロックは、他のトランザクションが排他的な読込みや書込みを実行するために表を手動でロックすることを防止します。そのため、他のトランザクションは、次の文を使用して同時に表をロックすることはできません。

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

共有表ロック（S） 共有表ロックは、次の文で指定される「表」について自動的に取得されます。

```
LOCK TABLE table IN SHARE MODE;
```

許可される操作 : トランザクションによって保持される共有表ロックでは、他のトランザクションによる、表の問合せ、SELECT . . . FOR UPDATE 文による特定行のロック、または LOCK TABLE . . . IN SHARE MODE 文の実行のみが許可されます。したがって、他のトランザクションによる更新は許可されません。複数のトランザクションが、同じ表について同時に共有表ロックを保持できます。ただし、この場合、トランザクションは表を更新できません（FOR UPDATE 句を指定した SELECT 文の結果として行ロックを保持できたとしても更新できません）。そのため、共有表ロックを保持しているトランザクションがその表を更新できるのは、他のトランザクションが同じ表について共有表ロックを保持していない場合のみです。

禁止される操作 : トランザクションによって保持される共有表ロックは、他のトランザクションが同じ表を変更するのを禁止します。また、他のトランザクションが次の文を実行することも禁止します。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

共有行排他表ロック（SRX） 共有行排他表ロック（「共有副排他表ロック、SSX」ともいう）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のみが、特定の表の共有行排他表ロックを取得できます。トランザクションによって保持される共有行排他表ロックでは、他のトランザクションは、問合せ、または FOR UPDATE 句を指定した SELECT 文による特定行のロックを許可されますが、表の更新は許可されません。

禁止される操作 : トランザクションによって保持される共有行排他表ロックは、他のトランザクションによる行排他表ロックの取得、および同じ表の変更を防止します。また、共有行排他表ロックでは、他のトランザクションが共有ロック、共有行排他ロックおよび排他表ロックを取得することが禁止されます。つまり、他のトランザクションが次の文を実行できなくなります。

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

排他表ロック（X） 排他表ロックは、最も多くの制限が課されるモードの表ロックであり、ロックを保持するトランザクションが表に排他的に書込みアクセスすることを許可します。排他表ロックは、次の文で指定された「表」について取得されます。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

許可される操作 : 1 つのトランザクションのみが、表の排他表ロックを取得できます。排他表ロックでは、他のトランザクションは同じ表に対する問合せのみを実行できます。

禁止される操作 : トランザクションによって保持されている排他表ロックは、他のトランザクションによる DML 文の実行とその表に対するロックの適用を禁止します。

DML 文について自動的に取得される DML ロック

ここまでで、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。この後の項では、各種の DML 操作を実行するために Oracle がどのようにデータを自動ロックするかをまとめます。

表 27-3 に、この後の項で説明する情報をまとめておきます。

表 27-3 DML 文が取得するロック

DML 文	行ロック	表ロックのモード
SELECT ... FROM table		
INSERT INTO table ...	X	RX
UPDATE table ...	X	RX
DELETE FROM table ...	X	RX
SELECT ... FROM table ... FOR UPDATE OF ...	X	RS
LOCK TABLE table IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X
	X: 排他 RX: 行排他	RS: 行共有 S: 共有 SRX: 共有行排他

問合せのデフォルト・ロック 問合せはデータを読み込むだけなので、他の SQL 文に対してほとんど干渉しない SQL 文です。INSERT 文、UPDATE 文および DELETE 文には、文の一部として暗黙の問合せが含まれている場合があります。問合せには、次の種類の文が含まれます。

```
SELECT

INSERT . . . SELECT . . . ;

UPDATE . . . ;

DELETE . . . ;
```

次の文は**含まれません**。

```
SELECT . . . FOR UPDATE OF . . . ;
```

次の特性は、FOR UPDATE 句を使用しないすべての問合せに当てはまります。

- 問合せはデータ・ロックを取得しない。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せて更新することができます。FOR UPDATE 句が指定されていない問合せでは、データ・ロックが取得されないので他の操作はブロックされないため、Oracle ではこの問合せを「非ブロッキング問合せ」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、UPDATE、DELETE および SELECT ... FOR UPDATE のデフォルト・ロック INSERT 文、UPDATE 文、DELETE 文および SELECT ... FOR UPDATE 文のロックの特性は次のとおりです。

- DML 文を含むトランザクションは、その文の修正対象の行に対して排他行ロックを取得する。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は、ロックされている行の更新や削除は実行できません。
- DML 文を含むトランザクションは、副問合せや暗黙の問合せ（WHERE 句の中にある問合せ）によって選択される行に対して行ロックを取得する必要がない。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、共有行排他ロックまたは排他表ロックをすでに保持している場合、行排他表ロックは取得されません。トランザクションが行共有表ロックをすでに保持している場合、Oracle はこのロックを行排他表ロックに自動的に変換します。

DDL ロック（ディクショナリ・ロック）

処理中の DDL 操作によってスキーマ・オブジェクト（表など）が影響を受けたり参照される間、そのオブジェクトの定義は DDL ロックによって保護されます。（DDL 文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成する場合を考えます。そのユーザーの単一文トランザクションのために、Oracle はプロシージャ定義で参照されるすべてのスキーマ・オブジェクトについての DDL ロックを自動的に取得します。その DDL ロックにより、プロシージャのコンパイル完了前に、プロシージャで参照されるオブジェクトの変更または削除が防止されます。

Oracle は、ディクショナリ・ロックを必要とする DDL トランザクションのために、ディクショナリ・ロックを自動的に取得します。ユーザーは DDL ロックを明示的に要求できません。DDL 操作中には、修正や参照の対象となる個々のスキーマ・オブジェクトのみがロックされます。データ・ディクショナリ全体がロックされることはありません。

DDL ロックは、排他 DDL ロック、共有 DDL ロックおよびブレ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、リソースの排他 DDL ロックが必要です。たとえば ALTER TABLE 操作で表に列を追加している間は、DROP TABLE 操作でその表を削除できません。その逆も同様です。

排他 DDL ロックの取得では、別の操作によってすでにそのスキーマ・オブジェクトに対して別の DDL ロックが保持されている場合、その取得は古い DDL ロックが解除されるまで待機し、その後に処理されます。

また、DDL 操作は変更対象のスキーマ・オブジェクトに対する DML ロック（データ・ロック）も取得します。

共有 DDL ロック

ある種の DDL 操作では、競合する DDL 操作によって破壊的な干渉が起きないようにし、かつ一方では類似の DDL 操作についてデータの同時実行性を確保するため、リソースの共有 DDL ロックが必要です。たとえば CREATE PROCEDURE 文の実行時には、その文を含むトランザクションは、参照されるすべての表について共有 DDL ロックを取得します。他のトランザクションは同じ表を参照するプロシージャを同時に作成できるため、同じ表について同時実行の共有 DDL ロックを取得できます。ただし、参照中の表について排他 DDL ロックを取得できるトランザクションはありません。また、参照中の表を変更または削除できるトランザクションはありません。その結果、共有 DDL ロックを保持するトランザクションでは、参照中のスキーマ・オブジェクトの定義がそのトランザクションの存続期間にわたって変化しないことが保証されます。

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 および CREATE TABLE（CLUSTER パラメータが含まれていない場合）を含む DDL 文のスキーマ・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 21 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 操作も許可するためにブレーク（中断）できるため、「ブレーク可能解析ロック」と呼ばれます。

解析ロックは SQL 文の実行の解析フェーズで取得され、共有プールの中にその文の共有 SQL 領域が残っている限り保持されます。

DDL ロックの存続期間

DDL ロックの存続期間は、DDL ロックの種類によって異なります。排他ロックと共有 DDL ロックは、DDL 文実行の継続中と自動コミット中ずっと存続します。解析ロックは、対応する SQL 文が共有プールに残っている限り存続します。

DDL ロックとクラスタ

クラスタに対する DDL 操作は、クラスタおよびそのクラスタ内のすべての表とスナップショットに対して排他 DDL ロックを取得します。クラスタ内の表やスナップショットに対する DDL 操作は、その表やスナップショットに対する共有 DDL ロックまたは排他 DDL ロックに加えて、そのクラスタに対する共有ロックも取得します。クラスタに対する共有 DDL ロックは、最初の操作の処理中に、別の操作がそのクラスタを削除するのを防止します。

ラッチと内部ロック

ラッチと内部ロックは、内部データベース構造とメモリー構造を保護します。ユーザーは、それらの出現や存続期間を制御する必要がないため、そのいずれもユーザーからはアクセスできません。次の説明内容は、Oracle Enterprise Manager または SQL*Plus の LOCKS モニターと LATCHES モニターについて理解する上で役立ちます。

ラッチ

ラッチは、システム・グローバル領域（SGA）内の共有データ構造を保護するための単純な下位レベルの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する、非常に短い間のみラッチを取得します。ラッチのインプリメンテーション（特に、プロセスがラッチを待機するかどうか、およびどれくらいの時間待機するか）は、オペレーティング・システムに依存します。

内部ロック

内部ロックは、ラッチよりも高水準で複雑なメカニズムであり、いろいろな目的のために機能します。

ディクショナリ・キャッシュ・ロック これらのロックの存続期間は非常に短く、ディクショナリ・キャッシュ内のエントリが修正されたり使用されている間、そのエントリについて保持されます。これらのロックは、解析されている文が矛盾したオブジェクト定義を参照しないことを保証します。

ディクショナリ・キャッシュ・ロックは、共有または排他で保持されます。共有ロックは、解析が完了すると解除されます。排他ロックは、DDL 操作が完了すると解除されます。

ファイルとログの管理ロック これらのロックは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 つのプロセスのみがそれを変更できるようにします。別のロックは、REDO ログ・ファイルの使用とアーカイブを調整します。データベースを複数インスタンスが共有モードでマウントしたり、1 つのインスタンスが排他モードでマウントすることを保証するために、データ・ファイルがロックされます。ファイルとログのロックはファイルの状態を示すので、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 を使用している場合は特に重要です。

> **追加情報：** ロックの詳細は、『Oracle8*i* 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、整合性および文レベルの読込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。ただし、デフォルトのロック・メカニズムを置き換えることもできます。デフォルト・ロックの置換えは、次のような状況で有効です。

- アプリケーションで、トランザクション・レベルの読込み一貫性または「反復可能読込み」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成されるデータが、他のトランザクションによる変更を反映せず、そのトランザクションの存続期間にわたって一貫性を維持しなければならない場合。トランザクション・レベルの読込み一貫性は、明示的ロック、読込み専用トランザクションまたは直列可能トランザクションを使用するか、デフォルトのロックを変更することによって実現できます。
- 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る場合。

Oracle の自動ロックは、次の 2 つのレベルで置換できます。

トランザクション　Oracle のデフォルト・ロックは、次の SQL 文を含むトランザクションによって上書きされます。

- SET TRANSACTION ISOLATION LEVEL コマンド
- LOCK TABLE コマンド（表をロックするコマンド、またはビューを使用する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

これらの文で取得されたロックは、トランザクションがコミットまたはロールバックされた後で解除されます。

セッション　セッションでは、ALTER SESSION 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。

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などの基準を満たすものにする必要があります。

追加情報： SQL 文 LOCK TABLE および SELECT ... FOR UPDATE の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

明示的ロックの下でのデータ同時実行性の例

LOCK TABLE 文や FOR UPDATE 句を指定した SELECT 文が使用されるときに、Oracle がデータの同時実行性、整合性および一貫性をどのように維持するかを次に示します。

注意： 簡単にするため、ORA-00054 のメッセージ・テキストは記載しません。そのメッセージ・テキストは「リソースビジー、NOWAIT が指定されていません。」というものです。ユーザーが入力するテキストは**太字**になっています。

トランザクション 1	時点	トランザクション 2
LOCK TABLE scott.dept **IN ROW SHARE MODE;** 文が処理される。	1	
	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	
	9	**LOCK TABLE scott.dept** **IN EXCLUSIVE MODE** **NOWAIT;** ORA-00054
	10	**LOCK TABLE scott.dept** **IN SHARE ROW EXCLUSIVE** **MODE NOWAIT;** ORA-00054

トランザクション 1	時点	トランザクション 2
	11	**LOCK TABLE scott.dept** **IN SHARE ROW EXCLUSIVE** **MODE NOWAIT;** ORA-00054
	12	**UPDATE scott.dept** **SET loc = 'NEW YORK'** **WHERE deptno = 20;** 1 行処理。
	13	**ROLLBACK;**
SELECT loc **FROM scott.dept** **WHERE deptno = 20** **FOR UPDATE OF loc;** LOC - - - - - - DALLAS 1 行選択。	14	
	15	**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）
ROLLBACK;	16	
	17	1 行処理。 （競合するロックが解除された） **ROLLBACK;**
LOCK TABLE scott.dept **IN 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。

トランザクション 1	時点	トランザクション 2
	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。
	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

トランザクション 1	時点	トランザクション 2
	31	**LOCK TABLE scott.dept** **IN ROW EXCLUSIVE** **MODE NOWAIT;** ORA-00054
	32	**LOCK TABLE scott.dept** **IN SHARE MODE NOWAIT;** ORA-00054
	33	**SELECT loc** **FROM scott.dept** **WHERE deptno = 20;** LOC - - - - - - DALLAS 1 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	
	40	**LOCK TABLE scott.dept** **IN EXCLUSIVE MODE;** ORA-00054

トランザクション 1	時点	トランザクション 2
	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）
UPDATE scott.dept **SET deptno = 30** **WHERE deptno = 20;** 1 行処理。	47	
COMMIT;	48	
	49	0 行選択。 （T1 が競合するロックを解除）
SET TRANSACTION READ ONLY;	50	

トランザクション 1	時点	トランザクション 2
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - BOSTON	51	
	52	**UPDATE scott.dept** **SET loc = 'NEW YORK'** **WHERE deptno = 10;** 1 行処理。
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - BOSTON （T1 から未コミット・データは見えない）	53	
	54	**COMMIT;**
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - （T2 のコミット後も同じ結果が参照される）	55	
COMMIT;	56	
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - NEW YORK （コミットされたデータが参照される）	57	

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用して次のような処理をする文を PL/SQL ブロックに含めることができます。

- 特定のタイプのロックを要求する。
- 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 ロック管理サービスの詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

28

データの整合性

Does one's integrity ever lie in what he is not able to do?

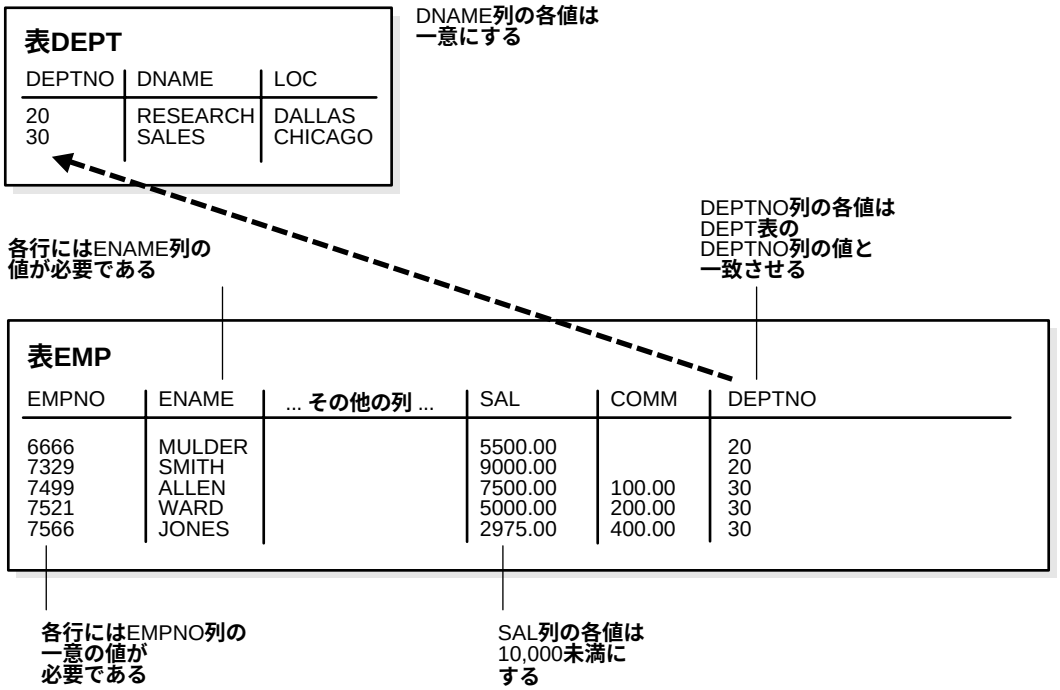
Flannery O'Connor: *Wise Blood*

この章では、データベースに関連する業務規則を施行し、表への無効な情報入力を防止する方法について説明します。この章の内容は、次のとおりです。

データ整合性の定義

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 28-1 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規則について考えてみましょう。

図 28-1 データ整合性の例



それぞれの表のある列には、その列に入っているデータに制約をかける固有の規則があることに注意してください。

データ整合性のタイプ

この項では、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規則について説明します。

NULL

NULL は、単一の列に対して定義される規則で、その列に NULL が入っている（値がない）行の挿入や更新を許可または禁止します。

一意の列値

列（または列の集合）に対して定義される一意の列値は、その列（または列の集合）に含まれる値が一意である場合に限って、行の挿入または更新を許可します。

主キー値

キー（列または列の集合）に対して定義される主キー値は、表の中の各行がキーの値によって一意に識別できることを指定します。

参照整合性

1 つの表のキー（列または列の集合）に対して定義される規則であり、そのキーの値が関連する表のキーの値（参照値）と一致することを保証します。

また、参照整合性には、参照先のデータに対してどのようなタイプのデータ操作を許可するか、およびその操作の結果として依存データがどのような影響を受けるかについて指示する規則が含まれています。参照整合性に関連する規則は、次のとおりです。

RESTRICT（制限）	参照先のデータの更新または削除を禁止する。
SET TO NULL（NULL 設定）	参照先のデータが更新または削除されると、対応する依存データがすべて NULL に設定される。
SET TO DEFAULT（デフォルト設定）	参照先のデータが更新または削除されると、対応する依存データがすべてデフォルト値に設定される。
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 を使用します。）

複雑な整合性チェック

複雑な整合性チェックは、列（または列の集合）に対して設定されるユーザー定義の規則であり、その列の値に基づいて、行の挿入、更新、削除を許可または禁止します。

Oracle がデータの整合性を施行する方法

Oracle では、前述のデータ整合性規則をそれぞれ定義し、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・トリガーを使用して簡単に定義できます。

整合性制約

整合性制約は、表の列に規則を定義する宣言手法です。Oracle では、次の整合性制約をサポートしています。

- NOT NULL 制約。列に含まれる NULL に関連した規則です。
- UNIQUE キー制約。一意の列値に関連した規則です。
- PRIMARY KEY 制約。1 次識別値に関連した規則です。
- FOREIGN KEY 制約。参照整合性に対応付けられた規則です。現在、Oracle では、次の参照整合性アクションを定義する際に FOREIGN KEY 整合性制約の使用をサポートしています。
  - UPDATE No Action と DELETE No Action
  - DELETE CASCADE
  - DELETE SET NULL
- CHECK 制約。複雑な整合性規則の場合に使用します。

注意： 子表と親表が分散データベースの別のノードにある場合、宣言整合性制約を使用して参照整合性の施行はできません。ただし、データベース・トリガーを使用して、分散データベースで参照整合性を施行することはできます（後述）。

データベース・トリガー

Oracle では、データベース・トリガー（挿入、更新または削除の各操作で自動的に起動されるストアド・データベース・プロシージャ）を使用することによって、宣言以外の形の整合性制約を施行できます。データの整合性の施行で使用されるデータベース・トリガーの詳細と例は、第 20 章「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、整合性制約を使用します。整合性制約を定義することによって、データベースの情報に関連付ける業務規則を施行できます。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。この表では、どの行でもこの列に 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 文または UPDATE 文がこの整合性制約に違反すると、Oracle はその文をロールバックし、通知エラー・メッセージを戻します。

Oracle で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業務規則を施行する。
- ストアド・プロシージャを使用してデータへのアクセスを完全に制御する。
- トリガーされるストアド・データベース・プロシージャを使用して業務規則を施行する（第 20 章「トリガー」を参照）。

宣言の容易さ

整合性制約は、SQL コマンドを使用して定義します。表を定義したり変更したりするときに追加のプログラミングをする必要はありません。SQL 文は簡単に記述でき、プログラミング・エラーが少なくてすみ、Oracle が整合性制約の機能を制御します。これらの理由で、宣言整合性制約は、アプリケーション・コードやデータベース・トリガーよりも優れています。また、ストアド・プロシージャを使用してデータ・アクセスを制御することによってデータの整合性を解決するという方法もありますが、整合性制約の場合は非定型のデータ・アクセスという柔軟性を犠牲にすることがないため、宣言アプローチを使用するほうがストアド・プロシージャより優れています。

規則の集中化

整合性制約は、表に対して（アプリケーションに対してではなく）定義され、データ・ディクショナリに格納されます。どのアプリケーションから入力されるデータも、表に対応付けられている同じ整合性制約を遵守する必要があります。業務規則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はありますが、それを実現する方法は、整合性制約で使用されている宣言アプローチよりはるかに複雑です。

アプリケーション開発の生産性の最大化

整合性制約によって施行される業務規則を変更する場合、管理者が整合性制約を変更するだけで、すべてのアプリケーションは変更後の制約を自動的に遵守するようになります。それに対して、それぞれのデータベース・アプリケーションのコードで業務規則を施行する場合、開発者はすべてのアプリケーションのソース・コードを修正して、その修正したアプリケーションを再コンパイルし、デバッグしてテストする必要があります。

ユーザーへの即時フィードバック

Oracle は、各整合性制約ごとに、特定の情報をデータ・ディクショナリに格納します。Oracle が SQL 文を実行しチェックする前でも、データベース・アプリケーションは、その情報を使用して整合性制約の違反をユーザーに即時にフィードバックするように、データベース・アプリケーションを設計できます。たとえば SQL*Forms アプリケーションは、文を発行する前でも、データ・ディクショナリに格納されている整合性制約の定義を使用して、フォームのフィールドに入力される値の違反をチェックできます。

優れたパフォーマンス

整合性制約宣言の意味は明確に定義されており、個々の宣言規則ごとにパフォーマンスの最適化が実現されます。Oracle 問合せオプティマイザは、宣言を使用して、データをより詳細に認識し、問合せのパフォーマンスを全体として向上させます。（また、アプリケーション・コードとデータベース・トリガーから整合性規則を取り去ることによって、必要なときのみチェックが行われることが保証されます。）

データ・ロードの柔軟性と整合性違反の識別

大量のデータをロードする場合、制約チェックによるオーバーヘッドをなくすために、整合性制約を一時的に使用禁止にできます。データのロードが完了した後、整合性制約を使用可能にするのは簡単であり、整合性制約に違反した新しい行を別の例外表に自動的に報告させることもできます。

整合性制約のパフォーマンス・コスト

データ整合性の規則を施行することによる利点は、パフォーマンスを多少犠牲にして初めて獲得できます。一般に、整合性制約を組み込むことの「コスト」は、多くても、制約を評価する SQL 文を実行するのと同じです。

整合性制約のタイプ

列値の入力時の制限として課すことのできる整合性制約には、次のものがあります。

- NOT NULL 整合性制約
- UNIQUE キー整合性制約
- PRIMARY KEY 整合性制約
- 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）を使用できます。NOT NULL 制約がある場合、表の列値が NULL でないということが求められます。たとえば、EMP 表のすべての行で ENAME 列に必ず値を入力するように求める NOT NULL 制約を定義できます。

図 28-2 に、NOT NULL 整合性制約を示します。

図 28-2 NOT NULL 整合性制約

表EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17–DEC–85	9,000.00		20
7499	ALLEN	VP_SALES	7329	20–FEB–90	7,500.00	100.00	30
7521	WARD	MANAGER	7499	22–FEB–90	5,000.00	200.00	30
7566	JONES	SALESMAN	7521	02–APR–90	2,975.00	400.00	30

NOT NULL CONSTRAINT (行のこの列にはNULLを使用できない)

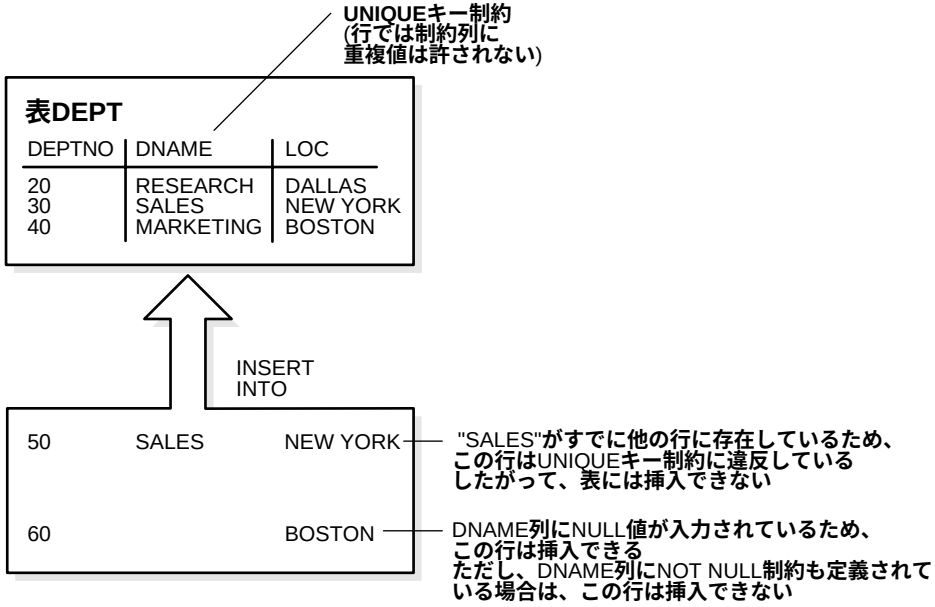
NOT NULL制約なし (どの行でもこの列にNULLを使用できる)

UNIQUE キー整合性制約

UNIQUE キー整合性制約では、列または列の集合（キー）のすべての値が一意である必要があります。つまり、指定した列または列の集合について、表の 2 つの行の値が重複することは許されません。

たとえば、図 28-3 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 28-3 UNIQUE キー制約

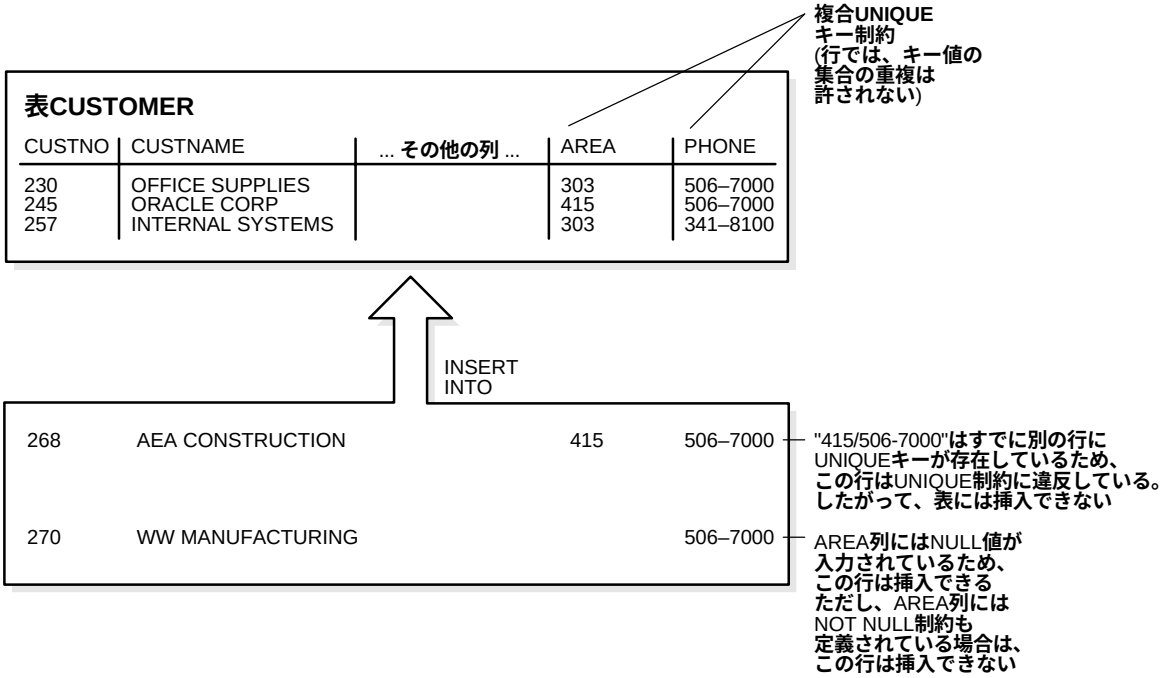


一意キー

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」のシノニムとして使用されることがありますが、「キー」という用語は整合性制約の定義に使用されている列または列の集合のみを指して使用されるので、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 28-4 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 28-4 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」を同じ表の中に重複しては入力できません。これにより、誤って電話番号が重複するのを避けられます。

UNIQUE キー制約と索引

Oracle は、索引を使用して一意の整合性制約を施行します。（図 28-4 で、Oracle は、複合一意キーに対する一意索引を暗黙的に作成することにより、UNIQUE キー制約を施行しています。）したがって、複合 UNIQUE キー制約には、複合索引に対して課されているのと同じ制限があります。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ることはできません。UNIQUE キー制約が作成されたときに使用できる索引がある場合、制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

UNIQUE キー整合性制約と NOT NULL 整合性制約を組み合わせる

図 28-3 と図 28-4 では、同じ列に NOT NULL 制約も定義するのでない限り、UNIQUE キー制約は NULL の入力を許可します。実際、NULL はどの値とも等しいとみなされることがないので、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、一般的に使用されます。これらの組合せにより、ユーザーは一意キーに必ず値を入力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もなくなります。

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 つ以上の列グループの値は、その行の一意識別子を構成します。つまり、それぞれの行は、その主キー値によって指定されます。

Oracle に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）。

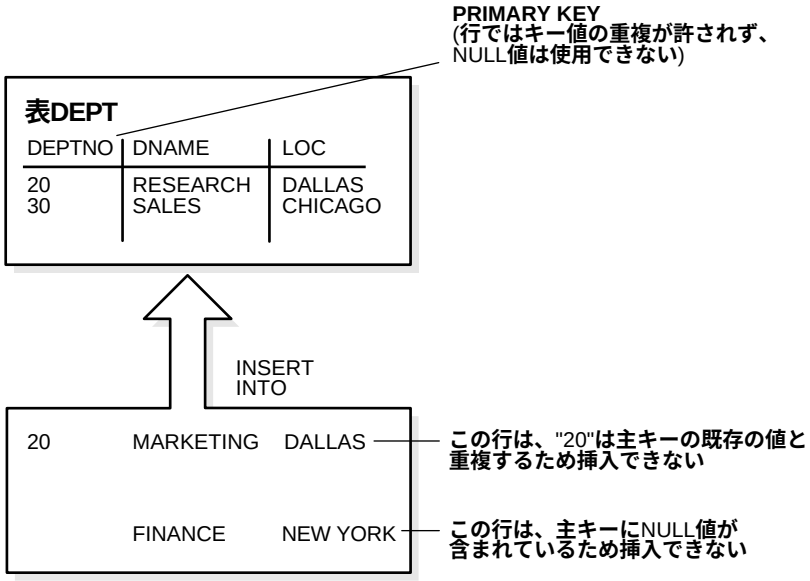
主キー

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、次の利点を考慮して、すべての表に主キーを指定してください。

- 表のそれぞれの行を一意に識別できる。
- 重複する行が表に存在しない。

図 28-5 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 28-5 主キー制約



PRIMARY KEY 制約と索引

Oracle では、すべての PRIMARY KEY 制約が索引を使用して施行されます。図 28-5 では、DEPTNO 列に対して作成された主キー制約は、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成する。
- その列に対して NOT NULL 制約を暗黙のうちに作成する。

Oracle は、索引を使用し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、制約を作成に使用する CREATE TABLE 文または ALTER TABLE 文に ENABLE 句を組み込んで、この索引に格納オプションを指定することもできます。主キー制約の作成時に使用できる索引がある場合、主キー制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

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ることが必要です。参照整合性規則により、これらの関係が保たれることが保証されます。

参照整合性制約に関連する用語は、次のとおりです。

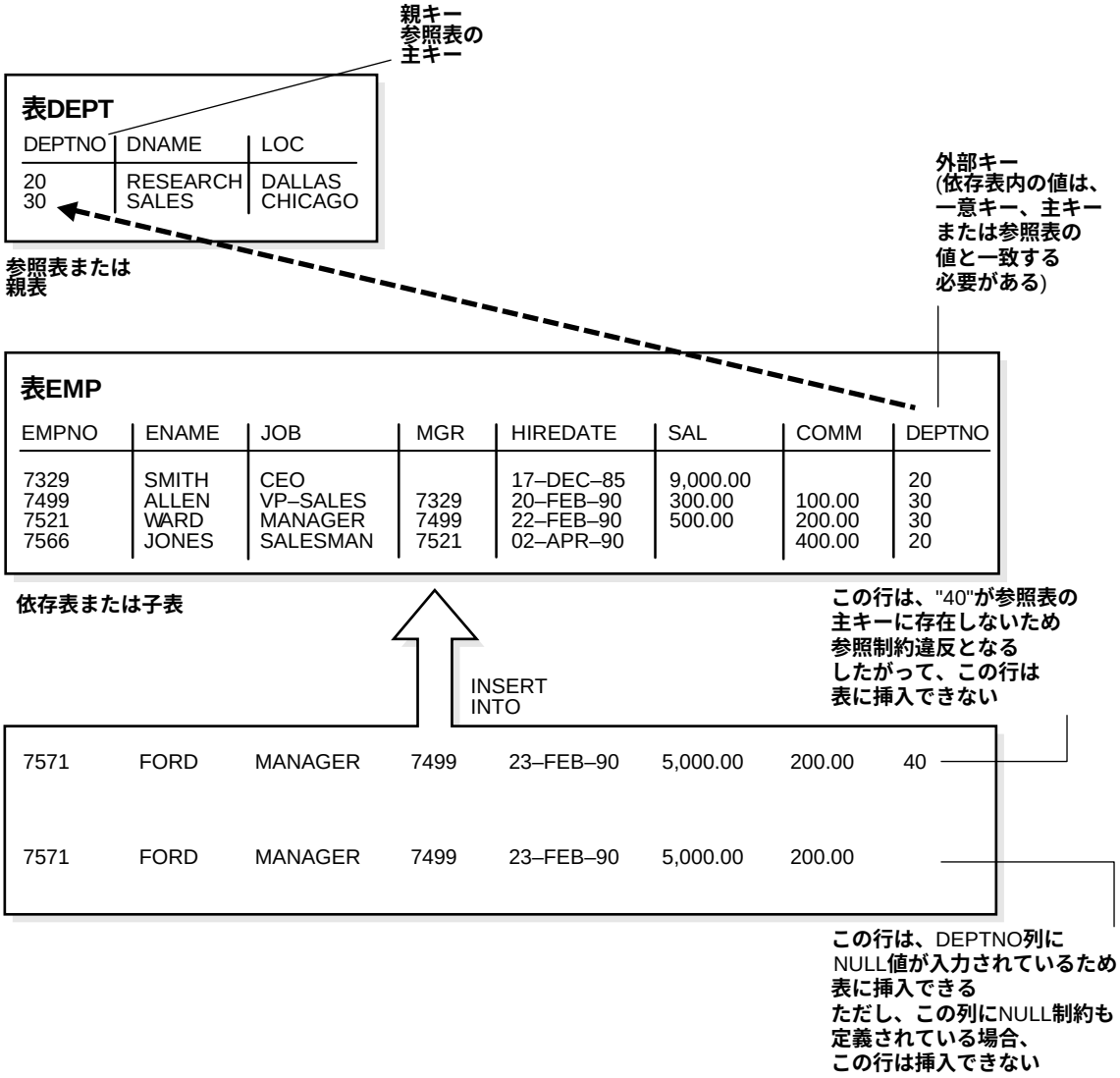
外部キー	参照整合性制約の定義に含まれている列または列の集合のうち、参照キー（次を参照）を参照するもの。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。

参照整合性制約では、表の各行の外部キー値は親表の値と一致している必要があります。

図 28-6 に、EMP 表の DEPTNO 列に対して定義された外部キーを示します。これにより、この列のすべての値が DEPT 表の主キー（すなわち DEPTNO 列）の値と一致することが保証されます。このため、EMP 表の DEPTNO 列に間違った部門番号が存在することはありません。

外部キーは、複数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 28-6 参照整合性制約

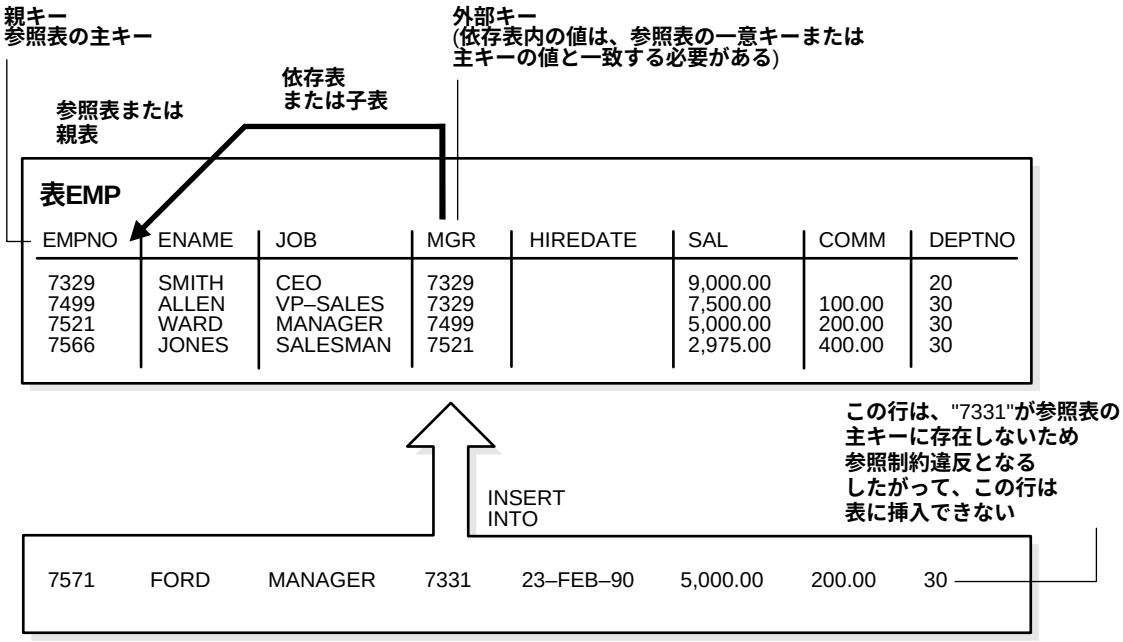


自己参照型整合性制約

図 28-7 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 28-7 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ある必要がある）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 28-7 単一表の参照制約



NULL と外部キー

リレーショナル・モデルでは、外部キーの値が、参照主キーか一意キーの値、または NULL のどちらかに一致することが許されています。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）規格では、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かまいません。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）を使用し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、すべて非 NULL または一部 NULL という値を指定できます。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、DELETE No Action および DELETE CASCADE です。

注意： Oracle の FOREIGN KEY 整合性制約でサポートされていない他の参照アクションは、データベース・トリガーを使用して施行できます。詳細は、第 20 章「トリガー」を参照してください。

UPDATE No Action と DELETE No Action　No Action（デフォルト）オプションは、結果のデータが参照整合性制約に違反する場合に参照キー値を更新または削除できないことを指定します。たとえば、主キー値が外部キーの中の値によって参照されている場合は、依存データであるため、参照先の主キー値を削除できません。

DELETE Cascade　DELETE CASCADE アクションは、参照キー値を含む行が削除された場合に、子表のうち依存外部キー値を含むすべての行も削除すること（DELETE CASCADE）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。

DELETE Set Null DELETE Set Null アクションは、参照キー値を含む行が削除された場合に、子表のうち依存外部キー値を含むすべての行の値を NULL に設定すること（SET NULL）を指定します。たとえば、EMPNO が EMP 表内の MGR を参照する場合は、管理者を削除すると、その管理者の部下である従業員全員の行の MGR 値が NULL に設定されます。

参照アクションに関する DML 制限 表 28-1 に、親表の主キー値または一意キー値および子表の外部キー値に対する異なる参照アクションごとに可能な DML 文の概要を示します。

表 28-1 UPDATE No Action と DELETE No Action で許される DML 文

DML 文	親表に対して発行	子表に対して発行
INSERT	親キー値が一意であれば常に発行できる。	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 の場合にのみ発行できる。
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。
DELETE Set Null	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 TRUE または UNKNOWN であることが必要です。DML 文の結果が、CHECK 制約の条件が FALSE に評価される場合、その文はロールバックされます。

チェック条件

CHECK 制約を使用すると、チェック条件を指定することによって、特定の目的の、または洗練された整合性規則を施行できます。CHECK 制約の条件には次のような制限があります。

- 挿入または更新する行の値を使用し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 関数の SYSDATE、UID、USER または 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 設定を使用します。これらのデフォルトは、CHECK 制約定義の中でそれらのファンクションに NLS パラメータを明示的に指定すれば、上書きできます。

追加情報： NLS 機能の詳細は、『Oracle8*i* NLS ガイド』を参照してください。

複数の CHECK 制約

単一の列に、定義の中でその列を参照する複数の CHECK 制約を指定できます。1 つの列に対して定義できる CHECK 制約の数に制限はありません。

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と役立ちます。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、28-15 ページの図 28-7 で定義されたものである。
- 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 列のみを対象にします。

EMP 表に最初の行を挿入する場合を考えます。現在は行が存在しません。MGR 列が EMPNO 列の既存の値を参照できない場合、どうすれば行を入力できるでしょうか？この操作を実行するには、次の 3 つの方法が考えられます。

- MGR 列に NOT NULL 制約が定義されていない場合には、第 1 行の MGR 列に NULL を入力できる。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の入力を許可するために、Oracle は、文を実行（つまり、新しい行を挿入）してから、その新しい行の MGR 列に対応する EMPNO が入っている行がその表に含まれているかどうかをチェックします。
- SELECT 文のネストを伴う INSERT 文など、複数行を挿入する INSERT 文により、相互に参照しあう行を挿入できる。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、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。28-20 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、次の使用例を考えます。

- 会社が買収された。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。

この時点では、この表は図 28-8 のような状態になっています。

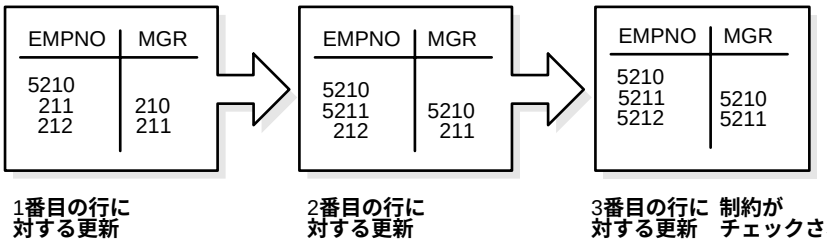
図 28-8 更新前の EMP 表

EMPNO	MGR
210	
211	210
212	211

```
UPDATE emp
  SET empno = empno + 5000,
      mgr = mgr + 5000;
```

制約は、各 MGR 値が EMPNO 値と一致するかどうかを検査するように定義されていますが、Oracle では文の完了後に制約チェックを効率的に実行されるので、この文は有効です。図 28-9 に、制約チェックの前に SQL 文全体のアクションが実行される流れを示します。

図 28-9 制約チェック



この項の例は、INSERT 文と UPDATE 文を実行した場合の制約チェックのメカニズムを示すものでした。UPDATE 文、INSERT 文および DELETE 文など、すべてのタイプの DML 文でも、同じメカニズムを使用しています。

また、これらの例は、自己参照型整合性制約を使用してチェックのメカニズムを示すものでした。NOT NULL、UNIQUE キーおよび 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用されています。

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」では、制約が満たされているかどうかのチェックがコミット時にしか実行されない。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、その文は即時にロールバックされます。

制約によって「アクション」（DELETE CASCADE など）が発生する場合は、遅延制約か即時制約かに関係なく、そのアクションは、アクションを発生させた文の一部として実行されます。

制約の属性

制約は、「遅延可能」または「遅延不可」、および「初期遅延」または「初期即時」のどちらかに定義できます。これらの属性は、制約ごとに異なるものを指定できます。それらの定義は、CONSTRAINT 句の中で次のキーワードを使用して指定します。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE

> **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*i* SQL リファレンス』を参照してください。

制約について、追加、削除、使用可能と使用禁止の切替えまたは妥当性検査を実行できます（28-21 ページの「制約の状態」を参照）。また、制約の属性も変更できます（28-23 ページの「制約の状態変更」を参照）。

SET CONSTRAINTS モード

SET CONSTRAINTS 文は、特定のトランザクションのために、制約を DEFERRED または IMMEDIATE のどちらかに指定します（構文的にも意味的にも ANSI SQL92 規格に準拠）。この文を使用すると、制約名のリストまたはすべての制約（ALL）を指定して、そのモードを設定できます。

SET CONSTRAINTS モードは、トランザクションの存続期間中、または別の SET CONSTRAINTS 文によってモードが再設定されるまで有効です。

SET CONSTRAINTS ... IMMEDIATE は、指定された制約を各制約文の実行直後にチェックするように指示します。Oracle は、トランザクション内の遅延制約をチェックし、次にそのトランザクション内のそれ以降の文の制約を即時にチェックします（チェック制約が一貫しており、他に SET CONSTRAINTS 文が発行されない場合）。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ません）。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用します。遅延可能制約を削除しても、その索引は残ります。（制約を使用禁止にしても格納情報は残るので、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用します。

制約の状態

CREATE TABLE 文または ALTER TABLE 文を使用すると、整合性制約を表レベルで使用可能または使用禁止にできます。また、次のように、制約を VALIDATE または NOVALIDATE に設定し、ENABLE または DISABLE と組み合わせることもできます。

- ENABLE を指定すると、入ってくるすべてのデータが制約に準拠していることが保証される。
- DISABLE を指定すると、入ってくるデータは、制約に準拠しているかどうかに関係なく許可される。
- VALIDATE を指定すると、既存のデータが制約に準拠していることが保証される。
- NOVALIDATE は、一部の既存データが制約に準拠していない可能性があることを意味する。

さらに、次のようになります。

- ENABLE VALIDATE は ENABLE と同じである。制約がチェックされ、すべての行が保持されることが保証されます。
- ENABLE NOVALIDATE は、制約はチェックされるが、すべての行について TRUE でなくても許されることを意味する。これにより、既存の行が制約に違反することは許され、しかも、新しい行や変更した行に対してはすべて有効であることが保証されます。

  ALTER TABLE 文の ENABLE NOVALIDATE オプションを使用すると、表内のすべてのデータの妥当性を最初に検査せずに、使用禁止にした制約について制約チェックを再開できます。
- DISABLE NOVALIDATE は DISABLE と同じである。制約はチェックされず、TRUE でなくてもかまいません。
- DISABLE VALIDATE を指定すると制約が使用禁止になり、制約の索引が削除され、対象となる列の変更は許されなくなります。

  UNIQUE 制約の場合、DISABLE VALIDATE 状態では、ALTER TABLE コマンドの EXCHANGE PARTITION オプションを使用して、非パーティション表からパーティション表にデータを効率よくコピーできます。

これらの状態間の遷移は、次の規則で制御されます。

- NOVALIDATE が指定されていない限り、ENABLE は暗黙的に VALIDATE を意味する。
- VALIDATE が指定されていない限り、DISABLE は暗黙的に NOVALIDATE を意味する。
- VALIDATE と NOVALIDATE には、ENABLE 状態と DISABLE 状態に関するデフォルトの含意はない。
- 一意キーまたは主キーが DISABLE 状態から ENABLE 状態に移る場合に、既存の索引がなければ一意キーが自動的に作成される。同様に、一意キーまたは主キーが ENABLE から DISABLE に移る場合に、一意索引で使用可能にされていれば、一意索引は削除される。
- 制約が NOVALIDATE 状態から VALIDATE 状態に移った場合は、すべてのデータをチェックする必要がある。（この操作は、きわめて低速の場合があります。）ただし、VALIDATE から NOVALIDATE に移っても、データがチェックされたことが無視されるだけです。
- 単一の制約を ENABLE NOVALIDATE 状態から ENABLE VALIDATE 状態に移動すると、読込み、書込みまたは他の DDL 文は阻止されない。パラレルに実行できます。

追加情報： ENABLE、DISABLE、VALIDATE および NOVALIDATE CONSTRAINT オプションの使用方法の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

制約の状態変更

ALTER TABLE コマンドの MODIFY CONSTRAINT オプションを使用すると、次の制約の状態を変更できます。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE
- RELY または NORELY
- USING INDEX . . .
- ENABLE または DISABLE
- VALIDATE または NOVALIDATE
- EXCEPTIONS INTO . . .

追加情報： これらの制約の状態の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

29

データベース・アクセスの制御

Allow me to congratulate you, sir.You have the most totally closed mind 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、データベース・ユーザーおよびセキュリティ・ドメイン
- ユーザーの認証
- ユーザー表領域の設定と割当て制限
- ユーザー・グループ PUBLIC
- ユーザー・リソースの制限とプロファイル
- ライセンス

データベース・セキュリティ

データベース・セキュリティには、データベースとデータベースに格納されているオブジェクトに対するユーザー・アクションを許可したり禁止する機能が関係しています。Oracle では、スキーマとセキュリティ・ドメインを使用して、データへのアクセスを制御したり、さまざまなデータベース・リソースの使用を制限します。

Oracle では、包括的な任意アクセス制御が提供されています。「任意アクセス制御」は、特定のオブジェクトに対するすべてのユーザー・アクセスを、権限によって調整します。権限とは、特定のオブジェクトに規定の方法でアクセスするための許可です。たとえば、表への問合せを実行する許可はその一例です。権限は、他のユーザーの裁量で任意に付与されるものなので、「任意セキュリティ機能」という語を使用しています。権限の詳細は、第 30 章「権限、ロールおよびセキュリティ・ポリシー」を参照してください。

スキーマ、データベース・ユーザーおよびセキュリティ・ドメイン

「ユーザー」（「ユーザー名」ともいう）は、データベース内で定義されている名前であり、この名前を使用してオブジェクトに接続したりアクセスできます。「スキーマ」は、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、プロシージャおよびパッケージなどのオブジェクトの集合に名前を付けたものです。スキーマとユーザーは、データベース管理者がデータベース・セキュリティを管理するのに役立ちます。

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 またはプリコンパイラ・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用して接続する必要があります。

データベース・ユーザーを作成すると、同じ名前の対応するスキーマがこのユーザー用に作成されます。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、交換可能な語として使用されることがあります。

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成したり、既存のデータベースを変更する場合、セキュリティ管理者は、ユーザーのセキュリティ・ドメインについていくつか決定する必要があります。たとえば、次のようなことを決定します。

- ユーザー認証情報を、データベース、オペレーティング・システムまたはネットワークの認証サービスのどれを使用して維持管理するか。
- ユーザーのデフォルト表領域と一時表領域の設定。
- ユーザーがアクセス可能な表領域のリスト（存在する場合）と、そのリストに含まれている表領域に対応付ける割当て制限。

- ユーザーのリソース制限プロファイル。ユーザーが使用できるシステム・リソースの量を制限します。
- データベース操作の実行に必要な、オブジェクトへの適切なアクセス権をユーザーに提供する権限、ロールおよびセキュリティ方針。

この章では、このうち最初の 4 つのセキュリティ・ドメイン・オプションについて説明します。権限、ロールおよびセキュリティ方針については、第 30 章「権限、ロールおよびセキュリティ・ポリシー」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： 特別なユーザーである SYS および SYSTEM の詳細と、セキュリティ管理者の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

ユーザーの認証

データベース・ユーザー名が無許可で使用されないようにするため、Oracle では次のように複数の異なる方法で標準データベース・ユーザーの妥当性を検査します。

- 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
- ユーザーのためにトランザクションを実行する中間層アプリケーションの Oracle データベースによる認証

通常は、検査を簡素化するため、前述のどれか 1 つの方法を使用してデータベースのすべてのユーザーを認証します。ただし、Oracle では、同じデータベース・インスタンス内で前述の方法をすべて使用できます。

さらに、Oracle では、ネットワークのセキュリティを確実にするため、送信時にパスワードが暗号化されます。

データベース管理者は特別なデータベース操作を実行するため、データベース管理者に対しては特別な認証手順が必要になります。

オペレーティング・システムによる認証

一部のオペレーティング・システムでは、オペレーティング・システムの維持している情報を、Oracle がユーザーの認証のために使用できます。オペレーティング・システムによる認証の利点は、次のとおりです。

- ユーザーがより簡単に Oracle に接続できる（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する必要がなくなります。ただし、ユーザー名は Oracle データベース内に保持されます。
- データベースのユーザー名エントリと、オペレーティング・システムの監査証跡が対応する。

データベース・ユーザーの認証にオペレーティング・システムを使用する場合は、分散データベース環境とデータベース・リンクに関していくつかの特別な考慮事項があります。詳細は、第 33 章「分散データベース」を参照してください。

> **追加情報：** オペレーティング・システムによる認証の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

ネットワークによる認証

Oracle は、後述のようにネットワークによる複数の認証方法をサポートしています。

サードパーティベースの認証テクノロジ

ネットワーク認証サービス（DCE、Kerberos または SESAME など）を使用できる場合、Oracle はこれらのネットワーク・サービスによる認証を受け入れることができます。Oracle でネットワーク認証サービスを使用するには、Oracle8*i* Enterprise Edition と Oracle Advanced Security が必要です。

> **追加情報：** ネットワーク認証サービスを使用する場合は、ネットワーク・ロールとデータベース・リンクについて特別な考慮事項があります。ネットワーク認証の詳細は、『Oracle8*i* 分散システム』を参照してください。また、Oracle Advanced Security の詳細は、『Oracle8*i* Advanced Security 管理者ガイド』を参照してください。

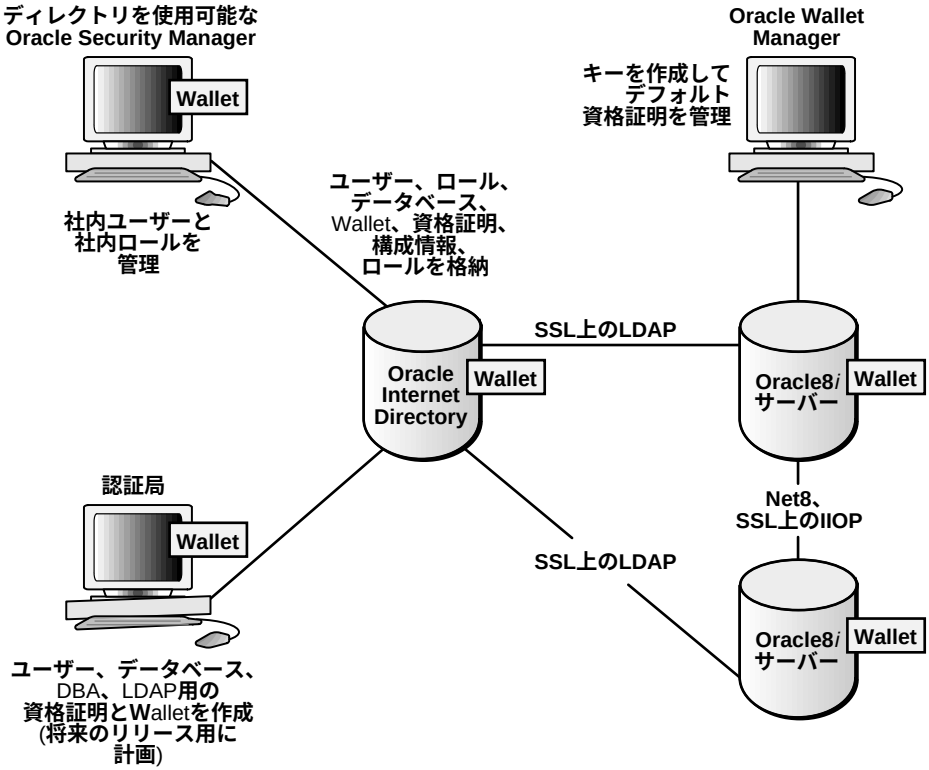
公開鍵インフラストラクチャベースの認証

公開鍵暗号化システムに基づく認証システムは、ユーザー・クライアントにデジタル証明書を発行します。ユーザー・クライアントはそれを使用し、認証サーバーを直接関与させないで、社内のサーバーを直接認証します。Oracle は、公開鍵と証明書を使用するための公開鍵インフラストラクチャ（PKI）を提供します。PKI のコンポーネントは、次のとおりです。

- Secure Sockets Layer（SSL）による認証および保護セッション・キー管理。
- Oracle Call Interface（OCI）および PL/SQL 関数。ユーザー指定のデータに秘密鍵と証明書を使用してシグネチャを付け、証明書とトラスト・ポイントを使用してデータのシグネチャを検査します。
- 「Oracle Wallet」。これは、ユーザーの秘密鍵、ユーザー証明書および一連のトラスト・ポイント（ユーザーが信頼するルート証明書のリスト）を含むデータ構造です。
- 「Oracle Wallet Manager」。Oracle のクライアントとサーバー上でユーザー・キーを保護し、X.509v3 証明書を管理します。
- X.509v3 証明書。認証局（Oracle 外部）から取得します。証明書が Oracle Wallet にロードされ、認証が使用可能になります。
- 「Directory-enabled Oracle Security Manager」。管理を容易にし、セキュリティ・レベルを高めるために、集中的な権限管理を提供します。Directory-enabled Oracle Security Manager により、Oracle Internet Directory または Lightweight Directory Access Protocol（LDAP）準拠の任意のディレクトリにロールを格納し、取り出すことができます。
- 「Oracle Internet Directory」。Oracle8*i* データベース上に構築された LDAP v3 準拠のディレクトリです。X.509 証明書を使用して認証されたユーザーについて、セキュリティ属性や権限など、ユーザーおよびシステム構成環境を管理できます。Oracle Internet Directory は認証レベルのアクセス制御を施行します。これにより、特定のユーザー（社内のセキュリティ管理者など）のみに、ディレクトリの特定の属性の読込み、書込みまたは更新権限を限定できます。また、SSL 暗号化を介したディレクトリの問合せと応答の保護と認証もサポートしています。

図 29-1 に、Oracle の公開鍵インフラストラクチャを示します。

図 29-1 Oracle の公開鍵インフラストラクチャ



注意： Oracle で公開鍵インフラストラクチャベースの認証を使用するには、Oracle8*i* Enterprise Edition と Oracle Advanced Security が必要です。

リモート認証

Oracle は、Remote Dial-In User Service（RADIUS）を介したユーザーのリモート認証をサポートしています。RADIUS は、ユーザー認証、許可およびアカウント処理に使用される標準軽量プロトコルです。Oracle で RADIUS を介したユーザーのリモート認証を使用するには、Oracle8*i* Enterprise Edition と Oracle Advanced Security が必要です。

追加情報： Oracle Advanced Security の詳細は、『Oracle8*i* Advanced Security 管理者ガイド』を参照してください。

Oracle データベースによる認証

Oracle では、データベースに格納されている情報を使用して、データベースに接続しようとするユーザーを認証できます。

Oracle でデータベースによる認証を使用する場合は、対応するパスワードを指定してそれぞれのユーザーを作成します。ユーザーは、接続の確立時に正しいパスワードを入力します。この方法により、データベースの無許可使用が防止されます。ユーザーのパスワードは、暗号化された形式でデータ・ディクショナリに格納されます。ユーザーは、いつでも自分のパスワードを変更できます。

接続時のパスワード暗号化

パスワードの機密性を保護するため、Oracle ではネットワーク（クライアント / サーバーおよびサーバー / サーバー）接続時にパスワードを暗号化できます。クライアントおよびサーバーのマシンでこの機能を使用可能にすると、パスワードは修正 DES（データ暗号化規格）アルゴリズムを使用して暗号化され、その後ネットワーク経由で送信されます。パスワードをネットワークへの侵入者から保護するために、接続のパスワード暗号化を使用可能にすることを強くお薦めします。

> **追加情報：** ネットワーク・システムにおけるパスワード暗号化の詳細は、『Oracle8*i* 分散システム』を参照してください。

アカウントのロック

指定された試行回数の範囲内でユーザーがシステムにログインできない場合、ユーザーのアカウントがロックされることがあります。アカウントの構成方法によって、一定の時間の後に自動的にロックが解除されるか、またはデータベース管理者がロックを解除する必要があります。

CREATE PROFILE 文は、ユーザーが試行できるログインの失敗回数、および自動ロック解除前にアカウントがロックされたままになっている時間を構成するために使用します。プロファイルの詳細は、29-18 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はロック解除されないので、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時間切れ

パスワードの存続期間と時間切れのオプションを使用すると、データベース管理者はパスワードの存続期間を指定できます。その期間を過ぎると、パスワードは期限切れになり、そのアカウントにログインする前にパスワードを変更する必要があります。パスワードの期限が切れた後にデータベース・アカウントへの最初のログイン試行で、ユーザーのアカウントは猶予期間に入り、その猶予期間が終わるまで、ユーザーがログインするたびに警告メッセージが発行されます。

ユーザーは、猶予期間内にパスワードを変更するよう求められます。パスワードが猶予期間内に変更されなければ、そのアカウントはロックされ、それ以降、そのアカウントにはデータベース管理者の介入がなければログインできなくなります。

また、データベース管理者がパスワードの状態を時間切れに設定することもできます。この場合は、ユーザーのアカウントの状態が時間切れに変更され、そのユーザーがデータベースにログインするには、自分で、またはデータベース管理者に依頼してパスワードを変更する必要があります。

パスワード履歴

パスワード履歴オプションは、新しく指定される各パスワードを調べて、指定された期間内に、または指定されたパスワード変更回数の間に、同じパスワードが再使用されないようにするためのものです。データベース管理者は、CREATE PROFILE 文を使用してパスワードの再使用のルールを構成できます。

パスワードの複雑度の検証

複雑度の検証は、パスワードを推定してシステムに入ろうとする侵入者に対して、各パスワードが十分保護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、デフォルトのパスワード複雑度検証ルーチンでは、各パスワードに次の条件が求められます。

- 4 文字以上の長さ。
- ユーザー ID と同じでない。
- 少なくとも 1 文字のアルファベット、1 文字の数字および 1 文字の句読点が含まれている。
- 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。

複数層の認証と許可

複数層環境では、Oracle は権限を制限し、すべての層を通じてクライアント認証を保持し、クライアントのために実行されるアクションを監査することによって、中間層アプリケーションのセキュリティを制御します。

クライアント、アプリケーション・サーバーおよびデータベース・サーバー

複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントと 1 つ以上のデータベース・サーバーとの間のインタフェースとして機能します。

このアーキテクチャでは、アプリケーション・サーバーを使用して Web ブラウザなどのクライアントの資格証明を検査できます。また、データベース・サーバーは、アプリケーション・サーバーが自身のために実行する操作と、クライアントのためにアプリケーション・サーバーが実行する操作を監査できます。たとえば、前者の操作はデータベース・サーバーへの接続要求などで、後者の操作はクライアント上で表示する情報の要求などです。

複数層環境における認証は、次のようなトラスト・リージョンに基づいています。

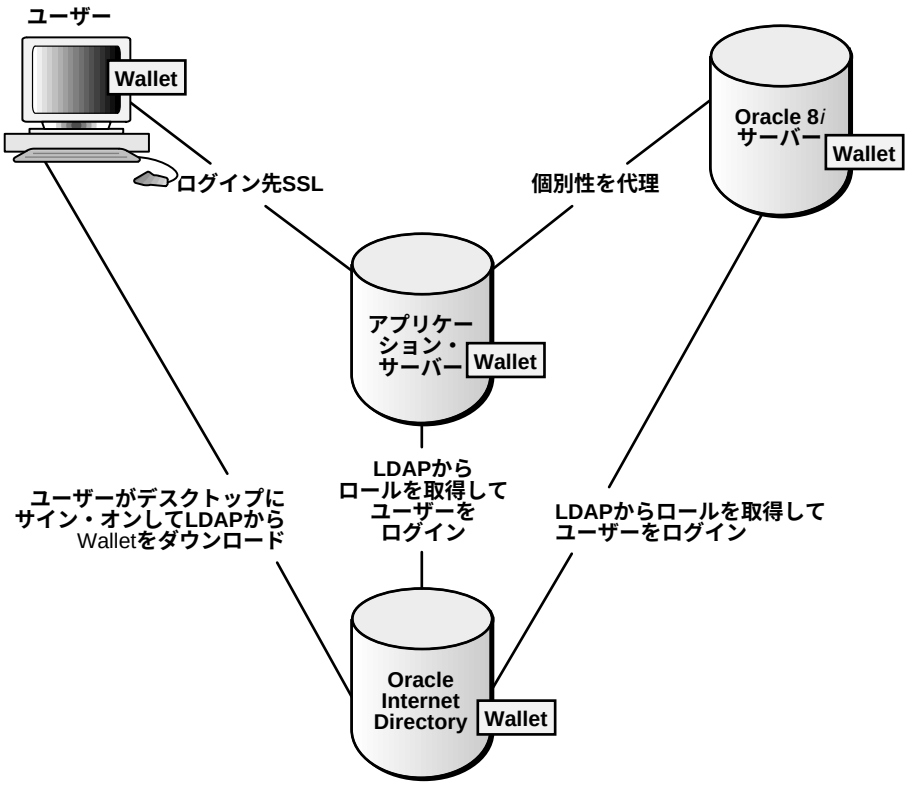
- クライアントは、通常はパスワードまたは X.509 証明書を使用して、アプリケーション・サーバーに認証の証明を提供する。
- アプリケーション・サーバーは、クライアント認証を検査し、自身をデータベース・サーバーに対して認証する。
- データベース・サーバーは、アプリケーション・サーバー認証をチェックし、クライアントが存在するかどうかを検証し、アプリケーション・サーバーがこのクライアントへの接続権限を持っているかどうかを検証する。

アプリケーション・サーバーは、接続中のクライアントのロールを使用可能にすることもできます。また、これらのロールは、認証リポジトリとして機能するディレクトリから取得できます。アプリケーション・サーバーが要求できるのは、これらのロールを使用可能にすることだけです。データベースは、次のことを検証します。

- クライアントがこれらのロールを付与されているかどうか（内部のロール・リポジトリをチェックするか、ディレクトリ内でロール付与を検証する）。
- アプリケーション・サーバーは、ユーザーのロールを使用して、そのユーザーのために接続する権限を付与されているかどうか。

図 29-2 に、複数層認証の例を示します。

図 29-2 複数層の認証



中間層アプリケーションのセキュリティの問題

中間層アプリケーションには、次のようにセキュリティ上の問題が多数あります。

責任

クライアントは、実行するトランザクションの責任を負う必要があります。たとえば、トランザクションに法的義務が伴っている場合（クライアントが Web ブラウザを使用して電子バンキング・アプリケーションにアクセスする場合など）は、アプリケーション・サーバーやデータベース・サーバーではなく、クライアントがトランザクションに責任を負います。

区別　データベース・サーバーは、Web サーバー・トランザクション、ブラウザ・クライアントのための Web サーバー・トランザクションおよびデータベースに直接アクセスするクライアントを、区別できる必要があります。

最小限度の権限　中間層アプリケーションは、すべてのクライアントのためにすべてのトランザクションを実行するための全権限をアプリケーション・サーバーに与えるのではなく、アプリケーション・サーバーの権限を、それがかわってトランザクションを実行するクライアントにバインドできる必要があります。

複数層環境における認証の問題

複数層アプリケーションは、すべての接続層を通じてクライアントの認証をメンテナンスします。この操作を必要とするのは、要求元クライアントの認証が失われると、有効な監査レコードをメンテナンスできなくなるためです。また、アプリケーション・サーバーがクライアントのために実行する操作と、アプリケーション自身のための操作を区別できなくなります。

複数層環境における制限付きの権限

複数層環境での権限は、要求された操作の実行に必要な権限に限定されます。

クライアントの権限　クライアントの権限は、複数層環境では最小限度になります。操作は、クライアントにかわってアプリケーション・サーバーが実行します。

アプリケーション・サーバーの権限　複数層環境におけるアプリケーション・サーバーの権限は、クライアント操作の実行中に望ましくない操作や不要な操作を実行できないように制限されます。

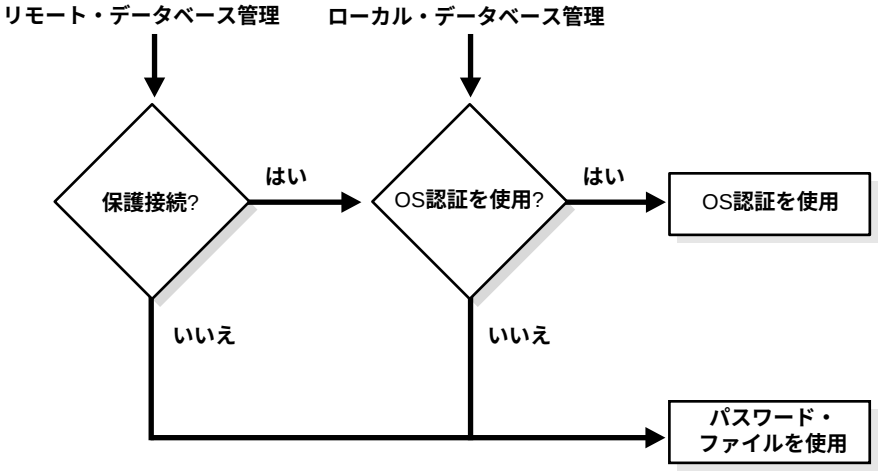
データベース管理者の認証

データベース管理者は、通常のデータベース・ユーザーが実行できない特別な操作（データベースの停止や起動など）を実行します。Oracle では、データベース管理者のユーザー名に対して、さらに安全性の高い認証方式を使用します。

データベース管理者の認証方式として、オペレーティング・システムによる認証とパスワード・ファイルによる認証のどちらを使用するかを選択できます。

図 29-3 に、データベースをローカルに（データベースが存在する同じマシン上で）管理するか、1 つのリモート・クライアントから多数の異なるデータベース・マシンを管理するかに応じた、データベース管理者の認証方式の選択肢を示します。

図 29-3 データベース管理者の認証方法



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・システム固有の Oracle 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・ファイルを使用することによって、SYSDBA または SYSOPER 権限を付与されたデータベース・ユーザー名を追跡します。データベース管理者にこれらの権限があると、次のアクションを実行できます。

SYSOPER	STARTUP、SHUTDOWN、ALTER DATABASE OPEN / MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG および RECOVER を実行できます。また、RESTRICTED SESSION 権限もこの権限に含まれます。
SYSDBA	ADMIN OPTION オプションが指定されているすべてのシステム権限、および SYSOPER システム権限が含まれます。CREATE DATABASE と時間ベースの回復が許可されます。

追加情報：『Oracle8*i* 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・オブジェクトを作成すると、そのオブジェクトはそのユーザーのデフォルト表領域に格納されます。ユーザーのデフォルト表領域は、ユーザー作成時に設定し、ユーザー作成後に変更できます。

一時表領域

一時セグメントの作成を必要とする SQL 文をユーザーが実行すると、ユーザーの一時表領域にそのセグメントが割り当てられます。

表領域のアクセスと割当て制限

データベースのどの表領域についても、ユーザーごとに「表領域の割当て制限」を設定できます。これにより、次の 2 つのことが可能になります。

- ユーザーが適切な権限を付与されている場合は、そのユーザーが指定された表領域を使用してスキーマ・オブジェクトを作成することを許可できる。
- 指定された表領域にあるユーザーのスキーマ・オブジェクトの記憶域に割り当てられたスペースの量を制限できる。

デフォルトでは、各ユーザーはデータベースの表領域での割当て制限を課されていません。このため、ユーザーになんらかのタイプのスキーマ・オブジェクトを作成する権限がある場合、そのユーザーには、オブジェクトを作成するときの表領域割当て制限が設定されているか、十分な表領域割当て制限を設定されている別のユーザーのスキーマにそのオブジェクトを作成する権限も必要になります。

ユーザーには、2 種類の表領域割当て制限を設定できます。1 つは表領域内のディスク領域を特定の量に制限する割当て制限（バイト、KB または MB 単位）、もう 1 つは表領域内のディスク領域を無制限とする割当て制限です。ユーザーのオブジェクトが表領域内の領域を使用しすぎないように、特定量の割当て制限を設定する必要があります。

表領域の割当て制限および一時セグメントは、相互に影響し合いません。

- 一時セグメントが作成されても、それはユーザーに課されている割当て制限の領域には加算されない。Oracle によって一時セグメントに自動的に作成されるスキーマ・オブジェクトは、SYS の所有となるため、割当て制限は適用されません。
- 一時セグメントは、ユーザーが割当て制限を課されていない表領域内に作成できる。

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用できます。

PUBLIC には、任意のシステム権限、オブジェクト権限またはロールを付与（または取消し）できます。権限とロールの詳細は、第 30 章「権限、ロールおよびセキュリティ・ポリシー」を参照してください。ただし、アクセス権に対する厳密なセキュリティを維持するため、PUBLIC に付与する権限とロールは、**すべて**のユーザーに関係のあるもののみにしてください。

PUBLIC に対してなんらかのシステム権限やオブジェクト権限を付与するか取り消すと、データベース中のすべてのビュー、プロシージャ、ファンクション、パッケージおよびトリガーが再コンパイルされます。

PUBLIC には次の制限があります。

- PUBLIC に対して表領域割当て制限は設定できないが、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/SYNONYM を使用して）PUBLIC として作成できるが、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・セグメントを作成することはできますが、そのセグメントは PUBLIC ユーザー・グループが所有することにはなりません。すべてのロールバック・セグメントは SYS が所有します。ロールバック・セグメントの詳細は、第 4 章「データ・ブロック、エクステントおよびセグメント」を参照してください。

ユーザー・リソースの制限とプロファイル

ユーザーのセキュリティ・ドメインの一部として、各ユーザーが使用できる各種のシステム・リソースの容量に制限を設定できます。これにより、CPU 時間などの貴重なシステム・リソースが無制限に浪費されるのを防止できます。

システム・リソースに多額の費用がかかる大規模なマルチ・ユーザー・システムにおいて、このリソース制限機能はたいへん有効です。1 人以上のユーザーが過度にリソースを消費すると、データベースの他のユーザーに有害な影響を及ぼす可能性があります。シングル・ユーザー・データベースまたは小規模のマルチ・ユーザー・データベースの場合、ユーザーがシステム・リソースを消費してもそれほど有害な影響はないため、システム・リソース機能はそれほど重要ではありません。

ユーザーのリソース制限とパスワード管理の作業環境は、そのユーザーのプロファイルを使用して管理します。プロファイルとは、そのユーザーに割り当てることができる一連のリソース制限に名前を付けたものです。それぞれの Oracle データベースに指定できるプロファイルの数に、制限はありません。Oracle において、セキュリティ管理者は、プロファイルによるリソース制限の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。

追加情報： セキュリティ管理者の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

リソース制限を設定した場合、ユーザーがセッションを作成すると、パフォーマンスがわずかに低下します。これは、ユーザーがデータベースに接続した時点で、そのユーザーのすべてのリソース制限データがロードされるためです。

システム・リソースのタイプと制限

Oracle では、CPU 時間と論理読込みを含め、いくつかのタイプのシステム・リソースの使用を制限できます。一般に、それぞれのリソースは、セッション・レベル、コール・レベル、またはその両方で制御できます。

セッション・レベル　ユーザーがデータベースに接続するたびに、セッションが作成されます。それぞれのセッションは、Oracle を実行するコンピュータの CPU 時間とメモリーを消費します。複数のリソース制限をセッション・レベルで設定できます。

ユーザーがセッション・レベルのリソース制限を超えると、現行の文は終了（ロールバック）し、セッションの制限に達したことを示すメッセージが戻されます。この時点では、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）のみです。それ以外の操作を実行すると、エラーが発生します。トランザクションがコミットまたはロールバックされた後も、現行のセッションではユーザーはどんな作業も完了できません。

コール・レベル

SQL 文が実行されるたびに、いくつかのステップが実行されて文が処理されます。この処理では、データベースに対して複数のコールが異なる実行フェーズの一部として発行されます。1 回のコールで過度にシステムが使用されないように、複数のリソース制限をコール・レベルで設定できます。

ユーザーがコール・レベルのリソース制限を超えると、Oracle は文の処理を停止し、その文をロールバックし、エラーを戻します。ただし、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 時間ですみます。ただし、大量のデータや冗長な問合せを伴う SQL 文は CPU 時間を大量に消費することがあるため、他の処理に使用できる CPU 時間が少なくなります。

CPU 時間が無制限に消費されないようにするため、1 回のコール当りの CPU 時間と、1 つのセッション中に Oracle コールに使用される CPU 時間の合計を制限できます。これらの制限は、コールやセッションに使用される 1/100 秒（0.01 秒）単位の CPU 時間で設定し、測定されます。

論理読込み

入出力 (I/O) は、データベース・システムで最もリソースの使用量が多い操作の 1 つです。I/O を集中的に実行する SQL 文は、メモリーとディスクの使用を独占することがあるため、他のデータベース操作がこれらのリソースをめぐって競合する原因になる可能性があります。

単一の原因による過度の I/O が発生しないようにするため、Oracle では、1 コール当り、および 1 セッション当りの論理データ・ブロック読込み数を制限できます。論理データ・ブロック読込みには、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込みの数として設定し、測定されます。

その他のリソース

Oracle では、さらに、その他のいくつかのセッション・レベルのリソース制限が提供されています。

- ユーザー当りの「同時実行セッション」数の制限。各ユーザーは、事前に定義された数まで同時実行セッションを作成できます。
- セッションの「アイドル時間」の制限。1 つのセッションでの Oracle コールと Oracle コールの間隔がアイドル時間制限に達すると、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、ユーザーがインスタンスから切断されたことを示すエラーを受け取ります。この制限は、分単位の経過時間として設定します。

  注意： セッションがアイドル時間の制限を超えたために異常終了すると、その少し後に、異常終了したセッションの後処理としてプロセス・モニター（PMON）バックグラウンド・プロセスが クリーン・アップを実行します。PMON がこのプロセスを完了するまでは、異常終了したセッションも、セッションまたはユーザー・レベルのリソース制限に加算されます。

- セッション当りの経過接続時間の制限。セッションの持続時間が経過時間制限を超えると、現行のトランザクションがロールバックされ、セッションが削除され、そのセッションのリソースがシステムに戻されます。この制限は、分単位の経過時間として設定します。

  注意： Oracle は、経過アイドル時間や経過接続時間を絶えず監視しているわけではありません。絶えず監視するとすれば、システム・パフォーマンスが低下します。そのかわり数分ごとにチェックします。このため、Oracle がこの制限を施行してからセッションを異常終了させるまでの間に、セッションはこの制限をわずかに（5 分など）超える可能性があります。

- セッションのプライベート SGA 領域（プライベート SQL 領域に使用される）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用するシステムの場合のみです。それ以外のシステムの場合、プライベート SQL 領域は PGA 内にあります。この制限は、インスタンスの SGA に使用するメモリーのバイト数として設定します。KB または MB で指定するには、"K" または "M" の文字を使用します。

  追加情報： リソース制限を使用可能および使用禁止にする手順は、『Oracle8*i* 管理者ガイド』を参照してください。

プロファイル

プロファイルとは、Oracle データベースの有効なユーザー名に対して割り当てることができるリソース制限に名前を付けた集合です。プロファイルを使用すると、リソース制限を簡単に管理できます。

プロファイルを使用する場合

ユーザー・プロファイルを作成して管理する必要があるのは、リソース制限がデータベース・セキュリティ方針の要件になっている場合だけです。プロファイルを使用するには、まずデータベース内の関連のあるユーザーを、いくつかのタイプに分類します。関連するユーザーの権限を管理するためにロールを使用するのと同様に、関連するユーザーのリソース制限を管理するためにプロファイルを使用します。データベースのすべてのタイプのユーザーを網羅するのに必要なプロファイルの数を決定し、それぞれのプロファイルに適切なリソース制限を決定します。

プロファイルのリソース制限の値の決定

プロファイルを作成し、そのプロファイルに含めるリソース制限を決定する前に、各リソース制限について適切な値を決定する必要があります。これらの値は、典型的なユーザーが実行する操作のタイプを基準として決定できます。たとえば、あるクラスのユーザーが通常は大量の論理データ・ブロック読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 オプションを使用することによって、CONNECT_TIME、LOGICAL_READS_PER_SESSION および LOGICAL_READS_PER_CALL の制限値についての情報を収集できます。詳細は、第 31 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 SQL*Plus）のモニター機能、特に統計モニターを使用して収集できます。

ライセンス

通常、Oracle のライセンスには、名前付きユーザーの最大数、または同時接続ユーザーの最大数が決められています。データベース管理者（DBA）は、そのサイトでライセンス契約を守ることに責任があります。Oracle のライセンス機能は、あるインスタンスに同時に接続されているセッションの数を追跡して制限したり、データベース内に作成されるユーザーの数を制限することによって、DBA モニター・システムの使用を支援するものです。

ライセンスされたセッション数より多くのセッションを接続したり、ライセンスされたユーザーより多くのユーザーを作成することが DBA にとって必要になった場合は、Oracle ライセンスをアップグレードして、制限を適切に変更できます。（Oracle ライセンスをアップグレードする場合は、必ずソフトウェアの販売元にご確認ください。）

注意： Oracle が Oracle アプリケーション（Oracle Office など）に組み込まれている場合、一部の古いオペレーティング・システムで実行されている場合または一部の国で使用するために購入した場合には、セッションの集合数にもユーザーの集合グループにもライセンスはありません。そのような場合に限り、Oracle ライセンス・メカニズムは適用されないので、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*i* 管理者ガイド』を参照してください。

同時使用ライセンス

「同時使用ライセンス」では、「同時実行ユーザー」の数が指定されます。同時実行ユーザーとは、ある時点で、指定されたコンピュータのデータベースに同時に接続できるセッションのことです。この数には、すべてのバッチ・プロセスとオンライン・ユーザーが含まれます。単一のユーザーが複数の同時実行セッションに関係している場合、その各セッションはセッションの合計数に別々に加算されます。データベースに直接接続されるセッションの数を少なくするために多重化ソフトウェア（TP モニターなど）を使用している場合、同時実行ユーザーの数は多重化フロントエンドへの個々の入力の数になります。

同時使用ライセンス・メカニズムによって、DBA は次のことを実行できます。

- LICENSE_MAX_SESSIONS パラメータを設定することにより、あるインスタンスに接続できる同時実行セッションの数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 コマンドを使用して）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降の同時実行セッションの最大数を調べることができます。V$LICENSE ビューには、現行のライセンス制限の設定値、現行のセッション数およびインスタンスが起動して以降の同時実行セッションの最大数（セッションの「高水位標」）が表示されます。DBA は、この情報を使用してシステムのライセンス・ニーズを評価し、システムのアップグレードを計画できます。

Oracle Parallel Server で実行しているインスタンスの場合は、それぞれのインスタンスに独自の同時使用制限と警告制限を設定できます。各インスタンスの制限の合計が、サイトの同時使用ライセンスを超えないようにしてください。

着信データベース・リンク用に作成されるセッションも含め、同時使用制限はすべてのユーザー・セッションに適用されます。Oracle によって作成されたセッションや再帰的なセッションには適用されません。外部の多重化ソフトウェアを介して接続されるセッションは、Oracle ライセンス・メカニズムによって別個にカウントされることはありませんが、それぞれのセッションは Oracle ライセンスの合計に個別に加算されます。これらのセッションを考慮する責任は、DBA にあります。

名前付きユーザー・ライセンス

「名前付きユーザー・ライセンス」では、名前付きユーザーの数がライセンスで指定されます。「名前付きユーザー」とは、指定されたコンピュータで Oracle の使用を認められている個人のことです。それぞれのユーザーが同時に実行できるセッションの数や、データベースの同時実行セッションの数に、制限は設定されません。

名前付きユーザー・ライセンスによって DBA は、データベース・リンクを介して接続するユーザーを含め、データベースの中で定義されるユーザーの数に制限を設定できます。この制限に達すると、新しいユーザーは作成できなくなります。このメカニズムでは、データベースにアクセスする個々の人のユーザー名はデータベース内で一意であり、1 つのユーザー名が複数の人の間で共有されることはないことが前提となっています。

DBA は、これらの制限をデータベースのパラメータ・ファイルに設定することによ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 コマンドを使用して）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*i* Parallel Server 概要および管理』を参照してください。

30

権限、ロールおよびセキュリティ・ポリシー

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、ユーザーが実行できるシステム操作とスキーマ・オブジェクトに対して実行できるアクセスを、権限、ロールおよびセキュリティ・ポリシー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
  - 表のセキュリティに関するトピック
  - ビューのセキュリティに関するトピック
  - プロシージャのセキュリティに関するトピック
  - 型のセキュリティに関するトピック
- ロール
- ファイン・グレイン・アクセス・コントロール
- アプリケーション・コンテキスト

権限

「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（セッションを作成する）
- 表を作成する
- 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で必要な作業を実行できるようになります。なお、権限は、必要な作業を実行する上で本当にその権限を必要としているユーザーにのみ付与します。必要でない権限まで付与すると、セキュリティを維持できなくなる可能性があります。ユーザーは次の 2 つの方法で権限を受け取ることができます。

- 権限を明示的にユーザーに付与する。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、更新および削除する権限を、CLERK という名前のロールに付与し、このロール CLERK をユーザー SCOTT や BRIAN に付与できます。

ロールを使用することによって権限の管理が容易になり、改善されるため、通常、権限は個々のユーザーではなくロールに付与します。

権限は次の 2 つに分類できます。

- システム権限
- スキーマ・オブジェクト権限

> **追加情報：** すべてのシステム権限およびスキーマ・オブジェクト権限の詳細リストと、権限管理の手順は、『Oracle8*i* 管理者ガイド』を参照してください。

システム権限

システム権限とは、特定のアクションを実行する権限、または特定のタイプのスキーマ・オブジェクトに対してアクションを実行する権限のことです。たとえば、表領域を作成する権限や、データベース内の任意の表から行を削除する権限などがシステム権限です。システム権限には 60 以上の種類があります。

システム権限の付与と取消し

システム権限は、ユーザーやロールに対して付与したり取り消すことができます。システム権限をロールに付与すると、そのロールを使用してシステム権限を管理できます（たとえば、ロールを使用すると権限を選択的に使用できるようになります）。

注意： 通常、エンド・ユーザーはシステム権限に関連する機能を必要としないので、システム権限は、管理者やアプリケーション開発者にのみ付与するようにしてください。

次のどちらかを使用して、ユーザーやロールにシステム権限を付与したり、その権限を取り消します。

- Oracle Enterprise Manager の「Grant System Privileges/Roles」ダイアログ・ボックスおよび「Revoke System Privileges/Roles」ダイアログ・ボックス
- SQL コマンド GRANT および REVOKE

システム権限を付与または取消しできるユーザー

他のユーザーにシステム権限を付与したり、他のユーザーのシステム権限を取り消したりできるのは、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）のみです。

スキーマ・オブジェクト権限

スキーマ・オブジェクト権限（「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、ファンクションまたは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。各タイプのスキーマ・オブジェクトごとに、異なるオブジェクト権限があります。たとえば、DEPT 表から行を削除する権限は、1 つのオブジェクト権限です。

一部のスキーマ・オブジェクト（クラスタ、索引、トリガーおよびデータベース・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 システム権限が必要です。

スキーマ・オブジェクトとそのシノニムは、権限に関しては等価です。つまり、表、ビュー、順序、プロシージャ、ファンクションまたはパッケージについて付与されるオブジェクト権限は、その基礎となるオブジェクトを名前で参照するかシノニムで参照するかにかかわらず、適用されます。

たとえば、JWARD.EMP という表があり、それに JWARD.EMPLOYEE というシノニムがある場合に、ユーザー JWARD が次の文を発行するとします。

```
GRANT SELECT ON emp TO swilliams;
```

ユーザー SWILLIAMS は、名前またはシノニム JWARD.EMPLOYEE を使用して表を参照することによって、JWARD.EMP を問い合せることができます。

```
SELECT * FROM jward.emp;
SELECT * FROM jward.employee;
```

表、ビュー、プロシージャ、ファンクションまたはパッケージに対するオブジェクト権限を、そのオブジェクトの「シノニム」に付与した場合、結果はシノニムを使用しない場合と同じです。たとえば、JWARD が EMP 表に対する SELECT 権限を SWILLIAMS に付与する場合、JWARD は次のどちらの文でも使用できます。

```
GRANT SELECT ON emp TO swilliams;
GRANT SELECT ON employee TO swilliams;
```

シノニムが削除された場合、そのシノニムを指定して権限が付与されていた場合でも、基礎になるスキーマ・オブジェクトに対して付与された権限はすべて有効なままです。

スキーマ・オブジェクト権限の付与と取消し

スキーマ・オブジェクト権限は、ユーザーやロールに対して付与したり取り消すことができます。オブジェクト権限をロールに付与する場合、その権限を選択的に使用可能にできます。ユーザーやロールのオブジェクト権限を付与したり取り消すには、それぞれ SQL コマンドの GRANT と REVOKE を使用するか、Oracle Enterprise Manger の「Add Privilege to Role / User」ダイアログ・ボックスと「Revoke Privilege from Role / User」ダイアログ・ボックスを使用します。

スキーマ・オブジェクト権限を付与できるユーザー

ユーザーは、自分のスキーマに含まれているスキーマ・オブジェクトに関しては、自動的にすべてのオブジェクト権限が付与されています。ユーザーは、自分が所有するスキーマ・オブジェクトに対するオブジェクト権限を、他のユーザーまたはロールに付与できます。GRANT 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。このオプションを指定しなければ、権限受領者はその権限を使用できますが、別のユーザーには権限を付与できません。

表のセキュリティに関するトピック

表に対するスキーマ・オブジェクト権限は、DML および DDL 操作のレベルで表のセキュリティ機能を実現します。

データ操作言語（DML）操作

DELETE、INSERT、SELECT および UPDATE の各権限は、それぞれ表またはビューに対する DELETE、INSERT、SELECT および 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せたり変更する必要があるユーザーとロールにのみ付与してください。

追加情報： これらの DML 操作の詳細は、『Oracle8*i* 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 権限を選択的に使用します。

たとえば、データ入力者が EMP 表の SAL 列を変更しないようにするには、SAL 列を含めずに、選択的な INSERT 権限と UPDATE 権限を付与します。（また、SAL 列を含まないようにしたビューを使用すると、同じことをさらに高いセキュリティ・レベルで実現できます。）

データ定義言語（DDL）操作

ALTER、INDEX および REFERENCES の各権限は、表に対する DDL 操作の実行を許可します。これらの権限によって、他のユーザーは表の依存性を変更または作成できるようになるため、この権限は慎重に付与する必要があります。表に対して DDL 操作を実行するユーザーには、さらにその他のシステム権限やオブジェクト権限が必要です（たとえば、表にトリガーを作成するには、その表に対する ALTER TABLE オブジェクト権限と CREATE TRIGGER システム権限の両方が必要です）。

INSERT 権限および UPDATE 権限と同じように、表の特定の列に REFERENCES 権限を付与できます。REFERENCES 権限を付与されたユーザーは、付与の対象となった表を、自分の表の中に作成する外部キーの親キーとして使用できます。外部キーが存在すると、親キーに対して実行される可能性のあるデータ操作や表の変更が制限されるため、この動作は特殊な権限によって制御されます。列固有の REFERENCES 権限によって、権限受領者が使用できるのは、指定された列（当然、親表の主キーまたは一意キーを最低 1 つ含む）に制限されます。主キー、一意キーおよび整合性制約の詳細は、第 28 章「データの整合性」を参照してください。

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するさまざまな 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。

ビューの作成に必要な権限

ビューを作成するには、次の要件を満たしている必要があります。

- CREATE VIEW システム権限（自分のスキーマ内にビューを作成する）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成する）が、明示的にまたはロールによって付与されている必要がある。

- 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 または DELETE の各オブジェクト権限、または SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE または DELETE ANY TABLE の各システム権限が明示的に付与されている必要がある。これらの権限は、ロールを通じ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る必要がある。これらの権限がなければ、権限受領者はビューにアクセスできません。

ビューによる表セキュリティの強化

ビューを使用するのに必要な権限は、そのビュー自体に対する適切な権限のみです。ビューの基礎となるベース・オブジェクトに対する権限は必要ありません。

ビューを使用することにより、表に対してさらに 2 つのセキュリティ・レベル（列レベルのセキュリティと値ベースのセキュリティ）が追加されます。

- ビューは、実表の中から選択した列のアクセスを提供できる。たとえば、EMP 表の中から EMPNO、ENAME および MGR の 3 列のみを表示するようにビューを定義できます。

  ```
  CREATE VIEW emp_mgr AS
      SELECT ename, empno, mgr FROM emp;
  ```

- ビューにより、表の情報に対して、値ベースのセキュリティを実現できる。ビュー定義の中に WHERE 句を使用すると、実表の中から選択した行のみが表示されます。次の 2 つの例を考えてみます。

  ```
  CREATE VIEW lowsal AS
      SELECT * FROM emp
      WHERE sal < 10000;
  ```

  LOWSAL ビューでは、EMP 表のうち給与値が 10000 未満のすべての行にアクセスできます。LOWSAL ビューでは、EMP 表のすべての列にアクセスできることに注目してください。

  ```
  CREATE VIEW own_salary AS
      SELECT ename, sal
      FROM emp
      WHERE ename = USER;
  ```

  OWN_SALARY ビューでは、ENAME がビューの現ユーザーに一致している行にのみアクセスできます。OWN_SALARY ビューは USER 関数を使用します。この関数の値は、常に現ユーザーを参照します。なお、このビューでは、列レベルのセキュリティと値ベースのセキュリティを併用しています。

プロシージャのセキュリティに関するトピック

プロシージャ（スタンドアロン・プロシージャ、ファクションおよびパッケージを含む）に対する「スキーマ・オブジェクト権限」は、EXECUTE のみです。この権限は、プロシージャの実行、またはそのプロシージャをコールする他のプロシージャのコンパイルを必要とするユーザーにのみ付与してください。

プロシージャの実行とセキュリティ・ドメイン

特定のプロシージャに対する EXECUTE オブジェクト権限を持っているユーザーは、そのプロシージャを実行したり、それを参照するプログラム・ユニットをコンパイルできます。このプロシージャがコールされるときには、実行時権限チェックは実行されません。EXECUTE ANY PROCEDURE システム権限を付与されているユーザーは、データベース内の任意のプロシージャを実行できます。

プロシージャの実行権限は、ロールを通してユーザーに付与できます。ロールの詳細は、30-18 ページの「PL/SQL ブロックとロール」を参照してください。

起動者権限プロシージャの場合は、参照オブジェクトに対する追加の権限が必要ですが、定義者権限プロシージャの場合は不要です（18-9 ページの「定義者権限と起動者権限」を参照）。

定義者権限 定義者権限プロシージャのユーザーに必要なのは、そのプロシージャを実行する権限のみで、そのプロシージャがアクセスする基礎となるオブジェクトの権限は不要です。これは、定義者権限プロシージャは、その実行者に関係なく、所有者のセキュリティ・ドメインで動作するからです。プロシージャの所有者は、参照オブジェクトに対する必要なオブジェクト権限をすべて持つ必要があります。定義者権限プロシージャのユーザーに付与する権限が少ないほど、データベース・アクセスを厳密に制御できます。

定義者権限プロシージャを使用して、データベースのセキュリティ・レベルを強化できます。定義者権限プロシージャを記述し、ユーザーには EXECUTE 権限のみを付与することにより、ユーザーがそのプロシージャを介さなければ参照オブジェクトにアクセスできないようにできます（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）。

定義者権限ストアド・プロシージャの所有者の現行の権限は、常にそのプロシージャを実行する前にチェックされます。参照オブジェクトに対して必要な権限が、定義者権限プロシージャの所有者から取り消されていると、所有者もその他のユーザーも、そのプロシージャを実行できません。

注意： トリガーの実行には、定義者権限プロシージャと同じパターンが適用されます。ユーザーは、実行する権限を付与されている SQL 文を実行できます。SQL 文の実行結果として、トリガーが起動されます。トリガー・アクション内の文は、トリガーを所有するユーザーのセキュリティ・ドメインで実行されます。

起動者権限 起動者権限プロシージャは、使用可能になっているロールも含め、すべての起動者の権限で実行されます。起動者権限プロシージャのユーザーには、そのプロシージャの名前が起動者のスキーマ内で解決されるようにアクセスする、基礎となるオブジェクトに対する権限が必要です。

- 起動者のスキーマ内で解決される外部参照（DML 文中や動的 SQL 文中の名前など）の場合、起動者は、基礎となるオブジェクトにアクセスするための権限が必要である。
- 他のすべてのオブジェクト（ファンクションやプロシージャなど）の場合、所有者の権限はコンパイル時にチェックされ、実行時チェックは行われない。

18-10 ページの「外部参照の変換」を参照してください。

DML 文または動的 SQL 文に埋め込まれているプログラム参照は、実行時に効率的に再コンパイルされるため、実行時には起動者の権限でチェックされます。

ほとんどの DBMS_* パッケージなど、Oracle が提供する多数のパッケージは、起動者権限で実行されます。つまり、所有者（SYS）ではなく現行ユーザーとして実行されます。ただし、DBMS_RLS パッケージなど、いくつか例外があります（30-21 ページの「ファイン・グレイン・アクセス・コントロール」を参照）。

> **追加情報：** Oracle が提供するパッケージの詳細は、『Oracle8*i* パッケージ・プロシージャ リファレンス』を参照してください。

複数のプログラム・ユニットからなるソフトウェア・バンドルを作成し、一部のプログラム・ユニットには定義者権限を付与し、残りのプログラム・ユニットには起動者権限を付与して、プログラムのエントリ・ポイントを限定できます（「コントロール・ステップイン」）。エントリ・ポイント・プロシージャを実行する権限を持つユーザーは、内部プログラム・ユニットを間接的に実行できますが、内部プログラムを直接コールすることはできません。

プロシージャの作成または変更に必要なシステム権限

プロシージャを作成するには、ユーザーには CREATE PROCEDURE または CREATE ANY PROCEDURE システム権限が必要です。プロシージャを変更する、つまりプロシージャを手動で再コンパイルするには、そのプロシージャを所有しているか、ALTER ANY PROCEDURE システム権限を持っていなければなりません。

プロシージャを所有するユーザーは、プロシージャ本体で参照されるスキーマ・オブジェクトに対する権限も持っている必要があります。プロシージャを作成するには、プロシージャによって参照されるすべてのオブジェクトに対して、必要な権限（システム権限やオブジェクト権限）を明示的に付与されてい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。これには、作成するプロシージャ内からコールするプロシージャに対する EXECUTE 権限も含まれます。

また、トリガーでは、参照オブジェクトに対する権限をトリガーの所有者に対して明示的に付与することが必要です。権限が明示的に、またはロールを介して付与されていても、無名 PL/SQL ブロックは任意の権限を使用できます。

パッケージとパッケージ・オブジェクト

パッケージに対する EXECUTE オブジェクト権限を付与されているユーザーは、パッケージ内の任意の（パブリック）プロシージャまたはファンクションを実行したり、任意の（パブリック）パッケージ変数の値をアクセスまたは修正できます。パッケージの構成体に対して特定の EXECUTE 権限は付与できません。したがって、データベース・アプリケーションのプロシージャ、ファンクションおよびパッケージを開発する場合は、セキュリティの確立に関して 2 つの選択肢を考慮してください。これらの選択肢について、次に示す例で具体的に説明します。

例 1 この例では、2 つのパッケージの本体の中に 4 つのプロシージャを作成します。

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO emp . . .
    END hire;
  PROCEDURE fire(...) IS
    BEGIN
      DELETE FROM emp . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE EMP SET sal = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE EMP SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

このプロシージャを実行するためのアクセス権限を付与するには、次の文を使用して、パッケージに対する EXECUTE 権限を付与します。

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

つまり、EXECUTE 権限はパッケージに対して付与されるため、これによってパッケージ・オブジェクト全体にわたる一様なアクセスが提供されます。

例 2 この例では、単一のパッケージ本体にある 4 つのプロシージャ定義を示します。さらに 2 つのスタンドアロン・プロシージャと 1 つのパッケージを作成し、メイン・パッケージ内で定義したプロシージャへのアクセスを提供します。

```
CREATE PACKAGE BODY employee_changes AS
  PROCEDURE change_salary(...) IS BEGIN ...END;
  PROCEDURE change_bonus(...) IS BEGIN ...END;
  PROCEDURE insert_employee(...) IS BEGIN ...END;
  PROCEDURE delete_employee(...) IS BEGIN ...END;
END employee_changes;

CREATE PROCEDURE hire
  BEGIN
    employee_changes.insert_employee(...)
  END hire;

CREATE PROCEDURE fire
  BEGIN
    employee_changes.delete_employee(...)
  END fire;

PACKAGE raise_bonus IS
  PROCEDURE give_raise(...) AS
    BEGIN
      employee_changes.change_salary(...)
    END give_raise;

  PROCEDURE give_bonus(...)
    BEGIN
      employee_changes.change_bonus(...)
    END give_bonus;
```

この方法を使用すると、実際に作業を実行するプロシージャ（EMPLOYEE_CHANGES パッケージ内のプロシージャ）は 1 つのパッケージ内で定義され、宣言されたグローバル変数やカーソルなどを共有できます。トップ・レベルのプロシージャの HIRE と FIRE、および追加のパッケージ RAISE_BONUS を宣言することにより、選択的な EXECUTE 権限をメイン・パッケージ内のプロシージャに対して付与できます。

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

型のセキュリティに関するトピック

この項では、型、メソッドおよびオブジェクトに対する権限について説明します。

システム権限

Oracle8*i* では、名前付きの型（オブジェクト型、VARRAY およびネストした表）について、次のシステム権限が定義されています。

権限	許可される操作
CREATE TYPE	自分のスキーマ内で名前付きの型を作成する。
CREATE ANY TYPE	任意のスキーマ内で名前付きの型を作成する。
ALTER ANY TYPE	任意のスキーマ内で名前付きの型を変更する。
DROP ANY TYPE	任意のスキーマ内で名前付きの型を削除する。
EXECUTE ANY TYPE	任意のスキーマ内で名前付きの型を使用および参照する。

CONNECT ロールと RESOURCE ロールには、CREATE TYPE システム権限が含まれています。DBA ロールには、これらの権限すべてが含まれています。

オブジェクト権限

名前付きの型に適用されるオブジェクト権限は、EXECUTE のみです。名前付きの型に対する EXECUTE 権限があれば、ユーザーはその型を使用して次の操作を実行できます。

- 表を定義する。
- リレーショナル表に列を定義する。
- 名前付きの変数またはパラメータを宣言する。

EXECUTE 権限により、ユーザーは、型コンストラクタを含め、その型のメソッドを起動できます。これは、ストアド PL/SQL プロシージャに対する EXECUTE 権限と同じです。

メソッド実行モデル

メソッドの実行は、他のストアド PL/SQL プロシージャと同じです。詳細は、30-7 ページの「プロシージャのセキュリティに関するトピック」を参照してください。

型の作成と型を使用した表の作成に必要な権限

型を作成するには、次の要件を満たしている必要があります。

- 自分のスキーマ内で型を作成するには CREATE TYPE システム権限が、他のユーザーのスキーマ内で型を作成するには CREATE ANY TYPE システム権限が必要である。この 2 つの権限は、明示的に、またはロールを介して取得できます。
- 型の所有者には、その型の定義で参照されている他のすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要がある。所有者が、ロールを介して必要な権限を取得することはできません。

- 型の所有者が他のユーザーに型へのアクセス権を付与する意図を持っている場合は、GRANT OPTION で参照された型への EXECUTE 権限を付与されるか、ADMIN OPTION で EXECUTE ANY TYPE システム権限を付与される必要がある。どちらかの権限がなければ、権限不足のため、型の所有者は型へのアクセス権を他のユーザーに付与できません。

型を使用して表を作成するには、30-4 ページの「表のセキュリティに関するトピック」で説明した表の作成要件だけでなく、次の要件を満たす必要があります。

- 表の所有者には、その表で参照されているすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要がある。所有者が、ロールを介して必要な権限を取得することはできません。
- 表の所有者が他のユーザーに表へのアクセス権を付与する意図を持っている場合は、GRANT OPTION で参照された型への EXECUTE 権限を付与されるか、ADMIN OPTION で EXECUTE ANY TYPE システム権限を付与される必要がある。どちらかの権限がなければ、権限不足のため、表の所有者は型へのアクセス権を他のユーザーに付与できません。

例

CONNECT および RESOURCE ロールを持つユーザーが存在するとします。

- USER1
- USER2
- USER3

USER1 は、自分のスキーマで次の DDL を実行します。

```
CREATE TYPE type1 AS OBJECT (
  attr1 NUMBER);

CREATE TYPE type2 AS OBJECT (
  attr2 NUMBER);

GRANT EXECUTE ON type1 TO user2 ;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION ;
```

USER2 は、自分のスキーマで次の DDL を実行します。

```
CREATE TABLE tab1 OF user1.type1 ;
CREATE TYPE type3 AS OBJECT (
  attr3 user1.type2);
CREATE TABLE tab2 (
  col1 user1.type2);
```

次の文は、正常に実行されます。USER2 は、USER1 の TYPE2 に対して GRANT OPTION 付きの EXECUTE 権限を持っているためです。

```
GRANT EXECUTE ON type3 TO user3 ;
GRANT SELECT on tab2 TO user3 ;
```

ただし、次の権限付与は正しく実行されません。USER2 は、USER1 の TYPE1 に対して GRANT OPTION 付きの EXECUTE を持っていないためです。

```
GRANT SELECT ON tab1 TO user3;
```

USER3 は、次の文を正常に実行できます。

```
CREATE TYPE type4 AS OBJECT (
  attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

型アクセスとオブジェクト・アクセスについての権限

DML コマンドについて列レベルと表レベルの権限が存在する場合は、列オブジェクトと行オブジェクトの両方に適用されます。

オブジェクト表の SELECT 権限があれば、ユーザーは表からオブジェクトとその属性にアクセスでき、UPDATE 権限があればその行を構成するオブジェクトの属性を変更できます。また、オブジェクト表の INSERT 権限があれば、ユーザーは表に新規オブジェクトを作成でき、DELETE 権限があれば行、つまりオブジェクトを削除できます。

同様に、列オブジェクトには表権限と列権限が適用されます。インスタンスを検索するだけでは、型情報は明らかになりません。ただし、クライアントは、型インスタンスのイメージを解釈するために、名前付きの型の情報にアクセスする必要があります。クライアントからこの種の型情報を要求されると、Oracle は型の EXECUTE 権限をチェックします。

次のスキーマを考えてみます。

```
CREATE TYPE emp_type (
    eno NUMBER, ename CHAR(31), eaddr addr_t);
CREATE TABLE emp OF emp_t;
```

さらに、次の 2 つの問合せについて考えます。

```
SELECT VALUE(emp) FROM emp;
SELECT eno, ename FROM emp;
```

どちらの問合せの場合も、Oracle は、EMP 表に対するユーザーの SELECT 権限をチェックします。最初の問合せの場合、ユーザーは、EMP_TYPE の型情報を取得してデータを解釈する必要があります。問合せによって EMP_TYPE 型がアクセスされると、Oracle はユーザーの EXECUTE 権限をチェックします。

ただし、2 番目の問合せを実行しても、名前付きの型が含まれていないため、Oracle は型の権限をチェックしません。

さらに、前の項のスキーマを使用して、USER3 は次の問合せを実行できます。

```
SELECT tab1.col1.attr2 from user2.tab1 tab1;
SELECT attr4.attr3.attr2 FROM tab3;
```

USER3 は基礎となる型に対する明示的な権限を持っていませんが、USER3 によるどちらの SELECT 文も成功するので注意してください。型および表の所有者が、GRANT OPTION 付きの必要な権限を持っているためです。

Oracle は、次の要求に対する権限をチェックし、クライアントがそのアクションに必要な権限を持っていなければエラーを戻します。

- オブジェクトの REF 値を使用してオブジェクト・キャッシュ内でオブジェクトをピンするときには、Oracle は、そのオブジェクトを含んでいるオブジェクト表に対する SELECT 権限をチェックする。
- 既存のオブジェクトを修正したり、オブジェクト・キャッシュからオブジェクトをフラッシュするときには、Oracle は目的のオブジェクト表に対する UPDATE 権限をチェックする。
- 新しいオブジェクトをフラッシュするときには、Oracle は目的のオブジェクト表に対する INSERT 権限をチェックする。
- オブジェクトを削除するときには、Oracle は目的の表に対する DELETE 権限をチェックする。
- 名前付きの型のオブジェクトをピンするときには、Oracle はそのオブジェクトに対する EXECUTE 権限をチェックする。

クライアントの 3GL アプリケーション内でオブジェクトの属性を変更するときには、Oracle はオブジェクト全体を更新します。したがって、ユーザーにはオブジェクト表に対する UPDATE 権限が必要です。アプリケーションがオブジェクト表の特定の列に対応する属性を変更する場合でも、その列のみに対する UPDATE 権限では不十分です。したがって、Oracle は、オブジェクト表に対する列レベルの権限はサポートしていません。

型の依存性

プロシージャや表などのストアド・オブジェクトと同様に、他のオブジェクトから参照される型を、依存性があるといいます。表が型に依存する特殊な問題点がいくつかあります。アクセス時に型定義に依存するデータが表に含まれている場合は、その型を変更すると、その表に格納されているすべてのデータにアクセスできなくなります。変更によってこのような結果になるのは、型に必要な権限が取り消される場合や、型または依存型が削除される場合です。このどちらかのアクションが発生すると、表は無効になり、アクセスできなくなります。

必要な権限が再び付与されると、権限がないために無効になった表は自動的に有効になり、アクセスできるようになります。依存型が削除されていたために無効になった表は、アクセス不能のままで、実行できるアクションは削除のみです。

型に対する権限を取り消したり型を削除すると重大な影響が生じるため、デフォルトでは SQL コマンド REVOKE および DROP TYPE は限定されたセマンティクスをインプリメントします。つまり、どちらのコマンドでも、名前付きの型に表または型依存性があると、エラーが戻されてコマンドは異常終了します。ただし、どちらのコマンドも、FORCE コマンドを使用すると常に正常終了し、表への依存性がある場合は依存性が無効になります。

追加情報： REVOKE、DROP TYPE および FORCE オプションの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

ロール

Oracle では、ロールを使用することにより、簡単かつ制御された権限管理を実現できます。「ロール」は、関連する権限のグループに名前を付けたもので、ユーザーまたは他のロールに付与します。ロールは、エンド・ユーザーのシステム権限とスキーマ・オブジェクト権限を容易に管理できるように設計されています。ただし、ロールは、アプリケーション開発者が使用することを目的としたものではありません。ストアド・プログラム構成体の中からスキーマ・オブジェクトにアクセスする権限は、直接付与する必要があるからです。プロシージャに関する制限の詳細は、30-19 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さらに容易になります。

権限管理に要する労力の削減	複数のユーザーに対して同一の権限セットを明示的に付与するかわりに、関連するユーザー・グループのための権限をまとめて 1 つのロールに付与しておき、そのグループの各メンバーにはそのロールを付与するだけですみます。
動的な権限管理	あるグループの権限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すみます。グループのロールを付与した全ユーザーのセキュリティ・ドメインには、そのロールに対して加えられる変更が自動的に反映されます。
権限の選択的な可用性	あるユーザーに付与したロールを、選択的に使用可能または使用禁止にできます。この機能によって、どのような状況でもユーザー権限を個々に制御できます。
アプリケーションによる認識	ロールの存在はデータ・ディクショナリに記録されます。したがって、ユーザーが特定のユーザー名でアプリケーションを実行したときに、アプリケーションがディクショナリに問い合せ、自動的に特定のロールを使用可能（または使用禁止）にするようにアプリケーションを設計できます。
アプリケーション固有のセキュリティ	ロールの使用はパスワードを使用して保護できます。正しいパスワードを入力するとロールが使用可能になるようなアプリケーションを作成できます。パスワードを知らないユーザーは、ロールを使用可能にできません。

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手順は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

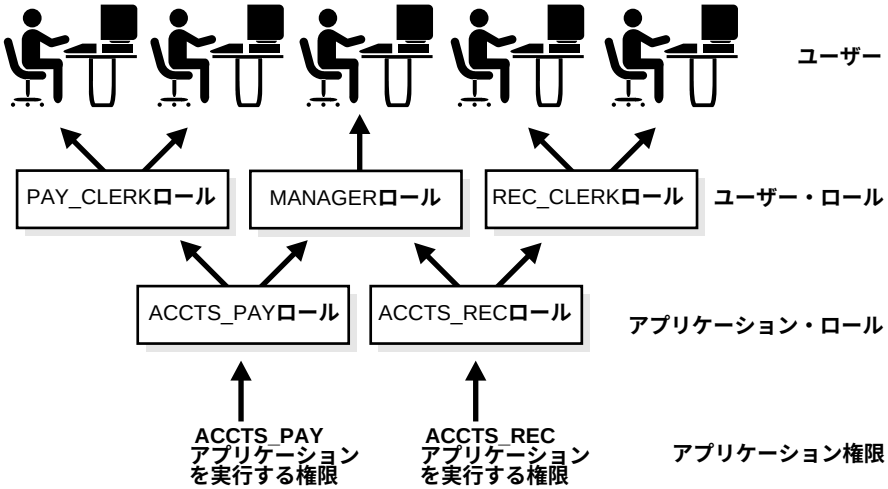
ロールの一般的な使用方法

通常は、次のどちらかの目的でロールを作成します。

- データベース・アプリケーションに対する権限の管理
- ユーザー・グループに対する権限の管理

図 30-1 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 30-1 ロールの一般的な使用方法



アプリケーション・ロール

アプリケーション・ロールには、特定のデータベース・アプリケーションを実行するために必要な権限をすべて付与します。そして、そのアプリケーション・ロールを、他のロールや特定のユーザーに対して付与します。1 つのアプリケーションに対して複数の異なるロールを設定し、アプリケーション使用時のデータ・アクセスの量や範囲に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成されるロールです。ユーザーの権限は、アプリケーションのロールと権限をユーザー・ロールに付与し、そのユーザー・ロールを適切なユーザーに付与することによって管理します。

ロールのメカニズム

データベース・ロールには次の機能があります。

- ロールには、システム権限またはスキーマ・オブジェクト権限を付与できる。
- 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、循環的に付与することはできません（たとえば、ロール B があらかじめロール A に付与されている場合、ロール A をロール B には付与できません）。
- 任意のロールを、任意のデータベース・ユーザーに付与できる。
- 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、権限を選択的に使用できます。
- 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含まれる間接的に付与された全ロールは、すべて暗黙のうちに使用可能になります。

ロールの付与と取消し

ユーザーや別のロールに対してロールを付与したり取り消すには、次の方法があります。

- Oracle Enterprise Manager の「Grant System Privileges/Roles」ダイアログ・ボックスおよび「Revoke System Privileges/Roles」ダイアログ・ボックス
- SQL 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同じオプションを使用します。また、ロールは、Oracle を実行しているオペレーティング・システムや、ネットワーク・サービスを使用することによっても、ユーザーに対して付与したり取り消したりできます。

追加情報： ロール管理の詳細は、『Oracle8*i* 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*i* 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すことができます。つまり、このオプションにより、選択的なロールの管理が可能になります。

ロールの命名

各ロール名はデータベース内で一意にする必要があり、ユーザー名とロール名を同一にすることはできません。スキーマ・オブジェクトとは異なり、ロールはスキーマに「含まれている」わけではありません。そのため、ロールを作成したユーザーを削除しても、そのロールに影響はありません。

ロールとユーザーのセキュリティ・ドメイン

各ロールと各ユーザーは、それぞれ独自のセキュリティ・ドメインを持っています。ロールのセキュリティ・ドメインには、ロールそのものに付与されている権限と、そのロールに付与されたロールに対して付与されている権限が含まれます。

ユーザーのセキュリティ・ドメインには、対応するスキーマ内のすべてのスキーマ・オブジェクトの権限、そのユーザーに対して付与されている権限、およびそのユーザーに対して付与されていて「現在使用可能」になっているロールの権限が含まれます。（1 つのロールをあるユーザーに対して使用可能にし、別のユーザーに対しては使用禁止にすることもできます。）さらに、ユーザーのセキュリティ・ドメインには、ユーザー・グループ PUBLIC に対して付与されている権限とロールも含まれます。

PL/SQL ブロックとロール

PL / SQL ブロック内でのロールの使用方法は、それが無名ブロックであるか名前付きブロック（ストアド・プロシージャ、ファンクションまたはトリガー）であるか、および定義者権限と起動者権限のどちらで実行されるかに応じて異なります。

定義者権限を持つ名前付きブロック

定義者権限で実行される名前付き PL / SQL ブロック（ストアド・プロシージャ、ファンクションまたはトリガー）では、すべてのロールは使用禁止になっています。ロールは権限チェックに使用されず、定義者権限プロシージャ内ではロールを設定できません。

SESSION_ROLES ビューは、現在使用可能になっているすべてのロールを示します。定義者権限で実行される名前付き PL/SQL ブロックが SESSION_ROLES を問い合せると、問合せは行を戻しません。

起動者権限と無名ブロック

起動者権限で実行される名前付き PL/SQL ブロックと、無名 PL/SQL ブロックは、使用可能になっているロールを通じて付与された権限に基づいて実行されます。起動者権限を持つ PL/SQL ブロック内での権限チェックにはロールが使用され、動的 SQL を使用してセッション中にロールを設定できます。

起動者権限と定義者権限の説明は 18-9 ページの「定義者権限と起動者権限」、動的 SQL の詳細は 16-19 ページの「PL/SQL の動的 SQL」を参照してください。

データ定義言語の文とロール

ユーザーがデータ定義言語（DDL）文を正常に実行するには、その文に応じて 1 つ以上の権限が必要になります。たとえば、表を作成するには、CREATE TABLE または CREATE ANY TABLE システム権限が必要です。別のユーザーの表のビューを作成するには、CREATE VIEW または CREATE ANY VIEW システム権限だけでなく、その表に対する SELECT オブジェクト権限または SELECT ANY TABLE システム権限も必要です。

Oracle では、特定の DDL 文での特定の権限の使用を制限することによって、ロールを介して受け取った権限への依存性を回避します。次の規則は、DDL 文に対する権限の制限を示しています。

- DDL 操作の実行をユーザーに許可するすべてのシステム権限およびスキーマ・オブジェクト権限は、ロールを介して受け取った場合でも使用可能。

  例 :

  – システム権限 : CREATE TABLE、CREATE VIEW および CREATE PROCEDURE の各権限。

  – スキーマ・オブジェクト権限 : 表に対する ALTER および INDEX の各権限。

  例外 : 表に対する REFERENCES オブジェクト権限は、それがロールを介して付与された場合には、表の外部キー定義には使用できません。

- DDL 文の発行に必要な DML 操作の実行をユーザーに許可するすべてのシステム権限およびオブジェクト権限は、ロールを介して受け取った場合は使用不可。

  例 :

  – 表に対する SELECT ANY TABLE システム権限や SELECT オブジェクト権限がロールを介して付与されている場合、それらの権限を使用して別のユーザーの表に基づくビューを作成することはできません。

ロールを介して受け取った権限の使用許可と使用制限について、次の例で具体的に説明します。

例 : 次のようなユーザーを想定します。

- CREATE VIEW システム権限を持つロールが付与されている。
- 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成できない。作成されたビューにアクセスすると、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 表のビューは作成できる。

事前定義済みのロール

ロール CONNECT、RESOURCE、DBA、EXP_FULL_DATABASE および 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提供されており、Oracle データベース内の他のロールと同じ方法で修正できます。

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・システムを使用して、データベース・ロールの付与と取消しや、パスワードの認証を管理できます。この機能は、すべてのオペレーティング・システムで利用できるとは限りません。

> **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・システム固有の Oracle マニュアルを参照してください。

分散環境におけるロール

分散データベース環境でロールを使用する場合は、必要なすべてのロールを分散（リモート）セッションのデフォルト・ロールとして設定する必要があります。ローカル・データベース・セッション内からリモート・データベースに接続しているときは、ロールを使用可能にできません。たとえば、リモート・サイトでロールを使用可能にしようとするリモート・プロシージャは実行できません。

追加情報： 分散データベース環境の詳細は、『Oracle8*i* 分散システム』を参照してください。

ファイン・グレイン・アクセス・コントロール

ファイン・グレイン・アクセス・コントロールにより、ファンクションにセキュリティ・ポリシーを適用し、セキュリティ・ポリシーを表やビューに対応付けることができます。これらのセキュリティ・ポリシーは、データがアクセスされるかどうか（非定型問合せなど）に関係なく、データベース・サーバーによって自動的に施行されます。

次のことができます。

- SELECT、INSERT、UPDATE および DELETE に異なるポリシーを使用する。
- 必要な場合（給与情報など）にのみセキュリティ・ポリシーを使用する。
- パッケージ化されたアプリケーションの最上位に基本ポリシーを構築するなど、各表に複数のポリシーを使用する。

PL/SQL パッケージ DBMS_RLS を使用すると、セキュリティ・ポリシーを管理できます。このパッケージを使用して、作成したポリシーを追加、削除、使用可能と使用禁止の切替えおよびリフレッシュを行います。PL/SQL パッケージの使用方法は、18-11 ページの「パッケージ」を参照してください。

追加情報： パッケージのインプリメンテーションの詳細は、『Oracle8*i* パッケージ・プロシージャ リファレンス』を参照してください。

動的述語

作成したセキュリティ・ポリシーを実現するファンクションまたはパッケージは、述語（WHERE 条件）を戻します。この述語によって、ポリシーで設定されたとおりにアクセスが制御されます。再書込みされた問合せは、完全に最適化され、共有可能になります。

セキュリティ・ポリシーの例

次のセキュリティ・ポリシーの例を考えます。

HR という人事管理アプリケーションでは、EMPLOYEES は ALL_EMPLOYEES 表のビューであり、どちらのオブジェクトも APPS スキーマに属しています。この表とビューを作成するコマンドは、次のとおりです。

```
CREATE TABLE all_employees
    (employee_id  NUMBER(15),
     emp_name     VARCHAR2(30),
     mgr_id       NUMBER(15),
     user_name    VARCHAR2(8), .... );
CREATE VIEW employees AS SELECT * FROM all_employees;
```

社内のユーザーのロールに基づいて EMPLOYEES ビューへのアクセスを制限するために、セキュリティ・ポリシー・ファンクションを作成する必要があります。このポリシーの述語は、HR_ACCESS パッケージ内の SECURE_PERSON ファンクションで生成できます。このパッケージはスキーマ APPS に属し、HR アプリケーションに関連するすべてのセキュリティ・ポリシーをサポートするファンクションが含まれています。また、すべてのアプリケーション・コンテキストは、APPS_SEC 名前領域にあります。この例のアプリケーション・コンテキストを作成するには、次のコマンドを使用します。

```
CREATE CONTEXT hr_role USING apps_sec.hr_role
```

セキュリティ・ポリシー・ファンクションを作成するコマンドは、次のとおりです。

```
CREATE PACKAGE BODY hr_access IS
    FUNCTION secure_person(obj_schema VARCHAR2, obj_name VARCHAR2)
             RETURN VARCHAR2 IS
        d_predicate VARCHAR2(2000);
    BEGIN
        IF SYS_CONTEXT ('apps_sec', 'hr_role') = 'EMP' THEN
            d_predicate = 'emp_name = sys_context(''userenv'', ''user'')';
        IF SYS_CONTEXT ('apps_sec', 'hr_role') = 'MGR' THEN
            d_predicate = 'mgr_id = sys_context(''userenv'', ''uid''))';
        ELSE
             d_predicate = '1=2';  -- deny access to other users,
                                   -- may use something like 'keycol=null'
        RETURN d_predicate;
    END secure_person;
END hr_access;
```

次のステップでは、EMPLOYEES ビューのポリシー（この例では PER_PEOPLE_SEC）を、動的述語を生成する HR_ACCESS.SECURE_PERSON ファンクションに対応付けます。

```
DBMS_RLS.ADD_POLICY('apps', 'employees', 'per_people_sec', 'apps'
                    'hr_access.secure_person',  'select, update, delete');
```

これで、EMPLOYEES ビューに関係する SELECT、UPDATE および DELETE 文では、アプリケーション・コンテキスト HR_ROLE の値に基づいて、3 つの述語の 1 つが選択されます。

また、ALL_EMPLOYEES 表を保護していたのと同じセキュリティ・ポリシー・ファンクションを使用して、ADDRESSES 表を保護する動的述語を生成できるので注意してください。この 2 つの表には、データへのアクセスを制限するために同じポリシーが適用されているからです。

> **追加情報：** セキュリティ・ポリシーの設定方法の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』を参照してください。

アプリケーション・コンテキスト

アプリケーション・コンテキストにより、ファイン・グレイン・アクセス・コントロールの実現が容易になります。また、関数にセキュリティ・ポリシーをインプリメントし、そのセキュリティ・ポリシーをアプリケーションに対応付けることができます。各アプリケーションには、固有のコンテキストを対応付けることができます。ユーザーは、そのコンテキストを任意に（SQL*Plus などを通じて）変更することは許されません。

アプリケーション・コンテキストにより、アプリケーションに関係する属性に基づき、パラメータベースの柔軟なアクセス制御が可能になります。たとえば、人事管理アプリケーションのコンテキスト属性には「職位」、「部門」および「国」などを含め、受注管理アプリケーションの属性には「顧客番号」や「営業地区」などを含めることができます。

次のことができます。

- コンテキスト値の基礎に述語を使用する。
- 述語内でコンテキスト値をバインド変数として使用する。
- ユーザー属性を設定する。
- ユーザー属性にアクセスする。

アプリケーション・コンテキストを定義する手順

1. アプリケーションのコンテキストを検査して設定する関数を使用して、PL/SQL パッケージを作成します。ログイン時にイベント・トリガーを使用して、ログイン・ユーザーの初期コンテキストを設定できます。
2. CREATE CONTEXT を使用して一意のコンテキスト名を指定し、作成した PL/SQL パッケージに対応付けます。
3. 次のどちらかの操作を行います。
   - ファイン・グレイン・アクセス・コントロールをインプリメントするポリシー関数内で、アプリケーション・コンテキストを参照する。
   - ログイン時にイベント・トリガーを使用して、ユーザーの初期コンテキストを設定する。たとえば、ユーザーの従業員番号を問い合せて、「従業員番号」コンテキスト値として設定できます。
4. アプリケーション・コンテキストを参照します。

追加情報： PL/SQL パッケージの作成方法は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle8*i* パッケージ・プロシージャリファレンス』を参照してください。

31

監査

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、権限およびスキーマ・オブジェクトの監査対象の限定

監査の基礎知識

「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用されます。

- 動作が不審なアクティビティの調査。たとえば、無許可ユーザーが表からデータを削除しようとした場合、セキュリティ管理者は、そのデータベースへのすべての接続と、そのデータベースにあるすべての表からの行の削除（成功および失敗）をすべて監査できます。
- 特定のデータベース・アクティビティに関するデータの監視と収集。たとえば、データベース管理者は、更新された表、実行された論理 I/O の回数、またはピーク時に接続していた同時実行ユーザーの数などに関する統計を収集できます。

監査機能

ここでは、Oracle の監査メカニズムの概要について説明します。

監査のタイプ

Oracle は次の 3 つのタイプの一般監査機能をサポートします。

文監査	SQL 文が処理する特定のスキーマ・オブジェクトではなく、文のタイプに関してのみ実行する SQL 文の選択的な監査。文監査のオプションは通常、適用範囲が広く、オプションごとに何種類かの関連したアクションの使用方法を監査します。たとえば、AUDIT TABLE は、それがどの表に対して発行されたかには関係なく、いくつかの DDL 文を追跡します。文監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
権限監査	AUDIT CREATE TABLE など、システム・アクションを実行する強力なシステム権限の使用方法を監査する選択的な監査。権限監査は、対象となる権限の使用方法のみを監査するので、監査対象が文監査より限定されています。権限監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
スキーマ・オブジェクト監査	AUDIT SELECT ON EMP など、特定のスキーマ・オブジェクトの特定の文に対する選択的な監査。スキーマ・オブジェクト監査は対象が非常に限定されており、特定のスキーマ・オブジェクトに対する特定の文のみを監査します。スキーマ・オブジェクト監査は、データベースのすべてのユーザーに常に適用されます。

監査の対象

Oracle では、監査オプションの対象範囲を広くしたり狭くしたりできます。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、または文が実行されるごとの文の実行の監査
- 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。この表の情報を使用しやすくするため、複数の事前定義済みのビューが提供されています。

> **追加情報：** これらのビューの作成方法は、『Oracle8*i* 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常に記録されます。

- ユーザー名
- セッション識別子
- 端末識別子
- アクセスされたスキーマ・オブジェクトの名前
- 実行または試行された操作
- 操作の完了コード
- 日時のタイム・スタンプ
- 使用されたシステム権限

オペレーティング・システム監査証跡はコード化されていて読むことはできませんが、次のようなデータ・ディクショナリ・ファイルとエラー・メッセージにデコードできます。

アクション・コード　実行または試行された操作の記述。これらのコードとその記述のリストは、AUDIT_ACTIONS データ・ディクショナリ表にあります。

使用された権限	操作の実行に使用されたシステム権限の記述。これらのコードとその記述は、SYSTEM_PRIVILEGE_MAP 表にあります。
完了コード	試行された操作の結果の記述。成功した操作からは、値 0 が戻されます。失敗した操作からは、操作が異常終了した理由を説明する Oracle エラー・コードが戻されます。これらのコードの詳細は、『Oracle8*i* エラー・メッセージ』を参照してください。

監査のメカニズム

この項では、Oracle の監査機能で使用されているメカニズムについて説明します。

監査レコードが生成される場合

監査情報の記録機能は、使用可能または使用禁止にできます。この機能により、許可されたデータベース・ユーザーは監査オプションをいつでも設定できますが、監査情報の記録を制御する操作はセキュリティ管理者のために確保されています。

> **追加情報：** 監査を使用可能および使用禁止にする手順は、『Oracle8*i* 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 16 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常に記録します。

インスタンス起動	インスタンスを起動した 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、およびデータベース監査が使用可能になっていたかどうかを記述した監査レコードが生成されます。データベース監査証跡は起動が正常に完了しないと使用可能にならないため、この情報は 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ため、管理者が、データベース監査使用禁止の状態のままデータベースを再起動して、監査されないアクションを実行できないようになっています。
インスタンス停止	インスタンスを停止した OS ユーザー、そのユーザーの端末識別子および日時のタイム・スタンプを記述した監査レコードが生成されます。
管理者権限によるデータベースへの接続	Oracle に管理者権限によって接続する 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 からはアクセスできるようになっていないオペレーティング・システムの場合、これらの監査証跡レコードは、バックグラウンド・プロセスのトレース・ファイルと同じディレクトリにある Oracle 監査証跡ファイルに格納されます。

> **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・システム固有の Oracle マニュアルを参照してください。

監査オプションが有効になる時期

データベース・ユーザーがデータベースに接続した時点で有効になっていた文監査オプションと権限監査オプションは、そのセッションの持続期間中は有効です。文監査または権限監査のオプションの設定または変更の結果は、セッション中には有効になりません。修正した文監査オプションまたは権限監査オプションは、現行のセッションを終了し、新しいセッションを作成する時点で有効になります。一方、オブジェクト監査オプションについて変更した内容は、現行セッションでただちに有効になります。

分散データベースの監査

監査機能はサイト自律性を持っています。つまり、インスタンスは、直接接続しているユーザーが発行する文のみを対象に監査します。ローカル Oracle ノードは、リモート・データベースで発生するアクションを監査できません。リモート接続はデータベース・リンクのユーザー・アカウントを介して確立されるので、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 33 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

オペレーティング・システムの監査証跡が使用可能になっている場合に、Oracle は監査証跡レコードをオペレーティング・システムの監査証跡に送ることができます。その他のオペレーティング・システムの場合、これらの監査レコードが、他の Oracle トレース・ファイルと似た形式でデータベース外のファイルに書き込まれることもあります。

> **追加情報：** この機能がオペレーティング・システムでインプリメントされているかどうかを調べるには、プラットフォーム固有の Oracle マニュアルを参照してください。

Oracle では、オペレーティング・システムの監査証跡（または監査レコードを含むオペレーティング・システム・ファイル）に監査レコードを記録できない場合にも、常に特定のアクションを引き続き監査できます。オペレーティング・システムの監査証跡に記録できないのは、多くの場合、オペレーティング・システムの監査証跡またはファイル・システムが満杯になっており、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・システムが満杯にならないようにしてください。ほとんどのオペレーティング・システムでは、このような状況を回避できるように十分な情報と警告が管理者に提供されます。ただし、データベースの監査証跡を使用するように監査を構成すれば、その危険性を回避でき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、関連する文のグループに対する選択的な監査のことです。文は、次の 2 つに分類できます。

- 特定のタイプのデータベース構造体やスキーマ・オブジェクトに関する DDL 文。ただし、監査の対象として特定の構造体やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT TABLE はすべての CREATE TABLE 文と DROP TABLE 文を監査します）。
- 特定のタイプのデータベース構造体やスキーマ・オブジェクトに関する DML 文。ただし、監査の対象として特定の構造体やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT SELECT TABLE は、表、ビューまたはスナップショットのどれであるかに関係なく、すべての SELECT ... FROM TABLE / VIEW / SNAPSHOT 文を監査します）。

文監査では、監査の対象範囲を広げてすべてのデータベース・ユーザーのアクティビティを監査したり、範囲を限定して特定のデータベース・ユーザーのアクティビティだけを監査できます。

権限監査

権限監査は、システム権限の使用を許可されている文を選択的に監査します。たとえば、SELECT ANY TABLE システム権限の監査は、SELECT ANY TABLE システム権限を使用して実行されるユーザーの文を監査します。任意のシステム権限の使用を監査できます。

権限監査では、システム権限の監査の前に、所有者権限とスキーマ・オブジェクト権限が必ず検査されます。所有者権限とスキーマ・オブジェクト権限がアクションを許可するのに十分である場合、そのアクションは監査されません。

類似の文監査オプションと権限監査オプションを両方設定しても、監査レコードは 1 つしか生成されません。たとえば、文オプション TABLE とシステム権限の CREATE TABLE の両方を監査すると、表が作成されるたびに監査レコードが 1 つだけ生成されます。

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 TABLE では CREATE TABLE、ALTER TABLE および DROP TABLE 文が監査されますが、権限監査オプション CREATE TABLE では CREATE TABLE 文のみが監査されます。これは、CREATE TABLE 権限を必要とするのが CREATE TABLE 文のみであるためです。

文監査と同様に、権限監査では、すべてのデータベース・ユーザーのアクティビティを監査したり、特定のデータベース・ユーザーのアクティビティだけを監査できます。

スキーマ・オブジェクト監査

スキーマ・オブジェクト監査は、特定のスキーマ・オブジェクトの特定の DML 文（問合せを含む）、および GRANT 文と REVOKE 文の選択的な監査です。スキーマ・オブジェクト監査では、特定の表に対する SELECT 文や DELETE 文など、スキーマ・オブジェクト権限によって許可される操作と、それらの権限を制御する GRANT 文と REVOKE 文が監査されます。

表、ビュー、順序、スタンドアロンのストアド・プロシージャとストアド・ファンクションおよびパッケージを参照する文を監査できます（パッケージ内のプロシージャは個別には監査できません）。

クラスタ、データベース・リンク、索引またはシノニムを参照する文は、直接監査できません。ただし、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*i* 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、パッケージおよびトリガーを含む）は、その定義の基礎を形成するスキーマ・オブジェクトを参照します。そのため、ビューとプロシージャに関する監査には、いくつかの固有の特性があります。ビューやプロシージャを使用すると、その結果として複数の監査レコードが生成される可能性があります。ビューやプロシージャの使用は、使用可能になっている監査オプションに依存します。ビューやプロシージャを使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）に依存します。

次の一連の SQL 文について考えます。

```
AUDIT SELECT ON emp;

CREATE VIEW emp_dept AS
  SELECT empno, ename, dname
    FROM emp, dept
    WHERE emp.deptno = dept.deptno;

AUDIT SELECT ON emp_dept;

SELECT * FROM emp_dept;
```

この EMP_DEPT に対する問合せの結果、2 つの監査レコードが生成されます。1 つは EMP_DEPT ビューの問合せについての監査レコード、もう 1 つは実表 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用して共有プールに配置する時点で、判別されます。そのビューまたはプロシージャをいったんフラッシュして共有プールに再配置するまで、これらの監査オプションは設定されたままになります。スキーマ・オブジェクトを監査すると、キャッシ内のスキーマ・オブジェクトが無効になるため、スキーマ・オブジェクトを再ロードすることになります。ベース・スキーマ・オブジェクトの監査オプションに対する変更は、共有プール内のビューとプロシージャには認識されません。

前述の例の続きとして、EMP 表に対する SELECT 文の監査をオフにすると、EMP_DEPT ビューを使用しても EMP 表の監査レコードは生成されなくなります。

文、権限およびスキーマ・オブジェクトの監査対象の限定

Oracle では、文、権限およびスキーマ・オブジェクトのそれぞれの監査の対象を、次の 3 つの分野に限定できます。

- 監査される SQL 文の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（文監査と権限監査のみ）

成功した文の実行と失敗した文の実行の監査

文、権限およびスキーマ・オブジェクトの監査では、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、監査する文が成功しなかった場合にも、アクションを監査できます。

失敗した文の実行を監査できるのは、有効な SQL 文を実行し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の実行のみを監査する場合に使用します。
- WHENEVER NOT SUCCESSFUL オプション。監査対象の文の失敗した文の実行のみを監査する場合に使用します。
- 前述のどちらのオプションも使用しない。成功した文と失敗した文の両方の実行を監査する場合に使用します。

BY SESSION 監査と BY ACCESS 監査

ほとんどの監査オプションでは、1 つのユーザー・セッションで監査対象の文が複数回発行された場合の監査レコードの生成方法を指定できます。ここでは、AUDIT 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中に、ユーザーおよびスキーマ・オブジェクト当たり 1 つの監査レコードのみを監査証跡に挿入します。

セッションとは、ユーザーが Oracle データベースに接続してから切断するまでの期間です。

例 1 この例では、次のような状況を想定します。

- SELECT TABLE 文監査オプションを BY SESSION に設定する。
- JWARD でデータベースに接続し、DEPT という表に対して 5 つの SELECT 文を発行した後、そのデータベースとの接続を切断する。
- SWILLIAMS でデータベースに接続し、表 EMP に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断する。

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 でデータベースに接続し、DEPT という表に対して 5 つの SELECT 文、表 EMP に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断する。

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 オプションを使用すると、Oracle ではアクセスするたびに監査レコードが生成され格納されます。したがって、この監査構成の場合、BY SESSION は BY ACCESS と等価です。

BY ACCESS

監査を BY ACCESS に設定すると、カーソル内で監査対象の操作が実行されるたびに、監査証跡に監査レコードが 1 つ挿入されます。カーソルを再使用させるイベントは、次のとおりです。

- カーソルを再使用するためにオープンしておく、Oracle Forms などのアプリケーション
- 新しいバインド変数を使用してカーソルを再実行すること
- PL/SQL ループ内で実行される文で、1 つのカーソルを再使用するために PL/SQL エンジンにより文が最適化される場合

監査は、カーソルが共有されているかどうかの影響を受けないので注意してください。それぞれのユーザーは、カーソルの最初の実行時に自分の監査証跡レコードを作成します。

例 次のような場合を想定します。

- SELECT TABLE 文監査オプションを BY ACCESS に設定する。
- JWARD でデータベースに接続し、DEPT という表に対して 5 つの SELECT 文を発行した後、そのデータベースとの接続を切断する。

- SWILLIAMS でデータベースに接続し、表 DEPT に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断する。

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 と BY ACCESS のどちらかを指定できます。ただし、次のような一部の監査オプションでは、BY ACCESS しか設定できません。

- DDL 文を監査するすべての文監査オプション
- DDL 文を監査するすべての権限監査オプション

他のすべての監査オプションでは、デフォルトで BY SESSION が設定されています。

ユーザー別の監査

文監査オプションと権限監査オプションでは、任意のユーザーが発行した文を監査するか、特定のユーザー・リストに含まれるユーザーが発行する文を監査するかを選択できます。特定のユーザーに限定すると、生成される監査レコードの数は最小限に抑えられます。

> **追加情報：** ユーザー別の監査の詳細は、『Oracle8*i* 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せたり更新するためにユーザー SCOTT および BLAKE によって発行される文を監査するには、次の文を発行します。

```
AUDIT SELECT TABLE, UPDATE TABLE
     BY scott, blake;
```

32

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用される構造、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用される構造
- ロールフォワードとロールバック
- 回復パフォーマンスの改善
- Recovery Manager
- データベースのアーカイブ・モード
- 制御ファイル
- データベースのバックアップ
- 耐障害性

追加情報： バックアップおよびリカバリ構造の作成とメンテナンスに必要な手順の詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

データベース回復の基礎知識

データベース管理者の主な責任の 1 つは、ハードウェア、ソフトウェア、ネットワーク、プロセスまたは発生する可能性のある障害に対処する準備をしておくことです。そのような障害がデータベース・システムの操作に影響する場合、通常は速やかにデータベースを回復し、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すむようにする必要があります。

回復プロセスは、発生した障害のタイプ、その障害の影響を受けた構造および実行する回復のタイプに応じて異なります。消失または破損したファイルがなければ、インスタンスを再起動するだけで回復できます。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）の操作を容易にするユーティリティです。32-14 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*i* バックアップおよびリカバリ・ガイド』を参照してください。

エラーと障害

いくつかの問題が原因で、Oracle データベースの通常の操作が停止したり、ディスクへのデータベース I/O が影響を受けることがあります。この後の項では、最も一般的なタイプの障害について説明します。障害によっては、回復が自動的に実行される場合や、データベース・ユーザーやデータベース管理者による処置がほとんど、またはまったく必要ない場合もあります。

ユーザー・エラー

データベース管理者は、ユーザー・エラー（表を誤って削除してしまうなど）をほとんど防止できません。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・エラーは減少します。また、管理者が事前に効果的な回復方法を準備しておけば、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の異常終了など）。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現行のトランザクションをロールバックし、そのプロセスが使用していたリソースを解放して、この障害を解決します。エラーになったユーザー・プロセスまたはサーバー・プロセスは、自動的に回復されます。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、インスタンスは正常に機能できなくなります。このため、インスタンスを停止して再起動する必要があります。

ネットワーク障害

システムでネットワーク（ローカル・エリア・ネットワーク、電話回線など）を使用してクライアント・ワークステーションをデータベース・サーバーに接続したり、複数のデータベース・サーバーを接続して分散データベース・システムを形成している場合に、ネットワーク障害（電話接続の異常終了やネットワーク通信ソフトウェアの障害など）が発生すると、データベース・システムの通常の操作に割込みが発生することがあります。次に例を示します。

- クライアント・アプリケーションの正常な実行に割り込み、プロセス障害を引き起こす。前の項で説明したとおり、この場合は Oracle バックグラウンド・プロセス PMON が、接続を切断されたユーザー・プロセスに対応する、異常終了したサーバー・プロセスを検出し、そのサーバー・プロセスを解決します。
- 分散トランザクションの 2 フェーズ・コミットに割り込む。ネットワークの問題が訂正されると、この問題に関連していたそれぞれのデータベース・サーバーの Oracle バックグラウンド・プロセス RECO が、分散データベース・システムのすべてのノードにある未解決の分散トランザクションを、自動的に解決します。分散データベース・システムの詳細は、第 33 章の「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 コマンドや STARTUP FORCE コマンド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 クラッシュ回復またはインスタンス回復では、データベースは、インスタンス障害が発生する直前のトランザクション一貫性状態に回復します。「クラッシュ回復」では単一インスタンス構成のデータベースを回復し、「インスタンス回復」では Oracle Parallel Server 構成でのデータベースを回復します。

インスタンス障害からの回復は自動的に実行されます。たとえば、Oracle Parallel Server を使用している場合、障害インスタンスのインスタンス回復は、他のインスタンスによって実行されます。単一インスタンス構成では、データベースの再起動時にデータベースのクラッシュ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、オープンされます）。必要であれば、マウント状態からオープン状態に推移する時点で、クラッシュ回復が自動的にトリガーされます。

クラッシュまたはインスタンス回復は、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・ファイルを回復します。これを「キャッシュ回復」といいます。
2. データベースをオープンします。Oracle は、すべてのトランザクションがロールバックされるのを待ってからデータベースを使用可能にするのではなく、キャッシュ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。
3. 障害発生時にアクティブであったシステム全体のトランザクションすべてを DEAD とマークし、これらのトランザクションを含むロールバック・セグメントを PARTLY AVAILABLE とマークします。
4. SMON による回復の一部として、デッド・トランザクションをロールバックします。これを「トランザクション回復」といいます。
5. インスタンス障害の発生時に 2 フェーズ・コミットが行われていたために保留になっている分散トランザクションを解決します。
6. 新規トランザクションでは、デッド・トランザクションによってロックされている行を検出すると、デッド・トランザクションを自動的にロールバックしてロックを解除できます。ファースト・スタート・リカバリを使用している場合は、トランザクション全体ではなく、データ・ブロックのみが即時にロールバックされます。

追加情報： インスタンス回復の説明は、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』を参照してください。

インスタンス回復のチューニングの説明は、『Oracle8*i* チューニング』を参照してください。

メディア（ディスク）障害

Oracle データベースの操作に必要なファイルの書込みまたは読込み時にエラーが発生することがあります。記憶域メディア上のファイルの読み書きにかかわる物理的な問題が原因となるため、このような障害は「メディア障害」と呼ばれます。

一般的なメディア障害に、ディスク・ドライブのすべてのファイルが失われるディスク・ヘッド・クラッシュがあります。データ・ファイル、オンライン REDO ログ・ファイルおよび制御ファイルなど、データベースの関連ファイルはすべてディスク・クラッシュの影響を受けます。

メディア障害からの回復の方法は、関係しているファイルによって異なります。

> **追加情報：** 回復方法の説明は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

メディア障害がデータベース操作に及ぼす影響 メディア障害はデータ・ファイル、オンライン REDO ログ・ファイルおよび制御ファイルなど、Oracle データベースの操作に必要なさまざまなタイプのファイルに影響します。

オンライン REDO ログ・ファイルや制御ファイルでメディア障害が発生した場合、オンライン REDO ログ・ファイルや制御ファイルが「多重化」されているかどうかに応じて、それ以降のデータベース操作が異なります（多重化することをお薦めします）。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 が破損制御ファイルに対して読込みまたは書込みを試行した時点でデータベース操作は停止します（すべてのチェックポイントやログ・スイッチなどで頻繁に発生します）。

データ・ファイルに影響するメディア障害は、読込みエラーと書込みエラーの 2 つに分類されます。読込みエラーの場合、Oracle がデータ・ファイルを読み込めないことを発見すると、オペレーティング・システムのエラーと、ファイルが見つからない、オープンできない、または読み込めないことを示す Oracle のエラーがアプリケーションに戻されます。Oracle は続行されますが、このエラーは読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、書込みエラーが発生します。

Oracle がデータ・ファイルに書き込めなかった場合に、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れば、Oracle はエラーを DBW*n* 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、Oracle によりインスタンスが停止されます。Oracle が正常に機能するには、SYSTEM 表領域のすべてのファイルがオンラインになっている必要があるので、この例外が適用されます。同様の理由で、アクティブ・ロールバック・セグメントを含む表領域のデータ・ファイルはオンラインである必要があります。

Oracle がデータ・ファイルに書き込めず、データが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、DBW*n* 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・ファイルを使用したクラッシュまたは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。ただし、データ・ファイルが永続的に破損し、アーカイブされていない場合は、最新のコールド・バックアップからデータベース全体を復元する必要があります。

読込み専用表領域の回復 クラッシュまたはインスタンス回復時には、読込み専用データ・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、読込み専用になる前に作成されたバックアップからは復元されなかったということです。読込み専用の表領域を、その表領域が読込み専用になる前に作成されたバックアップから復元した場合は、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。

データベースの回復で使用される構造

Oracle データベースのいくつかの構造により、データは障害から保護されます。この項では、データベース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メディア回復を開始するために、Oracle はバックアップ・ファイルを使用して、破損したデータ・ファイルまたは制御ファイルを復元します。破損したと思われるデータ・ファイル、表領域またはデータベースの現行コピーをバックアップ・コピーと置き換えることを、データベースのその部分を「復元する」といいます。

Oracle では、次のように、データベース・バックアップを実行するためのオプションがいくつか提供されています。

- Recovery Manager
- オペレーティング・システム・ユーティリティ
- Export ユーティリティ
- Enterprise Backup Utility

追加情報：『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

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。Oracle バックグラウンド・プロセス LGWR は、オンライン REDO ログを使用して、対応付けられているインスタンスによって実行されたすべての変更を即時に記録します。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使用され、進行中のデータベース変更が記録されます。

アーカイブ済み（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ための、Oracle データベースを構成できます。アーカイブ対象のオンライン REDO ログ・ファイルは、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使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、メディア回復などの操作用に過去の REDO ログ情報を保存します。

データ・ファイルは、バックアップから復元された場合や、クリーン・データベース停止でクローズされていない場合は、最新のものでない可能性があります。これらのデータ・ファイルには、アーカイブ済み REDO ログまたはオンライン REDO ログ（あるいはその両方）に記録された変更を適用して更新する必要があります。このプロセスを「回復」といいます。

詳細は、32-17 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用されます。一般に、データベースのロールバック・セグメントには、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古いデータ値が格納されます。

特に、ロールバック・セグメントの情報は、データベースの回復時に 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」ために使用されます。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は、ロールバック・セグメントを使用して、すべてのコミットされていないデータがデータ・ファイルから削除された後です。

制御ファイル

一般に、データベースの制御ファイルには、データベースの物理構造の状態が格納されます。Oracle は、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、32-20 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 のバッファ・キャッシュ内のデータベース・バッファは、最低使用頻度アルゴリズムを使用して、必要な場合にのみディスクに書き込まれます。DBW*n* プロセスはこのアルゴリズムを使用してデータベース・バッファをデータ・ファイルに書き込むので、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る可能性があります。

インスタンス障害が発生すると、次の 2 つの問題が起きる可能性があります。

- トランザクションによって変更されたデータ・ブロックがコミット時にデータ・ファイルに書き込まれず、REDO ログ・ファイルに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。
- ロールフォワード・フェーズの完了後に、データ・ファイルには障害発生時にコミットされていなかった変更が含まれている場合がある。このようにコミットされていない変更は、トランザクションの一貫性を保つためにロールバックする必要があります。このような変更は、障害発生前にデータベースに保存されてたものか、ロールフォワード・フェーズで取り込まれたものです。

この矛盾を解決するには、通常、2 つの別個のステップを実行して回復を実行します。つまり、REDO ログを使用してロールフォワードし（キャッシュ回復）、ロールバック・セグメントを使用してロールバックします（トランザクション回復）。

REDO ログとロールフォワード

REDO ログとは、データ、索引およびロールバック・セグメントなど、データベース・バッファに加えられたすべての変更を、コミット済みかどうかに関係なく記録するオペレーティング・システム・ファイルの集合です。各 REDO ログ・エントリは、データベースに対する 1 つのアトミック変更を記述する変更ベクトルのグループです。REDO ログは、メモリー内のデータベース・バッファに対する変更内容で、データ・ファイルにまだ書き込まれていない情報を保護します。

インスタンスまたはディスク障害から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。これを「キャッシュ回復」といいます。

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・ファイルが使用され、さらにアーカイブ済み REDO ログ・ファイルが使用されることもあります。

ロールフォワード後のデータ・ブロックには、コミットされたデータがすべて含まれています。また、障害発生前にデータ・ファイルに保存された変更や、REDO ログに記録され、ロールフォワード中に取り込まれた変更も含まれている場合があります。

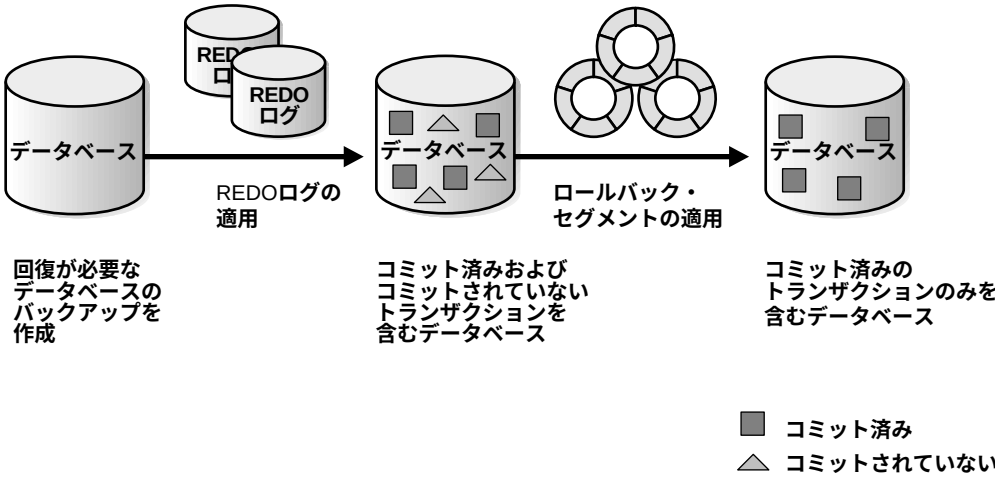
ロールバック・セグメントとロールバック

ロールバック・セグメントは、特定のデータベース操作時に取り消す必要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・フェーズでそれ以前に適用されたコミットされていないトランザクションの効果を取り消します。

ロールフォワードの実行後、コミットされていない変更をすべて取り消す必要があります。REDO ログ・ファイルをデータベースに再適用してすべての変更を反映させた後、対応するロールバック・セグメントを使用することになります。ロールバック・セグメントは、コミットされていなかったのに障害発生前にデータ・ファイルに保存されたか、ロールフォワード時にデータベースに適用されたトランザクションを識別して取り消すために使用します。このプロセスを「ロールバック」または「トランザクション回復」といいます。

図 32-1 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 32-1 基本的な回復手順 : ロールフォワードとロールバック



Oracle では、必要に応じて、複数のトランザクションを同時にロールバックできます。障害発生時にアクティブであったシステム全体のすべてのトランザクションは、DEAD とマークされます。SMON がデッド・トランザクションをロールバックするのを待つのではなく、ブロックしているトランザクションそのものを新しいトランザクションによって回復し、トランザクションで必要な行ロックを取得できます。

回復パフォーマンスの改善

ほとんどの状況では、データベース障害の発生時には迅速に回復することがきわめて重要です。Oracle は、できるだけ短時間で回復するために、次のようにさまざまな方法を提供します。

- パラレル回復
- ファースト・スタート・リカバリ
- 透過的アプリケーション・フェイルオーバー

回復のパラレル実行

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。パラレル回復を使用すると、複数のプロセスが REDO ログ・ファイルにある変更を同時に適用します。

注意： Oracle8*i* では、Recovery Manager での並行性が制限されます。Oracle8*i* Enterprise Edition を使用すると、並行性が無制限になります。

パラレル回復を実行するには、次の 3 つの方法があります。

- パラレル回復は、複数の Oracle Enterprise Manager セッションを生成し、各セッションで別々のデータ・ファイル・セットに対し RECOVER DATAFILE コマンドを発行することによって手動で実行できる。ただし、この方法を使用すると、各 Oracle Enterprise Manager セッションが REDO ログ・ファイル全体を読み込むことになります。
- Recovery Manager の RESTORE および RECOVER コマンドを使用すると、回復処理のすべての段階を自動的にパラレル化できる。Oracle は、1 つのプロセスを使用してログ・ファイルを順次読み込み、REDO 情報を複数の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複数のセッションを使用する必要はありません。また、自動パラレル回復用に設定する初期化パラメータもあります。詳細は、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』を参照してください。
- SQL*Plus の RECOVER コマンドを使用してパラレル回復を実行できる。詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。
- SQL コマンド ALTER DATABASE RECOVER を使用して、パラレル回復を実行できる。ただし、この方法はお薦めしません。

パラレル回復を活用できる状況

一般に、パラレル回復は、複数の異なるディスク上にあるいくつかのデータ・ファイルを同時実行で回復する際の回復時間を短縮するうえで最も効果的です。多数の異なるディスク・ドライブにある大量のデータ・ファイルのクラッシュ回復（インスタンス障害が発生した後に実行する回復）とメディア回復を行うのは、パラレル回復のよい候補です。

パラレル回復によってパフォーマンスが向上するかどうかは、オペレーティング・システムで非同期 I/O がサポートされているかどうかにも依存します。非同期 I/O がサポートされて いない場合にパラレル回復を使用すると、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、パラレル回復を使用しても回復時間はわずかに短縮されるだけです。

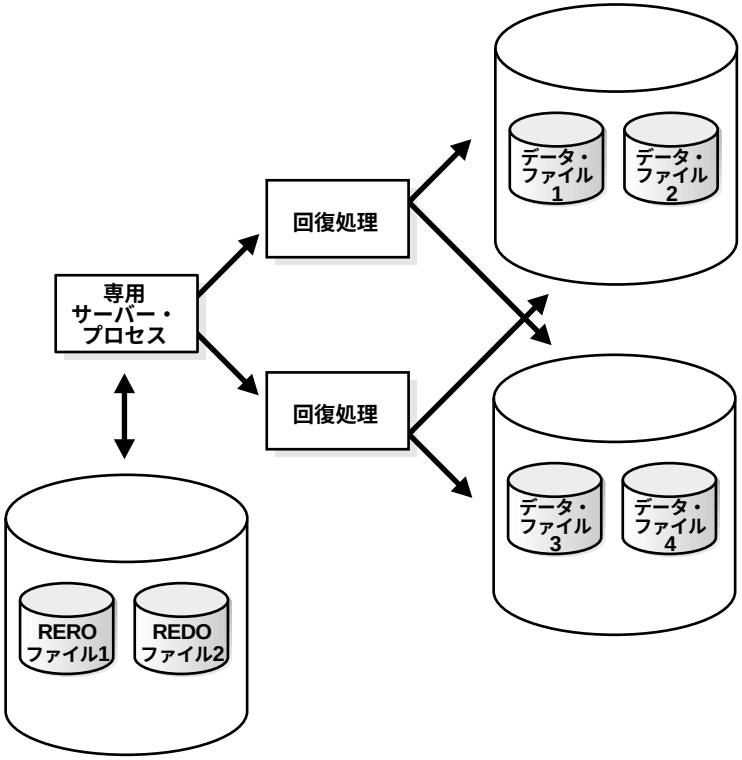
追加情報： システムで非同期 I/O がサポートされているかどうかを判別するには、使用しているオペレーティング・システムのマニュアルを参照してください。

回復プロセス

典型的なパラレル回復の状況では、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 32-2 に、典型的なパラレル回復セッションを示します。

図 32-2 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であるため、必要な回復プロセスの数は、回復で使用するディスク・ドライブの数によって異なります。

一般に、パラレル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。

ファースト・スタート・リカバリ

ファースト・スタート・リカバリは、ロールフォワードの所要時間を短縮し、回復を有界で予測可能なものにするアーキテクチャです。また、システム障害のために異常終了したトランザクションの場合は、回復時のロールバック時間が不要になります。ファースト・スタート・リカバリの内容は、次のとおりです。

- ファースト・スタート・チェックポイント
- ファースト・スタート・オン・デマンド・ロールバック
- ファースト・スタート・パラレル・ロールバック

ファースト・スタート・チェックポイント

ファースト・スタート・チェックポイントは、REDO スレッド（ログ）内でクラッシュ回復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。各 DBW*n* プロセスは、絶えずバッファをディスクに書き込んでチェックポイント位置を先に進めます。このため、通常の処理中には、オーバーヘッドは生じないか、最小限度ですみます。ファースト・スタート・チェックポイント機能により、クラッシュ回復とインスタンス回復のパフォーマンスは向上しますが、メディア回復のパフォーマンスは向上しません。

クラッシュ回復やインスタンス回復の所要時間が厳しく制限される状況では、回復処理のパフォーマンスを改善できます。クラッシュ回復やインスタンス回復の所要時間は、ロールフォワード・フェーズで読込みまたは書込みを必要とするデータ・ブロック数にほぼ比例します。ロールフォワード中に処理を必要とするデータ・ブロック数について、制限つまり上限を指定できます。Oracle サーバーは、指定されたロールフォワード上限に合わせてチェックポイント書込み率を自動的に調整し、書込み数を最小限度に抑えます。

動的初期化パラメータ FAST_START_IO_TARGET を設定すると、クラッシュ回復またはインスタンス回復時に読込みを必要とするブロック数を制限できます。このパラメータの値が大きいほど、多数のバッファが書き込まれる必要があるので、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、回復する必要の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。また、動的初期化パラメータ LOG_CHECKPOINT_INTERVAL および LOG_CHECKPOINT_TIMEOUT も、ファースト・スタート・チェックポイントに影響します。

追加情報： FAST_START_IO_TARGET 値の設定方法の詳細は『Oracle8*i* チューニング』、チェックポイントの詳細は『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

ファースト・スタート・オン・デマンド・ロールバック

デッド・トランザクションが他のトランザクションに必要な行ロックを保持している場合、「ファースト・スタート・オン・デマンド・ロールバック」は、問題のデータ・ブロックのみを即時に回復し、デッド・トランザクションの残りの部分についてはバックグラウンドで回復処理が実行されます。これにより、大きいデッド・トランザクションによってロックされているデータにアクセスするユーザーにとっては、データベースの可用性が改善されます。ファースト・スタート・ロールバックが使用可能になっていない場合、ユーザーはデッド・トランザクション全体が回復されるまで待たなければ、行ロックを取得できません。

ファースト・スタート・パラレル・ロールバック

ファースト・スタート・パラレル・ロールバックにより、サーバー・プロセスのグループを使用してトランザクションの集合をパラレルに回復できます。この手法が使用されるのは、パラレル回復の所要時間が、シリアルな回復の所要時間より短いと SMON が判断した場合です。

透過的アプリケーション・フェイルオーバーによって障害を隠す方法

迅速な回復処理では、データの使用不能期間は最短になりますが、ユーザー・セッションの障害による中断は処理されません。ユーザーは、データベースへの接続を再確立する必要があり、進行中だった作業は失われる可能性があります。Oracle8*i* の透過的アプリケーション・フェイルオーバー（TAF）により、アプリケーションの状態を保ち、障害発生時に進行中だった問合せを再開して、多数の障害をユーザーから隠すことができます。開発者は、TAF を活用するアプリケーションを構築し、トランザクションに影響するものを含めて全障害をユーザーに透過的なものにして、これらの機能を拡張できます。

追加情報： 透過的アプリケーション・フェイルオーバーの詳細は、『Oracle8*i* チューニング』を参照してください。

Recovery Manager

Recovery Manager は、すべてのデータベース・ファイル（データ・ファイル、制御ファイルおよびアーカイブ済み REDO ログ・ファイル）のバックアップを作成するプロセスと、バックアップからファイルを復元または回復するプロセスを管理する Oracle ユーティリティです。

追加情報： Recovery Manager の詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

リカバリ・カタログ

Recovery Manager は、「リカバリ・カタログ」というリポジトリを維持しています。リカバリ・カタログには、バックアップ・ファイルとアーカイブ済みログ・ファイルに関する情報

が含まれています。Recovery Manager は、このリカバリ・カタログを使用して、復元操作とメディア回復の両方を自動化します。

リカバリ・カタログには、次の情報が含まれています。

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

リカバリ・カタログをメンテナンスしているのは、Recovery Manager だけです。バックアップ済みデータベースのデータベース・サーバーが、リカバリ・カタログに直接アクセスすることはありません。Recovery Manager は、バックアップ・データ・ファイルの集合、アーカイブ済み REDO ログ、バックアップ制御ファイルおよびデータ・ファイルのコピーに関する情報を、長期保存のためにリカバリ・カタログに伝播します。

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成します。リカバリ・カタログは Oracle データベースに格納されるので、Recovery Manager を使用してリカバリ・カタログのバックアップを作成できます。

リカバリ・カタログが破棄されて使用可能なバックアップがない場合は、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・カタログを使用しない操作

リカバリ・カタログの使用は必須ではありませんが、使用することをお薦めします。リカバリ・カタログのほとんどの情報は制御ファイルからも入手できるので、Recovery Manager では制御ファイルのみを使用する操作モードをサポートしています。この操作モードは、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。

Recovery Manager には、リカバリ・カタログを使用する場合にのみ使用可能な機能があります。

追加情報： リカバリ・カタログの作成方法と、リカバリ・カタログの使用を必要とする Recovery Manager の機能の詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

パラレル化

Recovery Manager では、非ブロック化 UPI を使用して複数のログオン・セッションを確立し、複数の操作を同時に実行することにより、操作をパラレル化できます。同時実行される操作は、データ・ファイルの別々のセットを処理するものである必要があります。

注意： Oracle8*i* Enterprise Edition を使用すると、並行性が無制限になります。Oracle8*i* では、Recovery Manager チャネルを一度に 1 つしか割当てできないので、1 つのストリームに対する並行性は制限されます。

backup、**copy** および **restore** の各コマンドのパラレル化は、Recovery Manager で内部処理されます。必要なのは、次の情報を指定することだけです。

- 1 つ以上の順次 I/O デバイスのリスト
- バックアップ、コピーまたはリストアするオブジェクト

Recovery Manager はコマンドを順次実行します。つまり、前のコマンドを完了してから次のコマンドを開始します。並列性が活用されるのは、1 つのコマンドのコンテキスト内に限られます。したがって、10 個のデータ・ファイル・コピーが必要な場合は、**copy** コマンドを 10 回発行するのではなく、10 回のコピーをすべて指定した **copy** コマンドを 1 回発行してください。

レポートの生成

report および **list** コマンドを使用すると、バックアップとイメージ・コピーに関する情報が得られます。これらのコマンドの出力は、メッセージ・ログ・ファイルに書き込まれます。

report コマンドによって生成されるレポートは、次のような質問に対する答えを提供します。

- バックアップが必要なファイル
- しばらくバックアップを作成していなかったファイル
- 削除できるバックアップ・ファイル

report need backup コマンドと **report unrecoverable** コマンドを定期的に使用することによって、回復の実行に必要なバックアップを使用可能にし、回復が妥当な時間内で実行されるよう指定できます。**report deletable** コマンドは、冗長であるため、または **recover** コマンドでは使用できなくなったために削除可能となった、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・ファイルに格納されているスキーマ・オブジェクトに対して、ログに記録されていない操作が実行された場合、そのデータ・ファイルは「回復不能」と見なされます。

（バックアップがないデータ・ファイルは、回復不能とは見なされません。そのファイルを作成した時点から記録が開始されたログがまだ残っていれば、**create datafile** コマンドを使用して、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。このコマンドを使用すると、どのバックアップまたはコピーが使用可能であるかがわかります。

- 指定したデータ・ファイル（複数可）のバックアップまたはコピー
- 指定した表領域（複数可）のメンバーであるデータ・ファイルのバックアップまたはコピー
- 指定した名前または指定した範囲内（あるいはその両方）のアーカイブ・ログのバックアップまたはコピー
- 指定したデータベースそのもの

データベースのアーカイブ・モード

データベースは、NOARCHIVELOG モード（メディア回復が使用禁止）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禁止）

データベースを NOARCHIVELOG モードで使用する場合、オンライン REDO ログのアーカイブは使用禁止になります。データベースの制御ファイルの情報は、満杯のグループをアーカイブする必要はないことを示します。したがって、満杯のグループが非アクティブになると、そのグループは即時に LGWR プロセスが再使用できるようになります。

NOARCHIVELOG モードでデータベースを保護できるのは、インスタンス障害が発生した場合だけで、ディスク（メディア）障害の場合は保護できません。クラッシュ回復またはインスタンス回復に使用できるのは、データベースへの最新の変更（オンライン REDO ログのグループに格納されている）のみで、それ以外の情報は使用できません。Oracle は、変更がデータ・ファイルに安全に記録されるまで、必要かもしれないオンライン REDO ログ・ファイルを上書きしないので、クラッシュ回復とインスタンス回復のニーズを満たすにはこの情報があれば十分です。ただし、メディア回復は実行できません。

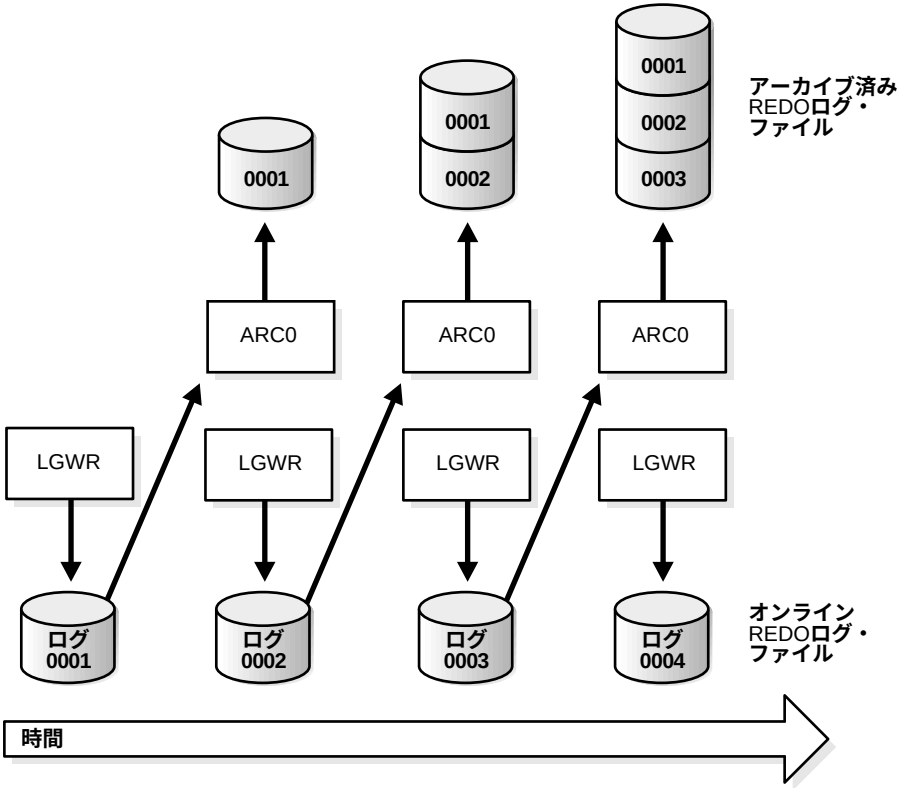
ARCHIVELOG モード（メディア回復が使用可能）

データベースを ARCHIVELOG モードで使用する場合、オンライン REDO ログのアーカイブは使用可能になります。データベースの制御ファイルの情報は、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を示します。

図 32-3 に、データベースのオンライン REDO ログ・ファイルが ARCHIVELOG モードでどのように使用されるか、また、満杯のグループをアーカイブするプロセス（この図では ARC0）によってアーカイブ済み 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 32-3 ARCHIVELOG モードでのオンライン REDO ログ・ファイルの使用



自動アーカイブと ARC*n*（アーカイバ）バックグラウンド・プロセス

インスタンスの構成にバックグラウンド・プロセス ARC0（アーカイバ）を追加すると、オンライン REDO ログ・ファイルのグループがアクティブでなくなった場合に、そのプロセスが自動的にこれらのグループをアーカイブできます。自動アーカイブを使用すると、データベース管理者は、満杯のグループを追跡して記録し、アーカイブする作業を手動で実行する必要がなくなります。ARCHIVELOG モードで稼働するほとんどのデータベース・システムで自動アーカイブを使用しているのは、このような便宜上の理由のためです。データの大量ロードなど、作業負荷が大きい場合は、初期化パラメータ LOG_ARCHIVE_MAX_PROCESSES を設定して、複数のアーカイバ・プロセス（ARC9 まで）を構成できます。

LOG_ARCHIVE_START 初期化パラメータを設定して、インスタンス起動時に自動アーカイブを要求すると、Oracle は LOG_ARCHIVE_MAX_PROCESSES で指定された数の ARC*n* プロセスをインスタンス起動時に開始します。それ以外の場合、インスタンス起動時には、ARC*n* プロセスは開始されません。

ただし、データベース管理者は、自動アーカイブをいつでも対話形式で開始したり停止できます。インスタンス起動時に自動アーカイブを指定しないで、後からデータベース管理者が自動アーカイブを開始すると、ARC*n* バックグラウンド・プロセスが作成されます。ARC*n* は、インスタンスの存続期間中は、自動アーカイブが一時的にオフになり、再度オンになっても残っています。ただし、ALTER SYSTEM コマンドで LOG_ARCHIVE_MAX_PROCESSES を設定すると、ARC*n* プロセスの数を動的に変更できます。

ARC*n* は、常に最小の順序番号から順にグループをアーカイブします。また、ARC*n* は、満杯のグループがアクティブでなくなると、そのグループを自動的にアーカイブします。すべての自動アーカイブの記録が、ARC*n* プロセスによって ARC*n* 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。

ARC*n* があるグループをアーカイブするときにエラーを検出しても（アーカイブ先が無効または満杯の場合など）、ARC*n* は引き続きそのグループをアーカイブます。また、エラーは ARC*n* トレース・ファイルと ALERT ファイルにも書き込まれます。問題が解決されない場合、最終的にはすべてのオンライン REDO ログ・グループが満杯になってもアーカイブされていないと、LGWR が使用できるグループがなくなるため、システムは停止します。したがって、問題が検出された場合は、問題を解決して ARC*n* がアーカイブを続行できるようにするか（アーカイブ先の変更など）、または問題が解決されるまで手動でグループをアーカイブしてください。

手動アーカイブ

データベースが ARCHIVELOG モードで動作している場合、自動アーカイブが使用可能かどうかには関係なく、必要に応じてアクティブでないオンライン REDO ログ・ファイルのグループのうち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手動でアーカイブする必要があります。

ほとんどのシステムでは、グループが非アクティブになってアーカイブ可能になったかどうかをデータベース管理者が監視する必要はないようにするため、自動アーカイブを使用します。さらに、自動アーカイブを使用禁止にした場合に、手動アーカイブを十分な速度で実行しなければ、アクティブでないグループを再使用できるようになるまで LGWR が強制的に待機状態に入り、データベース操作が一時的に停止することがあります。

手動アーカイブ・オプションは、データベース管理者が次の操作を実行するために提供されています。

- 問題が発生して（アーカイブ済み 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バージョンが消失したり、破損した場合にグループを再アーカイブする。

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 ARC*n* バックグラウンド・プロセスが存在していても、オンライン REDO ログ・ファイル・グループのアーカイブは、ユーザー・プロセスが実行します。

制御ファイル

データベースの制御ファイルは、データベースの正常な起動と操作に必要な小さいバイナリ・ファイルです。制御ファイルはデータベースの使用時に Oracle によって絶えず更新されるので、データベースがオープンされている間は書込み可能になっている必要があります。なんらかの理由で制御ファイルにアクセスできない場合、データベースは正常に機能しなくなります。

それぞれの制御ファイルは、1 つの Oracle データベースにのみ対応付けられます。

制御ファイルの内容

制御ファイルには、インスタンスからアクセスするデータベースが起動時と通常の操作時に必要とする、関連データベースの情報が格納されています。制御ファイルの情報を変更できるのは、Oracle だけです。データベース管理者やエンド・ユーザーは、データベースの制御ファイルを編集できません。

制御ファイルには、次のような情報が格納されています。

- データベース名
- データベースを作成したときのタイムスタンプ
- 対応するデータ・ファイルとオンライン REDO ログ・ファイルの名前と位置

- 表領域情報
- データ・ファイルのオフライン範囲
- ログ履歴
- アーカイブ・ログ情報
- バックアップ・セットとバックアップ部分の情報
- バックアップ・データ・ファイルと REDO ログ情報
- データ・ファイル・コピー情報
- 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成されます。データベース名には、DB_NAME 初期化パラメータに指定した名前、または CREATE DATABASE 文で使用した名前が使用されます。

データベースのデータ・ファイルやオンライン REDO ログ・ファイルが追加、改名または削除されるたびに、制御ファイルが更新され、この物理構造の変化が反映されます。これらの変更は、次の目的のために記録されます。

- データベースの起動時に、オープンするデータ・ファイルとオンライン REDO ログ・ファイルを Oracle が識別できるようにする。
- データベースの回復が必要な場合に必要なファイルや使用可能なファイルを Oracle が識別できるようにする。

したがって、データベースの物理構造を変更した後は、ただちに制御ファイルのバックアップを作成してください。

> **追加情報：** データベースの制御ファイルのバックアップを作成する方法は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

制御ファイルには、チェックポイントに関する情報も記録されます。チェックポイント・プロセス（CKPT）は、オンライン REDO ログ内のチェックポイント位置に関する情報を、制御ファイルに 3 秒ごとに記録します。この情報は、データベースの回復時に使用され、オンライン REDO ログ・グループの特定のポイントより前に記録されている REDO エントリはデータベースの回復には不要である（すでにデータ・ファイルに書き込まれている）ことを Oracle に伝えます。

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。ただし、制御ファイルのコピーを他のディスクに保存してあれば、データベースを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が永続的に消失した（複数のディスクで障害が発生した）場合は、インスタンスが異常終了し、メディア回復が必要になります。ただし、現行の制御ファイルを使用できず、制御ファイルの古いバックアップを使用するのであれば、メディア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成し、それぞれのコピーを異なる物理ディスクに格納してください。

データベースのバックアップ

Oracle ミラー化ログ、Oracle ミラー化制御ファイルおよびアーカイブ・ログを使用すると、メディア障害から回復できますが、回復処理中は一部またはすべてのデータが使用不能になることがあります。回復レベルを上げるために、少なくともデータ・ファイルと制御ファイルには、オペレーティング・システムまたはハードウェアのデータ冗長性を使用することをお薦めします。これにより、システムは完全に使用可能な状態で、メディア障害のいずれか 1 つが回復可能になることが保証されます。

Oracle データベースについてどのようなバックアップおよびリカバリの計画を考案するとしても、データベースのデータ・ファイルと制御ファイルのバックアップを作成しておくことは、これらのファイルを破損しかねないメディア障害の可能性に備えた保護方針の一環として絶対に必要です。以降の項では、使用可能な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： 詳細およびデータベース・バックアップ実行のガイドラインは、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

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用すると、そのバックアップの時点まで回復できます。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（オンラインおよびアーカイブ済みの REDO ログ・ファイルを使用する）してからでなければ、データベースは使用可能になりません。

バックアップとアーカイブ・モード

全体バックアップによって取得されたデータ・ファイルは、どのタイプのメディア回復の方式にも有効です。

- データベースが NOARCHIVELOG モードで動作している場合に、ディスク障害が発生してデータベースを構成するファイルの一部またはすべてが破損した場合は、一貫性のある最新の全体バックアップを使用してデータベースを「復元」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 ログを使用できないので、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、この方法には依存しないでください。

- データベースが ARCHIVELOG モードで動作している場合に、ディスク障害が発生してデータベースを構成するファイルの一部またはすべてが破損した場合は、最新の全体バックアップで収集したデータ・ファイルを「データベース回復」の一部として使用できる。

  必要なデータ・ファイルを全体バックアップから復元した後は、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、復元したデータ・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用することによって、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元できます。

部分データベース・バックアップ

「部分データベース・バックアップ」とは、全体バックアップより小規模なバックアップのことで、データベースがオープンまたは停止しているときに実行します。部分データベース・バックアップには、次のものがあります。

- 個々の表領域のすべてのデータ・ファイルのバックアップ

- 1 つのデータ・ファイルのバックアップ
- 制御ファイルのバックアップ

部分バックアップが有効なのは、ARCHIVELOG モードでデータベースを実行している場合だけです。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元されたデータ・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確保できます。

データ・ファイルのバックアップ

部分バックアップには、データベースの一部のデータ・ファイルのみを含めることができます。個々の特定のデータ・ファイルまたはその集合のバックアップは、データベースの他のデータ・ファイル、オンライン REDO ログ・ファイルおよび制御ファイルのバックアップとは別個に作成できます。データ・ファイルのバックアップは、オフラインでもオンラインでも作成できます。

オンラインとオフラインのどちらのデータ・ファイル・バックアップを作成するかは、データ可用性の要件のみに依存しています。バックアップの対象となるデータを常に使用可能な状態にしておく必要がある場合は、オンラインのデータ・ファイル・バックアップが唯一の選択肢です。

制御ファイルのバックアップ

部分バックアップの別の形態は、制御ファイルのバックアップです。制御ファイルは、対応付けられたデータベースの物理ファイル構造を追跡し記録するので、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成する必要があります。

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のバックアップを作成します。データ・ファイル 1 にはデータ・ディクショナリが含まれています。

制御ファイルを多重化しておくと、単一の制御ファイルが失われた場合の保護対策になります。ただし、ディスク障害によってデータ・ファイルが破損し、部分的な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用する必要があります。このため、制御ファイルを多重化してあるとしても、データベースの構造が変更されるたびに作成する制御ファイル・バックアップの代替策にはなりません。

不完全な回復または Point-in-Time 回復の前に Recovery Manager を使用し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元します。

Export および Import ユーティリティ

Export および Import は、Oracle のデータを Oracle データベースから外部へ、または外部から Oracle データベース内部へ移動するために使用するユーティリティです。Export は、Oracle データベースのデータをオペレーティング・システムのファイルに、Oracle データベース・フォーマットで書き出すユーティリティです。エクスポートされるファイルには、データベースに対して作成されたスキーマ・オブジェクトに関する情報が格納されます。Import は、エクスポートされたファイルを読み込んで、対応する情報を既存のデータベースに復旧するユーティリティです。Export と Import は、Oracle データの移動を目的として設計されていますが、Oracle データベースのデータを保護する補助的な方法としても利用できます。

追加情報：『Oracle8*i* ユーティリティ・ガイド』を参照してください。

読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、読込み専用表領域についてはバックアップを作成できます。表領域を読込み専用にした後は、その表領域のバックアップをただちに作成してください。表領域が読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成する必要はありません。

読込み専用表領域を読み書き可能表領域に変更した場合、その表領域の通常のバックアップを再開する必要があります。これはオフラインの読み書き可能表領域をオンラインに戻した時点でバックアップを再開するのと同様です。

読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、ハードウェア障害またはシステムを中断させるその他の障害に備えて、Oracle には「自動化されたスタンバイ・データベース」機能が用意されています。スタンバイ・データベースは、耐障害性と災害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。（もう 1 つの選択肢は、データベース・レプリケーションを使用することです。この機能の説明は第 34 章の「データベースのレプリケーション」を参照してください。）

障害回復の計画

システム障害やその他の災害から迅速かつ確実に回復するには、綿密な計画を練る以外に方法はありません。詳しい手順を記述した定型的な計画を作成しておく必要があります。スタンバイ・データベース・システムをインプリメントするにしても、単一データベース・システムを使用するにしても、突然の障害に備えてどのように対処するかを計画しておく必要があります。

自動化されたスタンバイ・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障害回復を容易にする、信頼性の高い支援メカニズムが提供されています。このメカニズムを、「自動化されたスタンバイ・データベース」といいます。プライマリ・サイトにアーカイブされているログ・ファイルの自動出荷依頼を通じて、最大 4 つのスタンバイ・システムを一貫したメディア回復状態に保つことができます。1 次システムで障害が発生した場合は、スタンバイ・システムの 1 つをアクティブ化できるので、システムはすぐに使用可能になります。Oracle には、スタンバイ・システムの作成とメンテナンスに伴う操作のためのコマンドと内部確認機能が用意されており、それによって障害回復スキーム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用されます。プライマリ・データベースで REDO ログがアーカイブされると、必ずそのログがリモート・サイトに転送され、スタンバイ・データベースに適用されます。したがって、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・データベースよりログ 1 つまたは 2 つ分だけ遅れます。

自動化されたスタンバイ・データベースによって、電源障害などの長期の停止期間や、火災、洪水または地震などの自然災害からデータが保護されます。スタンバイ・データベースは障害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・データベースは、読込み専用でオープンできます。これにより、データベースをレポート作成に使用できます。スタンバイ・データベースを読込み専用でオープンすると、REDO ログはキューに入れられ、適用されません。データベースがスタンバイ・モードに戻ると、キューに入れられたログと新しいアーカイブ・ログがただちに適用されます。

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*i* バックアップおよびリカバリ・ガイド』を参照してください。

第 IX 部

分散データベースとレプリケーション

第 IX では、分散データベース・アーキテクチャとネットワーク上でのデータ・レプリケーションについて説明します。

第 IX 部には、次の章が含まれています。

33

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
- 異機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 33-1 に、典型的な Oracle 分散データベース・システムを示します。

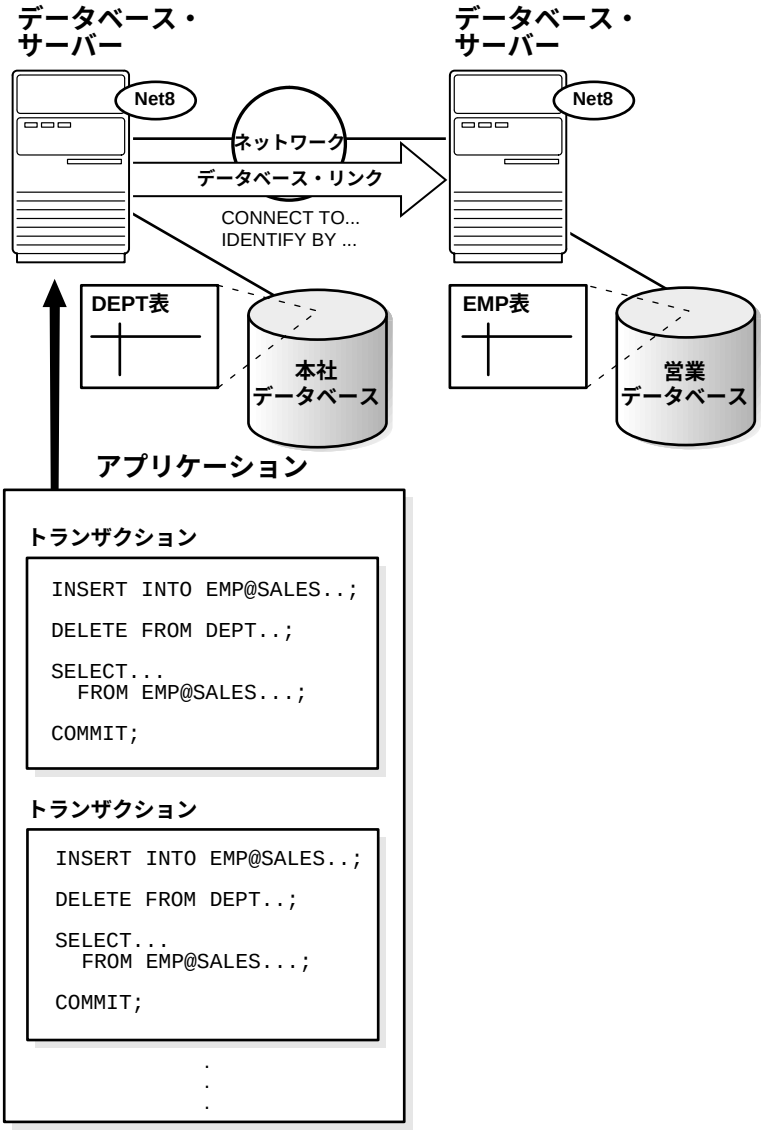
クライアントとサーバー

「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」とはサーバーに対して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 33-1 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 33-1 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 33-1 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。以降の各項では、Oracle 分散データベース・システムのうちネットワーク関連の事項についてさらに詳しく説明します。

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、ネットワークを介したデータベース間通信に Oracle のネットワーキング・ソフトウェアである Net8 を使用します。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、それと同様にして、複数のネットワークにまたがる各データベース・サーバーが相互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・トランザクションと分散トランザクションをサポートします。

Net8 によって、システムを使用するアプリケーションでは、SQL 要求の送信やデータの受信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、『Oracle8*i* Net8 管理者ガイド』を参照してください。

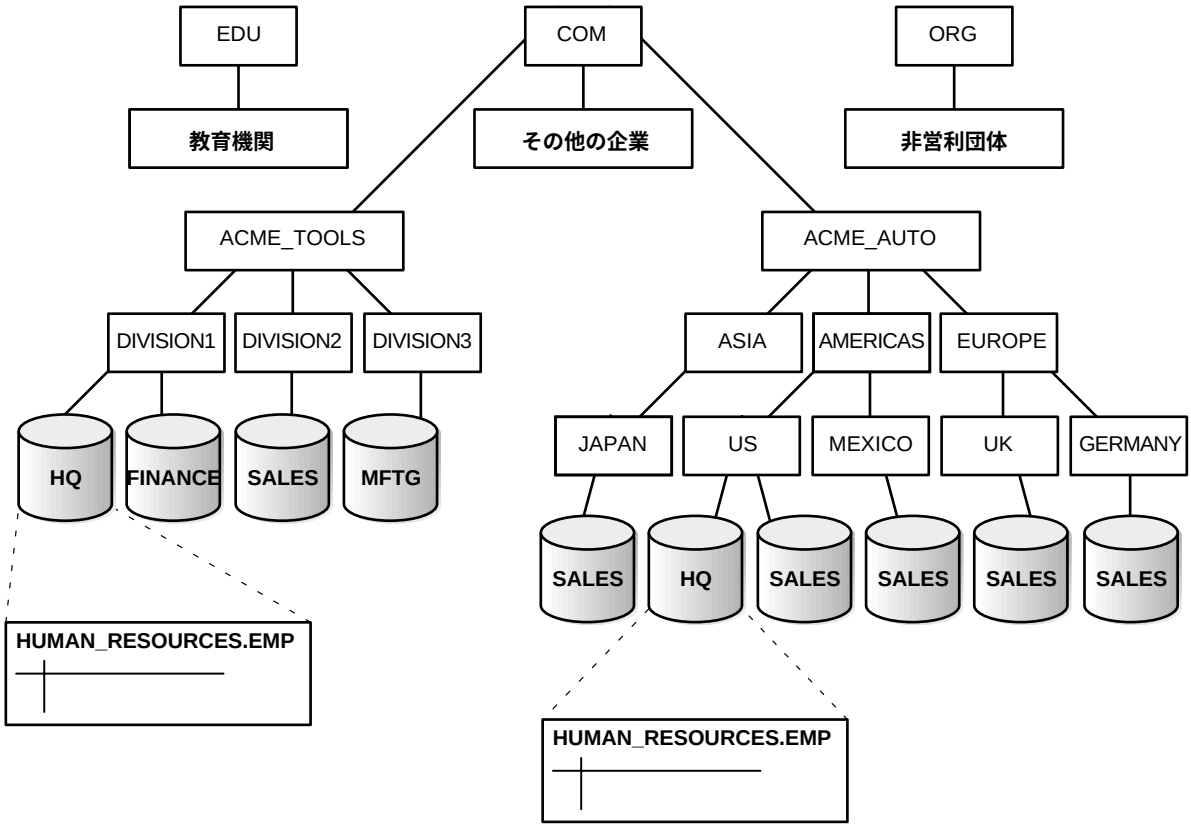
Oracle Names

Oracle ネットワークでは、オプションとして Oracle Names を使用することによって、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場合は、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 サーバーを使用することによって、分散データベース・アクセスの構成が容易になります。

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 が作成す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 33-2 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 33-2　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にする必要があります。たとえば、ネットワーク・ドメイン US.AMERICAS.ACME_AUTO.COM と UK.EUROPE.ACME_AUTO.COM には、それぞれ SALES データベースが存在します。

```
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」を使用します。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すデータベースのグローバル名と同じであるため、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・リンクは本質的に透過的です。

たとえば、次の SQL 文では、ローカル・データベース内に、リモート・データベース SALES.US.AMERICAS.ACME_AUTO.COM へのパスを記述するデータベース・リンクが作成されます。

```
CREATE DATABASE LINK sales.us.americas.acme_auto.com ... ;
```

データベース・リンクの作成後、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。次の項では、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・リンクを使用する例を示します。

> **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*i* 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用し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。また、Oracle では、オブジェクトのローカル名へのアプリケーション参照を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」を作成し、分散データベース・システム内のスキーマ・オブジェクトへの参照を解決します。たとえば、問合せでは、表が存在するデータベースを含めた完全修飾名を指定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に、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・リンクを暗黙的に使用します。

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、分散データベース・システムに加えることができます。ただし、分散データベース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理解している必要があります。

たとえば、分散データベース・アプリケーションでは、Oracle8*i* 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、意味には明確な違いがあります。

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 に分散し、バックエンドのデータベース・サーバーがデータベースへの共有アクセスを管理するようにします。そのため、分散データベース・アプリケーション処理システムは、「クライアント / サーバー」データベース・アプリケーション・システムとも呼ばれます。

Oracle 分散データベース・システムでは、分散処理アーキテクチャを採用しています。たとえば、Oracle Server は、他の Oracle Server が管理するデータを要求するときにはクライアントとして動作します。

分散データベースとデータベース・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、両者は異なっています。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・オブジェクトのコピーを 1 つ管理します。多くの分散データベース・アプリケーションは、分散トランザクションを使用する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 34 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、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・テクノロジに依存していますが、データベース・レプリケーションを使用するアプリケーションには、純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、パフォーマンスの改善やアプリケーションの可用性の保護に役立ちます。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、ネットワーク通信量を最小化して最大のパフォーマンスを達成できる可能性があります。しかもそのアプリケーションは、ローカル・サーバーに障害が発生した場合にも、レプリケートされたデータが存在する他のサーバーがアクセス可能である限り継続できます。

追加情報： Oracle のレプリケーション機能の詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。

異機種間分散データベース

Oracle の「異機種間分散データベース・システム」では、データベース・システムのうち最低 1 つは非 Oracle システムです。アプリケーションからは、異機種間分散データベース・システムは、1 つのローカル Oracle データベースに見えます。ローカル Oracle サーバーは、データの分散と異機種性を隠すことができます。Oracle サーバーは、非 Oracle システムにアクセスするために、Oracle8*i* 異機種間サービスと非 Oracle システム固有の異機種間サービス・エージェントを使用します。

異機種間サービス

異機種間サービスは、Oracle8*i* サーバー内の統合されたコンポーネントであり、Oracle の次世代 Open Gateway 製品を使用可能にするテクノロジです。異機種間サービスは、将来の Oracle Gateway 製品や他の異機種間アクセス機能のための共通アーキテクチャと管理メカニズムを提供するだけでなく、Oracle Open Gateway の旧リリースのユーザー向けに上位互換性機能を提供します。

異機種間サービス・エージェント

アクセスしたい非 Oracle システムごとに、異機種間サービスでは、特定の非 Oracle システムへアクセスするための「エージェント」が必要です。異機種間サービス・エージェントは、非 Oracle システム、および Oracle サーバーの異機種間サービス・コンポーネントと通信します。エージェントは、Oracle サーバーにかわって SQL、プロシージャおよびトランザクション要求を非 Oracle システム側で実行します。

バージョン 8 の Oracle Gateway は、非 Oracle システムにプロシージャまたは SQL を使用してアクセスする、異機種間サービス・エージェントの Oracle 製品名です。ただし、異機種間サービス・エージェントは、Oracle Transparent Gateways や Oracle Procedural Gateways 以外の製品として使用可能になります。このマニュアルでは、より一般的な用語である「異機種間サービス・エージェント」を使用します。Oracle Open Gateway バージョン 8 を購入した場合は、"Oracle Gateway version 8" という用語を「異機種間サービス・エージェント」と置き換えてください。

バージョン 8 ゲートウェイのインストールと構成の詳細は、『Oracle Open Gateway Installation and User's Guide』を参照してください。

機能

異機種間サービスの機能は、次のとおりです。

- 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 SQL アクセス。非 Oracle システムからのデータを、1 つのローカル・データベースに格納されているかのようにして Oracle 環境に統合します。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。
- プロシージャ型アクセス。メッセージング・システムやキューイング・システムなどのプロシージャ型システムには、Oracle8*i* サーバーから PL/SQL のリモート・プロシージャ・コールを使用してアクセスします。
- 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に、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 方言を使用して、Oracle アプリケーションから非 Oracle システムに直接アクセスできます。
- ストアド・プロシージャへのアクセス。SQL ベースの非 Oracle システムに含まれているストアド・プロシージャは、PL/SQL リモート・プロシージャであるかのようにアクセスされます。
- 各国語サポート。異機種間サービスはマルチバイト・キャラクタセットをサポートしており、非 Oracle システムと Oracle8*i* サーバー間でキャラクタ・セットを変換します。
- マルチスレッド・エージェント。マルチスレッド・エージェントは、オペレーティング・システムのスレッド化機能を使用します。マルチスレッド・エージェントにより、マルチスレッド・サーバー機能が活用され、必要なプロセス数が減少します。

- エージェント自動登録。エージェント自動登録により、リモート・ホスト上で異機種間サービスの構成データを更新するプロセスが自動化され、異機種間データベース・リンクを介した正常動作が保証されます。
- インタフェース管理。アクティブな異機種間サービス・エージェントと、各エージェントにアクセスしているユーザー・セッションのグラフィック表現を提供します。

注意： 前述の機能のすべてが異機種間サービス・エージェントや Oracle Gateway でサポートされているとは限りません。サポートしている機能の詳細は、異機種間サービス・エージェントまたは Oracle Open Gateway のマニュアル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。この後の項では、分散データベース内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。

分散問合せの最適化

「分散問合せの最適化」は、Oracle8*i* のデフォルト機能であり、分散 SQL 文で参照されるリモート表からデータを取り出す場合に、サイト間で必要となるデータ転送の量を低減します。

分散問合せの最適化では、Oracle のコストベース・オプティマイザを使用して、リモート表から必要なデータのみを抽出する SQL 式を検索または生成し、そのデータをリモート・サイトで処理し、結果を最終処理のためにローカル・サイトに送信します。これにより、すべての表データを処理のためにローカル・サイトに転送する場合に比べて、必要なデータ転送量が減少します。

DRIVING_SITE、NO_MERGE および INDEX ヒントなど、コストベースのオプティマイザ・ヒントを使用すると、Oracle でデータが処理される場所とデータへのアクセス方法をきめ細かく制御できます。

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;
```

注意： リモート更新には 1 つ以上のリモート・ノードからデータを取り出す副問合せが含まれることがありますが、更新操作は 1 つのリモート・ノードのみで発生するので、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・ユニットを使用して実行できます。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・ユニットが実行する必要がある作業や、結果をコール元のアプリケーションに戻す方法を理解した上でコーディングする必要があります。

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・ノードのみを参照する場合、そのトランザクションはリモート・トランザクションであって、分散トランザクションではありません。

2 フェーズ・コミット・メカニズム

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 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あることが保証される必要があります。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、更新またはリモート・プロシージャ・コールなど、あらゆるタイプの操作が含まれるトランザクションに当てはまります。

非分散データベースにおけるトランザクション制御の一般的なメカニズムの詳細は、第 17 章「トランザクションの管理」を参照してください。分散データベースでは、ネットワーク全体で同じ特性によってトランザクション制御を調整し、ネットワーク障害やシステム障害が発生した場合にもデータの一貫性が保たれる必要があります。

Oracle の「2 フェーズ・コミット・メカニズム」は、分散トランザクションに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のみになるように保証するものです。また、2 フェーズ・コミット・メカニズムは、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙の DML 操作も保護します。

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*i* 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

最小限の労力で、システムで作業するユーザーに対して、Oracle 分散データベース・システムの機能を透過的にすることができます。このような透過性の目標は、分散データベース・システムを単一の Oracle データベースであるかのように見せることです。その結果、システムの開発者およびユーザーは、分散データベース・アプリケーションの開発の難航やユーザーの生産性低下を招く原因になりかねない複雑な作業から解放されます。この後の各項では、分散データベース・システムでの透過性についてさらに説明します。

位置の透過性

Oracle 分散データベース・システムには、アプリケーションの開発者や管理者がデータベース・オブジェクトの物理的な位置をアプリケーションやユーザーから隠せるようにする機能があります。「位置の透過性」は、アプリケーションが接続しているノードに関係なく、表などのデータベース・オブジェクトをユーザーが広く参照できる場合に存在します。位置の透過性には、次のような利点があります。

- リモート・データへのアクセスが単純。データベース・ユーザーはデータベース・オブジェクトの物理的な位置を知る必要がないためです。
- 管理者は、エンドユーザーや既存のデータベース・アプリケーションに影響を与えることなくデータベース・オブジェクトを移動できる。

多くの場合、管理者と開発者は、シノニムを使用することによって、アプリケーション・スキーマ内の表およびサポートするオブジェクトについての位置の透過性を確立します。たとえば、次の文は、別のリモート・データベースにある表のシノニムをデータベース内に作成します。

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.acme_auto.com
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.acme_auto.com
```

こうすれば、次のような問合せによってリモート表にアクセスするかわりに、

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.acme_auto.com e,
       scott.dept@sales.us.americas.acme_auto.com d
  WHERE e.deptno = d.deptno;
```

アプリケーションは、次のように、リモート表の位置を指定する必要のない単純な問合せを発行できます。

```
SELECT ename, dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

シノニムに加えて、開発者はビューやストアド・プロシージャを使用して、分散データベース・システムで作業するアプリケーションの位置の透過性を確立することもできます。

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、更新およびトランザクションの透過性も提供されます。たとえば、SELECT、INSERT、UPDATE および DELETE などの標準的な SQL 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、SAVEPOINT および ROLLBACK によってトランザクションを制御します。分散トランザクションを制御するために複雑なプログラミングや他の特殊な操作を実行する必要はありません。

- 1 つのトランザクション内の文から、任意の数のローカル表またはリモート表を参照できる。
- 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、またはすべてロールバックされるかのどちらか一方になります。
- 分散トランザクションのコミット時にネットワーク障害またはシステム障害が発生すると、トランザクションは自動的、透過的かつグローバルに解決される。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用して接続が確立される場合。
- 分散 SQL 文が実行される場合。
- 分散トランザクションがコミットされる場合。

なによりも大きな利点は、分散データベース・システムのノード間の SCN の調整により、文レベルとトランザクション・レベルの両方で、グローバルな分散読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。Oracle のレプリケーション機能の詳細は、『Oracle8*i* レプリケーション・ガイド』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様に、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。以降の各項では、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動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、別々に管理され明確に区別される個々のデータ・リポジトリです。Oracle 分散データベースでのサイト自律性には、次のような利点があります。

- システムのノードは、企業の論理的な組織体系、または「互いに対等」な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担当ドメインが小さくなり、管理しやすくなります。
- 1 つのノードの障害によって分散データベースの他のノードを中断させる可能性が少ない。1 つのデータベースとネットワークが使用できる限りは、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・データへのアクセスには、グローバル・カタログは不要です。
- 各ノードで単独でソフトウェアをアップグレードできる。

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よいわけではありません。

たとえば、サーバー間接続用に作成するリンクをサポートするために、各データベースでユーザー・アカウントの追加が必要になることがあります。この後の各項では、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 / サーバー間接続とサーバー間の相互接続でのログイン・パケットの暗号化

この後の項では、Oracle 分散データベース・システムを構成する場合の追加の考慮事項について説明します。

ユーザー・アカウントとロールのサポート

分散データベース・システムでは、システムを使用するアプリケーションをサポートするのに必要なユーザー・アカウントとロールを慎重に計画する必要があります。

- サーバー間の相互接続を確立するために必要なユーザー・アカウントは、分散データベース・システム内のすべてのデータベースで使用可能である必要がある。
- アプリケーション権限を分散データベース・アプリケーション・ユーザーから使用できるようにするために必要なロールは、分散データベース・システム内のすべてのデータベースに存在している必要がある。

分散データベース・システム内のノードのためにデータベース・リンクを作成する場合、各サイトでそのリンクを使用するサーバー間の相互接続をサポートするには、どんなユーザー・アカウントとロールが必要かを決定してください。

> **追加情報：** システムで各種データベース・リンクをサポートするために必要なユーザー・アカウントの詳細は、『Oracle8*i* 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

通常、分散環境では、ユーザーは多数のネットワーク・サービスにアクセスする必要があります。各ユーザーが各ネットワーク・サービスにアクセスできるようにするための認証を個別に構成する必要があると（特に大規模システムの場合）、セキュリティ管理が煩雑になります。

グローバル認証サービスの使用は、分散環境のセキュリティ管理作業を簡素化するための一般的な手法です。

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 Manager。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。
- 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（DCE など）の枠内で機能させる必要がある場合、Oracle 分散データベース環境では Oracle の Advanced Security を使用できる。Oracle Advanced Security 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。詳細は、『Oracle8*i* Advanced Security 管理者ガイド』を参照してください。

データの暗号化

Oracle Advanced Security は、ネットワーク・データの暗号化とチェックサムを使用してデータの読込みや変更を禁止するという点でも、Net8 とその関連製品を拡張します。このオプションは、RSA Data Security RC4 またはデータ暗号化規格（DES）暗号化アルゴリズムを使用することによって、許可なしでは表示できないようにデータを保護します。

データが修正、削除または再実行されていないことを確認するため、Oracle Advanced Security のセキュリティ・サービスでは、暗号保護メッセージ・ダイジェストを生成し、それをネットワーク内で送信される各パケットに含めることができます。

> **追加情報：** これらの機能を含め、Oracle Advanced Security の各機能の詳細は、『Oracle8*i* Advanced Security 管理者ガイド』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に、データベース管理者は、次の複数の選択肢の中から使用するツールを選択できます。

- Enterprise Manager
- サードパーティの管理ツール
- SNMP のサポート

Enterprise Manager

Enterprise Manager は、Oracle のデータベース管理ツールです。Oracle Enterprise Manager のグラフィカル・コンポーネント（Enterprise Manager / GUI）を使用すると、グラフィカル・ユーザー・インタフェース（GUI）を利用してデータベース管理作業を実行できます。

Oracle Enterprise Manager の行モード・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用しやすいインタフェースが用意されています。Oracle Enterprise Manager を使用すると、次の作業を実行できます。

- データベースの起動、シャットダウン、バックアップおよびリカバリなど、従来の管理作業を実行する。これらの作業を実行する場合に、Oracle Enterprise Manager のグラフィカル・インタフェースを使用すると、SQL 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 を使用すると、単一のデータベースを管理したり、複数のデータベースを同時に管理できます。
- データベース管理作業を集中化する。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、PL / SQL および Oracle Enterprise Manager のコマンドを動的に実行する。Oracle Enterprise Manager を使用して、文を入力、編集および実行できます。Oracle Enterprise Manager は、実行した文の履歴のメンテナンスもします。

  このため、再び入力しなくても文を再実行できます。これは、分散データベース・システムで一連の長い文を繰り返し実行する必要がある場合に特に役立つ機能です。
- グラフィカル・ユーザー・インタフェースが使用できない場合や、使用しないほうがよい場合には、Oracle Enterprise Manager の行モード・インタフェースによって管理作業を実行する。

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 の検索や問合せができます。SNMP は、広く使用されている次のような多数のネットワーク管理システムの基礎として受け入れられている標準です。

- HP の OpenView
- Digital の POLYCENTER Manager on Net View
- 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で異なるキャラクタ・セットを使用するクライアント / サーバー環境がサポートされています。クライアントで使用されるキャラクタ・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。サーバーで使用されるキャラクタ・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 : 各国語サポートの詳細は、『Oracle8*i* NLS ガイド』を参照してください。

34

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 graces 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
- レプリケーションのオブジェクト、グループおよびサイト
- マルチマスター・レプリケーション
- スナップショット・レプリケーション
- マルチマスターおよびスナップショットのハイブリッド構成
- レプリケート環境の管理
- レプリケーションの競合
- 特殊レプリケーション・オプション

追加情報 : 『Oracle8*i* レプリケーション・ガイド』には、データベース・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。あるサイトで適用された変更は、ローカルに獲得されて格納されてから、各リモート・ロケーションに転送され、適用されます。レプリケーションにより、ユーザーは共有データに高速でローカル・アクセスでき、データ・アクセスに代替手段が存在するため、アプリケーションの可用性が保護されます。あるサイトが使用不能になっても、ユーザーは引き続き残りのサイトを問い合せて更新できます。

レプリケーションのオブジェクト、グループおよびサイト

この後の項では、レプリケーション・システムの基本的なコンポーネント（レプリケーション・サイト、レプリケーション・グループおよびレプリケーション・オブジェクトなど）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 のレプリケーション機能を使用すると、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、索引およびシノニムなど）をレプリケートできます。図 34-1 の `SCOTT.EMP` および `SCOTT.BONUS` は、レプリケーション・オブジェクトの一例です。

レプリケーション・グループ

レプリケーション環境の場合、Oracle は、「レプリケーション・グループ」を使用してレプリケーション・オブジェクトを管理します。レプリケーション・グループ内で相互に関連したデータベース・オブジェクトを編成すると、多数のオブジェクトを管理しやすくなります。多くの場合、特定のデータベース・アプリケーションのサポートに必要なスキーマ・オブジェクトを編成するために、レプリケーション・グループを作成して使用します。つまり、レプリケーション・グループとスキーマが相互に対応付けられている必要はありません。レプリケーション・グループ内のオブジェクを複数のデータベース・スキーマから選択し、スキーマには異なるレプリケーション・グループのメンバーであるオブジェクトを含めることができます。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なれません。

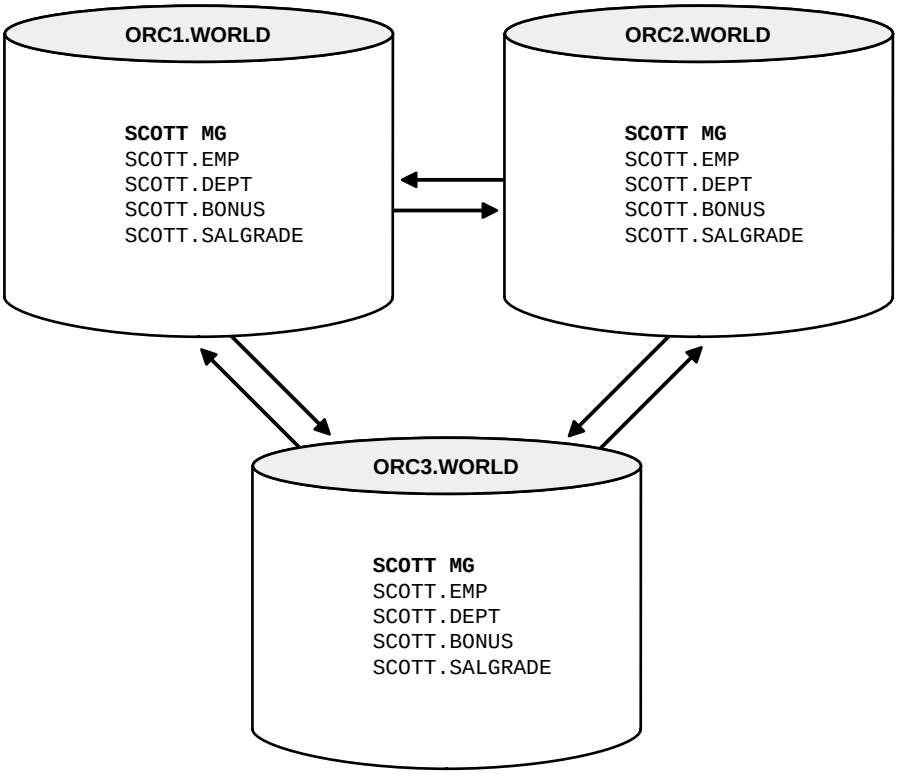
注意： 読込み専用スナップショットがスナップショット・グループに属する必要はなく、マスター・グループの一部であるマスター表に基づくものでなくてもかまいません。

この章では、基本またはアドバンスト・レプリケーションに使用するマテリアライズド・ビューを表すために、「マテリアライズド・ビュー」ではなく「スナップショット」という用語を使用しています。「スナップショット」という用語は廃止され、このマニュアルの将来のリリースでは「マテリアライズド・ビュー」に置き換えられます。SQL 文では、キーワード SNAPSHOT および MATERIALIZED VIEW は同じ意味です。

マテリアライズド・ビューの詳細は、10-17 ページの「マテリアライズド・ビュー」を参照してください。

マルチマスター・レプリケーション環境では、レプリケーション・グループを「マスター・グループ」と呼びます。各サイトの対応するマスター・グループには、同じレプリケーション・グループの集合を含める必要があります（34-2 ページの「レプリケーション・オブジェクト」を参照）。図 34-1 に、各マスター・サイトでレプリケートされたオブジェクトの正確なレプリカを含むマスター・グループ "SCOTT_MG" を示します。

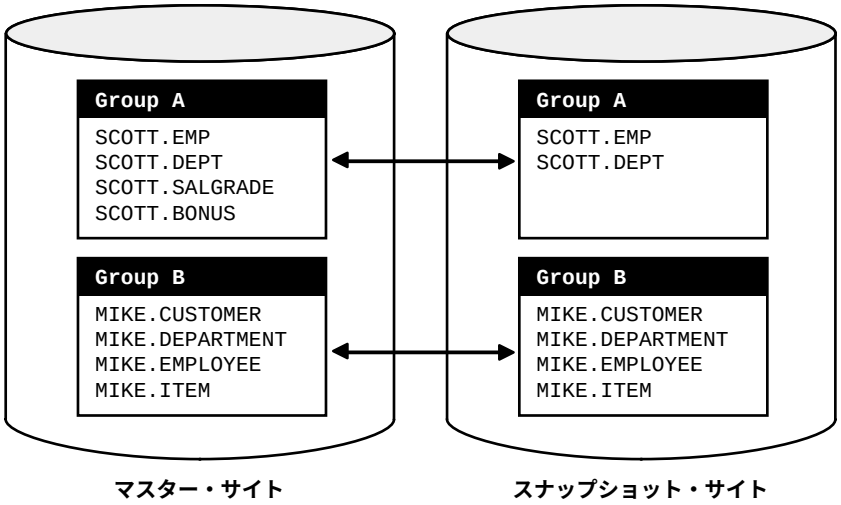
図 34-1 すべてのサイトに同じレプリケーション・オブジェクトを含むマスター・グループ SCOTT_MG



スナップショット・サイトでは、編成はスナップショット・グループを使用してメンテナンスされます。「スナップショット・グループ」により、ターゲット・マスター・グループにあるオブジェクトの部分的または完全なコピーがメンテナンスされます。図 34-2 では、スナップショット・サイトにあるスナップショット・グループ「グループ A」は、マスター・サイトにあるマスター・グループ「グループ A」の部分的なレプリカのみをメンテナンスし、「グループ B」のスナップショット・グループとマスター・グループは完全なレプリカをメンテナンスしています。

また、図 34-2 に、各サイトに複数のレプリケーション・グループを格納できることを示します。

図 34-2 マスター・グループに対応するスナップショット・グループ



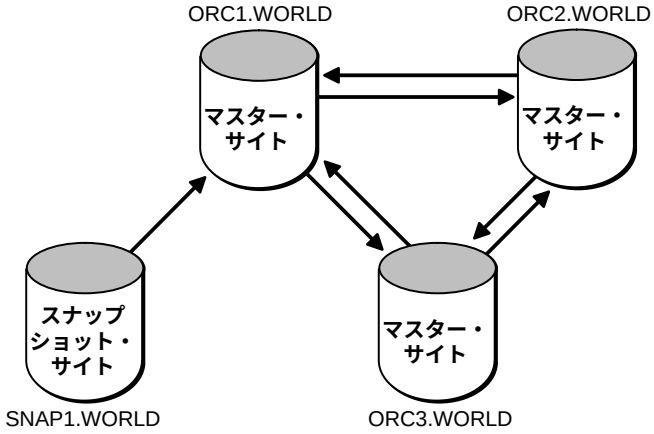
レプリケーション・サイト

レプリケーション・グループは、複数の「レプリケーション・サイト」に存在できます。レプリケーション環境では、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・レプリケーション環境にあるすべてのマスター・サイトは、相互に直接通信して、レプリケーション・グループ内のデータとスキーマの変更を波及させます。マスター・サイトにあるレプリケーション・グループを、特に「マスター・グループ」と呼びます。また、すべてのマスター・グループは、「マスター定義サイト」を 1 つだけ持ちます（たとえば、図 34-3 の ORC1.WORLD を、マスター定義サイトにすることができます）。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読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・サイトにある表スナップショットには、レプリケーション・グループ内のすべての表データ、またはそのサブセットを含めることができます。ただし、これらは、マスター・サイトにある表と 1 対 1 の対応付けを持つ単純スナップショットである必要があります。たとえば、スナップショットには、レプリケーション・グループ内の選択した表のみのスナップショットを含めることができます。また、特定のスナップショットを、特定のレプリケート表から選択した部分のみにすることもできます。スナップショット・サイトにあるレプリケーション・グループを、特に「スナップショット・グ

ループ」と呼びます。スナップショット・グループには、他のレプリケーション・オブジェクトを含めることもできます。

図 34-3 3 つのマスター・サイトと 1 つのスナップショット・サイト



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」を使用すると、相互に対等な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 34-4 に、マルチマスター・レプリケーション・システムを示します。

マルチマスター環境でマスター・サイトとして稼働する Oracle データベース・サーバーは、すべての表レプリカのデータを自動的に集中させることによって、グローバルなトランザクションの一貫性とデータの整合性を保証します。

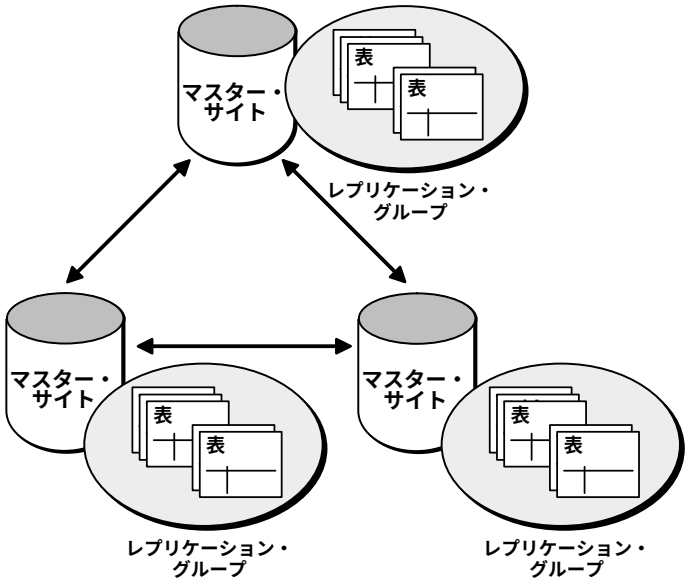
マルチマスター・レプリケーションの使用方法

マルチマスター・レプリケーションは、特殊な要件を伴うさまざまなタイプのアプリケーション・システムで使用できます。次の使用例では、マルチマスター・レプリケーションの使用方法をいくつか説明します。

フェイルオーバー・サイト

マルチマスター・レプリケーションは、ミッション・クリティカルなデータベースの可用性保護に使用できます。たとえば、マルチマスター・レプリケーション環境では、データベース内のすべてのデータをレプリケートし、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に備えてフェイルオーバー・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、このようなフェイルオーバー・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。

図 34-4 マルチマスター・レプリケーション・システム

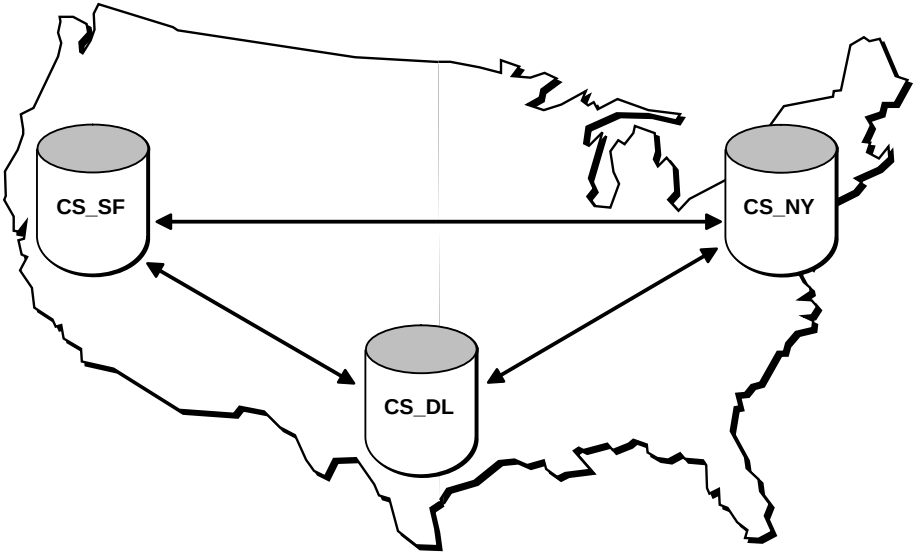


アプリケーション負荷の分散

マルチマスター・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。

通常、アプリケーション負荷分散要件を持つアプリケーションは、カスタマ・サービス指向のアプリケーションなどです。（アプリケーション負荷は、更新可能スナップショットを使用して分散させる方法もあります。詳細は、34-8 ページの「スナップショット・レプリケーション」を参照してください。

図 34-5 複数の更新アクセス・ポイントをサポートしているマルチマスター・レプリケーション



スナップショット・レプリケーション

スナップショットには、特定の時点からのターゲット・マスター表の完全または部分的なレプリカが含まれています。スナップショットは、読込み専用でも更新可能でもかまいません。

読込み専用スナップショット

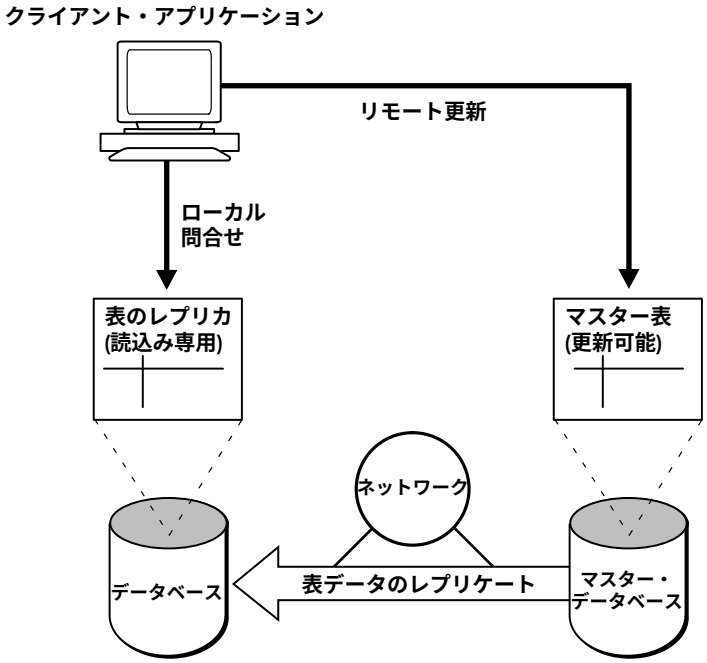
基本構成では、スナップショットは、プライマリ、つまり「マスター」サイトから発行される表データへの読込み専用アクセスを提供します。アプリケーションは、ネットワークが使用できるかどうかには関係なく、ローカルなデータ・レプリカのデータを問い合せることができ、ネットワーク・アクセスを回避します。ただし、更新操作が必要な場合は、システム全体のどのアプリケーションも、プライマリ・サイトのデータにアクセスする必要があります。図 34-6 に、基本的な読込み専用レプリケーションを示します。

読込み専用スナップショットの利点は、次のとおりです。

- マスター表がマスター・グループに属する必要はない。
- 複雑なスナップショットをサポートできる（スナップショットの基礎として 1 つ以上の表を使用し、集約演算、結合演算、集合演算または CONNECT BY 句を含めることができます）。

- ローカル・アクセスを提供して応答時間と可用性を改善する。
- マスター・サイトからの問合せをオフロードする。

図 34-6 読込み専用スナップショットのレプリケーション

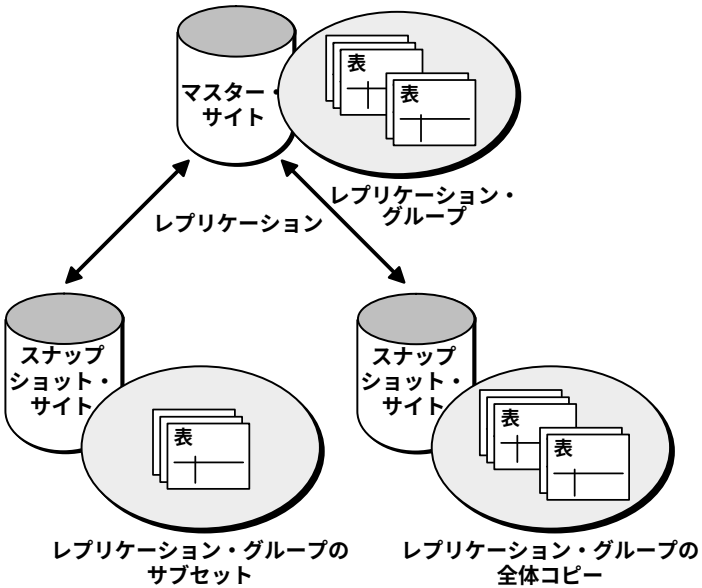


更新可能スナップショット

より拡張された構成では、ターゲット・マスター表の行をユーザーが挿入、更新および削除できるように、「更新可能スナップショット」を作成できます。更新可能スナップショットには、ターゲット・マスター表のデータ・セットのサブセットのみを含めることもできます。図 34-7 に、更新可能スナップショットを使用するレプリケーション環境を示します。

更新可能スナップショットは、マルチマスター・レプリケーションのサポート用にセットアップされたマスター・サイトにある表に基づいています。実際には、更新可能スナップショットは、マスター・サイトにあるマスター・グループに基づくスナップショット・グループの一部である必要があります。

図 34-7 更新可能スナップショットのレプリケーション



更新可能スナップショットには次のプロパティがあります。

- 更新可能スナップショットは、常に単一の表に基づき、増分（または「高速」）リフレッシュできる。
- 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・マスター表に波及させる。必要であれば、更新は他のすべてのマスター・サイトにもカスケードします。
- Oracle は、更新可能スナップショットを、読込み専用スナップショットの場合と同じリフレッシュ・グループの一部としてリフレッシュする。(「リフレッシュ・グループ」は、トランザクションの一貫性を保つ編成上のメカニズムです)。

更新可能スナップショットには次の利点があります。

- ユーザーは、マスター・サイトから切断されているときにも、ローカルにレプリケートされたデータセットを問い合せて更新できる。
- ターゲット・マスター表のデータ・セットのうち、選択したサブセットのみをレプリケートすることにより、データ・セキュリティが向上する。
- マルチマスター・レプリケーションに比べてフットプリントが小さい。

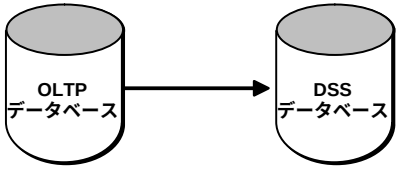
スナップショット・レプリケーションの使用方法

スナップショット・レプリケーションは、いくつかのタイプのアプリケーションに使用できます。次の使用例では、スナップショット・レプリケーションの使用方法をいくつか説明します。

情報のオフロード

読込み専用スナップショット・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は、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を切り離してください。

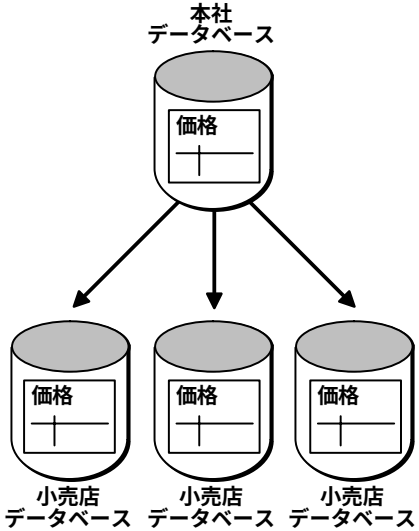
図 34-8 情報のオフロード



情報の配布

読込み専用スナップショット・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ます。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る必要があります。この目標を達成するため、小売店ごとに価格表の独自コピーを用意し、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します。

図 34-9 情報の配布



情報輸送

読込み専用および更新可能スナップショット・レプリケーションは、情報輸送メカニズムとして使用できます。たとえば、読込み専用スナップショット・レプリケーションを使用して、実働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。

非接続環境

更新可能スナップショット・レプリケーションは、非接続コンポーネントを使用して作動するトランザクション処理アプリケーションの実行に使用できます。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。各営業担当はラップトップ・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・ネットワークや中央のデータベース・システムからは切断されています。オフィスに戻った各営業担当は、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。

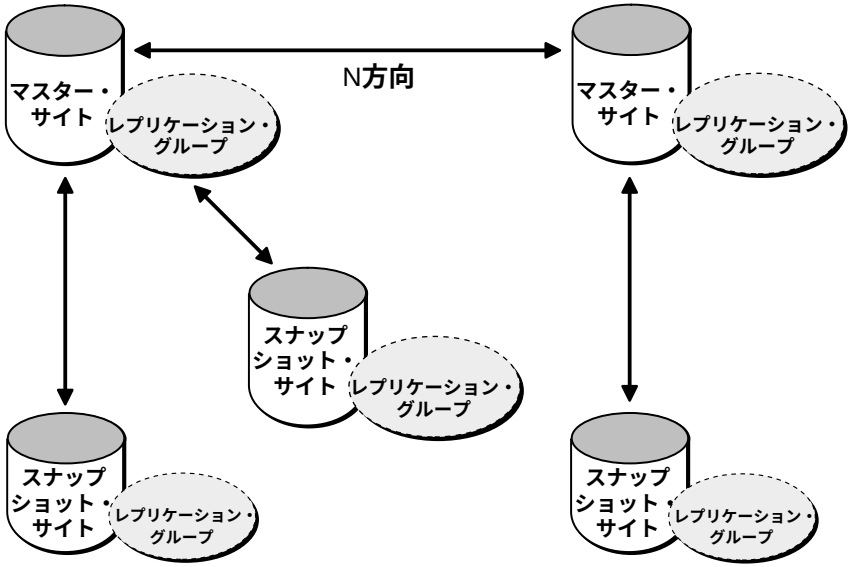
スナップショット環境を営業部などに展開できるように、データベース管理者は「実行テンプレート」を使用して、マスター・サイトでスナップショット環境を事前に作成できます。これにより、スナップショット環境を独自に、容易かつ安全に分散させてインストールできます。実行テンプレートにより、DBA はスナップショット環境を 1 度作成するだけで、必要に応じて何度でもターゲット・スナップショット・サイトに展開できます。

マルチマスターおよびスナップショットのハイブリッド構成

マルチマスター・レプリケーションと更新可能スナップショットを「ハイブリッド」（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すことができます。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 34-10 のように、2 つのマスター間の *n* 重（またはマルチマスター）レプリケーションでは、2 つの地域をサポートするデータベース間の全表レプリケーションをサポートできます。表全体または表のサブセットを各地域のサイトにレプリケートするために、マスターに対してスナップショットを定義できます。

図 34-10 ハイブリッド構成



スナップショットとレプリケートされたマスターの主な違いは、次のとおりです。

- レプリケートされたマスターはレプリケートされる表全体のデータを含む必要があるが、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用すると、各トランザクションごとの変更を、その発生時にレプリケートできる。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ために競合が発生する場合、マスター・サイトはその競合を検出して解決する。

レプリケート環境の管理

レプリケーション環境を管理し、監視しやすいように、複数のツールを使用できます。Oracle Replication Manager は、環境を容易に管理できるように強力な GUI インタフェースを提供し、レプリケーション管理 API は、レプリケーション管理用にカスタマイズされたスクリプトを作成できるように、標準的なアプリケーション・プログラミング・インタフェース（API）を提供します。また、レプリケーション・カタログにより、レプリケート環境に関する情報を常に把握できます。

レプリケーション・カタログ

レプリケーション・レプリケーション環境にあるすべてのマスター・サイトとスナップショット・サイトには、「レプリケーション・カタログ」があります。サイトのレプリケーション・カタログは、そのサイトにあるレプリケーション・オブジェクトとレプリケーション・グループに関する管理情報をメンテナンスする、データ・ディクショナリ表およびビューの固有の集合です。レプリケーション環境に関係するすべてのサーバーは、そのレプリケーション・カタログ内の情報を使用して、レプリケーション・グループ内のオブジェクトのレプリケーションを自動化できます。

レプリケーション管理 API と管理要求

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）を使用します。サーバーの「レプリケーション管理 API」は、管理者が Oracle のレプリケーション機能を構成するために使用する、プロシージャとファンクションをカプセル化した PL/SQL パッケージの集合です。Oracle Replication Manager では、作業を実行するために、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用します。

「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 を使用して新しいマスター・グループを作成する場合、作業を完了させるため Replication Manager は、DBMS_REPCAT.CREATE_MASTER_REPGROUP プロシージャをコールします。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 Replication Manager

マスター・レプリケーション環境とスナップショット・レプリケーション環境の両方をサポートしているレプリケーション環境の場合、構成および管理作業が煩雑になる可能性があります。このようなレプリケーション環境の管理を支援するため、Oracle には、洗練された管理ツールである Oracle Replication Manager が用意されています。Oracle Replication Manager の使用方法と使用例は、他の項を参照してください。

レプリケーションの競合

非同期のマルチマスター環境と更新可能スナップショット・レプリケーション環境は、異なるサイトから発行された 2 つのトランザクションが、ほぼ同時に同じ行を更新するときなどに発生する可能性があるレプリケーションの競合を取り扱う必要があります。

データ競合が発生した場合は、それが業務規則に従って解決されることと、すべてのサイトでデータが収束されることを保証するメカニズムが必要です。

Oracle レプリケーションは、レプリケート環境で発生する競合をロギングするだけでなく、多様な競合解決方法を提供します。これにより、業務規則に従って競合を解決する競合解決システムを、データベースに合わせて定義できます。Oracle の事前構築済みの競合解決方法では解決できない固有の状況が発生した場合は、独自の競合ルーチンを作成して使用することもできます。

> **追加情報：** Oracle8*i* レプリケーション・ガイド には、データの競合を回避するデータベースの設計方法と、この種の競合が発生した場合にそれを解決する競合解決ルーチンの作成方法が記載されています。また、レプリケーション管理 API を使用して競合解決ルーチンを作成する方法も記載されています。

特殊レプリケーション・オプション

一部のアプリケーションには、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。その場合、典型的な行レベルのレプリケーションでは、ネットワークに大量のデータ変更による負荷がかかる可能性があります。このような問題を回避するため、レプリケーション環境で動作するバッチ処理アプリケーションでは、Oracle の「プロシージャ・レプリケーション」を使用して、データ・レプリカを収集するための簡単なストアド・プロシージャ・コールをレプリケートできます。プロシージャ・レプリケーションでは、アプリケーションが表の更新に使用するストアド・プロシージャのコール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用するには、システムのデータを修正するパッケージをすべてのサイトにレプリケートする必要があります。パッケージをレプリケートした場合は、各サイトごとに、そのパッケージのための「ラッパー」を生成する必要があります。アプリケーションがローカル・サイトにあるパッケージ・プロシージャをコールしてデータを修正すると、そのラッパーによって、レプリケート環境内の他のすべてのサイトでも最終的に同じパッケージ・プロシージャがコールされることが保証されます。プロシージャ・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 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用してデータをレプリケートする場合、データをレプリケートするプロシージャは、レプリケートされたデータの整合性を保証する必要があります。。つまり、そのようなプロシージャは、レプリケーションの競合を回避するか、あるいはそれを検出して解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用されるのは、通常、データベースが大規模なバッチ操作でしか使用できない場合です。そのような状況では、多数のトランザクションが同じデータを要求することがないため、レプリケーションの競合はほとんど発生しません。

同期（リアルタイム）データ伝播

非同期データ伝播は、レプリケーション環境の標準構成です。ただし、Oracle では、特殊な要件を持つ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る必要がある場合にのみ使用してください。

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）を使用するレプリケート環境を作成することもできます。

注意： レプリケーション・データのリアルタイム伝播を使用するレプリケーション・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。

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・システムにより、レプリケーションの競合を防止できます。

第 X 部

付録

第 X 部に含まれる付録は、次のとおりです。

- 付録 A「オペレーティング・システム固有の情報」

A

オペレーティング・システム固有の情報

このマニュアルでは、特定のオペレーティング・システムで Oracle を使用する際の詳細情報について、他の Oracle マニュアルに言及していることがあります。正式名称は実際のオペレーティング・システムによって異なりますが、これらの Oracle マニュアルを『インストレーションおよび構成ガイド』と呼びます。

この付録では、このマニュアル内でオペレーティング・システム別の Oracle のマニュアルを参照しているすべての箇所と、オペレーティング・システム（OS）によって異なる初期化パラメータのリストを記載します。Oracle を複数のオペレーティング・システムで使用している場合は、それらのオペレーティング・システム間でアプリケーションの移植性を確保するために、この付録の情報が参考になります。

次のリストは、このマニュアルに記載されたオペレーティング・システム別のトピックを、ページ順に一覧したものです。

- データ・ファイル、ファイル・ヘッダーのサイズ : 3-16 ページの「データ・ファイル」
- データ・ブロック、サイズ : 4-3 ページの「データ・ブロック」
- ロールバック・セグメント、トランザクションあたりの数 : 4-20 ページの「トランザクションとロールバック・セグメント」
- DBA の認証 : 5-3 ページの「管理者権限での接続」および 29-11 ページの「データベース管理者の認証」
- 管理者権限、前提条件 : 5-3 ページの「管理者権限での接続」
- Net8、Net8 ソフトウェアに含まれるドライバ : 6-5 ページの「Net8 の機能」
- プログラム・グローバル領域（PGA）: 7-15 ページの「PGA のサイズ」
- ソフトウェア・コード領域、共有または非共有 : 7-17 ページの「ソフトウェア・コード領域」
- Oracle の構成 : 8-2 ページの「プロセスのタイプ」
- バックグラウンド・プロセス、作成 : 8-6 ページの「バックグラウンド・プロセス」
- バックグラウンド・プロセス、DBW*n* プロセス : 8-8 ページの「データベース・ライター（DBWn）」
- バックグラウンド・プロセス、ARC*n*: 8-12 ページの「アーカイバ・プロセス（ARCn）」
- 専用サーバー、管理操作のための要求 : 8-20 ページの「マルチスレッド・サーバーの限定的運用」
- クライアント / サーバー通信 : 8-22 ページの「専用サーバー構成」
- Net8、ネットワーク・ドライバの選択とインストール : 8-25 ページの「プログラム・インタフェース・ドライバ」
- 通信ソフトウェア : 8-26 ページの「オペレーティング・システムの通信ソフトウェア」
- 索引、索引ブロックのオーバーヘッド : 10-26 ページの「索引ブロックの形式」
- ユーザーの認証 : 29-4 ページの「オペレーティング・システムによる認証」
- オペレーティング・システムによるロール管理 : 30-20 ページの「オペレーティング・システムとロール」
- 監査 : 31-5 ページの 「オペレーティング・システムの監査証跡に必ず記録されるイベント」および 31-6 ページの「OS 監査証跡に対する監査」
- パラレル回復と非同期 I/O: 32-11 ページの「パラレル回復を活用できる状況」

索引

数字

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

の

は

ひ

ふ

へ

ま

む

め

も

ゆ

よ

ら

り

る

れ

ろ

わ