Oracle® Database

概要

11*g* リリース 1（11.1）

部品番号 : E05765-03

2008 年 11 月

ORACLE®

Oracle Database 概要 , 11*g* リリース 1（11.1）

部品番号 : E05765-03

Oracle Database Concepts, 11*g* Release 1 (11.1)

原本部品番号 : B28318-05

原本著者 : Richard Strohm

原本協力者 : Lance Ashdown, Mark Bauer, Michele Cyran, Steve Fogel, Janis Greenberg, Sumit Jeloka, Paul Lane, Diana Lorentz, Jack Melnick, Sheila Moore, Antonio Romero, Vivian Schupmann, Cathy Shea, Douglas Williams, Omar Alonso, Penny Avril, Hermann Baer, Sandeepan Banerjee, Bill Bridge, Sandra Cheevers, Carol Colrain, Vira Goorah, Mike Hartstein, John Haydu, Wei Hu, Ramkumar Krishnan, Vasudha Krishnaswamy, Bill Lee, Bryn Llewellyn, Rich Long, Paul Manning, Mughees Minhas, Valarie Moore, Gopal Mulagund, Muthu Olagappan, Jennifer Polk, Kathy Rich, John Russell, Bob Thome, Randy Urbano, Mark Van de Wiel, Michael Verheij, Ron Weiss, Steve Wertheimer

制限付権利の説明

目次

3 表領域、データファイルおよび制御ファイル

4 トランザクションの管理

5 スキーマ・オブジェクト

6 スキーマ・オブジェクトの依存性

7 データ・ディクショナリ

8 メモリー・アーキテクチャ

9 プロセス・アーキテクチャ

10 アプリケーション・アーキテクチャ

11 Oracle Database ユーティリティ

12 データベースとインスタンスの起動と停止

第 III 部　Oracle Database の機能

13　データの同時実行性と整合性

14 管理性

15 バックアップおよびリカバリ

16 ビジネス・インテリジェンス

17 高可用性

18 大規模データベース（VLDB）

19 コンテンツ管理

20 データベース・セキュリティ

21 データ整合性

22 トリガー

23 情報の統合

第 IV 部　Oracle データベース・アプリケーション開発

24　SQL

25 サポートされているアプリケーション開発言語

26 Oracle データ型

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 データベース・サーバーの全機能について説明します。Oracle データベース・サーバーがどのように機能するかを説明します。この情報は、他のマニュアルに記載されている多くの実用的な情報を理解する上で概念的な基盤となります。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 データベース・サーバーを対象にしています。

この章の内容は次のとおりです。

対象読者

Oracle Database 概要は、データベース管理者、システム管理者およびデータベース・アプリケーションの開発者を対象にしています。

このマニュアルを使用するには、次の知識が必要です。

- リレーショナル・データベースの一般的な概念
- **第 1 章「Oracle Database の概要」**の概念と用語
- Oracle を実行しているオペレーティング・システムの環境

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。アメリカ国外からの場合は、+1-407-458-2479 にお電話ください。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 以前のリリースの Oracle をアップグレードする方法の詳細は、『Oracle Database アップグレード・ガイド』を参照してください。
- Oracle データベース・サーバーの管理方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- Oracle データベース・アプリケーションの開発に関する情報は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- Oracle データベースのパフォーマンスの最適化に関する情報は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- データ・ウェアハウスとビジネス・インテリジェンスに関する情報は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

- このマニュアルに記載されたユーティリティの詳細は、『Oracle Database ユーティリティ』を参照してください。

マニュアル・セットに含まれるマニュアルの多くでは、Oracle のインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているか、およびその使用方法については、『Oracle Database サンプル・スキーマ』を参照してください。

表記規則

このドキュメントでは、次の表記規則が使用されます。

規則	意味
太字	太字は、操作に関連付けられているグラフィカル・ユーザー・インタフェース（GUI）要素、または本文中や用語集で定義されている用語を示します。
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、コード例、画面に表示されるテキストまたはユーザーが入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.com/lang/jp/support/index.html
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

```
http://www.oracle.com/technology/global/jp/documentation/index.html
```

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=3
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

```
http://www.oracle.com/lang/jp/index.html
http://www.oracle.com/technology/global/jp/index.html
```

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

第 I 部

Oracle の概要

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

- 第 1 章「Oracle Database の概要」

1

Oracle Database の概要

この章では、Oracle データベース・サーバー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。

- Oracle Database アーキテクチャ
- Oracle Database の機能
- Oracle Database アプリケーション開発

Oracle Database アーキテクチャ

データベースとは、1 単位として扱われるデータの集合です。データベースの目的は、関連する情報の格納や取出しです。データベース・サーバーは、情報管理の鍵となります。一般に**サーバー**では、多数のユーザーが同時に同じデータへアクセスできるように、マルチユーザー環境において大量のデータが確実に管理されます。権限のないアクセスに対して保護機能を備えながら、障害のリカバリについても効率のよい解決方法を提供します。

Oracle Database は、エンタープライズ・グリッド・コンピューティング用に設計された最初のデータベースで、最も柔軟性があり、コスト効率の高い方法で情報とアプリケーションを管理できます。エンタープライズ・グリッド・コンピューティングでは、業界標準のモジュール化した記憶域とサーバーのラージ・プールが作成されます。このアーキテクチャによって、各新規システムをコンポーネントのプールから迅速にプロビジョニングできます。容量は、必要に応じてリソース・プールから容易に追加または再割当てできるため、ワークロードのピークに備えて余分なハードウェアを追加する必要はありません。

データベースには、**物理構造**および**論理構造**があります。物理構造と論理構造は分離されているので、論理記憶構造へのアクセスに影響を与えずに、データの物理記憶域を管理できます。

この項の内容は、次のとおりです。

- グリッド・アーキテクチャの概要
- アプリケーション・アーキテクチャの概要
- 物理データベース構造の概要
- 論理データベース構造の概要
- スキーマと一般的なスキーマ・オブジェクトの概要
- Oracle Database データ・ディクショナリの概要
- Oracle Database インスタンスの概要
- データベース・アクセスの概要
- Oracle Database ユーティリティの概要

グリッド・アーキテクチャの概要

グリッド・コンピューティングは、よりリジリエンスがあり低コストの企業情報システムを生成する情報技術（IT）アーキテクチャです。グリッド・コンピューティングを使用すると、ビジネスのニーズの変化に合せて、独立したモジュール・ハードウェアおよびソフトウェア・コンポーネントのグループを必要に応じて接続および再結合できます。

グリッド形式のコンピューティングは、企業の IT に共通する次の問題を解決します。

- 専用のハードウェア・リソースが十分に活用されていないアプリケーション・サイロ
- 保守にコストがかかり変更が難しい不便な大規模システム
- 全体として企業で十分に活用されていない断片化した情報

その他のコンピューティング・モデルと比較して、グリッド形式で設計および実装された IT システムは、サービス品質の向上、コストの削減および柔軟性の向上を実現します。優れたサービス品質は、シングル・ポイント障害がないこと、堅牢なセキュリティ・インフラストラクチャおよびポリシードリブンの集中管理により達成されます。低コストは、リソース使用率の向上と、管理および保守コストの大幅な削減によるものです。ソフトウェアおよびハードウェアのスタックを特定のタスク専用にするのではなく、すべてのリソースをプールしておき必要に応じて割り当てることで、容量を活用して冗長な機能が発生しないようにします。グリッド・コンピューティングにより、小規模な個別のハードウェア・コンポーネントも活用できるため柔軟性が増し、各コンポーネントのコストが削減され、企業は変化するニーズに合せたリソースの割当てが可能になります。

アプリケーション・アーキテクチャの概要

データベースの最も一般的なアーキテクチャは、クライアント / サーバーと複数層の 2 種類です。コンピューティング環境でインターネット・コンピューティングが普及するにつれて、多数のデータベース管理システムが複数層環境に移行しつつあります。

この項の内容は、次のとおりです。

クライアント / サーバー・アーキテクチャ

Oracle データベース・システムでは、**クライアント / サーバー・アーキテクチャ**を使用することにより、分散処理を容易に活用できます。このアーキテクチャでは、データベース・システムは 2 つの部分に分割されます。フロントエンドの**クライアント**とバックエンドの**サーバー**です。

クライアント クライアントは、データベース・サーバー上で実行される操作について要求を出すデータベース・アプリケーションです。サーバーが管理するデータの要求、処理および提示を行います。クライアント・ワークステーションは、このようなジョブにあわせて最適化できます。たとえば、クライアントで必要なディスク容量の縮小や、グラフィック機能を活用できます。通常、クライアントは、データベース・サーバーとは異なるコンピュータ上で実行されます。多数のクライアントを 1 つのサーバーに対して同時に実行できます。

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

複数層アーキテクチャ : アプリケーション・サーバー

従来の**複数層アーキテクチャ**の構成要素は、次のとおりです。

- 操作を開始するクライアントまたは起動側プロセス。
- 操作の各部分を実行する 1 つ以上のアプリケーション・サーバー。**アプリケーション・サーバー**には、アプリケーション・ロジックの大部分が含まれています。また、クライアント用のデータにアクセスし、なんらかの問合せ処理を実行することにより、データベース・サーバーの負荷をある程度軽減するプロセスです。アプリケーション・サーバーは、クライアントと複数のデータベース・サーバー間のインタフェースとして働き、セキュリティ・レベルを高める役割を果します。
- 操作で使用するほとんどのデータを格納するエンド・サーバーまたはデータベース・サーバー。

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

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

プロキシ認証が使用されている場合、クライアントの識別は、接続のすべての層で保持されます。

複数層アーキテクチャ：サービス指向アーキテクチャ

サービス指向アーキテクチャ（SOA）は、アプリケーション機能がサービスにカプセル化されている複数層アーキテクチャです。通常、SOA サービスは Web サービスとして実装されます。Web サービスには HTTP プロトコルを使用してアクセスでき、WSDL や SOAP など XML ベースの一連のオープン標準に基づいています。

Oracle Database 11*g* から、Oracle Database は、従来の複数層または SOA 環境において Web サービス・プロバイダとして機能できるようになりました。

関連項目：

- Web サービス・プロバイダとしての Oracle Database の詳細は、10-6 ページの「Web サービス・プロバイダとしての Oracle Database」を参照してください。
- データベースを使用した Web サービスの使用方法の詳細は、『Oracle XML DB 開発者ガイド』を参照してください。

物理データベース構造の概要

次の項では、データファイル、制御ファイル、REDO ログ・ファイル、アーカイブ REDO ログ・ファイル、パラメータ・ファイル、アラートおよびトレース・ログ・ファイルおよびバックアップ・ファイルなど、Oracle データベースの物理構造について説明します。

この項の内容は、次のとおりです。

データファイル

すべての Oracle データベースには、すべてのデータベース・データを含む 1 つ以上の物理**データファイル**があります。表や索引などの論理データベース構造のデータは、データベースに割り当てられたデータファイルに物理的に格納されます。

データファイルの特性は次のとおりです。

- 1 つ以上のデータファイルによって、表領域と呼ばれるデータベース記憶域の論理単位が形成されます。
- 1 つのデータファイルは、1 つの表領域のみに対応付けられます。
- データファイルがいっぱいになると自動的に拡張されるように定義できます。

通常のデータベース操作中、データファイル内のデータが必要に応じて読み取られ、Oracle Database のメモリー・キャッシュに格納されます。たとえば、あるユーザーがデータベースの表に入っている一部のデータにアクセスする場合、要求された情報がデータベースのメモリー・キャッシュに存在しなければ、その情報は適切なデータファイルから読み取られて、メモリーに格納されます。

修正されたデータや新規のデータは、必ずしも即時にデータファイルに書き込まれるわけではありません。ディスクへのアクセス回数を減らし、パフォーマンスを向上させるために、データはメモリー内に蓄積されてから、適切なデータファイルに一度に書き込まれます。これらの操作は、バックグラウンド・プロセスである**データベース・ライター・プロセス**によって決定されます。

一時表領域に格納されるデータファイルを**一時ファイル**と呼びます。一時ファイルは、3-18 ページの「一時データファイル」で説明している制限事項の対象になります。

> **関連項目：** Oracle Database のメモリーおよびプロセス構造の詳細は、1-10 ページの「Oracle Database インスタンスの概要」を参照してください。

制御ファイル

すべての Oracle データベースには、**制御ファイル**があります。この制御ファイルには、次のような情報を含む、データベースの物理構造を指定するエントリが含まれています。

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

Oracle Database では、制御ファイルを**多重化**できます。つまり、制御ファイルに関連する障害から保護するために、同じ制御ファイルの多数のコピーが同時に維持されます。

Oracle データベースの**インスタンス**が起動するたびに、データベース操作のためにオープンする必要のあるデータファイル、一時ファイルおよび REDO ログ・ファイルが、制御ファイルによって識別されます。データベースの物理的な構造が変更されると（データファイルや REDO ログ・ファイルの新規作成など）、制御ファイルは、その変更を反映するように Oracle Database によって自動的に修正されます。制御ファイルは、データベースのリカバリにも使用されます。

> **関連項目：** 第 3 章「表領域、データファイルおよび制御ファイル」

オンライン REDO ログ・ファイル

すべての Oracle Database には、2 つ以上のオンライン **REDO ログ・ファイル**の集合があります。これらのオンライン REDO ログ・ファイルと REDO ログ・ファイルのアーカイブ・コピーは、データベースの REDO ログと呼ばれます。**REDO ログ**は、データに加えられた変更をすべて記録する REDO エントリ（**REDO レコード**とも呼ばれます）で構成されます。万一障害が発生して、修正済のデータがデータファイルに書き込まれなかった場合でも、その変更は REDO ログから取得できるため、実行した作業が失われることはありません。

REDO ログ自体にかかわる障害から保護するため、Oracle Database では 2 つ以上の REDO ログのコピーを異なるディスク上に維持できるように、**多重 REDO ログ**を作成できます。

> **関連項目：** 1-21 ページの「データベースのバックアップおよびリカバリ機能の概要」

アーカイブ REDO ログ・ファイル

アーカイブ REDO ログ・ファイルは、データベースにより生成されるオンライン REDO ログ・ファイルのオフライン・コピーです。データベースが `ARCHIVELOG` モードの場合は、Oracle Database により REDO ログ・ファイルが自動的にアーカイブされます。オンライン REDO ログの自動アーカイブを有効にすることをお薦めします。

パラメータ・ファイル

パラメータ・ファイルには、特定のインスタンスおよびデータベースの構成パラメータのリストが格納されます。パラーメータ・ファイル（pfile）およびサーバー・パラメータ・ファイル（spfile）のどちらを使用しても、初期化パラメータをサーバー側ディスク・ファイルに永続的に格納して管理できます。サーバー・パラメータ・ファイルには次のような利点があります。

- アクティブなインスタンスでパラメータ値が変更された場合、ファイルは同時に更新されます。
- Real Application サービス・データベースのすべてのインスタンスからアクセスできるよう、ファイルは集中配置されます。

初期化パラメータを動的に維持する手段として、サーバー・パラメータ・ファイルを作成することをお薦めします。

関連項目：

- 12-4 ページの「初期化パラメータ・ファイルとサーバー・パラメータ・ファイル」
- パラメータ・ファイルの作成と変更については、『Oracle Database 管理者ガイド』を参照してください。

アラート・ファイルとトレース・ログ・ファイル

それぞれのサーバーとバックグラウンド・プロセスは、対応付けられたトレース・ファイルに書き込むことができます。プロセスによって内部エラーが検出されると、エラーについての情報がそのプロセスのトレース・ファイルに書き込まれます。トレース・ファイルに書き込まれる情報には、データベース管理者向けのものと Oracle サポート・サービス向けのものがあります。また、トレース・ファイル情報は、アプリケーションとインスタンスのチューニングにも使用されます。アラート・ファイルは、アラート・ログとも呼ばれる特殊なトレース・ファイルです。データベースのアラート・ログは、メッセージとエラーの履歴ログです。

次の機能は、トレース・ファイルとアラート・ファイルの情報の収集と解釈の自動化とサポートを提供します。

- **自動診断リポジトリ**（ADR）は、トレース・ファイルおよびその他のエラー診断データを格納し構成するためのシステム管理リポジトリです。ADR はデータベースで発生したすべての重大なエラーを包括的に表示し、問題の診断と今後の解決に必要なすべての関連データを維持します。同じタイプのインシデントが頻繁すぎる場合、ADR は大量のデータを制御し、診断情報の過度のダンピングを回避します。
- インシデント・パッケージ化サービス（IPS）は重大なエラーに関連する診断データおよびテスト・ケース・データを ADR から抽出し、オラクル社への送信のためにデータをパッケージ化します。

関連項目： 詳細は、『Oracle Database 管理者ガイド』を参照してください。

バックアップ・ファイル

ファイルをリストアするには、バックアップ・ファイルで置き換えます。通常、ファイルをリストアするのは、メディア障害やユーザー・エラーによりオリジナルのファイルが破損したり削除された場合です。

ユーザー管理バックアップおよびリカバリでは、バックアップの試行リカバリを実行する前に、実際にバックアップ・ファイルをリストアする必要があります。

サーバー管理バックアップおよびリカバリでは、リカバリが必要な時点で適切なバックアップ・ファイルを適用するなどのリカバリ処理のみでなく、バックアップのスケジューリングなどのバックアップ処理も管理されます。

関連項目：

- 第 15 章「バックアップおよびリカバリ」
- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』

論理データベース構造の概要

この項では、データ・ブロック、エクステント、セグメントおよび表領域などの論理記憶構造について説明します。これらの論理記憶構造によって、Oracle Database のディスク領域の使用をきめ細かく制御できます。

この項の内容は、次のとおりです。

Oracle Database データ・ブロック

最少レベルとして、Oracle データベースのデータは**データ・ブロック**に格納されます。1 つのデータ・ブロックは、ディスク上の特定のバイト数の物理データベース領域に対応します。標準のブロック・サイズは、DB_BLOCK_SIZE 初期化パラメータで指定します。また、他のブロック・サイズを 4 つまで指定できます。データベースは、Oracle Database データ・ブロック内の空きデータベース領域を使用して領域を割り当てます。

エクステント

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

セグメント

エクステントの上位に位置する論理データベース記憶域のレベルは、**セグメント**です。セグメントは、表、索引、ロールバック・セグメント、またはセッション、トランザクションおよび SQL パーサーでの一時使用のために割り当てられるエクステントの集合です。物理データベース構造に関して、すべてのエクステントは同じ表領域に存在するセグメントに属しますが、異なるデータファイルにある場合があります。

セグメントのエクステントがいっぱいになると、Oracle Database は、そのセグメントに別のエクステントを動的に割り当てます。エクステントは必要に応じて割り当てられるため、1 つのセグメントの各エクステントはディスク上で連続している場合と連続していない場合があります。

表領域

データベースは、関連するデータ・ブロック、エクステントおよびセグメントがグループ化された**表領域**と呼ばれる論理記憶単位に分けられます。たとえば、通常、表領域はすべてのアプリケーション・オブジェクトをグループ化し、管理操作を容易にします。

各データベースは、論理的に 2 つ以上の表領域に分けられます。各表領域に対して 1 つ以上のデータファイルが明示的に作成され、すべての論理構造のデータが表領域に物理的に格納されます。表領域内のデータファイルのサイズを組み合せると、表領域全体の記憶容量になります。

Oracle データベースには、SYSTEM および SYSAUX という表領域が含まれています。この 2 つの表領域は、データベースの作成時に Oracle Database により自動的に作成されます。システム・デフォルトでは、従来型の Oracle 表領域である**小型ファイル表領域**が作成されます。SYSTEM および SYSAUX 表領域は、小型ファイル表領域として作成されます。

また、Oracle Database では、多数の小型ファイルではなく単一の大型ファイルで構成されている**大型ファイル表領域**も作成できます。大型ファイル表領域により、Oracle Database では、64 ビット・システムの機能を利用して超大規模ファイルを作成および管理できます。この結果、Oracle Database は 800 万 TB までサイズを拡張できるようになりました。Oracle Managed Files では、大型ファイル表領域によりデータファイルがユーザーに対して完全に透過的になります。つまり、基礎となるデータファイルではなく表領域に対する操作を実行できます。

関連項目：

- 第 2 章「データ・ブロック、エクステントおよびセグメント」
- 第 3 章「表領域、データファイルおよび制御ファイル」
- 2-16 ページの「UNDO セグメントおよび自動 UNDO 管理の概要」
- 1-16 ページの「読取り一貫性」
- 1-21 ページの「データベースのバックアップおよびリカバリ機能の概要」

オンライン表領域とオフライン表領域 表領域は、**オンライン**または**オフライン**に設定できます。ユーザーが表領域内の情報にアクセスできるように、通常、表領域はオンラインになっています。ただし、管理を単純化するために、場合によっては、1 つの表領域をオフラインにしてデータベースの一部を使用できないようにすると同時に、データベースの残りの部分には通常どおりアクセスできるようにすることもできます。

読取り専用表領域 表領域を**読取り専用**、つまり表領域のデータを変更できないように設定できます。読取り専用表領域の主な目的は、データベースの静的な部分を大量にバックアップしたりリカバリする必要をなくすことです。読取り専用表領域のファイルは Oracle Database によって更新されることがないため、CD-ROM や WORM ドライブのような読取り専用メディアに常駐させることができます。

スキーマと一般的なスキーマ・オブジェクトの概要

スキーマとは、データベース・オブジェクトの集合です。スキーマはデータベース・ユーザーによって所有され、そのユーザーと同じ名前を持ちます。**スキーマ・オブジェクト**は、データベースのデータを直接参照する論理構造です。表領域とスキーマとの間に関連付けはありません。同じスキーマのオブジェクトを異なる表領域に含めることができます。また、ある表領域に異なるスキーマのオブジェクトを含めることができます。スキーマ・オブジェクトには、**表**、**ビュー**および**索引**などの構造が含まれます。ここでは、最も一般的なスキーマ・オブジェクトの定義について説明します。

この項の内容は、次のとおりです。

- 表
- 索引
- ビュー
- クラスタ
- シノニム

表

表は、Oracle データベースにおけるデータ記憶域の基本単位です。データベースの表には、ユーザーがアクセスできるすべてのデータが保持されています。それぞれの表には**列**と**行**があります。たとえば、従業員情報の表には、`employee_number` という列があり、その列の各行には従業員番号が格納されます。

索引

索引とは、表に対応付けられているオプションの構造です。索引を作成し、データ検索のパフォーマンスを向上できます。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。それと同じように、Oracle データベースの索引を使用すると表データにアクセスできます。

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

索引は、表の 1 つ以上の列に対して作成されます。作成された索引は、Oracle Database によって自動的に使用され、メンテナンスされます。表データに対する変更（新しい行の追加、行の更新または削除など）は、関連するすべての索引にも自動的に取り込まれます。

ビュー

ビューとは、1 つ以上の表または他のビューに含まれているデータの表現がカスタマイズされたものです。ビューは、ストアド・クエリーとみなすこともできます。ビューに実際のデータは含まれていません。データはビューの基礎となる表からデータを導出します。これらの表を、ビューの**実表**と言います。

いくつかの制限付きでビューを表と同様に問合せ、更新、挿入および削除できます。ビューが更新可能な場合、ビューに対して実行する操作は、すべてビューの実表にも影響します。

ビューでは、事前に定義された表の行および列の集合のみにアクセスを制限することで、表のセキュリティを提供できます。また、データの複雑さを隠し、複雑な問合せを格納します。

クラスタ

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

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

シノニム

シノニムとは、表、ビュー、マテリアライズド・ビュー、順序、演算子、プロシージャ、ファンクション、パッケージ、Java クラスのスキーマ・オブジェクト、ユーザー定義オブジェクト型または他のシノニムの別名です。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶域は必要ありません。

> **関連項目：** これらのスキーマ・オブジェクトと他のスキーマ・オブジェクトの詳細は、第 5 章「スキーマ・オブジェクト」を参照してください。

Oracle Database データ・ディクショナリの概要

各 Oracle データベースには、データベースの参照として使用される、表およびビューの集合である**データ・ディクショナリ**があります。たとえば、データ・ディクショナリには、データベースの論理構造と物理構造の両方に関する情報が格納されます。また、データ・ディクショナリには、Oracle データベースの有効なユーザー、データベース内の表に定義された整合性制約に関する情報、スキーマ・オブジェクトに割り当てられている領域の量およびその使用率などの多くの情報も格納されます。

データ・ディクショナリは、データベースの作成時に作成されます。常にデータベースの正確な状態が反映されるように、データ・ディクショナリは、データベース構造の変更など、特定の処理が行われるたびに、Oracle Database により自動的に更新されます。データベース・ユーザーはデータ・ディクショナリを変更できません。各種のデータベース・プロセスでは、データ・ディクショナリを基盤として、進行する作業の記録、検証および指示が実行されます。たとえば、データベースの操作中、Oracle Database は、スキーマ・オブジェクトが存在しているかどうか、およびユーザーが適切なアクセス権を持っているかどうかを検証するためにデータ・ディクショナリを読み取ります。

> **関連項目：** 詳細は、第 7 章「データ・ディクショナリ」を参照してください。

Oracle Database インスタンスの概要

Oracle データベース・サーバーは、Oracle Database と 1 つ以上の Oracle データベース・インスタンスで構成されます。データベースを起動するたびに、システム・グローバル領域（SGA）と呼ばれる共有メモリー領域が割り当てられ、Oracle Database バックグラウンド・プロセスが起動されます。バックグラウンド・プロセスと SGA を組み合せて、Oracle データベース・**インスタンス**と呼びます。

Oracle Real Application Clusters　ハードウェア・アーキテクチャ（共有ディスク・システムなど）によっては、複数のコンピュータで、データ、ソフトウェアまたは周辺装置へのアクセスを共有できます。Oracle Real Application Clusters（Oracle RAC）は、相互接続を使用して相互に通信する複数のクラスタ化コンピュータ上で実行される 2 つ以上の Oracle データベース・インスタンスで構成されます。Oracle RAC は、Oracle Clusterware を使用して、共有ディスクに常駐する共有データベースにアクセスします。このように相互接続された複数のコンピュータの処理能力を組み合せることで、Oracle RAC はシステムの冗長性、ほぼ線形のスケーラビリティおよび高可用性を提供します。また、Oracle RAC は OLTP とデータ・ウェアハウス・システムの両方にとって大きな利点があり、すべてのシステムとアプリケーションはクラスタ環境を効率的に活用できます。

Oracle RAC 環境では、アプリケーションをそのまま拡張して増大するデータ処理需要に対処でき、アプリケーション・コードを変更する必要がありません。ノードやストレージなどのリソースを追加すると、これらのリソースの処理能力は Oracle RAC により個々のコンポーネントの制限を超えて拡張されます。

> **関連項目：**『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』

ユーザーが Oracle データベース・サーバーに接続すると、Oracle データベース・インスタンスへと接続されます。データベース・インスタンスは、SGA のみでなくその他のメモリー領域も割り当て、Oracle Database バックグラウンド・プロセスの他のプロセスも起動して、それらのユーザーに対応します。次の各項では、Oracle Database の様々なメモリー領域およびプロセスについて説明します。

- Oracle Database バックグラウンド・プロセス
- インスタンスのメモリー構造

Oracle Database バックグラウンド・プロセス

Oracle データベースは、メモリー構造とプロセスを使用してデータベースの管理やアクセスを行います。すべてのメモリー構造は、データベース・システムを構成するコンピュータのメイン・メモリー内に存在します。**プロセス**は一連の処理を実行できるオペレーティング・システムのメカニズムです。オペレーティング・システムによっては、「ジョブ」または「タスク」という用語を使用します。Oracle データベース・サーバーは、サーバー・プロセスおよびバックグラウンド・プロセスを含む Oracle プロセス、およびユーザー・プロセスの 3 つのタイプのプロセスを使用します。ほとんどのシステムでは、Oracle プロセスとユーザー・プロセスはそれぞれ別々のコンピュータで実行されます。

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

  > **関連項目：** 最も一般的なバックグラウンド・プロセスの詳細は、9-5 ページの「Oracle Database バックグラウンド・プロセス」を参照してください。

- **ユーザー・プロセス**（クライアント・プロセスとも呼ばれる）を作成してメンテナンスを実行すると、OCI または OCCI プログラムなどのアプリケーション・プログラムや、Oracle Enterprise Manager などの Oracle のツール製品のソフトウェア・コードを実行できます。ほとんどの環境では、クライアント・プロセスは別のマシン（ラップトップ、デスクトップなど）で実行されます。後述のように、ユーザー・プロセスは、プログラム・インタフェースを介してサーバー・プロセスとの通信を管理します。
- Oracle Database は、接続されたユーザー・プロセスの要求を処理するために、**サーバー・プロセス**を作成します。サーバー・プロセスは、関連付けられたユーザー・プロセスの要求を実行するために、ユーザー・プロセスと通信し、Oracle Database と対話します。たとえば、ユーザーが SGA の**データベース・バッファ**に存在しないデータを問い合せた場合、関連付けられたサーバー・プロセスが、データファイルから SGA に適切な**データ・ブロック**を読み取ります。

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

  関連項目： 第 9 章「プロセス・アーキテクチャ」

インスタンスのメモリー構造

Oracle Database では、様々な目的でメモリー構造が作成および使用されます。たとえば、実行中のプログラム・コード、ユーザー間で共有されるデータおよび接続している各ユーザーのプライベート・データ領域を格納するときに、メモリーが使用されます。Oracle Database には、基本的な 2 つのメモリー構造が関連付けられています。

- システム・グローバル領域（SGA）は、SGA コンポーネントと呼ばれる共有メモリー構造のグループで、1 つの Oracle データベース・インスタンスに関するデータと制御情報が含まれています。SGA は、すべてのサーバーおよびバックグラウンド・プロセスで共有されます。SGA に保存されるデータの例としては、キャッシュ・データ・ブロックや共有 SQL 領域があります。
- プログラム・グローバル領域（PGA）はサーバー・プロセスまたはバックグラウンド・プロセス用のデータや制御情報を含むメモリー領域です。PGA は、サーバー・プロセスまたはバックグラウンド・プロセスの起動時に、Oracle Database によって作成される非共有メモリーです。PGA にアクセスできるのはプロセスのみです。それぞれのサーバー・プロセスおよびバックグラウンド・プロセスには専用の PGA があります。

  関連項目： 詳細は、第 8 章「メモリー・アーキテクチャ」を参照してください。

データベース・アクセスの概要

この項では、Oracle Net Services およびデータベースの起動方法について説明します。この項の内容は、次のとおりです。

- ネットワーク接続
- データベースの起動
- Oracle Database の動作

ネットワーク接続

Oracle Net Services は、Oracle Database とネットワークの通信プロトコルとの間のインタフェースであり、分散処理と分散データベースの実現を支援します。通信プロトコルにより、ネットワーク上でのデータの送受信方法が定義されます。Oracle Net Services は、TCP/IP、HTTP、FTP および WebDAV など、主要なネットワーク・プロトコルによる通信をすべてサポートしています。

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

Oracle Net は Oracle Net Services のコンポーネントであり、クライアント・アプリケーションから Oracle Database サーバーへのネットワーク・セッションを確立して維持します。ネットワーク・セッションの確立後は、Oracle Net はクライアント・アプリケーションとデータベース・サーバーのためのデータ伝達手段として機能し、これらの間でメッセージを交換します。Oracle Net は、ネットワーク内の各コンピュータに配置されているのでこれらのジョブを実行できます。

関連項目：

- ネットワーク接続の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。
- データベースでの WebDAV の使用方法は、『Oracle XML DB 開発者ガイド』を参照してください。

データベースの起動

Oracle データベースを起動して、システム全体で使用可能にするには、次の 3 つの操作を行います。

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

データベース管理者は、Oracle Enterprise Manager、SQL*Plus の `STARTUP` 文、`srvctl` コマンドライン・ツールまたは Express Edition の `START` コマンドを使用してこれらの手順を実行できます。Oracle Database は、インスタンスを起動するときに、サーバー・パラメータ・ファイル（spfile）または初期化パラメータ・ファイル（pfile）を読み取って、初期化パラメータの値を判別します。次に、SGA を割り当ててバックグラウンド・プロセスを作成します。

関連項目： 第 12 章「データベースとインスタンスの起動と停止」

Oracle Database の動作

次の例は、Oracle Database が実行する操作の最も基本的なレベルです。ここでは、ユーザー・プロセスとそれに対応するサーバー・プロセスが、ネットワークで接続された別々のコンピュータに存在する Oracle Database 構成について説明します。

1. **インスタンス**は、Oracle Database が稼働しているコンピュータ（通常は**ホスト**または**データベース・サーバー**と呼ばれる）で起動されています。
2. アプリケーションを実行しているコンピュータ（**ローカル・コンピュータ**または**クライアント・ワークステーション**）は、**ユーザー・プロセス**でアプリケーションを実行します。クライアント・アプリケーションは、適切な Oracle Net Services ドライバを使用して、サーバーへの**接続**を確立しようとします。
3. サーバーは、適切な Oracle Net Services ドライバを実行しています。サーバーはアプリケーションからの接続要求を検出すると、ユーザー・プロセスのために専用サーバー・プロセスを作成します。
4. ユーザーは SQL 文を実行して、トランザクションをコミットします。たとえば、ユーザーは表の行に入っている名前を変更します。
5. サーバー・プロセスは SQL 文を受け取り、同一の SQL 文を含む共有 SQL 領域がないかどうか**共有プール**（SGA コンポーネント）をチェックします。共有 SQL 領域が見つかった場合、サーバー・プロセスはユーザーが要求したデータへのアクセス権を持っているかどうかチェックし、既存の共有 SQL 領域を使用して SQL 文を処理します。共有 SQL 領域が見つからなかった場合は、新しい共有 SQL 領域をその SQL 文に割り当て、解析と処理を実行します。

6. サーバー・プロセスは、実際のデータファイル（表）から必要なデータ値を取得するか、またはシステム・グローバル領域内に格納されている必要なデータ値を取得します。
7. サーバー・プロセスは、システム・グローバル領域内のデータを修正します。データベース・ライター・プロセス（DBW*n*）は、書込みが効率よく行われるタイミングを判断して、修正したブロックをディスクに書き込みます。トランザクションがコミットされると、ログ・ライター・プロセス（LGWR）がそのトランザクションを REDO ログ・ファイルに即時に記録します。
8. トランザクションが成功すると、サーバー・プロセスは、ネットワークを介してアプリケーションにメッセージを送信します。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体を通じて、他のバックグラウンド・プロセスも実行されていて、介入が必要かどうかを監視しています。さらに、データベース・サーバーは、他のユーザーのトランザクションを管理し、同一のデータを要求する異なるトランザクション間の競合を防止します。

関連項目： バックグラウンド・プロセスの詳細は、第 9 章「プロセス・アーキテクチャ」を参照してください。

Oracle Database ユーティリティの概要

Oracle Database には、データ転送、データ維持およびデータベース管理用の複数のユーティリティが用意されています。それぞれのユーティリティについては、第 11 章「Oracle Database ユーティリティ」で簡単に、また、『Oracle Database ユーティリティ』で詳細に説明しています。

Oracle Database の機能

この項の内容は、次のとおりです。

- Oracle Real Application Testing の概要
- 同時実行性機能の概要
- 管理性機能の概要
- 診断性機能の概要
- データベースのバックアップおよびリカバリ機能の概要
- 高可用性機能の概要
- ビジネス・インテリジェンス機能の概要
- コンテンツ管理機能の概要
- セキュリティ機能の概要
- データ整合性とトリガーの概要
- 情報統合機能の概要

Oracle Real Application Testing の概要

ハードウェアやソフトウェアのアップグレードおよびアプリケーション・パッチなどのシステム変更は、規格への準拠やセキュリティのため、また競合性を保つために企業にとって不可欠です。Oracle Real Application Testing は、実環境のアプリケーションにおけるシステム変更の効果を、実際の環境にデプロイする前にテスト環境で完全に評価するのに役立ちます。Oracle Real Application Testing は次の 2 つの機能で構成されています。

- データベースの再実行
- SQL Performance Analyzer

データベースの再実行

データベースの再実行では、原則的に本番ワークロード環境をテスト・システムで再作成することにより、システム変更を実際にテストします。これは、本番システムのワークロードを取得し、その後、完全に同じタイミング、同時実行性および元のワークロードのトランザクションの特性でテスト・システムを再実行することで実現します。再実行により、好ましくない結果、新たな競合ポイントおよびパフォーマンスの低下などの変更の影響を完全に評価できます。新たなエラーやパフォーマンスの相違などの潜在的な問題を識別するために、詳細な分析およびレポートが提供されます。

データベースの再実行により、企業は変更を迅速にテストし、作業の全体的な成功に自信を持ち、非常に低いリスクで新しいテクノロジを採用することが可能です。

データベースの再実行を使用し、次のタイプのシステム変更の影響を評価できます。

- データベースのアップグレード、パッチ、パラメータおよびスキーマの変更
- シングル・インスタンスから Oracle Real Application Clusters および自動ストレージ管理への変換などの構成の変更
- 記憶域、ネットワークおよびインターコネクトの変更
- オペレーティング・システム・パッチ、アップグレード、パラメータの変更およびハードウェアの移行

SQL Performance Analyzer

SQL 実行計画に影響を与える変更は、システム・パフォーマンスおよび可用性に重大な影響を与える場合があります。その結果、DBA は変更によりパフォーマンスが低下した SQL 文の特定と修正に、非常に時間をかけることになります。

SQL Performance Analyzer は、各 SQL 文のパフォーマンスの相違を識別することにより、完全な SQL ワークロードに対する変更による全体への影響を評価する手順を自動化します。この機能では、ワークロード・パフォーマンスに対する変更による最終的な影響を表したレポートが提供されます。パフォーマンスが低下した SQL 文に対しては、適切な実行計画の詳細およびチューニングの推奨事項が提供されます。これにより、DBA はエンド・ユーザーに影響を与える前に悪い結果を解決し、時間とコストを大幅に削減して、本番環境に対するシステム変更が最終的に改善されることを確認できます。

SQL Performance Analyzer を使用し、すべてのタイプのシステム変更による SQL パフォーマンスへの影響を分析できます。一般的なシステム変更には次のものがあります。

- データベースのアップグレード
- オペレーティング・システム、ハードウェアまたはデータベースの構成変更
- データベース初期化パラメータの変更
- 新しい索引またはマテリアライズド・ビューの追加などのスキーマの変更
- オプティマイザ統計の収集
- SQL プロファイルの作成などの SQL チューニング・アクション

> **関連項目：** SQL Performance Analyzer の使用方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

同時実行性機能の概要

すべての情報管理システムには、次の重要な要件があります。

- マルチユーザー・システムのデータの**同時実行性**を最大限に高めます。
- データを一貫した方法で読み取り、修正します。最初のユーザーがデータを使用し終わるまで、参照中または変更中のデータを（他のユーザーが）変更することはできません。
- データベース・システムの多数のユーザーから最大限の生産性を引き出すために高いパフォーマンスを実現します。

Oracle Database には、これらの要件を満たす複数のソフトウェア・メカニズムが組み込まれています。この項の内容は、次のとおりです。

- 同時実行性
- 読取り一貫性
- キャッシングのメカニズム
- ロックのメカニズム

同時実行性

マルチユーザー・データベース管理システムの最大の機能は、**同時実行性**、つまり多数のユーザーによる同一データへの同時アクセスです。十分な同時実行性制御機能がなければ、データが不当に更新または変更され、データ整合性が損われる可能性があります。

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

Oracle Database では、様々なタイプのロックとマルチバージョン一貫性モデルを使用することで、これらの問題を解決します。これらの機能は、**トランザクション**の概念に基づいています。

トランザクションは、読取り一貫性を実現するための Oracle Database の重要な機能です。コミット済の（またはコミットされていない）SQL 文から構成されているトランザクションは、次のように機能します。

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

同時実行性と一貫性に関するこれらの機能がトランザクションで十分に活用されるようにするのは、アプリケーション設計者の責任です。

関連項目： 第 4 章「トランザクションの管理」

読取り一貫性

Oracle Database が提供する読取り一貫性は、次の目的を達成します。

- 文が参照するデータの集合が、ある特定の時点で一貫しており、文の実行中に変化しません（文レベルの読取り一貫性）。
- データベース・データを読み取るユーザーは、その同じデータを書き込むユーザーまたはそれを読み取る他のユーザーのために待機しません。
- データベース・データを書き込むユーザーは、その同じデータを読み取るユーザーのために待機しません。
- 同時実行のトランザクションで同一行を更新しようとする場合にのみ、データを書き込むユーザーは他の書込みユーザーに対して待機します。

Oracle Database に実装されている読取り一貫性は、各ユーザーが自分専用のデータベースのコピーを操作しているような状況を実現します。これは、マルチバージョンの一貫性モデルとも呼ばれます。

> **関連項目：** 第 13 章「データの同時実行性と整合性」

読取り一貫性、UNDO レコードおよびトランザクション マルチバージョン一貫性モデルを管理するために、Oracle Database はシステム・グローバル領域内の現在の情報と UNDO レコードにある情報を使用して、問合せのために表データの**読取り一貫性を備えたビュー**を作成します。更新が発生すると、データの元の値が、データベースの UNDO レコードに記録されます。この更新がコミットされていないトランザクションの一部であるかぎり、変更されたデータに後から問い合せたユーザーは、データの元の値を参照することになります。トランザクションがコミットされた時点で初めて、トランザクションの変更が確定します。トランザクションがコミットされた後に開始された問合せのみ、そのコミットされたトランザクションによって加えられた変更を参照することになります。

読取り専用トランザクション デフォルトでは、Oracle Database により文レベルの読取り一貫性が保証されます。1 つの問合せによって戻された一連のデータは、ある特定の時点での一貫性を保っています。ただし、場合によっては、トランザクション・レベルの読取り一貫性が必要になることもあります。これは、1 つのトランザクション内で複数の問合せを実行するときに、そのトランザクション内の問合せが、途中でコミットされた他のトランザクションの結果を参照しないように、1 つのトランザクション内のすべての問合せに同一時点を基準とした読取り一貫性を持たせる機能です。複数の表に対して多数の問合せを実行し、更新を行わない場合は、**読取り専用トランザクション**として定義するコマンドを使用してトランザクションを開始できます。

> **関連項目：** トランザクション・レベルの読取り一貫性の詳細は、『Oracle Database 概要』を参照してください。

キャッシングのメカニズム

Oracle Database では、データベース・パフォーマンスを最適化するために、ユーザー・データ、ログ・データ、ディクショナリ・データおよびその他のタイプのデータがメモリー内にキャッシュされます。

Oracle Database では問合せ結果もキャッシュされるため、問合せが繰り返されると、問合せを再度処理して記憶域からデータを読み取るのではなく、データベースによりキャッシュから結果が戻されます。キャッシュされた結果は、共有プールの専用領域に格納されます。問合せ結果のキャッシュからの問合せの取得は、問合せの再実行より高速です。問合せ結果のキャッシュにより、データベースのメモリーに結果を明示的にキャッシュできます。頻繁に実行される問合せの場合は特に、問合せ結果のキャッシュを使用することでパフォーマンスが向上します。

ロックのメカニズム

Oracle Database は、**ロック**を使用してデータへの同時アクセスを制御します。情報の更新時には、更新が発行またはコミットされるまで、その情報にはデータ・サーバーによりロックされます。発行またはコミットされるまで、ロックされている情報は誰も変更できません。これにより、システムのデータ整合性が保証されます。

Oracle Database には、拡大されない一意の行レベル・ロック機能が用意されています。他のデータ・サーバーでは行グループ全体や表全体を扱うためにロックが拡大されますが、Oracle Database では更新中の情報行のみが常にロックされます。データベースには、実際の行自体を使用してロック情報が組み込まれるので、ロックできる行数に制限がなく、ユーザーは不必要な遅延なしで同時に作業できます。

自動ロック Oracle Database のロックは自動的に実行されるため、ユーザーの操作は必要ありません。要求される処理によっては、必要に応じて暗黙的ロックが SQL 文に対して発生します。

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

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

管理性機能の概要

Oracle データベース・システムの運用管理者はデータベース管理者（DBA）と呼ばれ、Oracle データベースの作成、円滑な運用の保証および使用の監視を担当します。Oracle Database には、多数のアラートとアドバイザに加えて、次の項で説明する機能が用意されています。

- 自己管理データベース
- 自動メンテナンス・タスク
- Oracle Enterprise Manager
- SQL Developer および SQL*Plus
- 自動メモリー管理
- 自動ストレージ管理
- Automatic Database Diagnostic Monitor
- SQL チューニング・アドバイザ
- SQL アクセス・アドバイザ
- ストリーム・チューニング・アドバイザ
- スケジューラ
- データベース・リソース・マネージャ

自己管理データベース

Oracle Database は高度な自己管理機能を備えており、日常的な DBA タスクが自動化され、領域、メモリーおよびリソース管理が簡素化されます。Oracle Database の自己管理データベース機能には、自動 UNDO 管理、自動サーバー・メモリー管理、Oracle Managed Files、空き領域管理および Recovery Manager（RMAN）が含まれます。

自動メンテナンス・タスク

Oracle Database では、統計収集や領域リカバリなど、定期的なメンテナンス・タスクは自動的にスケジューリングされます。これらのタスクは、メンテナンス・ウィンドウと呼ばれる Oracle Scheduler の一連のウィンドウから実行されます。これらのメンテナンス・ウィンドウの開始時間と期間を制御し、消費する CPU および I/O リソースの量を制限できます。

Oracle Enterprise Manager

Oracle Enterprise Manager は、データベース環境を集中管理するためのシステム管理ツールです。Oracle Enterprise Manager は、グラフィカル・コンソール、Oracle Management Server、Oracle Intelligent Agent、共通サービスおよび管理ツールを組み合せることで、Oracle 製品を管理するための包括的なシステム管理プラットフォームを提供します。

クライアント・インタフェースである Oracle Enterprise Manager コンソールから、次のタスクを実行できます。

- データベース、Oracle Application Server サーバー、アプリケーションおよびサービスなど、Oracle 環境全体の管理
- 複数のデータベースの診断、変更およびチューニング
- 複数システムでのタスクの様々な時間間隔によるスケジュール
- ネットワーク全体にわたるデータベースの状態の監視
- 多数の場所からの複数のネットワーク・ノードおよびサービスの管理
- 他の管理者とのタスクの共有
- 管理タスクを容易にするための関連ターゲットのグループ化
- 統合された Oracle およびサード・パーティ製のツールの起動
- Oracle Enterprise Manager 管理者による表示のカスタマイズ

SQL Developer および SQL*Plus

Oracle SQL Developer は、次のタスクの実行に便利な方法を提供するグラフィカル開発ツールです。

- データベース・オブジェクトの参照、作成、編集、削除
- PL/SQL コードの編集およびデバッグ
- SQL 文およびスクリプトの実行
- データの操作およびエクスポート
- レポートの作成および表示

SQL Developer により、標準の Oracle データベース認証を使用してターゲットである Oracle データベース・スキーマに接続できます。接続が完了すると、データベース内のオブジェクトに対する操作を実行できます。選択した MySQL、Microsoft SQL Server、Microsoft Access などのサード・パーティ製（Oracle 製品ではない）データベースのスキーマに接続し、それらのデータベース内のメタデータやデータを表示し、それらのデータベースを Oracle に移行できます。

SQL*Plus は、非定型データベース文を入力して実行するための基本的なコマンドライン・ツールです。SQL*Plus を使用すると、SQL 文と PL/SQL ブロックを実行したり、他の多数のタスクを実行できます。

> **関連項目：** これらのツールの詳細は、『Oracle Database SQL Developer ユーザーズ・ガイド』および『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。

自動メモリー管理

Oracle Database 11*g* リリース 1 から、Oracle データベースでシステム・グローバル領域（SGA）メモリーおよびインスタンス・プログラム・グローバル領域（PGA）メモリーを完全に自動的に管理できるようになりました。インスタンスで使用可能な合計メモリー・サイズのみを指定すれば、処理要求を満たすために、Oracle Database により SGA とインスタンス PGA 間で動的にメモリーが交換されます。この機能は、自動メモリー管理と呼ばれます。このメモリー管理モードでは、データベースにより個々の SGA コンポーネントおよび PGA のサイズも動的にチューニングされます。

> **関連項目：** 詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

自動ストレージ管理

自動ストレージ管理により、すべてのタイプのデータベース・ファイルの管理が自動化され、簡素化されます。データベース・ファイルは使用可能なすべてのディスク間で自動的に分散され、記憶域構成に変更があるたびにデータベース記憶域のバランスが自動的に調整されます。また、自動ストレージ管理により、データベース・ファイルのミラー化による冗長性が実現されます。

Oracle Database にはネットワーク・ファイル・システム（NFS）のサポートが標準で組み込まれており、NFS の OS サポートに依存しません。これにより、NFS を使用してアクセスされるネットワーク接続ストレージの管理性および診断性が向上します。

Automatic Database Diagnostic Monitor

Automatic Database Diagnostic Monitor（ADDM）を使用して、特定のインスタンスで取得した 1 対の自動ワークロード・リポジトリ（AWR）スナップショットによって定義された任意の時間帯のパフォーマンス解析を実行できます。分析はトップ・ダウンで行われるため、最初に兆候を識別し、次にパフォーマンス上の問題の根本的原因を突き止めます。ADDM は、システムで問題のない領域も示します。たとえば、システム・パフォーマンスに大きな影響を与えない待機イベント・クラスは、初期段階で識別され、チューニングの対象から除かれます。これにより、システム・パフォーマンス全体にほとんど、またはまったく影響を与えない項目にかける時間と労力を削減できます。

問題の診断に加えて、ADDM は可能な解決策を提案します。該当する場合、ADDM は複数の解決策を提案し、DBA はその中から選択できます。推奨事項の生成中、ADDM はハードウェアの変更、データベース構成の変更、スキーマ・オブジェクトの変更、アプリケーションの変更および他のアドバイザへの参照など様々な変更を考慮します。

> **関連項目：** Automatic Database Diagnostic Monitor の詳細は『Oracle Database 2 日でデータベース管理者』を、自動ワークロード・リポジトリの詳細は『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

SQL チューニング・アドバイザ

Oracle Database では、SQL チューニング・アドバイザと呼ばれるサーバー・ユーティリティを提供します。SQL チューニング・アドバイザは、1 つ以上の SQL 文を入力情報として取得し、自動 SQL チューニング・アドバイザを起動してその文の SQL チューニングを実行します。SQL チューニング・アドバイザの出力は、アドバイスまたは推奨事項の形式で、各推奨事項の原理およびその予想されるメリットも提供します。推奨事項は、オブジェクトの統計の収集、新しい索引の作成、SQL 文の再構築、SQL プロファイルの作成に関連しています。ユーザーは、SQL 文のチューニングを完了するために推奨事項を受け入れるかどうかを選択できます。

> **関連項目：** 詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

SQL アクセス・アドバイザ

SQL アクセス・アドバイザは、スキーマ変更の推奨事項を作成します。このアドバイザは、SQL 問合せを最適化するための索引やマテリアライズド・ビューなどのアクセス構造の作成を提案できます。また、問合せのパフォーマンスを改善するためのパーティション表、索引またはマテリアライズド・ビューも提案できます。

SQL アクセス・アドバイザは、SQL ワークロードを入力として使用します。ワークロードは、現在および最新の SQL アクティビティ、SQL リポジトリまたは開発環境によるユーザー定義のワークロードなどを含む様々なソースから選択できます。その後、アドバイザは、ワークロード全体のパフォーマンス向上のための変更を推奨します。

> **関連項目：** 詳細は、『Oracle Database 2 日でパフォーマンス・チューニング・ガイド』を参照してください。

ストリーム・チューニング・アドバイザ

Streams トポロジは、Streams 環境におけるデータベース、データベースに構成された Streams コンポーネントおよびそれらのコンポーネント間のメッセージ・フローを表現したものです。ストリーム・パフォーマンス・アドバイザは、スループットおよび待機時間の測定も含む Streams トポロジのパフォーマンス測定を報告します。ストリーム・パフォーマンス・アドバイザは、Streams トポロジのボトルネックも識別し、修正します。また、ストリーム・パフォーマンス・アドバイザは、Streams トポロジの Streams コンポーネントを調査し、パフォーマンス改善の方法を推奨します。

> **関連項目：** 詳細は、『Oracle Streams 概要および管理』を参照してください。

スケジューラ

Oracle Database には、管理タスクを簡素化し、複雑なスケジューリング・ニーズに対処する豊富な機能セットを提供するために、`DBMS_SCHEDULER` パッケージにファンクションとプロシージャのコレクションが用意されています。これらのファンクションは総称してスケジューラと呼ばれ、どの PL/SQL プログラムからでもコールできます。

スケジューラを使用すると、データベース管理者とアプリケーション開発者はデータベース環境で各種タスクが発生する時期と場所を制御できます。たとえば、データベース管理者は、バックアップや夜間のデータ・ウェアハウスのロードおよび抽出など、データベース・メンテナンス・ジョブをスケジュールして監視できます。

データベース・リソース・マネージャ

従来、Oracle データベースなど、システム上で実行される各種アプリケーション間のリソース管理は、オペレーティング・システムによって調整されていました。データベース・リソース・マネージャは、データベース内の実行スケジュールを制御することで、各種セッション間のリソースの分配を制御します。実行させるセッションとその実行時間を制御することで、データベース・リソース・マネージャでは、リソース分配とプラン・ディレクティブとを確実に一致させることができます（したがって、ビジネス目標とも一致します）。

> **関連項目：** データベース・リソース・マネージャの詳細は、第 14 章「管理性」を参照してください。

診断性機能の概要

Oracle Database 11*g* から、Oracle Database には、問題を防止、検出、診断および解決するための高度な障害診断性インフラストラクチャが含まれています。ターゲットとなる問題は、データベース・コードの不具合、メタデータの破損、顧客データの破損などが原因の重大なエラーです。インフラストラクチャの目標と、その目標を達成するための Oracle テクノロジの詳細は、14-6 ページの「障害診断性インフラストラクチャ」を参照してください。

データベースのバックアップおよびリカバリ機能の概要

どのようなデータベース・システムでも、システムやハードウェアの障害が発生する可能性があります。バックアップおよびリカバリ方針の目的は、障害によるデータ消失からデータベースを保護し、データ消失が発生した場合、データベースを再構築することです。

RMAN によるバックアップおよびリカバリとユーザー管理バックアップおよびリカバリ　データベースのバックアップは、すべてのバックアップおよびリカバリ方針の基礎となるものです。バックアップとは、データのコピーです。このコピーには、データファイル、制御ファイルおよびサーバー・パラメータ・ファイルなど、データベースの重要な部分を含めることができます。メディア・リカバリでは、リストアされたバックアップ・データファイルや個々のデータ・ブロックに、REDO ログまたは増分バックアップが適用されます。失われた変更を再適用することで、リカバリはバックアップをある時点にロールフォワードします。

バックアップおよびリカバリ方針の実装には、次のソリューションを適用できます。

- Recovery Manager（RMAN）。このツールは、Oracle データベース上で実行中のセッションと統合し、バックアップの履歴データの Recovery Manager リポジトリの維持など、様々なバックアップおよびリカバリ・アクティビティを実行します。RMAN は、コマンドラインまたは Enterprise Manager からアクセスできます。
- ユーザー管理バックアップおよびリカバリ。このソリューションでは、ホスト・オペレーティング・システム・コマンドと SQL*Plus リカバリ・コマンドを組み合せてバックアップおよびリカバリを実行します。

前述のソリューションはいずれも Oracle でサポートされており、完全にドキュメントに記載されていますが、RMAN はデータベースのバックアップおよびリカバリの推奨ソリューションです。RMAN は、複数のバックアップおよびリカバリの方法や機能へのアクセスなど、ユーザー管理バックアップおよびリカバリにはないものを提供しています。次に、最も顕著な機能を示します。

- 増分バックアップ
- ブロック・メディア・リカバリ
- 未使用ブロックの圧縮
- バイナリ圧縮
- 暗号化バックアップ

RMAN またはユーザー管理メソッドのいずれを使用した場合も、物理バックアップは、データ・ポンプ・エクスポート・ユーティリティで作成したスキーマ・オブジェクトの論理バックアップで補足できます。リストアおよびリカバリの後に、データ・ポンプ・インポートを使用してデータを再作成できます。

> **関連項目：** バックアップ方法の詳細は、15-6 ページの「Recovery Manager によるバックアップとユーザー管理バックアップ」を、データ・ポンプの詳細は、『Oracle Database ユーティリティ』を参照してください。

Oracle Flashback テクノロジ　Oracle のほとんどのフラッシュバック機能は論理レベルで稼働するため、データベース・オブジェクトの表示と操作が可能です。Oracle の論理レベルのフラッシュバック機能は RMAN に依存しません。そのため、RMAN がバックアップ方法の一部であるかどうかにかかわらず使用可能です。フラッシュバック・ドロップ以外の論理フラッシュバック機能は、各データベースの更新および更新により上書きされた値の結果レコードである、UNDO データに依存します。Oracle Database には、次の論理フラッシュバック機能が用意されています。

- Oracle Flashback Query
- Oracle Flashback Version Query
- Oracle Flashback Transaction Query
- Oracle Flashback Transaction
- Oracle Flashback Table
- Oracle Flashback Drop
- フラッシュバック・データ・アーカイブ

関連項目： これらの機能の詳細は、15-12 ページの「Oracle Flashback テクノロジ」を参照してください。

データ・リカバリ・アドバイザ　Oracle Database には、データ・リカバリ・アドバイザ・ツールが用意されています。このツールを使用すると、永続的なデータ障害が自動的に診断され、適切な修復オプションが提示されて、ユーザーのリクエストで修復が実行されます。データ・リカバリ・アドバイザは、Oracle のバックアップおよびリカバリ・ソリューションの単一のエントリ・ポイントを提供します。データ・リカバリ・アドバイザは、Enterprise Manager Database Control、Grid Control コンソールまたは RMAN コマンドライン・クライアントを介して使用できます。

関連項目：

- Oracle のバックアップおよびリカバリ方法の詳細は、第 15 章「バックアップおよびリカバリ」を参照してください。
- このツールの詳細は、15-11 ページの「データ・リカバリ・アドバイザ」を参照してください。

高可用性機能の概要

ほとんど常時に近い可用性を提供するように構成されたコンピューティング環境は、高可用性システムと呼ばれます。通常、この種のシステムには、障害が発生してもシステムを使用できるように冗長ハードウェアおよびソフトウェアがあります。適切に設計された高可用性システムでは、単一の障害箇所が回避されます。

Oracle Database には、計画されていた停止または計画外の停止の際に高可用性を提供する多数の製品と機能が用意されています。次の項で説明するこれらの機能を様々な組合せで使用して、特定の高可用性ニーズを満たすことができます。

Oracle Real Application Clusters　Oracle Real Application Clusters（Oracle RAC）では、Oracle Database がパッケージ・アプリケーションまたはカスタム・アプリケーションを複数のクラスタ・サーバーにわたって変わりなく実行できます。この機能により、最高レベルの可用性と非常に柔軟なスケーラビリティが実現されます。クラスタ・サーバーの 1 つが停止しても、Oracle Database は稼働中のサーバー上で稼働し続けます。処理能力を上げる必要がある場合には、データへのアクセスを妨げることなく、別のサーバーを追加できます。

Oracle Data Guard　Oracle Data Guard は、本番 Oracle データベースが障害、災害、エラーおよびデータ破損に対処できるように、1 つ以上のスタンバイ・データベースを作成、保守、管理および監視する包括的なサービスのセットを提供します。Data Guard では、これらのスタンバイ・データベースは、トランザクション一貫性のある本番データベースのコピーとして維持されます。計画停止または計画外停止で本番データベースが使用できなくなった場合、Data Guard により任意のスタンバイ・データベースが本番用に切り替えられるため、停止時間が大幅に短縮されます。

Oracle Streams　Oracle Streams を使用すると、データベース内またはデータベース間で、データ・ストリーム内のデータ、トランザクションおよびイベントを伝播させ、管理できます。Streams には一連の要素が用意されており、データ・ストリームに入れる情報、ノード間でのストリームのルーティング方法、各ノードに入った時のストリーム内でのイベントの処理およびストリームの終了方法を制御できます。

Oracle Flashback テクノロジ　フラッシュバック・テクノロジは、データの表示を切り替えて異なる時点でのデータの状態を参照できるように、一連の機能を提供します。フラッシュバック機能を使用すると、スキーマ・オブジェクトおよび履歴データの過去のバージョンを問い合せできます。また、変更分析やセルフ・サービス修復を実行して、データベースのオンライン中に論理破損からリカバリできます。フラッシュバック・テクノロジは Oracle Database に固有のもので、行、トランザクション、表、表領域、データベースを含むすべてのレベルでのリカバリをサポートします。

オンライン表再定義　Oracle Enterprise Manager には、スクリプトが自動的に生成されて表のオンライン再編成が実行される「オブジェクトの再編成」ウィザードが用意されています。ユーザーに表への完全なアクセスを許可しながら、すべての再定義プロセスが実行されます。

自動ストレージ管理　自動ストレージ管理（ASM）は、ファイルシステムとボリューム・マネージャを直接 Oracle カーネル内で垂直に統合します。ASM は使用可能なすべての記憶域にファイルを分散します。データ破損からデータを保護するために、ASM では SAME（Stripe And Mirror Everything）の概念が拡張され、ディスク・レベル全体ではなくデータベース・ファイル・レベルでのミラー化を可能にする柔軟性が拡張されています。ASM を使用する DBA は、**ディスク・グループ**と呼ばれる規模の大きなオブジェクトを作成および管理します。ディスク・グループは、1 つの論理単位として管理される一連のディスクを表します。基礎となるデータベース・ファイルのファイルの命名と配置を自動化することにより、DBA の時間を節約でき、標準のベスト・プラクティスに確実に準拠できます。

Recovery Manager　データベースのバックアップおよびリカバリを管理する Oracle Database のユーティリティです。RMAN は、要求されたバックアップ、リストアまたはリカバリ操作の最も効率的な実行方法を判断し、処理のためにこれらの操作を Oracle Database サーバーに発行します。Recovery Manager とサーバーは、データベース構造に対する変更を自動的に識別し、必要な操作を動的に調整して変更に対処します。

フラッシュ・リカバリ領域　フラッシュ・リカバリ領域は、Oracle Database 内のすべてのリカバリ関連ファイルおよびアクティビティに対する統合された記憶域の場所です。この機能を有効化すると、すべての RMAN バックアップ、アーカイブ・ログ、制御ファイルの自動バックアップおよびデータファイルのコピーが、指定されたファイルシステムまたは自動ストレージ管理のディスク・グループに自動的に書き込まれます。このディスク領域の管理は、RMAN およびデータベース・サーバーが行います。フラッシュ・リカバリ領域を使用することにより、テープへの書込みのボトルネックがなくなります。さらに、データベースのメディア・リカバリが必要な場合に、データファイルのバックアップをすぐに使用できます。

関連項目： 第 17 章「高可用性」

ビジネス・インテリジェンス機能の概要

この項では、次のビジネス・インテリジェンス機能について説明します。

データ・ウェアハウス

データ・ウェアハウスは、トランザクション処理ではなく問合せおよび分析用に設計されたリレーショナル・データベースです。通常は、トランザクション・データから導出された履歴データが格納されますが、他のソースからのデータも格納できます。データ・ウェアハウスにより、分析のワークロードがトランザクションのワークロードから分離され、組織は複数のソースからのデータを連結できます。

データ・ウェアハウス環境には、リレーショナル・データベースの他に、ETL ソリューション、オンライン分析処理（OLAP）エンジン、クライアント分析ツール、データを収集してビジネス・ユーザーに配信するプロセスを管理する他のアプリケーションがあります。

抽出、変換、ロード（ETL） データ・ウェアハウスをビジネス分析に役立てるには、データ・ウェアハウスに定期的にデータをロードする必要があります。データをロードするには、1 つ以上の業務系システムからデータを抽出し、ウェアハウスにコピーする必要があります。ソース・システムからデータを抽出してデータ・ウェアハウスに取り込む処理は、一般に **ETL** と呼ばれ、抽出、変換、ロードを表します。

データ・ウェアハウスでのビットマップ索引 索引を使用する目的は、特定のキー値が含まれる行へのポインタを提供することです。通常の索引では、そのキー値を持つ行に対応する各キーの ROWID のリストが格納されます。Oracle Database では、各 ROWID とともに各キー値が繰り返し格納されます。従来の B ツリー索引を使用して大きな表に完全な索引を作成すると、領域という面で膨大なコストがかかります。索引は表データの何倍にも膨れ上がることがあるからです。

ビットマップ索引では、ROWID のリストのかわりに各キー値のビットマップがデータベースに格納されます。それに対して、ビットマップ索引は通常、表中で索引を付けるデータのサイズより小さくてすみます。データ・ウェアハウス環境では、通常、大量のデータと非定型問合せを扱うものの、同時データ操作言語（DML）トランザクションのレベルは高くありません。この種のアプリケーションに対するビットマップ索引の利点は次のとおりです。

- 多くの種類の非定型問合せの応答時間が短縮されます。
- 他の方式の索引と比べて記憶域必要量が減少します。
- 比較的少数の CPU や小型のメモリーが搭載されたハードウェアであっても、パフォーマンスが劇的に向上します。
- パラレル DML およびロード中に効果的にメンテナンスできます。

さらに、通常、ディメンション表やファクト表の結合を含む一般的なデータ・ウェアハウスの問合せにおいて、ビットマップ結合索引は通常のビットマップ索引と同じ領域使用で、問合せのパフォーマンスを向上します。

マテリアライズド・ビュー

マテリアライズド・ビューは、問合せ結果を別々のスキーマ・オブジェクトに格納して、表データへのアクセスを提供します。通常のビューは記憶域を占めず、データも含まれていませんが、マテリアライズド・ビューには、1 つ以上の実表やビューに対する問合せで得られた行が含まれます。問合せが実表ではなくマテリアライズド・ビューにアクセスして実行されるため、問合せ応答時間は向上します。マテリアライズド・ビューの格納場所は、実表と同じデータベースでも、異なるデータベースでもかまいません。

マテリアライズド・ビューをその実表と同じデータベースに格納すると、**クエリー・リライト**を通じて問合せのパフォーマンスをさらに改善できます。クエリー・リライトは、実表ではなくマテリアライズド・ビューを使用するように SQL 問合せを自動的にリライトするメカニズムです。クエリー・リライトにより、開発者はマテリアライズド・ビューを活用するためにアプリケーションのコードを書き換える必要はありません。クエリー・リライトは、データ・ウェアハウス環境で特に役立ちます。

表の圧縮

Oracle では、すべてのタイプのデータ、バックアップおよびネットワーク通信量を、アプリケーションに対して透過的な方法で圧縮する、包括的なデータ圧縮機能が提供されています。これらの機能には、OLTP ワークロードをターゲットとした圧縮も含まれています。この圧縮の使用は、記憶域の使用量の削減および問合せのパフォーマンスを向上すると同時に、書込みのパフォーマンスのオーバーヘッドを最低限にできます。表の圧縮は、すべてのリレーショナル・データの圧縮に使用できます。構造化されていないコンテンツの圧縮には、SecureFiles 圧縮を使用します。重複除外により、SecureFiles データの冗長コピーを自動的に除外できます。RMAN バックアップの高速化のために、新しい、より高速な圧縮アルゴリズムが含まれるようになりました。これにより、ディスクの領域要件を減らすために、データ・ポンプ・エクスポートも圧縮できるようになりました。また、Data Guard での REDO データの圧縮により、ネットワーク通信量の削減および迅速なギャップ解決が可能になります。

関連項目： 5-7 ページの「表の圧縮」

パラレル実行

Oracle Database で SQL 文をパラレル実行すると、同時に多数のプロセスが連携して 1 つの SQL 文を実行します。文の実行に必要なタスクを多数のプロセスで分割して実行すると、Oracle Database では、1 つのプロセスで実行する場合と比較して、より高速に実行できます。これを**パラレル実行**または**パラレル処理**と呼びます。パラレル実行では、大規模データベースでの、データ処理集中型の操作における応答時間が大幅に短縮されます。

分析 SQL

Oracle Database には、データベースで分析操作を実行するための多数の SQL 操作があります。これには、ランキング、移動平均、累計、対比レポート、期間対比などがあります。

OLAP 機能

Oracle オンライン分析処理（OLAP）は、複数のディメンション間でのデータ分析において、システム固有の多次元記憶域を提供し、思考スピードと同じ速さの応答時間を実現します。データベースは、時系列計算、予測、加算および非加算演算子による高度な集計および割当演算子などの豊富な機能で分析をサポートします。この機能により、Oracle データベースはビジネス・インテリジェンスのすべての領域と高度な分析アプリケーションをサポートする、完全な分析プラットフォームとなります。Oracle OLAP はデータベースに完全に統合されているため、標準的な SQL の管理、問合せおよびレポート・ツールを使用できます。

データ・マイニング

Oracle Data Mining では、データがデータベースを出ることはありません。データ、データ準備、モデル構築およびモデルのスコアリングの結果は、すべてデータベースに残ります。これにより、Oracle Database はアプリケーション開発者に、データ・マイニングをデータベース・アプリケーションとシームレスに統合するためのインフラストラクチャを提供します。通常、データ・マイニングは、コール・センター、ATM、E-Business リレーショナル管理（ERM）およびビジネス・プランニングなどで使用されます。Oracle Data Mining では、PL/SQL API、Java API、モデルのスコアリングを行う SQL 関数、および Oracle Data Miner と呼ばれるグラフィカル・ユーザー・インタフェースをサポートします。

> **関連項目：** Oracle Data Mining の詳細は、第 16 章「ビジネス・インテリジェンス」を参照してください。

大規模データベース（VLDB）

パーティション化は、大規模データベース（VLDB）の管理のための重要な機能です。パーティション化が扱う基本的な課題は、データの増加です。また、パーティション化によりデータベースは大きいデータセットへのスケーラビリティを持つと同時に、管理リソースやハードウェア・リソースを過度に増やすことなく、一貫性のあるパフォーマンスを維持します。パーティション化により、表、索引または索引構成表を**パーティション**と呼ばれる小さな単位に分割できます。SQL DML 文を使用したパーティション表にアクセスする場合、アプリケーションの変更は不要です。

パーティション化により可用性、管理性およびパフォーマンスが改善するため、様々なアプリケーションにとって大きな利点となります。

情報ライフサイクル管理（ILM）情報ライフサイクル管理（ILM）とは、その有効なライフサイクルを通じてデータを管理するための一連のプロセスおよびポリシーです。ILM ソリューションを実装することの利点の 1 つは、適切な記憶域の階層を活用すると同時に、業務や規制への準拠に必要なデータを維持して、コストを削減することです。パーティション化は、ILM ソリューションをデータベース内に実装するための機能です。

> **関連項目：** VLDB の詳細は、第 18 章「大規模データベース（VLDB）」を参照してください。

コンテンツ管理機能の概要

Oracle Database には、XML、テキスト、オーディオ、ビデオ、イメージ、医用画像および空間など、豊富な各種コンテンツをすべて処理するためのデータ型が組み込まれています。これらのデータ型は、データベースではシステム固有の型として表示されます。これらは、いずれも SQL を使用して問い合せることができます。これらのデータ型のいずれかまたはすべてに属するデータを、1 つの SQL 文に含めることができます。

この項の内容は、次のとおりです。

- Oracle Database の XML
- LOB
- SecureFiles
- Oracle Text
- Oracle Ultra Search
- Oracle Multimedia
- Oracle Spatial

Oracle Database の XML

Oracle XML DB は、高パフォーマンスの XML の格納と取得に関連する一連の Oracle Database テクノロジです。SQL データ・モデルと XML データ・モデルの相互運用が可能な方法でサポートするシステム固有の XML を提供します。Oracle XML DB の機能は、次のとおりです。

- World Wide Web Consortium（W3C）XML、XML スキーマ・モデルと、XML をナビゲートおよび問い合せる標準的なアクセス方法のサポート。データ・モデルは、Oracle Database に取り込まれます。
- SQL によるアクセス中の、XML データの格納、問合せ、更新および変換の実行。
- SQL データに対する XML 操作の実行。
- ファイル / フォルダ /URL 形式を使用して XML などのデータベース・コンテンツを整理および管理するための、単純で負荷の少ない XML リポジトリ。
- XML データを格納および管理するための、格納形式、コンテンツおよびプログラム言語に依存しないインフラストラクチャ。このインフラストラクチャは、データベース内に格納された XML コンテンツをナビゲートおよび問い合せる新しい方法を提供します。たとえば、Oracle XML DB リポジトリは XML ドキュメント階層を管理することにより、この操作を容易にします。
- 業界標準の XML へのアクセスおよび更新方法。この標準には、W3C XPath 推奨事項および ISO-ANSI SQL/XML 標準が含まれます。FTP、HTTP(S) および WebDAV を使用して XML コンテンツを Oracle Database との間で移動できます。業界標準の API により XML コンテンツへの Java、C および PL/SQL を使用したプログラム的なアクセスと操作が可能になります。
- XML 固有のメモリー管理と最適化。
- 信頼性、可用性、スケーラビリティおよびセキュリティなど、企業レベルの Oracle Database 機能の XML コンテンツへの実装。

Oracle XML DB は、Oracle XML Developer's Kit（XDK）とともに使用し、Oracle Application Server または Oracle Database のいずれかの中間層で実行するアプリケーションを構築できます。

LOB

LOB データ型 `BLOB`、`CLOB`、`NCLOB` および `BFILE` を使用すると、非構造化データ（テキスト、グラフィック・イメージ、ビデオ・クリップおよびサウンド・ウェーブなど）の大きなブロックを、バイナリ形式または文字形式で格納し、操作できます。このデータ型を使用すると、個々のデータに対する効率的なランダム・アクセスが可能です。

> **関連項目：** SecureFiles LOB の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

SecureFiles

SecureFiles は Oracle Database 11*g* の新機能であり、イメージ、オーディオ、ビデオ、PDF およびスプレッドシートなどのファイルのコンテンツの格納に最適なソリューションを提供します。従来は、リレーショナル・データはデータベースに格納され、半構造化のものを含む構造化されていないコンテンツはファイルとしてファイル・システムに格納されていました。SecureFiles は、ファイルの記憶域の選択における主要なパラダイム・シフトです。SecureFiles は、Oracle Database の利点を保持しながら、従来のファイル・システムと比べ、ファイル・データの配布を高速化するように設計されています。SecureFiles は非構造化コンテンツの格納に最適なデータベース・システムおよびファイル・システムのアーキテクチャ属性を提供します。

テクノロジの主要な利点　SecureFiles には、通常はハイエンドなファイル・システムに見られる次のような拡張機能が含まれています。

- **重複除外 :** Oracle Database は、複数の同じ SecureFiles データを自動的に検出し、コピーを 1 つのみ格納して、記憶域を節約できます。コピーを 1 つのみ格納するだけでなく、SecureFiles は他の重複データへの参照も維持します。重複除外はアプリケーションに対して完全に透過的であるため、記憶域の管理の簡略化に加えて、特にコピー操作においてパフォーマンスが大幅に改善します。重複の検出は LOB セグメントで行われます。`lob_storage_clause` により、パーティション・レベルの重複除外の指定が可能となります。そのため、パーティション化された SecureFiles 列では、パーティションまたはサブパーティションにまたがる重複の検出は行われません。

  関連項目 : 重複除外の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

  SecureFiles の重複除外は、**Advanced Compression** オプションの一部です。

- **圧縮 :** SecureFiles データは業界標準の圧縮アルゴリズムを使用して圧縮されています。圧縮によって、記憶域が大幅に節約されるだけでなく、I/O、バッファ・キャッシュ要件、REDO 生成、および暗号化のオーバーヘッドを削減することでパフォーマンスが向上します。圧縮により記憶域が節約されない場合、またはデータがすでに圧縮されている場合、SecureFiles はその列の圧縮を自動的にオフにします。圧縮はサーバー側で行われ、SecureFiles データへのランダムな読取りと書込みを可能にします。SecureFiles は、様々な圧縮の度合い（MEDIUM（デフォルト）および HIGH）を提供します。圧縮の度合いは、記憶域の節約と待機時間との間のトレードオフとなります。

  関連項目 : 圧縮の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

  SecureFiles の圧縮は、**Advanced Compression** オプションの一部です。

- **暗号化 :** Oracle Database 11*g* では、Oracle は SecureFiles への暗号化機能を拡張し、透過的データ暗号化（TDE）構文を使用するようになりました。Oracle Database は、表内のすべての SecureFiles 列の自動キー管理をサポートし、データ、バックアップ、REDO ログ・ファイルを透過的に暗号化および複合化します。アプリケーションでは何も変更せずに、TDE セマンティクスを使用した Oracle Database 11*g* SecureFiles の利点を活用できます。SecureFiles では、次の暗号化アルゴリズムをサポートしています。
  - 3DES168: サイズが 168 ビットの鍵を使用する Triple Data Encryption Standard
  - AES128: サイズが 128 ビットの鍵を使用する Advanced Encryption Standard
  - AES192: サイズが 192 ビットの鍵を使用する Advanced Encryption Standard（デフォルト）
  - AES256: サイズが 256 ビットの鍵を使用する Advanced Encryption Standard

  関連項目 : 暗号化の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

  SecureFiles の暗号化は、**Advanced Security** オプションの一部です。

- **ファイル・システムに類似したロギング :** 現在のファイル・システムでは、ファイル・システム・メタデータの実行中のログを記録できます。メタデータはジャーナルと呼ばれる実行中のログに記録されます。ジャーナルは、緩いペースでフラッシュされるため、パフォーマンスが向上し、`fsck` などのファイル・システム・チェック操作の実行が不要になります。SecureFiles のファイル・システムに類似したロギングにより、従来と変わらぬ高いパフォーマンスのジャーナル記録を実行できます。また、ファイル・システムに類似したロギングではソフトな破損が許可されるため、ブロック内にエラーが発見された場合、SecureFiles はブロックに LOB で埋めた文字列を戻します。これにより、アプリケーションは既知の無効なデータを参照してエラーを検知でき、LOB の削除（以前の LOB 実装では不可能だった処理）または他の手段でリカバリを実行できます。

前述のファイル・システムの拡張機能に加え、SecureFiles では Oracle Database の次のような拡張機能を利用できます。

- トランザクション、読取り一貫性、およびフラッシュバック
- LOB インタフェースとの 100% の下位互換性
- 読取り可能なスタンバイ、一貫性バックアップおよびポイント・イン・タイム・リカバリ
- ファイングレイン監査およびラベル・セキュリティ
- XML 索引、XML 問合せおよび XPath
- Oracle Real Application Clusters
- 自動ストレージ管理
- パーティション化と ILM
- メタデータおよびファイルのコンテンツの検索

高パフォーマンス　SecureFiles は、徹底的な高パフォーマンスとスケーラビリティを目指し設計されています。SecureFiles は、基本的な読取りおよび書込み操作では、ファイル・システムと同様のパフォーマンスを実現します。SecureFiles で最適化されたアルゴリズムは、LOB の 10 倍のパフォーマンスを提供します。SecureFiles はファイル・システムの機能を超えるスケーラビリティを実現します。組織では、大規模な SMP システムを使用したスケールアップ、または 1 つのシステム・イメージを維持しながら Oracle Real Application Clusters で何百ものコンピュータを使用したスケールアウトのいずれも可能です。CPU とディスクのスケーリングは、独立して透過的に実行できます。Oracle Database 11*g* を使用して、組織はペタバイト、エクサバイト級のすべてのタイプのコンテンツを格納できます。

Oracle Text

Oracle Text は、文書またはテキスト形式のコンテンツに索引を付けることで、情報の高速で正確な検索を実現します。Oracle Text を使用すると、テキスト検索と通常のデータベース検索を 1 つの SQL 文にまとめることができます。テキスト形式のコンテンツ、メタデータまたは属性に基づいて文書を検索できるため、Oracle Database がすべてのデータ管理の単一の統合ポイントとなります。

Oracle Text の SQL API は単純で直感的であり、アプリケーション開発者と DBA はテキスト索引を作成、メンテナンスしてテキスト検索を実行できます。

Oracle Ultra Search

Oracle Ultra Search を使用すると、Web サイト、データベース表、ファイル、宛先リスト、Oracle Application Server Portals およびユーザー定義データ・ソースに索引を付けて検索できます。この検索機能では、Oracle Ultra Search を使用して各種の検索アプリケーションをビルドできます。

Oracle Multimedia

Oracle Multimedia はイメージ、オーディオおよびビデオの開発を簡略化する多数のサービスを提供します。従来の他のリレーショナル・データと同様に、Oracle Multimedia のオブジェクトは、表の列としてアクセスされます。マルチメディア・コンテンツは、データベース内で内部的に保存および管理することも、データベース内のコンテンツへの参照を保存することで外部的に保存および管理することもできます。Java および PL/SQL API を使用すると、メタデータの抽出、イメージ形式の変換、サムネイル・イメージの生成が可能で、アプリケーションの開発とメンテナンスのコストが大幅に削減されます。Oracle JDeveloper、Application Express および Oracle Application Server Portal などのアプリケーション開発ツールと統合することにより、アプリケーション開発者は、メディアを駆使したアプリケーションを簡単に作成およびメンテナンスできます。また、Oracle Multimedia には、シングル・フレームおよびマルチフレームのイメージ、波形、3D ボリュームのスライス、ビデオ・セグメント、構造化レポートなどの Digital Imaging and Communications in Medicine（DICOM）のコンテンツと同様のサポートが用意されています。

関連項目： Oracle Multimedia の詳細は、第 19 章「コンテンツ管理」を参照してください。

Oracle Spatial

Oracle Database には組込みの空間機能があり、位置コンテンツ（資産、ビル、道路、地区、営業地域など）を格納し、索引を付けて管理し、データベースの機能を使用して位置関係を問い合せることができます。Oracle Spatial オプションによって、線形参照のサポートや座標系などの拡張空間機能が追加されます。

関連項目： Oracle Spatial の詳細は、第 19 章「コンテンツ管理」を参照してください。

セキュリティ機能の概要

Oracle Database には、データベースへのアクセス方法と使用方法を制御するセキュリティ機能が用意されています。次のような理由から、セキュリティ・メカニズムが必要です。

- データベースへの不正なアクセスの防止
- スキーマ・オブジェクトへの不正なアクセスの防止
- ユーザー・アクションの監査

スキーマは、各データベース・ユーザーに、そのユーザーと同じ名前によって対応付けられます。デフォルトでは、各データベース・ユーザーは、対応するスキーマ内のすべてのオブジェクトを作成およびアクセスする権利を持っています。

データベース・セキュリティは**システム・セキュリティ**と**データ・セキュリティ**の 2 つのカテゴリに分類できます。

システム・セキュリティによって、システム・レベルにおけるデータベースのアクセスと使用を制御できます。システム・セキュリティ・メカニズムでは、ユーザーがデータベースへの接続を認可されているかどうか、データベース監査がアクティブになっているかどうかと、ユーザーが実行できるシステム操作がチェックされます。たとえば、次のものが含まれます。

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

データ・セキュリティによって、スキーマ・オブジェクト・レベルにおけるデータベースのアクセスと使用を制御できます。たとえば、データ・セキュリティにより、次の事項が決定します。

- 特定のスキーマ・オブジェクトにアクセスできるユーザーと、そのスキーマ・オブジェクトに対して各ユーザーが許可されている特定のタイプのアクション（たとえば、ユーザー `SCOTT` は `employees` 表に対して `SELECT` 文と `INSERT` 文を発行できますが、`DELETE` 文は発行できません）。
- 各スキーマ・オブジェクトに対して監査されるアクション。
- 無許可ユーザーが Oracle Database を介さずにデータにアクセスするのを防ぐためのデータ暗号化。

セキュリティのメカニズム

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

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

- ネットワーク、データベースおよびアプリケーションを使用してエンティティの識別を検証するための認証
- ユーザーの識別とロールにリンクされているアクセスと処理および制限事項を限定するための認可処理
- 表または行と同様にオブジェクトに対するアクセス制限
- セキュリティ・ポリシー
- データベース監査

> **関連項目：** セキュリティ・メカニズムの詳細は第 20 章「データベース・セキュリティ」を参照してください。

データ整合性とトリガーの概要

データは、データベース管理者やアプリケーション開発者によって決められたとおり、特定のビジネス・ルールを遵守する必要があります。たとえば、ビジネス・ルールによって、`INVENTORY` 表の `sale_discount` 列には 9 よりも大きな数値を入れないように指定されている場合を考えます。`INSERT` 文や `UPDATE` 文がこの整合性規則に違反しようとすると、Oracle Database はその無効な文を取り消し、アプリケーションにエラーを戻します。Oracle Database は、データ整合性規則を管理するために、整合性制約とデータベース・トリガーを提供します。

> **注意：** データベース・トリガーによって整合性規則を定義または規定できますが、データベース・トリガーが整合性制約と同じ機能を持つわけではありません。特に、データベース・トリガーは、すでに表にロードされているデータをチェックしません。したがって、整合性制約によって整合性規則を規定できない場合にのみ、データベース・トリガーを使用することをお薦めします。

この項の内容は、次のとおりです。

- 整合性制約
- トリガー

整合性制約

整合性制約は、表の列に対してビジネス・ルールを定義する宣言の方法です。整合性制約は表のデータに関する文であり、常に次の規則に従って機能します。

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

整合性制約は表の定義の一部としてデータ・ディクショナリに格納されるため、すべてのデータベース・アプリケーションは同じ一連の規則を遵守する必要があります。規則が変更されても、整合性制約はデータベース・レベルで一度変更すればよいため、アプリケーションごとに何度も変更する必要はありません。**キー**とは、特定タイプの整合性制約の定義に含まれる列または列の集合です。キーによって、リレーショナル・データベースの異なる表と列の間の関連が示されます。キーの中にある個々の値を、**キー値**と呼びます。

Oracle Database がサポートする整合性制約は次のとおりです。

- **NOT NULL** 制約は、表の列に NULL（空のエントリ）を許可しません。
- **一意**制約は、列（または列の集合）に重複値を許可しません。**一意キー**は、一意制約の定義に含まれる列（または列の集合）です。
- **主キー**制約は、列（または列の集合）に重複値と NULL を許可しません。**主キー**は、表の主キー制約の定義に含まれる列（または列の集合）です。主キーの値は、表内の各行を一意に識別します。主キーは、各表に 1 つのみ定義できます。
- **外部キー**制約は、参照整合性制約とも呼ばれ、列または列の集合の値が、それぞれ別の表の一意キーまたは主キーの値と一致している必要があります。外部キー制約は、参照データが変更された場合に依存データを処理する方法を Oracle Database に指示する、参照整合性アクションも定義します。**外部キー**は、外部キー制約の定義に含まれる列（または列の集合）です。**参照キー**は、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です。
- チェック制約は、制約の論理式を満たさない値を許可しません。

> **関連項目：** 整合性制約の詳細は、第 21 章「データ整合性」を参照してください。

トリガー

トリガーとは、PL/SQL、Java または C 言語で記述され、表またはビューが変更された場合や、なんらかのユーザー・アクションまたはデータベース・システム・アクションが発生した場合に暗黙的に実行（起動）されるプロシージャです。

トリガーは Oracle Database 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。

> **関連項目：** トリガーの詳細は、第 22 章「トリガー」を参照してください。

情報統合機能の概要

分散環境とは、相互にシームレスに通信する様々なシステムで構成されるネットワークです。分散環境では、各システムをノードと呼びます。ユーザーが直接接続するシステムをローカル・システムと呼びます。このユーザーがアクセスする他のシステムをリモート・システムと呼びます。分散環境では、アプリケーションからローカル・システムやリモート・システムのデータにアクセスして相互に交換できます。すべてのデータに同時にアクセスして変更できます。

この項の内容は、次のとおりです。

- 分散 SQL
- Oracle Streams
- Oracle Database Gateway と Generic Connectivity

分散 SQL

同一機種分散データベース・システムとは、1 つ以上のコンピュータに常駐する複数の Oracle データベースで構成されるネットワークです。分散 SQL を使用すると、アプリケーションとユーザーは、1 つのデータベースにアクセスまたは変更する場合と同じくらい容易に、複数のデータベースにあるデータを同時にアクセスまたは変更できます。

Oracle 分散データベース・システムは、1 つの Oracle データベースであるかのように見せることができます。企業は、この分散 SQL 機能を使用して、すべての Oracle データベースを 1 つであるかのように見せることで、分散システムの複雑さを軽減できます。

Oracle Database は、データベース・リンクを使用して、あるデータベース上のユーザーがリモート・データベース内のオブジェクトにアクセスできるようにします。ローカル・ユーザーは、リモート・データベースのユーザーでなくても、リモート・データベースへのリンクにアクセスできます。

位置の透過性 アプリケーションとユーザーがデータの物理的な位置を意識しないですむ（透過的である）とき、位置の透過性が実現されます。たとえば、複数のデータベースの表データを結合するビューによって、位置の透過性が実現されます。つまり、この場合、ビューのユーザーはデータの物理的な位置を知る必要がありません。

SQL とトランザクションの透過性 Oracle Database は、問合せ、更新およびトランザクションの透過性を提供します。たとえば、SELECT、INSERT、UPDATE および DELETE などの標準 SQL 文は、非分散データベース環境の場合と同様にデータを操作します。また、アプリケーションでは標準 SQL 文 COMMIT、SAVEPOINT および ROLLBACK を使用してトランザクションを制御します。Oracle Database では、分散トランザクションのデータ整合性が 2 フェーズ・コミット・メカニズムを使用して保証されるため、分散システム内のすべてのノードにトランザクションをコミットするように指示できます。この要求を満たせない場合は、すべてのノードでトランザクションがロールバックされます。

> **関連項目：** 2 フェーズ・コミット・メカニズムの詳細は、『Oracle Database 管理者ガイド』を参照してください。

分散問合せの最適化 分散問合せの最適化では、コスト・ベースの最適化を使用して、リモート表から必要なデータのみを抽出する SQL 式を検索または生成し、そのデータをリモート・サイト（または場合によってはローカル・サイト）で処理し、結果を最終処理のためにローカル・サイトに送信します。この操作により、すべての表データをローカル・サイトに転送して処理する場合に比べて、同じ所要時間に必要なデータ転送量が減少します。

Oracle Streams

Oracle Streams を使用すると、データベース内またはデータベース間で、データ・ストリーム内のデータ、トランザクションおよびイベントを伝播させ、管理できます。ストリームにより、パブリッシュされた情報がサブスクライブした宛先に送付されます。

Oracle Streams では、ストリームに挿入する情報、ノード間でのストリームのフローまたはルーティング方法、各ノードで発生するストリーム内のイベント、およびストリームの終了方法を制御できます。ストリームで動作する要素の構成を指定することで、メッセージ・キューイングやデータ・レプリケーションなど、特定の要件に対処できます。

取得 Oracle Streams は、イベントを暗黙的および明示的に取得してステージング領域に置きます。DML 操作や DDL 操作などのデータベース・イベントは、REDO ログ・ファイルのマイニングによって暗黙的に取得されます。洗練されたサブスクリプション規則により、どのイベントを取得するかを決定できます。

ステージング ステージング領域は、取得したイベントを格納し管理するキューです。データベース表に対する変更は論理変更レコード（LCR）としてフォーマットされ、サブスクライバが使用するまでステージング領域に格納されます。ステージングは、LCR の安全な保管場所を提供し、LCR データの監査および追跡をサポートします。

使用 ステージング領域に置かれたメッセージは適用エンジンで使用され、そこで変更がデータベースに適用されるか、アプリケーションで使用されます。柔軟な適用エンジンにより、標準またはカスタムの適用ファンクションを使用できます。明示的デキューがサポートされるため、アプリケーション開発者は Oracle Streams を使用してメッセージを確実に交換できます。また、データへの変更の適用も通知できます。

メッセージ・キューイング Oracle Streams Advanced Queuing は、Oracle Streams の柔軟なインフラストラクチャの上にビルドされています。このコンポーネントは、イベント処理用の統一フレームワークを提供します。アプリケーションやワークフローで生成されたイベント、または REDO ログやデータベース・トリガーから暗黙的に取得されたイベントを、1 つのキューで取得できます。これらのイベントは、様々な方法で使用できます。ユーザー定義ファンクションまたはデータベース表の操作で自動的に適用するか、明示的にデキューするか、使用側アプリケーションに通知を送信できます。これらのイベントは、どの段階でも変換できます。使用側アプリケーションが異なるデータベースにある場合、イベントは適切なデータベースに自動的に伝播します。このようなイベントに対する操作を自動的に監査でき、ユーザー指定の期間のみ履歴を保持できます。

データ・レプリケーション レプリケーションとは、データベース・オブジェクトを複数のデータベース内で維持することです。Oracle Streams には強力なレプリケーション機能が用意されており、この機能を使用して分散オブジェクトの複数コピーの同期を維持できます。

どの情報が関連しているかは自動的に判別され、その情報を必要としているユーザーと共有されます。このアクティブな情報共有には、DML によるデータ変更を含むデータベースでのイベントの取得と管理、および他のデータベースやアプリケーションへのイベントの伝播が含まれます。データ変更には、レプリカ・データベースに直接適用する方法と、ユーザー定義プロシージャをコールして宛先データベースで代替処理を実行する方法があります。たとえば、データ・ウェアハウスのロードに使用したステージング表を移入できます。

Oracle Streams はオープンな情報共有ソリューションであり、Oracle と Oracle 以外のシステム間での異機種間レプリケーションをサポートしています。Transparent Gateway を使用すると、Oracle データベースで実行された DML 変更を Oracle 以外のプラットフォームに適用できます。

Oracle Streams は、マテリアライズド・ビューと完全に相互運用可能です。マテリアライズド・ビューは、更新可能または読取り専用の、データの特定時点のコピーを維持できます。マテリアライズド・ビューには、値ベースの選択基準と一致するマスター表全体のコピー、またはその各行の定義済サブセットが含まれます。多重化マテリアライズド・ビューも使用でき、この場合は 1 つのマテリアライズド・ビューが別のマテリアライズド・ビューのサブセットとなります。マテリアライズド・ビューは、トランザクション一貫性を持つバッチ更新を介して、対応するマスター表から定期的に更新またはリフレッシュされます。

Oracle Database Gateway と Generic Connectivity

Oracle Database Gateway と Generic Connectivity により、Oracle 分散データベース機能が Oracle 以外のシステムへと拡張されます。Generic Connectivity は汎用ソリューションですが、Oracle Database Gateway は調整されたソリューションであり、特定の Oracle 以外のシステム向けに特別にコーディングされています。Oracle Database は、Oracle 以外のデータ・ソース、Oracle 以外のメッセージ・キューイング・システムおよび非 SQL アプリケーションで動作でき、他のベンダーの製品およびテクノロジとの相互運用性が保証されます。

Oracle Database Gateway と Generic Connectivity は、分散 SQL を使用する場合は同期アクセスに、Oracle Streams を使用する場合は非同期アクセスに使用できます。Oracle Streams 環境に Transparent Gateway を導入することで、Oracle データベースから Oracle 以外のデータベースへのデータ・レプリケーションが可能になります。

Oracle Database Gateway と Generic Connectivity は、サード・パーティの SQL 言語、データ・ディクショナリおよびデータ型を Oracle Database フォーマットに変換し、Oracle 以外のデータ・ストアをリモート Oracle データベースのように取り扱えるようにします。これらの機能により、企業は各種システムをシームレスに統合し、全社的に連結されたビューを提供できます。

関連項目： 第 23 章「情報の統合」

Oracle Database アプリケーション開発

SQL と PL/SQL は、Oracle Database アプリケーション開発スタックの中核となります。

- ほとんどの企業のバックエンドで SQL を実行
- データベースにアクセスする Web アプリケーションも SQL（Java クラスにより JDBC としてラップ）を使用
- エンタープライズ・アプリケーション統合アプリケーションは SQL 問合せから XML を生成
- コンテンツ・リポジトリは SQL 表の最上位に作成

SQL と PL/SQL は、単純で広く知られた統一データ・モデルを提供します。SQL と PL/SQL は多数のアプリケーションでスタンドアロンとして使用されますが、Java（JDBC）、Oracle Call Interface（OCI）、Oracle C++ Call Interface（OCCI）または XSU（XML SQL Utility）からも直接コールできます。ストアド・パッケージ、プロシージャおよびトリガーは、いずれも PL/SQL または Java で記述できます。

この項の内容は、次のとおりです。

Oracle SQL の概要

Structured query language（**SQL**）は、データベースを定義して操作するためのプログラミング言語です。SQL データベースはリレーショナル・データベースです。これは、データが一連の単純なリレーションに基づいて格納されるということです。

SQL 文

Oracle データベース内の情報の操作はすべて、SQL 文を使用して実行されます。SQL 文は、SQL テキストの文字列です。次のように、文は、完全な SQL 文と等価である必要があります。

```
SELECT last_name, department_id FROM employees;
```

> **注意：** SQL 文の終了の表現は、プログラミング環境により異なります。このマニュアル・セットでは、デフォルトの SQL*Plus 文字列であるセミコロン（;）を使用します。

正常に実行できるのは完全な SQL 文のみです。次のような不完全文を実行しようとすると、テキストの不足を示すエラーが発生します。

```
SELECT last_name
```

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

データ定義言語（DDL）文は、スキーマ・オブジェクトを作成、変更、維持および削除します。DDL 文には、データベースやデータベース内の特定のオブジェクトにアクセスする権限を、あるユーザーから他のユーザーに付与するための文も含まれます。

データ操作言語（DML）文は、データを操作します。表の行の問合せ、挿入、更新および削除はすべて DML 操作です。最も使用頻度の高い SQL 文は、`SELECT` 文です。この文によって、データベースからデータを取り出します。表やビューのロックや SQL 文の実行計画の検討も DML 操作です。

トランザクション制御文は、DML 文によって行われる変更を管理します。これによって、ユーザーはいくつかの変更をグループ化して論理トランザクションを作成できます。この文に含まれるのは、`COMMIT`、`ROLLBACK` および `SAVEPOINT` などです。

セッション制御文を使用すると、ユーザーは自分のカレント・セッションのプロパティを制御できます。つまり、ロールを使用可能や使用禁止にしたり、言語設定を変更できます。セッション制御文には、`ALTER SESSION` および `SET ROLE` の 2 つがあります。

システム制御文は、Oracle データベース・インスタンスのプロパティを変更します。`ALTER SYSTEM` は唯一のシステム制御文です。この文は、共有サーバーの最小数などの設定値の変更のために使用します。また、セッションの停止やその他のシステム全体のタスクも実行できます。

埋込み SQL 文は、DDL 文、DML 文およびトランザクション制御文を、Oracle プリコンパイラとともに使用されるプログラムなどの手続き型言語プログラムに取り込んだものです。この文に含まれるのは、`OPEN`、`CLOSE`、`FETCH` および `EXECUTE` などです。

関連項目： SQL の詳細は、第 24 章「SQL」を参照してください。

PL/SQL の概要

PL/SQL は、SQL に対する Oracle の手続き型言語拡張機能です。PL/SQL では、SQL が持つ平易さと柔軟性に、`IF ... THEN`、`WHILE`、`LOOP` などのルーチンを含む、構造化プログラミング言語の手続き型機能が結合されています。

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

- PL/SQL コードは、データベースに格納できます。アプリケーションとデータベース間のネットワークの通信量が軽減されるため、アプリケーションとシステムのパフォーマンスが向上します。PL/SQL がデータベースに格納されていない場合でも、アプリケーションは個々に SQL 文をデータベースに送信するかわりに、PL/SQL のブロックを送信できます。そのため、ネットワーク通信量が減少します。
- PL/SQL コードのシステム固有のコンパイルは非常に簡単で、パフォーマンスが大幅に改善します。
- データ・アクセスを PL/SQL コードによって制御できます。別のアクセス・ルートが与えられないかぎり、PL/SQL のユーザーはアプリケーション開発者が意図したとおりにのみデータにアクセスできます。
- Oracle は PL/SQL Server Pages をサポートしているため、アプリケーションのロジックを Web ページから直接起動できます。

次の項では、データベース内に定義し、一括して格納できる PL/SQL プログラム・ユニットについて説明します。

プロシージャと**ファンクション**は、一連の SQL 文および PL/SQL 文です。これらの文は、特定の問題を解決したり一連の関連タスクを実行することを目的として、1 つの単位としてグループ化されています。プロシージャとファンクションは、作成後にコンパイル済の形式でデータベース内に格納され、ユーザーまたはデータベース・アプリケーションによって実行されます。プロシージャとファンクションは同じです。ただし、ファンクションはユーザーに必ず 1 つの値を戻します。これに対して、プロシージャは値を戻しません。

パッケージによって、関連したプロシージャ、ファンクション、変数およびその他の構成メンバーをカプセル化し、データベースの単位としてまとめて格納できます。パッケージによって、機能性が拡大されます。たとえば、グローバル・パッケージ変数を宣言して、パッケージ内のプロシージャで使用できます。パッケージのすべてのオブジェクトを一度に解析およびコンパイルし、メモリーにロードできるため、パフォーマンスも改善できます。

関連項目： PL/SQL の詳細は、第 24 章「SQL」を参照してください。

Java の概要

Java は、アプリケーション・レベルのプログラムに効果を発揮するオブジェクト指向のプログラミング言語です。Oracle Database にはあらゆるタイプの JDBC ドライバが用意されており、Java アプリケーションからのデータベース・アクセスを拡張します。Java ストアド・プロシージャはアクセス制御の面で移植可能でセキュアであり、Java 以外の従来型アプリケーションでも Java を透過的に起動できます。また、Java コードのシステム固有のコンパイルは非常に簡単で、パフォーマンスが大幅に改善します。

関連項目： Java の詳細は、25-19 ページの「Java の概要」を参照してください。

アプリケーション・プログラミング言語（API）の概要

Oracle Database の開発者は、アプリケーション開発に使用する言語を C、C++、Java、COBOL、PL/SQL、PHP および Visual Basic の中から選択できます。どの言語でも、データベースの機能全体が使用可能です。言語固有の標準がすべてサポートされます。開発者は、最も慣れている言語、または特定のタスクに最も適した言語を選択できます。たとえば、アプリケーションでは、サーバー側で Java を使用して動的 Web ページを作成し、PL/SQL を使用してデータベースにストアド・プロシージャを実装し、C++ を使用して中間層に計算集中型のロジックを実装できます。

関連項目： 次のマニュアルは、様々な Oracle API について説明します。

- 『Pro*C/C++ プログラマーズ・ガイド』
- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Pro*COBOL プログラマーズ・ガイド』
- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database Data Cartridge Java API Reference』

また、詳細は、第 25 章「サポートされているアプリケーション開発言語」を参照してください。

アプリケーション開発環境の概要

Oracle では、異なるアプリケーション開発者のニーズに対応するために、様々なアプリケーション開発環境を提供しています。

- Oracle Application Express は、データベース中心の Web アプリケーションを開発およびデプロイするためのホスティングされた宣言的開発環境です。Web ブラウザのみを使用することで、プログラミング経験があまりなくても、迅速かつ安全な専門のアプリケーションを開発およびデプロイできます。Application Express エンジンは Oracle データベース内に存在し、PL/SQL で記述されています。エンジンは、データベース表に格納されたデータから、アプリケーションをリアルタイムでレンダリングします。アプリケーションを作成または拡張すると、Oracle Application Express はデータベース表内のメタデータを作成または変更します。アプリケーションの実行時、Application Express エンジンはメタデータを読み取り、アプリケーションを表示します。Oracle Application Express は、データベース内のセッションの状態も透過的に管理します。アプリケーション開発者は、標準的な SQL バインド変数構文とともに、簡単な置換でセッションの状態を取得および設定できます。Application Express は Web ベースのアプリケーションを構築するためのツールであり、アプリケーション開発環境も Web ベースです。

関連項目： 詳細は、『Oracle Database Application Express 2 日で開発者ガイド』を参照してください。

- PHP（自己参照型の頭字語で PHP - Hypertext Preprocessor の略語）は通常、動的な Web ページを作成するために HTML に埋め込まれる一般的なスクリプト言語です。PHP は急速に発達している Web 2.0 アプリケーションに最適です。PHP の oci8 への拡張は、安定したパフォーマンスの高い PHP データベース・ドライバであり、Oracle データベースと完全に統合されています。Oracle Database で PHP を使用すると、データの問合せと更新、ストアド・プロシージャとファンクションの実行、イメージのロード、およびスケーラブルでグローバルなアプリケーションの構築を簡単に実行できます。

  関連項目： 詳細は、『Oracle Database 2 日で PHP 開発者ガイド』を参照してください。

- Microsoft Windows 環境では、Oracle は次の開発環境を提供しています。
  - Oracle Data Provider for .NET（ODP.NET）の特徴は、.NET 環境から Oracle データベースへの最適化されたデータ・アクセスです。ODP.NET の使用により、開発者は Oracle Real Application Clusters、XML DB、Advanced Security などの Oracle の高度なデータベース機能を利用できるようになります。データ・プロバイダは、C#、Visual Basic .NET などの任意の .NET 言語から使用できます。

データ型の概要

SQL 文中の列値と定数は、それぞれ特定の記憶域形式、制約および値の有効範囲に対応付けられた**データ型**を持っています。表の作成時には、その各列のデータ型を指定する必要があります。

Oracle Database では、次の複数のカテゴリで多くのデータ型を使用できます。

- 文字データ型、数値データ型、DATETIME データ型などのスカラー・データ型
- データベースのデータについてのファイングレイン編成方法とアクセス方法を提供するための、可変長配列（VARRAY 型）やネストした表などのコレクション型
- Oracle データベース以外のデータでの作業を簡略化するための、ANSI がサポートされているデータ型
- 組込みデータ型または ANSI がサポートされているデータ型では不十分な場合に新しい型を定義するための、SQL ベースのインタフェースである、提供されるデータ型。これらの型の動作は C/C++、Java および PL/SQL で実装できます。

さらに、組込みデータ型または以前に作成したオブジェクト型、オブジェクト参照およびコレクション型から、ユーザー定義のオブジェクト型を作成できます。ユーザー定義型のメタデータは、SQL、PL/SQL、Java および他の公開インタフェースで使用可能なスキーマに格納されます。

ユーザー定義オブジェクト型は、基礎となる永続データ（属性）と関連する動作（メソッド）を指定するという点で、ネイティブの SQL データ型と異なります。オブジェクト型は、実社会のエンティティを抽象化したもので、抽象データ型（ADT）とも呼ばれます。

関連項目：

- Oracle の組込みデータ型および提供されるデータ型の完全なリストは、『Oracle Database SQL 言語リファレンス』を参照してください。
- 第 26 章「Oracle データ型」
- 『Oracle Database オブジェクト・リレーショナル開発者ガイド』

グローバリゼーションの概要

Oracle データベースは世界中どこにでもデプロイでき、Oracle データベースの単一インスタンスに世界中のユーザーがアクセスできます。各ユーザーには、それぞれの所在地に固有の言語と形式で情報が表示されます。

Globalization Development Kit（GDK）により開発プロセスが簡素化され、多言語市場に向けたインターネット・アプリケーションの開発コストが削減されます。GDK を使用すると、1 つのプログラムで世界各地のあらゆる言語によるテキストを処理できます。

> **関連項目：** グローバリゼーションの詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

第 II 部

Oracle データベース・アーキテクチャ

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

- 第 2 章「データ・ブロック、エクステントおよびセグメント」
- 第 3 章「表領域、データファイルおよび制御ファイル」
- 第 4 章「トランザクションの管理」
- 第 5 章「スキーマ・オブジェクト」
- 第 6 章「スキーマ・オブジェクトの依存性」
- 第 7 章「データ・ディクショナリ」
- 第 8 章「メモリー・アーキテクチャ」
- 第 9 章「プロセス・アーキテクチャ」
- 第 10 章「アプリケーション・アーキテクチャ」
- 第 11 章「Oracle Database ユーティリティ」
- 第 12 章「データベースとインスタンスの起動と停止」

2

データ・ブロック、エクステントおよびセグメント

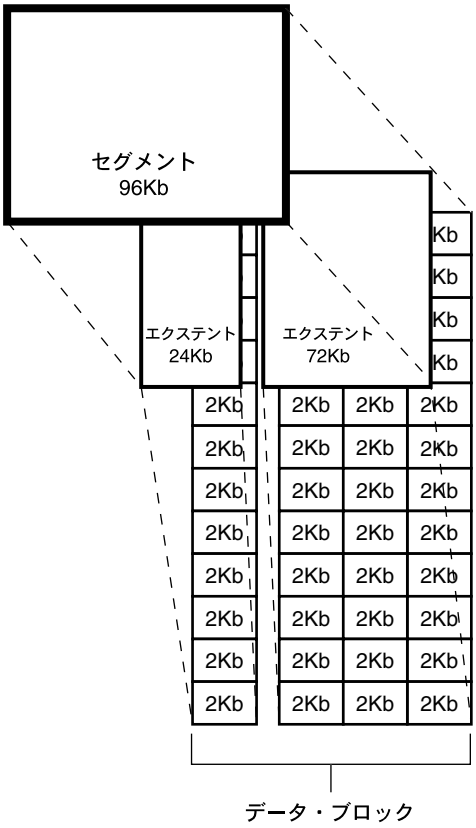
この章では、Oracle データベース・サーバーの論理的な記憶域構造の性質と、それらの構造の相互関係について説明します。

この章の内容は、次のとおりです。

データ・ブロック、エクステントおよびセグメントの概要

Oracle Database では、データベース内のすべてのデータに対して論理データベース領域が割り当てられます。データベース領域の割当て単位は、データ・ブロック、エクステントおよびセグメントです。図 2-1 は、これらのデータ構造間の関係を示します。

図 2-1 セグメント、エクステントおよびデータ・ブロックの関係



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

次の論理データベース領域のレベルは、**エクステント**と呼ばれます。エクステントは、特定数の連続したデータ・ブロックで、特定のタイプの情報を格納するために割り当てられています。

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

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

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

追加のエクステントを割り当てることはできますが、各ブロックは別々に割り当てられます。あるエクステントを特定のインスタンスに割り当てる場合、そのブロックは、即座に空きリストに割り当てられます。しかし、そのエクステントが特定のインスタンスに割り当てられない場合、そのブロックは最高水位標が移動するときのみ割り当てられます。**最高水位標**は、セグメント内の使用済領域と未使用領域間の境界です。

注意： 空き領域を自動的に管理することをお薦めします。2-5 ページの「空き領域管理」を参照してください。

データ・ブロックの概要

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

標準のブロック・サイズは、`DB_BLOCK_SIZE` 初期化パラメータで指定します。また、非標準のブロック・サイズを 5 つまで指定できます。このデータ・ブロック・サイズは、不必要な I/O を避けるための最大値の範囲内で、オペレーティング・システムのブロック・サイズの倍数です。Oracle Database データ・ブロックは、Oracle Database が使用および割当て可能な最小の格納単位です。

この項の内容は、次のとおりです。

- データ・ブロックの形式
- 空き領域管理
- PCTFREE、PCTUSED と行連鎖

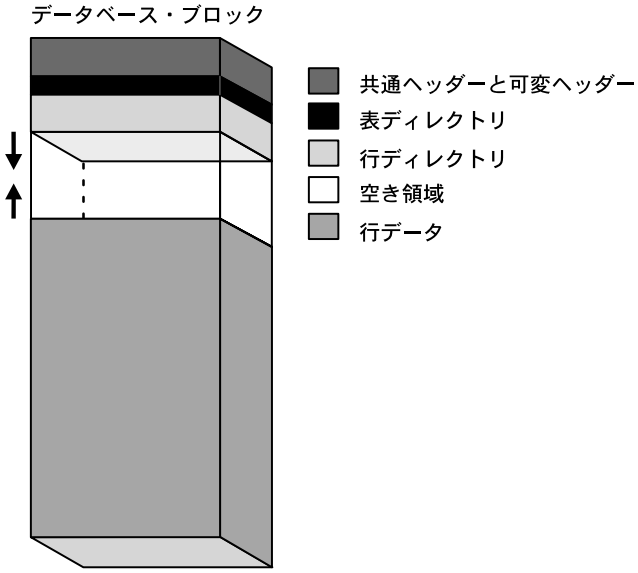
関連項目：

- データ・ブロック・サイズの詳細は、オペレーティング・システム固有の Oracle Database マニュアルを参照してください。
- 3-13 ページの「マルチ・ブロック・サイズ」

データ・ブロックの形式

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

図 2-2 データ・ブロックの形式



この項では、データ・ブロックの次のコンポーネントについて説明します。

- ヘッダー（共通と可変）
- 表ディレクトリ
- 行ディレクトリ
- オーバーヘッド
- 行データ
- 空き領域

ヘッダー（共通と可変）

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

表ディレクトリ

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

行ディレクトリ

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

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

オーバーヘッド

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

行データ

データ・ブロックのこの部分には、表データまたは索引データが格納されます。行は、複数のブロックにまたがることがあります。

関連項目： 2-6 ページの「行連鎖と移行」

空き領域

空き領域は、新しい行の挿入や、追加の領域を必要とする行の更新（後続 NULL が NULL 以外の値に更新される場合など）に割り当てられます。

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

空き領域管理

空き領域は、自動または手動で管理できます。

空き領域は、データベース・セグメント内で自動的に管理できます。セグメント内の空き領域と使用済領域は、空きリストと対照的に、ビットマップを使用して追跡されます。自動セグメント領域管理には、次の利点があります。

- 使用しやすい
- 領域の使用率が改善（特に、行のサイズが変化に富んでいるオブジェクトについて）
- 様々な同時アクセスに合せた実行時の最適な調整
- パフォーマンスや領域使用率の点で、複数インスタンスの優れた動作

ローカル管理表領域を作成するときに、自動セグメント領域管理を指定します。この指定は、この表領域でそれ以降作成されたすべてのセグメントに適用されます。

関連項目：『Oracle Database 管理者ガイド』

この項の内容は、次のとおりです。

データ・ブロック内の空き領域の可用性と最適化

2 種類の文、つまり、DELETE 文と UPDATE 文によって、1 つ以上のデータ・ブロックの空き領域を増加できます。UPDATE 文は、既存の値をより小さな値に更新します。これらの文を使用すると、解放された領域は次の条件を満たす後続の INSERT 文で使用できます。

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

解放された領域は、データ・ブロック内の空き領域の主な領域と連続しているとはかぎりません。Oracle Database では、データ・ブロックの空き領域が連続した領域にまとめられるのは、(1) INSERT 文または UPDATE 文が使用するブロックに新しい行断片が十分に収まる大きさの空き領域があり、かつ、(2) その空き領域が断片化しているために、行断片をブロックの連続した部分に挿入できない場合にかぎられます。前述の場合に圧縮が行われないと、Oracle Database はデータ・ブロックの空き領域を絶えず圧縮しようとするため、データベース・システムのパフォーマンスが低下します。

行連鎖と移行

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

それに対して、2 番目の状況は、1 つのデータ・ブロック内にすでに収まっていた行が更新されて、行全体の長さが増加したが、ブロックの空き領域がすでにいっぱいになっている場合です。この場合、Oracle Database はその行全体のデータを新しいデータ・ブロックに**移行**します（その行全体が新しいブロックに収まる場合）。Oracle Database により、移行された行の元の行断片は、行の移行先の新しいブロックを指すように保たれます。そのため、移動された行の ROWID は変わりません。

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

関連項目：

- 行および行断片の形式の詳細は、5-5 ページの「行の形式とサイズ」を参照してください。
- ROWID の詳細は、5-7 ページの「行断片の ROWID」を参照してください。
- ROWID の詳細は、26-15 ページの「物理 ROWID」を参照してください。
- 行の連鎖と移行を減らして I/O パフォーマンスを改善する方法は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

PCTFREE、PCTUSED と行連鎖

手動管理の表領域の場合、PCTFREE および PCTUSED という 2 つの領域管理パラメータで、空き領域の使用を制御し、特定セグメントのすべてのデータ・ブロック内に行の挿入および更新を実行できます。これらのパラメータは、表またはクラスタ（専用のデータ・セグメントを持つ）を作成または変更するときに指定します。記憶域パラメータ PCTFREE は、索引（専用の索引セグメントを持つ）を作成または変更するときにも指定できます。

この項の内容は、次のとおりです。

- PCTFREE パラメータ
- PCTUSED パラメータ
- PCTFREE と PCTUSED の機能

注意： この説明は、LOB データ型（BLOB、CLOB、NCLOB と BFILE）には適用されません。これらのデータ型は、PCTFREE 記憶域パラメータや空きリストを使用しません。

詳細は、26-12 ページの「LOB データ型の概要」を参照してください。

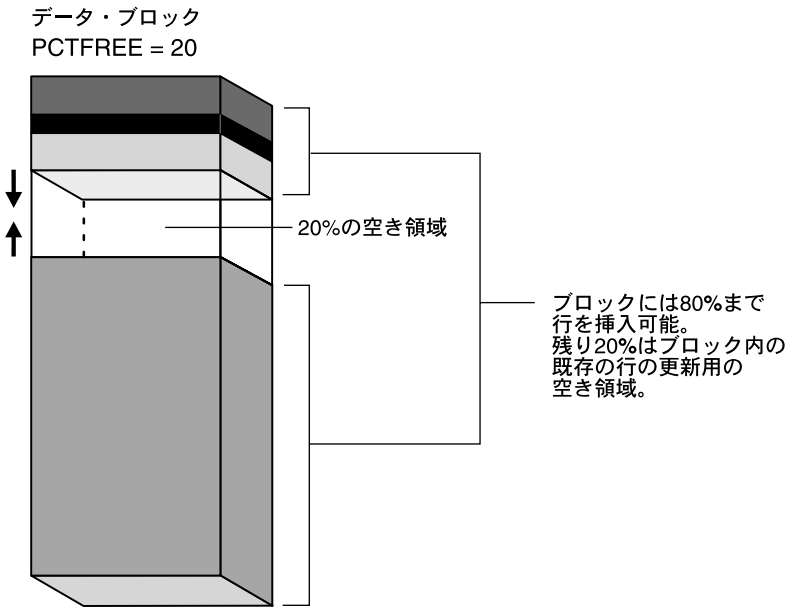
PCTFREE パラメータ

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

```
PCTFREE 20
```

これによって、この表のデータ・セグメント内の各データ・ブロックの 20% が空き状態で維持されます。この空き領域は、各ブロック内の既存の行が更新される場合に使用されます。行データとオーバーヘッドの合計が合計ブロック・サイズの 80% になるまで、新規の行を行データ領域に追加し、それに対応する情報をオーバーヘッド領域の可変部分に追加できます。図 2-3 は、PCTFREE を示しています。

図 2-3 PCTFREE



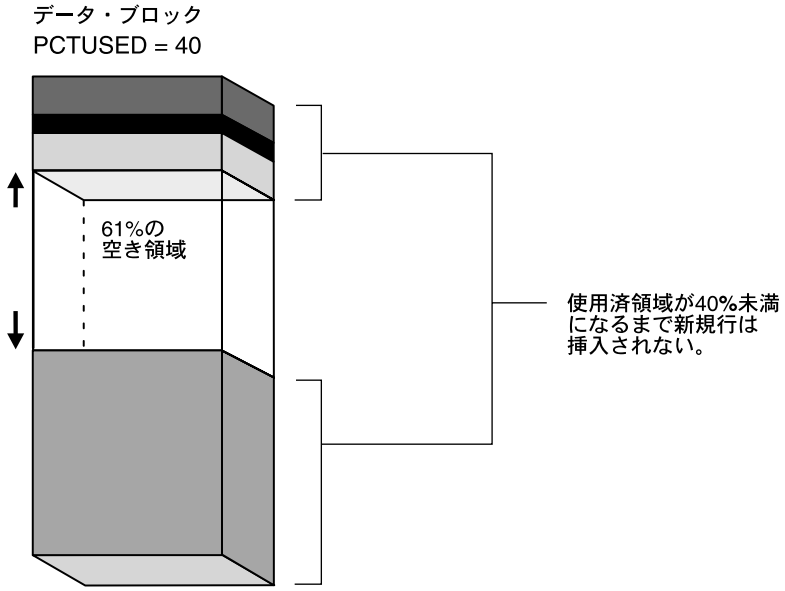
PCTUSED パラメータ

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

```
PCTUSED 40
```

この場合、この表のデータ・セグメントとして使用されるデータ・ブロックは、ブロックの使用領域の総量が 39% 以下になるまでは、新しい行の挿入に使用できないものとみなされます（ブロックの使用領域がそれ以前に PCTFREE に達していたと仮定します）。図 2-4 は、このことを示しています。

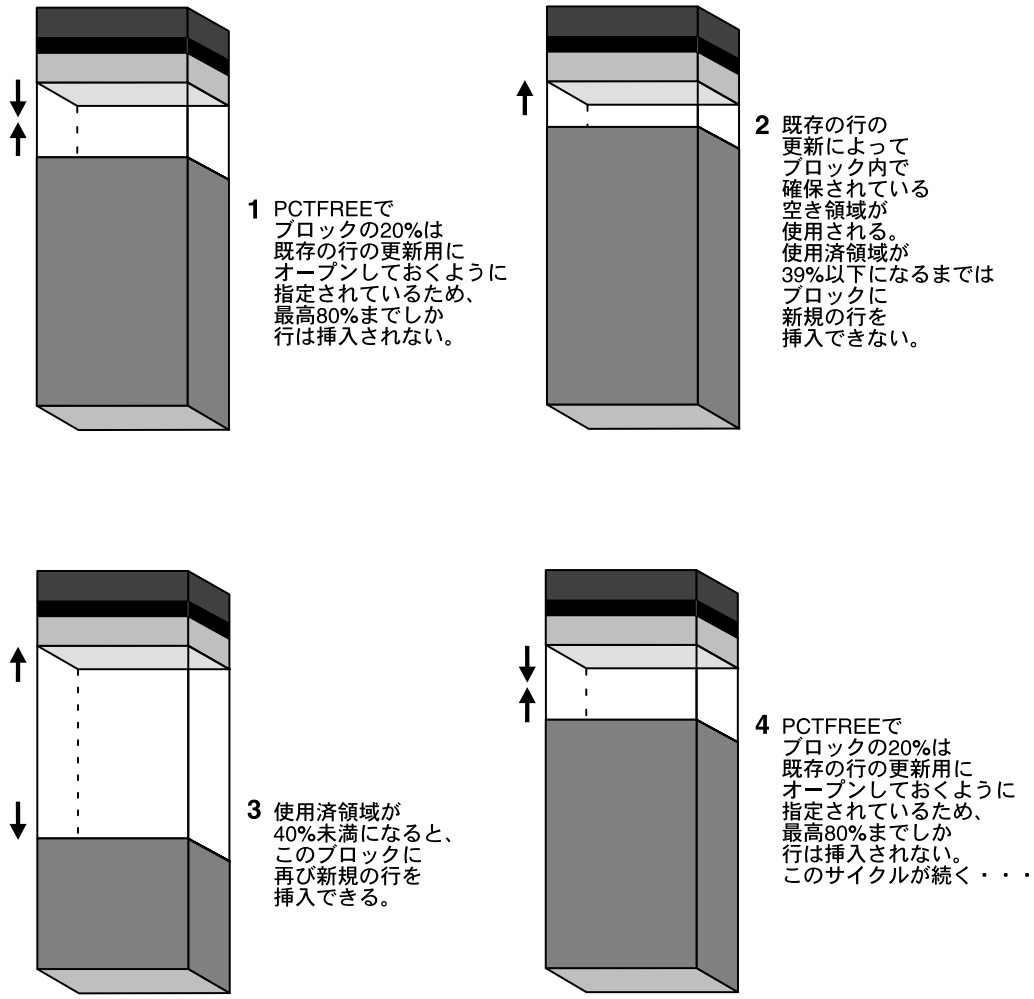
図 2-4 PCTUSED



PCTFREE と PCTUSED の機能

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

図 2-5 PCTFREE と PCTUSED によるデータ・ブロックの空き領域の管理



新しく割り当てられたデータ・ブロックでは、挿入に使用できる領域は、ブロック・サイズからブロック・オーバーヘッドと空き領域（PCTFREE）の合計を引いた大きさです。既存データの更新は、ブロック内の使用可能な領域をどれでも使用できます。このため、更新によりブロックの使用可能な領域は、更新用に確保されている領域の PCTFREE より少なくできますが、挿入には使用できません。

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

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

エクステントの概要

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

この項の内容は、次のとおりです。

エクステントの割当て

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

セグメントの初期エクステントのデータ・ブロックがいっぱいになり、新しいデータを保持するためさらに多くの領域が必要な場合、Oracle Database はそのセグメントに**増分エクステント**を自動的に割り当てます。増分エクステントは、それより前にそのセグメントに割り当てられていたエクステントのサイズと同一またはそれ以上の後続エクステントです。

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

注意： この章は、1 つのサーバー・プロセスによって SQL 文を解析して実行する、シリアル操作に適用されます。複数のサーバー・プロセスを必要とするパラレル SQL 文では、エクステントは少し異なる方法で割り当てられます。

エクステントの数とサイズの決定

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

ディレクトリ管理表領域またはローカル管理表領域を使用できます。前者ではデータ・ディクショナリ表に依存して領域の使用率が追跡され、後者ではビットマップを（データ・ディクショナリ表のかわりに）使用して使用済領域と空き領域が追跡されます。ローカル管理表領域の方がパフォーマンスと管理性に優れているため、エクステント管理のタイプを明示的に指定しないかぎり、SYSTEM 以外の永続表領域のデフォルトはローカル管理となります。

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

- 均一エクステントの場合は、エクステント・サイズを指定するか、またはデフォルト・サイズである 1MB を使用できます。各エクステントに、指定のデータベース・ブロック・サイズのデータベース・ブロックが 5 つ以上含まれていることを確認してください。エクステントがローカルに管理される一時表領域の場合は、この種の割当てしか使用できません。
- システム管理のエクステントの場合は、他のエクステントの最適サイズが Oracle Database により自動的に決定されます。この場合、最小エクステント・サイズは 64KB です。表領域が SEGMENT SPACE MANAGEMENT AUTO を使用して作成され、データベースのブロック・サイズが 16KB 以上の場合は、Oracle Database により最小サイズ 1MB のエクステントが作成されてセグメント・サイズが管理されます。これは、永続表領域のデフォルトです。

ローカル管理表領域の場合、表領域レベルでは記憶域パラメータ INITIAL、NEXT、PCTINCREASE および MINEXTENTS を指定できません。ただし、セグメント・レベルでは指定できます。この場合は、INITIAL、NEXT、PCTINCREASE および MINEXTENTS も併用してセグメントの初期サイズが計算されます。セグメント・サイズの計算後は、内部のアルゴリズムにより各エクステントのサイズが決定されます。

関連項目：

- 3-11 ページの「表領域内の領域管理」
- 3-6 ページの「大型ファイル表領域」
- 『Oracle Database 管理者ガイド』

エクステントの割当て方法

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

ローカル管理表領域では、Oracle Database は最初に表領域内で候補となるデータファイルを判別します。次に、そのデータファイルのビットマップ内で必要数の隣接する空きブロックを検索し、空き領域を検索し、新しいエクステントにこの空き領域を割り当てます。そのデータファイルに十分な隣接する空き領域がない場合は、Oracle Database は他のデータファイルを調べます。

注意： ローカル管理表領域を使用することをお薦めします。

エクステントの割当て解除

Oracle Database が提供するセグメント・アドバイザは、オブジェクト内での領域の断片化のレベルに応じて、オブジェクトに再利用可能な領域があるかどうかを判断するのに役立ちます。

関連項目：

- セグメント領域の再利用のガイドラインは、『Oracle Database 管理者ガイド』を参照してください。
- SQL 構文およびセマンティクスについては、『Oracle Database SQL 言語リファレンス』を参照してください。

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

- 表やクラスタの所有者、または DELETE ANY 権限を持つユーザーは、TRUNCATE...DROP STORAGE 文を使用して表やクラスタを切り捨てることができます。
- データベース管理者（DBA）は、次の SQL 構文を使用して未使用のエクステントの割当てを解除できます。

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

- OPTIMAL サイズが指定されているロールバック・セグメントについては、Oracle Database により定期的に 1 つ以上のエクステントの割当てが解除されることがあります。

エクステントが解放されると、Oracle Database はデータファイル内のビットマップを変更するか（ローカル管理表領域の場合）、データ・ディクショナリを更新して（ディクショナリ管理表領域の場合）、再度取得されたエクステントを使用可能領域として反映させます。解放されたエクステントのブロックにあるデータにはアクセスできなくなります。

この項の内容は、次のとおりです。

- クラスタ化されていない表のエクステント
- クラスタ化表のエクステント
- マテリアライズド・ビューとそのログのエクステント
- 索引内のエクステント
- 一時セグメント内のエクステント
- ロールバック・セグメント内のエクステント

関連項目：

- 『Oracle Database 管理者ガイド』
- 『Oracle Database SQL 言語リファレンス』

クラスタ化されていない表のエクステント

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

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

ディクショナリ管理表領域の場合は、使用可能エクステントより大きいエクステントがセグメントに必要になると、Oracle Database は再生済の連続したエクステントを識別し、それらを結合して大きいエクステントにします。これを、エクステントの**結合**と呼びます。ローカル管理表領域の場合は、新しいエクステントが 1 つ以上のエクステントから再生されたかどうかに関係なく、すべての連続した空き領域が新しいエクステントへの割当てに使用可能なため、エクステントを結合する必要はありません。

クラスタ化表のエクステント

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

マテリアライズド・ビューとそのログのエクステント

Oracle Database は、マテリアライズド・ビューとマテリアライズド・ビュー・ログのエクステントを、表やクラスタの場合と同じ方法で割当て解除します。

関連項目： 5-17 ページの「マテリアライズド・ビューの概要」

索引内のエクステント

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

一時セグメント内のエクステント

Oracle Database により一時セグメントを必要とする文の実行が完了すると、Oracle Database によって一時セグメントが自動的に削除され、そのセグメントに割り当てられていたエクステントは、対応する表領域に戻されます。単一のソートでは、その文を発行したユーザーの一時表領域に専用の一時セグメントが作成されます。その後、そのエクステントは表領域に戻されます。

それに対して複数のソートでは、ソート専用に設定されている一時表領域のソート・セグメントを使用できます。これらのソート・セグメントは、インスタンスごとに 1 回のみ割り当てられ、ソート後には戻されずに残るため、他の複数ソートでそのセグメントを使用できます。

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

関連項目：

- 2-15 ページの「一時セグメントの概要」
- 5-10 ページの「一時表」

ロールバック・セグメント内のエクステント

Oracle Database では、データベースのロールバック・セグメントを定期的にチェックし、最適サイズを超えていないかどうか確認されます。ロールバック・セグメントが最適サイズを超えている（つまり、エクステントが多すぎる）場合、ロールバック・セグメントから 1 つ以上のエクステントが Oracle Database により自動的に割当て解除されます。

セグメントの概要

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

この項の内容は、次のとおりです。

- データ・セグメントの概要
- 索引セグメントの概要
- 一時セグメントの概要
- UNDO セグメントおよび自動 UNDO 管理の概要

データ・セグメントの概要

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

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

このデータ・セグメントは、CREATE 文を使用して表またはクラスタを作成する時点で Oracle Database により作成されます。

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

注意： Oracle Database は、マテリアライズド・ビューとマテリアライズド・ビュー・ログのセグメントを、表やクラスタの場合と同じ方法で作成します。

関連項目：

- マテリアライズド・ビューとマテリアライズド・ビュー・ログの詳細は、『Oracle Database アドバンスト・レプリケーション』を参照してください。
- 構文は、『Oracle Database SQL 言語リファレンス』を参照してください。

索引セグメントの概要

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

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

一時セグメントの概要

Oracle Database では、問合せの処理中に、SQL 文の解析と実行の中間段階で、一時的な作業領域が必要になることがよくあります。Oracle Database は、**一時セグメント**と呼ばれるこのディスク領域を自動的に割り当てます。通常、Oracle Database では一時セグメントは、ソート操作のデータベース領域として必要です。ソート操作がメモリー内で実行できる場合、または Oracle Database が索引を使用して操作を実行できる別の方法を見つけた場合、Oracle Database では一時セグメントは作成されません。

この項の内容は、次のとおりです。

一時セグメントを必要とする操作

次の文では、一時セグメントの使用が必要になる場合があります。

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

一部の索引付けされていない結合および相関副問合せでも、一時セグメントの使用が必要になることがあります。たとえば、問合せに `DISTINCT` 句、`GROUP BY` および `ORDER BY` が含まれている場合、Oracle Database では 2 つの一時セグメントが必要になる可能性があります。

一時表とその索引のセグメント

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

関連項目： 5-10 ページの「一時表」

一時セグメントが割り当てられる方法

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

この項の内容は、次のとおりです。

問合せ用の一時セグメントの割当て 一時セグメントは、ユーザー・セッション中に必要に応じて、Oracle Database により文を発行したユーザーの一時表領域の 1 つに割り当てられます。これらの一時表領域は、`TEMPORARY TABLESPACE` 句を付けた `CREATE USER` または `ALTER USER` 文で指定します。

注意： 永続表領域をユーザーの一時表領域として割り当てることはできません。

ユーザーに定義されている一時表領域がない場合、SYSTEM 表領域がデフォルトの一時表領域として使用されます。一時セグメントのエクステントの記憶特性は、一時セグメントが作成された表領域のデフォルト値によって決まります。一時セグメントは、その文が完了すると Oracle Database により削除されます。

一時セグメントの割当てと割当て解除は頻繁に発生するため、一時セグメント専用に 1 つ以上の表領域を作成してください。これによって、ディスク・デバイス間で I/O を分散させるとともに、SYSTEM 表領域やその他の表領域に一時セグメントが保持されることによって生じるそれらの表領域の断片化を避けることもできます。

注意： SYSTEM 表領域がローカル管理の場合は、データベースの作成時にデフォルトの一時表領域を定義する必要があります。ローカル管理の SYSTEM 表領域は、デフォルトの一時記憶域として使用できません。

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

関連項目：

- 3-6 ページの「大型ファイル表領域」
- ユーザーの一時セグメント表領域を割り当てる方法の詳細は、第 20 章「データベース・セキュリティ」を参照してください。

一時表および索引用の一時セグメントの割当て その表への最初の INSERT が発行されるときに、Oracle Database により一時表のセグメントが割り当てられます。（これは、CREATE TABLE AS SELECT によって内部で発行される挿入操作の場合もあります。）一時表に対する最初の INSERT 文では、表とその索引にセグメントが割り当てられ、索引のルート・ページが作成され、LOB セグメントが割り当てられます。

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

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

関連項目： 5-10 ページの「一時表」

UNDO セグメントおよび自動 UNDO 管理の概要

Oracle Database では、データベースに対して行われた変更を無効にするための情報が維持されます。この情報はトランザクションのアクションのレコードで構成され、総称的に **UNDO** と呼ばれます。UNDO は UNDO 表領域の UNDO セグメントに格納されます。UNDO の情報は Oracle Database により次の目的で使用されます。

- アクティブ・トランザクションのロールバック
- 終了したトランザクションのリカバリ
- 読取り一貫性の提供
- 論理の破損からのリカバリ

ROLLBACK 文が発行されると、UNDO レコードは、コミットされていないトランザクションによってデータベースに加えられた変更を取り消すために使用されます。データベース・リカバリ中には、UNDO レコードは、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すべての変更を取り消すために使用されます。UNDO レコードでは、あるユーザーによるデータの変更中にそのデータにアクセスしているユーザーのために、そのデータの修正前のイメージを維持することで読取り一貫性が実現されています。読取り一貫性の詳細は、13-3 ページの「Oracle Database データの同時実行性と整合性の管理方法」を参照してください。

Oracle Database では、UNDO 情報および領域の管理用に、**自動 UNDO 管理**と呼ばれる完全に自動化されたメカニズムが提供されています。この管理モードでは、すべての現行セッションで、UNDO 表領域の UNDO セグメントおよび領域がサーバーにより自動的に管理されます。

自動 UNDO 管理により、ロールバック・セグメント領域の管理の複雑さが解消されます。また、UNDO 情報を必要とする長時間実行される問合せに対応するために、最善の状態で保存された UNDO 情報を提供できるよう、システムによりシステム自体が自動的にチューニングされます。自動 UNDO 管理は、Oracle Database の新しいインストールのデフォルトです。インストール・プロセスにより、UNDO 表領域が自動的に作成されます。

Oracle Database に用意されている UNDO アドバイザは、UNDO 環境の確立に関するアドバイスを提供し、その自動化に役立ちます。

この項の内容は、次のとおりです。

- 手動 UNDO 管理
- UNDO 割当て
- 自動 UNDO 保存

> **関連項目：** UNDO アドバイザおよびアドバイザの使用方法は、『Oracle Database 2 日でデータベース管理者』を参照してください。また、自動 UNDO 管理の使用方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

手動 UNDO 管理

データベース・システムは、手動 UNDO 管理モードでも実行できます。手動 UNDO 管理モードでは、UNDO 領域はロールバック・セグメントを介して管理され、UNDO 表領域は使用されません。

Oracle Database の以前のリリースでは、手動 UNDO 管理モードがデフォルトでした。自動 UNDO 管理に変更するには、まず UNDO 表領域を作成し、初期化パラメータを変更する必要がありました。Oracle Database のリリースが 9*i* 以降の場合に自動 UNDO 管理に変更する際には、『Oracle Database アップグレード・ガイド』を参照してください。

注意： ロールバック・セグメントの領域管理は複雑です。自動 UNDO 管理を使用することをお薦めします。

UNDO 割当て

自動 UNDO 管理モードの場合、システムは UNDO セグメントへのトランザクションの割当てを排他的に制御します。また、UNDO セグメントの領域割当てを制御します。問題のあるトランザクションは、UNDO 領域のほとんどを使用してしまうことがあるため、システム全体が機能できなくなります。リソース・マネージャ・ディレクティブの `UNDO_POOL` は、大きいトランザクションを制御するより明示的な方法です。このディレクティブによって、データベース管理者はユーザーをコンシューマ・グループに分けて、それぞれのグループに最大 UNDO 領域の上限を割り当てることができます。あるグループが使用する UNDO 領域の合計が上限を超えると、そのグループのユーザーは、UNDO 領域が他のメンバーのトランザクションの終了によって解放されるまで、いかなる更新も実行できません。

`UNDO_POOL` のデフォルト値は、`UNLIMITED` です。ユーザーは UNDO 表領域が保持しているのと同じ大きさの UNDO 領域を使用できます。データベース管理者は、`UNDO_POOL` ディレクティブを使用して特定のユーザーを制限できます。

自動 UNDO 保存

トランザクションのコミット後、ロールバックやトランザクション・リカバリの目的には UNDO データは不要になります。ただし、一貫性読取りの目的では、長時間実行される問合せで、データ・ブロックの古いイメージを生成するためにこの古い UNDO 情報が必要になる場合があります。さらに、古い UNDO 情報の可用性が、Oracle Flashback 機能の成功に関わる場合もあります。これらの理由から、古い UNDO 情報をできるだけ長期間保存することをお薦めします。UNDO 表領域に新しいトランザクション用の領域がある場合には、古い UNDO 情報を保存できます。表領域の使用可能な領域が不足すると、データベースにより、コミットされたトランザクションの古い UNDO 情報の上書きが開始されます。

現行の UNDO 表領域の最善の UNDO 保存を行うため、Oracle Database によりシステムが自動的にチューニングされます。データベースにより、使用統計が収集され、これらの統計および UNDO 表領域サイズに基づいて UNDO 保存期間がチューニングされます。UNDO 表領域が、最大サイズが指定されていない状態で `AUTOEXTEND` オプションを使用して構成されている場合には、UNDO 保存チューニングは多少異なります。この場合には、データベースにより、実行時間が最も長い問合せより少し長くなるように、UNDO 保存期間がチューニングされます(領域が許す場合)。

> **関連項目：** UNDO 保存の自動チューニングの詳細は、『Oracle Database 管理者ガイド』を参照してください。

3

表領域、データファイルおよび制御ファイル

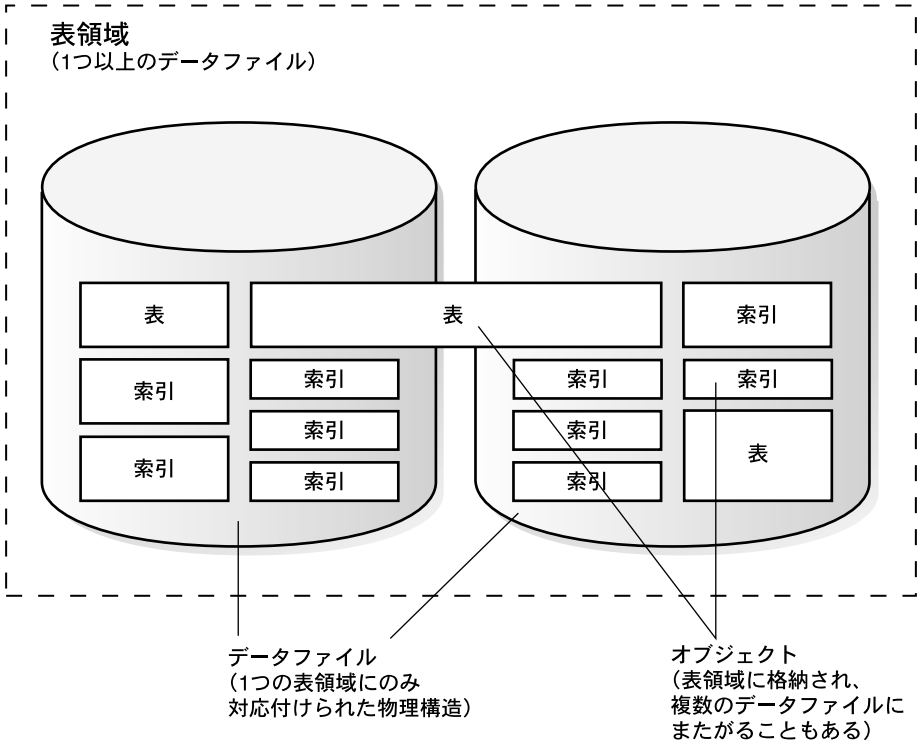
この章では、Oracle データベースの主要な論理データベース構造である表領域と、それぞれの表領域に対応する物理データファイルについて説明します。

この章の内容は、次のとおりです。

表領域、データファイルおよび制御ファイルの概要

Oracle Database では、**表領域**にデータを論理的に格納し、対応する表領域に関連付けられた**データファイル**にデータを物理的に格納します。図 3-1 に、この関係を示します。

図 3-1 データファイルと表領域



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

- Oracle データベースは、データベースのすべてのデータがまとめて格納される、表領域と呼ばれる 2 つ以上の論理記憶単位からなっています。SYSTEM および SYSAUX 表領域は必須であり、TEMP という 3 番目の表領域はオプションです。
- Oracle データベース内の各表領域は、データファイルと呼ばれる 1 つ以上のファイルからなっています。データファイルは、Oracle Database を実行しているオペレーティング・システムに適合する物理構造です。
- データベースのデータは、データベースの各表領域を構成するデータファイル内にまとめて格納されます。たとえば、最も単純な Oracle データベースには、1 つのデータファイルで構成される表領域が 1 つあります。他のデータベースは、それぞれ 2 つのデータファイルから構成される 3 つの表領域を持つ場合もあります（合計で 6 つのデータファイルになります）。

この項の内容は、次のとおりです。

- Oracle Managed Files
- データベースへの多くの領域の割当て

Oracle Managed Files

Oracle Managed Files により、データベース管理者は、Oracle データベースを構成するオペレーティング・システム・ファイルを直接管理する必要がなくなります。操作を指定するときには、ファイル名ではなくデータベース・オブジェクトを使用します。Oracle Database では、標準ファイル・システム・インタフェースを内部で使用して、次のデータベース構造のファイルを必要に応じて作成および削除します。

- 表領域
- REDO ログ・ファイル
- 制御ファイル

初期化パラメータを介して、特定タイプのファイルに使用するファイル・システム・ディレクトリを指定します。Oracle Database では、一意のファイルである Oracle Managed File が作成され、不要になると削除されます。

関連項目：

- 『Oracle Database 管理者ガイド』
- 14-15 ページの「自動ストレージ管理」

データベースへの多くの領域の割当て

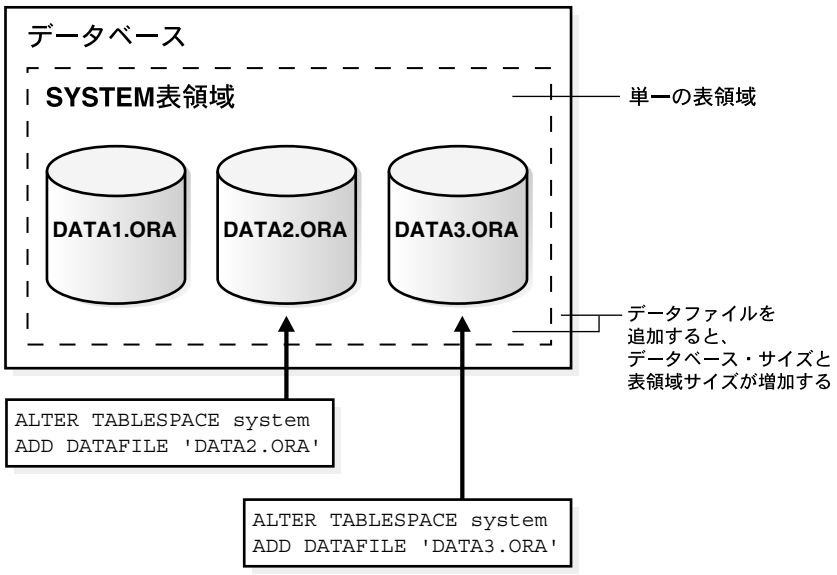
表領域のサイズは、表領域を構成するデータファイルのサイズです。データベースのサイズは、データベースを構成する表領域の合計サイズです。

データベースのサイズを拡張するには、次の 3 つの方法があります。

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

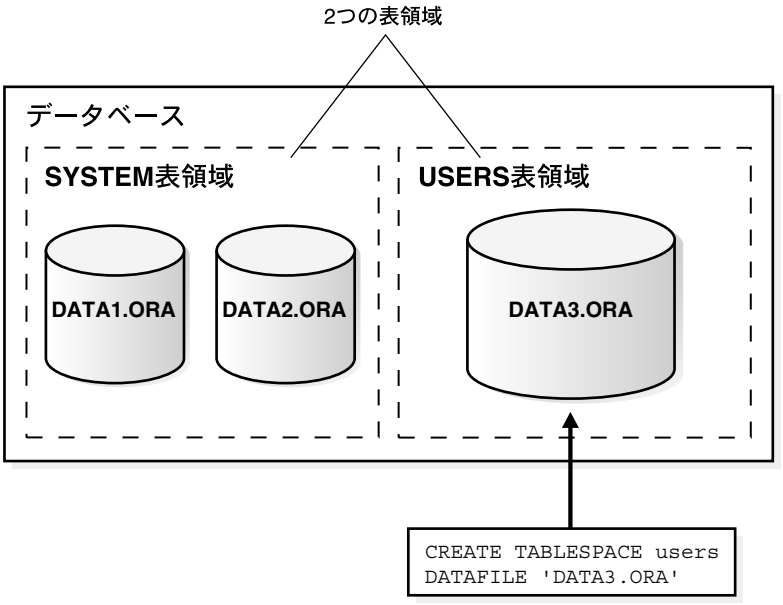
既存の表領域にデータファイルを追加すると、その表領域に割り当てられているディスク領域の容量を増やすことになります。図 3-2 は、このようにして領域を拡大する方法を示しています。

図 3-2 表領域にデータファイルを追加してデータベースを拡大



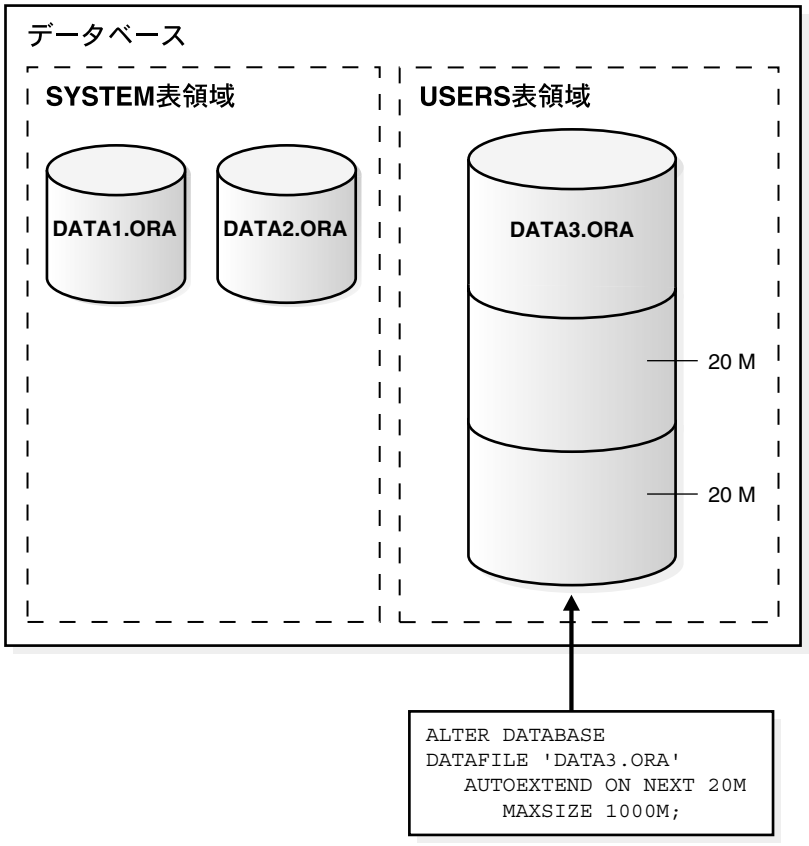
別の方法として、新しい表領域を作成し（少なくとも 1 つのデータファイルを作成）、データベースのサイズを大きくすることもできます。図 3-3 は、この方法を示しています。

図 3-3 新しい表領域を追加してデータベースを拡大



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

図 3-4 データファイルを動的にサイズ変更してデータベースを拡大



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

表領域の概要

データベースは、表領域と呼ばれる 1 つ以上の論理記憶単位に分割されます。表領域は**セグメント**と呼ばれる論理記憶単位に分けられ、セグメントはさらに**エクステント**に分けられています。エクステントは連続するブロックのコレクションです。

この項で表領域に関して取り上げる項目は、次のとおりです。

- 大型ファイル表領域
- SYSTEM 表領域
- SYSAUX 表領域
- UNDO 表領域
- デフォルト一時表領域
- 複数の表領域の使用
- 表領域内の領域管理
- マルチ・ブロック・サイズ
- オンライン表領域とオフライン表領域
- 読取り専用表領域
- 一時表領域
- データベース間での表領域の移動

関連項目：

- セグメントとエクステントの詳細は、第 2 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。
- 表領域の作成と構成の詳細は、『Oracle Database 管理者ガイド』を参照してください。

大型ファイル表領域

Oracle Database では、**大型ファイル表領域**を作成できます。これにより、Oracle Database では、多数の小型ファイルではなく単一の大型ファイルで構成された表領域を格納できます。この表領域により、Oracle Database では、64 ビット・システムの機能を利用して超大規模ファイルを作成および管理できます。この結果、Oracle Database は 800 万 TB までサイズを拡張できるようになりました。

Oracle Managed Files では、大型ファイル表領域によりデータファイルがユーザーに対して完全に透過的になります。つまり、基礎となるデータファイルではなく表領域に対する操作を実行できます。大型ファイル表領域では、表領域はディスク領域の管理、バックアップおよびリカバリなどの主要単位となります。また、大型ファイル表領域により、新規データファイルを追加したり複数ファイルを取り扱う必要がなくなり、Oracle Managed Files と自動ストレージ管理によるデータファイル管理が簡素化されます。

システム・デフォルトでは、従来型の Oracle Database 表領域である小型ファイル表領域が作成されます。`SYSTEM` および `SYSAUX` 表領域タイプは、常にシステム・デフォルトのタイプを使用して作成されます。

大型ファイル表領域がサポートされるのは、自動セグメント領域管理を使用するローカル管理表領域の場合のみです。ただし、これには 2 つの例外があり、ローカル管理の UNDO 表領域と一時表領域は、セグメントが手動管理される場合にも大型ファイル表領域として使用できます。

Oracle データベースには、大型ファイル表領域と小型ファイル表領域の両方を含めることができます。各種の表領域は、データファイルを明示的に参照しない SQL 文の実行時には区別できません。

ユーザーが複数の表領域の一時領域を消費できるように、一時表領域のグループを作成できます。表領域グループは、データベースのデフォルト一時表領域としても指定できます。これは、ソート操作用の一時表領域が多数必要になる可能性がある大型ファイル表領域を使用する際に役立ちます。

この項の内容は、次のとおりです。

大型ファイル表領域の利点

- 大型ファイル表領域を使用すると、Oracle データベースの記憶容量を大幅に増やすことができます。小型ファイル表領域には最大 1024 ファイルを格納できますが、大型ファイル表領域には、小型ファイル表領域の 1024 倍までの大きさになるファイルを 1 つのみ格納します。小型ファイル表領域と大型ファイル表領域では、表領域の合計容量は同じです。ただし、各データベースのデータファイルには 64KB という制限があるため、1 つのデータベースには小型ファイル表領域よりも 1024 倍の数の大型ファイル表領域を含めることができ、大型ファイル表領域を使用するとデータベースの合計容量が 3 桁も大きくなります。つまり、最大ブロック・サイズが 32KB の大型ファイル表領域を使用した場合、Oracle データベースの最大サイズは 800 万 TB です。
- 大型ファイル表領域により、データファイルの必要数が減少することで、超大規模データベースにおけるデータファイルの管理が簡素化されます。また、パラメータを調整して、データファイル情報に必要な SGA 領域を減らし、制御ファイルのサイズを小型化することもできます。
- データファイルの透過性が得られることでデータベース管理が簡素化されます。

大型ファイル表領域の考慮事項

- 大型ファイル表領域は、自動ストレージ管理、または動的に拡張可能な論理ボリュームとストライプ化または RAID をサポートしている他の論理ボリューム・マネージャとともに使用することを意図しています。
- パラレル実行と Recovery Manager によるバックアップのパラレル化はデメリットを伴うため、ストライプ化をサポートしていないシステムでは、大型ファイル表領域を作成しないでください。
- ディスク・グループで使用可能な空き領域がなくなる可能性があり、別のディスク・グループに新規データファイルを追加しなければ表領域を拡張できない場合は、大型ファイル表領域を使用しないでください。
- 大型ファイル・サイズをサポートしていないプラットフォームで、大型ファイル表領域を使用しないでください。表領域の容量に制限がある場合があります。サポートされる最大ファイル・サイズについては、オペレーティング・システム固有のマニュアルを参照してください。
- データを従来の表領域ではなく大型ファイル表領域に格納すると、データベースのオープン、チェックポイントおよび DBWR プロセスのパフォーマンスが改善されます。ただし、データファイルのサイズが大きくなると、破損ファイルのリストアや新規データファイルの作成に時間がかかるようになる場合があります。

> **関連項目：** 大型ファイル表領域の作成、変更および管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。

SYSTEM 表領域

Oracle Database には、データベースの作成時に自動的に作成される SYSTEM という表領域が含まれています。SYSTEM 表領域は、データベースのオープン中は常にオンラインになっています。

ローカル管理表領域の利点を活用するには、ローカル管理の SYSTEM 表領域を作成する方法と、既存のディクショナリ管理の SYSTEM 表領域をローカル管理形式に移行する方法があります。

ローカル管理の SYSTEM 表領域を持つデータベースには、ディクショナリ管理表領域を作成できません。トランスポータブル機能を使用するとディクショナリ管理表領域をプラグインできますが、書込み可能にすることはできません。

この項の内容は、次のとおりです。

データ・ディクショナリ

SYSTEM 表領域には、データベース全体のデータ・ディクショナリ表が必ず含まれます。

PL/SQL プログラム・ユニットの説明

ストアド PL/SQL プログラム・ユニット（プロシージャ、ファンクション、パッケージおよびトリガー）のために格納されるデータは、すべて SYSTEM 表領域にあります。データベースにこれらのプログラム・ユニットの多くを含める場合、データベース管理者は必要なスペースを SYSTEM 表領域に確保する必要があります。

関連項目：

- ローカル管理の SYSTEM 表領域を作成する方法、またはローカル管理の SYSTEM 表領域に移行する方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- SYSTEM 表領域の永続的なオンライン条件の詳細は、3-13 ページの「オンライン表領域とオフライン表領域」を参照してください。
- PL/SQL プログラム・ユニットの領域要件の詳細は、第 24 章「SQL」および第 22 章「トリガー」を参照してください。

SYSAUX 表領域

SYSAUX 表領域は、SYSTEM 表領域の補助表領域です。多数のデータベース・コンポーネントでは、デフォルトのデータ格納場所として SYSAUX 表領域が使用されます。そのため、データベースの作成時またはアップグレード時には、必ず SYSAUX 表領域が作成されます。

注意： メディア障害などの原因で SYSAUX 表領域が使用不可能になると、一部のデータベース機能が失敗する場合があります。

SYSAUX 表領域は、SYSTEM 表領域に常駐しないデータベース・メタデータの集中格納場所となります。これにより、シード・データベースとユーザー定義データベースの両方で、デフォルトで作成される表領域の数が減少します。

Oracle Database では、通常のデータベース操作中には SYSAUX 表領域の削除や名前変更は許可されません。SYSAUX 用のトランスポータブル表領域はサポートされません。

関連項目： SYSAUX 表領域を使用するデータベース・コンポーネントの詳細は、『Oracle Database 管理者ガイド』を参照してください。

UNDO 表領域

UNDO 表領域は、ロールバック情報の格納にのみ使用する特別な表領域です。UNDO 表領域に、タイプの異なるセグメント（表や索引など）を作成することはできません。UNDO 表領域は、データベースが自動 UNDO 管理モード（デフォルト）の場合にのみ使用されます。データベースに複数の UNDO 表領域を含めることができますが、使用中となるのは常に 1 つの UNDO 表領域のみです。UNDO データは、データベースにより自動的に作成および維持される UNDO セグメントを使用して、UNDO 表領域内で管理されます。

トランザクション内で最初の DML 操作が実行されると、そのトランザクションは、現在の UNDO 表領域内の UNDO セグメント（トランザクション表）にバインド（割当て）されます。インスタンスに対し専用の UNDO 表領域が割り当てられていない場合は、トランザクションがシステム UNDO セグメントにバインドされます。

各 UNDO 表領域には一連のデータファイルが含まれ、ローカルで管理されます。他の表領域と同様、UNDO ブロックはエクステントの中でグループ化され、各エクステントの状態がビットマップで表示されます。任意の時点で、エクステントはトランザクション表に割り当てられ（て使用され）るか、または解放されます。

大型ファイル UNDO 表領域を作成できます。

関連項目：

- UNDO 表領域の管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 3-6 ページの「大型ファイル表領域」

UNDO 表領域の作成

UNDO 表領域は、Oracle Database の各新規インストールで自動的に作成されます。Oracle Database の旧リリースには、UNDO 表領域が含まれておらず、かわりにロールバック・セグメントが使用されていました。これは、手動 UNDO 管理モードと呼ばれます。Oracle Database 11*g* にアップグレードすると、UNDO 表領域を作成して自動 UNDO 管理モードを有効化することで自動 UNDO 管理に移行できます。詳細は、『Oracle Database アップグレード・ガイド』を参照してください。

デフォルト一時表領域

SYSTEM 表領域がローカル管理の場合は、データベースの作成時にデフォルトの一時表領域を 1 つ以上定義する必要があります。ローカル管理の SYSTEM 表領域は、デフォルトの一時記憶域として使用できません。

SYSTEM がディクショナリ管理されており、データベースの作成時にデフォルトの一時表領域を定義しない場合は、デフォルトの一時記憶域に引き続き SYSTEM が使用されます。ただし、デフォルトの一時表領域が推奨され、今後のリリースでは必須となることを伝える警告を ALERT.LOG で受け取ります。

デフォルトの一時表領域の指定方法

データベース作成の際は、CREATE DATABASE 文に対する DEFAULT TEMPORARY TABLESPACE 拡張機能を使用して、デフォルトの一時表領域を指定します。

大型ファイル一時表領域を作成できます。この種の表領域では、すべての一時表領域と同様に、データファイルではなく一時ファイルが使用されます。

注意： デフォルトの一時表領域を永続表領域化またはオフライン化することはできません。

関連項目：

- デフォルトの一時表領域の定義と変更の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 3-6 ページの「大型ファイル表領域」

複数の表領域の使用

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

複数の表領域を使用すると、次のタスクを実行できます。

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

データベース管理者は、次のことを実行できます。

- 新規表領域の作成
- 表領域へのデータファイルの追加
- 表領域に作成したセグメントのデフォルトのセグメント記憶域設定の設定と変更
- 表領域を読取り専用または読取り / 書込みに設定
- 表領域を一時表領域または永続表領域に設定
- 表領域名の変更
- 表領域の削除
- データベース間およびプラットフォーム間での表領域の移動

表領域内の領域管理

表領域では、領域がエクステント単位で割り当てられます。表領域では、次の 2 つの方法で空き領域と使用済領域を追跡できます。

- **ローカル管理表領域**: ビットマップによるエクステント管理
- **ディクショナリ管理表領域**: データ・ディクショナリによるエクステント管理

表領域の作成時に、この 2 つの領域管理方法からどちらかを選択できます。選択した管理方法は、後で DBMS_SPACE_ADMIN PL/SQL パッケージを使用して変更できます。

この項の内容は、次のとおりです。

関連項目： 2-10 ページの「エクステントの概要」

ローカル管理表領域

自身のエクステントを管理する表領域では、各データファイルのビットマップが保持され、そのデータファイル内のブロックの解放または使用状態が追跡されます。ビットマップ内の各ビットは、1 ブロックまたは 1 ブロック・グループに対応します。エクステントが割り当てられるか、再利用のために解放されると、Oracle Database では、新しいブロックの状態を表すためにビットマップの値が変更されます。

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

- エクステントのローカル管理によって、隣接する空き領域が自動的に追跡され、使用可能エクステントを結合する必要がなくなります。
- エクステントのローカル管理によって、再帰的な領域管理操作が回避されます。エクステント内の領域を使用または解放することにより、データ・ディクショナリ表やロールバック・セグメントの領域を使用または解放する他の操作が生じる場合に、ディクショナリ管理表領域内でこの種の再帰的な操作が発生することがあります。

ローカルに管理されるエクステントのサイズは、システムにより自動的に決定されます。また、ローカル管理表領域内では、すべてのエクステントを同じサイズにすることもでき、さらにオブジェクト記憶域オプションがオーバーライドされます。

ローカル管理の永続表領域または一時表領域をそれぞれ作成するには、CREATE TABLESPACE または CREATE TEMPORARY TABLESPACE 文の LOCAL 句を指定します。

ローカル管理表領域内のセグメント領域管理

CREATE TABLESPACE 文を使用したローカル管理表領域の作成では、SEGMENT SPACE MANAGEMENT 句により、セグメント内の空き領域および使用領域の管理方法を指定します。次のいずれかを選択します。

- AUTO

  このキーワードを指定すると、Oracle Database で、セグメント内の空き領域の管理にビットマップが使用されます。この場合のビットマップは、行の挿入に使用できるブロックの領域に対応するセグメント内の各データ・ブロックの状態を示すマップです。データ・ブロックに、ある程度の空き領域ができると、新しい状態がビットマップに反映されます。Oracle Database では、ビットマップを使用することで、空き領域の管理がさらに自動化されます。そのため、このような領域管理は、自動セグメント領域管理と呼ばれます。

  自動セグメント領域管理を使用するローカル管理表領域は、小型ファイル表領域（従来型）または大型ファイル表領域として作成できます。AUTO がデフォルトです。

- MANUAL

  このキーワードを指定すると、Oracle Database では、セグメント内の空き領域の管理に空きリストが使用されます。空きリストとは、行の挿入に使用できる領域が確保されたデータ・ブロックのリストです。

関連項目：

- 構文は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 自動セグメント領域管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 2-11 ページの「エクステントの数とサイズの決定」
- 一時表領域の詳細は、3-14 ページの「一時表領域」を参照してください。

ディクショナリ管理表領域

Oracle9*i* でデータベースを作成した場合は、ディクショナリ管理表領域を使用できます。表領域で、エクステントの管理にデータ・ディクショナリを使用する場合、Oracle Database では、エクステントが割り当てられたり再利用のために解放されるたびにデータ・ディクショナリ内の適切な表が更新されます。また、Oracle Database では、ディクショナリ表の更新ごとに、ロールバック情報が格納されます。ディクショナリ表とロールバック・セグメントはデータベースの一部であるため、これらが占める領域は、他のすべてのデータと同じ領域管理操作の対象となります。

注意： 表領域の作成時にエクステント管理を指定しない場合は、ローカル管理がデフォルトとなります。

マルチ・ブロック・サイズ

Oracle Database は、マルチ・ブロック・サイズをデータベースでサポートします。SYSTEM 表領域には、**標準ブロック・サイズ**が使用されます。標準ブロック・サイズは、データベースが作成されたときに設定され、任意の有効サイズに設定可能です。初期化パラメータ DB_BLOCK_SIZE を設定して、標準ブロック・サイズを指定します。指定できる値は 2K ～ 32K です。

初期化パラメータ・ファイルまたはサーバー・パラメータ・ファイルで、これらのブロック・サイズごとにバッファ・キャッシュ内のサブキャッシュを構成できます。インスタンスの実行中にサブキャッシュも構成できます。任意のブロック・サイズの表領域を作成できます。標準ブロック・サイズは、SYSTEM 表領域および他の多くの表領域で使用されます。

注意： パーティション・オブジェクトのパーティションはすべて、単一ブロック・サイズの表領域に常駐する必要があります。

マルチ・ブロック・サイズが主に役立つのは、表領域を OLTP データベースから企業のデータ・ウェアハウスへ移動する場合です。これにより、ブロック・サイズが異なるデータベース間での移動が容易になります。

関連項目：

- 3-15 ページの「データベース間での表領域の移動」
- データ・ウェアハウス環境で表領域を移動する方法の詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

オンライン表領域とオフライン表領域

データベース管理者は、Oracle データベースがオープンされているときはいつでも、その表領域（SYSTEM 表領域を除く）を**オンライン**（アクセス可能）または**オフライン**（アクセス不能）にできます。Oracle Database が常にデータ・ディクショナリにアクセスできるようにするため、SYSTEM 表領域は、データベースのオープン中は常にオンラインです。

表領域は通常はオンラインになっており、データベース・ユーザーはその表領域内のデータを使用できます。一方、データベース管理者は、メンテナンスまたはバックアップおよびリカバリの目的に表領域をオフラインで使用できます。

表領域のオフライン化

Oracle Database では、表領域がオフライン化されると、その表領域に含まれるオブジェクトを SQL 文で参照できなくなります。アクティブなトランザクションで、この表領域のデータを参照する文がすでに完了している場合、トランザクション・レベルでの影響はありません。Oracle Database では、すでに完了しているこれらの文に対応するロールバック・データが、SYSTEM 表領域の遅延ロールバック・セグメントに保存されます。表領域が再びオンライン化されたとき、Oracle Database では、必要に応じて、ロールバック・データが表領域に適用されます。

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

表領域をオンラインにできるのは、その表領域を作成したデータベース内のみに限られます。これは、必要なデータ・ディクショナリ情報がそのデータベースの SYSTEM 表領域内に維持されているためです。Oracle Database 以外のユーティリティでは、オフライン表領域の読取りまたは編集はできません。したがって、オフライン表領域を他のデータベースへ転送することはできません。

Oracle Database では、特定のエラーが発生すると、表領域がオンラインからオフラインに自動的に切り替わります。たとえば、データベース・ライター・プロセス DBW*n* が、表領域のデータファイルへの書込みに数回失敗すると、Oracle Database では、表領域がオンラインからオフラインに自動的に切り替わります。オフライン表領域にある表にアクセスしようとしたユーザーは、エラーを受け取ります。このようにディスク I/O が失敗してしまう原因がメディア障害の場合は、問題を解決してから、表領域をリカバリする必要があります。

関連項目：

- オンライン表領域をデータベース間で転送する方法の詳細は、3-15 ページの「データベース間での表領域の移動」を参照してください。
- データ転送ツールの詳細は、『Oracle Database ユーティリティ』を参照してください。

読取り専用表領域

読取り専用表領域の主な目的は、データベースの静的な部分を大量にバックアップしたりリカバリする必要をなくすことです。読取り専用表領域のファイルは Oracle Database によって更新されることがないため、CD-ROM や WORM ドライブのような読取り専用メディアに常駐させることができます。

注意： 表領域をオンライン化できるのは、表領域が作成されたデータベース内のみです。そのため、読取り専用表領域はアーカイブ要件を満たしていない場合があります。

読取り専用表領域は変更できません。読取り専用表領域を更新するには、最初に表領域を読取り / 書込み可能にします。表領域の更新後に、その表領域を読取り専用に再設定します。

読取り専用表領域は変更できないため、読取り / 書込みが可能に設定されていないかぎり、バックアップを繰り返す必要はありません。また、データベースのリカバリが必要な場合、読取り専用表領域は修正されていないため、リカバリする必要がありません。

関連項目：

- 表領域を読取り専用または読取り / 書込みモードに変更する方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- `ALTER TABLESPACE` 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- リカバリの詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

一時表領域

ソート操作用の領域は、ソート専用に指定した 1 つ以上の一時表領域を使用すると、さらに効率的に管理できます。それにより、ソート領域の割当ておよび割当て解除に伴う領域管理操作のシリアライズを効率的に解消できます。1 つの SQL 操作で、ソート操作用の一時表領域を複数使用できます。たとえば、非常に大きな表の索引を作成し、索引作成中のソート操作を複数の表領域に分散できます。

結合、索引作成、順序付け、集計の計算（`GROUP BY`）およびオプティマイザ統計の収集など、ソートを使用するすべての操作には、一時表領域の機能が役立ちます。Oracle Real Application Clusters を使用することで、パフォーマンスが大幅に向上します。

この項の内容は、次のとおりです。

ソート・セグメント

1 つ以上の一時表領域は、ソート・セグメントにのみ使用できます。ユーザーが一時セグメント用に指定する表領域とは異なり、一時表領域としてはユーザーが使用可能な任意の表領域を使用できます。永続スキーマ・オブジェクトを一時表領域に格納することはできません。

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

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

> **関連項目：** セグメントの詳細は、第 2 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。

一時表領域の作成

一時表領域の作成には、`CREATE TABLESPACE` または `CREATE TEMPORARY TABLESPACE` 文を使用します。

> **関連項目：**
>
> - `TEMPFILES` の詳細は、3-18 ページの「一時データファイル」を参照してください。
> - ローカル管理表領域とディクショナリ管理表領域の詳細は、3-11 ページの「表領域内の領域管理」を参照してください。
> - 構文は、『Oracle Database SQL 言語リファレンス』を参照してください。
> - ソートおよびハッシュ結合用に一時表領域を設定する方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

データベース間での表領域の移動

トランスポータブル表領域により、異なるプラットフォームにまたがる場合も、Oracle データベース間でそのサブセットを移動できます。表領域のクローンを作成して別のデータベースにプラグインする（データベース間で表領域をコピーする）方法と、ある Oracle データベースから表領域をアンプラグして他の Oracle データベースにプラグインする（データベース間で表領域を移動する）方法があります。

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

表領域はプラットフォーム間で転送できます。（すべてではありませんが、多くのプラットフォームでプラットフォーム間の表領域転送がサポートされています。）この機能を使用して、次のことが可能です。

- コンテンツ・プロバイダは、より簡単かつ効率的な方法で、構造化データを公開し、異なるプラットフォームで Oracle Database を実行している顧客に配布できます。
- データ・ウェアハウス環境からデータ・マート（多くの場合、小規模なプラットフォーム上で稼働）へのデータの配布を簡素化できます。
- 異機種間クラスタで読取り専用表領域を共有できます。
- プラットフォーム間でデータベースを移行できます。

この項の内容は、次のとおりです。

表領域リポジトリ

表領域リポジトリとは、表領域セットの集合です。表領域リポジトリはファイル・グループ・リポジトリ上に作成されますが、表領域リポジトリには、データベース間の表領域の移動またはコピーに必要なファイルのみが含まれます。表領域リポジトリには様々な表領域セットを格納でき、特定の表領域セットの異なるバージョンも格納できます。表領域リポジトリ内の 1 つのバージョンの表領域セットは、次のファイルで構成されます。

- 表領域セットのデータ・ポンプ・エクスポート・ダンプ・ファイル
- データ・ポンプのエクスポート・ログ・ファイル
- 表領域セットを構成するデータファイル

関連項目：『Oracle Streams 概要および管理』

表領域の別のデータベースへの移動またはコピー方法

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

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

COMPATIBLE 初期化パラメータが 10 以上に設定された Oracle Database では、最初に表領域のデータファイルをオープンするときに、各ファイルが属するプラットフォームが識別されます。これらのファイルではファイル・ヘッダー・ブロックのディスク・フォーマットが同一で、このブロックがファイルの識別と検証に使用されます。読取り専用ファイルとオフライン・ファイルは、読取り / 書込みモードにするかオンライン化されると互換性レベルが上がります。このため、Oracle Database 10*g* より前の読取り専用の表領域は、プラットフォーム間のトランスポータブル機能を使用する前に、少なくとも 1 回は読取り / 書込みにする必要があります。

注意： ローカル管理の SYSTEM 表領域を持つデータベースには、ディクショナリ表領域を作成できません。トランスポータブル機能を使用するとディクショナリ管理表領域をプラグインできますが、書込み可能にすることはできません。

関連項目：

- プラットフォーム間の表領域の転送も含めて、表領域を別のデータベースに移動またはコピーする方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- インポート / エクスポートの詳細は、『Oracle Database ユーティリティ』を参照してください。
- DBMS_FILE_TRANSFER パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ファイルをコピーまたは転送する方法の詳細は、『Oracle Streams 概要および管理』を参照してください。

データファイルの概要

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

Oracle Database では、表領域のデータファイルを作成する際に、指定されたディスク領域に加えてファイル・ヘッダーのために必要なオーバーヘッドの領域が割り当てられます。データファイルが作成されると、Oracle Database を実行するオペレーティング・システムで、ファイルから古い情報や認可を消去した後、Oracle Database に割り当てます。ファイルが大きい場合には、この処理にかなりの時間がかかることがあります。どのデータベースでも最初の表領域は常に SYSTEM 表領域です。このため、Oracle Database では、データベースの作成時に、SYSTEM 表領域にデータベースの最初のデータファイルが自動的に割り当てられます。

この項の内容は、次のとおりです。

- データファイルの内容
- データファイルのサイズ
- オフライン・データファイル
- 一時データファイル

> **関連項目：** 各オペレーティング・システムでのデータファイルのファイル・ヘッダーに必要な領域の詳細は、オペレーティング・システム固有の Oracle Database のマニュアルを参照してください。

データファイルの内容

データファイルが初めて作成されたときに、割り当てられたディスク領域がフォーマットされますが、ユーザー・データは含まれません。Oracle Database では、対応付けられた表領域の将来のセグメントのデータを保持する領域が確保されます。これは Oracle Database 専用の領域です。Oracle Database では、表領域内のデータが増加すると、対応付けられたデータファイル内の空き領域を使用してセグメントにエクステントを割り当てます。

表領域内のスキーマ・オブジェクトに対応付けられたデータは、物理的には、表領域を構成する 1 つ以上のデータファイルに格納されます。スキーマ・オブジェクトは、特定のデータファイルには対応していないことに注意してください。データファイルは、特定の表領域内の任意のスキーマ・オブジェクトのデータ用のリポジトリです。Oracle Database では、スキーマ・オブジェクトに対応付けられたデータの領域は、表領域の 1 つ以上のデータファイル内に割り当てられます。このため、1 つのスキーマ・オブジェクトが 1 つ以上のデータファイルにまたがる場合があります。表の**ストライプ化**（データが複数のディスク上に分散されている状態）を使用しないかぎり、データベース管理者とエンド・ユーザーは、どのデータファイルにスキーマ・オブジェクトが格納されるかを制御できません。

> **関連項目：** 領域使用の詳細は、第 2 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。

データファイルのサイズ

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

注意： 拡張するには、オペレーティング・システムに十分な領域が必要です。

関連項目： データファイルのサイズ変更の詳細は、『Oracle Database 管理者ガイド』を参照してください。

オフライン・データファイル

SYSTEM 以外の表領域は、いつでもオフライン化またはオンライン化できます。表領域をオフライン化またはオンライン化すると、その表領域を構成するすべてのデータファイルは 1 単位としてオフライン化またはオンライン化されます。

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

一時データファイル

ローカル管理の一時表領域には、次の点を除き、通常のデータファイルと同様の一時データファイル（**一時ファイル**）があります。

- 一時ファイルは、常に NOLOGGING モードに設定されます。
- 一時ファイルを読取り専用にすることはできません。
- ALTER DATABASE 文では一時ファイルを作成できません。
- メディア・リカバリでは一時ファイルが認識されません。
  - BACKUP CONTROLFILE では、一時ファイルに関する情報が生成されません。
  - CREATE CONTROLFILE では、一時ファイルに関する情報を指定できません。
- 一時ファイルを作成またはサイズ変更する場合、指定したファイル・サイズのディスク領域が常に保証されるとはかぎりません。ある種のファイル・システム（たとえば、UNIX）でのディスク・ブロックは、ファイルの作成またはファイルのサイズ変更時ではなく、ブロックがアクセスされる前に割り当てられます。

  注意： このため、一時ファイルの作成とサイズ変更が素早く行えますが、一時ファイルにアクセスしたときに、ディスク領域が不足する可能性があります。

- 一時ファイルの情報は、ディクショナリ・ビュー DBA_TEMP_FILES と動的パフォーマンス・ビュー V$TEMPFILE には表示されますが、DBA_DATA_FILES や V$DATAFILE ビューには表示されません。

関連項目： ローカル管理表領域の詳細は、3-11 ページの「表領域内の領域管理」を参照してください。

制御ファイルの概要

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。データベースの使用中に、制御ファイルは Oracle Database によって絶えず更新されるため、データベースがオープンされている間は書込み可能になっている必要があります。なんらかの理由で制御ファイルにアクセスできない場合、データベースは正常に機能できなくなります。

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。

この項の内容は、次のとおりです。

制御ファイルの内容

制御ファイルには、起動時と通常の操作時に必要な、インスタンスからアクセス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 Database のみです。データベース管理者やユーザーは、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 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・ピースの情報
- バックアップ・データファイルと REDO ログ情報
- データファイルのコピーの情報
- カレントのログ順序番号
- チェックポイント情報

データベース名とタイムスタンプは、データベース作成時に作成されます。データベース名には、`DB_NAME` 初期化パラメータに指定した名前、または `CREATE DATABASE` 文で使用した名前が使用されます。

データベースのデータファイルや 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。

- Oracle Database の起動時にオープンするデータファイルと REDO ログ・ファイルの識別
- Oracle Database のリカバリが必要になった場合、必要なファイルや使用可能なファイルの識別

したがって、データベースの物理構造を変更した後は（`ALTER DATABASE` 文を使用）、ただちに制御ファイルのバックアップを作成してください。

制御ファイルには、チェックポイントに関する情報も記録されます。チェックポイント・プロセス（CKPT）は、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。Oracle Database では、データベースをリカバリする際にこの情報を使用して、REDO ログ・グループの特定のポイントより前に記録されている REDO エントリはデータベースのリカバリには不要である（すでにデータファイルに書き込まれている）ことを指定します。

関連項目： データベースの制御ファイルのバックアップを作成する方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

多重制御ファイル

REDO ログ・ファイルと同様に、Oracle Database では、同じデータベースに対する同一の制御ファイルを同時に複数オープンし、書き込むことができます。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、Oracle Database が破損した制御ファイルへのアクセスを試みたときに現行インスタンスが失敗します。ただし、現行の制御ファイルのコピーを他のディスクに保存してあれば、データベースをリカバリしなくてもインスタンスを再起動できます。

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場合は、インスタンスが異常終了し、メディア・リカバリが必要になります。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場合、メディア・リカバリは容易ではありません。次の原則を遵守することをお薦めします。

- 各データベースで多重制御ファイルを使用する
- 各コピーを異なる物理ディスクに格納する
- オペレーティング・システムのミラー化機能を使用する
- バックアップを監視する

4

トランザクションの管理

この章では、トランザクションを定義し、トランザクションを使用して作業を管理する方法について説明します。

この章の内容は、次のとおりです。

トランザクションの概要

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

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

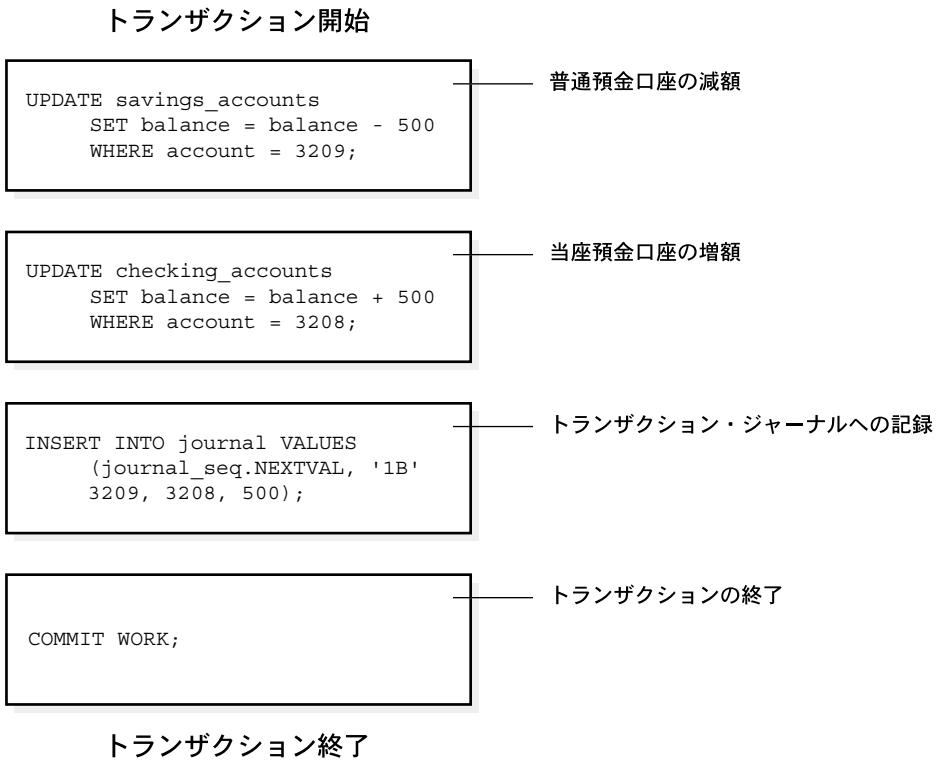
トランザクションの概念を具体的に説明するため、銀行業務データベースについて考えてみます。銀行の顧客が普通預金口座から当座預金口座へ預金を振り替える場合、トランザクションは次の 3 つの別々の操作で構成されます。

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

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

図 4-1 に銀行のトランザクションの例を示します。

図 4-1 銀行のトランザクション



この項の内容は、次のとおりです。

文の実行とトランザクションの制御

正常に実行される SQL 文とコミットされるトランザクションとは異なります。正常に実行されるとは、単一の文が次の条件を満たしたことを意味します。

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

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

コミットとは、ユーザーが明示的または暗黙的に、このトランザクションの変更内容を確定するように要求することを意味します。明示的な要求は、ユーザーが `COMMIT` 文を発行すると発生します。暗黙的な要求は、アプリケーションの正常終了後またはデータ定義言語（DDL）操作の完了後に発生します。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。トランザクションのコミット後に発行された問合せでは、コミットされた変更を参照できます。

トランザクションを開始する前に、`SET TRANSACTION ... NAME` 文を使用して、トランザクションに名前を付けることができます。名前を付けることで、長時間実行のトランザクションの監視およびインダウト分散トランザクションの解決が容易になります。

関連項目： 4-7 ページの「トランザクションの命名」

文レベルのロールバック

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

SQL 文の**実行中**にエラーが検出されると、文レベルのロールバックが発生します。この種のエラーの一例として、主キーに重複値を挿入しようとした場合があります。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。**解析中**に構文エラーなどのエラーが検出された場合は、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。

SQL 文でエラーが発生した場合は、SQL での実行予定の作業のみが影響を受けます。現行のトランザクションにおけるこれまでの作業に影響を及ぼすことはありません。文が DDL 文の場合、その文の直前に実行された暗黙的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的セーブポイントを直接参照できません。

関連項目： 13-16 ページの「デッドロック」

再開可能領域割当て

Oracle Database では、領域割当てエラーが発生した場合、大規模なデータベース操作の実行を一時停止および再開することが可能です。このため、Oracle データベース・サーバーからユーザーにエラーを戻すかわりに、管理者は対処措置を取ることができます。エラー条件が修正されると、一時停止中の操作が自動的に再開されます。

再開可能モードで文が実行されるのは、クライアントが `ALTER SESSION` 文を使用して、明示的にセッションの再開可能セマンティクスを使用可能にした場合のみです。

次のような場合、再開可能領域割当ては一時停止されます。

- 領域が足りない場合
- エクステントの最大値に達した場合
- 領域割当て制限を超えた場合

非再開可能領域割当ての場合、これらの条件ではエラーが発生し、文がロールバックされます。

文を一時停止すると、トランザクションが自動的に一時停止されます。このため、トランザクションのすべてのリソースが、一時停止文および再開文を介して保持されます。

エラー条件が（たとえば、ユーザーの介入、または他の問合せによりソート領域が解放された結果として）消滅すると、一時停止中の文が自動的に実行を再開します。

> **関連項目：** 再開可能領域割当てを使用可能にする方法、修正可能な条件および再開可能にすることが可能な文の詳細は、『Oracle Database 管理者ガイド』を参照してください。

トランザクション管理の概要

Oracle Database 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 Database はそのトランザクションを使用可能な UNDO 表領域に割り当てて、新しいトランザクションのロールバック・エントリを記録します。

トランザクションは、次のいずれかの状況が発生すると終了します。

- `COMMIT` 文または `ROLLBACK` 文が、`SAVEPOINT` 句なしで発行される場合。
- `CREATE`、`DROP`、`RENAME` または `ALTER` などの DDL 文が実行される場合。カレント・トランザクションに DML 文が含まれている場合、Oracle Database 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 Database の接続を切断する場合。カレント・トランザクションはコミットされます。
- ユーザー・プロセスが異常終了する場合。カレント・トランザクションはロールバックされます。

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

この項の内容は、次のとおりです。

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取り消す必要があります。

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ます。

- Oracle Database により UNDO 情報が生成されています。この UNDO 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含まれます。
- Oracle Database 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ています。REDO ログ・レコードには、データ・ブロックおよびロールバック・ブロックに対する変更内容が含まれています。これらの変更は、トランザクションがコミットされる前にディスクに移動することがあります。
- SGA のデータベース・バッファに変更が加えられます。これらの変更は、トランザクションが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、バックグラウンド・プロセスであるデータベース・ライター（DBW*n*）によりデータファイルに必ずしもすぐに書き込まれるわけではありません。データベースにとって最も効率的な時点で書き込まれます。この書込みは、トランザクションがコミットされる前に実行されたり、トランザクションがコミットされた後しばらくしてから実行されます。

トランザクションをコミットすると、次の処理が実行されます。

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

注意： デフォルトの動作では、LGWR により REDO がオンライン REDO ログ・ファイルに同時に書き込まれ、トランザクションは REDO がディスクに移動するまで待機してから、コミットをユーザーに戻します。ただし、アプリケーション開発者は、トランザクションのコミット待機時間を短縮するために、REDO が非同期に書き込まれ、REDO がディスクに移動するまでトランザクションが待機する必要がないよう指定できます。

関連項目：

- 非同期コミットの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 13-3 ページの「ロックのメカニズムの概要」
- バックグラウンド・プロセスである LGWR および DBW*n* の詳細は、9-4 ページの「Oracle Database プロセスの概要」を参照してください。

トランザクションのロールバック

ロールバックとは、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 Database は、UNDO 表領域（またはロールバック・セグメント）を使用して古い値を格納します。REDO ログには、変更の履歴が記録されます。

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

次のすべてのタイプのロールバックで、同じ手順が使用されます。

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

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

1. Oracle Database 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 UNDO 表領域を使用して取り消されます。
2. Oracle Database により、そのトランザクションのすべてのデータ・ロックが解放されます。
3. トランザクションが終了します。

関連項目：

- 4-6 ページの「トランザクションのセーブポイント」
- 13-3 ページの「ロックのメカニズムの概要」
- リカバリ中にコミット済およびコミットされていない変更に対して実行される処理の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

トランザクションのセーブポイント

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、ロング・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、ロング・トランザクション内の任意のポイントで作業に任意にマークを設定できます。これにより、そのトランザクション内の宣言されたセーブポイントからトランザクション内の現時点までの間に実行された作業を、後でロールバックできるようになります。たとえば、大規模で複雑な一連の更新のどこにでもセーブポイントを設定できるため、エラーが発生してもすべての文を再発行する必要はありません。

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。そうすると、あるファンクションが失敗しても、そのファンクション開始前の状態にデータを復帰し、パラメータを修正してから再実行したり、リカバリ作業を実行するのが容易になります。

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

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

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

トランザクションはアクティブであり、継続できます。

セッションが待機中の場合は、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でも行のロックは解放されません。トランザクションがロックを取得できない場合に異常終了しないように、`UPDATE` または `DELETE` 文を発行する前に、`FOR UPDATE ... NOWAIT` を使用します。（これは、ロールバックしたセーブポイントの前に取得されたロックを参照します。セーブポイントの後に実行された文は完全にロールバックされているため、このセーブポイントの後に取得された行ロックが解放されます。）

トランザクションの命名

トランザクションの名前には、簡単で覚えやすいテキスト文字列を使用できます。この名前は、トランザクションの目的を連想できるように命名します。分散トランザクションのコミット・コメントをトランザクションの名前に置換することには、次の利点があります。

- 名前を付けることで、長時間実行のトランザクションの監視およびインダウト分散トランザクションの解決が容易になります。
- アプリケーションのトランザクション ID とともにトランザクション名を表示できます。たとえば、システムの活動状況を監視する場合、データベース管理者は Enterprise Manager 内のトランザクション名を表示できます。
- 互換性が Oracle9*i* 以上に設定されている場合、トランザクション名は、トランザクション監査 REDO レコードに書き込まれます。
- LogMiner は、トランザクション名を使用して、REDO ログ内のトランザクション監査レコードから特定のトランザクションを検索できます。
- トランザクション名を使用して、`V$TRANSACTION` などのデータ・ディクショナリ・ビュー内の特定のトランザクションを検索できます。

この項の内容は、次のとおりです。

- トランザクションの命名方法
- コミット・コメント

トランザクションの命名方法

トランザクションを開始する前に、`SET TRANSACTION ... NAME` 文を使用して、トランザクションに名前を付けます。

トランザクションに名前を付けるときは、ID と対応付けます。トランザクション名は一意の必要がないので、所有者が複数のトランザクションに同じトランザクション名を付けることも可能です。トランザクションを識別できるように任意の名前を付けることができます。

コミット・コメント

以前のリリースでは、コミット・コメントを使用してコメントをトランザクションに対応付けることができました。ただし、コメントをトランザクションに対応付けられるのは、トランザクションがコミットされているときのみです。

下位互換性を維持するため、コミット・コメントは引き続きサポートされます。ただし、トランザクション名を使用することをお薦めします。コミット・コメントは名前付きトランザクションでは処理されません。

注意：将来のリリースでは、コミット・コメントは使用できなくなります。

関連項目：

- 分散トランザクションの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- トランザクション命名構文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

2 フェーズ・コミット・メカニズム

分散データベースでは、Oracle Database 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整合性が保たれるようにする必要があります。

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

2 フェーズ・コミット・メカニズムは、分散トランザクションに関係するすべてのデータベース・サーバーが、そのトランザクション内の文のすべてをコミットするか、またはすべてを取り消す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的 DML 操作も保護します。

Oracle Database 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` 文は、自動的に 2 フェーズ・コミット・メカニズムをトリガーしてトランザクションをコミットします。データベース・アプリケーション本体に分散トランザクションを含めるのに、コーディングや複雑な構文の記述は必要ありません。

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、各ローカルの Oracle データベースの `RECO` プロセスが、関係するすべてのノード上でインダウト分散トランザクションを自動的にコミットするか、またはロールバックします。

長時間にわたる障害が発生した場合、Oracle Database では、各ローカル管理者が障害によって発生したインダウト分散トランザクションを手動でコミットまたは取り消すことができます。このオプションによって、ローカル・データベース管理者は、長時間にわたる障害の結果として無期限にロックされる可能性のあるリソースを解放できます。

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

関連項目： 『Oracle Database Heterogeneous Connectivity 管理者ガイド』

自律型トランザクションの概要

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離れてなんらかの SQL 操作を実行し、その操作をコミットまたは取り消してから、コール側トランザクションのコンテキストに戻って引き続き実行できます。

起動後の自律型トランザクションは、コール側のメイン・トランザクションの影響をまったく受けません。メイン・トランザクションによって加えられたがコミットされていない変更は取り扱わず、ロックやリソースをメイン・トランザクションと共有することもありません。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 Database は、この種のデッドロックを検出するとエラーを戻します。アプリケーション開発者は、デッドロック状況を回避する責任があります。

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

自律型 PL/SQL ブロック

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

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

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

自律型ブロックが他の自律型ブロックまたはそれ自身を起動する場合、コールされたブロックが、コール側ブロックとトランザクション・コンテキストを共有することはありません。ただし、自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取り消されません。

関連項目：『Oracle Database PL/SQL 言語リファレンス』

5

スキーマ・オブジェクト

この章では、ユーザー・スキーマに含まれる様々な種類のデータベース・オブジェクトについて説明します。

この章の内容は、次のとおりです。

スキーマ・オブジェクトの概要

スキーマとは、データの論理構造の集合、すなわちスキーマ・オブジェクトのことです。スキーマはデータベース・ユーザーによって所有され、そのユーザーと同じ名前を持ちます。各ユーザーが単一のスキーマを所有します。スキーマ・オブジェクトは、SQL で作成および操作できます。スキーマ・オブジェクトには、次のタイプのオブジェクトがあります。

- クラスタ
- 制約
- データベース・リンク
- データベース・トリガー
- ディメンション
- 外部プロシージャ・ライブラリ
- 索引および索引タイプ
- Java クラス、Java リソースおよび Java ソース
- マテリアライズド・ビューおよびマテリアライズド・ビュー・ログ
- オブジェクト表、オブジェクト型およびオブジェクト・ビュー
- 演算子
- 順序
- ストアド・ファンクション、プロシージャおよびパッケージ
- シノニム
- 表および索引構成表
- ビュー

データベースには他に、次のタイプのオブジェクトも保存されており、SQL で作成および操作できますが、スキーマには含まれていません。

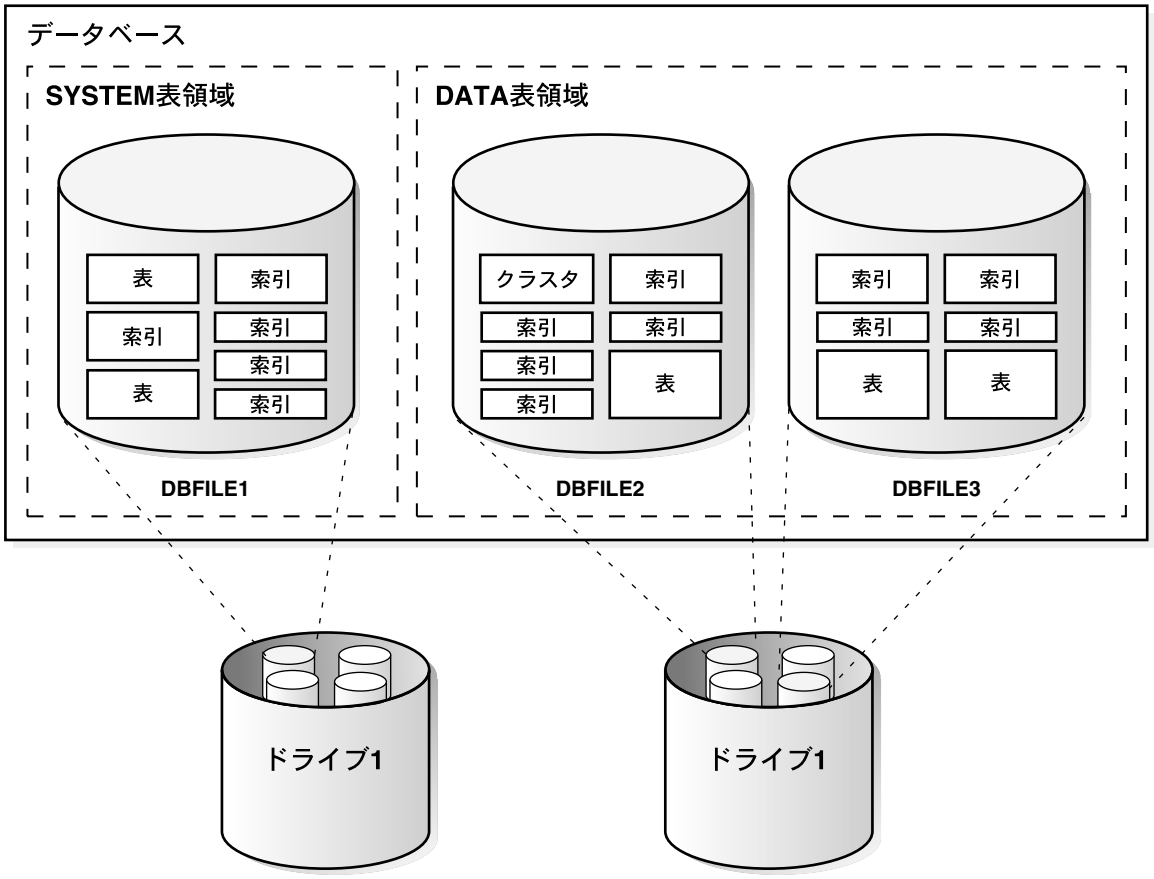
- コンテキスト
- ディレクトリ
- パラメータ・ファイル（PFILE）およびサーバー・パラメータ・ファイル（SPFILE）
- プロファイル
- ロール
- ロールバック・セグメント
- 表領域
- ユーザー

スキーマ・オブジェクトは、論理的なデータ記憶域構造です。各スキーマ・オブジェクトは、その情報を格納するディスク上の物理ファイルと 1 対 1 では対応しません。Oracle Database では、スキーマ・オブジェクトはデータベースの表領域内に論理的に格納されます。各オブジェクトのデータは、物理的には、表領域の 1 つ以上のデータファイルに格納されます。Oracle Database では、表、索引およびクラスタなどのいくつかのオブジェクトに関して、表領域のデータファイル内のオブジェクトに割り当てるディスク領域を指定できます。

スキーマと表領域には関連がありません。表領域は異なるスキーマのオブジェクトを含むことができ、スキーマの各オブジェクトは異なる表領域に含めることができます。

図 5-1 に、オブジェクト、表領域およびデータファイルの関連を示します。

図 5-1 スキーマ・オブジェクト、表領域およびデータファイル



関連項目：『Oracle Database 管理者ガイド』

表の概要

表は、Oracle データベースにおけるデータ記憶域の基本単位です。データは**行**と**列**に格納されます。表は、**表名**（employees など）と列の集合で定義されます。各列には、**列名**（employee_id、last_name および job_id など）、**データ型**（VARCHAR2、DATE または NUMBER など）および**幅**を割り当てます。幅は、DATE データ型の場合のようにデータ型によって事前に決定されている場合があります。列が NUMBER データ型の場合は、幅のかわりに**精度**と**スケール**を定義します。

各列に**整合性制約**と呼ばれるルールを指定できます。たとえば、NOT NULL 整合性制約では、列の各行には値が含まれている必要があります。

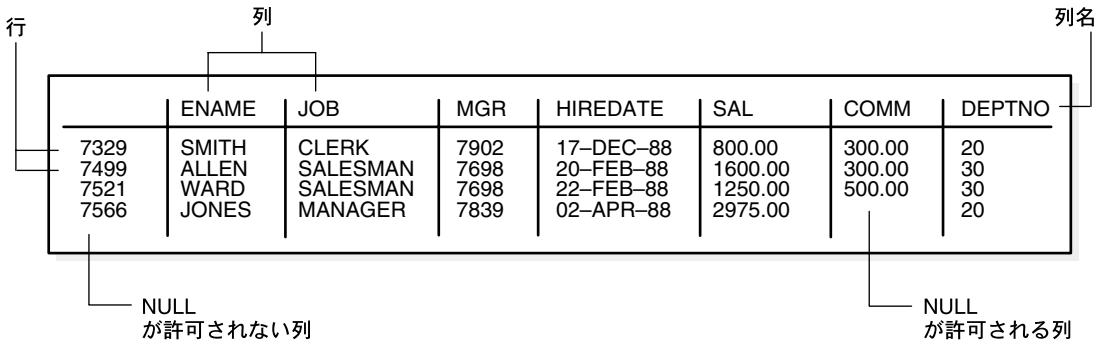
表には**仮想列**を含めることができます。仮想列は通常の列とは異なり、ディスク上の領域を使用しません。詳しく説明すると、データベースによりユーザー指定の一連の式または関数が計算され、必要に応じて仮想列から値が導出されます。仮想列は問合せ、DML および DDL 文で使用できます。仮想列の索引付け、統計の収集および整合性制約の作成が可能です。このため、通常の列と同じように扱われます。

また、データファイルに格納される前にデータを暗号化する表の列を指定できます。暗号化を使用すると、ユーザーがオペレーティング・システム・ツールにより直接データファイル内部を参照して、データベース・アクセス制御メカニズムから逃れることを防止できます。

表の作成後に、SQL 文を使用してデータ行を挿入します。行は、1 つのレコードに対応する列情報の集まりです。挿入した表データは、SQL を使用して問合せ、削除または更新できます。

図 5-2 にサンプル表を示します。

図 5-2 EMP 表



この項の内容は、次のとおりです。

- 表データの格納方法
- 表の圧縮
- 値がないことを意味する NULL
- 列のデフォルト値
- パーティション表
- ネストした表
- 一時表
- 外部表

関連項目：

- 表の管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 透過的なデータ暗号化の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。
- 仮想列の参照情報は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 第 26 章「Oracle データ型」
- 第 21 章「データ整合性」

表データの格納方法

Oracle Database では、表を作成する際に、表の将来のデータを保持するために、表領域にデータ・セグメントが自動的に割り当てられます。表のデータ・セグメントに対する領域の割当てと使用は、次の方法で制御できます。

- データ・セグメントに対して記憶域パラメータを設定します。これにより、データ・セグメントのエクステントの領域の容量を制御できます。
- データ・セグメントに対して、PCTFREE パラメータと PCTUSED パラメータを設定します。これにより、データ・セグメントのエクステントを構成するデータ・ブロック内の空き領域の使用を制御できます。

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

表のデータ・セグメント（クラスタ表を扱っている場合はクラスタ用のデータ・セグメント）は、表所有者のデフォルト表領域、または CREATE TABLE 文で指定された特定の表領域に作成されます。

関連項目： 2-7 ページの「PCTFREE、PCTUSED と行連鎖」

この項の内容は、次のとおりです。

行の形式とサイズ

次のような場合、表の行データが大きすぎて、1 つのデータ・ブロック内に収まらない場合があります。

- 最初に行を挿入するとき、大きすぎて 1 つのデータ・ブロック内に収まらない場合

  Oracle Database では、行連鎖により、その行のデータは、セグメント用に確保された 1 つ以上のデータ・ブロックの**連鎖**に格納されます。多くの場合、行連鎖は、行が大きい場合に発生します。大きい行の例には、LONG または LONG RAW データ型の列を含む行、2KB ブロックに VARCHAR2(4000) の列が含まれる行、または行に含まれる列の数が多すぎる場合があります。そのような場合の行連鎖は、避けられません。

- 1 つのデータ・ブロックに収まっていた行が更新されて、行全体の長さが増加したが、更新後の行を保持する十分な空き領域がない場合

  Oracle Database では、行の移行により、行全体が新しいブロックに収まることを前提として、行全体が新しいデータ・ブロックに移動されます。移行された行の元の行断片には、行の移行先の新しいブロックへのポインタまたは「転送先アドレス」が含まれます。そのため、移動された行の ROWID は変わりません。

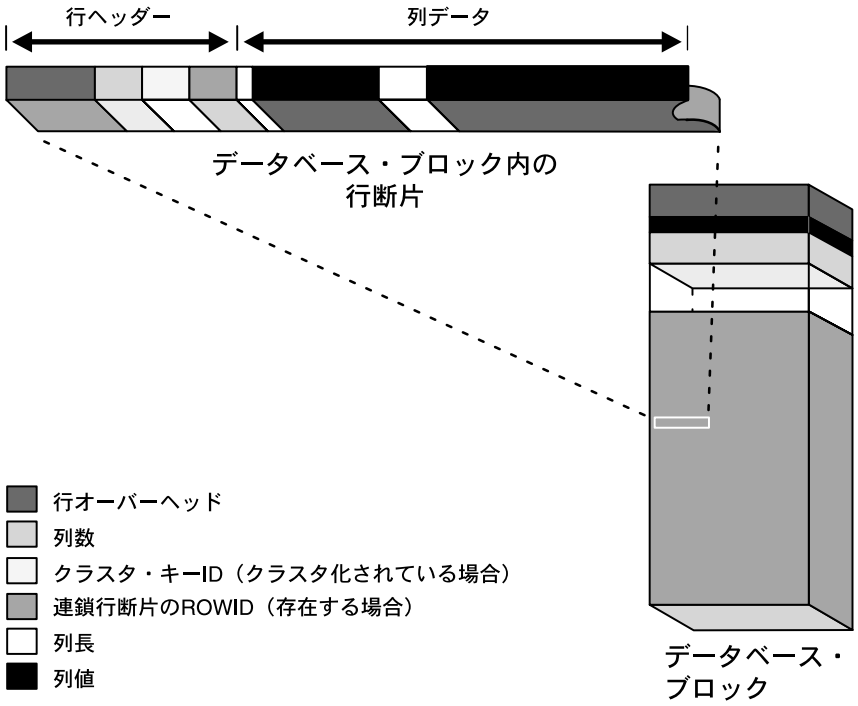
- 行に 255 を超える列が含まれる場合

  Oracle Database では、1 つの行断片に格納できる列数は、255 列のみです。したがって、1000 列の行を表に挿入すると、データベースでは通常、複数のブロックに連鎖された 4 つの行断片が作成されます。

行が連鎖または移行されると、行の情報を取り出す際に、Oracle Database が複数のデータ・ブロックをスキャンする必要があるため、データの取出しに必要な I/O が増加します。たとえば、索引の読取りに対して 1 回の I/O を行い、移行されていない行の表を読み取るために 1 回の I/O を行う場合、データベースでは、移行された行の実際の行データを取得するために追加の I/O が必要になります。

それぞれの行断片は、連鎖されていても連鎖されていなくても、行の全部または一部の列の**行ヘッダー**とデータを含みます。また、個々の列が複数の行断片にまたがり、結果として複数のデータ・ブロックにまたがっている場合もあります。図 5-3 に、行断片の形式を示します。

図 5-3 行断片の形式



行ヘッダーは、データの前に位置し、次のような情報を含んでいます。

- 行断片
- 連鎖（連鎖された行断片の場合のみ）
- 行断片内の列
- クラスタ・キー（クラスタ化されたデータの場合のみ）

1 つのブロックに完全に収まる行は、最低 3 バイトの行ヘッダーを持っています。各行は、行ヘッダー情報の後に列の長さとデータを格納します。列の長さは、250 バイト以下のデータを格納する列については 1 バイト、250 バイトよりも大きなデータを格納する列については 3 バイトを必要とし、列データの直前に位置します。列データのために必要となる領域は、データ型によって異なります。列のデータ型が可変長である場合、値を保持するために必要な領域は、データの更新によって変動します。

領域を節約するために、NULL を含む列は、列の長さ（ゼロ）のみを格納します。Oracle Database では、NULL 列のデータは格納されません。また、Oracle Database では、末尾にある NULL 列の場合は、列の長さも格納されません。

注意： また、各行は、データ・ブロック・ヘッダーの行ディレクトリで 2 バイトを使用します。

クラスタ化された行には、クラスタ化されていない行と同じ情報が格納されます。また、クラスタ化された行には、それらの行が属するクラスタ・キーを参照する情報も格納されます。

関連項目：

- クラスタ化された行や表の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 5-39 ページの「クラスタの概要」
- 2-6 ページの「行連鎖と移行」
- 5-8 ページの「値がないことを意味する NULL」
- 2-4 ページの「行ディレクトリ」

行断片の ROWID

ROWID は、それぞれの行断片を位置またはアドレスで識別します。状況によっては、ROWID が行断片に割り当てられた後、ROWID が変更される場合があります。たとえば、行の移動が可能な場合、ROWID は、パーティション・キーの更新、フラッシュバック表の操作、表の縮小操作などのために変更される可能性があります。行の移動が無効な場合は、Oracle Database ユーティリティを使用して行をエクスポートおよびインポートした場合に ROWID が変更される可能性があります。

関連項目： 26-15 ページの「物理 ROWID」

列の順序

1 つの表内のすべての行は、列の順序が同一です。列は通常、`CREATE TABLE` 文でリストされた順序で格納されますが、これは保証されません。たとえば、表に `LONG` データ型の列があると、Oracle Database では、この列は常に最後に格納されます。また、表を変更して新しい列を追加すると、その新しい列は、最後に格納されます。

一般的には、行に必要な領域を少なくするために、NULL を頻繁に格納する列を最後の列にします。ただし、作成する表に `LONG` 列が含まれている場合、NULL 列を最後に置く利点はなくなります。

表の圧縮

Oracle Database の表の圧縮機能は、データベース・ブロック内の重複値の排除に基づいています。データベース・ブロック（ディスク・ページと呼ぶこともあります）に格納された圧縮データは自己完結型です。つまり、ブロック内の非圧縮データを再作成するために必要な情報は、すべてそのブロック内で使用できます。ブロックのすべての行と列にある重複値は、いったんブロックの先頭に格納されます。これを、そのブロックのシンボル表と呼びます。この種の値のすべての出現箇所は、シンボル表の短い参照で置き換えられます。

先頭にあるシンボル表を除き、圧縮データベース・ブロックは通常のデータベース・ブロックとほぼ同じです。通常のデータベース・ブロックに対して動作するデータベース機能とファンクションは、すべて圧縮データベース・ブロックに対しても動作します。圧縮可能なデータベース・オブジェクトは、表やマテリアライズド・ビューなどです。パーティション表の場合は、一部またはすべてのパーティションを圧縮するように選択できます。表領域、表または表のパーティションについて、圧縮属性を宣言できます。表領域レベルで宣言すると、その表領域に作成されたすべての表がデフォルトで圧縮されます。表（またはパーティションや表領域）の圧縮属性を変更すると、変更はその表に送られる新規データにのみ適用されます。そのため、1 つの表またはパーティションに圧縮ブロックと通常のブロックの両方が含まれる場合があります。これにより、圧縮によってデータ・サイズが大きくならないことが保証されます。ブロック・サイズが大きくなる場合、そのブロックに圧縮は適用されません。

表の圧縮の使用

圧縮は、圧縮された表に対してデータの挿入、更新、バルク挿入またはバルク・ロードが実行されている間に行われます。この種の操作には次のものが含まれます。

- ダイレクト・パス SQL*Loader
- CREATE TABLE および AS SELECT 文
- パラレル INSERT（または APPEND ヒントを指定したシリアル INSERT）文
- 単一行挿入または配列挿入
- 単一行更新または配列更新

データベースにある既存のデータも、ALTER TABLE および MOVE 文を介して圧縮形式に移行できます。この操作では表に排他ロックが適用されるため、操作が完了するまでは更新およびロードできなくなります。この状況を許容できない場合は、Oracle Database オンライン再定義ユーティリティ（DBMS_REDEFINITION PL/SQL パッケージ）を使用できます。

データ圧縮は、LOB のすべての改良型と、行の外に格納される VARRAY や CLOB に格納される XML データ型など、LOB から導出されるデータ型を除く、すべてのデータ型に使用できます。

表の圧縮は、データベースへのデータのバルク・ロードや、単一行挿入または配列挿入、単一行更新または配列更新の際に実行されます。圧縮に関連したオーバーヘッドは、その時点でほぼ参照可能になります。このオーバーヘッドは、圧縮を検討する際に考慮する必要のある重要なトレードオフです。

圧縮された表またはパーティションは、Oracle Database の他の表やパーティションと同様に変更できます。圧縮されたデータの削除には、圧縮されていないデータの削除と同じだけ時間がかかります。新規データの挿入についても同様です。圧縮データの更新速度は、低下する場合があります。Oracle Database では、圧縮された表に対するすべての DML 操作（挿入、更新、削除）をサポートするため、表の圧縮は、データ・ウェアハウス・アプリケーションのみでなく OLTP アプリケーションにも適しています。これら両方の環境では、データは、その読取り専用部分やほとんど変更されない部分（履歴データなど）の圧縮状態が維持されるように編成する必要があります。

値がないことを意味する NULL

NULL は、ある行のある列に値が入っていないことを意味します。NULL は、データがない、不明である、または適切でないことを示します。他の値（ゼロなど）を暗示する目的では、NULL は使用できません。NOT NULL 整合性制約または主キー整合性制約が列に対して定義されていない場合にかぎり、列に NULL 値を入力できます。これらの制約が列に対して定義されている場合、その列に値を持たない行は挿入できません。

データ値を持つ 2 つの列の間にはさまれた NULL はデータベースに格納されます。このような場合、NULL には列の長さ（ゼロ）を格納する 1 バイトのみが必要となります。

行の末尾にある NULL には、記憶域は不要です。これは、新しい行のヘッダーが、前行の残りの列が NULL であることを知らせるためです。たとえば、表の最後の 3 列が NULL であれば、その 3 列には情報は格納されません。列が多い表では、ディスク領域を節約するため、NULL を含む可能性の高い列は最後に定義する必要があります。

NULL とその他の値との比較の大部分は、定義によって TRUE にも FALSE にもならず、UNKNOWN となります。SQL 文の中で NULL を識別するには、IS NULL 述語を使用します。NULL を NULL 以外の値に変換するには、SQL 関数の NVL を使用します。

クラスタ・キー列の値が NULL の場合または索引がビットマップ索引の場合を除いて、NULL に索引を付けることはできません。

関連項目：

- IS NULL および NVL 関数を使用した比較については、『Oracle Database SQL 言語リファレンス』を参照してください。
- 5-24 ページの「索引と NULL」
- 5-34 ページの「ビットマップ索引と NULL」

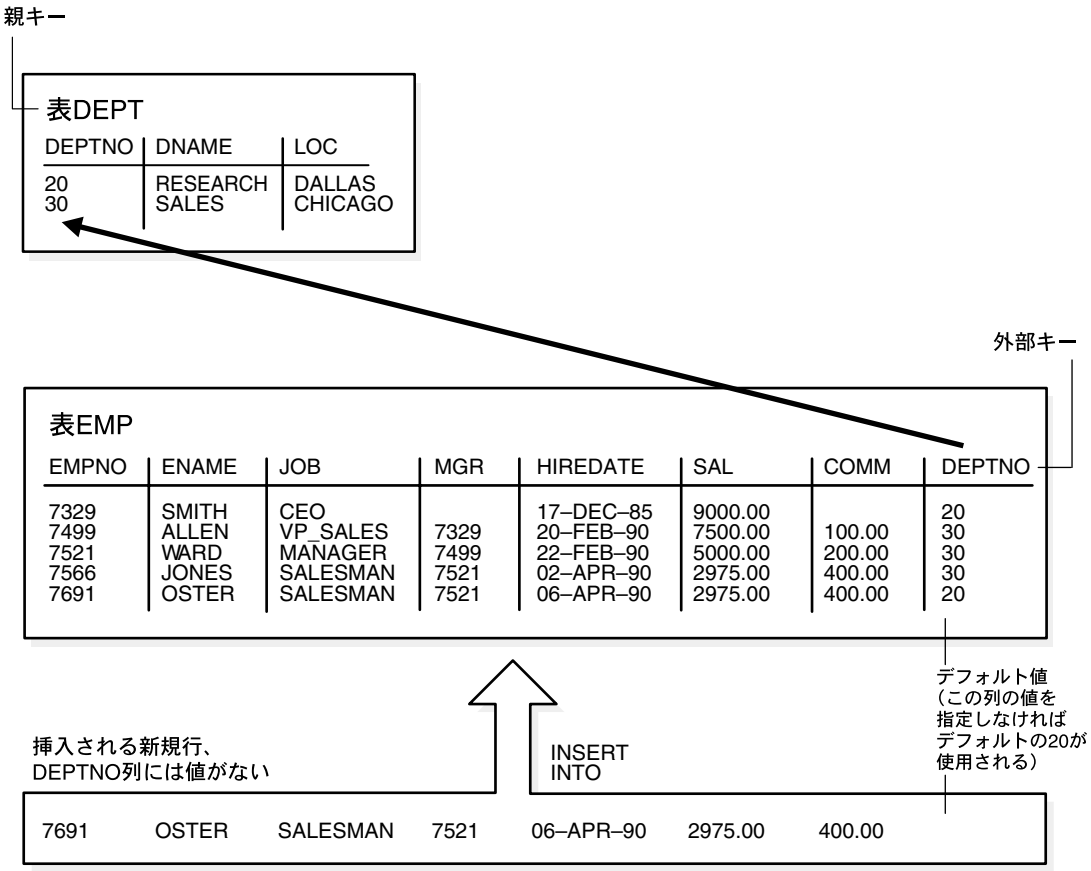
列のデフォルト値

列に対してデフォルト値が明示的に定義されていない場合、その列のデフォルトは暗黙的に NULL になります。新しい行が挿入され、列に対する値が省略されたか、キーワードの DEFAULT が提供された場合、デフォルト値が自動的に入力されるように、表の列にデフォルト値を割り当てることもできます。

列のデフォルト値は、実際に INSERT 文が値を指定している場合と同様に機能します。デフォルトのリテラルまたは式のデータ型は、その列のデータ型に一致しているか、あるいはそれに変換可能である必要があります。

デフォルト値を持つ行が挿入されると、整合性制約チェックが行われます。たとえば、図 5-4 では、従業員の部門番号に対する値を持たない行が、emp 表に挿入されています。部門番号に値が指定されていないため、Oracle Database により、deptno 列のデフォルト値の 20 が挿入されます。デフォルト値を挿入した後、Oracle Database では、deptno 列に定義された外部キー整合性制約のチェックが行われます。

図 5-4 デフォルトの列値



表の列のデータ型によっては、NOT NULL 制約およびデフォルト値の両方が設定された列を追加すると、データベースにより操作が最適化され、表が DML 用にロックされる時間が短縮される場合があります。

関連項目： 整合性制約の詳細は、第 21 章「データ整合性」を参照してください。

パーティション表

パーティション表では、データをパーティションやサブパーティションと呼ばれる小さく管理しやすい単位に分割できます。索引も同様にパーティション化できます。各パーティションは個別に管理し、他のパーティションから独立して操作できます。これにより、各パーティションの可用性とパフォーマンスに適合する構造を提供しています。

注意： ディスク使用量とメモリー使用量（特にバッファ・キャッシュ）を減少させるために、表とパーティション表をデータベースに圧縮形式で格納できます。通常は、これにより読取り専用操作のパフォーマンスが向上します。また、問合せの実行も高速化されます。ただし、CPU オーバーヘッドの面で少しコストがかかります。

関連項目：

- 16-9 ページの「表の圧縮」
- 『Oracle Database VLDB およびパーティショニング・ガイド』

ネストした表

データ型として別の表を指定した表を作成できます。つまり、表の列値として別の表を**ネスト**できます。Oracle データベース・サーバーは、親表の行の外にネストした表のデータを、ネストした表の列に対応付けた**格納表**を使用して格納します。親行には、ネストした表のインスタンスと対応付けられた一意の集合識別子が入れられます。

関連項目：

- ネストした表の詳細は、『Oracle Database オブジェクト・リレーショナル開発者ガイド』を参照してください。
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

一時表

Oracle Database では、永続的な表のみでなく、トランザクションまたはセッションの期間中にのみ存在するセッション用データを保持する**一時表**を作成できます。

`CREATE GLOBAL TEMPORARY TABLE` 文では、トランザクションまたはセッションに固有の一時表が作成されます。トランザクション固有の一時表では、データはトランザクションの期間中のみ存在します。セッション固有の一時表では、データはセッションの期間中のみ存在します。一時表には、セッションのプライベート・データが含まれます。各セッションで表示したり変更できるのは、そのセッションの独自データのみです。一時表のデータについては、DML ロックは取得されません。各セッションに固有のプライベート・データがあるため、一時表には `LOCK` 文は無効です。

セッション固有の一時表に発行された `TRUNCATE` 文は、独自のセッションのデータを切り捨てます。同じ表を使用した他のセッションのデータは切り捨てません。

一時表での DML 文では、データ変更の REDO ログは生成されません。ただし、データの UNDO ログと UNDO ログの REDO ログは生成されます。セッションの終了時には、一時表からデータが自動的に削除されます。たとえば、ユーザーがログオフした場合や、セッション障害またはインスタンス障害の発生時など、セッションが異常終了した場合です。

一時表の索引は、`CREATE INDEX` 文を使用して作成します。一時表に作成された索引も一時的で、その索引のデータは一時表のデータと同じセッションまたはトランザクション・スコープを持ちます。

一時表と永続表の両方にアクセスするビューを作成できます。また、一時表のトリガーを作成することも可能です。

Oracle Database ユーティリティは、一時表の定義をエクスポートおよびインポートできます。ただし、ROWS 句を使用しても、データ行はエクスポートされません。同様に、一時表の定義はレプリケートできますが、そのデータはレプリケートできません。

この項の内容は、次のとおりです。

セグメントの割当て

一時表は一時セグメントを使用します。永続表とは異なり、一時表とその索引の場合、作成時にセグメントが自動的に割り当てられることはありません。かわりに、最初の INSERT（または CREATE TABLE AS SELECT）の実行時にセグメントが割り当てられます。このため、最初の INSERT の前に SELECT、UPDATE または DELETE が実行されると、表は空のように見えます。

一時表で DDL 文（ALTER TABLE、DROP TABLE、CREATE INDEX など）を実行できるのは、バインドされているセッションがない場合のみです。セッションは、INSERT の実行時に一時表にバインドされます。セッションは、セッションの終了時に TRUNCATE によってアンバインドされます。またトランザクション固有の一時表の場合は COMMIT または ROLLBACK によってアンバインドされます。

一時セグメントは、トランザクション固有の一時表の場合はトランザクションの終了時に、セッション固有の一時表の場合はセッションの終了時に割当て解除されます。

関連項目： 2-13 ページの「一時セグメント内のエクステント」

親トランザクションと子トランザクション

トランザクション固有の一時表には、ユーザー・トランザクションとその子トランザクションからアクセスできます。ただし、2 つのトランザクションが、同じセッションで特定のトランザクション固有の一時表を同時に使用することはできません。異なるセッションであれば、複数のトランザクションが使用できます。

ユーザーのトランザクションが一時表への INSERT を実行すると、その子トランザクションはその後一時表を使用できなくなります。

子トランザクションが一時表への INSERT を実行すると、その子トランザクションの終了時に、一時表に対応するデータが削除されます。その後は、ユーザー・トランザクションまたは他の任意の子トランザクションから、一時表にアクセスできます。

外部表

外部表は、外部ソースのデータに対して、データベース内の表にあるデータのようにアクセスします。データベースに接続し、DDL を使用して外部表のメタデータを作成できます。外部表の DDL には、Oracle Database の列型を記述する部分と、外部データの Oracle Database データ列へのマッピングを記述する部分（アクセス・パラメータ）が含まれます。

外部表には、データベースに格納されているデータは示されません。また、外部ソースにおけるデータの格納方法も示されません。かわりに、外部表レイヤーでのサーバーに対するデータの表示方法が記述されています。外部表の定義に合うようにデータファイルのデータに必要な変換を行うのは、アクセス・ドライバおよび外部表レイヤーの役割です。

外部表は読取り専用なため、DML 操作はできず、索引は作成できません。また、仮想列もサポートされていません。

この項の内容は、次のとおりです。

- アクセス・ドライバ
- 外部表を使用したデータのロード
- 外部表へのパラレル・アクセス

アクセス・ドライバ

外部表の作成時には、その型を指定します。外部表には型ごとにアクセス・ドライバがあり、これにより外部表の型固有のアクセス・パラメータが提供されます。アクセス・ドライバにより、データ・ソースからのデータが外部表の定義に合うように処理されることを保証されます。

外部表のコンテキストでは、データのロードは、外部表からデータを読み取り、データベース内の表にロードする操作を指します。データのアンロードは、データベース内の表からデータを読み取り、外部表に挿入する操作を指します。

外部表のデフォルトの型は `ORACLE_LOADER` です。これを使用して、外部表から表データを読み取り、データベースにロードできます。また、Oracle Database には、`ORACLE_DATAPUMP` 型も用意されています。この型を使用すると、データをアンロード（データベース内の表からデータを読み取って外部表に挿入）し、Oracle データベースに再ロードできます。

外部表の定義は、データ・ソース内のデータの記述とは個別に保持されます。これは次のことを意味します。

- ソース・ファイルのフィールドは、外部表内の列より多くすることも、少なくすることもできます。
- データ・ソース内のフィールドのデータ型は、外部表内の列と異なるものにすることができます。

外部表を使用したデータのロード

外部表の主な用途は、データをデータベース内の実在する表にロードする際の行ソースとすることです。外部表を作成すると、それを `SELECT` 句のソースとして、`CREATE TABLE AS SELECT` または `INSERT INTO ... AS SELECT` 文を使用できます。

> **注意：** 外部表にデータを挿入したり、外部表のレコードを更新することはできません。外部表は読取り専用です。

SQL 文を通して外部表にアクセスするときは、外部表のフィールドは一般の表のフィールドとまったく同じように使用できます。特に、SQL のビルトインのファンクション、PL/SQL ファンクション、または Java 関数の引数として使用できます。これにより、外部ソースからのデータを操作できます。データ・ウェアハウスではこの方法により、単純なデータ型変換よりもさらに洗練された変換が可能です。また、データ・ウェアハウスでこのメカニズムを使用してデータのクレンジングを行うこともできます。

外部表に列オブジェクトを格納することはできませんが、コンストラクタ関数を使用すると外部表内の属性から列オブジェクトを作成できます。

外部表へのパラレル・アクセス

外部表のメタデータを作成すると、SQL を使用して外部データに直接またはパラレルで問合せができます。この結果、外部表がビューとして動作するため、外部データに対する SQL 問合せが外部データをデータベースにロードすることなく実行できるようになります。

外部表へのパラレル・アクセスの程度は、標準パラレル・ヒントを使用して PARALLEL 句で指定します。外部表でパラレル化を使用すると、外部表を導出するデータファイルに同時アクセスができます。単一のファイルに同時アクセスができるかどうかは、アクセス・ドライバの実装およびアクセスされるデータファイルの属性（レコードのフォーマットなど）に依存します。

関連項目：

- 外部表、外部接続およびディレクトリの管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 外部表からのロードのチューニングの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 外部表とインポートおよびエクスポートの詳細は、『Oracle Database ユーティリティ』を参照してください。
- 外部表の作成と問合せの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

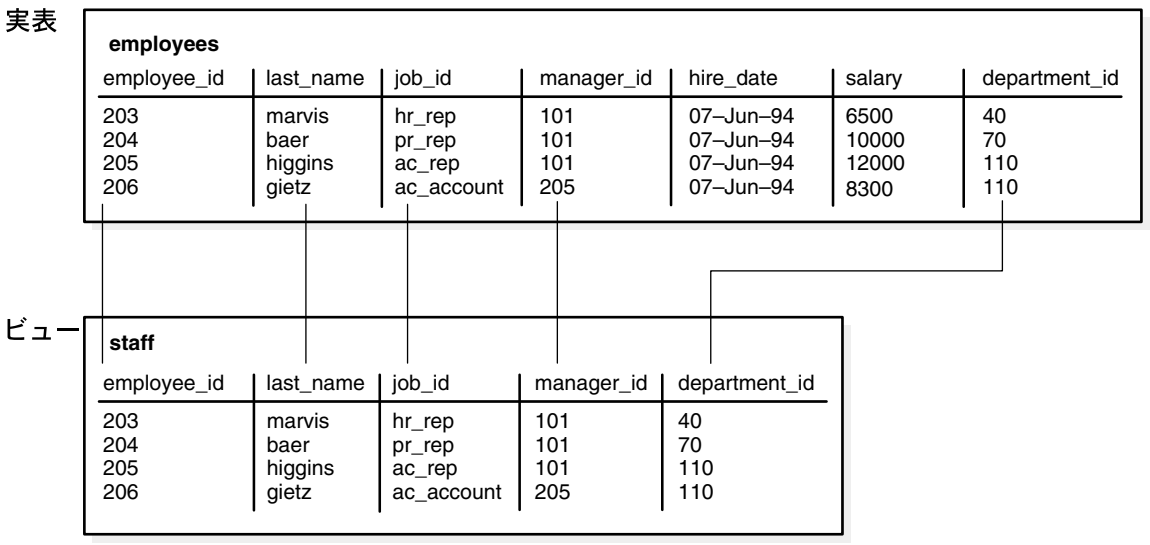
ビューの概要

ビューとは、1 つ以上の表または他のビューに含まれているデータを調整して表現したものです。ビューは問合せの出力を受け取り、それを表として扱います。そのためビューは、ストアド・クエリーまたは仮想表とみなすこともできます。多くの場合、ビューは表と同じように使用できます。

たとえば、employees 表には、いくつかの列と多数の情報行があります。ユーザーにはそのうちの 5 つの列または特定の行しか参照させないようにする場合は、他のユーザーがアクセスできるようにビューを作成できます。

図 5-5 に、実表 employees から導出された staff というビューの例を示します。ビューが、実表の 5 つの列のみを表示していることに注目してください。

図 5-5 ビューの例



ビューは表から導出されるため、ビューと表には多数の類似点があります。たとえば、ビューには表と同じように最大 1000 個の列を定義できます。ビューは問合せが可能なだけでなく、いくつかの制限付きでビューのデータを更新、挿入および削除できます。実際にビューに対して実行されるすべての操作は、そのビューの実表のデータに影響し、実表の整合性制約とトリガーの対象になります。

ビューではトリガーを明示的に定義できませんが、ビューから参照される実表に対しては定義できます。Oracle Database では、ビューの論理的制約の定義はサポートされません。

関連項目：『Oracle Database SQL 言語リファレンス』

この項の内容は、次のとおりです。

- ビューの格納方法
- ビューの使用方法
- ビューのメカニズム
- 依存性とビュー
- 更新可能な結合ビュー
- オブジェクト・ビュー
- インライン・ビュー

ビューの格納方法

表とは異なり、ビューには記憶域は割り当てられず、実際にデータが格納されることもありません。ビューは、参照する表からデータを抽出または導出する問合せによって定義されます。これらの表は**実表**と呼ばれます。実表としては、実際の表またはビュー（マテリアライズド・ビューを含む）を使用できます。ビューは他のオブジェクトを基盤としているため、データ・ディクショナリ内のビューの定義（ストアド・クエリー）以外には記憶域が必要ありません。

ビューの使用方法

ビューは、実表に含まれるデータをいろいろな表現形式で表現する手段を提供します。様々なユーザーにあわせてデータの表現形式を変化させることができるため、ビューは非常に強力な機能です。通常、ビューは次の目的で使用されます。

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

  たとえば、図 5-5 は、`staff` ビューに実表 `employees` の列 `salary` または `commission_pct` が含まれない仕組みを示しています。

- データの複雑さを隠します。

  たとえば、複数の表の関連した列または行を 1 つにまとめる**結合処理**によって、単一のビューを定義できます。ただし、ビューからは、この情報が複数の表から抽出されたものであるということはわかりません。

- ユーザーの文を単純化します。

  たとえば、ユーザーが結合の方法を知らなくても、複数の表から情報を選択できるようにします。

- 実表とは異なる視点からデータを提示します。

  たとえば、ビューが基礎にしている表に影響を与えずに、ビューの列名を変更できます。

- 実表の定義の変更からアプリケーションを分離します。

  たとえば、ビューを定義している問合せが表の 4 つの列の中の 3 つの列を参照している場合、たとえ 5 番目の列が追加されてもビューの定義は影響を受けないため、ビューを使用しているすべてのアプリケーションも影響を受けません。

- ビューなしでは表現できなかった問合せを表現します。

  たとえば、GROUP BY ビューと表を結合してビューを定義できます。また、UNION ビューと表を結合してビューを定義することもできます。

- 複雑な問合せを保存します。

  たとえば、ある問合せで表情報に対して複雑な計算を実行したとします。この問合せをビューとして保存しておくと、そのビューに問合せを行うたびに同じ計算が実行されます。

> **関連項目：** GROUP BY ビューまたは UNION ビューの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

ビューのメカニズム

Oracle Database では、ビューの定義は、ビューを定義する問合せテキストとしてデータ・ディクショナリに保存されます。SQL 文でビューを参照すると、Oracle Database では次のように処理されます。

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

Oracle Database では、新しい共有 SQL 領域内のビューを参照する文は、既存の共有 SQL 領域に同様の文が含まれていない場合にのみ解析されます。したがって、ビューを使用すると、共有 SQL に対応付けられているメモリーの使用量を減らせるという利点があります。

この項の内容は、次のとおりです。

- ビューのグローバリゼーション・サポート・パラメータ
- ビューに対する索引の使用

ビューのグローバリゼーション・サポート・パラメータ

文字列リテラルまたはグローバリゼーション・サポート・パラメータを引数として持つ SQL 関数（TO_CHAR、TO_DATE および TO_NUMBER など）が組み込まれているビューを評価する際、Oracle Database では、セッションのグローバリゼーション・サポート・パラメータから、これらのパラメータのデフォルト値を取得します。これらのデフォルト値は、ビュー定義の中でグローバリゼーション・サポート・パラメータを明示的に指定することでオーバーライドできます。

> **関連項目：** グローバリゼーション・サポートの詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

ビューに対する索引の使用

Oracle Database では、元の問合せをビュー定義の問合せとマージする際に、元の問合せを変換することによって、ビューに対する問合せで索引を使用するかどうかが決まります。

次のビューを考えてみます。

```
CREATE VIEW employees_view AS
  SELECT employee_id, last_name, salary, location_id
    FROM employees JOIN departments USING (department_id)
    WHERE departments.department_id = 10;
```

ここで、ユーザーは次の問合せを発行します。

```
SELECT last_name
  FROM employees_view
  WHERE employee_id = 9876;
```

Oracle Database では、次のような問合せが最終的に作成されます。

```
SELECT last_name
  FROM employees, departments
  WHERE employees.department_id = departments.department_id AND
        departments.department_id = 10 AND
        employees.employee_id = 9876;
```

可能な場合、Oracle Database は、ビューに対する問合せを、ビュー定義の問合せおよび基礎となるビューの問合せとマージします。Oracle Database により、マージされた問合せは、ビューを参照せずに問合せを発行したかのように最適化されます。そのため、Oracle Database では、列がビュー定義で参照されても、またはビューに対するユーザーの問合せで参照されても、参照される実表の列に対する索引を使用できます。

Oracle Database では、ユーザーの発行した問合せとビュー定義をマージできないこともあります。その場合、Oracle Database は参照された列の索引すべてを使用できるとはかぎりません。

関連項目： 問合せの最適化の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

依存性とビュー

ビューは、他のオブジェクト（表、マテリアライズド・ビューまたは他のビュー）を参照する問合せによって定義されるため、参照されるオブジェクトに依存します。Oracle Database では、ビューの依存性は、自動的に処理されます。たとえば、いったん削除されたビューの実表が再作成された場合、Oracle Database は、新しい実表がそのビューの既存の定義と一致するかどうかを確認します。

関連項目： 第 6 章「スキーマ・オブジェクトの依存性」

更新可能な結合ビュー

結合ビューは、FROM 句（**結合**）に複数の表またはビューを持ち、かつ DISTINCT、AGGREGATION、GROUP BY、START WITH、CONNECT BY および ROWNUM のいずれの句も使用せず、また集合演算（UNION ALL、INTERSECT など）も使用しないビューとして定義されます。

更新可能な結合ビューとは、2 つ以上の実表またはビューを含み、UPDATE、INSERT および DELETE の各操作が実行可能である結合ビューです。ビューのどの列が更新可能であるかを示す情報は、データ・ディクショナリ・ビュー ALL_UPDATABLE_COLUMNS、DBA_UPDATABLE_COLUMNS および USER_UPDATABLE_COLUMNS に含まれています。本質的に更新可能であるためには、ビューは次の構成メンバーを含むことはできません。

- 集合演算子
- DISTINCT 演算子
- 集計関数または分析関数
- GROUP BY 句、ORDER BY 句、CONNECT BY 句または START WITH 句
- SELECT 構文のリストのコレクション式
- SELECT 構文のリストの副問合せ
- 結合（例外あり）

更新可能でないビューは、INSTEAD OF トリガーを使用して変更できます。

関連項目：

- 『Oracle Database 管理者ガイド』
- 更新可能ビューの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 22-9 ページの「INSTEAD OF トリガー」

オブジェクト・ビュー

Oracle オブジェクト・リレーショナル・データベースの場合、**オブジェクト・ビュー**を使用すると、リレーショナル・データを、それがオブジェクト型として格納されているかのように検索、更新、挿入および削除できます。また、オブジェクト、REF およびコレクション（ネストした表と VARRAY）などのオブジェクト・データ型である列を持つビューを定義できます。

関連項目：

- 『Oracle Database オブジェクト・リレーショナル開発者ガイド』
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

インライン・ビュー

インライン・ビューは、スキーマ・オブジェクトではありません。SQL 文内でビューのように使用できる、別名（相関名）を持つ副問合せです。

関連項目：

- 副問合せの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- ビューを使用するインライン問合せの例は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

マテリアライズド・ビューの概要

マテリアライズド・ビューは、データの集計、計算、レプリケートおよび分配に使用できるスキーマ・オブジェクトです。この種のビューは、データ・ウェアハウス、意思決定支援および分散コンピューティングやモバイル・コンピューティングなど、様々なコンピュータ環境に適しています。

- データ・ウェアハウスでは、マテリアライズド・ビューは、合計値や平均値など、集計されたデータを計算して格納するために使用されます。通常、このような環境では、マテリアライズド・ビューには集計データが格納されるため、**サマリー**と呼ばれます。また、マテリアライズド・ビューは、集計操作の有無に関係なく、結合の計算にも使用できます。互換性が Oracle9*i* 以上に設定されていると、フィルタ選択を含む問合せにマテリアライズド・ビューを使用できます。

  オプティマイザでは、可能かつ必要な場合を自動的に認識して、マテリアライズド・ビューを使用して問合せのパフォーマンスを改善できます。要求は、マテリアライズド・ビューを使用するように、オプティマイザによって自動的に透過的にリライトされます。その後、問合せは、基礎となるディテール表やビューではなく、マテリアライズド・ビューに送られます。

- 分散環境では、マテリアライズド・ビューは、分散サイトでデータをレプリケートし、複数のサイトで実行される更新を競合解消方法に従って同期化するために使用されます。レプリカとしてのマテリアライズド・ビューは、他の方法ではリモート・サイトからのアクセスを必要とするデータへのローカル・アクセスを提供します。

- モバイル・コンピューティング環境では、マテリアライズド・ビューは中央のサーバーからモバイル・クライアントにデータのサブセットをダウンロードし、中央のサーバーから定期的にリフレッシュし、クライアントで実行された更新を中央のサーバーに伝播させるために使用されます。

マテリアライズド・ビューは、いくつかの点で索引に似ています。

- 記憶域を使用します。
- マスター表のデータに変更があった場合は、リフレッシュする必要があります。
- クエリー・リライトに使用すると、SQL の実行効率が改善されます。
- その存在は、SQL アプリケーションとユーザーに対して透過的です。

索引とは異なり、マテリアライズド・ビューには、SELECT 文を使用して直接アクセスできます。必要なリフレッシュのタイプによっては、INSERT、UPDATE または DELETE 文で直接アクセスすることも可能です。

マテリアライズド・ビューはパーティション化できます。マテリアライズド・ビューは、そのパーティション表および 1 つ以上の索引上で定義できます。

この項の内容は、次のとおりです。

- ビューの制約定義
- マテリアライズド・ビューのリフレッシュ
- マテリアライズド・ビュー・ログ

関連項目：

- 5-22 ページの「索引の概要」
- 『Oracle Database VLDB およびパーティショニング・ガイド』
- データ・ウェアハウス環境におけるマテリアライズド・ビューの詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

ビューの制約定義

データ・ウェアハウス・アプリケーションは、Oracle データベースにあるマルチディメンションのデータを、リレーショナル・スキーマにおける参照整合性（RI）制約を特定することによって認識します。RI 制約は、各表の間での主キーや外部キーの関係を表します。Oracle Database データ・ディクショナリに問い合せることにより、アプリケーションは RI 制約を認識し、それによってデータベース内のマルチディメンションのデータを認識できるようになります。一部の環境では、スキーマの複雑性またはセキュリティ上の理由から、データベース管理者はビューをファクト表およびディメンション表で定義します。Oracle Database では、ビューの制約のための機能が提供されています。各ビュー間の制約定義を許可すると、データベース管理者は実表制約をビューに伝播できるようになり、それによってアプリケーションが制限付きの環境でもマルチディメンションのデータを認識できるようになります。

ビューで定義できるのは、論理的な制約、つまり宣言制約であって Oracle Database によって規定されていない制約のみです。これらの制約の目的は、ビジネス・ルールの規定ではなく、マルチディメンション・データを識別することです。ビューでは次の制約を定義できます。

- 主キー制約
- 一意制約
- 参照整合性制約

ビューの制約が宣言制約の場合、有効な状態は DISABLE、NOVALIDATE のみです。ただし、ビューの制約はより洗練されたクエリー・リライトのために使用される場合があるため、RELY または NORELY の状態も認められます。RELY 状態にあるビューの制約は、リライトの整合性レベルがトラステッド・モードに設定されている場合に、クエリー・リライトを許可します。

注意： ビューの制約定義が本質的には宣言制約であっても、ビュー上の操作は基礎となる実表で定義された整合性制約の対象であり、ビューの制約は実表における制約を通して施行できます。

マテリアライズド・ビューのリフレッシュ

Oracle Database では、マスター表の変更後にマテリアライズド・ビューをリフレッシュして、マテリアライズド・ビューのデータを管理します。リフレッシュ方法には、増分リフレッシュ（**高速リフレッシュ**）または完全リフレッシュがあります。高速リフレッシュ方法を使用する場合、マスター表に対する変更は**マテリアライズド・ビュー・ログ**または**ダイレクト・ローダー・ログ**に記録されます。

マテリアライズド・ビューは、必要時に、または定期的にリフレッシュできます。また、マスター表と同じデータベース内のマテリアライズド・ビューは、トランザクションによってマスター表の変更がコミットされるたびにリフレッシュできます。

マテリアライズド・ビュー・ログ

マテリアライズド・ビュー・ログとは、マスター表に定義されているマテリアライズド・ビューの増分リフレッシュを実行できるように、マスター表の変更を記録するスキーマ・オブジェクトです。

マテリアライズド・ビュー・ログは、それぞれ 1 つのマスター表に対応付けられています。マテリアライズド・ビュー・ログは、マスター表と同じデータベースおよびスキーマに格納されます。

関連項目：

- ウェアハウス環境におけるマテリアライズド・ビューとマテリアライズド・ビュー・ログの詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。
- レプリケーションに使用されるマテリアライズド・ビューの詳細は、『Oracle Database アドバンスト・レプリケーション』を参照してください。

ディメンションの概要

ディメンションは、1 対の列または列セット間の階層（親子）関係を定義します。子レベルの値は、それぞれが親レベルの 1 つの値にのみ対応付けられます。階層関係は、ある階層レベルから次のレベルへの**機能上の依存関係**です。ディメンションは、列相互の論理関係のコンテナで、それにはデータ記憶域は割り当てられません。

CREATE DIMENSION 文では、次の事項を指定します。

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

ディメンション内の列は、同じ表のもの（**非正規化**）でも複数の表のもの（**完全正規化**または**一部正規化**）でもかまいません。複数の表からの列によるディメンションを定義するには、HIERARCHY 句の JOIN 句を使用して表を接続します。

たとえば、正規化されている time ディメンションには、date 表、month 表および year 表と、各 date 行を month 行に、各 month 行を year 行に接続する結合条件を含めることができます。完全な非正規化の time ディメンションでは、date、month および year 列はすべて同じ表に格納されます。正規化されているかどうかに関係なく、列相互の階層関係を CREATE DIMENSION 文で指定する必要があります。

関連項目：

- ウェアハウス環境におけるディメンションの使用方法は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。
- ディメンションの作成の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

シーケンス・ジェネレータの概要

シーケンス・ジェネレータは、連続的な数値を生成します。また、ディスク I/O やトランザクション・ロックなどのオーバーヘッドを発生させずに、一意の連続的な数を生成するため、マルチユーザー環境では特に有効です。たとえば、2 人のユーザーが employees 表に新しい従業員の行を同時に挿入しているとします。順序を使用して employee_id 列に一意の従業員番号を生成すると、一方のユーザーは、他方のユーザーが従業員番号を入力するのを待機する必要はありません。どちらのユーザーにも、順序により、正しい値が自動的に生成されるためです。

したがって、このシーケンス・ジェネレータによってシリアライズ（2 つのトランザクションの文が連続番号を同時に生成する必要がある状態）が低減されます。このシリアライズを避けることによって、トランザクションのスループットが改善され、ユーザーの待ち時間が大幅に短縮されます。

順序番号とは、データベース内で定義される最大 38 桁の整数です。順序の定義には、次のように一般的な情報を指定します。

- 順序の名前
- 昇順または降順
- 数値の増分値
- Oracle Database により生成される順序番号の集合をメモリー内にキャッシュするかどうか

Oracle Database では、特定のデータベースのすべての順序定義が、SYSTEM 表領域内の単一のデータ・ディクショナリ表に行として保存されます。したがって、SYSTEM 表領域は常時オンラインであるため、すべての順序定義がいつでも使用できます。

順序番号は、順序を参照する SQL 文によって使用されます。文を発行して、新しい順序番号を生成することも、現行の順序番号を使用することもできます。ユーザーのセッションにおける文が順序番号を生成すると、その特定の順序番号はそのセッションのみに使用可能となります。順序を参照する各ユーザーは、現行の順序番号にアクセスできます。

順序番号は表からは独立して生成されます。そのため、同じシーケンス・ジェネレータを複数の表に対して使用できます。順序番号の生成は、データに対して一意の主キーを自動的に生成する場合や、複数の行または表にまたがるキーを統合する場合に有効です。最終的にロールバックされたトランザクションで生成されて使用された個々の順序番号はスキップされます。必要に応じて、アプリケーションは、これらの順序番号を受け取って再利用するように作成することもできます。

注意：順序番号をなくすことができないアプリケーションの場合、Oracle Database の順序は使用できません。かわりに、データベースの表に順序番号を保存する方法があります。データベースの表を使用してシーケンス・ジェネレータを実装する際は、注意が必要です。シングル・インスタンスの構成においても、高頻度で順序値を生成する際は、順序値を保存する行のロックにかかるコストに対応付けられたパフォーマンスのオーバーヘッドが発生します。

関連項目：

- 順序の使用がパフォーマンスに与える影響については、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- CREATE SEQUENCE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

シノニムの概要

シノニムとは、表、ビュー、マテリアライズド・ビュー、順序、プロシージャ、ファンクション、パッケージ、型、Java クラスのスキーマ・オブジェクト、ユーザー定義オブジェクト型または他のシノニムの別名です。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶域は必要ありません。

シノニムは、セキュリティと便利さのために使用されます。たとえば、次のような利点があります。

- オブジェクトの名前と所有者を隠します。
- 分散データベースのリモート・オブジェクトの位置の透過性を実現します。
- データベース・ユーザー側の SQL 文を単純化します。
- ファイングレイン・アクセス・コントロールを行う際に、特化したビューに似た制限付きのアクセスを使用可能にします。

パブリック・シノニムとプライベート・シノニムの両方を作成できます。**パブリック**・シノニムは、PUBLIC という名前の特別なユーザー・グループが所有しており、データベースのすべてのユーザーがアクセスできます。**プライベート**・シノニムは、特定のユーザーのスキーマに含まれており、そのユーザーが他のユーザーに対するシノニムの使用許可を制御します。

シノニムは、分散システム内の位置を含めて、基礎になるオブジェクトの識別を隠すため、分散データベース環境でもそれ以外の環境でもきわめて役立ちます。基礎となるオブジェクトを改名または移動する必要がある場合、シノニムのみを再定義すればよいので便利です。シノニムに基づくアプリケーションは、今後も変更なしに機能し続けることができます。

また、分散データベース・システム内のユーザーのために、シノニムによって SQL 文を単純化できます。実表の識別を隠したり、SQL 文の複雑さを低減するために、データベース管理者はパブリック・シノニムを作成する場合があります。その理由と作成方法を次の例で説明します。この例では、次のような状況を想定します。

- ユーザー JWARD が所有するスキーマ内に SALES_DATA という表があります。
- SALES_DATA 表に対する SELECT 権限が PUBLIC に付与されています。

この時点で、データベースのユーザーが、次のような SQL 文を使用して SALES_DATA 表を問い合せる必要があります。

```
SELECT * FROM jward.sales_data;
```

問合せを実行するには、表を含むスキーマと表の名前の両方を含める必要があることに注意してください。

データベース管理者が次の SQL 文によって、パブリック・シノニムを作成するとします。

```
CREATE PUBLIC SYNONYM sales FOR jward.sales_data;
```

パブリック・シノニムが作成されると、ユーザーは、次のような単純な SQL 文で SALES_DATA 表を問い合せることができます。

```
SELECT * FROM sales;
```

パブリック・シノニム SALES は、表 SALES_DATA の名前とその表を含むスキーマ名を隠していることに注意してください。

索引の概要

索引は、表とクラスタに対応付けるオプションの構造体です。表の 1 つ以上の列に索引を作成し、その表に対する SQL 文の実行を高速化できます。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。同様に、Oracle Database の索引を使用すると表データに高速にアクセスできます。適切に使用すると、索引はディスク I/O を低減する基本的な手段になります。

表には、列の組合せが索引ごとに異なるかぎり、いくつもの索引を作成できます。列の組合せを個別に指定すると、同じ列を使用して複数の索引を作成できます。たとえば、次の文では有効な組合せを指定しています。

```
CREATE INDEX employees_idx1 ON employees (last_name, job_id);
CREATE INDEX employees_idx2 ON employees (job_id, last_name);
```

Oracle Database は、パフォーマンスの機能性を補足する複数の索引体系を提供しています。

- B ツリー索引
- B ツリークラスタ索引
- ハッシュ・クラスタ索引
- 逆キー索引
- ビットマップ索引
- ビットマップ結合索引

また、Oracle Database ではアプリケーションまたはカートリッジ固有のファンクション索引やドメイン索引もサポートしています。

索引の有無によって、SQL 文の表現を変更する必要はありません。索引は、単なるデータへの高速なアクセス・パスです。実行のスピードにのみ影響を与えます。索引の付いたデータ値では、索引はその値を含んでいる行の位置を直接示すポインタとして機能します。

索引は、関連する表のデータから論理的にも物理的にも独立しています。索引は、実表や他の索引に影響を及ぼさずにいつでも作成または削除できます。索引を削除しても、すべてのアプリケーションは機能を続けます。ただし、索引設定されていたデータへのアクセスは低速になる場合があります。索引は独立した構造体であるため、記憶域を必要とします。

作成された索引は、Oracle Database によって自動的にメンテナンスされ、使用されます。Oracle Database では、行の新規追加、更新または削除など、データに対する変更が、関連するすべての索引に自動的に反映され、ユーザーによる操作は不要です。

新たにいくつかの行が挿入されても、索引付きのデータ検索のパフォーマンスはほとんど一定です。ただし、表に対して数多くの索引が存在すると、更新、削除および挿入のパフォーマンスは低下します。これは、Oracle Database では表に関連する索引も更新する必要があるためです。

オプティマイザは、既存の索引を使用して別の索引を作成できます。この結果、索引作成がはるかに高速になります。

この項の内容は、次のとおりです。

- 索引レンジ・スキャン
- キー圧縮
- 逆キー索引
- ビットマップ索引
- ビットマップ結合索引

一意索引と非一意索引

索引には、一意索引と非一意索引の 2 種類があります。一意索引によって、キー列（1 つまたは複数）に重複値を持つ行が複数存在しないことが保証されます。非一意索引の場合は、列値にこのような制限はありません。

一意索引は、`CREATE UNIQUE INDEX` を使用して、明示的に作成することをお薦めします。主キーまたは一意制約による一意索引の作成では、新規索引の作成は保証されず、作成される索引が一意索引となる保証もありません。

> **関連項目：** 一意索引を明示的に作成する方法は、『Oracle Database 管理者ガイド』を参照してください。

参照用索引と非参照用索引

索引には、参照用と非参照用の 2 種類があります。非参照用索引は DML 操作により維持され、オプティマイザでは使用できません。

索引を使用不可にしたり削除する代替手段として、その索引を非参照用にすることもできます。

> **関連項目：**
>
> - 非参照用索引を作成する方法は、『Oracle Database 管理者ガイド』を参照してください。
> - 索引を非参照用にする方法は、『Oracle Database 管理者ガイド』を参照してください。

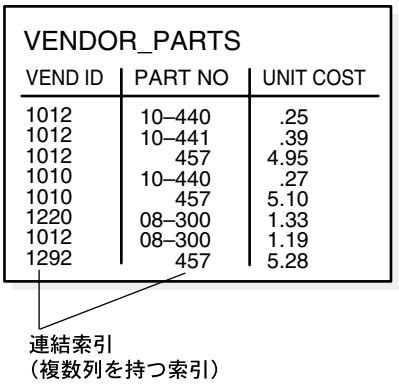
コンポジット索引

コンポジット索引（**連結索引**とも呼ばれる）は、表の中の複数の列に対して作成される索引です。コンポジット索引を構成する複数の列は、どんな順序で指定してもかまいません。また、コンポジット索引の列は表の中で隣接している必要もありません。

`SELECT` 文の `WHERE` 句でコンポジット索引のすべての列または列の先頭部分を参照する場合には、コンポジット索引によってデータの検索速度を向上させることができます。そのため、定義で使用する列の順序は重要です。通常、最も頻繁にアクセスされる列または最も選択的な列が最初になります。

図 5-6 は、VENDOR_ID および PART_NO 列にコンポジット索引を設定した VENDOR_PARTS 表を示しています。

図 5-6 コンポジット索引の例



通常のコンポジット索引は、最大 32 列で形成されます。ビットマップ索引の場合、列の最大数は 30 です。キー値は、データ・ブロック内の使用可能データ領域のおよそ 2 分の 1（オーバーヘッド分を差し引いたもの）を超えることはできません。

> **関連項目：** コンポジット索引の使用方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

索引とキー

索引とキーという用語は同じ意味で使用されることがありますが、これらの用語には違いがあります。**索引**は、データベース内に実際に格納される構造体であり、ユーザーは SQL 文を使用して作成、変更および削除します。作成された索引により、表データへの高速なアクセス・パスが提供されます。一方、**キー**は厳密に論理的な概念です。キーは、整合性制約と呼ばれる、データベースのビジネス・ルールを規定する Oracle Database のもう 1 つの機能に対応しています。

Oracle Database では索引を使用して一部の整合性制約を規定するため、キーと索引という 2 つの用語はよく同じ意味で使用されます。ただし、これらの用語を混同しないでください。

> **関連項目：** 第 21 章「データ整合性」

索引と NULL

索引内に複数の NULL 値を持つ行が存在する場合、それぞれの行は別個のものとみなされます。ただし、索引内に NULL でない同一の値を持つ行が 2 行以上存在する場合は、それらの行は同一とみなされます。したがって、一意索引では、行に NULL 値が含まれていると、同一の行として処理されません。この規則は、NULL 以外の値が存在しない場合、つまり、行全体が NULL の場合には適用されません。

Oracle Database では、すべてのキー列が NULL の表の行には、索引は作成されません。ただし、ビットマップ索引の場合や、クラスタ・キーの列値が NULL の場合は例外です。

> **関連項目：** 5-34 ページの「ビットマップ索引と NULL」

ファンクション索引

表に索引を設定する場合、その表の 1 つ以上の列を含むファンクションと式について、索引を作成できます。**ファンクション索引**では、ファンクションや式の値が計算され、索引に格納されます。ファンクション索引は、B ツリーまたはビットマップ索引として作成できます。

索引作成に使用するファンクションには、算術式あるいは PL/SQL ファンクション、パッケージ・ファンクション、C コールアウトまたは SQL 関数を含む式を使用できます。式に集計関数を含むことはできず、DETERMINISTIC にする必要があります。オブジェクト型を含む列の索引を作成する場合は、マップ・メソッドなど、そのオブジェクトのメソッドをファンクションとして使用できます。ただし、LOB 列、REF またはネストした表の列には、ファンクション索引を作成できません。また、オブジェクト型に LOB、REF またはネストした表が含まれる場合も作成できません。

この項の内容は、次のとおりです。

- ファンクション索引の使用方法
- ファンクション索引による最適化
- ファンクション索引の依存性

関連項目：

- 「ビットマップ索引」
- ファンクション索引の使用方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ファンクション索引の使用方法

ファンクション索引は、WHERE 句にファンクションを含む文を効率的に評価するメカニズムを提供します。式の値は計算されて、索引に格納されます。ただし、INSERT 文と UPDATE 文の処理中には、Oracle Database では文を処理するために従来どおりファンクションを評価する必要があります。

たとえば、次の索引を作成する場合を考えます。

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
```

Oracle Database では、次のような問合せを処理するときにこの索引を使用できます。

```
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

UPPER(*column_name*) または LOWER(*column_name*) でファンクション索引を定義すると、大文字 / 小文字区別の検索が容易になります。たとえば、次の索引があります。

```
CREATE INDEX uppercase_idx ON employees (UPPER(first_name));
```

この索引により、次のような問合せの処理が容易になります。

```
SELECT * FROM employees WHERE UPPER(first_name) = 'RICHARD';
```

また、ファンクション索引は、SQL 文に効率的な言語照合機能を提供するグローバリゼーション・サポート・ソート索引にも使用できます。

関連項目： 言語索引の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

ファンクション索引による最適化

オプティマイザを使用するには、ファンクション索引に関する統計を収集する必要があります。この統計がないと、SQL 文の処理に索引を使用できません。

オプティマイザでは、WHERE 句の式による問合せに、ファンクション索引に対する索引レンジ・スキャンを使用できます。たとえば、次の問合せで考えてみます。

```
SELECT * FROM t WHERE a + b < 10;
```

a+b の索引が作成されていれば、オプティマイザは索引レンジ・スキャンを使用できます。レンジ・スキャンのアクセス・パスは、述語（WHERE 句）の選択性が低い場合に特に便利です。また、式がファンクション索引で具象化されていると、より正確な式を必要とする述語の選択性をオプティマイザで見積ることができます。

オプティマイザは、SQL 文の式を解析し、文の式ツリーとファンクション索引を比較して、式のマッチングを実行します。この比較は大文字 / 小文字が区別され、ブランクは無視されます。

> **関連項目：** 統計収集の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ファンクション索引の依存性

ファンクション索引は、その索引を定義している式に使用されたファンクションに依存しています。ファンクションが PL/SQL ファンクションまたはパッケージ・ファンクションの場合は、ファンクションの仕様に変更があると索引は使用禁止になります。

ファンクション索引を作成するには、CREATE INDEX または CREATE ANY INDEX 権限が付与されている必要があります。

ファンクション索引を使用する手順は次のとおりです。

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

この後の各項では、追加の要件について説明します。

DETERMINISTIC ファンクション ファンクション索引に使用するユーザー記述の関数は、現在も将来も入力引数値の集合に対して常に同じ出力戻り値を戻すことを示すために、DETERMINISTIC キーワードで宣言する必要があります。

> **関連項目：**『Oracle Database パフォーマンス・チューニング・ガイド』

定義する関数に対する権限 索引所有者は、ファンクション索引の定義に使用する関数に対して EXECUTE 権限を持つ必要があります。Oracle Database では、EXECUTE 権限が取り消されると、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、基礎となる表に SELECT 権限を付与できます。

ファンクション索引の依存性の解決 ファンクション索引は、それを使用するファンクションに依存します。ファンクション、またはそのファンクションを含むパッケージの仕様部が再定義されると（または、索引所有者の EXECUTE 権限が取り消されると）、次の条件が成立します。

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

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

索引の格納方法

Oracle Database では、索引を作成する際に、索引のデータを保持するための索引セグメントが表領域内に自動的に割り当てられます。索引セグメントへの領域の割当てと、確保された領域の使用は、次の方法で制御できます。

- 索引セグメントのエクステントの割当ては、その索引セグメントに対して記憶域パラメータを設定すると制御できます。
- 索引セグメントのエクステントを構成するデータ・ブロック内の空き領域は、その索引セグメントの PCTFREE パラメータを設定すると制御できます。

索引セグメントの表領域は、所有者のデフォルト表領域または CREATE INDEX 文で指定された表領域です。索引を、その索引に対応する表と同じ表領域に格納する必要はありません。また、Oracle Database では、索引と表データを並行して取得できるため、索引とその表をそれぞれ別のディスク・ドライブ上の異なる表領域内に格納して、索引を使用する問合せのパフォーマンスを向上させることができます。

関連項目： 2-7 ページの「PCTFREE、PCTUSED と行連鎖」

この項の内容は、次のとおりです。

索引ブロックの形式

索引データのために使用できる領域は、Oracle Database ブロック・サイズから、ブロック・オーバーヘッド、エントリ・オーバーヘッド、ROWID および索引が作成される値 1 つ当たり 1 バイトの合計を引いたものです。

索引を作成する際に、Oracle Database は、索引を付ける列をフェッチしてソートし、ROWID を各行の索引値とともに保存します。次に、Oracle Database は、索引の一番下から順にロードします。たとえば、次の文を考えてみます。

```
CREATE INDEX employees_last_name ON employees(last_name);
```

Oracle Database は、last_name 列に基づいて employees 表をソートします。次に、このソート順に、last_name とそれに対応する ROWID という形で索引をロードします。索引を使用するときに、Oracle Database は、ソートされた last_name 値を迅速に検索し、その値に対応する ROWID 値を使用して、求める last_name 値を持つ行を見つけます。

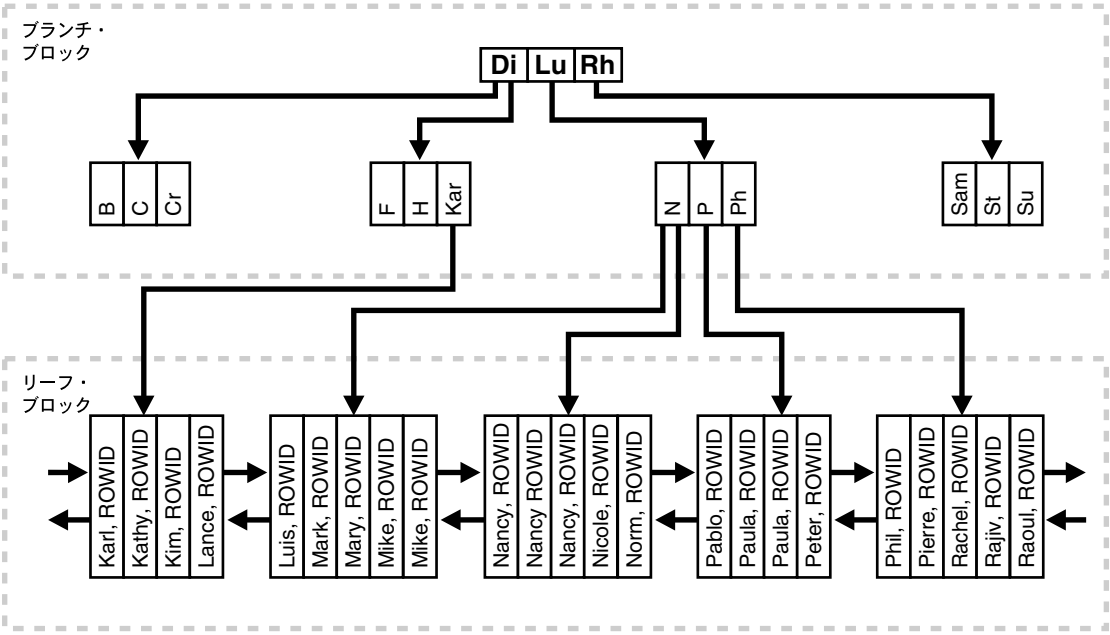
索引の内部構造

Oracle Database は、索引の格納に B ツリーを使用して、データ・アクセスを高速化します。索引がない場合は、データの順次スキャンを行って値を検索する必要があります。行数が *n* の場合、検索される行数の平均は *n*/2 になります。したがって、データの量が多くなるほど、正確な結果を得られません。

ブロック幅範囲（リーフ・ブロック）に分割されている、順序付けされた値リストを考えてみます。範囲のエンド・ポイントはブロックへのポインタとともに検索ツリーに格納できるため、*n* エントリのログ（*n*）タイムで値を検索できます。これが Oracle Database 索引の基本原理です。

図 5-7 に、B ツリー索引の構造を示します。

図 5-7 B ツリー索引の内部構造



B ツリー索引の上位ブロック（**ブランチ・ブロック**）には、下位レベルの索引ブロックを指す索引データが含まれます。最下位レベルの索引ブロック（**リーフ・ブロック**）には、すべての索引対象のデータ値と、実際の行を検出するための ROWID が含まれます。リーフ・ブロックは二重にリンクされます。文字データを持つ列の索引は、データベース・キャラクタ・セットにおける文字のバイナリ値を基盤にしています。

一意索引の場合、データ値ごとに ROWID が 1 つ存在します。非一意索引の場合、ROWID はソートするためのキーに含まれるため、非一意索引は索引キーと ROWID によってソートされます。クラスタ索引を除いて、NULL のみを含んでいるキーに索引は定義できません。2 つの行に NULL のみを含めても、一意索引には違反しません。

索引のプロパティ

ブロックには 2 つの種類があります。

- 検索用のブランチ・ブロック
- 値を格納するリーフ・ブロック

ブランチ・ブロック ブランチ・ブロックには次のものが格納されます。

- 2 つのキーの分岐を決定する際に必要な、最小のキー接頭辞
- キーを含む子ブロックへのポインタ

ブロックに *n* 個のキーがある場合、*n*+1 個のポインタがあります。キーおよびポインタの数は、ブロックのサイズによって制限されます。

リーフ・ブロック リーフ・ブロックはすべて、ルート・ブランチ・ブロックからの深さが同じになります。リーフ・ブロックには次のものが格納されます。

- すべての行の完全なキー値
- 表の行の ROWID

キーと ROWID のペアはすべて、左右の兄弟関係にリンクされています。これらは（キー、ROWID）によってソートされます。

B ツリー構造の利点

B ツリー構造の利点は、次のとおりです。

- ツリーのすべてのリーフ・ブロックは同じ深さであるため、索引内のどの位置にあるレコードを検索する場合にも、所要時間はほぼ同じになります。
- B ツリー索引は自動的にバランスが保たれます。
- 平均して、B ツリーのすべてのブロックの 4 分の 3 が満たされます。
- 完全一致や範囲検索など、広範囲な問合せに対して、B ツリーは優れた検索パフォーマンスを提供します。
- 挿入、更新および削除が効率的で、高速検索のためにキー順序が維持されます。
- B ツリーのパフォーマンスは、小さな表と大きな表のどちらでも優れているため、表のサイズが大きくなっても低下しません。

> **関連項目：** B ツリー索引の詳細は、コンピュータ・サイエンスのドキュメントを参照してください。

索引の一意スキャン

索引の一意スキャンは、データへの最も効率的なアクセス方法の 1 つです。このアクセス方法は、B ツリー索引からデータを戻す際に使用されます。一意（B ツリー）索引のすべての列が等価条件で指定されると、オプティマイザは一意スキャンを選択します。

索引レンジ・スキャン

索引レンジ・スキャンは、選択的なデータにアクセスする際の一般的な操作です。これは境界（両側に境界あり）または非有界（片側または両側）になります。データは、索引列の昇順で戻されます。値が同一の複数行は、ROWID によって昇順でソートされます。

キー圧縮

キー圧縮により、索引または索引構成表内で主キーの列値の各部を圧縮し、繰り返される値による記憶域のオーバーヘッドが低減します。

通常、索引のキーには、グループ化要素および一意要素という 2 つの要素があります。一意要素を持つようにキーを定義しなければ、Oracle Database がグループ化要素に ROWID を追加して一意要素を提供します。キー圧縮は、複数の一意要素で共有できるように、グループ化要素を分割し、格納する方法です。

この項の内容は、次のとおりです。

- 接頭辞エントリと接尾辞エントリ
- パフォーマンスと記憶域に関する考慮事項
- キー圧縮の使用方法

接頭辞エントリと接尾辞エントリ

キー圧縮により、索引キーが接頭辞エントリ（グループ化要素）と接尾辞エントリ（一意要素）に分割されます。圧縮するために、接頭辞エントリが索引ブロック内の接尾辞エントリ間で共有されます。圧縮されるのは、B ツリー索引のリーフ・ブロックのキーのみです。ブランチ・ブロックの場合、キーの接尾辞は切り捨てることができますが、キーは圧縮されません。

キー圧縮は、複数の索引ブロック間ではなく、1 つの索引ブロック内で実行されます。接尾辞エントリは、索引行の圧縮版を形成します。各接尾辞エントリは、同じ索引ブロックに格納されている接頭辞エントリを参照します。

デフォルトで、接頭辞は、最終列を除く、すべてのキー列で構成されます。たとえば、3 列 (column1, column2, column3) からなるキーの場合、デフォルトの接頭辞は (column1, column2) です。値リスト (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4) の場合は、接頭辞内で重複する (1,2)、(1,3) が圧縮されます。

また、接頭辞の長さは、接頭辞に含まれる列数として指定できます。たとえば、接頭辞の長さを 1 と指定すると、その接頭辞は column1、接尾辞は (column2, column3) となります。値リスト (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4) の場合は、接頭辞内で重複する 1 が圧縮されます。

非一意索引の接頭辞の最大長はキー列の数であり、一意索引の接頭辞の最大長は、キー列の数から 1 を差し引いた値です。

接頭辞エントリは、それと等しい値を持つ接頭辞エントリが索引ブロックに含まれていない場合にのみ、索引ブロックに書き込まれます。接頭辞エントリは、索引ブロックに書き込まれた直後から共有可能になり、最後に削除した参照側の接尾辞エントリが索引ブロックから消去されるまで使用可能のままです。

パフォーマンスと記憶域に関する考慮事項

キー圧縮を使用すると、領域が大幅に節約になり、索引ブロック当たりで格納できるキー数が増え、I/O が減少してパフォーマンスが向上します。

ただし、索引の記憶域必要量は減少しますが、索引スキャン中にキー列値を再構築するための CPU タイムが増大することがあります。また、各接頭辞エントリには、対応する 4 バイトのオーバーヘッドが生じるため、記憶域のオーバーヘッドが大きくなります。

キー圧縮の使用方法

キー圧縮は、次のように様々な状況で役立ちます。

- 通常の非一意索引の場合、Oracle Database は、重複キーを格納する際にキーに ROWID を追加して、重複行を分割します。キー圧縮を使用する場合、Oracle Database では、重複キーは、ROWID を除き接頭辞エントリとして索引ブロックに格納されます。残りの行は、ROWID のみからなる接尾辞エントリです。
- これと同じ動作は、(stock_ticker,transaction_time) など、(**項目 , タイムスタンプ**) 形式のキーを持つ一意索引にも見られます。多数の行が同じ stock_ticker 値を持ち、transaction_time によって一意性が保たれます。特定の索引ブロックでは、stock_ticker 値が接頭辞エントリとして一度のみ格納されます。索引ブロックの他のエントリでは、transaction_time 値が共通の stock_ticker 接頭辞エントリを参照する接尾辞エントリとして格納されます。
- VARRAY または NESTED TABLE データ型を含む索引構成表の場合は、コレクション・データ型の要素ごとにオブジェクト識別子が繰り返されます。キー圧縮を使用すると、重複するオブジェクト識別子の値を圧縮できます。

ただし、キー圧縮を使用できない場合があります。たとえば、単一の属性キーを持つ一意索引の場合、一意要素はありますが、共有するグループ化要素がないため、キー圧縮は使用できません。

関連項目： 5-35 ページの「索引構成表の概要」

逆キー索引

逆キー索引を作成する場合は、標準の索引とは異なり、列の順序は保ちながら、索引が定義されている各列（ROWID を除く）のバイトを逆にします。索引に対する変更が少数のリーフ・ブロックに集中する Oracle Real Application Clusters では、この機能によりパフォーマンスの低下を防ぐことができます。索引のキーを逆にすることにより、挿入値は索引のリーフ・キー全体に分散されます。

逆キーを使用すると、その索引に対する索引レンジ・スキャンは実行できなくなります。逆キー索引では辞書的に連続しているキーが隣接して格納されないため、キー指定フェッチまたは全索引（表）スキャンしか実行できません。

状況によっては、逆キー索引を使用することで、OLTP の Oracle Real Application Clusters アプリケーションが高速になる場合があります。たとえば、電子メール・アプリケーションでメール・メッセージの索引を保存し、ユーザーが古いメッセージを保存する場合、索引は最新のメールに対するポインタと同様に古いメッセージに対するポインタもメンテナンスする必要があります。

REVERSE キーワードは、逆キー索引を作成するための簡単なメカニズムを備えています。CREATE INDEX 文のオプションの索引仕様部に REVERSE キーワードを指定します。

```
CREATE INDEX i ON t (a,b,c) REVERSE;
```

逆キー索引を標準の索引に REBUILD（再作成）するには、キーワード NOREVERSE を指定します。

```
ALTER INDEX i REBUILD NOREVERSE;
```

NOREVERSE キーワードを指定しないで逆キー索引を再作成すると、逆キー索引が再作成されて生成されます。

ビットマップ索引

索引を使用する目的は、特定のキー値が含まれる行へのポインタを提供することです。通常の索引では、そのキー値を持つ行に対応する各キーの ROWID のリストが格納されます。Oracle Database では、各 ROWID とともに各キー値が繰り返し格納されます。**ビットマップ索引**では、ROWID のリストのかわりに、各キー値のビットマップを使用します。

ビットマップの各ビットは、存在する ROWID に対応します。ビットが設定されている場合は、対応する ROWID を持つ行にはキー値が含まれることになります。マッピング機能がビット位置を実際の ROWID に変換するため、内部的には別の表現が使用されていても、ビットマップ索引は通常の索引と同じ機能を果します。異なるキー値の数が多くない場合、ビットマップ索引は領域を有効に使用できます。

ビットマップ索引は、WHERE 句に指定されたいくつかの条件に対応する索引を効率的にマージします。一部の条件は満たしているがすべては満たしていない行については、表自体にアクセスする前に除外します。これによって、応答時間が短縮されます。多くの場合は劇的に改善されます。

この項の内容は、次のとおりです。

データ・ウェアハウス・アプリケーションの場合の利点

ビットマップ索引は、大量のデータと非定型問合せを扱うものの、同時実行トランザクションのレベルは高くないデータ・ウェアハウス・アプリケーションに対して効果的です。この種のアプリケーションに対するビットマップ索引の利点は次のとおりです。

- 多くの種類の非定型問合せの応答時間が短縮されます。
- 他の方式の索引と比べて使用領域が実質的に縮小されます。
- 機能の低いハードウェアでもパフォーマンスが劇的に向上します。
- パラレル DML とロードを効果的に実行できます。

従来の B ツリー索引を使用して大きな表に完全な索引を作成すると、領域という面で膨大なコストがかかります。索引は表中のデータの何倍にも膨れ上がることがあるからです。それに対して、ビットマップ索引は通常、表中で索引を付けるデータのサイズより小さくてすみます。

ビットマップ索引は、数多くの並列トランザクションがデータを更新する OLTP アプリケーションには適していません。むしろ、ユーザーがデータの更新より問合せを実行することが多い、データ・ウェアハウス・アプリケーションの意思決定支援システムに適しています。

ビットマップ索引はまた、主として大小の比較による問合せの対象とされる列には適していません。たとえば、通常特定の値との比較における WHERE 句に現れる給与の列には、B ツリー索引の方が適しています。ビットマップ化された索引は、特に AND、OR および NOT 演算子と組み合せた等価問合せでのみ有効です。

ビットマップ索引は、Oracle Database オプティマイザおよび実行エンジンと統合されます。また、他の Oracle Database 実行メソッドとシームレスに組み合せて使用できます。たとえば、オプティマイザが 2 つの表をハッシュ結合するように決めたとします。片方の表ではビットマップ索引を、もう一方の表では通常の B ツリー索引を使用します。オプティマイザはビットマップ索引と他の使用可能なアクセス方法（通常の B ツリー索引または全表スキャンなど）を検討し、可能な場合にはパラレル化も考慮に入れながら、最適な方法を選択します。

パラレル問合せおよびパラレル DML は、従来の索引のみでなく、ビットマップ索引に対しても機能します。パーティション表のビットマップ索引は、ローカル索引であることが必要です。索引のパラレル作成と連結索引もサポートされています。

カーディナリティ

ビットマップ索引の使用によるメリットは、カーディナリティの低い列、つまり表内の行数に比べて個別値の数が少ない列において、最も大きくなります。列の個別値の数が表内の行数の 1% より少ない場合、または列内の値が 100 回以上繰り返される場合は、列はビットマップ索引の候補となります。値の繰返しが少なく、カーディナリティが高い列でも、その列が問合せの WHERE 句で複雑な条件に含まれることが頻繁にある場合は、ビットマップ索引の候補になります。

たとえば、100 万行ある表で個別の値が 10,000 個ある列は、ビットマップ索引の候補です。この列に対しては、B ツリー索引よりビットマップ索引の方が効果的です。この列を他の列と組み合せて問い合せることが多い場合は、特に効果的です。

B ツリー索引は、カーディナリティの高いデータ、つまり CUSTOMER_NAME や PHONE_NUMBER といった可能な値の多いデータにおいて最も効率的です。B ツリー索引は、索引を設定したデータよりも大きくなる場合があります。一方、ビットマップ索引は、適切に使用すれば B ツリー索引より相当小さなサイズですみます。

非定型の問合せ（またそれと同様の状況）では、ビットマップ索引を使用すると問合せのパフォーマンスが劇的に向上します。問合せの WHERE 句の AND 条件と OR 条件については、結果のビットマップを ROWID に変換する前に、対応するブール演算をビットマップに直接実行すると解決をスピードアップできます。結果の行数が少なければ、全表スキャンを行う必要はないため、問合せはすぐに終了します。

ビットマップ索引の例

表 5-1 に、ある会社の顧客データの一部を示します。

表 5-1 ビットマップ索引の例

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

MARITAL_STATUS、REGION、GENDER および INCOME_LEVEL はすべて、カーディナリティの低い列です。可能な値は、MARITAL_STATUS および REGION には 3 つ、GENDER には 2 つ、INCOME_LEVEL には 4 つしかありません。そのため、これらの列にはビットマップ索引を作成するのが適切といえます。一方、CUSTOMER# はカーディナリティの高い列であるため、ビットマップ索引は作成しません。この場合は、一意の B ツリー索引を使用する方が、表示と検索が効率的になります。

表 5-2 に、この例の REGION 列に対するビットマップ索引を示します。これは、各地域に対応する 3 つのビットマップからなっています。

表 5-2 サンプル・ビットマップ

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

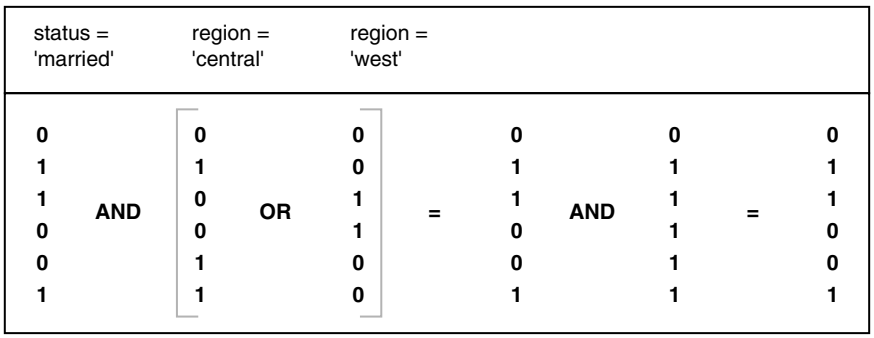
ビットマップの各エントリまたはビットは CUSTOMER 表の 1 つの行に対応しています。各ビットの値は対応する行の値に依存しています。たとえば、ビットマップ REGION='east' では、1 が最初のビットとなっています。これは CUSTOMER 表の最初の行で、REGION が east になっているためです。REGION の値が east の行は他にないため、ビットマップ REGION='east' の残りのビットは 0 です。

顧客の人口統計を調べているアナリストが、「既婚の顧客のうち central または west 地域に住んでいる人はどれくらいいるのか」尋ねてきたとします。この質問は、次の SQL 問合せで表現できます。

```
SELECT COUNT(*) FROM CUSTOMER
    WHERE MARITAL_STATUS = 'married' AND REGION IN ('central','west');
```

ビットマップ索引を使用すると、図 5-8 に示すとおり、結果のビットマップから該当する値を数えて、この問合せを非常に効果的に処理できます。基準に該当する特定の顧客を識別するときは、結果のビットマップを使用して表にアクセスできます。

図 5-8 ビットマップ索引を使用した問合せの実行



ビットマップ索引と NULL

他のほとんどのタイプの索引とは異なり、ビットマップ索引には NULL 値を持つ行が含まれる場合があります。NULL の索引作成は、集計関数 COUNT による問合せなど、いくつかのタイプの SQL 文に使用できます。

パーティション表に対するビットマップ索引

他の索引の場合と同じように、ビットマップ索引もパーティション表に対して作成できます。ただし、制限が 1 つだけあります。ビットマップ索引はパーティション表に対してローカルであることが必要です。グローバル索引にはできません。グローバル・ビットマップ索引は、非パーティション表でのみサポートされます。

関連項目：

- パーティション表と、ローカルおよびグローバル索引の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。
- 『Oracle Database VLDB およびパーティショニング・ガイド』
- NULL 値の索引作成の例を含めて、ビットマップ索引の使用方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ビットマップ結合索引

単一表には、ビットマップ索引以外に、複数の表を結合するためのビットマップ索引であるビットマップ結合索引を作成できます。ビットマップ結合索引は、制限を事前に実行するため、結合するデータのボリュームを削減する効率的な方法です。ビットマップ結合索引は、表列の値ごとに、対応する行の ROWID を 1 つ以上の別の表に格納します。データ・ウェアハウス環境では、結合条件はディメンション表の主キー列とファクト表の外部キー列の等価内部結合になります。

ビットマップ結合索引は、格納に関して、事前に結合をマテリアライズするマテリアライズド結合ビューよりもはるかに効率的です。これは、マテリアライズド結合ビューがファクト表の ROWID を圧縮しないためです。

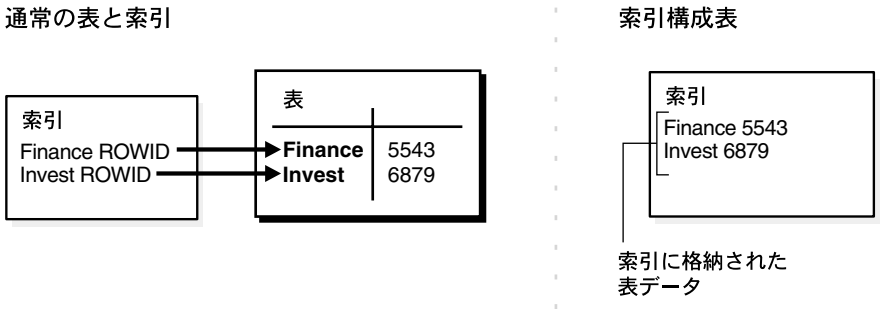
> **関連項目：** ビットマップ結合索引の詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

索引構成表の概要

索引構成表には、プライマリ B ツリーの改良型となる記憶域組織があります。データが順不同のコレクション（ヒープ）として格納される通常の表（ヒープ構成表）とは異なり、索引構成表のデータは主キーによるソート形式で B ツリー索引構造に格納されます。B ツリーの各索引エントリには、索引構成表の行の主キー列値のみでなく、非キー列値も格納されます。

索引構成表は、図 5-9 に示されているとおり、通常の表と 1 つ以上の列の索引からなる構成にやや類似していますが、データベース・システムは表と B ツリー索引という 2 つの記憶域構造を別個にメンテナンスするかわりに、1 つの B ツリー索引のみをメンテナンスします。また行の ROWID ではなく、非キー列値が索引エントリに格納されます。そのため、B ツリーの各索引エントリには `<primary_key_value,non_primary_key_column_values>` が含まれます。

図 5-9 索引構成表と比較した通常の表の構造



アプリケーションは、通常の表と同じように、SQL 文を使用して索引構成表を操作します。ただし、データベース・システムは、対応する B ツリー索引を操作することですべての操作を実行します。

表 5-3 に、索引構成表と通常の表の違いをまとめます。

表 5-3 索引構成表と通常の表の比較

通常の表	索引構成表
ROWID が行を一意に識別します。主キーはオプションで指定できます。	主キーが行を一意に識別します。主キーを指定する必要があります。
ROWID 疑似列の物理 ROWID で 2 次索引を作成できます。	ROWID 疑似列の論理 ROWID で 2 次索引を作成できます。
アクセスは ROWID に基づきます。	アクセスは論理 ROWID に基づきます。
順次スキャンはすべての行を戻します。	全索引スキャンはすべての行を戻します。
他の表とともにクラスタに格納できます。	クラスタには格納できません。
LONG データ型の列と LOB データ型の列を格納できます。	LOB 列は格納できますが、LONG 列は格納できません。
仮想列を格納できます（サポートされているのはリレーショナル・ヒープ表のみです）。	仮想列は格納できません。

この項の内容は、次のとおりです。

- 索引構成表の利点
- 行オーバーフロー領域付きの索引構成表
- 索引構成表の 2 次索引
- 索引構成表のビットマップ索引
- パーティション索引構成表
- ヒープ構成表および索引構成表の UROWID 列の B ツリー索引
- 索引構成表アプリケーション

索引構成表の利点

索引構成表は、主キーまたはその有効な接頭辞である任意のキーの使用により、表の行に対するアクセスを高速化します。B ツリーのリーフ・ブロックに行の非キー列が存在することにより、追加のブロック・アクセスが回避されます。さらに、行が主キーの順序で格納されるため、主キー（または有効な接頭辞）によるレンジ・アクセスでは最小限のブロック・アクセスのみが行われます。

アクセス頻度の高い列へのアクセスをさらに高速化するには、行オーバーフローのセグメント（次を参照）を使用して、アクセス頻度の低い非キー列を B ツリーのリーフ・ブロックからオプションの（ヒープ構造の）オーバーフロー・セグメントにプッシュできます。これにより、B ツリーのリーフ・ブロックに実際に格納される行の部分のサイズおよび内容を制限できるため、各リーフ・ブロック内の行数を多く、かつ B ツリーを小さくできる場合があります。

主キー索引を持つヒープ構成表では主キー列が表と索引の両方に格納されますが、ここでは主キー列値が B ツリー索引のみに格納されるため、そのような重複は起こりません。

行は主キーの順序で格納されるため、キー圧縮を使用すると多大な追加の記憶域が確保できます。

索引構成表の 2 次索引で主キーベースの論理 ROWID を使用すると、物理 ROWID とは逆に、高可用性が確保できます。これは ROWID の論理的性質により、実表の行が移動する表の再編成操作の後にも、2 次索引が使用不可能にはならないためです。同時に、論理 ROWID で物理推測を使用すると、2 次索引ベースの索引構成表のアクセス・パフォーマンスを実現することも可能です。これは通常の表に対する 2 次索引ベースのアクセス・パフォーマンスに匹敵します。

関連項目：

- 5-30 ページの「キー圧縮」
- 5-37 ページの「索引構成表の 2 次索引」
- 索引構成表の作成とメンテナンスの詳細は、『Oracle Database 管理者ガイド』を参照してください。

行オーバーフロー領域付きの索引構成表

B ツリー索引エントリはキー値と ROWID のみで成り立っているので、サイズは通常小さくてすみます。ただし索引構成表では、B ツリー索引エントリはすべての行で成り立っているため大きくなる場合があります。この場合、B ツリー索引の稠密クラスタ・プロパティは破棄されることがあります。

この問題に対処するために、Oracle Database には OVERFLOW 句が用意されています。必要に応じて、オーバーフロー表領域を指定して、行を次の 2 つの部分に分割できます。分割した各部分はそれぞれ、索引とオーバーフロー記憶域セグメントに格納されます。

- 索引エントリ（主キー列すべての列値、行のオーバーフロー部分を指す物理 ROWID、およびオプションで一部の非キー列を含む）
- オーバーフロー部分（残りの非キー列の列値を含む）

OVERFLOW では、PCTTHRESHOLD と INCLUDING の 2 つの句を使用して、行を 2 つの部分に分けて格納するかどうか、またその場合どの非キー列で行を分割するかの Oracle Database による判断方法を制御できます。PCTTHRESHOLD を使用すると、ブロック・サイズの割合としてしきい値を指定できます。非キー列値のすべてが指定したサイズ制限に収まる場合、行は分割されません。そうでない場合は、サイズ制限に収まらない最初の非キー列から開始して、残りの非キー列すべてが表の行オーバーフロー・セグメントに格納されます。

INCLUDING 句を使用すると、列名を指定して、その列の後の CREATE TABLE 文に現れる非キー列を行オーバーフロー・セグメントに格納させることができます。PCTTHRESHOLD ベースの制限のため、場合によっては追加の非キー列をオーバーフローに格納する必要があることに注意してください。

関連項目： OVERFLOW 句の使用例は、『Oracle Database 管理者ガイド』を参照してください。

索引構成表の 2 次索引

索引構成表の 2 次索引がサポートされることにより、主キーでもその接頭辞でもない列を使用して、索引構成表に効率よくアクセスできます。

Oracle Database は、論理行識別子を使用して索引構成表の 2 次索引を組み立てます。この識別子は、表の主キーに基づいており、**論理 ROWID** と呼ばれます。論理 ROWID には、行のブロック位置を識別する**物理推測**が含まれます。Oracle Database は、これらの物理推測を使用して、主キー検索をバイパスし、索引構成表のリーフ・ブロックを直接プローブできます。索引構成表の行には永続物理アドレスがないため、行が新しいブロックに移動すると物理推測が陳腐化することがあります。

通常の表の場合、2 次索引によるアクセスでは、行を含むデータ・ブロックをフェッチするために、2 次索引のスキャンと付加的な I/O が必要になります。索引構成表の場合、2 次索引によるアクセスは、次のように物理推測を使用するかどうかと、その正確さに応じて異なります。

- 物理推測を使用しない場合、アクセスでは 2 次索引のスキャンとその後に主キー索引のスキャンという、2 つの索引スキャンが行われます。
- 正確な物理推測を使用する場合、アクセス時には、2 次索引スキャンと追加の I/O によって行を含むデータ・ブロックがフェッチされます。
- 不正確な物理推測を使用する場合、アクセス時には 2 次索引スキャンと I/O によって間違ったデータ・ブロック（物理推測が示すブロック）がフェッチされてから、主キー索引がスキャンされます。

関連項目： 26-18 ページの「論理 ROWID」

索引構成表のビットマップ索引

Oracle Database では、パーティション化または非パーティション化された索引構成表のビットマップ索引がサポートされています。索引構成表のビットマップ索引の作成には、マッピング表が必要です。

マッピング表

マッピング表は、索引構成表の論理 ROWID を格納するヒープ構成表です。具体的には、マッピング表の各行には対応する索引構成表の行の論理 ROWID が格納されます。すなわちマッピング表では、索引構成表の行の論理 ROWID とマッピング表の行の物理 ROWID の 1 対 1 のマッピングができます。

索引構成表のビットマップ索引はヒープ構成表のビットマップ索引に似ていますが、索引構成表のビットマップ索引で使用される ROWID は実表の ROWID ではなく、マッピング表の ROWID です。索引構成表にはそれぞれ 1 つのマッピング表があり、その索引構成表で作成されたすべてのビットマップ索引がそのマッピング表を使用します。

ビットマップ索引には、ヒープ構成と索引構成のどちらの実表においても、検索キーを使用してアクセスします。キーが見つかると、ビットマップのエントリは物理 ROWID に変換されます。ヒープ構成表の場合、この物理 ROWID は実表へのアクセスに使用されます。索引構成表の場合は、マッピング表へのアクセスに使用されます。マッピング表にアクセスすると、論理 ROWID が取得できます。この論理 ROWID は索引構成表へのアクセスに使用されます。

索引構成表のビットマップ索引は論理 ROWID を格納しませんが、性質は論理的です。

注意： 索引構成表で行を移動すると、その索引構成表に作成されたビットマップ索引の使用禁止状態が変化します。索引構成表で行を移動すると、マッピング表の一部の論理 ROWID エントリにおける物理推測が無効になります。それでも、索引構成表には主キーを使用してアクセスできます。

パーティション索引構成表

列値の範囲、ハッシュまたはリストごとに索引構成表をパーティション化できます。パーティション化列は、主キー列のサブセットを形成する必要があります。通常の表と同様、パーティション化索引構成表にはローカル・パーティション化された（同一キーおよび非同一キーの）索引もグローバル・パーティション化された（同一キーの）索引もサポートされています。

ヒープ構成表および索引構成表の UROWID 列の B ツリー索引

`UROWID` データ型の列は、索引構成表の行を特定する論理主キーベースの ROWID を保持できます。Oracle Database では、ヒープ構成表または索引構成表の `UROWID` データ型の索引をサポートしています。索引の `UROWID` 列では、等価述語をサポートしています。等価以外の述語、または `UROWID` データ型の列の順序付けのための述語の場合、索引は使用されません。

索引構成表アプリケーション

主キーベースのアクセスにおける優れた問合せパフォーマンス、高可用性の要素、および記憶域に関する要件の少なさから、索引構成表は次のようなアプリケーションに最適といえます。

- オンライン・トランザクション処理（OLTP）
- インターネット（検索エンジンやポータルなど）
- E-Commerce（電化製品店やカタログなど）
- データ・ウェアハウス
- 分析関数

アプリケーション・ドメイン索引の概要

Oracle Database は、複合データ型（文書、空間データ、イメージおよびビデオ・クリップなど）の索引に対応し、特化した索引作成テクニックを使用するために、**拡張索引作成機能**を提供します。この機能を使用すると、アプリケーション固有の索引管理ルーチンを**索引タイプ**・スキーマ・オブジェクトとしてカプセル化し、オブジェクト型の表の列や属性の**ドメイン索引**（アプリケーション固有の索引）を定義できます。また、拡張可能索引作成機能により、アプリケーション固有の**演算子**を効率的に処理できます。

ドメイン索引の構造と内容は、**カートリッジ**と呼ばれるアプリケーション・ソフトウェアによって制御されます。Oracle データベース・サーバーは、ドメイン索引の作成、メンテナンスおよび検索のために、このアプリケーションと対話します。索引構造そのものは、索引構成表として Oracle データベースに格納するか、ファイルとして外部に格納できます。

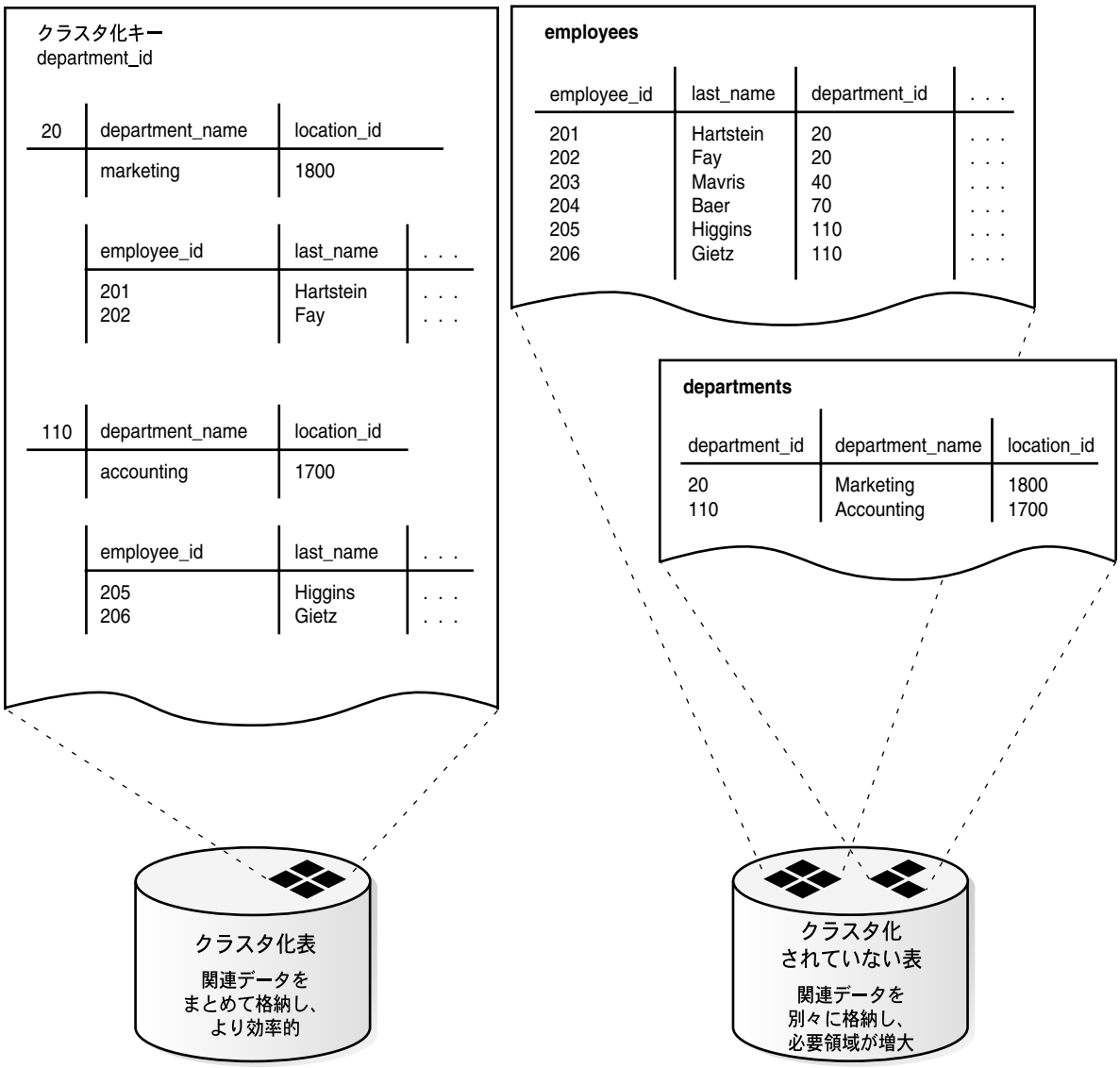
> **関連項目：** Oracle データベースの拡張性アーキテクチャ内のデータ・カートリッジの使用方法は、『Oracle Database データ・カートリッジ開発者ガイド』を参照してください。

クラスタの概要

クラスタは、表データを格納するためのオプションの方法です。クラスタは同じデータ・ブロックを共有する表のグループです。これらの表グループは共通の列を共有し、頻繁に併用されます。たとえば、`employees` および `departments` 表は `department_id` 列を共有しています。`employees` 表と `departments` 表をクラスタ化すると、Oracle Database では `employees` 表と `departments` 表における各部門の行すべてが、同じデータ・ブロックに物理的に格納されます。

図 5-10 に、employees 表と departments 表をクラスタ化する際の例を示します。

図 5-10 クラスタ化表データ



クラスタは、異なる表の関連した行を、同じデータ・ブロック内にまとめて格納するため、クラスタを適切に使用すると、次のような利点があります。

- クラスタ化表を結合する場合のディスク I/O が減少します。
- クラスタ化表を結合する場合のアクセス時間が改善されます。
- クラスタでは、特定の行のクラスタ・キー列が**クラスタ・キー値**になります。各クラスタ・キー値は、その値を持つ行がいくつかの表に含まれているとしても、クラスタとクラスタ索引にそれぞれ一度のみ格納されます。そのため、クラスタとして関連付けた表データと索引データを格納するために必要な記憶域は、クラスタ化していない表に必要な記憶域よりも少なくなります。たとえば、図 5-10 では、各クラスタ・キー（各 department_id）は、employees 表と departments 表の両方で同じ値を持つ多数の行に対して一度のみ格納されていることに注意してください。

  関連項目： クラスタの作成および管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。

ハッシュ・クラスタの概要

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

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

ハッシングは、データ検索のパフォーマンスを改善するために表データを格納するオプションの方法です。ハッシングを使用するには、ハッシュ・クラスタを作成し、そのクラスタに表をロードします。Oracle Database では、表の行はハッシュ・クラスタ内に物理的に格納され、ハッシュ関数の結果に従って取得されます。

ソート済のハッシュ・クラスタを使用すると、データが挿入順に使用されるアプリケーションでは、データの検索を高速化できます。

Oracle Database により、ハッシュ関数を使用して生成される数値の分布（**ハッシュ値**と呼ばれる）は、特定のクラスタ・キーの値に基づいています。ハッシュ・クラスタのキーは、索引クラスタのキーと同様に、単一の列またはコンポジット・キー（複数列のキー）にすることができます。Oracle Database では、ハッシュ・クラスタ内で行を検索または格納するために、ハッシュ関数を行のクラスタ・キー値に適用します。結果として得られるハッシュ値は、クラスタ内の 1 つのデータ・ブロックに対応します。Oracle Database は、発行された文による読取り / 書込みをそのデータ・ブロックに対して実行します。

ハッシュ・クラスタは、索引を持つクラスタ化されていない表、または索引クラスタにかわるものです。索引付きの表または索引クラスタでは、Oracle Database は、別個の索引に格納されたキー値を使用して、表内の行を見つけます。索引付きの表またはクラスタの行を検索または格納するには、次のように最低 2 回の I/O を実行する必要があります。

- 索引内でキー値を検索または格納するために 1 回以上の I/O
- 表またはクラスタ内の行を読取り / 書込みするためにさらに 1 回の I/O

> **関連項目：** ハッシュ・クラスタの作成と管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。

6

スキーマ・オブジェクトの依存性

オブジェクト A の定義がオブジェクト B を参照している場合、A は B に依存します。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 Database 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。こうした自動依存性管理により、A によって B の古いバージョンが使用されることがなく、B の変更後に A を明示的に再コンパイルする必要もほとんどなくなります。

この項の内容は、次のとおりです。

スキーマ・オブジェクトの依存性の概要

スキーマ・オブジェクトのタイプによっては、定義の中で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、サブプログラムの本体の中には、他のオブジェクトを参照する SQL 文を含めることができます。オブジェクト A の定義がオブジェクト B を参照している場合、A は（B に関する）**依存オブジェクト**、B は（A に関する）**参照オブジェクト**です。

次の問合せでは、現在のデータベース内で他のオブジェクトに依存するオブジェクトのタイプが示されます。

```
SELECT DISTINCT TYPE
  FROM DBA_DEPENDENCIES
     [ORDER BY TYPE]
```

次の問合せでは、現在のデータベース内で他のオブジェクトから参照されるオブジェクトのタイプが示されます。

```
SELECT DISTINCT REFERENCED_TYPE
  FROM DBA_DEPENDENCIES
    [ORDER BY REFERENCED_TYPE]
```

例 6-1 の SQL*Plus スクリプトは、前述の 2 つの問合せによる出力を示しています。

例 6-1 依存オブジェクト・タイプと参照オブジェクト・タイプの表示

```
SQL> SELECT DISTINCT TYPE
  2    FROM DBA_DEPENDENCIES
  3      ORDER BY TYPE;

TYPE
------------------
DIMENSION
EVALUATION CONTXT
FUNCTION
INDEX
INDEXTYPE
JAVA CLASS
JAVA DATA
MATERIALIZED VIEW
OPERATOR
PACKAGE
PACKAGE BODY
PROCEDURE
RULE
RULE SET
SYNONYM
TABLE
TRIGGER
TYPE
TYPE BODY
UNDEFINED
VIEW
XML SCHEMA

22 rows selected.

SQL> SELECT DISTINCT REFERENCED_TYPE
  2    FROM DBA_DEPENDENCIES
  3      ORDER BY REFERENCED_TYPE;

REFERENCED_TYPE
------------------
EVALUATION CONTXT
```

```
FUNCTION
INDEXTYPE
JAVA CLASS
LIBRARY
NON-EXISTENT
OPERATOR
PACKAGE
PROCEDURE
SEQUENCE
SYNONYM
TABLE
TYPE
VIEW
XML SCHEMA

15 rows selected.

SQL>
```

参照オブジェクトの定義を変更した場合、その変更の内容によって、依存オブジェクトがエラーなしで機能し続ける場合と、そうでない場合があります。たとえば、表を削除した場合、その表に基づくビューは使用できなくなります。

一部の依存性のみを無効にするスキーマ・オブジェクトの変更例として、例 6-2 に示す 2 つのビューを考えます。このビューは、HR.EMPLOYEES 表を基にしたものです。

EMPLOYEES 表の EMAIL 列を延長するよう指定するとします。この表を次のように変更します。

```
ALTER TABLE employees MODIFY email VARCHAR2(100);
```

COMMISSIONED ビューは、その選択リストに EMAIL が含まれていないため、無効化されません。一方 SIXFIGURES ビューは、表内のすべての列が選択されているため、無効化の対象となります。

```
select object_name, status from user_objects where object_type = 'VIEW';

OBJECT_NAME          STATUS
-------------------- -------
COMMISSIONED         VALID
SIXFIGURES           INVALID
```

例 6-2 一部の依存性を無効化するスキーマ・オブジェクトの変更

```
CREATE VIEW commissioned AS
SELECT first_name, last_name, commission_pct FROM employees
WHERE commission_pct > 0.00;

CREATE VIEW sixfigures AS
SELECT * FROM employees
WHERE salary >= 100000;

select object_name, status from user_objects where object_type = 'VIEW';

OBJECT_NAME          STATUS
-------------------- -------
COMMISSIONED         VALID
SIXFIGURES           VALID
```

ビューは、その問合せ内で参照されているすべてのオブジェクトに依存します。例 6-3 に示した oc_inventories ビューは、オブジェクト型 inventory_typ、ファンクション warehouse_typ、表 inventories および warehouse に依存しています。

例 6-3 複数のオブジェクトに依存するビュー

```
CREATE TYPE inventory_typ
  OID '82A4AF6A4CD4656DE034080020E0EE3D'
  AS OBJECT
    ( product_id          NUMBER(6)
    , warehouse           warehouse_typ
    , quantity_on_hand    NUMBER(8)
    ) ;
/
CREATE OR REPLACE VIEW oc_inventories OF inventory_typ
  WITH OBJECT OID (product_id)
  AS SELECT i.product_id,
            warehouse_typ(w.warehouse_id, w.warehouse_name, w.location_id),
            i.quantity_on_hand
     FROM inventories i, warehouses w
     WHERE i.warehouse_id=w.warehouse_id;
```

注意：

- CREATE 文を実行すると、すべての依存性が自動的に更新されます。
- 動的 SQL 文では、依存性は作成されません。たとえば次の文を実行しても、tab1 に対する依存性は作成されません。

```
EXECUTE IMMEDIATE 'SELECT * FROM tab1 ...'
```

オブジェクト依存性の問合せ

静的データ・ディクショナリ・ビュー USER_DEPENDENCIES、ALL_DEPENDENCIES および DBA_DEPENDENCIES には、データベース・オブジェクト間の依存性が表示されます。

utldtree.sql SQL スクリプトを実行すると、オブジェクト依存性ツリーに関する情報が表示されるビュー DEPTREE、ビュー IDEPTREE およびあらかじめソートされ整形表示される DEPTREE が作成されます。

関連項目： DEPTREE、IDEPTREE および utldtree.sql スクリプトの詳細は、『Oracle Database リファレンス』を参照してください。

オブジェクトのステータス

各データベース・オブジェクトには、表 6-1 に示されているいずれかのステータス値が割り当てられています。

表 6-1 データベース・オブジェクトのステータス

ステータス	意味
有効	データ・ディクショナリ内の現在の定義を使用して正常にコンパイルされた。
コンパイル・エラー	直前のコンパイル実行中にエラーが発生した。
無効	参照先のオブジェクトが変更されたために無効のマークが付けられている。（無効になるのは依存オブジェクトのみ。）
権限取消し	参照オブジェクトに対するアクセス権限が取り消された。（権限取消しの対象となるのは依存オブジェクトのみ。）

注意： 静的データ・ディクショナリ・ビュー USER_OBJECTS、ALL_OBJECTS および DBA_OBJECTS では、「コンパイル・エラー」、「無効」、「権限取消し」はすべて同じものとみなされ、INVALID と表示されます。

依存オブジェクトの無効化

オブジェクト A がオブジェクト B に、オブジェクト B がオブジェクト C にそれぞれ依存している場合、A、B はそれぞれ、B、C への**直接依存性**を持ち、A は C への**間接依存性**を持ちます。

直接依存性は、それ自体に影響を与える変更（参照オブジェクトのシグネチャに対する変更）を参照オブジェクトに対して加えた場合にのみ無効になります。

間接依存性は、それ自体に影響を与えない変更を参照オブジェクトに対して加えることにより無効にできます。たとえば、C を変更したことにより B が無効になると、A（および B への直接依存性と間接依存性のすべて）も無効になります。これを**連鎖的な無効化**と呼びます。

表 6-2 は、各オブジェクトに対して、それが依存する他のオブジェクトに変更があった場合どのような影響があるかを示したものです。

表 6-2 オブジェクト・ステータスに影響を与える操作

操作	影響のあった依存オブジェクトのステータス
ALTER TABLE *table* ADD *column*	次の場合は INVALID ■ 依存オブジェクト（ビューを除く）で、*table* に SELECT * が使用された場合。 ■ 依存オブジェクトで、*table*%rowtype が使用された場合。 ■ 依存オブジェクトで、列リストを指定することなく *table* に対して INSERT が実行された場合。 ■ 依存オブジェクトが、SQL の結合が含まれる問合せ内の *table* を参照している場合。 ■ 依存オブジェクトが、PL/SQL 変数を参照する問合せ内の *table* を参照している場合。 これ以外の場合は変更なし。
ALTER TABLE *table* {MODIFY | RENAME | DROP | SET UNUSED} *column* ALTER TABLE *table* DROP CONSTRAINT *not_null_constraint*	次の場合は INVALID ■ 依存オブジェクトが *column* を直接参照している場合。 ■ 依存オブジェクトで、*table* に SELECT * が使用された場合。 ■ 依存オブジェクトで、*table*%rowtype が使用された場合。 ■ 依存オブジェクトで、列リストを指定することなく *table* に対して INSERT が実行された場合。 これ以外の場合は変更なし。

表 6-2 オブジェクト・ステータスに影響を与える操作（続き）

操作	影響のあった依存オブジェクトのステータス
CREATE OR REPLACE VIEW *view* オンライン表再定義（DBMS_REDEFINITION）	新しい定義の列リストと以前の定義の列リストが異なり、少なくとも次の 1 つに該当する場合は INVALID ■ 依存オブジェクトが、ビューまたは表の新しい定義の中で修正または削除された列を参照している場合。 ■ 依存オブジェクトで、*view*%rowtype または *table*%rowtype が使用されている場合。 ■ 依存オブジェクトで、列リストを指定することなくビューまたは表に対して INSERT が実行された場合。 ■ ビューの新しい定義に新しい列が追加され、かつ SQL の結合が含まれている問合せ内のビューまたは表を依存オブジェクトが参照している場合。 ■ ビューの新しい定義に新しい列が追加され、PL/SQL を参照する問合せ内のビューまたは表を依存オブジェクトが参照している場合。 ■ 依存オブジェクトが、RELIES ON 句の中のビューまたは表を参照している場合。 これ以外の場合は変更なし。
CREATE OR REPLACE SYNONYM *synonym*	次の場合は INVALID ■ 新旧の *synonym* のターゲットが異なり、かつ一方が表ではない場合。 ■ *synonym* の新旧のターゲットがどちらも表で、かつその 2 つの表の列リストまたは付与権限が異なる場合。 ■ *synonym* の新旧のターゲットがどちらも表で、かつ依存オブジェクトがビューであり、そのビューが参照している列が以前のターゲットの一意索引には関与し、新しいターゲットの一意索引には関与していない場合。 これ以外の場合は変更なし。
RENAME {TABLE|VIEW|SEQUENCE|SYNONYM}	INVALID
DROP INDEX	次の場合、索引が構築されている表の依存性は INVALID ■ 索引がファンクション索引であり、かつ依存オブジェクトがトリガーである場合。 ■ 索引が一意索引であり、依存オブジェクトがビューであって、かつそのビューが、一意索引に関与している列を参照している場合。
DROP *object*	INVALID

表 6-2 オブジェクト・ステータスに影響を与える操作（続き）

操作	影響のあった依存オブジェクトのステータス
CREATE OR REPLACE {PROCEDURE|FUNCTION}	コール・シグネチャが変更された場合は INVALID。コール・シグネチャとは、パラメータ・リスト（パラメータの順序、名前および型）、戻り型、純粋度[1]、決定性、並列度、パイプライン処理および（プロシージャまたはファンクションが C または Java で実装されている場合は）実装の各プロパティを指します。 プロシージャ本体やファンクション本体への変更など、これ以外の変更に対しては有効。
CREATE OR REPLACE PACKAGE	次の場合は INVALID ■ 依存オブジェクトが、削除または名前変更されたパッケージ項目を参照している場合。 ■ 依存オブジェクトが参照しているパッケージ・プロシージャまたはパッケージ・ファンクションのコール・シグネチャまたはエントリ・ポイント番号[2] が変更された場合。 参照されているプロシージャまたはファンクションにオーバーロードの候補が複数ある場合、それらの候補のコール・シグネチャまたはエントリ・ポイント番号が変更されるか、候補が追加または削除されると、依存オブジェクトは無効。 ■ 依存オブジェクトが参照しているパッケージ・カーソルのコール・シグネチャ、行タイプまたはエントリ・ポイント番号が変更された場合。 ■ 依存オブジェクトが参照しているパッケージ・タイプまたはパッケージ・サブタイプの定義が変更された場合。 ■ 依存オブジェクトが参照しているパッケージの変数または定数の名前、データ型、初期値、オフセット番号のいずれかが変更された場合。 ■ パッケージ純粋度[1] が変更された場合。 これ以外の場合は変更なし。
CREATE OR REPLACE PACKAGE BODY	変更なし。
REVOKE *DML-object-privilege*[3] ON *object* FROM *user*	*object* に依存している *user* のオブジェクトはすべて INVALID[4]
REVOKE *DML-object-privilege*[3] ON *object* FROM PUBLIC	*object* に依存しているデータベース内のオブジェクトはすべて INVALID[4]

[1] **純粋度**とは、SQL 問合せ内部で PL/SQL ファンクションを起動する際に不測の影響（予期せぬデータ変更など）が生じるのを回避するための一連のルールです。**パッケージ純粋度**とは、パッケージ初期化ブロック内のコードの純粋度を指します。

[2] プロシージャやファンクションの**エントリ・ポイント番号**は、PL/SQL パッケージ・コードにおける位置によって決まります。PL/SQL パッケージの最後に追加されたプロシージャやファンクションには、新しいエントリ・ポイント番号が付与されます。

[3] DML オブジェクト権限は、SELECT、INSERT、UPDATE、DELETE および EXECUTE です。

[4] 再有効化する場合には、再コンパイルする必要はありません。詳細は、6-10 ページの「無効な PL/SQL オブジェクトの即時再有効化」を参照してください。

この項の内容は、次のとおりです。

- セッションの状態と参照パッケージ
- セキュリティ認可

セッションの状態と参照パッケージ

パッケージ構成を参照するセッションごとに、各パッケージの独自のインスタンスが存在します。このインスタンスには、パブリック変数とプライベート変数、カーソルおよび定数の永続的な状態が含まれます。その後、セッションのインスタンス化されたパッケージのいずれかが無効になった後、再度有効化されると、そのセッションのパッケージのインスタンス化は、状態も含めてすべて失われる可能性があります。

セキュリティ認可

ユーザーや PUBLIC に対して DML オブジェクト権限やシステム権限の付与または取消しが実行されたことを検知すると、Oracle Database は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 Database 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。

無効化を回避するためのガイドライン

依存オブジェクトの無効化を回避するには、次のガイドラインに従ってください。

- パッケージ最後尾への新しい項目の追加
- ビューによる各表の参照

パッケージ最後尾への新しい項目の追加

パッケージに新しい項目を追加する場合、その項目はパッケージの最後尾に追加します。これにより、パッケージの最上位にある既存項目は、エントリ・ポイント番号が維持され、無効になることはありません。

たとえば、次のパッケージを考えてみます。

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
END;
```

次のように pkg1 の最後尾に項目を追加した場合、get_var ファンクションを参照する依存性が無効になることはありません。

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
```

次のように get_var ファンクションと set_var プロシージャの間に項目を挿入すると、set_var ファンクションを参照する依存性は無効になります。

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE assert_var (v VARCHAR2);
  PROCEDURE set_var (v VARCHAR2);
END;
```

ビューによる各表の参照

ビューを使用して表を間接的に参照します。これにより、次のような操作が可能となります。

- 依存ビューまたは依存 PL/SQL オブジェクトを無効にすることなく表に列を追加
- 依存オブジェクトを無効にすることなく、ビューが参照していない列を修正または削除

文 CREATE OR REPLACE VIEW を実行した場合、新しい ROWTYPE と以前の ROWTYPE が一致していれば、既存のビューまたはその依存性が無効になることはありません。

オブジェクトの再有効化

有効でないオブジェクトを参照する場合は、そのオブジェクトが有効化され使用可能な状態になる必要があります。オブジェクトは、参照された時点で自動的に有効になるため、ユーザーがなんらかの操作を明示的に行う必要はありません。

有効でないオブジェクトは、コンパイル・エラー、権限取消し、無効のいずれかのステータスを持ちます。これらの用語の定義は、表 6-1 を参照してください。

この項の内容は、次のとおりです。

コンパイル時にエラーが発生したオブジェクトの再有効化

コンパイル中にエラーが発生したオブジェクトは、コンパイラで自動的に再有効化することはできません。こうしたオブジェクトは、コンパイラにより再コンパイルされ、エラーが発生しなければ再有効化されます。エラーが発生した場合は無効のままです。

権限が取り消されたオブジェクトの再有効化

コンパイラでは、権限が取り消されたオブジェクトに対して、その参照オブジェクトすべてに対するアクセス権限があるかどうかのチェックが行われます。アクセス権限がある場合は、権限が取り消されたオブジェクトは、再コンパイルされることなく再有効化されます。アクセス権限がない場合は、それを示すエラー・メッセージがコンパイラにより表示されます。

無効な SQL オブジェクトの再有効化

SQL コンパイラでは、無効なオブジェクトの再コンパイルが行われます。オブジェクトは、再コンパイル中にエラーが発生しなければ再有効化されます。エラーが発生した場合は無効のままです。

無効な PL/SQL オブジェクトの再有効化

PL/SQL コンパイラでは、無効な PL/SQL プログラム・ユニット（プロシージャ、ファンクションまたはパッケージ）について、それ自体に影響を与えるような変更が参照オブジェクトに対して加えられているかどうかのチェックが行われます。変更されている場合、コンパイラでは無効なオブジェクトの再コンパイルが行われます。オブジェクトは、再コンパイル中にエラーが発生しなければ再有効化されます。エラーが発生した場合は無効のままです。変更されていなければ、無効なオブジェクトは再コンパイルされることなく再有効化されます。「無効な PL/SQL オブジェクトの即時再有効化」を参照してください。

無効な PL/SQL オブジェクトの即時再有効化

PL/SQL コンパイラでは、無効な PL/SQL プログラム・ユニット（プロシージャ、ファンクションまたはパッケージ）について、それ自体に影響を与えるような変更が参照オブジェクトに対して加えられているかどうかのチェックが行われます。変更されていなければ、無効なオブジェクトは再コンパイルされることなく再有効化されます。連鎖的な無効化により無効になったオブジェクトの再有効化は通常、即時に行われます。

たとえば、次の表、パッケージおよびプロシージャを考えてみます。

```
CREATE TABLE tab1(n NUMBER);

CREATE OR REPLACE PACKAGE pkg1 IS
  TYPE rec1 IS tab1%ROWTYPE;  -- pkg1 depends on tab1
  PROCEDURE p(n NUMBER);
END pkg1;

CREATE OR REPLACE PROCEDURE proc1 IS
BEGIN
  pkg1.p(5);  -- proc1 depends on pkg1
END proc1;
```

次の文を実行すると、pkg1（tab1 に依存）が無効になります。これにより、proc1（pkg1 に依存）も連鎖的に無効となります。

```
ALTER TABLE tab1 ADD(v VARCHAR2(20));
```

ただし、pkg1.p のシグネチャは変更されていないため、PL/SQL コンパイラでは、proc1 を再コンパイルすることなく再度有効にすることができます。

スキーマの範囲内での名前解決

SQL 文で参照されているオブジェクトの名前は、1 つ以上の要素で構成されています。各構成要素は、ピリオドで区切られます（例：hr.employees.department_id）。

Oracle Database では、次の手順に従ってオブジェクト名の解決が実行されます。

1. オブジェクト名の**先頭の要素**を修飾します。

   オブジェクト名に要素が 1 つしかない場合は、その要素が最初の要素となります。要素が 2 つ以上ある場合は、最も左にあるピリオドの左側の要素が先頭の要素です。たとえば、hr.employees.department_id では、hr が先頭の要素です。

   先頭の要素を修飾する手順は次のとおりです。

   a. オブジェクト名が、SELECT 文の FROM 句に含まれる表名であり、かつ 2 つ以上の構成要素を持つ場合は、手順 d に進みます。それ以外の場合は、手順 b に進みます。

   b. 現在のスキーマ内で、最初の要素と一致する名前を持つオブジェクトが検索されます。

      見つかった場合は、手順 2 に進みます。それ以外の場合は、手順 c に進みます。

   c. 最初の要素と一致するパブリック・シノニムが検索されます。

      見つかった場合は、手順 2 に進みます。それ以外の場合は、手順 d に進みます。

   d. 最初の要素と名前が一致するスキーマが検索されます。

      該当するスキーマが見つかった場合、オブジェクト名に後続の要素があれば、手順 e に進みます。それ以外の場合は、エラーが戻され、オブジェクト名を修飾できません。

e. 手順 d で見つかったスキーマ内で、オブジェクト名の 2 番目の要素と一致する名前を持つ組込みファンクションが検索されます。

該当する組込みファンクションが見つかった場合は、スキーマによりその組込みファンクションが再定義されています。オフジェクト名は、スキーマによって定義された同名のファンクションではなく、元の組込みファンクションに解決されます。手順 2 に進みます。

該当する組込みファンクションが見つからなかった場合は、エラーが戻され、オブジェクト名を修飾できません。

2. スキーマ・オブジェクトは修飾されています。オブジェクト名の残りの要素は、このスキーマ・オブジェクトの有効な部分と一致する必要があります。

たとえば、オブジェクト名が `hr.employees.department_id` である場合、`hr` がスキーマとして修飾されます。`employees` が表として修飾されている場合、`department_id` はその表の列に対応する必要があります。また、`employees` がパッケージとして修飾されている場合、`department_id` はそのパッケージのパブリック定数、変数、プロシージャまたはファンクションに対応する必要があります。

Oracle Database による参照の解決では、オブジェクトは、他のオブジェクトが存在しないという事実に依存する場合があります。このような状況が発生するのは、依存オブジェクトで使用されている参照の解釈が、別のオブジェクトの存在により多義的になる場合です。

関連項目： 詳細は、『Oracle Database 管理者ガイド』を参照してください。

ローカル依存性の管理

ローカル依存性の管理が行われるのは、Oracle Database において、単一データベース内のオブジェクト間で依存性が管理される場合です。たとえば、プロシージャ内の文が、同じデータベース内の表を参照する場合があります。

リモート依存性の管理

リモート依存性の管理が行われるのは、Oracle Database において、ネットワークを介した分散環境で依存性が管理される場合です。たとえば、Oracle Forms トリガーは、データベース内のスキーマ・オブジェクトに依存する場合があります。また、分散データベースにおいて、ローカル・ビューの定義問合せがリモート表を参照する場合があります。

Oracle Database では、分散データベースの依存性の管理も行われます。たとえば、Oracle Forms アプリケーション内のトリガーが表を参照する場合があります。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。Oracle Database では、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。

この項の内容は、次のとおりです。

- 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
- その他のリモート・オブジェクト間の依存性
- アプリケーションの依存性

ローカルおよびリモートのデータベース・プロシージャ間の依存性

ファンクション、パッケージ、トリガーなど、ストアド・プロシージャの相互間の依存性は、分散データベース・システムにおいてはタイムスタンプのチェックまたはシグネチャのチェックを使用して管理されます（6-13 ページの「タイムスタンプのチェック」および 6-15 ページの「シグネチャのチェック」を参照してください）。

タイムスタンプとシグネチャのどちらでリモート依存性を管理するかについては、動的初期化パラメータ `REMOTE_DEPENDENCIES_MODE` で指定します。

関連項目：『Oracle Database PL/SQL 言語リファレンス』

その他のリモート・オブジェクト間の依存性

Oracle Database では、ローカル・プロシージャとリモート・プロシージャの間の依存性を除いては、リモート・スキーマ・オブジェクト間の依存性は管理されません。

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

表の変更後にビューとプロシージャが使用され、そのビューとプロシージャがエラーとなった場合でも、ローカル・ビューとプロシージャは無効になりません。この場合、ビューとプロシージャを、エラーが戻されないように手動で変更する必要があります。このような場合は、依存オブジェクトを不必要に再コンパイルするより、依存性を管理しないことをお薦めします。

アプリケーションの依存性

データベース・アプリケーションのコードでは、接続されているデータベース内のオブジェクトを参照できます。たとえば、OCI およびプリコンパイラ・アプリケーションは、無名 PL/SQL ブロックを発行できます。Oracle Forms アプリケーション内のトリガーは、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。Oracle Database では、アプリケーション間の依存性の追跡は、自動的には行われません。

関連項目： 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・システム固有のマニュアルを参照してください。

リモート・プロシージャ・コール（RPC）の依存性管理

リモート・プロシージャ・コール（RPC）の依存性管理が行われるのは、分散データベース・システムでローカル・ストアド・プロシージャによりリモート・プロシージャがコールされる場合です。

この項の内容は、次のとおりです。

タイムスタンプのチェック

タイムスタンプのチェックの依存性モデルでは、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、変更または置換された時刻）がデータ・ディクショナリに記録されます。タイムスタンプは、プロシージャが作成、変更または置換された時刻の記録です。さらに、コンパイル済プロシージャには、それが参照するリモート・プロシージャごとに、スキーマ、パッケージ名、プロシージャ名、タイムスタンプなどの情報も含まれています。

依存プロシージャが使用されると、Oracle Database 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カレント・タイムスタンプを比較します。比較結果に応じて、次の 2 つの処理が発生します。

- タイムスタンプが一致すると、ローカル・プロシージャとリモート・プロシージャはコンパイルされずに実行されます。
- リモートで参照されたプロシージャのいずれかのタイムスタンプの比較結果が一致しない場合、ローカル・プロシージャは無効となり、コール元の環境にエラーが戻されます。さらに、新しいタイムスタンプを持つそのリモート・プロシージャに依存する他のすべてのローカル・プロシージャも無効になります。たとえば、いくつかのローカル・プロシージャがリモート・プロシージャをコールしており、そのリモート・プロシージャが再コンパイルされたとします。ローカル・プロシージャの 1 つが実行され、リモート・プロシージャのタイムスタンプと一致しないことがわかると、そのリモート・プロシージャに依存するローカル・プロシージャはすべて無効になります。

ローカル・プロシージャ本体の文によりリモート・プロシージャが実行されると、実際のタイムスタンプが比較されます。分散データベースの通信リンクを使用してタイムスタンプが比較されるのは、このときのみです。したがって、ローカル・プロシージャの文のうち無効なプロシージャ・コールより前にあるすべての文は、正常に実行される可能性があります。無効なプロシージャ・コールより後の文はまったく実行されません。コンパイルが必要です。

無効なプロシージャ・コールの実行方法によっては、その前に実行された DML 文がロールバックされます。たとえば、次の例では、PL/SQL ブロックの変更全体がロールバックされるため、UPDATE の結果はロールバックされます。

```
BEGIN
UPDATE table set ...
invalid_proc;
COMMIT;
END;
```

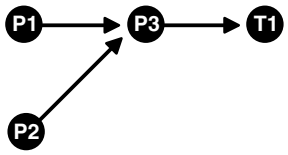
ただし、次の例では、UPDATE の結果は確定されます。この場合、ロールバックされるのは PROC コールのみです。

```
UPDATE table set ...
EXECUTE invalid_proc;
COMMIT;
```

タイムスタンプを使用して PL/SQL プログラム・ユニット間の依存性を処理する場合は、プログラム・ユニットまたは関連するスキーマ・オブジェクトを変更するたびに、そのすべての依存ユニットに無効のマークが付けられるため、実行するためには再コンパイルする必要があります。

各プログラム・ユニットでは、作成時または再コンパイル時にサーバーで設定されたタイムスタンプが保持されます。図 6-1 は、それを図示したものです。まず、プロシージャ P1 および P2 は、ストアド・プロシージャ P3 をコールします。ストアド・プロシージャ P3 は表 T1 を参照します。この例では、各プロシージャはいずれも表 T1 に依存しています。P3 は T1 に直接依存しますが、P1 および P2 は T1 に間接的に依存します。

図 6-1 依存関係



P1 および P2 が P3 と同じサーバー上にある場合に P3 が変更されると、すぐに無効のマークが付けられます。P1 および P2 がコンパイルされた状態では、P3 のタイムスタンプのレコードが含まれています。そのため、プロシージャ P3 が変更され再コンパイルされた場合、P3 上のタイムスタンプは、P1 および P2 のコンパイル中に P3 に対して記録された値とは一致しなくなります。

P1 および P2 は、クライアント・システム上または分散環境内の別の Oracle Database インスタンス上にある場合、実行時にタイムスタンプ情報を使用してこの 2 つのプロシージャに無効のマークが付けられます。

この依存性モデルのデメリットは、必要以上に制限が多いということです。ネットワークを介した依存オブジェクトは、必ずしも必要でないときにも再コンパイルされることが多いため、パフォーマンスが低下します。

さらに、クライアント側アプリケーションが PL/SQL バージョン 2 を使用して作成されている場合、クライアント側では、タイムスタンプ・モデルによりアプケーションの実行が完全にブロックされる状況が生じることがあります。PL/SQL バージョン 1 が、ストアド・プロシージャをサポートしていなかったため、クライアント側で PL/SQL バージョン 1 を使用していた Oracle Forms などの初期のリリースのツールでは、この依存性モデルは使用されていませんでした。

クライアント側の PL/SQL バージョン 2 と統合された Oracle Forms のリリースの場合、タイムスタンプ・モデルで問題が発生する場合があります。たとえば、そのアプリケーションが使用するクライアント側 PL/SQL プロシージャがクライアント・サイトで再コンパイルされないかぎり、アプリケーションはインストール中に無効になります。また、クライアント側のプロシージャがサーバー側のプロシージャに依存しており、そのサーバー側のプロシージャが変更または自動的に再コンパイルされた場合は、クライアント側の PL/SQL プロシージャを再コンパイルする必要があります。ただし、多くのアプリケーション環境（Forms ランタイム・アプリケーションなど）では、クライアントで使用できる PL/SQL コンパイラはありません。このため、アプリケーションはまったく実行できません。このような場合、クライアント・アプリケーションの開発者は、すべての顧客に対してそのアプリケーションの新しいバージョンを再配布する必要があります。

シグネチャのチェック

Oracle Database には、**RPC シグネチャ**を使用するリモート依存性の機能もあります。RPC シグネチャ機能の影響を受けるのは、リモート依存性のみです。ローカル依存性は、その環境でいつでも再コンパイルできるため影響を受けません。

プロシージャの RPC シグネチャには、次の項目に関する情報が含まれます。

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

> **注意：** 重要なのはパラメータの型とモードのみです。パラメータの名前は RPC シグネチャには影響しません。

RPC シグネチャ依存性モデルが有効である場合に、依存単位に親単位のプロシージャへのコールが含まれており、かつ、そのプロシージャの RPC シグネチャ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・ユニットへの依存性により、その依存単位が無効になります。プログラム・ユニットは、パッケージ、ストアド・プロシージャ、ストアド・ファンクションまたはトリガーのいずれかです。

タイムスタンプのみの依存性モデルに伴う問題をできるかぎり解消するため、Oracle Database には、RPC シグネチャを使用するリモート依存性の機能が追加されています。RPC シグネチャ機能の影響を受けるのは、リモート依存性のみです。ローカル（同一サーバー内での）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。

コンパイル済のストアド・プログラム・ユニットには、それぞれに RPC シグネチャが関連付けられています。RPC シグネチャは、次の基準に従って各ユニットを識別しています。

- ユニットの名前（パッケージ、プロシージャまたはファンクションの名前）
- サブプログラムの各パラメータの型
- パラメータのモード（IN、OUT および IN OUT）
- パラメータの個数
- ファンクションの戻り値の型

ユーザーは、リモート依存性の管理に RPC シグネチャとタイムスタンプのどちらを使用するかを制御できます。

関連項目： 6-19 ページの「リモート依存性の制御」

RPC シグネチャ依存性モデルが使用されている場合に、依存単位に親単位のサブプログラムへのコールが含まれており、かつ、そのサブプログラムの RPC シグネチャ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・ユニットへの依存性により、その依存単位が無効になります。

たとえば、ボストンのサーバー（BOSTON_SERVER）に格納されているプロシージャ get_emp_name を考えます。このプロシージャは、次のように定義されています。

```
CREATE OR REPLACE PROCEDURE get_emp_name (
   emp_number   IN  NUMBER,
   hire_date    OUT VARCHAR2,
   emp_name     OUT VARCHAR2) AS
BEGIN
   SELECT ename, to_char(hiredate, 'DD-MON-YY')
      INTO emp_name, hire_date
      FROM emp
      WHERE empno = emp_number;
END;
```

get_emp_name が BOSTON_SERVER でコンパイルされると、RPC シグネチャはタイムスタンプとともに記録されます。

次のように、カリフォルニアにある別のサーバーで、PL/SQL コードにより、BOSTON_SERVER という DB リンクで識別される get_emp_name がコールされるとします。

```
CREATE OR REPLACE PROCEDURE print_ename (emp_number IN NUMBER) AS
   hire_date     VARCHAR2(12);
   ename         VARCHAR2(10);
BEGIN
   get_emp_name@BOSTON_SERVER(emp_number, hire_date, ename);
   dbms_output.put_line(ename);
   dbms_output.put_line(hire_date);
END;
```

このカリフォルニアのサーバー・コードがコンパイルされるときに、次の処理が実行されます。

- ボストンにあるサーバーに接続されます。
- get_emp_name の RPC シグネチャが、カリフォルニアのサーバーに転送されます。
- RPC シグネチャが、print_ename のコンパイルされた状態で記録されます。

実行時には変更の有無にかかわらず、カリフォルニアのサーバーからボストンのサーバーへのリモート・プロシージャのコール中に、print_ename のコンパイルされた状態で保存されている get_emp_name の記録済 RPC シグネチャが、ボストンのサーバーまで送信されます。

タイムスタンプ依存性モードが有効な場合、タイムスタンプに不一致があると、コール側プロシージャにエラー・ステータスが戻されます。

ただし、RPC シグネチャ・モードが有効な場合は、タイムスタンプの不一致は無視され、カリフォルニアのサーバーの print_ename のコンパイル済状態にある get_emp_name の記録済 RPC シグネチャが、ボストンのサーバーにある get_emp_name の現在の RPC シグネチャと比較されます。この 2 つのシグネチャが一致した場合、コールは正常に完了します。一致しない場合は、エラー・ステータスが print_name プロシージャに戻されます。

ボストンのサーバー上にある get_emp_name プロシージャが変更されているか、または新しいリリースのサーバーがインストールされたなどの理由で、そのタイムスタンプがカリフォルニアのサーバー上にある print_name プロシージャに記録されているタイムスタンプと異なる可能性があります。カリフォルニアのサーバーで RPC シグネチャ・リモート依存性モードが有効になっているかぎり、タイムスタンプに不一致があっても、get_emp_name のコール時にエラーが発生することはありません。

注意： DETERMINISTIC、PARALLEL_ENABLE および純粋度情報は、RPC シグネチャ・モードでは表示されません。リモート・システム上のファンクションが別の設定で再定義された場合、これらの設定に基づく最適化は自動的には再考慮されません。そのため、SQL 文でリモート・ファンクションへのコールが（間接的にでも）発生した場合、またはファンクション索引でリモート・ファンクションが（間接的にでも）使用された場合、問合せ結果は正しくないことがあります。

この項の内容は、次のとおりです。

- データ型クラスの切替え
- プロシージャ・シグネチャの変更例

データ型クラスの切替え

RPC シグネチャは、あるクラスのデータ型を別のクラスに切り替えた場合に変更されます。1 つのデータ型クラスには、複数のデータ型が含まれる可能性があります。同じクラスの中でパラメータのデータ型を別のデータ型に変更した場合は、RPC シグネチャは変更されません。

表 6-3 は、データ型クラスとそこに含まれるデータ型をリストにまとめたものです。NCHAR や TIMESTAMP など、表 6-3 にリストされていないデータ型は、いずれのクラスの一部でもありません。したがって、それらのデータ型を変更した場合は必ず、RPC シグネチャに不一致が発生します。

表 6-3 データ型クラス

データ型クラス	クラス内のデータ型
Character	CHAR CHARACTER
VARCHAR	VARCHAR VARCHAR2 STRING LONG ROWID
RAW	RAW LONG RAW
Integer	BINARY_INTEGER PLS_INTEGER SIMPLE_INTEGER BOOLEAN NATURAL NATURALN POSITIVE POSITIVEN
Number	NUMBER INT INTEGER SMALLINT DEC DECIMAL REAL FLOAT NUMERIC DOUBLE PRECISION
Date	DATE TIMESTAMP TIMESTAMP WITH TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE INTERVAL YEAR TO MONTH INTERVAL DAY TO SECOND

モード デフォルトのパラメータ・モード IN の明示的指定に変更があっても、サブプログラムの RPC シグネチャは変更されません。たとえば、次の 2 つを入れ換えます。

```
PROCEDURE P1 (Param1 NUMBER);
PROCEDURE P1 (Param1 IN NUMBER);
```

入れ換えても、RPC シグネチャは変更されません。これ以外のパラメータ・モードの変更が行われた場合、RPC シグネチャは変更されます。

デフォルトのパラメータ値 デフォルトのパラメータ値の指定を変更しても、RPC シグネチャは変更されません。たとえば、P1 は、次の 2 つの例では同じ RPC シグネチャを持っています。

```
PROCEDURE P1 (Param1 IN NUMBER := 100);
PROCEDURE P1 (Param1 IN NUMBER := 200);
```

コール側で新しいデフォルト値を取得できるようにする必要がある場合、アプリケーション開発者はコールされたプロシージャを再コンパイルする必要があります。ただし、デフォルトのパラメータ値の割当てが変更されても、RPC シグネチャに基づいて無効化されることはありません。

プロシージャ・シグネチャの変更例

この章ですでに説明した Get_emp_names プロシージャの本体を、次のように変更するとします。

```
DECLARE
   Emp_number  NUMBER;
   Hire_date   DATE;
BEGIN
-- date format model changes

   SELECT Ename, To_char(Hiredate, 'DD/MON/YYYY')
      INTO Emp_name, Hire_date
      FROM Emp_tab
      WHERE Empno = Emp_number;
END;
```

プロシージャ仕様部は変更されていないため、その RPC シグネチャも変更されていません。

ただし、プロシージャ仕様部が次のように変更されたとします。

```
CREATE OR REPLACE PROCEDURE Get_emp_name (
   Emp_number  IN  NUMBER,
   Hire_date   OUT DATE,
   Emp_name    OUT VARCHAR2) AS
```

この場合、パラメータ Hire_date が別のデータ型になっているため、それに応じて本体が変更された場合、RPC シグネチャは変更されます。

ただし、そのパラメータの名前が When_hired に変更されても、データ型は VARCHAR2、モードは OUT のままの場合、RPC シグネチャは変更されません。仮パラメータの名前が変更されても、そのユニットの RPC シグネチャは変更されません。

次の例を考えてみます。

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_type IS RECORD (
        Emp_number NUMBER,
        Hire_date  VARCHAR2(12),
        Emp_name   VARCHAR2(10));
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type);
END;

CREATE OR REPLACE PACKAGE BODY Emp_package AS
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type) IS
   BEGIN
       SELECT Empno, Ename, TO_CHAR(Hiredate, 'DD/MON/YY')
           INTO Emp_data
           FROM Emp_tab
           WHERE Empno = Emp_data.Emp_number;
   END;
END;
```

レコードのフィールド名が変更されるようにパッケージ仕様部が変更された場合、型がそのままである場合は、RPC シグネチャには影響しません。たとえば、次のパッケージ仕様部は、前述したパッケージ仕様部の例と同じ RPC シグネチャを持っています。

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_type IS RECORD (
        Emp_num    NUMBER,         -- was Emp_number
        Hire_dat   VARCHAR2(12),   -- was Hire_date
        Empname    VARCHAR2(10));  -- was Emp_name
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type);
END;
```

型が以前のままである場合は、パラメータ型の名前を変更しても RPC シグネチャは変更されません。たとえば、次のような Emp_package のパッケージ仕様部は、最初のものと同じです。

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_record_type IS RECORD (
        Emp_number NUMBER,
        Hire_date  VARCHAR2(12),
        Emp_name   VARCHAR2(10));
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_record_type);
END;
```

リモート依存性の制御

動的初期化パラメータ REMOTE_DEPENDENCIES_MODE は、タイムスタンプと RPC シグネチャのどちらで依存性モデルを有効にするかを制御します。

- 初期化パラメータ・ファイルに次のような指定がある場合 :

  ```
  REMOTE_DEPENDENCIES_MODE = TIMESTAMP
  ```

この場合は、タイムスタンプのみを使用して依存性が解決されます（明示的な上書きが動的に行われない場合）。

- 初期化パラメータ・ファイルに次のような指定がある場合 :

  ```
  REMOTE_DEPENDENCIES_MODE = SIGNATURE
  ```

この場合は、RPC シグネチャを使用して依存性が解決されます（明示的な上書きが動的に行われない場合）。

- モードは、DDL 文を使用することによって動的に変更できます。たとえば、次の例では、カレント・セッションの依存性モデルが変更されます。

  ```
  ALTER SESSION SET REMOTE_DEPENDENCIES_MODE =
      {SIGNATURE | TIMESTAMP}
  Thise example alters the dependency model systemwide after startup:
  ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE =
      {SIGNATURE | TIMESTAMP}
  ```

REMOTE_DEPENDENCIES_MODE パラメータが init.ora パラメータ・ファイル内で指定されていないか、DDL 文 ALTER SESSION または ALTER SYSTEM を使用してパラメータが指定されていない場合は、TIMESTAMP がデフォルト値となります。そのため、明示的に REMOTE_DEPENDENCIES_MODE パラメータまたは適切な DDL 文を使用しないかぎり、サーバーはタイムスタンプ依存性モデルを使用して動作します。

REMOTE_DEPENDENCIES_MODE=SIGNATURE を使用する場合：

- リモート・プロシージャのパラメータのデフォルト値を変更しても、リモート・プロシージャをコールするローカル・プロシージャは無効にされません。リモート・プロシージャへのコールでパラメータが指定されない場合は、デフォルト値が使用されます。この場合、無効化 / 再コンパイルは自動的には行われないため、以前のデフォルト値が使用されます。新しいデフォルト値を使用するには、コール側のプロシージャを手動で再コンパイルする必要があります。
- オーバーロードされた新しいプロシージャ（既存プロシージャと同名の新規プロシージャ）をパッケージへ追加しても、リモート・プロシージャをコールするローカル・プロシージャは無効にされません。このオーバーロードの結果として、ローカル・プロシージャからの既存のコールがタイムスタンプ・モードで再バインドされた場合、ローカル・プロシージャは無効にされないため、RPC シグネチャ・モードでは、この再バインドは行われません。新しい再バインドを成功させるには、ローカル・プロシージャを手動で再コンパイルする必要があります。
- 新しい型と以前の型が同じになるように既存のパッケージ・プロシージャのパラメータの型が変更された場合でも、ローカルのコール側プロシージャは自動的に無効化または再コンパイルされません。新しい型のセマンティクスを取得するには、コール側プロシージャを手動で再コンパイルする必要があります。

この項の内容は、次のとおりです。

依存性の解決

REMOTE_DEPENDENCIES_MODE = TIMESTAMP（デフォルト値）の場合、プログラム・ユニット間の依存性は、実行時にタイムスタンプを比較して処理されます。コールされたリモート・プロシージャのタイムスタンプが、コールされたプロシージャのタイムスタンプが一致しない場合、コール側の（依存）ユニットは無効になり、再コンパイルする必要があります。この場合、ローカル PL/SQL コンパイラがない場合は、コール側のアプリケーションを処理できません。

タイムスタンプ依存性モードでは、RPC シグネチャは比較されません。ローカル PL/SQL コンパイラがある場合は、コール側プロシージャが実行されると自動的に再コンパイルされます。

REMOTE_DEPENDENCIES_MODE = SIGNATURE の場合、コール側のユニットの記録済タイムスタンプは、まず最初にコールされたリモート・ユニット内の現在のタイムスタンプと比較されます。2 つのタイムスタンプが一致した場合、コールが進行します。タイムスタンプが一致しない場合は、コールされたリモート・サブプログラムの RPC シグネチャは、コール側サブプログラムに記録されたままの状態で、コールされたサブプログラムの現在の RPC シグネチャと比較されます。この 2 つのシグネチャが一致しない場合は、6-17 ページの「データ型クラスの切替え」に記載されている基準を使用した結果、コール側セッションにエラーが戻されます。

依存性を管理するための提案

次のガイドラインに従って、REMOTE_DEPENDENCIES_MODE パラメータを設定してください。

- サーバー側の PL/SQL ユーザーは、パラメータを TIMESTAMP に設定して（またはこれをデフォルト値にして）、タイムスタンプ依存性モードにできます。
- サーバー側の PL/SQL ユーザーは、分散システムがある場合、不要な再コンパイルを回避するために、RPC シグネチャ依存性モードの使用を選択できます。
- クライアント側の PL/SQL ユーザーは、パラメータを SIGNATURE に設定する必要があります。この設定をすると、次のことができます。
  - プロシージャを再コンパイルしなくても、クライアント・サイトで新しいアプリケーションをインストールできます。
  - タイムスタンプの不一致を発生させずに、サーバーをアップグレードできます。
- サーバー側で RPC シグネチャ・モードを使用するときは、パッケージ仕様部のプロシージャ（またはファンクション）宣言の最後に新しいプロシージャを追加します。新しいプロシージャを宣言リストの中間に追加すると、依存プロシージャに対して不必要に無効化され再コンパイルされる可能性があります。

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 Database 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 Database 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。

7

データ・ディクショナリ

この章では、**データ・ディクショナリ**と呼ばれる、各 Oracle データベースの読取り専用の参照表とビューの中核的なセットについて説明します。

この章の内容は、次のとおりです。

データ・ディクショナリの概要

データ・ディクショナリは Oracle データベースで最も重要な部分の 1 つであり、データベースに関する情報を提供する**読取り専用**の表の集合です。データ・ディクショナリには次のものが含まれます。

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

データ・ディクショナリは、他のデータベース・データと同様に、表とビューによって構成されています。特定のデータベースのデータ・ディクショナリ表とビューは、すべてそのデータベースの SYSTEM 表領域に格納されます。

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

> **関連項目：** SYSTEM 表領域の詳細は、3-6 ページの「大型ファイル表領域」を参照してください。

この項の内容は、次のとおりです。

データ・ディクショナリの構造

データ・ディクショナリは、次の要素で構成されています。

実表 : 対応するデータベースについての情報を格納する基礎になる表。これらの表に読取り / 書込みできるのは Oracle Database のみです。これらの表は正規化され、データのほとんどは、暗号形式で格納されているため、ユーザーが直接アクセスすることはほとんどありません。

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

SYS（データ・ディクショナリの所有者）

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

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

データ・ディクショナリの使用方法

データ・ディクショナリの主な使用方法は次の 3 つです。

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

この項の内容は、次のとおりです。

Oracle Database でのデータ・ディクショナリの使用方法

データ・ディクショナリの実表内のデータは、Oracle Database を機能させるために必要です。したがって、Oracle Database のみがデータ・ディクショナリ情報の書込みや変更を行います。データベースをアップグレードまたはダウングレードしたとき、Oracle Database は、データ・ディクショナリ表を変更するスクリプトを提供します。

注意： データ・ディクショナリ表内のデータの変更や削除はユーザーに実行させないでください。

データベースの操作時に、Oracle Database はデータ・ディクショナリを読み取って、スキーマ・オブジェクトが存在すること、およびユーザーのアクセス権が適切に付与されていることを確認します。また、Oracle Database は、データベース構造、監査、権限付与およびデータの変更を反映するために、継続的にデータ・ディクショナリを更新します。

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

この項の内容は、次のとおりです。

データ・ディクショナリ・ビューのパブリック・シノニム

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

高速アクセスのためのデータ・ディクショナリのキャッシュ

Oracle Database は、データベース操作中に絶えずデータ・ディクショナリにアクセスして、ユーザー・アクセスの妥当性チェックやスキーマ・オブジェクトの状態を検証します。そのため、データ・ディクショナリ情報の大部分は SGA の**ディクショナリ・キャッシュ**に格納されます。すべての情報は、最低使用頻度（LRU）アルゴリズムを使用してメモリーに格納されます。

通常、キャッシュに保持されるのは、解析情報です。表とそれらの列について記述している `COMMENTS` 列は、頻繁にアクセスされないかぎりキャッシュには保持されません。

他のプログラムとデータ・ディクショナリ

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

データ・ディクショナリの使用方法

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

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

データ・ディクショナリは、ビューのセットによって構成されています。多くの場合、表 7-1 で示すように、そのセットは、類似した情報が格納されている 3 つのビューで構成され、それぞれが接頭辞によって区別されます。

表 7-1 データ・ディクショナリ・ビューの接頭辞

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

列の集合は、次の例外を除き、ビュー全体で同一です。

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

> **関連項目：** データ・ディクショナリ・ビューとその列の完全なリストは、『Oracle Database リファレンス』を参照してください。

この項の内容は、次のとおりです。

接頭辞が USER のビュー

通常のデータベース・ユーザーが最も頻繁に使用するのは、接頭辞が USER のビューです。これらのビューには、次のような特長があります。

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

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

```
SELECT object_name, object_type FROM USER_OBJECTS;
```

接頭辞が ALL のビュー

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

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

接頭辞が DBA のビュー

接頭辞が DBA のビューには、データベース全体のグローバル・ビューが示されます。DBA ビューへの問合せを発行するのは管理者のみであるため、これらのビューのシノニムは作成されません。このため、DBA ビューに問合せを発行するには、管理者は、次のようにして所有者 SYS をビューの名前の接頭辞として指定する必要があります。

```
SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;
```

ANY システム権限を持つユーザーがその権限をデータ・ディクショナリに対して使用しないように、データ・ディクショナリ保護を実装することをお薦めします。データ保護を使用可能に（O7_DICTIONARY_ACCESSIBILITY を false に設定）すると、SYS スキーマ内のオブジェクト（ディクショナリ・オブジェクト）へのアクセスは、SYS スキーマを持つユーザーに限定されます。これらのユーザーは SYS であり、SYSDBA として接続するユーザーです。

> **関連項目：** システム権限の制限の詳細は、『Oracle Database 管理者ガイド』を参照してください。

DUAL 表

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

> **関連項目：** DUAL 表の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

動的パフォーマンス表

Oracle Database は、操作中ずっと、カレント・データベース・アクティビティを記録する一連の仮想表を保持しています。これらの表のことを**動的パフォーマンス表**と呼びます。

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

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

> **関連項目：** 動的パフォーマンス・ビューのシノニムおよび列の完全なリストは、『Oracle Database リファレンス』を参照してください。

データベース・オブジェクト・メタデータ

DBMS_METADATA パッケージには、データベース・オブジェクトの完全な定義を抽出するためのインタフェースが用意されています。これらの定義は、XML または DDL 文のいずれかで表現されます。2 つのインタフェース・スタイルには次の特長があります。

- 柔軟で洗練された、プログラム制御用のインタフェース
- 単純化された、非定型問合せ用のインタフェース

> **関連項目：** DBMS_METADATA の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

8

メモリー・アーキテクチャ

この章では、Oracle Database インスタンスのメモリー・アーキテクチャについて説明します。この章の内容は、次のとおりです。

- Oracle Database メモリー構造の概要
- システム・グローバル領域の概要
- プログラム・グローバル領域の概要
- メモリー管理方法の概要
- ソフトウェア・コード領域

> **関連項目：** メモリーの構成方法および管理方法は、『Oracle Database 管理者ガイド』を参照してください。

Oracle Database メモリー構造の概要

Oracle Database では、メモリーを使用して次のような情報を格納します。

- プログラム・コード
- 現在はアクティブかどうかを問わず、接続されているセッションについての情報
- プログラムの実行時に必要な情報（行をフェッチしている問合せの現在の状態など）
- Oracle Database プロセス間で共有され、やりとりされる情報（ロック情報など）
- ストレージ・デバイスに永続的に保存されるキャッシュ・データ（データ・ブロックおよび REDO ログ・エントリなど）

基本的なメモリー構造

Oracle Database に関連する基本的なメモリー構造は、次のような領域で構成されています。

- ソフトウェア・コード領域

  ソフトウェア・コード領域とは、実行中または実行される可能性があるコードを格納するためのメモリー部分です。Oracle Database のコードはソフトウェア領域に格納されます。この場所は通常ユーザー・プログラムの格納場所とは異なり、排他的であるかまたは保護されています。

- システム・グローバル領域（SGA）

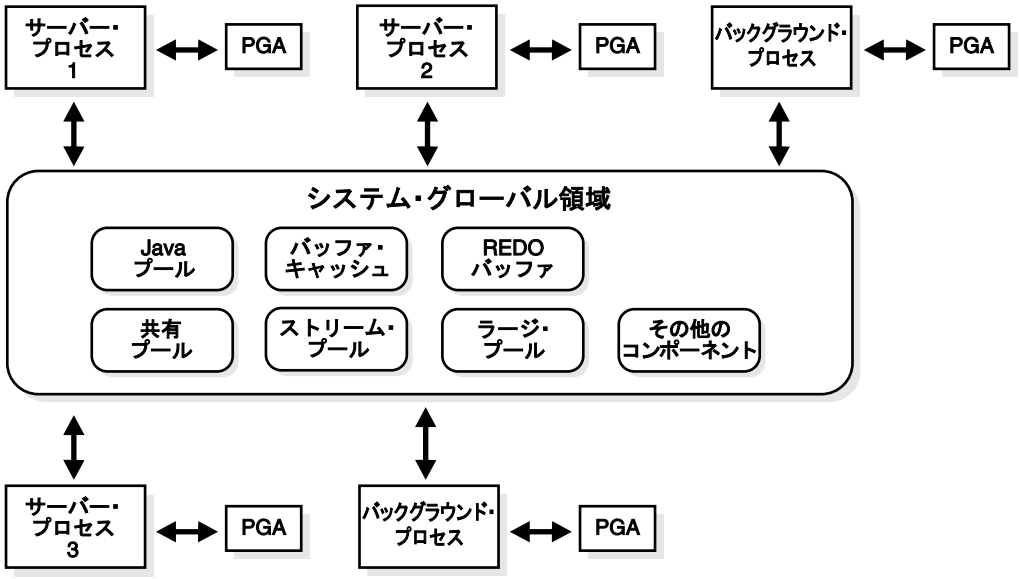
  SGA は、SGA コンポーネントと呼ばれる共有メモリー構造のグループで、1 つの Oracle Database インスタンスに関するデータと制御情報が保存されています。SGA は、すべてのサーバーおよびバックグラウンド・プロセスで共有されます。SGA に保存されるデータの例としては、キャッシュ・データ・ブロックや共有 SQL 領域があります。

- プログラム・グローバル領域（PGA）

  PGA は、サーバー・プロセス用のデータや制御情報を含むメモリー領域です。PGA は、サーバー・プロセスの起動時に、Oracle Database によって作成される非共有メモリーです。PGA にアクセスできるのはサーバー・プロセスのみです。各サーバー・プロセスには PGA が 1 つあります。バックグラウンド・プロセスにも独自の PGA が割り当てられます。すべての個々の PGA で使用されるメモリーの合計は、**合計インスタンス PGA メモリー**と呼ばれ、個々の PGA の集合は、**合計インスタンス PGA**、または単に**インスタンス PGA** と呼ばれます。個々の PGA ではなく、データベース初期化パラメータを使用してインスタンス PGA のサイズを設定します。

図 8-1 にこれらのメモリー構造の関係を示します。

図 8-1 Oracle Database のメモリー構造



関連項目：

- 8-3 ページの「システム・グローバル領域の概要」
- 8-10 ページの「プログラム・グローバル領域の概要」
- 8-15 ページの「ソフトウェア・コード領域」
- 9-4 ページの「Oracle Database プロセスの概要」

システム・グローバル領域の概要

Oracle Database インスタンスは、システム・グローバル領域（SGA）および一連のデータベース・プロセスにより構成されます。インスタンスを起動すると、Oracle Database が自動的に SGA 用のメモリーを割り当て、インスタンスを停止すると、オペレーティング・システムがそのメモリーの割当てを解除します。各インスタンスには、それぞれ専用の SGA があります。

SGA は読取り / 書込み可能です。ユーザーにかわって実行するすべてのデータベース・バックグラウンド・プロセスおよびサーバー・プロセスにより、インスタンスの SGA に含まれている情報が読み取られ、データベース操作中に一部のサーバー・プロセスにより SGA への書込みが実行されます。

SGA の一部には、バックグラウンド・プロセスがアクセスする必要のある、データベースとインスタンスの状態に関する一般的な情報が含まれています。この部分を**固定 SGA** と呼びます。ここには、ユーザー・データは格納されません。SGA には、ロック情報などのプロセス間でやりとりされる情報も格納されます。

システムが共有サーバー・アーキテクチャを使用している場合、要求キューとレスポンス・キュー、および PGA 領域の内容の一部は SGA に格納されます。

8-3 ページの図 8-1 で示すように、SGA は、いくつかのメモリー・**コンポーネント**で構成されています。このコンポーネントは、特定のクラスのメモリー割当て要求を満たすために使用するメモリーのプールです。

次に、最も重要な SGA コンポーネントを示します。

- データベース・バッファ・キャッシュ
- REDO ログ・バッファ
- 共有プール
- ラージ・プール
- Java プール
- ストリーム・プール

関連項目：

- Oracle Database インスタンスの詳細は、12-2 ページの「Oracle インスタンスの概要」を参照してください。
- 8-10 ページの「プログラム・グローバル領域の概要」
- 9-15 ページの「ディスパッチャの要求キューとレスポンス・キュー」

データベース・バッファ・キャッシュ

データベース・バッファ・キャッシュは SGA の一部であり、データファイルから読み取ったデータ・ブロックのコピーが保持される領域です。インスタンスに同時接続されたユーザーはすべて、データベース・バッファ・キャッシュへのアクセスを共有します。

この項の内容は、次のとおりです。

- データベース・バッファ・キャッシュの編成
- LRU アルゴリズムと全表スキャン

データベース・バッファ・キャッシュの編成

このキャッシュのバッファは、書込みリストおよび最低使用頻度リスト（LRU）の 2 つのリストで編成されます。**書込みリスト**には使用済バッファが保持されます。使用済バッファとは、修正されたが、まだディスクに書き込まれていないデータを含むバッファです。**LRU** リストは、使用可能バッファ、使用中バッファおよび書込みリストに移動していない使用済バッファを保持します。**使用可能バッファ**は、有効なデータが含まれておらず、使用可能なバッファです。**使用中バッファ**は、現在アクセスされているバッファです。

Oracle Database プロセスからバッファにアクセスするとき、そのバッファは LRU リストの最高使用頻度（MRU）側に移動します。さらに多くのバッファが LRU リストの MRU 側へ移動されるにつれて、使用済バッファは LRU リストの LRU 側に向かって古くなります。

Oracle Database ユーザー・プロセスでデータの特定の部分が初めて必要になると、データベース・バッファ・キャッシュ内のデータを検索します。キャッシュ内にデータが見つかった場合（**キャッシュ・ヒット**）、プロセスはデータをメモリーから直接読み取ることができます。キャッシュ内にデータが見つからなかった場合（**キャッシュ・ミス**）、プロセスはデータにアクセスする前に、ディスク上のデータファイルからキャッシュ内のバッファにデータ・ブロックをコピーする必要があります。キャッシュ・ヒットによるデータのアクセスは、キャッシュ・ミスによるデータのアクセスよりも高速です。

プロセスはキャッシュ内にデータ・ブロックを読み取る前に、使用可能バッファを見つける必要があります。プロセスは、LRU リストの LRU 側から検索を開始します。使用可能バッファが見つかるか、検索したバッファ数が制限しきい値に達するまで、プロセスは検索を続けます。

ユーザー・プロセスが LRU リストの検索時に使用済バッファを見つけた場合、プロセスはそのバッファを書込みリストに移動してから検索を続けます。使用可能バッファが見つかると、プロセスはディスクからバッファにデータ・ブロックを読み取って、バッファを LRU リストの MRU 側に移動します。

使用可能バッファが見つからず、検索したバッファ数が制限しきい値に達すると、Oracle Database ユーザー・プロセスは LRU リストの検索を停止し、使用済バッファの一部をディスクに書き込むように、DBW0 バックグラウンド・プロセスに信号を送ります。

関連項目： DBW*n* プロセスの詳細は、9-7 ページの「データベース・ライター・プロセス（DBWn）」を参照してください。

LRU アルゴリズムと全表スキャン

ユーザー・プロセスは、全表スキャンを実行するときに、表のブロックをバッファに読み取って LRU リストの（MRU 側ではなく）LRU 側に入れます。通常、全表スキャンされた表は一時的に必要なため、使用頻度の高いブロックをキャッシュ内に残しておくために、全表スキャンされた表のブロックをすぐにキャッシュから移動する必要があるためです。

表スキャンに関連するブロックのこのデフォルトの動作は、表ごとに制御できます。全表スキャン時に表のブロックをリストの MRU 側に格納するように指定するには、表やクラスタの作成時または変更時に CACHE 句を使用します。それ以後の表へのアクセスで I/O を防止するために、小さな参照表や大きな静的履歴表にこの動作を指定することがあります。

関連項目： CACHE 句の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

REDO ログ・バッファ

REDO ログ・バッファとは、SGA 内の循環バッファであり、データベースに加えられた変更についての情報を保持します。この情報は、REDO エントリに格納されます。**REDO エントリ**には、INSERT、UPDATE、DELETE、CREATE、ALTER または DROP の各操作によってデータベースに加えられた変更の再構築または再実行に必要な情報が含まれます。REDO エントリは、必要に応じてデータベースのリカバリ時に使用されます。

REDO エントリは、Oracle Database プロセスによってユーザーのメモリー領域から SGA 内の REDO ログ・バッファにコピーされます。REDO エントリは、バッファ内で連続した領域を占めます。バックグラウンド・プロセス LGWR は、REDO ログ・バッファをディスク上のアクティブな REDO ログ・ファイル（または REDO ログ・ファイルのグループ）に書き込みます。

関連項目：

- REDO ログ・バッファがディスクに書き込まれる方法の詳細は、9-9 ページの「ログ・ライター・プロセス（LGWR）」を参照してください。
- REDO ログ・ファイルとグループの詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

共有プール

SGA の共有プール部分には、ライブラリ・キャッシュ、ディクショナリ・キャッシュ、結果キャッシュ、パラレル実行メッセージ用バッファおよび制御構造が含まれます。

この項の内容は、次のとおりです。

- ライブラリ・キャッシュ
- ディクショナリ・キャッシュ
- 結果キャッシュ

ライブラリ・キャッシュ

ライブラリ・キャッシュには、共有 SQL 領域、プライベート SQL 領域（共有サーバー構成の場合）、PL/SQL プロシージャおよびパッケージの他、ロックやライブラリ・キャッシュ・ハンドルなどの制御構造が含まれます。

共有 SQL 領域にはすべてのユーザーがアクセス可能であるため、ライブラリ・キャッシュは SGA 内の共有プールに含まれます。

共有 SQL 領域とプライベート SQL 領域 Oracle Database では、実行される各 SQL 文が、共有 SQL 領域とプライベート SQL 領域によって表されます。Oracle Database は、2 人のユーザーが同じ SQL 文を実行する場合にそれを認識し、これらのユーザーのために共有 SQL 領域を再利用します。ただし、各ユーザーは、その文のプライベート SQL 領域のコピーを各自で持つ必要があります。

共有 SQL 領域には、特定の SQL 文の解析ツリーと実行計画が含まれます。Oracle Database では、多数のユーザーが同じアプリケーションを実行するときには特に、複数回実行される SQL 文に 1 つの共有 SQL 領域を使用することによってメモリーを節約します。

Oracle Database では、新しい SQL 文が解析された時点でメモリーが共有プールから割り当てられ、共有 SQL 領域に格納されます。このメモリーのサイズは、文の複雑度に応じて決められます。共有プール全体が割当て済の場合、Oracle Database では、新しい文の共有 SQL 領域のための空き領域が十分になるまで、修正した LRU アルゴリズムを使用してその共有プールからエントリの割当てを解除できます。Oracle Database が共有 SQL 領域の割当て解除する場合、対応付けられていた SQL 文は次の実行時に再解析され、別の共有 SQL 領域に再び割り当てられます。

関連項目：

- 8-11 ページの「プライベート SQL 領域」
- 『Oracle Database パフォーマンス・チューニング・ガイド』

PL/SQL プログラム・ユニットと共有プール Oracle Database では、PL/SQL プログラム・ユニット（プロシージャ、ファンクション、パッケージ、無名ブロックおよびデータベース・トリガー）が、個々の SQL 文と同じ方法で処理されます。また、プログラム・ユニットを解析およびコンパイル済の形で保持するために、Oracle Database により共有領域が割り当てられます。さらに、ローカル変数、グローバル変数、パッケージ変数（パッケージ・インスタンス化とも呼ばれる）および SQL 実行用のバッファなど、プログラム・ユニットを実行するセッションに固有な値を保持するために、Oracle Database によりプライベート領域が割り当てられます。複数のユーザーが同じプログラム・ユニットを実行している場合、単一の共有領域はすべてのユーザーによって使用されますが、各ユーザーは、自分のセッションに固有の値を保持するプライベート SQL 領域のコピーを個別にメンテナンスします。

PL/SQL プログラム・ユニット内に含まれる個々の SQL 文は、先の項で説明したように処理されます。PL/SQL プログラム・ユニット内の起点には関係なく、これらの SQL 文は解析された表現を保持するために共有領域を使用し、その文を実行するセッションごとにプライベート SQL 領域を使用します。

共有プール内のメモリーの割当てと再利用 一般に、共有プール内のあらゆる項目（共有 SQL 領域やディクショナリ行）は、修正 LRU アルゴリズムに基づいてフラッシュされるまでは、そのまま存在します。新しい項目に領域を割り当てるために共有プール内にさらに領域が必要になると、定期的には使用されていない項目のメモリーが解放されます。修正 LRU アルゴリズムを使用すると、多数のセッションが使用している共有プール項目は、その項目を作成したプロセスが終了しても、その項目が使用されているかぎりメモリーに残ります。その結果、マルチユーザー Oracle Database システムに関連する SQL 文のオーバーヘッドと処理が、最小限に抑えられます。

SQL 文が実行のために Oracle Database に送られると、Oracle Database では次のメモリー割当ての手順が自動的に実行されます。

1. Oracle Database は、同一の文について共有 SQL 領域がすでに存在しているかどうかを確認するため、共有プールをチェックします。その文のための共有 SQL 領域がすでに存在する場合、その領域は、その文の後続の新しいインスタンスを実行するために使用されます。一方、その文のための共有 SQL 領域が存在しない場合は、Oracle Database により新しい共有 SQL 領域が共有プール内に割り当てられます。どちらの場合も、ユーザーのプライベート SQL 領域が、その文を含む共有 SQL 領域に対応付けられます。

   注意： 共有 SQL 領域がオープンしているカーソルに対応していても、しばらく使用されていないカーソルであれば、その共有 SQL 領域を共有プールからフラッシュできます。オープンしているカーソルが、その後その文を実行するために使用される場合、その文は Oracle Database により再解析されて新しい共有 SQL 領域が共有プール内に割り当てられます。

2. Oracle Database はセッションのためにプライベート SQL 領域を割り当てます。プライベート SQL 領域の位置は、セッションのために確立される接続のタイプによって異なります。

Oracle Database では、次のような場合にも共有プールから共有 SQL 領域がフラッシュされます。

- 表、クラスタまたは索引の統計を更新または削除するために `ANALYZE` 文を使用する場合、分析されたスキーマ・オブジェクトを参照する文を含んでいるすべての共有 SQL 領域は、共有プールからフラッシュされます。フラッシュされた文を次に実行するときには、その文はスキーマ・オブジェクトの新しい統計を反映するように新しい共有 SQL 領域で解析されます。
- スキーマ・オブジェクトが SQL 文で参照され、なんらかの方法で修正されると、共有 SQL 領域は**無効**になり（無効のマークが付けられます）、その文を次に実行するときには解析する必要があります。
- データベースのグローバル・データベース名を変更すると、すべての情報が共有プールからフラッシュされます。
- 管理者はインスタンス起動後のパフォーマンス（データ・バッファ・キャッシュではなく、共有プールに関するパフォーマンス）を査定するために、共有プール内のすべての情報を手動でフラッシュできます。こうすると、現行インスタンスを停止する必要がありません。文 `ALTER SYSTEM FLUSH SHARED_POOL` は、このフラッシュを実行するのに使用します。

  関連項目：

  - プライベート SQL 領域の位置の詳細は、8-6 ページの「共有 SQL 領域とプライベート SQL 領域」を参照してください。
  - SQL 文の無効化と依存性の問題の詳細は、第 6 章「スキーマ・オブジェクトの依存性」を参照してください。
  - `ALTER SYSTEM FLUSH SHARED_POOL` の使用方法は、『Oracle Database SQL 言語リファレンス』を参照してください。
  - `V$SQL` および `V$SQLAREA` 動的ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ディクショナリ・キャッシュ

データ・ディクショナリとは、データベース、データベースの構造およびそのユーザーについての参照情報を含むデータベースの表およびビューの集合のことです。Oracle Database は、SQL 文の解析時にデータ・ディクショナリに頻繁にアクセスします。このアクセスは、Oracle Database の操作を続けるためには不可欠です。

データ・ディクショナリは Oracle Database によって頻繁にアクセスされるので、ディクショナリ・データを保持するために、メモリー内に 2 箇所の特別な場所が指定されています。一方の領域は、データのブロック全体を保持するバッファのかわりに行としてデータを保持するため、**データ・ディクショナリ・キャッシュ**または**行キャッシュ**と呼ばれます。ディクショナリ・データを保持するメモリー内の他方の領域は、ライブラリ・キャッシュです。これら 2 つのキャッシュは、データ・ディクショナリ情報にアクセスするすべての Oracle Database ユーザー・プロセスによって共有されます。

関連項目： 第 7 章「データ・ディクショナリ」

結果キャッシュ

結果キャッシュは、同一のインフラストラクチャを共有している SQL 問合せ結果キャッシュと PL/SQL ファンクション結果キャッシュで構成されています。

DBMS_RESULT_CACHE パッケージには、たとえばキャッシュされた結果をすべてフラッシュしたり、結果のキャッシングの有効化 / 無効化をシステム全体で切り替えたりするための管理サブプログラムが用意されています。動的パフォーマンス・ビュー V$RESULT_CACHE_* を使用すると、開発者や DBA は、特定の SQL 問合せまたは PL/SQL ファンクションにキャッシュ・ヒットしたかなどを判断できます。

クライアント結果キャッシュでは、キャッシングがクライアント側で行われるという点を除けば、結果キャッシュと同じように結果のキャッシュが行われます。

関連項目：

- 結果キャッシュのサイズ設定の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- DBMS_RESULT_CACHE パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 動的パフォーマンス・ビュー（$V）の詳細は、『Oracle Database リファレンス』を参照してください。
- クライアント結果キャッシュの詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

SQL 問合せ結果キャッシュ

問合せの結果および問合せ断片を、SQL 問合せ結果キャッシュのメモリーにキャッシュできます。その後、データベースはキャッシュされた結果を使用して、これらの後続の問合せおよび問合せ断片の実行に対応します。問合せを再実行するより SQL 問合せ結果キャッシュから結果を取得する方が速いため、頻繁に実行される問合せパフォーマンスは、結果がキャッシュされている場合には大幅に向上します。ユーザーは、結果キャッシュ・ヒントを使用して問合せまたは問合せ断片に注釈を付け、結果を SQL 問合せ結果キャッシュに保存することを指定できます。

RESULT_CACHE_MODE 初期化パラメータの設定内容により、SQL 問合せ結果キャッシュをすべての問合せに使用するか（可能な場合）、注釈付きの問合せにのみ使用するかを制御できます。

データまたはキャッシュされた結果の構成に使用されているデータベース・オブジェクトのメタデータがトランザクションで変更されると、キャッシュされた結果はデータベースにより自動的に無効化されます。

関連項目： RESULT_CACHE_MODE 初期化パラメータの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

PL/SQL ファンクション結果キャッシュ

PL/SQL ファンクションは、1 つ以上のパラメータ化問合せを発行し、それを入力とした計算の結果を返すのに使用されることがあります。これらの問合せによりアクセスするデータ（ショッピング・アプリケーションの商品カタログなど）の中には、変更される頻度がファンクションをコールする頻度に比べてはるかに低いものもあります。PL/SQL ファンクションのソース・テキストには、その結果をキャッシュするよう要求する構文や、正確性を確保するため、表のリストに対して DML が実行された場合にはキャッシュをパージすることを要求する構文を含めることができます。また、ファンクションの起動に使用された実引数の組が、キャッシュに対する参照キーになります。結果がキャッシュされたファンクションを起動し、それがキャッシュ・ヒットした場合は、そのファンクション本体は実行されず、かわりにキャッシュに置かれた値が即座に返されます。

関連項目： PL/SQL ファンクション結果キャッシュの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

ラージ・プール

データベース管理者は、次の目的で大量のメモリーを割り当てるために、**ラージ・プール**と呼ばれるオプションのメモリー領域を構成できます。

- 共有サーバーおよび Oracle XA インタフェース用のセッション・メモリー（トランザクションが複数のデータベースと対話する環境で使用）
- I/O サーバー・プロセス
- Oracle Database のバックアップおよびリストア操作

ラージ・プールからセッション・メモリーを共有サーバー、Oracle XA またはパラレル問合せバッファ用に割り当てることにより、Oracle Database では共有プールと主に共有 SQL のキャッシュに使用して、共有 SQL キャッシュの縮小によるパフォーマンスのオーバーヘッドを回避できます。

さらに、Oracle Database のバックアップおよびリストア操作用のメモリー、I/O サーバー・プロセス用のメモリーおよびパラレル・バッファ用のメモリーは、数百 KB のバッファ内で割り当てられます。ラージ・プールのほうが、共有プールよりも適切に、これらの大量のメモリー要求を満たすことができます。

ラージ・プールには、LRU リストはありません。これに対して、共有プール内で確保されている領域では、その共有プールから割り当てられる他のメモリーと同じ LRU リストが使用されます。

関連項目：

- 共有サーバー用にラージ・プールからセッション・メモリーを割り当てる方法の詳細は、9-14 ページの「共有サーバー・アーキテクチャ」を参照してください。
- Oracle XA の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- ラージ・プール、共有プール内の領域確保および I/O サーバー・プロセスの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- パラレル実行用にメモリーを割り当てる方法は、16-11 ページの「パラレル実行の概要」を参照してください。

Java プール

Java プールのメモリーは、サーバー・メモリー内で JVM に含まれるセッション固有の Java コードとデータすべてに使用されます。Java プールのメモリーの使用方法は、Oracle Database の実行モードにより異なります。

Java Pool Advisor 統計は、Java に使用されるライブラリ・キャッシュ・メモリーに関する情報を提供し、Java プールのサイズの変化が解析率に及ぼす影響を予測します。Java Pool Advisor は、`statistics_level` を `TYPICAL` 以上に設定すると内部的にオンに設定されます。これらの統計は、アドバイザをオフにするとリセットされます。

関連項目：『Oracle Database Java 開発者ガイド』

ストリーム・プール

ストリーム・プールは、Oracle Streams によって排他的に使用されます。ストリーム・プールには、バッファ・キュー・メッセージが保存され、Oracle Streams の取得プロセスおよび適用プロセス用のメモリーが用意されています。

明示的に構成しないかぎり、ストリーム・プールのサイズはゼロから開始されます。プール・サイズは、Oracle Streams の使用時に必要に応じて動的に増加します。

関連項目：『Oracle Streams 概要および管理』

プログラム・グローバル領域の概要

Oracle Database により、各サーバー・プロセスにプログラム・グローバル領域（PGA）が割り当てられます。PGA は SQL 文の処理や、ログオンおよびその他のセッション情報の保持に使用されます。メモリー管理の目的では、すべての PGA の集合は**インスタンス PGA** と呼ばれます。初期化パラメータを使用してインスタンス PGA のサイズを設定すると、データベースにより個々の PGA に必要に応じてメモリーが分配されます。

注意： バックグラウンド・プロセスにも独自の PGA が割り当てられます。ここでは、サーバー・プロセス PGA のみを説明します。

この項の内容は、次のとおりです。

- PGA の内容
- 専用サーバー・モードおよび共有サーバー・モードにおける PGA メモリーの使用量

関連項目： セッションの詳細は、9-4 ページの「接続とセッション」を参照してください。

PGA の内容

PGA メモリーの内容は、インスタンスが共有サーバー・オプションを起動しているかどうかにより変わります。一般的に、PGA メモリーは次の領域に分割されます。

- セッション・メモリー
- プライベート SQL 領域

セッション・メモリー

セッション・メモリーとは、セッションの変数（ログイン情報）およびセッションに関係する他の情報を保持するために割り当てられたメモリーです。共有サーバーでは、セッション・メモリーが共有されます（プライベートではありません）。

関連項目：

- セッションの詳細は、9-4 ページの「接続とセッション」を参照してください。
- 『Oracle Database Net Services 管理者ガイド』

プライベート SQL 領域

プライベート SQL 領域には、バインド変数値、問合せ実行状況の情報、および問合せ実行作業領域などのデータが格納されます。SQL 文を発行するそれぞれのセッションには、プライベート SQL 領域があります。同一の SQL 文を発行するそれぞれのユーザーは、1 つの共有 SQL 領域を使用する専用のプライベート SQL 領域を持っています。これにより、プライベート SQL 領域の多くは、同一の共有 SQL 領域に対応付けることができます。

プライベート SQL 領域の位置は、セッションのために確立される接続のタイプによって異なります。セッションが専用サーバーを介して接続されている場合、プライベート SQL 領域はサーバー・プロセスの PGA 内にあります。ただし、セッションが共有サーバーを介して接続されている場合、プライベート SQL 領域の一部は SGA 内に保持されます。

この項の内容は、次のとおりです。

- カーソルと SQL 領域
- プライベート SQL 領域のコンポーネント
- SQL 作業領域

関連項目： 8-6 ページの「共有 SQL 領域とプライベート SQL 領域」

カーソルと SQL 領域 Oracle Database プリコンパイラ・プログラムや OCI プログラムのアプリケーション開発者は、**カーソル**、つまり特定のプライベート SQL 領域へのハンドルを明示的にオープンし、それらのカーソルをプログラムの実行中に名前付きリソースとして使用できます。Oracle Database が一部の SQL 文のために暗黙的に発行する再帰カーソルにも共有 SQL 領域が使用されます。

プライベート SQL 領域は、ユーザー・プロセスが管理します。ユーザー・プロセスが割り当てることのできるプライベート SQL 領域の数は初期化パラメータ `OPEN_CURSORS` によって制限されますが、プライベート SQL 領域の割当ておよび割当て解除は、使用するアプリケーション・ツールに大きく依存します。このパラメータのデフォルト値は 50 です。

プライベート SQL 領域は、対応するカーソルがクローズされるか、文ハンドルが解放されるまで存在します。Oracle Database では、文の実行が完了するとランタイム領域は解放されますが、持続領域は待機状態が維持されます。持続領域を解放し、アプリケーションのユーザーが必要とするメモリー容量を最小限に抑えるには、オープンされているカーソルのうち再利用しないものを、すべてアプリケーション開発者側でクローズします。

関連項目： 24-5 ページの「カーソル」

プライベート SQL 領域のコンポーネント カーソルのプライベート SQL 領域は、存続期間が異なる次の 2 つの領域に分割されます。

- **持続領域 :** この領域にはバインド変数値が格納されます。持続領域は、カーソルのクローズ時にのみ解放されます。
- **ランタイム領域 :** この領域は、Oracle Database における実行要求の最初の処理で作成されます。次の構造が含まれています。
  - 問合せ実行状況の情報

    たとえば、全表スキャンの場合、この領域にはスキャンの進行情報が格納されます。
  - SQL 作業領域

    これらの領域は、ソートやハッシュ結合など、メモリー集中型の操作に必要な場合に割り当てられます。詳細は、この章で後から説明します。

  DML では、文の実行が終了するとランタイム領域が解放されます。問合せの場合は、すべての行がフェッチされた後、または問合せが取り消された後に解放されます。

SQL 作業領域 SQL 作業領域は、次のようなメモリー集中型の操作をサポートするために割り当てられます。

- ソートベースの演算子（ORDER BY、GROUP BY、ロールアップ、ウィンドウ機能）
- ハッシュ結合
- ビットマップ・マージ
- ビットマップ作成

たとえば、ソート演算子は作業領域（ソート領域とも呼ばれる）を使用して、行の集合のメモリー内ソートを実行します。同様に、ハッシュ結合演算子も作業領域（ハッシュ領域とも呼ばれます）使用して、左入力からハッシュ表を構築します。これら 2 つの演算子で処理するデータが作業領域に収まらない場合は、入力データを小さく分割します。これにより、メモリー内でデータの一部を処理できます。残りのデータは、処理待ちとして一時ディスク記憶域に収められます。ビットマップ演算子では、対応する作業領域が非常に小さい場合でもディスクに収まらないことはありませんが、ビットマップ演算の複雑度は使用する作業領域のサイズに反比例します。このため、これらの演算子は、作業領域が大きくなればより高速に実行できます。

作業領域のサイズは、制御およびチューニング可能です。自動 PGA メモリー管理が有効になっている場合、作業領域サイズはデータベースにより自動的にチューニングされます。詳細は、8-13 ページの「メモリー管理方法の概要」を参照してください。

通常、作業領域を増やすとメモリーの使用は増えるものの、演算子の種類によってはパフォーマンスが著しく改善されます。オプションの使用により、入力データや関連の SQL 演算子が割り当てた補助メモリー構造に応じた作業領域のサイズを十分確保できます。このオプションを使用しない場合、一部の入力データが一時ディスク記憶域に収容できなくなるため、応答時間が長くなります。作業領域のサイズが入力データ・サイズに比べ、きわめて小さい場合は、データを分割して複数に分けて渡すことが必要となります。このため、演算子の応答時間が大幅に増加します。

専用サーバー・モードおよび共有サーバー・モードにおける PGA メモリーの使用量

システムが使用するアーキテクチャが専用サーバーなのか、共有サーバーなのかにより、PGA メモリーの割当て方法が変わることがあります。表 8-1 にこれらの相違点を示しています。

表 8-1 専用サーバーと共有サーバーのメモリー割当て方法の相違点

メモリー領域	専用サーバー	共有サーバー
セッション・メモリーの性質	プライベート	共有
持続領域の位置	PGA	SGA
SELECT 文の一部のランタイム領域の位置	PGA	PGA
DML/DDL 文のランタイム領域の位置	PGA	PGA

メモリー管理方法の概要

メモリー管理では、Oracle データベースのインスタンス・メモリー構造に対する最適なサイズが、データベースでの需要に応じて維持されます。管理が必要なメモリーは、システム・グローバル領域（SGA）メモリーおよびインスタンス・プログラム・グローバル領域（インスタンス PGA）メモリーです。インスタンス PGA メモリーとは、すべての個別 PGA に対するメモリー割当ての集合のことです。

Oracle Database では、様々なメモリー管理方法がサポートされており、どの方法を使用するかは初期化パラメータの設定により選択できます。自動メモリー管理方法を有効化することをお薦めします。

SGA およびインスタンス PGA に対する自動メモリー管理

Oracle Database 11*g* から、Oracle Database で SGA メモリーおよびインスタンス PGA メモリーを完全に自動的に管理できるようになりました。ユーザーが指定するのはインスタンスで使用される合計メモリー・サイズのみで、Oracle Database により、処理要求を満たすために必要に応じて SGA とインスタンス PGA の間でメモリーが動的に交換されます。この機能は、自動メモリー管理と呼ばれます。このメモリー管理方法を使用すると、データベースでは、個々の SGA コンポーネントのサイズおよび個々の PGA のサイズが動的にチューニングされます。

SGA に対する自動共有メモリー管理

SGA のサイズをより直接的に制御する場合には、自動メモリー管理を無効化し、自動共有メモリー管理を有効化します。自動共有メモリー管理では、SGA に対するターゲット・サイズおよび最大サイズをユーザーが設定します。データベースは、指定したターゲットに対して SGA の合計サイズをチューニングし、SGA コンポーネントすべてのサイズを動的にチューニングします。

SGA に対する手動共有メモリー管理

個々の SGA コンポーネントのサイズを完全に制御するには、自動メモリー管理および自動共有メモリー管理をともに無効化します。これによって、手動共有メモリー管理が有効化されます。このモードでは、ユーザーが個々の SGA コンポーネントのサイズを設定し、それにより SGA 全体のサイズが決定します。またユーザーは処理ごとに、個々の SGA コンポーネントのチューニングを手動で行います。

インスタンス PGA に対する自動 PGA メモリー管理

自動メモリー管理を無効化した上で、自動共有メモリー管理または手動共有メモリー管理を有効化すると、自動 PGA メモリー管理も暗黙的に有効化されます。自動 PGA メモリー管理では、PGA に対するターゲット・サイズをユーザーが設定します。データベースは、ターゲットに対してインスタンス PGA のサイズをチューニングし、個々の PGA のサイズを動的にチューニングします。自動 PGA メモリー管理は、インスタンス PGA に対するデフォルトの管理方法であるため、ターゲット・サイズを明示的に設定しないかぎり、適切なデフォルト値の計算と構成がデータベースによって自動的に行われます。

インスタンス PGA に対する手動 PGA メモリー管理

Oracle Database の以前のリリースでは、DBA が作業領域の最大サイズを SQL 演算子（ソートやハッシュ結合など）のタイプごとに手動で指定する必要がありました。しかし、ワークロードは常に変化するため、この作業は非常に困難であることがわかりました。Oracle Database の現行リリースではこの手動 PGA メモリー管理もサポートされていますが、自動 PGA メモリー管理を有効化することをお薦めします。

メモリー管理方法の概要

表 8-2 で、各メモリー管理方法を説明します。自動メモリー管理を有効化しない場合は、SGA および PGA のそれぞれに対して個別にメモリー管理方法を構成する必要があります。

注意： 自動メモリー管理を有効化しない場合、インスタンス PGA に対しては自動 PGA メモリー管理がデフォルトの方法となります。

表 8-2 Oracle Database のメモリー管理モード

メモリー管理モード	対象	設定項目	Oracle Database により自動チューニングされる項目
自動メモリー管理	SGA および PGA	■ Oracle インスタンスの合計メモリー・ターゲット・サイズ ■ （オプション）Oracle インスタンスの最大メモリー・サイズ	■ 合計 SGA サイズ ■ SGA コンポーネント・サイズ ■ インスタンス PGA サイズ ■ 個々の PGA サイズ
自動共有メモリー管理 （自動メモリー管理は無効化）	SGA	■ SGA のターゲット・サイズ ■ （オプション）SGA の最大サイズ	SGA コンポーネント・サイズ
手動共有メモリー管理 （自動メモリー管理および自動共有メモリー管理は無効化）	SGA	■ 共有プール・サイズ ■ バッファ・キャッシュ・サイズ ■ Java プール・サイズ ■ ラージ・プール・サイズ	-
自動 PGA メモリー管理	PGA	インスタンス PGA のターゲット・サイズ	個々の PGA サイズ
手動 PGA メモリー管理 （非推奨）	PGA	SQL 演算子の各タイプに対する作業領域の最大サイズ	-

関連項目： 自動メモリー管理が使用できないプラットフォームもあります。『Oracle Database 管理者ガイド』を参照してください。

データベース・インストール時のメモリー管理オプションおよびデフォルト設定

Database Configuration Assistant（DBCA）を使用してデータベースを作成し、基本インストール・オプションを選択した場合、デフォルトでは自動メモリー管理が有効化されます。拡張インストールを選択した場合、Database Configuration Assistant（DBCA）では、次の 3 つのメモリー管理構成からいずれかを選択できます。

- 自動メモリー管理
- 自動共有メモリー管理および自動 PGA メモリー管理
- 手動共有メモリー管理および自動 PGA メモリー管理

`CREATE DATABASE` SQL 文を使用してデータベースを作成する際、必要なメモリー初期化パラメータを指定することによりメモリー管理モードを選択しない場合は、デフォルトで手動共有メモリー管理および自動 PGA メモリー管理が構成されます。

> **関連項目：** メモリー管理およびメモリー管理初期化パラメータの詳細は、『Oracle Database 管理者ガイド』を参照してください。

ソフトウェア・コード領域

ソフトウェア・コード領域とは、実行中または実行される可能性があるコードを格納するためのメモリー部分です。Oracle Database のコードはソフトウェア領域に格納されます。この場所は通常ユーザー・プログラムの格納場所とは異なり、排他的であるかまたは保護されています。

ソフトウェア領域のサイズは通常固定されており、ソフトウェアの更新時か再インストール時にかぎり変化します。これらの領域に必要なサイズは、オペレーティング・システムによって異なります。

ソフトウェア領域は読取り専用であり、共有または非共有でインストールされます。Oracle Database コードは、メモリー内に複数のコピーを格納することなくすべてのユーザーがアクセスできるよう、可能なかぎり共有されます。これにより、実際の主メモリーが節約され、全体のパフォーマンスが改善されます。

ユーザー・プログラムは共有でも非共有でもかまいません。Oracle Forms や SQL*Plus などの Oracle のツール製品およびユーティリティによっては、共有でインストールできるものもありますが、共有にできないものもあります。Oracle Database の複数のインスタンスが同じコンピュータ上で実行されている場合、それらのインスタンスは異なるデータベースにおいても同じ Oracle Database コード領域を使用できます。

注意： ソフトウェアを共有でインストールするオプションは、すべてのオペレーティング・システムで使用できるわけではありません（たとえば、Windows が稼働している PC では使用できません）。

詳細は、オペレーティング・システム別の Oracle Database マニュアルを参照してください。

9

プロセス・アーキテクチャ

この章では、Oracle データベース・システムのプロセスと、Oracle データベース・システムで使用可能な各種構成について説明します。

この章の内容は、次のとおりです。

プロセスの概要

Oracle Database に接続されたすべてのユーザーは、Oracle Database インスタンスにアクセスするために、2 つのコード・モジュールを実行する必要があります。

- アプリケーションまたは Oracle のツール製品 : データベース・ユーザーは、データベース・アプリケーション（プリコンパイラ・プログラムなど）や Oracle のツール製品（SQL*Plus など）を実行します。これらは、Oracle データベースに SQL 文を発行します。
- Oracle データベース・サーバー・コード : 各ユーザーのために実行される Oracle データベース・コードで、アプリケーションの SQL 文を解釈および処理します。

これらのコード・モジュールは、プロセスによって実行されます。**プロセス**は制御のスレッド、つまり一連の処理を実行できるオペレーティング・システムのメカニズムです。（オペレーティング・システムによっては、**ジョブ**または**タスク**という用語を使用します）。プロセスには、通常、実行するプライベート・メモリー領域があります。

この項の内容は、次のとおりです。

マルチ・プロセス Oracle システム

マルチ・プロセス Oracle（**マルチユーザー Oracle** とも呼ばれる）は、Oracle コードの各部分とユーザーのためのその他のプロセスを実行するために、複数のプロセスを使用します。ユーザー用プロセスは、接続中のユーザーごとに個別のプロセスの場合や、複数のユーザーが共有する 1 つ以上のプロセスの場合があります。データベースの主な利点の 1 つが、複数のユーザーが同時に必要とするデータを管理できることにあるため、多くのデータベース・システムはマルチユーザー・システムです。

Oracle Database インスタンスの各プロセスは、特定のジョブを実行します。Oracle Database とデータベース・アプリケーションの処理を複数のプロセスに分割することにより、1 つのデータベース・インスタンスに複数のユーザーおよびアプリケーションが同時に接続でき、優れたシステム・パフォーマンスを維持できます。

プロセスのタイプ

Oracle Database システムのプロセスは、次の 2 つの大きなグループに分類できます。

- ユーザー・プロセスは、アプリケーションまたは Oracle のツール製品のコードを実行します。
- Oracle Database プロセスは、Oracle データベース・サーバーのコードを実行します。プロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。

プロセスの構造は、オペレーティング・システムおよび Oracle Database オプションの選択に応じて、Oracle Database の構成ごとに異なります。接続ユーザー用のコードは、専用サーバーまたは共有サーバーとして構成できます。

専用サーバーの場合は、それぞれのユーザーについて、データベース・アプリケーションを実行するプロセス（ユーザー・プロセス）と、Oracle データベース・サーバー・コードを実行するプロセス（専用サーバー・プロセス）が異なります。

共有サーバーの場合は、データベース・アプリケーションを実行するプロセス（ユーザー・プロセス）と、Oracle データベース・サーバー・コードを実行するプロセスが異なります。Oracle データベース・サーバー・コードを実行する各サーバー・プロセス（**共有サーバー・プロセス**）は、マルチ・ユーザー・プロセスとして機能します。

図 9-1 に、専用サーバー構成を示します。接続ユーザーごとに専用のユーザー・プロセスが生成され、Oracle Database で複数のバックグラウンド・プロセスが実行されています。

図 9-1 Oracle Database インスタンス

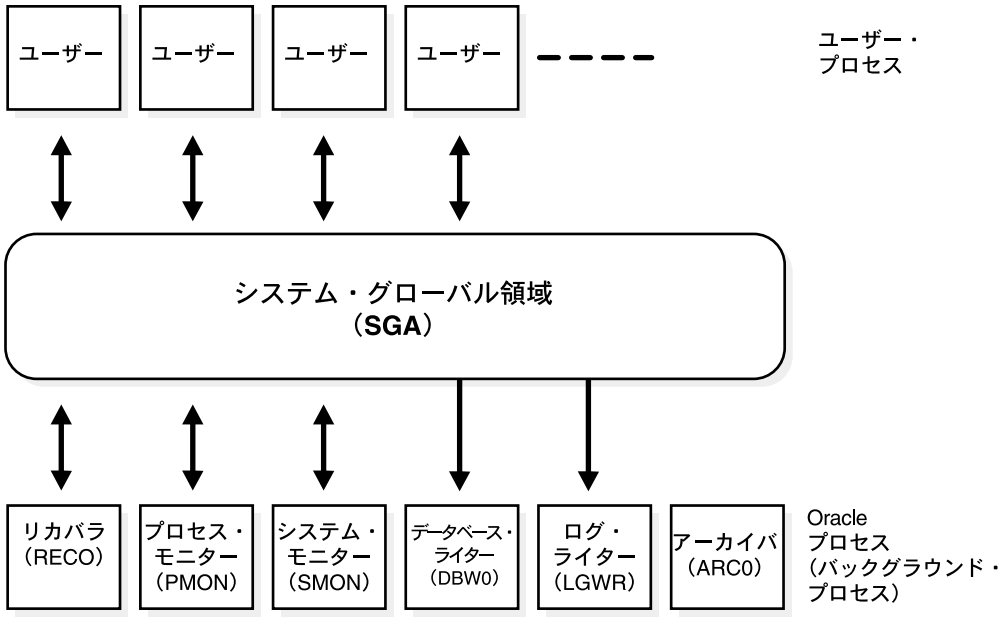


図 9-1 では、Oracle Database と同じコンピュータで複数の同時実行ユーザーがアプリケーションを実行しています。このような特別の構成は、通常、メインフレームまたはミニコンピュータで稼働します。

関連項目：

- 9-4 ページの「ユーザー・プロセスの概要」
- 9-4 ページの「Oracle Database プロセスの概要」
- 9-18 ページの「専用サーバー構成」
- 9-14 ページの「共有サーバー・アーキテクチャ」
- 構成の選択の詳細は、オペレーティング・システム固有の Oracle Database マニュアルを参照してください。

ユーザー・プロセスの概要

Oracle Database では、ユーザーがアプリケーション・プログラム（Pro*C プログラムなど）または Oracle のツール製品（Oracle Enterprise Manager や SQL*Plus など）を実行するとき、**ユーザー・プロセス**を作成してユーザーのアプリケーションを実行します。

接続とセッション

接続および**セッション**という用語は、**ユーザー・プロセス**という用語と密接に関連していますが、意味的には大きな差異があります。

接続とは、ユーザー・プロセスと Oracle Database インスタンスとの通信経路のことです。通信経路は、使用可能なプロセス間通信メカニズム（1 台のコンピュータでユーザー・プロセスと Oracle Database の両方を実行する場合）、またはネットワーク・ソフトウェア（データベース・アプリケーションと Oracle Database を別々のコンピュータで実行し、ネットワークを介して通信する場合）を使用して確立されます。

セッションとは、ユーザー・プロセスによるユーザーと Oracle Database インスタンスとの特定の接続です。たとえば、ユーザーは、SQL*Plus を開始するときに、有効なユーザー名とパスワードを入力する必要があります。そうすることにより、そのユーザーのためのセッションが確立されます。セッションは、ユーザーが接続した時点から、接続を切断するかデータベース・アプリケーションを終了する時点まで続きます。

同じユーザー名を使用して、1 人の Oracle Database ユーザーのセッションを複数作成し、それらを同時に存在させることができます。たとえば、`SCOTT/TIGER` というユーザー名 / パスワードを持つユーザーが、同じ Oracle Database インスタンスに複数回接続できます。

共有サーバーのない構成では、Oracle Database は、各ユーザー・セッションに対して 1 つのサーバー・プロセスを作成します。しかし、共有サーバーを使用すると、多くのユーザー・セッションで 1 つのサーバー・プロセスを共有できます。

関連項目： 9-14 ページの「共有サーバー・アーキテクチャ」

Oracle Database プロセスの概要

この項では、Oracle データベース・サーバー・コードを実行する 2 種類のプロセス（サーバー・プロセスとバックグラウンド・プロセス）について説明します。また、Oracle Database プロセスに関するデータベース・イベントが記録されるトレース・ファイルとアラート・ログについても説明します。

この項の内容は、次のとおりです。

- Oracle Database サーバー・プロセス
- Oracle Database バックグラウンド・プロセス
- Oracle Database トレース・ファイルとアラート・ログ

Oracle Database サーバー・プロセス

Oracle Database では、インスタンスに接続されたユーザー・プロセスの要求を処理するために、**サーバー・プロセス**が作成されます。アプリケーションと Oracle Database が同一のコンピュータで実行している場合は、ユーザー・プロセスとそれに対応するサーバー・プロセスを 1 つのプロセスに結合して、システムのオーバーヘッドを軽減できる場合もあります。ただし、アプリケーションと Oracle Database がそれぞれ別のコンピュータで実行している場合は、ユーザー・プロセスは常に専用のサーバー・プロセスを通じて Oracle Database と通信します。

各ユーザー・アプリケーションのために作成されたサーバー・プロセス（または、結合されたユーザー / サーバー・プロセスのサーバー部）は、次の 1 つ以上の操作を実行できます。

- アプリケーションを介して発行された SQL 文を解析し、実行します。
- 必要なデータ・ブロックが SGA 内に存在しない場合、そのブロックをディスク上のデータファイルから SGA の共有データベース・バッファに読み取ります。
- アプリケーションが情報を処理できるような方法で結果を戻します。

Oracle Database バックグラウンド・プロセス

パフォーマンスを最大化し、多数のユーザーに対処するために、マルチプロセス Oracle Database システムでは、**バックグラウンド・プロセス**と呼ばれる追加の Oracle Database プロセスを使用します。

An Oracle Database instance can have many background processes; not all are always present. バックグラウンド・プロセスには、多数の種類があります。バックグラウンド・プロセスの詳細は、`V$BGPROCESS` ビューを参照してください。Oracle Database インスタンスには、次のようなバックグラウンド・プロセスがあります。

- アーカイバ・プロセス（ARCn）
- チェックポイント・プロセス（CKPT）
- データベース・ライター・プロセス（DBWn）
- ジョブ・キュー・プロセス
- ログ・ライター・プロセス（LGWR）
- プロセス・モニター・プロセス（PMON）
- キュー・モニター・プロセス（QMNn）
- リカバラ・プロセス（RECO）
- システム・モニター・プロセス（SMON）
- その他の Oracle Database バックグラウンド・プロセス

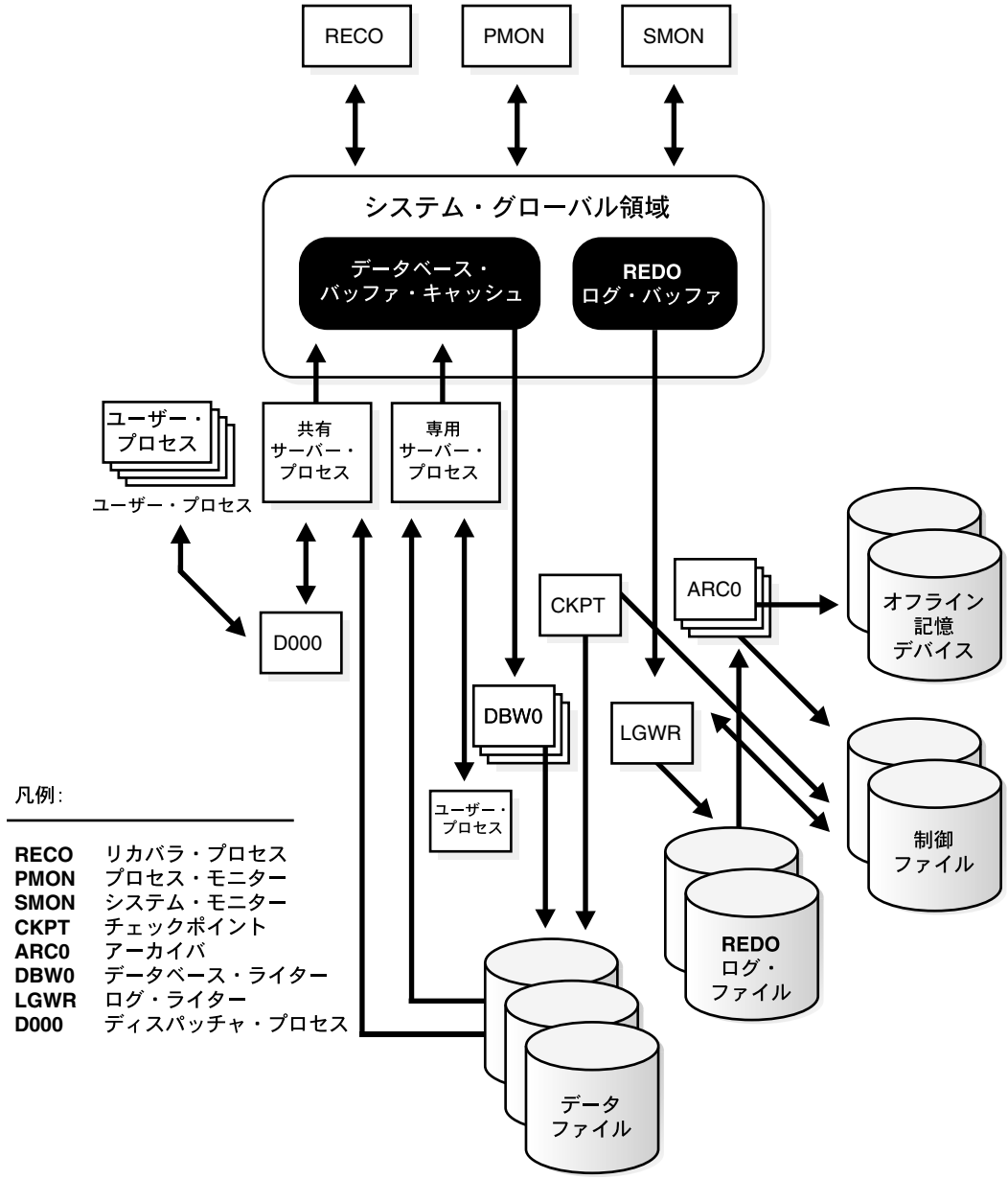
多くのオペレーティング・システムでは、インスタンスが起動するときに、バックグラウンド・プロセスが自動的に作成されます。

図 9-2 に、Oracle データベースの各部分とそれぞれのバックグラウンド・プロセスとの相互作用を示します。その後、各プロセスについて説明します。

関連項目：

- 詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』および『Oracle Clusterware 管理およびデプロイメント・ガイド』を参照してください。図 9-2 には、Oracle Real Application Clusters は示されていません。
- プロセスが作成される方法の詳細は、オペレーティング・システム固有のマニュアルを参照してください。

図 9-2 マルチ・プロセスの Oracle Database インスタンスのバックグラウンド・プロセス



アーカイバ・プロセス（ARC*n*）

アーカイバ・プロセス（ARCn）は、ログ・スイッチの発生後、REDO ログ・ファイルを指定のストレージ・デバイスにコピーします。また、トランザクションの REDO データを収集し、そのデータをスタンバイの宛先に転送できます。ARC*n* プロセスは、データベースが ARCHIVELOG モードに設定され、自動アーカイブが使用可能になっている場合にのみ存在します。

データのバルク・ロード中など、アーカイブに伴う大量のワークロードが予想される場合は、初期化パラメータ LOG_ARCHIVE_MAX_PROCESSES を使用してアーカイバ・プロセスの最大数を増加できます。ALTER SYSTEM 文では、このパラメータの値を動的に変更し、ARC*n* プロセス数を増減させることができます。

関連項目：

- 9-13 ページの「Oracle Database トレース・ファイルとアラート・ログ」
- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』
- ARC*n* プロセスの使用方法の詳細は、オペレーティング・システム固有のマニュアルを参照してください。

チェックポイント・プロセス（CKPT）

Oracle Database では、チェックポイントが発生すると、チェックポイントの詳細を記録するためにすべてのデータファイルのヘッダーを更新する必要があります。この処理は、CKPT プロセスによって実行されます。CKPT プロセスは、ブロックをディスクに書き込みません。この書込みは、常に DBW*n* が実行します。

Oracle Enterprise Manager の System_Statistics モニターによって表示される **DBWR チェックポイント**の統計には、完了したチェックポイント要求の数が示されます。

関連項目： Oracle Real Application Clusters による CKPT の詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

データベース・ライター・プロセス（DBW*n*）

データベース・ライター・プロセス（DBWn）は、バッファの内容をデータファイルに書き込みます。DBW*n* プロセスは、データベース・バッファ・キャッシュ内の変更された（使用済）バッファをディスクに書き込みます。ほとんどのシステムでは 1 つのデータベース・ライター・プロセス（DBW0）があれば十分ですが、追加プロセス（DBW1 ～ DBW9 および DBWa ～ DBWj）を構成して、システムでデータを頻繁に変更する場合に書込みのパフォーマンスを改善できます。これらの追加 DBW*n* プロセスは、ユニプロセッサ・システムでは効果がありません。

データベース・バッファ・キャッシュ内のバッファが変更されると、そのバッファには**使用済**のマークが付きます。**コールド**・バッファとは、LRU アルゴリズムに従って最近使用されたことがないバッファです。DBW*n* プロセスは、ユーザー・プロセスが新規ブロックをキャッシュに読み取るために使用できるクリーンなコールド・バッファを検出できるように、使用済のコールド・バッファをディスクに書き込みます。ユーザー・プロセスによってバッファが使用済の状態になると、使用可能バッファの数が減少します。使用可能バッファの数が少なすぎると、ディスクからキャッシュにブロックを読み取る必要があるユーザー・プロセスは、使用可能バッファを検出できません。ユーザー・プロセスが使用可能バッファを常に検出できるように、DBW*n* はバッファ・キャッシュを管理します。

使用済のコールド・バッファをディスクに書き込むことにより、DBW*n* は、使用頻度の高いバッファをメモリー内に保ちながら、使用可能バッファを検索するときのパフォーマンスを向上させます。たとえば、頻繁にアクセスされる小さな表または索引の一部であるブロックは、それらをディスクから再び読み取らなくても済むように、キャッシュ内に置かれます。LRU アルゴリズムは、バッファの内容をディスクに書き込むときに、すぐに使用されそうなデータが書き込まれてしまわないように、より頻繁にアクセスされるブロックをバッファ・キャッシュ内に保持します。

DBW*n* プロセスの個数は、初期化パラメータ DB_WRITER_PROCESSES で指定します。DBW*n* プロセスの最大数は 20 です。起動時にユーザーが最大数を設定しない場合は、Oracle Database により CPU 数とプロセッサ・グループ数に基づいて DB_WRITER_PROCESSES の設定方法が決まります。

DBW*n* プロセスは、次のような場合に使用済バッファをディスクに書き込みます。

- サーバー・プロセスがしきい値の数までバッファをスキャンしても、クリーンな再利用可能バッファが見つからない場合は、DBW*n* に書込み信号が送られます。DBW*n* は、他の処理中に使用済バッファをディスクに非同期に書き込みます。
- DBW*n* は、定期的にバッファを書き込んで、REDO スレッド（ログ）内のインスタンス・リカバリを開始する位置（**チェックポイント**）を進めます。このログ位置は、バッファ・キャッシュ内の最も古い使用済バッファによって決まります。

どの場合でも、効率を向上させるために DBW*n* はバッチ（マルチブロック）書込みを実行します。マルチブロック書込みで書き込まれるブロックの数は、オペレーティング・システムによって異なります。

関連項目：

- 8-4 ページの「データベース・バッファ・キャッシュ」
- DB_WRITER_PROCESSES の設定のアドバイスと、DBW0 プロセスまたは複数 DBW*n* プロセスのパフォーマンスの監視とチューニング方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』

ジョブ・キュー・プロセス

ジョブ・キュー・プロセスは、バッチ処理で使用します。ジョブ・キュー・プロセスはユーザー・ジョブを実行します。ジョブ・キュー・プロセスは、ジョブを PL/SQL 文または Oracle Database インスタンスのプロシージャとしてスケジュールするために使用するスケジューラ・サービスであると考えることができます。開始日と間隔を指定すると、ジョブ・キュー・プロセスは、指定された間隔でジョブの実行を試行します。

ジョブ・キュー・プロセスは動的に管理されます。このため、ジョブ・キューのクライアントは、必要なときに、より多くのジョブ・キュー・プロセスを使用できます。新規プロセスが使用するリソースは、そのプロセスがアイドル状態になると解放されます。

動的ジョブ・キュー・プロセスは、指定された間隔で多数のジョブを同時に実行できます。ジョブ・キュー・プロセスは、ユーザー・ジョブが CJQ プロセスにより割り当てられると、それを実行します。ジョブ・キュー・プロセスは、次のように機能します。

1. コーディネータ・プロセス CJQ0 は、システムの JOB$ 表から、実行する必要があるジョブを定期的に選択します。選択された新規ジョブは、時間順に並べ替えられます。
2. CJQ0 プロセスは、ジョブを実行するためにジョブ・キュー・スレーブ・プロセス（J000 ～ J999）を動的に生成します。
3. ジョブ・キュー・プロセスは、実行のために CJQ プロセスによって選択されたジョブの 1 つを実行します。ジョブ・キュー・プロセスは、一度に 1 つずつジョブを実行します。
4. ジョブ・キュー・プロセスは、1 つのジョブの実行が終了すると、次のジョブの問合せを行います。実行予定のジョブが存在しない場合、ジョブ・キュー・プロセスはスリープ状態に入り、定期的な周期で起動すると次のジョブの問合せを行います。新しいジョブがない場合、プリセットした周期が経過した後に強制終了します。

初期化パラメータ JOB_QUEUE_PROCESSES は、1 つのインスタンスで同時に実行できるジョブ・キュー・プロセスの最大数を示します。ただし、クライアントは、すべてのジョブ・キュー・プロセスがジョブの実行に使用可能であると想定することはできません。

注意： 初期化パラメータ JOB_QUEUE_PROCESSES が 0（ゼロ）に設定されている場合、コーディネータ・プロセスは起動しません。

関連項目： ジョブ・キューの詳細は、『Oracle Database 管理者ガイド』を参照してください。

ログ・ライター・プロセス（LGWR）

ログ・ライター・プロセス（LGWR）は、REDO ログ・バッファ管理、つまりディスク上の REDO ログ・ファイルへの REDO ログ・バッファの書込みを行います。LGWR は、最後の書込み以後にバッファにコピーされた REDO エントリすべてを、REDO ログ・ファイルに書き込みます。

REDO ログ・バッファは循環バッファです。LGWR が REDO ログ・バッファから REDO ログ・ファイルに REDO エントリを書き込んだ後、サーバー・プロセスはディスクに書き込まれた REDO ログ・バッファのエントリ上に新しいエントリをコピーできます。REDO ログへのアクセスが頻繁なときにも、新しいエントリを書き込めるよう常にバッファ内の領域を空けておくため、LGWR が行う書込みは通常は非常に高速になります。

LGWR は、バッファの 1 つの連続した部分をディスクに書き込みます。LGWR は、次のものを書き込みます。

- ユーザー・プロセスがトランザクションをコミットしたときのコミット・レコード
- REDO ログ・バッファ
  - 3 秒ごと
  - REDO ログ・バッファが 3 分の 1 になったとき
  - 必要に応じ、DBW*n* プロセスが修正済のバッファをディスクに書き込むとき

注意： DBW*n* が修正済バッファを書き込むには、バッファの変更に関連するすべての REDO レコードがディスクに書き込まれている必要があります（**事前書込みプロトコル**）。DBW*n* は、いくつかの REDO レコードが書き込まれていないことを見つけた場合に、REDO レコードをディスクに書き込むように LGWR に信号を送り、REDO ログ・バッファの書込みの完了を待機してからデータ・バッファを書き出します。

LGWR は、ミラー化されたアクティブな REDO ログ・ファイルのグループにも同時に書き込みます。グループ内のファイルのいずれかが破損している場合や使用できない場合、LGWR はそのグループ内の他のファイルへの書込みを続け、このエラーのログを LGWR トレース・ファイルやシステム・アラート・ログに記録します。グループ内のすべてのファイルが破損している場合や、アーカイブされていないためにグループ全体が使用できない場合、LGWR は機能を続行できません。

ユーザーが COMMIT 文を発行すると、LGWR は REDO ログ・バッファ内にコミット・レコードを入れ、トランザクションの REDO エントリとともにそれを即時にディスクに書き込みます。対応するデータ・ブロックの変更は、それらをより効率的に書き込めるようになるまで延期されます。これを**高速コミット**・メカニズムと呼びます。トランザクションのコミット・レコードを含む REDO エントリのアトミックな書込みは、トランザクションがコミットされたかどうかを判別する 1 つのイベントです。Oracle Database では、データ・バッファがまだディスクに書き込まれていない場合でも、コミットしたトランザクションに対する成功コードが戻されます。

注意： より多くのバッファ領域が必要な場合に、LGWR はトランザクションがコミットされる前に REDO ログ・エントリを書き込むこともあります。後でトランザクションがコミットされた場合にのみ、これらのエントリは確定します。

ユーザーがトランザクションをコミットすると、そのトランザクションには**システム変更番号（SCN）**が割り当てられます。Oracle Database では、このシステム変更番号は、該当するトランザクションの REDO エントリとともに REDO ログに記録されます。SCN は、Real Application Clusters および分散データベースでリカバリ操作を同期させることができるように、REDO ログに記録されます。

アクティビティが高いときには、LGWR は**グループ・コミット**を使用して REDO ログ・ファイルに書き込むこともあります。たとえば、ユーザーがトランザクションをコミットする場合を考えます。LGWR は、トランザクションの REDO エントリをディスクに書き込む必要がありますが、その際、他のユーザーは `COMMIT` 文を発行します。ただし、LGWR は前の書込み操作を完了するまで、他のユーザーのトランザクションを REDO ログ・ファイルに書き込んでコミットすることはできません。最初のトランザクションのエントリを REDO ログ・ファイルに書き込んだ後、待機中の（まだコミットされていない）トランザクションの REDO エントリのリスト全体を 1 回の操作でディスクに書き込むことができるため、トランザクションのエントリを個々に処理するときよりも、必要となる I/O を低減できます。したがって、Oracle Database では、ディスク I/O が最小化され、LGWR のパフォーマンスが最大化されます。コミットの要求が高い頻度で続くと、REDO ログ・バッファからの（LGWR による）毎回の書込みに複数のコミット・レコードが含まれることがあります。

関連項目：

- 8-5 ページの「REDO ログ・バッファ」
- 9-13 ページの「Oracle Database トレース・ファイルとアラート・ログ」
- SCN およびその使用方法の詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。
- SCN およびその使用方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- LGWR のパフォーマンスの監視およびチューニング方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

プロセス・モニター・プロセス（PMON）

ユーザー・プロセスが失敗すると、**プロセス・モニター（PMON）**がプロセスをリカバリします。PMON は、データベース・バッファ・キャッシュをクリーン・アップしたり、ユーザー・プロセスが使用していたリソースを解放します。たとえば、アクティブ・トランザクション表の状態をリセットしてロックを解除し、アクティブ・プロセスのリストからプロセス ID を削除します。

PMON は、ディスパッチャ・プロセスとサーバー・プロセスの状態を定期的にチェックし、実行を停止したプロセスがあれば再起動します（ただし、Oracle Database が意図的に終了させたプロセスは除きます）。また、PMON は、インスタンスおよびディスパッチャ・プロセスに関する情報をネットワーク・リスナーに登録します。

SMON と同じように、PMON は、処理が必要かどうかを定期的にチェックし、別のプロセスで起動する必要が検出された場合にコールできます。

キュー・モニター・プロセス（QMN*n*）

キュー・モニター・プロセスは、メッセージ・キューを監視する Oracle Streams Advanced Queuing のオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。これらのプロセスは、ジョブ・キュー・プロセスと同様に、プロセスの障害がインスタンス障害の原因とならない点で他の Oracle Database バックグラウンド・プロセスと異なります。

関連項目：

- 23-8 ページの「Oracle Streams アドバンスト・キューイング」
- 『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』

リカバラ・プロセス（RECO）

リカバラ・プロセス（RECO）は、分散データベース構成で使用されるバックグラウンド・プロセスであり、分散トランザクションに関連する障害を自動的に解決します。ノードの RECO プロセスは、インダウト分散トランザクションにかかわる他のデータベースに自動的に接続されます。RECO プロセスは、関係するデータベース・サーバー間の接続を再確立するときに、すべてのインダウト・トランザクションを自動的に解決し、解決されるインダウト・トランザクションに対応する行を各データベースの保留中のトランザクション表からすべて削除します。

RECO プロセスがリモート・サーバーとの接続に失敗した場合、RECO は決められた間隔で自動的に再接続しようとします。ただし、RECO が次の接続を試行するまで待機する時間は増えていきます（指数関数的に増加します）。RECO プロセスは、インスタンスが分散トランザクションを許可している場合にのみ存在します。同時分散トランザクションの数に制限はありません。

> **関連項目：** 分散トランザクション・リカバリの詳細は、『Oracle Database 管理者ガイド』を参照してください。

システム・モニター・プロセス（SMON）

システム・モニター・プロセス（SMON）は、インスタンスの起動時に必要に応じてリカバリを実行します。また、SMON は、使用されなくなった一時セグメントをクリーン・アップする操作と、ディクショナリ管理表領域内で連続した使用可能エクステントを 1 つに結合する操作を受け持ちます。ファイル読取りエラーやオフライン・エラーが原因で、インスタンス・リカバリ時に終了済トランザクションがスキップされた場合、SMON はその表領域やファイルがオンラインに戻った時点でトランザクションをリカバリします。SMON は、処理が必要かどうかを定期的にチェックします。他のプロセスは、必要性が検出された場合に SMON をコールできます。

Real Application Clusters では、あるインスタンスの SMON プロセスは、障害が発生した CPU またはインスタンスについてもインスタンス・リカバリを実行できます。

> **関連項目：** SMON の詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

その他の Oracle Database バックグラウンド・プロセス

他にも実行されているバックグラウンド・プロセスがいくつかあります。次の SQL 問合せを発行して、システムで実行中のバックグラウンド・プロセスを表示できます。

```
SELECT * FROM V$BGPROCESS
WHERE  PADDR != '00'
ORDER BY NAME;
```

次のようなバックグラウンド・プロセスが含まれる場合があります。

- インスタンスごとの ACMS（メモリー・サービスへのアトミック制御ファイル）プロセスは、Oracle RAC 環境で、分散 SGA メモリーの更新が、成功した場合にグローバルにコミットされ、失敗した場合はグローバルに異常終了することを保証するためのエージェントです。
- DBRM（データベース・リソース・マネージャ）プロセスは、リソース・プランおよびその他のリソース・マネージャ関連のタスクを設定します。

  関連項目： データベース・リソース・マネージャの使用方法の詳細は、14-18 ページの「データベース・リソース・マネージャの概要」を参照してください。

- DIA0（障害診断性プロセス 0）（現在は 0 のみを使用）は、ハングを検出しデッドロックを解決します。
- DIAG（障害診断性）プロセスは、診断ダンプを実行し、グローバル `oradebug` コマンドを実行します。
- イベント監視コーディネータ（EMNC）は、データベースのイベント管理および通知に使用されるバックグラウンド・サーバー・プロセスです。
- FBDA（フラッシュバック・データ・アーカイバ・プロセス）は、フラッシュバック・データ・アーカイブに記録済テーブルの履歴行をアーカイブします。記録済テーブルとは、フラッシュバック・アーカイブに対して使用可能なテーブルです。記録済テーブルに DML を含んでいるトランザクションがコミットすると、このプロセスは、行の事前イメージをフラッシュバック・アーカイブに保存します。また、現在の行にメタデータを保存します。

  さらに FBDA は、領域、編成、および保存に対するフラッシュバック・データ・アーカイブを自動的に管理し、記録済トランザクションのアーカイブがどの程度発生するかを追跡します。
- GTX0-j（グローバル・トランザクション）プロセスは、Oracle RAC 環境において、XA グローバル・トランザクションを透過的にサポートします。データベースは、XA グローバル・トランザクションのワークロードに基づいて、これらのプロセスの数を自動調整します。グローバル・トランザクション・プロセスは、Oracle RAC 環境でのみ見られます。
- MMAN は、内部的なデータベース・タスクで使用します。
- MMNL は、セッション履歴の取得、メトリック計算など、管理性に関連する軽量のバックグラウンド・タスクを頻繁に実行します。
- MMON は、管理性に関連する様々なバックグラウンド・タスクを実行します。次に例を示します。
  - 指定のメトリックがしきい値に違反したときにアラートを発行します。
  - 追加プロセス（MMON スレーブ）を起動してスナップショットを取ります。
  - 前回変更された SQL オブジェクトの統計値を取得します。
- ARB*n* は、自動ストレージ管理インスタンスで実際の再バランス・データ・エクステントの移動を実行します。この種のバックグラウンド・プロセスは同時に多数実行される場合があるため、ARB0、ARB1 などと呼びます。
- プロセス・スポーナ（PSP0）は、Oracle プロセスを起動します。

- RBAL は、自動ストレージ管理インスタンスでディスク・グループに対するバランス再調整アクティビティを調整します。このプロセスは、自動ストレージ管理ディスクに対するグローバル・オープンを実行します。
- 領域管理コーディネータ（SMCO）プロセスは、予防的な領域の管理および領域の再生など、様々な領域管理関連タスクの実行を調整します。これにより、自動的にスレーブ・プロセス（W*nnn*）が起動され、タスクが実装されます。
- 時間の仮想キーパー（VKTM）は、実時間（秒ごとに更新）および参照時間カウンタ（20 ミリ秒ごとに更新され、優先順位が上がった場合にのみ利用可能）を提供します。

関連項目：

- Oracle Clusterware バックグラウンド・プロセスの詳細は、『Oracle Clusterware 管理およびデプロイメント・ガイド』を参照してください。
- Oracle Real Application Clusters バックグラウンド・プロセスの詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。
- ASM バックグラウンド・プロセスの詳細は、『Oracle Database ストレージ管理者ガイド』を参照してください。

Oracle Database トレース・ファイルとアラート・ログ

Oracle Database 11*g* からは、問題を防止、検出、診断および解決するための高度な障害診断性インフラストラクチャが含まれるようになりました。特にターゲットとなる問題は、データベース・コードの不具合、メタデータの破損、顧客データの破損などが原因の重大なエラーです。

致命的なエラーが発生すると、それに対してインシデント番号が割り当てられ、エラーに対する診断データ（トレース・ファイルなど）が即座に取得され、この番号がタグ付けされます。その後、データは自動診断リポジトリ（ADR）（データベースの外部にあるファイル・ベースのリポジトリ）に保存され、後でインシデント番号を使用して取り出して分析できます。

それぞれのサーバーとバックグラウンド・プロセスは、対応付けられた**トレース・ファイル**に書き込むことができます。プロセスによって内部エラーが検出されると、エラーについての情報がそのプロセスのトレース・ファイルに書き込まれます。内部エラーが発生して情報がトレース・ファイルに書き込まれた場合、管理者は Oracle サポート・サービスに連絡してください。

バックグラウンド・プロセスに対応付けられているすべてのトレース・ファイルのファイル名には、そのトレース・ファイルを生成したプロセスの名前が含まれます。唯一の例外は、ジョブ・キュー・プロセス（Jnnn）が生成するトレース・ファイルです。

トレース・ファイル内の追加情報は、アプリケーションやインスタンスをチューニングするための手引きにもなります。バックグラウンド・プロセスは、該当する場合は、いつもトレース・ファイルにこの情報を書き込みます。

また、各データベースには、`alert.log` があります。データベースのアラート・ログは、次のようなメッセージとエラーの履歴ログです。

- 発生したすべての内部エラー（ORA-00600）、ブロック障害エラー（ORA-01578）およびデッドロック・エラー（ORA-00060）。
- 管理的な操作。たとえば、SQL 文の `CREATE/ALTER/DROP DATABASE/TABLESPACE` と、Oracle Enterprise Manager または SQL*Plus 文の `STARTUP`、`SHUTDOWN`、`ARCHIVE LOG`、および `RECOVER` など。
- 共有サーバーとディスパッチャ・プロセスの機能に関係するいくつかのメッセージとエラー。
- マテリアライズド・ビューの自動リフレッシュにおけるエラー。

Oracle Database では、オペレータのコンソール上に情報を表示するかわりに、アラート・ログを使用してこれらのイベントの記録を保管します。（多くのシステムではコンソール上にもこの情報が表示されます。）管理操作が成功すると、タイムスタンプとともに「completed」というメッセージがアラート・ログに書き込まれます。

関連項目：

- SQL トレース機能を使用可能にする方法については、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- エラー・メッセージの詳細は、『Oracle Database エラー・メッセージ』を参照してください。

共有サーバー・アーキテクチャ

共有サーバー・アーキテクチャでは、それぞれの接続に対する専用サーバー・プロセスが必要ありません。ディスパッチャが、複数の受信ネットワーク・セッション要求を共有サーバー・プロセスのプールに導きます。サーバー・プロセスの共有プールにあるアイドル状態の共有サーバー・プロセスは、共通キューからの要求をピックアップします。このことは、少数の共有サーバーで、多数の専用サーバーと同じ量の処理を行えることを意味します。また、各ユーザーに必要なメモリーの量が比較的少ないため、メモリー管理やプロセス管理も容易であり、多くのユーザーをサポートできます。

共有サーバー・システムでは、次に示すように、様々なプロセスが多数必要です。

- ユーザー・プロセスをディスパッチャまたは専用サーバーに接続するネットワーク・リスナー・プロセス（リスナー・プロセスは、Oracle Database ではなく Oracle Net Services の一部）
- 1 つ以上のディスパッチャ・プロセス
- 1 つ以上の共有サーバー・プロセス

共有サーバー・プロセスでは、Oracle Net Services または SQL*Net バージョン 2 が必要です。

注意： 共有サーバーを使用する際には、ユーザー・プロセスは、Oracle Database インスタンスと同一のコンピュータで実行されている場合でも、Oracle Net Services または SQL*Net バージョン 2 を介して接続する必要があります。

インスタンスが起動されると、ネットワーク・リスナー・プロセスは、ユーザーが Oracle Database への接続に使用する通信経路をオープンして確立します。その後、各ディスパッチャ・プロセスは、接続要求をリスニングするアドレスをリスナー・プロセスに割り当てます。データベース・クライアントで使用するネットワーク・プロトコルごとに、最低 1 つ以上のディスパッチャ・プロセスを構成し起動する必要があります。

ユーザー・プロセスが接続を要求すると、リスナーはその要求を調べてユーザー・プロセスが共有サーバー・プロセスを使用できるかどうかを決定します。使用できる場合には、リスナー・プロセスは負荷が最も軽いディスパッチャ・プロセスのアドレスを戻し、ユーザー・プロセスはディスパッチャに直接接続します。

ディスパッチャと通信できないユーザー・プロセスもあるため、ネットワーク・リスナー・プロセスはそれらをディスパッチャに接続できません。この場合、つまりユーザー・プロセスが専用サーバーを要求すると、リスナーは専用サーバーを作成して適切な接続を確立します。

Oracle Database の共有サーバー・アーキテクチャでは、アプリケーションのスケーラビリティが向上し、データベースへのクライアントの同時接続数が増加します。このアーキテクチャでは、アプリケーション自体を一切変更することなく、既存のアプリケーションをスケールアップできます。

関連項目：

- 9-18 ページの「共有サーバーの限定的運用」
- ネットワーク・リスナーの詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

この項の内容は、次のとおりです。

ディスパッチャの要求キューとレスポンス・キュー

ユーザーからの要求は、ユーザーの SQL 文の一部である単一のプログラム・インタフェース・コールです。ユーザーがコールすると、そのディスパッチャが要求を**要求キュー**に入れ、次に使用可能な共有サーバー・プロセスがそこから要求を取り出します。

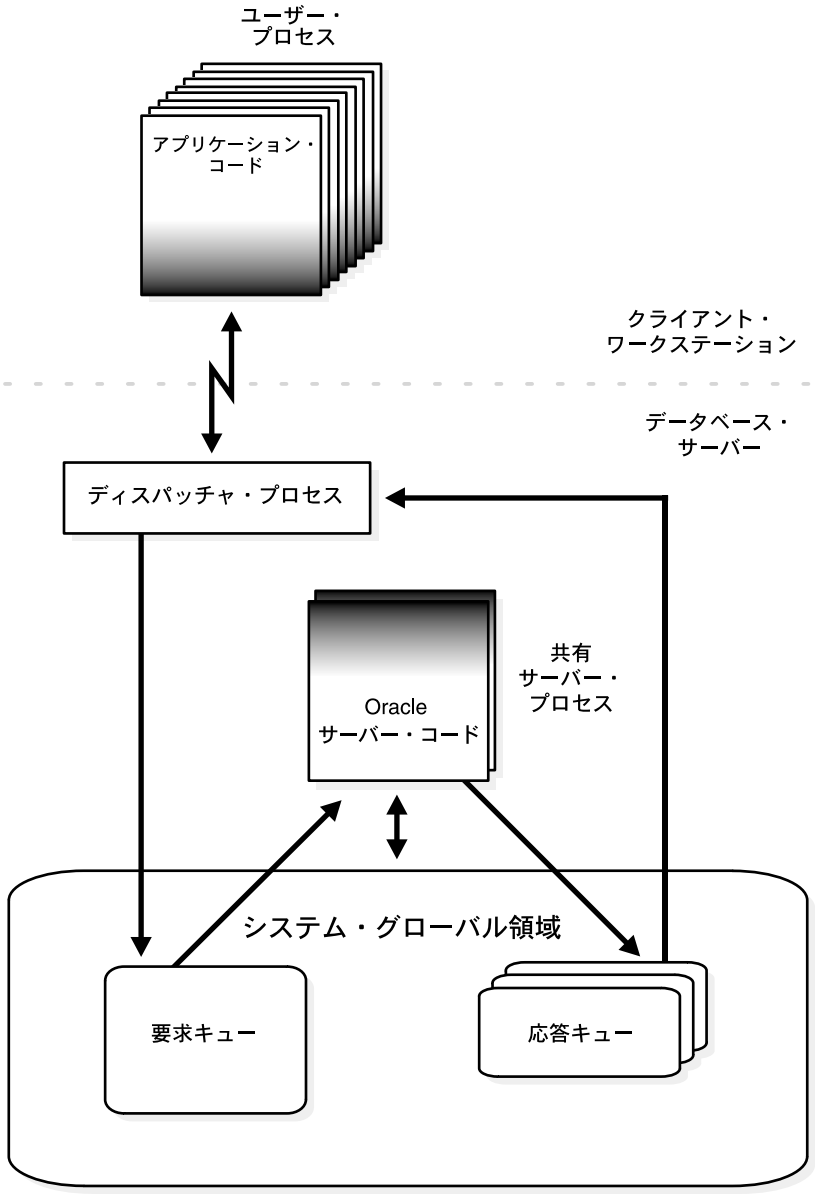
要求キューは SGA 内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。共有サーバー・プロセスは、共通の要求キューをチェックして新しい要求がないかどうかを調べ、先入れ先出し方式に基づいて新しい要求をピックアップします。1 つの共有サーバー・プロセスがキュー内の要求を 1 つピックアップし、その要求を完了するのに必要なデータベースに対するコールをすべて出します。

サーバーは要求を完了すると、コール・ディスパッチャの**レスポンス・キュー**に応答を入れます。各ディスパッチャには、SGA 内に固有のレスポンス・キューがあります。ディスパッチャは、完了した要求を適切なユーザー・プロセスに戻します。

たとえば、注文入力システムでは、それぞれの事務担当のユーザー・プロセスがディスパッチャに接続し、事務担当が出したそれぞれの要求がそのディスパッチャに送られ、そしてディスパッチャがその要求を要求キューに入れます。次に使用可能な共有サーバー・プロセスは、要求をピックアップして処理し、レスポンス・キューに応答を入れます。事務担当の要求が完了しても、事務担当はディスパッチャに接続されたままですが、その要求を処理した共有サーバー・プロセスは解放されるため、別の要求で使用できます。1 人の事務担当が顧客と話し合っている間に、別の事務担当は同じ共有サーバー・プロセスを使用できます。

図 9-3 に、ユーザー・プロセスがプログラム・インタフェースを介してディスパッチャと通信する方法と、ディスパッチャがユーザー要求を共有サーバー・プロセスに伝達する方法を示します。

図 9-3 共有サーバー構成とプロセス



この項の内容は、次のとおりです。

- ディスパッチャ・プロセス（Dnnn）
- 共有サーバー・プロセス（Snnn）

ディスパッチャ・プロセス（D*nnn*）

ディスパッチャ・プロセスは、ユーザー・プロセスが限定された数のサーバー・プロセスを共有できるようにすることで、共有サーバー構成をサポートします。共有サーバーでは、共有サーバー・プロセスの数がユーザーの数より少なくてすみます。そのため、特にクライアント・アプリケーションとサーバーが別々のコンピュータ上で稼働するようなクライアント / サーバー環境では、共有サーバーによってより多くのユーザーをサポートできます。

1 つのデータベース・インスタンスに対し、複数のディスパッチャ・プロセスを作成できます。ただし、Oracle Database で使用する各ネットワーク・プロトコルごとに、少なくとも 1 つのディスパッチャを作成する必要があります。データベース管理者は、オペレーティング・システムの制限とプロセス当たりの接続数に応じて、ディスパッチャ・プロセスを適切な数だけ起動する必要があります。また、インスタンスの実行中にディスパッチャ・プロセスを追加および削除できます。

注意： ディスパッチャに接続するそれぞれのユーザー・プロセスは、両方のプロセスが同一のコンピュータで実行されている場合であっても、Oracle Net Services または SQL*Net バージョン 2 を介して接続する必要があります。

共有サーバー構成では、ネットワーク・リスナー・プロセスがクライアント・アプリケーションからの接続要求を待機し、それぞれのクライアント・アプリケーションをディスパッチャ・プロセスにルーティングします。クライアント・アプリケーションをディスパッチャに接続できない場合、リスナー・プロセスは専用サーバー・プロセスを起動し、クライアント・アプリケーションを専用サーバーに接続します。リスナー・プロセスは、Oracle Database インスタンスの一部ではなく、Oracle Database と連携するネットワーキング・プロセスの一部です。

関連項目：

- 9-14 ページの「共有サーバー・アーキテクチャ」
- ネットワーク・リスナーの詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

共有サーバー・プロセス（S*nnn*）

共有サーバー構成では、各**共有サーバー・プロセス**は複数のクライアント要求を処理します。共有サーバー・プロセスには専用サーバー・プロセスと同じ機能がありますが、特定のユーザー・プロセスとは対応付けられていません。共有サーバー・プロセスは、共有サーバー構成におけるクライアント要求に対してサービスを提供します。

共有サーバー・プロセスの PGA には、ユーザーに関係した（すべての共有サーバー・プロセスからアクセス可能であることが必要な）データは含まれません。共有サーバー・プロセスの PGA には、スタック領域とプロセス固有の変数のみが含まれています。

セッション関連の情報はすべて SGA 内に入れられます。各共有サーバー・プロセスは、サーバーがどのセッションからの要求でも処理できるように、すべてのセッションのデータ領域にアクセスできる必要があります。セッションのデータ領域ごとに、SGA 内に領域が割り当てられます。ユーザーのプロファイル内のリソース制限 `PRIVATE_SGA` を必要な領域の容量に設定すると、セッションが割り当てることのできる領域の容量を制限できます。

Oracle Database は、要求キューの長さに基づいて、共有サーバー・プロセスの数を動的に調整します。作成できる共有サーバー・プロセスの数は、初期化パラメータ `SHARED_SERVERS` と `MAX_SHARED_SERVERS` の値で指定した範囲です。

関連項目：

- 各種インスタンス構成における PGA の内容の詳細は、8-10 ページの「プログラム・グローバル領域の概要」を参照してください。
- リソース制限とプロファイルの詳細は、第 20 章「データベース・セキュリティ」を参照してください。

共有サーバーの限定的運用

インスタンスの停止、インスタンスの起動、およびメディア・リカバリを含む、特定の管理アクティビティは、ディスパッチャ・プロセスに接続されている間は実行できません。ディスパッチャ・プロセスに接続されている間にこれらのアクティビティを実行しようとすると、エラー・メッセージが出されます。

これらのアクティビティは、一般的には管理者権限を使用して接続しているときに実行されます。共有サーバーにより構成されたシステムで管理者権限を使用して接続する場合は、ディスパッチャ・プロセスではなく専用サーバー・プロセスを使用することを接続文字列で明言してください（SERVER=DEDICATED）。

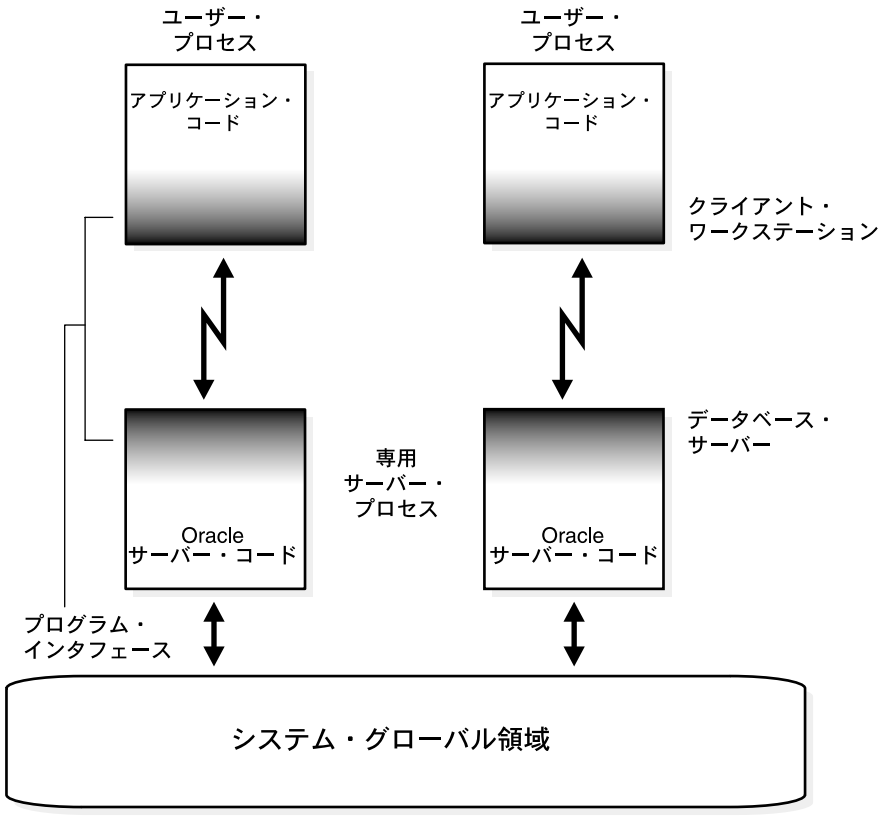
関連項目：

- オペレーティング・システム固有のマニュアル
- 正しい接続文字列の構文は、『Oracle Database Net Services 管理者ガイド』を参照してください。

専用サーバー構成

図 9-4 では、専用サーバー・アーキテクチャを使用して、Oracle Database が 2 台のコンピュータで実行されています。この構成では、データベース・アプリケーションが 1 台のコンピュータのユーザー・プロセスによって実行され、対応付けられた Oracle データベース・サーバーがもう 1 台のコンピュータ上のサーバー・プロセスによって実行されます。

図 9-4 Oracle Database での専用サーバー・プロセスの使用



ユーザー・プロセスとサーバー・プロセスは、相互に分離した別のプロセスです。ユーザー・プロセスごとに作成された別々のサーバー・プロセスは、このサーバー・プロセスが対応付けられたユーザー・プロセスのためにのみ活動するため、**専用サーバー・プロセス**（または**シャドウ**・プロセス）と呼ばれます。

この構成では、ユーザー・プロセス数とサーバー・プロセス数の比率が 1 対 1 に維持されます。ユーザーが活発にデータベース要求をしていない場合でも、専用サーバー・プロセスはそのまま残ります（ただし、活動していない状態の場合、オペレーティング・システムによってはページアウトされることもあります）。

図 9-4 では、ユーザー・プロセスとサーバー・プロセスが、ネットワークで接続された別々のコンピュータで実行されています。専用サーバー・アーキテクチャは、同じコンピュータ上でクライアント・アプリケーションと Oracle データベース・サーバー・コードの両方が実行される場合にも使用されますが、この 2 つのプログラムが 1 つのプロセスで実行されている場合は、ホスト・オペレーティング・システムはその 2 つの分離を維持できません。UNIX はそのようなオペレーティング・システムの一例です。

専用サーバー構成では、ユーザー・プロセスとサーバー・プロセスは異なるメカニズムを使用して通信します。

- ユーザー・プロセスと専用サーバー・プロセスを同じコンピュータで実行するようにシステムが構成される場合、プログラム・インタフェースは、ホスト・オペレーティング・システムのプロセス間通信メカニズムを使用してジョブを実行します。
- ユーザー・プロセスと専用サーバー・プロセスを別々のコンピュータで実行する場合は、プログラム・インタフェースがプログラム間の通信メカニズム（ネットワーク・ソフトウェアや Oracle Net Services など）を提供します。
- 専用サーバー・アーキテクチャによって、効率が低下する場合もあります。専用サーバー・プロセスによる注文入力システムの例を考えてみます。顧客から注文が入り、事務担当がデータベースにその注文を入力します。トランザクションの大部分では、事務担当が顧客と話し合っており、事務担当のユーザー・プロセス専用のサーバー・プロセスはアイドル状態になっています。サーバー・プロセスは、トランザクションの大部分で不要であり、他の事務担当が注文を入力するときにシステムの処理速度は遅くなります。このようなアプリケーションでは、共有サーバー・アーキテクチャを選択してください。

関連項目：

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

通信リンクの詳細は、前述のドキュメントを参照してください。

データベース常駐接続プーリング

データベース常駐接続プーリング（DRCP）は、一般的な Web アプリケーション使用のシナリオにおいて、データベース・サーバー内で接続プールを提供します。DRCP は、プールされたサーバーを作成するために、データベース・セッションと組み合せたサーバー・フォアグラウンドから構成される専用サーバーをプールします。

通常、Web アプリケーションは、データベースへの接続を取得し、その接続を短時間使用した後、接続を解放します。DRCP では、複数の Web アプリケーションのスレッドおよびプロセスが、サーバー・プールを共有して接続ニーズを満たすことができます。

DRCP は、中間層プロセス内のスレッド間で接続を共有する、中間層接続プールを補完します。また、DRCP は、複数の中間層プロセスにわたってデータベース接続を共有できるようにします。これらの中間層プロセスは、中間層ホストが同一の場合と異なる場合があります。

DRCP は、多数のクライアント接続のサポートに必要な主要データベース・リソースを大幅に削減できます。DRCP は、データベース・サーバーに必要なメモリーの量を削減し、データベース・サーバーおよび中間層両方のスケーラビリティを向上させます。すぐに利用できるサーバーのプールにより、クライアント接続を再作成するコストも削減されます。

DRCP は、中間層接続プーリングを実行できない PHP や Apache サーバーなど、マルチプロセスのシングルスレッド・アプリケーション・サーバーを含むアーキテクチャに特に有効です。DRCP を使用すると、データベースは同時接続数を数万にまで増やすことができます。

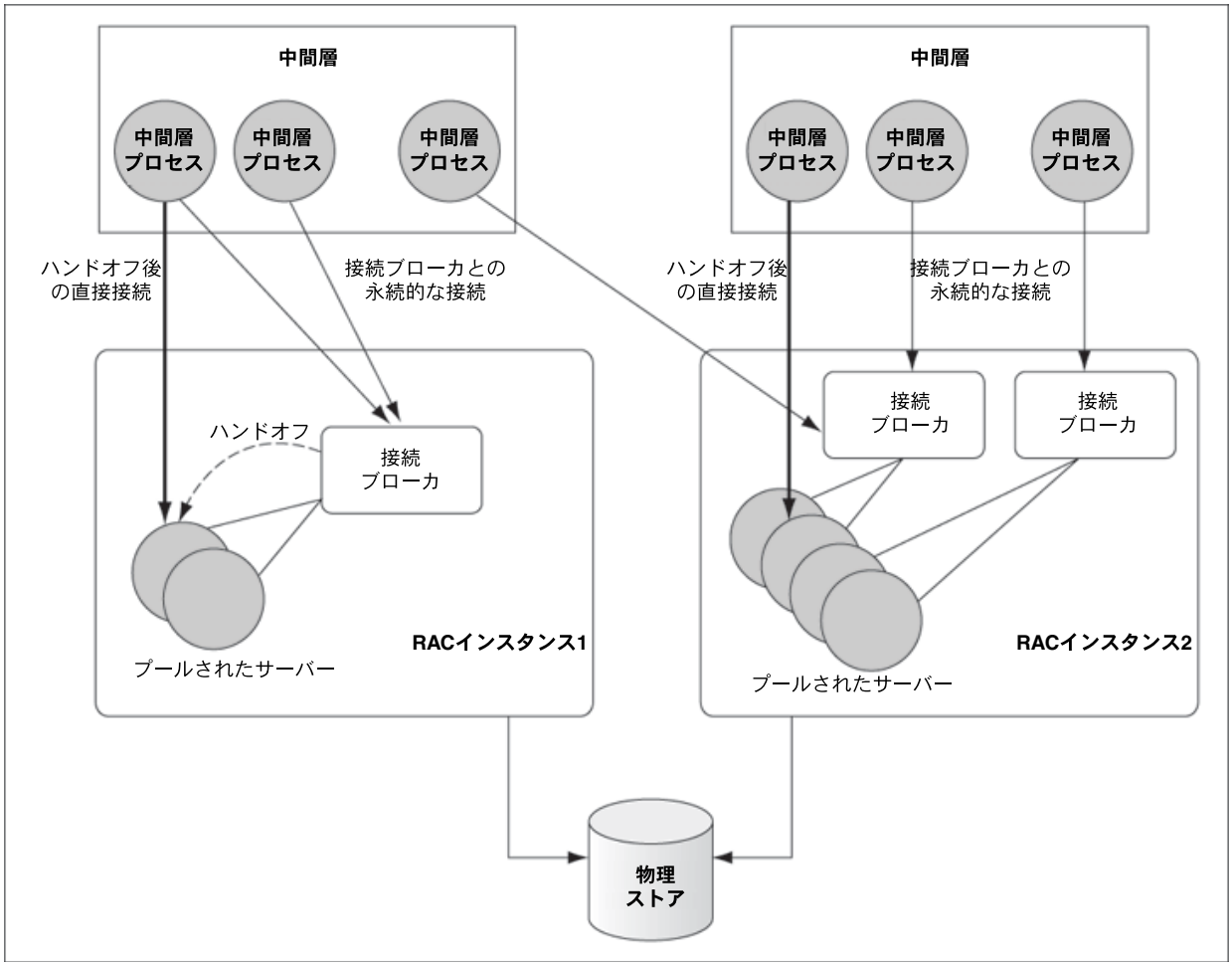
関連項目：

- 『Oracle Database 管理者ガイド』の「データベース常駐接続プーリングについて」
- 『Oracle Call Interface プログラマーズ・ガイド』

プールされたサーバーのモデルは、Oracle に接続するためのデフォルトとして使用されている専用モデルに密接に従っています。プールされたサーバー・モデルは、短時間サーバーを必要とする接続のすべてにサーバーを専用とすることによるオーバーヘッドを排除します。DRCP では、プールされたサーバーを取得して、同じような他の接続のために、自発的にそのサーバーを解放できます。プールされたサーバーは、接続によって取得されると、プールへ解放されるまでは、その接続に対しては基本的に専用サーバーへと変わります。データベース常駐接続プールから接続を取得したクライアントは、接続ブローカと呼ばれる Oracle バックグラウンド・プロセスに接続します。接続ブローカはプール機能を実装しており、プールされたサーバーをクライアント・プロセスからのインバウンド接続間で多重化します。

クライアントがバックエンド・データベース処理を実行する必要がある場合、接続ブローカはプールから、プールされたサーバーを選択してクライアントに割り当てます。その後クライアントは、要求が処理されるまでプールされたサーバーに直接接続します。サーバーでのクライアント要求の処理が終了すると、サーバーはプールに戻され、クライアント接続は接続ブローカ・プロセスにリストアされます。図 9-5 は、データベース常駐接続プーリングを示しています。

図 9-5 接続ブローカ・プロセスを使用して接続を処理する専用サーバー・プロセス



データベース常駐接続プーリングの使用

データベース常駐接続プーリングを使用すると、データベース側のメモリー不足を考慮せずに、中間層ハードウェアの自由自在な拡張が可能になります。これは、専用サーバー・プロセスの比較的小さなプールによって、より多くの中間層のプロセスを処理できるためです。

デフォルトの接続プール名は、SYS_DEFAULT_CONNECTION_POOL です。データベース常駐接続プーリングを使用するには、データベース管理者は、明示的にプールを起動する必要があります。次に例を示します。

SQL*Plus に SYSDBA としてログインします。

次のコマンドを実行します。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL('SYS_DEFAULT_CONNECTION_POOL');
```

注意： 現在、デフォルトの接続プールのみがサポートされています。

関連項目：『Oracle Database 管理者ガイド』の「データベース常駐接続プーリングの有効化」

共有プールに接続するには、データベース接続文字列中のサーバー・タイプが POOLED に設定されている必要があります。次に例を示します。

```
ServerPool = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=somehost)
       (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=testdb)(SERVER=POOLED)))
```

簡単な接続ネーミング・メソッドを使用して共有プールに接続することもできます。POOLED キーワードをデータベース・サービス名で使用する必要があります。次に例を示します。

```
CONNECT joeuser@myhost.example.com:1521/mydb:POOLED
Enter password: password
```

注意： この機能を簡単に説明するため、この例では、デプロイ済のシステムで通常使用されるパスワード管理テクニックを実行していません。本番環境では、Oracle Database のパスワード管理ガイドラインに従って、すべてのサンプル・アカウントを無効化してください。パスワード管理ガイドラインとその他のセキュリティ上の推奨事項は、『Oracle Database セキュリティ・ガイド』を参照してください。

この項の内容は、次のとおりです。

接続クラス

接続クラスは、アプリケーションに必要な接続タイプの論理名を定義します。2 人の異なるユーザーは、接続またはセッションをユーザー間で共有できません。たとえば、ユーザー HR 向けに作成されたセッションは、HR の以降の要求に対してのみ割り当てられます。接続クラスは、任意のユーザーのセッション間でさらに分離を可能にします。接続クラスは、同じデータベース・ユーザーとして接続している異なるアプリケーションが、アプリケーションに対応する論理名を使用して、セッションを識別できるようにします。これにより、DRCP では、特定の接続クラスに属すセッションが、接続クラスを越えて共有されることはありません。

セッション純粋度

セッション純粋度は、アプリケーションに新しいセッションが必要か（PURITY=NEW）、プールされたセッションを再利用するようにアプリケーション・ロジックで設定されているか（PURITY=SELF）を指定します。プールされたセッションの再利用が可能なアプリケーションの場合、要求された接続クラスを持つ空きセッションがそのアプリケーションに割り当てられます。

接続クラスおよびセッション純粋度は、DRCP 接続の属性としてクライアントで指定されます。接続クラスのデフォルト値は、*username*.SHARED です。デフォルトでは、純粋度が SELF である場合に同じ *username* を持つセッションが共有されます。

純粋度のデフォルト値は、NEW です。異なるアプリケーション・シナリオでは、デフォルトが異なる可能性があります。アプリケーションによる DRCP の使用方法の詳細は、それぞれのアプリケーション・マニュアルを参照してください。

> **関連項目：** 接続クラスおよびセッション純粋度の使用方法の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

プログラム・インタフェース

プログラム・インタフェースとは、データベース・アプリケーションと Oracle Database の間のソフトウェア・レイヤーです。プログラム・インタフェースには次のような機能があります。

- セキュリティ・バリアを実現し、クライアント・ユーザー・プロセスによる SGA への破壊的なアクセスを防ぎます。
- 通信メカニズムとして機能し、情報要求の書式設定、データの受渡し、エラーのトラップと通知を行います。
- 特に異機種コンピュータ間、または外部ユーザー・プログラム・データ型に対してデータを変換し解釈します。

Oracle コードはサーバーとして機能し、データ・ブロックから行をフェッチするなど、**アプリケーション**（クライアント）のためにデータベース・タスクを実行します。Oracle コードには、Oracle Database ソフトウェアとオペレーティング・システム固有のソフトウェアの両方のコード部分が含まれます。

この項の内容は、次のとおりです。

- プログラム・インタフェースの構造
- プログラム・インタフェース・ドライバ
- オペレーティング・システムの通信ソフトウェア

プログラム・インタフェースの構造

プログラム・インタフェースは、次の要素から構成されます。

- Oracle コール・インタフェース（OCI）または Oracle ランタイム・ライブラリ（SQLLIB）
- クライアント側またはユーザー側のプログラム・インタフェース
- 様々な **Oracle Net Services ドライバ**（プロトコル固有の通信ソフトウェア）
- オペレーティング・システムの通信ソフトウェア
- サーバーまたは Oracle Database 側のプログラム・インタフェース（OPI とも呼ばれます）

ユーザー側と Oracle Database 側のプログラム・インタフェースは、どちらもドライバのような仕組みで Oracle ソフトウェアを実行します。

Oracle Net Services は、クライアント・アプリケーション・プログラムと Oracle データベース・サーバーが通信ネットワーク内の別々のコンピュータに存在することを可能にするプログラム・インタフェース部分です。

プログラム・インタフェース・ドライバ

ドライバとは、通常はネットワークを介してデータを転送するソフトウェアの一部です。ドライバは、接続、切断、エラーの通知、エラーのテストなどの操作を実行します。ドライバは、通信プロトコルに固有であり、必ずデフォルトのドライバが存在します。

複数のドライバ（非同期または DECnet ドライバなど）をインストールし、その 1 つをデフォルト・ドライバとして選択しますが、各ユーザーは接続の時点で必要なドライバを指定すると、他のドライバを使用できます。プロセスが異なる場合は、異なるドライバを使用できます。1 つのプロセスは、異なる Oracle Net Services ドライバを使用した単一または複数のデータベース（ローカルまたはリモートのどちらでも）に同時接続できます。

関連項目：

- ドライバの選択、インストールおよび追加方法の詳細は、システムのインストレーションおよび構成ガイドを参照してください。
- Oracle Database にアクセスして実行時にドライバを選択する方法の詳細は、システムの Oracle Net Services マニュアルを参照してください。
- 『Oracle Database Net Services 管理者ガイド』

オペレーティング・システムの通信ソフトウェア

ユーザー側を Oracle Database 側のプログラム・インタフェースに接続する最下位層のソフトウェアは、ホスト・オペレーティング・システムによって提供される通信ソフトウェアです。たとえば、DECnet、TCP/IP、LU6.2、ASYNC などです。通信ソフトウェアは Oracle から提供されることもありますが、通常、ハードウェア・ベンダーまたはサード・パーティのソフトウェア・サプライヤから別途、購入します。

関連項目： 各システムの通信ソフトウェアの詳細は、オペレーティング・システム固有の Oracle Database マニュアルを参照してください。

10

アプリケーション・アーキテクチャ

この章では、アプリケーション・アーキテクチャについて定義し、分散処理環境で Oracle データベース・サーバーとデータベースのアプリケーションがどのように機能するかを説明します。このマニュアルの内容は、ほとんどすべてのタイプの Oracle Database システム環境に適用されます。

この章の内容は、次のとおりです。

クライアント / サーバー・アーキテクチャの概要

Oracle Database システムの環境では、データベース・アプリケーションとデータベースは 2 つの部分に分離されています。フロントエンド、すなわち**クライアント**部分と、バックエンド、すなわち**サーバー**部分です。このため、**クライアント / サーバー・アーキテクチャ**と名付けられています。クライアントでは、データベース情報にアクセスし、キーボード、スクリーンおよびマウスなどのポインティング・デバイスを使用してユーザーと対話する、データベース・アプリケーションが実行されます。サーバーでは、Oracle Database ソフトウェアが実行され、Oracle データベースへの同時実行の共有データ・アクセスに必要な機能が処理されます。

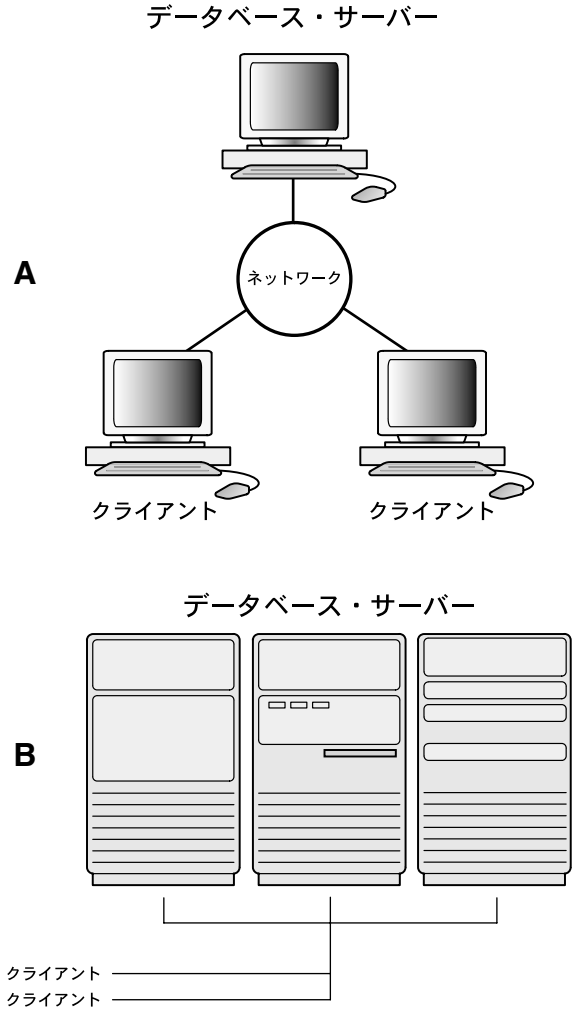
クライアント・アプリケーションと Oracle Database を同じコンピュータで実行することもできますが、クライアント部分とサーバー部分を別々のコンピュータで実行し、それらのコンピュータをネットワークで接続するほうがさらに効率的です。次の各項では、Oracle Database クライアント / サーバー・アーキテクチャの様々な形態について説明します。

分散処理とは、異なるシステムにある複数のプロセッサを使用して個々の作業を処理することです。図 10-1 に、Oracle Database システムでの分散処理の例を示します。

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

注意： この章の内容は、1 つのサーバー上に 1 つのデータベースがある環境に適用されます。**分散データベース**では、あるサーバー（Oracle Database）から別のサーバー上のデータベースへのアクセスが必要となる場合があります。

図 10-1 クライアント / サーバー・アーキテクチャと分散処理



分散処理環境での Oracle Database クライアント / サーバー・アーキテクチャには、次のような利点があります。

- クライアント・アプリケーションではデータ処理は実行されません。そのかわり、クライアント・アプリケーションは、ユーザーに入力を要求し、必要なデータをサーバーに要求し、そのデータを分析してクライアント・ワークステーションまたは端末の表示機能（グラフィックスやスプレッドシート）を使用して表示します。
- クライアント・アプリケーションはデータの物理的な位置に依存しません。データを他のデータベース・サーバーに移動したり分散しても、アプリケーションはほとんど、またはまったく変更することなく、機能を続行できます。
- Oracle Database では、基礎となるオペレーティング・システムのマルチタスキング機能と共有メモリー機能を活用できます。その結果、クライアント・アプリケーションに最高度の同時実行性、データ整合性およびパフォーマンスが提供されます。
- クライアント・ワークステーションや端末はデータの表示用に最適化でき（グラフィックスやマウスのサポート提供など）、サーバーはデータの処理と格納用に最適化できます（大容量のメモリーとディスク領域の搭載など）。
- ネットワーク化された環境では、安価なクライアント・ワークステーションを使用して、サーバーのリモート・データに効率的にアクセスできます。

- Oracle Database は、システムの拡張とともに必要に応じて**スケーリング**できます。複数のサーバーを追加して、データベース処理の負荷をネットワーク全体に分散させたり（**水平スケール**）、Oracle Database をミニコンピュータやメインフレームなどに移動して、より大規模なシステムのパフォーマンス上の利点を活用（**垂直スケール**）できます。どちらの場合も、Oracle Database にはシステム間での移植性があるため、すべてのデータとアプリケーションをほとんど、またはまったく変更することなく維持できます。
- ネットワーク化された環境の場合、共有データは、システムのすべてのコンピュータに格納されるのではなく、サーバーに格納されます。このため、同時アクセスの管理が簡単で効率的になります。
- ネットワーク化された環境では、クライアント・アプリケーションからサーバーにデータベース要求を送るときに SQL 文を使用できます。サーバーが受け取った SQL 文はそのサーバーで処理され、その結果がクライアント・アプリケーションに戻されます。ネットワークを介してやりとりされるのは要求と結果のみであるため、ネットワーク通信量は最小限ですみます。

関連項目：

- Oracle Net Services の詳細は、10-7 ページの「Oracle Net Services の概要」を参照してください。
- 分散データベースでのクライアントおよびサーバーの詳細は、『Oracle Database 管理者ガイド』を参照してください。

複数層アーキテクチャの概要

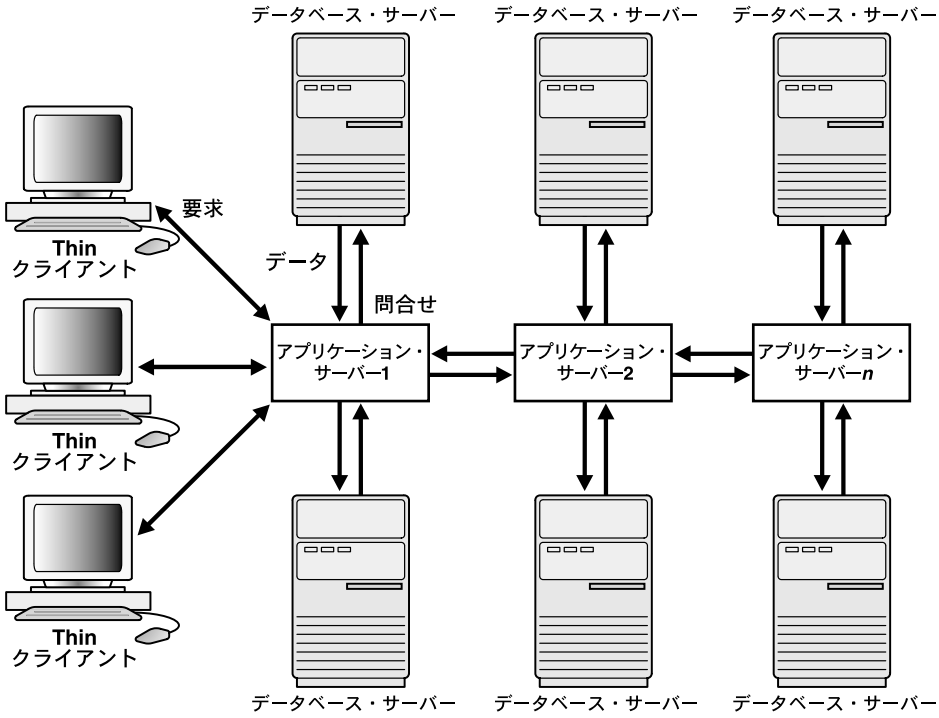
従来の複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントとデータベース・サーバー間のインタフェースとして機能します。このアーキテクチャは、インターネットの普及により重要度が増しています。

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

- Web ブラウザなど、クライアントの資格証明の検証
- データベース・サーバーへの接続
- 要求された操作の実行

図 10-2 は、複数層アーキテクチャの例を示しています。

図 10-2 複数層アーキテクチャ環境の例



この項の内容は、次のとおりです。

- クライアント
- アプリケーション・サーバー
- データベース・サーバー

クライアント

クライアントは、データベース・サーバー上で実行される操作について要求を出します。このクライアントは、Web ブラウザでも他のエンド・ユーザー・プロセスでもかまいません。複数層アーキテクチャでは、クライアントは 1 つ以上のアプリケーション・サーバーを介してデータベース・サーバーに接続します。

アプリケーション・サーバー

アプリケーション・サーバーは、クライアントにデータ・アクセスを提供します。また、クライアントと、さらに高度なセキュリティ・レベルを提供する 1 つ以上のデータベース・サーバーとの間のインタフェースとして機能します。さらに、クライアントのために一部の問合せ処理も実行するため、データベース・サーバーの負荷がある程度軽減されます。

アプリケーション・サーバーは、クライアントのためにデータベース・サーバー上で操作を実行するとき、そのクライアントの識別を想定します。アプリケーション・サーバーの権限は、クライアント操作中に不要な操作や望ましくない操作を実行できないように制限されます。

データベース・サーバー

データベース・サーバーは、クライアントにかわってアプリケーション・サーバーから要求されたデータを提供します。残りのすべての問合せ処理は、データベース・サーバーによって実行されます。

Oracle データベース・サーバーは、個々のクライアントのかわりにアプリケーション・サーバーが実行した操作や、アプリケーション自体のために実行した操作を監査できます。たとえば、クライアント操作がクライアントに表示する情報の要求であったり、アプリケーション・サーバーの操作がデータベース・サーバーへの接続要求であったりします。

関連項目： 第 20 章「データベース・セキュリティ」

Web サービス・プロバイダとしての Oracle Database

Oracle Database 11*g* から、Oracle Database は、従来の複数層またはサービス指向アーキテクチャ（SOA）環境で Web サービス・プロバイダとして機能できるようになりました。**SOA** は、アプリケーションの機能がサービスにカプセル化されている複数層アーキテクチャです。サービスは、ネットワーク上の相互運用可能なマシン間の相互作用をサポートするよう設計されています。動的に検出することが可能で、使用可能な関数およびコール・シーケンスで問い合せることもできます。

通常、SOA サービスは Web サービスとして実装されます。Web サービスには HTTP プロトコルを使用してアクセスでき、WSDL や SOAP など XML ベースの一連のオープン標準に基づいています。

XML DB の一部として実装される Oracle Database の Web サービス機能は、DBA が明示的に有効化する必要があります。アプリケーションは、データベース Web サービスを介して次のことを実行できます。

- SQL または XQuery 問合せの発行および XML での結果の受信
- スタンドアロン PL/SQL ファンクションの起動および結果の受信
- PL/SQL パッケージ・ファンクションの起動および結果の受信

複数層環境では、クライアントとアプリケーション・サーバーのどちらでも、データベース Web サービスを起動できます。

注意： データベース Web サービスを使用すると、アプリケーション・サーバーがなくても、アプリケーション環境に簡単に Web サービスを追加できます。ただし、Oracle Fusion Middleware などのアプリケーション・サーバーを介して Web サービスを起動すると、SOA 環境で、セキュリティ、スケーラビリティ、UDDI 登録および信頼できるメッセージ機能の面でさらに利点があります。データベース Web サービスは Oracle Fusion Middleware と簡単に統合できるため、SOA ソリューションの最適化に役立つ適切な方法です。SOA および Web サービスの詳細は、Oracle Fusion Middleware のマニュアルを参照してください。

関連項目： データベース Web サービスを有効化する方法および使用方法の詳細は、『Oracle XML DB 開発者ガイド』を参照してください。

Oracle Net Services の概要

Oracle Net Services により、分散、異機種間コンピューティング環境において、企業全体の接続性が確保されます。また、Oracle Net Services では、クライアント・アプリケーションから Oracle データベースへのネットワーク・セッションも使用可能です。

Oracle Net Services は、広範囲のネットワークでサポートされている通信プロトコルや Application Program Interface（API）を使用して、Oracle Database に分散データベースと分散処理の機能を提供します。

- 通信プロトコルとは、アプリケーションからネットワークにアクセスする方法およびデータをパケットに副分割してネットワークを介した送信を行う方法を規定する規則です。
- API は、ネットワークの場合、通信プロトコルを介してリモート・プロセス間通信を確立する手段を提供する、一連のサブルーチンです。

ネットワーク・セッションの確立後は、Oracle Net Services はクライアント・アプリケーションおよびデータベース・サーバーのためのデータ伝達手段として機能します。また、クライアント・アプリケーションとデータベース・サーバー間の接続確立と維持の他、これらの間でのメッセージ交換を受け持ちます。Oracle Net Services は、ネットワーク内の各コンピュータに配置されているのでこれらのジョブを実行できます。

Oracle Net Services により、位置の透過性、構成と管理の一元化および迅速なインストールと構成が実現されます。また、システム・リソースの最大化とパフォーマンスの改善が可能です。Oracle Database の**共有サーバー**・アーキテクチャによって、アプリケーションのスケーラビリティおよびデータベースへのクライアントの同時接続数が向上します。**Virtual Interface（VI）**プロトコルでは、メッセージ機能の負荷のほとんどを高速ネットワーク・ハードウェアに配置することで、より重要なタスクのために CPU を解放します。

> **関連項目：** これらの機能の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

この項の内容は、次のとおりです。

Oracle Net Services の動作

Oracle がサポートする業界標準のネットワーク・プロトコルでは、データベース・サーバー上で稼働する Oracle Database プロセスおよびネットワークの他のコンピュータ上で稼働する Oracle Database アプリケーションのユーザー・プロセスとの間のインタフェースが提供されます。

Oracle Database プロトコルでは、Oracle アプリケーションのインタフェースから SQL 文が取り出され、それらをパッケージ化してから、サポートされている業界標準の高レベルのプロトコルまたはプログラム・インタフェースを介して Oracle Database に送信します。また、Oracle Database から応答を取り込んでパッケージ化し、同じ高レベルの通信メカニズムを介してアプリケーションに送り返します。これらの機能はすべて、ネットワーク・オペレーティング・システムからは独立して実行されます。

Oracle Database を実行するオペレーティング・システムによっては、データベース・サーバーの Oracle Net Services ソフトウェアにドライバ・ソフトウェアが含まれており、ドライバ・ソフトウェアによって追加の Oracle Database バックグラウンド・プロセスが起動される場合があります。

> **関連項目：** Oracle Net Services の動作の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

リスナー

インスタンスの起動時に、**リスナー・プロセス**によって Oracle Database への通信経路が確立されます。ユーザー・プロセスが接続要求を出すと、リスナーは共有サーバー・ディスパッチャ・プロセスと専用サーバー・プロセスのどちらを使用するかを判断し、適切な接続を確立します。

また、リスナー・プロセスは、データベース間の通信経路も確立します。Oracle Real Application Clusters など、単一のコンピュータ上で複数のデータベースやインスタンスが稼働している場合は、**サービス名**を使用すると、インスタンスを同じコンピュータ上の他のリスナーに自動的に登録できます。サービス名では複数のインスタンスを識別でき、インスタンスは複数のサービスに属することができます。サービスに接続するクライアントは、必要なインスタンスを指定する必要がありません。

サービス情報の登録

動的サービス登録により、複数のデータベースやインスタンスの管理に伴うオーバーヘッドが低減します。リスナーがクライアントの要求を送るサービスの情報はリスナーに登録されます。サービス情報は、**サービス登録**と呼ばれる機能を介してリスナーに動的に登録することも、`listener.ora` ファイルに静的に構成することもできます。

サービス登録では、インスタンスのバックグラウンド・プロセスである **PMON プロセス**を利用して、インスタンス情報、インスタンスと**共有サーバー**・ディスパッチャの現在の状態と負荷をリスナーに登録します。登録済情報により、リスナーはクライアントの接続要求を適切なサービス・ハンドラに送ることができます。サービス登録では、`listener.ora` ファイル内での構成は必要ありません。

初期化パラメータ `SERVICE_NAMES` では、インスタンスが属するデータベース・サービスが識別されます。起動時に、各インスタンスは、同じサービスに属している他のインスタンスのリスナーに登録します。データベース操作中に、各サービスのインスタンスは CPU の使用と現行の接続カウントに関する情報を、同じサービスのすべてのリスナーに渡します。これにより、動的なロード・バランシング機能と接続フェイルオーバー機能が使用可能になります。

関連項目：

- 9-14 ページの「共有サーバー・アーキテクチャ」
- サーバー・プロセスの詳細は、9-18 ページの「専用サーバー構成」を参照してください。
- リスナーの詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。
- Oracle Real Application Clusters におけるインスタンスの登録とクライアント / サーバー接続の詳細は、それぞれのプラットフォームに対応した Oracle Real Application Clusters インストール・ガイドおよび『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

11

Oracle Database ユーティリティ

この章では、Oracle Database ユーティリティでのデータ転送、データ・メンテナンスおよびデータベース管理について説明します。

この章の内容は、次のとおりです。

Oracle Database ユーティリティの概要

Oracle Database ユーティリティでは、次のタスクを実行できます。

- データ・ポンプ・エクスポート / インポートを使用した、データベース間でのデータおよびメタデータの高速移動
- Metadata API を使用した、データベース・オブジェクト用メタデータの完全な表現の抽出と操作
- データ・ポンプ API を使用した、サイトに関するデータおよびメタデータの全体または一部のデータベース間における移動
- SQL*Loader を使用してオペレーティング・システム・ファイルから、または外部表を使用して外部ソースから Oracle Database 表にデータをロード
- ADR コマンド・インタプリタ（ADRCI）を使用した Oracle Database の診断データの管理
- LogMiner との SQL インタフェースを介した REDO ログ・ファイルの問合せ
- DBVERIFY を使用した、オフライン（バックアップなど）データベースまたはデータファイルの物理データ構造の整合性チェックの実行
- DBNEWID ユーティリティを使用した、運用データベースの内部データベース識別子（DBID）とデータベース名（DBNAME）のメンテナンス

関連項目： この章で説明されているすべてのユーティリティの詳細は、『Oracle Database ユーティリティ』を参照してください。

データ・ポンプ・エクスポート / インポートの概要

Oracle のデータ・ポンプ・テクノロジにより、データベース間でデータとメタデータを超高速で移動できます。このテクノロジは、Oracle Database のデータ移動ユーティリティであるデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポートのベースとなっています。

データ・ポンプでは、ジョブでデータとメタデータのサブセットを移動する必要があるかどうかを指定できます。これは、エクスポートおよびインポートのパラメータを介して実装されるデータ・フィルタとメタデータ・フィルタを使用して実行されます。

この項の内容は、次のとおりです。

データ・ポンプ・エクスポート

データ・ポンプ・エクスポート（以降、読みやすいようにエクスポートと呼びます）は、データとメタデータをダンプ・ファイル・セットと呼ばれる一連のオペレーティング・システム・ファイルにアンロードするユーティリティです。ダンプ・ファイル・セットは、データ・ポンプ・インポート・ユーティリティを使用して他のシステムに移動してロードできます。

ダンプ・ファイル・セットは、表データ、データベース・オブジェクトのメタデータおよび制御情報を含む 1 つ以上のディスク・ファイルから構成されています。各ファイルは固有のバイナリ形式で記述され、データ・ポンプ・インポートでのみ読取りが可能です。インポート操作中に、データ・ポンプ・インポート・ユーティリティはこれらのファイルを使用して、ダンプ・ファイル・セット内で各データベース・オブジェクトを検索します。

データ・ポンプ・インポート

データ・ポンプ・インポート（以降、読みやすいようにインポートと呼びます）は、エクスポートされたダンプ・ファイル・セットをターゲット・システムにロードするユーティリティです。ダンプ・ファイル・セットは、表データ、データベース・オブジェクトのメタデータおよび制御情報を含む 1 つ以上のディスク・ファイルから構成されています。各ファイルは固有のバイナリ形式で記述されます。

また、インポートを使用すると、ソース・データベースから中間ファイルなしでターゲット・データベースに直接ロードできます。これにより、エクスポート操作とインポート操作を同時に実行でき、全体の所要時間が短縮されます。これをネットワーク・インポートと呼びます。

インポートでは、実際に SQL を実行せずに、インポート・ジョブが実行される SQL DDL をすべて確認することもできます。これは、インポートの `SQLFILE` パラメータを介して実装されます。

データ・ポンプ API の概要

データ・ポンプ API は、サイトに関するデータとメタデータの全体または一部をデータベース間で移動するための高速メカニズムを提供します。データ・ポンプ API を使用するには、`DBMS_DATAPUMP` PL/SQL パッケージに用意されているプロシージャを使用します。データ・ポンプ・エクスポートおよびデータ・ポンプ・インポート・ユーティリティは、データ・ポンプ API をベースとしています。

関連項目：

- データ・ポンプ API の機能の詳細は、『Oracle Database ユーティリティ』を参照してください。
- `DBMS_DATAPUMP` パッケージの説明は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Metadata API の概要

Metadata API は、次の手段を提供します。

- XML 形式によるオブジェクトのメタデータの検索
- SQL DDL への変換など、様々な方法による XML の変換
- 検索により抽出されたオブジェクトを再作成するための XML の発行

Metadata API を使用するには、`DBMS_METADATA` PL/SQL パッケージに用意されているプロシージャを使用します。Metadata API の目的上、データベース内のすべてのエンティティは、1 つのオブジェクト型に属するオブジェクトとしてモデル化されます。たとえば、表 `scott.emp` はオブジェクトで、そのオブジェクト型は `TABLE` です。オブジェクトのメタデータをフェッチする場合は、オブジェクト型を指定する必要があります。

関連項目：

- Metadata API の使用方法の詳細は、『Oracle Database ユーティリティ』を参照してください。
- `DBMS_DATAPUMP` パッケージの説明は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

SQL*Loader の概要

SQL*Loader では、データが外部ファイルから Oracle データベースの表にロードされます。データファイル内のデータの書式にほとんど制限のない、強力なデータ解析エンジンです。SQL*Loader を使用すると次の操作ができます。

- 同じロード・セッション中に複数のデータファイルからデータをロードします。
- 同じロード・セッション中に複数の表にデータをロードします。
- データのキャラクタ・セットを指定します。
- データを選択的にロードします（レコードをその値に基づいてロードできます）。
- ロード前に SQL 関数を使用してデータを操作します。
- 指定の列に一意のシーケンシャル・キー値を生成します。
- オペレーティング・システムのファイル・システムを使用してデータファイルにアクセスします。
- データをディスク、テープまたは名前付きパイプからロードします。
- トラブルシューティングに役立つ洗練されたエラー・レポートを生成します。
- 任意の複合オブジェクト・リレーショナル・データをロードします。
- LOB およびコレクションのロードにセカンダリ・データファイルを使用します。
- 従来型パスによるロードまたはダイレクト・パス・ロードを使用します。従来型パスによるロードは大きな柔軟性を備えていますが、ダイレクト・パス・ロードは優れたパフォーマンスを発揮します。

典型的な SQL*Loader セッションは、SQL*Loader の動作を制御する制御ファイルと 1 つ以上のデータファイルを入力として使用します。SQL*Loader の出力は、Oracle データベース（データのロード場所）、ログ・ファイル、不良ファイル、および存在する場合は廃棄ファイルです。

> **関連項目：** LogMiner の詳細は、『Oracle Database ユーティリティ』を参照してください。

外部表の概要

外部表機能は、既存の SQL*Loader 機能を補完するものです。この機能により、外部ソースのデータに対して、データベース内の表にあるデータのようにアクセスできます。外部表は `ORACLE_DATAPUMP` アクセス・ドライバを使用して記述できます。外部表に対しては、データ操作言語（DML）の操作も索引作成も許可されません。したがって、ステージング表への追加の索引付けを必要とするデータ・ロード状況には、SQL*Loader の方が適しています。

外部表機能を使用するには、プラットフォーム上にあるデータファイルのファイル形式とレコード形式に関してある程度の知識が必要です。また、外部表を作成し、それに対する問合せを実行するには、SQL についても十分な知識が必要になります。

> **関連項目：** 5-12 ページの「外部表」

LogMiner の概要

Oracle LogMiner を使用すると、SQL インタフェースを介して REDO ログ・ファイルを問い合せることができます。ユーザー・データまたはデータベース・ディクショナリに対して行われたすべての変更は、Oracle DatabaseREDO ログ・ファイルに記録されます。したがって、REDO ログ・ファイルには、リカバリ操作の実行に必要な情報がすべて含まれています。

LogMiner 機能は、コマンドライン・インタフェースまたは Oracle LogMiner Viewer の Graphical User Interface（GUI）を介して入手できます。LogMiner Viewer は、Oracle Enterprise Manager に付属しています。

REDO ログ・ファイルに含まれるデータの使用方法を次に示します。

- アプリケーション・レベルで発生したエラーなど、データベースの論理的な破損の開始時点を特定します。これにより、データベースを破損直前の状態までリストアできます。
- ユーザー・エラーを検出し、可能な場合は訂正します。これは論理的な破損よりも実行する可能性が高い使用例です。ユーザー・エラーには、`WHERE` 句の値が正しくないために間違った行を削除する、行を不正な値で更新する、間違った索引を削除するなどがあります。
- トランザクション・レベルでファイングレイン・リカバリを実行するために必要な操作を判断します。既存の依存性をよく理解し、考慮している場合は、表ベースの UNDO 操作を実行して一連の変更をロールバックできます。
- 傾向分析を介してパフォーマンスのチューニングと容量計画を実行します。更新および挿入頻度の最も高い表を判断できます。この情報からディスク・アクセス統計の履歴が得られ、チューニングに活用できます。
- 監査後処理を実行します。REDO ログ・ファイルには、データベース上で実行された DML 文と DDL 文、その実行順序および実行者の追跡に必要な全情報が含まれています。

DBVERIFY ユーティリティの概要

DBVERIFY は、物理データ構造の整合性チェックを実行する外部コマンドライン・ユーティリティです。オフライン・データベース、オンライン・データベースおよびバックアップ・ファイルに対して使用できます。DBVERIFY を使用するのは、主として、リストア前にバックアップ・データベース（またはデータファイル）が有効かどうかを確認する必要がある場合で、データ破損エラーが発生した場合には診断に役立つ情報としても使用できます。

DBVERIFY はオフライン・データベースに対して実行できるため、整合性チェックは非常に高速です。

DBVERIFY でのチェック対象は、キャッシュ管理ブロック（つまりデータ・ブロック）のみです。DBVERIFY はデータファイル専用であるため、制御ファイルや REDO ログには機能しません。

DBVERIFY には 2 つのコマンドライン・インタフェースが用意されています。最初のインタフェースでは、チェック対象となる 1 つのデータファイルのディスク・ブロックを指定します。第 2 のインタフェースでは、チェック対象となるセグメントを指定します。

DBNEWID ユーティリティの概要

DBNEWID は、運用データベースの内部の一意データベース識別子（DBID）とデータベース名（DBNAME）を変更できるデータベース・ユーティリティです。このユーティリティを使用すると、次の情報を変更できます。

- データベースの DBID のみ
- データベースの DBNAME のみ
- データベースの DBNAME と DBID の両方

したがって、データベースのコピーを手動で作成し、制御ファイルを再作成して新規の DBNAME と DBID を指定できます。また、シード・データベースと手動でコピーしたデータベースを、同じ Recovery Manager リポジトリにともに登録できます。

ADRCI: ADR コマンド・インタプリタ

ADRCI はコマンドライン・ツールで、Oracle Database 11*g* に導入された障害診断性インフラストラクチャの一部です。ADRCI では次の操作ができます。

- 自動診断リポジトリ（ADR）内の診断データの表示
- Oracle サポートへインシデントおよび問題の情報を送信するための zip ファイルへのパッケージ化

診断データには、インシデントや問題の説明、トレース・ファイル、ダンプ、状態モニター・レポート、アラート・ログ・エントリなどが含まれます。

ADRCI には豊富なコマンド・セットが用意されており、対話モードで使用することもスクリプト内で使用することもできます。また、ADRCI では、SQL*Plus で SQL および PL/SQL コマンドのスクリプトを実行するのと同じ方法で、ADRCI コマンドのスクリプトを実行できます。

> **関連項目：** ADRCI の詳細は、『Oracle Database ユーティリティ』を参照してください。

12

データベースとインスタンスの起動と停止

この章では、Oracle データベース・インスタンスおよびデータベースの起動と停止に関連する手順を説明します。

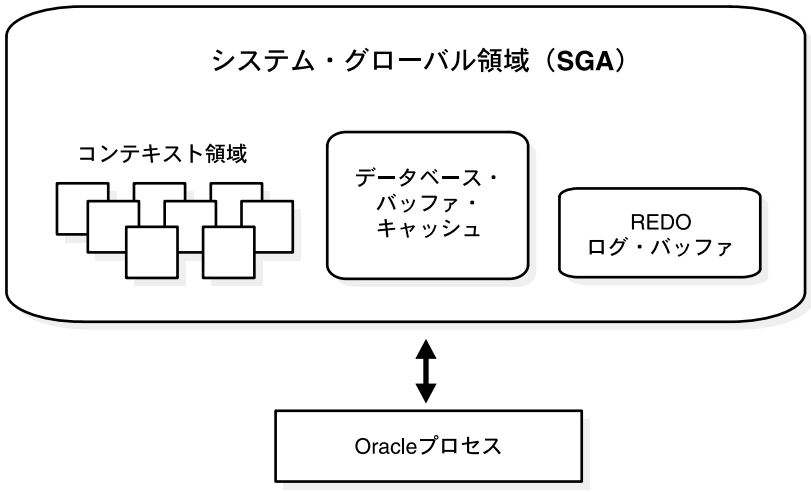
この章の内容は、次のとおりです。

Oracle インスタンスの概要

稼働中のすべての Oracle Database は、Oracle データベース・インスタンスに対応付けられます。データベース・サーバー上で（コンピュータの種類に関係なく）データベースを起動すると、Oracle Database によってシステム・グローバル領域（SGA）と呼ばれるメモリー領域が割り当てられ、1 つ以上の Oracle Database プロセスが開始されます。この SGA と Oracle Database プロセスを組み合せて、Oracle **インスタンス**と呼びます。インスタンスのメモリーとプロセスは、対応付けられたデータベースのデータを効果的に管理し、データベースを使用する 1 人以上のユーザーのために機能します。

図 12-1 に Oracle データベース・インスタンスを示します。

図 12-1 Oracle インスタンス



関連項目：

- 第 8 章「メモリー・アーキテクチャ」
- 第 9 章「プロセス・アーキテクチャ」

この項の内容は、次のとおりです。

- インスタンスとデータベース
- 管理者権限での接続
- 初期化パラメータ・ファイルとサーバー・パラメータ・ファイル

インスタンスとデータベース

インスタンスを起動した後、Oracle Database は指定されたデータベースにインスタンスを対応付けます。これは**マウントされたデータベース**です。これでデータベースをオープンできるようになり、許可されたユーザーがアクセスできるようになります。

複数のインスタンスを同時に同じコンピュータ上で実行し、それぞれ専用の物理データベースにアクセスさせることができます。大規模クラスタ・システムでは、Oracle Real Application Clusters により、複数のインスタンスが 1 つのデータベースをマウントできます。

データベース管理者のみが、インスタンスを起動してデータベースをオープンできます。データベースがオープンされている場合、データベース管理者はそのデータベースを停止してクローズできます。データベースが**クローズ**されている場合、ユーザーはそのデータベース内のデータにアクセスできません。

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

管理者権限での接続

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

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

SYSDBA 権限で接続すると、ユーザーは SYS が所有するスキーマ内に置かれます。SYSOPER として接続すると、そのユーザーはパブリック・スキーマ内に置かれます。SYSOPER 権限は、SYSDBA 権限のサブセットです。

関連項目：

- 各オペレーティング・システムにおける管理者権限の機能の詳細は、オペレーティング・システム固有の Oracle Database マニュアルを参照してください。
- データベース管理者を認証するときのパスワード・ファイルと認証方式の詳細は、第 20 章「データベース・セキュリティ」を参照してください。

初期化パラメータ・ファイルとサーバー・パラメータ・ファイル

インスタンスを起動するには、Oracle Database は**初期化パラメータ・ファイル**または**サーバー・パラメータ・ファイル**を読み取る必要があります。これらのファイルには、インスタンスおよびデータベースの構成パラメータのリストが格納されます。Oracle Database では従来より、初期化パラメータはテキスト形式の初期化パラメータ・ファイルに格納されます。また、サーバー側のバイナリ形式のサーバー・パラメータ・ファイル（SPFILE）に保持するように選択することもできます。

サーバー・パラメータ・ファイルに格納されている初期化パラメータは永続的です。つまり、インスタンスの実行中に行われたパラメータへの変更は、インスタンスを停止して起動しても残ります。

初期化パラメータは、基本と拡張という 2 つのグループに分かれています。ほとんどの場合、妥当なパフォーマンスが得るには、基本パラメータのみを設定およびチューニングする必要があります。ただし最適なパフォーマンスを得るために、拡張パラメータの変更が必要になる場合があります。

ほとんどの初期化パラメータは、次のグループのいずれかに属しています。

- ファイルなどの名前を指定するパラメータ。
- 最大数などの制限を設定するパラメータ。
- SGA のサイズなどの容量に影響するパラメータ。これを**可変パラメータ**と呼びます。

初期化パラメータは主に、Oracle Database に次のことを指定します。

- インスタンスを起動するデータベースの名前
- SGA のメモリー構造に使用するメモリーの容量
- いっぱいになった REDO ログ・ファイルの処理方法
- データベースの制御ファイルの名前と位置
- データベースの UNDO 表領域の名前

> **関連項目：**『Oracle Database 管理者ガイド』

この項の内容は、次のとおりです。

サーバー・パラメータ・ファイルと Hardware Assisted Resilient Data

オラクル社の Hardware Assisted Resilient Data（HARD）initiative は、データ破損を未然に防ぐよう設計された包括的なプログラムです。Oracle Database では、ストレージ・デバイス内に Oracle のデータ検証アルゴリズムを実装することにより、破損データが永続ストレージに書き込まれるのを防ぐことができます。Oracle Database 11*g* から、HARD に準拠したストレージ・システムで使用可能な新しい形式でサーバー・パラメータ・ファイルを作成できます。データベースは、新旧どちらの形式でも、サーバー・パラメータ・ファイルを読み取りおよび書き込みできます。

> **関連項目：** サーバー・パラメータ・ファイルの作成および管理方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

パラメータ値の変更方法

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

一部のパラメータは、インスタンスの実行中に ALTER SESSION 文または ALTER SYSTEM 文を使用して動的に変更できます。サーバー・パラメータ・ファイル（SPFILE）を使用していないかぎり、ALTER SYSTEM 文を使用して行われる変更は、現行インスタンスにのみ有効になります。次回のインスタンス起動時に変更内容を確認するには、テキスト初期化パラメータ・ファイルを手動で更新する必要があります。

サーバー・パラメータ・ファイルの使用時には、ALTER SYSTEM SET 文を使用してメモリーまたはディスク内（あるいはその両方）のパラメータ値を変更できます。データベースにより、新しい値と古い値（存在する場合）がアラート・ログに出力されます。基本的なパラメータを変更すると、不正な値がサーバー・パラメータ・ファイルに書き込まれるのを防ぐため、予防策として、データベースにより検証ステップが実行されます。

Oracle Database は、データベース・ソフトウェアで提供される最初の初期化パラメータ・ファイル、または Database Configuration Assistant により作成される最初の初期化パラメータ・ファイルに値を提供します。構成、オプションおよびデータベースのチューニング計画に応じて、これらの Oracle 提供の初期化パラメータを編集し、他のパラメータを追加できます。該当する初期化パラメータが初期化パラメータ・ファイルに特に組み込まれていない場合には、Oracle Database によりデフォルトが指定されます。データベースを初めて作成する場合は、パラメータ値の変更数を最小限に抑えることをお薦めします。

関連項目：

- 初期化パラメータの詳細とサーバー・パラメータ・ファイルの使用方法は、『Oracle Database 管理者ガイド』を参照してください。
- すべての初期化パラメータの説明は、『Oracle Database リファレンス』を参照してください。

インスタンスとデータベースの起動の概要

Oracle データベースを起動して、システム全体で使用可能にするには、次の 3 つの操作を行います。

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

データベース管理者は、SQL*Plus の STARTUP 文や Enterprise Manager を使用してこれらの手順を実行できます。

関連項目：『Oracle Database 2 日でデータベース管理者』

この項の内容は、次のとおりです。

インスタンスの起動方法

Oracle Database は、インスタンスを起動するときに、サーバー・パラメータ・ファイル（SPFILE）または初期化パラメータ・ファイルを読み取って、初期化パラメータの値を決定します。その後、データベース情報用の共有メモリー領域である SGA を割り当てて、バックグラウンド・プロセスを作成します。この時点では、これらのメモリー構造とプロセスにデータベースは対応付けられていません。

インスタンスが起動されると、明示的に設定されたすべてのパラメータは、パラメータの有効な構文でアラート・ログに書き込まれます。必要な場合には、このテキストをコピーして新しいパラメータ・ファイルに貼り付け、インスタンスを再起動できます。

関連項目：

- SGA の詳細は、第 8 章「メモリー・アーキテクチャ」を参照してください。
- バックグラウンド・プロセスの詳細は、第 9 章「プロセス・アーキテクチャ」を参照してください。

この項の内容は、次のとおりです。

- インスタンスの起動の制限モード
- 異常な状況での強制起動

インスタンスの起動の制限モード

インスタンスを制限モードで起動したり、既存のインスタンスを制限モードに変更できます。これにより、接続は `RESTRICTED SESSION` システム権限を付与されているユーザーのみに制限されます。

異常な状況での強制起動

異常な状況では、直前のインスタンスが正常に停止していないことがあります。たとえば、インスタンスのプロセスのいずれかが正常に終了していないことがあります。そのような状況では、次回インスタンスを通常起動するときにデータベースからエラーが戻されます。この問題を解決するには、前のインスタンスからの残留 Oracle Database プロセスをすべて終了してから、新しいインスタンスを起動する必要があります。

データベースのマウント方法

インスタンスは、データベースをマウントして、データベースをそのインスタンスに関連付けます。データベースをマウントすると、そのインスタンスはデータベース制御ファイルを見つけてオープンします。制御ファイルは、インスタンスの起動に使用されるパラメータ・ファイルの `CONTROL_FILES` 初期化パラメータで指定します。Oracle Database は、制御ファイルを読み取って、データベースのデータファイルと REDO ログ・ファイルの名前を取得します。

マウント直後のデータベースはまだクローズされているため、データベース管理者のみがそのデータベースにアクセスできます。データベース管理者は、特定のメンテナンス操作を完了するまでの間、データベースをクローズしたままにしておくことができます。ただし、データベースに対して通常の操作を行うことはまだできません。

この項の内容は、次のとおりです。

- Oracle Real Application Clusters でのデータベースのマウント方法
- クローン・データベースのマウント方法

Oracle Real Application Clusters でのデータベースのマウント方法

Oracle Database で、複数のインスタンスが同じデータベースを同時にマウントすることが可能な場合、データベース管理者は、複数のインスタンスがデータベースを利用できるように、CLUSTER_DATABASE 初期化パラメータを使用できます。CLUSTER_DATABASE パラメータのデフォルト値は false です。Oracle RAC をサポートしない Oracle Database のバージョンでは、CLUSTER_DATABASE を false にのみ設定できます。

データベースをマウントする最初のインスタンスの CLUSTER_DATABASE パラメータが false の場合は、最初のインスタンスのみがデータベースをマウントできます。この CLUSTER_DATABASE パラメータが true に設定されていると、他のインスタンスも CLUSTER_DATABASE パラメータが true に設定されている状態でデータベースをマウントできます。データベースをマウントできるインスタンスの数は、データベースの作成時に指定した最大数によって決まります。

関連項目：

- 『Oracle Real Application Clusters インストレーションおよび構成』
- 『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』

1 つのデータベースでの複数のインスタンスの使用方法の詳細は、前述のマニュアルを参照してください。

クローン・データベースのマウント方法

クローン・データベースは、表領域のポイント・イン・タイム・リカバリに使用できる、データベースの特殊コピーです。表領域のポイント・イン・タイム・リカバリを実行する場合は、クローン・データベースをマウントし、表領域を目的の時点までリカバリします。次に、クローンからプライマリ・データベースにメタデータをエクスポートし、リカバリされた表領域からデータファイルをコピーします。

関連項目： クローン・データベースと表領域のポイント・イン・タイム・リカバリの詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

データベースのオープン時の動作

マウントされたデータベースをオープンすると、そのデータベースで通常のデータベース操作を実行できるようになります。有効なユーザーは、オープン状態のデータベースに接続して、データベースの情報にアクセスできます。通常、データベース管理者は、データベースをオープンして一般に使用できる状態にします。

Oracle Database では、データベースをオープンするときに、オンライン・データファイルと REDO ログ・ファイルをオープンします。データベースを前回停止したときに表領域がオフラインであった場合は、データベースを再オープンしたとき、その表領域と対応するデータファイルはオフラインのままです。

データベースをオープンするとき、データファイルまたは REDO ログ・ファイルが欠如していると、Oracle Database からエラーが戻されます。そのデータベースをオープンするには、破損または欠落したファイルのバックアップからリカバリする必要があります。

関連項目： オフライン表領域をオープンする方法は、3-13 ページの「オンライン表領域とオフライン表領域」を参照してください。

この項の内容は、次のとおりです。

クラッシュ・リカバリおよびインスタンス・リカバリ

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（LRU）アルゴリズムを使用して、必要な場合にのみディスクに書き込まれます。このアルゴリズムがデータベース・ライター・プロセスで使用され、データベース・バッファがデータファイルに書き込まれる方法により、データファイルにはコミットされていないトランザクションにより変更されたデータ・ブロックや、コミット済トランザクションからの変更が欠落しているデータ・ブロックが含まれている場合があります。

インスタンス障害が発生すると、次の 2 つの問題が発生する可能性があります。

- トランザクションにより変更されたデータ・ブロックは、コミット時にデータファイルに書き込まれず、REDO ログにのみ表示される場合があります。そのため、REDO ログにはリカバリ時にデータベースに再適用する必要のある変更が含まれます。
- ロールフォワード・フェーズの後、データファイルには障害時にコミットされていなかった変更が含まれている可能性があります。これらの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証するためにロールバックする必要があります。これらの変更は、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・フェーズ中に取り込まれたものです。

データベース管理者によるインスタンスの終了や停電などの理由から、前回のデータベースのクローズが正常に行われていない場合、Oracle Database では、データベースを再びオープンするときに、インスタンス・リカバリまたはクラッシュ・リカバリが自動的に実行されます。

クラッシュ・リカバリは、シングル・インスタンス・データベースの障害時、または Oracle Real Application Clusters データベースの全インスタンスの障害時のリカバリに使用します。インスタンス・リカバリでは、障害の発生していないインスタンスが Oracle Real Application Clusters データベース内で障害インスタンスをリカバリする場合について説明します。

クラッシュ・リカバリとインスタンス・リカバリを行う目的は、終了したインスタンスのキャッシュ内にあるデータ・ブロックの変更をリストアし、オープン状態になっていた REDO スレッドをクローズすることです。インスタンス・リカバリおよびクラッシュ・リカバリでは、オンライン REDO ログ・ファイルと現行のオンライン・データファイルのみが使用されます。Oracle Database では、終了したインスタンスの **REDO スレッド**も同時にリカバリされます。

暗号化された表領域を持つデータベースをリカバリする場合（たとえば、`SHUTDOWN ABORT` やデータベース・インスタンスを停止させるような致命的なエラーの後）、リカバリ・プロセスがデータ・ブロックと REDO を復号化できるように、データベースのマウント後、データベースをオープンする前に、Oracle Wallet をオープンする必要があります。

クラッシュ・リカバリとインスタンス・リカバリは、2 つの異なる操作を伴います。1 つはオンライン REDO レコードに含まれるコミット済トランザクションとコミットされていないトランザクションの両方を適用し、現行のオンライン・データファイルをロールフォワードすることです。もう 1 つは、コミットされていないトランザクションで行われた変更を元の状態までロールバックすることです。

クラッシュ・リカバリとインスタンス・リカバリには、次の共通する特性があります。

- 現行の（障害または `SHUTDOWN ABORT` の後もディスクに残っている）オンライン・データファイルを使用して変更を REDO します。
- 使用するのはオンライン REDO ログのみで、アーカイブ・ログを使用する必要はありません。
- リカバリ時間は、終了したインスタンスの数、最後のチェックポイント以降に終了した各 REDO スレッドで生成された REDO の量、REDO ログ・ファイルの数とサイズ、チェックポイント間隔およびパラレル・リカバリ設定などのユーザー構成可能な要素により制御されます。

Oracle Database では、次の 2 つの場合にこのリカバリが自動的に実行されます。

- シングル・インスタンス・データベースまたは Oracle RAC データベースの全インスタンスの障害後に、最初にデータベースをオープンするとき（クラッシュ・リカバリ）。
- Oracle RAC 構成に含まれる全インスタンスではなく一部のインスタンスに障害が発生した時点（インスタンス・リカバリ）。リカバリは、構成内で正常に稼働しているインスタンスにより自動的に実行されます。

重要な点は、クラッシュ・リカバリでもインスタンス・リカバリでも、Oracle Database によって自動的に REDO が適用されることです。REDO ログを提供するためにユーザーが介入する必要はありません。ただし、データベース・サーバーでパラメータを設定して、インスタンスの存続期間とクラッシュ・リカバリのパフォーマンスをチューニングできます。また、インスタンス・リカバリのロールフォワード・フェーズとロールバック・フェーズを別々にチューニングすることもできます。

このジレンマを解決し、システム障害から正常にリカバリするために、Oracle Database では通常、2 つの個別のステップが使用されます。1 つは REDO ログを使用したロールフォワード（キャッシュ・リカバリ）、もう 1 つはロールバック・セグメントまたは UNDO セグメントを使用したロールバック（トランザクション・リカバリ）です。

この項の内容は、次のとおりです。

- キャッシュ・リカバリ
- トランザクション・リカバリ

キャッシュ・リカバリ このジレンマを解決し、システム障害から正常にリカバリするために、Oracle Database では通常、2 つの個別のステップが使用されます。1 つは REDO ログを使用したロールフォワード（キャッシュ・リカバリ）、もう 1 つはロールバック・セグメントまたは UNDO セグメントを使用したロールバック（トランザクション・リカバリ）です。

オンライン REDO ログは、データ、索引およびロールバック・セグメントなどのデータベース・ブロックに対する変更をすべて記録する、オペレーティング・システム・ファイルの集合です。変更がコミット済かどうかは問いません。Oracle Database ブロックに対する変更は、すべてオンライン・ログに記録されます。

インスタンス障害またはメディア障害からリカバリする最初のステップを、**キャッシュ・リカバリ**または**ロールフォワード**と呼びます。このステップでは、REDO ログに記録された変更すべてをデータファイルに再適用する必要があります。REDO ログにはロールバック・データも記録されているため、ロール・フォワードでは対応するロールバック・セグメントも再生成されます。

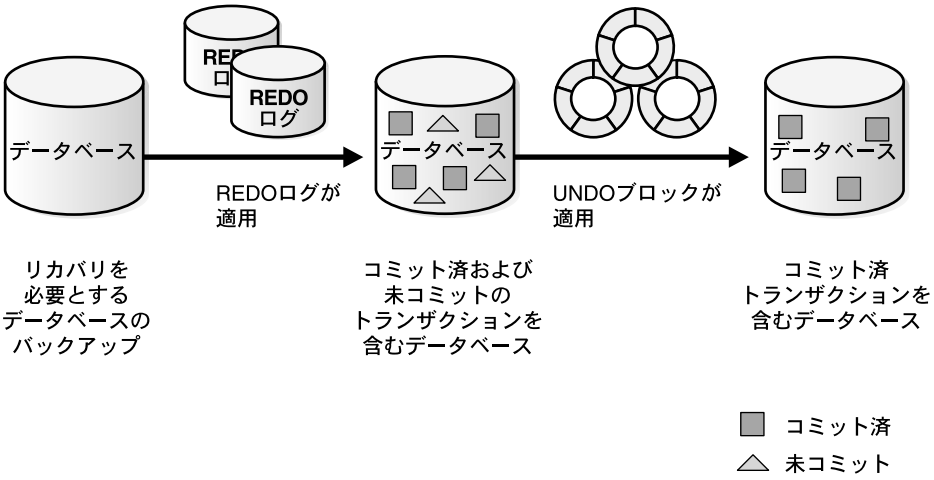
ロールフォワードでは、必要に応じて多数の REDO ログ・ファイルをたどり、データベースを後の時点まで進行させます。通常、ロールフォワードにはオンライン REDO ログ・ファイルが含まれ（インスタンス・リカバリまたはメディア・リカバリ）、アーカイブ REDO ログ・ファイルが含まれる場合（メディア・リカバリのみ）もあります。

ロールフォワード後のデータ・ブロックには、コミット済の変更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ていたか、REDO ログに記録されてキャッシュ・リカバリ時に取り込まれた、コミットされていない変更も含まれることがあります。

トランザクション・リカバリ ロールフォワード後は、コミットされなかった変更を取り消す必要があります。Oracle Database は UNDO ブロックを適用して、障害発生前に書き込まれたか、キャッシュ・リカバリ中に REDO の適用により取り込まれたデータ・ブロック内の、コミットされていない変更をロールバックします。このプロセスを**ロールバック**または**トランザクション・リカバリ**と呼びます。

図 12-2 に、ロールフォワードとロールバックを示します。この 2 つは、どのタイプのシステム障害からのリカバリにも必要なステップです。

図 12-2 リカバリの基本ステップ : ロールフォワードとロールバック



Oracle Database は、必要に応じて同時に複数のトランザクションをロールバックできます。障害時点でアクティブになっていたトランザクションすべてに、終了マークが付けられます。終了したトランザクションが SMON によりロールバックされるのを待たずに、新規トランザクションでは阻止しているトランザクション自体をリカバリし、必要な行ロックが取得されます。

関連項目 :

- インスタンス・リカバリ・メカニズムとインスタンスおよびクラッシュ・リカバリのチューニング指示は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- UNDO の詳細は、2-16 ページの「UNDO セグメントおよび自動 UNDO 管理の概要」を参照してください。

UNDO 領域の取得と管理

データベースをオープンするときに、インスタンスにより UNDO 表領域の取得が試行されます。複数の UNDO 表領域が存在する場合には、使用する UNDO 表領域を UNDO_TABLESPACE 初期化パラメータで指定できます。このパラメータが指定されていない場合、データベース内で使用可能な最初の UNDO 表領域が選択されます。

関連項目 : UNDO 表領域の詳細は、2-16 ページの「UNDO セグメントおよび自動 UNDO 管理の概要」を参照してください。

インダウト分散トランザクションの解決

場合によっては、データベースが異常にクローズし、1 つ以上の分散トランザクションが**インダウト**（コミットもロールバックもされない状態）になることがあります。データベースを再オープンし、リカバリが完了すると、RECO バックグラウンド・プロセスがただちに自動的に実行されて、インダウト分散トランザクションが矛盾のないように解決されます。

関連項目 : 分散トランザクションの障害からのリカバリの詳細は、『Oracle Database 管理者ガイド』を参照してください。

データベースの読取り専用モードでのオープン

データベースは、ユーザー・トランザクションによってデータが変更されるのを防ぐために、読取り専用モードでオープンできます。読取り専用モードでは、データベース・アクセスは読取り専用トランザクションに限定され、データファイルや REDO ログ・ファイルに書き込むことはできません。

制御ファイル、オペレーティング・システムの監査証跡、トレース・ファイルおよびアラート・ログなど、他のファイルへのディスク書込みは、読取り専用モードでも実行できます。データベースを読取り専用モードでオープンしても、ソート操作用の一時表領域には影響しません。ただし、データベースを読取り専用モードでオープンしている間は、永続表領域はオフライン化できません。また、読取り専用モードでは、ジョブ・キューを使用できません。

データベース・リカバリや、REDO データを生成しないでデータベースの状態を変更する操作には、制限はありません。たとえば、読取り専用モードでは、次の操作を実行できます。

- データファイルのオフラインとオンラインを切り替えることができます。
- オフラインのデータファイルと表領域をリカバリできます。
- 制御ファイルは、データベースの状態の更新に引き続き使用できます。

読取り専用モードの利点の 1 つは、**スタンバイ・データベース**が一時レポート用データベースとして使用できることです。

> **関連項目：** データベースを読取り専用モードでオープンする方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

読取り専用データベースの制限

- アプリケーションで、読取り専用データベースが実行されている間は、データベース・オブジェクトを作成することはできません。たとえば、グローバル一時表などのデータベース表に対して行の挿入、削除、更新またはマージをする場合にデータベース・オブジェクトが作成されます。データベースの順序を操作する場合にデータベース・オブジェクトが作成されます。行のロック、`EXPLAIN PLAN` の実行または DDL の実行時にデータベース・オブジェクトが作成されます。`DBMS_SCHEDULER` などの Oracle 提供 PL/SQL パッケージにあるファンクションやプロシージャの多くは、データベース・オブジェクトを作成します。アプリケーションで、これらのファンクションおよびプロシージャが呼び出されるか、前述の操作が実行されると、データベース・オブジェクトが書き込まれ、読取り専用ではなくなります。
- 読取り専用データベースを実行する場合は、データベース・リンクを含む進行中のトランザクションをコミットまたはロールバックした後にのみ、別のデータベース・リンクを使用できます。これは、現在のトランザクションが読取り専用のとき、最初のデータベース・リンクで汎用 `SELECT` 文を実行する場合にも該当します。
- 読取り専用データベースでは、PL/SQL ストアド・プロシージャをコンパイルまたは再コンパイルできません。リモート・プロシージャ・コールによる無効な PL/SQL を最小限に抑えるには、読取り専用データベースに対してリモート・プロシージャ・コールを行うセッションで、`REMOTE_DEPENDENCIES_MODE=SIGNATURE` を使用します。
- 読取り専用データベースでは、データベースで 1 度もコールされていないリモート・プロシージャは、それが読取り専用のリモート・プロシージャであっても、起動することはできません。この制限は、匿名の PL/SQL ブロックや SQL 文におけるリモート・プロシージャ・コールに適用されます。この場合、リモート・プロシージャ・コールをストアド・プロシージャに格納するか、データベースが読取り専用になる前にそのデータベースでリモート・プロシージャを起動できます。

データベースとインスタンスの停止の概要

データベースとそれに対応付けられたインスタンスを停止するには、次の 3 つの操作を行います。

1. データベースをクローズします。
2. データベースをアンマウントします。
3. インスタンスを停止します。

データベース管理者は、これらの手順を Enterprise Manager を使用して実行できます。Oracle Database では、インスタンスの停止時にこの 3 つの手順が自動的に実行されます。

関連項目：『Oracle Database 2 日でデータベース管理者』

この項の内容は、次のとおりです。

データベースのクローズ

データベースをクローズすると、Oracle Database により SGA にあるすべてのデータベース・データとリカバリのためのデータが、それぞれデータファイルと REDO ログ・ファイルに書き込まれます。次に、Oracle Database がすべてのオンライン・データファイルと REDO ログ・ファイルをクローズします。(オフライン表領域のオフライン・データファイルは、すでにクローズされています。オフラインになっていた表領域とそのデータファイルは、この後でデータベースを再オープンしたとき、それぞれオフラインでクローズされたままになっています。) この時点でデータベースはクローズされているため、通常の操作は実行できません。データベースがクローズされても、まだマウントされていれば、制御ファイルはオープンされたままになります。

インスタンスの終了によるデータベースのクローズ

万一の非常時には、オープン状態のデータベースのインスタンスを終了させて、データベースをクローズし、ただちに完全に停止させることができます。SGA のバッファにあるすべてのデータをデータファイルと REDO ログ・ファイルに書き込む操作が省略されるため、このプロセスは高速に実行されます。次にデータベースを再オープンするときに必要なリカバリは、Oracle Database で自動的に実行されます。

注意： データベースがオープンされているときにシステム障害や停電が発生すると、インスタンスは事実上終了するため、データベースを再オープンした時点でリカバリが実行されます。

データベースのアンマウント

データベースがクローズされると、Oracle Database はデータベースをアンマウントしてインスタンスから切り離します。この時点ではインスタンスはコンピュータのメモリーに残っています。

データベースをアンマウントすると、Oracle Database では、データベースの制御ファイルがクローズされます。

インスタンスの停止

データベース停止の最後の操作は、インスタンスの停止です。インスタンスを停止すると、SGA がメモリーから削除され、バックグラウンド・プロセスが停止します。

インスタンスの異常停止

異常な状況では、すべてのメモリー構造がメモリーから削除されない、またはバックグラウンド・プロセスの 1 つが終了されないなど、インスタンスが正常に停止しないことがあります。前のインスタンスの一部が残っていると、その後にインスタンスを起動しようとしても、ほとんどの場合に失敗します。このような状況では、データベース管理者は、まず残っている前のインスタンスを削除してから新しいインスタンスを起動するか、SQL*Plus または Enterprise Manager で `SHUTDOWN ABORT` 文を発行することで、新しいインスタンスの起動を強制実行できます。

> **関連項目：** インスタンスとデータベースの起動および停止の詳細は、『Oracle Database 管理者ガイド』を参照してください。

第 III 部

Oracle Database の機能

第 III 部では、Oracle Database の中核となる機能分野について説明します。

第 III 部には、次の章が含まれています。

13

データの同時実行性と整合性

この章では、マルチユーザー・データベース環境での Oracle Database によるデータ整合性の維持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整合性の概要

シングル・ユーザーのデータベースでは、他のユーザーが同時に同じデータを変更するということがないため、ユーザーは何も心配せずにデータベース内のデータを変更できます。ただし、マルチユーザー・データベースでは、複数のトランザクション内の文によって、同じデータが同時に更新される可能性があります。同時に実行される複数のトランザクションでは、意味のある一貫した結果を出すことが必要です。したがって、マルチユーザー・データベースではデータの同時実行性と整合性の制御が重要になります。

- **データ同時実行性**とは、多数のユーザーが同時にデータにアクセスできることを意味します。
- **データ整合性**は、各ユーザーにデータの一貫したビューが表示され、その中にユーザー自身のトランザクションや他のユーザーのトランザクションによる参照可能な変更も含まれることを意味します。

複数のトランザクションが同時に実行される場合のトランザクションの首尾一貫した動作を記述するために、データベース研究者は**シリアライズ可能性**と呼ばれるトランザクションの分離モデルを定義してきました。トランザクション動作のシリアライズ可能なモードでは、複数のトランザクションは、同時にではなく 1 つずつ（シリアルに）実行されるように見えます。

一般に、この程度までのトランザクションの分離が望ましいとされますが、このモードで多数のアプリケーションを実行すると、アプリケーションのスループットがかなり低下する可能性があります。同時に実行される各トランザクションの完全な分離とは、あるトランザクションが問合せを実行している表に対して、他のトランザクションが挿入を実行できない状態を指します。つまり、現実には、トランザクションの完全な分離とパフォーマンスとの間の妥協点を考慮する必要があります。

Oracle Database の 2 つの分離レベルは、整合性を維持し、高いパフォーマンスを実現する操作モードをアプリケーション開発者に提供します。

> **関連項目：** データベースに対応付けられたビジネス・ルールを規定するデータ整合性の詳細は、第 21 章「データ整合性」を参照してください。

この項の内容は、次のとおりです。

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（SQL-92）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 つの回避可能な現象とは、次のものです。

- 内容を保証しない読取り : 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取る現象
- 非リピータブル・リード（ファジー読取り）: あるトランザクションが以前に読み取ったデータをもう 1 度読み取ったときに、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
- 仮読取り : 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

SQL-92 では、それぞれの分離レベルで実行されるトランザクションで起こり得る現象という観点で 4 つの分離レベルを定義しています。表 13-1 に、各分離レベルを示します。

表 13-1 分離レベル別による回避可能な読取り現象

分離レベル	内容を保証しない読取り	非リピータブル・リード	仮読取り
非コミット読取り	あり	あり	あり
コミット読取り	なし	あり	あり
リピータブル・リード	なし	なし	あり
シリアライズ可能	なし	なし	なし

Oracle Database で使用できる分離レベルは、SQL-92 の一部ではない読取り専用モードと、コミット読取りおよびシリアライズ可能です。デフォルトはコミット読取りです。

> **関連項目：** コミット読取りとシリアライズ可能な分離レベルの詳細は、13-3 ページの「Oracle Database データの同時実行性と整合性の管理方法」を参照してください。

ロックのメカニズムの概要

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間で破壊的な相互作用が起きないようにするメカニズムです。

リソースには、次の 2 つの一般的なタイプのオブジェクトが含まれます。

- ユーザー・オブジェクト。たとえば、表と行（構造体とデータ）
- ユーザーには見えないシステム・オブジェクト。たとえば、メモリー内の共有データ構造やデータ・ディクショナリ行

> **関連項目：** ロックの詳細は、13-14 ページの「Oracle Database のデータ・ロックの方法」を参照してください。

Oracle Database データの同時実行性と整合性の管理方法

Oracle Database は、マルチバージョン一貫性モデルや、様々なタイプのロックおよびトランザクションを使用することによって、マルチユーザー環境でのデータ整合性を維持します。この項の内容は、次のとおりです。

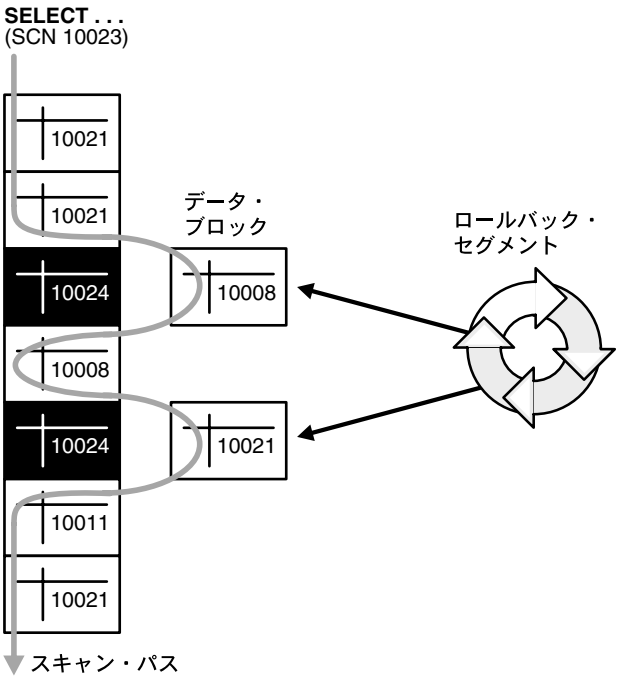
- マルチバージョン同時実行性制御
- 文レベルの読取り一貫性
- トランザクション・レベルの読取り一貫性
- Oracle Real Application Clusters における読取り一貫性
- Oracle Database の分離レベル
- コミット読取り分離とシリアライズ可能な分離の比較
- 分離レベルの選択

マルチバージョン同時実行性制御

Oracle Database では、ある問合せで参照されるデータのすべてが同一時点でのデータになるように、問合せに対して自動的に読取り一貫性が提供されます（**文レベルの読取り一貫性**）。Oracle Database の読取り一貫性は、1 つのトランザクション内のすべての問合せに対しても提供できます（**トランザクション・レベルの読取り一貫性**）。

Oracle Database 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・セグメントには、コミットされていないトランザクションまたは最近コミットされたトランザクションによって変更されたデータの元の値が含まれます。Oracle Database では、図 13-1 に示すように、ロールバック・セグメントのデータを使用して文レベルの読取り一貫性を提供します。

図 13-1 トランザクションと読取り一貫性



問合せが実行段階に入ると、現行のシステム変更番号（SCN）が決定されます。図 13-1 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・ブロックを読み取ると、監視されたシステム変更番号（SCN）が書き込まれたブロックが使用されます。変更されたデータを含むブロック（もっと後の SCN）があると、それらはロールバック・セグメント内のデータに基づいて再構成され、問合せには再構成されたデータが戻されます。そのため、問合せを実行するたびに、問合せの実行開始時点で記録された SCN に関してコミット済のすべてのデータが戻されます。問合せの実行中に発生する他のトランザクションの変更は参照されません。このようにして、各問合せに対して一貫性のあるデータが戻されることが保証されます。

文レベルの読取り一貫性

Oracle Database では、常に**文レベル**の読取り一貫性が保たれています。これによって、1 つの問合せによって戻されるすべてのデータは、その問合せが開始された時点のものであることが保証されます。そのため、問合せは、内容が保証されないデータも、その問合せの実行中にコミットされたトランザクションによって変更されたデータもまったく参照しません。問合せの実行中にその問合せで参照できるのは、その問合せが開始される前にコミットされたデータのみです。問合せは、文の実行の開始後にコミットされた変更は参照しません。

あらゆる問合せに対して一貫した結果セットが保証され、これによって、データ整合性が保証されます。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決定します。

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。

注意： SELECT 構文のリストにファンクションが含まれる場合、データベースにより、親 SQL レベルではなく PL/SQL ファンクション・コード内で実行される SQL の文レベルにおいて、文レベルの読取り一貫性が適用されます。たとえば、ファンクションは、別のユーザーがデータを変更およびコミットした表にアクセスする場合があります。ファンクション内の SELECT の実行ごとに、読取り一貫性を備えた新しいスナップショットが設定されます。

トランザクション・レベルの読取り一貫性

Oracle Database では、**トランザクション・レベルの読取り一貫性**を保つこともできます。シリアライズ可能なモード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。したがって、シリアライズ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ます。トランザクション・レベルの読取り一貫性によってリピータブル・リードが実現され、問合せで実体のないデータが検索されることもありません。

Oracle Real Application Clusters における読取り一貫性

Oracle Real Application Clusters（Oracle RAC）では、読取り一貫性のあるブロックのイメージをインスタンス間で転送する際に、キャッシュ・フュージョンというキャッシュ間のブロック転送メカニズムを使用します。Oracle RAC ではデータ・ブロックに対するリモート要求に対応するため、待機時間の短い高速相互接続を使用してこれを行います。

関連項目：『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』

Oracle Database の分離レベル

Oracle Database は、表 13-2 に示すトランザクション分離レベルを提供します。

表 13-2 トランザクション分離レベル

分離レベル	説明
コミット読取り	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 Database 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）データが読み取られることはありません。 Oracle Database では、問合せで読み取ったデータを他のトランザクションで変更できるため、問合せを 2 回実行する間に最初の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、非リピータブル・リードと仮読取り（実体のない読取り）の現象の両方が起こり得ます。
シリアライズ可能	シリアライズ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の変更と、そのトランザクション自身が INSERT 文、UPDATE 文および DELETE 文で実行した変更のみが参照されます。シリアライズ可能トランザクションでは、非リピータブル・リードや仮読取りの現象は起きません。
読取り専用	読取り専用トランザクションでは、そのトランザクションを開始した時点でコミット済の変更のみが参照され、INSERT 文、UPDATE 文および DELETE 文は使用できません。

この項の内容は、次のとおりです。

- 分離レベルの設定
- コミット読取り分離
- シリアライズ可能な分離

分離レベルの設定

アプリケーション設計者、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文のいずれかを使用して設定できます。

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ ONLY;
```

各トランザクションを SET TRANSACTION 文で開始する場合のネットワーキングおよび処理のコストを節約するために、ALTER SESSION 文を使用して、後続のすべてのトランザクションのトランザクション分離レベルを設定することもできます。

```
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;

ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
```

関連項目： これらの SQL 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

コミット読取り分離

Oracle Database のデフォルト分離レベルは、コミット読取りです。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。Oracle Database では、問合せごとに、それぞれ独自のマテリアライズド・ビュー時間を基準として実行されます。このため、ある問合せを複数回実行する間に非リピータブル・リードと仮読取り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。コミット読取り分離は、競合するトランザクションの数が少ない環境に適した分離レベルです。

シリアライズ可能な分離

シリアライズ可能な分離は、次のような環境に適しています。

- 大規模データベースを使用し、短いトランザクションでごく少数の行しか更新しない環境
- 2 つの同時実行トランザクションが同一の行を更新するようなことが比較的少ない環境
- 比較的長時間にわたって実行されるトランザクションが主に読取り専用である環境

シリアライズ可能な分離では、同時実行トランザクションが実行できる変更は、トランザクションが順に 1 つずつ実行される場合に実行できるデータベース変更のみです。具体的には、Oracle Database では、シリアライズ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、シリアライズ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決定できる場合のみです。

この決定の効率をよくするために、Oracle Database では、データ・ブロック内に格納されている制御情報、つまりブロック内の行のうち、どの行にコミットされた変更が含まれていて、どの行にコミットされていない変更が含まれているかを示す情報が使用されます。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。

Oracle Database では、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決定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更新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。この設定により、Oracle Database では、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ために十分な記憶域が各ブロックに割り当てられます。

シリアライズ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、Oracle Database では次のようなエラーが生成されます。

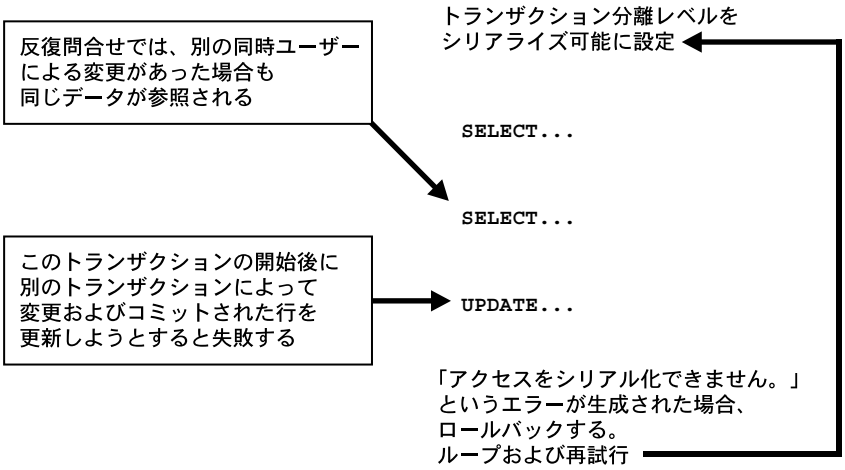
```
ORA-08177: Cannot serialize access for this transaction
```

シリアライズ可能トランザクションが失敗し、「アクセスをシリアル化できません。」というエラーが出た場合、アプリケーションは次のいずれかのアクションを実行できます。

- 実行した作業をそのポイントまでコミットします。
- 追加の（異なる）文を実行します（トランザクション内で以前に設定したセーブポイントまでロールバックした後など）。
- トランザクション全体を取り消します。

図 13-2 に、「アクセスをシリアル化できません。」というエラーになったトランザクションをロールバックして再試行するアプリケーションの例を示します。

図 13-2 シリアライズ可能トランザクションの失敗



コミット読取り分離とシリアライズ可能な分離の比較

Oracle Database では、アプリケーション開発者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。コミット読取りとシリアライズ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 Database の読取り一貫性によるマルチバージョン同時実行性制御モデルと、排他的な行レベルのロッキングが実装されることによって競合が削減されます。これらの分離レベルは、実際のアプリケーションのデプロイメントを考慮して設計されています。

この項の内容は、次のとおりです。

- トランザクション集合の一貫性
- 行レベル・ロック
- 参照整合性
- 分散トランザクション

トランザクション集合の一貫性

Oracle Database におけるコミット読取りとシリアライズ可能な分離レベルを理解するため、次のような使用例を考えます。データベースの表の集合（またはデータの任意の集合）があり、それらの表内の行を特定の順序で何度か読み取り、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）のどの読取りでも、コミット済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。一部の読取り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、他の読取り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 Database では、コミット読取りモードで実行されるトランザクションには文単位でのトランザクション集合の一貫性が提供されます。シリアライズ可能なモードでは、トランザクション単位でのトランザクション集合の一貫性が提供されます。

表 13-3 に、Oracle Database でのコミット読取りトランザクションとシリアライズ可能トランザクションの主な違いをまとめます。

表 13-3 コミット読取りトランザクションとシリアライズ可能トランザクション

動作	コミット読取り	シリアライズ可能
内容を保証しない書込み	なし	なし
内容を保証しない読取り	なし	なし
非リピータブル・リード	あり	なし
仮読取り	あり	なし
ANSI/ISO SQL 92 への準拠	あり	あり
マテリアライズド・ビュー読取り時間	文	トランザクション
トランザクション集合の一貫性	文レベル	トランザクション・レベル
行レベル・ロック	あり	あり
読取りが書込みをブロックするか	なし	なし
書込みが読取りをブロックするか	なし	なし
別の行の書込みが書込みをブロックするか	なし	なし
同一行の書込みが書込みをブロックするか	あり	あり
阻止しているトランザクションを待機するか	あり	あり
「アクセスをシリアル化できません。」が発生するか	なし	あり
阻止しているトランザクションの終了後にエラーが発生するか	なし	なし
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	なし	あり

行レベル・ロック

コミット読取りトランザクションとシリアライズ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取り消されてロックが解除されるまで、待ち状態になります。この最初のトランザクションがロールバックした場合、どの分離モードでも、待ち状態のトランザクションは最初のトランザクションが存在しなかったかのように、以前にロックされた行を変更できるようになります。

ただし、阻止している最初のトランザクションがコミットされてロックが解除された場合、コミット読取りトランザクションは、目的の更新処理を続行します。また、シリアライズ可能トランザクションの場合には、そのシリアライズ可能トランザクションの開始後になされた変更を他のトランザクションがコミットしているため、「アクセスをシリアル化できません。」エラーを出して失敗します。

参照整合性

Oracle Database は、読取り一貫性トランザクションでもシリアライズ可能トランザクションでも読取りロックを使用しないため、あるトランザクションで読み取った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、読み取った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かぎり、シリアライズ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われることがあります。

関連項目： 参照整合性とシリアライズ可能トランザクションの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

注意： Oracle Real Application Clusters では、コミット読取りトランザクションとシリアライズ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、一部のノードのみがコミットされることがないように、複数の物理データベースが 2 フェーズ・コミットによって保護され、1 つのトランザクションによって複数の物理データベースのデータが更新されます。そのような環境では、Oracle かどうかを問わず、**シリアライズ可能**トランザクションに関与するすべてのサーバーで、シリアライズ可能な分離モードがサポートされている必要があります。

シリアライズ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シリアライズ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。トランザクションを取り消して再試行できるのは、リモート・サーバーがシリアライズ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、**コミット読取り**トランザクションでは、シリアライズ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。

関連項目：『Oracle Database 管理者ガイド』

分離レベルの選択

アプリケーション設計者およびアプリケーション開発者は、アプリケーションのコーディング以外にアプリケーションのパフォーマンスと一貫性のニーズに基づいて分離レベルを選択する必要があります。

多数の同時実行ユーザーが次々にトランザクションを送る環境では、トランザクションの予想到着率および応答時間の要求の面から、トランザクションのパフォーマンス要件を評価する必要があります。高いパフォーマンスが必要な環境では、分離レベルを選択する際に一貫性と同時実行性とのバランスを考慮する必要がよくあります。

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取りによっては書込みがブロックされないという点を考慮する必要があります。

Oracle Database の 2 つの分離モードは、行レベル・ロックと Oracle Database のマルチバージョン同時実行性制御システムとの組合せによって、高水準の一貫性、同時実行性およびパフォーマンスを提供します。Oracle Database では、読取り側と書込み側が互いに処理をブロックしません。そのため、問合せで一貫性のあるデータが参照される一方で、コミット読取りとシリアライズ可能などちらの分離レベルでも、コミットされてい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。

この項の内容は、次のとおりです。

コミット読取り分離

多くのアプリケーションでは、コミット読取りが最適な分離レベルです。コミット読取り分離では、同時実行性が大幅に向上する一方で、一部のトランザクションでは、仮読取りや非リピータブル・リードのために、一貫性のない結果が生じる危険性が多少高くなります。

トランザクションの到着率が高く、高いパフォーマンスが求められる多くの環境では、シリアライズ可能な分離レベルによって実現されるよりも大きなスループットと高速な応答時間が必要になる場合があります。サポートするユーザーが少数で、トランザクション到着率が非常に低い環境では、仮読取りおよび非リピータブル・リードによって間違った結果が生じる危険性が非常に低くなります。コミット読取り分離は、いずれの環境にも適しています。

Oracle Database のコミット読取り分離では、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、コミット読取り分離で十分に対応できます。

コミット読取り分離モードでは、アプリケーションのロジックで「アクセスをシリアル化できません。」というエラーをトラップしたり、処理をロールバックしてトランザクションを再起動する必要はありません。大部分のアプリケーションでは、同じ問合せを 2 回繰り返して発行するという機能的必要があるトランザクションはごく少数のため、仮読取りおよび非リピータブル・リードの防止は重要ではありません。したがって、前述のようなエラー・チェックや再試行のコードを各トランザクションで記述する必要を避けるために、多くの開発者はコミット読取りを選択します。

シリアライズ可能な分離

Oracle Database のシリアライズ可能な分離は、2 つの同時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、長時間にわたって実行されるトランザクションが主に読取り専用である環境に適しています。この分離モードが最も適しているのは、大規模なデータベースを使用し、短いトランザクションでごく少数の行のみが更新される環境です。

シリアライズ可能な分離モードは、仮読取りと非リピータブル・リードを防止することである程度まで一貫性を向上できるため、読取り / 書込みトランザクションが 1 つの問合せを複数回実行する場合に重要になることがあります。

他の処理系のシリアライズ可能な分離では書込みのみでなく読取りについてもブロックがロックされるのに対し、Oracle Database では非ブロック化問合せときめ細かい行レベル・ロックが提供され、それらによって読取り / 書込みの競合が少なくなります。読取り / 書込みの競合が多く発生するアプリケーションでは、Oracle Database のシリアライズ可能な分離は、他のシステムより大幅に高いスループットを提供します。このため、Oracle Database 上ではシリアライズ可能な分離に適していても、他のシステムではシリアライズ可能な分離に適さないアプリケーションもあります。

Oracle Database のシリアライズ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ため、読取り / 書込みトランザクションで複数の一貫した問合せを発行する必要がある場合は、この分離レベルが適しています。シリアライズ可能なモードでは、`READ ONLY` トランザクションで実現される一貫性が確保される一方で、`INSERT`、`UPDATE` および `DELETE` トランザクションも実行できるため、サマリー・データを生成してそのデータをデータベースに格納する報告書作成アプリケーションでは、シリアライズ可能なモードを使用できます。

注意： 副問合せを持つ DML 文を含むトランザクションは、読取り一貫性を保証するためにシリアライズ可能な分離を使用する必要があります。

アプリケーション開発者がシリアライズ可能トランザクションをコーディングする場合には、「アクセスをシリアル化できません。」のエラーのチェックとトランザクションの取消しおよび再試行のために、追加の作業が必要になります。他のデータベース管理システムでデッドロックを管理する際にも、同様の追加コーディングが必要です。社内標準を遵守する場合や、複数のデータベース管理システム上で実行するアプリケーションを作成する場合には、シリアライズ可能なモードに対応したトランザクションの設計が必要なことがあります。シリアライズの可能性エラーをチェックして、再試行するトランザクションは、Oracle Database のコミット読取りモード、つまりシリアライズの可能性エラーを生成しないモードで使用できます。

シリアライズ可能なモードにあまり適さないのは、大量の短い更新トランザクションでアクセスするのと同じ行を、比較的長いトランザクションで更新する環境です。このような環境では、長時間実行のトランザクションが行を最初に変更するという可能性は低いため、頻繁なロールバックが必要となり、作業が無駄になります。従来型の読取りをロックしたシリアライズ可能なモードの即時実行であり、この環境には適していません。長時間実行されるトランザクションは、読取りトランザクションであっても、短い更新トランザクションの処理を阻止したり、逆に短いトランザクションが長時間実行のトランザクションの処理を阻止する場合があるためです。

アプリケーション開発者は、シリアライズ可能なモードを使用するときはトランザクションのロールバックと再試行に要するコストを考慮する必要があります。デッドロックが頻繁に発生する読取りロックのシステムと同様に、シリアライズ可能なモードを使用するときには、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使用されます。

ほとんどの環境では、「アクセスをシリアル化できません。」のエラーを受け取った後で再起動されたトランザクションと別のトランザクションの間で、二度目の競合が発生することはめったにありません。このため、他のトランザクションと競合する可能性の高い文は、シリアライズ可能トランザクション内の可能なかぎり前の部分で実行するとよい場合があります。ただし、そのトランザクションが正常に完了する保証はないため、再試行の回数を制限するようにアプリケーションをコーディングしておく必要があります。

Oracle Database のシリアライズ可能なモードには SQL-92 との互換性があり、読取り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。アプリケーション設計者は、Oracle Database での読取りは、他のシステムとは異なり、書込みをブロックしないという点を考慮する必要があります。データベースの一貫性をアプリケーション・レベルでチェックするトランザクションでは、`SELECT FOR UPDATE` などのコーディング技法を使用することが必要な場合があります。シリアライズ可能なモードを使用しているアプリケーションを他の環境から Oracle Database に移植する場合には、この問題を検討する必要があります。

データベースの静止

システムを**静止状態**にできます。`SYS` と `SYSTEM` 以外のアクティブなセッションがない場合は、システムは静止状態になっています。アクティブなセッションは、現行のトランザクション、問合せ、フェッチ、PL/SQL プロシージャの中にあるセッション、または現在いずれかの共有リソース（エンキューなど）を保持しているセッションとして定義されています（エンキューは、データベース・リソースへのアクセスをシリアライズする共有メモリー構造で、セッションまたはトランザクションに関連付けられています）。データベース管理者は、システムが静止状態にある際に進行処理のできる唯一のユーザーです。

データベース管理者は、システムが静止状態でなければ安全に行えない特定のアクションを実行できます。このアクションには次のようなものがあります。

- 同時のユーザー・トランザクションや問合せがあると失敗する可能性のあるアクション。たとえば、同時トランザクションが同じ表にアクセスしている場合は、データベース表のスキーマを変更できません。
- 同時のユーザー・トランザクションや問合せに対して、好ましくない中間効果を持つ可能性のあるアクション。たとえば、大きい表 T と、それを操作する PL/SQL パッケージがあるとします。表 T を 2 つの表 T1 および T2 に分割し、PL/SQL パッケージを変更して、元の表 T のかわりに新規の表 T1 および T2 を参照させることができます。

  データベースが静止状態になると、次を実行できます。

```
CREATE TABLE T1 AS SELECT ... FROM T;
CREATE TABLE T2 AS SELECT ... FROM T;
DROP TABLE T;
```

  これで元の PL/SQL パッケージを削除して再作成できます。

継続的に操作する必要のあるシステムでは、このようなアクションをデータベースを停止せずに実行する機能は重要です。

データベース・リソース・マネージャは、システムの静止中に SYS または SYSTEM 以外のユーザーが開始したアクションをすべてブロックします。これらのアクションは、システムが通常の（静止中でない）状態に戻ると実行されます。ユーザーに対して、静止状態によるそれ以外のエラー・メッセージはありません。

データベースの静止方法 データベース管理者は、ALTER SYSTEM QUIESCE RESTRICTED 文を使用してデータベースを静止させます。ユーザー SYS および SYSTEM のみが、ALTER SYSTEM QUIESCE RESTRICTED 文を発行できます。この文を発行すると、オープンしているデータベースのすべてのインスタンスに対して、次のような影響があります。

- Oracle Database はすべてのインスタンスのデータベース・リソース・マネージャに、アクティブでないすべてのセッション（SYS と SYSTEM 以外）がアクティブにならないように指示します。SYS と SYSTEM 以外のユーザーは、新しいトランザクション、問合せ、フェッチまたは PL/SQL 操作を開始できません。
- Oracle Database は、SYS と SYSTEM 以外のユーザーが開始した、すべてのインスタンスの既存トランザクションがすべて終了（コミットまたは終了）するまで待機します。また、Oracle Database では、SYS と SYSTEM 以外のユーザーが開始した、トランザクション外部にあるすべてのインスタンスの実行中の問合せ、フェッチおよび PL/SQL プロシージャがすべて終了するまで待機します。ただし、問合せが連続する複数の OCI フェッチによって実行されている場合、Oracle Database はそのすべての終了までは待機しません。現在実行中のフェッチのみを終了させ、次のフェッチはブロックします。また、Oracle Database では共有リソース（エンキューなど）を保持するすべてのセッション（SYS と SYSTEM のセッション以外）が、そのリソースを解放するまで待機します。これらの操作がすべて終了した後、Oracle Database はデータベースを静止状態にして、QUIESCE RESTRICTED 文の実行を完了します。
- 共有サーバー・モードでインスタンスが実行中の場合、Oracle Database はデータベース・リソース・マネージャに、そのインスタンスに対するログイン（SYS と SYSTEM 以外）をブロックするよう指示します。インスタンスが共有サーバー・モード以外で実行中の場合は、Oracle Database ではそのインスタンスに対するユーザーのログインは制限されません。

静止状態の間は、インスタンスのリソース・マネージャ・プランは変更できません。

ALTER SYSTEM UNQUIESCE 文を発行すると、実行中のすべてのインスタンスが通常モードに戻り、ブロックされたアクションがすべて実行可能になります。管理者は、v$blocking_quiesce ビューを問い合せて、静止の完了をブロックしているセッションを判断できます。

関連項目：

- 『Oracle Database SQL 言語リファレンス』
- 『Oracle Database 管理者ガイド』

Oracle Database のデータ・ロックの方法

ロックは、同じ**リソース**、つまりユーザー・オブジェクト（表や行など）またはユーザーには見えないシステム・オブジェクト（メモリー内の共有データ構造やデータ・ディクショナリ行など）のどちらかにアクセスする複数のトランザクション間で、破損を招く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 Database 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 Database 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。Oracle Database ではまた、必要に応じて、手動でもデータをロックできます。

関連項目： 13-17 ページの「ロックの種類」

この項の内容は、次のとおりです。

トランザクションとデータ同時実行性

Oracle Database 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整合性を実現します。Oracle Database 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、アプリケーション設計者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 Database によって自動的に管理されます。

Oracle Database 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 文について暗黙的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 Database のデフォルトのロック・メカニズムは、最高水準のデータ同時実行性を実現しながら、データ整合性を保証するために、最も低い制限レベルでデータをロックします。

関連項目： 13-26 ページの「明示的（手動）データ・ロック」

この項の内容は、次のとおりです。

ロックのモード

Oracle Database 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・モードは、関連リソースの共有を防止します。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・モードでは、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。データの読取り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継続時間中は保持され、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取り、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取り消すと、Oracle Database では、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、Oracle Database では、セーブポイントまでロールバックすると、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 Database は、低い制限の表ロックを、より高い制限の適切な表ロックに自動的に変換します。たとえば、トランザクションが表の行をロックするために、`FOR UPDATE` 句を指定した `SELECT` 文を使用する場合を考えます。その結果、排他行ロックとその表に対する行共有表ロックが取得されます。後で、トランザクションがロック中の 1 つ以上の行を更新する場合、行共有表ロックは自動的に行排他表ロックに変換されます。

ロックの段階的拡大が発生するのは、あるレベル（行レベルなど）で多数のロックが保持されている場合に、各ロックをデータベースが上位レベル（表レベルなど）の別のロックに変更した場合です。たとえば、あるユーザーが表の中の多数の行をロックした場合、データベースによっては、そのユーザーが保持する複数の行ロックを単一の表ロックに自動的に拡大する場合があります。ロックの数は少なくなりますが、ロックされている対象の制限は大きくなります。

Oracle Database では、ロックの段階的拡大が発生することはありません。ロックの段階的拡大はデッドロックの可能性を大幅に増します。たとえば、システムがトランザクション T1 のためにロックを拡大しようとしますが、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。トランザクション T2 の処理を進めるのに同じデータのロックの段階的拡大が必要な場合、デッドロックが発生します。

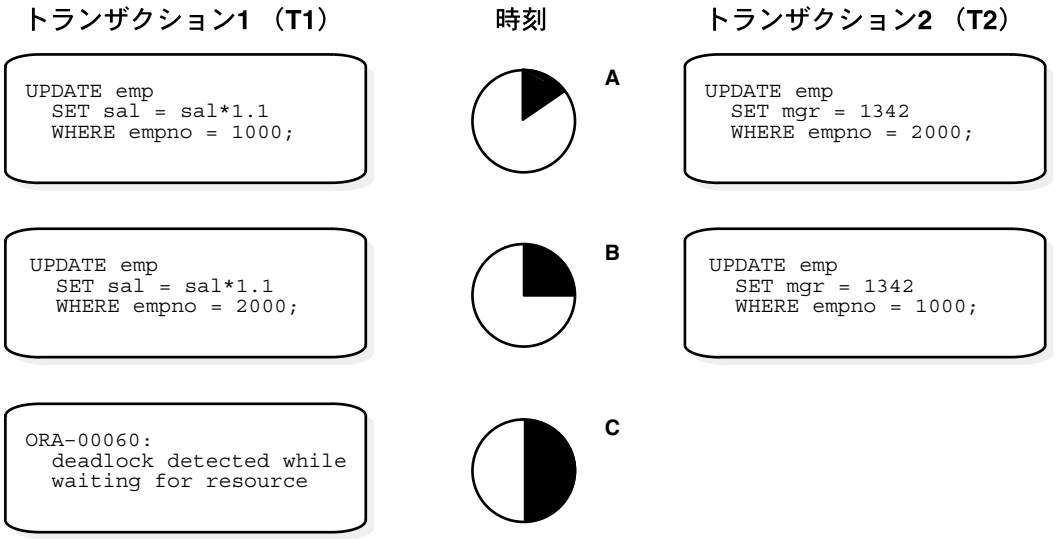
関連項目： 13-19 ページの「表ロック（TM）」

デッドロック

デッドロックが発生するのは、2 人以上のユーザーが、相手がロックしているデータを待機している場合です。デッドロックが発生すると、一部のトランザクションは処理を継続できなくなります。図 13-3 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例を示します。

図 13-3 において、各トランザクションはそれ自体が更新しようとする行に対して行ロックを保持するため、A の時点では問題は存在しません。各トランザクションの処理は、終了せずに進行します。ただし、各トランザクションは、次に、他方のトランザクションが現在保持している行を更新しようとします。したがって、どちらのトランザクションも処理の進行や終了に必要となるリソースを取得できないため、結局、B の時点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 13-3 デッドロック状態の 2 つのトランザクション



この項の内容は、次のとおりです。

- デッドロックの検出
- デッドロックの回避

デッドロックの検出

Oracle Database 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データ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 Database のデフォルト・ロックを明示的にオーバーライドする場合です。Oracle Database 自体はロックの段階的拡大を実行せず、また、問合せに読取りロックを使用しませんが行レベルのロック（ページ・レベルのロックではありません）は使用するため、Oracle Database でデッドロックはまれにしか起こりません。

> **関連項目：** 手動によるロックの取得の詳細は、13-26 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、暗黙的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 Database は、データへの同時アクセスを制御し、各ユーザー間の有害な干渉を防止するために、様々な種類のロックを自動的に使用します。Oracle Database 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的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 Database 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種類のロックを自動的に取得します。

Oracle Database のロックは、表 13-4 に示された 3 つの一般的なカテゴリのいずれかに分類されます。

表 13-4 ロックの種類

ロック	説明
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。

次の各項では、DML ロック、DDL ロックおよび内部ロックについて説明します。

DML ロック

DML ロック（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整合性を保証することです。DML ロックは、同時に実行される矛盾する複数の DML 操作または DDL 操作の破損を招く干渉を防ぎます。DML 文では、表レベル・ロックおよび行レベル・ロックの両方が自動的に取得されます。

注意： 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、Enterprise Manager の Lock Monitor で使用される略称です。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。

この項の内容は、次のとおりです。

- 行ロック（TX）
- 表ロック（TM）
- DML 文について自動的に取得される DML ロック

行ロック（TX）

行レベル・ロックは、主に、2 つのトランザクションによる同じ行の変更を防ぐ目的で使用されます。トランザクションで行を変更する必要がある場合、行ロックが取得されます。

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 Database が行レベルのロックからロックの単位を拡大することはありません。行ロックでは、最もきめの細かいロックが実現されるため、最高の同時実行性とスループットが得られます。

マルチバージョン同時実行性制御と行レベル・ロックを組み合せると、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。

- データの読取りが、同じデータ行の書込みを待機しない場合。
- データの書込みが、同じデータ行の読取りを待機しない場合。ただし、読取りのロックを要求する `SELECT ... FOR UPDATE` を使用している場合は除きます。
- 他の書込みが同時に同じ行を更新しようとする場合にかぎり、書込みは他の書込みを待機します。

注意： 保留中の分散トランザクションにおける非常に特殊な状況では、データを読み取るために、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行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、他のトランザクションがその行を変更できないように**常に**排他的にロックされます。ただし、インスタンス障害のためにトランザクションが終了すると、トランザクション全体がリカバリされる前に、ブロック・レベルのリカバリによって行が使用可能になります。行ロックは、前にリストされた文の結果として、Oracle Database によって常に自動的に取得されます。

ある行について行ロックを取得したトランザクションは、その行に対応する表についての表ロックも取得します。表ロックがあると、カレント・トランザクション内のデータ変更をオーバーライドする DDL 操作の競合が回避されます。

関連項目： 13-24 ページの「DDL ロック」

表ロック（TM）

表レベル・ロックは、DML 操作中に表が削除されないようにするなど、主に同時実行 DDL 操作で同時実行性制御を行うために使用されます。DDL 文または DML 文が表に対するものである場合、表ロックが取得されます。表ロックは、DML 操作の同時実行性には影響しません。パーティション表の場合、表およびサブパーティションの両方のレベルで表ロックを取得できます。

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こと、およびトランザクションと競合する可能性がある DDL 操作を防止するという 2 つの目的のためです。すべての表ロックは同じ表に対する排他 DDL ロックを防止するため、そのようなロックを必要とする DDL 操作も防止されます。たとえば、コミットされていないトランザクションが、ある表について表ロックを保持している場合、その表を変更したり削除することはできません。

表ロックは、行共有（RS）、行排他（RX）、共有（S）、共有行排他（SRX）および排他（X）のいずれかのモードで保持できます。表ロックのモードの制限は、他の表ロックが同じ表について取得し、保持できるモードを決定します。

表 13-5 に、文が取得する表ロックのモードを示します。表の最後の 5 列は、表ロックで許可（可）および禁止（否）される操作を示します。

表 13-5 表ロックのまとめ

SQL 文	表ロックのモード	RS	RX	S	SRX	X
SELECT...FROM *table*...	なし	可	可	可	可	可
INSERT INTO *table* ...	RX	可	可	否	否	否
UPDATE *table* ...	RX	可 *	可 *	否	否	否
DELETE FROM *table* ...	RX	可 *	可 *	否	否	否
SELECT ... FROM *table* FOR UPDATE OF ...	RX	可 *	可 *	否	否	否
LOCK TABLE *table* IN ROW SHARE MODE	RS	可	可	可	可	否
LOCK TABLE *table* IN ROW EXCLUSIVE MODE	RX	可	可	否	否	否
LOCK TABLE *table* IN SHARE MODE	S	可	否	可	否	否
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE	SRX	可	否	否	否	否
LOCK TABLE *table* IN EXCLUSIVE MODE	X	否	否	否	否	否

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。また、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や、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も説明します。

関連項目： 13-26 ページの「明示的（手動）データ・ロック」

この項の内容は、次のとおりです。

- 行共有表ロック（RS）
- 行排他表ロック（RX）
- 共有表ロック（S）
- 共有行排他表ロック（SRX）
- 排他表ロック（X）

行共有表ロック（RS） 行共有表ロック（**副共有表ロック、SS** とも呼ぶ）は、この表ロックを保持しているトランザクションが、その表の行をロックし、それらの行を更新する予定であることを示します。次の SQL 文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。

```
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** とも呼ぶ）は、一般に、このロックを保持しているトランザクションが、表の中または SELECT ... FOR UPDATE で発行された行に対して 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。

```
SELECT ... FROM table ... FOR UPDATE OF ... ;

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取り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 ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

共有表ロック（S） 共有表ロックは、次の文で指定される表について自動的に取得されます。

```
LOCK TABLE table IN SHARE MODE;
```

許可される操作 : トランザクションによって保持される共有表ロックでは、他のトランザクションは、表の問合せ（SELECT ... FOR UPDATE の使用は除く）、または LOCK TABLE ... IN SHARE MODE 文の正常な実行のみが許可されます。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。そのため、共有表ロックを保持しているトランザクションがその表を更新できるのは、他のトランザクションが同じ表について共有表ロックを保持していない場合のみです。

禁止される操作 : トランザクションによって保持される共有表ロックは、他のトランザクションが同じ表を変更するのを禁止します。また、他のトランザクションが次の文を実行することも禁止します。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

共有行排他表ロック（SRX） 共有行排他表ロック（**共有副排他表ロック、SSX** とも呼ぶ）は、共有表ロックよりも多くの制限を課します。共有行排他表ロックは、次の文で指定された表について取得されます。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

許可される操作 : 一度に 1 つのトランザクションのみが、特定の表の共有行排他表ロックを取得できます。トランザクションによって保持される共有行排他表ロックでは、他のトランザクションは表の問合せ（SELECT ... FOR UPDATE の使用は除く）は許可されますが、表の更新は許可されません。

禁止される操作 : トランザクションによって保持される共有行排他表ロックは、他のトランザクションによる行排他表ロックの取得、および同じ表の変更を防止します。また、共有行排他表ロックでは、他のトランザクションが共有ロック、共有行排他ロックおよび排他表ロックを取得することが禁止されます。つまり、他のトランザクションが次の文を実行できなくなります。

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

排他表ロック（X） 排他表ロックは、最も多くの制限が課されるモードの表ロックであり、ロックを保持するトランザクションが表に排他的に書込みアクセスすることを許可します。排他表ロックは、次の文で指定された表について取得されます。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

許可される操作 : 1 つのトランザクションのみが、表の排他表ロックを取得できます。排他表ロックでは、他のトランザクションは同じ表に対する問合せのみを実行できます。

禁止される操作 : トランザクションによって保持されている排他表ロックは、他のトランザクションによる DML 文の実行とその表に対するロックの適用を禁止します。

DML 文について自動的に取得される DML ロック

これまで、各種データ・ロックのタイプと、どのモードで保持できるか、いつ取得できるか、いつ取得されるか、何を禁止するかについて説明しました。次の項では、異なる DML 操作に対する Oracle Database でのデータの自動的なロックについて説明します。

表 13-6 に、次の項で説明する情報の概要を示します。

表 13-6 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	RX
`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

この項の内容は、次のとおりです。

- 問合せのデフォルト・ロック
- INSERT、UPDATE、DELETE および SELECT ... FOR UPDATE のデフォルト・ロック

問合せのデフォルト・ロック 問合せはデータを読み取るのみであるため、他の SQL 文に対してほとんど干渉しない SQL 文です。INSERT 文、UPDATE 文および DELETE 文には、文の一部として暗黙的問合せが含まれている場合があります。問合せには、次の種類の文が含まれます。

```
SELECT

INSERT ... SELECT ... ;

UPDATE ... ;

DELETE ... ;
```

次の文は含まれません。

```
SELECT ... FOR UPDATE OF ... ;
```

次の特性は、FOR UPDATE 句を使用しないすべての問合せに当てはまります。

- 問合せはデータ・ロックを取得しません。そのため、特定の行に対して問合せが実行される場合も含め、問合せが実行されている表を、他のトランザクションが問い合せて更新できます。FOR UPDATE 句が指定されていない問合せでは、データ・ロックが取得されないので他の操作はブロックされないため、Oracle Database ではこの問合せを**非ブロック化問合せ**と呼ぶことがあります。
- 問合せでは、データ・ロックの解除を待つ必要がなく、常に処理を進めることができます。（待ち状態の分散トランザクションがある場合、ごくまれに、問合せにデータ・ロックの待機が必要な場合があります。）

INSERT、UPDATE、DELETE および SELECT ... FOR UPDATE のデフォルト・ロック INSERT、UPDATE、DELETE および SELECT ... FOR UPDATE 文のロックの特徴は次のとおりです。

- DML 文を含むトランザクションは、その文の修正対象の行に対して排他行ロックを取得します。ロックしているトランザクションがコミットまたはロールバックされるまで、その他のトランザクションは、ロックされている行の更新や削除は実行できません。
- DML 文を含むトランザクションは、副問合せや暗黙的問合せ（WHERE 句の中にある問合せ）によって選択される行に対して行ロックを取得する必要がありません。DML 文の中の副問合せや暗黙的問合せは、問合せの開始時点での一貫性が保証され、元の DML 文自体の影響を参照しません。
- トランザクション内の問合せは、同じトランザクション内の前の位置にある DML 文によって加えられた変更を参照できますが、そのトランザクションより後で開始されたその他のトランザクションの変更は参照できません。
- DML 文を含むトランザクションは、必要な排他行ロックに加えて、処理の対象となる行を含む表に対して少なくとも行排他表ロックを取得します。トランザクションがその表について、共有ロック、共有行排他ロックまたは排他表ロックをすでに保持している場合、行排他表ロックは取得されません。トランザクションが行共有表ロックをすでに保持している場合、Oracle Database はこのロックを行排他表ロックに自動的に変換します。

DDL ロック

処理中のデータ・ディクショナリ・ロック（DDL）操作によってスキーマ・オブジェクトが影響を受けたり参照される間、そのオブジェクトの定義は DDL ロックによって保護されます。DDL 文がトランザクションを暗黙のうちにコミットすることに注意してください。たとえば、ユーザーがプロシージャを作成する場合を考えます。そのユーザーの単一文トランザクションのために、Oracle Database 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 Database 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。

この項の内容は、次のとおりです。

排他 DDL ロック

13-24 ページの「共有 DDL ロック」の項に示されている DDL 操作を除き、ほとんどの DDL 操作では、同じスキーマ・オブジェクトの変更や参照を実行する可能性のある他の DDL 操作によって破壊的な干渉が起きないように、リソースの排他 DDL ロックが必要です。たとえば ALTER TABLE 操作で表に列を追加している間は、DROP TABLE 操作でその表を削除できません。その逆も同様です。

排他 DDL ロックの取得では、別の操作によってすでにそのスキーマ・オブジェクトに対して別の DDL ロックが保持されている場合、その取得は古い DDL ロックが解除されるまで待機し、その後に処理されます。

また、DDL 操作は変更対象のスキーマ・オブジェクトに対する DML ロック（データ・ロック）も取得します。

共有 DDL ロック

ある種の DDL 操作では、競合する DDL 操作によって破壊的な干渉が起きないようにし、かつ一方では類似の DDL 操作についてデータの同時実行性を確保するため、リソースの共有 DDL ロックが必要です。たとえば CREATE PROCEDURE 文の実行時には、その文を含むトランザクションは、参照されるすべての表について共有 DDL ロックを取得します。他のトランザクションは同じ表を参照するプロシージャを同時に作成できるため、同じ表について同時実行の共有 DDL ロックを取得できます。ただし、参照中の表について排他 DDL ロックを取得できるトランザクションはありません。また、参照中の表を変更または削除できるトランザクションはありません。その結果、共有 DDL ロックを保持するトランザクションでは、参照中のスキーマ・オブジェクトの定義がそのトランザクションの継続時間にわたって変化しないことが保証されます。

スキーマ・オブジェクトに対する共有 DDL ロックは、AUDIT、NOAUDIT、COMMENT、CREATE（または 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）などの DDL 文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。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 領域が残っているかぎり保持されます。

関連項目： 第 6 章「スキーマ・オブジェクトの依存性」

DDL ロックの継続時間

DDL ロックの継続時間は、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かぎり存続します。

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マテリアライズド・ビューに対して排他 DDL ロックを取得します。クラスタ内の表やマテリアライズド・ビューに対する DDL 操作は、その表やマテリアライズド・ビューに対する共有 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、それらの出現や継続時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。次の項の説明内容は、Enterprise Manager を使用したロックとラッチの監視について理解する上で役立ちます。

この項の内容は、次のとおりです。

- ラッチ
- 内部ロック

ラッチ

ラッチは、システム・グローバル領域（SGA）内の共有データ構造を保護するための単純な下位レベルのシリアライズ・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。

この項の内容は、次のとおりです。

- ディクショナリ・キャッシュ・ロック
- ファイルとログの管理ロック
- 表領域とロールバック・セグメントのロック

ディクショナリ・キャッシュ・ロック これらのロックの継続時間は非常に短く、ディクショナリ・キャッシュ内のエントリが修正されたり使用されている間、そのエントリについて保持されます。これらのロックは、解析されている文が矛盾したオブジェクト定義を参照しないことを保証します。

ディクショナリ・キャッシュ・ロックは、共有または排他で保持されます。共有ロックは、解析が完了すると解除されます。排他ロックは、DDL 操作が完了すると解除されます。

ファイルとログの管理ロック これらのロックは様々なファイルを保護します。たとえば、あるロックは制御ファイルを保護し、一度に 1 つのプロセスのみがそれを変更できるようにします。別のロックは、REDO ログ・ファイルの使用とアーカイブを調整します。データベースを複数インスタンスが共有モードでマウントしたり、1 つのインスタンスが排他モードでマウントすることを保証するために、データファイルがロックされます。ファイルとログのロックはファイルの状態を示すため、必然的に長い間保持されます。

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ている必要があります。ロールバック・セグメントは、1 つのセグメントに 1 つのインスタンスしか書き込めないようにロックされています。

明示的（手動）データ・ロック

データの同時実行性、整合性および文レベルの読取り一貫性を確保するために、Oracle Database は常に自動的にロックを実行します。ただし、Oracle Database では、デフォルトのロック・メカニズムを置き換えることもできます。デフォルト・ロックの置換えは、次のような状況で有効です。

- アプリケーションで、トランザクション・レベルの読取り一貫性または**リピータブル・リード**が必要な場合。つまり、アプリケーション内の問合せによって作成されるデータが、他のトランザクションによる変更を反映せず、そのトランザクションの継続時間にわたって一貫性を維持する必要がある場合。トランザクション・レベルの読取り一貫性は、明示的ロック、読取り専用トランザクションまたはシリアライズ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オーバーライドすると実現できます。
- アプリケーションで、あるトランザクションが他のトランザクションの完了まで待機せずに済むように、そのトランザクションがリソースに排他的アクセスできるようにする必要がある場合。

Oracle Database の自動ロックは、トランザクション・レベルまたはセッション・レベルでオーバーライドできます。

トランザクション・レベルでは、次の SQL 文を含むトランザクションによって Oracle Database のデフォルト・ロックがオーバーライドされます。

- `SET TRANSACTION ISOLATION LEVEL` 文
- `LOCK TABLE` 文（表をロックする文、またはビューを使用するときにその基礎となる実表をロックする文）
- `SELECT ... FOR UPDATE` 文

これらの文で取得されたロックは、トランザクションがコミットまたはロールバックされた後で解除されます。

セッション・レベルでは、ALTER SESSION 文を使用し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 Database 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、データ整合性が保証される、データ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。

関連項目： SQL 文 LOCK TABLE および SELECT ... FOR UPDATE の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

Oracle Database のロック・マネージメント・サービス

アプリケーション開発者は、Oracle Database のロック・マネージメント・サービスを使用して次のような処理をする文を PL/SQL ブロックに含めることができます。

- 特定のタイプのロックを要求します。
- そのロックに、同一のインスタンスまたは別のインスタンス内にある別のプロシージャからも識別できる、一意の名前を指定します。
- ロック・タイプを変更します。
- ロックを解除します。

確保したユーザー・ロックは、Oracle Database 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 Database 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 Database ロックと矛盾することはありません。

Oracle Database のロック・マネージメント・サービスは、DBMS_LOCK パッケージ内のプロシージャを介して利用できます。

関連項目：

- Oracle Database Lock Management サービスの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- DBMS_LOCK の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Oracle Flashback Query の概要

Oracle Flashback Query を使用すると、履歴データを表示および修復できます。特定の実時間またはユーザー指定のシステム変更番号（SCN）による、データベースへの問合せを実行できます。

フラッシュバック問合せでは、Oracle Database のマルチバージョン読取り一貫性機能を使用しており、必要に応じて UNDO を適用することでデータをリストアします。Oracle Database 11*g* では、UNDO 保存期間と呼ばれるパラメータが自動的にチューニングされます。UNDO 保存期間とは、古い UNDO 情報（コミットされたトランザクションの UNDO 情報）が上書き可能になるまでの時間を示します。データベースにより、使用統計が収集され、これらの統計および UNDO 表領域サイズに基づいて UNDO 保存期間がチューニングされます。

フラッシュバック問合せを使用すると、その日の朝、前日または前週のデータベースの状態を問い合せることができます。この操作のスピードは、問い合せるデータの量と、そのデータを元へ戻すための変更の回数にのみ依存します。

特定の行またはトランザクションの履歴を問い合せることができます。データベースに格納されている UNDO データを使用すると、ある行の全バージョンを表示して以前のバージョンに戻すことができます。フラッシュバック・トランザクション問合せの履歴は、トランザクション・レベルでデータベースに対して行われた変更を検査する上で役立ちます。

表の各行を監査し、その行を変更したトランザクションと変更日時に関する情報を取得できます。トランザクション ID を使用すると、LogMiner を介したトランザクション・マイニングを実行して、トランザクションの詳細情報を取得できます。

関連項目：

- UNDO 保存の自動チューニングおよび LogMiner の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 2-18 ページの「自動 UNDO 保存」

表示する日付と時間を設定します。次に SQL 問合せを実行すると、設定した時間におけるデータに対して問合せが行われます。権限を持つユーザーは、管理者の介入なしにエラーを訂正したりリストアされたデータをバック・アウトできます。

SQL の `AS OF` 句を使用すると、問合せで表ごとに異なるスナップショットを選択できます。スナップショットを表に関連付ける操作は、表の修飾と呼ばれます。表をスナップショットで修飾しない場合は、デフォルトのスナップショットが使用されます。スナップショットが指定されていない表はすべて、同じデフォルト・スナップショットを取得します。

たとえば、過去 1 時間以内に作成されたすべての新規顧客アカウントを検索する問合せを作成するとします。異なる `AS OF` 句で修飾された同じ表の 2 つのインスタンスに対して、集合演算を実行できます。

DML 操作と DDL 操作では、表の修飾を使用して、副問合せ内でスナップショットを選択できます。`INSERT TABLE AS SELECT` や `CREATE TABLE AS SELECT` などの操作を副問合せで表の修飾とともに使用して、行が誤って削除された表を修復できます。表の修飾には、バインド変数、定数、文字列、日付の演算など、任意の式を使用できます。カーソルをオープンし、スナップショット値（タイムスタンプまたは SCN）を動的にバインドし、表を修飾できます。

関連項目：

- すべての Oracle Flashback 機能の概要は、1-22 ページの「高可用性機能の概要」を参照してください。
- `AS OF` 句の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

この項の内容は、次のとおりです。

フラッシュバック問合せの利点

この項では、フラッシュバック問合せを使用する利点について説明します。

- アプリケーションの透過性

  レポート生成ツールなど、問合せのみを行うパッケージ・アプリケーションは、ログオン・トリガーを使用してフラッシュバック問合せモードで実行できます。アプリケーションは、コードを変更することなく、透過的に実行できます。フラッシュバック問合せ時の、データベースの一貫したバージョンがあるため、アプリケーションのすべての制約は意識することなく確実に遵守されます。

- アプリケーションのパフォーマンス

  アプリケーションにリカバリ処理が必要な場合は、SCN を保存し、その SCN にフラッシュバックしてリカバリを実行できます。これは、アプリケーションが明示的なバージョニングを実行した場合に、データ・セットを保存して後にリストアするよりもはるかに容易で高速です。フラッシュバック問合せを使用すると、明示的なバージョニングによって発生するロギングのコストを排除できます。

- オンライン操作

  フラッシュバック問合せはオンラインで行う操作です。オブジェクトがフラッシュバック問合せ内で問合せを受けている間、他のセッションから同時実行される DML や問合せが許可されます。それらの操作のスピードは影響を受けません。さらに、別のセッションが同じオブジェクトの異なる時間や SCN に同時にフラッシュバックすることもできます。フラッシュバック問合せ自体のスピードは、問合せが過去にさかのぼる時間の長さに比例して適用される UNDO の量に依存します。

- 管理の容易さ

  ユーザー側では、適切な保存間隔の設定や必要な権限の保持などの他に、追加の管理は必要ありません。また、過去のバージョンは必要に応じて自動的に構成されるため、追加のロギングを実施する必要もありません。

注意：

- フラッシュバック問合せで UNDO は行われません。これはあくまでも問合せのメカニズムです。UNDO は、フラッシュバック問合せの出力を取得してから、独自に様々な状況で実行できます。
- フラッシュバック問合せでは、変更を知らせるメッセージは表示されません。この処理は LogMiner が行います。
- フラッシュバック問合せは、過去に戻す必要のある行がわかっている場合には、変更を取り消すことができ、非常に効率的です。フラッシュバック問合せを使用して表全体を過去に戻すことができますが、これには表全体のコピーが必要になるため、表が大きい場合は非常にコストがかかります。
- フラッシュバック問合せは、列の変更または表の削除や切捨てを行う DDL 操作では機能しません。
- 通常、LogMiner は変更履歴の取得には大変便利ですが、変更はデルタ（挿入、更新、削除）で通知され、行の変更前と変更後のイメージでは通知されません。SQL の `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA` 文では、最小限のロギングのみが補足され、変更された行のすべての列がログに保存されるものではありません。

フラッシュバック問合せの用途

この項では、フラッシュバック問合せの用途について説明します。

- セルフサービス修復

  偶然、表から重要な行を数行削除してしまい、それをリカバリするとします。修復するには、時間をさかのぼって削除した行を探し、それを現在の表に挿入します。

- 電子メール・アプリケーションやボイス・メール・アプリケーション

  過去に、メールを削除している場合があります。フラッシュバック問合せを使用すると、時間をさかのぼって削除したメールを現在のメッセージ・ボックスに再度挿入して、この削除したメールをリストアできます。

- 口座残高

  以前の口座残高を、特定の日付で表示できます。

- パッケージ・アプリケーション

  パッケージ・アプリケーション（レポート生成ツールなど）では、アプリケーション・ロジックに変更を加えることなく、フラッシュバック問合せを利用できます。ユーザーに対して、特定の時間または SCN におけるデータベースの一貫したバージョンが示されるため、アプリケーションで必要となる制約はすべて確実に遵守されます。

また、監査情報の検査後にフラッシュバック問合せを使用して、データのビフォア・イメージを確認できます。DSS 環境では、OLTP システムから一貫した時点でのデータを抽出するために使用できます。

関連項目：

- Oracle Flashback Query の使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- `DBMS_FLASHBACK` パッケージの説明は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- UNDO 表領域と保存期間の設定の詳細は、『Oracle Database 管理者ガイド』を参照してください。

14

管理性

Oracle Database 11*g* は、自己管理データベースに対するオラクル社の取組みにとって大きな進歩を示すものです。この製品によって、多くの日常的な管理作業が自動化され、パフォーマンス診断、SQL チューニング、領域管理、メモリー管理などの主要な DBA 機能が大幅に簡素化されます。また、データベースの主要なコンポーネントの管理について DBA をガイドする複数のアドバイザが導入され、潜在的なメリットとともに特定の推奨事項が提示されるようになりました。さらに、Oracle Database 11*g* は、問題を予測して事前にアラートを送信するため、対処的ではなく予防的なデータベース管理が容易となります。

この章の内容は、次のとおりです。

Oracle Database 11*g* のインストールと開始

Oracle Universal Installer は、Oracle のソフトウェアをインストールするための GUI ツールです。OUI を使用すると、総合的な前提条件（オペレーティング・システムのバージョン、ソフトウェア・パッチ、容量など）の確認、選択したソフトウェア・コンポーネントのインストール、インストール後のすべての構成の実施など、すべてのインストール作業が自動化されます。

インストールは、日常的な監視と管理に必要なインフラストラクチャを自動的に設定する自己完結型の処理です。Oracle Enterprise Manager Database Management Console は、管理者がデータベースの管理作業を開始できるように自動的に構成されます。手動による構成は必要ありません。Oracle Enterprise Manager Database Console には、アラート通知、ジョブ・スケジューリング、ソフトウェア管理も含めて、単一のデータベースを管理するための基本的な機能すべてが用意されています。さらに、データベース、リスナー、管理フレームワークなど、すべての Oracle Database サーバーのコンポーネントが自動的に起動および停止するように構成されます。

> **関連項目：** Oracle Enterprise Manager の詳細は、14-18 ページの「構成の管理」を参照してください。

この項の内容は、次のとおりです。

データベース作成の簡素化

データベース作成アシスタント（DBCA）は、データベース作成用の GUI ツールです。DBCA を使用すると、スタンドアロン・データベース、Oracle Real Application Clusters データベース、スタンバイ・データベースなど、データベースについて可能な構成をすべて作成できます。データベースの作成過程で、DBCA は、ディスク・ベースの自動バックアップの設定とデータベースの LDAP サーバー（使用可能な場合）への登録をガイドします。DBCA を使用して作成されたデータベースは、完全に設定され、あらゆる点で使用の準備が整います。

インスタント・クライアント

インスタント・クライアントは、OCI、OCCI、JDBC-OCI または ODBC ドライバを使用して作成した完全な Oracle クライアント・アプリケーションをデプロイする最も簡単な方法です。インスタント・クライアントは、必要な Oracle クライアント・ライブラリを一連の小さいファイルで提供します。インストールは、いくつかの共有ライブラリをクライアント・コンピュータのディレクトリにコピーする簡単なものです。このディレクトリに、オペレーティング・システムのライブラリ・パス変数（`LD_LIBRARY_PATH` や `PATH`）を介してアクセスできる場合、アプリケーションはインスタント・クライアント・モードで動作します。インスタント・クライアントによるデプロイメントに `ORACLE_HOME` 環境は不要です。また、完全な Oracle クライアントのインストールで提供される多数のコードやデータファイルも不要です。したがって、クライアント・アプリケーションに要するディスク領域を大幅に削減します。完全な `ORACLE_HOME` 環境で実行する同一のアプリケーションと比較しても、インスタント・クライアントを使用してデプロイされたアプリケーションの機能やパフォーマンスに遜色はありません。

> **関連項目：**
>
> - JDBC、OCI および OCCI の詳細は、第 24 章「SQL」および第 25 章「サポートされているアプリケーション開発言語」を参照してください。
> - インスタント・クライアントの詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

自動アップグレード

Database Upgrade Assistant（DBUA）を使用すると、いくつかの簡単な質問に答えることで、Oracle Real Application Clusters（Oracle RAC）やスタンバイを含めたあらゆるデータベース構成をアップグレードできます。DBUA は、適切なリソースが使用可能であることを自動的に確認し、最も効率的な方法（アップグレード処理を開始する前のデータベースをバックアップし、廃止および非推奨の初期化パラメータを置換するなど）に従って、操作の正常な終了を確認します。

アップグレード処理は再開可能で、中断したところから自動的に再開できます。アップグレード処理にかかる時間を見積ることもできます。

基本的な初期化パラメータ

Oracle Database には、様々な環境に応じて操作を最適化する多数の初期化パラメータが用意されています。ほとんどの場合はデフォルト値のままで十分で、明示的な設定を必要とするパラメータはごく少数です。

約 30 種類の基本的なパラメータがあります。それ以外のパラメータは、熟練した DBA が固有の要件にあわせて Oracle Database の動作を調整するために確保されており、このような要件のない DBA には負担となりません。

関連項目：『Oracle Database 管理者ガイド』

データのロード、転送およびアーカイブ

データ・ポンプを使用すると、Oracle Database との間でのデータやメタデータのロードとアンロードを大幅に高速化できます。ロードまたはアンロードの複数のパラレル・ストリームが最大のスループットとなるように、自動的に管理およびスケジュールされます。

トランスポータブル表領域機能により、複数の Oracle データベース間で表領域を迅速に移動できます。表領域の転送に必要なのは、データファイルのコピーと表領域構造情報の統合のみであるため、同じデータのエクスポート / インポートやアンロード / ロードより大幅に高速化されます。また、トランスポータブル表領域を使用すると索引データも移動できるため、表データをインポートまたはロードする際に必要な索引の再作成を回避できます。

データ・ポンプ機能とクロス・プラットフォーム・トランスポータブル表領域機能を併用すると、データベースとの間のデータ移動で使用できる強力で使いやすい高性能なツールとなります。

関連項目：

- 11-2 ページの「データ・ポンプ・エクスポート / インポートの概要」
- 3-15 ページの「データベース間での表領域の移動」

インテリジェント・インフラストラクチャ

Oracle Database には洗練された自己管理インフラストラクチャがあります。これによって、データベースが自らを理解するための情報を取得し、この情報を使用してワークロードの変化に対応したり、潜在的な問題を自動的に解決します。自己管理インフラストラクチャの内容は、次のとおりです。

- 自動ワークロード・リポジトリ
- 自動メンテナンス・タスク
- 障害診断性インフラストラクチャ
- サーバー生成アラート
- アドバイザ・フレームワーク
- ハング・マネージャ

自動ワークロード・リポジトリ

自動ワークロード・リポジトリ（AWR）は、問題の検出および自己チューニング用に Oracle Database で使用されるパフォーマンス統計を含んだ、組込みリポジトリです。Oracle Database は、定期的に、その稼働統計とワークロード情報すべてのスナップショットを作成し、AWR にそのスナップショットを格納します。スナップショットに含まれるデータは、Automatic Database Diagnostic Monitor（ADDM）によって分析されます。スナップショット間の違いを比較し、システム負荷の影響に応じてどの SQL 文を取得するか判断します。これにより、一定期間に取得が必要な SQL 文の数が削減されます。デフォルトでは、スナップショットは 1 時間ごとに作成され、8 日間 AWR に格納され、その後自動的にパージされます。スナップショットの作成頻度および格納期間は、どちらも変更可能です。

特定の期間に作成されたスナップショットは、その他の類似したワークロード期間と比較するためベースラインに保存できます。ベースラインに含まれるスナップショットは、自動 AWR パージ・プロセスからは除外され、無期限に保存されます。Oracle Database では、固定ベースライン、変動ウィンドウ・ベースライン、ベースライン・テンプレートなど各種ベースラインを使用できます。固定ベースラインは、過去における連続した固定期間に対応します。システムが最適な状態で動作しているときに取得された固定ベースラインにより、パフォーマンスが低下している期間に取得されたその他のベースラインやスナップショットを比較し、時間の経過によるパフォーマンス低下を分析できます。変動ウィンドウ・ベースラインは、AWR 保存期間内に存在するすべての AWR データに対応します。このベースラインは、AWR 保存期間全体の AWR データを使用してメトリックしきい値を計算できるため、適応しきい値を使用する場合に便利です。ベースライン・テンプレートを使用して、今後の連続した期間のベースラインを作成できます。ベースライン・テンプレートには単一と繰返しの 2 種類があります。単一のベースライン・テンプレートを使用して、今後の連続した単一の期間のベースラインを作成できます。将来取得する期間をあらかじめわかっている場合に便利なテンプレートです。繰返しのベースライン・テンプレートを使用すると、繰返し時間スケジュールに基づいてベースラインの作成および削除ができます。これは、Oracle Database で連続した期間を継続して自動的に取得する場合に便利なテンプレートです。

AWR は Oracle Database の自己管理機能すべての基礎となっています。AWR は、データベースの使用状況の履歴を Oracle Database に提供する情報源であり、この情報によって ADDM は、潜在的なパフォーマンス上の問題を診断し、解決できます。

> **関連項目：** 自動ワークロード・リポジトリの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

自動メンテナンス・タスク

AWR に格納された情報を分析することで、データベースは、日常的なメンテナンス・タスクの必要性を識別できます。自動メンテナンス・タスクのインフラストラクチャ（AutoTask）によって、Oracle Database は、このような操作を自動的にスケジューリングできます。AutoTask は、メンテナンス・ウィンドウと呼ばれる Oracle Scheduler の一連のウィンドウで実行される自動メンテナンス・タスクをスケジューリングします。メンテナンス・ウィンドウは、Oracle Scheduler のウィンドウ・グループ MAINTENANCE_WINDOW_GROUP のウィンドウです。

MAINTENANCE_WINDOW_GROUP には、デフォルトで、週の各曜日用のウィンドウが含まれています。平日用のウィンドウ（月曜から金曜）は、午後 10 時から 4 時間の間オープン（アクティブ）になるよう構成されます。週末用のウィンドウ（土曜と日曜）は、午前 6 時に開始され、その後 20 時間オープンになります。開始時刻と終了時刻、頻度、週当たりの実施日数など、これらのメンテナンス・ウィンドウのすべての属性をカスタマイズできます。また、グループからのメンテナンス・ウィンドウの追加および削除も可能です。

これらのメンテナンス・ウィンドウで、AutoTask により自動的にスケジューリングされるタスクは次のとおりです。

- オプティマイザ統計の収集
- 自動セグメント・アドバイザ
- SQL チューニング・アドバイザ

Oracle Enterprise Manager または PL/SQL パッケージ・プロシージャを使用して、どのメンテナンス・ウィンドウでこれらのタスクのいずれを実行するかを調整できます。

自動メンテナンス・タスクのリソース割当ての制限

通常のデータベース操作に対する自動メンテナンス・タスクの影響は、デフォルトでデータベース・リソース・マネージャのリソース計画によって制限されています。デフォルトの計画を修正するか、独自のリソース計画を作成し、システム全体のレベルまたは個々のメンテナンス・ウィンドウのレベルでそれらをアクティブ化することが可能です。AutoTask により、すべての自動メンテナンス・タスクが、特定のリソース・コンシューマ・グループに属する Oracle Scheduler ジョブとして実行されます。リソース・プランにより、これらのリソース・コンシューマ・グループに割り当てられる CPU リソースが制限されます。ユーザー・アプリケーションをリソース・コンシューマ・グループに割り当てることができるため、その他のメンテナンス・タスクに関連するものだけでなく、アプリケーションに関連するメンテナンス・タスクへのリソース割当ても調整できます。

関連項目：

- 自動メンテナンス・タスクの管理の詳細は、『Oracle Database 管理者ガイド』および『Oracle Database 2 日でデータベース管理者』を参照してください。
- 14-23 ページの「Oracle Scheduler」
- 14-18 ページの「データベース・リソース・マネージャの概要」
- 自動セグメント・アドバイザの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- SQL チューニング・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

障害診断性インフラストラクチャ

Oracle Database には、問題を防止、検出、診断および解決するための高度な障害診断性インフラストラクチャが含まれています。ターゲットとなる問題は、データベース・コードの不具合、メタデータの破損、顧客データの破損などが原因の重大なエラーです。この高度な障害診断性インフラストラクチャの目的は、次のとおりです。

- 問題の予防的な検出
- 問題検出後の損害および障害の制限
- 問題の診断時間の短縮
- 問題の解決時間の短縮
- ユーザーと Oracle サポートとのやりとりの簡略化

これらの目的を達成するための鍵は、次のテクノロジです。

- エラー検出時に重大なエラーをさらに詳細に分析する状態モニターにより、ヘルス・チェック・レポートが作成され、これらのレポートはそのエラーに対して収集された診断データに追加されます。また DBA はヘルス・チェックを手動で起動し、レポートを取得できます。
- 重大なエラーの最初の発生時に包括的な診断データを取得する最初の障害のデータ・キャプチャ。
- 分析を行いやすくするためのトレースおよびダンプ形式の標準化。
- インシデント・パッケージ化サービス（重要なエラーに関連するすべての診断情報を、Oracle サポートへの送信に適したアーカイブに自動的にパッケージ化）。
- データ破損の問題の表示、問題の範囲の評価および修復オプションの推奨を行うデータ・リカバリ・アドバイザ。
- Oracle サポートによる SQL 障害に関連したユーザーの問題の再現を支援する SQL テスト・ケース・ビルダー。
- サポート・ワークベンチ（重要なエラー診断情報の取得、Oracle サポートへのそのエラーの送信、およびサービス・リクエストの入力を行うことでユーザーを支援するガイド付きのワークフロー）。

> **関連項目：** 障害診断性インフラストラクチャおよびサポート・ワークベンチの詳細は、『Oracle Database 管理者ガイド』を参照してください。

この項では、この新しいインフラストラクチャの次の 2 つのコンポーネントについて説明します。

- 自動診断リポジトリ
- インシデント・パッケージ化サービス

自動診断リポジトリ

自動診断リポジトリ（ADR）は、トレース、アラート・ログ、状態モニター・レポートなどデータベース診断データのファイルベース・リポジトリです。自動診断リポジトリのディレクトリ構造は、複数インスタンスおよび複数製品の間で統一されています。Oracle Database 11*g* から、データベース、自動ストレージ管理（ASM）およびその他の Oracle 製品やコンポーネントでは、すべての診断データは ADR に保存されています。各製品の各インスタンスでは、診断データは固有の ADR ホーム・ディレクトリの下に保存されています。たとえば、共有記憶域と ASM がある Oracle Real Application Clusters 環境では、データベース・インスタンスと ASM インスタンスにはそれぞれ ADR 内にホーム・ディレクトリがあります。ADR の統合ディレクトリ構造、製品とインスタンス間における一貫性のある診断データ形式、および統合されたツール・セットによりユーザーと Oracle サポートは、複数インスタンス間で診断データを相関付けて、分析できます。

インシデント・パッケージ化サービス

DBA は、重大なエラーに関連するすべての診断データ（トレース、ヘルス・チェック・レポート、SQL テスト・ケース他）を自動的かつ簡単に収集し、Oracle サポートへの送信に適した zip ファイルにパッケージ化できます。重大なエラーに関連する診断データにはすべてそのエラーのインシデント番号がタグ付けされているため、DBA がトレース・ファイルおよびその他のファイルを検索して分析が必要なファイルを決定する必要はありません。必要なファイルはすべて自動的にインシデント・パッケージ化サービスにより識別され、パッケージに追加されます。

関連項目：

- コンポーネントの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- ADR 使用の詳細は、『Oracle Database Net Services 管理者ガイド』および『Oracle Database Net Services リファレンス』を参照してください。

サーバー生成アラート

空き領域不足などの自動的には解決できず、管理者への通知が必要な問題に対応するために、Oracle Database にはサーバー生成アラートが用意されています。Oracle Database が自らを監視してアラートを送信することで、タイミングのよい効率的な方法で管理者に問題を通知できます。

監視アクティビティは、データベースが通常の操作を実行するときに開始されます。したがって、問題が発生するとデータベースはただちにその問題に気付くことができます。Oracle Database が生成するアラートは、問題を通知するのみでなく、報告した問題の解決方法に関する推奨事項も提供します。これによって、問題を迅速に解決でき、潜在的な障害の防止に役立ちます。

アドバイザ・フレームワーク

Oracle Database には、データベース内の異なるサブシステムごとに多数のアドバイザが用意されています。これらのアドバイザは、対応するサブコンポーネントの操作をさらに最適化する方法を自動的に判断します。たとえば、SQL チューニング・アドバイザや SQL アクセス・アドバイザは、SQL 文をより迅速に実行するための推奨事項を提供します。メモリー・アドバイザは、試行錯誤の方式に頼らずに、様々なメモリー・コンポーネントのサイズ設定を支援します。セグメント・アドバイザは、無駄な領域の再生の提案や領域使用の増加傾向の分析など、領域に関する問題を処理します。一方、UNDO アドバイザは、UNDO 表領域の正しいサイズ設定をアドバイスします。様々なアドバイザがこの章全般にわたって説明されています。

複数のアドバイザが一貫した統一性のある方法で機能し、アドバイザ同士が相互にシームレスに対話できるように、Oracle Database にはアドバイザ・フレームワークが用意されています。アドバイザ・フレームワークは、アドバイザの起動と結果報告に関する一貫した方法を提供します。これらのアドバイザは、主にデータベースが自身のパフォーマンスを最適化するために使用しますが、特定のサブコンポーネントの機能を知るために、管理者が起動することもできます。

関連項目： アドバイザの使用方法の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

ハング・マネージャ

共有リソースまたはその他の Oracle Database プロセス、セッションおよびトランザクションからのリクエスト・サービスへの制限付きアクセスの取得を試行するアクティブなエンティティは、ハングする可能性があります。ハング連鎖は、単一のプロセスがハングの原因となり、それぞれが次のプロセスが保持しているリソースを待機している複数のプロセスの連鎖です。

システムを使用できなくなるため、Oracle Database でハングが発生するとコスト面で多大な影響があります。ハングは、具体的には次のような問題の原因となります。

- システム停止が拡大します。解決策が見つかるまで頻繁に停止し、合計停止時間が長くなります。
- 問題の原因を特定するためのハングの分析には時間がかかり、複雑でミスが発生する可能性があります。

ハング・マネージャは、ハングを検出して分析し、必要な診断データを取得する Oracle Database インフラストラクチャの 1 つです。ハング・マネージャは、Oracle RAC データベースおよび自動ストレージ管理（ASM）インスタンスでデフォルトで有効化されています。ハング・マネージャのデータはトレース・ファイルに出力されます。

パフォーマンス診断とトラブルシューティング

AWR 内にデータが取得されると、Automatic Database Diagnostic Monitor（ADDM）によって、Oracle Database は自らのパフォーマンスを診断し、識別された問題の解決方法を判断します。ADDM は、AWR 統計の取得ごとに自動的に実行され、パフォーマンス診断データが使用可能になります。

ADDM は、AWR に取得されたデータを調査して分析し、システムの主要な問題を事前に判断します。ほとんどの場合、ADDM は解決策を提示し、予想されるメリットを数量化します。ADDM では、コンポーネント間の共通判断尺度として時間を使用し、システムのパフォーマンスに対して総合的にアプローチします。ADDM は、システムが最も多く時間を消費している領域を識別します。ADDM は、単に症状を示すのではなく、問題を掘り下げて根本原因を識別し、その問題がシステム全体に及ぼす影響を報告します。推奨事項を提示する場合は、予想されるメリットを時間の観点から報告します。一貫して時間を使用することによって、複数の問題や推奨事項の影響を比較検討できます。

ADDM は、データベースが大部分の時間を費やしているアクティビティに注目し、複雑な問題を分類するツリーをドリルダウンします。ADDM が検出する一般的な問題は、次のとおりです。

- CPU のボトルネック
- 貧弱な接続管理
- 過度の解析
- ロックの競合
- I/O 能力
- サイズ設定が小さすぎる Oracle Database メモリー構造（PGA、バッファ・キャッシュ、ログ・バッファなど）
- 負荷が大きい SQL 文
- 時間がかかる PL/SQL および Java
- 負荷が大きいチェックポイントとその原因（小さいログ・ファイル、強引な MTTR 設定など）
- Oracle RAC 固有の問題

潜在的なパフォーマンスの問題報告に加え、ADDM は、システムで問題のない領域も示します。I/O やメモリーなど、システム・パフォーマンスに大きな影響を与えないサブコンポーネントは、初期段階で分類ツリーから除かれてリストされます。これによって、管理者は、それらのサブコンポーネントでの処理による成果が少ないことを速やかに理解できます。

パフォーマンスに関わる問題の解決策のために、膨大な量の診断データを収集し、データ分析に多くの時間を費やす必要はなくなりました。わずかなマウス操作のみで、ADDM が作成した推奨事項に従うことができます。

アプリケーションと SQL チューニング

Oracle Database は、SQL のチューニング処理を完全に自動化します。ADDM は、システム・リソースを大量に使用し、それが原因でパフォーマンス上の問題を引き起こす SQL 文を識別します。さらに、CPU と共有メモリーの使用量の観点で上位を占める SQL 文は、AWR に自動的に格納されます。このように、高い負荷の SQL 文の識別は、Oracle Database で自動的に実施されるため、人的な介入は不要です。

Oracle Database は、リソース消費の点で上位を占める SQL 文を識別すると、自動 SQL チューニング・アドバイザを使用してその SQL 文を自動的に分析し、解決策を示します。自動 SQL チューニングは、SQL チューニング・アドバイザと呼ばれるアドバイザによって提供されます。SQL チューニング・アドバイザは、1 つ以上の SQL 文を入力情報として取得し、チューニング・アドバイスとともに適切にチューニングされた計画を作成します。SQL チューニング・アドバイザの起動以外に、管理者が行う作業はありません。

解決策は、事前定義のヒューリスティックスを使用して、外部ツールではなくオプティマイザから直接発行されます。これには次の利点があります。a）実行計画と SQL パフォーマンスの最終的な責任を負うシステム・コンポーネントによるチューニングが行われます。b）チューニング処理は完全にコスト・ベースで、問合せオプティマイザに対する変更と拡張の責任は当然オプティマイザが負います。c）チューニング処理では SQL 文の過去の実行統計が考慮され、その文に対するオプティマイザ設定がカスタマイズされます。d）問合せオプティマイザにとって有用と考えられるものに基づいて、通常の統計とともに補足情報が収集されます。

自動 SQL チューニング・アドバイザの推奨事項は、次のカテゴリのいずれかに分類されます。

- 統計分析 : 自動 SQL チューニング・アドバイザは、欠落している統計や古くなった統計について各問合せオブジェクトをチェックし、適切な統計を収集するように提案します。推奨事項が実装されない場合は、欠落している統計を補い、古くなった統計を修正するために補足情報も収集します。Oracle Database はオプティマイザ統計を自動的に収集するため、自動統計収集機能が使用禁止にされていないかぎり、この収集に問題はありません。
- SQL プロファイリング : 自動 SQL チューニング・アドバイザは、自らの見積りを検証し、見積りエラーを取り除くために補足情報を収集します。また、SQL 文の過去の実行履歴に基づいて、最初の行またはすべての行など、カスタマイズされたオプティマイザ設定の形式で補足情報を収集します。補足情報を使用して SQL プロファイルを作成し、SQL プロファイルを作成するための推奨事項を作成します。次に、適切にチューニングされた計画を生成するために、問合せオプティマイザを（標準モードで）使用可能にします。SQL プロファイルの最も強力な点は、構文を一切変更せずに問合せをチューニングできることです。したがって、パッケージ化されたアプリケーションに埋め込まれている SQL 文をチューニングするための、データベース常駐の解決策であることが証明されます。
- アクセス・パス分析 : 自動 SQL チューニング・アドバイザは、各表へのアクセスを大幅に改善するために新しい索引が問合せで使用できるかどうかを検討し、適切な場合は、それらの索引の作成を提案します。
- SQL 構造分析 : 自動 SQL チューニング・アドバイザは、不適切な計画に関係している SQL 文の識別を試み、その SQL 文を再構築するための適切な提案を行います。提示する再構築には、SQL コードの意味的な変更の他に、構文的な変更が含まれる場合もあります。

アクセス・パス分析と SQL 構造分析は、管理者と開発者がアプリケーション・コードへのアクセス権を保持している、開発中のアプリケーションや社内の本番アプリケーションのパフォーマンスをチューニングする場合に役立ちます。

SQL アクセス・アドバイザは、指定されたワークロードのスキーマ設計を自動的に分析し、ワークロードに応じた索引、ファンクション索引、パーティションおよびマテリアライズド・ビューの作成、保持または削除を提案できます。単一文の使用例では、アドバイザは現行の文に影響する調整のみを推奨します。完全なビジネス・ワークロードについては、ワークロード全体に与える影響を考慮した後に推奨を行います。

推奨事項の生成中、SQL アクセス・アドバイザは、問合せに見込まれるパフォーマンスの改善に加えて、挿入、更新、削除などのデータ操作アクティビティに新しい索引、パーティションとマテリアライズド・ビューを追加することによる影響を考慮します。SQL アクセス・アドバイザがワークロードのフィルタ処理を完了した後、可能なすべての解決策を識別している間に、プロセスを非同期に中断し、その時点までの最適な解決策を取得できます。

SQL アクセス・アドバイザには、使いやすいインタフェースが用意されているため、システムに関する知識はほとんど必要ありません。本番システムからデータを収集し、SQL アクセス・アドバイザを実行できる別のコンピュータにデータを移動できるため、本番システムに影響を与えずに SQL アクセス・アドバイザを実行できます。

> **関連項目：** SQL チューニング・アドバイザおよび SQL アクセス・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

メモリーの管理

Oracle Database のメモリー管理では、自動または手動による、システム・グローバル領域（SGA）およびプログラム・グローバル領域（PGA）のメモリー・コンポーネントの動的なサイズ変更が可能です。

自動メモリー管理

新しいデータベース・インストールは、デフォルトで、SGA および PGA の様々なコンポーネントを自動的にチューニングするよう構成されています。データベース・パラメータ `MEMORY_TARGET` を変更するだけで、簡単かつ高レベルなメモリー割当ての調整を行うことができます。このパラメータを使用してデータベースにさらにシステム・メモリーを割り当てると、データベースは、最適なデータベース・パフォーマンスを実現するために様々なコンポーネントのサイズを自動的に調整します。

各コンポーネントのパフォーマンスは、Oracle データベース・インスタンスで監視されます。インスタンスは、内部ビューと統計を使用して、自動的にサイズ設定されたコンポーネントにメモリーを適切に分配する方法を決定します。したがって、ワークロードの変化に応じて、新しいワークロードで最適なパフォーマンスとなるようにメモリーが再分配されます。データベースは、長期と短期の傾向を考慮しながら最適な分配を実現します。

各コンポーネントに最小値を指定することで、自動チューニングされたコンポーネントのサイズを部分的に制御できます。これは、特定のコンポーネントが正しく機能するためにアプリケーションで必要なメモリーの最小量がわかっている場合に役立ちます。

サーバー・パラメータ・ファイル（SPFILE）を使用している場合、自動チューニングされたコンポーネントのサイズは停止しても保持されます。したがって、システムは最後の停止で中断した場所から再開します。

手動メモリー管理およびメモリー・アドバイザ

複数のメモリー・コンポーネントに対する割当てをより細かく制御する場合は、手動メモリー管理を有効化します。現在のコンポーネント・サイズやこれらのサイズを変更することによる予想される影響が視覚的に表示される、一連のメモリー・アドバイザを利用できます。

共有プール・アドバイザでは、ライブラリ・キャッシュによる使用を追跡することで最適な共有プール・サイズが判断されます。ライブラリ・キャッシュに使用可能なメモリー容量は、Oracle データベース・インスタンスの解析率に大きく影響することがあります。共有プール・アドバイザ統計は、ライブラリ・キャッシュ・メモリーに関する情報を提供します。これにより、共有プール・サイズの変更内容によって共有プール内のオブジェクトがエージ・アウトする影響を予想できます。

バッファ・キャッシュ・アドバイザでは、バッファ・キャッシュの最適サイズが判断されます。新規インスタンスに手動でメモリーを構成する際に、バッファ・キャッシュの適正サイズを知ることは困難です。通常は、最初にキャッシュ・サイズを見積った後、インスタンス上で代表的なワークロードを実行して関連統計を検査し、キャッシュ構成の過不足を調べます。バッファ・キャッシュ・アクティビティの検査には、多数の統計を使用できます。これには、`V$DB_CACHE_ADVICE` ビューとバッファ・キャッシュ・ヒット率が含まれます。

Java Pool Advisor は、Java に使用されるライブラリ・キャッシュ・メモリーに関する情報を提供し、Java プールのサイズの変化が解析率に及ぼす影響を予測します。

ストリーム・プール・アドバイザでは、ストリーム・プールの最適サイズが判断されます。V$STREAMS_POOL_ADVICE ビューでは、STREAMS_POOL_SIZE パラメータの様々な値における収容されるバイト数と収容されないバイト数が見積もられます。このビューを使用して、ストリームおよびロジカル・スタンバイ・データベースの STREAMS_POOL_SIZE パラメータをチューニングできます。V$STREAMS_POOL_ADVICE ビューおよび CPU 使用率に関する AWR レポートは、ストリーム・パフォーマンスのチューニングに役立ちます。

プログラム・グローバル領域（PGA）アドバイザは、サーバーおよびバックグラウンド・プロセスのすべての PGA に割り当てる合計メモリー量である PGA_AGGREGATE_TARGET に対する適切な設定の判断に役立ちます。

関連項目：

- 第 8 章「メモリー・アーキテクチャ」
- メモリー・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 手動および自動によるメモリー管理用の様々な初期化パラメータの詳細、およびサーバー・パラメータ・ファイルの詳細は、『Oracle Database 管理者ガイド』を参照してください。

領域の管理

Oracle Database は、その領域使用を自動的に管理して、領域に関する潜在的な問題のアラートを送信し、可能な解決策を提案します。領域を容易に管理する上で役立つ Oracle Database の機能には、次のものがあります。

- 自動 UNDO 管理
- Oracle Managed Files
- 空き領域管理
- 予防的な領域の管理
- インテリジェント容量計画
- 領域の再生

関連項目：『Oracle Database ストレージ管理者ガイド』

自動 UNDO 管理

以前のリリースの Oracle Database では、ロールバック・セグメントを使用して UNDO を格納していました。これらのロールバック・セグメントの領域管理は複雑でした。自動 UNDO 管理により、UNDO 表領域の領域が自動的に管理されるため、ロールバック・セグメントの管理の複雑さが解消されます。自動 UNDO 管理は、UNDO が上書きされるまでに保存される時間の長さも最適にチューニングします。UNDO 保存の自動チューニングにより、古い UNDO 情報が必要な長期間実行される問合せおよび一部の Oracle Flashback 機能の成功率が向上します。

データベースでロールバック・セグメントを使用するよう構成できますが、自動 UNDO 管理はデフォルトです。データベースのインストール時に、自動的に拡張する UNDO 表領域が自動的に作成されます。

UNDO 保存の自動チューニングでは、固定サイズの UNDO 表領域を使用するとさらによい結果を得られます。この理由またはその他の理由で UNDO 表領域を固定サイズに変更する場合には、UNDO アドバイザを使用して割り当てる適切な固定サイズを決定できます。長期間実行される問合せまたは Oracle Flashback 操作に必要な UNDO 保存期間を指定すると、UNDO アドバイザにより必要な UNDO 表領域のサイズが提示されます。UNDO アドバイザは、最も長時間実行する問合せと UNDO 生成率を含めたシステム・アクティビティ統計に基づいて提案を行います。アドバイザ情報には、次の項目が含まれます。

- 現行の UNDO 保存
- 現行の UNDO 表領域サイズ
- 最長の問合せ継続時間
- 最善の UNDO 保存
- 現行の UNDO 保存に必要な UNDO 表領域のサイズ

関連項目：

- 2-16 ページの「UNDO セグメントおよび自動 UNDO 管理の概要」
- UNDO の管理および UNDO アドバイザ実行の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。
- UNDO 表領域および UNDO 保存の詳細は、『Oracle Database 管理者ガイド』を参照してください。

Oracle Managed Files

Oracle Managed Files を使用すると、Oracle データベースを構成するファイルを直接管理する必要がなくなります。Oracle Database では、必要に応じて標準ファイル・システム・インタフェースを使用してファイルが作成および削除されます。これにより、データベース・ファイルの作成および削除という日常的なタスクが自動化されます。

空き領域管理

Oracle Database では、従来のディクショナリ・ベースの領域管理のみでなく、ビットマップを使用して表の空き領域を管理できます。実装がビットマップ化されるため、表の領域に関連するチューニングは大部分が不要になり、ピーク負荷時のパフォーマンスが改善されます。また、Oracle Database ではデータファイルの自動拡張機能も用意されているため、ファイルは格納されているデータ量に基づいて自動的に拡張できます。これによって、データベース管理者は、全データベース・ファイルの領域使用率を手動で追跡して再編成する必要がなくなります。

予防的な領域の管理

Oracle Database では、領域使用の監視に、他に影響しないタイミングのよいチェックが導入されています。一般的な領域の割当ておよび割当て解除操作の過程で、領域の使用状況が自動的に監視され、使用可能な空き領域が事前定義のしきい値を下回るとアラートが発行されます。領域監視機能はデフォルトで設定されており、パフォーマンスに影響することなく、すべての表領域タイプで一律に使用できます。また、Oracle Enterprise Manager や SQL を使用しても同じ機能を使用できます。監視は、データベース内での領域の割当てや解放と並行して実行されるため、情報が必要な場合は、領域使用情報をただちに取得できることが保証されます。

通知は、サーバー生成アラートを使用して送信されます。データベース内の特定の領域に関連するイベントが発生すると、アラートがトリガーされます。たとえば、表領域の領域使用しきい値を超えた場合や再開可能セッションで領域不足状況が発生した場合は、アラートが発生します。アラートは、修正措置を講じるために即時に送信されます。アラート情報を受信して表領域に領域を追加することで、中断したところから一時停止操作を続行できます。

データベースには、デフォルトのアラートしきい値が設定されています。指定の表領域に対するデフォルトは上書きできます。また、Oracle Enterprise Manager を介してデータベース全体に新しいデフォルトを設定できます。

インテリジェント容量計画

領域は、オブジェクトの領域要件を予測することが困難なため、あるいはオブジェクトの増加傾向を予測できないために、過剰に割り当てられる可能性があります。頻繁に更新される表では、結果的にセグメントに多数の内部断片化が発生し、行連鎖となる可能性もあります。これらの問題は、パフォーマンス低下の問題から領域損失の問題に至る多種多様な問題に発展する可能性があります。Oracle Database は、これらの難問に対応するために複数の機能を提供しています。

Oracle Database では、指定された表のサイズを、その表の構造と推定行数に基づいて予測できます。これは、オブジェクトの作成や再作成の前にサイズを見積ることができる強力な What If ツールです。表領域に異なるエクステント管理ポリシーがある場合は、内部断片化が最も少なくなるような表領域の判断に、このツールが役立ちます。

増加傾向レポートによって、容量計画の次のステップである増加計画に進みます。ほとんどのデータベース・システムは、時間が経つにつれて大きくなります。増加計画は、リソース・プロビジョニングの重要な側面です。Oracle Database は、このプロセスを支援するために、AWR に領域の使用履歴を記録し、この情報を使用して今後のリソース所要量を予測します。

領域の再生

Oracle Database には、領域使用を最適化するために、セグメントを縮小してデータを適切に再編成する機能があります。セグメントを縮小すると、表領域内の他のセグメントが未使用領域を使用できるようになり、問合せや DML 操作のパフォーマンスが改善される可能性があります。

セグメントの縮小機能は、セグメント内で使用されている領域を圧縮し、空いた領域をセグメントから割当て解除します。割当て解除された領域は表領域に返され、その表領域の他のオブジェクトが使用できるようになります。データが点在している表は、全表スキャンでパフォーマンス上の問題が発生する可能性があります。縮小を実行することで、表のデータが圧縮され、セグメントの最高水位標が押し下げられます。これによって、全表スキャンの読取りブロック数が少なくなり、実行が高速化されます。

セグメントの縮小はオンラインで実行される操作です。セグメントを縮小している間も、縮小されている表に対して問合せおよび DML 操作を実行できます。また、セグメントの縮小は適切に実施されます。これは、領域の圧縮と再生に関するオンラインによる表再定義の利点です。データベース内の 1 つまたはすべてのオブジェクトについて、夜間ジョブとしてセグメントの縮小をスケジューリングできます。その際、他の領域をデータベースに割り当てる必要はありません。

セグメントの縮小は、自動セグメント領域管理を使用している表領域内で行の移動が可能なヒープ、IOT、IOT オーバーフロー・セグメント、LOB、LOB セグメント、マテリアライズド・ビューおよび索引に対して実行されます。索引が設定されている表でセグメントの縮小を実行すると、圧縮のために行が移動してもその索引は自動的に保持されます。圧縮は純然たる物理操作であってアプリケーションには影響がないため、ユーザー定義のトリガーは起動されません。

注意： セグメントの縮小は、行の移動が可能な表でのみ実行できます。オブジェクトの ROWID を明示的に記録するアプリケーションは縮小できません。これは、そのアプリケーションによってオブジェクト内の行の物理的な位置が追跡管理されるためです。

縮小の候補となるセグメントを簡単に識別するために、Oracle Database はセグメント・アドバイザを自動的に実行し、データベース全体を評価します。セグメント・アドバイザは、個々のオブジェクトについて増加傾向分析を実行し、余分な領域がオブジェクトに残されているかを 7 日ごとに判断します。その後、領域の再生目標を使用して、縮小するオブジェクトの候補を選択します。

注意： セグメント・アドバイザでは、UNDO 表領域および一時表領域は評価されません。

ワークロード・リポジトリ内の事前計算済統計の使用に加え、セグメント・アドバイザは、考慮中のオブジェクトをサンプリングして、そのオブジェクトに関する統計を絞り込みます。この操作にはリソースが必要ですが、より正確な分析の実行に使用できます。

セグメントの縮小によって行連鎖が少なくなり、Oracle Database ではオンライン再定義を行って連鎖行を削除することが推奨されていますが、実際、セグメント・アドバイザにより、しきい値を上回るいくつかの連鎖行が検出されます。たとえば、更新の際に行サイズが増加して、ブロックに収容できない場合、セグメント・アドバイザは、セグメントを再編成して I/O パフォーマンスを向上させることを推奨します。

注意： セグメント・アドバイザは、挿入により作成された連鎖行は検出しません。

関連項目：

- 行連鎖の詳細は、2-6 ページの「行連鎖と移行」を参照してください。
- セグメント・アドバイザの使用方法の詳細は、『Oracle Database 管理者ガイド』および『Oracle Database 2 日でデータベース管理者』を参照してください。

自動ストレージ管理

自動ストレージ管理（ASM）によって、Oracle データベース・ファイル専用に構築されたファイル・システムとボリューム・マネージャの垂直統合が提供されます。ASM は、使用可能なすべてのリソースにわたって I/O 負荷を分散してパフォーマンスを最適化し、手動による I/O チューニングの必要性を取り除きます（データベース・ファイルの分散によってホットスポットを回避します）。ASM では、データベースを停止して記憶域割当てを調整せずにデータベース・サイズを拡大でき、動的データベース環境の管理に役立ちます。

ASM では、ディスク・グループと呼ばれる記憶域プールを定義して、Oracle カーネルにより、そのディスク・グループでのデータベース・ファイルのファイル名および配置を管理できます。記憶域割当ては、`CREATE DISKGROUP`、`ALTER DISKGROUP` および `DROP DISKGROUP` などの SQL 文を使用して、ディスクの追加や削除などによって変更できます。ディスク・グループは、Oracle Enterprise Manager および Database Configuration Assistant（DBCA）でも管理できます。

Oracle Database には、記憶域リソース用の簡素化された管理インタフェースが用意されています。ASM は、手動による I/O パフォーマンスのチューニングを排除します。記憶域は一連のディスク・グループに仮想化され、用意されている冗長性オプションにより、高水準の保護が使用可能になります。ASM をバランスの自動再調整と併用すると、連続した記憶域の構成変更が容易になります。データベース・ファイルは使用可能なすべての記憶域にまたがって分散され、パフォーマンスとリソース使用率が最適化されます。手動による格納操作が自動化されることで記憶域の管理オーバーヘッドが削減され、より多くの大規模データベースを効率的に管理できるようになります。

次の項では、ASM の基本概念について説明します。

- 自動ストレージ管理インスタンス

  ASM インスタンスは、ディスク・グループ内のディスクを管理する特殊な Oracle インスタンスです。データベース・インスタンスから ASM ファイルにアクセスするには、ASM インスタンスを構成して実行する必要があります。データベースの作成に Database Configuration Assistant を使用した場合は、この構成が自動的に実行されます。ASM インスタンスでは、データベースをマウントできません。ASM インスタンスは、データベース・インスタンスのデータ・レイアウト調整のみを実行します。データベース・インスタンスは、ASM インスタンスにアクセスせずに、ディスク・グループ内のディスクに対してダイレクト I/O を実行します。

- ディスク・グループ

  ディスク・グループは、1 つの論理単位として管理される 1 つ以上の ASM ディスクです。ディスク・グループのデータ構造は自己完結型で、ディスク・グループの一部のディスク領域を使用します。ディスク・グループ内の ASM ディスクは、データベースの稼働中に追加または削除できます。ディスク・グループの構成に変更があっても、そこに含まれる全ディスクへの均一の I/O 負荷を保証するために、データ・バランスが再調整されます。

- 自動ストレージ管理ファイル

  ASM では、データベースで必要になるとファイルが作成されます。各ファイルには、末尾にドットで区切られた数値ペアが付いた完全修飾名が割り当てられます。ASM のファイル名には、よりわかりやすい別名を作成できます。ASM ファイルの別名を調べるには、ASM インスタンスから `V$ASM_ALIAS` データ・ディクショナリ・ビューを問い合せます。ただし、通常、ユーザーがファイル名を知る必要はありません。

- 自動ストレージ管理ディスク

  記憶域は、ディスク・グループから ASM ディスク単位で追加および削除されます。ASM ディスクは、完全な物理ディスク、ストレージ・アレイからの論理ユニット番号（LUN）、または NAS フィルタで事前に作成されたファイルの場合があります。個々の ASM ディスクは、最適な I/O パフォーマンスを取得するために相互に独立している必要があります。たとえば、ストレージ・アレイの場合は、一対のミラー化された物理ディスクであるハードウェアを表す LUN を、単一の ASM ディスクとして ASM に指定できます。

  関連項目： ASM の詳細は、『Oracle Database ストレージ管理者ガイド』を参照してください。

バックアップおよびリカバリ

Oracle Database には、バックアップおよびリカバリの管理を容易にする複数の機能が用意されています。たとえば、次の機能があります。

- Recovery Manager
- 平均リカバリ時間（MTTR）
- セルフサービスによるエラー修正

Recovery Manager

Oracle Recovery Manager（RMAN）は、バックアップ操作とリカバリ操作を単純化して自動化し、パフォーマンスを改善する強力なツールです。Recovery Manager によって、一時的なバックアップ構成、ユーザー指定のリカバリ期間に基づくバックアップとアーカイブ・ログの自動管理、再起動可能なバックアップとリストア、テスト・リストア / リカバリが使用可能になります。RMAN では、バックアップの存続期間を制御するリカバリ期間が実装されます。これにより、データベースまたは表領域のポイント・イン・タイム・リカバリを実行することで、論理エラーを検出して影響を受けるオブジェクトを修正するための期間を設定できます。また、Recovery Manager では、リカバリ期間内にデータベースを特定の時点までリストアする上で不要になったバックアップは、自動的に期限切れになります。制御ファイルの自動バックアップでは、Recovery Manager リポジトリが使用不可能な場合にも、データベースをリストアまたはリカバリできます。

DBCA は、オンディスク・バックアップ手続きを自動的にスケジューリングできます。ユーザーに必要な唯一の作業は、自動バックアップを実行する時間枠を指定することです。リカバリに関連するすべてのファイルとアクティビティ用に統一された Oracle データベース内の格納位置は、フラッシュ・リカバリ領域と呼ばれ、初期化パラメータ `DB_RECOVERY_FILE_DEST` で定義できます。制御ファイル、アーカイブ・ログ・ファイル、フラッシュバック・ログ、Recovery Manager のバックアップなど、メディア障害からデータベースを完全にリカバリするために必要なファイルはすべて、フラッシュ・リカバリ領域の一部です。

Oracle データベースのリカバリを高速化、単純化および自動化するには、フラッシュ・リカバリ領域に十分な領域を割り当てる必要があります。フラッシュ・リカバリは、この位置に格納されているファイルをインテリジェントな方法で実際に管理して、領域使用を最大にし、領域不足状況を可能なかぎり回避します。指定の Recovery Manager 保存ポリシーに従って、フラッシュ・リカバリ領域では、その構成に基づいて不要になった古いバックアップやアーカイブ・ログが自動的に削除されます。

増分バックアップにより、前回のバックアップ以降に変更されたブロックのみをバックアップできます。ブロック・チェンジ・トラッキング機能が使用可能な場合、Oracle Database はすべてのデータベース変更の物理位置を記録します。Recovery Manager は、変更追跡ファイルを自動的に使用して、増分バックアップ時に読み取る必要があるブロックを判断し、そのブロックに直接アクセスしてバックアップします。これによって、日次バックアップに必要な時間が短縮され、ネットワークを介してバックアップを実行する際のネットワーク帯域幅を節約でき、バックアップ・ファイルの記憶域が削減されます。

増分バックアップを使用して、以前に作成されたバックアップを更新できます。次第に増加する更新されたバックアップを使用して、Recovery Manager の増分バックアップとデータファイルのイメージ・コピーをマージすると、増分バックアップで取得した変更を組み込んだ最新のバックアップを作成できます。データベース全体を繰返しバックアップする必要はありません。指定のデータベースに対して完全なデータベース・バックアップを 1 度作成し、その後は増分バックアップを使用することで、更新された完全なバックアップを保持できます。次第に増加する更新されたバックアップに基づくバックアップ方法によって、データベースのメディア・リカバリに必要な時間を最小限にできます。

関連項目：

- 『Oracle Database 管理者ガイド』
- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』

平均リカバリ時間（MTTR）

Oracle Database では、システム障害からの平均リカバリ時間（MTTR）を秒単位で指定することで、データベースの停止時間を適切に制御できます。ユーザーによる MTTR の指定と動的初期化パラメータを併用すると、データベースの可用性を改善できます。システム障害からのリカバリ所要時間に時間制限を設定すると、Oracle Database では障害時にシステム上で実行されていたアプリケーション・アクティビティに関係なく、その時間内にシステムを再起動できることが自動的かつ透過的に確認されます。これにより、システム障害の発生後は最短時間で復旧できます。

オンライン・ログ・ファイルが小さいほど、DBWR では増分チェックポイントが実行され、物理書込みが増えることになります。これはデータベースのランタイム・パフォーマンスを低下させる場合があります。また、FAST_START_MTTR_TARGET を設定すると、最小ログ・ファイル・サイズによって MTTR が必要とするよりも多くの増分チェックポイントが発生する場合があります。

ログ・ファイル・サイズ・アドバイザでは、現行の FAST_START_MTTR_TARGET 設定と MTTR 統計に基づいて最適の最小ログ・ファイル・サイズが決定されます。最小ログ・ファイル・サイズが最適とみなされるのは、FAST_START_MTTR_TARGET が必要とするよりも多数の増分チェックポイントを発生させない場合です。

MTTR アドバイザにより、余分な物理書込みに関して様々な MTTR 設定がシステム・パフォーマンスに与える効果を評価できます。MTTR アドバイザが使用可能になっている場合は、システムで典型的なワークロードが実行された後に、V$MTTR_TARGET_ADVICE を問合せして、現行の MTTR でのキャッシュ書込み数に対する他の MTTR 設定での予想キャッシュ書込み数の比率を表示できます。たとえば、比率 1.2 は、キャッシュ書込み数が 20% 増加することを示します。

様々な MTTR 設定とそれに対応するキャッシュ書込み比を調べることで、リカバリおよびパフォーマンスのニーズに適合する MTTR の値を決定できます。V$MTTR_TARGET_ADVICE でも、直接の書込みなどの合計物理書込み数に関する比率と、読取りなどの合計 I/O に関する比率が表示されます。

> **関連項目：** MTTR アドバイザ使用の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

セルフサービスによるエラー修正

Oracle Flashback テクノロジによって、データを時間的な前後関係の中で表示できます。データベースがオンラインの間に、スキーマ・オブジェクトの過去のバージョンの問合せ、履歴データの問合せ、変更分析の実行、またはセルフサービスの修復を実施して、論理的な破損から情報をリカバリできます。

これによって、リカバリ方法とは変更されたデータの単なる操作となりました。エラーのリカバリにかかる時間は、誤りが作成された時間に等しくなります。

関連項目：

- 1-22 ページの「高可用性機能の概要」
- 15-12 ページの「Oracle Flashback テクノロジ」
- 13-28 ページの「Oracle Flashback Query の概要」

構成の管理

Oracle Enterprise Manager には、構成の変更や違いを検出し、強制的に最も効率的な構成パラメータ設定を規定する複数の強力な構成管理機能があります。これらの機能は、基礎となるホストおよびオペレーティング・システムにも適用されます。

Oracle Enterprise Manager は、最も効率的な方法によるパラメータ設定、セキュリティ設定、記憶域およびファイル領域の状況、および推奨機能の使用などについて、Oracle システムすべての構成を継続的に監視します。基準に満たないシステムには、システム構成に固有の問題に関する詳細な説明とともに、フラグが自動的に設定されます。たとえば、自動 UNDO 管理やローカル管理表領域などの新しい機能を使用していない場合は、Oracle Enterprise Manager ではそれらの機能を使用するようにアドバイスされます。システム構成の自動監視によって、最も効率的な構成管理が促進され、管理者のワークロードおよび可用性、パフォーマンスまたはセキュリティに関するリスクが軽減されます。

また、Oracle Enterprise Manager は、新しい重要なパッチ（セキュリティに関する重要なパッチなど）に関するアラートを発行し、そのパッチを必要とするすべてのシステムにフラグを設定します。さらに、インストールに使用可能な個別パッチを見つけるために、Oracle Enterprise Manager のパッチ・ウィザードを起動できます。

関連項目：『Oracle Enterprise Manager 概要』

ワークロードの管理

Oracle Database には、次のリソース管理機能があります。

- データベース・リソース・マネージャの概要
- サービスの概要

データベース・リソース・マネージャの概要

データベース・リソース・マネージャには、Oracle データベース・システム内での操作に優先順位を設定する機能が用意されています。オンライン・ユーザーについて応答時間を最短にするために、優先順位の高いコンシューマがリソースを取得しますが、バッチ・ジョブやレポートなど、優先順位の低いコンシューマの所要時間が長くなることがあります。このため、よりきめ細かいリソース制御が可能で、コンシューマ・グループについて自動コンシューマ・グループ切替え、アクティブ・セッションの最大数の制御、問合せ実行時間の見積りと UNDO プール割当て制限などの機能を提供します。

コンシューマ・グループごとに、同時にアクティブなセッションの最大数を指定できます。この制限に達すると、データベース・リソース・マネージャは以降の要求をすべてキューに入れ、それらの要求は既存のアクティブ・セッションが完了した後でのみ実行します。

データベース・リソース・マネージャを使用することで、オペレーティング・システムでは十分に管理できない次の多くのリソース割当ての問題が解決されます。

- 過度のオーバーヘッド。過度のオーバーヘッドは、サーバー・プロセスの数が多いときに、Oracle データベースのサーバー・プロセス間におけるオペレーティング・システムのコンテキストのスイッチングに起因して発生します。
- 非効率的なスケジューリング。オペレーティング・システムは、Oracle データベース・サーバーが非効率的なラッチを保持している間はそのスケジュールを停止します。
- 不適当なリソースの割当て。オペレーティング・システムは、すべてのアクティブなプロセスにリソースを均等に分配します。また、オペレーティング・システムは、ある作業を別の作業に優先させることはできません。
- データベース固有のリソース管理ができない状態。

データベース・リソース・マネージャを使用して、次のことができます。

- システムの負荷およびユーザー数に関係なく、特定のユーザーに最小限の処理リソースを保証します。
- 異なるユーザーおよびアプリケーション間に、CPU タイムの割合または 1 秒当たりの I/O リクエスト数を割り当て、使用可能な処理リソースを分配します。

  たとえば、データ・ウェアハウスでは、バッチ・ジョブよりも ROLAP（リレーショナル・オンライン分析処理）アプリケーションに対する CPU の割合を高くする場合があります。共有記憶域が構成され、I/O リソース管理が有効な場合は、データベースにより発行可能な 1 秒当たりの最大 I/O リクエスト数、または 1 秒当たりの最大 MB 数も設定できます。
- ユーザー・グループのメンバーが実行する処理の並列度を制限します。
- **アクティブなセッションのプール**を作成します。

  このプールは、ユーザーが所属するグループ内で同時にアクティブになれるユーザー・セッションで指定された最大数で構成されます。最大数を超えるセッションは、実行待ちのキューに入りますが、キューに入れられたジョブが終了するまでのタイムアウト周期を指定できます。
- 管理者が定義した基準に基づいて、あるグループから他のグループへのユーザーの切替えを自動的に行うことができます。

  特定のユーザー・グループのメンバーによって、実行時間、I/O 使用量または I/O リクエスト数が指定された最大値を超えるセッションが作成された場合は、リソース要件の異なる別のユーザー・グループにセッションを自動的に切り替えることができます。
- 実行時間が事前定義済の制限を超えると推定される場合、その処理が実行されないようにします。

  注意： ユーザーの切替えや予防操作は、CPU 時間のみでなく、I/O の総量にも基づいています。

  関連項目： 自動切換えの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- **UNDO プール**の作成。

  UNDO プールは、ユーザーが所属するグループが使用できる UNDO 領域の総量からなります。
- 特定のリソース割当て方法を使用するようにインスタンスを構成します。

  インスタンスを停止して再起動しなくても、セットアップ作業を業務時間中から夜間に変更するなど、方法を動的に変更できます。
- 静止の完了をブロックするセッションを識別します。

企業全体の目標を達成するには、あるユーザーと他のユーザーが使用するリソースのバランスをとったり、処理に割り当てるシステム・リソースを重要度により分割します。

データベース・リソース・マネージャの概念

リソースは、データベース管理者が指定するリソース・プランに従ってユーザーに割り当てられます。次の用語は、リソース・プランを指定するのに使用します。

リソース・プランで、各ユーザー（リソース・コンシューマ・グループ）へのリソースの分配方法を指定します。

リソース・コンシューマ・グループを使用して、リソース要件ごとにユーザー・セッションをグループ化します。リソース・コンシューマ・グループは、ユーザー・ロールとは異なり、1 人のデータベース・ユーザーが、異なるリソース・コンシューマ・グループに割り当てられた異なるセッションを持つことができます。

リソース割当て方法は、特定のリソースを割り当てる際に使用する方針を決めます。リソース割当て方法は、リソース・プランとリソース・コンシューマ・グループで使用されます。

リソース・プラン・ディレクティブは、各リソース割当て方法でパラメータを指定して、コンシューマ・グループを特定のプランに割り当て、コンシューマ・グループ間でリソースを分配するための手段です。

データベース・リソース・マネージャでは、プラン内にサブプランと呼ばれるプランも作成できます。**サブプラン**では、アプリケーションの複数のユーザー間でリソースをさらに副分割できます。

レベルは、利用可能なユーザー間で未使用のリソースの分配を指定するメカニズムを提供します。リソース割当ては、8 レベルまで指定できます。

関連項目：

- データベース・リソース・マネージャの使用方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- リソース・プランのチューニング方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

サービスの概要

サービスとは、共通の属性、サービス・レベルのしきい値および優先順位を持つアプリケーション・グループまたは大規模なアプリケーションのサブセットを表します。アプリケーションの機能は、サービスで識別されるワークロードに分割できます。たとえば、Oracle E-Business Suite では、総勘定元帳、売掛管理、受注などのモジュールごとにサービスを定義できます。Oracle Mail では、IMAP プロセス、ポストマン、ガベージ・コレクタ、モニターなどのサービスを定義できます。サービスの範囲は、Oracle データベースまたはクラスタ内の複数データベースの 1 つ以上のインスタンスに及ぶことが可能です。また、1 つのインスタンスで複数のサービスをサポートできます。

サービスを提供しているインスタンス数は、アプリケーションに対して透過的です。サービスは、競合するアプリケーションを管理するために単一のシステム・イメージを提供し、各ワークロードを 1 単位として管理できます。

中間層のアプリケーションおよびクライアントは、サービス名を TNS 接続データの接続の一部として指定することで、サービスを選択します。たとえば、Web サーバーまたはアプリケーション・サーバーのデータ・ソースは、1 つのサービスにルーティングするように設定されます。Net Easy*Connection を使用すると、この接続に、サービス名とネットワーク・アドレスが含まれます。たとえば、サービス :IP のようになります。

スケジューラ、パラレル実行および Oracle Streams Advanced Queuing など、サーバー側の作業では、サービス名がワークロード定義の一部として設定されます。スケジューラの場合、ジョブはジョブ・クラスに割り当てられ、ジョブ・クラスがサービス内で実行されます。パラレル実行およびパラレル DML の場合は、問合せコーディネータがサービスに接続します。パラレル実行プロセスは、問合せの期間を通じてそのサービスを継承します。Oracle Streams Advanced Queuing の場合は、サービスを使用してストリーム・キューがアクセスされます。サービスのもとで実行中の作業は、サービスのしきい値と属性を継承し、サービスの一部として測定されます。

データベース・リソース・マネージャは、サービスをコンシューマ・グループおよび優先順位にバインドします。これによって、優先順位の順にデータベース内のサービスが管理されます。たとえば、優先順位の高いオンライン・ユーザーと優先順位の低い内部レポート・アプリケーションに対して個別のサービスを定義できます。同様に、ゴールド、シルバーおよびブロンズなどのサービスを定義して、同じアプリケーションの要求がサービスされる優先順位を設定できます。

システムに対して複数のサービスを計画する場合は、サービス間の相対優先順位を指定します。このようにして、データベース・リソース・マネージャは、最初に優先順位の最も高いサービスを処理し、次の優先順位のサービスが後に続くようにしています。

この項の内容は、次のとおりです。

サービスを使用したワークロード管理

AWR を使用すると、サービス用の新規集計ディメンションを使用してワークロードのパフォーマンスを分析できます。AWR は、すべてのサービスに関する応答時間と CPU 使用メトリック、パフォーマンス統計とリソース統計の待機イベント、しきい値ベースのアラートおよびパフォーマンス索引を自動的にメンテナンスします。

サービス、モジュールおよび処理タグは、サーバーのサービス内で操作を識別します。（`MODULE` と `ACTION` はアプリケーションで設定されます。）エンド・ツー・エンドの監視で、サービス、モジュールおよび処理レベルによる集計とトレーシングを実行し、負荷が大きい操作を識別できます。Oracle Enterprise Manager は、応答時間と CPU 使用のサービス品質しきい値を管理し、上位のサービスを監視し、各サービスで上位を占めるモジュールと処理へのドリルダウンを提供します。

AWR では、サービス集計によるパフォーマンス管理は、セッションによる監視が意味を持たない場合に有意です。たとえば、接続プーリングまたはトランザクション処理モニターを使用するシステムでは、セッションが共有されるため、アカウンタビリティを実現するのは困難です。

サービス、モジュールおよび処理タグは、作業と処理フローを区別する大小の境界となります。この集計レベルにより、ともに実行される SQL のグループを（サービス、モジュールおよび処理の各レベルで）チューニングできます。これらの統計は、サービス品質の管理、リソース使用の査定、サービス間の相対優先順位の調整、およびチューニングが必要な箇所の参照に使用できます。Oracle Real Application Clusters（Oracle RAC）を使用すると、現在のパフォーマンスに基づいて様々なインスタンス上でサービスをプロビジョニングできます。

接続時ルーチンとランタイム・ルーチンのアルゴリズムによって、サービスを提供しているインスタンス全体でワークロードのバランスが調整されます。サーバー側の接続ロード・バランシングのメトリックは、サービス・パフォーマンスを含めるように拡張されます。接続は、現在のサービス・パフォーマンスに従ってインスタンス全体で共有されます。ロード・バランシングにサービス・パフォーマンスを使用することで、サイズとワークロードが異なり、競合する優先順位を持つ複数のノードを解決できます。また、中断または障害が発生しているノードへの作業の送信を防止できます。

AWR は、サービス・パフォーマンスのメトリックを継続的に保持します。これらのメトリックは、中間層サーバーおよび TP モニターからのランタイム要求を Oracle RAC にルーティングする場合に使用できます。たとえば、Oracle JDBC 接続プールは、サービスを提供しているインスタンスにランタイム要求をルーティングするときに、サービス・データを使用します。

サービスを使用した高可用性

Oracle RAC は、サービスを使用して、割込みなしのデータベース操作を実現します。サービスは、Oracle RAC をサポートする高可用性フレームワークである Oracle Clusterware と緊密に統合されています。障害が発生すると、障害に影響されないノードおよびインスタンスで、割込みなしでサービスが続行されます。障害の影響を受けたサービスの要素は、Oracle Clusterware によって速やかにリカバリされ、リカバリ・セッションは正常に稼働しているシステム全体で自動的にバランスが保たれます。

計画された休止の場合、Oracle RAC には、サービスを再配置、使用禁止および使用可能にするためのインタフェースがあります。再配置は、サービスを別のインスタンスに移行し、必要な場合はセッションが切断されます。Oracle Clusterware システムが、メンテナンスまたは修復時に発生する予定外の障害に応答しないように、予定された休止の開始時に、メンテナンスが実行されるノードではサービスが使用禁止になります。サービスは、休止の終了時に使用可能になります。

サービス・ベースのこれらの操作をサービス・ベースのスキーマ・プリコンパイル（DBMS_SCHEMA_COPY）と組み合せることで、予定されている多数の休止による停止時間を最小限にします。たとえば、アプリケーションのアップグレード、オペレーティング・システムのアップグレード、ハードウェアのアップグレードや修理、ローリング・アップグレードが承認された Oracle のパッチ、およびパラメータの変更は、一度に 1 つ以上のサービスを分離して実装できます。

Oracle RAC に組み込まれている継続的なサービスは、アプリケーションおよび中間層のサーバーにまで拡張されます。サービスの状態が変化（たとえば、起動、停止または再開不可）すると、新しいステータスがイベントおよびコールアウトを介して関連するサブスクライバに通知されます。アプリケーションは、障害を迅速に検出し、障害に対応する接続プールをバランシングし、障害の発生したコンポーネントが修復されたときに再度接続プールをバランシングするために、この通知を使用できます。たとえば、インスタンスでサービスを開始する場合は、ただちにそのサービスで作業をトリガーするためにイベントおよびコールアウトが使用されます。

インスタンスでサービスが停止すると、そのインスタンスでサービスを使用しているアプリケーションを中断するために、イベントが使用されます。通知を使用すると、TCP タイムアウトによるクライアントの待機を回避できます。イベントは、Oracle JDBC 接続プール、Oracle Data Provider for .Net 接続プール、および透過的アプリケーション・フェイルオーバー（TAF）などの Oracle Call Interface に統合されます。

Oracle Data Guard を使用すると、本番サービスは本番サイトで提供されます。その他のスタンバイ・サイトでは、読取り専用モードでの動作中にレポート・サービスを提供できます。Oracle RAC と Data Guard Broker の統合により、フェイルオーバー、スイッチオーバーおよび保護モードが変更されると、当初の本番サイトの本番サービスが解体され、新しい本番サイトに本番サービスが構築されます。サービスをローカルに管理する Oracle Clusterware と遷移を管理する Data Guard の間では、コマンドの変更が制御されています。Data Guard の遷移が完了すると、Oracle Clusterware は高可用性操作の管理を自動的に再開します。

関連項目：

- 『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『Oracle Database パフォーマンス・チューニング・ガイド』
- 14-23 ページの「Oracle Scheduler」
- 14-18 ページの「データベース・リソース・マネージャの概要」

Oracle Scheduler

Oracle Database には、豊富な機能を備えたジョブ・スケジューラが組み込まれています。指定日時（平日の毎晩 11:00 など）または指定イベントの発生時（在庫が特定のレベルを下回ったときなど）に、実行するジョブをスケジューリングできます。会計四半期ごとの最後の勤務日などのカスタム・カレンダを定義できます。

DBMS_SCHEDULER パッケージまたは Oracle Enterprise Manager を使用して、ジョブ、プログラムおよびスケジュールなどのスケジューラ・オブジェクトを作成および操作します。スケジューラ・オブジェクトは標準データベース・オブジェクトであるため、システム権限およびオブジェクト権限により、これらのオブジェクトへのアクセスを制御できます。

プログラム・オブジェクト（またはプログラム）には、引数のデフォルト値を含む、スケジューラで実行されるコマンドに関するメタデータが含まれます。スケジュール・オブジェクト（スケジュール）には、実行日時および反復パターンに関する情報が含まれます。ジョブ・オブジェクト（ジョブ）は、プログラムをスケジュールに対応付けるもので、スケジューラで操作する主要なオブジェクトです。同じプログラムを参照し、異なるスケジュールで実行される複数のジョブを作成できます。ジョブは、プログラム引数のデフォルト値を上書きする可能性があるため、複数のジョブが同じプログラムを参照できますが、異なる引数値を提供します。

スケジューラは、Oracle Enterprise Manager および SQL*Plus から使用できる各種ビューで、ジョブの包括的なロギングを提供します。指定した状態変更が発生したときにイベントを発生させるジョブを構成できます。アプリケーションによりイベントを処理し、適切な処理を実行できます。たとえば、ジョブが異常終了した場合、スケジューラは DBA に電子メールをページングまたは送信できます。

スケジューラには**連鎖**も含まれています。連鎖とは、連携してタスクを実行する名前付きのステップのグループです。連鎖内の各ステップは 1 つのプログラム、副鎖またはイベントであり、各ステップが実行されるタイミングと、各ステップ間の依存性を決定するルールを指定します。連鎖の一例は、「プログラム A および B を実行します。ただし、プログラム A および B が正常に完了した場合のみプログラム C を実行し、それ以外の場合はプログラム D を実行します。」のようになります。

スケジューラはデータベース・リソース・マネージャに統合されています。スケジューラ・ジョブをリソース・コンシューマ・グループに対応付け、異なる時間に異なるリソース・プランを自動的にアクティブ化する、ウィンドウと呼ばれるスケジューラ・オブジェクトを作成できます。リソース・プランに変更がある場合、ジョブを実行すると、ジョブに割り当てられたリソース内の変更を参照できます。スケジューラ・ジョブでは、スケジューリングの際に、スケジュール・オブジェクトではなくウィンドウに名前を付けることができます。そのようなジョブは、名前付きのウィンドウが開かれる際に実行されます。また、ウィンドウはウィンドウ・グループにグループ化でき、スケジュール時にウィンドウ・グループに名前を付けることが可能です。そのようなジョブは、名前付きのウィンドウ・グループの任意のウィンドウが開かれる際に実行されます。

> **関連項目：** スケジューラの概要およびスケジューラの使用方法と管理方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

スケジューラの機能

スケジューラには、企業向けの複雑なスケジューリング機能が用意されています。この機能を使用すると次の操作ができます。

- ジョブ実行スケジュール
- 時間ベースのスケジューリング
- イベントベースのスケジューリング
- 複数ステップ・ジョブの定義
- ビジネス要件をモデル化するジョブ・プロセスのスケジュール
- ジョブの管理と監視
- クラスタ環境におけるジョブの実行と管理

ジョブ実行スケジュール

ジョブ・スケジューラの最も基本的な機能は、ジョブの実行のスケジュールです。スケジューラでは、時間ベースとイベントベースの両方のスケジューリングがサポートされています。

時間ベースのスケジューリング

時間ベースのスケジューリングを使用すると、ユーザーは固定した日時（2006 年 1 月 23 日午前 1 時など）、繰返しスケジュール（毎週月曜日など）または定義済ルール（1 か月おきの最終日曜日、感謝祭を定義する 11 月の第 4 木曜日など）を指定できます。

ユーザーは、既存のスケジュールを組み合せることで、最小限の労力で新しい複合スケジュールを作成できます。たとえば、休日および平日スケジュールがすでに定義されている場合、平日スケジュールから休日スケジュールを除外することで、勤務日スケジュールを簡単に作成できます。

企業では、通常のカレンダとは対照的に会計カレンダを使用する場合が多いため、会計四半期の最後の勤務日にジョブをスケジュールする必要があります。スケジューラでは、ユーザーが毎月最後の勤務日のみでなく、会計四半期ごとの最後の勤務日も定義できる、ユーザー定義の頻度がサポートされています。

イベントベースのスケジューリング

イベントベースのスケジューリングでは、名前が意味するように、リアルタイムのイベントに基づいてジョブがトリガーされます。イベントは、ファイルの到着などシステムの状態の変化として定義されます。イベントに基づいてスケジューリングすると、ジョブを実行する日時について正確な時間が事前にわからない状況に対応できます。

複数ステップ・ジョブの定義

スケジューラでは、シングルステップまたは複数ステップのジョブがサポートされています。複数ステップ・ジョブは、連鎖を使用して定義されます。連鎖は、依存性ルールを使用して結合された複数のステップで構成されます。各ステップはタスクを表すため、連鎖を使用すると、ユーザーはタスク A およびタスク B が正常に完了した 1 時間後にタスク C を実行するなど、タスク間の依存性を指定できます。

ビジネス要件をモデル化するジョブ・プロセスのスケジュール

スケジューラを使用すると、ビジネス要件をモデル化する方法でジョブを処理できます。限られたコンピューティング資源を競合するジョブ間で適切に割り当てることができるため、ビジネス・ニーズにあわせてジョブ処理を実行できます。共通の特性および動作を共有するジョブを、ジョブ・クラスと呼ばれる大きいエンティティにグループ化できます。クラス間の優先順位は、各クラスに割り当てるリソースを制御することで設定します。これにより、重要なジョブに優先順位が設定され、完了に必要となる十分なリソースが確実に割り当てられます。また、ジョブ・クラス内で各ジョブの優先順位を設定することも可能です。

スケジューラには、スケジュールに基づいて優先順位を変更する機能も用意されています。重要なジョブの定義は変化することがあるため、スケジューラを使用して随時異なるクラス優先順位を定義できます。

ジョブの管理と監視

ジョブが作成されてから完了するまでには、複数の状態をたどります。すべてのスケジューラ・アクティビティは記録され、ジョブのステータスや完了時刻などの情報を容易に追跡できます。この情報はビューに格納されます。ビューの情報の問合せには、Oracle Enterprise Manager または SQL 問合せを使用できます。各ビューにはジョブとその実行に関する情報が表示され、ジョブの適切なスケジューリングと管理に役立ちます。たとえば、ユーザー scott に関して失敗したジョブをすべて容易に追跡できます。

ジョブの監視を容易にするために、ユーザーはスケジューラにフラグを設定して、予期しない動作が発生した場合にイベントを発生させ、指定したイベントが発生した場合に実行される処理を指定できます。たとえば、ジョブが失敗した場合、管理者に通知する必要があります。

クラスタ環境におけるジョブの実行と管理

クラスタとは、同じタスクを共同で実行するデータベース・インスタンスの集合です。Oracle Real Application Clusters は、アプリケーションを変更せずにスケーラビリティと信頼性を提供します。スケジューラは、このようなクラスタ環境におけるジョブの実行を全面的にサポートしています。システムの負荷バランスを調整してパフォーマンスを改善するために、ジョブを実行するサービスを指定することもできます。

関連項目：

- `DBMS_SCHEDULER` パッケージと `DBMS_FILE_TRANSFER` パッケージを使用したファイル転送の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 固定ユーザー・データベース・リンクの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

15

バックアップおよびリカバリ

バックアップおよびリカバリ・プロシージャにより、データベースのデータ消失が防止され、データは消失した場合にも再構築されます。この章では、バックアップおよびリカバリ方針を設計する際の基礎となる概念について説明します。

この章の内容は、次のとおりです。

関連項目：

- 1-21 ページの「データベースのバックアップおよびリカバリ機能の概要」
- バックアップとリカバリの概念およびタスクの詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

バックアップおよびリカバリの概要

バックアップとは、データのコピーです。このコピーには、ユーザー・データが含まれるデータファイル、および構成情報が含まれるサーバー・パラメータ・ファイルや制御ファイルなど、データベースの重要な部分を含めることができます。

バックアップの主要な目的は、予期しないデータ消失やアプリケーション・エラーに対する防護策です。たとえば、ディスク障害が原因でデータファイルが失われたとします。データのバックアップをリストアし、メディア・リカバリを介して失われたデータを再構築できます。メディア・リカバリとは、データベース・ファイルのバックアップのリストア、ロールフォワードおよびロールバックにかかわる各種操作を意味します。

Oracle データベースのバックアップおよびリカバリの実行には、**Recovery Manager**（RMAN）を使用する方法と、ユーザーが管理する方法の 2 つがあります。RMAN は、データベース・ファイルのバックアップ、リストアおよびリカバリに使用される Oracle Database ユーティリティです。Oracle Database の機能であるため、別途インストールする必要はありません。また、バックアップにはオペレーティング・システム・コマンド、メディア・リカバリには SQL*Plus を使用することもできます。この方法はユーザー管理バックアップおよびリカバリとも呼ばれ、Oracle で全面的にサポートされています。ただし、より堅牢で管理作業が簡素化されるため、RMAN を使用することをお薦めします。

Oracle Flashback テクノロジは、従来のバックアップおよびリカバリに代わるテクノロジです。フラッシュバック機能を使用すると、バックアップからのデータのリストアを行わずに、過去のデータ状態を表示することや特定の時間にあわせてデータを前後に移動することが可能です。1 つのコマンドを発行することで、データベース全体または単一の表を過去のある時点に巻き戻しできます。Oracle Database のフラッシュバック機能は、該当する状況のほとんどにおいて、メディア・リカバリよりもさらに効率的でわかりやすい機能です。

どのようなバックアップおよびリカバリ・ツールを使用している場合でも、フラッシュ・リカバリ領域を構成してリカバリ関連ファイルを管理することをお薦めします。

フラッシュ・リカバリ領域

フラッシュ・リカバリ領域は、オプションで Oracle Database によって管理されるディレクトリ、ファイル・システムまたは自動ストレージ管理ディスク・グループで、バックアップおよびリカバリ・ファイルの一元的な格納場所です。フラッシュ・リカバリ領域は、Database Configuration Assistant を使用してデータベースを作成する際に構成することも、後から追加することもできます。

Oracle Database では、フラッシュ・リカバリ領域にアーカイブ・ログを作成できます。RMAN はフラッシュ・リカバリ領域にバックアップを格納し、メディア・リカバリ中にそれをフラッシュ・リカバリ領域からリストアできます。フラッシュ・リカバリ領域は、テープ用のディスク・キャッシュとしても機能します。

Oracle Database のリカバリ・コンポーネントがフラッシュ・リカバリ領域と対話して、リカバリ領域に格納されたファイルを使用してデータベースを完全にリカバリできるようにします。メディア障害後のデータベースのリカバリに必要なファイルは、すべてフラッシュ・リカバリ領域に属します。

フラッシュ・リカバリ領域には、次に示すリカバリ関連ファイルが格納されます。

- 現行の制御ファイル
- オンライン REDO ログ
- アーカイブ REDO ログ
- フラッシュバック・ログ
- 制御ファイルの自動バックアップ
- データファイルおよび制御ファイルのコピー
- バックアップ・ピース

Oracle Database ではディスク領域の制限を定義して、フラッシュ・リカバリ領域に使用できるディスク領域を制限できます。このディスク制限により、ディスク全体をフラッシュ・リカバリ領域専用にするのではなく、残りのディスク領域を他の用途に使用できます。この領域に、Oracle Database では認識されないオーバーヘッドは含まれません。たとえば、ディスク制限には、圧縮、ミラー化、または他のなんらかの冗長性メカニズムの使用によるファイル・システムの余分なサイズは含まれません。

Oracle Database および RMAN は、使用領域がリカバリ用のディスク領域の制限に達するまで、フラッシュ・リカバリ領域にファイルを作成します。新しいファイル用の領域が必要になった場合、Oracle Database は、失効したファイル、冗長なファイルまたは 3 次記憶装置にバックアップ済のファイルをフラッシュ・リカバリ領域から削除します。使用可能なディスク領域が 15% 未満になると Oracle Database により警告が発行されますが、制限されたディスク領域の 100% に達するまで使用し続けることができます。

フラッシュ・リカバリ領域は、大きいほど有用になります。推奨ディスク制限は、データベース・サイズ、増分バックアップのサイズ、およびテープにコピーされていない全アーカイブ・ログのサイズの合計です。

関連項目：

- ファイル削除の優先順位を定義するルールと、フラッシュ・リカバリ領域に関するその他の情報は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- フラッシュ・リカバリ領域の設定および管理方法は、『Oracle Database 管理者ガイド』を参照してください。

データベースのバックアップ

この項では、物理バックアップについて説明します。この項の内容は、次のとおりです。

データベース・バックアップについて

データベース・バックアップは、物理または論理のいずれかです。物理バックアップはバックアップおよびリカバリ方針の中心概念であり、物理データベース・ファイルのコピーです。物理バックアップを作成するには、RMAN ユーティリティを使用する方法と、オペレーティング・システムのユーティリティを使用する方法があります。

これに対して、論理バックアップには、表やストアド・プロシージャなどの論理データが含まれます。Oracle Database のデータ・ポンプ・エクスポートなどのユーティリティを使用して、論理データを抽出してバイナリ・ファイルに格納できます。論理バックアップは、物理バックアップを補足できます。

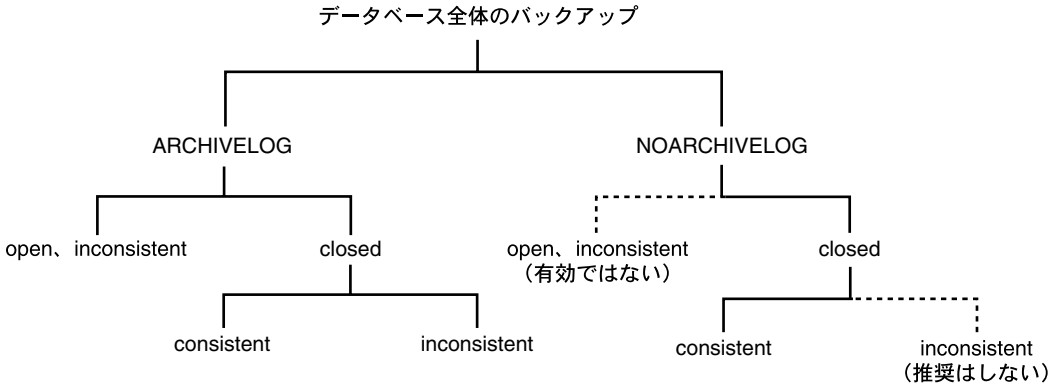
データベース・バックアップの主要な目的はデータ保護ですが、アーカイブ・データベース・バックアップを作成してデータを保存することもできます。たとえば、顧客のトランザクション・レコードを指定された期間保存するビジネス要件があるとします。RMAN を使用すると、一貫性を保つために必要な REDO と、データベースのアーカイブ・バックアップを外部記憶域に作成できます。不要なバックアップの削除を管理する RMAN の保存ポリシーから、このデータベース・バックアップを除外する期間を制御できます。

データベース全体のバックアップと部分バックアップ

データベース全体のバックアップとは、データベース内の各データファイルと制御ファイルのバックアップです。これは最も一般的なタイプのバックアップです。

図 15-1 に示すように、データベース全体のバックアップは ARCHIVELOG または NOARCHIVELOG モードで作成でき、**一貫性バックアップ**または**非一貫性バックアップ**のいずれかです。バックアップに一貫性を持たせるには、バックアップをリストアした後に REDO ログを適用する必要があります。

図 15-1 データベース全体のバックアップのオプション



部分バックアップには、個々の表領域またはデータファイルであるデータベースのサブセットが含まれます。表領域のバックアップとは、表領域を構成するデータファイルのバックアップです。オンラインかオフラインかにかかわらず、表領域のバックアップが有効なのはデータベースが ARCHIVELOG モードで稼働している場合のみです。これは、リストア後の表領域とデータベースの他の表領域との一貫性を保つには、REDO が必要であるためです。

データファイルのバックアップとは、1 つのデータファイルのバックアップです。この種のバックアップは、表領域のバックアップほど一般的ではありませんが、ARCHIVELOG モードのデータベースには有効です。

> **関連項目：** 論理バックアップの詳細は、『Oracle Database バックアップおよびリカバリ・リファレンス』および『Oracle Database ユーティリティ』を参照してください。

一貫性バックアップと非一貫性バックアップ

データベース・バックアップは、一貫性バックアップか非一貫性バックアップのいずれかです。この項ではその違いを説明します。

この項の内容は、次のとおりです。

一貫性バックアップの概要

データベースの一貫性バックアップにおいて、すべての読取り / 書込みデータファイルおよび制御ファイルでは、同じ**システム変更番号**（SCN）を使用してチェックポイントが実行されます。バックアップ内のファイルは、同じ SCN までのすべての変更内容が含まれていることが保証されます。非一貫性バックアップとは異なり、一貫性のあるデータベース全体のバックアップの場合、リストア後にリカバリする必要はありません。

一貫性のあるデータベース全体のバックアップを作成するには、NORMAL、IMMEDIATE または TRANSACTIONAL オプションを指定してデータベースを停止し、データベースがクローズしている間にバックアップを作成する必要があります。データベースが**読取り専用データベース**でないかぎり、インスタンス障害が発生した場合や、SHUTDOWN ABORT 文を発行した場合など、データベースが正常に停止していないと、常にそのデータファイルの一貫性が損われます。

データはすでに一貫した状態になっているため、一貫性のあるデータベース全体のバックアップをリストアした後は、そのままデータベースをオープンでき、リカバリ不要であることに注意してください。リストアしたデータファイル内のデータを正常にするための処理は不要です。そのため、メディア・リカバリやデータベースによるインスタンス・リカバリが実行されなくても、データベースの 1 年前の一貫性バックアップをリストアできます。

注意： REDO を適用せずに一貫性のあるデータベース全体のバックアップをリストアすると、バックアップの実行後に実行されたトランザクションはすべて失われます。

一貫性のあるデータベース全体のバックアップは、NOARCHIVELOG モードで稼働しているデータベースでのみ有効なバックアップ・オプションです。他のバックアップ・オプションでは一貫性のためリカバリを必要としますが、これはアーカイブ REDO ログがなければ不可能です。

一貫性のあるデータベース全体のバックアップは、ARCHIVELOG モードで稼働しているデータベースにも有効なバックアップ・オプションです。このタイプのバックアップをリストアしており、アーカイブ・ログが使用可能な場合は、2 つの選択肢があります。つまり、データベースを即時にオープンし、バックアップの作成後に実行されたトランザクションを消失させる方法と、アーカイブ・ログを適用して、消失したトランザクションをリカバリする方法です。

非一貫性バックアップの概要

データベースの非一貫性バックアップでは、読取り / 書込みデータファイルおよび制御ファイルは、同じ SCN までのチェックポイントの実行を保証されていません。バックアップのファイルには、異なる時点から取得されたデータが含まれている場合があります。つまり、変更内容が欠落している可能性があります。この状況が発生するのは、バックアップの実行中にデータファイルが変更されている場合などです。

データベースがオープン状態またはマウントされているときにデータベースをバックアップすると、非一貫性バックアップが作成されます。オンライン・データファイルのバックアップを**オンライン・バックアップ**と呼びます。オンライン・バックアップは、データベースを ARCHIVELOG モードで実行する必要があります。

ARCHIVELOG モードでデータベースを実行し、アーカイブ REDO ログおよびデータファイルをバックアップするかぎり、非一貫性バックアップを堅実なバックアップおよびリカバリ計画の基礎とできます。データベースを完全保護するバックアップの作成にデータベースを停止する必要がないため、非一貫性バックアップには優れた可用性があります。

Oracle Database のリカバリでは、データファイルのヘッダーにある最小 SCN から順にすべてのアーカイブ REDO ログとオンライン REDO ログを読み取り、ログからの変更をデータファイルに適用することで、非一貫性バックアップに一貫性を持たせます。非一貫性バックアップの作成後は必ず、アーカイブされていない REDO ログをアーカイブして、バックアップのリカバリに必要な REDO が揃っていることを確認します。バックアップ中に生成されたアーカイブ REDO ログがすべて揃っていないと、一貫性を保つために必要な REDO に欠落があるため、バックアップをリカバリできません。

> **関連項目：**『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』

Recovery Manager によるバックアップとユーザー管理バックアップ

RMAN の BACKUP コマンドでは、イメージ・コピーまたはバックアップ・セットが生成されます。イメージ・コピーとは、データファイル、制御ファイルまたはアーカイブ・ログを正確に複製したものです。オペレーティング・システムのユーティリティまたは Recovery Manager を使用して物理ファイルのイメージ・コピーを作成し、それ以上は処理を実行せずに、それをそのままリストアできます。

注意： オペレーティング・システムによるコピーとは異なり、RMAN ではファイル内のブロックが検証され、イメージ・コピーがリポジトリに記録されます。

バックアップ・セットとは、バックアップ・ピースと呼ばれる 1 つ以上の物理ファイルで構成される、固有のフォーマットによるバックアップです。バックアップ・セットには、複数のデータファイルが含まれます。バックアップ・セットの最小単位は、バックアップ・ピースと呼ばれるバイナリ・ファイルです。RMAN のみが作成およびアクセスするバックアップ・セットは、RMAN によるテープ・ドライブなどのシーケンシャル・デバイスへのバックアップの書込みが可能な唯一の形式です。

この項の内容は、次のとおりです。

- オンライン・バックアップ
- 制御ファイルのバックアップ
- アーカイブ REDO ログのバックアップ

オンライン・バックアップ

データベースではオンライン・バックアップ中にもファイルへの書込みが続くため、ブロック内に一貫性のないデータがバックアップされる可能性があります。たとえば、データベース・ライターによるブロックの更新中に、Recovery Manager またはオペレーティング・システムのユーティリティがそのブロックを読み取る場合を考えます。この場合、Recovery Manager またはコピー・ユーティリティは、ブロックの前半にある新しいデータと、後半にある古いデータを読み取る可能性があります。このブロックは分裂ブロックであり、このブロック内のデータには一貫性がないことを意味します。

オペレーティング・システムのユーティリティではなく Recovery Manager によるバックアップ中には、Oracle データベースはデータファイルを読み取ります。各ブロックを読み取って、ブロックが分裂しているかどうかを判断します。ブロックが分裂している場合、データベースは、有効なブロックを取得するまでブロックを読み取ります。

Recovery Manager ではなくオペレーティング・システムのユーティリティを使用してオンライン・データファイルのバックアップを実行する場合は、異なる方法で分裂ブロックを処理する必要があります。まず、`ALTER TABLESPACE BEGIN BACKUP` 文（個々の表領域をバックアップ）、または `ALTER DATABASE BEGIN BACKUP` 文（データベース全体をバックアップ）を使用して、ファイルをバックアップ・モードにする必要があります。オンライン・バックアップが完了したら、`ALTER TABLESPACE...END BACKUP` 文または `ALTER DATABASE END BACKUP` 文を実行して、ファイルのバックアップ・モードを解除する必要があります。

バックアップ・モードでファイルが更新されると、追加の REDO データがログに記録されます。この追加データは、オペレーティング・システムのユーティリティでバックアップが作成された分裂ブロックの修復に必要となります。

制御ファイルのバックアップ

制御ファイルのバックアップは、バックアップおよびリカバリの重要な側面です。制御ファイルがなければ、データベースのマウントもオープンもできません。RMAN では、`CONFIGURE CONTROLFILE AUTOBACKUP ON` を使用することで、バックアップ・ジョブを実行するたびに制御ファイルが自動的にバックアップされるように指定できます。自動バックアップではデフォルトのファイル名が使用されるため、Recovery Manager リポジトリが使用できない場合にも、Recovery Manager はこのバックアップをリストアできます。そのため、この機能は障害時リカバリにきわめて役立ちます。

制御ファイルの手動バックアップを実行するには、次の方法を使用できます。

- Recovery Manager の `BACKUP CURRENT CONTROLFILE` コマンドを実行すると、制御ファイルのバイナリ・バックアップがバックアップ・セットまたはイメージ・コピーとして作成されます。
- SQL 文 `ALTER DATABASE BACKUP CONTROLFILE` を実行すると、制御ファイルのバイナリ・バックアップが作成されます。
- SQL 文 `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` を実行すると、制御ファイルの内容が SQL スクリプト・ファイルにエクスポートされます。このスクリプトを使用して新規制御ファイルを作成できます。トレース・ファイルのバックアップには、1 つ重大なデメリットがあります。つまり、アーカイブ REDO ログのレコード、Recovery Manager によるバックアップおよびコピーが含まれません。このため、バイナリ・バックアップの方が適切です。

関連項目：

- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』
- 『Oracle Database バックアップおよびリカバリ・リファレンス』

アーカイブ REDO ログのバックアップ

アーカイブ REDO ログを使用して、バックアップをある時点にロールフォワードできます。最新のアーカイブ REDO ログでバックアップをリカバリするには、バックアップの作成後に生成されたすべてのログが使用可能であることが必要です。つまり、ログ 173 が欠落している場合、アーカイブ REDO ログ 100 ～ログ 200 までのリカバリはできません。この場合はログ 172 の適用後リカバリを停止し、`RESETLOGS` オプションを指定してデータベースをオープンする必要があります。

アーカイブ REDO ログはリカバリに不可欠であるため、それらのバックアップを定期的に作成する必要があります。メディア・マネージャを使用する場合は、定期的にテープにログをバックアップします。アーカイブ・ログのバックアップを作成するには、次の方法を使用できます。

- Recovery Manager の `BACKUP ARCHIVELOG` コマンド
- Recovery Manager の `BACKUP...PLUS ARCHIVELOG` コマンド
- オペレーティング・システムのユーティリティ

関連項目：

- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』
- 『Oracle Database バックアップおよびリカバリ・リファレンス』

データ修復が必要な問題

次に示す障害には、DBA が対応する必要があります。データベース・インスタンスがクラッシュする可能性がありますが、データ損失の原因になることや、バックアップからのリカバリが必要になることはほとんどありません。

- インスタンス障害
- ネットワーク障害
- Oracle Database のバックグラウンド・プロセスの障害
- データファイルの領域など、なんらかのリソース不足を原因とする文の実行の失敗

通常、メディア障害またはユーザー・エラーの場合に、データ・リカバリを行います。

この項の内容は、次のとおりです。

- メディア障害
- ユーザー・エラー

メディア障害

メディア障害は、データベース外部の問題により、Oracle Database 操作中にファイルの読取りまたは書込みができない場合に発生します。一般的なメディア障害には、ヘッド・クラッシュなどの物理的な障害、およびデータベース・ファイルの上書き、削除または破損が含まれます。メディア障害は、ユーザーまたはアプリケーションのエラーより頻度は低いですが、バックアップおよびリカバリ計画ではこれに備えておく必要があります。

オンライン REDO ログ・ファイルまたは制御ファイルのメディア障害後のデータベース操作は、**多重化**によりファイルが保護されているかどうかで異なります。オンライン REDO ログまたは制御ファイルが多重化されている場合は、データベースによりファイルのコピーが複数維持されています。

多重化されているオンライン REDO ログのコピーが 1 つ含まれるディスクがメディア障害により破損しても、データベースは通常、それほど問題なく稼働し続けます。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、データが完全に失われる場合があります。

制御ファイルが破損すると、多重化されているかどうかにかかわらず、データベースにより、破損した制御ファイルの読取りまたは書込みが試行された時点でデータベースが停止します。チェックポイントごとおよびオンライン REDO ログの切替え時などに、データベースは制御ファイルに頻繁にアクセスします。

メディア障害は、**読取りエラー**または**書込みエラー**のいずれかです。読取りエラーでは、インスタンスによるデータファイルの読取りが実行できません。また、ファイルが存在しない、ファイルを開けない、ファイルを読み取れないなどのエラーとともに、アプリケーションにオペレーティング・システム・エラーが戻されます。データベースは稼働し続けますが、読取りが失敗するたびにエラーが戻されます。次のチェックポイントでは、データベースにより、チェックポイント処理の一部としてデータファイル・ヘッダーへの書込みが試行されると書込みエラーが発生します。

データファイル書込みエラーの影響は、データファイルがどの表領域に存在するかによって異なります。**SYSTEM 表領域**のデータファイル、UNDO 表領域、またはアクティブなロールバック・セグメントのあるデータファイルにインスタンスが書込みを実行できない場合は、データベースによりエラーが発行され、データベースは停止します。SYSTEM 表領域のすべてのファイルおよび UNDO またはロールバック・セグメントを含むすべてのデータファイルは、データベースが正常に稼働するためにオンラインである必要があります。

インスタンスが前述のリストにないデータファイルへの書込みを実行できない場合、結果は、データベースが ARCHIVELOG モードで実行されているかどうかに依存します。ARCHIVELOG モードでは、データベースによりエラーがデータベース・ライター・トレース・ファイルに記録され、影響を受けたデータファイルがオフラインになります。このデータファイルを含む表領域のその他すべてのデータファイルはオンラインのままです。原因である問題を修正し、影響を受けた表領域をリストアしてリカバリします。

NOARCHIVELOG モードでは、データベース・ライター・バックグラウンド・プロセスが停止するとインスタンスも停止します。問題の原因により必要な対応が決まります。問題が一時的な場合は、通常、オンライン REDO ログ・ファイルを使用してクラッシュ・リカバリが実行されます。そのような場合には、メディア・リカバリは行わずにインスタンスを再起動できます。ただし、データファイルが破損している場合は、データベース全体の**一貫性バックアップ**をリストアする必要があります。

ユーザー・エラー

不適切な更新、表の内容の削除またはデータベース・オブジェクトの削除など、ユーザーまたはアプリケーションにより、データベースに不要な変更が行われる場合があります。適切なバックアップおよびリカバリ計画では、多数の Oracle Database 機能を使用して、データベースの可用性に対する影響を最小化し、DBA の労力を最低限に抑えて、必要な状態にデータベースを戻すことができます。

関連項目：

- データベース全体に対してポイント・イン・タイム・リカバリを実行する方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- 表領域のポイント・イン・タイム・リカバリを実行する方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- Oracle Database のフラッシュバック機能の使用方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

データ修復

15-8 ページの「データ修復が必要な問題」で説明されている問題の解決方法は複数あります。

データ・リカバリ・アドバイザは、診断および修復に関する様々な作業を自動的に行うことができる統合ソリューションです。データ・リカバリ・アドバイザは、障害を診断し、手動および自動の両方の修復オプションを提供し、場合によっては自動的に障害が修復されることもあります。

論理データの破損やユーザー・エラーによる問題を解決するために、メディア・リカバリのかわりとして Oracle Flashback を使用できます。Oracle Flashback 機能を使用すると、データベース全体またはデータベースのサブセットを以前の状態に巻き戻しできます。

メディア障害を修正するには、メディア・リカバリを使用します。メディア・リカバリでは、失われた変更を更新するために、バックアップに REDO または増分バックアップが適用されます。ブロック・メディア・リカバリはさらに特化されている操作で、1 つ以上のファイル内で少数のブロックのみが破損している場合に使用します。

この項の内容は、次のとおりです。

- データ・リカバリ・アドバイザ
- Oracle Flashback テクノロジ
- メディア・リカバリ

関連項目：

- 『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』
- 『Oracle Database バックアップおよびリカバリ・リファレンス』

データ・リカバリ・アドバイザ

Oracle Database には、**データ・リカバリ・アドバイザ**・ツールが組み込まれています。このツールを使用すると、永続的なデータ障害が自動的に診断され、適切な修復オプションが提示されて、ユーザーの要求に応じて実行されます。データ・リカバリ・アドバイザは、Enterprise Manager インタフェースまたは RMAN クライアントを介して使用できます。

チェッカは、データベースまたはそのコンポーネントの状態を評価するために状態モニターに登録されている診断操作またはプロシージャです。状態評価はデータ整合性チェックとも呼ばれ、対処的または予防的に起動できます。

障害は通常、対処的に検出されます。破損データが関与するデータベースの操作はエラーとなりますが、これにより自動的にデータ整合性チェックが起動され、エラーに関連する障害がないかデータベースが検索されます。障害が診断された場合、その障害は自動診断リポジトリ（ADR）に記録されます。状態モニターを使用して、または Recovery Manager の `VALIDATE` および `BACKUP` コマンドによるブロック障害のチェックによって、データ整合性チェックを予防的に起動することもできます。

データベースによって障害が検出され、ADR に格納された後にのみ、データ・リカバリ・アドバイザを使用して修復に関するアドバイスを生成し、障害を修復できます。各障害には、オープンまたはクローズというステータスがあります。また、クリティカル、高、低という優先度もあります。優先度がクリティカルの障害は、データベース全体が使用不可になるため、迅速に対応する必要があります。優先度が高の障害では、データベースが部分的に使用不可またはリカバリ不可能になるため、通常はできるだけ時間が経過しないうちに修復する必要があります。優先度が高の障害には、データ・ブロックの破損および致命的ではない I/O エラーが含まれます。優先度が低の障害は、より重要な障害が修正されるまで待機可能です。

データ・リカバリ・アドバイザは、最善の修復オプションおよびそのデータベースへの影響を自動的に判断します。通常、データ・リカバリ・アドバイザは、それぞれの障害または障害グループに対して手動および自動の両方の修復オプションを生成します。手動のオプションは、必須またはオプションのいずれかとして分類されます。

データ・リカバリ・アドバイザは、自動修復オプションを提示する前に、修復を完了するために必要なメディア・コンポーネントが使用可能かどうか、およびそのオプションが特定の環境に適しているかどうかを検証します。自動修復を選択すると、Oracle Database によって自動的に修復されます。データ・リカバリ・アドバイザ・ツールは、修復の成功を確認し、該当する障害をクローズします。

関連項目：

- RMAN コマンドライン・インタフェースでのデータ・リカバリ・アドバイザの使用方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- Enterprise Manager でのデータ・リカバリ・アドバイザの使用方法の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

Oracle Flashback テクノロジ

Oracle Database には、Oracle Flashback テクノロジと呼ばれる機能グループが提供されています。フラッシュバック・テクノロジは、バックアップからのデータベースのリストアを必要とせず、過去のデータの状態を表示したり、特定の時間にあわせてデータを前後に巻き戻したりする機能を備えています。データベースへの変更内容によりますが、フラッシュバック機能では、メディア・リカバリよりも迅速で、データベースの可用性に対する影響も少ない状態で、不要な変更を元に戻すことができます。

> **関連項目：** バックアップおよびリカバリには直接関係のない機能も含むすべての Oracle Flashback 機能の概要は、1-22 ページの「高可用性機能の概要」を参照してください。

この項の内容は、次のとおりです。

Oracle Flashback Database

Oracle Flashback Database を使用すると、Oracle データベースを以前の時点まで巻き戻し、論理データの破損やユーザー・エラーによる問題を解決できます。

フラッシュ・リカバリ領域が構成され、フラッシュバック・データベース機能が使用可能な場合、RMAN または SQL の `FLASHBACK DATABASE` コマンドを使用して、データベースを過去のある時点に戻すことができます。フラッシュバック・データベースは、物理ファイルのリストアを伴わないため、実際のメディア・リカバリではありません。迅速かつ容易であり、データベース全体のリストアを必要としないため、`RESTORE` および `RECOVER` コマンドを使用するよりもフラッシュバック・データベースの方が適切な場合があります。

フラッシュバック・データベースを使用すると、Oracle Database は、ブロックの過去のイメージを使用してデータベースに対する変更をバック・アウトします。これらのブロック・イメージは、Oracle Database により、通常のデータベース処理の実行中に、フラッシュバック・ログに書き込まれます。フラッシュバック・ログは順次書き込まれ、アーカイブされません。Oracle Database により、フラッシュバック・ログは、フラッシュ・リカバリ領域内で自動的に作成、削除およびサイズ変更されます。フラッシュバック・ログに注意する必要があるのは、パフォーマンスを監視する場合と、フラッシュバック・ログ用のフラッシュ・リカバリ領域に割り当てるディスク容量を決定する場合のみです。

`FLASHBACK DATABASE` を使用してデータベースの巻戻しに要する時間は、さかのぼる必要がある過去の時点や、目標時間以後のデータベース・アクティビティの量に比例します。データベース全体のリストアおよびリカバリの所要時間は、長時間かかる場合があります。フラッシュバック・ログ内のビフォア・イメージは、データベースを過去のある時点までリストアするためにのみ使用され、データベースを過去のある時点の一貫性のある状態まで戻すにはフォワード・リカバリが使用されます。Oracle Database はデータファイルを以前の特定時点まで戻しますが、初期化パラメータ・ファイルなどの補助ファイルは特定次点まで戻されません。

フラッシュバック・データベースは、Data Guard、リカバリ・アドバイザの補完、およびクローン・データベースの同期化にも使用できます。

関連項目：

- Oracle Flashback Database の使用方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- FLASHBACK DATABASE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- フラッシュバック・データベースが Oracle Data Guard を補完する方法の詳細は、『Oracle Data Guard 概要および管理』を参照してください。
- フラッシュバック・データベースの使用方法とリストア・ポイントの詳細は、『Oracle Database 高可用性概要』を参照してください。

Oracle Flashback Table

Oracle Flashback Table を使用すると、1 つの文で表を指定の時点まで巻き戻しできます。表データとともに、関連する索引、トリガーおよび制約をリストアできます。データベースがオンライン状態のときに、指定された表に対する変更のみが取り消されます。Oracle Flashback Table は、不良ディスクやデータ・セグメントと索引の非一貫性など、物理的な破損には対応しません。

Oracle Flashback Table は、セルフサービス修復ツールと同様に機能します。たとえば、偶然、表から重要な行を数行削除してしまい、それをリカバリするとします。FLASHBACK TABLE 文を使用すると、表を削除前の時点までリストアし、欠落していた表内の行を確認できます。

表とその内容を、特定の実時間またはユーザー指定のシステム変更番号（SCN）までリストアできます。Oracle Flashback Table を Oracle Flashback Version Query および Oracle Flashback Transaction Query とともに使用して、表をリストアする時点を調べます。

Oracle Flashback Table を正常に終了するには、システムに指定の SCN またはタイムスタンプを満たすために十分な UNDO 情報が必要であり、表に指定されている整合性制約には違反していない必要があります。また、表における行移動が有効化されている必要があります。

Oracle Flashback Table の保存された UNDO 情報の可用性は、自動的にチューニングされたシステムの UNDO 保存期間で制御されます。UNDO 保存期間とは、古い UNDO 情報（コミットされたトランザクションの UNDO 情報）が上書き可能になるまでの時間を示します。データベースにより、使用統計が収集され、これらの統計および UNDO 表領域サイズに基づいて UNDO 保存期間がチューニングされます。UNDO_RETENTION 初期化パラメータを設定することで、最低限の UNDO 保存期間をリクエストできます。

注意： UNDO 保存の自動チューニングは、データベースが自動 UNDO 管理モード（デフォルト）の場合にのみ実行されます。最低限の UNDO 保存期間のリクエストは、データベースで受け入れられる場合と受け入れられない場合があります。これは、システムの現在のトランザクション・アクティビティ、UNDO 表領域が自動拡張であるか固定サイズであるか、UNDO 表領域に RETENTION GUARANTEE を指定したかどうかなど、様々な要因に依存します。

UNDO 保存の自動チューニングの詳細は、『Oracle Database 管理者ガイド』を参照してください。

関連項目：

- 2-18 ページの「自動 UNDO 保存」
- Oracle Flashback Table の使用方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- UNDO_RETENTION 初期化パラメータおよび FLASHBACK TABLE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

Oracle Flashback Drop

Oracle Flashback Drop は、DROP TABLE 操作の効果を無効にします。フラッシュバック・ドロップは、ポイント・イン・タイム・リカバリなど、このような状況で使用できるその他のリカバリ・メカニズムより大幅に速く、最近のトランザクションの損失や停止時間は発生しません。

表を削除しても、データベースではその表に関連する領域はすぐには削除されません。かわりに、その表は名前が変更され、関連するオブジェクトとともにデータベースのごみ箱に配置されます。Oracle Database では、削除されたデータベース・オブジェクトにより占有されている領域が新しいデータの保存に必要になるまで、それらのオブジェクトはごみ箱を使用して管理されます。ごみ箱は、実際は、削除されたオブジェクトに関する情報を含むデータ・ディクショナリ表です。

> **関連項目：** Oracle Flashback Drop の使用方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

メディア・リカバリ

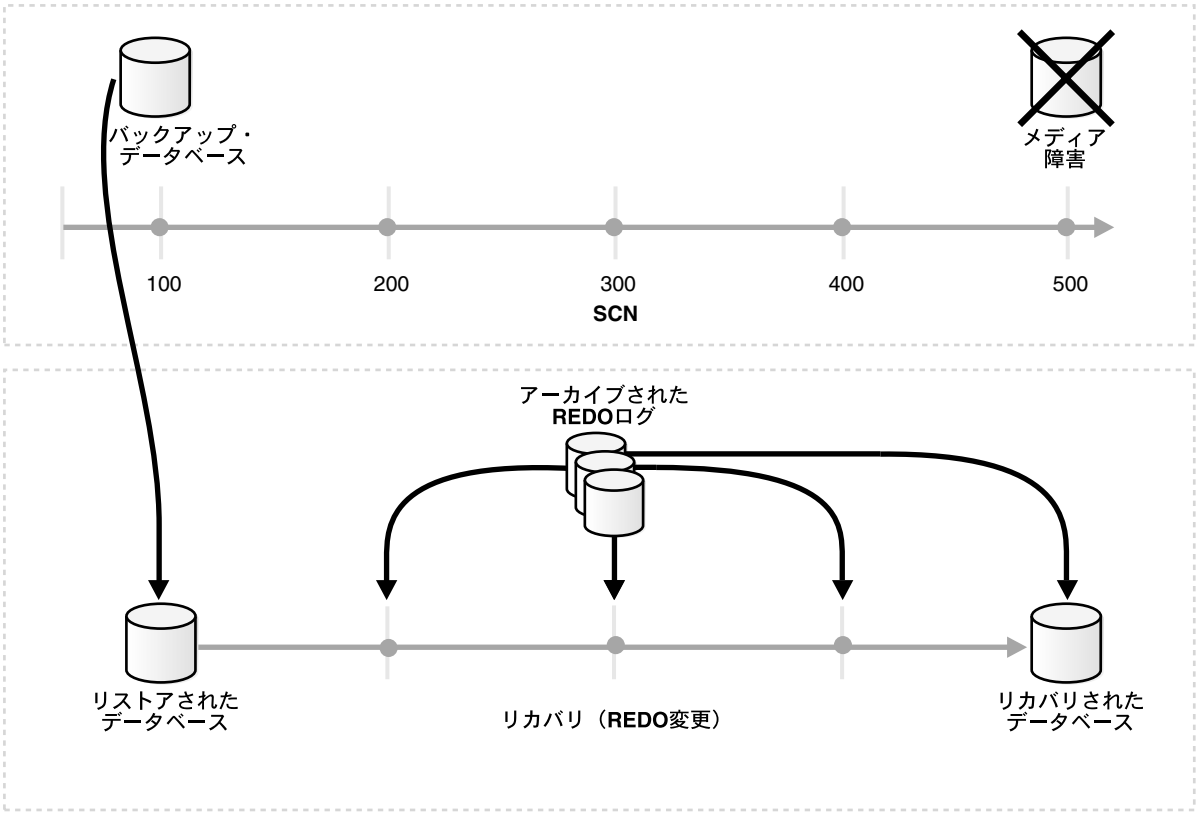
データファイルまたは制御ファイルの物理バックアップのリストアとは、それを再構築して Oracle データベースで使用可能にすることです。リストアされたデータファイルのリカバリとは、アーカイブ REDO ログとオンライン REDO ログ、つまり、バックアップの実行後にデータベースに対して行われた変更のレコードを適用して更新することです。Recovery Manager を使用すると、増分バックアップを使用してデータファイルをリカバリすることもできます。増分バックアップは、前回の増分バックアップ後に変更があったブロックのみを含むデータファイルのバックアップです。

必要なファイルがリストアされた後、ユーザーがメディア・リカバリを実行する必要があります。メディア・リカバリは、データベース・ファイルのリストア、ロールフォワードおよびロールバックのための各種操作を伴います。

メディア・リカバリでは、アーカイブ REDO ログとオンライン REDO ログが適用され、データファイルがリカバリされます。データファイルに変更がある度に、その変更はオンライン REDO ログに最初に記録されます。メディア・リカバリでは、オンラインおよびアーカイブ REDO ログに記録された変更がリストア後のデータファイルに選択的に適用され、ロールフォワードされます。

図 15-2 に、データベース上でのバックアップ、リストアおよびメディア・リカバリ実行の基本原理を示します。

図 15-2 メディア・リカバリ



メディア・リカバリとは異なり、Oracle Database では、クラッシュ・リカバリとインスタンス・リカバリはインスタンス障害の発生後に自動的に実行されます。クラッシュ・リカバリとインスタンス・リカバリでは、データベースはインスタンス障害直前のトランザクション一貫性のある状態までリカバリされます。クラッシュ・リカバリとは、単一インスタンス構成または Oracle Real Application Clusters 構成において、すべてのインスタンスがクラッシュした後にデータベースをリカバリすることです。これに対してインスタンス・リカバリとは、Oracle Real Application Clusters 構成において、障害が発生した 1 つ以上のインスタンスを稼働中のインスタンスでリカバリすることです。

この項の内容は、次のとおりです。

- データファイル・メディア・リカバリ
- ブロック・メディア・リカバリ
- 完全リカバリ
- データベースのポイント・イン・タイム・リカバリ
- RMAN によるリカバリとユーザー管理リカバリ

データファイル・メディア・リカバリ

データファイル・メディア・リカバリを使用するのは、消失または破損した現行のデータファイルまたは制御ファイルをリカバリする場合です。また、OFFLINE NORMAL オプションを指定せずに表領域がオフライン化され、消失した変更をリカバリする場合にも使用します。データファイル・メディア・リカバリとインスタンス・リカバリでは、どちらもデータベースの整合性を修復する必要があります。ただし、この 2 つのタイプのリカバリは、追加の機能に違いがあります。メディア・リカバリには次の特性があります。

- 破損データファイルのリストア後のバックアップに変更を適用します。
- アーカイブ・ログとオンライン・ログを使用できます。
- ユーザーが明示的に起動する必要があります。
- メディア障害（つまり、バックアップのリストアの必要性）は自動的に検出されません。ただし、バックアップのリストア後は、メディア・リカバリを介してリカバリする必要があることが自動的に検出されます。
- リカバリ時間は、Oracle Database 内部のメカニズムではなくユーザーのポリシー（バックアップ間隔、パラレル・リカバリ・パラメータ、前回のバックアップ以降のデータベース・トランザクション数など）でのみ制御されます。

オンライン・データファイルのメディア・リカバリが必要な場合は、データベースをオープンできません。また、メディア・リカバリが完了するまでは、メディア・リカバリを必要とするデータファイルはオンライン化できません。次の使用例では、メディア・リカバリが必要です。

- データファイルのバックアップをリストアする場合
- バックアップ制御ファイルをリストアする場合（全データファイルが最新の状態の場合も）
- データファイルが OFFLINE NORMAL オプションを指定せずに（ユーザーにより、または Oracle Database により自動的に）オフライン化される場合

データベースがインスタンスによりオープンされていれば、データファイルのメディア・リカバリはオフライン・データファイルにのみ機能します。

ブロック・メディア・リカバリ

ブロック・メディア・リカバリは、すべてのデータベース・ファイルをオンラインで使用可能な状態に保ったまま、個々のデータ・ブロックをリストアしてリカバリするテクニックです。破損が少数のブロックのみに限られている場合は、データファイル・リカバリよりもブロック・メディア・リカバリが適しています。

> **関連項目：** ブロック・メディア・リカバリの実行方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

完全リカバリ

完全リカバリでは、アーカイブ・ログとオンライン・ログに含まれる REDO 変更すべてがバックアップに適用されます。通常、完全メディア・リカバリは、メディア障害によりデータファイルまたは制御ファイルが破損した後に実行します。完全リカバリは、データベース、表領域またはデータファイルに対して実行できます。

データベース全体の完全リカバリを実行する場合は、次の操作が必要です。

- データベースのマウント
- リカバリ対象となるデータファイルすべてがオンライン化されているかどうかの確認
- データベース全体のバックアップのリストア
- Recovery Manager の `RECOVER DATABASE` コマンドの実行（正しい REDO ログと増分バックアップが適用される）

表領域またはデータファイルの完全リカバリを実行する場合は、次の操作が必要です。

- データベースがオープンされている場合は、リカバリ対象となる表領域またはデータファイルのオフライン化
- リカバリ対象となるデータファイルのバックアップのリストア
- オンライン REDO ログまたはアーカイブ REDO ログ、あるいはその両方の適用

データベースのポイント・イン・タイム・リカバリ

データベースのポイント・イン・タイム・リカバリは不完全リカバリとも呼ばれ、データベースの以前のバージョンを生成します。つまり、リストアされたバックアップ以後に生成された REDO レコードすべてを適用するわけではありません。通常、データベース全体のポイント・イン・タイム・リカバリは、次の場合に実行します。

- メディア障害によりオンライン REDO ログの一部または全体が破棄された場合。
- ユーザーが意図せずに表を削除したなど、ユーザー・エラーが原因でデータが消失した場合。
- アーカイブ REDO ログが欠落しているために完全リカバリを実行できない場合。
- バックアップ制御ファイルによって完全なリカバリが可能な場合。Recovery Manager を使用している場合は、シームレスで自動的です。

データベースのポイント・イン・タイム・リカバリを実行するには、リカバリが必要な時点より前に作成されたバックアップからデータファイルをすべてリストアし、リカバリの完了時に `RESETLOGS` オプションを指定してデータベースをオープンする必要があります。`RESETLOGS` 操作では、データベースの新規インカネーションが作成されます。つまり、ログ順序 1 から始まるログ順序番号の新規ストリームを含むデータベースです。

不完全リカバリの実行後は、`OPEN RESETLOGS` コマンドを使用してデータベースを読取り / 書込みモードでオープンする前に、まず読取り専用モードでオープンし、データを検査して適切な時点までリカバリされたことを確認することをお薦めします。正しい時点までリカバリされていない場合、`OPEN RESETLOGS` を実行していなければ、リカバリを再実行する方が簡単です。データベースを読取り専用モードでオープンし、リカバリが十分に実行されていないことが判明した場合は、単に必要な時点までリカバリを再実行します。必要以上の時点までリカバリされたことが判明した場合は、データベースをリストアしなおしてからリカバリを再実行する必要があります。

注意： フラッシュバック・データベースは、データベースのポイント・イン・タイム・リカバリの代替方法です。

関連項目： 15-12 ページの「Oracle Flashback Database」

表領域のポイント・イン・タイム・リカバリ 表領域のポイント・イン・タイム・リカバリ（TSPITR）機能を使用すると、1 つ以上の表領域をデータベースの残りの部分より古い時点までリカバリできます。TSPITR が最も役立つのは、次の操作が必要な場合です。

- 間違った削除操作や表切捨て操作からリカバリする場合
- 論理的に破損した表をリカバリする場合
- 不適切なバッチ・ジョブや、データベースのサブセットにのみ影響した他の DML 文からリカバリする場合
- 1 つの独立スキーマを物理データベースの残りの部分とは異なる時点までリカバリする場合（1 つの物理データベースの個別の表領域に複数の独立スキーマが存在する場合）
- バックアップからデータベース全体をリストアしてから完全データベース・ロールフォワードを実行するのではなく、大規模データベース（VLDB）上で 1 つの表領域をリカバリする場合

TSPITR には、次の制限があります。

- SYSTEM 表領域、UNDO 表領域、またはロールバック・セグメントが含まれる表領域では使用できません。
- 相互に依存するデータを含む表領域は、一緒にリカバリする必要があります。たとえば、2 つの表が個別の表領域にあり、外部キーの関係がある場合は、両方の表領域を同時にリカバリする必要があります。一方のみのリカバリはできません。Oracle Database でこの制限を規定できるのは、データベース制約で明示的に宣言されたデータ関連が検出される場合です。データベース制約で宣言されていない他のデータ関連が存在する可能性があります。Oracle Database ではこのような関連を検出できず、DBA は常に一貫した表領域セットをリストアするように注意する必要があります。

> **関連項目：** TSPITR の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』および『Oracle Database バックアップおよびリカバリ・リファレンス』を参照してください。

RMAN によるリカバリとユーザー管理リカバリ

物理ファイルをリカバリする場合は、2 つの基本的な方法から選択できます。次のことができます。

- Recovery Manager ユーティリティを使用してデータベースをリストアおよびリカバリする方法
- オペレーティング・システムのユーティリティを使用してバックアップをリストアしてから、SQL*Plus の RECOVER コマンドを実行してリカバリする方法

どちらの方法を選択した場合も、データベース、表領域またはデータファイルをリカバリできます。メディア・リカバリを実行する前に、どのデータファイルをリカバリするかを決定する必要があります。通常は、固定ビュー V$RECOVER_FILE を使用できます。このビューには、リカバリが必要な全ファイルと、リカバリを必要とするエラーの説明が表示されます。

> **関連項目：** リカバリ使用例で V$ ビューを使用する方法の詳細は、『Oracle Database バックアップおよびリカバリ・リファレンス』を参照してください。

Recovery Manager でのリストアおよびリカバリ Recovery Manager の基本的なリカバリ・コマンドは、RESTORE および RECOVER です。RESTORE を使用して、バックアップ・セットまたはディスク上のイメージ・コピーから、データファイルを現行または新規の場所にリストアします。アーカイブ REDO ログを含むバックアップ・セットもリストアできますが、通常、この操作は不要です。Recovery Manager では、リカバリに必要なアーカイブ・ログが自動的にリストアされ、リカバリの完了後に検出されるためです。Recovery Manager の RECOVER コマンドを使用してメディア・リカバリを実行し、アーカイブ・ログまたは増分バックアップを適用します。

> **関連項目：** RMAN を使用したリストアおよびリカバリの詳細は、『Oracle Database バックアップおよびリカバリ・リファレンス』を参照してください。

ユーザー管理リストアおよびリカバリ Recovery Manager を使用しない場合は、オペレーティング・システムのユーティリティを使用してバックアップをリストアしてから、SQL*Plus の RECOVER コマンドを実行してデータベースをリカバリできます。

> **関連項目：** オペレーティング・システムのユーティリティと SQL*Plus を使用したリストアおよびリカバリ方法の詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

16

ビジネス・インテリジェンス

この章では、ビジネス・インテリジェンスの基本概念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・ウェアハウスとビジネス・インテリジェンスの概要
- 抽出、変換、ロード（ETL）の概要
- データ・ウェアハウスのマテリアライズド・ビューの概要
- データ・ウェアハウスでのビットマップ索引の概要
- パラレル実行の概要
- 分析 SQL の概要
- OLAP 機能の概要
- データ・マイニングの概要

データ・ウェアハウスとビジネス・インテリジェンスの概要

データ・ウェアハウスは、トランザクション処理ではなく問合せおよび分析用に設計されたリレーショナル・データベースです。通常は、トランザクション・データから導出された履歴データが格納されますが、他のソースからのデータも格納できます。データ・ウェアハウスにより、分析のワークロードがトランザクションのワークロードから分離され、組織は複数のソースからのデータを連結できます。

データ・ウェアハウス環境には、リレーショナル・データベースの他に、ETL ソリューション、オンライン分析処理（OLAP）エンジン、Oracle Warehouse Builder、クライアント分析ツール、データを収集してビジネス・ユーザーに配信するプロセスを管理する他のアプリケーションがあります。

この項の内容は、次のとおりです。

データ・ウェアハウスの特性

すべてのデータ・ウェアハウスでは、次に示す基本的な特性が共通しています。

サブジェクト指向

データ・ウェアハウスは、データの分析に役立つように設計されています。たとえば、会社の売上データの詳細を知るために、売上専用のウェアハウスを構築できます。このウェアハウスを使用すると、「昨年度のこの品目の最大顧客は何か」などの質問に答えることができます。このようにデータ・ウェアハウスは主題（この場合は売上）別に定義できるため、データ・ウェアハウスはサブジェクト指向となっています。

統合

統合は、サブジェクト指向に密接に関連しています。データ・ウェアハウスには、様々なソースからのデータを一貫性のある形式で格納する必要があります。また、名前の競合や単位間の不一致などの問題を解決する必要があります。この目標が達成されて初めて、データ・ウェアハウスは統合されたことになります。

恒常的

恒常的とは、一度ウェアハウスに格納されたデータは変更されないことを意味します。ウエアハウスの目的が、発生した事柄を分析できることであるため、これは論理的です。

時系列

アナリストは、ビジネスの傾向を把握するために大量のデータを必要とします。これは、パフォーマンス要件があるために履歴データをアーカイブに移動する必要のあるオンライン・トランザクション処理（OLTP）システムとはきわめて対照的です。データ・ウェアハウスは時間経過による変化に焦点を当てており、これこそは時系列が意味するものです。

一般に、1 つ以上のオンライン・トランザクション処理（OLTP）データベースからデータ・ウェアハウスに、月次、週次または日次でデータが送信されます。データは、通常**ステージング・ファイル**内で処理されてから、データ・ウェアハウスに追加されます。データ・ウェアハウスのサイズは、数十 GB から数 TB になるのが普通です。また、データの大多数は少数の非常に大きなファクト表に格納されます。

データ・ウェアハウスと OLTP システムの相違点

データ・ウェアハウスと OLTP システムでは、要件が大きく異なります。ここでは、典型的なデータ・ウェアハウスと OLTP システムの相違点の例を示します。

ワークロード

データ・ウェアハウスは非定型問合せに適応するように設計されています。データ・ウェアハウスのワークロードは事前に不明な場合があります。このため、データ・ウェアハウスは様々な問合せ操作を適切に実行できるように最適化する必要があります。

OLTP システムでサポートされるのは、事前定義済の操作のみです。これらの操作のみをサポートするように、アプリケーションの特別なチューニングや設計が必要になる場合があります。

データ修正

データ・ウェアハウスは、バルク・データ修正テクニックを使用して（夜間または週次で実行される）ETL プロセスにより定期的に更新されます。データ・ウェアハウスをエンド・ユーザーが直接更新することはありません。

OLTP システムでは、エンド・ユーザーはデータベースに対して個々のデータ修正文を定期的に発行します。OLTP データベースは常に最新であり、各ビジネス・トランザクションの現在の状態が反映されます。

スキーマ設計

通常、データ・ウェアハウスでは、全体または一部が非正規化されたスキーマ（スター・スキーマなど）を使用して、問合せパフォーマンスが最適化されます。

OLTP システムでは、通常は完全に正規化されたスキーマを使用して更新 / 挿入 / 削除のパフォーマンスが最適化され、データ整合性が保証されます。

典型的な操作

典型的なデータ・ウェアハウスの問合せでは、数千または数百万行がスキャンされます。たとえば、「すべての顧客について先月の総売上を検索する」などです。

典型的な OLTP 操作は、少数のレコードにのみアクセスします。たとえば、「この顧客の現行の注文を検索する」などです。

履歴データ

通常、データ・ウェアハウスには、数か月または数年分のデータが格納されます。これは、履歴分析をサポートするためです。

OLTP システムには、通常は数週または数か月分のデータしか格納されません。OLTP システムでは、現行のトランザクションの要件を適切に満たすために必要な履歴データのみが格納されます。

データ・ウェアハウスのアーキテクチャ

データ・ウェアハウスとそのアーキテクチャは、組織の状況に応じて異なります。次の 3 つのアーキテクチャが一般的です。

- データ・ウェアハウスのアーキテクチャ（基本）
- データ・ウェアハウスのアーキテクチャ（ステージング領域あり）
- データ・ウェアハウスのアーキテクチャ（ステージング領域およびデータ・マートあり）

データ・ウェアハウスのアーキテクチャ（基本）

図 16-1 に、データ・ウェアハウスの単純なアーキテクチャを示します。エンド・ユーザーは、データ・ウェアハウスを介して、複数のソース・システムから導出されたデータに直接アクセスします。

図 16-1 データ・ウェアハウスのアーキテクチャ

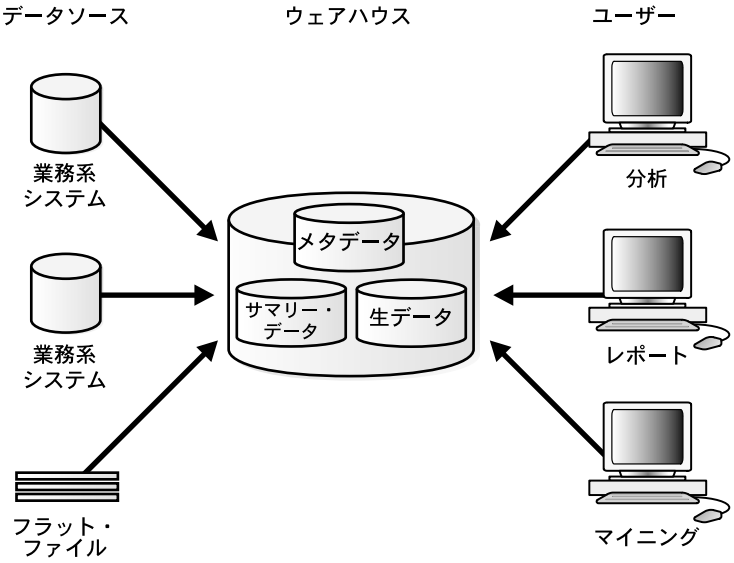


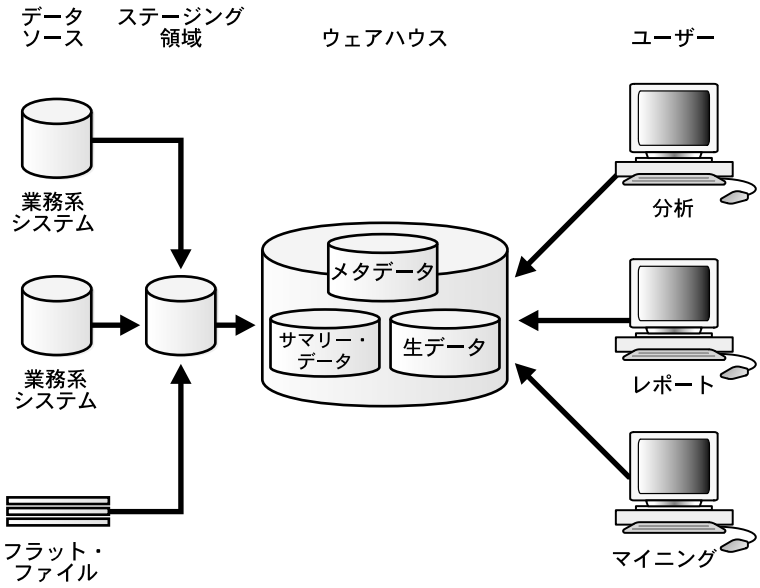
図 16-1 では、従来型 OLTP システムのメタデータと生データの他に、サマリー・データがあります。サマリーでは、時間のかかる処理が事前に行われるため、データ・ウェアハウスではきわめて重要です。たとえば、典型的なデータ・ウェアハウスの問合せでは、8 月の売上などが検索されます。

Oracle Database では、サマリーはマテリアライズド・ビューと呼ばれます。

データ・ウェアハウスのアーキテクチャ（ステージング領域あり）

図 16-1 で示しているように、業務系データをウェアハウスに入れる前に、クリーン・アップおよび処理する必要があります。この操作はプログラムによって実行できますが、ほとんどのデータ・ウェアハウスではかわりにステージング領域が使用されています。ステージング領域により、サマリーの作成とウェアハウス管理全般が簡素化されます。図 16-2 に、この典型的なアーキテクチャを示します。

図 16-2 データ・ウェアハウスのアーキテクチャ（ステージング領域あり）

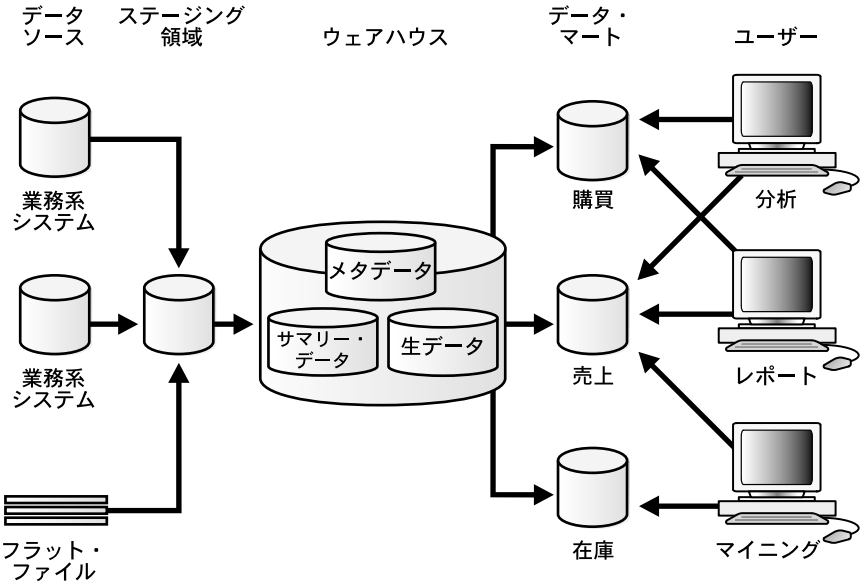


データ・ウェアハウスのアーキテクチャ（ステージング領域およびデータ・マートあり）

図 16-2 のアーキテクチャはごく一般的ですが、ウェアハウスのアーキテクチャを組織内の様々なグループ向けにカスタマイズできます。

そのためには、特定のビジネス・ライン向けに設計されたシステムであるデータ・マートを追加します。図 16-3 に、購買、売上および在庫が分離されている例を示します。この例では、財務アナリストは購買と売上の履歴データを分析できます。

図 16-3 データ・ウェアハウスのアーキテクチャ（ステージング領域およびデータ・マートあり）



関連項目：『Oracle Database データ・ウェアハウス・ガイド』

抽出、変換、ロード（ETL）の概要

データ・ウェアハウスをビジネス分析に役立てるには、データ・ウェアハウスに定期的にデータをロードする必要があります。データをロードするには、1 つ以上の業務系システムからデータを抽出し、ウェアハウスにコピーする必要があります。ソース・システムからデータを抽出してデータ・ウェアハウスに取り込む処理は、一般に **ETL** と呼ばれ、抽出、変換、ロードを表します。ETL では転送フェーズが省略され、プロセスの他のフェーズがそれぞれ個別であることを意味するため、ETL という頭字語ではあまりに単純すぎるでしょう。データのロードも含め、このプロセス全体のことを ETL と呼びます。ETL が詳細に定義された 3 つのステップではなく広範囲なプロセスを指すことを理解しておく必要があります。

ETL の方法論とタスクは以前からよく知られており、必ずしもデータ・ウェアハウス環境に固有のものではありません。様々な独自アプリケーションおよびデータベース・システムが、企業の IT のバックボーンとなっています。データの統合を試み、少なくとも 2 つのアプリケーションに同じ内容を提供して、アプリケーション間やシステム間で共有する必要があります。多くの場合、このデータの共有は、ETL に類似したメカニズムによって対処されています。

データ・ウェアハウス環境は同じ課題に直面しているのに加えて、データを多数のシステム間で交換するのみでなく、統合、再配置および連結する必要があり、それによりビジネス・インテリジェンスに新たに統一された情報ベースを提供するという問題を抱えています。また、データ・ウェアハウス環境ではデータ量が非常に大きくなる傾向があります。

ETL プロセスでどのような処理が発生するかを考えてみます。抽出時には必要なデータが識別され、データベース・システムやアプリケーションなど多数の異なるソースから抽出されます。通常、必要なデータの特定のサブセットを識別することはできません。したがって、必要以上のデータを抽出することになるため、該当データの識別は後に行われます。ソース・システムの機能（オペレーティング・システムのリソースなど）によっては、この抽出プロセスである程度の変換が発生することがあります。抽出されるデータのサイズは、ソース・システムとビジネス状況に応じて数百 KB ～数 GB です。2 つの（論理的に）同一の抽出間に要する時間についても、同じことが言えます。つまり、期間は数日、数時間、数分、ほぼリアルタイムなど様々です。たとえば、Web サーバーのログ・ファイルは、きわめて短時間のうちに簡単に数百 MB に達することがあります。

抽出したデータは、さらに処理するためにターゲット・システムまたは中間システムに物理的に転送する必要があります。選択した転送方法によっては、このプロセスでなんらかの変換処理を実行することもできます。たとえば、ゲートウェイを介してリモート・ターゲットに直接アクセスする SQL 文では、`SELECT` 文の一部として 2 つの列を連結できます。

ロード中にエラーが発生すると、そのエラーがログに記録され、操作を続行できます。

この項の内容は、次のとおりです。

トランスポータブル表領域

トランスポータブル表領域は、2 つの Oracle データベース間で大量のデータを移動する場合に最も高速な方法です。異なるコンピュータ・アーキテクチャやオペレーティング・システム間で表領域を転送することもできます。

従来、最もスケーラブルなデータ転送メカニズムでは、生データを含むフラット・ファイルを移動していました。この種のメカニズムでは、データをソース・データベースからアンロードするかファイルにエクスポートする必要がありました。それから、転送後にターゲット・データベースにロードまたはインポートする必要がありました。トランスポータブル表領域は、このアンロードしてから再ロードするステップを完全にバイパスします。

トランスポータブル表領域を使用すると、Oracle Database データファイル（表データ、索引および他のほぼすべての Oracle データベース・オブジェクトを含む）を、データベース間で直接転送できます。さらに、トランスポータブル表領域は、データ転送に加えてメタデータの転送用に、インポートおよびエクスポートに似たメカニズムを提供します。

データ・ウェアハウスのトランスポータブル表領域が最も一般的に利用されるのは、ステージング・データベースからデータ・ウェアハウスへとデータを移動する場合、またはデータ・ウェアハウスからデータ・マートにデータを移動する場合です。

テーブル・ファンクション

テーブル・ファンクションは、PL/SQL、C または Java で実装された変換のパイプライン実行とパラレル実行をサポートします。前述の使用例では、中間のステージング表を使用する必要がありません。中間のステージング表を使用すると、各種の変換ステップを経るデータ・フローが中断されます。

テーブル・ファンクションは、出力として行セットを生成できる関数として定義されます。また、入力として行セットを使用することもできます。テーブル・ファンクションによりデータベース機能が拡張され、次の操作が可能になります。

- 1 つの関数から複数の行を戻すことができます。
- SQL 副問合せ（複数行を選択）の結果が関数に直接渡されます。
- 関数の入力としてカーソルを使用します。
- 関数をパラレル化できます。
- 結果セットは作成後すぐに段階的に戻され、処理を進めることができます。これを段階的パイプライン処理と呼びます。

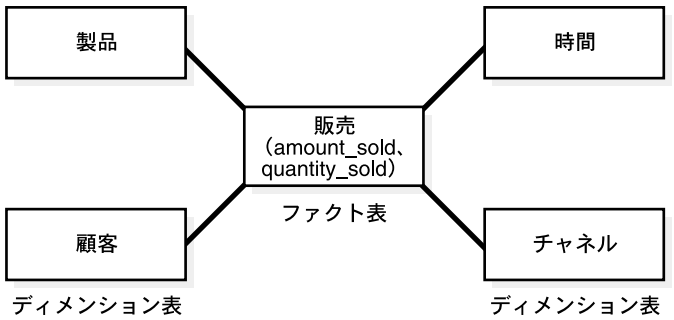
テーブル・ファンクションは、システム固有の PL/SQL インタフェースを使用して PL/SQL で定義するか、Oracle Data Cartridge Interface（ODCI）を使用して Java または C で定義できます。

外部表

外部表により、外部データを仮想表として使用できます。仮想表は、最初に外部データをデータベースにロードしなくても、パラレルで直接問い合せたり結合できます。これにより、SQL、PL/SQL および Java を使用して外部データにアクセスできます。

外部表を使用すると、ロード・フェーズと変換フェーズをパイプライン処理できます。データ・ストリームを中断することなく、変換プロセスをロード・プロセスとマージできます。データベース内のデータに対してさらに比較や変換などの処理を実行するために、データベース内でステージングする必要がなくなります。たとえば、従来型ロードの変換機能を、外部表からの `SELECT` とともにダイレクト・パス `INSERT AS SELECT` 文に使用できます。図 16-4 に、パイプライン処理の一般的な例を示します。

図 16-4 パイプライン処理されたデータ変換



外部表と通常の表の重要な違いは、外部で構成された表が読取り専用であることです。外部表では、DML 操作（`UPDATE/INSERT/DELETE`）および索引の作成ができません。

外部表は SQL*Loader を補完するもので、外部ソース全体を既存のデータベース・オブジェクトと結合して複雑な方法で変換する環境や、外部データ・ボリュームが大きく 1 度しか使用されない環境で特に役立ちます。これに対して SQL*Loader は、ステージング表で追加の索引付けが必要となるデータをロードする場合に、より適切な選択肢となります。これは、データが独立した複雑な変換に使用される操作や、以降の処理でデータの一部のみが使用される操作の場合も同様です。

表の圧縮

ヒープ構成表を圧縮することでディスク領域を節約できます。表の圧縮を考慮する必要のある典型的なタイプのヒープ構成表は、パーティション表です。

ディスク使用量とメモリー使用量（特にバッファ・キャッシュ）を減少させるために、表とパーティション表をデータベースに圧縮形式で格納できます。通常は、これにより読取り専用操作のパフォーマンスが向上します。また、問合せの実行も高速化されます。ただし、CPU オーバーヘッドの面で少しコストがかかります。

表の圧縮は、多数の外部キーを持つ表など、冗長度の高いデータに使用する必要があります。更新や他の DML アクティビティが多い表は圧縮しないでください。圧縮された表またはパーティションは更新可能ですが、この種の表の更新時にオーバーヘッドが発生し、更新アクティビティが高いときには領域が多少浪費されて圧縮の妨げとなる場合があります。

関連項目： 5-7 ページの「表の圧縮」

チェンジ・データ・キャプチャ

チェンジ・データ・キャプチャにより、Oracle Database リレーショナル表に対して追加、更新または削除されたデータが効率的に識別および取得され、変更データのアプリケーションでの使用が可能になります。

通常、データ・ウェアハウスでは、分析のためにリレーショナル・データを 1 つ以上のデータベースからデータ・ウェアハウスに抽出して転送する必要があります。チェンジ・データ・キャプチャにより、表全体ではなく変更があったデータのみがすばやく識別され、処理されて、変更データが後で使用できるようになります。

チェンジ・データ・キャプチャは、リレーショナル・データベースの外部でデータを段階的に処理するための中間フラット・ファイルに依存しません。ユーザー表に対する `INSERT`、`UPDATE` および `DELETE` 操作からの変更データを取得します。変更データはチェンジ・テーブルと呼ばれるデータベース・オブジェクトに格納され、制御された方法でアプリケーションで使用可能になります。

関連項目：『Oracle Database データ・ウェアハウス・ガイド』

データ・ウェアハウスのマテリアライズド・ビューの概要

データ・ウェアハウスでパフォーマンス改善のために採用されているテクニックの 1 つは、サマリーを作成することです。サマリーは特殊な集計ビューで、問合せを実行する前に高コストの結合および集計操作を計算して結果をデータベースの表に格納することで、問合せの実行時間を短縮します。たとえば、地域別、製品別の売上合計を含む表を作成できます。

このマニュアルおよびデータ・ウェアハウス関連書で言及しているサマリーまたは集計は、Oracle Database では**マテリアライズド・ビュー**というスキーマ・オブジェクトを使用して作成されます。マテリアライズド・ビューは、問合せパフォーマンスの改善や複製データの提供など、様々な役割を果します。

従来、サマリーを使用している組織は、手動によるサマリー作成、作成するサマリーの識別、サマリーの索引付けと更新、さらに使用サマリーに関するユーザーへのアドバイスなどの作業に、大量の時間と労力を費やしています。サマリー管理によりデータベース管理者のワークロードが軽減され、ユーザーは定義済のサマリーを知る必要がなくなります。データベース管理者は 1 つ以上のマテリアライズド・ビューを作成しますが、これがサマリーに相当します。エンド・ユーザーは、詳細データ・レベルで表やビューを問い合せます。

SQL 問合せは、Oracle Database のクエリー・リライト・メカニズムにより、サマリー表を使用するように自動的にリライトされます。このメカニズムにより、問合せから結果を戻すための応答時間が短縮されます。データ・ウェアハウス内のマテリアライズド・ビューは、エンド・ユーザーやデータベース・アプリケーションに対して透過的です。

通常、マテリアライズド・ビューにはクエリー・リライト・メカニズムを介してアクセスしますが、エンド・ユーザーやデータベース・アプリケーションはサマリーに直接アクセスする問合せを構成できます。ただし、サマリーに対する変更はそれを参照する問合せに影響するため、ユーザーにこの操作を許可するかどうかは重大な考慮事項となります。

Oracle Database では、一連のマテリアライズド・ビューの分析およびアドバイザ用のファンクションおよびプロシージャが DBMS_ADVISOR パッケージとして提供されており、スキーマの多数のマテリアライズド・ビューでの選択に役立ちます。これらのファンクションは総称して SQL アクセス・アドバイザと呼ばれ、どの PL/SQL プログラムからでもコールできます。SQL アクセス・アドバイザは、仮説によるワークロードまたはユーザー定義ワークロード、あるいは SQL キャッシュから取得したワークロードに基づいて、マテリアライズド・ビューを推奨します。SQL アクセス・アドバイザは、Oracle Enterprise Manager から実行するか、DBMS_ADVISOR パッケージを起動して実行します。

関連項目： マテリアライズド・ビューと SQL アクセス・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

データ・ウェアハウスでのビットマップ索引の概要

ビットマップ索引は、データ・ウェアハウス環境で広範囲に使用されています。通常、この環境では大量のデータと非定型問合せを扱いますが、同時 DML トランザクションのレベルは高くありません。この種のアプリケーションに対するビットマップ索引の利点は次のとおりです。

- 多くの種類の非定型問合せの応答時間が短縮されます。
- 他の方式の索引と比べて記憶域必要量が減少します。
- 比較的少数の CPU や小型のメモリーが搭載されたハードウェアであっても、パフォーマンスが劇的に向上します。
- パラレル DML およびロード中に効果的にメンテナンスできます。

従来の B ツリー索引を使用して大きな表に完全な索引を作成すると、領域という面で膨大なコストがかかります。索引は表中のデータの何倍にも膨れ上がることがあるからです。それに対して、ビットマップ索引は通常、表中で索引を付けるデータのサイズより小さくてすみます。

索引では、特定のキー値が含まれる行にポインタが提供されます。通常の索引では、キー値と、そのキー値を持つ行の ROWID の対応リストを格納します。ビットマップ索引では、ROWID のリストが各キー値のビットマップで置換されます。

ビットマップの各ビットは存在する ROWID に対応します。ビットが設定されている場合は、対応する ROWID を持つ行にはキー値が含まれることになります。マッピング機能がビット位置を実際の ROWID に変換するため、ビットマップ索引は通常の索引と同じ機能を果します。異なるキー値の数が多くない場合、ビットマップ索引は領域を節約できます。

ビットマップ索引は、WHERE 句に複数の条件を含む問合せに最も効率的です。一部の条件は満たしているがすべては満たしていない行については、表自体にアクセスする前に除外します。これによって、応答時間が短縮されます。多くの場合は劇的に改善されます。可能な値が小さいため、性別列はビットマップ索引の優れた候補となります。

パラレル問合せおよびパラレル DML は、従来の索引のみでなく、ビットマップ索引に対しても機能します。また、索引のパラレル作成と連結索引もサポートしています。

関連項目：『Oracle Database データ・ウェアハウス・ガイド』

パラレル実行の概要

Oracle Database で SQL 文をパラレル実行すると、同時に多数のプロセスが連携して 1 つの SQL 文を実行します。文の実行に必要なタスクを多数のプロセスで分割して実行すると、Oracle Database では、1 つのプロセスで実行する場合と比較して、より高速に実行できます。これは、**パラレル実行**または**パラレル処理**と呼ばれます。

パラレル実行では、意思決定支援システム（DSS）およびデータ・ウェアハウスに対応付けられた大規模なデータベースにおいて、集中データ処理の応答時間が大幅に削減されます。対称型マルチプロセッサ（SMP）、クラスタ化されたシステムおよび大規模クラスタ・システムでは、パラレル実行によりパフォーマンスが大幅に改善されます。それは 1 つの Oracle Database システムで、文の処理を多数の CPU で分割できるためです。特定のタイプのオンライン・トランザクション処理（OLTP）およびハイブリッド・システムに、パラレル実行を実装することもできます。

パラレル化とは、すべての問合せ作業を 1 つのプロセスで行わずに、多数のプロセスで同時に行えるように 1 つの作業を分割する考え方です。たとえば、1 つのプロセスで 12 か月分をすべて処理するかわりに、12 のプロセスが 1 年のそれぞれの月を分担して処理するなどがこの例です。これにより、パフォーマンスの大幅な改善が期待できます。

パラレル実行では、ハードウェア・リソースの使用が最適化され、システムのパフォーマンスの向上に役立ちます。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。

Oracle RAC 環境では、パラレル実行は、特定のサービスに対するサービスの配置によって制御されています。つまり、パラレル・プロセスはサービスを構成したノードで実行されます。Oracle Database のデフォルトの動作では、パラレル処理は、データベースに接続する際に使用したサービスが提供されるインスタンスでのみ実行されます。これは、パラレル・リカバリや `GV$queries` の処理などの他のパラレル操作には影響しません。

パラレル実行に適さない作業もあります。たとえば、OLTP 操作の多くは比較的高速であり数秒以内で完了されるので、全体の実行時間に対し、パラレル実行利用のオーバーヘッドが大きくなる傾向にあります。

> **関連項目：** パラレル実行を最大限に活用するためのパラメータ・ファイルおよびデータベースのチューニングの詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。また、Oracle RAC 環境でのパラレル実行に関する考慮事項は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

パラレル実行の動作

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のすべてを 1 つのサーバー・プロセスが実行します。たとえば、全表スキャン（`SELECT * FROM emp` など）を実行するには、図 16-5 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 16-5 シリアルの全表スキャン

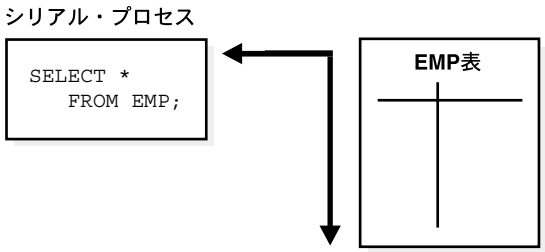
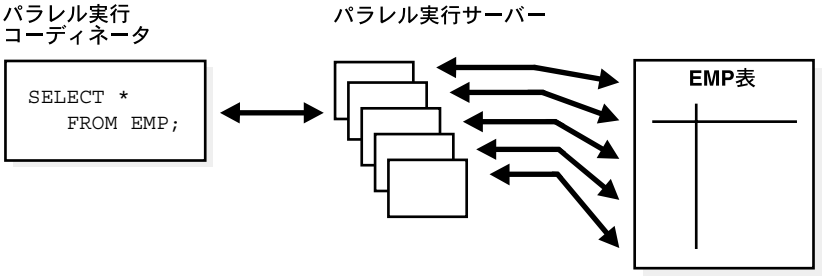


図 16-6 では、複数のパラレル実行サーバーが emp 表のスキャンを実行しています。この表はグラニュルと呼ばれるロード単位に動的に分割され（**動的パーティション化**）、各グラニュルは単一のパラレル実行サーバーにより読み取られます。グラニュルはコーディネータにより生成されます。各グラニュルは、emp 表の物理ブロックの範囲です。実行サーバーへのグラニュルのマッピングは静的ではなく、実行時に決定されます。実行サーバーがグラニュルに対応する emp 表の行の読取りを終了したとき、グラニュルがまだ残っている場合、コーディネータから別のグラニュルを取得します。この操作は、すべてのグラニュルが使い果され、emp 表全体が読み取られるまで継続されます。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、最終的な全表スキャンになります。

図 16-6 パラレルの全表スキャン



SQL 問合せの問合せ計画では、まずパラレル実行コーディネータが SQL 問合せ内の各演算子をパラレル・ピースに分割し、それらを問合せで指定した順序に従って実行します。次に、演算子を実行するパラレル実行サーバーによって作成された部分的な実行結果を統合します。1 つの操作に割り当てられたパラレル実行サーバーの数が、その操作の並列度（DOP）となります。同じ SQL 文中の複数の操作の並列度は、すべて同じです。

> **関連項目：** グラニュルの詳細およびマルチユーザー環境での Oracle Database によるタスクの分割および DOP の処理の詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

分析 SQL の概要

Oracle には、データベースで分析操作を実行するための多数の SQL 操作が導入されています。たとえば、ランキング、移動平均、累計、対比レポート、期間対比などです。これらの計算の一部は従来でも SQL を使用して実行できましたが、この構文はパフォーマンスを大幅に改善します。

この項の内容は、次のとおりです。

集計用 SQL

集計は、データ・ウェアハウスの基盤となる部分です。ウェアハウスでの集計効率を改善するために、Oracle Database には問合せとレポートを容易かつ高速にする GROUP BY 句の拡張機能が用意されています。これらの拡張機能を使用すると、次の操作ができます。

- 最も詳細なレベルから総計へと、集計レベルの昇順で集計します。
- 1 つの文を使用して、集計に可能なすべての組合せを計算します。
- 1 つの問合せでクロス集計レポートに必要な情報を生成します。

これらの拡張機能を使用すると、必要なデータ・グループを GROUP BY 句で正確に指定できます。これにより、CUBE 操作を実行しなくても複数のディメンション間で効率的な分析が可能になります。キューブ全体を計算すると大きな処理負荷が発生するため、キューブをグルーピング・セットで置き換えるとパフォーマンスを大幅に改善できます。CUBE、ROLLUP およびグルーピング・セットは、異なる方法でグループ化された行の UNION ALL に等価の結果セットを 1 つ生成します。

パフォーマンスを強化するために、これらの拡張機能をパラレル化できます。つまり、複数のプロセスがこれらの文をすべて同時に実行できます。これらの機能により集計計算の効率が向上し、データベースのパフォーマンスとスケーラビリティが強化されます。

意思決定支援システムで重要な概念の 1 つが多次元分析、つまり必要なディメンションのすべての組合せから企業を検査することです。**ディメンション**という用語は、質問の指定に使用される任意のカテゴリという意味で使用されます。最も一般的に指定されるディメンションは時間、地理、製品、部門および物流チャネルですが、潜在的なディメンションは企業のアクティビティと同様に制限はありません。通常、特定のディメンション値の集合に関連付けられたイベントやエンティティは、**ファクト**と呼ばれます。ファクトは、営業単位や各国通貨、収益、顧客数、生産高など、追跡する価値のあるデータです。

ここでは、多次元要求の例を使用して説明します。

- 1999 年と 2000 年について、州、国、地域というように地理ディメンションの集計レベルの昇順で全製品の総売上を表示します。
- 1999 年と 2000 年における南アメリカの地域別経費を示す事業のクロス集計分析を作成します。可能な小計をすべて組み込みます。
- 自動車製品の 2000 年の売上収益に従って、アジアにおける上位 10 人の営業担当について、コミッションのランキング・リストを作成します。

これらの要求には、いずれも複数のディメンションが含まれています。多次元による多くの質問では、通常は時間、地理または予算にまたがって集計されたデータと、データ・セットの比較が必要になります。

関連項目：『Oracle Database データ・ウェアハウス・ガイド』

分析用 SQL

Oracle は、分析 SQL 関数ファミリを使用した拡張 SQL 分析処理機能を備えています。これらの分析関数を使用して次の計算を実行できます。

- ランキングとパーセンタイル
- 変動ウィンドウの計算
- LAG/LEAD 分析
- FIRST/LAST 分析
- 線形回帰統計

ランキング関数には、累積分布、パーセント・ランクおよび N タイルがあります。変動ウィンドウの計算を使用すると、合計や平均などの移動集計と累積集計を求めることができます。LAG/LEAD 分析を使用すると、行間を直接参照できるため、期間ごとの変化を計算できます。FIRST/LAST 分析を使用すると、順序付けしたグループ内の最初と最後の値を求めることができます。

その他の機能として CASE 式があります。CASE 式は、様々な場合に役立つ if-then ロジックを提供します。

パフォーマンスを強化するために、分析関数をパラレル化できます。つまり、複数のプロセスがこれらの文をすべて同時に実行できます。これらの機能により計算が容易かつ効率的になり、データベースのパフォーマンスとスケーラビリティが強化され、簡素化が促進されます。

関連項目：『Oracle Database データ・ウェアハウス・ガイド』

モデル化用 SQL

Oracle の MODEL 句は、SQL 計算に新たなレベルの機能と柔軟性をもたらします。MODEL 句を使用すると、問合せ結果から多次元配列を作成し、この配列に計算式を適用して新規の値を計算できます。計算式には、基本的な演算や再帰型を使用した連立方程式を使用できます。一部のアプリケーションでは、MODEL 句で PC ベースのスプレッドシートを置き換えることができます。SQL でのモデルには、スケーラビリティ、管理性、コラボレーションおよびセキュリティにおける Oracle Database の長所が生かされています。中核となる問合せエンジンは、処理できるデータ数に制限がありません。データベース内でモデルを定義して実行すると、大きいデータセットを個別のモデル化環境の間で転送する必要がありません。モデルはワークグループ間で簡単に共有できるため、すべてのアプリケーションで計算の一貫性が確実に保たれます。モデルを共有できるのみでなく、アクセスも Oracle Database のセキュリティ機能で厳密に制御できます。MODEL 句は、その豊富な機能によりあらゆるタイプのアプリケーションを強化できます。

関連項目：『Oracle Database データ・ウェアハウス・ガイド』

OLAP 機能の概要

Oracle オンライン分析処理（OLAP）により、分析コンテンツが幅広くなり、問合せ応答時間が短縮されるため、SQL アプリケーションの能力が強化されます。SQL 問合せインタフェースにより、OLAP を認識せずに任意のアプリケーションでキューブおよびディメンションを問い合せることができるようになります。

OLAP オプションを使用すると、キューブ、ディメンションおよび階層に関する一連のリレーショナル・ビューが自動的に生成されます。SQL アプリケーションによってこれらのビューが問い合せられ、アナリストや意思決定者の役に立つ情報が豊富なこれらのオブジェクトのコンテンツが表示されます。また、実表などのシステム生成ビューを使用して、アプリケーションに求められる構造に適合するカスタム・ビューを作成することもできます。

アナリストは、任意の SQL 問合せおよび分析ツールを使用して、データを選択、表示および分析できます。お気に入りのツールやアプリケーション、または Oracle Database で提供されるいずれかのツール（Oracle Application Express や Business Intelligence Publisher など）を使用できます。

関連項目：『Oracle OLAP ユーザーズ・ガイド』

この項の内容は、次のとおりです。

多次元テクノロジの完全な統合

Oracle では、多次元のオブジェクトおよび分析をデータベースに統合することにより、多次元分析の利点と、Oracle Database の信頼性、可用性、セキュリティおよびスケーラビリティの利点を兼ね備えています。

Oracle OLAP は、Oracle Database に完全に統合されています。技術レベルでは、これは次のことを意味します。

- OLAP エンジンは、Oracle Database のカーネル内で動作します。
- ディメンション・オブジェクトは、システム固有の多次元形式で Oracle Database に格納されます。
- キューブおよび他のディメンション・オブジェクトは、Oracle データ・ディクショナリ内に存在する最上位のデータ・オブジェクトです。
- データ・セキュリティは、Oracle Database ユーザーおよびロールの権限の付与と取消しにより、標準的な方法で管理されます。
- アプリケーションは、SQL を使用してディメンション・オブジェクトを問い合せることができます。

組織は多大な利点を享受できます。Oracle OLAP には、簡素化によってもたらされる強みがあります。つまり、1 つのデータベース、標準的な管理とセキュリティ、標準的なインタフェースと開発ツールという強みです。

アプリケーション開発の容易さ

Oracle OLAP を使用すると、魅力的な分析コンテンツによってデータベースおよびアプリケーションの機能を簡単に高めることができます。Oracle の多次元のオブジェクトおよび計算にシステム固有の方法で SQL を使用してアクセスできるため、独自のインタフェースを使用するシステムと比較しても、任意のタイプのダッシュボード、レポート、ビジネス・インテリジェンスおよび分析アプリケーションは大幅に開発しやすくなります。さらに、SQL によるアクセスが可能であるということは、従来の限定された OLAP アプリケーション・コレクションのみでなく、任意のデータベース・アプリケーションが Oracle OLAP の分析機能を使用できることを意味します。

管理の容易さ

Oracle OLAP は Oracle Database に完全に埋め込まれているため、一般的なスタンドアロンの OLAP サーバーのように管理に習熟するまでに時間がかかることはありません。特別な管理スキルに投資せずに、既存の DBA スタッフを活用できます。

Oracle の埋込み型の OLAP テクノロジによる管理上の主な利点の 1 つに、キューブの自動メンテナンスがあります。スタンドアロンの OLAP サーバーの場合、キューブをリフレッシュするという作業負担はすべて管理者に負わされています。この作業は複雑で、間違いを起こしやすいジョブです。管理者は、変更されたデータをリレーショナル・ソースから抽出し、このデータをソース・システムからスタンドアロンの OLAP サーバーが稼働しているシステムに移動し、キューブをロードして再構築するプロシージャを作成する必要があります。また、管理者は、このプロセス中に変更された値のセキュリティに関する責任も負うことになります。

一方、Oracle OLAP を使用すると、キューブのリフレッシュは Oracle Database によって完全に処理されます。ディメンション・オブジェクトの失効が追跡され、ソース表のデルタが自動的に記録され、リフレッシュ・プロセス中に変更された値のみが自動的に適用されます。管理者は適切な間隔でリフレッシュをスケジュールするのみで、他のすべての作業は Oracle Database によって処理されます。

セキュリティ

Oracle OLAP では、Oracle Database の標準的なセキュリティ機能を使用して多次元データが保護されます。

一方、スタンドアロンの OLAP サーバーの場合、管理者はリレーショナル・ソース・システムと OLAP サーバー・システムで、セキュリティ管理が 2 回必要です。また、リレーショナル・システムからスタンドアロンの OLAP システムへデータを移動する際のセキュリティも管理する必要があります。

他に例のないパフォーマンスとスケーラビリティ

ビジネス・インテリジェンスおよび分析アプリケーションでのアクションのほとんどは、階層のドリルアップおよびドリルダウン、期間対比、親の構成比、将来期間の予測などの集計値の比較、およびこれに類似した多数の計算によって占められています。多くの場合、これらのアクションは本質的に、潜在する階層型の集計領域全体にわたってランダムに実行されます。Oracle OLAP では、定義された多次元領域ですべての集計が事前に計算または効率的に即時に計算されるため、従来のビジネス・インテリジェンス・アプリケーションとは比較にならないパフォーマンスを実現します。

Oracle OLAP の問合せでは Oracle 共有のカーソルが利用されるため、メモリー要件が大幅に削減され、パフォーマンスが向上します。

Oracle Real Application Clusters（Oracle RAC）を使用して Oracle Database をインストールすると、OLAP アプリケーションは、他の任意のアプリケーションと同様のパフォーマンス、スケーラビリティ、フェイルオーバーおよびロード・バランシング上の利点を享受できます。

コスト削減

これらのすべての機能はコスト削減につながります。既存のスタッフのスキルを活用できるため、管理コストを削減できます。さらに、ディメンション・オブジェクトのリフレッシュという、他のシステムでは管理者に任されている複雑なタスクは、Oracle Database によって管理されます。また、標準のセキュリティ機能も管理コストの削減に貢献します。アプリケーションの開発コストも削減できます。これは、SQL に関する知識を備えた大勢のアプリケーション開発者や SQL ベースの多数の開発ツールを利用できることが、アプリケーションのより迅速な開発とデプロイにつながるためです。Oracle OLAP は、SQL ベースの任意の開発ツールが利用できます。Oracle OLAP の効率的な集計管理、共有カーソルの使用、および低コストの商用コンポーネントから高度な拡張性を持つシステムを構築可能な Oracle RAC により、ハードウェアのコストも削減できます。

データ・マイニングの概要

Oracle Data Mining は、Oracle Database の埋込みデータ・マイニング・コンポーネントです。データがデータベースを出ることはなく、データ準備、モデル構築およびモデルのスコアリングはすべてデータベース内で実行されます。データがデータベースを出ることはないため、スケーラビリティ、管理性およびユーザー・アクセスが大幅に向上します。これにより、Oracle Database はアプリケーション開発者に、データ・マイニングをデータベース・アプリケーションとシームレスに統合するためのインフラストラクチャを提供します。データ・マイニングは、コール・センター、ATM、ERM およびビジネス・プランニングなどのアプリケーションでよく使用されます。

Oracle Database 11*g* では、Oracle Data Mining モデルは SYS スキーマのデータ・ディクショナリ・オブジェクトとして実装されます。一連の新しいデータ・ディクショナリ・ビューは、マイニング・モデルとそのプロパティを示します。新しいシステムおよびオブジェクト権限により、マイニング・モデル・オブジェクトへのアクセスが制御されます。

Oracle Data Mining 11*g* では、新たに一般化線形モデル（GLM）をサポートしています。Oracle Data Mining は、分類用と回帰用の 2 つの形式の GLM をサポートしています。

- バイナリ・ロジスティック回帰 : 分類用として使用され、スコアリング・データの各行の確率を予測します。従属変数（ターゲット）は、バイナリ形式で断定的です。たとえば、人口統計属性を使用して、販売促進に対する顧客の反応は高いか低いかを予測できます。
- 多変量線形回帰 : 回帰用として使用され、スコアリング・データの各行の連続体内における最良の推定値を予測します。たとえば、年齢層、所得水準、性別および居住地などの人口統計属性を使用して、顧客ごとの売上を予測できます。

Oracle Data Mining の GLM では、一般的に 30 以下の入力属性をサポートしている従来の実装とは異なり、何百何千もの入力属性を処理できます。

モデルの構築、テストおよびスコアリングなどのデータ・マイニング・アクティビティは、PL/SQL API、Java API および SQL データ・マイニング関数を使用して実行されます。Java API は、データ・マイニング規格 JSR 73 に準拠しています。Java API および PL/SQL API は、完全に相互運用可能です。

Oracle Data Mining では必要に応じて、ビニング、正規化および異常値処理などのアルゴリズムに必要なすべてのデータ準備を自動的に実行できます。また、ユーザー指定のデータ変換をアルゴリズム固有のデータ準備に統合することにより、テストおよびスコアリングを簡略化できます。このようなモデルはスーパーモデルです。

SQL データ・マイニング関数は、データ・マイニング・モデルをデプロイするための SQL 言語を操作します。データ・マイニング関数は、分類、回帰、クラスタ化および機能抽出モデルのスコアリングをサポートします。標準的な SQL 文のコンテキストでは、事前に作成されたモデルを新規データや将来の処理用として戻された結果に適用できます。

予測分析は、単純なルーチンでデータ・マイニング・プロセスを取得するテクノロジです。ワンクリック・データ・マイニングとも呼ばれる予測分析により、データ・マイニング・プロセスを簡略化して自動化します。このプロシージャにより、分析処理の結果が戻されます。モデルや他の中間オブジェクトは保存されません。DBMS_PREDICTIVE_ANALYTICS PL/SQL パッケージにより、予測分析に次のプロシージャが実装されます。

- EXPLAIN: ターゲット属性との関係が深い順に属性をランク付けします。
- PREDICT: ターゲット属性の値を予測します。
- PROFILE: 同じターゲット値を持つレコードを識別するルールを作成します。

Oracle Data Mining は、次のアルゴリズムをサポートしています（一般化線形モデルは、Oracle Database 11*g* の新機能です）。

- 分類では、Naive Bayes、ディシジョン・ツリー、一般化線形モデル（バイナリ・ロジスティック回帰）およびサポート・ベクター・マシン
- 回帰では、サポート・ベクター・マシンおよび一般化線形モデル（多変量線形回帰）
- アソシエーション（マーケット・バスケット分析）用の Apriori
- クラスタリングでは、*k*-Means および O-Cluster
- 属性重要度用の Minimum Description Length
- 変則検出用の One Class Support Vector Machine
- 機能抽出のための Non-Negative Matrix Factorization

関連項目：

- 『Oracle Data Mining 概要』
- 『Oracle Data Mining 管理者ガイド』
- 『Oracle Data Mining アプリケーション開発者ガイド』
- 『Oracle Data Mining Java API リファレンス』には、Oracle Data Mining の Java API を構成するクラスに関する Javadoc の記述が含まれます。
- PL/SQL API の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_DATA_MINING、DBMS_DATA_MINING_TRANSFORM および DBMS_PREDICTIVE_ANALYTICS に関する各章を参照してください。
- SQL Data Mining 関数の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

17

高可用性

この章では、データベースの可用性の概念を説明し、Oracle Database の高可用性製品と機能を紹介します。

> **注意：** 可用性は、データベース・ソフトウェア以外のユーザーの選択内容に影響されます。ハードウェア、アプリケーションとオペレーティング・システム・ソフトウェア、ストレージ・メディア、ネットワークの信頼性および運用プロセスは、すべて重要な要素です。

この章の内容は、次のとおりです。

関連項目：

- 『Oracle Database 高可用性概要』
- 『Oracle Database 高可用性ベスト・プラクティス』

これらのマニュアルでは、高可用性環境のデプロイのベスト・プラクティスに関する完全な情報が提供されており、高可用性をサポートする Oracle 製品および機能が説明されています。

高可用性の概要

企業は、競争力を高め、生産性を上げ、ユーザーがより豊富な情報に基づいてより迅速に意思決定することを支援するために、情報テクノロジ（IT）・インフラストラクチャを活用しています。ただし、このような利点とは別に、このインフラストラクチャに対する依存度も高まっています。収益や顧客を失う、罰則が課される、新聞での悪評が顧客と会社の評判に長く悪影響を及ぼすというリスクも存在します。今日の日進月歩の経済環境に置かれたすべての企業が成功して健全であり続ける上で、高可用性 IT インフラストラクチャの構築は非常に重要です。

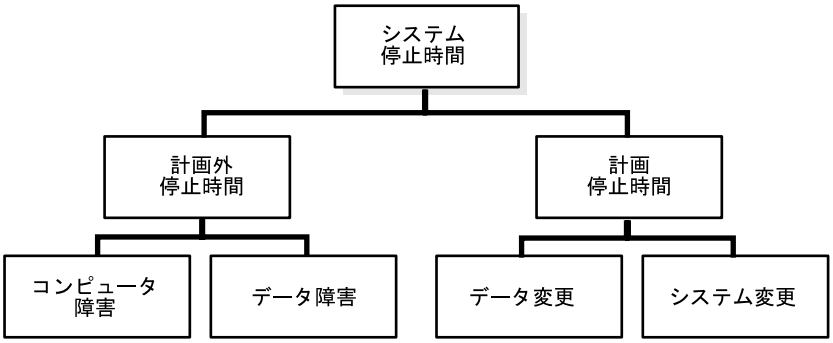
コンピューティング・テクノロジには、グリッド・コンピューティングと呼ばれる新しい IT アーキテクチャのデプロイの実現化に向けた流れもあります。グリッド・コンピューティング・アーキテクチャを使用すると、すべての企業のコンピューティング・ニーズに応じて多くのサーバーおよび記憶域を柔軟なオンデマンド・コンピューティング・リソースにプールできます。低コストのブレード・サーバー、小型で廉価なマルチプロセッサ・サーバー、モジュール記憶域テクノロジ、およびオープン・ソースのオペレーティング・システム（Linux など）のようなテクノロジ革新により、グリッド用の素材が提供されます。これらのテクノロジを利用し、Oracle Database に用意されているグリッド・テクノロジを活用することにより、企業は、非常に高品質なサービスをユーザーに提供するとともに、IT 費用を大幅に削減できます。Oracle Database を使用すると、パフォーマンス、スケーラビリティ、セキュリティ、管理性、機能またはシステムの可用性を犠牲にすることなく、エンタープライズ・グリッド・コンピューティングによるコスト面での利点を享受できます。

この章では、停止時間の原因を調査するとともに、損害の大きい停止時間を回避し、障害から迅速にリカバリするために Oracle Database に用意されているテクノロジを確認します。

停止時間の原因

高可用性の IT グリッド・インフラストラクチャ設計の課題の 1 つは、可能性のある停止時間の原因をすべて調査して対処することです。図 17-1 では、システムの停止時間を計画外停止時間と計画停止時間という 2 つの主なカテゴリに分けた図を示しています。フォルト・トレラントでリジリエンスのある IT インフラストラクチャを設計する際には、計画外停止時間と計画停止時間の両方の原因を考慮することが重要です。

図 17-1 停止時間の原因



計画外停止時間の原因は、コンピュータ障害またはデータ障害です。計画停止時間は主に、本番システムに適用する必要があるデータ変更またはシステム変更が原因です。次の項では、これらの停止時間の各原因を調査し、停止時間を回避するために適用できる Oracle テクノロジについて説明します。

コンピュータ障害に対する保護

コンピュータ障害が発生するのは、コンピュータ・システムまたはデータベース・サーバーに予期しないに障害が発生し、サービスが中断される場合です。多くの場合、コンピュータ障害の原因はハードウェアの故障にあります。このようなタイプの障害の最善の解決方法は、クラスタ・テクノロジと高速データベース・クラッシュ・リカバリを利用する方法です。推奨されるソリューションには、Oracle Real Application Clusters（Oracle RAC）を使用したエンタープライズ・グリッド、ファスト・スタート・リカバリ、Oracle Data Guard および Oracle Streams があります。

この項の内容は、次のとおりです。

Oracle Real Application Clusters および Oracle Clusterware を使用したエンタープライズ・グリッドの概要

企業は、Oracle RAC を使用して、複数のシステムにわたって可用性とスケーラビリティの高いデータベース・サーバーを構築できます。Oracle RAC 環境では、Oracle Database は単一の共有データベースに同時にアクセスしながら、クラスタ内の 2 つ以上のシステム上で稼働します。そのため、複数のハードウェア・システムにまたがる単一のデータベース・システムが、アプリケーションでは単一の統合されたデータベース・システムとして認識されます。これにより、すべてのアプリケーションに対して次のような可用性とスケーラビリティ上の利点がもたらされます。

- 目的の容量やビジネス・ニーズの変更に応じてシステムをスケーリングできる、容量計画における柔軟性と費用対効果
- 特にコンピュータ障害時におけるクラスタ内のフォルト・トレランス

次のリストに、Oracle RAC 環境の機能を示します。

エンタープライズ・グリッド : Oracle RAC を使用すると、エンタープライズ・グリッドが有効になります。エンタープライズ・グリッドは、プロセッサ、サーバー、ネットワークおよび記憶域などの標準的な商品価格のコンポーネントによる大規模な構成から構築されます。Oracle RAC は、企業の役に立つ処理システム用としてこれらのコンポーネントを利用できる唯一のテクノロジです。Oracle RAC とグリッドは、運用コストを劇的に削減でき、システムの適応性、予防性および敏捷性を高めることができる柔軟性も備えています。ノード、記憶域、CPU およびメモリーを動的にプロビジョニングすることにより、使用状況の改善によってコストをさらに削減しながら、サービス・レベルを簡単かつ効率的に維持できます。また、Oracle RAC は Oracle RAC データベースにアクセスするアプリケーションからは完全に透過的であるため、既存のアプリケーションを変更せずに Oracle RAC にデプロイできます。

スケーラビリティ : Oracle RAC では、容量を増加するためにクラスタにノードを追加できる柔軟性があります。これにより、システムを段階的に拡張してコストを節約でき、小規模な単一ノード・システムを大規模なシステムに置き換えなくてすむようになります。標準的な低コストのコンピュータとモジュール・ディスク・アレイによるグリッド・プールにより、Oracle Database を使用したこのソリューションが強化されます。既存のシステムを新しい大規模なノードに置き換えてシステムをアップグレードする方法とは対照的に、1 つ以上のノードをクラスタに段階的に追加できるため、容量のアップグレード・プロセスがより簡単で迅速になります。Oracle RAC に実装されているキャッシュ・フュージョン・テクノロジと Oracle Database でサポートされている InfiniBand により、アプリケーションを変更しなくても容量をほぼ直線比例的にスケーリングできます。

フォルト・トレランス : Oracle RAC クラスタ・アーキテクチャのもう 1 つの主な利点は、複数のノードによって実現される本質的なフォルト・トレランスです。物理ノードは独立して動作しているため、1 つ以上のノードに障害が発生してもクラスタ内の他のノードには影響しません。フェイルオーバーはグリッド上の任意のノードで発生します。極端な場合、1 つのノードを除くすべてのノードが停止しても、Oracle RAC システムはデータベース・サービスを提供し続けます。このアーキテクチャでは、メンテナンスのためにノードのグループを透過的にオンラインまたはオフラインにすると同時に、クラスタの残りの部分によってデータベース・サービスを提供し続けることができます。Oracle RAC には、接続プールをフェイルオーバーするために Oracle Application Server との統合が組み込まれています。この機能により、TCP タイムアウトが発生するまで数十分待機することなく、アプリケーションに障害が即座に通知されます。アプリケーションは適切なリカバリ・アクションを即座に実行できます。そして、グリッドのロード・バランシングによってロードが徐々に再分配されます。

Oracle Clusterware: Oracle RAC は、クラスタを管理するためのクラスタウェアの完全セットも備えています。Oracle Clusterware には、ノード・メンバーシップ、メッセージ・サービスおよびロックなど、クラスタの実行に必要なすべての機能が用意されています。Oracle Clusterware は、一般的なイベントおよび管理 API を備えた完全統合スタックであるため、Oracle Enterprise Manager から一元的に管理できます。クラスタをサポートするために追加ソフトウェアを購入する必要はありません。このため、サード・パーティのクラスタウェアを統合およびテストするために余計な作業を行わずにすみます。また、Oracle Clusterware は、Oracle Database を使用可能なすべてのプラットフォームにわたって同じインタフェースを使用し、同じように動作します。Oracle では Oracle RAC とともに使用するサード・パーティのクラスタウェアを引き続きサポートしますが、サード・パーティのクラスタウェアを使用する必要やメリットはありません。

Oracle Clusterware のフレームワークの高可用性機能はアプリケーションに応じて拡張できます。つまり、Oracle Database および Oracle RAC の同じ高可用性メカニズムを使用して、カスタム・アプリケーションの可用性を高めることができます。Oracle Clusterware を使用すると、アプリケーションを監視、再配置および再起動できるため、アプリケーションのフェイルオーバーをデータベースのフェイルオーバーと統合および調和できます。

サービス : Oracle RAC は、サービスと呼ばれるエンティティをサポートしています。このサービスを定義することにより、データベースのワークロードをグループ化し、サービスを提供するために割り当てられる最適なインスタンスに処理をルーティングできます。サービスは、データベース・ユーザーまたはアプリケーションのクラスを表します。ビジネス・ポリシーを定義してこれらのサービスに適用し、ピーク処理時用のノードの割当てや、サービス障害の自動処理などのタスクを実行します。サービスを使用してシステム・リソースを必要な場所と時間に応じて適用することにより、ビジネス上の目標を達成します。

関連項目 :『Oracle Database 2 日で Real Application Clusters ガイド』

ファスト・スタート・リカバリ

計画外停止時間の最も一般的な原因の 1 つは、システム障害またはクラッシュです。システム障害は、ハードウェア障害、電源障害、およびオペレーティング・システムまたはサーバーのクラッシュの結果として発生します。これらの障害が原因で発生する中断の程度は、影響を受けるユーザーの数やサーバーのリストアに要する時間によって異なります。高可用性システムは、障害が発生したときに障害から迅速かつ自動的にリカバリできるよう設計されています。重要なシステムのユーザーは、障害から迅速にリカバリし、リカバリまでの時間を予測可能にするという確約を IT 組織に求めます。この確約時間よりも停止時間が長いと、システム運用に直接影響を及ぼし、収益や生産性の低下に繋がる可能性があります。

Oracle Database では、システム障害およびクラッシュから非常に短時間でリカバリできます。ただし、短時間でのリカバリと同等に重要なのが予測可能性です。Oracle Database に含まれているファスト・スタート・リカバリ・テクノロジは、Oracle Database に固有のテクノロジで、データベース・クラッシュ・リカバリ時間を自動的にバインドします。データベースによってチェックポイント処理が自己チューニングされ、必要な Recovery Time Objective が守られます。これにより、リカバリ時間が短く、予測可能になり、サービス・レベル目標を達成する能力が高まります。ファスト・スタート・リカバリ機能は、負荷の高いデータベースのリカバリ時間を数十分から数秒に短縮できます。

関連項目： ファスト・スタート・リカバリの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

Oracle Data Guard

Oracle Data Guard により、高可用性、データ保護、および企業データの障害時リカバリが保証されます。Data Guard では、スタンバイ・データベースは、トランザクション一貫性のあるプライマリ（本番）・データベースのコピーとして維持されます。計画停止または計画外停止でプライマリ・データベースが使用できなくなった場合、Data Guard により任意のスタンバイ・データベースがプライマリ用に切り替えられるため、停止時間が最小限に抑えられます。Oracle クライアントに統合されている Data Guard のファスト・スタート・フェイルオーバーと高速アプリケーション通知を使用した自動フェイルオーバーにより、高度なデータ保護とデータ可用性を実現します。

関連項目：『Oracle Data Guard 概要および管理』

Oracle Streams

Oracle Streams を使用して、柔軟な高可用性環境を構成できます。Oracle Streams を使用すると、本番データベースのローカルまたはリモート・コピーを作成できます。人為的エラーや大災害が発生した場合、このコピーを使用して処理を再開できます。

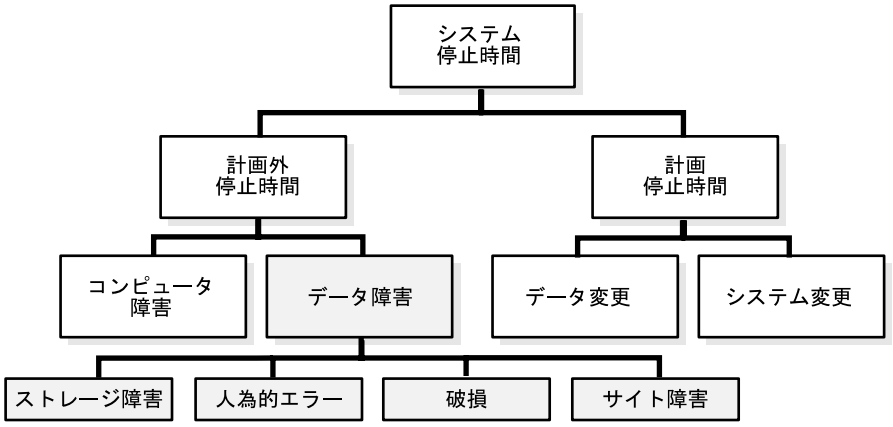
関連項目：『Oracle Streams 概要および管理』

データ障害に対する保護

データ障害とは、企業の重要なデータの消失、損害または破損です。データ障害の原因はコンピュータ障害よりも複雑かつ微妙です。この原因には、ストレージ・ハードウェアの障害、人為的エラー、破損またはサイト障害などがあります。

図 17-2 は、ストレージ障害、人為的エラー、破損およびサイト障害という 4 つのタイプのデータ障害に焦点を当てています。

図 17-2 データ障害が原因の停止時間



データ障害から保護してリカバリするためのソリューションの設計は非常に重要です。システムまたはネットワーク障害によってユーザーがデータにアクセスできなくなる可能性はありますが、適切なバックアップまたはリカバリ・テクノロジなしにデータ障害が発生すると、リカバリ作業に何時間もかかったり、データが失われる場合があります。

Oracle Database には、多くのデータ保護機能が用意されています。これらの多くの拡張機能を求める誘引により、データ保護およびリカバリを取り巻く新たな経済的側面が形成されています。過去 20 年間にわたって、ディスク容量は 3 桁大きくなった一方、MB 当たりのコストは大幅に小さくなりました。この傾向は一向に衰える兆しを見せません。これにより、バックアップ・メディアとしてのディスク記憶域のコストはテープに拮抗するようになりました。また、ディスク記憶域には、オンラインで使用できるという利点と、データにランダムにアクセスできるという利点もあります。

オラクル社ではこのような傾向に後押しされ、リカバリ計画について再考し、これらの経済的力学を利用した段階的なものとすることができました。Oracle Database で追加ディスク記憶域を使用することにより、バックアップとリカバリにかかる時間を数時間から数分に短縮できます。実質的に、廉価なディスク記憶域をコストの多大な停止時間と引き換えることが可能です。

この項の内容は、次のとおりです。

- ストレージ障害に対する保護
- 人為的エラーに対する保護

ストレージ障害に対する保護

単一データベース・インスタンスの記憶域のプロビジョニングは複雑であり、まして企業全体の記憶域のプロビジョニングなどは論外です。以前は、このプロセスに、次のステップが含まれていました。

1. 必要な領域の量の見積り
2. 最適なレイアウトの策定（ホット・スポットの発生を防止するためにデータファイルやアーカイブ・ファイルなどを格納する場所）
3. 論理ボリュームの作成
4. ファイル・システムの作成
5. データを保護およびミラー化する方法の定義と設定
6. データのバックアップおよびリカバリ計画の定義と実装
7. Oracle ソフトウェアのインストール
8. データベースの作成

この後には、パフォーマンスに悪影響を与えるホット・スポットの特定や、競合を減らすためのデータファイルの移動といった難しい作業が控えています。そして、いつかはディスク・クラッシュが発生したり領域が不足し、更新した記憶域構成のバランス再調整のために、ディスクの追加およびすべてのファイルの移動が必要になります。

ただし、この状況は、Oracle Database の自動ストレージ管理（ASM）という機能によって大きく変化しました。ASM では、垂直に統合されたファイル・システムおよびボリューム・マネージャが Oracle カーネルに直接提供されているため、データベース記憶域のプロビジョニング作業が大幅に削減されます。

ASM により、特化されたストレージ製品の費用、インストールおよびメンテナンスが不要になり、可用性が高まります。また、データベース・アプリケーション用の固有の機能が提供されます。ASM は、使用可能なすべての記憶域にまたがって分散され、パフォーマンスを最適化し、ミラー化も行い、データ破損に対する保護機能となります。ASM では、SAME（Stripe And Mirror Everything）の概念が拡張され、ディスク・レベル全体ではなくデータベース・ファイル・レベルでのミラー化を可能にする柔軟性が拡張されています。

最も重要なことは、ASM により、データおよびディスクの管理に伴う複雑な仕組みが排除され、ミラー化の設定、ディスクの追加および削除のプロセスが簡略化されることです。（大規模なデータ・ウェアハウスのように）数百、時には数千のファイルを管理するかわりに、ASM を使用するデータベース管理者は、より規模の大きなオブジェクトであるディスク・グループを作成および管理します。ディスク・グループは、1 つの論理単位として管理される一連のディスクを表します。基礎となるデータベース・ファイルのファイルの命名と配置を自動化することにより、DBA の時間を節約でき、標準のベスト・プラクティスに確実に準拠できます。

必要な場合、ASM に固有のミラー化メカニズムをストレージ障害に対する保護として使用できます。ミラー化はデフォルトで有効になっており、3 重ミラー化も可能です。ASM ミラー化により、障害グループを使用してデータの保護レベルを強化できます。障害グループは、障害を許容できる共通のリソース（ディスク・コントローラまたはディスク・アレイ全体）を共有するディスクのセットです。

ASM 障害グループを定義した後、データの冗長コピーは別の障害グループにインテリジェントに格納されるため、ストレージ・サブシステム内の任意のコンポーネントに障害が発生しても、データは透過的に保護されて使用可能であることが保証されます。また、ASM では、データの保護を強化するために Hardware Assisted Resilient Data 機能（後述の「データ破損に対する保護」の項を参照）をサポートしています。

関連項目：

- 『Oracle Database 2 日でデータベース管理者』
- 『Oracle Database ストレージ管理者ガイド』

人為的エラーに対する保護

停止時間の原因に関するほとんどの調査では、停止時間の最大の原因として人為的エラーがあげられています。不注意による重要データの削除や、UPDATE 文の不適切な WHERE 句により、意図した以上の行が更新されるなどの人為的エラーを可能なかぎり防止し、それに対する予防措置が機能しなかった場合は、元に戻す必要があります。Oracle Database には、このようなエラーが発生した場合に、エラーを迅速に診断してリカバリを行うために役立つ管理者用の使用しやすい強力なツールがあります。また、エンド・ユーザーが管理者の介入なしで問題からのリカバリを行えるようにする機能も装備しているため、管理者のサポート作業が軽減され、損失データや損傷データのリカバリ時間が短縮されます。

次の各項では、人為的エラーを防止する Oracle の機能について説明します。

人為的エラーに対する保護

エラーを回避する最善の方法は、ビジネスの実行に本当に必要なデータおよびサービスにユーザー・アクセスを制限することです。Oracle Database は、ユーザーの認証、および管理者による職務実行に必要な権限のみのユーザーへの付与を可能にすることで、アプリケーション・データへのアクセスを制御する様々なセキュリティ・ツールを提供しています。また、Oracle Database のセキュリティ・モデルには、仮想プライベート・データベース（VPD）機能を使用して行レベルでデータ・アクセスを制限できます。この機能を使用すると、アクセスする必要のないデータからユーザーをさらに隔離できます。

Oracle Flashback テクノロジ

権限を持つユーザーが誤りを作成した場合、そのエラーを修復するツールが必要です。Oracle Database は、フラッシュバックと呼ばれる人為的エラーを修復する一連のテクノロジを提供します。フラッシュバックにより、データ・リカバリに大きな変化がもたらされます。以前は、数分の内に破壊されたデータベースのリカバリに、数時間かかる場合がありました。フラッシュバック・テクノロジを使用することにより、エラーの発生にかかった時間と同じ時間内で修復が可能になります。フラッシュバック・テクノロジは非常に使用しやすく、1 つの短いコマンドを使用してデータベース全体をリカバリできるため、複雑な手順は必要ありません。フラッシュバックは Oracle Database に固有のテクノロジで、次の機能があります。

- 人為的エラーを迅速に分析して修復する SQL インタフェース
- 間違った顧客注文の削除など、部分的な損害を対象としたファイングレイン分析と修復
- ある月の全顧客オーダーが削除された場合など、より広範囲の損害の修正と、それによる停止時間の短縮
- 行、トランザクション、表、表領域およびデータベース全体を含むすべてのレベルでのリカバリ

表 17-1 は、フラッシュバック・テクノロジによる人為的エラーの修復方法を示しています。

表 17-1 Oracle Flashback テクノロジによる人為的エラーに対する保護

機能	説明
フラッシュバック問合せ	Oracle Flashback Query を使用すると、過去のある時点のデータを問い合せることができます。フラッシュバック問合せを使用して、誤って削除または変更された可能性がある損失データを表示し、再構築できます。次に例を示します。 `SELECT * FROM employee AS OF TIMESTAMP TO_TIMESTAMP('19-APR-05 02:00:00 PM') WHERE …` この文の場合、従業員表から特定の日付の 2:00 p.m. 時点における行が表示されます。開発者は、この機能を使用して、エラーを遅延なく元に戻し修正する作業を管理者ではなくエンド・ユーザー自身にさせるセルフサービス・エラー修正機能をアプリケーションに組み込むことができます。フラッシュバック問合せは、管理が容易です。これは、過去のある時点のデータを再構築するために必要な情報が、データベースによって自動的に保持されるためです。
フラッシュバック・バージョン問合せ	フラッシュバック・バージョン問合せにより、データベースの行レベルで行われた変更を参照できます。これは SQL の拡張機能で、指定した時間にわたり、ある行の異なるすべてのバージョンを取得できます。次に例を示します。 `SELECT * FROM employee VERSIONS BETWEEN TIMESTAMP TO_TIMESTAMP('19-APR-05 02:00:00 PM') AND TIMESTAMP TO_TIMESTAMP('19-APR-05 03:00:00 PM') WHERE …` この文の場合、4 月 19 日の 2:00 ～ 3:00 p.m. の間に異なるトランザクションによって変更された各行とともに、行の各バージョンが表示されます。この機能を使用すると、データがいつどのように変更されたかを把握でき、それをユーザー、アプリケーションまたはトランザクションまでトレースバックできます。このため、データベース内の論理的な破損の原因を追跡してそれを解決できます。また、アプリケーション開発者は、フラッシュバック・バージョン問合せを使用してコードをデバッグすることもできます。
フラッシュバック・トランザクション	Oracle Flashback Transaction により、トランザクションおよびその依存トランザクションを元に戻します。`DBMS_FLASHBACK.TRANSACTION_BACKOUT()` プロシージャは、データベースがオンラインのまま、トランザクションとその依存トランザクションをロールバックします。このリカバリ操作では UNDO データを使用して、影響されたデータを元の状態に戻す補正トランザクションが作成および実行されます。`DBA_FLASHBACK_TRANSACTION_STATE` ビューを問い合せることにより、トランザクションが依存性ルールを使用して元に戻されたか、次のいずれかの方法によって強制実行されたかについて、トランザクションの現在の状態を参照できます。 ■ 非競合行を元に戻す ■ UNDO SQL を適用する Oracle Flashback Transaction では、データベースがオンラインのまま 1 つのコマンドを使用して、特定のトランザクションまたは一連のトランザクションとその依存トランザクションを簡単かつ迅速に元に戻すことができるため、論理リカバリ時の可用性が向上します。
フラッシュバック・トランザクション問合せ	フラッシュバック・トランザクション問合せにより、データベースのトランザクション・レベルで行われた変更を参照できます。これは、SQL の拡張機能で、トランザクションによって行われたすべての変更を参照できます。次に例を示します。 `SELECT * FROM FLASHBACK_TRANSACTION_QUERY WHERE XID = '000200030000002D';` この問合せの場合、このトランザクションによって行われたすべての変更結果が表示されます。また、これを補正する SQL 文も戻されるため、この文を使用して、このトランザクションによって行われたすべての行への変更を元に戻すことができます。データベース管理者やアプリケーション開発者は、このような精密なツールを使用することにより、データベースまたはアプリケーション内の論理的な問題を正確に診断して解決できます。
フラッシュバック・データベース	Oracle データベースを過去のある時点の状態に戻す場合、従来の方法では、ポイント・イン・タイム・リカバリを使用します。ただし、ポイント・イン・タイム・リカバリでは、データベース全体をバックアップからリストアし、エラーがデータベースに伝播する直前の状態を修復するため、数時間、あるいは数日もかかる場合があります。データベースのサイズが増加し続けると、データベース全体の単なるリストアに、数時間あるいは数日かかります。 フラッシュバック・データベースは、ポイント・イン・タイム・リカバリを実行するための計画です。これにより、Oracle データベースを以前の時点まで即座に巻き戻し、論理データの破損やユーザー・エラーによる問題を解決できます。この機能では、フラッシュバック・ログを使用して、変更されたブロックの旧バージョンを取得します。これは、連続的バックアップやストレージ・スナップショットと似ています。リカバリを実行する必要がある場合、フラッシュバック・ログが迅速に再実行され、エラーが発生した時点までデータベースがリストアされ、変更されたブロックのみがリストアされます。フラッシュバック・データベースは高速で、リカバリ時間は時間単位から分単位まで短縮されます。たとえば、次のコマンドを発行し、データベースを 2:05 p.m. までリカバリします。 `FLASHBACK DATABASE TO TIMESTAMP TO_TIMESTAMP('19-APR-05 02:05:00 PM');`

表 17-1 Oracle Flashback テクノロジによる人為的エラーに対する保護（続き）

機能	説明
フラッシュバック表	フラッシュバック表により、表または一連の表を過去の指定の時点に戻します。多くの場合、フラッシュバック表を使用すると、より複雑なポイント・イン・タイム・リカバリ操作を実行しなくてすむようになります。次に例を示します。 `FLASHBACK TABLE orders, order_items TIMESTAMP TO_TIMESTAMP('07-APR-2005 02:33:00 PM');` このコマンドの場合、現在時刻と過去の指定したタイムスタンプの間に行われた orders および order_items 表の更新が巻き戻されます。フラッシュバック表により、この操作がオンラインで適切に実行され、各表の間の参照整合性制約が保持されます。フラッシュバック表は、表または一連の関連表に対して巻戻しまたは元に戻すボタンのように機能します。
フラッシュバック・ドロップ	データベース・オブジェクトを誤ってドロップ、つまり削除してしまう誤りは、ユーザーが犯しやすい誤りです。これは、実際には本番データベースに接続されているとき、テスト・データベースに接続されていると勘違いした場合に発生します。この誤りにはすぐに気付くことになりますが、そのときにはすでに遅く、削除した表とその索引、制約およびトリガーを簡単にリカバリすることは不可能です。削除した後、オブジェクトは永遠に失われてしまいます。索引は再構築できますが、重要な表や他のオブジェクト（パーティションまたはクラスタなど）に対してはポイント・イン・タイム・リカバリを実行する必要があり、この作業には非常に時間がかかり、最新のトランザクションが失われる可能性があります。 フラッシュバック・ドロップは、Oracle Database でオブジェクトを削除するときの安全策となります。ユーザーが表を削除すると、Oracle Database によってこの表はごみ箱に入れられます。ごみ箱は、削除されたすべてのオブジェクトが入れられた仮想コンテナです。オブジェクトを完全に削除することを選択するか、この表が含まれる表領域の容量が上限に達するまで、オブジェクトはごみ箱内に残ります。削除した表とその依存オブジェクトはごみ箱から元に戻すことができます。たとえば、次のコマンドを使用すると、表 employee とその依存オブジェクトの削除は元に戻されます。 `FLASHBACK TABLE employee TO BEFORE DROP;`
フラッシュバック・リストア・ポイント	Oracle データベースのポイント・イン・タイム・リカバリ操作が必要な場合、データをロールバックする時間またはシステム変更番号（SCN またはトランザクション時間）を決定する必要があります。Oracle Database では、リストア・ポイントが使用されます。これは、フラッシュバック・データベース、フラッシュバック表および Recovery Manager（RMAN）とともに使用するときに SCN または実時間のかわりに使用できるユーザー定義のラベルです。リストア・ポイントには、データベースが良好な状態であった既知の時間を記録する機能があるため、データベースに対して実行された不適切なアクションを迅速かつ簡単に巻き戻すことができます。また、以前のデータベースのリカバリを介してフラッシュバックし、リセット・ログを開く機能も用意されています。保証付きリストア・ポイントを使用すると、データベースを巻き戻すために必要な UNDO を保持することにより、主なデータベースの変更（データベースのバッチ・ジョブ、アップグレードまたはパッチなど）を簡単に元に戻すことができます。リストア・ポイントを使用すると、既知の時間まで簡単に巻き戻すことが可能になります。 Oracle Data Guard 環境では、この機能を使用して、読取り / 書込みモードでオープンされたフィジカル・スタンバイ・データベースで変更を後でフラッシュバックし、このデータベースを本番データベースと同期が維持されているフィジカル・スタンバイ・データベースに戻すこともできます。スイッチオーバー操作の後に論理エラーが見つかった場合、プライマリおよびスタンバイ・データベースをスイッチオーバー操作の前の時点または SCN までフラッシュバックできます。また、スタンバイ・データベースを本番データベースと迅速に同期させるには、2 つのデータベースが同一でなくなった後に生成されたすべての REDO データを適用するかわりに、本番データベースの増分バックアップをスタンバイ・データベースに適用します。これにより、2 つのデータベースを再同期化させるために要する時間を大幅に短縮できます。
フラッシュバック・ログを使用したブロック・リカバリ	ブロック・リカバリを使用して必要に応じて、フラッシュバック・ログから最新のデータ・ブロックのコピーを取得し、リカバリ時間を短縮できます。また、インスタンス・リカバリ中にブロックが破損しても、インスタンス・リカバリは失敗しません。ブロックは自動的に破損としてマークが設定され、V$DATABASE_BLOCK_CORRUPTION 表の RMAN 破損リストに追加されます。これにより、後で RMAN の RECOVER BLOCK コマンドを発行し、関連するブロックを修正できます。
フラッシュバック・データ・アーカイブ	フラッシュバック・データ・アーカイブには、表の存続期間中におけるトランザクションのすべての変更を追跡して格納する機能があります。このインテリジェント機能をアプリケーションに組み込む必要はありません。フラッシュバック・データ・アーカイブは、レコード・ステージング・ポリシーおよび監査レポートに準拠させるときに役に立ちます。

関連項目：

- Oracle Flashback Query の詳細は、第 13 章「データの同時実行性と整合性」を参照してください。
- Oracle Flashback Database と Oracle Flashback Table の詳細は、第 15 章「バックアップおよびリカバリ」を参照してください。
- フラッシュバック・トランザクションの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

SQL ベースの LogMiner ログ・アナライザ

Oracle ログ・ファイルには、Oracle データベースのアクティビティおよび履歴に関する有用な情報が豊富に含まれています。ログ・ファイルには、データベース・リカバリに必要なすべてのデータが含まれています。これらのファイルには、データベース内のデータおよびメタデータに加えられたすべての変更も記録されています。LogMiner は、SQL を使用した REDO ログ・ファイルの読取り、分析および解析が可能な完全なリレーショナル・ツールです。LogMiner によるログ・ファイルの分析を使用すると、データへの変更の追跡または監査、チューニングおよび容量計画に関する補足情報の提供、複雑なアプリケーションのデバッグに関する重要な情報の取得、または削除したデータのリカバリが可能になります。

関連項目： 11-5 ページの「LogMiner の概要」

データ破損に対する保護

破損は、I/O スタック内の不良コンポーネントによって引き起こされます。たとえば、データベースにより、更新トランザクションの結果として I/O が発行されます。データベース I/O は次に対して渡されます。

1. オペレーティング・システムの I/O コード
2. ファイル・システム
3. ボリューム・マネージャ
4. デバイス・ドライバ
5. ホスト・バス・アダプタ
6. ストレージ・コントローラ
7. I/O が最終的に書き込まれるディスク・ドライブ

I/O シーケンス内のいずれかのコンポーネントに不具合またはハードウェア障害があると、データ内の一部のビットが反転され、破損データがデータベースに書き込まれる場合があります。データベース制御情報やユーザー・データが破損することもあり、その場合は、データベースの機能性や可用性が深刻な打撃を受けるおそれがあります。同様に、ディスク障害もデータベース・ファイルに損害を与え、データベースのリカバリにバックアップが必要となる場合があります。

データ破損に対する保護の内容は、次のとおりです。

- Oracle Hardware Assisted Resilient Data（HARD）
- 書込み欠落
- データ・リカバリ・アドバイザ
- フラッシュ・バックアップおよびリカバリ

Oracle Hardware Assisted Resilient Data（HARD）Oracle Hardware Assisted Resilient Data（HARD）により、データ破損を未然に防ぎます。データ破損はほとんどありませんが、発生した場合にはデータベース、つまりビジネスに壊滅的な影響があります。ストレージ・デバイス内に Oracle のデータ検証アルゴリズムを実装することにより、破損データが永続ストレージ上のデータベース・ファイルに書き込まれるのを防ぐことができます。上位のソフトウェアから下位のハードウェアまでの、このようなエンドツーエンドの検証は、Oracle とストレージ・パートナが提供する独自の機能です。

Oracle が保護情報を検証してデータベース・ブロックに追加し、この保護情報がストレージ・デバイスによって検証されます。HARD により、データベースとストレージ間の I/O パスへの破損データの侵入が検出され、データベース業界で今までは防止が不可能だった大規模障害が除去されます。RAID はデータを物理的に確実に保護することで、ストレージ業界から幅広く支持されていますが、HARD は物理ビットの保護からビジネス・データの保護へと進化することで、データ保護を 1 つ上のレベルに高めています。

> **関連項目：** HARD の詳細は、`http://www.oracle.com/technology/deploy/availability/htdocs/HARD.html` を参照してください。

書込み欠落 書込み欠落はデータ破損のもう 1 つの形態ですが、こちらの方が迅速な検出および修復がより困難です。データ・ブロックの書込み欠落が発生するのは、次のような場合です。

- I/O サブシステムによってブロック書込みの完了が確認されていますが、実際には永続ストレージで書込みは行われていません。後続のブロック読取り時に、I/O サブシステムによってデータ・ブロックの失効バージョンが戻され、これを使用してデータベースの他のブロックが更新され、データベースが破損する場合があります。
- 書込み I/O は完了しましたが、別の場所に書き込まれたため、後続の読取り操作では失効値が戻されます。
- 別のノードでの書込み I/O の後、1 つのクラスタ・ノードからの読取り I/O によって失効データが戻されます。たとえば、NFS キャッシング・ポリシーに Oracle RAC との互換性がない場合、このようになる可能性があります。

データ・ブロック破損の防止および検出 Oracle Database 11*g* より前は、RMAN によって検出されたブロック破損は `V$DATABASE_BLOCK_CORRUPTION` に記録されていました。Oracle Database 11*g* では、RMAN を含む複数のデータベース・コンポーネントおよびユーティリティがブロック破損を検出し、そのビューに記録できるようになりました。Oracle Database では、ブロック破損が検出または修復されるとこのビューが自動的に更新されます（たとえば、ブロック・メディア・リカバリまたはデータファイル・リカバリなどを使用）。これには、ブロック破損の検出に要する時間が短縮されるという利点があります。

データ・リカバリ・アドバイザ データ・リカバリ・アドバイザを使用すると、永続的な（ディスク上の）データ障害が自動的に診断され、適切な修復オプションが提示されて、ユーザーのリクエストで修復操作が実行されます。この機能は、次のとおりです。

- 障害診断
- 障害の影響評価
- 修復生成
- 修復の実現可能性チェック
- 修復自動化
- データ整合性およびデータベースのリカバリ可能性の検証
- 破損の早期検出
- データ検証と修復の統合

データ・リカバリ・アドバイザの初期リリースは、Oracle RAC をサポートしていません。また、データ・リカバリ・アドバイザを使用して Data Guard 構成でプライマリ・データベースは管理できますが、データ・リカバリ・アドバイザを使用してフィジカル・スタンバイ・データベースはトラブルシューティングできません。Oracle Enterprise Manager 11*g* Grid Control を使用している場合、データ・リカバリ・アドバイザでは、修復計画の推奨時にスタンバイ・データベースの存在のみが考慮されます。

フラッシュ・バックアップおよびリカバリ 企業データのバックアップのかわりとなるものは存在しません。まれに複数の障害により、ストレージ・サブシステム内でミラー化されているデータであっても使用不可能になることがあります。Oracle では、すべてのデータを適切にバックアップし、以前のバックアップからデータをリストアし、障害発生直前までにそのデータに行われた変更をリカバリする、オンライン・ツールを提供しています。

大規模なデータベース・システムをバックアップすることは、簡単ではありません。大規模なデータベースは、多数の異なるディスクに分散された数百単位のファイルで構成されている場合があります。重要なファイルのバックアップを怠ると、データベース・バックアップ全体が使用不可能になることがあります。通常、破損したこのようなファイルは、それが必要になって初めて検出されます。Recovery Manager（RMAN）は、Oracle Database のバックアップ、リストアおよびリカバリ・プロセスの管理ツールです。RMAN により、バックアップ・ポリシーが作成および維持され、すべてのバックアップおよびリカバリ・アクティビティが分類整理されます。バックアップおよびリストア時にすべてのデータ・ブロックについて破損を分析することにより、破損データがバックアップに伝播するのを防止できます。最も重要なのは、Recovery Manager を使用すると、必要なすべてのデータファイルがバックアップされ、データベースがリカバリ可能であることが保証されることです。

Recovery Manager は、ユーザー指定の期間内でデータベースをリストアするために必要なファイルを自動的に追跡します。Recovery Manager は、データベースの残りの部分がオンラインのまま、中断されていた操作を自動的に再開し、破損したログ・ファイルを処理し、個々のデータ・ブロックをリストアできます。

RMAN は、データベースのバックアップおよびリカバリ機能を大幅に強化します。RMAN は、フラッシュ・リカバリ領域へのすべてのデータのバックアップおよびリカバリを自動管理できます。フラッシュ・リカバリ領域は、Oracle データベース内のすべてのリカバリ関連ファイルおよびアクティビティ用の統合されたディスク・ベースの格納位置です。テープのかわりにディスクにバックアップすることで、より高速のバックアップが可能になります。しかしさらに重要なことは、データベースのメディア・リカバリが必要な場合に、データベースのリカバリ時間を大幅に短縮する、データファイルのバックアップをすぐに使用できることです。

フラッシュ・リカバリ領域のファイルは、Recovery Manager により管理されます。RMAN は、フラッシュ・リカバリ領域のすべてのバックアップを作成し、この領域を管理します。アーカイバは REDO データをフラッシュ・リカバリ領域に書き込み、RMAN では、不要になった古いバックアップおよびアーカイブ REDO ログが自動的に削除（またはテープに移動）されます。`RETENTION POLICY` を 7 日間のリカバリ期間に設定すると、RMAN により、データベースを 7 日前までリカバリするために必要なすべてのバックアップが保持されます。7 日より前の時点までリカバリする必要がある場合は、テープからデータがリストアされます。Oracle Enterprise Manager には、ベスト・プラクティスの実装を含む、フラッシュ・バックアップおよびリカバリを実行するための完全インタフェースが用意されています。

増分バックアップは、Oracle8 データベースで最初にリリースされてから RMAN の一部となっています。増分バックアップにより、前回のバックアップ以降に変更されたブロックのみが記録されます。Oracle Database には、ブロック・チェンジ・トラッキングが実装されているため、増分バックアップをより速く取得できます。Oracle Database は、すべてのデータベース変更の物理位置を追跡します。RMAN は、この変更追跡情報を自動的に使用して、増分バックアップ時に読み取る必要があるブロックを判断し、そのブロックに直接アクセスしてバックアップします。この増分バックアップを、前に作成したイメージ・バックアップとマージすることにより、リカバリ時間を最小限に抑えることができます。次第に増加する更新されたバックアップに基づくバックアップ方法によって、メディア・リカバリに必要な時間を最小限にできます。バックアップ方法の変更追跡部分を使用して増分バックアップを行うことにより、日次バックアップに必要な時間が短縮され、ネットワークを介してバックアップを実行する際のネットワーク帯域幅を節約でき、ログに記録されていない変更がデータベースをリカバリでき、バックアップ・ファイルの記憶域が削減され、データベースのリカバリ時間が短縮されます。

Oracle Database のバックアップおよびリカバリには、他にも多くの革新的機能が用意されています。

- バックアップの圧縮
- リストア時に欠落または破損したバックアップが検出されたときの以前のバックアップへの自動フェイルオーバー
- 以前のポイント・イン・タイム・リカバリを介した自動リカバリ、またはリセット・ログを使用したリカバリ
- リカバリ時の新規データベースおよび一時ファイルの自動作成
- バックアップまたはリストアでの自動チャネル・フェイルオーバー
- 表領域の自動ポイント・イン・タイム・リカバリ
- ミラー分割を高速化するための全データベースの `BEGIN BACKUP` コマンド
- リカバリ並列性の向上（2 ～ 4 倍）によるリカバリの高速化
- 表領域名の変更
- アーカイブ REDO ログ用のプロキシ（サード・パーティ）・バックアップ
- 時間枠ベースのバックアップの制限
- クロス・プラットフォーム・トランスポータブル表領域

関連項目：

- バックアップおよびリカバリ・ソリューションの詳細は、第 15 章「バックアップおよびリカバリ」を参照してください。
- RMAN とバックアップおよびリカバリ・ソリューションの詳細は、『Oracle Database バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

サイト障害に対する保護

データ保護機能により、サイトの処理を不可能にする壊滅的な事象からサイトを長期的に保護します。例としては、ファイルの破損、自然災害、停電、通信障害、さらにはテロなどがあります。Oracle Database は、本番データベースのローカルまたはリモート・コピーを作成し保持する、様々なデータ保護ソリューションを提供します。破損または災害が発生した場合、データのユーザーはリモート・データベースにアクセスすることで自分のタスクを続行できます。

データ保護の最もシンプルな形式は、データベース・バックアップのオフサイト・ストレージです。データ・センターが適正な時間内にサービスを再開できない場合、バックアップを他のサイトのシステムでリストアでき、ユーザーはこのバックアップ・システムに接続できます。ただし、他のシステムでバックアップをリストアするには時間がかかり、しかもバックアップは完全には最新でない可能性があります。より迅速なリカバリを可能にし、災害時でさえも継続的なデータベース・サービスを可能にする機能として、Oracle は Data Guard を提供しています。これについては、次の各項で説明します。

- Oracle Data Guard およびスタンバイ・データベース
- データ損失ゼロの REDO 転送
- リアルタイム適用およびフラッシュバック・データベース
- Data Guard Broker
- ファスト・スタート・フェイルオーバー

Oracle Data Guard およびスタンバイ・データベース Data Guard は、あらゆる Oracle Database 障害時リカバリ計画の基盤となるものです。Data Guard には、本番データベースのスタンバイ・コピーを設定および保持できる機能が用意されています。このスタンバイ・データベースは、本番データベースから見て地球の反対側に配置することも、同じデータ・センターに配置することも可能です。Data Guard には、複雑なタスクを自動化する拡張機能があり、優れた監視、アラートおよび制御メカニズムが用意されています。このため、データベースはデータ・センターの災害に耐えることができます。また、フェイルオーバーが必要な場合にはサーバーをスタンバイ・データベースに動的に追加できるため、Data Guard はグリッド・クラスタ間で透過的に動作します。

Oracle Data Guard は、Redo Apply テクノロジを使用するフィジカル・スタンバイ・データベース、スナップショット・スタンバイ・データベース、および SQL Apply テクノロジを使用するロジカル・スタンバイ・データベースをサポートしています。

- フィジカル・スタンバイ・データベースおよび Redo Apply

  Redo Apply モードの Data Guard は、フィジカル・スタンバイ・データベースと呼ばれる本番データベースのコピーを保持し、それを本番データベースと常に同期させます。プライマリ・データベースの REDO データは、スタンバイに送信され、メディア・リカバリの際に物理的に適用されます。スタンバイ・データベースは、物理的にプライマリと同一です（ただし、遅延が生じる場合があります）。また、REDO の適用中にスタンバイ・データベースを読取り専用モードでオープンできるため、スタンバイ・データベースを使用して本番データベースからレポート処理をオフロードすることもできます。スタンバイ・データベースで作成されたバックアップを使用して本番データベースのリカバリを実行できるため、バックアップ処理を本番データベースからオフロードすることもできます。

  フィジカル・スタンバイ・データベースは、障害やデータ・エラーからの保護機能として有効です。エラーまたは災害の発生時に、フィジカル・スタンバイ・データベースをオープンして使用することで、データ・サービスをアプリケーションおよびエンド・ユーザーに提供できます。スタンバイ・データベースへの変更の適用には優れたメディア・リカバリ・メカニズムが使用されているため、スタンバイ・データベースはすべてのアプリケーションでサポートされ、最大のトランザクション・ワークロードにも簡単に効率よく対処できます。

- スナップショット・スタンバイ・データベース

  スナップショット・スタンバイ・データベースは、フィジカル・スタンバイ・データベースから作成する更新可能なスタンバイ・データベースです。スナップショット・スタンバイ・データベースは、プライマリ・データベースから REDO データを受け取ってアーカイブしますが、スタンバイ・データベースが読取り / 書込みモードでオープンされている間は、プライマリ・データベースの REDO データを適用しません。このため、通常、時間が経つにつれてスナップショット・スタンバイ・データベースとプライマリ・データベースの相違が大きくなります。また、スナップショット・スタンバイ・データベースをローカルで更新すると、この相違はさらに大きくなります。

  REDO データはスナップショット・スタンバイ・データベースをフィジカル・スタンバイ・データベースに戻すまで適用されないため、ローカルで行われたスナップショット・スタンバイ・データベースの更新はすべて破棄されます。単一コマンドを使用して、スナップショット・スタンバイ・データベースをフィジカル・スタンバイ・データベースに戻すことができます。この時点で、スナップショット・スタンバイ状態で行われた変更は破棄され、Redo Apply により、アーカイブされた REDO を使用してフィジカル・スタンバイ・データベースとプライマリ・データベースが自動的に再同期されます。

- ロジカル・スタンバイ・データベースおよびSQL Apply

  SQL Applyモードの Data Guard は、プライマリ・データベースから REDO データを受け取り、これを SQL トランザクションに変換し、オープン状態のスタンバイ・データベースに適用します。ロジカル・スタンバイ・データベースは、物理的にはプライマリ・データベースと異なる場合もありますが、論理的にはプライマリ・データベースと同一で、プライマリ・データベースが破棄された場合には、処理を引き継ぐことができます。トランザクションは SQL を使用してオープン状態のデータベースに適用されるため、スタンバイ・データベースは、他のタスクに対して同時に使用でき、本番データベースと物理構造が異なる場合もあります。たとえば、ロジカル・スタンバイ・データベースは、意思決定支援用として使用することや、プライマリ・データベースにはない別の索引やマテリアライズド・ビューを使用してレポート用として最適化することが可能です。

  最も重要なのは、SQL Apply はデータ保護機能であることです。これは、SQL Apply は、ログ・ファイル内の変更前の値とロジカル・スタンバイ・データベース内の変更前の値を比較することにより、論理的な破損をチェックできるためです。これにより、ロジカル・スタンバイ・データベースは、起こり得る最大範囲の破損からデータを保護できるようになります。

  ロジカル・スタンバイ・データベースは、リカバリ中に読取り / 書込み I/O が可能なため、SQL Apply によって REDO ログへの変更が適用されている間も、スタンバイ・データベースを問い合せることができます。

データ損失ゼロの REDO 転送 フィジカルとロジカルの両方のスタンバイ・データベースで、同じ転送サービスが使用されます。Data Guard では、同期転送方式と非同期転送方式を選択できます。Data Guard の同期転送サービスにより、データ損失ゼロの保護が実現されるため、プライマリ・データベースに障害が発生した場合、以前にコミットしたすべてのトランザクションの保持に必要な REDO データがスタンバイ・サイトで使用可能になります。

また、REDO データをスタンバイ・サイトに非同期方式で転送することもできます。これにより、潜在的なデータ損失を最小限に抑えると同時に、遠距離転送であっても最適なパフォーマンスを実現し、ネットワーク障害からデータを保護できます。

リアルタイム適用およびフラッシュバック・データベース リアルタイム適用により、Data Guard の適用サービスでは、現行ログ・ファイルがスタンバイ・データベースにアーカイブされるのを待機せずに、REDO データをプライマリ・データベースから受信すると同時にスタンバイ・データベースに適用できます。これにより、スタンバイ・データベースをプライマリ・データベースと密接に同期できるため、最新かつリアルタイムのレポートを使用できるようになります。また、スイッチオーバーおよびフェイルオーバー時間を短縮できるため、ビジネスの計画停止時間と計画外停止時間の短縮につながります。さらに、フラッシュバック・データベースをプライマリとスタンバイの両方のデータベースで使用することを選択して、データベースを過去のある時点の状態に短時間で戻し、ユーザー・エラーを撤回できます。あるいは、スタンバイ・データベースにフェイルオーバーすると決定したが、これらのユーザー・エラーはすでにスタンバイ・データベースに適用されている場合（リアルタイム適用が有効であったため）、スタンバイ・データベースを安全な過去のある時点の状態にフラッシュバックできます。これら 2 つの機能を使用すると、スタンバイ・データベースを最新の状態に維持するか、または本番データベースでの人為的エラーがスタンバイ・データベースに伝播することを防ぐために Redo Apply を遅延させるかを選択するという、ときには必要な妥協が解消されます。

Data Guard Broker プライマリ・データベースとスタンバイ・データベース、およびこれらによる各種対話は、SQL*Plus を使用して管理できます。Data Guard には、管理を容易にするために、Data Guard Broker 分散管理フレームワークも用意されています。これにより、Data Guard 構成の作成、保守および監視が自動化されて一元化されます。Oracle Enterprise Manager または Broker 独自の特別なコマンドライン・インタフェース（DGMGRL）を使用すると、Broker の管理機能を利用できます。使用しやすい Oracle Enterprise Manager GUI からマウスをシングルクリックするのみで、プライマリ・データベースからいずれかのタイプのスタンバイ・データベースに対するフェイルオーバー処理を開始できます。Broker と Oracle Enterprise Manager を使用すると、スタンバイ・データベースを簡単に管理かつ操作できます。フェイルオーバーやスイッチオーバーなどのアクティビティを容易にすることにより、エラーが発生する可能性が大幅に小さくなります。

ファスト・スタート・フェイルオーバー ファスト・スタート・フェイルオーバーを使用すると、本番データベースからスタンバイ・データベースへのデータベース処理のフェイルオーバーを人間の介在なしに完全に自動化できるため、フォルト・トレラントなスタンバイ・データベース環境の作成が可能になります。ファスト・スタート・フェイルオーバーにより、管理者は、フェイルオーバー操作を起動および実装するために複雑な手動ステップを実行する必要がなくなるため、プライマリ・データベースが消失する事態が発生しても、同期化された指定のスタンバイ・データベースに自動的に迅速かつ確実にフェイルオーバーできます。これにより、停止時間は大幅に短縮されます。ファスト・スタート・フェイルオーバーは、Oracle Enterprise Manager または Data Guard Broker を使用して設定します。Data Guard 環境を監視するオブザーバにより、必要に応じてフェイルオーバーが自動的にトリガーされて完了します。ファスト・スタート・フェイルオーバーが発生した後、構成への再接続時に、古いプライマリ・データベースは Broker によって新しいスタンバイ・データベースとして自動的に復元されます。これにより、Data Guard 構成では構成における障害時の保護を簡単かつ迅速にリストアできるため、Data Guard 構成の堅牢性が向上します。これらの機能により、Data Guard は、透過的なビジネスの継続性の維持に役立つのみでなく、障害時リカバリ構成の管理コストも削減します。

関連項目：

- 『Oracle Data Guard 概要および管理』
- 『Oracle Data Guard Broker』

計画メンテナンス中の停止時間の回避

特に複数のタイムゾーンのユーザーをサポートしているグローバル・エンタープライズでは、計画停止時間は運用に大きな損害を与える可能性があります。このような場合、計画的な中断を最小化するようシステムを設計することが重要です。計画停止時間には、ルーチン操作、定期的なメンテナンスおよび新規デプロイメントが含まれます。

ルーチン操作とは頻繁なメンテナンス作業を指し、これには、バックアップ、パフォーマンス管理、ユーザーやセキュリティの管理およびバッチ操作があります。また、パッチのインストールやシステムの再構成などの定期メンテナンスは、データベース、アプリケーション、オペレーティング・システム、ミドルウェアまたはネットワークを更新するために必要となる場合があります。新規デプロイメントは、ハードウェア、オペレーティング・システム、データベース、アプリケーション、ミドルウェアまたはネットワークへの大規模なアップグレード時に必要となる場合があります。アップグレードを実行するタイミングのみでなく、その変更によるアプリケーション全体への影響も考慮する必要があります。

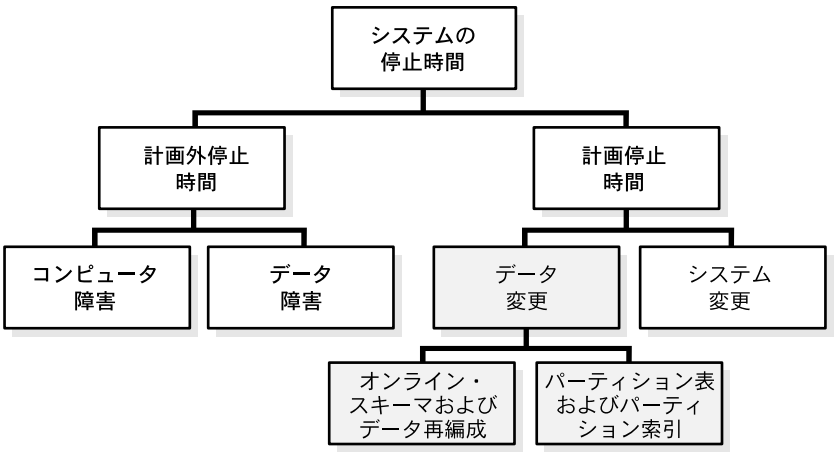
インターネットは、データのグローバル共有を容易にしましたが、データ可用性に対する新しい問題と要件も提起しました。世界各国のユーザーが 1 日 24 時間データにアクセスするため、メンテナンスのための時間帯はなくなったも同然です。計画停止時間による運用の中断は、計画外停止時間による中断と同様に深刻な問題となっています。ユーザーが影響を受けない時間枠は、存在しません。データベースに格納されるデータ量が膨大になると、メンテナンス操作に非常に時間がかかります。このような作業は、データのユーザーに影響を与えずに行うことが重要です。

この項の内容は、次のとおりです。

データ変更の停止時間の回避

図 17-3 は、オンライン・スキーマおよびデータ再編成、パーティション表とパーティション索引、およびリソースの動的プロビジョニングという、データ変更が原因である 3 つのタイプの停止時間に焦点を当てています。

図 17-3 データ変更が原因の停止時間



この項の内容は、次のとおりです。

- オンライン・スキーマおよびデータ再編成
- パーティション表とパーティション索引

オンライン・スキーマおよびデータ再編成

Oracle Database では、データベース操作やユーザーに対するデータ可用性を阻害せずに、ほとんどのメンテナンス操作を実行できます。データベースがオンライン状態で、エンド・ユーザーがデータの読取りまたは更新を行っている場合でも、索引の追加、再作成または断片化解消が可能です。同様に、オンライン状態での表の再配置や断片化解消も可能です。表の再定義、表タイプの変更、列の追加、削除または名前変更、および記憶域パラメータの変更は、基盤となるデータの表示または更新を行っているエンド・ユーザーの操作を中断せずに実行できます。Oracle Database には、次の機能があります。

- 表の索引、権限付与、制約および他の特性の簡単なクローニングのサポート
- LONG データ型から LOB データ型へのオンライン変換
- 主キーを必要としない一意索引の使用
- シノニムを介して参照される PL/SQL ストアド・プロシージャおよびビューの本体を、これらを参照する他の PL/SQL パッケージの再コンパイルなしに変更する機能
- オンラインでセグメントの縮小を実行し、セグメント内で使用される領域をオンラインで適切に圧縮することにより、システムやデータの可用性に影響を与えずに領域管理を実行する機能

パーティション表とパーティション索引

データベースが大きくなると、その管理作業が極端に膨大になることがあります。管理者は、データベース表および索引をパーティション化する機能を使用することにより、大きい表を、小さく管理の容易な表に分割できます。パーティション化により、大半の操作やスキーマ変更をオンライン状態に維持しながら、メンテナンス作業を一度に 1 パーティションずつ行うことができます。このため、メンテナンス中にデータが受ける影響は最小限になります。また、パーティションにより、パラレル実行が可能になるため、ほとんどの操作の実行時間が大幅に短縮されます。

パーティションの他の利点としては、障害の伝播防止があります。メディア障害や破損などの障害は、障害が発生したディスク上のパーティション以外には伝播されません。影響を受け、リカバリを必要とするのは、そのパーティションのみです。これにより、リカバリ時間が短縮されるのみでなく、障害が発生したパーティションのリカバリ中に、他の影響を受けていないパーティションをオンライン状態に維持できます。

一般に、大きい表の中のすべてのデータが、同じアクセス特性を持っているわけではありません。通常、保留中のオーダーはクローズされたオーダーよりも頻繁にアクセスされ、または前四半期の販売実績は 3 年前の四半期の販売実績よりも頻繁に分析されます。パーティション化により、データのインテリジェントなストレージ管理が可能になります。頻繁にアクセスされるデータは、最速のディスク上に格納し、アクセスの集中するデータは、多数のドライブにストライプ化できます。

システム変更の停止時間の回避

計画的なシステム変更が行われるのは、ルーチン操作、定期的なメンテナンス操作および新規デプロイメントを実行する場合です。計画的なシステム変更には、データベース内の組織データ構造の外部で行われる運用環境の計画的変更が含まれます。

計画的なシステム変更によるサービス・レベルでの影響は、次の条件に応じて大きく異なります。

- 計画停止の性質と範囲
- 変更の実装前に行われるテストおよび検証作業
- 影響を最小限に抑えるための適切なテクノロジと機能

この項の内容は、次のとおりです。

- ローリング・パッチ更新
- ローリング・リリース・アップグレード
- リソースの動的プロビジョニング

ローリング・パッチ更新

Oracle Database は、ローリング方式による Oracle RAC システムのノードへのパッチの適用をサポートします。ローリング・アップグレードを実行するには、次の高度な手順を実行する必要があります。

1. パッチ・ソフトウェアをプライマリ・アップグレード・ノードにコピーします。
2. アップグレード・ノードで Oracle RAC インスタンスを停止します。
3. アップグレード・ノードですべての Oracle プロセスを停止します。
4. OUI を起動し、アップグレード・ノードでアップグレード・プロセスを完了します。

Oracle RAC システムの実行中は、データベース・クライアントにかわってすべてのノードがトランザクションをアクティブに処理します。パッチ適用の手順 1 では、パッチが適用される最初のインスタンスを静止させます。手順 2 では、Oracle パッチ・ツール（Opatch）を使用して、静止中のインスタンスにパッチを適用します（たとえば、インスタンス 1 の Oracle ホームが更新されます）。手順 3 では、パッチが適用されたインスタンスが再度アクティブ化され、クラスタに再結合します。これで、Oracle RAC システムでは、1 つのインスタンスがクラスタ内の他のノードよりも高いメンテナンス・レベルで実行されるようになりました。

Oracle RAC システムは、このパッチが元の問題を解決し、他の問題を引き起こしていないことを確認するために、任意の期間にわたってこの複合モードで実行できます。次に、この手順がクラスタ内の残りのノードに対して繰り返されます。クラスタ内のすべてのノードにパッチが適用されると、ローリング・パッチ更新が完了し、すべてのノードで同じバージョンの Oracle Database ソフトウェアが実行されるようになります。さらに、Opatch にはパッチの適用をロールバックする機能もあります。この機能では、更新されたインスタンスで異常な動作が見られた場合、クラスタ全体を停止することなく、問題のパッチをアンインストール、すなわちロールバックできます。ロールバック手順はパッチ適用手順と同じですが、この場合、Opatch ユーティリティにより、以前に適用したパッチが削除されます。

ローリング・リリース・アップグレード

Oracle Database は、ローリング方式によるデータベース・ソフトウェア・アップグレードのインストールおよびパッチ・セットの適用をサポートします。ここでは、Data Guard SQL Apply およびロジカル・スタンバイ・データベースが使用され、データベースの停止時間はほとんどありません。

また、Oracle Clusterware も、ソフトウェア・アップグレードおよびパッチ・セットの適用についてローリング方式によるアップグレードをサポートします。さらに、Oracle Database 11*g* では、ASM もローリング・リリース・アップグレードをサポートします。

フィジカル・スタンバイ・データベースのユーザーは、ローリング・アップグレードの期間中に構成へのロジカル・スタンバイ・データベースの追加を選択することも、一時ロジカル・スタンバイ・データベースを使用して第 2 のスタンバイ・データベースの作成を回避することもできます。一時ロジカル・スタンバイ・データベースは、Oracle Database 11*g* における Data Guard の新機能で、ローリング・データベース・アップグレードを実行するために、フィジカル・スタンバイ・データベースをロジカル・スタンバイ・データベースに一時的に変換できます。このため、アップグレードが完了した後、ロジカル・スタンバイ・データベースをフィジカル・スタンバイ・データベースに戻すことができます。一時ロジカル・スタンバイ・データベースを使用すると、アップグレードを実行するために別のロジカル・スタンバイ・データベースを作成する必要がなくなります。

> **関連項目：** 一時ロジカル・スタンバイ・データベースを使用したローリング・アップグレードの段階的な手順の詳細は、『Oracle Data Guard 概要および管理』を参照してください。

Oracle では、ローリング・アップグレードおよびローリング・パッチ更新をサポートしているため、企業のデータベース管理者が管理タスクを行うためのメンテナンス時間枠がほとんど不要となり、24 時間 365 日の連続稼働が可能になります。

リソースの動的プロビジョニング

Oracle Database は、動的再構成に対するサポートの拡張を継続しているため、必要に応じて、サービスを中断せずにハードウェア内の変更に対応できます。Oracle Database は、次のようなハードウェア構成への変更を動的にサポートします。

- SMP サーバーのプロセッサの追加および削除

  Oracle Database がオペレーティング・システムを監視して、CPU 数の変更を検出します。CPU_COUNT 初期化パラメータをデフォルトに設定した場合、データベースのワークロードは、新規に追加されたプロセッサを動的に利用できます。

- 共有メモリー割当ての動的拡大と縮小、およびメモリーの自動オンライン・チューニング

  Oracle Database では、自動メモリー管理を使用して、SGA、PGA およびサブコンポーネント間でメモリーを自動的に移動して最適なパフォーマンスを確保できます。また、初期化パラメータ SGA_TARGET、PGA_AGREEGATE_TARGET および SERVER_MEMORY_TARGET を動的に変更して、アクティブなインスタンスのメモリーを追加および削除できます。

- Oracle RAC クラスタ内のノードの追加および削除
- データベース・アクティビティを中断しない、ASM ディスクの追加および削除
- データベース記憶域全体での I/O ロードの自動バランス再調整
- データファイルのオンライン移動

前述の機能により、システム変更の影響が解消され、エンタープライズ・グリッド・コンピューティングにおける基本要件である必要に応じた容量のプロビジョニングが真に可能になります。

> **関連項目：** Oracle のメモリー管理機能の詳細は、『Oracle Database 管理者ガイド』を参照してください。

Maximum Availability Architecture（MAA）のベスト・プラクティス

IT インフラストラクチャの実装を成功させる鍵は、運用面のベスト・プラクティスです。技術のみでは不十分です。Oracle の Maximum Availability Architecture（MAA）は、可用性の高いシステムを構築するための完全に統合された実証済のブループリントです。MAA のブループリントには、Oracle RAC および Oracle Clusterware、Data Guard、Recovery Manager、フラッシュバック・テクノロジ、Streams、Enterprise Manager を含む Oracle Database の主要機能を統合的に使用して高可用性を実現する方法が詳述されています。

Oracle テクノロジが関与する HA システムの設計は複雑であったり当て推量が含まれるものであってはならないという理念に基づいて概念化されているため、MAA の設計と構成の推奨事項は、広範囲にわたって検討およびテストされており、最適なシステムの可用性と信頼性を実現できます。また、MAA は、サーバー、記憶域、ネットワーキングおよびアプリケーション・サーバーを含む可用性の高いシステムの他の重要なコンポーネントの構成と統合にも対処しています。MAA に基づいたシステム・アーキテクチャを持つ企業は、高可用性とデータ保護に関するビジネス上の要件を迅速かつ効率的に満たすことができます。

MAA の詳細は、http://www.oracle.com/technology/deploy/availability/htdocs/maa.htm を参照してください。

18

大規模データベース（VLDB）

この章では、VLDB 戦略の重要な要素としてパーティション化に重点を置きながら、VLDB の概要を示します。この章の内容は、次のとおりです。

- パーティション化の基礎知識
- パーティション索引の概要
- パフォーマンスの改善のためのパーティション化

関連項目： パーティション化など VLDB の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。

注意： この機能を使用できるのは、Partitioning Option を購入した場合のみです。

パーティション化の基礎知識

パーティション化とは、大規模な表や索引を、**パーティション**というより小さくて管理しやすい部分に分割して、この種の表や索引をサポートするときの主な問題に対処します。パーティション表にアクセスする際、SQL 問合せと DML 文を変更する必要はありません。ただしパーティションを定義すると、DDL 文は表や索引全体ではなく、個々のパーティションへのアクセスやその操作ができるようになります。パーティション化を行うと、このようにしてラージ・データベース・オブジェクトの管理を簡素化できます。また、パーティション化は、アプリケーションに対して完全に透過的です。

表や索引の各パーティションは、列名、データ型、制約といった論理属性は同じものを持つ必要がありますが、PCTFREE、PCTUSED および表領域といった物理属性は別のものを持つことができます。

パーティション化は様々なタイプのアプリケーションで有効ですが、特に大量のデータを管理するアプリケーションで便利です。多くの場合、OLTP システムでは管理性と可用性が改善され、データ・ウェアハウス・システムではパフォーマンスと管理性が向上します。

パーティション化には次のような利点があります。

- パーティション化により、表全体でなくパーティション・レベルでのデータのロード、索引の作成や再作成、バックアップやリカバリといったデータ管理操作が可能になります。このため、これらの操作にかかる時間が大幅に短縮されます。
- パーティション化により、問合せのパフォーマンスが改善されます。多くの場合、問合せの結果は表全体ではなくパーティションのサブセットにアクセスして得ることができます。一部の問合せでは、このテクニック（**パーティション・プルーニング**という）によってパフォーマンスを大幅に改善できます。
- パーティション化により、定期的なメンテナンス操作による停止時間の影響を大幅に軽減できます。

  パーティションのメンテナンス操作におけるパーティションの独立性により、同じ表や索引にある異なるパーティションのメンテナンス操作を同時に行うことができます。また、メンテナンス操作の影響を受けないパーティションに対して、SELECT 操作と DML 操作を同時に実行できます。
- クリティカルな表および索引がパーティション化されて、メンテナンス操作の時間枠、リカバリ時間および障害による影響が低減されると、ミッション・クリティカルなデータベースの可用性が向上します。
- パーティション化は、アプリケーションに何の変更も加えずに実装できます。たとえば、表にアクセスする SELECT 文や DML 文を一切変更せずに、非パーティション表をパーティション表に変換できます。パーティション化を活用するためにアプリケーションのコードを書き換える必要はありません。

この項の内容は、次のとおりです。

パーティション・キー

パーティション表の各行は、単一パーティションに明示的に割り当てられます。パーティション・キーは、各行のパーティションを決定する 1 つ以上の列のセットです。Oracle Database では、パーティション・キーの使用により、挿入、更新および削除操作が適切なパーティションに自動的に送られます。パーティション・キーには次のような特長があります。

- 1 ～ 16 の順序付けられた列のリストからなります。
- LEVEL、ROWID または MLSLABEL の各擬似列、および ROWID 型の列を含むことはできません。
- NULL 値可能な列を含むことができます。

パーティション表

表は最大 1024K-1 のパーティションに分割できます。どの表もパーティション化できますが、LONG データ型または LONG RAW データ型の列を含む表は例外です。ただし、CLOB または BLOB の各データ型の列を含む表は使用できます。

注意： ディスク使用量とメモリー使用量（特にバッファ・キャッシュ）を減少させるために、表とパーティション表をデータベースに圧縮形式で格納できます。通常は、これにより読取り専用操作のパフォーマンスが向上します。また、問合せの実行も高速化されます。ただし、CPU オーバーヘッドの面で少しコストがかかります。

関連項目： 16-9 ページの「表の圧縮」

パーティション索引構成表

索引構成表に対してレンジ・パーティション化、リスト・パーティション化またはハッシュ・パーティション化を実行できます。パーティション索引構成表は、索引構成表の管理性、可用性およびパフォーマンスの改善に非常に役立ちます。さらに、索引構成表を使用するデータ・カートリッジも、格納したデータをパーティション化する機能を活用できます。

索引構成表のパーティション化には、次の規定があります。

- パーティション列は、主キー列のサブセットであることが必要です。
- 2 次索引はローカルにもグローバルにもパーティション化できます。
- OVERFLOW データ・セグメントは、常に表パーティションと同一レベルでパーティション化されます。

パーティション化方法

用意されているパーティション化方法は次のとおりです。

- レンジ・パーティション化では、各パーティションに設定したパーティション・キー値の範囲に基づいて、データが各パーティションにマップされます。
- 時間隔パーティション化は、表に挿入されたデータがレンジ・パーティションすべてを超える場合に、指定された間隔のパーティションを自動的に作成するようデータベースに指示するレンジ・パーティション化の拡張機能です。
- ハッシュ・パーティション化では、パーティションがほぼ同じサイズになるよう、パーティション間で均等に行を分配するハッシュ・アルゴリズムに基づいて、データが各パーティションにマップされます。
- リスト・パーティション化では、各パーティションの記述で、不連続な値のリストを指定して各パーティションへの行のマップ方法を明示的に制御できます。
- 参照パーティション化では、参照制約内で参照された表のパーティション化スキームに基づいて、表をパーティション化できます。
- コンポジット・パーティション化は、さらにデータをサブパーティションに分割するため、2 つのパーティション化方法を組み合せたものです。
  - コンポジット・レンジ - レンジ・パーティション化では、データはレンジ方式でパーティション化されてから、各パーティション内でレンジ方式でサブパーティション化されます。
  - コンポジット・レンジ - ハッシュ・パーティション化では、データはレンジ方式でパーティション化されてから、各パーティション内でハッシュ方式でサブパーティション化されます。
  - コンポジット・レンジ - リスト・パーティション化では、データはレンジ方式でパーティション化されてから、各パーティション内でリスト方式でサブパーティション化されます。
  - コンポジット・リスト - レンジ・パーティション化では、データはリスト方式でパーティション化されてから、各パーティション内でレンジ方式でサブパーティション化されます。
  - コンポジット・リスト - ハッシュ・パーティション化では、データはリスト方式でパーティション化されてから、各パーティション内でハッシュ方式でサブパーティション化されます。
  - コンポジット・リスト - リスト・パーティション化では、データはリスト方式でパーティション化されてから、各パーティション内でリスト方式でサブパーティション化されます。
- システム・パーティション化では、任意の表をアプリケーションが制御するパーティション化ができます。

パーティション索引の概要

パーティション表と同様、パーティション索引も管理性、可用性、パフォーマンスおよびスケーラビリティを改善します。パーティション索引は独立的にパーティション化する（グローバル索引）ことも、表のパーティション化方法に自動でリンクする（ローカル索引）こともできます。一般に、OLTP アプリケーションにはグローバル索引、データ・ウェアハウス・アプリケーションや DSS アプリケーションにはローカル索引を使用する必要があります。また、管理が容易であるため、できるだけローカル索引を使用するようにしてください。使用するパーティション索引の種類を決定する際には、次のガイドラインを順番に考慮する必要があります。

1. 表のパーティション化列が索引キーのサブセットの場合は、ローカル索引を使用します。この場合、他のガイドラインを考慮する必要はありません。それ以外の場合は、ガイドライン 2 に進みます。
2. 索引が一意索引の場合は、グローバル索引を使用します。この場合、他のガイドラインを考慮する必要はありません。それ以外の場合は、ガイドライン 3 に進みます。
3. 管理性を優先する場合は、ローカル索引を使用します。この場合、他のガイドラインを考慮する必要はありません。それ以外の場合は、ガイドライン 4 に進みます。
4. アプリケーションが OLTP で、ユーザーがすばやい応答時間を必要とする場合は、グローバル索引を使用します。アプリケーションが DSS で、ユーザーがスループットを重視している場合は、ローカル索引を使用します。

> **関連項目：** パーティション索引および使用するタイプを決定する方法の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』および『Oracle Database 管理者ガイド』を参照してください。

この項の内容は、次のとおりです。

ローカル・パーティション索引

ローカル・パーティション索引は、他のタイプのパーティション索引よりも管理が容易です。ローカル・パーティション索引でも可用性が拡張されており、DSS 環境では一般的となっています。その理由は、ローカル索引の各パーティションが必ず表の 1 つのパーティションに対応付けられている、同一レベル・パーティション化にあります。これにより、Oracle Database では、索引パーティションと表パーティションの同期を自動的に維持することが可能になり、表と索引の各ペアをそれぞれ独立させることができます。1 つのパーティションのデータを無効または使用不可能にする処理は、単一パーティションのみに影響します。

ローカル・パーティション索引は、表に対してパーティションまたはサブパーティションのメンテナンス操作を実行するときに、より高い可用性を提供します。ローカルの非同一キー索引と呼ばれる索引タイプは、履歴データベースにきわめて有効です。このタイプの索引では、索引列の左接頭辞に対してパーティション化が存在しません。

> **関連項目：** 同一キー索引の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。

パーティションは、明示的にローカル索引に追加することはできません。基礎となる表にパーティションを追加した場合のみ、新しいパーティションがローカル索引に追加されます。同様に、パーティションを明示的にローカル索引から削除することもできません。基礎となる表からパーティションを削除した場合のみ、ローカル索引のパーティションが削除されます。

ローカル索引は一意索引にできます。ただしローカル索引を一意索引にするには、表のパーティション化キーが索引のキー列の一部である必要があります。一意のローカル索引は、OLTP 環境では有効です。

グローバル・パーティション索引

Oracle Database には、レンジ・パーティションおよびハッシュ・パーティションという 2 種類のグローバル・パーティション索引があります。

この項の内容は、次のとおりです。

- グローバル・レンジ・パーティション索引
- グローバル・ハッシュ・パーティション索引
- グローバル・パーティション索引のメンテナンス

グローバル・レンジ・パーティション索引

グローバル・レンジ・パーティション索引は、パーティション化の程度とパーティション化キーが表のパーティション化方法から独立しているため、柔軟といえます。この索引は通常は OLTP 環境で使用され、あらゆる個別レコードへの効率的なアクセスを提供します。

グローバル索引の最も上位のパーティションのパーティション・バウンドには、すべての値に MAXVALUE を指定する必要があります。これにより、基礎になる表のすべての行を索引に含めることができます。グローバル同一キー索引は、一意の索引にも、一意でない索引にもできます。

最上位のパーティションには常に MAXVALUE のパーティション・バウンドがあるため、グローバル索引にはパーティションを追加できません。新規に最上位パーティションを追加する場合は、ALTER INDEX SPLIT PARTITION 文を使用します。グローバル索引のパーティションが空の場合、ALTER INDEX DROP PARTITION 文を発行すると明示的に削除できます。グローバル索引のパーティションにデータが含まれている場合は、パーティションを削除すると、次に上位のパーティションに UNUSABLE マークが設定されます。グローバル索引の最上位パーティションは削除できません。

グローバル・ハッシュ・パーティション索引

グローバル・ハッシュ・パーティション索引では、索引が直線的に大きくなっていく場合に競合が分散され、パフォーマンスが改善されます。つまり、索引挿入のほとんどは索引の右端でのみ発生します。

グローバル・パーティション索引のメンテナンス

デフォルトでは、ヒープ構成表のパーティションに次の操作を行うと、グローバル索引すべてに UNUSABLE マークが設定されます。

```
ADD (HASH)
COALESCE (HASH)
DROP
EXCHANGE
MERGE
MOVE
SPLIT
TRUNCATE
```

操作のための SQL 文に UPDATE INDEXES 句を追加すると、これらの索引をメンテナンスできます。グローバル索引をメンテナンスすることには、次の 2 つの利点があります。

- 操作の間中、索引が使用可能かつオンラインな状態にあります。そのため、他のアプリケーションがこの操作の影響を受けません。
- 操作の後、索引を再作成する必要がありません。

例:

```
ALTER TABLE DROP PARTITION P1 UPDATE INDEXES;
```

注意: この機能はヒープ構成表のみにサポートされています。

関連項目: UPDATE INDEXES 句の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

グローバル非パーティション索引

グローバル非パーティション索引は、非パーティション索引と同様に動作します。この索引は通常は OLTP 環境で使用され、あらゆる個別レコードへの効率的なアクセスを提供します。

パーティション表に索引を作成する場合のその他の情報

パーティション表に対してビットマップ索引を作成できますが、ビットマップ索引はパーティション表に対して必ずローカルであるという制限があります。グローバル索引にすることはできません。

グローバル索引は一意索引にできます。ローカル索引は、パーティション化キーが索引キーの一部である場合、一意索引にしかできません。

OLTP アプリケーションでのパーティション索引の使用

ここでは OLTP アプリケーションにいくつかのガイドラインを示します。

- グローバル索引と一意のローカル索引を使用すると、索引パーティション・プローブの数が最小になるため、非一意のローカル索引を使用するよりもパフォーマンスが向上します。
- ローカル索引は、表に対してパーティションまたはサブパーティションのメンテナンス操作を実行するときに、より高い可用性を提供します。
- ハッシュ・パーティション化されたグローバル索引は、索引が直線的に大きくなっていく場合に競合を分散させることでパフォーマンスを改善します。つまり、索引挿入のほとんどは索引の右端でのみ発生します。

データ・ウェアハウス・アプリケーションと DSS アプリケーションでのパーティション索引の使用

ここではデータ・ウェアハウス・アプリケーションと DSS アプリケーションにいくつかのガイドラインを示します。

- データのロード時およびパーティションのメンテナンス操作時にはローカル索引のほうが管理が容易なため、ローカル索引の使用をお薦めします。
- ローカル索引を使用すると、索引キーに基づくレンジ問合せにより、多くの索引パーティションをパラレルでスキャンできるため、パフォーマンスを改善できます。

コンポジット・パーティションでのパーティション索引

ここではコンポジット・パーティションでパーティション索引を使用する際の考慮事項をあげます。

- サブパーティション索引は常にローカルで、デフォルトでは表のサブパーティションとともに格納されます。
- 表領域は、索引レベルでも索引のサブパーティションレベルでも指定できます。

パフォーマンスの改善のためのパーティション化

パーティション化を行うと、パフォーマンスや管理性を改善できます。その理由でパーティション化を使用する際の考慮事項を次に示します。

- パーティション・プルーニング
- パーティション・ワイズ結合

パーティション・プルーニング

Oracle Database は、パーティションとサブパーティションを明確に認識します。次に、アクセスを必要とするパーティションまたはサブパーティションをマークし、不要なパーティションやサブパーティションがこれらの SQL 文によるアクセスから除外（プルーニング）されるようにするために、SQL 文を最適化します。言い換えると、パーティション・プルーニングとは、問合せの際に不要な索引およびデータのパーティションやサブパーティションをスキップすることです。

SQL 文ごとに、指定された選択基準に応じて不必要なパーティションやサブパーティションが除外されます。たとえば、3 月の売上データのみに関する問合せであれば、残りの 11 か月に関するデータを取り出す必要はありません。このようなインテリジェント・プルーニング機能によって、データの量を大幅に低減できるため、問合せのパフォーマンスが実質的に向上します。

SQL アクセス・アドバイザは、指定されたワークロードのスキーマ設計を自動的に分析し、ワークロードに応じた索引、ファンクション索引、パーティションおよびマテリアライズド・ビューの作成、保持または削除を提案できます。

関連項目： SQL アクセス・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

オプティマイザは、アクセスされるパーティションまたはサブパーティションのすべての行がプルーニングによって使用される選択基準を満たすかどうかを判断する場合に、パフォーマンスを向上させるために、評価のときにそれらの基準を述語リスト（WHERE 句）から削除します。ただし、SQL 文によってパーティション化される列に TO_DATE 以外の関数が適用されると、オプティマイザはパーティションをプルーニングできません。同様に、ファンクション索引でないかぎり、SQL 文によって索引付きの列に関数が適用されると、オプティマイザは索引を使用できません。

基礎となる表のパーティションを排除できない場合にも、パーティションのプルーニングによって索引パーティションを排除できますが、これは索引と表が別々の列にパーティション化されている場合にかぎります。大規模な表に対する操作のパフォーマンスは、SQL 文がアクセスまたは変更する必要のあるデータ量を削減するパーティション索引を作成すると改善できます。

等価、レンジ、LIKE および IN-list の述語は、レンジ・パーティション化またはリスト・パーティション化によるパーティション・プルーニングの対象とされ、等価かつ IN-list の述語は、ハッシュ・パーティション化によるパーティション・プルーニングの対象とされます。

パーティション・プルーニングの例

cust_orders と呼ばれるパーティション表があります。cust_orders のパーティション・キーは order_date です。cust_orders には 1 月～ 6 月までの 6 か月分のデータがあり、各月のデータに対して 1 つのパーティションが設定されているとします。ここで次のような問合せを実行するとします。

```
SELECT SUM(value)
FROM cust_orders
WHERE order_date BETWEEN '28-MAR-98' AND '23-APR-98';
```

パーティション・プルーニングは次のように行われます。

- まず 1 月、2 月、5 月および 6 月のデータ・パーティションのパーティション絞込みが行われます。次に、下のいずれかが実行されます。
  - 索引の選択性が高い場合、3 月および 4 月のデータ・パーティションの索引スキャン

    または
  - 索引の選択性が低い場合、3 月および 4 月のデータ・パーティションの全体スキャン

パーティション・ワイズ結合

パーティション・ワイズ結合とは、結合列に沿ってパーティション化された 2 つの表を結合するための最適化のことです。パーティション・ワイズ結合では、結合の操作が小さな単位に分割され、それぞれが順番にまたはパラレルで実行されます。別の見方をすれば、パーティション・ワイズ結合は、パラレル結合の実行時にデータ配分を考慮することでパラレルのスレーブ間で交換されるデータの量を最小化するものです。

関連項目： パーティション化方法およびパーティション・ワイズ結合の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。

19

コンテンツ管理

この章では、Oracle のコンテンツ管理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。

コンテンツ管理の概要

Oracle Database には、リレーショナル・データ、オブジェクト・リレーショナル・データ、XML、テキスト、オーディオ、ビデオ、イメージおよび空間など、豊富な各種インターネット・コンテンツをすべて処理するためのデータ型が組み込まれています。これらのデータ型は、データベースではシステム固有の型として表示されます。これらは、いずれも SQL を使用して問い合せることができます。これらのデータ型のいずれかまたはすべてに属するデータを、1 つの SQL 文に含めることができます。

アプリケーションが進化し、より多様なセマンティクスを含むようになるにつれて、次の種類のデータを取り扱う必要性が生じています。

- 単純な構造化データ
- 複雑な構造化データ
- 半構造化データ
- 非構造化データ

従来、リレーショナル・モデルは単純な構造化データ、つまり単純な表に格納できるデータの処理において大成功を納めてきました。Oracle では、アプリケーションが複雑な構造化データ（コレクション、参照、ユーザー定義型など）を処理できるように、オブジェクト・リレーショナル機能が追加されています。Oracle Streams Advanced Queuing などのキューイング・テクノロジは、メッセージおよびその他の半構造化データを処理します。この章では、非構造化データをサポートする Oracle のテクノロジについて説明します。

構造化されていないデータは、標準構成要素に分解できません。従業員に関するデータは、氏名（文字列）、ID（番号など）、給与などに構造化できます。しかし、写真を与えられた場合は、データが実際には 0 と 1 からなる長いストリームで構成されていることがわかります。これらの 0 と 1 はピクセルのオンとオフを切り替えるために使用されているため、画面には写真を表示できますが、データベースへの格納用にそれよりも細かい構造へと分解することはできません。

テキスト、グラフィック・イメージ、ビデオ・クリップ、フル・モーション・ビデオおよびサウンド・ウェーブなどの非構造化データは大きくなりがちであり、通常の従業員レコードは数百バイトであるのに対して、マルチメディア・データはごく少量でも数千倍のサイズになることがあります。一部のマルチメディア・データはオペレーティング・システム・ファイルに常駐することがあり、データベースからアクセスすることが望ましいとされています。

Oracle Database における XML の概要

Extensible Markup Language（XML）はタグベースのマークアップ言語であり、開発者は独自のタグを作成して、アプリケーション間やシステム間でやりとりされるデータを記述できます。XML は企業間の共通の情報交換言語として広く採用されています。XML は判読可能なプレーン・テキストです。プレーン・テキストであるため、XML 文書や XML ベースのメッセージは、HTTP や FTP などの一般的なプロトコルを使用して簡単に送信できます。

Oracle XML DB では、XML はデータベース内のシステム固有のデータ型として扱われます。Oracle XML DB は別個のサーバーではありません。XML データ・モデルには、構造化されていないコンテンツと構造化データの両方が含まれています。アプリケーションでは、標準的な SQL と XML の演算子を使用して、SQL 問合せから複雑な XML 文書を作成したり、XML 文書を格納できます。

Oracle XML DB には、コンテンツ指向のアクセスとデータ指向のアクセスの両面で機能が用意されています。Oracle XML DB では、XML を文書（ニュース、ニュース項目など）とみなす開発者に、標準プロトコルと SQL からアクセスできる XML リポジトリが提供されます。

その他のユーザーにとっては、XML の構造化データ部分（請求書、住所など）の方が重要です。Oracle XML DB では、この種のユーザーにシステム固有の XML 型、XML スキーマ、XPath、XSLT、DOM などのサポートが提供されます。通常は、データ指向のアクセスの方が問合せ集中型です。

Oracle XML Developer's Kit（XDK）には、XML 文書（ファイル・システム上にあるか、データベースに格納されているかを問わない）の読取り、操作、変換および表示のための基本ビルディング・ブロックが用意されています。これらは Java、C および C++ で使用できます。多数のシェアウェアや試用の XML コンポーネントとは異なり、Oracle XDK は本番環境で全面的にサポートされ、業務目的で再配布するためのライセンスが付属しています。Oracle XDK のコンポーネントは、次のとおりです。

- XML パーサー : Java、C、および C++ をサポートしています。このコンポーネントは、業界標準の DOM および SAX インタフェースを使用して XML を作成、解析します。
- XSLT プロセッサ : XML を HTML などの他のテキストベースの形式に変換またはレンダリングします。
- XML Schema Processor: Java、C および C++ をサポートしており、XML の単純データ型と複合データ型を使用できます。
- XML Class Generator: XSL スキーマから Java および C++ のクラスを自動的に生成し、Web フォームまたはアプリケーションから XML データを送信します。
- XML Java Beans: XML 文書とデータを表示し、Java コンポーネントを使用して変換します。
- XML SQL Utility: Java をサポートしており、SQL 問合せから XML 文書、DTD およびスキーマを生成します。
- XSQL Servlet: サーバー上の XML、SQL および XSLT を組み合せて動的な Web コンテンツを配信します。

関連項目：

- 『Oracle XML DB 開発者ガイド』
- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML Developer's Kit API Reference』
- 『Oracle Database XML Java API Reference』

LOB の概要

ラージ・オブジェクト（LOB）データ型 `BLOB`、`CLOB`、`NCLOB` および `BFILE` を使用すると、非構造化データ（テキスト、グラフィック・イメージ、ビデオ・クリップおよびサウンド・ウェーブなど）の大きなブロックを、バイナリ形式または文字形式で格納し、操作できます。このデータ型を使用すると、個々のデータに対する効率的なランダム・アクセスが可能です。

インターネット・アプリケーションや豊富なコンテンツを持つアプリケーションの発展に伴い、データベースで次の要件を満たすデータ型をサポートすることが必要になってきています。

- 圧縮、暗号化または重複除外によって非構造化データを格納できること
- 最大 128TB の大量のデータにあわせて最適化されていること
- データベースの内部または外部にある読取り専用オペレーティング・システム・ファイル内の大きな非構造化データへの一様なアクセス方法を提供すること

関連項目： 26-12 ページの「LOB データ型の概要」

`STORE AS SECUREFILE` オプション（リリース 11.1 で導入）のある LOB では、`CREATE TABLE` および `ALTER TABLE` 文で SQL パラメータ `DEDUPLICATE` を指定できます。これによって、LOB の列内の 2 つ以上の行で同一の LOB データがすべて同じデータ・ブロックを共有する（したがってディスク容量を節約する）ように指定できます。この機能は、`KEEP_DUPLICATES` によってオフになります。次のオプションも `SECUREFILE` とともに使用できます。

パラメータ `COMPRESS` は、LOB 圧縮をオンにします。LOB 圧縮は、`NOCOMPRESS` によってオフになります。

パラメータ ENCRYPT は、LOB 暗号化をオンにし、オプションで暗号化アルゴリズムを選択します。LOB 暗号化は、NOENCRYPT によってオフになります。

デフォルトはプレリリース 11.1 LOB パラダイムです。これは現在、オプション STORE AS BASICFILE によっても明示的に設定するようになっています。

次の SQL および PL/SQL 文、OCI 関数は、SECUREFILE 機能で使用します。

関連項目：

- 関連 SQL 関数の詳細と PL/SQL パッケージの相互参照については、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。
- 関数 OCILobGetDuplicateRegions()、OCILobSetOptions() および OCILobGetOptions() については、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

Oracle Text の概要

Oracle Text は、あらゆるドキュメントやテキスト形式のコンテンツに索引を付け、インターネット・コンテンツ管理アプリケーション、E-Business カタログ、ニュース・サービス、人材募集などに高速で正確な情報検索機能をもたらします。索引付けの対象は、ファイル・システムやデータベースに格納されているコンテンツでも、Web 上にあるコンテンツでもかまいません。

Oracle Text を使用すると、テキスト検索と通常のデータベース検索を 1 つの SQL 文にまとめることができます。ドキュメントは、テキスト形式のコンテンツ、メタデータまたは属性に基づいて検索できます。Oracle Text の SQL API により、単純で直感的な方法でテキスト索引を作成、メンテナンスしてテキスト検索を実行できます。

Oracle Text は Oracle Database と完全に統合されるため、本質的に高速でスケーラブルになります。Oracle Text の索引はデータベースにあり、Oracle Text の問合せは Oracle Database プロセス内で実行されます。Oracle Database オプティマイザは、どのような問合せについても最善の実行計画を選択でき、テキストと構造化された基準を使用する非定型問合せに最高のパフォーマンスを提供します。さらに次の利点を備えています。

- Oracle Text は、複数言語による問合せおよび索引付けをサポートします。
- XML 文書の検索に使用するセクションに索引を付けて定義できます。セクション検索により、問合せ対象を文書内のテキスト・ブロックに絞り込むことができます。Oracle Text では、XML セクションが自動的に作成されます。
- Oracle Text の索引は多数のテキスト列にまたがることができるため、複数の列にまたがるテキスト問合せに最高のパフォーマンスを提供します。
- Oracle Text は、ヒット数など、テキスト検索に共通する操作の面でパフォーマンスが強化されています。
- Oracle Text では、レプリケーションなどのスケーラビリティ機能を利用できます。
- Oracle Text はローカル・パーティション索引をサポートします。

この項の内容は、次のとおりです。

Oracle Text の索引タイプ

Oracle Text には、テキスト検索のニーズすべてに対処する 3 つの索引タイプがあります。

- 標準索引タイプは、従来の文書および Web ページの全テキスト検索に使用します。コンテキスト索引タイプは、必要なコンテンツを検索するための様々なテキスト検索機能を提供し、誤った結果のページを戻すことはありません。
- カタログ索引タイプは、E-Business カタログ専用に設計されています。このカタログ索引を使用すると、Web の処理速度で柔軟に検索およびソートできます。
- 分類索引タイプは、分類の構築やアプリケーションのルーティングに使用します。この索引は問合せの表に対して作成され、問合せにより分類基準やルーティング基準を定義します。

Oracle Text には、サブストリング索引と同一キー索引も用意されています。サブストリング索引付けにより、後方一致または中間一致によるワイルドカード問合せのパフォーマンスが改善されます。同一キー索引付けにより、前方一致によるワイルドカード問合せのパフォーマンスが改善されます。

Oracle Text のドキュメント・サービス

Oracle Text には、格納方法を問わずテキストを表示するための、多数のユーティリティが用意されています。

- Oracle Text は、Inso フィルタ処理テクノロジを介して 150 以上のドキュメント形式をサポートしています。これには、XML、PDF および Microsoft Office など、一般的なドキュメント形式がすべて含まれます。また、独自のカスタム・フィルタを作成することもできます。
- PDF、Microsoft Office などでフォーマットされたドキュメントを含め、あらゆるテキストの HTML バージョンを表示できます。
- テキストの HTML バージョンを表示すると検索用語が選択され、テキスト内の次または前の検索用語にナビゲートできます。
- Oracle Text によりテキスト内の各検索用語のオフセットや長さなどのマークアップ情報が提供され、サード・パーティのビューアなどで使用できます。

Oracle Text の問合せパッケージ

CTX_QUERY PL/SQL パッケージを使用すると、問合せのフィードバックを生成し、ヒット数をカウントし、ストアド・クエリー式を作成できます。

> **関連項目：** このパッケージの詳細は、『Oracle Text リファレンス』を参照してください。

Oracle Text の拡張機能

Oracle Text を使用すると、テキスト、メタデータまたは属性に基づいてドキュメントを検索、分類およびクラスタ化できます。

ドキュメント分類では、ドキュメントのコンテンツに基づいて処理が実行されます。処理では、後でドキュメントを検索したりユーザーに送信できるように、カテゴリ ID を割り当てることができます。その結果、カテゴリに分類されたドキュメントの集合、つまりストリームが生成されます。たとえば、ニュース項目の受信ストリームがあるとします。そこで金融というカテゴリを表すルールを定義できます。このルールは、実際には金融という主題に関するドキュメントを選択する 1 つ以上の問合せです。このルールは「株式または債券または収益」という形式にすることができます。このカテゴリのルールを満たすドキュメントが着信すると、アプリケーションはドキュメントに「金融」というタグを付けたり 1 人以上のユーザーに電子メールを送信するなどの処理を実行します。

クラスタ化とは、パターンをグループへと監視なしで分割することです。ユーザーはインタフェースを使用して、適切なクラスタ化アルゴリズムを選択できます。各クラスタには、コレクションに属するドキュメントのサブセットが含まれます。クラスタ内のドキュメントは、クラスタ外部のドキュメントではなく内部の他のドキュメントとの類似性が高いものとみなされます。クラスタを使用すると、コレクション内の類似ドキュメントを表示するなどの機能を構築できます。

関連項目：『Oracle Text アプリケーション開発者ガイド』

Oracle Ultra Search の概要

Oracle Ultra Search は Oracle Database および Oracle Text テクノロジを使用しており、複数のリポジトリにおける均一の検索および特定機能を提供します。この場合のリポジトリは、Oracle Database、ODBC 準拠の他のデータベース、IMAP メール・サーバー、Web サーバーで提供される HTML ドキュメント、ディスク上のファイルなどです。

Oracle Ultra Search はクローラを使用してドキュメントに索引付けします。ドキュメントはそのリポジトリに残り、得られた情報を使用してファイアウォール内で指定の Oracle データベースに残る索引が作成されます。また、Oracle Ultra Search には、コンテンツ管理ソリューションを構築するための API も用意されています。

Oracle Ultra Search の機能は次のとおりです。

- データベース内でテキスト検索に使用する完全なテキスト問合せ言語
- Oracle Database および SQL 問合せ言語との完全な統合
- 概念検索やテーマ分析などの拡張機能
- すべての一般ファイル形式（150 以上）の索引付け
- 中国語、日本語および韓国語（CJK）と Unicode のサポートを含む完全なグローバリゼーション

関連項目：『Oracle Ultra Search 管理者ガイド』

Oracle Multimedia の概要

Oracle Multimedia（旧称 Oracle *inter*Media）は、Oracle Database において、イメージ、Digital Imaging and Communications in Medicine（DICOM）、オーディオおよびビデオ・データを他の企業情報と統合された方法で格納、管理および取得できるようにする機能です。Oracle Multimedia は、Oracle Database の信頼性、可用性およびデータ管理機能を、従来型、医療、インターネット、電子商取引および多彩なメディアに対応するアプリケーション内のメディア・コンテンツおよび医用画像コンテンツにまで拡張します。

Oracle Multimedia では、次の機能によりメディア・コンテンツを管理します。

- データベースにメディア・データを格納して取得することによる、メディア・データと関連付けられたビジネス・データとの同期化
- 一般的なイメージ、オーディオおよびビデオ形式のサポート
- 形式およびアプリケーション・メタデータの XML 文書への抽出
- Oracle Multimedia サービスに対する完全なオブジェクト・インタフェースおよびリレーショナル・インタフェース
- 従来型および Web インタフェースによるアクセス
- 関連するリレーショナル・データおよび抽出されたメタデータを使用した問合せ
- サムネイル生成などのイメージ処理
- RealNetworks および Windows Media Streaming Servers による配信

Oracle Multimedia では、次の機能により DICOM コンテンツを管理します。

- データベースに医用画像データを格納して取得することによる、DICOM データと関連付けられたビジネス・データとの同期化
- Oracle Multimedia DICOM サービスに対する完全なオブジェクト・インタフェースおよびリレーショナル・インタフェース
- ユーザー指定が可能な XML 文書への DICOM メタデータの抽出
- 関連するリレーショナル・データおよび抽出されたメタデータを使用した問合せ
- サムネイル生成などのイメージ処理
- 新しい DICOM オブジェクトの作成
- ユーザー指定の一連の準拠ルールに基づく準拠検証
- 匿名にする属性のセットおよびそれらの属性を匿名にする方法を指定するユーザー定義のルールに基づく DICOM オブジェクトの匿名化
- Oracle Database の新しいリリースをインストールせずに実行時の動作（サポートされている DICOM 標準のバージョンなど）を更新する機能

関連項目：

- 『Oracle Multimedia ユーザーズ・ガイド』
- 『Oracle Multimedia リファレンス』
- 『Oracle Multimedia DICOM 開発者ガイド』
- 『Oracle Multimedia Java API Reference』
- 『Oracle Multimedia Servlets and JSP Java API Reference』
- 『Oracle Multimedia DICOM Java API Reference』

Oracle Spatial の概要

Oracle Spatial は、ロケーション対応アプリケーションや地理情報システム（GIS）アプリケーションのユーザーにとって、空間データの管理作業がより容易かつ自然なものになるよう設計されています。空間データが Oracle Database に格納されている場合は、簡単に操作し、取り出して、データベースに格納されている他のすべてのデータに関連付けることができます。

空間データの一般例は、ロード・マップに見いだすことができます。ロード・マップは、点、線および多角形を含む 2 次元オブジェクトであり、それぞれの要素で都市、道路および州や郡などの行政区分を表すことができます。ロード・マップは地理情報を視覚化したものです。地球上に存在する都市、道路および行政区分の位置は、レンダリングされるオブジェクトの相対位置や相対距離を保ったままで 2 次元の画面上や紙上に投影されます。

このようにレンダリングされるオブジェクトの地球上での位置（緯度と経度など）を示すデータが、空間データです。マップがレンダリングされるときには、この空間データを使用してオブジェクトの位置が 2 次元の紙上に投影されます。通常、この相対的空間データの格納、取得およびレンダリングには GIS が使用されます。

Oracle Spatial を使用して格納できる空間データのタイプには、GIS データの他にも、コンピュータ支援設計（CAD）システムやコンピュータ支援製造（CAM）システムからのデータがあります。CAD/CAM システムでは、地理的スケールでオブジェクトを操作するかわりに、自動車エンジンやプリント基板などの小さいスケールを処理します。

このようなシステム間の違いはデータのサイズおよび精度であり、データの複雑度ではありません。どのシステムでも、扱うデータ・ポイントの数は同じです。地理的スケールでは、橋の位置が 10 分の数インチ変わっても道路建設業者にとってほとんど問題になりませんが、エンジンのピストンの直径が 10 分の数インチでも違えばエンジンは動きません。

また、データの複雑度は、表示される領域の絶対スケールとは無関係です。たとえば、プリント基板の場合、その表面には数千のオブジェクトが印刷され、そのわずかな領域に、道路建設業者の図面に記載された詳細よりも複雑になりうる情報が含まれます。

この種のアプリケーションではすべて、非空間属性と空間属性の両方を持つ機能のなんらかのコレクションが格納、取得、更新または問合せされます。非空間属性の例は、name、soil_type、landuse_classification および part_number などです。空間属性は、座標のジオメトリ、つまり、機能の形状をベクターベースで表現したものです。

Oracle Spatial には SQL スキーマおよび関数が用意されており、Oracle Database 内にある空間機能の集まりを容易に格納、取得、更新および問合せできます。Oracle Spatial の構成は、次のとおりです。

- サポートされている地理的データ型の格納、構文およびセマンティクスを記述するスキーマ（MDSYS）
- 空間索引付けメカニズム
- 対象領域の問合せ、空間結合問合せおよび他の空間分析操作を実行するための演算子、ファンクションおよびプロシージャ
- ユーティリティおよびチューニング操作用のファンクションおよびプロシージャ
- トポロジのノード、辺および面に関するデータを操作するためのトポロジ・データ・モデル
- モデル化された機能またはオブジェクトをネットワーク内のノードおよびリンクとして表すためのネットワーク・データ・モデル
- GeoRaster データ（ラスター・イメージおよびグリッド・データとそれに関連するメタデータ）を格納、索引付け、問合せ、分析および配信できる機能である GeoRaster

> **関連項目：**『Oracle Spatial GeoRaster 開発者ガイド』および『Oracle Spatial トポロジおよびネットワーク・データ・モデル開発者ガイド』

20

データベース・セキュリティ

この章では、Oracle Database のデータベース・セキュリティの概要について説明します。

この章の内容は、次のとおりです。

- データベース・セキュリティの概要
- 透過的データ暗号化の概要
- 認証方式の概要
- 認可の概要
- 表、ビュー、シノニムまたは行に対するアクセス制限の概要
- セキュリティ・ポリシーの概要
- データベース監査の概要

関連項目： この章の内容の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

データベース・セキュリティの概要

データベース・セキュリティには、データベースとデータベースに格納されたオブジェクトに対するユーザー・アクションを許可または禁止する機能が関係しています。Oracle Database 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、様々なデータベース・リソースの使用を制限します。

Oracle Database は、包括的な任意アクセス制御を提供します。**任意アクセス制御**では、権限を使用して、特定のオブジェクトに対するすべてのユーザー・アクセスを制御します。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ます。

この項の内容は、次のとおりです。

データベース・ユーザーとスキーマ

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

セキュリティ・ドメイン

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

- ユーザーが行える操作（権限とロール）
- ユーザーに対する表領域の割当て制限（利用可能なディスク領域）
- ユーザーのためのシステム・リソース制限（CPU 処理時間など）

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

権限

権限は、特定のタイプの SQL 文を実行するための権利です。たとえば、次のことをする権利が権限です。

- データベースへの接続（セッションの作成）
- スキーマ内への表の作成
- 他のユーザーの表からの行の選択
- 他のユーザーのストアド・プロシージャの実行

関連項目：

- 権限の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 20-13 ページの「権限の概要」

ロール

Oracle Database では、ロールを使用して、簡単かつ制御された権限管理を提供します。**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。

> **関連項目：** ロール・プロパティの詳細は、20-14 ページの「ロールの概要」を参照してください。

記憶域の設定と割当て制限

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

この項の内容は、次のとおりです。

デフォルト表領域

各ユーザーには、**デフォルト表領域**が対応付けられます。ユーザーがスキーマ・オブジェクトを作成する権限と、指定されたデフォルト表領域に対する割当て制限を持っている場合、表、索引またはクラスタの作成時にそのスキーマ・オブジェクトを物理的に格納する表領域を指定しないと、そのユーザーのデフォルト表領域が使用されます。スキーマ・オブジェクトの位置が指定されていない場合、デフォルトの表領域は、Oracle Database に領域使用を指定する情報を提供します。

一時表領域

各ユーザーは、**一時表領域**を持っています。ユーザーが一時セグメントの作成を必要とする SQL 文（索引の作成など）を実行するときには、そのユーザーの一時表領域が使用されます。すべてのユーザーの一時セグメントを別の表領域に割り当てることによって、一時表領域は一時セグメントと他のタイプのセグメントとの間の I/O 競合を低減します。

表領域割当て制限

Oracle Database では、スキーマ内のオブジェクトに対して使用できるディスク領域の総量を制限できます。**割当て制限**（領域制限）は、ユーザーが使用可能な表領域ごとに設定できます。これによって、特定のスキーマのオブジェクトが使用するディスク領域の容量を選択的に制御できます。

プロファイルとリソース制限

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

- ユーザーが確立できる同時セッションの数
- ユーザーのセッションおよびSQL 文による Oracle Database への 1 回のコールに使用可能な CPU 処理時間
- ユーザーのセッションおよびSQL 文による Oracle Database への 1 回のコールに使用可能な論理 I/O の総量
- ユーザーのセッションで使用可能なアイドル時間の総量
- ユーザーのセッションで使用可能な接続時間の総量
- パスワード制限
  - 複数のログインが失敗したときにアカウントをロックする機能
  - パスワードの期限切れと猶予期間
  - パスワードの再利用と複雑さに関する制限

関連項目：

- プロファイルとリソース制限の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 20-12 ページの「プロファイル」

透過的データ暗号化の概要

Oracle Database は、認証、認可および監査形式でセキュリティを提供します。認証では、正当なユーザーのみがシステムにアクセスできることが保証されます。認可では、ユーザーがアクセスを許可されているリソースにのみアクセスできることが保証されます。監査では、ユーザーが保護されたリソースにアクセスした場合のアカウンタビリティが保証されます。これらのセキュリティ・メカニズムによりデータベース内のデータが効率的に保護されますが、ただし、データが格納されているオペレーティング・システム・ファイルへのアクセスは防止されません。

透過的データ暗号化を使用すると、オペレーティング・システム・ファイルに格納されているデータベース列内の機密データを暗号化できます。さらに、データベース外部のセキュリティ・モジュールにおける、暗号化キーのセキュアな格納および管理が提供されます。

外部のセキュリティ・モジュールを使用すると、通常のプログラム機能が、暗号化などのセキュリティ関連機能から分離されます。したがって、DBA とセキュリティ管理者との間の管理業務を分割できます。この方法では、管理者にデータへの包括的なアクセス権限が付与されないため、セキュリティが強化されます。外部セキュリティ・モジュールでは、暗号化キーの生成、暗号化と復号化の実行およびデータベース外部へのキーの安全な格納が実行されます。

透過的データ暗号化は、機密扱いのキーによりデータを暗号化して認可を規定する、キーベースのアクセス制御システムです。指定された表内の暗号化された列の数に関係なく、暗号化された列が含まれるデータベース表ごとにキーを 1 つのみ使用できます。次に、各表の列の暗号化キーが、データベース・サーバーのマスター・キーにより暗号化されます。データベースにはキーは格納されません。そのかわり、外部セキュリティ・モジュールの一部である Oracle Wallet に格納されます。

データベース列を暗号化する前に、マスター・キーを生成または設定する必要があります。このマスター・キーは、データベース列について `ENCRYPT` 句を含む SQL コマンドを発行したときに自動的に生成された、列の暗号化キーの暗号化に使用されます。

関連項目： 透過的データ暗号化の使用方法の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

表領域の暗号化

表領域の暗号化は、このリリースで導入された新機能です。表領域の暗号化を使用すると、表領域全体を暗号化できます。これにより、表領域に格納されているすべてのデータが保護されます。認可されたユーザーが表領域のデータにアクセスすると、データはそのユーザーに対して透過的に復号化されます。

表領域の暗号化により、暗号化する列を決定するためにアプリケーションを細かく分析する必要がなくなります。表領域の暗号化を使用して、機密データが含まれている可能性のある表全体を暗号化できます。

透過的暗号化 / 復号化は、データへの論理アクセスごとではなく、ディスク I/O の際に行われます。このため、パフォーマンスが向上します。

> **関連項目：** 表領域の暗号化の使用方法の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

認証方式の概要

認証とは、データ、リソースまたはアプリケーションの使用を希望するエンティティ（ユーザーやデバイスなど）の識別を検証することです。識別を検証することで、その後の対話に関する信頼関係が確立されます。また、認証により、アクセスとアクションを特定の識別にリンクでき、アカウンタビリティが有効化されます。認証後は、認可プロセスでそのエンティティが実行できるアクセスと処理のレベルを許可または制限できます。

通常は、簡素化のためにデータベース・ユーザー全員に同じ認証方式が使用されますが、Oracle Database では 1 つのデータベース・インスタンスで一部またはすべての方式を使用できます。Oracle Database では、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。さらに、Oracle Database では、ネットワーク認証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・ユーザーの識別を検証し、データベース・ユーザー名の無許可使用を防止するために、ここで説明する認証方式を任意に組み合せて使用できます。

- オペレーティング・システムによる認証
- ネットワークによる認証
- Oracle Database による認証
- 複数層の認証と認可
- Secure Socket Layer プロトコルによる認証
- データベース管理者の認証

> **関連項目：** 認証方式の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

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・システムがユーザー認証のために管理している情報を、Oracle Database で使用することができます。この方式には、次の利点があります。

- オペレーティング・システムから認証を受けたユーザーは、より簡単に Oracle Database に接続できます。ユーザー名やパスワードを指定する必要はありません。たとえば、オペレーティング・システムによる認証を受けたユーザーは、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 /
  ```

- ユーザー認証はオペレーティング・システムで集中管理され、Oracle Database がユーザーの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、ユーザー名は引き続きデータベース内で管理されます。
- データベースとオペレーティング・システムの監査証跡には、同じユーザー名が使用されます。

データベース・ユーザーの認証にオペレーティング・システムを使用する場合は、分散データベース環境とデータベース・リンクの管理に特別な注意が必要です。

ネットワークによる認証

Oracle Database 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
- 公開鍵インフラストラクチャベースの認証
- リモート認証

> **注意：** これらの方式を使用するには、Oracle Database Enterprise Edition と Oracle Advanced Security オプションが必要です。

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 Database では、これらのネットワーク・サービスによる認証を使用できます。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。

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 Database 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鍵管理。
- Oracle Call Interface（OCI）および PL/SQL ファンクション。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、信頼できる証明書を使用してデータのシグネチャを検証します。
- 信頼できる証明書。実在者との同一性が確認されたときにユーザー証明書の署名者として信頼できるサード・パーティ・エンティティの識別です。
- 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（信頼できる認証局）を含むデータ構造です。
- Oracle Wallet Manager。Oracle Wallet 内のセキュリティ資格証明の管理および編集に使用される、スタンドアロンの Java アプリケーションです。

- X.509v3 証明書。信頼できるエンティティ、つまり Oracle Database 外部の認証局から取得します（および署名を受けます）。
- Oracle Internet Directory。X.509 証明書により認証されたユーザーなど、ユーザーのセキュリティ属性と権限を管理します。Oracle Internet Directory は、属性レベルのアクセス制御を規定し、特定の属性の読取り、書込みまたは更新権限を、管理者などの特定のユーザーに限定できます。
- Oracle Enterprise Security Manager。集中的な権限管理を提供することで、管理を容易にし、セキュリティ・レベルを高めます。これにより、Oracle Internet Directory にロールを格納し、取り出すことができます。
- Oracle Enterprise Login Assistant。ユーザーの Wallet のオープンとクローズを行って、アプリケーションによる安定した SSL ベースの通信の使用を可能または不可能にする Java ベースのツールです。

リモート認証

Oracle Database は、Remote Authentication 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 Database による認証

Oracle Database 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。

データベース認証を使用するように Oracle Database を設定するには、対応するパスワードを指定して各ユーザーを作成します。ユーザーは、接続の確立時にそのパスワードを入力する必要があります。ユーザーが正しいパスワードを入力しない場合は接続が拒否されるため、データベースの無許可での使用が防止されます。Oracle Database では、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、無許可での変更が防止されます。ただし、ユーザーはいつでも自分のパスワードを変更できます。

データベース認証の機能は、次のとおりです。

- パスワード暗号化
- アカウントのロック
- パスワードの存続期間と期限切れ
- パスワードの複雑度の検証

パスワード暗号化

パスワードの機密性を保護するために、Oracle Database では常に、ネットワークを介してパスワードが送信される前に暗号化されます。Oracle Database では、変更された AES（Advanced Encryption Standard）アルゴリズムによってパスワードが暗号化されます。

アカウントのロック

指定した連続試行回数の範囲内でログインできない場合、Oracle Database ではユーザーのアカウントがロックされることがあります。一定の時間の後に自動的にロックが解除されるか、またはデータベース管理者によるロックの解除が必要になるように、アカウントを構成できます。データベース管理者が手動でアカウントをロックすることもできるため、その場合はデータベース管理者による明示的なロック解除が必要です。

パスワードの存続期間と期限切れ

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。その期間をすぎるとパスワードは期限切れになり、アカウントへのログインが再度許可されるには、パスワードを変更する必要があります。猶予期間を設定できます。この期間中は、データベース・アカウントへのログインを試行するたびに、パスワードの変更を求める警告メッセージが発行されます。その期間内にパスワードを変更しないと、アカウントはロックされます。それ以降、そのアカウントには、データベース管理者の介入がなければログインできなくなります。

また、データベース管理者がパスワードの状態を期限切れに設定することもできます。この場合は、ユーザーのアカウントの状態が期限切れに変更されます。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利用されないようにするためのものです。

パスワードの複雑度の検証

複雑度の検証は、パスワードを推定してシステムに入ろうとする侵入者に対して、各パスワードが十分保護可能な複雑なものであることをチェックすることです。

Oracle Database では、デフォルトのパスワード複雑度検証ルーチンにより、各パスワードが次の要件を満たしているかどうかがチェックされます。

- 長さが 8 文字以上 30 文字以下であること
- ユーザー名、ユーザー名の逆スペル、または数字を追加したユーザー名とは異なること
- サーバー名、または 1 ～ 100 の数字を追加したサーバー名とは異なること
- `welcome1`、`oracle1`、`user1234`、数字付きのアルファベット順の文字列、または `change_on_install` のように、単純なパスワードでないこと
- 少なくとも 1 文字のアルファベットと 1 文字の数字が含まれていること
- 前のパスワードと 3 文字以上異なっていること

> **関連項目：** Oracle Database によるパスワードの複雑度の検証の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

複数層の認証と認可

複数層環境では、Oracle Database は権限を制限し、すべての層を通じてクライアント識別を保持し、クライアントのために実行されるアクションを監査することによって、中間層アプリケーションのセキュリティを制御します。トランザクション処理モニターなど、大規模な中間層を使用するアプリケーションでは、中間層に接続するクライアントの識別性を維持する必要があります。ただし、中間層が持つ利点の 1 つに**接続プーリング**があります。これにより、複数のユーザーが個別に接続しなくても 1 つのデータ・サーバーにアクセスできます。この種の環境では、接続を迅速に確立および切断できる機能が必要です。

この種の環境では、Oracle データベース管理者は Oracle Call Interface（OCI）を使用して、各ユーザーのデータベース・パスワード認証を可能にする**軽量セッション**を作成できます。これにより、中間層を介して実際のユーザーの識別性が保たれ、個別のデータベース接続によるオーバーヘッドは生じません。

パスワードあり、またはパスワードなしで軽量セッションを作成できます。ただし、中間層が外部またはファイアウォールにある場合は、軽量セッションごとに専用パスワードを設定する方がセキュリティが向上します。内部アプリケーション・サーバーの場合は、パスワードなしの軽量セッションの方が適している場合があります。

Oracle Database 11*g* では、サーバー側接続プールを実装できます。これにより、様々なアプリケーションおよびアプリケーション・プロセスで、データベース接続を共有できます。サーバー側接続プールでサポートされているのは、パスワードベースの認証です。アドバンスト・セキュリティ・オプション（ASO）およびエンタープライズ・ユーザーは、現在はサポートされていません。

関連項目： サーバー側接続プールの詳細は、『Oracle Database 管理者ガイド』および『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

Secure Socket Layer プロトコルによる認証

Secure Socket Layer（SSL）プロトコルは、アプリケーション・レイヤー・プロトコルです。外部的またはグローバルに識別されたユーザー（外部ユーザーまたはグローバル・ユーザー）は、SSL を使用してデータベースを認証できます。

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 Database 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、厳密認証、オペレーティング・システムによる認証、またはパスワード・ファイルによる認証のいずれかを選択できます。データベースをローカルに（データベースが格納されているコンピュータ上で）管理する場合と、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・コンピュータを管理する場合では、異なる選択肢が適用されます。

厳密認証では、複数のデータベースに対する SYSDBA および SYSOPER のアクセスを集中管理できます。データベース管理においてこの種の認証を検討する必要があるのは、パスワード・ファイルのセキュリティに不安がある場合、サイトにきわめて厳格なセキュリティ要件が存在する場合、またはデータベースから識別管理を分離する場合です。

通常、データベース管理者のオペレーティング・システム認証には、データベース管理者のオペレーティング・システム・ユーザー名を特別なグループに入れること、または特別な処理を実行する権利を付与することが含まれます。（UNIX システムでは、これは **dba** グループです。）

データベースは、パスワード・ファイルを使用することによって、次の操作を実行できる 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。

- SYSOPER は、データベース管理者が STARTUP、SHUTDOWN、ALTER DATABASE OPEN/MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG および RECOVER を実行できるようにします。また、RESTRICTED SESSION 権限が含まれます。
- SYSDBA 権限には、ADMIN OPTION および SYSOPER 権限とともに、すべてのシステム権限が含まれます。CREATE DATABASE と時間ベースのリカバリを許可します。

関連項目：

- 認証と分散データベースの概念の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- Oracle Advanced Security オプションの詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。
- データベース管理者の認証の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 認証の詳細は、オペレーティング・システム固有の Oracle Database マニュアルを参照してください。

認可の概要

認可は、主に次の 2 つのプロセスに分かれています。

- データへのアクセス、処理または変更を特定のユーザーにのみ許可します。
- ユーザーによるアクセスまたは操作に様々な制限を適用します。ユーザーに適用（または除外）する制限は、スキーマ、表または行などのオブジェクトや、時間（CPU、接続またはアイドル時間）などのリソースに適用できます。

ここでは、この種の制限を個々のユーザーまたはユーザー・グループに対して適用または除外するための基本概念とメカニズムについて説明します。

この項の内容は、次のとおりです。

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 タイム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使用すると、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。

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。セキュリティ管理者は、プロファイルによるリソース制限の規定を全体的に使用可能または使用禁止に設定できます。

リソース制限を設定した場合、ユーザーがセッションを作成すると、パフォーマンスがわずかに低下します。これは、ユーザーが Oracle Database に接続した時点で、そのユーザーのすべてのリソース制限データがロードされるためです。

関連項目：

- セキュリティ管理者の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- データベース管理者の認証の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

この項の内容は、次のとおりです。

システム・リソースのタイプと制限

Oracle Database では、CPU タイムと論理読取りを含め、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。

この項の内容は、次のとおりです。

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。各セッションは、Oracle Database を実行するコンピュータの CPU タイム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。

Oracle Database では、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、現行の文が終了（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、現行トランザクションはコミットされます）のみです。それ以外の操作を実行すると、エラーが発生します。トランザクションがコミットまたはロールバックされた後も、カレント・セッションではユーザーはどんな作業も完了できません。

コール・レベル SQL 文が実行されるたびに、いくつかのステップが実行され文が処理されます。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、Oracle Database では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 Database は文の処理を停止してその文をロールバックし、エラーを戻します。ただし、カレント・トランザクションのそれ以前の文の結果はそのまま残り、そのユーザーのセッションは接続されたままになります。

CPU タイム SQL 文やその他のタイプのコールが Oracle Database 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 タイムが必要になります。平均的なコールであれば、わずかな CPU タイムですみます。ただし、大量のデータや冗長な問合せを伴う SQL 文は CPU タイムを大量にコンシュームすることがあるため、他の処理に使用できる CPU タイムが少なくなります。

CPU タイムが無制限に消費されないようにするため、1 回のコール当たりの CPU タイムと、1 つのセッション中に Oracle Database コールに使用される CPU タイムの合計を制限します。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 タイムで設定し、測定されます。

Logical Reads（論理読取り） 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 Database では、1 コール当たり、および 1 セッション当たりの論理データ・ブロック読取り数を制限できます。論理データ・ブロック読取りには、メモリーとディスクの両方からの論理データ・ブロック読取りが含まれます。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取りの数として設定し、測定されます。

その他のリソース Oracle Database 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。

- **ユーザー当た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 Database コール間の時間がアイドル制限時間に達すると、カレント・トランザクションがロールバックされて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。

  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。
- セッション当たりの経過接続時間の制限。セッションの持続時間が経過制限時間を超えると、カレント・トランザクションがロールバックされ、セッションが削除され、そのセッションのリソースがシステムに戻されます。この制限は、分単位の経過時間として設定します。

  Oracle Database 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。絶えず監視した場合、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 Database がこの制限を実施してセッションを異常終了させるまでに、セッションはこの制限をわずかに（5 分程度）超える可能性があります。
- セッションのプライベート SGA 領域（プライベート SQL 領域に使用）の容量の制限。この制限が重要になるのは、共有サーバーの構成を使用するシステムの場合のみです。それ以外のシステムの場合、プライベート SQL 領域は PGA 内にあります。この制限は、インスタンスの SGA に使用するメモリーのバイト数として設定します。KB または MB で指定するには、**K** または **M** の文字を使用します。

関連項目： リソース制限を使用可能および使用禁止にする手順は、『Oracle Database 管理者ガイド』を参照してください。

プロファイル

システム・リソースのコンテキストにおけるプロファイルとは、Oracle Database の有効なユーザー名に対して割り当てることができるリソース制限に名前を付けた集合です。プロファイルを使用すると、リソース制限を簡単に管理できます。また、パスワード方針を管理する手段としても使用できます。

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

この項の内容は、次のとおりです。

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・ポリシーの要件になっている場合のみです。プロファイルを使用するには、まずデータベース内の関連のあるユーザーを、いくつかのタイプに分類します。関連するユーザーの権限を管理するためにロールを使用するのと同様に、関連するユーザーのリソース制限を管理するためにプロファイルを使用します。データベースのすべてのタイプのユーザーを網羅するのに必要なプロファイルの数を決定し、それぞれのプロファイルに適切なリソース制限を決定します。

プロファイルのリソース制限の値の決定 プロファイルを作成し、そのプロファイルに含めるリソース制限を決定する前に、各リソース制限について適切な値を決定します。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。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。

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。

権限の概要

権限は、特定のタイプの SQL 文を実行するため、または別のユーザーのオブジェクトにアクセスするための権利です。

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。権限は、その権限を本当に必要としているユーザーにのみ付与します。必要でない権限まで付与すると、セキュリティを維持できなくなる可能性があります。ユーザーは次の 2 つの方法で権限を受け取ることができます。

- 権限を明示的にユーザーに付与します。たとえば、`employees` 表にレコードを挿入する権限を、ユーザー `SCOTT` に明示的に付与できます。
- 権限をロール（名前付きの権限グループ）に付与した上で、そのロールを 1 人以上のユーザーに付与します。たとえば、`employees` 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する必要があります。

権限は次の 2 つに分類できます。

- システム権限
- スキーマ・オブジェクト権限

> **関連項目：** すべてのシステム権限およびスキーマ・オブジェクト権限のリストと、権限管理の手順は、『Oracle Database 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 100 以上の種類があります。

スキーマ・オブジェクト権限

スキーマ・オブジェクト権限とは、次のように特定のスキーマ・オブジェクトに対して特定のアクションを実行する権限です。

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、`departments` 表から行を削除する権限は、1 つのオブジェクト権限です。

クラスタ、索引、トリガーおよびデータベース・リンクなど、一部のスキーマ・オブジェクトには、対応付けられたオブジェクト権限はありません。これらのオブジェクトの使用は、システム権限によって決定されます。たとえば、クラスタを変更するには、ユーザーはそのクラスタを所有しているか、または `ALTER ANY CLUSTER` システム権限が必要です。

スキーマ・オブジェクトとそのシノニムは、権限に関しては等価です。つまり、表、ビュー、順序、プロシージャ、ファンクションまたはパッケージについて付与されるオブジェクト権限は、その基礎となるオブジェクトを名前で参照するかシノニムを使用するかにかかわらず、適用されます。

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**に付与すると、シノニムを使用しない場合と同じ結果になります。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。

> **関連項目：** スキーマ・オブジェクト権限の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

ロールの概要

ロールを使用することで、権限の管理および制御が容易になります。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、ユーザーや他のロールに付与します。各ロール名はデータベース内で一意にする必要があり、どのユーザー名または他のどのロール名とも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。

ロールを使用すると、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ます。ただし、ストアド・プログラム構成メンバーの中からスキーマ・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ため、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ていません。

表 20-1 に、データベース内の権限管理を簡易化するロール・プロパティを示します。

表 20-1 ロールのプロパティ

プロパティ	説明
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じの権限の集合を明示的に付与するかわりに、関連するユーザー・グループの権限をまとめて 1 つのロールに付与することで、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する必要がある場合、修正が必要なのは、そのロールの権限のみです。すべてのユーザーにグループのロールが付与されるセキュリティ・ドメインでは、ロールに対す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。

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

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

関連項目： アプリケーションからロールを使用可能にする手順は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

この項の内容は、次のとおりです。

- ロールの一般的な使用方法
- ロールのメカニズム
- オペレーティング・システムとロール

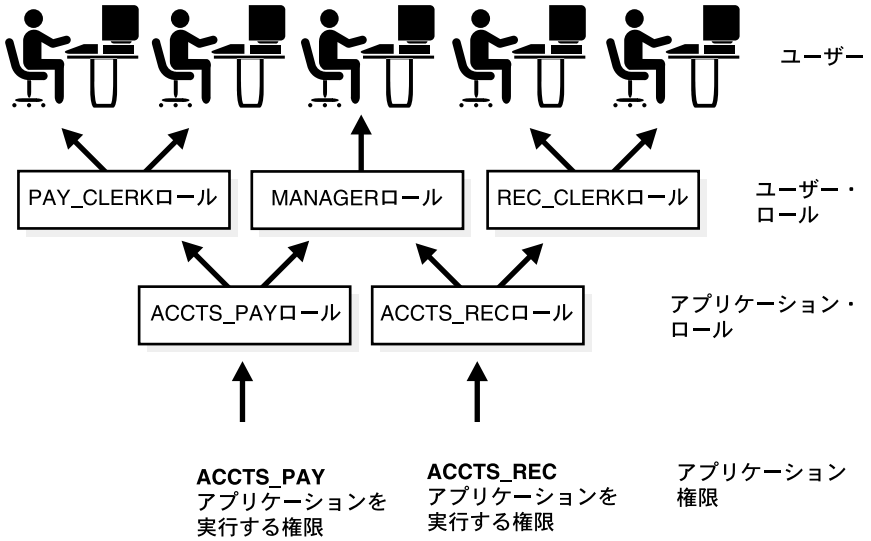
ロールの一般的な使用方法

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 20-1 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 20-1 ロールの一般的な使用方法



この項の内容は、次のとおりです。

- アプリケーション・ロール
- ユーザー・ロール

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、その保護アプリケーション・ロールを、他のロールや特定のユーザーに対して付与します。1 つのアプリケーションに対して複数の異なるロールを設定し、アプリケーション使用時のデータ・アクセスの量や範囲にあわせて異なる権限セットを各ロールに割り当てることができます。

ユーザー・ロール ユーザー・ロールは、共通の権限要件を持つデータベース・ユーザーのグループのために作成されるロールです。ユーザーの権限は、保護アプリケーション・ロールと権限をユーザー・ロールに付与し、そのユーザー・ロールを適切なユーザーに付与することによって管理します。

ロールのメカニズム

データベース・ロールには次の機能があります。

- ロールには、システム権限またはスキーマ・オブジェクト権限を付与できます。
- ロールには、別のロールを付与できます。ただし、ロールをそのロール自体に付与したり、循環的に付与することはできません。たとえば、ロール B があらかじめロール A に付与されている場合、ロール A をロール B には付与できません。
- 任意のロールを、任意のデータベース・ユーザーに付与できます。
- ユーザーに付与した各ロールは、任意の時点で使用可能または使用禁止にできます。ユーザーのセキュリティ・ドメインには、そのユーザーに対して現在使用可能になっているすべてのロールの権限が含まれており、ユーザーに対して現在使用禁止になっているロールの権限は除外されています。Oracle Database では、データベース・アプリケーションとユーザーがロールを使用可能または使用禁止にできるため、権限を選択的に使用できます。
- 間接的に付与されたロールとは、ロールに対して付与されたロールです。この種のロールは、ユーザーに対して明示的に使用可能または使用禁止にすることができます。ただし、別のロールを含んだロールを使用可能にすることによって、直接的に付与されたロールに含まれる間接的に付与された全ロールは、すべて暗黙のうちに使用可能になります。

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かぎりません。

保護アプリケーション・ロール

Oracle Database では、保護アプリケーション・ロールが用意されています。このロールは、認可された PL/SQL パッケージでのみ有効にできます。このメカニズムは、アプリケーションを起動するロールの有効化を制限します。

パスワードが、アプリケーションのソース・コードに埋め込まれていない場合、または表に格納されていない場合、セキュリティは強化されます。そのかわりに、ロールの有効化を認可する PL/SQL パッケージを指定して、保護アプリケーション・ロールを作成できます。パッケージの識別は、ロールを有効化するのに十分な権限があるかどうかの決定に使用します。ロールの有効化の前に、アプリケーションはユーザーがプロキシを介して接続しているかどうかをチェックするなど、認証およびカスタマイズ認可を実行できます。

ユーザーが定義者権限プロシージャ内のセキュリティ・ドメインを変更できないという制限により、保護アプリケーション・ロールは実行者権限プロシージャ内でのみ有効化できます。

関連項目：

- デフォルト・ロールの詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 保護アプリケーション・ロールの詳細は、『Oracle Database 2 日でセキュリティ・ガイド』を参照してください。
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

表、ビュー、シノニムまたは行に対するアクセス制限の概要

この項では、ユーザーではなくオブジェクトに関連する制限について説明します。制限により、それらはアクセスまたは変更を試みるエンティティにかかわらず保護されます。

このような保護を提供するには、特定の表、ビュー、シノニムまたは行へのアクセスを制限するためのポリシーを設計して使用します。これらのポリシーにより、制限を確立する動的な述語を指定するように設計されたファンクションが起動されます。また、確立されたポリシーをグループ化し、そのポリシー・グループを特定のアプリケーションに適用することもできます。

このように確立された保護に対する脅威や侵害が発生する場合には、通知を受け取る必要があります。通知を受け取ると、防御を強化するか、不適切なアクションとそれを実行したエンティティに対処できます。

この項の内容は、次のとおりです。

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、ファンクションを使用してセキュリティ・ポリシーを実装し、セキュリティ・ポリシーを表、ビューまたはシノニム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規定されます。

次のことができます。

- SELECT、INSERT、UPDATE および DELETE（および、行レベルのセキュリティ・ポリシーの場合は INDEX）に異なるポリシーを使用します。
- 必要な場合（給与情報など）にのみセキュリティ・ポリシーを使用します。
- パッケージ化されたアプリケーションの最上位に基本ポリシーを作成するなど、各表に複数のポリシーを使用します。
- 異なるアプリケーションのポリシーは、ポリシー・グループを使用して区別します。各ポリシー・グループは、1 つのアプリケーションに属するポリシーの集合です。データベース管理者は駆動コンテキストと呼ばれるアプリケーション・コンテキストを指定して、有効になっているポリシー・グループを示します。表、ビューまたはシノニムにアクセスすると、ファイングレイン・アクセス・コントロールのエンジンが駆動コンテキストを検索して有効なポリシー・グループを決定し、そのポリシー・グループの対応付けられたポリシーすべてを規定します。

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（またはポリシー・グループ）の追加、削除、使用可能と使用禁止の切替えおよびリフレッシュを行います。

関連項目：

- パッケージの実装の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ファイングレイン・アクセス・コントロールの詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

動的な述語

ビューにセキュリティ・ルールが埋め込まれている場合ではなく、実表またはビューが DML 文で参照される場合、動的な述語は文の解析時に取得されます。

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。この述語により、ポリシーの指定に従って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。

表、ビューまたはシノニムに対する動的な述語は、PL/SQL インタフェースを介してセキュリティ・ポリシーに対応付けられた PL/SQL ファンクションにより生成されます。

アプリケーション・コンテキスト

アプリケーション・コンテキストを使用すると、アプリケーションにファンクション・ベースのセキュリティ・ポリシーを対応付けることができるため、ファイングレイン・アクセス・コントロールの適用が容易になります。

各アプリケーションには固有のコンテキストがあり、ユーザーが任意に（SQL*Plus などを介して）変更することはできません。コンテキスト属性には、セキュリティ・ポリシーを実装するファンクションからアクセスできます。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。

そのため、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を使用した、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。

次のことができます。

- コンテキスト値のベースに述語を使用します。
- 述語内でコンテキスト値をバインド変数として使用します。
- ユーザー属性を設定します。
- ユーザー属性にアクセスします。

関連項目：

- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

動的コンテキスト

ポリシーでは、それが静的、共有、状況依存、動的のいずれであるかを指定することでランタイム効率を識別できます。

静的ポリシーの場合は、オブジェクトにアクセスするエンティティに関して同じ条件文字列を生成すると、それが 1 度実行されて SGA にキャッシュされます。同じオブジェクトにアクセスする文に対するポリシーの場合、ポリシー関数は再実行されず、かわりにキャッシュされた条件が使用されます。

これは共有の静的ポリシーの場合も同様で、サーバーは最初に、同じポリシー・タイプの同じポリシー関数によって生成されキャッシュに置かれた条件を検索します。共有の静的ポリシーは、ほぼすべてのオブジェクトが同じ関数を共有し、ポリシーが静的であるため、ホスト上のデータ・パーティションに最適です。

ポリシーに状況依存のラベルを付けると、サーバーは常に文の解析時にポリシー関数を実行し、戻り値をキャッシュしません。カーソルが最後に使用された後にサーバーがコンテキストの変化を検出しないかぎり、文の実行時にポリシー関数が再評価されることはありません。（複数のクライアントが 1 つのデータベース・セッションを共有するセッション・プーリングの場合は、クライアント切替え時に中間層でコンテキストをリセットする必要があります）。

状況依存ポリシーが共有される場合、サーバーは最初に、同じデータベース・セッションで同じポリシー・タイプの同じポリシー関数によって生成されキャッシュに置かれた述語を検索します。セッション・メモリー内で述語が検出されると、ポリシー関数は再実行されず、セッションのプライベート・アプリケーション・コンテキストが変化するまでは、キャッシュに置かれた値が有効になっています。

動的ポリシーの場合、サーバーでは述語が常にシステムまたはセッション環境の影響を受けるとみなされ、文の解析または実行するたびにポリシー関数を再実行します。

ファイングレイン監査

ファイングレイン監査を行うと、データ・アクセスを内容に基づいて監視できます。INSERT、UPDATE および DELETE 操作と問合せをきめ細かく監査できます。たとえば、中央の税務当局は、職員によるアクセス違反から保護するために、アクセスされたデータの判別に十分な詳細を使用して所得申告情報へのアクセスを追跡する必要があります。特定のユーザーが特定の表に対して SELECT 権限を使用したということがわかるだけでは十分ではありません。ファイングレイン監査は、このより詳細な機能を提供します。

通常、ファイングレイン監査方針は、選択的監査の条件として表オブジェクトに対する単純なユーザー定義 SQL 条件に基づいています。フェッチ中に戻される行が方針の条件を満たすと、その問合せが監査対象となります。後で Oracle Database は自律型トランザクションを使用してユーザー定義の監査イベント・ハンドラを実行し、イベントを処理します。

ファイングレイン監査をユーザー・アプリケーションに実装するには、DBMS_FGA パッケージを使用する方法と、データベース・トリガーを使用する方法があります。

> **関連項目：** ファイングレイン監査の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

セキュリティ・ポリシーの概要

この項の内容は、次のとおりです。

- システム・セキュリティ・ポリシー
- データ・セキュリティ・ポリシー
- ユーザー・セキュリティ・ポリシー
- パスワード管理ポリシー
- 監査方針

システム・セキュリティ・ポリシー

各データベースでは、セキュリティ管理者と呼ばれる 1 人以上の管理者が、セキュリティ・ポリシーのあらゆる側面の保守を担当します。データベース・システムが小規模な場合は、データベース管理者がセキュリティ管理者を兼務する場合もあります。ただし、データベース・システムが大規模な場合は、特定の管理者または管理者グループが専任でセキュリティ管理者を務める場合があります。

すべてのデータベースについて、セキュリティ・ポリシーを開発する必要があります。また、セキュリティ・ポリシーには後述する複数のサブポリシーを組み込む必要があります。

この項の内容は、次のとおりです。

- データベース・ユーザーの管理
- ユーザー認証
- オペレーティング・システムのセキュリティ

データベース・ユーザーの管理

データベース・システムの規模と、データベース・ユーザーの管理に必要な作業量によっては、データベース・ユーザーの作成、変更または削除に必要な権限をセキュリティ管理者にのみ付与できます。また、データベース・ユーザーを管理するための権限を複数の管理者に付与することもできます。どちらの場合にも、データベース・ユーザーを管理するための強力な権限は、信頼できるユーザーにのみ付与する必要があります。

ユーザー認証

データベース・ユーザーは、データベース・パスワード、ホスト・オペレーティング・システム、ネットワーク・サービスまたは Secure Sockets Layer（SSL）を使用して、Oracle Database で**認証**を受ける（正しいユーザーとして検証される）ことができます。

関連項目： 20-5 ページの「認証方式の概要」

オペレーティング・システムのセキュリティ

該当する場合は、Oracle Database および他のデータベース・アプリケーションを実行するオペレーティング・システム環境について、次のセキュリティ上の問題も考慮する必要があります。

- データベース管理者には、ファイルを作成および削除するためのオペレーティング・システム権限が必要です。
- 通常のデータベース・ユーザーには、データベース関連ファイルを作成または削除するためのオペレーティング・システム権限を付与しません。
- オペレーティング・システムでユーザーのデータベース・ロールが識別される場合、セキュリティ管理者にはオペレーティング・システム・アカウントのセキュリティ・ドメインを変更するためのオペレーティング・システム権限が必要です。

データ・セキュリティ・ポリシー

データ・セキュリティには、オブジェクト・レベルにおけるデータベースのアクセスと使用を制御するメカニズムが含まれています。データ・セキュリティ・ポリシーにより、特定のスキーマ・オブジェクトにアクセスするユーザーと、各ユーザーがオブジェクトに対して許可されるアクションのタイプが決定されます。たとえば、ユーザー `scott` が `employees` 表を使用する場合、`SELECT` および `INSERT` 文は発行できますが、`DELETE` 文は発行できません。データ・セキュリティ・ポリシーでは、スキーマ・オブジェクトごとに監査対象となるアクションがある場合は、そのアクションも定義する必要があります。

データ・セキュリティ・ポリシーは、主にデータベース内のデータに必要なセキュリティのレベルによって決定されます。たとえば、ユーザーにスキーマ・オブジェクトの作成を許可する場合や、ユーザーのオブジェクトへのアクセス権限をシステムの他のユーザーに付与することを許可する場合、データベース内のデータ・セキュリティはほとんど必要ありません。また、オブジェクトを作成するための権限と、ロールおよびユーザーに対するオブジェクトへのアクセス権限を付与するための権限を、データベース管理者またはセキュリティ管理者のみに限定する必要がある場合は、データ・セキュリティの厳密な制御が必要になります。

データ・セキュリティ全体は、データの機密性をベースとする必要があります。情報に機密性がない場合は、データ・セキュリティ・ポリシーを緩和できます。ただし、データの機密性が高い場合は、オブジェクトへのアクセスを常に厳密に制御するように、セキュリティ・ポリシーを開発する必要があります。

データ・セキュリティの実装手段には、システム権限、オブジェクト権限およびロールなどがあります。ロールとは、ユーザーにまとめて付与できるようにグループ化された権限の集合です。ビュー定義で表データへのアクセスを制限できるため、ビューでデータ・セキュリティを実現することもできます。これにより、機密データを含む列を除外できます。

データ・セキュリティのもう1つの実装手段は、ファイングレイン・アクセス・コントロールを介して、関連するアプリケーション・コンテキストを使用することです。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・ポリシーを表またはビューに対応付けることができます。実際には、セキュリティ・ポリシー関数により SQL 文に追加する WHERE 条件が生成され、表またはビューのデータ行に対するユーザー・アクセスが制限されます。アプリケーション・コンテキストは、アクセス制御の決定に使用される情報を格納するための保護データ・キャッシュです。

関連項目：

- ビューの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- ファイングレイン・アクセス・コントロールとアプリケーション・コンテキストの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

ユーザー・セキュリティ・ポリシー

この項では、ユーザー・セキュリティ・ポリシーの様々な側面について説明します。この項の内容は、次のとおりです。

一般ユーザー・セキュリティ

すべてのタイプのデータベース・ユーザーについて、パスワード・セキュリティと権限管理を考慮してください。

ユーザー認証をデータベースで管理する場合、セキュリティ管理者は、データベース・アクセス・セキュリティを維持するためにパスワード・セキュリティ・ポリシーを開発する必要があります。たとえば、データベース・ユーザーは各自のパスワードを定期的に変更する必要があります。ユーザーにパスワードの変更を強制することで、無許可のデータベース・アクセスを減らすことができます。

また、あらゆるタイプのユーザーについて、権限管理に関連する問題を考慮する必要があります。たとえば、多数のユーザー、アプリケーションまたはオブジェクトを伴うデータベースの場合は、ロールを活用してユーザーが使用可能な権限を管理できます。また、データベースで使用されるユーザー名が少数の場合には、ロールを使用せずに、ユーザーに対して権限を明示的に付与する方が容易な場合があります。

エンド・ユーザー・セキュリティ

セキュリティ管理者は、エンド・ユーザー・セキュリティのポリシーを定義する必要があります。データベースに多数のユーザーがいる場合、セキュリティ管理者はどのユーザーをまとめて各ユーザー・グループに分類するかを決定し、各グループのユーザー・ロールを作成できます。また、必要な権限またはアプリケーション・ロールを各ユーザー・ロールに付与し、そのユーザー・ロールを各ユーザーに付与できます。例外を考慮して、どの権限を個々のユーザーに明示的に付与する必要があるかについても決定する必要があります。

様々なデータベース・ユーザー・グループが必要とする共通の権限を付与して管理するには、ロールを使用するのが最も簡単な方法です。また、Oracle Advanced Security のエンタープライズ・ユーザーおよびエンタープライズ・ロール機能を介して、ユーザーとその認可をディレクトリ・サービスで集中管理することもできます。

管理者セキュリティ

セキュリティ管理者には、データベース管理者セキュリティに対処するポリシーが必要です。たとえば、データベースが大規模で数タイプのデータベース管理者が存在する場合、セキュリティ管理者が関連する管理権限を複数の管理ロールにグループ化するように決定できます。これにより、管理ロールを適切な管理者ユーザーに付与できます。また、データベースが小規模で管理者が少人数であれば、管理ロールを 1 つ作成して管理者全員に付与する方が便利な場合があります。

SYS および SYSTEM としての接続の保護 SYS および SYSTEM にデフォルトのパスワードを使用した場合は、データベースを作成した直後に SYS および SYSTEM 管理ユーザー名のパスワードを変更してください。SYS または SYSTEM として接続すると、ユーザーにはデータベースを変更するための強力な権限が付与されます。

他の管理ユーザー名を作成させるオプションをインストールしている場合、そのユーザー名のアカウントは最初はロック状態で作成されます。

管理者接続の保護 管理権限を使用してデータベースに接続できるユーザーは、データベース管理者のみに限定する必要があります。次に例を示します。

```
CONNECT SYS/AS SYSOPER|SYSDBA
Enter password: enter the password
```

SYSOPER として接続するユーザーには、基本的な運用タスク（STARTUP、SHUTDOWN およびリカバリ操作など）を実行するための権限が付与されます。SYSDBA として接続するユーザーには、これらの権限に加えて、データベースまたはデータベース内のオブジェクトに対してあらゆる操作（CREATE、DROP および DELETE など）を実行するための無制限の権限が付与されます。SYSDBA を使用すると、ユーザーは SYS スキーマに置かれるため、データ・ディクショナリ表を変更できます。

アプリケーション開発者セキュリティ

セキュリティ管理者は、データベースを使用するアプリケーション開発者について、特別なセキュリティ・ポリシーを定義する必要があります。アプリケーション開発者には、必要なオブジェクトを作成するための権限を付与できます。または、オブジェクトの作成権限を、開発者からオブジェクト作成要求を受け取るデータベース管理者にのみ付与する方法もあります。

> **関連項目：** アプリケーション開発者の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

アプリケーション開発者とその権限 データベース・アプリケーション開発者は、担当のジョブを達成するために特別な権限グループを必要とする一意のデータベース・ユーザーです。エンド・ユーザーとは異なり、開発者には、CREATE TABLE、CREATE PROCEDURE などのシステム権限が必要です。ただし、開発者には特定のシステム権限のみを付与し、データベース内で実行可能な操作全体を制限する必要があります。

多くの場合、アプリケーション開発者が実行できる操作はデータベースのテストに限定され、本番データベースで実行することは許可されません。この制限により、アプリケーション開発者がエンド・ユーザーとの間でデータベース・リソースをめぐって競合しないことと、本番データベースに有害な影響を及ぼさないことが保証されます。アプリケーションが細部に至るまで開発されテストされた後は、本番データベースへのアクセスが許可され、本番データベースの適切なエンド・ユーザーが使用可能になります。

セキュリティ管理者はロールを作成して、典型的なアプリケーション開発者に必要な権限を管理できます。

通常、アプリケーション開発者には開発プロセスの一部としてオブジェクトを作成するための権限が付与されますが、セキュリティ管理者は各アプリケーション開発者が使用できるデータベース領域とそのサイズに対する制限を維持する必要があります。たとえば、セキュリティ管理者はアプリケーション開発者ごとに次の制限を特別に設定または制限します。

- 開発者が表または索引を作成できる表領域
- 開発者がアクセスできる各表領域の割当て制限

どちらの制限も、開発者のセキュリティ・ドメインを変更することで設定できます。

アプリケーション管理者セキュリティ

多数のデータベース・アプリケーションが関連する大規模データベース・システムの場合は、次のタイプのタスクを担当するアプリケーション管理者を割り当てることを考慮してください。

- アプリケーション・ロールの作成と、各アプリケーション・ロールの権限の管理
- データベース・アプリケーションで使用するオブジェクトの作成と管理
- アプリケーション・コードと Oracle Database のプロシージャおよびパッケージの管理と必要に応じた更新

通常、アプリケーション管理者は、アプリケーションを設計したアプリケーション開発者です。ただし、アプリケーション管理者には、データベース・アプリケーションをよく理解している他のユーザーを指定する場合もあります。

パスワード管理ポリシー

パスワードに依存するデータベース・セキュリティ・システムでは、常にパスワードの機密を保つ必要があります。ただし、パスワードは盗まれたり忘れたり誤用する可能性があります。データベース・セキュリティを厳密に管理できるように、Oracle Database のパスワード管理ポリシーは DBA とセキュリティ管理者によりユーザー・プロファイルを介して制御されます。

関連項目：

- 20-7 ページの「Oracle Database による認証」
- パスワード保護の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

監査方針

セキュリティ管理者は、各データベースの監査手順に関する方針を定義する必要があります。問題のあるアクティビティが疑われる場合を除き、データベース監査は使用禁止にするように決定できます。監査が必要な場合は、データベース監査の詳細レベルを決定します。通常、システム監査の後は、疑わしいアクティビティの発生源が判別された後で、さらに限定的なタイプの監査を実行します。次の項では、監査について説明します。

データベース監査の概要

監査とは、選択したユーザー・データベース・アクションを監視して記録する処理のことです。実行された SQL 文のタイプなどの個々のアクション、または名前、アプリケーション、時間などの様々な要因の組合せをベースとして使用できます。Oracle Database の指定の要素がアクセスされたり、その内容などに変更があった場合に、セキュリティ・ポリシーに基づいて監査を実行できます。

監査は、通常次のような目的で使用されます。

- 特定のスキーマ、表、行または影響を受ける内容に対して実行された現行のアクションの今後のアカウンタビリティの有効化。
- 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。

Enterprise Manager を使用して、監査関連の初期化パラメータを表示および構成し、文監査およびスキーマ・オブジェクト監査の監査オブジェクトを管理できます。たとえば、Enterprise Manager では、現行の監査対象の文、権限およびオブジェクトのプロパティが示されます。各オブジェクトのプロパティを参照でき、プロパティを基準にして監査オブジェクトを検索できます。また、オブジェクト、文および権限に対する監査をオンまたはオフにできます。

監査のタイプとレコード

Oracle Database では、監査オプションの対象範囲を広くしたり狭くできます。次の監査ができます。

- 文の正常な実行、失敗した実行、またはその両方
- ユーザー・セッションごと、または文が実行されるごとの文の実行
- すべてのユーザーまたは特定のユーザーのアクティビティ

Oracle Database の監査機能では、表 20-2 のように複数の異なるメカニズムを使用できます。

表 20-2 監査のタイプ

監査のタイプ	意味 / 説明
文監査	SQL 文が処理する特定のスキーマ・オブジェクト別ではなく、文のタイプ別による SQL 文の監査。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。また文監査では、選択されたデータベースのユーザー、またはデータベースの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・アクションを使用可能にする強力なシステム権限の使用方法の監査。権限監査は、対象となる権限の使用方法のみを監査するため、監査対象が文監査より限定されています。権限監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
スキーマ・オブジェクト監査	`AUDIT SELECT ON employees` など、特定のスキーマ・オブジェクトの特定の文に対する監査。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。
ファイングレイン監査	内容に基づくデータ・アクセスとアクションの監査。セキュリティ管理者は `DBMS_FGA` を使用して、ターゲット表の監査方針を作成します。DML 文のブロックから戻された行が監査条件と一致すると、監査証跡に監査イベント・エントリが挿入されます。

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・システム監査証跡と呼ばれるオペレーティング・システム・ファイルのどちらかに格納できます。

この項の内容は、次のとおりです。

データベース監査証跡 データベース監査証跡は、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 SYS.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、複数の事前定義済のビューが提供されています。

監査対象のイベントと設定されている監査オプションに応じて、監査証跡には様々な種類の情報が記録されます。ただし、次の情報は、特定の監査アクションにとって意味がある場合、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
インスタンス番号
プロセス識別子
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

分散データベースの監査 監査機能には、サイト自律性があります。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 Database ノードは、リモート・データベースで発生するアクションを監査できません。リモート接続はデータベース・リンクのユーザー・アカウントを介して確立されるため、リモート Oracle Database 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。

オペレーティング・システム監査証跡 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 Database 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。それ以外の場合は、監査レコードは、他の Oracle Database トレース・ファイルに類似した形式のデータベース外のファイルに書き込まれます。

Oracle Database 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の監査を続行できます。オペレーティング・システムの監査証跡に記録できないのは、多くの場合、オペレーティング・システムの監査証跡またはファイル・システムがいっぱいになっており、新しいレコードが入らないことが原因です。

オペレーティング・システム監査を構成するシステム管理者は、監査証跡またはファイル・システムがいっぱいにならないようにしてください。ほとんどのオペレーティング・システムでは、このような状況を回避できるように十分な情報と警告が管理者に提供されます。ただし、データベースの監査証跡を使用するように監査を構成すれば、その危険性を回避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 Database によって防止されるためです。

オペレーティング・システム監査レコード オペレーティング・システム監査証跡はエンコードされていますが、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**は、実行または試行された操作の記述です。これらのコードは AUDIT_ACTIONS データ・ディクショナリ表に記述されます。
- **使用された権限**は、操作の実行に使用されたシステム権限の記述です。これらのコードはすべて SYSTEM_PRIVILEGE_MAP 表に記述されます。
- **完了コード**は、試行された操作の結果の記述です。成功した操作からは、値 0（ゼロ）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 Database エラー・コードが戻されます。

  関連項目：

  - 事前定義済のビューの作成方法と使用方法は、『Oracle Database 管理者ガイド』を参照してください。
  - 監査の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
  - 完了コードのリストは、『Oracle Database エラー・メッセージ』を参照してください。

オペレーティング・システム監査証跡に常に記録される場合 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、ある種のデータベース関連アクションは常にオペレーティング・システム監査証跡に記録されます。

- インスタンス起動時には、インスタンスを起動したオペレーティング・システム・ユーザー、そのユーザーの端末識別子、日時のタイムスタンプ、およびデータベース監査が使用可能になっていたかどうかを記述した監査レコードが生成されます。データベース監査証跡は起動が正常に完了しないと使用可能にならないため、この情報はオペレーティング・システム監査証跡に記録されます。起動時のデータベース監査の状態を記録することは監査フラグとしても機能するため、管理者がデータベース監査を使用禁止にした状態で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することは禁止されます。
- インスタンス停止時には、インスタンスを停止したオペレーティング・システム・ユーザー、そのユーザーの端末識別子および日時のタイムスタンプを記述した監査レコードが生成されます。
- 管理者権限による接続時には、管理者権限を使用して Oracle Database に接続中のオペレーティング・システム・ユーザーの詳細を記述した監査レコードが生成されます。このレコードにより、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。

Oracle Database から監査証跡にアクセスできないように設定されたオペレーティング・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 Database 監査証跡ファイルに格納されます。

監査レコードが作成される場合 許可されたデータベース・ユーザーは独自の監査オプションをいつでも設定できますが、監査情報の記録を使用可能または使用禁止にするのはセキュリティ管理者です。

データベースの監査機能が使用可能になっている場合、監査レコードは文実行の実行フェーズで生成されます。

PL/SQL プログラム・ユニット内の SQL 文は、プログラム・ユニットの実行時に必要に応じて監査されます。

監査証跡レコードの生成と挿入は、ユーザーのトランザクションのコミットからは独立して実行されます。つまり、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。

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。セッション中に文監査または権限監査のオプションを設定または変更しても、そのセッション中は有効になりません。修正した文監査オプションまたは権限監査オプションは、カレント・セッションを終了し、新しいセッションを作成する時点で有効になります。一方、オブジェクト監査オプションについて変更した内容は、カレント・セッションでただちに有効になります。

SYS ユーザーや、SYSDBA または SYSOPER を介して接続したユーザーによる操作は、AUDIT_SYS_OPERATIONS 初期化パラメータを使用して完全に監査できます。SYS により実行され、正常終了した SQL 文は、常に監査対象となります。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の場合、生成された監査レコードはオペレーティング・システムの位置に送られます。SYS スキーマ内の通常のデータベース監査証跡とは別の場所に送信すると、監査のセキュリティが向上します。

関連項目：

- 監査を使用可能および使用禁止にする手順は、『Oracle Database セキュリティ・ガイド』を参照してください。
- SQL 文処理の様々なフェーズや共有 SQL の詳細は、第 24 章「SQL」を参照してください。

21

データ整合性

この章では、整合性制約を使用してデータベースに関連するビジネス・ルールを規定し、表への無効な情報入力を防止する方法について説明します。

この章の内容は、次のとおりです。

データ整合性の概要

列データが、データベース管理者やアプリケーション開発者によって事前定義された規則に準拠するのは重要なことです。

たとえば、データベース表の列に含まれるデータは、その列に設定された特定の規則の制約を受けることがあります。このような制約は、ある表と別の表のデータ列の関連付けに影響を与える可能性があります。

この項の内容は、次のとおりです。

データ整合性規則

この項では、様々な種類のデータ整合性を規定するために表の列に適用できる規則について説明します。

NULL 規則 : NULL 規則は、単一の列に対して定義される規則で、その列での 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規則は、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かぎって、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規則は、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。

参照整合性規則 : 参照整合性規則は、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:** 参照先のデータの更新または削除を禁止します。
- **SET NULL:** 参照先のデータが更新または削除されると、対応する依存データがすべて `NULL` に設定されます。
- **SET DEFAULT:** 参照先のデータが更新または削除されると、対応する依存データがすべてデフォルト値に設定されます。
- **CASCADE:** 参照先のデータが更新されると、対応する依存データもすべて更新されます。参照先の行が削除されると、対応する依存行もすべて削除されます。
- **NO ACTION:** 参照先のデータの更新または削除を禁止します。これは、文の最後にチェックされるか、または制約が遅延されている場合はトランザクションの最後にチェックされるという点が `RESTRICT` とは異なります。（Oracle Database 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 : 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 Database でデータ整合性を規定する方法

Oracle Database では、前述のデータ整合性規則を定義し、規定できます。これらの規則のほとんどは、整合性制約またはデータベース・トリガー（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）を使用して簡単に定義できます。

子表と親表が分散データベースの別のノードにある場合、宣言整合性制約を使用して参照整合性の規定はできません。ただし、データベース・トリガーを使用して、分散データベースで参照整合性を規定することはできます。

> **関連項目：** データ整合性の規定に使用されるトリガーの例は、第 22 章「トリガー」を参照してください。

制約の状態

- ENABLE を指定すると、入ってくるすべてのデータが制約に準拠していることが保証されます。
- DISABLE を指定すると、入ってくるデータは、制約に準拠しているかどうかに関係なく許可されます。
- VALIDATE を指定すると、既存のデータが制約に準拠していることが保証されます。
- NOVALIDATE は、一部の既存データが制約に準拠していない可能性があることを意味します。

さらに、次のようになります。

- ENABLE VALIDATE は ENABLE と同じです。制約がチェックされ、すべての行が保持されることが保証されます。
- ENABLE NOVALIDATE は、制約はチェックされますが、すべての行について TRUE でなくても許されることを意味します。これにより、既存の行が制約に違反することは許され、しかも、新しい行や変更した行に対してはすべて有効であることが保証されます。

  ALTER TABLE 文の ENABLE NOVALIDATE オプションを使用すると、表内のすべてのデータの妥当性を最初にチェックせずに、使用禁止にした制約について制約チェックを再開できます。
- DISABLE NOVALIDATE は DISABLE と同じです。制約はチェックされず、TRUE でなくてもかまいません。
- 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。

  一意制約の場合、DISABLE VALIDATE 状態では、ALTER TABLE 文の EXCHANGE PARTITION 句を使用して、非パーティション表からパーティション表にデータを効率よくロードできます。

これらの状態間の遷移は、次の規則で制御されます。

- NOVALIDATE が指定されていない場合、ENABLE は暗黙的に VALIDATE を意味します。
- VALIDATE が指定されていない場合、DISABLE は暗黙的に NOVALIDATE を意味します。
- VALIDATE と NOVALIDATE には、ENABLE 状態と DISABLE 状態に関するデフォルトの含意はありません。
- 一意キーまたは主キーが DISABLE 状態から ENABLE 状態に移る場合に、既存の索引がなければ一意索引が自動的に作成されます。同様に、一意キーまたは主キーが ENABLE から DISABLE に移る場合に、一意索引で使用可能にされていれば、一意索引は削除されます。
- 制約が NOVALIDATE 状態から VALIDATE 状態に移る場合は、すべてのデータをチェックする必要があります（この操作は、きわめて低速の場合があります）。ただし、VALIDATE から NOVALIDATE に移っても、データがチェックされたことが無視されるだけです。
- 単一の制約の ENABLE NOVALIDATE 状態から ENABLE VALIDATE 状態への移動では、読取り、書込みまたは他の DDL 文はブロックされません。パラレルに実行できます。

関連項目：

- 制約の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 制約の管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。

整合性制約の概要

Oracle Database 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ビジネス・ルールを規定できます。DML 文を実行した結果が整合性制約に違反すると、Oracle Database 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、`employees` 表の `salary` 列に、どの行でもこの列に 10,000 より大きい数値を入れないという規則を規定する整合性制約を定義するとします。`INSERT` 文または `UPDATE` 文がこの整合性制約に違反すると、Oracle Database 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。

この項の内容は、次のとおりです。

- 整合性制約の利点
- 整合性制約のパフォーマンス・コスト

整合性制約の利点

この項では、他の代替方法と比較して、データベースの表に関連付けられた整合性制約が優れている点を説明します。その利点は次のとおりです。

- データベース・アプリケーションのコードでビジネス・ルールを規定します。
- ストアド・プロシージャを使用してデータへのアクセスを完全に制御します。
- トリガーされるストアド・データベース・プロシージャを使用してビジネス・ルールを規定します。

関連項目： 第 22 章「トリガー」

この項の内容は、次のとおりです。

- 宣言の容易さ
- 規則の集中化
- アプリケーション開発の生産性の最大化
- ユーザーへの即時フィードバック
- データ・ロードの柔軟性と整合性違反の識別

宣言の容易さ

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。SQL 文を使用して整合性制約を定義するため、追加のプログラミングなしに表を定義または変更できます。SQL 文は記述が容易で、プログラミングのエラーを避けられます。Oracle Database が SQL 文の機能を制御します。

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ランダムなデータ・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。

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 Database 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。）

規則の集中化

整合性制約は、（アプリケーションではなく）表に対して定義され、データ・ディクショナリに格納されます。したがって、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。アプリケーション・コードではなく、アプリケーション・コードの集中管理された整合性制約内にビジネス・ルールを保持することによって、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。

アプリケーション開発の生産性の最大化

整合性制約によって規定されるビジネス・ルールを変更する場合、管理者が整合性制約を変更するだけで、すべてのアプリケーションは変更後の制約を自動的に遵守するようになります。それに対して、それぞれのデータベース・アプリケーションのコードでビジネス・ルールを規定する場合、開発者はすべてのアプリケーションのソース・コードを修正して、その修正したアプリケーションを再コンパイルし、デバッグしてテストする必要があります。

ユーザーへの即時フィードバック

Oracle Database 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 Database が SQL 文を実行しチェックする前でも、データベース・アプリケーションが、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設計できます。たとえば Oracle 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。

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レポートさせることもできます。

整合性制約のパフォーマンス・コスト

データ整合性の規則を規定することによる利点は、パフォーマンスを多少犠牲にして初めて獲得できます。一般に、整合性制約を組み込む場合のコストは、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

整合性制約を使用して、通常列と仮想列の両方で値の入力時に制限を設定できます。次の制約を使用できます。

- NOT NULL 整合性制約
- 一意キー制約
- 主キー整合性制約
- 参照整合性制約
- チェック整合性制約

関連項目： 仮想列の概要は 5-3 ページの「表の概要」を、仮想列の参照情報は『Oracle Database SQL 言語リファレンス』を参照してください。

NOT NULL 整合性制約

デフォルトでは、表のすべての列で NULL を使用できます。**NULL** は、値がないことを意味します。NOT NULL 制約がある場合、表の列値に NULL を使用できません。

表のタイプおよび列のデータ型によっては、追加する列に NOT NULL 制約が使用され、かつデフォルト値が設定されている場合、データベースでの操作が最適化され、表が DML 用にロックされる時間が短縮されることがあります。追加する列のデフォルト値が記述されたメタデータが、データベースの表に保存されます。そのため、列を追加してもデータベースで各行にデフォルト値を移入する必要がありません。これにより、表がロックされる時間が最小限になります。

NOT NULL 制約を持つ列は、行が 1 つも含まれていない表またはデフォルト値が指定された表にのみ追加できます。

関連項目：

- 表への列の追加方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- ALTER TABLE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

一意キー制約

一意キー制約では、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。

この項の内容は、次のとおりです。

- 一意キー
- 一意キー整合性制約と NOT NULL 整合性制約の組合せ

一意キー

一意キー制約の定義に含まれている列を、**一意キー**と呼びます。一意キーが 2 つ以上の列で構成されている場合、その列グループのことを**コンポジット一意キー**と呼びます。

「一意キー」という用語は、「一意キー制約」または「一意索引」のシノニムとして誤用されることがよくあります。しかし、「キー」とは、整合性制約の定義に使用される列または列のグループにすぎません。

たとえば、一意キー制約では、市外局番と電話番号を何回でも入力できますが、特定の市外局番と電話番号の組合せが表内で重複することは許可されません。これにより、同じ電話番号が誤って入力されるのを防止できます。

一意キー整合性制約と NOT NULL 整合性制約の組合せ

一意キーと NOT NULL 整合性制約の両方を持つ列が一般的に使用されます。これらの組合せにより、ユーザーは一意キーに必ず値を入力することになり、さらに新しい行データが既存の行データと衝突することもなくなります。

注意： 2 つ以上の列に対する一意制約の検索メカニズムにより、一部が NULL のコンポジット一意キー制約の非 NULL 列で同一の値は許されません。

主キー整合性制約

データベースでは、各表に最大 1 つの主キー制約を定義できます。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 Database では、主キー整合性制約の実装により、次の 2 つのことが保証されます。

- 表の指定された列または列の集合の中に、2 つの行が重複する値を持つことはありません。
- 主キー列では、NULL は許可されません。つまり、それぞれの行の主キー列には値が必ず存在します。

この項の内容は、次のとおりです。

- 主キー
- 主キー制約と索引

主キー

表の主キー整合性制約の定義に含まれる列を、主キーと呼びます。主キーは必須ではありませんが、次の利点を考慮して、すべての表に主キーを指定してください。

- 表のそれぞれの行を一意に識別できます。
- 重複する行が表に存在しません。

主キー制約と索引

Oracle Database では、すべての主キー制約で索引が使用されます。列に主キー制約を作成すると、次の要素が暗黙的に作成されます。

- その列に対して一意の索引
- その列に対する NOT NULL 制約

コンポジット主キー制約は、コンポジット索引に課されているのと同じ 32 列までに制限されています。この索引の名前は、制約の名前と同じ名前です。また、制約の作成に使用する CREATE TABLE 文または ALTER TABLE 文に ENABLE 句を組み込んで、索引の記憶域オプションを指定することもできます。主キー制約の作成時に使用できる索引がある場合、主キー制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

参照整合性制約

リレーショナル・データベースの中の異なる表は共通の列で関連付けることができ、列の関係を管理する規則が守られていることが必要です。参照整合性規則により、これらの関係が保たれることが保証されます。

表 21-1 に、参照整合性制約に関連する用語を示します。

表 21-1 参照整合性制約の用語

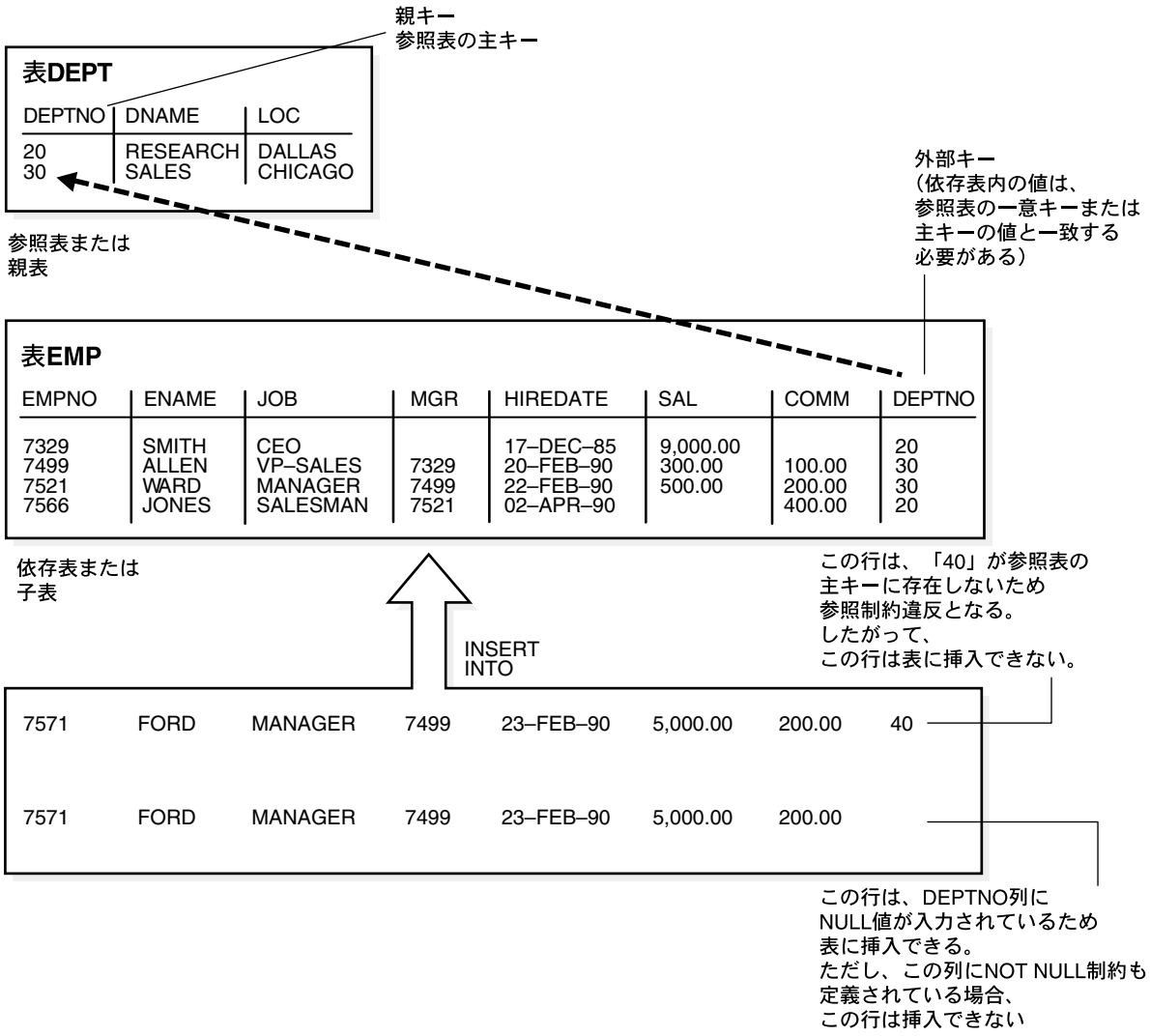
用語	定義
外部キー	参照整合性制約の定義に含まれている列または列の集合のうち、参照キーを参照するもの。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。

参照整合性制約では、表の各行の外部キー値は親表の値と一致している必要があります。

図 21-1 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（つまり deptno 列）の値と一致することが保証されます。このため、emp 表の deptno 列に間違った部門番号が存在することはありません。

外部キーは、複数列として定義できます。ただし、コンポジット外部キーの列数とデータ型は、参照先のコンポジット主キーまたは一意キーと同じであることが必要です。コンポジット主キーおよび一意キーは 32 列までに制限されているため、コンポジット外部キーも 32 列までに制限されます。

図 21-1 参照整合性制約



この項の内容は、次のとおりです。

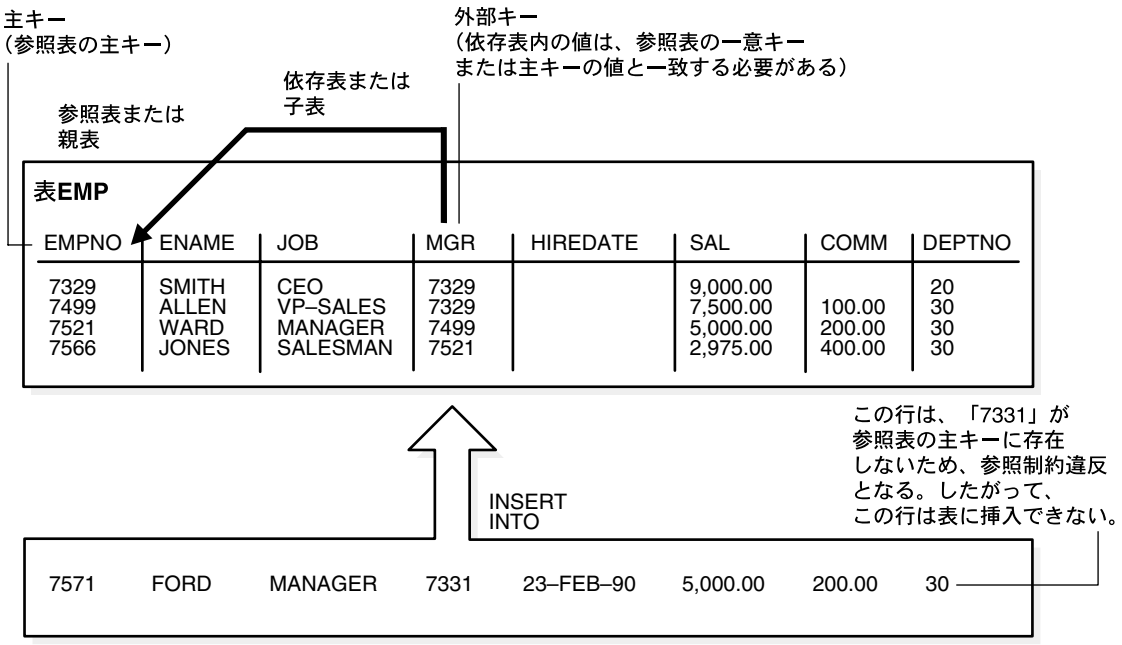
- 自己参照型整合性制約
- NULL と外部キー
- 参照整合性制約によって定義されるアクション
- 同時実行性制御、索引および外部キー

自己参照型整合性制約

図 21-2 に示す別のタイプの参照整合性制約のことを、自己参照型整合性制約と呼びます。このタイプの外部キーは、同じ表の親キーを参照します。

図 21-2 では、参照整合性制約によって、emp 表の mgr 列のすべての値は、同じ行ではなくても、同じ表内の empno 列に現在存在する値と一致していることが保証されます。これは、すべての管理職は従業員である必要もあるためです。この整合性制約により、間違った従業員番号が mgr 列に存在する可能性がなくなります。

図 21-2 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、外部キーの値は、参照先の主キーまたは一意キーの値と一致するか、NULL であることが可能です。コンポジット外部キーのいずれかの列が NULL の場合、そのキーの NULL 以外の部分は、親キーの対応部分と一致している必要はありません。

参照整合性制約によって定義されるアクション

参照整合性制約では、参照先の親キー値が修正された場合に子表の依存行に対して実行される特定のアクションを指定できます。Oracle Database の外部キー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 および DELETE NO ACTION、および DELETE CASCADE です。

注意： Oracle Database の外部キー整合性制約でサポートされていない他の参照アクションは、データベース・トリガーを使用して規定できます。

詳細は、第 22 章「トリガー」を参照してください。

この項の内容は、次のとおりです。

- DELETE NO ACTION

  DELETE CASCADE

  DELETE SET NULL
- 参照アクションに関する DML 制限

DELETE NO ACTION NO ACTION（デフォルト）オプションは、結果のデータが参照整合性制約に違反する場合に参照キー値を更新または削除できないことを指定します。たとえば、主キー値が外部キーの中の値によって参照されている場合は、依存データであるため、参照先の主キー値を削除できません。

DELETE CASCADE 参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行も削除されます。たとえば、親表の行が削除され、この行の主キー値が子表の 1 つ以上の外部キー値によって参照されている場合は、子表の中のこの主キー値を参照する行も子表から削除されます。

DELETE SET NULL 参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行の値が **NULL に設定**されます。たとえば、employee_id が TMP 表内の manager_id 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 manager_id 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 21-2 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 21-2 UPDATE NO ACTION と DELETE NO ACTION で許可される DML 文

DML 文	親表に対して発行	子表に対して発行
INSERT	親キー値が一意であれば常に発行できます。	外部キーの値が親キーに存在するか、外部キーの一部またはすべてが NULL の場合にのみ発行できます。
UPDATE NO ACTION	文の実行後に、参照される親キー値のない行が子表内に残らない場合は発行できます。	文の実行後も新しい外部キー値によって参照キー値が参照される場合は発行できます。
DELETE NO ACTION	子表のどの行も親キー値を参照していない場合は発行できます。	常に発行できます。
DELETE CASCADE	常に発行できます。	常に発行できます。
DELETE SET NULL	常に発行できます。	常に発行できます。

同時実行性制御、索引および外部キー

Oracle Database では、親キーとそれに依存する外部キーとの関連において、同時実行性制御が最適化されます。ロックの動作は、外部キーの列が索引付けされているかどうかに依存します。外部キーが索引付けされていない場合、子表が頻繁にロックされ、デッドロックが発生し、同時実行性が低下する可能性があります。このため外部キーは、ほとんどの場合、索引付けが必要です。唯一の例外は、対応する一意キーまたは主キーの更新や削除が発生しないことが確実な場合です。

この項の内容は、次のとおりです。

- 外部キーの索引がない場合
- 外部キーの索引

外部キーの索引がない場合 次のような場合、データベースは子表に対する表ロックを取得します。

- 子表の外部キー列に索引が存在しない場合。

  たとえば、`hr.departments` 表が、索引付けされていない外部キー `department_id` を含む `hr.employees` の親である場合など。

- セッションで、親表の主キーが変更されるか（行の削除や主キーの属性の変更など）、親表にデータがマージされた場合。親表に対する挿入では、子表の表ロックは取得されません。

  たとえば、図 21-3 に示すように、データベース・セッションで、`departments` 表から行 3 が削除された場合。

図 21-3 外部キーの索引がない場合のロック・メカニズム

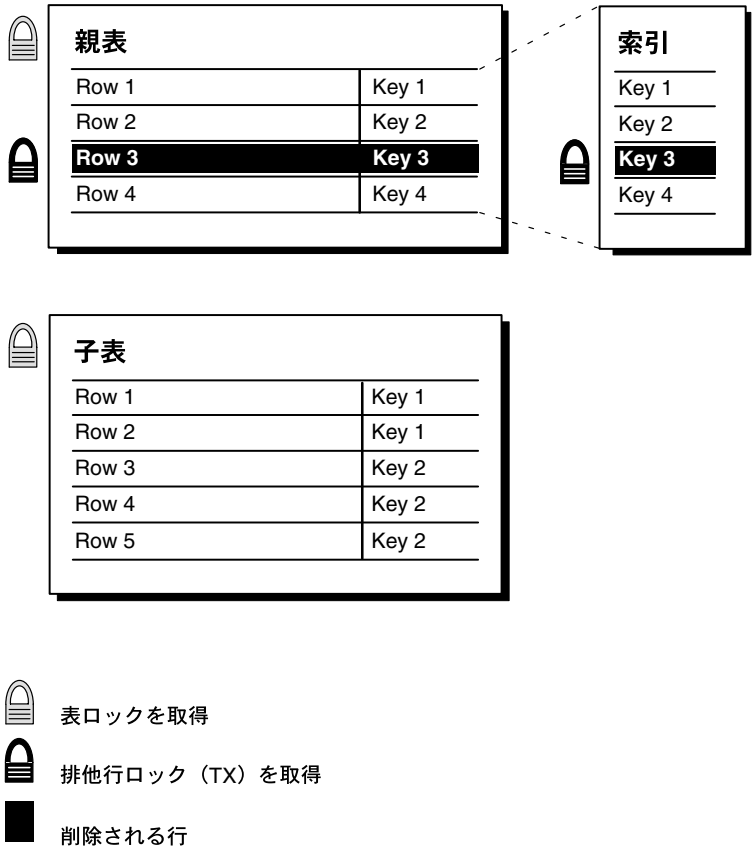


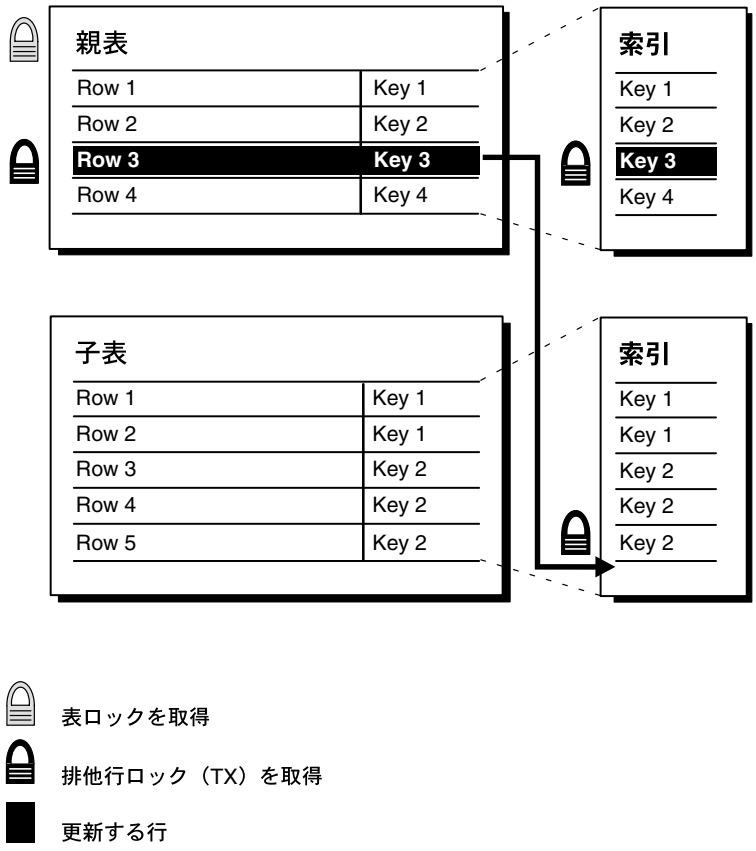
図 21-3 では、子表の外部キー列に索引が作成されていないため、親表の行 3 の削除において子表に対する共有表ロックが取得されます。このロックでは、他のトランザクションは問合せはできますが、表の更新はできません。たとえば、employees の電話番号は、departments 行の削除中に更新できません。表ロックは、departments 表への DML が完了すると、即時に解除されます。複数の行に影響がある場合、ロックは行ごとに取得および解除されます。

注意： 子表に対する DML では、親表の表ロックは取得されません。

外部キーの索引 子表の外部キー列が索引付けされている場合、親表に対する DML では親表の表ロックが取得されます。このロックにより、トランザクションは排他表ロックを取得できませんが、親表に対する DML の発生時に、親表または子表に対する DML は阻止されません。この状況は、子表への更新が発生している間に、親表への更新または削除が発生する場合に適しています。

図 21-4 に、子表の外部キー列が索引付けされている場合の例を示します。親表は departments、子表は employees です。セッションで、departments の行 3 が更新されます。departments に対する DML によって、employees への更新が阻止されることはありませんが、departments の行の更新および削除は、employees の索引に対する行レベルのロックが解除されるまで待機する必要があります。

図 21-4 外部キーが索引付けされている場合のロック・メカニズム



子表で ON DELETE CASCADE を指定すると、親表からの削除によって、子表からも削除することができます。たとえば、departments からレコードを削除した場合、削除された部門の従業員に関するレコードを employees から削除できます。この場合、待機とロックに関するルールは、親表から行を削除した後に子表から行を削除する場合と同じです。

チェック整合性制約

列または列の集合に対するチェック制約では、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 文の結果でチェック制約の条件が FALSE に評価される場合、その文はロールバックされます。

この項の内容は、次のとおりです。

チェック条件

チェック制約を使用すると、チェック条件を指定することによって、特定の目的の整合性規則を規定できます。チェック制約の条件には次のような制限があります。

- 挿入または更新する行の値を使用して評価されるブール式であることが必要です。
- 副問合せ、順序、SQL 関数 SYSDATE、UID、USER または USERENV、あるいは疑似列 LEVEL または ROWNUM を含むことはできません。

文字列リテラルまたはグローバリゼーション・サポート・パラメータを引数に指定した SQL 関数（TO_CHAR、TO_DATE および TO_NUMBER など）が組み込まれているチェック制約を評価する際、デフォルトでは Oracle Database はデータベースのグローバリゼーション・サポート設定を使用します。これらのデフォルトは、チェック制約定義の中でそれらのファンクションにグローバリゼーション・サポート・パラメータを明示的に指定して、オーバーライドできます。

> **関連項目：** グローバリゼーション・サポート機能の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

複数のチェック制約

1 つの列に、定義された列を参照する複数のチェック制約を設定できます。1 つの列に対して定義できるチェック制約の数に制限はありません。

単一の列に対して複数のチェック制約を作成する場合は、その目的が競合しないように慎重に設計してください。また、条件が特定の順序で評価されるとは考えないでください。Oracle Database では、チェックの条件が矛盾しないかどうかは検証されません。

制約チェックのメカニズム

制約が存在する場合に許可されるアクションのタイプを把握する上で、Oracle Database 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。この例では、次のような状況を想定します。

- emp 表は、21-10 ページの図 21-2 で定義されたものです。
- 自己参照型制約によって、mgr 列のエントリが empno 列の値に依存しています。わかりやすくするため、これ以降の説明では、emp 表の empno 列と mgr 列のみを対象にします。

emp 表に最初の行を挿入する場合を考えます。現在は行が存在しません。mgr 列が empno 列の既存の値を参照できない場合に行を入力する方法について考えます。この操作を実行するには、次の 3 つの方法が考えられます。

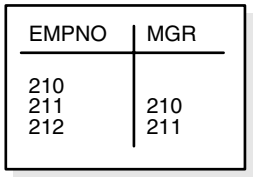
- mgr 列に NOT NULL 制約が定義されていない場合には、第 1 行の mgr 列に NULL を入力できます。外部キーには NULL が許されるため、この行は表に正しく挿入されます。
- empno 列と mgr 列の両方に同じ値を入力できます。このことから、Oracle Database が文の実行完了後に制約チェックを実行することがわかります。親キーと外部キーに同じ値を指定した行の入力を許可するために、Oracle Database は、文を実行（つまり、新しい行を挿入）してから、その新しい行の mgr 列に対応する empno のある行がその表に含まれているかどうかをチェックします。

- SELECT 文のネストを伴う INSERT 文など、複数行を挿入する INSERT 文により、相互に参照しあう行を挿入できます。たとえば、最初の行は empno 列が 200 で mgr 列が 300、2 番目の行は empno 列が 300 で mgr 列が 200 という挿入を実行できます。

  このことから、制約チェックは文の実行が完了するまで遅延されていることがわかります。すべての行が挿入されてから、制約違反がないかどうかすべての行が調べられます。また、**トランザクション**が完了するまで制約のチェックを遅延させることもできます。

同じ自己参照型の整合性制約に関して、次の使用例を考えます。会社が買収された場合です。この買収に伴い、すべての従業員番号の現在の設定値に 5000 を加算して、新しい会社の従業員番号と調和させる必要があります。管理職番号は実際には従業員番号であるため、これらの値にも 5000 を加算する必要があります（図 21-5 を参照）。

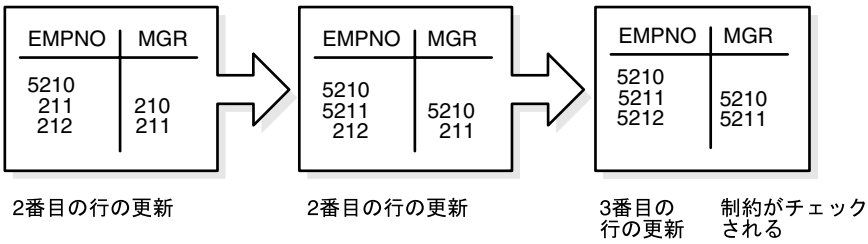
図 21-5 更新前の EMP 表




```
UPDATE employees
  SET employee_id = employee_id + 5000,
      manager_id = manager_id + 5000;
```

制約は、各 mgr 値が empno 値と一致するかどうかを検証するように定義されていますが、Oracle Database では文の完了後に制約チェックが効率的に実行されるため、この文は有効です。図 21-6 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 21-6 制約チェック



この項の例は、INSERT 文と UPDATE 文を実行した場合の制約チェックのメカニズムを示しています。これと同じメカニズムが、UPDATE、INSERT および DELETE 文を含むすべてのタイプの DML 文に使用されます。

また、これらの例は、自己参照型整合性制約を使用してチェックのメカニズムを示すものでした。これと同じメカニズムが、次にあげるすべてのタイプの制約に使用されます。

- NOT NULL
- 一意キー
- 主キー
- すべてのタイプの外部キー制約
- チェック制約

関連項目： 21-16 ページの「遅延制約チェック」

デフォルト列値と整合性制約チェック

デフォルト値は、文の解析前に INSERT 文の一部として組み込まれます。このため、デフォルトの列値はすべての整合性制約チェックの対象になります。

遅延制約チェック

制約の妥当性のチェックは、トランザクション終了時まで**遅延**できます。

- **遅延制約**では、制約が満たされているかどうかのチェックがコミット時にしか実行されません。遅延制約違反があると、コミット時にそのトランザクションは取り消されます。
- **即時制約**の場合（つまり遅延制約でない場合）、チェックはそれぞれの文が終わった時点で実行されます。制約違反があると、その文は即時にロールバックされます。

制約によって**アクション**（DELETE CASCADE など）が発生する場合は、遅延制約か即時制約かに関係なく、そのアクションは、アクションを発生させた文の一部として実行されます。

この項の内容は、次のとおりです。

制約の属性

制約は、**遅延可能**または**遅延不可**、および**初期遅延**または**初期即時**のどちらかに定義できます。これらの属性は、制約ごとに異なるものを指定できます。それらの定義は、CONSTRAINT 句の中で次のキーワードを使用して指定します。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE

制約について、追加、削除、使用可能と使用禁止の切替え、または妥当性チェックを実行できます。また、制約の属性も変更できます。

> **関連項目：** 制約属性とそのデフォルト値の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

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 SQL-92 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継続時間中、または別の SET CONSTRAINTS 文によってモードが再設定されるまで有効です。

SET CONSTRAINTS ... IMMEDIATE は、指定された制約を各制約文の実行直後にチェックするように指示します。チェック制約が一貫しており、他に SET CONSTRAINTS 文が発行されない場合、Oracle Database はまずトランザクション内で事前に遅延されている制約をチェックし、次にそのトランザクション内のそれ以降の文の制約を即時にチェックします。制約チェックが失敗すると、エラーが通知されます。この場合、COMMIT を発行するとトランザクション全体が取り消されます。

ALTER SESSION 文にも、SET CONSTRAINTS IMMEDIATE または DEFERRED の句があります。これらの句では、暗黙的にすべての遅延制約が設定されます（つまり、制約名のリストは指定できません）。これらのオプションを指定することは、カレント・セッションで各トランザクションの最初に SET CONSTRAINTS 文を発行することと同じ意味を持ちます。

COMMIT が成功するかどうかを調べる 1 つの方法は、トランザクションの最後に**即時**制約を設定することです。トランザクションの最後の文で制約を IMMEDIATE に設定すれば、予期しないロールバックを回避できます。いずれかの制約がチェックを通らなかった場合は、エラーを訂正してから、トランザクションをコミットできます。

SET CONSTRAINTS 文は、トリガー内では許可されていません。

SET CONSTRAINTS は分散型の文としても有効です。SET CONSTRAINTS ALL 文が出現すると、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出現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。マテリアライズド・ビューに対して遅延一意制約および遅延外部キー制約を設定すれば、リフレッシュ操作を高速かつ完全に完了できます。

遅延可能な一意制約は、常に一意でない索引を使用します。遅延可能制約を削除しても、その索引は残ります。制約を使用禁止にしても格納情報は残るため、この方法は便利です。遅延可能でない一意制約と主キーは、制約が規定される前に一意でない索引がキー列に置かれる場合には、一意でない索引も使用します。

22

トリガー

この章では、データベース・トリガーについて説明します。データベース・トリガーとは、PL/SQL または Java で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、あるい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。

この章の内容は、次のとおりです。

トリガーの概要

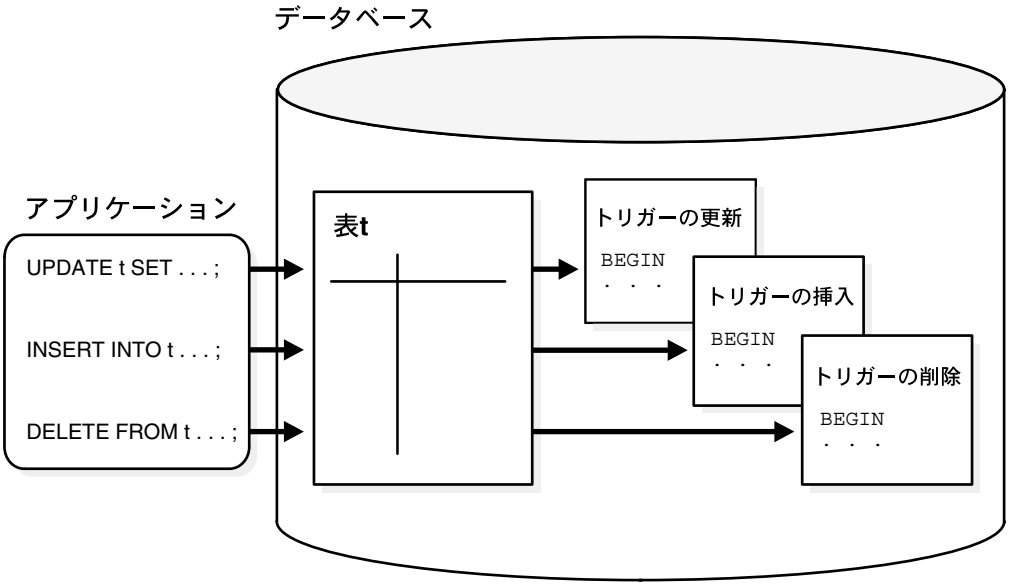
次の操作が発生した場合に起動するトリガーを記述できます。

1. 発行するユーザーを問わず、特定の表またはビューに対する DML 文（INSERT、UPDATE、DELETE）
2. データベース内の特定または任意のスキーマやユーザーが発行する DDL 文（主に CREATE または ALTER）
3. データベース内の特定または任意のスキーマやユーザーが発行する、ログオンやログオフ、エラーまたは起動や停止などのデータベース・イベント

トリガーは、ストアド・プロシージャに似ています。データベースに格納されているトリガーには、1 単位として実行可能な SQL や PL/SQL 文を格納でき、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。トリガーは、接続ユーザーや使用中のアプリケーションに関係なく、トリガー・イベントが発生した時点で Oracle Database によって暗黙的に起動されます。

図 22-1 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 22-1 トリガー



また、トリガーは C プロシージャを起動し、計算集中型の操作に使用できます。

関連項目：

- トリガーとストアド・プロシージャの類似点の詳細は、第 24 章「SQL」を参照してください。
- 22-5 ページの「トリガー・イベントまたはトリガーを実行する文」

この項の内容は、次のとおりです。

- トリガーの使用方法

トリガーの使用方法

トリガーは Oracle Database 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。トリガーには、その他に次のような使用方法があります。

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

関連項目： トリガーの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

この項の内容は、次のとおりです。

トリガーの使用上の注意

トリガーはデータベースのカスタマイズに使用すると便利ですが、必要な場合にのみ使用してください。トリガーを使用しすぎると相互依存性が複雑になるため、大規模アプリケーションのメンテナンスが困難になります。たとえば、トリガーが起動すると、そのトリガー・アクションに含まれる SQL 文によって他のトリガーが起動され、**トリガーが連鎖状態になる**ことがあります。これにより、意図しない影響を生じるおそれがあります。

トリガーと宣言整合性制約との比較

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

- 子表と親表が分散データベースの別々のノードにあるときに、参照整合性を規定する場合
- 整合性制約では定義できない複雑なビジネス・ルールを規定する場合
- 必要な参照整合性規則を、次の整合性制約を使用して規定できない場合
  - `NOT NULL`、一意
  - 主キー
  - 外部キー
  - チェック
  - `DELETE CASCADE`
  - `DELETE SET NULL`

関連項目： 整合性制約の詳細は、21-3 ページの「Oracle Database でデータ整合性を規定する方法」を参照してください。

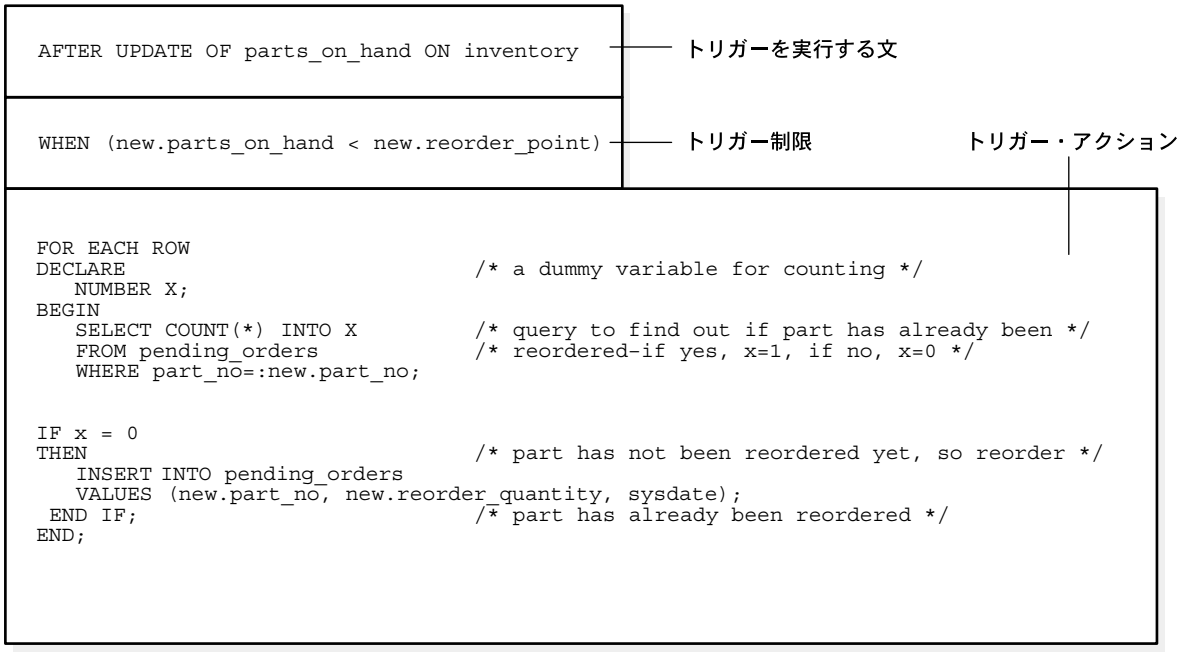
トリガーのコンポーネント

トリガーには次の 3 つの基本部分があります。それぞれについてこの項で説明します。

- トリガー・イベントまたはトリガーを実行する文
- トリガー制限
- トリガー・アクション

図 22-2 に、各トリガー・コンポーネントを示します。この図は、正確な構文を示すためのものではありません。各トリガー・コンポーネントについては、この後で詳しく説明します。

図 22-2 REORDER トリガー



トリガー・イベントまたはトリガーを実行する文

トリガー・イベントまたはトリガーを実行する文とは、トリガーを起動させる SQL 文、データベース・イベントまたはユーザー・イベントのことです。トリガー・イベントは、次に示すものの 1 つ以上からなります。

- 特定の表（または、場合によってはビュー）に対する INSERT、UPDATE、または DELETE 文
- スキーマ・オブジェクトに対する CREATE、ALTER 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、22-4 ページの図 22-2 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、UPDATE または DELETE 文を発行すると、トリガーが起動されます。複数のタイプの SQL 文でトリガーが起動される場合は、条件述語を使用して、トリガーを実行する文のタイプを検出できます。これにより、トリガーを起動した文のタイプに応じて異なるコードを実行するトリガーを作成できます。

トリガー制限

トリガー制限にはブール式を指定します。トリガーが起動されるには、このブール式が true になる必要があります。トリガー制限の評価が false または unknown になった場合、トリガー・アクションは実行されません。この例では、次のようにしてトリガーを制限しています。

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

この場合、引当可能部品の数量が現行の再発注数量より少なくなるまで、トリガーは起動しません。

トリガー・アクション

トリガー・アクションは、SQL 文またはコードを含むプロシージャ（PL/SQL ブロック、Java プログラムまたは C コールアウト）です。これらの SQL 文やコードは、次のイベントが発生すると実行されます。

- トリガーを実行する文が発行されたとき
- トリガー制限が true と評価されたとき

ストアド・プロシージャと同様に、トリガー・アクションでは次のことが可能です。

- SQL 文、PL/SQL 文または Java 文の使用
- 変数、定数、カーソル、例外など、PL/SQL の言語構造の定義
- Java 言語構造の定義
- ストアド・プロシージャの起動

トリガーが行トリガーの場合、トリガー・アクション内の文から、トリガーによって処理されている行の列値にアクセスできます。各列の古い値と新しい値には、相関名によってアクセスできます。

トリガーのタイプ

この項では、様々なタイプのトリガーについて説明します。

- 行トリガーと文トリガー
- BEFORE および AFTER トリガー
- 複合トリガー
- INSTEAD OF トリガー
- システム・イベントとユーザー・イベントのトリガー

行トリガーと文トリガー

トリガーを定義するときに、次のようにトリガー・アクションの実行回数を指定できます。

- 多数の行を更新する UPDATE 文によって起動されるトリガーなどの場合、トリガーを実行する文の影響を受ける行ごとに 1 回ずつ
- 影響を受ける行数に関係なく、トリガーを実行する文ごとに 1 回ずつ

この項の内容は、次のとおりです。

- 行トリガー
- 文トリガー

行トリガー

行トリガーは、表がトリガーを実行する文の処理の影響を受けるたびに実行されます。たとえば、UPDATE 文が表の複数の行を更新した場合、UPDATE 文が処理する行ごとに 1 つずつ行トリガーが起動されます。トリガーを実行する文の影響を受ける行がなければ、行トリガーは実行されません。

トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータまたは影響を受ける行数に依存する場合には、行トリガーが便利です。たとえば、22-4 ページの図 22-2 は、トリガーを実行する文の影響を受ける各行の値を使用する行トリガーの一例です。

文トリガー

文トリガーは、トリガーを実行する文の影響を受ける表の行数とは関係なく、また、影響を受ける行がない場合にも、トリガーを実行する文の実行ごとに 1 回起動されます。たとえば、DELETE 文が表の複数の行を削除すると、文レベルの DELETE トリガーが 1 回のみ起動されます。

トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータや、影響を受ける行に依存しない場合には、文トリガーが便利です。文トリガーの使用例を次に示します。

- 現在時刻やカレント・ユーザーについて複雑なセキュリティ・チェックを実行する場合
- 単一の監査レコードを生成する場合

BEFORE および AFTER トリガー

トリガーを定義するときに、**トリガーのタイミング**を指定できます。これは、トリガー・アクションを、トリガーを実行する文の前に実行するのか、後に実行するのかを指定するものです。BEFORE と AFTER は、文トリガーと行トリガーの両方に適用されます。

DML 文によって起動される BEFORE トリガーと AFTER トリガーは表にのみ定義でき、ビューには定義できません。ただし、ビューに対して INSERT、UPDATE または DELETE 文を発行すると、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、データベースまたはスキーマにのみ定義でき、特定の表には定義できません。

関連項目：

- 22-9 ページの「INSTEAD OF トリガー」
- BEFORE および AFTER トリガーを使用して DML 文と DDL 文に関する情報を発行する方法は、22-10 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。

この項の内容は、次のとおりです。

BEFORE トリガー

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

- トリガー文を実行してよいかどうかを、トリガー・アクションによって決める場合。BEFORE トリガーを使用することにより、トリガー・アクションで例外が発生した場合に、トリガー文で無駄な処理を実行して結果的にロールバックすることになるのを避けられます。
- トリガーを実行する INSERT または UPDATE 文を実行する前に、特定の列値を調べる場合。

AFTER トリガー

AFTER トリガーは、トリガー文の実行後にトリガー・アクションを実行します。

トリガー・タイプの組合せ

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

- **BEFORE 文トリガー**

  トリガー文を実行する前にトリガー・アクションが実行されます。

- **BEFORE 行トリガー**

  トリガーを実行する文の影響を受ける各行が修正され、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かぎりトリガー・アクションが実行されます。

- **AFTER 文トリガー**

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

- **AFTER 行トリガー**

  トリガーを実行する文の影響を受ける各行が修正され、該当する整合性制約が適用された後で、トリガー制限に違反していないかぎ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 行トリガーは、行をロックします。

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、employees 表の UPDATE 文に対して BEFORE 文トリガーを 2 つ指定できます。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 Database マテリアライズド・ビュー・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。

関連項目： トリガー・タイプの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

複合トリガー

複合トリガーは表の単一のトリガーで、次の 4 つのタイミング・ポイントのそれぞれにアクションを指定できます。

- 起動文の前
- 起動文が影響する各行の前
- 起動文が影響する各行の後
- 起動文の後

複合トリガーの本体では、各タイミング・ポイント用のコードによるアクセスが可能な PL/SQL の共通の状態がサポートされています。起動文が完了すると、その起動文がエラーの原因になった場合でも、共通の状態は自動的に破棄されます。

複合トリガーの効果は、各タイミング・ポイントに対する単一のトリガーを使用して実現できる効果に似ています（各タイミング・ポイントには、これらの単一のトリガーが共有する状態を保持する補助パッケージを使用して、アクションをコード化する必要があります）。複合トリガーの明らかな長所は、必要なコードが単一のコンパイル・ユニットの中で管理されることですが、さらに重要な長所は、複合トリガーの状態の存続期間が自動的に起動文の継続時間中に制限されるということです。

複合トリガーは、各行の変更を明らかにし、文の後のタイミングで本体として変更に作用するファクトを累積する場合に役立ちます。（テーブルの変更エラーを避けるために）この方法を強制される場合もあります。たとえば、ディテール表での変更に応じて、正規化されていない集計値をマスター表の中で更新する場合、または監査表を更新する場合などは、この方法のパフォーマンスのほうが優れている場合があります。

関連項目：『Oracle Database PL/SQL 言語リファレンス』

INSTEAD OF トリガー

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

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

この項の内容は、次のとおりです。

ビューの変更

ビューを変更する操作では、次のように曖昧な結果を生じることがあります。

- ビューで行を削除する操作は、実表から行を実際に削除する操作と、値をいくつか更新してその行がビューに選択されないようにする操作のどちらかを意味します。
- ビューに行を挿入する操作は、実表に新しい行を実際に挿入する操作と、既存の行を更新してビューに表示されるようにする操作のどちらかを意味します。
- 結合が含まれるビューで列を更新すると、ビューに表示されない別の列の意味が変わってしまうことがあります。

オブジェクト・ビューには、その他にも問題があります。たとえば、オブジェクト・ビューの主な使用方法は、マスターとディテールの関連を表すことです。この操作で結合が関係してくるのは避けられませんが、結合の変更は本質的に曖昧です。

その結果、変更可能なビューについては、たくさんの制限事項があります。INSTEAD OF トリガーは、それ以外の手段では変更できないリレーショナル・ビューのみでなく、オブジェクト・ビューに対しても使用できます。

INSTEAD OF トリガーを使用せずにデータを挿入、更新または削除でき、かつ後述の条件に一致する場合、このビューは**本質的に変更可能**です。ビューが本来は変更可能であっても、挿入、更新または削除する値の妥当性検査を実行する必要があります。この場合にも、INSTEAD OF トリガーを使用できます。変更される行の妥当性チェックがトリガー・コードによって実行され、有効であれば、基礎となる表に変更が伝播されます。

INSTEAD OF トリガーにより、OCI を使用してクライアント側のオブジェクト・ビューのインスタンスを変更することもできます。オブジェクト・ビューによって具体化されたオブジェクトを、クライアント側のオブジェクト・キャッシュ内で変更し、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場合は、INSTEAD OF トリガーを指定する必要があります。ただし、単にオブジェクト・キャッシュ内のビュー・オブジェクトを確保して読み取る場合、これらのトリガーを定義する必要はありません。

関連項目：

- 『Oracle Database オブジェクト・リレーショナル開発者ガイド』
- 『Oracle Call Interface プログラマーズ・ガイド』
- INSTEAD OF トリガーの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

変更不可能なビュー

ビュー問合せに次の構成メンバーが含まれている場合、そのビューは本質的に変更不可能であるため、そのビューに対しては挿入、更新または削除を実行できません。

- 集合演算子
- 集計関数
- GROUP BY、CONNECT BY、または START WITH 句
- DISTINCT 演算子
- 結合（ただし、一部の結合ビューは更新可能）

ビューに疑似列または式が含まれる場合、そのビューを更新する唯一の方法は、その疑似列または式を参照しない UPDATE 文を使用することです。

関連項目： 5-16 ページの「更新可能な結合ビュー」

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の列の要素は、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の列に INSTEAD OF トリガーを定義すると、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。

関連項目：

- ネストした表のトリガーの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- CREATE TRIGGER 文の詳細は、『Oracle Database PL/SQL 言語リファレンス』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
  - Data Guard ロール遷移
  - 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。DBMS_AQ パッケージは、特定のアクションを実行するデータベース・トリガーの使用例です。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。

この項の内容は、次のとおりです。

イベントの発行

イベントの発行では、Oracle Streams Advanced Queuing のパブリッシュ / サブスクライブ・メカニズムが使用されます。**キュー**は、各種サブスクライバに必要な主題を含むメッセージ・リポジトリの役割を果します。トリガーは、特定のシステム・イベントまたはユーザー・イベントの発生時に、DBMS_AQ パッケージを使用してメッセージをエンキューします。

関連項目：

- パブリッシュ / サブスクライブの Oracle Streams Advanced Queuing 実装の詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』を参照してください。
- DBMS_AQ パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベントの属性

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

システム・イベント

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

- STARTUP トリガーは、インスタンスによってデータベースがオープンされるときに起動します。この属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- SHUTDOWN トリガーは、サーバーがインスタンスの停止操作を開始する直前に起動します。このトリガーを使用して、データベースの停止時にサブスクライバ・アプリケーションを完全に停止できます。インスタンスが異常停止する場合、このトリガーは起動しません。SHUTDOWN トリガーの属性には、システム・イベント、インスタンス番号およびデータベース名があります。
- SERVERERROR トリガーは、特定のエラーの発生時、または、エラー番号を指定しなければ任意のエラーの発生時に起動します。このトリガーの属性には、システム・イベントとエラー番号があります。
- DB_ROLE_CHANGE トリガーは、Data Guard 構成でロール遷移（フェイルオーバーまたはスイッチオーバー）が発生したときに起動します。このトリガーでは、ロール遷移発生時にユーザーに通知されるため、新規プライマリ・データベースでクライアント接続を処理でき、アプリケーションは引き続き動作できます。

ユーザー・イベント

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

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

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

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

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

関連項目：

- 22-6 ページの「行トリガー」
- 22-7 ページの「BEFORE および AFTER トリガー」
- システム・イベントとユーザー・イベントのトリガーを使用したイベント発行の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーの実行

トリガーは使用可能または使用禁止のいずれかです。

表 22-1 トリガーのモード

トリガーのモード	定義
使用可能	使用可能にされているトリガーは、トリガーを実行する文が発行され、トリガー条件（指定されている場合）の評価が TRUE である場合に、そのトリガー・アクションを実行します。
使用禁止	使用禁止にされているトリガーは、トリガーを実行する文が発行され、トリガー条件（指定されている場合）の評価が TRUE であっても、そのトリガー・アクションを実行しません。

使用可能なトリガーの場合、Oracle Database は次の処理を自動的に行います。

- Oracle Database は、1 つの SQL 文により複数のトリガーが起動される場合は、指定どおりの起動順序で各トリガーを実行します。最初に文レベル、次に行レベルのトリガーを起動します。
- Oracle Database が様々なタイプのトリガーに対して、指定された時点で整合性制約のチェックを実行し、整合性制約の違反を防止します。
- Oracle Database は問合せと制約に対して、読取り一貫性ビューを提供します。
- Oracle Database はトリガー・アクションのコードで参照されるトリガーとスキーマ・オブジェクトの間の依存性を管理します。
- Oracle Database はトリガーが分散データベースのリモート表を更新する場合には、2 フェーズ・コミットを使用します。
- Oracle Database は特定の 1 つの文について同じタイプのトリガーが複数存在する場合、順序不定で各トリガーを起動します。つまり、同じ文に対する同じタイプのトリガーが、特定の順序で起動されるとはかぎりません。

この項の内容は、次のとおりです。

トリガーの実行モデルと整合性制約のチェック

トリガーの本体にある文によって、他のトリガーが起動される場合、そのトリガーは**連鎖的**と呼びます。Oracle Database では、最高 32 個のトリガーが同時に連鎖できます。

リレーショナル・データベースでは、SQL 文によって処理される行の順序が保証されません。そのため、行の処理の順序に依存するトリガーは作成しないでください。

> **関連項目：** トリガーの作成およびそれらが起動される順序の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

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取り（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ランザクションによって変更されたものである場合、起動されたトリガーの本体にある SQL 文は、次のガイドラインに従います。

- 問合せは、参照先の表の読取り一貫性のある現行のマテリアライズド・ビューと、同一トランザクション内で変更されたデータを参照します。
- 更新は、既存のデータのロックが解除されるまで待機してから、処理を続行します。

PL/SQL トリガーの記憶域

PL/SQL トリガーは、ストアド・プロシージャと同じように、コンパイル済の形式で Oracle Database に格納されます。`CREATE TRIGGER` 文がコミットされると、コンパイル済 PL/SQL コードがデータベースに格納され、トリガーのソース・コードは共有プールからフラッシュされます。

関連項目： トリガーのコンパイルおよび保存の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーの実行

Oracle Database は、サブプログラムの実行と同じ手順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、ユーザーがトリガー文を実行する権限を持っていれば、トリガーを起動する権限が付与されることです。この違いを除き、トリガーはストアド・サブプログラムと同じ方法でチェックされ、実行されます。

関連項目： ストアド・サブプログラムの詳細は、『Oracle Database PL/SQL 言語リファレンス』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 Database 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・プロシージャと同様に扱われます。これらは、データ・ディクショナリに挿入されます。

関連項目： 第 6 章「スキーマ・オブジェクトの依存性」

23

情報の統合

この章の内容は、次のとおりです。

> **関連項目：**『Oracle Database 2 日でデータ・レプリケーションおよび統合ガイド』

Oracle における情報統合の概要

企業が成長するにつれて、複数のデータベースおよびアプリケーション間で情報を共有できることに対する重要性が高まってきます。顧客はオンラインまたは営業担当を介して発注するため、企業は OLTP の更新、データベース・イベントおよびアプリケーション・メッセージをパートナとも共有する必要があります。この情報は、異機種間で複製されたデータベース、メッセージ・キューイング・システム、データ・ウェアハウスのステージング領域、オペレーショナル・データ・ストア、その他のアプリケーションおよびスタンバイ・データベースなど、様々な宛先にルーティングする必要があります。

情報を共有するには、次の 3 つの基本的な方法があります。情報を 1 つのデータベースに連結できます。これによって、それ以上の統合は必要なくなります。情報は分散した状態にしておき、それらの情報を統合するツールを提供できます。これによって、情報が 1 つの仮想データベース内にあるかのように見せることができます。あるいは、情報の共有もできます。これによって、複数のデータ・ストアおよびアプリケーションで情報を保持できます。この章では、情報の統合と共有を中心に説明します。

> **関連項目：** 情報を連結する機能の詳細は、第 16 章「ビジネス・インテリジェンス」を参照してください。

Oracle には、分散した情報を統合するために、分散 SQL が用意されています。分散 SQL を使用すると、位置の透過性とデータ整合性を保ちながら、複数のデータベース間に分散されたデータに同時にアクセスして更新できます。

Oracle Streams は、Oracle Database における非同期の情報共有インフラストラクチャです。Oracle Streams は、Oracle Database の REDO ログをマイニングしてデータへのデータ操作言語（DML）変更とデータ定義言語（DDL）変更を取得し、変更されたデータを他のアプリケーションやデータベースで使用できるようにします。このように、Oracle Streams は、きわめて柔軟な非同期レプリケーション・ソリューションとイベント通知フレームワークを提供します。Oracle Streams では、アプリケーションによるメッセージの明示的なエンキューとデキューをサポートしているため、完全な非同期メッセージ・ソリューションも提供します。このソリューションである Oracle Streams Advanced Queuing を使用すると、顧客、パートナおよび仕入先との情報交換と、ビジネス・プロセスの調整が可能になります。

Oracle Streams と分散 SQL は、どちらも Oracle Database Gateway、Generic Connectivity、および Messaging Gateway を使用して、Oracle 以外のシステム内のデータにアクセスして更新します。Oracle Database は、Oracle 以外のデータ・ソース、Oracle 以外のメッセージ・キューイング・システム、および非 SQL アプリケーションで動作でき、他のベンダーの製品やテクノロジとの相互運用性が保証されます。ここでは、各ソリューションについて詳しく説明します。

分散環境とは、相互にシームレスに通信する様々なシステムで構成されるネットワークです。分散環境では、各システムをノードと呼びます。ユーザーが直接接続するシステムをローカル・システムと呼びます。このユーザーがアクセスする他のシステムをリモート・システムと呼びます。分散環境では、アプリケーションからローカル・システムやリモート・システムのデータにアクセスして相互に交換できます。すべてのデータに同時にアクセスして変更できます。

分散環境では、ネットワークを介した大量のデータに対するアクセスを増加させることができますが、その一方で、データの位置とネットワークにまたがるアクセスの複雑性は隠す必要があります。

企業が分散環境で正常に運用するには、次の操作が必要です。

- Oracle Database 間でのデータ交換
- アプリケーション間での通信
- 顧客、パートナおよび仕入先との情報交換
- データベース間でのデータのレプリケーション
- Oracle 以外のデータベースとの通信

統合されたアクセス

同一機種分散データベース・システムとは、1 つ以上のコンピュータに常駐する複数の Oracle Database で構成されるネットワークです。

この項の内容は、次のとおりです。

分散 SQL

分散 SQL を使用すると、アプリケーションとユーザーは、1 つのデータベースにアクセスまたは変更する場合と同じくらい容易に、複数のデータベースにあるデータを同時にアクセスまたは変更できます。

Oracle 分散データベース・システムはユーザーに対して透過的であるため、単一の Oracle Database であるかのように見せることができます。企業は、この分散 SQL 機能を使用して、すべての Oracle Database を 1 つであるかのように見せることで、分散システムの持つ複雑さを軽減できます。

Oracle Database は、データベース・リンクを使用して、あるデータベース上のユーザーがリモート・データベース内のオブジェクトにアクセスできるようにします。ローカル・ユーザーは、リモート・データベースのユーザーでなくても、リモート・データベースへのリンクにアクセスできます。

> **関連項目：** データベース・リンクの詳細は、『Oracle Database 管理者ガイド』を参照してください。

位置の透過性

Oracle 分散データベース・システムを使用すると、アプリケーション開発者と管理者は、データベース・オブジェクトの物理位置をアプリケーションとユーザーから隠すことができます。位置の透過性が存在するのは、ユーザーがアプリケーションの接続先ノードに関係なく表などのデータベース・オブジェクトを全体的に参照できる場合です。位置の透過性には、次のような複数の利点があります。

- データベース・ユーザーがデータベース・オブジェクトの物理位置を知る必要がないため、リモート・データへのアクセスが単純です。
- 管理者は、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ずに、データベース・オブジェクトを移動できます。通常、管理者と開発者はシノニムを使用して、アプリケーション・スキーマ内の表およびサポート・オブジェクトの位置の透過性を確立します。

開発者の場合は、シノニムの他にもビューとストアド・プロシージャを使用して、分散データベース・システムで動作するアプリケーションの位置の透過性を確立できます。

> **関連項目：**
>
> - シノニムとビューの詳細は、第 5 章「スキーマ・オブジェクト」を参照してください。
> - ストアド・プロシージャの詳細は、第 24 章「SQL」を参照してください。

SQL と COMMIT の透過性

Oracle の分散データベース・アーキテクチャも、問合せ、更新およびトランザクションの透過性を提供します。たとえば、SELECT、INSERT、UPDATE および DELETE などの標準 SQL 文の動作は、非分散データベース環境の場合と同じです。また、アプリケーションでは標準 SQL 文 COMMIT、SAVEPOINT および ROLLBACK を使用してトランザクションを制御します。

ローカル・データベースでのトランザクションとは異なり、分散トランザクションでは複数データベース上にあるデータを変更する必要があります。その結果、Oracle Database ではトランザクションでの変更のコミットや取消しを自己完結型ユニットとして処理する必要があるため、分散トランザクション処理はより複雑になります。つまり、トランザクション全体がコミットされるか、トランザクション全体がロールバックされます。

Oracle Database では、分散トランザクションのデータ整合性が 2 フェーズ・コミット・メカニズムを使用して保証されます。準備フェーズでは、トランザクション内の開始ノードが他の参加ノードに対して、トランザクションをコミットまたは取り消すように指定します。コミット・フェーズでは、開始ノードがすべての参加ノードに対して、トランザクションのコミットを要求します。この要求を満たせない場合は、すべてのノードがトランザクションを取り消します。2 フェーズ・コミット・メカニズムは完全に透過的であるため、分散トランザクション制御のための複雑なプログラミングや他の特別な操作を必要としません。

関連項目： 4-8 ページの「2 フェーズ・コミット・メカニズム」

分散問合せの最適化

分散問合せの最適化により、分散 SQL 文で参照されているリモート表からトランザクションでデータを取り出すときに、サイト間で必要なデータ転送量が減少します。分散問合せの最適化では、Oracle Database 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（または場合によってはローカル・サイト）で処理し、結果を最終処理のためにローカル・サイトに送信します。

この操作により、すべての表データをローカル・サイトに転送して処理する場合に比べて、同じ所要時間に必要なデータ転送量が減少します。DRIVING_SITE、NO_MERGE、INDEX など、各種のオプティマイザ・ヒントを使用すると、Oracle Database のデータ処理の実行される場所とデータへのアクセス方法を制御できます。

関連項目： オプティマイザとヒントの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

情報の共有

どのような統合であっても、その中心は社内の様々なアプリケーション間でデータを共有することです。**レプリケーション**とは、データベース・オブジェクトを複数のデータベース内で維持することです。この機能は、多数の企業が直面しているスケーラビリティ、可用性およびパフォーマンス上の問題に対するソリューションを提供します。たとえば、レプリケーションにより企業の Web サイトのパフォーマンスを改善できます。在庫表など、ローカル・ユーザーから頻繁に問合せがあるリモート表をローカルにレプリケートすることにより、ネットワーク上で送信されるデータの量が大幅に減少します。ローカル・ユーザーを中央の単一コピーのかわりに複数のローカル・コピーにアクセスさせることにより、分散データベースでネットワークを介して繰り返し情報を送信する必要がなくなるため、データベース・アプリケーションのパフォーマンスを最大化する上で役立ちます。Oracle Streams には強力なレプリケーション機能が用意されており、この機能を使用して分散オブジェクトの複数コピーの同期を維持できます。

多くの企業は、ビジネス・プロセスを自動化して業務を管理するために、様々な自律型分散アプリケーションを開発しています。ただし、これらのアプリケーションは相互に通信して、ビジネス・プロセスおよびタスクを一貫性のある方法で調整する必要があります。また、トレース可能なイベント履歴を保持しながら、インターネットなどの低コスト・チャネルを通じて顧客、パートナおよび仕入先と情報を効率的に交換する必要もあります。これは、過去のものとなった書類による情報交換では満たされていた要件です。

疎結合されたアプリケーションの場合、柔軟な Oracle Streams インフラストラクチャの最上位に構築された Oracle Streams Advanced Queuing を使用できます。Oracle Streams Advanced Queuing は、イベント処理用の統一フレームワークを提供します。

アプリケーションやワークフローで生成されたイベント、または REDO ログやデータベース・トリガーから暗黙的に取得されたイベントを、1 つのキューで取得およびステージングできます。これらのイベントは、様々な方法で使用できます。ユーザー定義ファンクションまたはデータベース表の操作で自動的に適用するか、明示的にデキューできます。使用側アプリケーションに通知を送信することもできます。これらのイベントは、どの段階でも変換できます。使用側アプリケーションが異なるデータベースにある場合、イベントは適切なデータベースに自動的に伝播します。このようなイベントに対する操作を自動的に監査でき、ユーザー指定の期間のみ履歴を保持できます。

この項の内容は、次のとおりです。

- Oracle Streams
- マテリアライズド・ビュー

Oracle Streams

Oracle Streams を使用すると、データベース内またはデータベース間で、データ・ストリーム内のデータ、トランザクションおよびイベントを伝播させ、管理できます。ストリームにより、パブリッシュされた情報がサブスクライブした宛先にルーティングされます。変更を必要とするユーザーは、Oracle Streams の新機能を実装することで、既存の機能を犠牲にする必要がありません。

Oracle Streams では、ストリームに入れる情報、ノード間でのストリーム・フローまたはルーティング、各ノードに入ったときにストリーム内でイベントに発生する処理およびストリームの終了方法を制御できます。ストリームで動作する要素の構成を指定することで、メッセージ・キューイングやデータ・レプリケーションなど、特定の要件に対処できます。

Oracle Streams は、様々な使用例における情報共有の要件を満たしています。Oracle Streams Advanced Queuing には、データベース統合メッセージ・キューイング機能やイベント管理機能が用意されています。また、Oracle には、Oracle Streams を使用したイベント通知、レプリケーションおよびデータ・ウェアハウス・ロード・ソリューションを作成するためのツールが組み込まれています。

Oracle Streams の全機能を利用して、複数のユースケースにまたがる構成を作成し、新しいクラスのアプリケーションを有効化できます。ほとんどのデプロイメントとその関連メタデータには互換性があります。たとえば、データ・ウェアハウスをロードするように構成されたシステムを簡単に拡張して、双方向のレプリケーションを使用可能にできます。完全な再構成は必要としません。

この項の内容は、次のとおりです。

- Oracle Streams アーキテクチャ
- Oracle Streams によるレプリケーション
- Oracle Streams アドバンスト・キューイング
- データベース変更通知
- チェンジ・データ・キャプチャ
- 異機種間環境
- Oracle Streams のユースケース

関連項目：『Oracle Streams 概要および管理』

Oracle Streams アーキテクチャ

Oracle Streams のアーキテクチャはきわめて柔軟です。図 23-1 に示すように、Oracle Streams には次の 3 つの基本要素があります。

- 取得
- ステージング
- 使用

図 23-1 Oracle Streams の情報フロー

取得 Oracle Streams は、データベース変更を暗黙的および明示的に取得し、その変更をステージング領域に置きます。DML 変更や DDL 変更などのデータベース変更は、暗黙的に取得できます。ルールによって、取得される変更が判断されます。取得された変更を表す情報は、論理変更レコード（LCR）として書式設定され、ステージング領域に置かれます。

Oracle Streams には、暗黙的な取得を行うために、取得プロセスおよび同期取得という 2 つのコンポーネントがあります。取得プロセスは REDO ログをマイニングして、データベース変更を検出します。取得プロセスでは、アーカイブ・ログ・ファイルの他に、オンライン REDO ログのマイニングもサポートされています。オンライン REDO ログのマイニングの場合、変更データの REDO 情報は書込みと同時にマイニングされ、取得の待機時間を短縮します。同期取得では、指定されたデータベース・オブジェクトに DML 変更が行われると、その変更は内部メカニズムを使用して取得されます。

通常、取得プロセスは、大量のデータベース・オブジェクトに対する変更、および DML と DDL 変更の両方の取得に使用されます。同期取得は、少量のデータベース・オブジェクトに対する DML 変更の取得に使用されます。

Oracle Streams では、明示的取得もサポートされています。ユーザー・アプリケーションでは、イベントを表すメッセージをステージング領域に明示的にエンキューできます。これらのメッセージは、適用プロセスで使用できるように LCR として書式設定するか、他のユーザー・アプリケーションが明示的なデキューにより使用できるように書式設定できます。

ステージング 取得されたメッセージは、ステージング領域に置かれます。ステージング領域とは、取得されたメッセージを格納して管理するキューです。LCR および他のタイプのメッセージは、サブスクライバが使用するまでステージング領域に格納されます。LCR のステージングにより、LCR データの監査と追跡のみでなく保持のための、セキュリティ機能を備えた領域が提供されます。

サブスクライバは、ステージング領域の内容を検査して、特定のイベントを表すメッセージが必要かどうかを判断します。この場合のサブスクライバは、ユーザー・アプリケーション、通常は他のシステム上にある別のステージング領域への伝播、または適用プロセスです。サブスクライバは必要に応じて一連のルールを評価し、メッセージがサブスクリプションに規定されている基準を満たしているかどうかを判断できます。基準を満たしているメッセージは、サブスクライバにより使用されます。

サブスクライバがユーザー・アプリケーションの場合、そのアプリケーションはメッセージを使用するためにステージング領域からデキューします。サブスクライバが別のステージング領域への伝播の場合、メッセージはそのステージング領域に伝播します。サブスクライバが適用プロセスの場合は、適用プロセスによりデキューされて使用されます。

ステージング領域に置かれたメッセージは、必要に応じて同じデータベースの他のステージング領域またはリモート・データベースのステージング領域に伝播されます。ネットワーク・ルーティングを簡素化してネットワーク通信量を削減するために、メッセージをすべてのデータベースとアプリケーションに送信する必要はありません。サブスクライブするシステムに到達するまで、1 つ以上のシステムのステージング領域を経由して送ることができます。すべてのシステムがメッセージをサブスクライブする必要はないため、特定のシステムで適用される

メッセージについて柔軟性が得られます。1 つのステージング領域が複数のデータベースからのメッセージをステージングして、セットアップと構成を簡素化できます。

メッセージがステージング領域に入るか、伝播されるか、ステージング領域を出た後に、それを変換できます。変換とは、保持しているデータの取得および適用または変更に参加するオブジェクトの形式を変更することです。変換には、特定のデータベースにある表の特定列のデータ型表現を変更すること、1 つのデータベースにある表にのみ列を追加すること、または特定のデータベースにある表にデータのサブセットを組み込むことなどがあります。

使用 ステージング領域のメッセージは、暗黙的または明示的に使用されます。適用プロセスは、メッセージにカプセル化されているデータベース変更を暗黙的にデータベースに適用します。Oracle Streams の適用プロセスには柔軟性があります。メッセージの標準またはカスタムの適用が可能です。カスタムの適用により、適用時にデータの操作や他のアクションを実行できます。明示的デキューがサポートされるため、アプリケーション開発者は Oracle Streams を使用してメッセージを確実に交換できます。また、Oracle Streams の変更取得および伝播機能を活用することで、データ変更をアプリケーションに通知できます。

Oracle Streams によるレプリケーション

Oracle Streams は、該当する情報を自動的に判別し、その情報を必要としているユーザーと共有するための情報共有テクノロジです。このアクティブな情報共有には、DML 変更や DDL 変更がカプセル化されているメッセージを含むデータベース内でのメッセージの取得と管理、および他のデータベースやアプリケーションへのメッセージの伝播が含まれます。データ変更には、レプリカ・データベースに直接適用する方法と、ユーザー定義のサブプログラムをコールして宛先データベースで代替処理を実行する方法があります。たとえば、そのようなサブプログラムでは、データ・ウェアハウスのロードに使用したステージング表を移入できます。

この項の内容は、次のとおりです。

- DML 変更および DDL 変更の取得
- 有向ネットワークでの変更の伝播
- 競合の解消と変更の適用

関連項目：『Oracle Streams レプリケーション管理者ガイド』

DML 変更および DDL 変更の取得 Oracle Streams のレプリケーション構成は、レプリケートするオブジェクトまたはオブジェクトの集合を指定することから始まります。Oracle Streams の暗黙的な取得メカニズムのどちらか一方または両方を使用すると、指定したオブジェクトに対する変更を効率的に取得し、元のシステムにはほとんど影響を与えずに 1 つ以上のリモート・システムにレプリケートできます。取得プロセスを使用すると、データ変更（DML）と構造変更（DDL）の両方を REDO ログから抽出できます。同期取得を使用した場合には、DML 変更が行われるとそれらの変更を取得する内部メカニズムが使用されます。取得された変更は、ステージング領域にパブリッシュされます。

取得プロセスを使用したログベースの取得では、表に対する変更が REDO ログに記録されるという事実を利用して、誤動作やメディア障害が発生した場合にもリカバリ可能性が保証されます。変更を REDO ログから直接取得することにより、システムのオーバーヘッドが最小限で済みます。Oracle Database は、データベース上でのアクティビティ履歴情報を含む REDO 情報を読取り、分析および解析できます。また、Oracle Streams では情報をマイニングして変更データを取得プロセスに配信できます。

同期取得を使用した変更の取得は、DML 変更が比較的少数のデータベース・オブジェクトにレプリケートされる環境に最適です。同期取得では内部メカニズムを使用して、行われた DML 変更が取得されたことを確認します。

Oracle Streams テクノロジを利用するレプリケート・データベースは、同一である必要はありません。その一部となっているデータベースは、Oracle Streams を使用して異なるデータ構造を維持し、データを適切なフォーマットに変換できます。Oracle Streams には、変更の取得時、他のデータベースへの伝播中または宛先サイトでの適用中など、複数の時点でストリームを変換する機能が用意されています。これらの変換機能は、Oracle Streams フレームワークに登録されているユーザー定義ファンクションです。この変換機能は、たとえば、表の特定列のデータ型表現を変更したり、表の列名や表名を変更する際に使用できます。

各サイトにあるデータを内容に基づいてサブセット化することもできます。たとえば、レプリカでは、部門 ID 列に基づいて特定の部門に所属する従業員のみが表に含まれることを指定するルールを使用できます。Oracle Streams では変更が自動的に管理され、レプリカ内のデータとサブセット・ルールの基準が確実に一致することになります。

有向ネットワークでの変更の伝播 ステージング領域にあるメッセージは、他のデータベースのステージング領域に送信できます。Oracle Streams の有向ネットワーク機能により、パススルーとして中間データベースを介して変更を送ることができます。どのデータベースでの変更も、パブリッシュし、ネットワーク上のあらゆる場所に直接または他のデータベースを介して伝播できます。ステージング領域にあるキューのルールベースのパブリッシュ・サブスクライブ機能を使用すると、データベース管理者は各宛先データベースに伝播する変更を選択し、メッセージが宛先へと横断するルーティングを指定できます。

このため、たとえば、会社が特定のスキーマに対する変更をすべて取得し、ロンドンにあるヨーロッパ本社にはヨーロッパの顧客に対する変更のみを伝播し、ロンドン支社に関連する変更のみを適用して各現地支社で適用するサイト固有の情報は転送するように、レプリケーションを構成できます。

この有向ネットワーク・アプローチは Wide Area Network（WAN）でも一般的であり、以降の宛先に対する変更は、それぞれの宛先に直接送信するのではなく、単一サイトへとネットワークを横断させてから、後で他の宛先へと展開させることができます。

競合の解消と変更の適用 ステージング領域に置かれたメッセージは適用プロセスで使用され、そこでメッセージが表す変更がデータベース・オブジェクトに適用されるか、アプリケーションで使用されます。ユーザー定義の適用プロシージャにより、適用するメッセージ全体を制御できます。

カスタム適用機能を使用する場合は、表に対する各種 DML 操作（INSERT、UPDATE または DELETE）を処理するために個別のプロシージャを定義できます。たとえば、このカスタム適用機能を使用すると、`salary` 表内の従業員の値に基づいて、給与が $100,000 を超えている従業員については、`employees` 表に対する削除の適用をすべてスキップするプロシージャを記述できます。給与が $100,000 未満の従業員の場合の削除と同様に、`employees` 表に対する挿入と更新はデフォルトの適用エンジンを使用して引き続き適用されます。

カスタム適用機能を使用して、データのカスタム変換を実行することもできます。たとえば、元のサイトで行われた 1 つの表に対する変更を、リモート・ロケーションにある 3 つの異なる表に適用する必要が生じることがあります。

レプリケーション環境でのリモート・データベースは読取り / 書込み用に完全にオープンにできるため、ソース・データベースの同一コピーである必要はありません。リモート・データベースが他の手段で更新されることもあるため、適用プロセスでは変更の適用前に競合が検出されます。このような競合は、組込みまたはカスタムの解消メカニズムを使用して自動的に解消することもできます。

Oracle Streams アドバンスト・キューイング

Oracle Streams Advanced Queuing には、データベースの統合のみでなく、最も堅牢で豊富な機能を備えたメッセージ・キューイング・システムに必要な多数の機能が用意されています。これらの機能により開発者の生産性が改善され、管理者にとっての運用上の負荷が軽減され、Oracle ベースの分散アプリケーションの構築および維持管理コストが削減されます。次の各項では、これらの機能について説明します。

この項の内容は、次のとおりです。

- 非同期によるアプリケーションの統合
- 拡張可能な統合アーキテクチャ
- 異機種間アプリケーションの統合
- 従来型アプリケーションの統合
- 標準ベース API のサポート

関連項目：『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』

非同期によるアプリケーションの統合 Oracle Streams Advanced Queuing は、分散アプリケーションの非同期による統合を実現します。メッセージのエンキューには、複数の方法が用意されています。メッセージは、取得プロセスまたは同期取得が暗黙的に取得するか、アプリケーションやユーザーが明示的に取得できます。

また、遅延と存続期間を指定してメッセージをエンキューできます。遅延を指定すると、メッセージを後で参照できるようにエンキューできます。アドバンスト・キューイングは、使用前にメッセージを順序付けする操作についても複数の方法をサポートしています。たとえば、メッセージの順序付けには、先入れ先出し方式と優先順位ベースがあります。

メッセージの使用方法も複数用意されています。自動適用の場合は、メッセージに対するユーザー指定アクションを起動できます。使用側アプリケーションでは、メッセージを明示的にデキューできます。デキューのブロック化と非ブロック化の両方がサポートされます。使用側アプリケーションは、PL/SQL、OCI または Java コールバック・ファンクションを使用して手続き型で通知を受け取るように選択できます。または、電子メールまたは HTTP ポストで通知を取得する方法もあります。使用側アプリケーションで自動適用を実行するように選択することも可能です。

拡張可能な統合アーキテクチャ Oracle Streams Advanced Queuing は、分散アプリケーションを開発して統合するための拡張可能なフレームワークを提供します。多くのアプリケーションは、Oracle Database をハブとする分散ハブ・アンド・スポーク・モデルを使用して統合されます。

Oracle データベース上の分散アプリケーションは、同じ Oracle データベース・サーバー・ハブのキューと通信します。Oracle Database の拡張可能なフレームワークにより、複数のアプリケーションが同じキューを共有できるため、サポートするアプリケーション数を増やすためにキューをさらに追加する必要がありません。

また、アドバンスト・キューイングはマルチ・コンシューマ・キューをサポートしており、1 つのメッセージを複数のアプリケーションで使用できます。新規のアプリケーションが追加されるたびに、これらのアプリケーションでは、Oracle データベース・サーバー・ハブと同じキューおよび同じメッセージを使用してビジネス・トランザクションを調整できます。この方法には、メッセージが 1 度しか配信されないという保証はそのままで、拡張性が得られるという利点があります。

アドバンスト・キューイングはコンテンツベースのパブリッシュ・サブスクライブ・モデルをサポートしています。この場合、アプリケーションはメッセージをパブリッシュし、コンシューマはパブリッシャ・アプリケーションを認識せずにメッセージにサブスクライブします。この種のモデルでは、既存のアプリケーションに変更を加えずに、コンシューマ・アプリケーションをハブに追加できます。

分散アプリケーションが異なる Oracle Database で実行されている場合は、ビジネス上の通信を適切な Oracle Database に自動的に伝播できます。伝播は、Oracle Streams Advanced Queuing システムにより自動的に管理され、アプリケーションに対しては透過的です。

異機種間アプリケーションの統合 従来、アプリケーションが異なる場合は、共通のデータ・モデルを通信に使用する必要がありました。このデータ・モデルは、メッセージ指向のミドルウェアでサポートされるデータ型に制限があったため、さらに限定されていました。Oracle Streams Advanced Queuing は、複数のデータ型のメッセージを格納できる `ANYDATA` キューをサポートしています。

アドバンスト・キューイングは、アプリケーションに Oracle タイプ・システムの全機能を提供します。これには、`NUMBER`、`DATE`、`VARCHAR` などのスカラー・データ型、継承を伴う Oracle Database オブジェクト型、XML データ用の追加演算子を持つ XMLType および `ANYDATA` のサポートが含まれます。特に、XMLType 型がサポートされるため、アプリケーション開発者はビジネス通信の拡張性と柔軟性に関する XML の全機能を活用できます。

Oracle Streams Advanced Queuing には、変換機能も用意されています。異なるデータ・モデルを持つアプリケーションは、独自のデータ・モデルとの間でメッセージをデキューまたはエンキューしながらメッセージを変換できます。このような変換マッピングは SQL 式として定義され、PL/SQL ファンクション、Java 関数または外部 C コールアウトを含むことができます。

従来型アプリケーションの統合 Oracle Messaging Gateway は、Oracle Database アプリケーションを WebSphere MQ（旧称 MQ Series）や Tibco などの他のメッセージ・キューイング・システムと統合します。メインフレーム上の多くの従来型アプリケーションは WebSphere MQ と通信するため、これらのアプリケーションを Oracle Database 環境に統合する必要があります。メッセージ・ゲートウェイは、Oracle 以外のメッセージ・キューを Oracle Streams キューであるかのように見せて、Oracle Streams キューと WebSphere MQ または Tibco のキューの間でメッセージを自動的に伝播します。

複数のパートナにまたがる分散アプリケーションは、Oracle Streams Advanced Queuing のインターネット・アクセス機能を使用して調整できます。これらの機能を使用すると、ビジネス・パートナやアプリケーションは、インターネット上のアドバンスト・キューイングのキューに注文を安全に入れることができます。権限を付与されていて認証を受けたビジネス・パートナのみが、このような操作を実行できます。

アドバンスト・キューイングのインターネット操作は、インターネットでの転送に HTTP（S）などの XML ベース・プロトコルを利用しており、メッセージはセキュリティを維持しつつファイアウォールを通過できます。インターネット通信のサポートにより、通信コストが劇的に削減されるため、ソリューション全体のコストも削減されます。

標準ベース API のサポート Oracle Streams Advanced Queuing は、業界標準の API である SQL、JMS および SOAP をサポートしています。SQL を使用してデータベースに行われた変更は、メッセージとして自動的に取得されます。

同様に、配布されたメッセージとデータベースの変更を、データベース表に適用して SQL で表示できます。メッセージは、業界標準の JMS を使用してエンキューおよびデキューできます。アドバンスト・キューイングには、メッセージのエンキューとデキューのための SOAP ベースの XML API も用意されており、OCI と OCCI をサポートしています。

データベース変更通知

クライアント・アプリケーションは、登録された問合せの結果セットが変更された場合、通知を受け取ることができます。たとえば、クライアントが `hr.employees` 表の問合せを登録し、あるユーザーが従業員を追加した場合、このアプリケーションは、新しい行が表に追加されたときにデータベース変更通知を受け取ることができます。`hr.employees` の新しい問合せでは、変更された結果セットが戻されます。データベース変更通知は多くの開発コンテキストに適していますが、キャッシュ・データに依存する中間層アプリケーションに特に有効です。

> **関連項目：** データベース変更通知の使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

チェンジ・データ・キャプチャ

Oracle Streams のインフラストラクチャを使用するチェンジ・データ・キャプチャ機能により、Oracle Database のリレーショナル表に対して追加、更新または削除されたデータが効率的に識別されて取得され、変更データが ETL ツールおよびアプリケーションで使用可能になります。Oracle Streams のチェンジ・データ・キャプチャ機能を使用すると、表全体ではなく変更があったデータのみがすばやく識別され、処理されます。

> **関連項目：**『Oracle Database データ・ウェアハウス・ガイド』

異機種間環境

Oracle Streams はオープンな情報共有ソリューションであり、Oracle と Oracle 以外のシステム間での異機種間レプリケーションをサポートしています。Oracle Database Gateway を使用すると、Oracle Database で実行された DML 変更を Oracle 以外のデータベースに適用できます。

Oracle Database ソースから Oracle 以外の宛先への DML 変更の取得と適用を実装するために、Oracle Database システムはプロキシとして機能し、通常は Oracle Database 宛先サイトで実行される適用プロセスを実行します。Oracle Database システムではさらに、Oracle Database Gateway を使用して Oracle 以外のシステムとの通信が行われます。

変更は Oracle Database 自体でデキューされ、ローカル適用プロセスにより Oracle Database Gateway を使用してネットワーク接続全体の Oracle 以外のシステムに対して適用されます。

Oracle 以外のデータベースから Oracle Database へと変更を伝播させる必要のあるユーザーは、Oracle 以外のデータベースに対する変更を取得するアプリケーションを記述します。このアプリケーションでは、トランザクション・ログを読み取るかトリガーを使用して変更を取得できます。その後、アプリケーションではこれらの変更をトランザクションにアセンブルし順序付けしてから、論理変更レコード（LCR）形式に変換して、ターゲットである Oracle Database のステージング領域にパブリッシュする必要があります。これらの変更の適用は、Oracle Streams の適用プロセスによって行われます。

関連項目：

- 23-14 ページの「Oracle Database Gateways」
- 『Oracle Streams レプリケーション管理者ガイド』

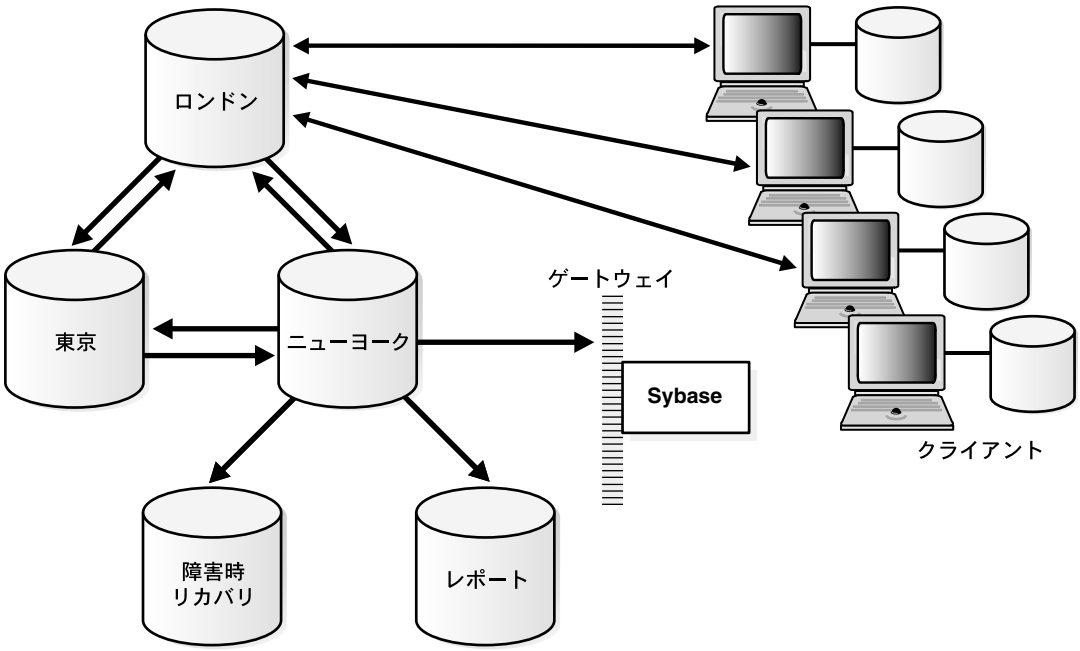
Oracle Streams のユースケース

Oracle Streams を使用すると、新しいクラスのアプリケーションを使用可能にするための構成を作成できます。また、すべてのデプロイメントとその関連メタデータには互換性があります。たとえば、レプリケーションのインストールは、データ・ウェアハウスをロードするように簡単に拡張し、双方向のレプリケーションを使用可能にすることができます。このため、完全な再構成を必要としません。

会社が、可用性、スケーラビリティおよびパフォーマンスを改善するために、Oracle Streams を使用して自社の Web サイトのコピーを複数維持するとします。その他の要件として、ニューヨークの本社にいるアナリストが非定型問合せを実行できるように、レポート用データベースには最新の情報を格納し、障害時リカバリ用データベースはニューヨーク本社とは別個に管理する必要があります。また、更新可能なマテリアライズド・ビューを使用して、現場の営業担当をサポートできるようにします。最後の要件は、Sybase データベースで管理されている既存のアプリケーションとデータを共有することです。

図 23-2 に、Oracle Streams の構成を示します。

図 23-2 Oracle Streams の構成



Oracle Streams は、ニューヨーク、ロンドンおよび東京という 3 つの地域別サイトで構成される N 方向の構成でデータをレプリケートするために使用されます。これらの各サイトでは、Oracle Streams の暗黙的な取得機能により、各地のサブスクライブ表に発生する変更が収集され、キューにローカルにステージングされます。各地で取得されたすべての変更は、その他の地域のそれぞれのデータベースに転送されます。目的は、各データベースで行われた変更を、その他すべてのデータベースに反映することです。これにより、サブスクライブされた世界中のオブジェクトの完全なデータが提供されます。

更新は各地域のデータベースで受信された時点で自動的に適用されるため、Oracle Streams の適用プロセスを使用して変更が適用されます。変更が適用されると、Oracle Streams により競合の有無をチェックし、検出された場合はそれを解消します。また、Oracle Streams を使用して Oracle 以外のデータベースとの間で特定の表データを交換することもできます。Oracle Database Gateway for Sybase を利用すると、Oracle Streams の適用プロセスは Oracle Database の場合と同じメカニズムを使用して変更を Sybase データベースに適用します。

レポートおよび障害時リカバリ用のデータベースは、ニューヨークのデータベース・サイトからホスティングされます。レポート用データベースは、該当するアプリケーション表の読取り専用コピーが格納されている、全機能を備えた Oracle Database です。レポート・サイトは、これらのアプリケーション表に対する変更を取得するように構成されていません。このレポート用データベースの構成と使用には制限が課されません。

ロンドンのサイトは、複数の更新可能マテリアライズド・ビュー・サイトのマスター・サイトとしても機能します。各営業担当は、データのうち必要な部分のみの更新可能なコピーを受け取ります。通常、これらのサイトは、前回のリフレッシュ以降に発生した注文をアップロードして変更をダウンロードするために、1 日に 1 度のみ接続します。

マテリアライズド・ビュー

Oracle Streams はマテリアライズド・ビューと全面的に相互運用可能です。マテリアライズド・ビューは、更新可能または読取り専用の、データの特定時点のコピーを維持するために使用できるスナップショットです。スナップショットは、値ベースの選択基準と一致するマスター表全体のコピー、またはその各行のサブセットを含めるように定義できます。多重化マテリアライズド・ビューも使用でき、この場合は 1 つのマテリアライズド・ビューが別のマテリアライズド・ビューに基づいています。マテリアライズド・ビューは、トランザクション一貫性を持つバッチ更新を介して、対応するマスター表から定期的に更新またはリフレッシュされます。

製品カタログは本社でのみ更新されるため、読取り専用のマテリアライズド・ビューを使用すると、更新された製品カタログを各種の営業支社に定期的に伝播させることができます。

マテリアライズド・ビューは専用接続を必要としないため、モバイル・コンピューティングに最適です。たとえば、会社が、営業担当用に更新可能なマテリアライズド・ビューを使用するように選択するとします。営業担当は各自のラップトップに注文を随時入力し、その日の終わりに地域の営業所に接続するだけで、これらの変更をアップロードして更新があればダウンロードできます。

> **関連項目：** レプリケーションにマテリアライズド・ビューを使用する方法は、『Oracle Database アドバンスト・レプリケーション』を参照してください。

Oracle Database でのデータの比較および収束

データベース・オブジェクトは、2 つ以上の Oracle Database で共有できます。データベース・オブジェクトを共有する方法の 1 つは、Oracle Streams またはマテリアライズド・ビューの機能を使用するレプリケーション環境を構成することです。通常、レプリケーション環境では、索引などのその他のタイプのデータベース・オブジェクトのみでなく、表などのデータを含むデータベース・オブジェクトも共有されます。あるデータベースの共有データベース・オブジェクトに変更が加えられると、その変更は転送され、そのデータベース・オブジェクトを共有するその他のデータベースでも変更が実行されます。この方法では、レプリケーション環境により、各データベースで共有データベース・オブジェクトの同期が維持されます。

これらの共有データベース・オブジェクトのデータが、そのデータベース・オブジェクトを共有するデータベースで同一でない場合があります。データの差異には、ネットワークの問題、コンピュータ・システムの問題およびレプリケーション構成エラーなど様々な要因があります。

DBMS_COMPARISON パッケージを使用すると、異なるデータベースのデータベース・オブジェクトを比較して、その相違を特定できます。このパッケージでは、異なるデータベースで一貫性を保てるよう、データベース・オブジェクトを収束することもできます。

DBMS_COMPARISON パッケージでは、次のタイプのデータベース・オブジェクトを比較および収束できます。

- 表
- 単一表ビュー
- マテリアライズド・ビュー
- 表、単一表ビューおよびマテリアライズド・ビューのシノニム

比較の結果は、DBA_COMPARISON、DBA_COMPARISION_SCAN、DBA_COMPARISON_COLUMNS および DBA_COMPARISON_ROW_DIF を含む複数のデータ・ディクショナリ・ビューに保存されます。

> **関連項目：**
>
> - 『Oracle Database 2 日でデータ・レプリケーションおよび統合ガイド』
> - 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
> - 『Oracle Database リファレンス』

Oracle 以外のシステムの統合

Oracle Database と Oracle 以外のデータベースとの統合に関して、Generic Connectivity と Oracle Database Gateway という 2 つのソリューションが用意されています。この 2 つのソリューションを使用すると、Oracle Database クライアントから Oracle 以外のデータ・ストアにアクセスできます。この 2 つのコンポーネントはサード・パーティの SQL 言語、データ・ディクショナリおよびデータ型を Oracle 形式に変換し、Oracle 以外のデータ・ストアをリモート Oracle Database として使用可能にします。これらのテクノロジにより、企業は各種システムをシームレスに統合し、全社的に連結されたビューを提供できます。

Generic Connectivity および Oracle Database Gateway は、分散 SQL による同期アクセスに使用できます。また、Oracle Database Gateway は、Oracle Streams による非同期アクセスに使用できます。Oracle Streams 環境に Oracle Database Gateway を導入することで、Oracle Database から Oracle 以外のデータベースへのデータ・レプリケーションが可能になります。

Generic Connectivity と Oracle Database Gateway は、どちらも Oracle Database 環境から Oracle 以外のシステムのデータに透過的にアクセスします。Oracle 分散データベース環境の場合と同様に、位置の透過性を Oracle 以外のシステムにあるオブジェクトへと拡張することも可能です。そのため、ユーザーは Oracle 以外のデータベースにあるオブジェクト用のシノニムを作成し、その物理位置を指定せずに参照できます。この透過性により、アプリケーション開発者はアプリケーションを Oracle 以外の様々なシステムのデータにアクセスするようにカスタマイズする必要がないため、開発作業が軽減され、アプリケーションのモバイル性が向上します。アプリケーションでは、システム固有のインタフェースを使用して Oracle 以外のシステムと相互に操作する必要がなく（この方法ではアプリケーション側に処理が集中する可能性があります）、Oracle Database システムと Oracle 以外のシステムの両方に一貫性のある Oracle Database インタフェースを使用できます。

この項の内容は、次のとおりです。

Generic Connectivity

Generic Connectivity は、ODBC または OLEDB ドライバを使用して、ODBC または OLEDB に準拠する Oracle 以外のシステムにアクセスする汎用ソリューションです。また、Oracle にゲートウェイ・ソリューションが用意されていない多数のデータ・ストアへのデータ・アクセスを提供します。これにより、ODBC や OLEDB などの業界標準を使用した透過的接続性が使用可能になります。Generic Connectivity では、Foxpro、Access、dBase などのローエンド・データ・ストアや Excel などの非リレーショナル・ターゲットにアクセスできます。

Oracle Database Gateways

Generic Connectivity は汎用ソリューションですが、Oracle Database Gateway は調整されたソリューションであり、Oracle 以外のシステム向けに特別にコーディングされています。Generic Connectivity よりも優れた機能性とパフォーマンスを持つ最適化されたソリューションを提供します。

Generic Connectivity は業界標準に依存しているのに対して、Oracle Database Gateway は Oracle 以外のシステムにシステム固有のインタフェースを使用してアクセスします。また、Oracle Database Gateway はエンドツーエンドで認証が行われます。Oracle Database には、Sybase、DB2、Informix および Microsoft SQL Server など、多数のソース用の Oracle Database Gateway が用意されています。

関連項目：『Oracle Database Heterogeneous Connectivity 管理者ガイド』

第 IV 部

Oracle データベース・アプリケーション開発

第 IV 部では、アプリケーション開発に使用できる Oracle 組込みの言語とデータ型について説明します。第 IV 部には、次の章が含まれています。

24

SQL

この章では、SQL の概要について説明します。

この章の内容は、次のとおりです。

- SQL の概要
- SQL 文
- カーソル
- 共有 SQL 領域
- 解析
- 問合せの処理
- SQL の処理
- オプティマイザの概要

関連項目：『Oracle Database SQL 言語リファレンス』

SQL の概要

SQL は、データベース・アクセス用の非手続き型言語です。処理内容を SQL で記述すると、SQL 言語コンパイラがデータベースをナビゲートし、指定されたタスクを実行するプロシージャを自動的に生成するという点で、SQL は非手続き型言語です。

Oracle SQL には、ANSI/ISO 標準 SQL 言語に対応する多くの拡張機能が組み込まれています。また、Oracle のツール製品とアプリケーションを利用すると、文を追加できます。SQL*Plus および Oracle Enterprise Manager などの Oracle ツールでは、Oracle データベースに対して ANSI/ISO 標準の SQL 文を実行できる他、これらのツール製品で利用可能な追加の文や機能を実行できます。

いくつかの Oracle のツール製品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 Database 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われる可能性があります。

関連項目：

- SQL 文および SQL のその他の部分（演算子、関数および書式モデルなど）の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- SQL 文との相違点など、SQL*Plus の文の詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

SQL 文

Oracle Database 内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。1 つの文は、識別子、パラメータ、変数、名前、データ型および SQL **予約語**から構成されます。SQL 予約語は、SQL で特別な意味を持ち、他の目的には使用できません。たとえば、`SELECT` と `UPDATE` は予約語のため、表名には使用できません。

SQL 文は、コンピュータ・プログラムまたは命令です。文は、完全な SQL 文と等価である必要があります。次に例を示します。

```
SELECT last_name, department_id FROM employees;
```

実行できるのは完全な SQL 文のみです。次のような不完全文を実行しようとすると、テキストの不足により SQL 文を実行できないことを示すエラーが発生します。

```
SELECT last_name
```

Oracle Database の SQL 文は、次のカテゴリに分類されます。

- データ操作言語文
- データ定義言語文
- トランザクション制御文
- セッション制御文
- システム制御文
- 埋込み SQL 文

関連項目： PL/SQL プログラム・ユニットで SQL 文を使用する方法の詳細は、第 22 章「トリガー」を参照してください。

データ操作言語文

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

- 1 つ以上の表またはビューからデータの取出しまたはフェッチを行います（SELECT）。フェッチはスクロールが可能です（24-5 ページの「スクロール可能カーソル」を参照してください）。
- 表またはビューに新しいデータ行を追加します（INSERT）。
- 表またはビューの既存の行の列値を変更します（UPDATE）。
- 行の更新または、条件付きで表またはビューに行の挿入を行います（MERGE）。
- 表またはビューから行を削除します（DELETE）。
- SQL 文の実行計画を表示します（EXPLAIN PLAN）。
- 表またはビューをロックして、一時的に他のユーザーのアクセスを制限します（LOCK TABLE）。

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

```
SELECT last_name, manager_id, commission_pct + salary FROM employees;

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

DELETE FROM employees WHERE last_name IN ('WARD','JONES');
```

DML エラー・ロギング

DML 文にエラーが発生した場合、エラー・コードおよび関連付けられたエラー・メッセージ・テキストがエラー・ロギング表に記録される間、DML 文は処理を続行できます。これは特に、長時間実行のバルク DML 文に役立ちます。DML 操作の完了後、エラー・ロギング表をチェックして、エラーのある行を修正できます。

エラー・ロギング表の名前、文タグおよび拒否の制限を指定する新規構文が DML 文に追加されています。拒否の制限は、文を強制終了する必要があるかどうかを決定します。パラレル DML 操作の場合、拒否の制限はスレーブごとに適用されます。パラレル操作に対して正確に規定される拒否の制限の値は、0（ゼロ）および無制限のみです。

データ変換エラーが発生した場合、Oracle Database は、列についてログに記録するための意味のある値を提供します。たとえば、障害が発生した変換演算子に対する最初のオペランドの値をログに記録します。値を導出できない場合、その列について NULL がログに記録されます。

関連項目：

- 24-9 ページの「SQL 文の処理の説明」
- DML エラー・ロギングの詳細は、『Oracle Database 管理者ガイド』を参照してください。
- DML エラー・ロギングの使用例は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。
- DML エラー・ロギングの構文は、『Oracle Database SQL 言語リファレンス』を参照してください。

データ定義言語文

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

- スキーマ・オブジェクトと他のデータベース構造（データベースとデータベース・ユーザーを含む）を作成、変更および削除します（CREATE、ALTER、DROP）。
- スキーマ・オブジェクトの名前を変更します（RENAME）。
- スキーマ・オブジェクトの構造を削除することなく、それらのオブジェクトの中のすべてのデータを削除します（TRUNCATE）。
- 権限とロールを付与および取り消します（GRANT、REVOKE）。
- 監査オプションをオンまたはオフに切り換えます（AUDIT、NOAUDIT）。
- データ・ディクショナリにコメントを追加します（COMMENT）。

DDL 文は、先行するコマンド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 employees TO scott;

REVOKE DELETE ON employees FROM scott;
```

関連項目：

- 24-11 ページの「DDL 文の処理」
- 第 20 章「データベース・セキュリティ」

トランザクション制御文

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

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

関連項目： 24-11 ページの「トランザクション制御処理」

セッション制御文

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

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

システム制御文

システム制御文は、Oracle データベース・インスタンスのプロパティを変更します。システム制御文は、ALTER SYSTEM のみです。この文は、設定値（共有サーバーの最小数など）の変更、セッションの終了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、手続き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。

- カーソルの定義、割当ておよび解放を行います（DECLARE CURSOR、OPEN、CLOSE）。
- データベースを指定し、Oracle Database に接続します（DECLARE DATABASE、CONNECT）。
- 変数名を割り当てます（DECLARE STATEMENT）。
- 記述子を初期化します（DESCRIBE）。
- エラー条件と警告の処理方法を指定します（WHENEVER）。
- SQL 文を解析および実行します（PREPARE、EXECUTE、EXECUTE IMMEDIATE）。
- データベースからデータを取り出します（FETCH）。

カーソル

カーソルとは、解析済の文と、処理に使用するその他の情報が保持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 Database ユーザーは Oracle Database ユーティリティの自動カーソル処理を使用しますが、プログラム・インタフェースを利用すると、アプリケーション設計者はカーソルを制御しやすくなります。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 文の解析に使用できます。

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、アプリケーション側で不必要なカーソルを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。

Oracle Database は暗黙的に再帰的 SQL 文を発行する必要があり、再帰カーソルが必要になる場合があります（主として DDL 文の場合）。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。これらの再帰カーソルに対して再帰コールが発行されます。1 つのカーソルで複数の再帰コールが実行されることもあります。それらの再帰カーソルでは、共有 SQL 領域も使用します。

スクロール可能カーソル

カーソルを実行すると、問合せの結果が結果セットと呼ばれる行の集合に入れられます。この集合は、順次またはランダムにフェッチできます。スクロール可能カーソルは、フェッチおよび DML 操作を順送りで行う必要がない場合のカーソルです。以前フェッチした行のフェッチ、結果セットの *n* 番目の行のフェッチ、および結果セットの現在位置から *n* 番目の行のフェッチのために、インタフェースが存在しています。

> **関連項目：** Oracle Call Interface（OCI）内でのスクロール可能カーソルの使用方法の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

共有 SQL 領域

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 Database はそのことを自動的に認識します。その文が最初に出現したときの処理に使用された SQL 領域は共有されます。つまり、その後に同じ文が出現すると、それを処理するためにこの領域が使用されます。したがって、一意の文に対しては 1 つの共有 SQL 領域しか存在しません。共有 SQL 領域は共有メモリー領域であるため、どの Oracle Database 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 Database 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的 SQL 文を評価します。

> **関連項目：** 共有 SQL の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』および『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

解析

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

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

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

アプリケーションが SQL 文の解析コールを出すことと、Oracle Database 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発行しなくても、その文を繰り返し実行できます。
- Oracle Database 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、再解析なしでその文を繰り返し実行できます。

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

> **関連項目：** 1-36 ページの「PL/SQL の概要」

SQL 文を解析するとその文の妥当性がチェックされますが、解析で識別されるのは文を実行する前に検出が可能なエラーのみです。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レポートもされません。

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のみですが、問合せは 1 行または数千行を戻します。問合せの結果は、常に**表形式**です。結果の行は、1 行ごとにまたはグループ単位で**フェッチ**され（取り出され）ます。

問合せ処理のみに関連する問題がいくつかあります。明示的な SELECT 文のみでなく、他の SQL 文に含まれる暗黙的問合せ（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。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、次のような特長があります。

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

SQL の処理

この項では、SQL 処理の基本について説明します。最初に、ほとんどのタイプの SQL 文を網羅する、一般的な SQL 文の実行のフローチャートを示します。次に、SQL 文の処理の段階について概説します。最後に、異なるタイプの SQL 文では、フローチャートおよび説明が異なる場合があることを説明します。

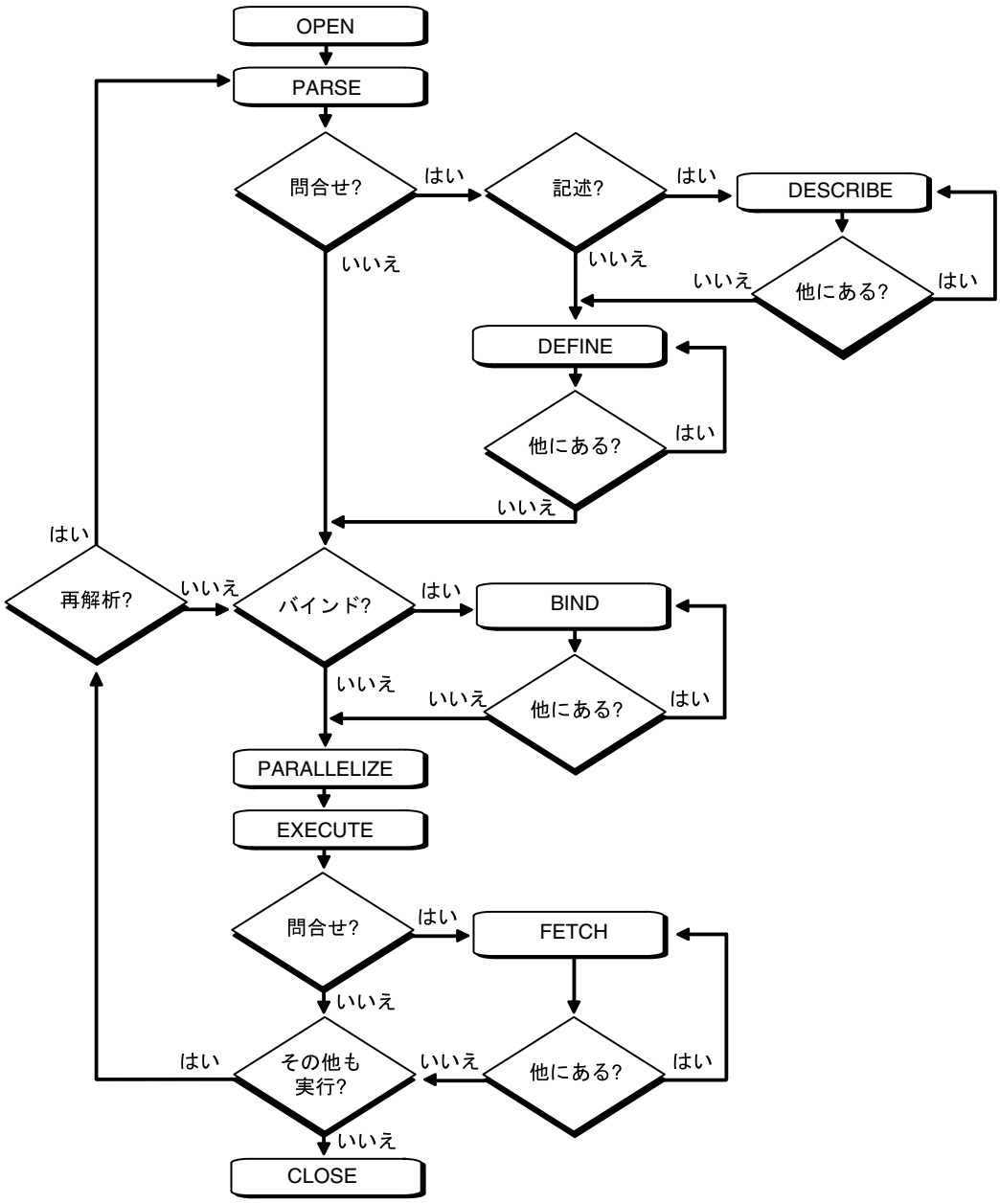
この項の内容は次のとおりです。

SQL 文の実行のフローチャート

図 24-1 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、Oracle Database でこれらの段階の実行順序が少し異なることもあります。たとえば、DEFINE の段階は、コーディングの仕方によっては FETCH の直前にくることがあります。

多くの Oracle のツール製品では、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、この情報は Oracle アプリケーションの作成に役立ちます。

図 24-1 SQL 文の処理の段階



SQL 文の処理の説明

この項では、1 つの例をあげて、SQL 文の実行中に何がどのように処理されているのか、処理の段階ごとに説明します。この例では、特定の DML 文を処理しますが、その内容は他のタイプの SQL 文にも一般化できます。それに続く項では、他のタイプの SQL 文を実行した場合、この説明とどのように異なるかについての情報を提供します。24-11 ページの「他のタイプの SQL 文の処理」を参照してください。

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は、Oracle Database への接続が確立され、`employees` 表を更新するための適切なスキーマに接続されているとします。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 employees SET salary = 1.10 * salary
    WHERE department_id = :department_id;
```

`Department_id` 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 `department_id` 値が使用されます。

次に、各タイプの文を処理するために必要な段階を示します。段階 7 はオプションで、段階 4、5、および 9 は図 24-1 で説明する問合せのみに適用されることに注意してください。

- 段階 1: カーソルのオープンまたは作成
- 段階 2: 文の解析
- 段階 3: 問合せの有無の判断
- 段階 4: 問合せの結果の記述（問合せのみ）
- 段階 5: 問合せの出力の定義（問合せのみ）
- 段階 6: 変数のバインド
- 段階 7: 文のパラレル化（オプション）
- 段階 8: 文の実行
- 段階 9: 問合せの行のフェッチ（問合せのみ）
- 段階 10: カーソルのクローズ

段階 1: カーソルのオープンまたは作成

プログラム・インタフェース・コールにより、カーソルがオープン（作成）されます。カーソルは、SQL 文からの要求で独立して作成されます。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 Database に渡され、SQL 文の解析済の表現が共有 SQL 領域にロードされます。文処理のこの段階では、多くのエラーを捕捉できます。

関連項目：

- 24-6 ページの「解析」
- 24-6 ページの「共有 SQL 領域」

段階 3: 問合せの有無の判断

この段階では、SQL 文が問合せで開始するかどうかを判断します。

関連項目： 24-6 ページの「解析」

段階 4: 問合せの結果の記述（問合せのみ）

記述段階が必要なのは、問合せがユーザーにより対話式で入力された場合など、問合せ結果の特性が不明な場合のみです。この場合は、記述段階で問合せ結果の特性（データ型、長さおよび名前）がわかります。

段階 5: 問合せの出力の定義（問合せのみ）

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。これらの変数は**定義変数**と呼ばれます。Oracle Database は、必要に応じてデータ型を変換します。（図 24-1「SQL 文の処理の段階」の「DEFINE」を参照。）

段階 6: 変数のバインド

この時点で、Oracle Database 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 Database は、文に含まれている変数の値を必要とします。例では、`department_id` の値が必要です。これらの値を取得する処理のことを、**変数のバインド**と呼びます。

プログラムでは、値を検出できる位置（メモリー・アドレス）を指定する必要があります。Oracle Database ユーティリティはアプリケーションのエンド・ユーザーに新しい値の入力を要求するプロンプトを表示するのみであるため、エンド・ユーザーはバインド変数を指定している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 Database 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 Database 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かぎり、それぞれの値のデータ型と長さも指定する必要があります。

関連項目：

- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Pro*C/C++ プログラマーズ・ガイド』（動的 SQL 方法 4 に関する項を参照）

値のデータ型および長さを指定する方法の詳細は、これらを参照してください。

段階 7: 文のパラレル化（オプション）

Oracle Database では、問合せ（`SELECT`、`INSERT`、`UPDATE`、`MERGE`、`DELETE`）および一部の DDL 処理（索引の作成、副問合せを含む表の作成、パーティションの操作など）をパラレル化できます。パラレル化すると、複数のサーバー・プロセスが SQL 文の処理を実行するため、処理を高速に完了できます。

関連項目： 第 16 章「ビジネス・インテリジェンス」

段階 8: 文の実行

この時点で、Oracle Database に必要なすべての情報およびリソースが用意され、文を実行できます。問合せまたは `INSERT` 文の場合は、データが変更されないため、行をロックする必要はありません。ただし、`UPDATE` 文または `DELETE` 文の場合は、トランザクションの `COMMIT`、`ROLLBACK` または `SAVEPOINT` が次に実行されるまで、文の影響を受けるすべての行がロックされます。この処理により、データ整合性を確実に維持できます。

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

段階 9: 問合せの行のフェッチ（問合せのみ）

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 10: 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。

他のタイプの SQL 文の処理

次の項では、DDL 文、トランザクション制御文およびその他の SQL 文の処理と、24-9 ページの「SQL 文の処理の説明」で説明した処理との相違について説明します。

この項の内容は、次のとおりです。

DDL 文の処理

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

トランザクション制御処理

通常、1 つのトランザクションを構成するアクションのタイプを考慮する必要があるのは、Oracle Database 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。作業を論理単位として完了し、データの一貫性が保たれるようにトランザクションを定義する必要があります。トランザクションには、1 つの論理作業単位に必要な部分を過不足なくすべて含める必要があります。

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

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

その他の処理タイプ

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。

オプティマイザの概要

すべての SQL 文で**オプティマイザ**が使用されます。オプティマイザは、指定されたデータへの最も効率的なアクセス手段を決定する Oracle Database の機能です。Oracle には、オプティマイザにジョブを適切に実行させるためのテクニックも用意されています。

たとえば、表または索引へのアクセス順序を変えることにより、SQL DML（SELECT、INSERT、UPDATE、MERGE または DELETE）文の処理は様々になります。Oracle Database で文を実行するときに使用する手順は、文の実行速度に大きく影響します。オプティマイザは、代替アクセス・パスの多数の要因を考慮します。

注意： オプティマイザは、Oracle Database 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、オプティマイザは入手可能な情報から適切なものに基づいて決定を行うことがあります。

オプティマイザが選択する内容は、そのアプローチ方法および目標に従って決まります。陳腐化したオブジェクトや統計のないオブジェクトは、自動的に分析されます。PL/SQL パッケージ DBMS_STATS を使用して、オプティマイザの統計も収集できます。

Oracle Database 11*g* では、次のものを含む、新しい拡張統計が導入されています。

- 複数列統計
- 1 つの列に対する関数の統計
- ビューに対する統計

さらに、統計を公開することなく収集できるようになりました。新しく収集された統計（保留中の統計）を公開する前にテストできます。

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設計者の方が、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。アプリケーション設計者は、SQL 文内にヒントを使用して文の実行方法を指定できます。

関連項目：

- DBMS_STATS の使用方法は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- オプティマイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

この項の内容は、次のとおりです。

SQL 計画の管理（SPM）

Oracle Database 11*g* では、オプティマイザが自動的に計画を管理し、検証された計画のみが使用されます。SQL 計画の管理（SPM）では、現在の計画よりも新しい計画のほうがより適切に実行されることが検証された場合にかぎり、その新しい計画を使用することによって、計画の更新を制御できます。

実行計画

DML 文を実行するために、Oracle Database で数多くの処理を必要とする場合があります。各処理では、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 Database が文の実行に使用する処理の組合せを、実行計画と呼びます。実行計画には、文がアクセスする表ごとのアクセス方法と表の順序付け（結合順序）が含まれています。実行計画の各処理は、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。

この項の内容は、次のとおりです。

ストアド・アウトライン

ストアド・アウトラインとは、実行計画の作成終了時にオプティマイザが生成する実行計画を抽象化したもので、主にヒントの集合として機能します。次にアウトラインを使用するとき、これらのヒントがコンパイルの様々な段階で適用されます。アウトライン・データは、OUTLN スキーマに格納されています。実行計画は、ストアド・アウトラインを編集することでチューニングできます。

ストアド・アウトラインの編集

アウトライン編集セッションの開始時には、ユーザーのスキーマにアウトラインのクローンが作成されます。その後の編集操作は、ユーザーが編集を完了し、それらを公開するまでそのクローンに対して行われます。このため、このユーザーによるいずれの編集内容も、それが明示的に保存されるまで、アウトラインのパブリック・バージョンを使用する残りのユーザー・グループに影響を与えることはありません。

注意： ストアド・アウトラインは、Oracle Database 11*g* では非推奨です。ストアド・アウトラインではなく、SQL 計画ベースラインを使用することをお薦めします。

実行計画および SQL 計画ベースラインの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

25

サポートされているアプリケーション開発言語

この章では、Oracle アプリケーション開発システム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。

- Oracle アプリケーション開発言語の概要
- C/C++ プログラミング言語の概要
- PL/SQL の概要
- Java の概要
- Microsoft プログラミング言語の概要
- レガシー言語の概要

関連項目： 第 24 章「SQL」

Oracle アプリケーション開発言語の概要

Oracle Database の開発者は、アプリケーション開発に使用する言語を C、C++、Java、COBOL、PL/SQL、Visual Basic および C# の中から選択できます。言語固有の標準がすべてサポートされます。開発者は、最も慣れている言語、または特定のタスクに最も適した言語を選択できます。たとえば、アプリケーションでは、サーバー側で Java を使用して動的 Web ページを作成し、PL/SQL を使用してデータベースにストアド・プロシージャを実装し、C++ を使用して中間層に計算集中型のロジックを実装できます。

Oracle には Pro* シリーズのプリコンパイラも用意されており、SQL と PL/SQL を C、C++、COBOL または FORTRAN アプリケーション・プログラムに埋め込むことができます。

関連項目：

- プログラミング環境を選択する方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 『Oracle Database グローバリゼーション・サポート・ガイド』
- 第 26 章「Oracle データ型」

C/C++ プログラミング言語の概要

この項の内容は、次のとおりです。

Oracle Call Interface（OCI）の概要

Oracle Call Interface（OCI）は Application Program Interface（API）です。この API により、C 言語によるシステム固有のファンクション・コールを使用して Oracle データベースにアクセスし、SQL 文の実行フェーズ全体を制御するアプリケーションを作成できます。OCI は、C のデータ型、コール規則、構文およびセマンティクスをサポートしています。OCI から Oracle Database の表データに直接アクセスしたり、Oracle Streams との間でデータをエンキューおよびデキューできます。

OCI の機能は、次のとおりです。

- ディスク領域を大幅に削減してアプリケーションをデプロイする方法であるインスタント・クライアント
- C アプリケーションからのスレッド管理、接続プール、グローバリゼーション・ファンクションおよびデータのダイレクト・パス・ロード（SQL*Loader ユーティリティ）
- N 層の認証
- Oracle Database オブジェクトを使用したアプリケーション開発の包括的サポート
- 外部データベースへのアクセス
- ハードウェアへの追加投資なしで増加するユーザーや要求にサービスを提供できるアプリケーション

OCI では、C ホスト・プログラミング言語を使用して、Oracle データベース内のデータおよびスキーマを操作できます。OCI には、実行時にアプリケーションにリンク可能な動的ランタイム・ライブラリ（OCI ライブラリ）という形式で、標準データベース・アクセスおよび検索関数のライブラリが用意されています。

問合せの結果を、OCI クライアント結果キャッシュのメモリーにキャッシュできます。その後、OCI クライアントはキャッシュされた結果を使用して、これらの後続の問合せの実行に対応します。データベース・コールを作成して問合せを再実行するより、結果キャッシュから結果を取得する方が速いため、頻繁に実行される問合せのパフォーマンスは、結果がキャッシュされている場合には大幅に向上します。OCI クライアントの結果キャッシュはプロセス単位であり、様々なセッション間で共有されます。

> **関連項目：** OCI クライアント結果キャッシュの詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

OCI では、Oracle Database のオブジェクト・リレーショナル機能がサポートされています。重要な要素は、**オブジェクト・キャッシュ**と呼ばれる作業領域をアプリケーション・プログラムから使用できるようにするコールの集合です。オブジェクト・キャッシュは、クライアント側にあるメモリー・ブロックで、プログラムがオブジェクト全体を格納し、サーバーとの間を往復せずにオブジェクト間をナビゲートできるようにします。

オブジェクト・キャッシュは、そのオブジェクト・キャッシュを使用するアプリケーション・プログラムが完全に制御および管理します。Oracle Database は、オブジェクト・キャッシュにアクセスできません。オブジェクト・キャッシュを使用するアプリケーション・プログラムは、サーバーとのデータの一貫性を維持しながら、同時発生のアクセスによる衝突から作業領域を保護する必要があります。

問合せの結果を、OCI クライアント結果キャッシュのメモリーにキャッシュできます。その後、OCI クライアントはキャッシュされた結果を使用して、これらの後続の問合せの実行に対応します。データベース・コールを作成して問合せを再実行するより、結果キャッシュから結果を取得する方が速いため、頻繁に実行される問合せのパフォーマンスは、結果がキャッシュされている場合には大幅に向上します。OCI クライアントの結果キャッシュはプロセス単位であり、様々なセッション間で共有されます。

> **関連項目：** OCI クライアント結果キャッシュの詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

OCI には、次の機能を持つ関数が用意されています。

- SQL を使用してサーバー上のオブジェクトにアクセスします。
- 横断ポインタまたは `REF` によってオブジェクト・キャッシュ内のオブジェクトをアクセスし、操作し、管理します。
- Oracle Database の日付および文字列、数値を C データ型に変換します。
- オブジェクト・キャッシュのメモリーのサイズを管理します。
- 一時型記述を作成します。一時型記述は、データベースに永続的には格納されません。

OCI の同時実行性が向上し、個々のオブジェクトにロックをかけることができるようになりました。また、複合オブジェクト検索をサポートすることにより、パフォーマンスが向上しています。

OCI を使用する開発者は、Object Type Translator（OTT）を使用して、Oracle Database オブジェクト型に対応する C 構造体データ型を生成できます。

> **関連項目：**『Oracle Call Interface プログラマーズ・ガイド』

Oracle C++ Call Interface（OCCI）の概要

Oracle C++ Call Interface（OCCI）は、オブジェクト指向機能、システム固有クラスおよびC++ プログラミング言語のメソッドを使用して Oracle Database にアクセスできる C++ API です。OCCI は OCI 上に作成され、その機能とパフォーマンスは、オブジェクト指向パラダイムのさらに使用しやすいインタフェースと結合されています。

この項の内容は、次のとおりです。

- OCCI の結合リレーショナルおよびオブジェクト・インタフェース
- OCCI のナビゲーショナル・アクセス用インタフェース

OCCI の結合リレーショナルおよびオブジェクト・インタフェース

結合リレーショナル API およびオブジェクト・クラスによって、SQL はデータベースにアクセスできます。これらのインタフェースを介して、SQL はサーバー上でオブジェクトまたはリレーショナル・データの作成、操作およびフェッチを実行します。アプリケーションは、ラージ・オブジェクト、オブジェクトや構造型、配列および参照など、サーバー上のすべてのデータ型にアクセスできます。

OCCI のナビゲーショナル・アクセス用インタフェース

ナビゲーショナル・アクセス用インタフェースは、C++ のインタフェースです。これによって、C++ オブジェクト形式のオブジェクト・リレーショナル・データに、SQL を使用せずにシームレスにアクセスし、変更できます。C++ オブジェクトは、透過的にアクセスされ、必要に応じてデータベースに格納されます。

OCCI のナビゲーショナル・アクセス用インタフェースを使用すると、オブジェクトを取得し、そのオブジェクトから他のオブジェクトへの参照内をナビゲートできます。サーバー・オブジェクトは、C++ クラスのインタフェースとしてアプリケーション・キャッシュでマテリアライズされます。アプリケーションは OCCI のオブジェクト・ナビゲーショナル・コールを使用して、サーバーのオブジェクトに対して次の機能を実行できます。

- オブジェクトの作成、アクセス、ロック、削除およびフラッシュ
- オブジェクト参照の取得および参照内のナビゲート

関連項目：『Oracle C++ Call Interface プログラマーズ・ガイド』

Oracle Type Translator の概要

Object Type Translator（OTT）は、オブジェクト型に対応する C 言語の構造体の宣言を自動生成するプログラムです。OTT は、Oracle Database のオブジェクト型に対応する C++ クラス定義を生成し、OCCI アプリケーションでネイティブな C++ オブジェクト・インタフェースに対して使用できるようにします。OTT は、Pro*C/C++ プリコンパイラと OCI サーバー・アクセス・パッケージを使用します。

関連項目：

- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Oracle C++ Call Interface プログラマーズ・ガイド』
- 『Pro*C/C++ プログラマーズ・ガイド』

Pro*C/C++ プリコンパイラの概要

Oracle プリコンパイラは、高水準のソース・プログラムに SQL 文を埋め込むことができるプログラミング・ツールです。このプリコンパイラは入力としてホスト・プログラムを受け入れ、埋込み SQL 文を標準の Oracle Database ランタイム・ライブラリ・コールに変換し、通常の方法でコンパイル、リンクおよび実行できるソース・プログラムを生成します。Oracle プリコンパイラは C、C++、COBOL および FORTRAN に使用できます（ただし、使用できないシステムもあります）。

Oracle Pro*C/C++ プリコンパイラを使用すると、C または C++ のソース・ファイルに SQL 文を埋め込むことができます。Pro*C/C++ は入力としてソース・ファイルを読み取り、C または C++ のソース・ファイルを出力します。このソース・ファイルでは、埋込み SQL 文が Oracle Database ランタイム・ライブラリ・コールで置換されてから、C または C++ コンパイラによりコンパイルされます。

Pro*C/C++ を使用すると高度にカスタマイズされたアプリケーションを作成できます。たとえば、最新のウィンドウおよびマウス・テクノロジを取り込むユーザー・インタフェースを作成できます。また、バックグラウンドで実行されてユーザーの介入を必要としないアプリケーションも作成できます。

さらに、Pro*C/C++ ではアプリケーションの詳細なチューニングが可能です。リソース使用、SQL 文の実行および各種のランタイム・インジケータを緊密に監視できます。この情報を使用してプログラムのパラメータを変更し、パフォーマンスを最大化します。

プリコンパイルすることでアプリケーション開発プロセスにステップが 1 つ加わりますが、所要時間は短縮されます。埋込み SQL 文は、手動ではなくプリコンパイラにより、Oracle Database ランタイム・ライブラリ（SQLLIB）のコールに変換されます。また、Pro*C/C++ プリコンパイラはホスト変数を分析し、構造と列のマッピングを定義し、`SQLCHECK=FULL` が指定されている場合には埋込み SQL 文のセマンティック分析を実行します。

Oracle Pro*C/C++ プリコンパイラを使用すると、プログラマは C および C++ プログラムでオブジェクト・データ型を使用することもできます。Pro*C/C++ を使用する開発者は、Object Type Translator（OTT）を使用して、Oracle Database のオブジェクト型とコレクションを C のデータ型にマップし、Pro*C/C++ アプリケーションで使用できます。

Pro*C/C++ 開発者は、プログラムから OCI 関数をコールすることもできます。

Pro*C/C++ は、オブジェクト型とコレクションのコンパイル時型チェック機能と、データベース型から C データ型への自動型変換を提供しています。Pro*C/C++ は、オブジェクトを作成したり破棄したりするための `EXEC SQL` 構文と、サーバーにあるオブジェクトにアクセスするための 2 つの方法を提供しています。

- Pro*C/C++ プログラムに埋め込まれている SQL 文と PL/SQL ファンクションまたはプロシージャ。
- オブジェクト・キャッシュへの単純なインタフェース。横断ポインタでオブジェクトにアクセスし、その後サーバー上で変更および更新します。

関連項目：

- 25-32 ページの「Microsoft プログラミング言語の概要」
- Pro*C/C++ プリコンパイラの詳細説明は、『Pro*C/C++ プログラマーズ・ガイド』を参照してください。

型記述の動的作成とアクセス

Oracle では、C API を提供しているため、型記述の動的な作成とアクセスが可能です。さらに、一時型記述、つまりデータベースに永続的に格納されない型記述も作成できます。

C API によって、OCIAnyData と OCIAnyDataSet の作成とアクセスが可能になります。

- OCIAnyData 型は、ある型の自己記述（型について）データ・インスタンスをモデル化します。
- OCIAnyDataSet 型は、ある型のデータ・インスタンスのセットをモデル化します。

Oracle では、これらのデータ型に対応する SQL データ型（Oracle の Open Type System）も用意されています。

- SYS.ANYTYPE は OCIType に対応します。
- SYS.ANYDATA は OCIAnyData に対応します。
- SYS.ANYDATASET は OCIAnyDataSet に対応します。

このようなデータについて、データベース表の列と SQL の問合せを作成できます。

C API は次の項を使用します。

- **一時型** : データベースに永続的に格納されない型記述（型メタデータ）。
- **永続型** : CREATE TYPE SQL 文を使用して作成された SQL 型。これらの型記述は、データベースに永続的に格納されます。
- **自己記述データ** : 実際の内容と一緒に型情報をカプセル化しているデータ。ANYDATA 型（OCIAnyData）はこのようなデータをモデル化します。SQL 型のどのデータ値も、ANYDATA に変換できるため、古いデータ値に変換しなおすことができます。不適切な変換を実行しようとすると、エラーが発生します。
- **自己記述多重集合** : 一連のデータ・インスタンス（同じ型のものすべて）をそれぞれの型記述と一緒にカプセル化したもの。

関連項目 :

- 『Oracle Database オブジェクト・リレーショナル開発者ガイド』
- 『Oracle Call Interface プログラマーズ・ガイド』

PL/SQL の概要

PL/SQL は、SQL に対する Oracle の手続き型言語拡張機能です。使用しやすく、SQL とシームレスで、堅牢かつ移植性を備えたセキュアなサーバー側に格納される手続き型言語を提供します。PL/SQL のコンパイラとインタプリタは Oracle Developer に埋め込まれており、開発者にクライアント側とサーバー側の両方で一貫性のある高度な開発モデルを提供します。また、PL/SQL ストアド・プロシージャは、Pro*C や Oracle Call Interface および Oracle Reports や Oracle Forms など、多数の Oracle Database クライアントからコールできます。

PL/SQL を使用すると、SQL 文を手続き型構造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表示されるが名前が付いていない、または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 Database 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 Database はそのプロシージャを解析し、その解析済の表現をデータベースに格納します。さらに Oracle Database では、ファンクション（プロシージャによく似ているもの）やパッケージ（プロシージャとファンクションをまとめたもの）を作成して保管することもできます。

関連項目：

25-19 ページの「Java の概要」

第 22 章「トリガー」

この項の内容は、次のとおりです。

PL/SQL の実行方法

PL/SQL の実行方法は、次のいずれかです。

解析済の実行

Oracle9*i* より前のバージョンでは、PL/SQL のソース・コードは、その名のとおり常にバイトコード表現にコンパイルされ、Oracle Database の一部として、また Oracle Forms などの製品に実装される移植性のある仮想コンピュータによって実行されていました。Oracle9*i* からは、システム固有の実行または解析済の実行を選択できます。

システム固有の実行

計算集約型のプログラム・ユニットで最大限のパフォーマンスが発揮されるように、データベースに格納されている PL/SQL プログラム・ユニットのソース・コードを直接コンパイルし、特定のプラットフォームのオブジェクト・コードにします。（このオブジェクト・コードは、Oracle Database にリンクされます。）

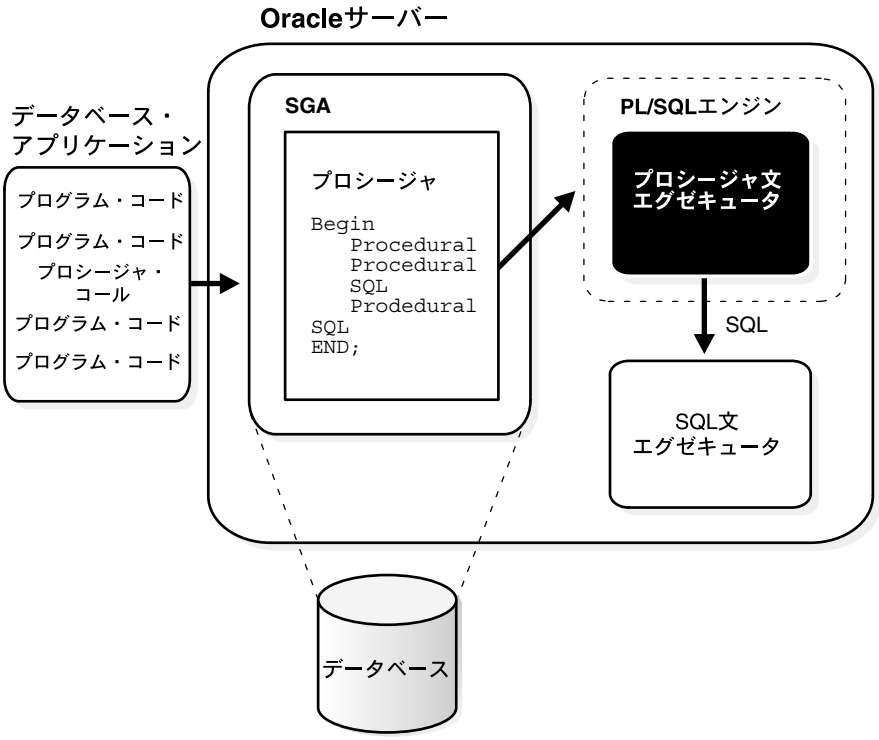
関連項目：『Oracle Database PL/SQL 言語リファレンス』

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 Database 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、この項では、Oracle Database に格納でき、Oracle Database の PL/SQL エンジンを使用して処理できるプログラム・ユニットについて説明します。それぞれの Oracle のツール製品の PL/SQL 機能については、該当する Oracle のツール製品のマニュアルに説明があります。

図 25-1 に、Oracle Database に含まれている PL/SQL エンジンを示します。

図 25-1 PL/SQL エンジンおよび Oracle Database



プログラム・ユニットは、データベースに格納されています。アプリケションがデータベースに格納されたプロシージャをコールすると、Oracle Database は、コンパイル済のプログラム・ユニットをシステム・グローバル領域（SGA）内の共有プールにロードします。PL/SQL 文エグゼキュータと SQL 文エグゼキュータは、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 Database
- Oracle Forms（バージョン 3 以上）
- SQL*Menu（バージョン 5 以上）
- Oracle Reports（バージョン 2 以上）
- Oracle Graphics（リリース 2 以上）

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 以上）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 Database 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 Call Interface（OCI）
- SQL*Plus
- Oracle Enterprise Manager

PL/SQL の言語構造

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

- 変数と定数
- カーソル
- 例外

関連項目： 『Oracle Database PL/SQL 言語リファレンス』

この項の内容は、次のとおりです。

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。

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

カーソル

カーソルは、Oracle Database 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。

関連項目： 24-5 ページの「スクロール可能カーソル」

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（ゼロ）による除算などの不正な操作や、PL/SQL に戻された Oracle Database 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と、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について、特定の例外ハンドラを作成できます。

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

また、動的 SQL を使用して DML 文または DDL 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` 文を発行できます。

関連項目：

- 動的 SQL のための 2 つのアプローチの比較については、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 動的 SQL の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

PL/SQL プログラム・ユニット

Oracle Database 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にアクセスし、情報を処理できます。プロシージャ、ファンクションおよびパッケージは、すべて PL/SQL プログラム・ユニットの例です。

ストアド・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 構成メンバー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、コール元から、入力のみ、出力のみまたは入出力の値が入るパラメータを指定できます。プロシージャとファンクションを使用すると、SQL が持つ平易さや柔軟性と、構造化プログラミング言語が持つプロシージャ的な機能性をあわせ持つことができます。

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元に値を戻さないのに対して、ファンクションは必ず 1 つの値を戻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。

プロシージャまたはファンクションは、次の方法で対話型で実行できます。

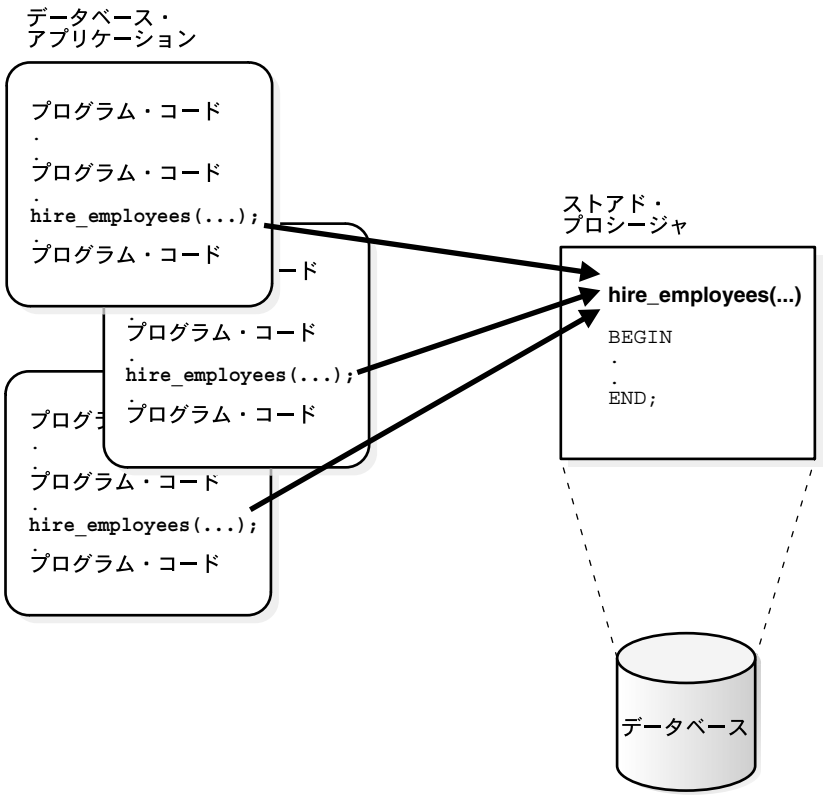
- SQL*Plus などの Oracle のツール製品を使用します。
- Oracle Forms やプリコンパイラのアプリケーションなど、データベース・アプリケーションのコード内で明示的にコールします。
- 別のプロシージャやトリガーのコード内で明示的にコールします。

関連項目：

- C または C++ のストアド・プロシージャをコールする方法の詳細は、『Pro*C/C++ プログラマーズ・ガイド』を参照してください。
- COBOL のストアド・プロシージャをコールする方法の詳細は、『Pro*COBOL プログラマーズ・ガイド』を参照してください。
- 各種アプリケーションのストアド・プロシージャをコールする方法の詳細は、個々のプログラマーズ・ガイドを参照してください。

図 25-2 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。

図 25-2 ストアド・プロシージャ



次のストアド・プロシージャの例では、従業員レコードが employees 表に挿入されます。

```
Procedure hire_employees (last_name VARCHAR2, job_id VARCHAR2, manager_id NUMBER,
hire_date DATE, salary NUMBER, commission_pct NUMBER, department_id NUMBER)

BEGIN
.
.
INSERT INTO employees VALUES (emp_sequence.NEXTVAL, last_name, job_id, manager_id,
hire_date, salary, commission_pct, department_id);
.
.
END
```

この例のデータベース・アプリケーションはすべて、hire_employees 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、次のような文で hire_employees プロシージャを実行できます。

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

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

関連項目：『Oracle Database PL/SQL 言語リファレンス』

この項の内容は、次のとおりです。

プロシージャの利点

ストアド・プロシージャは、次の分野で特長を発揮します。

- 定義者権限プロシージャでのセキュリティ

  ストアド・プロシージャは、データ・セキュリティの規定に役立ちます。定義者の権限で実行するプロシージャとファンクションを介してのみユーザーがデータにアクセスできるようにすると、ユーザーが実行できるデータベース操作を制限できます。

  たとえば、表を更新するプロシージャへのアクセス権は付与しても、その表自体へのアクセス権は付与しないこともできます。ユーザーがプロシージャを呼び出すと、そのプロシージャがプロシージャの所有者の権限で操作を実行します。プロシージャの実行権限しかなく、表データの問合せ、更新または削除の権限を持たないユーザーは、プロシージャを呼び出すことはできても、表のデータをそれ以外の方法では操作できません。

  関連項目： 25-14 ページの「ストアド・プロシージャの依存性の追跡」

- 実行者権限プロシージャで継承される権限およびスキーマのコンテキスト

  実行者権限プロシージャは、それをコールしたプロシージャから権限とスキーマのコンテキストを継承します。つまり、実行者権限プロシージャは特定のユーザーやスキーマに結び付けられておらず、実行者権限プロシージャの各コールはカレント・ユーザーの権限でカレント・ユーザーのスキーマ内で実行されます。アプリケーション開発者は、実行者権限プロシージャにより、基礎となるデータが複数のユーザー・スキーマに分割されていても、アプリケーション・ロジックを容易に一元化できます。

  たとえば、あるユーザーが、管理者として employees 表の更新プロシージャを実行する場合は、給与を更新できますが、同じプロシージャを事務担当として実行する場合には、その操作を住所データの更新のみに制限できます。

- パフォーマンスの改善
  - 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 Database に送信する場合に比べて、ネットワークを介して送信する必要のある情報量が少なくなります。
  - データベース内でプロシージャのコンパイル済形式をそのまま使用できるため、実行時にコンパイルする必要がありません。
  - プロシージャがすでにシステム・グローバル領域（SGA）の共有プール内にある場合は、ディスクから検索する必要がないため、ただちに実行を開始できます。

- メモリー割当て

  ストアド・プロシージャは、Oracle Database の共有メモリー機能を活用しているため、複数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済みます。同じコードを何人ものユーザーが共有すれば、アプリケーションに必要な Oracle Database メモリーを実質的に削減できます。

- 生産性の向上

  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、`employees` 表の従業員レコードを挿入、更新または削除するためのプロシージャを記述できます。これらのプロシージャは、各タスクの実行に必要な SQL 文を書き換えることなく、どんなアプリケーションからでもコールできます。データ管理の方法に変更があった場合も、修正する必要があるのはプロシージャのみで、プロシージャを使用するすべてのアプリケーションを修正する必要はありません。

- 整合性

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

  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利用できます。再びテストする必要はありません。プロシージャにより参照されるデータ構造が変更された場合には、プロシージャの再コンパイルのみが必要になります。プロシージャをコールするアプリケーションの修正は、必ずしも必要ありません。

プロシージャのガイドライン

ストアド・プロシージャの設計時には、次のガイドラインに従ってください。

- プロシージャは、単一の限定的なタスクを実行するように定義します。複数の個別のサブタスクを含む長いプロシージャは定義しないでください。多数のプロシージャに共通のサブタスクが存在すると、複数のプロシージャ・コードに不必要な重複が発生するためです。
- Oracle Database の他の機能によってすでに提供されている機能性を重複して提供するプロシージャは定義しないでください。たとえば、整合性制約を宣言すれば簡単に規定できる単純なデータ整合性規則の場合、それを規定するプロシージャは定義しないでください。

無名 PL/SQL ブロックとストアド・プロシージャ

ストアド・プロシージャを作成し、それをスキーマ・オブジェクトとしてデータベースに格納できます。いったん作成してコンパイルしたプロシージャは、再コンパイルしなくても実行可能な名前付きのオブジェクトになります。さらに、依存性情報がデータ・ディクショナリに格納されるため、それぞれのストアド・プロシージャの妥当性が保証されます。

ストアド・プロシージャとは別の方法として、無名 PL/SQL ブロックを Oracle のツール製品やアプリケーションから Oracle Database に送信し、無名 PL/SQL ブロックを作成できます。Oracle Database によって PL/SQL ブロックがコンパイルされ、SGA の共有プールにそのコンパイル済バージョンが入れられます。ただし、そのソース・コードやコンパイル済バージョンは、現行インスタンスの後も再利用できるようにデータベースに格納されることはありません。共有 SQL を使用すると、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、そのブロックを再利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 Database 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 Database 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**と呼びます。パッケージ内で定義されているプロシージャは、そのパッケージの一部とみなされます。

> **関連項目：** パッケージの利点の詳細は、25-15 ページの「PL/SQL パッケージ」を参照してください。

ストアド・プロシージャの依存性の追跡

ストアド・プロシージャは、その本体で参照するオブジェクトに依存します。Oracle Database 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 Database はこのような依存性の管理を自動的に処理します。

> **関連項目：** 依存性の追跡の詳細は、第 6 章「スキーマ・オブジェクトの依存性」を参照してください。

外部プロシージャ

Oracle Database 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 Database とは別のアドレス空間で実行されます。

> **関連項目：** 外部プロシージャの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

テーブル・ファンクション

テーブル・ファンクションは、出力用の行セットを生成する関数です。つまり、テーブル・ファンクションは、コレクション型インスタンスを戻します（NESTED TABLE データ型および VARRAY データ型）。テーブル・ファンクションは、SQL 文の FROM 句で通常の表のかわりに使用できます。

Oracle Database では、テーブル・ファンクションにより、ファンクションからの結果を**パイプライン処理**（Oracle Database の行ソースのように機能する）できます。これは、ODCITable インタフェースの実装や、ネイティブの PL/SQL 命令の使用により実現できます。

パイプライン処理を使用すると、Oracle Warehouse Builder（OWB）およびカートリッジ・グループなど、多くのアプリケーションでパフォーマンスが改善されます。

データ・ウェアハウス構築における抽出、変換、ロード（ELT）プロセスは、OLTP システムからデータを抽出します。抽出されたデータは、データ・ウェアハウスにロードされる前に、（PL/SQL などの手続き型言語で記述された）変換のシーケンスに渡されます。

Oracle Database では、テーブル・ファンクションおよび非テーブル・ファンクションのパラレル実行も可能です。パラレル実行では次の点が拡張されます。

- 関数は、副問合せのオペランドに対応する行の集合を直接受け入れることができます。
- 入力行の集合を、パラレル関数の複数のインスタンス間でパーティション化できます。関数の開発者は、関数のパラレル・インスタンス間で入力行をパーティション化する方法を指定します。

つまり、テーブル・ファンクションはビューに似ています。ただし、SQL で宣言を使用して変換を定義するかわりに、PL/SQL でプロシージャを使用して定義します。これは特に、ETL に通常必要な独特の複雑な変換に役立ちます。

関連項目：

- 16-6 ページの「抽出、変換、ロード（ETL）の概要」
- 『Oracle Database データ・カートリッジ開発者ガイド』
- 『Oracle Database PL/SQL 言語リファレンス』

PL/SQL 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。

Oracle Database には、データベースの機能性を拡張して SQL 機能への PL/SQL アクセスを提供する多数の PL/SQL パッケージがあります。たとえば、`ULT_HTTP` パッケージを使用すると、PL/SQL と SQL からの HTTP コールアウトでインターネット上のデータにアクセスしたり、Oracle Web Server Cartridges をコールできます。オラクル社が提供するパッケージは、アプリケーションの作成時や、独自のストアド・プロシージャを作成する場合に使用できます。

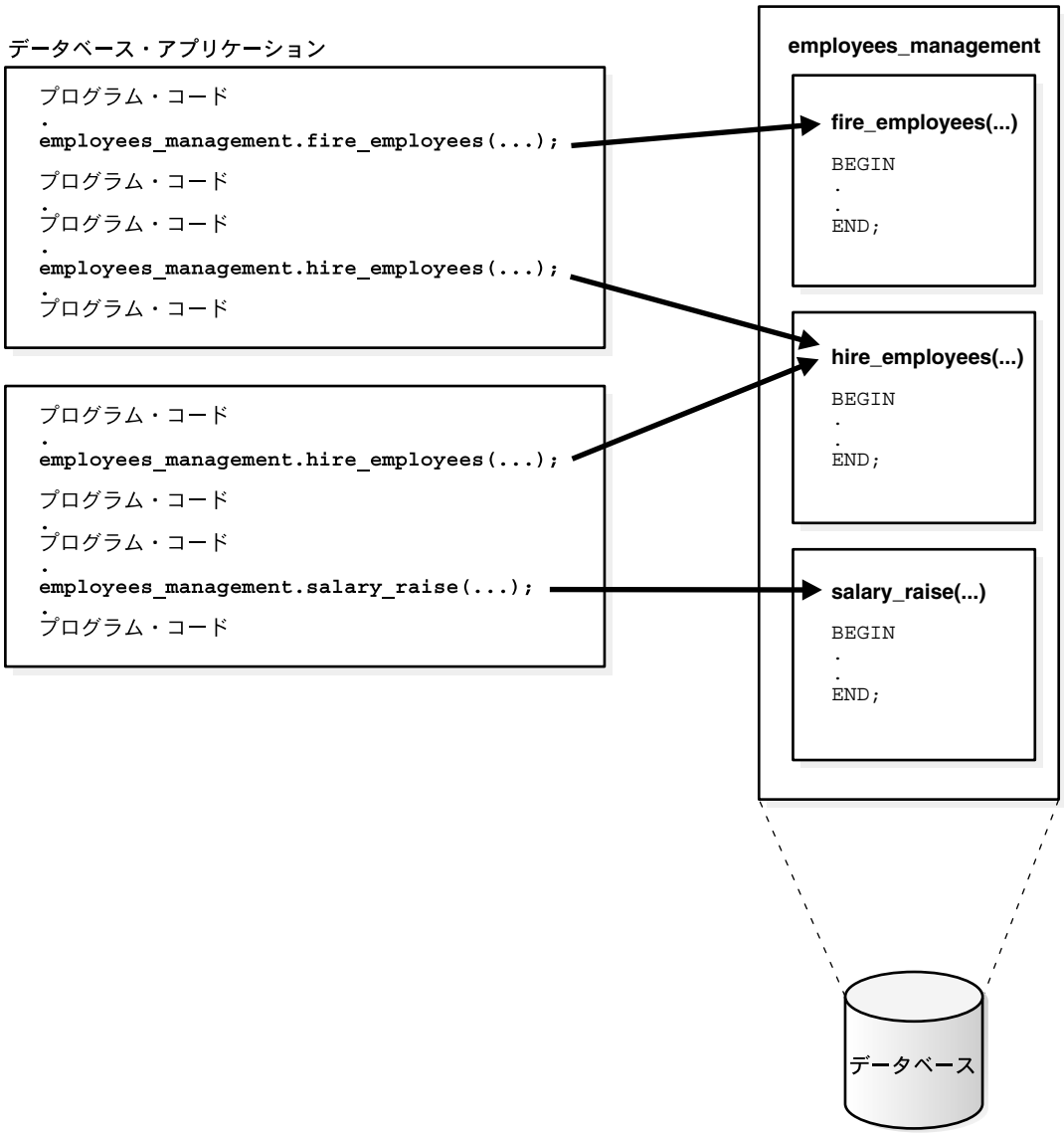
パッケージは、仕様部と本体の 2 つの部分に分けて作成します。パッケージの**仕様部**ではパッケージのすべてのパブリックな構成メンバーを宣言し、**本体**ではパッケージのすべての構成メンバー（パブリックとプライベート）を定義します。パッケージ本体は、パッケージと同じスキーマに作成する必要があります。パッケージを 2 つに分けることには、次のような利点があります。

- 柔軟な開発サイクルを実行できます。パッケージ本体を実際には作成せずに、仕様部を作成し、パブリック・プロシージャを参照するようにできます。
- パッケージ仕様部にパブリックに宣言されている仕様部とは別個に、パッケージ本体に含まれているプロシージャ本体を変更できます。プロシージャの仕様が変更されないかぎり、パッケージ内の変更されたプロシージャを参照するオブジェクトに無効のマークが付けられることはありません。つまり、これらのオブジェクトに、再コンパイルを要するマークが付けられることはありません。

注意： パッケージ本体およびパッケージの仕様部は、常に同じスキーマに存在する必要があります。

図 25-3 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 25-3 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。employees_management 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_employees パッケージ・プロシージャを実行できます。

```
EXECUTE employees_management.hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE, 500,
NULL, 20);
```

関連項目：

- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

パッケージの利点

パッケージは、次の分野で特長を発揮します。

- 関連するプロシージャと変数のカプセル化

  ストアド・パッケージを使用すると、ストアド・プロシージャ、変数、データ型などを**カプセル化**、つまりグループ化し、名前を付けて 1 単位としてデータベースに格納できます。この方針は、開発過程における優れた構成を提供します。プロシージャ構造をカプセル化すると、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されたユーザーは、そのパッケージのすべての構成メンバーにアクセスできるようになります。

- 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

  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パブリックとプライベートのどちらかに指定できます。パブリックはパッケージのユーザーが直接アクセスできることを意味します。プライベートはパッケージのユーザーには非表示になっていることを意味します。

  たとえば、パッケージに 10 個のプロシージャが含まれているとします。この中の 3 つのプロシージャのみをパブリックに設定して、パッケージのユーザーが実行できるように定義できます。残りのプロシージャはプライベートなので、パッケージ内のプロシージャによってのみアクセスできます。パブリックおよびプライベートのパッケージ変数を、PUBLIC への権限付与と混同しないでください。

  > **関連項目：** PUBLIC への権限付与の詳細は、第 20 章「データベース・セキュリティ」を参照してください。

- 優れたパフォーマンス

  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に対し、このロードは 1 回の操作で完了します。そのため、関連するパッケージ・プロシージャがコールされても、すでにメモリーにあるコンパイル済のコードを実行すると、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ない場合は、パッケージの構成メンバーを参照するスキーマ・オブジェクト（常に仕様部を介してアクセス）を再コンパイルする必要はありません。パッケージを使用すると、不必要な再コンパイルが最小限に抑えられるため、全体的なデータベース・パフォーマンスへの影響は少なくなります。

PL/SQL のコレクションとレコード

多数のプログラミング技法は、配列、構造体、リスト、ネストした表、セットおよびツリーなどのコレクション型を使用します。これらの技法をデータベース・アプリケーション内でサポートするために、PL/SQL にはデータ型 TABLE および VARRAY が用意されており、索引付き表、ネストした表および可変サイズ配列を宣言できます。

この項の内容は、次のとおりです。

コレクション

コレクションは、すべて同じ型の要素からなる順序付けられたグループです。各要素には、コレクション内での位置を決定する一意の添字が付いています。

コレクションの機能は、ほとんどの第 3 世代プログラミング言語に見られる配列と同じです。また、コレクションはパラメータとして渡すことができます。そのため、データベース表との間や、クライアント側アプリケーションとストアド・サブプログラムの間で、データの列を移動するときに使用できます。

レコード

%ROWTYPE 属性を使用すると、表の 1 行やカーソルからフェッチされる 1 行を表すレコードを宣言できます。ただし、ユーザー定義レコードの場合は、所有しているフィールドを宣言できます。

レコードには一意の名前を持つフィールドが含まれ、そのデータ型は異なっていてもかまいません。名前、給与および採用日など、従業員に関する各種データがあるとします。これらの項目は、型は異なりますが論理的には関連しています。項目ごとに 1 フィールドを含むレコードを使用すると、データを論理単位として扱うことができます。

> **関連項目：** コレクションおよびレコードの使用方法の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

PL/SQL Server Pages

PL/SQL Server Pages（PSP）は、特殊なタグにより PL/SQL スクリプトが埋め込まれているサーバー側の Web ページ（HTML または XML 形式）です。動的 Web ページを生成するために、通常、開発者は C または Perl を使用して、同じプログラム内でデータをフェッチし、Web ページ全体を生成する CGI プログラムを記述しています。この種の動的ページの開発とメンテナンスは、コストと時間がかかる作業です。

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

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

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

PL/SQL Gateway は HTTP クライアントから HTTP 要求を受信し、URL に指定されている PL/SQL ストアド・プロシージャを起動し、クライアントに HTTP 出力を戻します。PL/SQL Server Pages は、PSP コンパイラによって PL/SQL ストアド・プロシージャにコンパイルされます。ゲートウェイによりプロシージャが実行されると、動的な内容を持つ Web ページが生成されます。PSP は、次の 2 つの既存の PL/SQL Gateway のどちらかとともに使用します。

- Oracle Application Server の PL/SQL カートリッジ
- WebDB

関連項目： PL/SQL Server Pages の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

Java の概要

Java は、アプリケーション・レベルのプログラムに効果を発揮するオブジェクト指向のプログラミング言語です。Java の機能は、次のとおりです。

- Java 仮想マシン（JVM）。プラットフォームの独立性を確保するための基礎を提供します。
- 自動ストレージ管理機能。分散したメモリーを連続したメモリー領域に集めます。
- C 言語に基づく言語構文。強い型指定が必要です。

この項の内容は、次のとおりです。

Java およびオブジェクト指向プログラミングの用語

この項には、Oracle Database 環境における Java アプリケーション開発の基本用語が含まれます。

この項の内容は、次のとおりです。

クラス

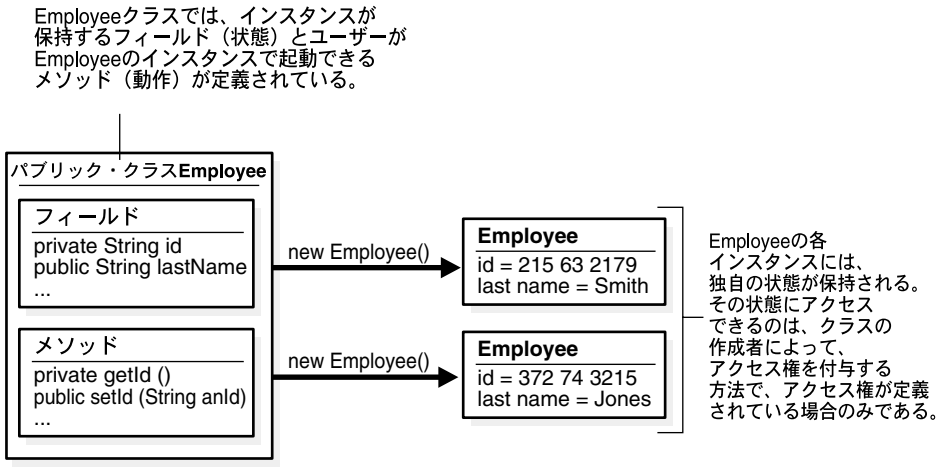
オブジェクト指向プログラミング言語はすべて、クラスの概念をサポートしています。表の定義では、クラスから共通の特性を共有するオブジェクト用のテンプレートが提供されます。それぞれのクラスには次の項目を含めることができます。

- 属性 - 特定クラスの各オブジェクトの静的変数またはインスタンス変数です。
- メソッド - クラスが定義するメソッドまたはあるクラスから拡張された任意のクラスが継承したメソッドを起動できます。

クラスからオブジェクトを作成すると、そのクラスのインスタンスが作成されます。インスタンスには、データまたは状態と呼ばれるオブジェクトのフィールドが含まれます。

図 25-4 に、2 つの属性である姓（`lastName`）と従業員識別子（`ID`）を使用して定義されている `Employee` クラスの例を示します。

図 25-4 クラスとインスタンス



インスタンスを作成するときは、属性には、ある従業員にのみ関係する個人のプライベートな情報を格納します。すなわち、従業員のインスタンスに含まれる情報は、該当する 1 人の従業員に対してのみ公開されます。図 25-4 の例は、従業員の 2 つのインスタンス Smith と Jones を示しています。各インスタンスには、個々の従業員に関連する情報が含まれています。

属性

インスタンス内の属性はフィールドと呼ばれます。インスタンスのフィールドは、リレーショナル表の行フィールドに類似しています。クラスにより、フィールドと各フィールドのタイプが定義されます。Java 内のフィールドを静的、パブリック、プライベート、保護付きまたはデフォルト・アクセスとして宣言できます。

- パブリック、プライベート、保護付きまたはデフォルト・アクセスのフィールドはそれぞれのインスタンス内に作成します。
- 静的フィールドは、従業員クラスのすべてのインスタンスで情報を使用できる点でグローバル変数に似ています。

言語仕様では、すべてのフィールドのデータの可視性規則が定義されます。可視性規則では、これらのフィールド内のデータにアクセス可能な環境を定義します。

メソッド

クラスでは、クラスのインスタンスを起動するメソッドも定義します。メソッドは Java で記述され、オブジェクトの動作を定義します。このように状態と動作をまとめることがカプセル化の本質であり、すべてのオブジェクト指向プログラミング言語の特長です。各従業員の `id` がプライベート・フィールドであることを宣言して `Employee` クラスを定義すると、他のオブジェクトは、メソッドがフィールドを戻したときにのみそのプライベート・フィールドにアクセスできます。この例では、オブジェクトは、`Employee.getId` メソッドを起動することで、従業員の識別子を取得します。

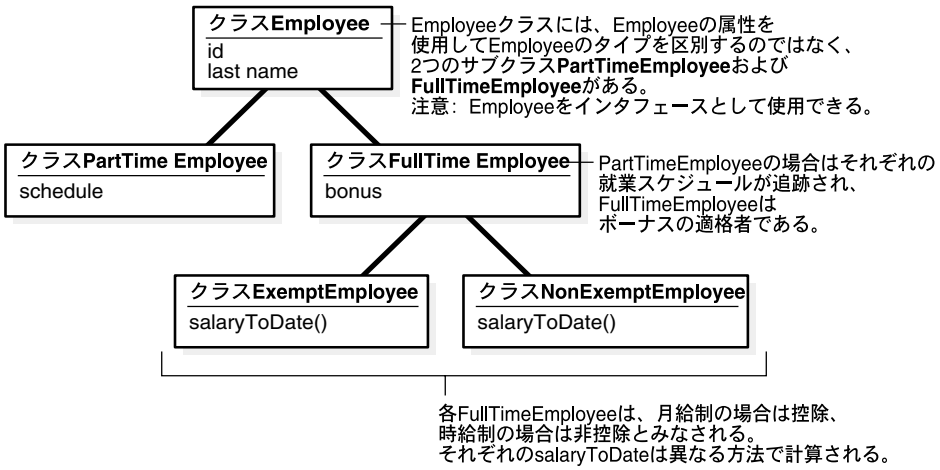
また、カプセル化では、`Employee.getId` メソッドをプライベートとして宣言すること、あるいは `Employee.getId` メソッドを記述しないでおくことができます。カプセル化を行うと、再利用可能なプログラム、または不正に使用されることのないプログラムを容易に記述できるようになります。カプセル化では、パブリックの宣言を行ったオブジェクトの機能のみがパブリックになります。他のすべてのフィールドとメソッドはプライベートです。プライベート・フィールドとプライベート・メソッドは、内部オブジェクトの処理に使用できます。

クラス階層

Java では、クラスをクラスの大きな階層内で定義します。階層の最上位にあるのは、Object クラスです。Java のすべてのクラスはスーパークラスの継承連鎖内を移動する過程で、あるレベルの `Object` クラスを継承しています。クラス B がクラス A を継承している場合、クラス B の各インスタンスには、クラス B で定義したすべてのフィールドとクラス A で定義したすべてのフィールドが含まれます。たとえば、図 25-5 では、`FullTimeEmployee` クラスは Employee クラスを継承しているため、`Employee` クラスで定義した `id` および `lastName` フィールドを含みます。また、`FullTimeEmployee` クラスでは、`FullTimeEmployee` にのみ含まれる別のフィールド `bonus` が追加されます。

クラス B のインスタンスでは、クラス A またはクラス B のいずれかで定義された任意のメソッドを起動できます。従業員の例では、`FullTimeEmployee` インスタンスでは、その独自クラスでのみ定義されているメソッド、または `Employee` クラスで定義されたメソッドを起動できます。

図 25-5 継承を使用した動作と状態のローカライズ



クラス B のインスタンスは、クラス A のインスタンスで置換可能です。これにより、コードの再利用性を改善するための、オブジェクト指向言語の強化された構成メンバーが継承されます。また、動作と状態を定義する新しいクラスを、階層内の適切な場所、しかもクラス・ライブラリの既存の機能が利用できる場所に作成できます。

インタフェース

Java でサポートされるのは 1 つの継承のみです。すなわち、各クラスは継承したクラスを 1 つのみ持ちます。1 つ以上のソースを継承する必要がある場合、Java はこれまでのような複雑さや混乱を招くことなく、複数継承に相当する継承をインタフェースを介して提供します。インタフェースはクラスに類似しています。ただし、インタフェースではメソッドを実装するかわりに、メソッドのシグネチャを定義します。このメソッドは、インタフェースを実装するように宣言したクラスに実装されます。1 つのクラスで同時に多数のインタフェースをサポートすると、複数継承が発生します。

ポリモフィズム

前述の Employee の例で、各種の従業員が給与累計で対応できるようにする必要があるとします。給与は、従業員の種類ごとに異なる方法で計算されます。

- FullTimeEmployee はボーナスの適格者です。
- NonExemptEmployee には時間外手当が支払われます。

従来の手続き型言語では、可能なケースをそれぞれ定義して長い SWITCH 文を記述します。

```
switch (employee.type) {
case: Employee
return employee.salaryToDate;
case: FullTimeEmployee
return employee.salaryToDate + employee.bonusToDate;
...
```

新しい種類の Employee を追加する場合は、SWITCH 文を更新する必要があります。データ構造を変更する場合は、その構造を使用する SWITCH 文をすべて変更する必要があります。

Java などのオブジェクト指向言語では、Employee クラスのうち、すでにこのクラスで定義されている以上の特殊な処理を必要とするサブクラスごとにメソッド compensationToDate を実装します。たとえば、NonExemptEmployee の compensationToDate メソッドを次のように実装するとします。

```
private float compensationToDate() {
return super.compensationToDate() + this.overtimeToDate();
}
```

FullTimeEmployee のメソッドを次のように実装します。

```
private float compensationToDate() {
return super.compensationToDate() + this.bonusToDate();
}
```

メソッド名 compensationToDate を共通して使用することで、使用している従業員の種類を知らなくても、同じメソッドを異なるクラスで起動して異なる結果を受け取ることができます。FullTimeEmployees と PartTimeEmployees を処理するために特殊なメソッドを記述する必要はありません。このように、異なるオブジェクトに対して同じメッセージに異なる方法で対応する機能は、ポリモフィズムと呼ばれます。

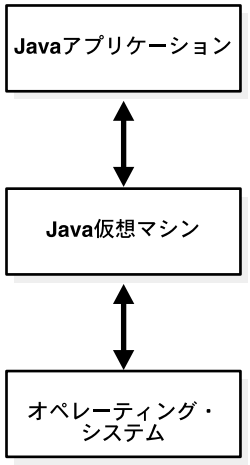
また、Employee をまったく継承しない新規のクラス Contractor を作成し、そこで compensationToDate メソッドを実装できます。給与累計を計算するプログラムは、フルタイム、パートタイム、外注契約者のいずれであるかに関係なく給与計算時にすべての従業員を反復し、それぞれで compensationToDate メソッドを起動した結果戻された値を合算します。メソッドのコール元が正常に動作することがわかっていれば、個々の compensationToDate メソッドを安全に変更できます。たとえば、既存のクラスに新規フィールドを安全に追加できます。

Java 仮想マシン（JVM）の概要

Java ソースは、他の高水準言語と同様、低水準の命令にコンパイルされます。Java では、これらの命令はバイトコードと呼ばれています（これらの命令サイズが一律 1 バイトの記憶単位であるためです）。C など他のほとんどの言語は、Intel または HP プロセッサに固有の命令など、コンピュータ固有の命令にコンパイルされます。Java ソースは、プラットフォームに依存しない、標準のバイトコード集合にコンパイルされ、Java 仮想マシン（JVM）と対話します。JVM は、Java コードを実行する特定のプラットフォーム用に最適化された個別のプログラムです。

図 25-6 に、Java によりプラットフォームの独立性がどのように保たれるかを示します。Java ソースはバイトコードにコンパイルされ、プラットフォームに依存しません。各プラットフォームには、オペレーティング・システムに固有の JVM をインストールします。ソースの Java バイトコードは、JVM を介して、プラットフォームに依存する適切な処理へと翻訳されます。

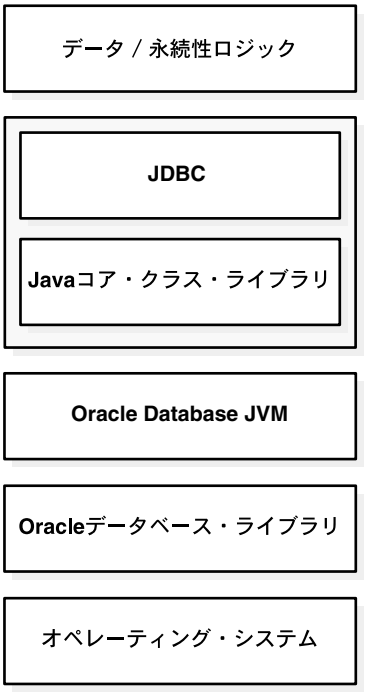
図 25-6 Java コンポーネント構造



Java プログラムの開発では、Java 言語で記述された事前定義のコア・クラス・ライブラリを使用します。Java コア・クラス・ライブラリは、基本言語サポート（java.lang）、I/O（java.io）およびネットワーク・アクセス（java.net）など、一般的な機能を提供するパッケージに論理的に分割されています。JVM とコア・クラス・ライブラリにより、Java プログラマの開発するプログラムが、Java をサポートするすべてのハードウェアおよびオペレーティング・システムで、確実に実行できるようなプラットフォームが提供されます。この概念は、「1 回プログラミングすれば、どこでも実行可能」という Java の考え方を推進するものです。

図 25-7 に、Oracle の Java アプリケーションが Java コア・クラス・ライブラリの最上位に配置され、さらに、JVM の最上位に配置される様子を示します。Oracle の Java サポート・システムはデータベース内に配置されるので、JVM はオペレーティング・システムと直接対話するかわりに、Oracle Database ライブラリと対話します。

図 25-7 Java コンポーネント構造



Sun 社により、Java 言語および JVM の仕様が提供されます。Java 言語仕様（JLS）で構文とセマンティクスなどが定義され、JVM 仕様でバイトコードを実行するコンピュータに必要な下位レベルの動作が定義されます。さらに、Sun 社からは、JVM の実装者が仕様に従ってコンパイルしたかどうかを判断するための互換性テスト一式が提供されます。このテスト一式は、Java Compatibility Kit（JCK）と呼ばれます。Oracle の JVM 実装は、JCK に完全に準拠しています。Java 方針の一部は、公開された仕様規格と、その企画への準拠を検証する簡単な方法とで構成されており、これにより、ベンダーはすべてのプラットフォーム上の Java に対し均一なサポートを提供できます。

Oracle Database で Java を使用する理由

データベースで Java アプリケーションを記述およびロードできるのは、Java が安全な言語であるためです。Java は、Java コードが格納されているオペレーティング・システムの改ざんを防止します。C など、一部の言語では、データベースにセキュリティ上の問題が生じる可能性があります。Java の場合は、その設計上、データベース内で容認できる安全な言語です。

Java は開発者にとって多数の利点がありますが、Java サーバー・アプリケーションをサポートする JVM をスケーラブルな方法で実装するのは難問です。この項では、これらの難問について説明します。

- マルチスレッド
- 自動記憶域管理
- フットプリント
- パフォーマンス
- 動的クラス・ロード

マルチスレッド

マルチスレッドのサポートは、通常、Java の主要なスケーラビリティ機能の 1 つと見られています。特に、Java を使用すると、Java 言語およびクラス・ライブラリにより、他の多数の言語に比べて共有サーバー・アプリケーションの作成が簡素化されます。しかし、どのような言語でも、信頼性が高いスケーラブルな共有サーバー・コードを記述するのは困難な作業です。

Oracle Database は、データベース・サーバーとして多数のユーザーの作業を効率的にスケジュールします。Oracle JVM は RDBMS サーバーの機能を使用して、多数のユーザーの Java 実行スケジュールを同時に設定します。Oracle Database は JLS と JCK に必要な Java 言語レベルのスレッドをサポートしていますが、データベースの有効範囲内でスレッドを使用するとスケーラビリティは向上しません。データベースの埋込みスケーラビリティを使用すると、共有サーバーの Java サーバーを記述する必要がなくなります。ユーザーのスケジューリングには、シングル・スレッド Java アプリケーションを作成してデータベースの機能を使用する必要があります。データベースが各アプリケーション間のスケジューリングを受け持つため、スレッドを管理しなくてもスケーラビリティが実現します。共有サーバーの Java アプリケーションを作成することもできますが、複数の Java スレッドを使用してもサーバーのパフォーマンスは向上しません。

マルチスレッドにより Java に生じる難問の 1 つは、スレッドと自動記憶域管理、つまりガベージ・コレクションとの間の相互作用です。汎用 JVM で実行されるガベージ・コレクタでは、どの Java 言語スレッドが実行されているか、あるいは基礎となるオペレーティング・システムでどのようにスケジュールされているかは認識されません。

- 非 Oracle モデル : シングル・ユーザーが単一の Java 言語レベル・スレッドにマップされ、同じ単一ガベージ・コレクタによって全ユーザーからのガベージがすべて管理されます。通常、存続期間やサイズの異なるオブジェクトの割当てと収集は、異なる技法で処理されます。頻繁に共有されるサーバー・アプリケーションの場合、結果はオペレーティング・システムがネイティブ・スレッドをサポートしているかどうかに応じて異なり、信頼性が低下したり、スケーラビリティが限定される可能性があります。この種の実装の場合、高水準のスケーラビリティは十分には実証されていません。
- Oracle JVM モデル : 数千のユーザーがサーバーに接続して同じ Java コードを実行する場合も、各ユーザーは各自の Java 仮想マシンで独自の Java コードを実行しているように感じます。Oracle JVM の役割は、Oracle RDBMS のスケーラブルなアプローチを使用して、オペレーティング・システムのプロセスとスレッドを利用することです。このアプローチの結果、一度に複数のユーザーからガベージが収集されることがなくなるため、JVM のガベージ・コレクタの信頼性と効率が向上します。

自動記憶域管理

ガベージ・コレクションは Java の自動記憶域管理の主機能であり、Java 開発者はメモリーの割当てと解放を明示的に行う必要がなくなります。そのため、C や C++ のプログラムで一般に問題となるメモリー・リークの大きな発生源が排除されます。この利点は代価も伴います。つまり、ガベージ・コレクションはプログラムの実行スピードとフットプリントのオーバーヘッドに影響します。このトレードオフを定性的 / 定量的に評価する多数の論文が記述されていますが、全体のコストは代替案と比較して妥当です。

ガベージ・コレクションは、高度な拡張性を持つ高速の Java プラットフォームを提供しようとする JVM 開発者に難問を課します。Oracle JVM は、これらの難問に次の方法で対処します。

- Oracle JVM は、複数のユーザーを効率的に管理できる Oracle Database のスケジューリング機能を使用します。
- ガベージ・コレクションは単一セッションで単一ユーザーに対してフォーカスされ、複数ユーザーにも一貫して実行されます。Oracle JVM の場合、ユーザー数が増加してもメモリー管理者のジョブの煩雑さや複雑さは変わらないため、大きな利点が得られます。メモリー管理者は単一セッションでオブジェクトの割当てと収集を実行し、通常はこれが単一ユーザーのアクティビティに変換されます。
- Oracle JVM は、使用するメモリーのタイプに応じて異なるガベージ・コレクション技法を使用します。これらの技法により、効率が向上し、オーバーヘッドが低減します。

フットプリント

Java プログラムの実行によるフットプリントは、次のように様々な要因の影響を受けます。

- プログラム自体のサイズ : クラス数、メソッド数およびそこに含まれるコードの量。
- プログラムの複雑さ : プログラム自体とは対照的に、プログラムの実行時に Oracle JVM が使用するコア・クラス・ライブラリの量。
- JVM が使用する状態の量 : JVM が割り当てるオブジェクトの数、そのサイズおよびコール間で保持する必要のある数。
- ガベージ・コレクタとメモリー管理者が実行中のプログラムの需要を処理する能力。通常、この能力は非決定論的です。オブジェクトが割り当てられるスピードと、他のオブジェクトによって保持される方法は、この要因の重要度に影響します。

スケーラビリティの観点から、多数の同時クライアントをサポートするには、ユーザー・セッションのフットプリントを最小限に抑えることが重要です。Oracle JVM は、Java バイト・コードなど、ユーザー用のすべての読取り専用データを共有メモリーに格納することで、ユーザー・セッションのフットプリントを最小限に抑えます。ユーザー・セッションのフットプリントが大きくならないように、コールとセッションのメモリーに対して、適切なガベージ・コレクション・アルゴリズムが適用されます。Oracle JVM は、次の 3 種類のガベージ・コレクション・アルゴリズムを使用して、ユーザーのセッション・メモリーをメンテナンスします。

- 存続期間の短いオブジェクトの場合はジェネレーション・スキャベンジ
- 単一コールの間に存在するオブジェクトの場合は、マーク / スイープ・コレクション
- 存続期間の長いオブジェクト、つまり、セッション中の複数コールにまたがって存在するオブジェクトの場合は、コレクタのコピー

パフォーマンス

Oracle JVM のパフォーマンスは、システム固有のコンパイラを実装することで拡張されます。Java は、JVM の最上位でプラットフォームに依存しないバイトコードを実行し、バイトコードは特定のハードウェア・プラットフォームと相互作用します。ソフトウェア内でレベルを追加すると、パフォーマンスが低下します。Java では、プラットフォームに依存しないバイトコードを解析するために仲介部分を通過する必要があるため、C のようにプラットフォーム依存言語に存在しない Java アプリケーションの場合は、ある程度の非効率が生じます。この問題に対処するために、複数の JVM サプライヤがシステム固有のコンパイラを作成しています。システム固有のコンパイラは、Java バイトコードをプラットフォームに依存するシステム固有のコードに変換します。これにより、解析処理が不要になり、パフォーマンスが改善されます。

表 25-1 に、システム固有の 2 つのコンパイル方法を示します。

表 25-1 システム固有のコンパイル方法

メソッド	説明
Just-In-Time（JIT）コンパイル	JIT コンパイラは、実行時に Java バイトコードをシステム固有（プラットフォーム固有）のコンピュータ・コードにすばやくコンパイルします。この場合、プラットフォームで実行される実行可能ファイルが生成されるかわりに、変換後に直接実行される Java バイトコードからのプラットフォーム依存コードが提供されます。頻繁に実行する Java コードには、この方法を使用する必要があります。この実行スピードは、C などの言語とほぼ同じです。
静的コンパイル	静的コンパイルでは、Java バイトコードが実行前にプラットフォームに依存しない C コードに変換されます。標準的な C コンパイラは、C コードをターゲット・プラットフォーム用の実行可能ファイルにコンパイルします。あまり変更されない Java アプリケーションには、このアプローチが適しています。このアプローチでは、最近の C コンパイラに見られる成熟して効率的なプラットフォーム固有のコンパイル技法が利用されています。

Oracle Database は、静的コンパイルを使用して、コア Java クラス・ライブラリである ORB および JDBC コードをコンパイル済の形式で提供します。これらのコードは、Oracle がサポートしているすべてのプラットフォーム間で適用できますが、JIT アプローチの場合は、プラットフォームごとに下位レベルのプロセッサ依存コードを記述してメンテナンスする必要があります。このシステム固有のコンパイル技法は、独自の Java コードに使用できます。

動的クラス・ロード

Java のもう 1 つの強力な機能は、動的クラス・ロードです。クラス・ローダーは、クラスがプログラムの実行中に使用される場合にのみディスクからロード（および解釈に必要な JVM 固有のメモリー構造に格納）されます。また、クラスを CLASSPATH 内で検索し、プログラムの実行中にロードします。このアプローチはアプレットにも使用できますが、サーバー環境では表 25-2 に示す問題を伴います。

表 25-2 動的クラス・ロードに伴う問題

問題	説明	解決策
予測可能性	クラス・ロード操作を初めて実行する場合は、重大なペナルティを伴います。単純なプログラムの場合は、Oracle JVM にそのニーズをサポートするための多数のコア・クラスがロードされる可能性があります。ロードされるクラスの数を、プログラマが簡単に予測または判断することはできません。	Oracle JVM は、他の Java 仮想マシンと同様に、クラスを動的にロードします。ワンタイム・クラス・ロードのスピードは同じです。ただし、クラスは共有メモリーにロードされるため、そのクラスの他のユーザーはクラスを再ロードする必要はなく、単に同じ事前ロード・クラスを使用します。
信頼性	動的クラス・ロードの利点は、プログラムの更新がサポートされることです。たとえば、サーバー上でクラスを更新すると、プログラムをダウンロードして動的にロードするクライアントでは、次回にそのプログラムを使用するときに更新内容が表示されます。サーバー・プログラムは、信頼性を強調する傾向があります。開発者は、各クライアントが特定のプログラム構成を実行することを知っておく必要があります。ロードする意図のない一部のクラスがクライアントによって意図せずにロードされないようにしてください。	Oracle Database では、アップロードおよび解決操作が、実行時にクラス・ロード操作と分離されます。開発した Java コードをサーバーにアップロードするには、loadjava ユーティリティを使用します。CLASSPATH を使用するかわりに、インストール時にリゾルバを指定します。リゾルバは CLASSPATH に似ていますが、クラスがあるスキーマを指定する必要があります。このように解決をクラス・ロードから分離することで、プログラム・ユーザーによる実行内容を常に把握できます。

Oracle の Java アプリケーション方針

Java の魅力は、ユビキタスと、それをアプリケーション開発に使用できるプログラマの数が増加していることです。オラクル社は、エンタープライズ・アプリケーションの開発者に、Java アプリケーションの作成、デプロイおよび管理のためのエンドツーエンドの Java ソリューションを提供します。この総合的なソリューションは、クライアント側とサーバー側のプログラム・インタフェース、Java 開発をサポートするためのツール、Oracle Database と統合された Java 仮想マシンで構成されています。これらの製品はいずれも、Java 標準との互換性を持っています。

Java プログラミング環境には、Oracle JVM に加えて次のコンポーネントがあります。

- PL/SQL に等価および相当する Java ストアド・プロシージャ。Java ストアド・プロシージャは、PL/SQL と緊密に統合されています。PL/SQL パッケージから Java ストアド・プロシージャをコールしたり、PL/SQL プロシージャを Java ストアド・プロシージャからコールできます。
- JDBC プログラミング・インタフェースを介して SQL データにアクセスできます。
- 開発、クラス・ロードおよびクラス管理の支援に使用されるツールとスクリプト。

この項の内容は、次のとおりです。

- Java ストアド・プロシージャ
- PL/SQL の統合と Oracle Database の機能
- JDBC
- JPublisher
- Java Message Service

Java ストアド・プロシージャ

Java ストアド・プロシージャは、PL/SQL ストアド・プロシージャと同様に、サーバーで実行するための Java で記述するプログラムです。Java ストアド・プロシージャには、SQL*Plus などの製品で直接起動する方法と、トリガーを使用して間接的に起動する方法があります。また、OCI、プリコンパイラまたは JDBC など、すべての Oracle Net クライアントからアクセスできます。

また、Java を使用して、PL/SQL に依存しない強力なプログラムを開発することもできます。Oracle Database には、Java プログラミング言語と JVM の完全準拠の実装が用意されています。

> **関連項目：** Java でストアド・プロシージャを記述する方法、PL/SQL からアクセスする方法および Java から PL/SQL 機能にアクセスする方法は、『Oracle Database Java 開発者ガイド』を参照してください。

PL/SQL の統合と Oracle Database の機能

既存の PL/SQL プログラムを Java から起動したり、Java プログラムを PL/SQL から起動できます。このソリューションでは、Java ベースのインターネット・コンピューティングの利点と機会を活用しながら、既存の資産を保護して利用できます。

JDBC

Java Database Connectivity（JDBC）は、SQL データにアクセスする Java 開発者向けの Application Program Interface（API）です。この API はクライアントとサーバーで使用できるため、同じコードをどちらにでもデプロイできます。

Oracle の JDBC では、動的 SQL を通じて、Java プログラムからデータベース内で定義されたオブジェクト型とコレクション型にアクセスできます。動的 SQL とは、実行される埋込み SQL 文がアプリケーションの実行前には認識されず、文の作成には入力が必要であることを意味します。この機能は、データベース内で定義されている型を、デフォルトまたはカスタマイズ可能なマッピングを通じて Java のクラスに変換します。また、Java および J2EE アプリケーションによるリソース使用を監視およびトレースし、データベース操作レベルに相関付けることもできます。

コア Java クラス・ライブラリには、JDBC API が 1 つのみ用意されています。ただし、JDBC は、ベンダーが特定のデータベースを必要に応じて特化するためのドライバを提供できるように設計されています。Oracle には、次の 3 種類の JDBC ドライバが用意されています。

表 25-3 JDBC ドライバ

ドライバ	説明
JDBC Thin ドライバ	JDBC Thin ドライバを使用すると、Oracle SQL のデータにアクセスする 100% の Pure Java アプリケーションおよびアプレットを作成できます。JDBC Thin ドライバは、他の Java アプレットと同様に Web ページから動的にダウンロードできるため、Web ブラウザ・ベースのアプリケーションとアプレットに特に適しています。
JDBC Oracle Call Interface ドライバ	JDBC Oracle Call Interface（OCI）ドライバは、クライアントまたは中間層にある Oracle 固有のネイティブ・コード（つまり非 Java）ライブラリにアクセスし、JDBC Thin ドライバに比べて豊富な機能セットとパフォーマンス向上をもたらしますが、サイズが極端に大きくなることと、クライアント側でのインストールというコストの問題があります。
サーバー側 JDBC 内部ドライバ	Oracle Database は、Java コードがサーバー上で実行される場合に、サーバー側内部ドライバを使用します。これにより、サーバーの JVM 内で実行中の Java アプリケーションは、JDBC を使用してローカルに定義されたデータ（つまり、同じコンピュータの同じプロセスにあるデータ）にアクセスできます。また、基礎となる Oracle RDBMS ライブラリを直接使用できるため、さらにパフォーマンスが向上し、Java コードと SQL データの間にネットワーク接続が介入することによるオーバーヘッドは発生しません。Oracle Database では、サーバー上で同じ Java-SQL インタフェースをサポートすることで、デプロイ時にコードの再処理を必要としません。

関連項目：

- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- JDBC プログラムの例は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

SQLJ

SQLJ を使用すると、開発者は Java プログラムでオブジェクト・データ型を使用できます。開発者は、JPublisher を使用して Oracle のオブジェクト型とコレクション型を Java クラスにマップし、アプリケーションで使用できます。

SQLJ では、Java コードに埋め込まれた SQL 文を使用してサーバー・オブジェクトにアクセスします。SQLJ は、SQL 文のオブジェクト型とコレクションのコンパイル時型チェック機能を提供します。構文は、ANSI 規格（SQLJ Consortium）に準拠しています。

Java クラスは、SQL のユーザー定義オブジェクト型として指定できます。この SQLJ 型の列または行を定義できます。また、この型のオブジェクトは、SQL プリミティブ型であるかのように問い合せて操作できます。さらに、次のことができます。

- クラスの静的フィールドを SQL で参照できるようにします。
- ユーザーによる Java コンストラクタのコールを可能にします。
- Java クラスとその対応する型との間の依存性を維持します。

JPublisher

Java Publisher（JPublisher）は、完全に Java で記述されたユーティリティで、次のユーザー定義データベース・エンティティを表す Java クラスを Java プログラムで生成します。

- SQL オブジェクト型
- オブジェクト参照型（REF 型）
- SQL コレクション型（VARRAY 型またはネストした表型）
- PL/SQL パッケージ

JPublisher によって、これらのエンティティから Java クラスへのマッピングを強い型指定のパラダイムで指定およびカスタマイズできます。

関連項目：『Oracle Database JPublisher ユーザーズ・ガイド』

Java Message Service

Java Message Service（JMS）は、Sun 社が Oracle、IBM および他のベンダーと共同開発したメッセージ処理標準です。JMS では、JMS アプリケーション用の一連のインタフェースが定義され、JMS プロバイダが実装する動作が指定されています。JMS には標準ベースの API が用意されており、社内および顧客やパートナとの間でビジネス・イベントを非同期に交換できるようにします。JMS により、分散環境の疎結合コンポーネント間で信頼性の高い通信が容易になり、企業の統合に必要な作業が大幅に簡素化されます。Java テクノロジとエンタープライズ・メッセージ処理を組み合せて、移植性の高いアプリケーションを開発できます。

Oracle Java Message Service は、JMS 標準に準拠した Oracle Streams 用の Java API です。複数のクライアント・アプリケーションでは、中央の JMS プロバイダ（Oracle Streams）を介して、あらゆるタイプのメッセージが送受信できます。JMS クライアントは、Java アプリケーションとメッセージ・クライアント・ランタイム・ライブラリで構成され、このランタイム・ライブラリが JMS インタフェースを実装して Oracle Streams と通信します。

Java メッセージ用の Oracle JMS は標準 JMS インタフェースをサポートしており、標準に含まれていない他の Oracle Streams 機能をサポートするための拡張機能も用意されています。Oracle JMS を使用すると、Oracle Streams で使用可能なキューにメッセージをエンキューおよびデキューできます。Oracle JMS の標準 JMS 機能は、次のとおりです。

- キューを使用した 2 点間通信
- トピックを使用したパブリッシュ・サブスクライブ通信
- 同期および非同期のメッセージ交換
- サブジェクト・ベースのルーティング

Oracle Streams には、標準 JMS 機能に対する拡張機能も用意されています。

- メッセージを受信するアプリケーションを指定する受信者リストを使用した point-to-multipoint Communication/point-to-multipoint 通信。
- キュー表、キューおよびサブジェクトを作成するための管理 API。
- 異なるデータベースにあるキュー間でのメッセージの自動伝播。これにより、アプリケーションでリモート・サブスクライバを定義できます。
- トランザクションを実行したセッションのサポート。1 つのトランザクションで JMS 操作と SQL 操作の両方を実行できます。
- 使用後のメッセージの保存。
- 例外処理。
- メッセージが参照可能になるまでの遅延の指定。

Microsoft プログラミング言語の概要

Oracle には、Visual Basic や Active Server Page など、COM ベースのプログラミング言語からの様々なデータ・アクセス方式が用意されています。たとえば、Oracle Objects for OLE（OO4O）や Oracle Provider for OLE DB があります。後者は、Microsoft 社の ActiveX Data Objects（ADO）で使用できます。Microsoft Office など、COM オートメーション・サーバーに対するサーバー側プログラミングは、COM オートメーション機能を介して使用可能です。さらに従来型の ODBC アクセスは、Oracle の ODBC ドライバを介して使用可能です。C/C++ アプリケーションでは、Oracle Call Interface（OCI）も使用できます。これらのデータ・アクセス・ドライバは、Oracle Database で優れたパフォーマンスを発揮するように設計されており、サード・パーティのドライバでは使用できないデータベースの拡張機能を提供します。

また、Oracle は Oracle Data Provider for .NET を介して最適の .NET データ・アクセス・サポートを提供し、.NET から Oracle 拡張機能にアクセスできるようにします。さらに、OLE DB .NET と ODBC .NET もサポートしています。

この項の内容は、次のとおりです。

- Open Database Connectivity
- Oracle Objects for OLE の概要
- Oracle Data Provider for .NET

Open Database Connectivity

Open Database Connectivity（ODBC）は、データベースに接続し、データベースに対して SQL 文を準備して実行するためのデータベース・アクセス・プロトコルです。アプリケーションを ODBC ドライバと組み合せることで、Excel のようなスプレッドシートに格納されているデータなど、あらゆるデータ・ソースにアクセスできます。ODBC は普及している標準 API であるため、ODBC 標準に準拠するアプリケーションを記述できます。ODBC ドライバは、ODBC 標準とアプリケーションがアクセスする特定のデータベースの間のマッピングをすべて実行します。ODBC 準拠のプログラムは、どのデータ・ソースへのアクセスにもデータ・ソース固有のドライバをそのまま使用でき、追加の開発作業を必要としません。

Oracle には ODBC インタフェースが用意されているため、ODBC 準拠であれば、どのようなアプリケーションからも Oracle の ODBC ドライバを使用して Oracle Database データベースにアクセスできます。たとえば、Visual Basic で記述されたアプリケーションでは、ODBC を使用して Oracle Database にアクセスできます。

Oracle Objects for OLE の概要

Oracle Objects for OLE（OO4O）を使用すると、Microsoft 社の COM オートメーションおよび ActiveX テクノロジをサポートしているプログラミング言語またはスクリプト言語を使用して、Oracle Database に格納されているデータに容易にアクセスできます。これには、Visual Basic、Visual C++、Visual Basic For Applications（VBA）、IIS Active Server Pages（VBScript および JavaScript）などが含まれます。

OO4O は次のソフトウェア・レイヤーで構成されています。

- OO4O オートメーション・サーバー
- Oracle Data Control
- Oracle Objects for OLE の C++ クラス・ライブラリ

OO4O オートメーション・サーバー

OO4O オートメーション・サーバーは、Oracle Database サーバーに接続して SQL 文と PL/SQL ブロックを実行し、結果にアクセスするための COM オートメーション・オブジェクトの集合です。

OO4O は、Visual Basic や Excel などで開発された典型的な 2 層クライアント / サーバー・アプリケーション、Microsoft Internet Information Server（IIS）や Microsoft Transaction Server における Web サーバー・アプリケーションのように複数層アプリケーション・サーバー環境にデプロイされたアプリケーション・サーバーなど、様々な環境で Oracle Database にアクセスするための主機能を備えています。

Oracle Data Control

Oracle Data Control（ODC）は、Oracle Database とビジュアル・コントロールとのデータ交換を簡素化するために設計された ActiveX Control です。ビジュアル・コントロールには、Visual Basic やカスタム・コントロールをサポートする他の開発ツールの編集、テキスト、リストおよびグリッド・コントロールなどがあります。

ODC は、Oracle Database からの情報の流れを処理するエージェントとして機能し、それにバインドされているグリッド・コントロールなどの視覚的データ認識コントロールとしても機能します。データ・コントロールでは、データの表示や編集などの各種ユーザー・インタフェース（UI）タスクが管理されます。また、データベース問合せも実行され、結果が管理されます。

関連項目：『Oracle Objects for OLE 開発者ガイド』

Oracle Objects for OLE の C++ クラス・ライブラリ

Oracle Objects for OLE の C++ クラス・ライブラリは、Oracle Object Server へのプログラム的なアクセスを提供する C++ クラスのコレクションです。このクラス・ライブラリは OLE オートメーションを使用して実装されますが、使用するのに OLE 開発キットも OLE 開発に関する知識も必要ありません。このライブラリにより、C++ 開発者は OO4O インタフェースにアクセスする COM クライアント・コードを記述せずに済みます。

関連項目：「スタート」メニューから、Oracle Objects for OLE の C++ クラス・ライブラリのヘルプを参照してください。

Oracle Data Provider for .NET

Oracle Data Provider for .NET（ODP.NET）は、Oracle Database 用のデータ・プロバイダの実装です。ODP.NET は Oracle システム固有の API を使用して、あらゆる .NET アプリケーションから Oracle Database のデータおよび機能への高速かつ信頼性の高いアクセスを提供します。また、Microsoft .NET Framework クラス・ライブラリで使用可能なクラスとインタフェースを使用し、継承します。

.NET Framework と同じように、ODP.NET では、ネイティブ・プロバイダによるプロバイダ固有の機能とデータ型の公開を可能にする ADO.NET モデルが使用されます。

ODP.NET を使用すると、開発者は Visual Basic .NET、C# およびその他の .NET 言語でプログラムを記述できます。

関連項目：『Oracle Data Provider for .NET 開発者ガイド』

レガシー言語の概要

この項の内容は、次のとおりです。

Pro*COBOL プリコンパイラの概要

Pro*COBOL プリコンパイラは、ホスト COBOL プログラムに SQL 文を埋め込むためのプログラミング・ツールです。Pro*COBOL は入力としてソース・ファイルを読み取り、COBOL ソース・ファイルを出力します。このソース・ファイルでは、埋込み SQL 文が Oracle ランタイム・ライブラリ・コールで置換されてから、COBOL コンパイラによりコンパイルされます。

Pro*C/C++ プリコンパイラと同様に、Pro*COBOL を使用すると高度にカスタマイズされたアプリケーションを作成できます。たとえば、最新のウィンドウおよびマウス・テクノロジを取り込むユーザー・インタフェースを作成できます。また、バックグラウンドで実行されてユーザーの介入を必要としないアプリケーションも作成できます。

さらに、Pro*COBOL ではアプリケーションの詳細なチューニングが可能です。リソース使用、SQL 文の実行および各種のランタイム・インジケータを緊密に監視できます。この情報を使用してプログラムのパラメータを調整し、パフォーマンスを最大化します。

関連項目：『Pro*COBOL プログラマーズ・ガイド』

Pro*FORTRAN プリコンパイラの概要

Oracle Pro*FORTRAN プリコンパイラを使用すると、ホスト FORTRAN プログラムに SQL を埋め込むことができます。

Pro*FORTRAN は、Windows ではサポートされません。

関連項目：『Pro*FORTRAN Supplement to the Oracle Precompilers Guide』

26

Oracle データ型

この章では、Oracle 組込みデータ型とその特性、および Oracle データ型を Oracle 以外のデータ型にマップする方法について説明します。

この章の内容は、次のとおりです。

Oracle データ型の概要

SQL 文中の列値と定数は、それぞれ特定の記憶域形式、制約および値の有効範囲に対応付けられた**データ型**を持っています。表の作成時には、その各列のデータ型を指定する必要があります。

Oracle の組込みデータ型は、次のカテゴリに分かれています。

- 文字データ型の概要
- 数値データ型の概要
- DATE データ型の概要
- LOB データ型の概要
- RAW および LONG RAW データ型の概要
- ROWID および UROWID データ型の概要

注意： PL/SQL には、定数と変数のその他のデータ型として、`BOOLEAN`、参照型、コンポジット型（コレクションとレコード）およびユーザー定義サブタイプがあります。

関連項目：

- PL/SQL データ型の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- 組込みデータ型の使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

次の各項では、各組込みデータ型についてさらに詳しく説明します。

文字データ型の概要

文字データ型は、文字コード体系（通常、キャラクタ・セットまたはコード・ページと呼ばれます）に対応するバイト値によって、文字（英数字）データを文字列に格納します。

データベースのキャラクタ・セットは、データベースの作成時に設定されます。キャラクタ・セットには、7 ビット ASCII（情報交換用米国標準コード）、EBCDIC（拡張 2 進化 10 進コード）、コード・ページ 500、日本語拡張 UNIX コード、Unicode UTF-8 などがあります。Oracle はシングルバイトおよびマルチバイト・コード体系をサポートしています。

関連項目：

- 文字データ型を選択する方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 文字データ変換の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

この項の内容は、次のとおりです。

CHAR データ型

CHAR データ型は、固定長の文字列を格納します。CHAR 列を含む表を作成するときは、CHAR 列の文字列の長さを 1 ～ 2000 までの値で指定する必要があります（単位はバイト数または文字数）。デフォルトは 1 バイトです。Oracle によって、次のことが保証されます。

- 表に行を挿入したり更新するとき、CHAR 列の値は固定長になります。
- 短い値を与えると、固定長にあわせて空白が埋め込まれます。
- 値が大きすぎると、Oracle Database からエラーが戻されます。

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

> **関連項目：** 空白埋め比較方法の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

VARCHAR2 および VARCHAR データ型

VARCHAR2 データ型には、可変長の文字列が格納されます。VARCHAR2 列を含む表を作成するときは、VARCHAR2 列の文字列の最大長を 1 ～ 4000 までの値で指定します（単位はバイト数または文字数）。各行の値は、Oracle Database により可変長フィールドとして列に格納されます。値が列の最大長を超えている場合は、Oracle Database からエラーが戻されます。VARCHAR2 と VARCHAR を使用すると、表が使用する空白を節約できます。

たとえば、列 VARCHAR2 を最大サイズ 50 文字で宣言するとします。シングルバイト・キャラクタ・セットで、特定の行の VARCHAR2 列に 10 文字を入れると、その行断片の列には 50 文字ではなく 10 文字（10 バイト）のみが格納されます。

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

> **関連項目：** 非空白埋め比較方法の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

VARCHAR データ型

VARCHAR データ型は、VARCHAR2 データ型と同義です。このため、考えられる動作変更を回避するために、可変長の文字列の格納には常に VARCHAR2 データ型を使用してください。

文字データ型の長さセマンティクス

グローバリゼーション・サポートは、文字データ型に様々なキャラクタ・セットを使用できるようにします。グローバリゼーション・サポートを使用すると、シングルバイトおよびマルチバイト・キャラクタ・セットの処理や、それらのキャラクタ・セット間での変換が可能になります。クライアント・セッションでは、データベース・キャラクタ・セットと異なるクライアント・キャラクタ・セットを使用できます。

文字データ型の列の長さを指定する場合は、文字のサイズを考慮してください。この問題は、文字データを含む列を持つ表の領域を見積るときに考慮する必要があります。

文字データ型の長さセマンティクスは、バイト数または文字数で測定できます。

- **バイト・セマンティクス**は、文字列を一連のバイトとして取り扱うことを意味します。これは、文字データ型のデフォルトです。
- **キャラクタ・セマンティクス**は、文字列を一連の文字として取り扱うことを意味します。文字は技術的に、データベース・キャラクタ・セットのコード・ポイントです。

シングルバイト・キャラクタ・セットの場合、キャラクタ・セマンティクスで定義された列は、バイト・セマンティクスで定義された列と基本的に同じです。キャラクタ・セマンティクスは、可変幅のマルチバイト文字列の定義に便利です。つまり、データ記憶域で実際に必要な長さを定義する際の複雑さを軽減します。たとえば、Unicode（UTF8）データベースで、VARCHAR2 列を定義する必要がある場合を考えます。この列には、5 つまでの漢字と 5 つの英字を格納できます。バイト・セマンティクスでは、（5 × 3 バイト）+（1 × 5 バイト）= 20 バイト必要です。キャラクタ・セマンティクスでは、列に 10 文字必要です。

VARCHAR2(20 BYTE) と SUBSTRB(*<string>*,1,20) は、バイト・セマンティクスを使用します。VARCHAR2(10 CHAR) と SUBSTR(*<string>*,1,10) は、キャラクタ・セマンティクスを使用します。

NLS_LENGTH_SEMANTICS パラメータによって、文字データ型の新しい列が、バイト・セマンティクスとキャラクタ・セマンティクスのどちらを使用するのかが決まります。デフォルトの長さセマンティクスはバイトです。データベースのすべての文字データ型列がバイト・セマンティクスを使用する（またはすべてキャラクタ・セマンティクスを使用する）場合、ユーザーはどの列がどちらのセマンティクスを使用するのかについて考慮する必要はありません。BYTE と CHAR の修飾子は、前述したように可能であれば使用しないでください。これは、これらの修飾子を使用すると、データベースのセマンティクスが混合されるためです。かわりに、サーバー・パラメータ・ファイル（SPFILE）または初期化パラメータ・ファイルで、NLS_LENGTH_SEMANTICS 初期化パラメータを適切に設定してください。列はデフォルトのセマンティクスを使用します。

関連項目：

- 26-6 ページの「Oracle Database での Unicode データの使用」
- Oracle のグローバリゼーション・サポート機能の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。
- 長さセマンティクスおよび適切な Unicode キャラクタ・セットの選択の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 既存の列をキャラクタ・セマンティクスへ移行する方法については、『Oracle Database アップグレード・ガイド』を参照してください。

NCHAR および NVARCHAR2 データ型

NCHAR と NVARCHAR2 は、Unicode 文字データを格納する Unicode データ型です。NCHAR と NVARCHAR2 データ型のキャラクタ・セットは、AL16UTF16 または UTF8 のいずれかのみです。また、データベースの作成時に各国語キャラクタ・セットとして指定されます。AL16UTF16 と UTF8 は、どちらも Unicode エンコーディングです。

- NCHAR データ型は、各国語キャラクタ・セットに対応する固定長の文字列を格納します。
- NVARCHAR2 データ型には、可変長の文字列が格納されます。

NCHAR または NVARCHAR2 列で表を作成すると、指定する最大サイズは、必ず文字長セマンティクス内になります。文字長セマンティクスは、デフォルトです。また、NCHAR または NVARCHAR2 の唯一の長さセマンティクスです。

たとえば、各国語キャラクタ・セットが UTF8 である場合、次の文は、最大バイト長を 90 バイトに定義します。

```
CREATE TABLE tab1 (col1 NCHAR(30));
```

この文は、最大文字長が 30 の列を作成します。最大バイト長は、最大文字長と各文字の最大バイト長から決まります。

この項の内容は、次のとおりです。

NCHAR

NCHAR 列の最大長は 2000 バイトです。2000 文字まで格納できます。実際のデータは、多くの場合、最大バイト数の 2000 になります。実行時には、同時に 2 つのサイズ制約を満たす必要があります。

NVARCHAR2

NVARCHAR2 列の最大長は 4000 バイトです。4000 文字まで格納できます。実際のデータは、多くの場合、最大バイト数の 4000 になります。実行時には、同時に 2 つのサイズ制約を満たす必要があります。

> **関連項目：** NCHAR および NVARCHAR2 データ型の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

Oracle Database での Unicode データの使用

Unicode は、ユーザーが使用するあらゆる言語のすべての文字を統一してエンコーディングしようとするものです。また、個人的に定義した文字を表す方法も提供します。Unicode を格納するデータベースの列は、あらゆる言語のテキストを格納できます。

グローバル化されたアプリケーションをデプロイする Oracle Database ユーザーは、Oracle Database に必ず Unicode データを格納する必要があります。データベースのキャラクタ・セットに関係なく、必ず Unicode に変換されるデータ型が必要です。

Oracle Database は、NCHAR、NVARCHAR2 および NCLOB によって、信頼できる Unicode データ型を実現しています。これらのデータ型は、Unicode エンコーディングへの変換が保証されているため、必ず文字長セマンティクスを使用します。NCHAR/NVARCHAR2 が使用するキャラクタ・セットは、UTF8 または AL16UTF16 のいずれかです。これは、データベースの作成時に設定される各国語キャラクタ・セットによって決まります。これらのデータ型では、データベースがデータベース・キャラクタ・セットとして Unicode を使用するしないにかかわらず、Unicode のキャラクタ・セットをそのデータベースに格納できます。

暗黙的な型変換

CHAR/VARCHAR2 に対するすべての暗黙的な変換に加え、Oracle Database は、NCHAR/NVARCHAR2 に対する暗黙的な変換もサポートしています。CHAR/VARCHAR2 と NCHAR/NVARCHAR2 間の暗黙的な変換もサポートされています。

LOB 文字データ型

文字データの LOB データ型は CLOB と NCLOB です。最大 8TB の文字データ（CLOB）または各国語キャラクタ・セット・データ（NCLOB）を格納できます。

関連項目： 26-12 ページの「LOB データ型の概要」

LONG データ型

注意： LONG 列を持つ表は作成しないでください。かわりに LOB 列（CLOB、NCLOB）を使用します。LONG 列は、下位互換性を維持するためにのみサポートされています。

また、既存の LONG 列を LOB 列に変換することをお薦めします。LOB 列に適用される制限は、LONG 列よりもはるかに少数です。さらに、LOB の機能性はリリースごとに拡張されますが、LONG の機能性はいくつかのリリースでは変化していません。

LONG と定義されている列には、最大 2GB までの情報を含む可変長の文字データを格納できます。LONG データは、異なるシステム間で移動しても適切に変換されるテキスト・データです。

LONG データ型の列は、ビュー定義のテキストを格納するために、データ・ディクショナリで使用されます。SELECT リストの LONG 列、UPDATE 文の SET 句、および INSERT 文の VALUES 句を使用できます。

関連項目：

- LONG データ型の制限の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- LONG RAW データ型の詳細は、26-14 ページの「RAW および LONG RAW データ型の概要」を参照してください。

数値データ型の概要

数値データ型には、正と負の固定小数点数と浮動小数点数、ゼロ、無限大および操作の未定義の結果である値（非数値つまり NAN）が格納されます。

この項の内容は、次のとおりです。

NUMBER データ型

NUMBER データ型には、固定小数点数および浮動小数点数が格納されます。事実上どんな大きさの数値でも格納可能です。Oracle Database が稼働するシステムであれば、最大 38 桁の精度で、システム間の移植性が保証されています。

NUMBER 列には、次の数値を格納できます。

- 1×10^{-130} ～ $9.99...9 \times 10^{125}$ の範囲の正の数。最大有効桁数は 38 です。
- -1×10^{-130} ～ $9.99...99 \times 10^{125}$ の範囲の負の数。最大有効桁数は 38 です。
- 0（ゼロ）。
- 正と負の無限大（Oracle Database バージョン 5 からのインポート時にのみ生成されます）。

数値列の場合は、次のように列を指定できます。

```
column_name NUMBER
```

また、任意で次のように**精度**（全体の桁数）と**スケール**（小数点以下の桁数）を指定することもできます。

```
column_name NUMBER (precision, scale)
```

精度を指定しないと、値は与えられたとおりに列に格納されます。スケールを指定しないと、スケールは 0（ゼロ）になります。

Oracle では、38 桁以下の精度で数値の移植性が保証されています。次のように、スケールのみを指定して、精度は指定しないこともできます。

```
column_name NUMBER (*, scale)
```

この場合、精度は 38 になり、指定したスケールが保持されます。

数値フィールドを指定する場合には、精度とスケールを指定する方法が優れています。これによって、入力の整合性をさらにチェックできます。

表 26-1 に、様々なスケール変更係数がデータの格納にどのように影響するかを示します。

表 26-1 スケール変更係数が数値データの格納に与える影響

入力データ	指定	格納方法
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	（精度を超えているため、受け入れられない）
7,456,123.89	NUMBER(7,-2)	7456100

負のスケールを指定すると、Oracle Database により小数点の左側の指定した桁数で実際のデータが丸められます。たとえば、(7,-2) という指定は、表 26-1 に示されているとおり、Oracle Database によって 100 の位で丸めることを意味します。

数値の入力および出力において、標準 Oracle Database のデフォルト小数点文字はピリオドです。たとえば、1234.56 というようになります。小数点文字は、数値の整数部と小数部を分離する文字です。デフォルトの小数点文字は、初期化パラメータ NLS_NUMERIC_CHARACTERS を使用して変更できます。また、セッションの存続中の小数点文字は、ALTER SESSION 文でも変更できます。現行のデフォルト小数点文字を使用していない数値を入力するには、TO_NUMBER 関数を使用します。

内部数値形式

Oracle Database では、数値データが可変長形式で格納されます。それぞれの値は科学表記法で格納され、指数を格納するために 1 バイト、仮数を格納するために最大 20 バイトが使用されます。結果の数値の精度は 38 桁に制限されます。Oracle Database では、先行ゼロと後続ゼロは格納されません。たとえば、数値 412 は 4.12×10^2 という形式で格納され、指数（2）を格納するために 1 バイト、仮数の 3 桁の有効数字（4、1、2）を格納するために 2 バイトが使用されます。負数の長さには、符号が含まれます。

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

```
ROUND((length(p)+s)/2))+1
```

数値が正数であれば *s* は 0（ゼロ）となり、数値が負数であれば *s* は 1 となります。

0（ゼロ）および正と負の無限大（Oracle Database バージョン 5 からのインポートによってのみ生成されます）は、一意の表現形式を使用して格納します。0（ゼロ）と負の無限大は、それぞれ 1 バイトを必要とします。正の無限大は 2 バイトを必要とします。

浮動小数点数

Oracle Database には、浮動小数点数専用に BINARY_FLOAT および BINARY_DOUBLE という 2 つの数値データ型が用意されています。この 2 つのデータ型は、NUMBER データ型の基本機能をすべてサポートしています。ただし、NUMBER は 10 進精度を使用しますが、BINARY_FLOAT および BINARY_DOUBLE は 2 進精度を使用します。これにより、算術計算が高速になり、通常は記憶域必要量が減少します。

BINARY_FLOAT と BINARY_DOUBLE は、近似値データ型です。どちらにも、10 進値の正確な表現ではなく近似値表現が格納されます。たとえば、BINARY_DOUBLE または BINARY_FLOAT では、値 0.1 を正確に表現できません。この 2 つは、通常は科学関係の計算に使用されます。両者の動作は、Java と XMLSchema のデータ型 FLOAT および DOUBLE に似ています。

この項の内容は、次のとおりです。

- BINARY_FLOAT データ型
- BINARY_DOUBLE データ型

BINARY_FLOAT データ型

BINARY_FLOAT は、32 ビット単精度の浮動小数点数データ型です。BINARY_FLOAT 値には、それぞれ長さを格納する 1 バイトを含めて 5 バイト必要です。

BINARY_DOUBLE データ型

BINARY_DOUBLE は、64 ビット倍精度の浮動小数点数データ型です。BINARY_DOUBLE 値には、それぞれ長さを格納する 1 バイトを含めて 9 バイト必要です。

注意： BINARY_DOUBLE と BINARY_FLOAT では、米国電気電子技術者協会（IEEE）のバイナリ浮動小数点演算に関する規格である IEEE 規格 754-1985（IEEE754）のほとんどが実装されます。浮動小数点数の Oracle Database 実装と IEEE754 との相違点の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

DATE データ型の概要

DATE データ型には、ある時点の値（日付と時刻）が表に格納されます。DATE データ型には、年（世紀を含む）、月、日、時、分および秒（真夜中から数える）が格納されます。

Oracle Database では、ユリウス暦の西暦前 4712 年 1 月 1 日から西暦 9999 年 12 月 31 日までの日付を格納できます。BCE（書式マスクでは 'BC'）が明確に指定されないかぎり、デフォルトとして日付は西暦として扱われます。

Oracle Database では、日付を格納する際、独自の内部書式が使用されます。日付データは、それぞれ世紀、年、月、日、時、分および秒に対応する 7 バイトの固定長フィールドに格納されます。

日付の入出力に対する標準的な Oracle の日付書式は、DD-MON-YY です。次に例を示します。

```
'13-NOV-92'
```

このデフォルト日付書式は、インスタンスごとにパラメータ NLS_DATE_FORMAT を使用して変更できます。ユーザー・セッションの存続中には、ALTER SESSION 文でも変更できます。標準 Oracle 日付書式ではない日付を入力するには、次のように書式マスク付きの TO_DATE 関数を使用してください。

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

Oracle Database では、時刻は HH:MI:SS の 24 時間形式で格納されます。デフォルトでは、時刻部分に何も指定しない場合、日付フィールドの時刻部分は 00:00:00 A.M.（真夜中）になります。時刻のみを入力した場合、日付部分のデフォルトは現在の月の 1 日になります。日付の時刻部分を入力するには、次のように時刻部分を表す書式マスクを指定して TO_DATE 関数を使用してください。

```
INSERT INTO birthdays (bname, bday) VALUES
    ('ANDY',TO_DATE('13-AUG-66 12:56 A.M.','DD-MON-YY HH:MI A.M.'));
```

この項の内容は、次のとおりです。

ユリウス日付の使用

ユリウス日付を使用すると、共通の基準からの通算日数を算定できます。（01-01-4712（西暦前）が基準になるため、現在の日付は 2,400,000 辺りになります。）ユリウス日付は、通常は整数ではなく、小数部が日付部分になります。Oracle Database では簡略化された方法が使用されるため、結果は整数値になります。ユリウス日付は、様々な計算方法と解釈があります。Oracle Database で使用する計算方法では、7 桁の数値（最も一般的に使用される日付の場合）が生成されます。08-APR-93 は 2449086 になります。

> **注意：** Oracle のユリウス日付は、他の日付アルゴリズムで生成されるユリウス日付と互換性がない場合があります。

日付データをユリウス日付に変換するために、日付関数（TO_DATE または TO_CHAR）に書式マスク 'J' を指定できます。たとえば、次の問合せはユリウス日付書式ですべての日付を戻します。

```
SELECT TO_CHAR (hire_date, 'J') FROM employees;
```

ユリウス日付を計算で使用する場合は、TO_NUMBER 関数を使用する必要があります。ユリウス日付を入力する場合は、TO_DATE 関数を使用できます。

```
INSERT INTO employees (hire_date) VALUES (TO_DATE(2448921, 'J'));
```

日付算術

Oracle 日付算術では、過去採用されてきた様々な暦の変則を考慮しています。たとえば、ユリウス暦からグレゴリウス暦への切替え（15-10-1582）では、その直前の 10 日間（05-10-1582 から 14-10-1582 まで）が排除されています。0 年は存在しません。

存在しない日付でもデータベースに入力できます。ただし、その日付は、日付算術では無視され、次の実在する日付として処理されます。たとえば、04-10-1582 の次の日は 15-10-1582 で、05-10-1582 の次の日も 15-10-1582 になります。

> **注意：** 日付算術に関するこの説明は、すべての国（アジア圏など）の日付規格に適用できるとはかぎりません。

世紀と西暦 2000 年

Oracle Database では、データを世紀情報と一緒に格納します。たとえば、Oracle Database には、単に 96 または 01 ではなく、1996 または 2001 が格納されます。DATE データ型は常に 4 桁の年を内部に格納し、データベース内部に格納される他のすべての日付も 4 桁の年となります。インポート、エクスポート、リカバリなどの Oracle Database ユーティリティでも、年を 4 桁で処理しています。

夏時間のサポート

Oracle Database は、サーバーの DATETIME データ型に対して夏時間をサポートしています。特定の地域のローカル時間に基づいて、DATETIME 値を挿入および問合せできます。DATETIME データ型の TIMESTAMP WITH TIME ZONE および TIMESTAMP WITH LOCAL TIME ZONE では、タイム・ゾーンも認識されます。

関連項目：

- 世紀および日付の書式マスクの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
- 日付書式コードの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

タイム・ゾーン

日付 / 時間データにタイム・ゾーンを含めることが可能です。また、小数秒をサポートしています。3 つの新しいデータ型が DATE に追加されています。これらのデータ型には、表 26-2 に示すような違いがあります。

表 26-2 タイム・ゾーン・データ型

データ型	タイム・ゾーン	小数秒
DATE	なし	なし
TIMESTAMP	なし	あり
TIMESTAMP WITH TIME ZONE	明示的	あり
TIMESTAMP WITH LOCAL TIME ZONE	関連	あり

TIMESTAMP WITH LOCAL TIME ZONE は、データベースのタイム・ゾーンで格納されます。ユーザーがデータを選択すると、値はユーザーのセッションのタイム・ゾーンに調整されます。

たとえば、サンフランシスコのデータベースは、システム・タイム・ゾーン = -8:00 です。ニューヨークのクライアント（セッション・タイム・ゾーン = -5:00）が、サンフランシスコのデータベースへの挿入またはデータベースからの選択を実行すると、TIMESTAMP WITH LOCAL TIME ZONE データが次のように調整されます。

- ニューヨークのクライアントは、サンフランシスコのデータベースの TIMESTAMP WITH LOCAL TIME ZONE 列に TIMESTAMP'1998-1-23 6:00:00-5:00' を挿入します。この挿入されたデータは、バイナリ値 1998-1-23 3:00:00 として、サンフランシスコのデータベースに格納されます。
- ニューヨークのクライアントがサンフランシスコのデータベースから挿入したデータを選択すると、ニューヨークに表示される値は、'1998-1-23 6:00:00' です。
- サンフランシスコのクライアントが同じデータを選択すると、値は '1998-1-23 3:00:00' です。

注意： 時間データへの予期しない結果を避けるために、組込み SQL 関数 DBTIMEZONE と SESSIONTIMEZONE を問い合せて、データベースとセッション・タイム・ゾーンを検証できます。データベース・タイム・ゾーンまたはセッション・タイム・ゾーンが手動で設定されていない場合、Oracle Database ではデフォルトでオペレーティング・システムのタイム・ゾーンを使用します。また、Oracle Database では、オペレーティング・システムのタイム・ゾーンが有効な Oracle タイム・ゾーンではない場合、デフォルト値として UTC を使用します。

関連項目： タイムスタンプ列のデータの作成と入力を実行する構文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

LOB データ型の概要

LOB データ型 BLOB、CLOB、NCLOB および BFILE を使用すると、非構造化データ（テキスト、グラフィック・イメージ、ビデオ・クリップおよびサウンド・ウェーブなど）の大きなブロックを、バイナリ形式または文字形式で格納し、操作できます。このデータ型を使用すると、個々のデータに対する効率的なランダム・アクセスが可能です。LONG データ型よりも、LOB データ型を常に使用することをお薦めします。LOB 列に対して（パラレル DML または DDL ではない）パラレル問合せを実行できます。

LOB データ型は、いくつかの点で LONG および LONG RAW データ型と異なります。次に例を示します。

- 表は複数の LOB 列を含むことができますが、LONG 列は 1 つしか含むことができません。
- 1 つ以上の LOB 列を含む表はパーティション化できますが、LONG 列を含む表はパーティション化できません。
- データベースのブロック・サイズによりますが、LOB の最大サイズは 128TB で、LONG の最大サイズは 2GB です。
- LOB はデータへのランダム・アクセスをサポートしていますが、LONG は順次アクセスしかサポートしていません。
- LOB データ型（NCLOB を除く）をユーザー定義オブジェクト型の属性として指定できますが、LONG データ型には指定できません。
- ローカル変数のように機能する一時 LOB を使用して、LOB データの変換を実行できます。一時内部 LOB（BLOB、CLOB および NCLOB）は、一時表領域に作成され、表には依存しません。ただし、LONG データ型の場合、一時構造は使用できません。
- LOB 列がある表は、レプリケートできますが、LONG 列がある表はできません。

SQL 文は、表に LOB 列を定義し、ユーザー定義オブジェクト型に LOB 属性を定義します。表内に LOB を定義するときは、各 LOB について表領域と記憶特性を明示的に指定できます。

LOB データ型は、インライン（表の中）、アウトライン（表領域の中で、LOB ロケータを使用）または外部ファイル（BFILE データ型）のいずれかに格納できます。Oracle9*i* 以上への互換性設定によって、LOB と一緒に SQL VARCHAR 演算子および SQL 関数を使用できます。

関連項目：

- LOB データ型と LONG および LONG RAW データ型の相違点の詳細リストは、『Oracle Database SQL 言語リファレンス』を参照してください。
- LOB 記憶域および LOB ロケータの詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

この項の内容は、次のとおりです。

BLOB データ型

BLOB データ型は、構造化されていないバイナリ・データをデータベースに格納します。BLOB は最大 128TB のバイナリ・データを格納できます。

BLOB は、トランザクションに全面的に参加します。DBMS_LOB パッケージ、PL/SQL または OCI によって BLOB 値に加えられる変更は、コミットまたはロールバックが可能です。ただし、BLOB ロケータは、複数のトランザクションまたはセッションにまたがることはできません。

CLOB および NCLOB データ型

CLOB および NCLOB データ型は、最大 128TB の文字データをデータベースに格納します。CLOB はデータベース・キャラクタ・セットを、また NCLOB は Unicode 各国語キャラクタ・セット・データを格納します。固定幅の Unicode キャラクタ・セットで内部的に可変幅の LOB データを格納すると、Oracle Database では、CLOB および NCLOB への効率的なキャラクタ・ベースのランダム・アクセスが可能になります。

CLOB と NCLOB は、トランザクションに全面的に参加します。DBMS_LOB パッケージ、PL/SQL または OCI によって CLOB または NCLOB 値に加えられる変更は、コミットまたはロールバックが可能です。ただし、CLOB および NCLOB ロケータは複数のトランザクションやセッションにまたがることはできません。NCLOB 属性を持つオブジェクト型は作成できませんが、オブジェクト型のメソッドで NCLOB パラメータを指定することはできます。

> **関連項目：** 各国語キャラクタ・セット・データおよび Unicode の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

BFILE データ型

BFILE データ型は、構造化されていないバイナリ・データを、データベースの外にあるオペレーティング・システムのファイルに格納します。BFILE 列または属性は、データが含まれる外部ファイルを参照するファイル・ロケータを格納します。格納可能な BFILE データの量は、オペレーティング・システムによって制限されます。

BFILE は読取り専用のため、変更はできません。サポートしているのはランダム読取りのみで（順次読取りはサポートしていません）、トランザクションには参加しません。基礎となるオペレーティング・システムは、BFILE についてファイルの整合性、セキュリティおよび継続性を維持する必要があります。データベース管理者は、ファイルが存在することと Oracle Database プロセスがオペレーティング・システムの読取り権をファイルに持っていることを確認する必要があります。

RAW および LONG RAW データ型の概要

注意： LONG RAW データ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションで大量のバイナリ・データを格納するには、BLOB および BFILE データ型を使用してください。

また、既存の LONG RAW 列を LOB 列に変換することをお薦めします。LOB 列に適用される制限は、LONG 列よりもはるかに少数です。さらに、LOB の機能性はリリースごとに拡張されますが、LONG RAW の機能性はいくつかのリリースでは変化していません。

RAW データ型と LONG RAW データ型は、Oracle Database が解釈しない（異なるシステム間での移動時には変換されない）データに使用します。これらのデータ型は、バイナリ・データやバイト文字列用に用意されています。たとえば、LONG RAW はグラフィックス、サウンド、文書またはバイナリ・データの配列の格納に使用できます。解釈は用途によって異なります。

RAW は、VARCHAR2 文字データ型と同じく可変長のデータ型です。ただし、Oracle Net Services（ユーザー・セッションをインスタンスに接続）およびインポート / エクスポート・ユーティリティでは、RAW データや LONG RAW データの送信時に、文字変換は実行されません。それとは対照的に、Oracle Net Services と Import/Export Utility は、データベース・キャラクタ・セットとユーザー・セッションのキャラクタ・セットが異なる場合に、これらのキャラクタ・セット間で CHAR データ、VARCHAR2 データおよび LONG データを自動的に変換します。

Oracle Database が RAW または LONG RAW データと CHAR データとの間の自動変換を実行する場合、バイナリ・データは 16 進数で表され、これらの 16 進文字は 1 文字で RAW データ 4 ビット分のデータを表します。たとえば、1 バイトの RAW データのビットが 11001011 の場合、これは 'CB' として表示および入力されます。

LONG RAW データは索引付けできませんが、RAW データは索引付けできます。

関連項目： LONG RAW データ型に関する他の制限の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

ROWID および UROWID データ型の概要

Oracle Database では、ROWID データ型にデータベース内の各行のアドレス（**ROWID**）が格納されます。

- **物理 ROWID** は、通常の表（索引構成表を除く）、クラスタ化表、表パーティションとサブパーティション、索引および索引パーティションとサブパーティションに行のアドレスを格納します。
- **論理 ROWID** は、索引構成表に行のアドレスを格納します。

ユニバーサル ROWID または UROWID という単一のデータ型は、論理 ROWID と物理 ROWID のみでなく、ゲートウェイを介してアクセスされる Oracle 以外の表など、外部表の ROWID もサポートしています。

UROWID データ型の列には、あらゆる種類の ROWID を格納できます。UROWID 列を使用するには、（ファイル形式の互換性を確保するための）COMPATIBLE 初期化パラメータの値を 8.1 以上に設定する必要があります。

関連項目： 26-20 ページの「Oracle 以外のデータベースの ROWID」

この項の内容は、次のとおりです。

ROWID 疑似列

Oracle データベースのそれぞれの表は、ROWID という名前の**疑似列**を内部的に持っています。SQL*Plus で SELECT * FROM... 文または DESCRIBE... 文を実行して表の構造のリストを表示しても、この疑似列は表示されません。また、表には疑似列のスペースが取得されません。ただし、それぞれの行のアドレスは、予約語 ROWID を列名として使用すると、SQL 問合せで取得できます。次に例を示します。

```
SELECT ROWID, last_name FROM employees;
```

ROWID 疑似列の値は、INSERT または UPDATE 文では設定できません。また、ROWID の値は削除できません。Oracle Database は、索引を組み立てるために、疑似列 ROWID 内の ROWID の値を内部的に使用します。

ROWID 疑似列の ROWID は他の表の列と同じように参照できますが（SELECT 構文のリストや WHERE 句で使用）、その ROWID はデータベースに格納されているわけではなく、またそのデータベースのデータでもありません。ROWID データ型の列を含む表を作成することはできますが、そのような列の ROWID 値が有効であるかどうかは保証できません。ユーザーは、ROWID 列に格納されているデータが、本当に有効な ROWID であることを確認する必要があります。

関連項目： 26-18 ページの「ROWID の使用方法」

物理 ROWID

物理 ROWID は、特定の表の 1 行への最高速のアクセスを提供します。物理 ROWID には、1 行のアドレス（特定のブロックまで）が含まれ、単一のブロック・アクセスでその行を取り出すことができます。

クラスタ化されていない表のすべての行には、行の行断片（行が複数の行断片にわたって連鎖している場合は、最初の行断片）の物理アドレスに対応する、一意の ROWID が割り当てられています。クラスタ化表の場合、同じデータ・ブロックにある異なる表の行の ROWID が同一の場合もあります。

特殊な状況では、ROWID が行断片に割り当てられた後、ROWID が変更される場合があります。たとえば、行の移動が可能な場合、ROWID は、パーティション・キーの更新、フラッシュバック表の操作、表の縮小操作などのために変更される可能性があります。行の移動が無効な場合は、Oracle Database ユーティリティを使用して行をエクスポートおよびインポートした場合に ROWID が変更される可能性があります。

物理 ROWID のデータ型は、次の 2 つの書式のどちらかです。

- **拡張 ROWID** 形式は、表領域関連のデータ・ブロック・アドレスをサポートし、パーティション表と索引内の行や、パーティション化されていない索引の行を効率よく識別します。Oracle8*i*（またはそれ以上の）サーバーによって作成される表と索引は、常に拡張 ROWID を持ちます。
- リリース 7 以前の Oracle Database で開発されたアプリケーションとの下位互換性を保つために、**制限付き ROWID** 形式も使用できます。

この項の内容は、次のとおりです。

拡張 ROWID

拡張 ROWID は、それぞれの選択された行の物理アドレスを BASE64 エンコーディングで表現したものです。エンコーディング文字は、A ～ Z、a ～ z、0 ～ 9、+ および / です。たとえば、次の問合せを考えます。

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

次のような行情報が戻されます。

```
ROWID              LAST_NAME
------------------ ----------
AAAAaoAATAAABrXAAA BORTINS
AAAAaoAATAAABrXAAE RUGGLES
AAAAaoAATAAABrXAAG CHEN
AAAAaoAATAAABrXAAN BLUMBERG
```

拡張 ROWID の書式は、OOOOOOFFFBBBBBBRRR という 4 つの部分からなっています。

- OOOOOO: **データ・オブジェクト番号**。データベース・セグメント（例では AAAAao）を識別します。同じセグメント内のスキーマ・オブジェクト（表のクラスタなど）は、同じデータ・オブジェクト番号を持っています。
- FFF: 行を含むデータファイルの表領域関連の**データファイル番号**（例ではファイル AAT）。
- BBBBBB: その行を含む**データ・ブロック**（例ではブロック AAABrX）。ブロック番号は、表領域**ではなく**データファイルに対する相対値です。このため、同じブロック番号の行が、同じ表領域の 2 つの異なるデータファイルに存在することもあります。
- RRR: ブロック内の**行**。

データ・オブジェクト番号は、データ・ディクショナリ・ビュー USER_OBJECTS、DBA_OBJECTS および ALL_OBJECTS から取得できます。たとえば、次の問合せは、SCOTT スキーマの employees 表のデータ・オブジェクト番号を戻します。

```
SELECT DATA_OBJECT_ID FROM DBA_OBJECTS
    WHERE OWNER = 'SCOTT' AND OBJECT_NAME = 'EMPLOYEES';
```

DBMS_ROWID パッケージを使用して、拡張 ROWID から情報を抽出したり、ROWID を拡張形式から制限付き形式に変換（またはその逆も）できます。

> **関連項目：** DBMS_ROWID パッケージの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

制限付き ROWID

制限付き ROWID は、それぞれの選択された行の物理アドレスをバイナリで表現したものです。SQL*Plus を使用して問合せを実行すると、このバイナリ表現が VARCHAR2 の 16 進数表現に変換されます。次のような問合せがあるとします。

```
SELECT ROWID, last_name FROM employees
    WHERE department_id = 30;
```

次のような行情報が戻されます。

```
ROWID              ENAME
------------------ ----------
00000DD5.0000.0001 KRISHNAN
00000DD5.0001.0001 ARBUCKLE
00000DD5.0002.0001 NGUYEN
```

前述のように、制限付き ROWID の VARCHAR2 の 16 進数表現は、**block.row.file** という 3 つの部分からなっています。

- その行を含む**データ・ブロック**（例ではブロック DD5）。ブロック番号は、表領域**ではなく**データファイルに対する相対値です。同じブロック番号の行が、同じ表領域の 2 つの異なるデータファイルに存在することもあります。
- その行を含むブロックの中の**行**（例では行 0、1、2）。各ブロックの行番号は、必ず 0 で始まります。
- その行を含む**データファイル**（例ではファイル 1）。すべてのデータベースの最初のデータファイルは必ずファイル 1 であり、ファイル番号はデータベース内で一意です。

ROWID の使用例

関数 SUBSTR を使用して、ROWID のデータをコンポーネントに分割できます。たとえば、SUBSTR を使用して拡張 ROWID を 4 つのコンポーネント（データベース・オブジェクト、ファイル、ブロックおよび行）に分割します。

```
SELECT ROWID,
       SUBSTR(ROWID,1,6) "OBJECT",
       SUBSTR(ROWID,7,3) "FIL",
       SUBSTR(ROWID,10,6) "BLOCK",
       SUBSTR(ROWID,16,3) "ROW"
       FROM products;

ROWID               OBJECT  FIL  BLOCK   ROW
------------------  ------  ---  ------  ----
AAAA8mAALAAAAQkAAA  AAAA8m  AAL  AAAAQk  AAA
AAAA8mAALAAAAQkAAF  AAAA8m  AAL  AAAAQk  AAF
AAAA8mAALAAAAQkAAI  AAAA8m  AAL  AAAAQk  AAI
```

または、SUBSTR を使用して制限付き ROWID を 3 つのコンポーネント（ブロック、行およびファイル）に分割します。

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE",
       SUBSTR(ROWID,1,8) "BLOCK",
       SUBSTR(ROWID,10,4) "ROW"
       FROM products;

ROWID               FILE  BLOCK     ROW
------------------  ----  --------  ----
00000DD5.0000.0001  0001  00000DD5  0000
00000DD5.0001.0001  0001  00000DD5  0001
00000DD5.0002.0001  0001  00000DD5  0002
```

ROWID は、表データの物理的な格納に関する情報を明らかにするのに便利です。たとえば、表の行の物理的な位置を確認する場合（表のストライプ化の場合など）、次のように拡張 ROWID を問い合せると、指定の表の行がいくつのデータファイルに含まれているかがわかります。

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID,7,3))) "FILES" FROM tablename;

   FILES
--------
       2
```

関連項目：

- 『Oracle Database SQL 言語リファレンス』
- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database パフォーマンス・チューニング・ガイド』

ROWID の使用例の詳細は、前述のマニュアルを参照してください。

ROWID の使用方法

Oracle Database では、ROWID を内部的に使用して索引が構築されます。索引のそれぞれのキーは、高速アクセスのために、対応する行のアドレスを指す ROWID に結び付けられています。エンド・ユーザーとアプリケーション開発者は、次のような重要な用途に ROWID を使用することもできます。

- ROWID は特定の行にアクセスする最も高速な方法です。
- ROWID は表の編成を把握するために使用できます。
- ROWID は表の中の行の一意識別子です。

DML 文の中で ROWID を使用する前に、ROWID が変更されないことを検証して確認する必要があります。目的の行を、削除されないようにロックしてください。無効な ROWID を使用してデータを要求すると、状況によっては、文が失敗します。

また、ROWID データ型を使用して定義した列を持つ表を作成することもできます。たとえば、データ型 ROWID の列を持つ例外表を定義して、整合性制約に違反するデータベース行の ROWID を格納できます。ROWID データ型を使用して定義されている列の動作は、表の他の列と類似しています。たとえば、値を更新できます。データ型 ROWID として定義されている列のそれぞれの値を、適切な列データとして格納するには 6 バイト必要です。

論理 ROWID

索引構成表の中の行は、永続的な物理アドレスを持たず、索引リーフに格納されます。挿入の結果として同一ブロック内で、または別のブロックに移動できます。したがって、物理アドレスに基づく行識別子は使用できません。かわりに、Oracle は論理行識別子を持つ索引構成表を提供します。この識別子は、表の主キーに基づいており、**論理 ROWID** と呼ばれます。Oracle Database は、これらの論理 ROWID を使用して、索引構成表の 2 次索引を構築します。

2 次索引に使用される論理 ROWID ごとに、**物理推測**が含まれます。これは、推測時、つまり、2 次索引の作成時または再作成時に、索引構成表の中の行のブロック位置を識別するものです。

Oracle Database は、推測を使用して、全キー検索を回避し、リーフ・ブロックを直接プローブできます。これにより、不揮発性の索引構成表の ROWID アクセスで、通常の表の物理 ROWID アクセスに匹敵するパフォーマンスを得られることが保証されます。ただし、揮発性の表では、推測が陳腐化するとプローブが失敗することがあります。その場合は、主キー検索を実行する必要があります。

2 つの論理 ROWID の値は、同じ主キー値および異なる推測を持つ場合に同一とみなされます。

この項の内容は、次のとおりです。

論理 ROWID と物理 ROWID の比較

論理 ROWID と物理 ROWID の類似点は、次のとおりです。

- 論理 ROWID には、ROWID 疑似列を介してアクセスできます。

  ROWID 疑似列を使用して、索引構成表から論理 ROWID を選択できます。SELECT ROWID 文は不透明構造を戻します。この構造は、表の主キーと行の物理推測（存在する場合）、およびなんらかの制御情報で内部的に構成されています。

  WHERE ROWID = *value* という形式の述語を使用して行にアクセスできます。この場合、*value* は SELECT ROWID から戻される不透明構造です。

- 論理 ROWID を介してアクセスするのは、特定の行へのアクセス方法としては最も高速ですが、複数のブロック・アクセスを必要とする場合があります。
- 行の論理 ROWID は、主キー値に変更がなければ変化しません。このため、行に対するすべての更新を通じて変化しない物理 ROWID に劣らず陳腐化しにくくなっています。
- 論理 ROWID は、UROWID データ型の列に格納できます。

物理 ROWID と論理 ROWID の違いの 1 つは、論理 ROWID を使用しても表の構成方法を表示できないことです。

注意： 不透明型は、その内部構造がデータベースに認識されない型です。データベースは、この型のための記憶域を提供します。型設計者は、ファンクション、通常は 3GL ルーチンを実装して、この型の内容へのアクセスを提供できます。

関連項目： 26-14 ページの「ROWID および UROWID データ型の概要」

論理 ROWID での推測

行の物理位置が変化した場合、論理 ROWID は推測が含まれていても引き続き有効ですが、推測は陳腐化し、その行へのアクセスが低速になることがあります。推測情報を動的に更新することはできません。ただし、索引構成表の 2 次索引に対し、索引を再作成して最新の推測を取得できます。索引構成表の 2 次索引を再作成するには、通常の表の索引を再作成する場合とは異なり、実表を読み取る必要があるため注意してください。

Oracle Database で推測が不用意に使用されないように、DBMS_STATS パッケージまたは ANALYZE 文で索引統計を収集し、推測の陳腐化を追跡する必要があります。特に、UROWID 列に推測を使用した ROWID を永続的に格納し、その ROWID を後から取り出して行のフェッチに使用するアプリケーションの場合は、この作業が重要になります。

DBMS_STATS パッケージまたは ANALYZE 文を使用して索引の統計を収集すると、Oracle Database により既存の推測がまだ有効かどうかがチェックされ、陳腐化した推測と有効な推測の比率がデータ・ディクショナリに記録されます。2 次索引を再作成（推測を再コンパイル）した後に、索引統計を収集しなおしてください。

通常、論理 ROWID に推論が付いていない場合に、揮発性の高い表へのアクセスが最も高速になります。表が静的な場合、または ROWID を取得してから使用するまでの時間が短いため行移動がない場合は、推測付きの論理 ROWID によるアクセスが最も高速です。

関連項目： 統計収集の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

Oracle 以外のデータベースの ROWID

Oracle Database アプリケーションは、SQL*Connect を使用して、Oracle 以外のデータベース・サーバーに対しても実行可能です。ROWID の形式は、その Oracle 以外のシステムの特性に応じて異なります。また、VARCHAR2 の 16 進形式への標準変換も使用できません。プログラムでは、ROWID データ型を使用できます。しかし、最大 256 バイトまでの 16 進形式への非標準変換を使用する必要があります。

Oracle 以外のデータベースの ROWID を、UROWID データ型の列に格納できます。

関連項目：

- Oracle 以外のシステムで ROWID を処理する場合の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。
- 26-14 ページの「ROWID および UROWID データ型の概要」

ANSI データ型、DB2 データ型および SQL/DS データ型の概要

表とクラスタを作成する SQL 文では、ANSI データ型と、IBM 製品である SQL/DS および DB2 からのデータ型も使用できます。Oracle Database では、Oracle データ型名とは異なる ANSI または IBM のデータ型名を認識し、それを列のデータ型名として記録してから、変換に基づいて、その列のデータを Oracle データ型に格納します。

関連項目： 変換の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

XML データ型の概要

Oracle には、XML データを処理するための XMLType データ型が用意されています。

XMLType データ型

XMLType は、他のユーザー定義型と同様に使用できます。XMLType は表やビューの列のデータ型として使用できます。XMLType の変数は PL/SQL ストアド・プロシージャのパラメータや戻り値として使用できます。また、XMLType は PL/SQL、SQL や Java で使用することも、JDBC や OCI を介して使用することも可能です。

XML の内容を操作する便利な関数が多数用意されています。これらの関数の多くは、SQL 関数および XMLType のメンバー関数として提供されます。たとえば、関数 extract は XMLType のインスタンスから特定のノードを抽出します。XMLType は、システム内の他のユーザー定義型と同様に、SQL 問合せに使用できます。

関連項目：

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML DB 開発者ガイド』
- アドバンスト・キューイングでの XMLType の使用方法の詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』を参照してください。
- 第 1 章「Oracle Database の概要」

URI データ型の概要

Uniform Resource Identifier（URI）は、一般化された URL の一種です。URL と同様にどのようなドキュメントでも参照でき、ドキュメントの特定部分を参照できます。URI には、ドキュメントの関連部分を指定する強力なメカニズムがあるため、URL よりも一般的です。`URIType` を使用すると、次の操作を実行できます。

- データベースの内部または外部にあるデータを指す表の列の作成
- `URIType` が提供する機能を使用したデータベース列の問合せ

関連項目：『Oracle XML DB 開発者ガイド』

オブジェクト・データ型およびオブジェクト・ビューの概要

オブジェクト型とその他のユーザー定義データ型を使用すると、アプリケーションのデータの構造と動作をモデル化するデータ型を定義できます。オブジェクト・ビューは、仮想的なオブジェクト表です。

関連項目：『Oracle Database オブジェクト・リレーショナル開発者ガイド』

データ変換

Oracle Database により、予期しているのとは異なるデータ型が与えられる場合があります。これは、Oracle Database で予期されたデータ型にデータを自動的に変換できる場合には許容されます。

関連項目： 暗黙的なデータ型変換の規則は、『Oracle Database SQL 言語リファレンス』を参照してください。

用語集

ADDM

「Automatic Database Diagnostic Monitor」を参照。

ADR

「自動診断リポジトリ」を参照。

ADRCI

Oracle Database 11*g* で導入された障害診断性インフラストラクチャの一部をなすコマンドライン・ツール。

AFTER トリガー（AFTER trigger）

トリガーの定義時に、トリガーのタイミングを指定できる。トリガー・アクションをトリガー文の前に実行するか、または後に実行するかを指定する。

AFTER トリガーは、トリガー文の実行後にトリガー・アクションを実行する。

BEFORE と AFTER は、文トリガーと行トリガーの両方に適用される。

関連項目：「トリガー」

ARCHIVELOG モード（ARCHIVELOG mode）

Oracle Database がいっぱいになったオンライン REDO ログをディスクにコピーするときの、データベースのモード。モードは、データベースの作成時に指定するか、ALTER DATABASE 文を使用して指定する。ALTER SYSTEM 文を使用するか、初期化パラメータ LOG_ARCHIVE_START を TRUE に設定すると、自動アーカイブを動的に使用可能にすることができる。

データベースを ARCHIVELOG モードで実行すると、NOARCHIVELOG モードに比べていくつか利点がある。次のことができる。

- オープン状態でユーザーがアクセスしているデータベースのバックアップ作成
- 必要な時点までのデータベースのリカバリ

ARCHIVELOG モードのデータベースを障害時に保護するために、アーカイブ・ログのバックアップを作成する。

ASM

「自動ストレージ管理」を参照。

Automatic Database Diagnostic Monitor（ADDM）

ADDM により、Oracle Database はそれ自体のパフォーマンスを診断し、識別された問題の解決方法を決定できる。ADDM は、AWR 統計の取得ごとに自動的に実行され、パフォーマンス診断データが使用可能になる。

AWR

「自動ワークロード・リポジトリ」を参照。

BEFORE トリガー（BEFORE trigger）

トリガーの定義時に、トリガーのタイミングを指定できる。トリガー・アクションをトリガー文の前に実行するか、または後に実行するかを指定する。

BEFORE トリガーは、トリガー文の実行前にトリガー・アクションを実行する。

BEFORE と AFTER は、文トリガーと行トリガーの両方に適用される。

関連項目：「トリガー」

DDL

データ定義言語。データ構造を定義または変更する CREATE/ALTER TABLE/INDEX などの文。

DML

データ操作言語。表のデータを変更する INSERT、UPDATE および DELETE などの文が含まれる。

DOP

操作の並列度。

LogMiner

管理者が SQL を使用してログ・ファイルの読取り、分析および解析を行うためのユーティリティ。あらゆる REDO ログ・ファイル、オンラインまたはアーカイブでも表示できる。Oracle Enterprise Manager アプリケーションである Oracle LogMiner Viewer により、GUI ベースのインタフェースが追加される。

NOT NULL 制約（NOT NULL constraint）

表の列値が NULL でないことを求めるデータ整合性制約。

関連項目：「NULL 値」

NULL 値（NULL value）

1 つの行の 1 つの列に値がないこと。NULL は、データがない、不明である、または適切でないことを示す。他の値（ゼロなど）を暗示する目的では、NULL は使用できない。

Oracle Enterprise Manager

異機種間環境を集中管理するための統合ソリューションを提供する Oracle のシステム管理ツール。Oracle 製品を管理するためのグラフィカル・コンソール、Oracle Management Servers、Oracle Intelligent Agent、共通サービスおよび管理ツールの組合せ。

Oracle RAC

関連項目：「Oracle Real Application Clusters」

Oracle Real Application Clusters

複数の同時インスタンスが 1 つの物理データベースを共有できるオプション。

関連項目：「インスタンス」

Oracle XA

Oracle Database 以外のトランザクション・マネージャでグローバル・トランザクションを調整できるようにする外部インタフェース。

Oracle アーキテクチャ（Oracle architecture）

データベースの管理で Oracle Database サーバーが使用するメモリーとプロセスの構造。

関連項目：「データベース」、「プロセス」、「サーバー」

Oracle クラスタウェア（Oracle Clusterware）

Oracle Real Application Clusters 10*g* に統合された移植性のあるクラスタ管理ソリューションで、Oracle コンポーネントの監視や再起動を行う。Oracle クラスタウェアは、クラスタ内で実行されているすべてのアプリケーションの管理および監視を行うようにプログラムすることが可能。Oracle クラスタウェアにより管理されているクラスタ内では、シングル・インスタンスと Oracle Real Application Clusters データベースのどちらも実行することができる。

Oracle プロセス（Oracle process）

Oracle Database コードを実行する。プロセスには、サーバー・プロセスとバックグラウンド・プロセスがある。

関連項目：「**プロセス**」、「**サーバー・プロセス**」、「**バックグラウンド・プロセス**」、「**ユーザー・プロセス**」

PL/SQL

オラクル社が開発した、SQL 言語を手続き型に拡張した言語。PL/SQL を使用すると、SQL 文を手続き型構造と混合して使用できる。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る。

関連項目：「**SQL**」

Recovery Manager（RMAN）

Oracle Database をバックアップ、リストアおよびリカバリするユーティリティ。Recovery Manager の実行時には、リカバリ・カタログと呼ばれる中央の情報リポジトリを使用するかどうかを選択できる。リカバリ・カタログを使用しない場合、Recovery Manager ではデータベースの制御ファイルを使用して、バックアップおよびリカバリ操作に必要な情報が格納される。Recovery Manager をメディア・マネージャと併用してファイルのバックアップを 3 次記憶装置に格納できる。

Recovery Point Objective（RPO）

IT ベースの組織でシステム障害の結果として損失しても支障のない最大データ量。RPO は、より完全なリカバリにかかるコストと、データ損失に伴う組織の損害との間で釣合いの取れる地点である。ビジネス・プロセスまたは組織の一般的なデータ損失の許容差を示す。5 時間または 2 日分のデータ損失など、時間の観点から測定される。

Recovery Time Objective（RTO）

IT ベースの組織でシステム障害後に停止しても支障のない最長時間。RTO は、より迅速なリカバリにかかるコストと、停止に伴う組織のコストとの間で釣合いの取れる地点である。RTO は、ビジネス・プロセスまたは組織の一般的な停止の許容差を示す。

REDO スレッド（redo thread）

インスタンスによって生成される REDO。単一インスタンス構成でデータベースが稼働している場合、そのデータベースの REDO スレッドは 1 つのみである。Oracle Real Application Clusters 構成で稼働している場合は、インスタンスごとに 1 つずつ、複数の REDO スレッドが生成される。

REDO ログ（redo log）

データファイルに書き込まれていないメモリー内の変更済データベース・データを保護する一連のファイル。REDO ログは、オンライン REDO ログとアーカイブ REDO ログの 2 部からなる場合がある。

関連項目：「**オンライン REDO ログ**」

REDO ログ・バッファ（redo log buffer）

REDO エントリ（データベースに対して行われた変更のログ）を格納するシステム・グローバル領域のメモリー構造。REDO ログ・バッファ内に格納された REDO エントリは、オンライン REDO ログ・ファイルに書き込まれ、データベースのリカバリが必要になったときに REDO ログ・ファイルが使用される。

関連項目：「**システム・グローバル領域**」

RMAN

関連項目：「**Recovery Manager**」

ROWID

データベース内の行に対するグローバルな一意の識別子。ROWID は、表に行が挿入されたときに作成され、その行が表から削除されたときに破棄される。

SQL

Structured Query Language。データにアクセスするための非手続き型言語。ユーザーが処理内容を SQL で記述すると、SQL 言語コンパイラはデータベースをナビゲートし、そのタスクを実行するプロシージャを自動的に生成する。Oracle SQL には、ANSI/ISO 標準 SQL 言語に対する拡張機能が多数含まれている。

関連項目：「**SQL*Plus**」、「**PL/SQL**」

SQL*Plus

Oracle Database に対して SQL 文を実行するために使用する Oracle のツール製品。

関連項目：「**SQL**」、「**PL/SQL**」

Unicode

世界のすべての言語のすべての文字を表現する方法。文字はコード・ポイントの順序として定義され、基本的なコード・ポイントの後に任意のサロゲート数が続く。64K のコード・ポイントがある。

Unicode 列（Unicode column）

Unicode を保持することが保証されている `NCHAR`、`NVARCHAR2` または `NCLOB` 型の列。

UTC

協定世界時。以前は、グリニッジ標準時（GMT）と呼ばれていた。

アーキテクチャ（architecture）

「**Oracle アーキテクチャ**」を参照。

一意キー制約（UNIQUE KEY constraint）

データ整合性制約。列または列の集合（キー）内のすべての値が一意である（つまり、指定した列または列の集合内では、表の 2 つの行の値が重複していない）必要がある。

関連項目：「**整合性制約**」、「**キー**」

一時セグメント（temporary segment）

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

関連項目：「**エクステント**」、「**セグメント**」

一時ファイル（tempfile）

一時表領域に属すファイル。`TEMPFILE` オプションを指定して作成される。一時表領域には、表などの永続データベース・オブジェクトを含めることはできず、通常はソートに使用される。

一貫性バックアップ（consistent backup）

メディア・リカバリを実行せずに RESETLOGS オプションを指定してオープンできる**データベース全体のバックアップ**。つまり、このバックアップの場合、一貫性を保つためにデータファイルに REDO を適用する必要はない。一貫性バックアップのすべてのデータファイルには、次の要件がある。

- ヘッダーに同一のチェックポイント・**システム変更番号**（SCN）が必要。ただし、読取り専用または NORMAL モードでオフライン化された表領域内のデータファイルの場合は、チェックポイント SCN より小さいクリーン SCN が割り当てられる。
- チェックポイント SCN より前の変更、つまりファジーでない変更は含まない。
- 制御ファイルに格納されているデータファイルのチェックポイント情報と一致する。

一貫性バックアップを実行できるのは、データベースを正しく停止した後のみ。バックアップが完了するまではデータベースをオープン状態にすることができない。

関連項目：「**非一貫性バックアップ**」

インスタンス（instance）

Oracle データベース・インスタンスは、システム・グローバル領域（SGA）および Oracle Database バックグラウンド・プロセスで構成される。データベースを起動するたびに、システム・グローバル領域（SGA）が割り当てられ、Oracle Database バックグラウンド・プロセスが起動される。SGA は、インスタンスが停止したときに割当てが解除される。

関連項目：「**バックグラウンド・プロセス**」、「**システム・グローバル領域**」、「**自動ストレージ管理インスタンス**」

エクステント（extent）

論理データベース記憶域の第 2 レベル。エクステントは、特定数の連続したデータ・ブロックで、特定のタイプの情報を格納するために割り当てられる。

関連項目：「**データ・ブロック**」、「**セグメント**」

演算子（operator）

メモリー管理における演算子とは、ソート、ハッシュ結合またはビットマップ・マージなどのデータ・フロー演算子を指す。

オブジェクト型（object type）

オブジェクト型は仕様部と本体の 2 部からなる。本体は、常に仕様部のタイプに依存する。

オンライン REDO ログ（online redo log）

Oracle Database のデータファイルと制御ファイルに対するすべての変更が記録される複数のファイルの集合。データベースに対して変更が行われるたびに、Oracle Database により REDO バッファに REDO レコードが生成される。LGWR プロセスは、REDO バッファの内容を REDO ログにフラッシュする。

関連項目：「**REDO ログ**」

オンライン・バックアップ（online backup）

データベースがオープン状態でデータファイルがオンラインになっている間に作成された 1 つ以上のデータファイルのバックアップ。データベースがオープン状態のときにユーザー管理バックアップを実行する場合は、ALTER TABLESPACE BEGIN BACKUP コマンドを発行して表領域をバックアップ・モードにする必要がある。ただし、データベースがオープン状態のときに Recovery Manager によるバックアップを実行する場合、表領域をバックアップ・モードにする必要はない。

外部キー（foreign key）

整合性制約。列または列の集合の値が、それぞれ関連する表の一意または主キーの値と一致する必要がある。

外部キー制約は、参照データが変更された場合に依存データを処理する方法を Oracle Database に指示する、参照整合性アクションも定義する。

関連項目：「**整合性制約**」、「**主キー**」

キー（key）

特定なタイプの整合性制約の定義に含まれる列または列の集合。キーによって、リレーショナル・データベースの異なる表と列の間の関連が示される。

関連項目：「**整合性制約**」、「**外部キー**」、「**主キー**」

キャッシュ・リカバリ（cache recovery）

インスタンス・リカバリの一部。Oracle Database により、コミット済かどうかを問わず、REDO ログ・ファイル内のすべての変更が、その対象となるデータ・ブロックに適用される。インスタンス・リカバリのロールフォワード・フェーズと呼ばれることもある。

キャラクタ・セマンティクス（character semantics）

文字列の長さが文字で測定される。

行（row）

表内のエンティティまたはレコードに関係する一連の属性または値。行は、1 つのレコードに対応する列情報の集まり。

関連項目：「**列**」、「**表**」

共有サーバー（shared server）

多数のユーザー・プロセスが少数のサーバー・プロセスを共有できるデータベース・サーバー構成。これにより、サーバー・プロセスの数を最低限に抑え、使用可能なシステム・リソースの使用効率を最大化できる。

関連項目：「**専用サーバー**」

共有プール（shared pool）

共有 SQL 領域などの共有メモリー構成メンバーを含むシステム・グローバル領域の一部。データベースに送られる個々の一意の SQL 文を処理するために、共有 SQL 領域が必要になる。

関連項目：「**システム・グローバル領域**」、「**SQL**」

クライアント（client）

クライアント / サーバー・アーキテクチャにおけるフロントエンドのデータベース・アプリケーションで、キーボード、ディスプレイおよびマウスなどのポインティング・デバイスを使用してユーザーと対話する。クライアント部分ではデータをアクセスしない。クライアント部分は、サーバー部分が管理するデータの要求、処理および提示のみを行う。

関連項目：「**クライアント / サーバー・アーキテクチャ**」、「**サーバー**」

クライアント / サーバー・アーキテクチャ（client/server architecture）

2 つの CPU 間で処理を分割するソフトウェア・アーキテクチャ。1 つの CPU は、トランザクションでクライアントとして機能し、サービスを要求して受け取る。もう 1 つはトランザクションでサービスを提供するサーバーとして機能する。

クラスタ（cluster）

表データを格納するためのオプションの構造。クラスタとは、物理的にまとめて格納される 1 つ以上の表のグループ。それらの表は、共通の列を共有しており、通常、一緒に使用されるため、まとめて格納される。関連する行が物理的にまとめて格納されているため、ディスク・アクセス時間が短縮される。

コミット（commit）

データベース内のデータに永続的な変更（挿入、更新、削除）を加える。変更をコミットする前は、変更を格納したり、データを以前の状態にリストアできるように新旧両方のデータが存在する。

関連項目：「**ロールバック**」

サーバー（server）

クライアント / サーバー・アーキテクチャで、Oracle ソフトウェアを実行し、同時実行の共有データ・アクセスに必要な機能を処理するコンピュータ。サーバーは、クライアント・アプリケーションから発行された SQL 文や PL/SQL 文を受け取って処理する。

関連項目：「**クライアント**」、「**クライアント / サーバー・アーキテクチャ**」

サーバー・パラメータ・ファイル（server parameter file）

Oracle Database ホスト上で管理される初期化パラメータ設定が含まれるバイナリ・ファイル。このファイルは、テキスト・エディタを使用して手動で編集することはできない。サーバー・パラメータ・ファイルは、最初は、`CREATE SPFILE` 文でテキストの初期化パラメータ・ファイルから作成される。また、Database Configuration Assistant を使用して直接作成することもできる。

サーバー・プロセス（server process）

サーバー・プロセスは、接続されているユーザー・プロセスからの要求を処理する。サーバー・プロセスは、対応付けられたユーザー・プロセスの要求を実行するために、ユーザー・プロセスと通信し、Oracle Database と対話する。

関連項目：「**プロセス**」、「**ユーザー・プロセス**」

索引（index）

表とクラスタに関連したオプションの構造。表の 1 つ以上の列に索引を作成し、その表のデータへのアクセスを高速化できる。

関連項目：「**クラスタ**」

索引セグメント（index segment）

各索引には、そのデータをすべて格納する索引セグメントが 1 つある。パーティション索引の場合は、各パーティションに 1 つずつ索引セグメントがある。

関連項目：「**索引**」、「**セグメント**」

索引タイプ（indextype）

ドメイン索引の管理でサポートされている一連の演算子とルーチンを指定して、新規の索引付けスキームを登録するオブジェクト。

サブタイプ（subtype）

ユーザー定義データ型の階層におけるサブタイプは、常にそのサブタイプのスーパータイプに依存している。

参照整合性（referential integrity）

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する。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る。

関連項目：「**キー**」

システム・グローバル領域（system global area: SGA）

共有メモリー構造のグループ。1 つの Oracle データベース・インスタンスに関するデータと制御情報が含まれている。複数のユーザーが同じインスタンスに同時に接続した場合、そのインスタンスの SGA 内のデータはユーザー間で共有される。したがって、SGA は共有グローバル領域と呼ばれることもある。

関連項目：「インスタンス」

システム変更番号（system change number: SCN）

特定の時点におけるデータベースのコミット済バージョンを定義するスタンプ。コミット済の各トランザクションには、Oracle Database により一意の SCN が割り当てられる。

自動 UNDO 管理モード（automatic undo management mode）

専用 UNDO 表領域の UNDO 領域が自動的に管理されるデータベースのモード。このモードでは、データベースにより UNDO 保存期間も自動的にチューニングされる。Oracle Database 11*g* 以降の新しいデータベース・インストールのデフォルト・モード。

関連項目：「手動 UNDO 管理モード」

自動診断リポジトリ（Automatic Diagnostic Repository）

システム全体のトレースとロギングを行うための中央リポジトリ。このリポジトリは、ファイルベースの階層データストアであり、ネットワークのトレースやロギングに関する情報など、様々な診断情報が蓄積される。

自動ストレージ管理（Automatic Storage Management: ASM）

Oracle Database ファイル専用に構築されたファイル・システムとボリューム・マネージャの両方が垂直方向に統合された機能。この機能により、stripe and mirror everything の概念が拡張されてパフォーマンスが最適化される一方、手動で I/O をチューニングする必要性がなくなる。

自動ストレージ管理インスタンス（Automatic Storage Management instance）

自動ストレージ管理ディスク・グループをマウントし、自動ストレージ管理ファイルをデータベース・インスタンスに対して使用可能にするために必要な管理機能を実行する Oracle データベース・インスタンス。自動ストレージ管理インスタンスは、データベースをマウントしない。

関連項目：「インスタンス」

自動ストレージ管理ディスク（Automatic Storage Management disk）

記憶域は、自動ストレージ管理ディスク・グループから自動ストレージ管理ディスク単位で追加および削除される。

自動ストレージ管理テンプレート（Automatic Storage Management template）

ファイル作成時に自動ストレージ管理で使用される属性のコレクション。テンプレートにより、複雑なファイル属性仕様が 1 つの名前にマッピングされ、ファイル作成処理が簡素化される。Oracle Database のファイル・タイプごとに、デフォルト・テンプレートが存在する。ユーザーはデフォルト・テンプレートの属性を変更するか、新規テンプレートを作成できる。

自動ストレージ管理ファイル（Automatic Storage Management file）

自動ストレージ管理ディスク・グループに格納される Oracle Database ファイル。ファイルの作成時には、特定のファイル属性が永続的に設定される。たとえば、冗長性レベル（`MIRROR`、`HIGH` または `UNPROTECTED`）やストライプ化ポリシーなどである。自動ストレージ管理ファイルは、オペレーティング・システムやそのユーティリティからは参照できないが、データベース・インスタンス、RMAN、または `ASMCMD` などの他の Oracle 提供ツールからは参照可能。

自動ワークロード・リポジトリ（Automatic Workload Repository: AWR）

すべての Oracle Database に組み込まれているリポジトリ。Oracle Database は、定期的に、その稼働統計とワークロード情報すべてのスナップショットを作成し、AWR にそのスナップショットを格納する。

シノニム（synonym）

表、ビュー、マテリアライズド・ビュー、順序、プロシージャ、ファンクション、パッケージ、型、Java クラスのスキーマ・オブジェクト、ユーザー定義オブジェクト型または他のシノニムの別名。

主キー（primary key）

主キー制約の定義に含まれる列（または列の集合）。主キーの値は、表内の各行を一意に識別する。各表には 1 つの主キーのみが定義できる。

関連項目：「**主キー制約**」

主キー制約（PRIMARY KEY constraint）

列または列の集合に重複値と NULL を許可しない整合性制約。

関連項目：「**整合性制約**」、「**キー**」

手動 UNDO 管理モード（manual undo management mode）

UNDO ブロックがユーザー管理ロールバック・セグメントに格納されるデータベース・モード。**自動 UNDO 管理モード**では、UNDO ブロックはシステム管理の専用 UNDO 表領域に格納される。

関連項目：「**自動 UNDO 管理モード**」

順序（sequence）

データ実表の数値列について、連続する一意の数値のリストを生成する。

障害グループ（failure group）

フォルト・トレランスが必要な共通のリソースを共有する自動ストレージ管理（ASM）のディスク・グループ内のディスクのサブセット。障害グループは、どの ASM ディスクをデータの冗長コピーの格納に使用するかを決定するために使用される。たとえば、特定のディスク・グループで、ディスク・コントローラ A にディスク 1 ～ 4 があり、ディスク・コントローラ B にディスク 5 ～ 8 があるとする。また、ディスク 1 ～ 4 を障害グループ A に、ディスク 4 ～ 8 を障害グループ B に割り当てるとする。障害グループ A のディスク上のファイル・エクステントの場合、ASM により、障害グループ B のディスク上にエクステントの冗長コピーが常に格納される。この方法では、ディスク・コントローラ A が停止しても、各ファイル・エクステントのコピーが少なくとも 1 つは使用可能である。現在のストレージ・ハードウェア構成に基づいた障害グループの定義は、ストレージ管理者が行う。特に障害グループの割当てを行わない場合、ディスク・グループの各ディスクは、独自の障害グループに自動的に配置される。

初期化パラメータ・ファイル（initialization parameter file）

初期化パラメータ設定が含まれるテキスト・ファイル。**サーバー・パラメータ・ファイル**とは異なり、このパラメータ・ファイルはバイナリではなく、データベース・ホストに配置する必要はない。データベース・サーバーは、テキストベースの初期化パラメータ・ファイルを読み取ることはできるが、書き込むことはできない。ファイルの変更にはテキスト・エディタを使用できる。

スーパータイプ（supertype）

「**サブタイプ**」を参照。

スキーマ（schema）

データベース・オブジェクトの集合。ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタおよびデータベース・リンクなどの論理構造が含まれる。スキーマには、そのスキーマを制御するユーザーの名前が付いている。

関連項目：「**論理構造**」

スタンバイ・データベース（standby database）

障害時の保護に使用できる本番データベースのコピー。スタンバイ・データベースを本番データベースからのアーカイブ REDO ログで更新し、常に最新の状態に保つことができる。災害によって本番データベースが破損した場合は、スタンバイ・データベースをアクティブ化して新規の本番データベースにできる。

制御ファイル（control file）

データベースの物理構造を記録し、データベース名、関連データベースおよび REDO ログ・ファイルの名前と位置、データベース作成のタイムスタンプ、カレント・ログ順序番号およびチェックポイント情報を含むファイル。

関連項目：「**物理構造**」、「**REDO ログ**」

整合性（integrity）

「**データ整合性**」を参照。

整合性制約（integrity constraint）

表の列に対してルールを定義する宣言メソッド。整合性制約によって、データベースに関連するビジネス・ルールが実施され、表への無効なエントリが防止される。

セグメント（segment）

論理データベース記憶域の第 3 レベル。セグメントはエクステントの集合で、それぞれ特定のデータ構造体に割り当てられ、それら全体が同じ表領域に格納されている。

関連項目：「**エクステント**」、「**データ・ブロック**」

セッション（session）

ユーザー・プロセスを使用した、ユーザーから Oracle データベース・インスタンスへの特定の接続。セッションは、ユーザーが接続した時点から、接続を切断するかデータベース・アプリケーションを終了する時点まで続く。

関連項目：「**接続**」、「**インスタンス**」、「**ユーザー・プロセス**」

接続（connection）

ユーザー・プロセスと Oracle データベース・インスタンスとの間の通信経路。

関連項目：「**セッション**」、「**ユーザー・プロセス**」

専用サーバー（dedicated server）

1 つのサーバー・プロセスが 1 つのユーザー・プロセスの要求を処理するデータベース・サーバー構成。

関連項目：「**共有サーバー**」

チェック制約（CHECK constraint）

列または列の集合に対するチェック制約では、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。DML 文の結果でチェック制約の条件が FALSE に評価される場合、その文はロールバックされる。

チェックポイント（checkpoint）

データベースの REDO スレッド内で SCN を定義するデータ構造。**制御ファイル**と各データファイルのヘッダーに記録される重要なリカバリ要素。

データ整合性（data consistency）

多数のユーザーがデータに同時にアクセスできる（同時実行性）マルチユーザー環境におけるデータ整合性とは、ユーザー自身および他のユーザーのトランザクションによる変更を表示可能にし、各ユーザーが整合性のとれたデータのビューを参照することを意味する。

関連項目：「**同時実行性**」

データ整合性（data integrity）

受入れ可能なデータに関する規格を指定するビジネス・ルール。このルールは整合性制約およびトリガーを使用してデータベースに適用され、無効なエントリが表に登録されることを防止する。

関連項目：「整合性制約」、「トリガー」

データ・セグメント（data segment）

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

各クラスタには、1 つのデータ・セグメントがある。クラスタ内のあらゆる表のデータが、そのクラスタのデータ・セグメントに格納される。

関連項目：「クラスタ」、「エクステント」、「セグメント」

データ・ディクショナリ（data dictionary）

特定のデータベースに関する読取り専用の参照として使用される中心的な一連の表やビュー。データ・ディクショナリには、次の情報が保存される。

- データベースの論理構造と物理構造
- データベースの有効なユーザー
- 整合性制約に関する情報
- スキーマ・オブジェクトに割り当てられている領域の量とその使用率

データ・ディクショナリはデータベースの作成時に作成され、データベースの構造が更新されると自動的に更新される。

データファイル（datafile）

ディスク上に Oracle が作成した物理ファイルで、表や索引などのデータ構造が含まれる。データファイルに、データベースのデータが保存される。データファイルが属することのできるデータベースは 1 つのみで、オペレーティング・システムのファイル・システムか、自動ストレージ管理のディスク・グループに配置される。

関連項目：「索引」、「物理構造」

データファイルのコピー（datafile copy）

次のどちらかによって生成されるディスク上のデータファイルのコピー。

- Recovery Manager の `COPY` コマンド
- オペレーティング・システムのユーティリティ

データ・ブロック（datablock）

Oracle Database 内にあるデータ記憶域の最小論理単位。論理ブロック、Oracle ブロックまたはページとも呼ばれる。1 つのデータ・ブロックは、ディスク上の特定のバイト数の物理データベース領域に対応する。

関連項目：「エクステント」、「セグメント」

データベース（database）

1 つの単位として取り扱われるデータの集まり。データベースの目的は、関連する情報の格納や取出しである。各 Oracle データベース・インスタンスは、1 つのデータベースにのみアクセスする。

データベース全体のバックアップ（whole database backup）

データベースに属する制御ファイルとすべてのデータファイルのバックアップ。

データベース・バッファ（database buffer）

システム・グローバル領域内の情報を格納する各種メモリー構造のタイプの 1 つ。データベース・バッファには、直前に使用されたデータのブロックが格納される。

関連項目：「**システム・グローバル領域**」

データベース・バッファ・キャッシュ（database buffer cache）

直前に使用されたデータのブロックが格納されるシステム・グローバル領域内のメモリー構造。

関連項目：「**システム・グローバル領域**」

データベース・ライター・プロセス（database writer process: DBW*n*）

データファイルにバッファの内容を書き込む Oracle バックグラウンド・プロセス。DBW*n* プロセスは、データベース・バッファ・キャッシュ内の変更された（使用済）バッファをディスクに書き込む。

関連項目：「**バッファ・キャッシュ**」

データベース・リンク（database link）

名前を持つスキーマ・オブジェクト。あるデータベースから別のデータベースへのパスを記述する。分散データベースでグローバル・オブジェクト名が参照されると、データベース・リンクが暗黙的に使用される。

データ・リカバリ・アドバイザ（Data Recovery Advisor）

永続的なデータ障害を自動的に診断してユーザーに修復オプションを提示し、ユーザーのリクエストで修復を実行する Oracle Database インフラストラクチャ。データ・リカバリ・アドバイザの目的は、自動データ修復用の集中管理ツールを提供して**平均リカバリ時間**を短縮し、Oracle Database の管理性と信頼性を向上することにある。

定義変数（define variables）

フェッチされた各値を受け取るために定義された変数（位置、サイズおよびデータ型）。

ディスク・グループ（disk group）

1 つの論理単位として管理される 1 つ以上の自動ストレージ管理ディスク。自動ストレージ管理ディスクは、グループ内のファイルの内容を保持しながらディスク・グループとの間で追加または削除できる。データを均等に再配分するために必要な最小量の I/O のみが自動的に実行される。ディスク・グループに対するすべての I/O は、そのグループの全ディスクに自動的に分散される。

ディスパッチャ・プロセス（dispatcher processes: D*nnn*）

オプションのバックグラウンド・プロセス。共有サーバー構成を使用している場合のみ存在する。使用中の通信プロトコルごとに、少なくとも 1 つのディスパッチャ・プロセス（D000、. . .、D*nnn*）が作成される。各ディスパッチャ・プロセスは、接続されたユーザー・プロセスから利用できる共有サーバー・プロセスへの要求の経路を指示し、適切なユーザー・プロセスにその応答を戻す。

関連項目：「**共有サーバー**」

問合せブロック（query block）

表に対する自己完結型 DML。問合せブロックは、トップレベルの DML または副問合せの場合がある。

関連項目：「**DML**」

同時実行性（concurrency）

多数のユーザーによる同一のデータへの同時アクセス。マルチユーザー・データベース管理システムには、データが正しく更新または変更され、データ整合性が維持されるように、適切な同時実行性制御が必要である。

関連項目：「**データ整合性**」

トランザクション（transaction）

1 つ以上の SQL 文を含む論理的な作業単位。トランザクション内のすべての文は、まとめてコミットまたはロールバックされる。

関連項目：「コミット」、「ロールバック」

トランザクション・リカバリ（transaction recovery）

トランザクション・リカバリでは、障害が発生したインスタンスのコミットされていないトランザクションをすべてロールバックする必要がある。これらは、コミットされていなかった進行中のトランザクションであり、Oracle Database で取り消す必要がある。コミットされていないトランザクションがディスクに保存されることがある。この場合 Oracle Database は、UNDO データを使用して、データファイルに書き込まれたがコミットされていない変更の効果を元に戻す。

トリガー（trigger）

表またはビューの変更時に、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。たとえば、INSERT、UPDATE または DELETE 操作によって起動される。

パーティション（partition）

表または索引の下位の管理しやすいピース。表はパーティション化キーに基づいてパーティション化される。たとえば、販売履歴表は販売日によってパーティション化され、各カレンダ四半期に 1 つのパーティションが割り当てられる。大規模な表の場合は、パーティションによって問合せパフォーマンスが向上し、表の管理が簡単になる。

バイト・セマンティクス（byte semantics）

文字列の長さがバイトで測定される。

バックグラウンド・プロセス（background process）

バックグラウンド・プロセスでは、各ユーザー・プロセスごとに実行される複数の Oracle プログラムによって処理されるようなファンクションを統合する。バックグラウンド・プロセスは、I/O を非同期的に実行し、他の Oracle プロセスを監視することにより、並列性を強化してパフォーマンスおよび信頼性を向上させる。

Oracle Database は、インスタンスごとに一連のバックグラウンド・プロセスを作成する。

関連項目：「インスタンス」、「プロセス」、「Oracle プロセス」、「ユーザー・プロセス」

バッファ・キャッシュ（buffer cache）

Oracle Database データ・ブロックのコピーを保持する SGA 部分。インスタンスに同時接続されたユーザー・プロセスはすべて、バッファ・キャッシュへのアクセスを共有する。

このキャッシュのバッファは、次の 2 つのリストで編成される。使用済リストおよび最低使用頻度リスト（LRU）である。使用済リストは、使用済バッファを保持する。使用済バッファとは、修正されたが、まだディスクに書き込まれていないデータを含むバッファである。最低使用頻度リスト（LRU）は、使用可能バッファ（変更されておらず使用可能）、使用中バッファ（現在アクセス中）および使用済リストに移動していない使用済バッファを保持する。

関連項目：「システム・グローバル領域」

非一貫性バックアップ（inconsistent backup）

一部のファイルにそのファイルのチェックポイントより後に行われた変更が含まれているバックアップ。このタイプのバックアップは、リカバリを実行しないと一貫性を維持できない。通常、非一貫性バックアップは、データベース・バックアップをオンライン化することで作成される。つまり、ファイルのバックアップ中はデータベースがオープン状態になっている。非一貫性バックアップを作成するには、次のどちらかの時点でデータベースがクローズ状態になっているときにデータファイルのバックアップを作成する方法もある。

- 1 つの Oracle データベース・インスタンス（または Oracle Real Application Clusters 構成のすべてのインスタンス）に障害が発生した直後
- `SHUTDOWN ABORT` を使用してデータベースを停止した後

非一貫性バックアップが役立つのは、データベースが `ARCHIVELOG` モードの場合のみである。

関連項目：「**一貫性バックアップ**」、「**オンライン・バックアップ**」、「**システム変更番号**」、「**データベース全体のバックアップ**」

ビュー（view）

ビューとは、1 つ以上の表のデータをユーザーの必要にあわせて調整したデータ表現。ビューは、ストアド・クエリーとみなすこともできる。ビューにデータは実際には含まれていない。データは基になる表から導出される。

表の場合と同じように、ビューに対しても、いくつかの制限付きで、問合せ、更新、挿入および削除を実行できる。ビュー上で実行されるすべての操作は、その実表に影響する。

表（table）

Oracle Database におけるデータ記憶域の基本単位。表データは、行と列に格納されている。

関連項目：「**列**」、「**行**」

表領域（tablespace）

関連する論理構造をまとめてグループ化したデータベース記憶域の単位。

関連項目：「**論理構造**」

品質保証契約（service level agreement: SLA）

サービス・プロバイダとサービス・コンシューマの間の契約。通常は、サービス内容、サービスの最大許容中断、サービス配信の評価の担当者、およびプロバイダが契約条件を履行できなかった場合の対応などが定められている。

物理構造（physical structures）

データファイル、REDO ログ・ファイルおよび制御ファイルを含めた Oracle Database の物理データベース構造。

関連項目：「**論理構造**」

物理バックアップ（physical backups）

ある場所から別の場所にコピーされた物理データベース・ファイル。この場合のファイルは、データファイル、アーカイブ REDO ログまたは制御ファイル。物理バックアップを作成するには、Recovery Manager を使用する方法と、UNIX の cp などのオペレーティング・システム・コマンドを使用する方法がある。

関連項目：「**論理バックアップ**」

フラッシュ・リカバリ領域（flash recovery area）

制御ファイルやオンライン REDO ログのコピー、アーカイブ・ログ、フラッシュバック・ログおよび RMAN のバックアップなど、リカバリ関連ファイルの格納に使用できるオプションのディスク位置。フラッシュ・リカバリ領域のファイルは、Oracle Database により自動的に管理される。ユーザーは、フラッシュ・リカバリ領域の最大サイズであるディスクの割当てを指定できる。

プログラム・グローバル領域（program global area: PGA）

サーバー・プロセス用のデータや制御情報を含むメモリー・バッファ。PGA はサーバー・プロセスの起動時に、Oracle Database によって作成される。PGA 内の情報は Oracle Database の構成によって異なる。

プロセス（process）

Oracle データベース・インスタンスにおけるプロセスごとに特定のジョブが実行される。Oracle Database とデータベース・アプリケーションの作業を複数のプロセスに分割することにより、複数のユーザーやアプリケーションが同時に 1 つのデータベース・インスタンスに接続できる。

関連項目：「**Oracle プロセス**」、「**ユーザー・プロセス**」

分散処理（distributed processing）

複数のコンピュータを使用するソフトウェア・アーキテクチャ。関連するジョブの集まりを分割処理する。分散処理により、1 つのコンピュータでのワークロードが軽減する。

平均リカバリ時間（mean time to recover: MTTR）

データベースのインスタンス・リカバリまたはメディア・リカバリの所要時間。たとえば、ディスク障害からのメディア・リカバリについて、目標を 10 分に設定できる。メディア・リカバリの MTTR には、検出速度、メディア・リカバリの実行に使用する方法のタイプおよびデータベースの規模など、様々な要因が影響する。

マウントされたデータベース（mounted database）

起動されており、オープンしているデータベースに対応付けられた制御ファイルを持つ**インスタンス**。データベースはオープンせずにマウントできる。通常、データベースをこの状態にするのは、メンテナンスやリストアおよびリカバリ操作を実行する場合である。

マテリアライズド・ビュー（materialized view）

マテリアライズド・ビューは、問合せ結果を別々のスキーマ・オブジェクトに格納して、表データへのアクセスを提供する。

関連項目：「**ビュー**」

ユーザー・プロセス（user process）

ユーザー・プロセスは、アプリケーションまたは Oracle のツール製品のコードを実行する。

関連項目：「**プロセス**」、「**Oracle プロセス**」

ユーザー名（user name）

Oracle Database および他のユーザーがユーザーを識別するための名前。すべてのユーザー名にはパスワードが関連付けられており、Oracle Database に接続するときはユーザー名とパスワードの両方を入力する必要がある。

優先順位逆転（priority inversion）

優先順位逆転は、優先順位の高いジョブが優先順位の低いジョブより少ないリソース量で実行される場合に発生する。したがって、予定した優先順位が逆転する。

読取り一貫性（read consistency）

マルチユーザー環境では、Oracle Database の読取り一貫性によって次の内容が保証される。

- 文が参照する一連のデータは、文の実行中不変である（文レベルの読取り一貫性）。
- データベース・データの読み書きでは、他のユーザーによる同じデータの読み書きを待機しない。データベース・データへの書込みでは、同時トランザクションで同じ行を更新している他のユーザーを待機するのみである。

関連項目：「**同時実行性**」、「**データ整合性**」

読取り専用データベース（read-only database）

ALTER DATABASE OPEN READ ONLY コマンドを使用してオープンされたデータベース。その名が示すように、読取り専用データベースは問合せ専用であり、変更はできない。Oracle Database では、**スタンバイ・データベース**を読取り専用モードで実行できる。つまり、プライマリ・データベースの最新の緊急時代替としての役割を果しながら問い合せることができる。

ラージ・プール（large pool）

Oracle Database のバックアップ処理とリストア処理、I/O サーバー・プロセスおよび共有サーバーと Oracle XA のセッション・メモリー用に、大量のメモリー割当てを提供するシステム・グローバル領域内のオプション領域。

関連項目：「**システム・グローバル領域**」、「**プロセス**」、「**共有サーバー**」、「**Oracle XA**」

列（column）

特定のデータ・ドメインを表すデータベース表内の垂直方向の領域。列には、列名および特定のデータ型がある。たとえば、従業員情報の表では、すべての従業員の採用年月日は 1 つの列で構成される。

関連項目：「**行**」、「**表**」

ロールバック（rolling back）

リカバリの**ロールフォワード**段階でデータベースに適用された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・セグメントを使用して取り消すこと。

関連項目：「**コミット**」、「**ロールフォワード**」

ロールバック・セグメント（rollback segment）

ロールバック情報を一時的に格納するために、データベース管理者が作成した論理データベース構造。ロールバック・セグメントには、コミットされるまでに、トランザクションの SQL 文で変更された古いデータが格納される。この UNDO 格納方法は非推奨になった。

関連項目：「**コミット**」、「**論理構造**」、「**セグメント**」、「**自動 UNDO 管理モード**」

ロールフォワード（rolling forward）

データファイルと制御ファイルに対する変更をリカバリするために、これらのファイルに REDO レコードまたは増分バックアップを適用すること。

関連項目：「**ロールバック**」

ログ・ライター・プロセス（log writer process: LGWR）

ログ・ライター・プロセス（LGWR）は、REDO ログ・バッファ管理、つまりディスク上の REDO ログ・ファイルへの REDO ログ・バッファの書込みを実行する。LGWR は、最後の書込み以後にバッファにコピーされた REDO エントリすべてを、REDO ログ・ファイルに書き込む。

関連項目：「**REDO ログ**」

論理構造（logical structures）

表領域、スキーマ・オブジェクト、データ・ブロック、エクステントおよびセグメントを含めた Oracle Database の論理構造。物理構造と論理構造は分離されているので、論理記憶構造へのアクセスに影響を与えずに、データの物理記憶域を管理できる。

関連項目：「**物理構造**」

論理バックアップ（logical backups）

Oracle Export Utility が SQL を使用してデータベース・データを読み取り、オペレーティング・システム・レベルでバイナリ・ファイルにエクスポートするバックアップ。その後、Oracle ユーティリティを使用してデータをデータベースにインポートできる。Oracle Export Utility を使用して実行したバックアップは、Recovery Manager によるバックアップと次の点で異なる。

- データベースの論理オブジェクトは、そのオブジェクトを含むファイルから独立してエクスポートされる。
- 論理バックアップは、異なるプラットフォーム上であっても異なるデータベースにインポートできる。Recovery Manager によるバックアップには、データベース間またはプラットフォーム間での移植性がない。

関連項目：「**物理バックアップ**」

索引

数字

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

れ

ろ

わ