Oracle9*i*

データベース概要

リリース 2（9.2）

2002 年 7 月

部品番号：J06245-01

ORACLE®

Oracle9*i* データベース概要 , リリース 2（9.2）

部品番号：J06245-01

原本名：Oracle9*i* Database Concepts, Release 2 (9.2)

原本部品番号：A96524-01

原本著者：Michele Cyran

原本協力者：Lance Ashdown, Cathy Baird, Sandeepan Banerjee, Mark Bauer, Ruth Baylis, Pradeep Bhanot, Janet Blowney, Allen Brumm, Ted Burroughs, Larry Carpenter, Donna Carver, Chandra Chandrasekar, Gary Chen, Amit Ganesh, Tom Grant, Mike Hartstein, John Haydu, Susan Hillson, Dominique Jeunot, Archna Kalra Johnson, Vishy Karra, Alex Keh, Susan Kotsovolos, Sushil Kumar, Tirthankar Lahiri, Paul Lane, Simon Law, Jeff Levinger, Yunrui Li, Bryn Llewellyn, Diana Lorentz, Lenore Luscher, Sheryl Maring, Ben Meng, Kuassi Mensah, Tony Morales, Ari Mozes, Subramanian Muralidhar, Ravi Murthy, Sujatha Muthulingam, Gary Ngai, Kant Patel, Ananth Raghavan, Jack Raitto, Beck Reitmeyer, Ann Rhee, Kathy Rich, John Russell, Vivian Schupmann, Ravi Shankar, Mark Smith, Richard Smith, Ekrem Soylemez, Marie St. Gelais, Debbie Steiner, Bob Thome, Anh-Tuan Tran, Randy Urbano, Stephen Vivian, Daniel Wong, Wanli Yang, Ruiling Zhang, Zulaikha, Valarie Moore

目次

4 データ・ディクショナリ

6 アプリケーション・アーキテクチャ

7 メモリー・アーキテクチャ

8 プロセス・アーキテクチャ

9 データベース・リソースの管理

第 IV 部　データ

10　スキーマ・オブジェクト

11 パーティション表とパーティション索引

12 システム固有のデータ型

13 オブジェクト・データ型とオブジェクト・ビュー

第 V 部　データ・アクセス

14　SQL、PL/SQL および Java

15 スキーマ・オブジェクト間の依存性

16 トランザクションの管理

17 トリガー

第 VI 部　パラレル SQL とダイレクト・ロード・インサート

18　SQL 文のパラレル実行

19　ダイレクト・パス・インサート

第 VII 部　データの保護

20　データの並行性と整合性

21 データ整合性

22 データベース・アクセスの制御

23 権限、ロールおよびセキュリティ・ポリシー

24 監査

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 サーバーの全機能について説明します。Oracle 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、Oracle サーバーの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 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

このマニュアルは、データベース管理者、システム管理者および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 サーバーの基礎知識」**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

このマニュアルの構成

このマニュアルは次の部と章で構成されています。

第 I 部 「Oracle の概要」

第 1 章「Oracle サーバーの基礎知識」

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

第 II 部 「データベース構造」

第 2 章「データ・ブロック、エクステントおよびセグメント」

Oracle データベース内の各種オブジェクトにデータがどのように格納され、記憶域がどのように割り当てられ、使用されるかについて説明します。

第 3 章「表領域、データ・ファイルおよび制御ファイル」

Oracle データベース内で、物理的な記憶域が、表領域と呼ばれる論理的な区画にどのように分割されているかについて説明します。リカバリに使用するファイル（制御ファイル）と表領域（データ・ファイル）に関連付けられている、物理的なオペレーティング・システム・ファイルについても説明します。

第 4 章「データ・ディクショナリ」

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

第 III 部 「Oracle インスタンス」

第 5 章「データベースとインスタンスの起動と停止」
Oracle インスタンスについて説明し、データベース管理者が Oracle データベース・システムへのアクセスを制御する方法について説明します。

第 6 章「アプリケーション・アーキテクチャ」
Oracle データ・サーバーを実行できる分散処理環境について説明します。

第 7 章「メモリー・アーキテクチャ」
Oracle データベース・システムで使用するメモリー構造について説明します。

第 8 章「プロセス・アーキテクチャ」
Oracle インスタンスのプロセス・アーキテクチャと、Oracle で利用できる各種のプロセス構成について説明します。

第 9 章「データベース・リソースの管理」
データ・リソース・マネージャを使用してリソース使用を制御する方法について説明します。

第 IV 部 「データ」

第 10 章「スキーマ・オブジェクト」
表、ビュー、数値順序およびシノニムなど、特定のユーザーのドメイン（スキーマ）内に作成できるデータベース・オブジェクトについて説明します。また、索引、マテリアライズド・ビュー、ディメンションおよびクラスタなど、データ検索を効率化するオプションの構造についても説明します。

第 11 章「パーティション表とパーティション索引」
パーティション化を利用して、大規模な表や索引を管理しやすい区画に分割する方法を説明します。

第 12 章「システム固有のデータ型」
Oracle データベースの表に格納できるリレーショナル・データのデータ型について説明します。たとえば、固定長の文字列、可変長の文字列、数値、日付、バイナリ・ラージ・オブジェクト（BLOB）などがあります。

第 13 章「オブジェクト・データ型とオブジェクト・ビュー」
Oracle が提供するオブジェクト拡張機能の概要について説明します。

第 V 部 「データ・アクセス」

第 14 章「SQL、PL/SQL および Java」

Oracle と対話するために使用する SQL（Structured Query Language）と、SQL への Oracle の手続き型言語機能拡張である PL/SQL について説明します。データベース内に格納される PL/SQL プログラム・ユニットである、プロシージャ、ファンクションおよびパッケージと呼ばれる手続き型言語構造について説明します。

第 15 章「スキーマ・オブジェクト間の依存性」

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

第 16 章「トランザクションの管理」

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

第 17 章「トリガー」

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

第 VI 部「パラレル SQL とダイレクト・パス・インサート」

第 18 章「SQL 文のパラレル実行」

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

第 19 章「ダイレクト・パス・インサート」

シリアル・インサートまたはパラレル・インサートのダイレクト・パス・インサート機能と、`NOLOGGING` 句について説明します。

第 VII 部「データの保護」

第 20 章「データの並行性と整合性」
マルチ・ユーザー環境において、Oracle が共有情報への同時アクセスを提供し、その情報の正確さを維持する仕組みについて説明します。さらに、複数のユーザーが同時に操作を実行しても相互に干渉し合わないようにするために Oracle が自動的に実行するメカニズムについて説明します。

第 21 章「データ整合性」
データ整合性と、整合性を規定するために使用できる整合性制約の宣言について説明します。

第 22 章「データベース・アクセスの制御」
データとデータベース・リソースへのユーザー・アクセスを制御する方法について説明します。

第 23 章「権限、ロールおよびセキュリティ・ポリシー」
システム・レベルとスキーマ・オブジェクト・レベルでのセキュリティについて説明します。

第 24 章「監査」
Oracle の監査機能がデータベース・アクティビティを追跡する仕組みについて説明します。

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

付録 B「廃止機能に関する情報」
データベースを以前のリリースの Oracle で作成した場合に重要な概念情報について説明します。

用語集
このマニュアルで使用する用語が説明されています。

関連文書

詳細は、次の Oracle リソース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、『Oracle9*i* データベース移行ガイド』を参照してください。
- Oracle サーバーの管理方法に関する情報は、『Oracle9*i* データベース管理者ガイド』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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- Oracle 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、『Oracle9*i* データベース・パフォーマンス・プランニング』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、『Oracle9*i* データ・ウェアハウス・ガイド』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、『Oracle9*i* サンプル・スキーマ』を参照してください。

リリース・ノート、インストレーション・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

`http://otn.oracle.co.jp/membership/`

OTN-J のユーザー名とパスワードを取得済みであれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

`http://otn.oracle.co.jp/document/`

表記規則

このマニュアル・セットの本文とコード例に使用されている表記規則について説明します。

本文の表記規則

本文中には、特別な用語が一目でわかるように様々な表記規則が使用されています。次の表は、本文の表記規則と使用例を示しています。

規則	意味	例
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。	この句を指定する場合は、**索引構成表**を作成します。
固定幅フォントの大文字	固定幅フォントの大文字は、システムにより指定される要素を示します。この要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージとメソッドの他、システム指定の列名、データベース・オブジェクトと構造体、ユーザー名、およびロールがあります。	この句は、NUMBER 列に対してのみ指定できます。 BACKUP コマンドを使用すると、データベースのバックアップを作成できます。 USER_TABLES データ・ディクショナリ・ビューの TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびサンプルのユーザー指定要素を示します。この要素には、コンピュータ名とデータベース名、ネット・サービス名、接続識別子の他、ユーザー指定のデータベース・オブジェクトと構造体、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニット、およびパラメータ値があります。 **注意：**一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。	sqlplus と入力して SQL*Plus をオープンします。 パスワードは orapwd ファイルに指定されています。 データ・ファイルと制御ファイルのバックアップを /disk1/oracle/dbs ディレクトリに作成します。 department_id、department_name および location_id の各列は、hr.departments 表にあります。 初期化パラメータ QUERY_REWRITE_ENABLED を true に設定します。 oe ユーザーで接続します。 これらのメソッドは JRepUtil クラスに実装されます。

規則	意味	例
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。*old_release* は、アップグレード前にインストールしたリリースです。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus またはその他のコマンドラインを示します。次のように、固定幅フォントで、通常の本文とは区別して記載しています。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例の記載上の表記規則と使用例を示します。

規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のどちらかを示します。	
	■ 例に直接関係のないコード部分が省略されていること。	CREATE TABLE ... AS *subquery*;
	■ コードの一部が繰返し可能であること。	SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関係のない数行のコードが省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fs1/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fs1/dbs/tbs_09.dbf 9 rows selected.

規則	意味	例
その他の表記	大カッコ、中カッコ、縦線および省略記号以外の記号は、示されているとおりに入力してください。	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
イタリック	イタリックの文字は、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

大文字	大文字は、システムにより指定される要素を示します。これらの用語は、ユーザー定義用語と区別するために大文字で記載されています。大カッコで囲まれている場合を除き、記載されているとおりの順序とスペルで入力してください。ただし、この種の用語は大 / 小文字区別がないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザー指定のプログラム要素を示します。たとえば、表名、列名またはファイル名を示します。
	注意：一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

第 I 部

Oracle の概要

第 I 部では、Oracle サーバー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 サーバーの基礎知識」

1

Oracle サーバーの基礎知識

この章では、Oracle 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。

- データベース構造と領域管理の概要
- データ・アクセスの概要
- メモリー構造とプロセスの概要
- アプリケーション・アーキテクチャの概要
- 分散データベースの概要
- データの並行性と一貫性の概要
- データベース・セキュリティの概要
- データベース管理の概要
- データ・ウェアハウスの概要
- 高可用性の概要
- コンテンツ管理の概要

注意： この章には、Oracle9*i* Standard Edition と Oracle9*i* Enterprise Edition の両方に関連する情報が含まれています。この章に記載されている機能とオプションの中には、Oracle9*i* Enterprise Edition を購入した場合にのみ使用可能なものも含まれています。Oracle9*i* Standard Edition と Oracle9*i* Enterprise Edition の相違点の詳細は、『Oracle9*i* データベース新機能』を参照してください。

データベース構造と領域管理の概要

Oracle データベースとは、1 単位として扱われるデータの集合です。データベースの目的は、関連する情報の格納や取出しです。データベース・サーバーは、情報管理の問題を解決する鍵となります。一般に**サーバー**では、多数のユーザーが同時に同じデータ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。このような管理を行いながら、一方では高いパフォーマンスを実現し、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。

データベースには、**論理構造**と**物理構造**があります。物理構造と論理構造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。

論理データベース構造

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

スキーマとスキーマ・オブジェクト

スキーマとは、データベース・オブジェクトの集合です。スキーマはデータベース・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。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。(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られます。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られます。)

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。

> **関連項目：** これらのスキーマ・オブジェクトと、ディメンション、シーケンス・ジェネレータ、シノニム、索引構成表、ドメイン索引、クラスタおよびハッシュ・クラスタなど、他のスキーマ・オブジェクトの詳細は、第 10 章「スキーマ・オブジェクト」を参照してください。

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。255 列以下のデータを含むデータベースの表の各行は、1 つ以上の行断片として格納されます。たとえば、従業員データベースの表には、従業員番号という列があり、その列の各行には従業員番号が格納されます。

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格納されていません。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。

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

ビューは事前に定義された行と列の集合のみにアクセスを限定して、表のセキュリティ・レベルを強化します。また、データの複雑さを隠し、複雑な問合せを格納します。

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。索引を作成すると、データ検索のパフォーマンスが向上します。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。それと同じように、Oracle の索引を使用すると表データにアクセスできます。

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

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

索引は、パーティションに分割できます。

関連項目： 第 11 章「パーティション表とパーティション索引」

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、ユーザーとアプリケーションに対して透過的です。クラスタ化表に格納されているデータは、クラスタ化されていない表のデータと同じように SQL を使用してアクセスします。

データ・ブロック、エクステントおよびセグメント

Oracle では、データ・ブロック、エクステントおよびセグメントなどの論理記憶構造によって、ディスク領域の使用をきめ細かく制御できます。

関連項目： 第 2 章「データ・ブロック、エクステントおよびセグメント」

Oracle データ・ブロック 最少レベルとして、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、初期化パラメータ DB_BLOCK_SIZE で指定します。また、他のブロック・サイズを 5 つまで指定できます。データベースは、Oracle データ・ブロック内の空きデータベース領域を使用して領域を割り当てます。

関連項目： 3-13 ページ「マルチ・ブロック・サイズ」

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、特定の論理構造に割り当てられるエクステントの集合です。次の表に、各種のセグメントを示します。

セグメント	説明
データ・セグメント	クラスタ化されていない表には、それぞれ 1 つのデータ・セグメントがあります。表のデータはすべて、データ・セグメントのエクステントに格納されます。 パーティション表の場合は、各パーティションに 1 つずつデータ・セグメントがあります。 各クラスタには、それぞれ 1 つのデータ・セグメントがあります。クラスタ内のあらゆる表のデータが、そのクラスタのデータ・セグメントに格納されます。
索引セグメント	各索引には、そのデータをすべて格納する索引セグメントが 1 つあります。 パーティション索引の場合は、各パーティションに 1 つずつ索引セグメントがあります。

セグメント	説明
一時セグメント	一時セグメントは、**SQL** 文の実行を完了するために一時的な作業領域が必要なときに、Oracle によって作成されます。SQL 文の実行が終了すると、一時セグメントのエクステントは後で使用できるようにシステムに戻されます。
ロールバック・セグメント	自動 UNDO 管理モードで操作している場合、データベース・サーバーは表領域を使用して UNDO 領域を管理します。「自動 UNDO 管理」を使用することをお薦めします。 ただし、手動 UNDO 管理モードで操作している場合、データベース管理者は、UNDO 情報を一時的に格納するために、データベースごとに 1 つ以上のロールバック・セグメントを作成します。 ロールバック・セグメント内の情報は、データベース・リカバリ中の次の場合に使用されます。 ■ 読込み一貫性を備えたデータベース情報を生成する場合。 ■ ユーザーが、コミットされていない**トランザクション**を**ロールバック**する場合。

Oracle は、1 つのセグメントの既存のエクステントがいっぱいになった時点で、領域を動的に割り当てます。つまり、あるセグメントのエクステントがいっぱいになると、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。

関連項目：

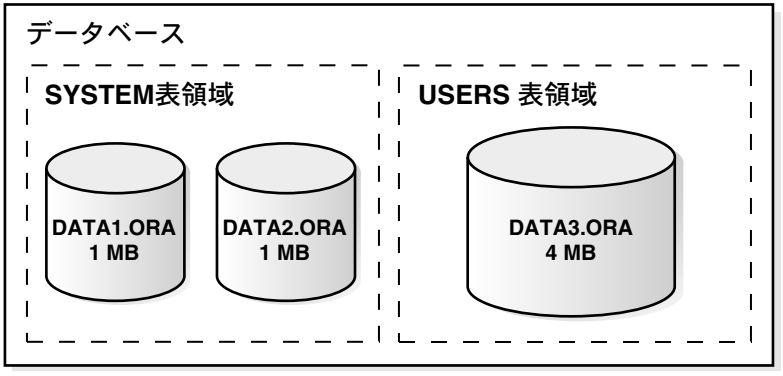
- 2-16 ページ「自動 UNDO 管理」
- 1-42 ページ「読込み一貫性」
- 1-52 ページ「データベースのバックアップおよびリカバリの概要」

表領域

データベースは、関連する論理構造がグループ化された**表領域**と呼ばれる論理記憶単位に分けられます。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。

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

図 1-1 データベース、表領域およびデータ・ファイル



この図から次のことがわかります。

- 各データベースは、論理的に 1 つ以上の表領域に分けられます。
- 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。
- 表領域のデータ・ファイルのサイズを合せると、表領域全体の記憶容量になります（SYSTEM 表領域は 2MB、USERS 表領域は 4MB の記憶容量を持ちます）。
- データベースの表領域の記憶容量を合せると、データベース全体の記憶容量になります（6MB）。

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

物理データベース構造

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

データ・ファイル

すべての Oracle データベースには、1 つ以上の物理**データ・ファイル**があります。データ・ファイルには、すべてのデータベース・データが格納されます。表や索引などの論理データベース構造のデータは、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 つのデータ・ファイルは、1 つのデータベースのみに対応付けられます。
- データ・ファイルには、データベースの領域をすべて使用してしまった場合にファイルが自動的に拡張されるように設定できる特性があります。
- 前述のように、1 つ以上のデータ・ファイルによって、表領域と呼ばれるデータベース記憶域の論理単位が形成されます。

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込まれ、Oracle のメモリー・キャッシュに格納されます。たとえば、あるユーザーがデータベースの表に入っている一部のデータにアクセスする場合を考えます。要求された情報がデータベースのメモリー・キャッシュに存在しない場合には、その情報は適切なデータ・ファイルから読み込まれて、メモリーに格納されます。

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

> **関連項目：** Oracle のメモリーおよびプロセス構造の詳細は、1-22 ページの「メモリー構造とプロセスの概要」を参照してください。

REDO ログ・ファイル

すべての Oracle データベースには、2 つ以上の **REDO ログ・ファイル**の集合があります。REDO ログ・ファイルの集合は、データベースの REDO ログと呼ばれます。REDO ログは、REDO エントリ（**REDO レコード**とも呼ばれる）で構成されます。

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

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

REDO ログ・ファイル内の情報は、データベース・データをデータ・ファイルに書き込む妨げとなるシステム障害やメディア障害が起きた場合に、データベースをリカバリするためにのみ使用されます。たとえば、予期しない停電によってデータベース操作が停止した場合、メモリー内のデータはデータ・ファイルに書き込まれずに失われます。ただし、電源が復旧した後、データベースがオープンされると、失われたデータがすべてリカバリされます。Oracle は、最新の REDO ログ・ファイル内にある情報をデータベースのデータ・ファイルに反映させることにより、停電が起きた時点までデータベースをリストアします。

リカバリ操作中に REDO ログ・ファイルの情報を反映させる処理のことを、**ロールフォワード**と呼びます。

> **関連項目：** REDO ログ・ファイルの詳細は、1-52 ページの「データベースのバックアップおよびリカバリの概要」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、データベースの物理構造を指定するエントリが含まれています。たとえば、制御ファイルには次のような情報が含まれます。

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

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

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

関連項目： データベース・リカバリにおける制御ファイルの使用の詳細は、1-52 ページの「データベースのバックアップおよびリカバリの概要」を参照してください。

データ・ユーティリティ

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

エクスポート・ユーティリティ エクスポート・ユーティリティを使用すると、異なるハードウェアやソフトウェア構成のプラットフォーム上にある Oracle データベース間でも、データ・オブジェクトを転送できます。エクスポート・ユーティリティでは、Oracle データベースからオブジェクト定義と表データが抽出され、通常はディスクまたはテープにある Oracle のバイナリ形式のエクスポート・ダンプ・ファイルに格納されます。

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

Oracle データベースに対してエクスポート・ユーティリティを実行すると、表などのオブジェクトとその関連オブジェクトが抽出されてから、エクスポート・ダンプ・ファイルに書き込まれます。

インポート・ユーティリティ インポート・ユーティリティは、ある Oracle データベースからエクスポート・ユーティリティによって抽出されたデータ・オブジェクトを、別の Oracle データベースに挿入します。エクスポート・ダンプ・ファイルを読み取れるのは、インポート・ユーティリティのみです。インポート・ユーティリティでは、エクスポート・ユーティリティによって Oracle データベースから抽出されたオブジェクト定義と表データが読み込まれます。

また、エクスポートおよびインポート・ユーティリティを使用すると、オフライン・インスタンシエーションなど、Oracle アドバンスト・レプリケーション機能の特定部分の処理が容易になります。

関連項目： 『Oracle9*i* レプリケーション』

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

関連項目： エクスポート、インポートおよびSQL*Loaderの詳細は、『Oracle9*i*データベース・ユーティリティ』を参照してください。

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・ディクショナリ**が1つずつあります。Oracleデータ・ディクショナリは、データベースの**読取り専用**の参照として使用される、表およびビューの集合です。たとえば、データ・ディクショナリには、データベースの論理構造と物理構造の両方に関する情報が格納されます。また、データ・ディクショナリには次の情報も格納されます。

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

データ・ディクショナリは、データベースの作成時に作成されます。常にデータベースの正確な状態が反映されるように、データ・ディクショナリは、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により自動的に更新されます。データベースでは、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は、スキーマ・オブジェクトが存在しているかどうか、およびユーザーが適切なアクセス権を持っているかどうかを検証するためにデータ・ディクショナリを読み込みます。

関連項目： 第4章「データ・ディクショナリ」

データ・アクセスの概要

この項では、Oracle における業界標準のデータ・アクセス言語への準拠と、データの一貫性および整合性の制御について説明します。この項の内容は、次のとおりです。

- 「SQL の概要」
- 「オブジェクトの概要」
- 「PL/SQL の概要」
- 「Java の概要」
- 「トランザクションの概要」
- 「データ整合性の概要」
- 「SQL*Plus の概要」

SQL の概要

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 のテキスト文字列であり、次のように完全な SQL 文と等価である必要があります。

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

正常に実行できるのは完全な SQL 文のみです。次のような不完全な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
- データ操作言語（DML）文
- トランザクション制御文
- セッション制御文
- システム制御文
- 埋込み SQL 文

データ定義言語（DDL）文 この種の文は、スキーマ・オブジェクトを作成、変更、保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 この種の文はデータを操作します。たとえば、表の行の問合せ、挿入、更新および削除はすべて DML 操作です。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 この種の文は、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 この種の文を使用すると、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 この種の文は、Oracle サーバー・インスタンスのプロパティを変更します。システム制御文は、ALTER SYSTEM のみです。この文は、共有サーバーの最小数などの設定値の変更、セッションの停止およびその他の作業のために使用します。

埋込み SQL 文 この種の文は、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。たとえば、OPEN、CLOSE、FETCH および EXECUTE などがあります。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 権限の詳細は、1-45 ページの「データベース・セキュリティの概要」を参照してください。
- トランザクション制御文の詳細は、1-17 ページの「トランザクションの概要」を参照してください。

オブジェクトの概要

Oracle オブジェクト・テクノロジは、Oracle のリレーショナル・テクノロジ上に作成された抽象レイヤーです。組込みデータベース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、新規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。

オブジェクト型は、ユーザー定義であり、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、発注情報など、実社会のエンティティを抽象化したものです。

オブジェクト型と、可変長配列やネストした表などの関連するオブジェクト指向機能により、データベース内のデータについて高水準の編成方法とアクセス方法が提供されます。データはオブジェクト・レイヤーでも列と表に格納されますが、顧客や発注情報など、データを意味のあるものにする実社会のエンティティに関連してデータを操作できます。データベースの問合せでは、列と表を考慮するかわりに単に顧客を選択できます。

内部的には、オブジェクトに関する文は基本的にはリレーショナル表および列に関する文であり、従来どおりリレーショナル・データ型を操作して、データをリレーショナル表に格納できます。ただし、オブジェクト指向の機能を利用するかどうかを選択するオプションも用意されています。オブジェクト指向の機能を使用しながら引き続きほとんどのリレーショナル・データを操作する方法と、完全にオブジェクト指向のアプローチを使用するように移行する方法があります。たとえば、オブジェクト・データ型をいくつか定義し、オブジェクトをリレーショナル表の列に格納できます。また、既存のリレーショナル・データのオブジェクト・ビューを作成し、このデータをオブジェクト・モデルに従って表示してアクセスしたり、各行がオブジェクトとなるようにオブジェクト表にオブジェクト・データを格納することもできます。

オブジェクトの利点

通常、オブジェクト型モデルは C++ や Java のクラス・メカニズムに似ています。クラスと同様に、オブジェクトを使用すると実社会の複雑なビジネス・エンティティとロジックをモデル化できます。また、オブジェクトには再利用性があるため、より迅速かつ効率的にデータベース・アプリケーションを開発できます。Oracle では、データベース内でオブジェクト型がネイティブにサポートされるため、アプリケーション開発者はアプリケーションで使用するデータ構造に直接アクセスできます。クライアント側オブジェクトと、データが格納されているリレーショナル・データベースの列と表の間には、マッピング・レイヤーは不要です。オブジェクトの抽象化とオブジェクト動作のカプセル化により、アプリケーションを簡単に理解してメンテナンスできます。

> **関連項目：**『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』

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 が持つ使用しやすさと柔軟性に、IF ... THEN、WHILE、LOOP などの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、データベースに集約して格納できます。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。
- データ・アクセスを PL/SQL コードによって制御できます。この場合、別のアクセス・ルートが与えられないかぎり、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。
- アプリケーションからデータベースに PL/SQL ブロックを送信できるため、大量のネットワーク通信を行わずに複雑な操作を実行できます。

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。

PL/SQL プログラム・ユニット

プログラム・ユニットとは、ストアド・プロシージャ、ファンクション、パッケージ、トリガーおよび無名トランザクションです。

プロシージャとファンクション **プロシージャ**と**ファンクション**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みの形式でデータベース内に格納され、ユーザーまたはデータベース・アプリケーションによって実行されます。

プロシージャは値を戻さないのに対して、ファンクションはユーザーに必ず 1 つの値を戻します。それ以外の点は同じです。

パッケージ **パッケージ**によって、関連したプロシージャ、ファンクション、変数およびその他の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）。また、パフォーマンスも改善できます（たとえば、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ます）。

データベース・トリガー データベース・トリガーは、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される PL/SQL、Java または C 言語のプロシージャです。データベース管理のために、データベース・トリガーをいろいろな方法で使用できます。たとえば、データ生成の自動化、データ修正の監査、複雑な**整合性制約**の規定、複雑なセキュリティ許可のカスタマイズに使用できます。

自律型ブロック 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。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。

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 の主機能は、サーバー・アプリケーションの開発に最適です。たとえば、次の機能があります。

- 簡素化 - Java はオブジェクト・モデルが一貫して規定されるため、サーバー・アプリケーションで使用される他のほとんどの言語より単純です。クラス・ライブラリの大規模な規格セットにより、あらゆるプラットフォームで Java 開発者が強力なツールとして使用できます。
- 移植性 - Java はプラットフォーム間での移植性を備えています。プラットフォーム依存コードは Java で記述しても、マシン間でシームレスに移動するプログラムを簡単に記述できます。ホスティング対象となるプラットフォームでは Graphical User Interface（GUI）を直接サポートしていない Oracle サーバー・アプリケーションでは、Java がプラットフォーム移植性に関して持っている少数の問題を回避する傾向があります。
- 自動記憶域管理 - Java Virtual Machine（JVM）では、プログラムの実行中にメモリー割当てと割当て解除がすべて自動的に実行されます。Java プログラマは、メモリーの割当てや解放を明示的に行うことはできません。かわりに、これらのブックキーピング操作は JVM に依存して実行されて、新規オブジェクトの作成時にメモリーが割り当てられ、オブジェクトが参照されなくなるとメモリーの割当てが解除されます。後者の操作は、ガベージ・コレクションと呼ばれます。
- 強い型指定 - Java の変数を使用する前に、それが保持するオブジェクトのクラスを宣言する必要があります。Java の強い型指定により、Java アプリケーションと PL/SQL アプリケーション間でコールするための妥当で安全なソリューションが得られ、Java コールと SQL コールが同じアプリケーション内で統合されます。
- ポインタを使用しない - Java の構文は C 言語に似ていますが、ダイレクト・ポインタやポインタ操作はサポートされません。プリミティブ型を除くすべてのパラメータは、値渡しではなく参照渡しです（つまり、オブジェクト識別性が保たれます）。Java には、メモリーの破損とリークを防ぐ C 言語の下位レベルのポインタへのダイレクト・アクセス機能はありません。

- 例外処理 - Java 例外はオブジェクトです。Java の場合、開発者は特定のクラスのメソッドがスローできる例外を宣言する必要があります。
- セキュリティ - Java バイト・コードの設計と JVM により、Java バイナリ・コードが改ざんされていないかどうかを組込みメカニズムで検証できます。Oracle9*i* とともに SecurityManager のインスタンスがインストールされ、このインスタンスと Oracle データベースのセキュリティとの結合によって、Java メソッドをコールできるユーザーが判断されます。
- リレーショナル・データベースへの接続性の標準 - JDBC と SQLJ により、Java コードでリレーショナル・データベース内のデータにアクセスして操作できます。Oracle には、ベンダーに依存しない移植性を持った Java コードでリレーショナル・データベースにアクセスできるように、ドライバが用意されています。

関連項目： 第 14 章「SQL、PL/SQL および Java」

XML の概要

XML（Extensible Markup Language）は、Web 上でデータを識別し、記述するための標準的な手段です。XML は階層データを記述するための、判読可能で、マシンで認識可能な一般構文であり、多様なアプリケーション、データベース、E-Commerce、Java、Web 開発、検索などに適用できます。

Oracle サーバーには、組込み型でパフォーマンスのよい XML 格納および検索テクノロジである Oracle XML DB が組み込まれています。Oracle XML DB により、W3C の XML データ・モデルが Oracle サーバーに完全に吸収され、XML のナビゲートと問合せのための新しい標準的なアクセス方式を提供します。リレーショナル・データベース・テクノロジと XML テクノロジの利点を、すべて同時に活用できます。XML データベースの主な側面を次に示します。

- システム固有のデータ型 - `XMLType` - XML の格納と操作に使用します。複数の記憶領域オプション（`CLOB`、分解されたオブジェクト・リレーショナル）を `XMLType` とともに使用可能であり、データベース管理者はオリジナルに対する忠実さ、問合せの容易さ、再生成の容易さなどの要件を満たす記憶域を選択できます。`XMLType` を使用すると、XML データに対する問合せや OLAP ファンクションなどの SQL 操作と、SQL データに対する XPath 検索や XSL 変換などの XML 操作を実行できます。アプリケーションの様々な側面で高いパフォーマンスが得られるように、`XMLType` について通常の SQL 索引または Oracle Text 索引を作成できます。
- システム固有の XML 生成には、組込み SQL 演算子と PL/SQL パッケージが用意されており、SQL 問合せの結果を XML フォーマットで戻します。
- XML リポジトリには、フォルダ、アクセス制御、FTP および WebDAV プロトコルのサポートとバージョニングの機能があります。これにより、アプリケーションでは XML データの操作時にファイルの抽象化を保持できます。

XML データベースを補完しているのが、Oracle XML Developer's Kit（XDK）です。XDK は、開発およびランタイム・サポートのために一般に使用されるビルディング・ブロックま

たはユーティリティのセットです。Oracle XDK には、XML 文書の読込み、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。多様なデプロイメント・オプションを提供するために、Oracle XDK を Java、JavaBeans、C、C++ および PL/SQL で使用できます。Oracle XDK は、XML パーサー、XSLT Processor、XML Schema Processor、XML Class Generator、XML Transviewer Beans、XML SQL Utility および XSQL Servlet で構成されています。

アドバンスト・キューイング（AQ）は、Oracle データベースのメッセージ・キューイング機能です。この機能を使用すると、Oracle データベースから SQL 操作の場合と同様にメッセージ・キューイング操作を実行できます。メッセージ・キューイング機能により、アプリケーションとキューを使用している Oracle データベース上のユーザーの間で非同期通信が可能になります。AQ は、メッセージのエンキュー、デキュー、伝播および保証付き配信と、メッセージを配信できない場合の例外処理を行います。メッセージ・キューイングは、XML メッセージのペイロードに `XMLType` を利用します。

関連項目：

- 第 12 章「システム固有のデータ型」
- 『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』

トランザクションの概要

トランザクションは、単一のユーザーによって実行される 1 つ以上の SQL 文を含む論理作業単位です。Oracle と互換性のある ANSI/ISO SQL 規格によると、トランザクションはユーザーの最初の実行 SQL 文で開始されます。ユーザーが明示的にコミットまたはロールバックすると、トランザクションは終了します。

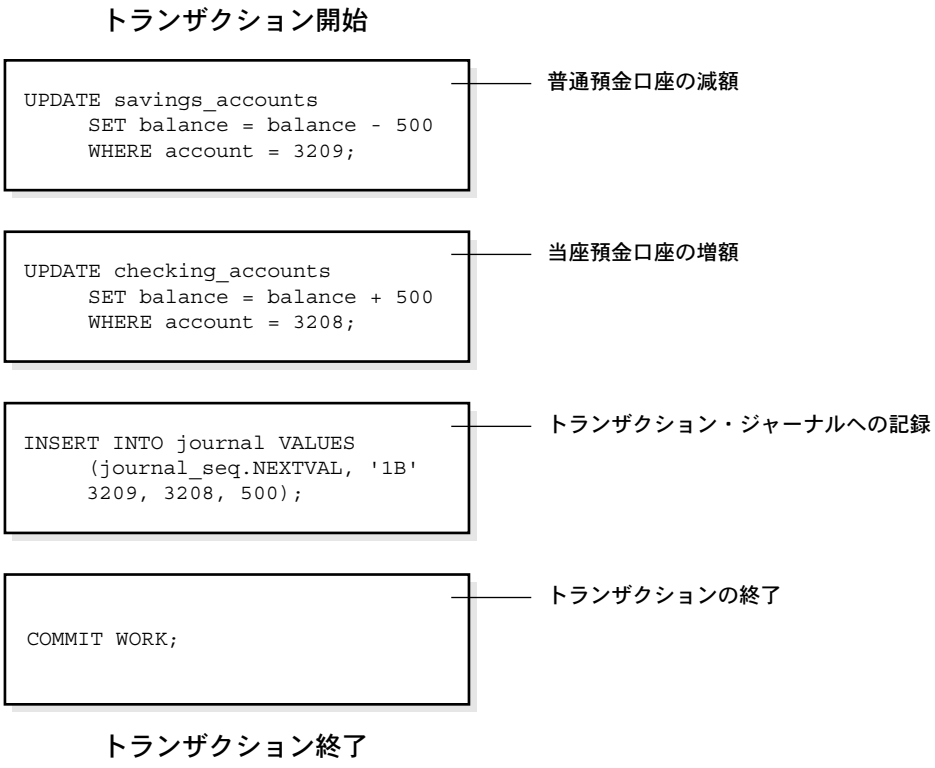
注意： Oracle9*i* は、SQL-99 コア仕様に準拠しています。

銀行のデータベースを考えてみます。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。つまり、普通預金口座の減額、当座預金口座の増額およびトランザクション・ジャーナルへのトランザクションの記録です。

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

図 1-2 に銀行のトランザクションの例を示します。

図 1-2 銀行のトランザクション



関連項目： Oracle の ANSI/ISO 規格への準拠の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

トランザクションのコミットとロールバック

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

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

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

セーブポイント

セーブポイントは、多数の SQL 文によるロング・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、トランザクションのカレント位置から、トランザクション内で宣言したセーブポイントまでの間に実行されたすべての作業を、選択的にロールバックできます。たとえば、大規模で複雑な一連の更新のどこにでもセーブポイントを設定できるため、エラーが発生してもすべての文を再発行する必要はありません。

トランザクションを使用したデータ整合性

トランザクション内の SQL 文が論理的にグループ化されるかぎり、ユーザーはトランザクションを使用してデータへの一貫した変更を保証できます。トランザクションには、1 つの論理作業単位に必要な部分を過不足なくすべて含める必要があります。トランザクションの開始から終了まで、参照される表データはすべて一貫した状態に維持されます。各トランザクションには、データに対して首尾一貫した 1 つの変更を加える SQL 文のみを含めます。

たとえば、銀行の例を思い出してください。2 つの口座間の送金（トランザクション）には、一方の口座の預金高を増やす処理（1 つの SQL 文）、他方の口座の預金高を減らす処理（1 つの SQL 文）、およびジャーナルにトランザクションを記録する処理（1 つの SQL 文）が必要です。これらの処理は、すべて失敗するか、すべて成功するかのどちらかです。つまり、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処理を、振替トランザクションに含めることはできません。そのような文は、別のトランザクションに組み込みます。

データ整合性の概要

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 はその無効な文をロールバックし、アプリケーションにエラーを戻します。Oracle は、データ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。

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。

整合性制約は表に定義され、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。

Oracle がサポートする整合性制約は次のとおりです。

- NOT NULL: 表の列に NULL（空のエントリ）を許可しません。
- 一意キー : 列（または列の集合）に重複値を許可しません。
- 主キー : 列（または列の集合）に重複値と NULL を許可しません。
- 外部キー : 列（または列の集合）の値が、それぞれ関連する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。また、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 に指示する、参照整合性アクションも定義します。
- CHECK: 制約の論理式を満たさない値を許可しません。

キー

キーは、いくつかのタイプの整合性制約の定義で使用されます。キーは特定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、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を記述するものです。キーの中にある個々の値を、**キー値**と呼びます。

次のようなタイプがあります。

- 主キー : 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。各表には 1 つの主キーのみが定義できます。
- 一意キー : 一意制約の定義に含まれる列（または列の集合）です。
- 外部キー : 参照整合性制約の定義に含まれる列（または列の集合）です。
- 参照キー : 同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキーです。

SQL*Plus の概要

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。SQL*Plus を通じて次の操作ができます。

- SQL 文と PL/SQL ブロックの入力、編集、格納、検索および実行
- 問合せ結果のフォーマット、計算の実行、格納、印刷および Web 出力の作成
- 表へのアクセスに使用する列の定義のリストと、SQL データベース間でのデータ・コピー
- エンド・ユーザーとの間のメッセージの送信と応答の受入れ
- データベース管理の実行

関連項目 : 『SQL*Plus ユーザーズ・ガイドおよびリファレンス』

メモリー構造とプロセスの概要

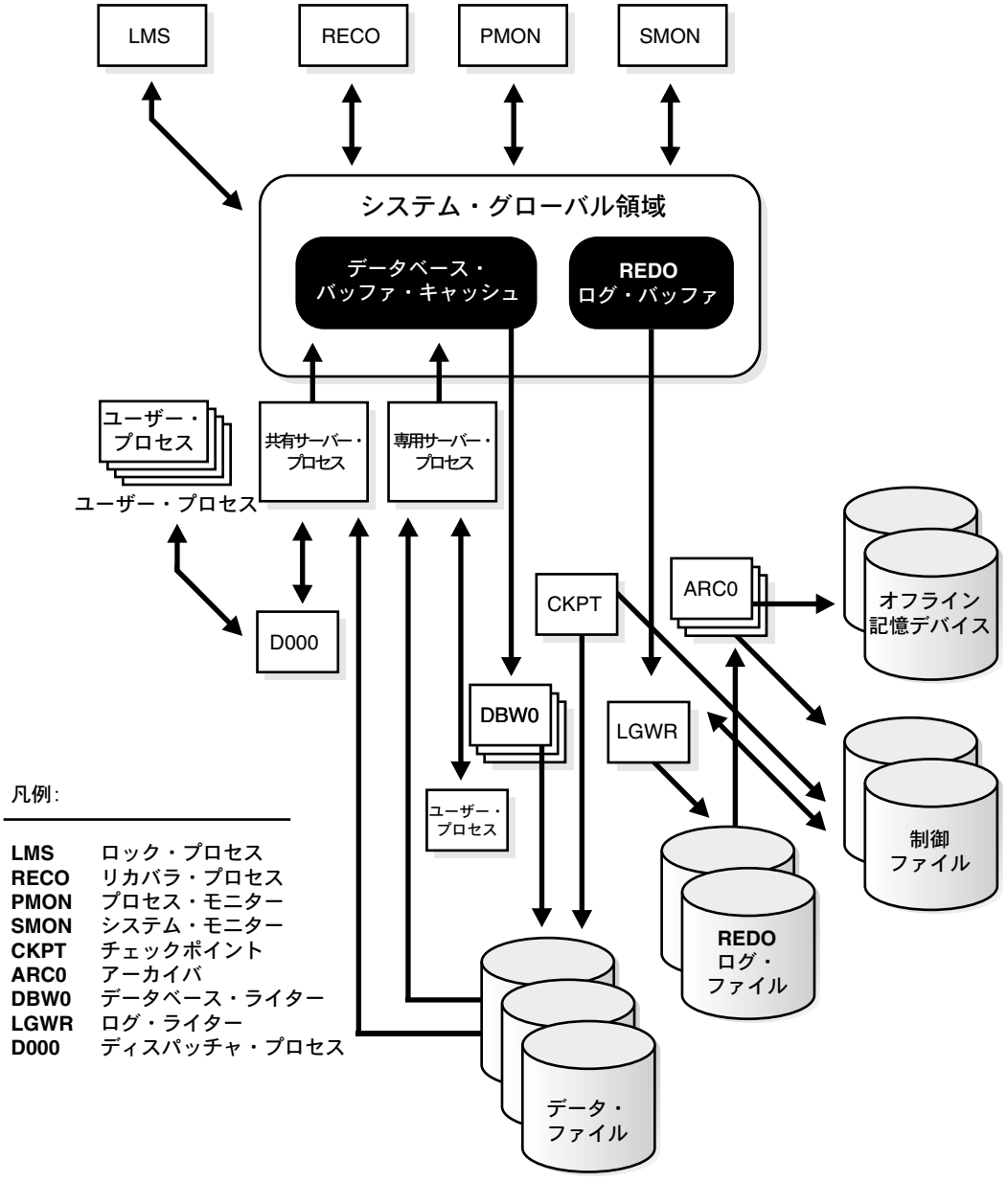
Oracle サーバーは、メモリー構造とプロセスを使用してデータベースの管理やアクセスを行います。すべてのメモリー構造は、データベース・システムを構成するコンピュータのメイン・メモリー内に存在します。**プロセス**とは、これらのコンピュータのメモリー内で実行されるジョブです。

この項で説明するアーキテクチャの機能によって、Oracle サーバーでは次の処理をサポートします。

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

図 1-3 に、Oracle サーバーのメモリーとプロセスの構造の代表的な例を示します。

図 1-3 Oracle のメモリー構造とプロセス



注意： UNIX 環境では、ほとんどの **Oracle プロセス**は、個別のプロセスではなく、1 つのマスター Oracle プロセスの一部です。Windows NT では、すべてのプロセスが少なくとも 1 つの**スレッド**で構成されています。スレッドは、プロセス内で個別に実行されます。スレッドによって、プロセス内での同時操作が可能になるため、プロセスはプログラムの異なる部分を別のプロセッサ上で同時に実行できます。スレッドは、Windows NT 上で順番に実行できる最も基本的なコンポーネントです。このマニュアルなどで記載する、UNIX のプロセスは、Windows NT ではスレッドのことと考えてください。

Oracle インスタンス

Oracle サーバーは、Oracle データベースと Oracle サーバー・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）が割り当てられ、Oracle バックグラウンド・プロセスが起動されます。バックグラウンド・プロセスとメモリー・バッファを組み合せて、Oracle **インスタンス**と呼びます。

Real Application Clusters: 複数インスタンス・システム

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機器へのアクセスを共有できます。Real Application Clusters を使用すると、1 つの物理データベースを共有する複数のインスタンスを実行できるため、このようなアーキテクチャの利点を活用できます。ほとんどのアプリケーションでは、Real Application Clusters によって、複数のマシン上のユーザーが、優れたパフォーマンスで 1 つのデータベースにアクセスできます。

本来、Real Application Clusters は可用性の高いシステムです。典型的な Real Application Clusters 環境である**クラスタ**では、計画停止または計画外停止に対して、継続的にサービスを提供できます。

Oracle サーバーは、メモリー構造とプロセスを使用してデータベースの管理やアクセスを行います。すべてのメモリー構造は、データベース・システムを構成するコンピュータのメイン・メモリー内に存在します。**プロセス**とは、これらのコンピュータのメモリー内で実行されるジョブです。

注意： Real Application Clusters を使用できるのは、Oracle9*i* Enterprise Edition のみです。

関連項目：『Oracle9*i* Real Application Clusters 概要』

メモリー構造

Oracle はメモリー構造を作成して使用し、いくつかのジョブを完了させます。たとえば、実行中のプログラム・コードやユーザー間で共有されるデータを格納するときに、メモリーが使用されます。2 つの基本的なメモリー構造が Oracle に関連付けられます。一方はシステム・グローバル領域で、他方はプログラム・グローバル領域です。この後、それぞれのメモリー領域について詳しく説明します。

システム・グローバル領域

システム・グローバル領域（SGA）は、1 つの Oracle インスタンスに関するデータと制御情報を含む共有メモリー領域です。SGA はインスタンスの起動時に割り当てられ、そのインスタンスの停止時に割当てが解除されます。各インスタンスには、それぞれ専用の SGA があります。

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

SGA に格納される情報は、**データベース・バッファ**、**REDO ログ・バッファ**および**共有プール**など、いくつかのタイプのメモリー構造に分けられます。

関連項目：

- 1-24 ページ「Oracle インスタンス」
- SGA および Oracle バックグラウンド・プロセスの詳細は、1-27 ページの「バックグラウンド・プロセス」を参照してください。

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

SGA の REDO ログ・バッファ REDO ログ・バッファは、**REDO エントリ**を格納します。REDO エントリは、データベースに対して加えられた変更のログです。REDO ログ・バッファ内に格納された REDO エントリは、**オンライン REDO ログ**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使用されます。REDO ログのサイズは固定です。

SGA の共有プール 共有プールには、共有 SQL 領域などの共有メモリー構成メンバーが格納されています。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 領域が必要になります。共有 SQL 領域には、解析ツリーや、対応する文の実行計画などの情報が格納されます。同じ文を発行する複数のアプリケーションが 1 つの共有 SQL 領域を共用するため、他に使用可能な共有メモリーがより多く残ります。

関連項目： 共有 SQL 領域の詳細は、1-11 ページの「SQL 文」を参照してください。

SGA のラージ・プール ラージ・プールとは、Oracle のバックアップおよびリストア操作、I/O **サーバー・プロセス**および**共有サーバー**と **Oracle XA**（複数のデータベースを必要とするトランザクションで使用）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オプションの領域です。

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

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

プログラム・グローバル領域

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

プロセス・アーキテクチャ

プロセスは制御のスレッド、つまり一連の手順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。通常プロセスには、それぞれが実行する独自のプライベート・メモリー領域があります。

Oracle サーバーには、通常 2 つのプロセスがあります。ユーザー・プロセスと Oracle プロセスです。

ユーザー（クライアント）プロセス

ユーザー・プロセスを作成してメンテナンスを実行すると、Pro*C/C++ プログラムなどのアプリケーション・プログラムや、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。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 プロセス

Oracle プロセスは、他のプロセスによって起動され、起動したプロセスにかわって処理を行います。各 Oracle プロセスとその特有の機能について、次に説明します。

サーバー・プロセス Oracle は、接続されたユーザー・プロセスの要求を処理するために、**サーバー・プロセス**を作成します。サーバー・プロセスは、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 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**を読み込みます。

Oracle 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。

ユーザー・プロセスとサーバー・プロセスが別個のプロセスになっているシステムと、1 つのプロセスにまとめられているシステムがあります。システムが共有サーバーを使用している場合、またはユーザー・プロセスとサーバー・プロセスが別々のマシン上で実行されている場合、ユーザー・プロセスとサーバー・プロセスは別のプロセスである必要があります。クライアント / サーバー・システムでは、ユーザー・プロセスとサーバー・プロセスが分離され、それぞれ別々のマシンで実行されます。

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

Oracle インスタンスは、それぞれ複数のバックグラウンド・プロセスを使用します。これらのプロセスの名前は、DBW*n*、LGWR、CKPT、SMON、PMON、ARC*n*、RECO、Jnnn、D*nnn*、LMS および QMN*n* です。

関連項目：

- 1-24 ページ「Oracle インスタンス」
- SGA の詳細は、1-25 ページの「システム・グローバル領域」を参照してください。

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

Oracle は事前書込みロギングを行うため、DBW*n* がトランザクションのコミット時にブロックを書き込む必要はありません。そのかわりに、DBW*n* はバッチ書込みを効率よく実行するように設計されています。通常、DBW*n* が書込みを行うのは、SGA に読み込む必要のある

データが増加し、空き状態のデータベース・バッファがほとんどなくなった場合のみです。LRU のデータから先に、データ・ファイルに書き込みます。DBW*n* は、チェックポイント機能などの他の機能のためにも書込みを行います。

> **関連項目：** コミットの詳細は、1-17 ページの「トランザクションの概要」を参照してください。

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

チェックポイント（CKPT）特定のタイミングで、SGA 内のすべての修正された**データベース・バッファ**が、DBW*n* によってデータ・ファイルに書き込まれます。このイベントは、**チェックポイント**と呼ばれます。チェックポイント・プロセスは、チェックポイント時に DBW*n* に信号を送り、データベースのすべてのデータ・ファイルと制御ファイルに最新のチェックポイントが反映されるように更新します。

システム・モニター（SMON）システム・モニターは、障害インスタンスの再起動時にリカバリを実行します。Real Application Clusters では、あるインスタンスの SMON プロセスが、障害を起こした他のインスタンスに対してインスタンス・リカバリを実行できます。また、SMON は、使用されなくなった**一時セグメント**をクリーン・アップし、リカバリ時にファイル読込みエラーまたはオフライン・エラーのためにスキップされたトランザクションをリカバリします。これらのトランザクションは最終的に、表領域またはファイルがオンライン状態に戻った時点で、SMON によってリカバリされます。また、SMON はディクショナリ管理表領域内で、使用可能エクステントを結合して連続した空き領域を作成し、割当てを容易にします。

プロセス・モニター（PMON）ユーザー・プロセスが障害を起こすと、**プロセス・モニター**がプロセスのリカバリを実行します。PMON は、キャッシュのクリーン・アップとプロセスで使用されていたリソースの解放を受け持ちます。また、ディスパッチャおよびサーバー・プロセスをチェックし、障害がある場合はそれらを再起動します。

アーカイバ（ARC*n*）アーカイバは、ログ・スイッチが発生すると、オンライン REDO ログ・ファイルをアーカイブ記憶域にコピーします。ほとんどのシステムの場合は、1 個の ARC*n* プロセス（ARC0）で十分ですが、動的初期化パラメータ `LOG_ARCHIVE_MAX_PROCESSES` を使用すると最高 10 個の ARC*n* プロセスを指定できます。ワークロードが現行の ARC*n* プロセス数に対して大きくなりすぎると、LGWR は最高 10 個まで別の ARC*n* プロセスを自動的に起動します。ARC*n* がアクティブになるのは、データベースが `ARCHIVELOG` モードで、自動アーカイブが使用可能になっているときのみです。

> **関連項目：** アーカイバの詳細は、1-54 ページの「REDO ログ」を参照してください。

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

ジョブ・キュー・プロセス（Jnnn）ジョブ・キュー・プロセスは、バッチ処理で使用します。また、動的に管理されます。このため、ジョブ・キュー・クライアントは必要に応じて使用するジョブ・キュー・プロセスを増やすことができます。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。

関連項目：

- ジョブ・キューの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

ディスパッチャ（D*nnn*）ディスパッチャは、オプションのバックグラウンド・プロセスです。これが存在するのは、共有サーバー構成を使用している場合のみです。使用中の通信プロトコルごとに、少なくとも 1 つのディスパッチャ・プロセス（D000、. . .、D*nnn*）が作成されます。各ディスパッチャ・プロセスは、接続されたユーザー・プロセスから利用できる共有サーバー・プロセスへの要求をルーティングし、適切なユーザー・プロセスにその応答を戻します。

ロック・マネージャ・サーバー（LMS）ロック・マネージャ・サーバー・プロセス（LMS）は、Real Application Clusters でインスタンス間のロックに使用されます。

関連項目： ロック・プロセスの構成の詳細は、1-24 ページの「Real Application Clusters: 複数インスタンス・システム」を参照してください。

キュー・モニター（QMN*n*） キュー・モニターは、Oracle Advanced Queuing のメッセージ・キューを監視するオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。

プログラム・インタフェースのメカニズム

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

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

通信ソフトウェアと Oracle Net Services

ユーザー・プロセスとサーバー・プロセスがネットワーク内の異なるコンピュータ上にある場合や、ユーザー・プロセスがディスパッチャ・プロセスを介して共有サーバー・プロセスに接続している場合、ユーザー・プロセスとサーバー・プロセスは Oracle Net Services を使用して通信します。**ディスパッチャ**は、オプションのバックグラウンド・プロセスです。これが存在するのは、共有サーバー構成を使用している場合のみです。

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

PC LAN でサポートされているプロトコルから大規模メインフレーム・コンピュータ・システムでサポートされているプロトコルまで、Oracle Net Services では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。

関連項目：『Oracle9*i* Net Services 管理者ガイド』

Oracle の動作例

次の例は、Oracle 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・プロセスが別々のマシン（ネットワークを介して接続）上に存在する Oracle の構成について、具体的に説明します。

1. **インスタンス**は、Oracle 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）上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・マシン**または**クライアント・ワークステーション**）は、**ユーザー・プロセス**でアプリケーションを実行します。クライアント・アプリケーションは、適切な 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**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。DBW*n* プロセスは、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、LGWR プロセスがそのトランザクションをオンライン REDO ログ・ファイルに即時に記録します。
8. トランザクションが成功すると、サーバー・プロセスは、ネットワークを介してアプリケーションにメッセージを送信します。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体を通じて、他のバックグラウンド・プロセスも実行されていて、介入が必要かどうかを監視しています。さらに、データベース・サーバーは、他のユーザーのトランザクションを管理し、同一のデータを要求する異なるトランザクション間の競合を防止します。

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

アプリケーション・アーキテクチャの概要

通常、データベースのアーキテクチャには、クライアント / サーバーと複数層の 2 種類があります。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。

クライアント / サーバー・アーキテクチャ

マルチプロセッシングでは、一連の関連ジョブに複数のプロセッサが使用されます。分散処理では、異なるプロセッサが関連するタスクのサブセットを集中処理することにより、1 つのプロセッサの負荷が低減され、システム全体のパフォーマンスと能力が向上します。

Oracle データベース・システムでは、**クライアント / サーバー・アーキテクチャ**を使用することにより、分散処理を容易に活用できます。このアーキテクチャでは、データベース・システムは 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**およびバックエンドの**サーバー**です。

クライアント

クライアントは、フロントエンドのデータベース・アプリケーションであり、ユーザーはキーボード、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アクセスします。クライアントではデータにアクセスしません。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合せて最適化できます。たとえば、必要なディスク容量の縮小や、グラフィック機能を活用できます。

通常、クライアントは、PC など、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 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 SQL 文や PL/SQL 文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、大容量のディスクや高速プロセッサを搭載できます。

複数層アーキテクチャ : アプリケーション・サーバー

複数層アーキテクチャの構成要素は、次のとおりです。

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

クライアントの識別は、接続のすべての層で保たれます。

分散データベースの概要

分散データベースは、同時に使用される複数のデータベース・サーバーによって管理されるデータベースのネットワークです。通常、単一の論理データベースとはみなされません。分散データベース内にあるすべてのデータベースのデータに、同時にアクセスや変更を実行できます。分散データベースの主な利点は、物理的には別々のデータベースに格納されているデータを論理的に結合し、ネットワーク上のすべてのユーザーに対してアクセスできることです。

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

データベース・リンクは、あるデータベースから別のデータベースへのパスを記述したものです。分散データベースで**グローバル・オブジェクト名**が参照されると、データベース・リンクが暗黙的に使用されます。

分散データベースでは、ネットワークを介した大量のデータに対するアクセスを増加させることができますが、その一方で、データの位置とネットワークをまたがるアクセスの複雑さを隠す必要もあります。分散データベース管理システムではさらに、各ローカル・データベースを非分散データベースと同様に管理できる必要があります。

> **関連項目：** 分散データベースの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

位置の透過性

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

サイト自律性

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

分散データ操作

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

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

```
SELECT * FROM employees@sales;
```

2 フェーズ・コミット

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

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

関連項目： 16-11 ページ「2 フェーズ・コミット・メカニズム」

レプリケーションの概要

レプリケーションとは、分散データベース・システムを構成する複数のデータベースの中に、表などのデータベース・オブジェクトをコピーして維持するプロセスのことです。あるサイトで適用された変更は、ローカルに獲得されて格納されてから、各リモート・ロケーションに転送され、適用されます。Oracle のレプリケーション機能は Oracle サーバーに完全に統合されている機能です。これは別のサーバーではありません。

レプリケーションでは、分散データベース・テクノロジを使用して複数のサイト間でデータが共有されますが、レプリケート・データベースと分散データベースは同じものではありません。分散データベースでは、データを多数の場所で使用できますが、特定の表は 1 箇所に格納されます。たとえば、employees 表を、db2 と db3 データベースも格納されている分散データベース・システムの db1 データベースのみに格納できます。レプリケーションは、同じデータを複数の場所で使用可能にすることを意味します。たとえば、employees 表は、db1、db2 および db3 で使用可能です。

関連項目： 『Oracle9*i* レプリケーション』

表レプリケーション

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

データは、マテリアライズド・ビューを使用してレプリケートできます。

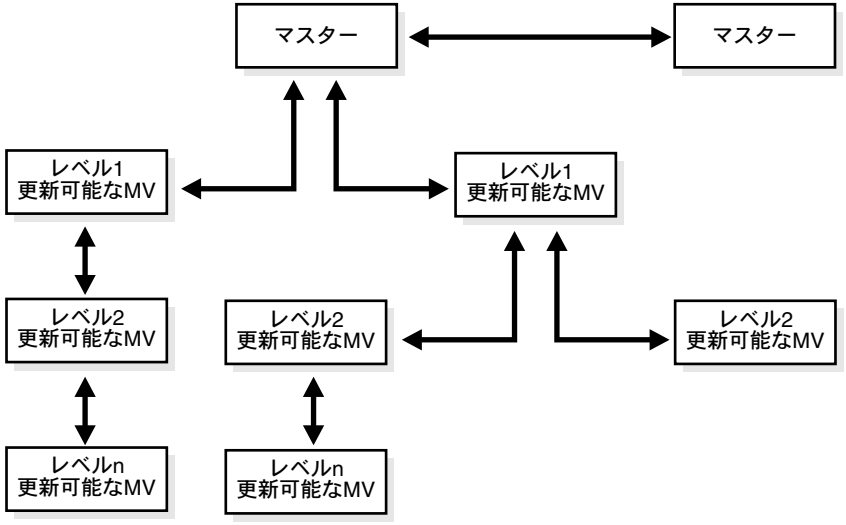
多重化マテリアライズド・ビュー

Oracle では、階層的で、更新可能なマテリアライズド・ビューをサポートしています。複数層レプリケーションによって、分散アプリケーションを設計する上での柔軟性が高められています。多重化マテリアライズド・ビューを使用すると、アプリケーションはレベル間に直接的な接続がないマルチレベルのデータ・サブセットを管理できます。

更新可能なマテリアライズド・ビューによって、行の挿入、更新および削除が可能になるため、ターゲットのマスター表へ変更を反映させることができます。同期および非同期レプリケーションをサポートします。

図 1-4 では、逆ツリー構造図で複数層アーキテクチャの例を示します。変更は、マスター（ルート）によって最も遠いところにあるマテリアライズド・ビューまで、ブランチに沿って上下に伝えられます。

図 1-4 複数層アーキテクチャ



競合解消 Oracle9*i* では、競合解消ルーチンを最上位レベルのマスター・サイトで定義します。また、必要に応じて更新可能なマテリアライズド・ビュー・サイトにもプルできます。このため、多重化マテリアライズド・ビューを持つことが可能になります。既存のシステム定義の競合解消方法もサポートされています。

さらに、ユーザーは独自の競合解消ルーチンを記述できます。ユーザー定義の競合解消方法は、`true` または `false` を戻す PL/SQL ファンクションです。`true` は、その方法がある列グループについて競合するすべての変更を正常に解消できたことを示します。

> **関連項目：** 多重化マテリアライズド・ビューの作成と管理の詳細は、『Oracle9*i* レプリケーション』と『Oracle9*i* SQL リファレンス』を参照してください。

Oracle Streams の概要

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とイベントを共有できます。ストリームにより、指定の情報が指定の宛先にルーティングされます。Oracle Streams には、分散エンタープライズ・アプリケーション、データ・ウェアハウスおよび高可用性ソリューションの作成と操作に必要な機能が用意されています。Oracle Streams のすべての機能は同時に使用できます。ニーズが変更された場合は、Oracle Streams の既存の機能を損なわずに新機能を実装できます。

Oracle Streams を使用すると、ストリームに入れる情報、データベース間でのストリーム・フローまたはルーティング、各データベース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。ストリームの特定の機能を構成することで、特定の要件に対処できます。Oracle Streams は、指定に基づいて DML の変更や DDL の変更など、データベース内でのイベントを自動的に取得して管理できます。また、ストリームにユーザー定義イベントを入れることもできます。その場合、Oracle Streams では、情報が他のデータベースやアプリケーションに自動的に伝播します。さらに、指定に基づいてイベントが接続先データベース側で適用されます。

Oracle Streams を使用すると次の操作ができます。

- データベースでの変更の取得。

  表、スキーマまたはデータベース全体に対する変更を取得するように、バックグラウンドの取得プロセスを構成できます。取得プロセスは REDO ログから変更を取得し、取得した変更をそれぞれフォーマットして論理変更レコード（LCR）を作成します。REDO ログ内で変更が生成されたデータベースは、ソース・データベースと呼ばれます。

- キューへのイベントのエンキュー。Oracle Streams のキューでは、LCR およびユーザー・メッセージという 2 種類のイベントを段階的に処理できます。

  取得プロセスは、LCR イベントを指定されたキューにエンキューします。キューでは、その LCR イベントを同じデータベース内で、または他のデータベースと共有できます。

また、ユーザー・イベントをユーザー・アプリケーションで明示的にエンキューすることもできます。このように明示的にエンキューされたイベントは、LCR またはユーザー・メッセージとして使用できます。

- キュー間でのイベントの伝播。この種のキューは、同じデータベースにあっても異なるデータベースにあってもかまいません。
- キューからのイベントのデキュー。

  バックグラウンド適用プロセスで、キューからイベントをデキューできます。また、ユーザー・アプリケーションでイベントを明示的にデキューすることも可能です。
- データベースでのイベントの適用。

  キュー内のすべてのイベント、または指定したイベントのみを適用するように、適用プロセスを構成できます。また、独自の PL/SQL サブプログラムをコールしてイベントを処理するように構成することもできます。

  LCR イベントが適用され、他のタイプのイベントが処理されるデータベースは、接続先データベースと呼ばれます。構成によっては、ソース・データベースと接続先データベースが同一の場合があります。

その他、Oracle Streams には次の機能があります。

- 取得した LCR 内のタグ
- 有向ネットワーク
- 自動競合検出および解決
- 変換
- 異機種間での情報の共有

関連項目：『Oracle9*i* Streams』

アドバンスト・キューイングの概要

アドバンスト・キューイングは、分散アプリケーションがメッセージを使用して非同期に通信するための基盤を提供します。メッセージは Oracle サーバーによる遅延取出しおよび遅延処理のために、キューに格納されます。これにより、トランザクション処理モニターやメッセージ指向のミドルウェアなどの追加ソフトウェアがなくても、信頼できる効率のよいキューイング・システムが提供されます。

メッセージは、クライアントとサーバー間、および異なるサーバー上のプロセス間でやりとりされます。効率的なメッセージ・システムには、内容ベースのルーティング、サブスクリプションおよび問合せが実装されています。

メッセージ・システムは、次の 2 つのタイプのどちらかに分類できます。

- 同期通信
- 非同期通信

同期通信 同期通信は、要求 / 応答のパラダイムに基づくもので、あるプログラムが別のプログラムに要求を送信し、応答が到着するまで待機するというものです。

この通信モデル（オンライン通信または接続通信とも呼ぶ）は、作業を進める前に応答を受信する必要があるプログラムに適しています。従来のクライアント / サーバーのアーキテクチャは、このモデルに基づいています。このモデルの大きな欠点は、コール側アプリケーションが動作するために、要求を受信するプログラムを使用可能かつ実行中の状態にしておく必要があるという点です。

非同期通信 切断モデルまたは遅延モデルでは、プログラムは非同期に通信し、要求をキューに入れてから次の作業に進みます。

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

ネットワーク、マシンおよびアプリケーションに障害が発生しても正しく動作する遅延実行の場合、要求は永続的に格納され、一度のみ処理される必要があります。このことは、永続キューイングとトランザクションの保護を組み合せることによって実現します。

> **関連項目：** 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』

異機種間サービスの概要

異機種間サービスは、非 Oracle データベース・システムへのアクセスに必要です。非 Oracle データベース・システムとは、次のようなシステムを指します。

- C 言語で記述された PL/SQL プロシージャ（つまり、外部プロシージャ）によってアクセスされるシステム
- SQL（つまり、Oracle Transparent Gateway と一般的な接続）を通してアクセスされるシステム
- プロシージャ（つまり、プロシージャ型ゲートウェイ）によってアクセスされるシステム

異機種間サービスによって、ユーザーは次の操作を実行できます。

- Oracle の SQL 文による非 Oracle システムに格納されているデータの取得
- Oracle プロシージャ・コールを使用した、Oracle 分散環境から非 Oracle システム、サービスまたは Application Program Interface（API）へのアクセス

異機種間サービスは、通常は次の 2 つのいずれかの方法で適用されます。

- Oracle Transparent Gateway を異機種間サービスと連係させて使用すると、Oracle Transparent Gateway が設計上対応している特定のベンダー固有の非 Oracle システムにアクセスできます。たとえば、Solaris 上の Oracle Transparent Gateway for Sybase を使用すると、Solaris プラットフォーム上で実行する Sybase 社のデータベース・システムにアクセスできます。
- 異機種間サービスの一般的な接続である ODBC または OLE DB インタフェースを使用して、非 Oracle データベースにアクセスできます。

関連項目：『Oracle9*i* Heterogeneous Connectivity 管理者ガイド』

データの並行性と一貫性の概要

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

- データを一貫した方法で読み込み、修正します。
- マルチユーザー・システムのデータの**並行性**を最大限に高めます。
- 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。

並行性

マルチユーザー・データベース管理システムの最大の関心は、**並行性**をどのように制御するか、つまり多数のユーザーによる同一データへの同時アクセスをどのように制御するかということです。十分な並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、データ整合性が損なわれる可能性があります。

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

Oracle では、様々なタイプのロックとマルチバージョン一貫性モデルを使用することで、このような問題を解決します。この 2 つの機能については、この項で後述します。これらの機能は、トランザクションの概念に基づいています。並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：** 並行性および一貫性機能の詳細は、1-19 ページの「トランザクションを使用したデータ整合性」を参照してください。

読込み一貫性

Oracle がサポートする読込み一貫性は、次のようなことを保証します。

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

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

読込み一貫性、UNDO レコードおよびトランザクション

マルチバージョンの一貫性モデルを管理するために、表に対する問合せ（読込み）や同時更新（書込み）の実行時に Oracle は読込み一貫性を備えたデータの集合を作成する必要があります。更新が発生すると、それによって変更されたデータの元の値が、データベースの 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。つまり、Oracle 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**読込み一貫性を備えたビュー**を作成します。

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

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

- 読込みユーザーのために生成される、読込み一貫性を備えたビューの開始点を示します。
- データベースの他のトランザクションが、変更データを読込みまたは更新のために参照できるタイミングを制御します。

読取り専用トランザクション

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

複数の表に対して多数の問合せを実行し、更新を行わない場合は、**読取り専用トランザクション**を使用します。トランザクションが読取り専用であると指定した後は、それぞれの問合せの結果が同一時点を基準にした読込み一貫性を持つことがわかっているため、任意の表に対して必要な数の問合せを実行できます。

ロックのメカニズム

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

ロックは、一貫性と整合性を保証するために使用されます。一貫性とは、ユーザーがデータを使用し終えるまで、参照中または変更中のデータが（他のユーザーによって）変更されないことを意味します。整合性とは、データベースのデータと構造が、すべての変更を正しい順序で反映していることを意味します。

ロックによって、データ整合性を保証すると同時に、無制限数のユーザーがデータに同時アクセスできる数を最大化します。

自動ロック

Oracle のロックは自動的に実行されるため、ユーザーの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 のロック・マネージャ（ロック管理機能）は、行レベルで表データを自動的にロックします。行レベルで表データをロックすることで、同一データへの競合が最小限に抑えられます。

ロックを設定する操作のタイプに応じて、異なるタイプの行ロックを保持します。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 つだけ設定できます。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 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。

関連項目： 20-34 ページ「明示的（手動）データ・ロック」

データベースの静止

データベース管理者は、同時に実行するデータベース管理者以外の処理を分離する必要が生じることがあります。つまり、同時に実行するデータベース管理者以外のトランザクション、問合せ、PL/SQL 文を分離します。このような分離を実行する方法の 1 つとして、データベースを停止してから、制限モードで再オープンする方法があります。データベースの静止機能は、別の方法によって分離を実行します。つまり、ユーザーに作業を中断させることなくシステムを静止状態にします。

データベース管理者は SQL 文を使用してデータベースを静止させます。システムが静止状態になると、データベース管理者は、データベース管理者以外の処理を分離して安全に実行できます。

関連項目： 20-16 ページ「データベースの静止」

データベース・セキュリティの概要

Oracle 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ためのセキュリティ機能が用意されています。たとえば、セキュリティ・メカニズムは次のように機能します。

- 権限のないデータベース・アクセスを防止します。
- 権限のないスキーマ・オブジェクトへのアクセスを防止します。
- ユーザーのアクションを監査します。

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

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

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

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

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

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

- 特定のスキーマ・オブジェクトにアクセスできるユーザーと、そのスキーマ・オブジェクトに対して各ユーザーが許可されている特定のタイプのアクション（たとえば、ユーザー `SCOTT` は `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 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

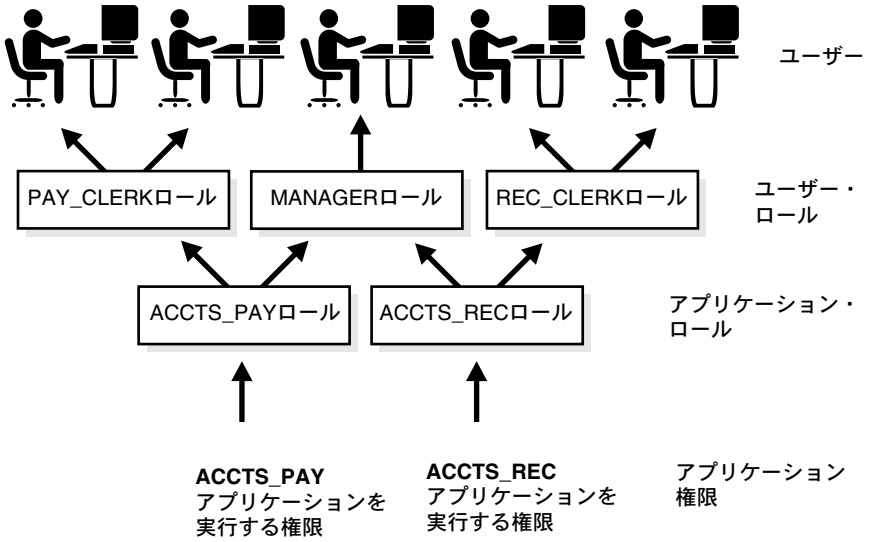
Oracle サーバーでは、**任意アクセス制御**が可能です。任意アクセス制御とは、権限に基づいて情報へのアクセスを制限する方法です。ユーザーがスキーマ・オブジェクトにアクセスするには、そのユーザーは適切な権限を割り当てられている必要があります。適切な権限が付与されているユーザーは、他のユーザーに自分で任意に権限を付与できます。このため、このタイプのセキュリティを**任意**セキュリティと呼びます。

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

- データベース・ユーザーとスキーマ
- 権限
- ロール
- 記憶域の設定と割当て制限
- プロファイルとリソース制限
- ユーザー・アクションの選択的な監査
- ファイングレイン監査

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

図 1-5 Oracle のセキュリティ機能



データベース・ユーザーとスキーマ

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

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

- ユーザーが行える操作（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

Oracle データベースの権限は、**システム権限**と**オブジェクト権限**の 2 つに分類できます。

システム権限 **システム権限**によって、ユーザーは特定のシステム全体に対する操作を行ったり、特定タイプのスキーマ・オブジェクトに対して特定の操作を行えます。たとえば、表領域を作成する権限やデータベース内にある任意の表の行を削除する権限はシステム権限です。多くのシステム権限は非常に強力であるため、それを使用できるのは管理者とアプリケーション開発者だけです。

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

権限の付与 ユーザーは権限が付与されると、データベース内のデータにアクセスしたり変更できます。ユーザーは、次のような 2 通りの形態で権限を付与されます。

- 権限はユーザーに対して明示的に付与できます。たとえば、`employees` 表にレコードを挿入するための権限を、ユーザー `SCOTT` に明示的に付与できます。

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

ロールによって、より簡単でより優れた権限管理が実現できるため、権限は通常特定のユーザーではなくロールに対して付与します。次の項では、ロールとその使用の詳細を説明します。

ロール

Oracle では、ロールを使用することで簡単かつ制御された権限管理を実現できます。**ロール**は、関連する権限のグループに名前を付けたもので、ユーザーまたは他のロールに付与します。

> **関連項目：** ロール・プロパティの詳細は、23-17 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

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

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

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

表領域割当て制限 Oracle は、スキーマ内のオブジェクトに使用可能なディスク領域の容量を制限できます。**割当て制限**（領域制限）は、ユーザーが使用可能な表領域ごとに設定できます。表領域割当て制限によるセキュリティ機能によって、特定のスキーマのオブジェクトが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
- 次の処理について使用可能な CPU 処理時間
  - ユーザーのセッション
  - SQL 文による Oracle への 1 回のコール
- 次の処理について使用可能な論理 I/O の総量
  - ユーザーのセッション
  - SQL 文による Oracle への 1 回のコール
- 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

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。

ユーザー・アクションの選択的な監査

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

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

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

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

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

Oracle では、すべてのタイプの監査について、正常に終了した文の実行、失敗した文の実行またはその両方という選択的な監査ができます。これにより、文を発行しているユーザーがその文を発行するための適切な権限を持っているかどうかにかかわらず、疑わしい文を監査できます。監査の結果は、**監査証跡**と呼ばれる表に記録されます。監査レコードを簡単に検索できるように、事前に定義された監査証跡のビューを使用できます。

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。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 権限を使用したことのみでなく、どのようなデータがアクセスされたかを判断できるように、十分な詳細が必要になります。ファイングレイン監査は、この機能を提供します。

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 は自律型トランザクションを使用してユーザー定義監査イベント・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。

データベース管理の概要

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。

関連項目： データベース管理タスクの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

Enterprise Manager の概要

Enterprise Manager は、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システム管理ツールです。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 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、*i*AS 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
- Enterprise Manager 管理者による表示のカスタマイズ

データベースのバックアップおよびリカバリの概要

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

- 様々なタイプの障害が発生した場合に必要となるデータベースのリカバリ
- どのような状況にも対応する柔軟なリカバリ操作
- バックアップおよびリカバリの処理中のデータの可用性（システムのユーザーが作業を継続できるようにする）

リカバリが重要な理由

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

障害のタイプ

状況によっては、Oracle データベースの操作が中断されます。次の表に、最も一般的な障害のタイプを示します。

障害	説明
ユーザー・エラー	エラー発生前の状態までデータベースをリカバリする必要があります。たとえば、ユーザーが誤って表を削除した場合を考えます。ユーザー・エラーからのリカバリを可能にし、その他の固有のリカバリ要件を満たすために、Oracle は厳密な時間管理によるリカバリ機能を用意しています。たとえば、ユーザーが誤って表を削除した場合、データベースは表が削除された直前の状態にリカバリできます。
文障害	Oracle プログラム内の文の処理中に論理的な障害があると発生します。文障害が発生すると、文による影響がある場合は Oracle によって自動的に取り消され、ユーザーに制御が戻されます。
プロセス障害	接続の異常切断やプロセスの異常終了など、Oracle にアクセスしているユーザー・プロセスでの障害が原因で発生します。バックグラウンド・プロセス PMON は障害を起こしたユーザー・プロセスを自動的に検出し、そのプロセスのうちコミットされていないトランザクションをロールバックし、そのプロセスで使用中だったリソースを解放します。

障害	説明
インスタンス障害	インスタンスで作業を継続できないような問題が発生すると、この障害が発生します。インスタンス障害は、停電などのハードウェア問題、またはオペレーティング・システム障害などのソフトウェア問題が原因で発生することがあります。インスタンス障害が発生した場合、システム・グローバル領域のバッファ内のデータは、データ・ファイルに書き込まれません。 インスタンス障害が発生すると、Oracle は自動的に**インスタンス・リカバリ**を実行します。Real Application Clusters 環境では、あるインスタンスに障害が発生すると、そのインスタンスの REDO を別のインスタンスがリカバリします。シングル・インスタンス・データベース、またはすべてのインスタンスに障害が発生した Real Application Clusters データベースでは、データベースの再起動時に Oracle がすべての REDO を自動的に適用します。
メディア（ディスク）障害	データベースを操作する上で必要なディスク上のファイルに対して読込み / 書込みを実行しているときに、エラーが発生することがあります。一般的な例は、ディスク・ヘッドの障害です。この場合、ディスク・ドライブ上のファイルはすべて失われます。 データ・ファイル、REDO ログ・ファイルおよび制御ファイルなど、様々なファイルが、このタイプのディスク障害によって影響を受ける可能性があります。また、データベース・インスタンスは正常に機能し続けることができないため、システム・グローバル領域のデータベース・バッファ内のデータをデータ・ファイルに書き込むことができません。 ディスク障害が発生した場合は、失われたファイルをリストアしてから**メディア・リカバリ**を実行する必要があります。インスタンス・リカバリとは異なり、メディア・リカバリはユーザーが開始する必要があります。メディア・リカバリは、障害のために失われたメモリー内のコミット済みデータなど、データ・ファイル内の情報がディスク障害の起こる直前の状態に対応するように、リストアされたデータ・ファイルを更新します。

Oracle では、ディスク障害など、発生する可能性のあるすべてのタイプのハードウェア障害から、完全なメディア・リカバリを実行できます。データベースを完全にリカバリするか、または部分的に特定の時点までリカバリできるように、オプションが用意されています。

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

リカバリに使用される構造

Oracle は、**REDO ログ**、**UNDO レコード**、**制御ファイル**および**データベース・バックアップ**など、いくつかの構造を使用して、インスタンス障害やディスク障害から完全にリカバリします。互換性が Oracle9*i* 以上に設定されている場合、UNDO レコードは UNDO 表領域またはロールバック・セグメントに格納できます。

> **関連項目：** UNDO レコードの管理の詳細は、2-16 ページの「自動 UNDO 管理」を参照してください。

REDO ログ **REDO ログ**とはデータ・ファイルに書き込まれていない、メモリー内の変更済みデータベース・データを保護するファイルの集合です。REDO ログは、**オンライン REDO ログ**と**アーカイブ REDO ログ**の 2 つの部分で構成できます。

オンライン REDO ログは、2 つ以上の**オンライン REDO ログ・ファイル**の集合であり、データベースに対して行われたすべての変更が、コミット済みかどうかを問わず記録されます。REDO エントリは、システム・グローバル領域の REDO ログ・バッファに一時的に格納され、バックグラウンド・プロセス LGWR はこの REDO エントリをオンライン REDO ログ・ファイルに順次書き込みます。LGWR は REDO エントリを絶えず書き込みます。つまり、ユーザー・プロセスがトランザクションをコミットするたびに、コミット・レコードを書き込みます。

いっぱいになったオンライン REDO ログ・ファイルは、オプションで**アーカイブ REDO ログ**を作成して再利用の前に手動で、または自動的にアーカイブすることができます。

アーカイブを使用可能または使用禁止にするには、データベースを次のいずれかのモードに設定します。

- `ARCHIVELOG`: いっぱいになったオンライン REDO ログ・ファイルは、循環方式で再利用される前にアーカイブされます。
- `NOARCHIVELOG`: いっぱいになったオンライン REDO ログ・ファイルはアーカイブされません。

`ARCHIVELOG` モードにすると、インスタンスおよびディスク障害からデータベースを完全にリカバリできます。データベースがオープンされ使用可能な場合にも、バックアップを取ることが可能です。ただし、管理操作を追加して、アーカイブ REDO ログをメンテナンスする必要があります。

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

UNDO レコード UNDO レコードは、UNDO 表領域またはロールバック・セグメントに格納できます。Oracle は、コミットされていないトランザクション内で変更があったブロックのビフォア・イメージにアクセスするなど、様々な目的で UNDO データを使用します。データベース・リカバリ中には、Oracle は REDO ログに記録された変更をすべて適用した後、UNDO 情報を使用して、コミットされていないトランザクションをロールバックします。

関連項目：

- UNDO 領域の管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- 起動時の UNDO メソッドの指定の詳細は、5-8 ページの「UNDO 領域の取得と管理」を参照してください。
- UNDO 領域の管理の詳細は、2-16 ページの「自動 UNDO 管理」を参照してください。

制御ファイル データベースの**制御ファイル**には、主としてデータベースのファイル構造と、LGWR によって書き込まれているカレント・ログ順序番号についての情報が保管されます。通常のリカバリ手順では、リカバリ操作を自動的に進行させるために制御ファイル内の情報を使用します。Oracle は、制御ファイルを**多重化**できます。つまり、多数の同じ制御ファイルが同時にメンテナンスされます。

データベースのバックアップ ディスク障害では、1 つ以上のファイルが物理的に破損している可能性があるため、メディア・リカバリでは、データベースの最新のオペレーティング・システム・バックアップから、破損したファイルをリストアする必要があります。

データベース・ファイルのバックアップを作成するには、Recovery Manager を使用する方法（推奨）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。Recovery Manager（RMAN）は、バックアップおよびリカバリの操作を管理する Oracle ユーティリティです。データベース・ファイル（データ・ファイル、制御ファイルおよびアーカイブ REDO ログ・ファイル）のバックアップを作成したり、バックアップを使用してデータベースのリストアやリカバリを行います。

データ・ウェアハウスの概要

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、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。

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、事前定義済みの操作のみです。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 1-6 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 1-6 データ・ウェアハウスのアーキテクチャ

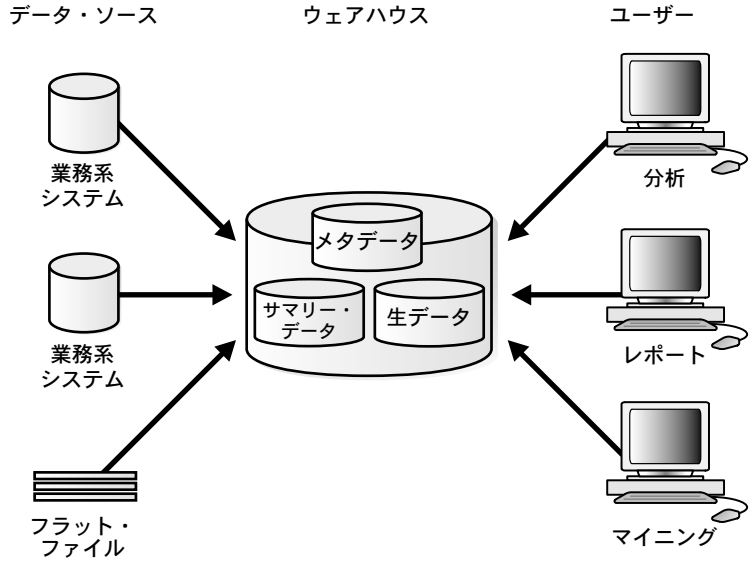


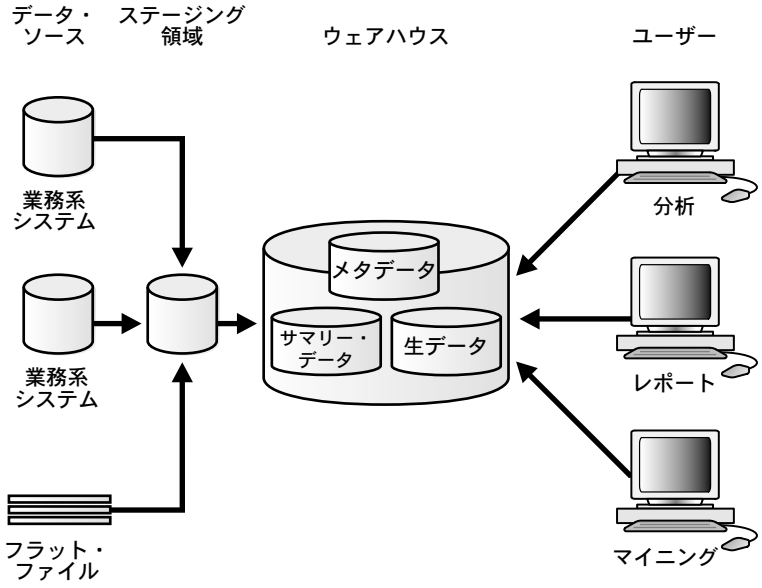
図 1-6 では、従来型 OLTP システムのメタデータと生データの他に、サマリー・データがあります。サマリーでは、時間のかかる処理が事前に行われるため、データ・ウェアハウスではきわめて重要です。たとえば、典型的なデータ・ウェアハウスの問合せでは、8 月の売上などが検索されます。

Oracle 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 1-6 では、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 1-7 に、この典型的なアーキテクチャを示します。

図 1-7 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）

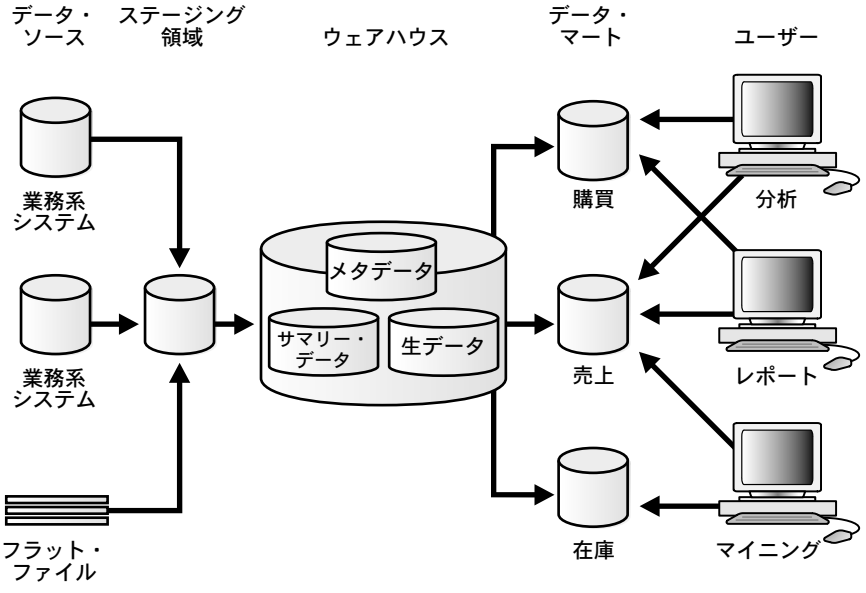


データ・ウェアハウスのアーキテクチャ（ステージング領域およびデータ・マートあり）

図 1-7 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 1-8 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 1-8 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：『Oracle9*i* データ・ウェアハウス・ガイド』

マテリアライズド・ビュー

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

マテリアライズド・ビューをその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改善できます。クエリー・リライトは、データ・ウェアハウス環境で特に役立ちます。

OLAP の概要

Oracle では、ビジネス・インテリジェンスをサポートするために、データベースにオンライン分析処理（OLAP）が統合されます。この統合により、Oracle データベースの管理しやすさ、拡張性および信頼性と SQL へのアクセス可能性を残しつつ、マルチディメンション・データベースの機能が提供されます。

リレーショナル管理システムと Oracle OLAP は、あらゆるレポートおよび分析アプリケーションをサポートするために、補完的な機能を提供します。アプリケーション開発者は、標準レポートと非定型レポートに SQL の OLAP 機能を使用するように選択できます。追加の分析機能が必要な場合は、Oracle OLAP を使用して、マルチディメンション計算、予測、モデル化および what-if による使用例などの機能を提供できます。これらの計算により、開発者は売上およびマーケティング分析、企業の予算および財務分析、需要プランニング・システムなど、洗練された分析およびプランニング・アプリケーションを作成できます。

データは、パフォーマンスとリソースの両面を考慮して、リレーショナル表またはマルチディメンション・オブジェクトに格納できます。格納場所に関係なく、データは OLAP エンジンで Java または SQL を使用して操作できます。リレーショナル・データ・ソースとマルチディメンション・データ・ソースの間で、データをレプリケートする必要はありません。

Oracle OLAP のコンポーネントは、次のとおりです。

- 迅速な計算のために最適化された計算エンジン
- マルチディメンション・データを一時的または永続的に格納するための分析作業領域
- マルチディメンション・データに対して算術、統計、モデル化および他の変換を実行するための OLAP データ操作言語
- マルチディメンション・データを SQL で使用可能にする Oracle OLAP への SQL インタフェース
- ビジネス・インテリジェンス用 Java アプリケーションを開発するための OLAP API
- OLAP API に対してマルチディメンション・データを定義する OLAP メタデータ・リポジトリ

> **関連項目：** Oracle OLAP の詳細は、『Oracle9*i* OLAP User's Guide』を参照してください。

チェンジ・データ・キャプチャの概要

チェンジ・データ・キャプチャにより、Oracle 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、取得され、変更データがアプリケーションで使用可能になります。

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：『Oracle9*i* データ・ウェアハウス・ガイド』

高可用性の概要

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。障害を起こす可能性のあるハードウェアやソフトウェアのコンポーネントには、それと同じタイプの冗長コンポーネントが用意されています。

障害が発生すると、フェイルオーバー・プロセスは、障害を起こしたコンポーネントによって実行されていた処理を、バックアップ・コンポーネントに移動します。このプロセスは、可能な場合は数ミリ秒以内にシステム全体のリソースを再マスター化し、部分的なトランザクションまたは障害を起こしたトランザクションをリカバリして、システムを正常な状態にリストアします。フェイルオーバーがユーザーに対して透過的であるほど、システムの可用性が高いことになります。

Oracle には、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。たとえば、多重 REDO ログ・ファイル、Recovery Manager、ファスト・スタート・リカバリ、LogMiner、フラッシュバック問合せ、パーティション化、透過的アプリケーション・フェイルオーバー、オンライン再編成、Advanced Replication、Oracle Data Guard およびスタンバイ・データベース、Real Application Clusters および Oracle Real Application Clusters Guard などがあります。これらの製品と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。

関連項目：

- 多重 REDO ログ・ファイルの詳細は、1-8 ページの「REDO ログ・ファイル」を参照してください。
- Recovery Manager の詳細は、『Oracle9*i* Recovery Manager ユーザーズ・ガイド』を参照してください。
- ファスト・スタートリカバリの詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- フラッシュバック問合せの詳細は、第 20 章「データの並行性と整合性」を参照してください。
- パーティション化の詳細は、第 11 章「パーティション表とパーティション索引」を参照してください。
- レプリケーションの詳細は、1-35 ページの「レプリケーションの概要」を参照してください。

透過的アプリケーション・フェイルオーバー

透過的アプリケーション・フェイルオーバー（TAF）により、アプリケーション・ユーザーは接続に失敗した場合にデータベースに自動的に再接続できます。アクティブなトランザクションはロールバックされますが、元の接続と同じデータベース接続が異なるノードを使用して新規に確立されます。これは、接続がどのように失敗した場合も同じです。

透過的アプリケーション・フェイルオーバーを使用すると、アプリケーションを処理中のインスタンスが 1 つでも残っているかぎり、クライアントは接続障害に気づきません。データベース管理者は、どのアプリケーションがどのインスタンスで実行されるかを制御し、アプリケーションごとにフェイルオーバー順序を作成します。

透過的アプリケーション・フェイルオーバーの影響を受ける要素

通常のクライアント / サーバーによるデータベース操作中は、クライアントとサーバーが通信できるように、クライアントがデータベースへの接続をメンテナンスします。サーバーが障害を起こすと、接続が切断されます。次回にクライアントが接続の使用を試みると、エラーが発行されます。この時点で、ユーザーはデータベースに再度ログインする必要があります。

ただし、透過的アプリケーション・フェイルオーバーを使用すると、Oracle はデータベースへの新規接続を自動的に取得します。これにより、ユーザーは元の接続が失敗しなかった場合と同様に作業を続けることができます。

アクティブなデータベース接続には、次のように複数の要素が関連付けられています。

- クライアント / サーバーのデータベース接続
- コマンドを実行するユーザーのデータベース・セッション
- フェッチに使用されるオープン・カーソル
- アクティブなトランザクション
- サーバー側プログラム変数

これらの要素の一部は、透過的アプリケーション・フェイルオーバーにより自動的にリストアされます。ただし、透過的アプリケーション・フェイルオーバーを使用可能にするために、他の要素をアプリケーション・コードに埋め込む必要が生じる場合があります。

関連項目：

- 『Oracle9*i* Net Services 管理者ガイド』
- 『Oracle9*i* Real Application Clusters 概要』

オンライン再編成のアーキテクチャ

データベース管理者は、ヒープ構成表のオンライン再編成など、表の定義に対して様々なオンライン操作を実行できます。これにより、ユーザーにはすべてのアクセス権を付与したままで表を再編成できます。

このオンライン・アーキテクチャには、次の機能が用意されています。

- 表の物理属性をオンラインで変更できます。表を新しい位置に移動したり、パーティション化できます。また、表をあるタイプの編成（ヒープ構成など）から別のタイプ（索引構成など）に変換できます。
- 多数の論理属性も変更できます。列名、型、サイズの変更、列の追加、削除またはマージができます。ただし、表の主キーは変更できないという制限があります。
- 索引構成表（IOT）の 2 次索引のオンライン作成およびオンライン再構築ができます。2 次索引では、ブロック・ヒント（物理推測）の効率的な使用がサポートされます。無効な物理推測はオンラインで修正できます。
- 索引をオンラインで作成すると同時に分析できます。論理 ROWID（索引構成表で 2 次索引とマッピング表に使用）の物理推測コンポーネントのオンライン修正も使用できます。
- IOT の 2 次索引に格納された論理 ROWID の物理推測コンポーネントを修正します。これにより、無効な物理推測のオンライン修正が可能になります。

Data Guard の概要

Oracle Data Guard では、破損、人為的エラーおよび災害など、すべての脅威から保護するために、本番データベースのリアルタイム・コピーであるスタンバイ・データベースが最高 9 つまでメンテナンスされます。本番（プライマリ）データベースに障害が発生した場合は、スタンバイ・データベースの 1 つにフェイルオーバーして新規のプライマリ・データベースにすることができます。また、現行のプライマリ・データベースとスタンバイ・データベースの間で本番処理をすばやく簡単に移行（スイッチオーバー）できるため、メンテナンスのための予定停止時間を短縮できます。

注意： スタンバイ・データベースに伝播できないような、ログに記録されない直接の書込みからプライマリ・データベースを保護するために、スタンバイ作成用のデータ・ファイルのバックアップを作成する前に、プライマリ・データベース側で FORCE LOGGING をオンにしてください。スタンバイ・データベースがアクティブになっている間は、データベース（または少なくとも重要な表領域）を FORCE LOGGING モードのままにしてください。

Data Guard 構成

Data Guard 構成は、緩やかに接続されたシステムの集合であり、単一のプライマリ・データベースと最高 9 つのスタンバイ・データベースで構成されます。スタンバイ・データベースには、フィジカル・スタンバイ・データベースとロジカル・スタンバイ・データベースを混在させることができます。Data Guard 構成でのデータベースは、同じデータ・センター内で LAN を使用して接続できます。また、最大限の災害保護のために、WAN を介して地理的に分散させて、Oracle Net Services で接続することもできます。

Data Guard 構成は、データベースを問わず配置できます。これは、その使用がアプリケーションに対して透過的であり、スタンバイ・データベースに適用するためにアプリケーション・コードを変更する必要がないためです。また、Data Guard を使用すると、データの保護レベルとアプリケーションのパフォーマンスが受ける影響のバランスを取るように構成をチューニングし、データ保護、可用性またはパフォーマンスを最大限に発揮するように保護モードを構成できます。

Data Guard のコンポーネント

アプリケーションのトランザクションがプライマリ・データベースに変更を加えると、変更内容は REDO ログにローカルに記録されます。この REDO ログは、**ログ転送サービス**によってスタンバイ・データベースに送信され、**ログ適用サービス**によって適用されます。

フィジカル・スタンバイ・データベースの場合、変更は管理リカバリ・モードで実行中の各フィジカル・スタンバイ・データベースに適用されます。ロジカル・スタンバイ・データベースの場合、変更はアーカイブ REDO ログから再生成された SQL を使用して適用されます。

フィジカル・スタンバイ・データベース フィジカル・スタンバイ・データベースは、物理的にプライマリ・データベースと同一です。プライマリ・データベースがオープンされてアクティブになっている間は、フィジカル・スタンバイ・データベースが（ログを適用して）リカバリを実行中であるか、アクセス・レポート用にオープンされています。フィジカル・スタンバイ・データベースは、本番データベースからフィジカル・スタンバイ・サイトに REDO データが送られている間で、リカバリを実行していないときに、読取り専用で問合せできます。

リカバリ操作では物理 ROWID を使用してブロックごとに変更を適用するため、フィジカル・スタンバイのオン・ディスク・データベース構造は、プライマリ・データベースのブロックベースの構造と同じにする必要があります。索引などのデータベース・スキーマは、同じである必要があります。また、データベースはオープンできません（読取り専用アクセスを除く）。オープンしていると、フィジカル・スタンバイ・データベースに異なる ROWID が使用され、リカバリを継続できなくなります。

ロジカル・スタンバイ・データベース ロジカル・スタンバイ・データベースは、標準的なOracle アーカイブ REDO ログを使用し、格納されている REDO レコードを SQL トランザクションに変換して、オープンされているスタンバイ・データベースに適用します。変更はエンド・ユーザーによるアクセスと並行して適用できますが、再生成される SQL トランザクションを介してメンテナンスされる表では、ロジカル・スタンバイ・データベースのユーザーに読取り専用アクセスが許可されます。データベースがオープンしているので、プライマリ・データベースとは物理的に異なります。データベース表には、対応するプライマリ・データベース表とは異なる索引と物理特性を持たせることができますが、スタンバイ・データ・ソースとしてのロールを果たすには、アプリケーション・アクセスの観点から論理的な一貫性を保つ必要があります。

Data Guard Broker Oracle Data Guard Broker により、複雑な作成およびメンテナンス・タスクが自動化され、監視、アラートおよび制御メカニズムが大幅に拡張されます。Data Guard Broker が使用するバックグラウンド・エージェント・プロセスは Oracle データベース・サーバーと統合され、各 Data Guard サイトに関連付けられており、Data Guard 構成全体に統一された監視および管理のインフラストラクチャを提供します。Data Guard 構成とのやりとりに 2 つのユーザー・インタフェースが用意されています。一方はコマンドライン・インタフェース（DGMGRL）で、他方は Data Guard Manager と呼ばれる GUI です。

Oracle Data Guard Manager は Oracle Enterprise Manager と統合されており、そのウィザードを使用すると構成を簡単に作成、管理および監視できます。この統合により、アラート用イベント・サービス、簡単に設定するための検出サービス、メンテナンスを容易にするジョブ・サービスの提供など、Enterprise Manager の他の機能を利用できます。

関連項目：

- 『Oracle9*i* Data Guard 概要および管理 』
- 『Oracle9*i* Data Guard Broker』

LogMiner の概要

LogMiner は、データベース管理者がログ・ファイルの読取り、分析および解析を SQL から実行できるリレーショナル・ツールです。LogMiner では、オンライン REDO ログとアーカイブ REDO ログの両方を表示できます。Enterprise Manager アプリケーションである LogMiner Viewer により、GUI ベースのインタフェースが追加されます。

LogMiner には REDO ログに格納されているデータにアクセスする機能があり、この機能を使用すると多数のデータベース管理タスクを実行できます。たとえば、次のタスクを実行できます。

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

関連項目：

- 『Oracle9*i* Data Guard 概要および管理 』
- LogMiner の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

Real Application Clusters

本来、Real Application Clusters は可用性の高いシステムです。典型的な Real Application Clusters 環境であるクラスタでは、計画停止または計画外停止に対して、継続的にサービスを提供できます。

Real Application Clusters は、標準的な Oracle 機能の最上位に、より高い可用性レベルを構築します。ファスト・スタート・リカバリやオンライン再編成など、シングル・インスタンスのすべての高可用性機能は、Real Application Clusters にも適用されます。ファスト・スタート・リカバリにより、平均リカバリ時間（MTTR）が大幅に短縮され、オンライン・アプリケーションのパフォーマンスに及ぼす影響が最小限ですみます。オンライン再編成により、予定の停止時間が短縮されます。ユーザーが基礎となるオブジェクトを更新している間も、多数の操作をオンラインで実行できます。Real Application Clusters では、これらの標準的な Oracle 機能がすべて保たれます。

通常のすべての Oracle 機能に加えて、Real Application Clusters はクラスタ化による冗長性を利用して、n ノード・クラスタで n-1 ノードの障害が発生した場合の可用性を実現します。

つまり、クラスタに 1 つでも使用可能なノードがあれば、すべてのユーザーがすべてのデータにアクセスできます。

関連項目：『Oracle9*i* Real Application Clusters 概要』

Real Application Clusters Guard

Oracle Real Application Clusters Guard は、Real Application Clusters の重要なコンポーネントです。Oracle Real Application Clusters Guard の機能は、次のとおりです。

- Oracle インスタンスを停止する障害からの、自動化されたファスト・リカバリとバインドされたリカバリ時間。
- 特定タイプの障害が発生した場合の診断データの自動取得。
- 規程されたプライマリ / セカンダリ構成。Oracle Net Services を介して接続しているクライアントは、クラスタ内の別のノードに接続されていても、プライマリ・ノードに適切にルーティングされます。
- 障害発生後に接続を再確立するときにクライアントに発生する遅延の排除。

Real Application Clusters を実行するデータベース・サーバーは、Oracle データベース、Real Application Clusters ソフトウェア、クライアント要求を受け入れる Oracle Net リスナーで構成されます。これらのソフトウェア・コンポーネントは、クラスタの各ノード上で実行されます。また、ハードウェア、オペレーティング・システムおよびポート固有の Cluster Manager から提供されるサービスを使用します。Cluster Manager は、クラスタ内のノードの状態を監視してレポートし、パックの動作を制御します。

関連項目：

- 『Oracle9*i* Real Application Clusters 概要』
- 『Oracle9*i* Real Application Clusters Real Application Clusters Guard I - Concepts and Administration』

コンテンツ管理の概要

Oracle は、パーソナライズされた豊富なコンテンツを作成、管理し、あらゆるデバイスに配信するための単一プラットフォームを提供します。企業の情報資産には文書、スプレッドシート、マルチメディア、プレゼンテーション、電子メールおよび HTML ファイルなどがありますが、すべてのユーザーがこれらの資産に簡単にアクセスでき、特別なサーバーや無関係なファイル・システムを必要としません。自動検索機能により、貴重なコンテンツを格納場所や使用言語に関係なく検出できます。

Oracle のコンテンツ管理機能を次に示します。

- Oracle Internet File System（9*i*FS）には、データベース内でコンテンツを格納および管理するためのファイル・システムのみでなく、コンテンツ管理アプリケーションを開発するための堅牢な開発プラットフォームも提供します。
- Oracle *inter*Media は、様々なメディア・ファイル（イメージ、オーディオ、ビデオ）からメタデータを抽出し、これらのファイルを Oracle データベース内で操作できるようにします。
- Oracle Text はデータベースに格納されているテキスト・コンテンツに索引を付け、これらの索引に対して洗練された内容ベースの問合せを実行できるようにします。Oracle データベースでは、Microsoft Office、Adobe PDF、HTML および XML 文書など、150 種類以上のファイル・タイプの文書に索引を付け、Oracle Text は 40 以上の言語をサポートしています。
- Oracle Ultra Search は Oracle Text 上に構築され、データベース、ファイル・システムおよび Web サイトに格納されたコンテンツについて、検索可能で統一された索引を提供します。
- Oracle eLocation を使用すると、リージョン・メタデータをコンテンツに追加して空間検索を実行できます。
- Dynamic Services と Syndication Server により、簡単にコンテンツを集約してサブスクライバに配信できます。
- Workspace Manager により、データベース内でコンテンツのバージョンを管理できます。
- Oracle XML Parser などの XML サービスにより、XML コンテンツを解析してレンダリングし、XML ベースのコンテンツを様々なフォーマットや対象ユーザーに合せて調整できます。
- Oracle9*i*AS Portal は、コンテンツをイントラネットやインターネットに配信するプロセスを簡素化し、コンテンツ・プロバイダが公開に使用するフレームワークを提供します。
- Oracle9*i*AS Wireless では、データベースから携帯情報端末にコンテンツをプッシュできます。

Oracle は、コンテンツを管理しやすい状態に保つと同時に、コンテンツを作成して配信するためのアクセスを提供します。Oracle Internet File System などの付属のインタフェースのみでなく、Java、XML および PL/SQL の各 API を介してコンテンツを作成、管理および配信できます。

Oracle Internet File System の概要

通常、重要なビジネス情報が文書、スプレッドシート、電子メールおよび Web ページに大量に格納されています。このデータは、あるユーザーのラップトップや部門別ファイル・サーバーにのみ存在することが多く、組織の他のユーザーからは隠されています。Oracle Internet File System は、すべての情報にアクセスする保護された、スケーラブルなファイル・サービスを作成します。

- Oracle Internet File System は、より多くの機能とインテリジェント機能を企業のファイル管理プロセスにもたらします。ユーザーは、文書に使用されている語句を検索し、チェックイン / チェックアウト機能を使用して、ディスク領域と文書のバージョニングを常に管理状態にしておくことができます。
- ユーザーは特別な研修を受けなくても、標準的な Web ブラウザ、Windows クライアントまたは電子メール・サーバーから、Oracle データベースに格納されているファイルやデータにアクセスできます。Oracle Internet File System は、HTTP、WebDAV、SMB、FTP、NFS、IMAP4 および SMTP など、最もポピュラーな業界標準をすべてサポートしています。
- Oracle Internet File System は、Oracle データベースのマルチレベル・セキュリティ・モデルを使用して、コンテンツを格納および管理するためのセキュアな方法を確立します。また、文書、バージョンおよびフォルダの各レベルでユーザー認証、アクセス権の定義およびアクセス制御を提供し、情報への無許可アクセスを防止します。
- 開発者は、新規文書タイプの迅速なサポートや、会社間での XML ベースのビジネス・ルールの検証と変換など、特定のアプリケーションの用途に合せて 9*i*FS をカスタマイズできます。

第 II 部

データベース構造

第 II 部では、Oracle データベースの基本的な構造上のアーキテクチャについて説明します。たとえば、物理的または論理的な記憶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「データ・ディクショナリ」

2

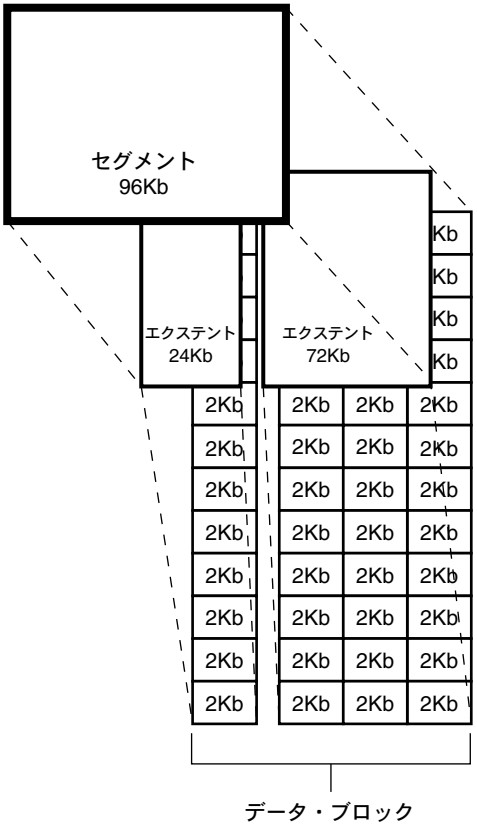
データ・ブロック、エクステントおよびセグメント

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

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

Oracle では、セグメントの領域は 1 エクステント単位で割り当てられます。あるセグメントの既存のエクステントがいっぱいになると、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み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。2-6 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 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。各オペレーティング・システムには、**ブロック・サイズ**があります。Oracle では、データはオペレーティング・システム・ブロックではなく、Oracle データ・ブロックの倍数を単位とする必要があります。

標準のブロック・サイズは、初期化パラメータ DB_BLOCK_SIZE 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 データ・ブロックは、Oracle が使用および割当て可能な最小の格納単位です。

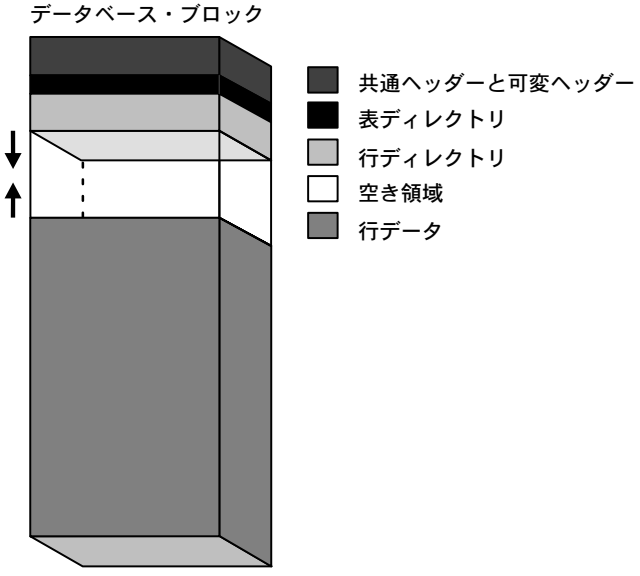
関連項目：

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

データ・ブロックの形式

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



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。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： 2-7 ページ「行連鎖と移行」

空き領域

空き領域は、新しい行の挿入や、追加の領域を必要とする行の更新（後続 NULL が NULL 以外の値に更新される場合など）に割り当てられます。発行された挿入が、あるデータ・ブロックで実際に行われるかどうかは、そのデータ・ブロック内の現在の空き領域の容量と、領域管理パラメータ PCTFREE の値によって決まります。

表やクラスタのデータ・セグメント、または索引の索引セグメントに割り当てられたデータ・ブロックでは、空き領域にトランザクション・エントリを格納することもできます。**トランザクション・エントリ**は、ブロック内の 1 つ以上の行にアクセスする INSERT、UPDATE、DELETE および SELECT...FOR UPDATE の各文ごとに、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み領域は、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：

- 『Oracle9*i* データベース管理者ガイド』
- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* 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』

データ・ブロック内の空き領域の可用性と圧縮

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。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、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 はその行のデータを、そのセグメント用に確保されたデータ・ブロックの**連鎖**（1 つ以上）に格納します。行連鎖は、データ型 LONG または LONG RAW の列が入っている行など、大きな行で最も頻繁に発生します。そのような場合の行連鎖は、避けられません。

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

関連項目：

- 行および行断片の形式の詳細は、10-6 ページの「行の形式とサイズ」を参照してください。
- ROWID の詳細は、10-8 ページの「行断片の ROWID」を参照してください。
- ROWID の詳細は、12-18 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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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。

関連項目： PCTFREE および PCTUSED パラメータの詳細は、付録 B「廃止機能に関する情報」を参照してください。

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 はそのセグメントに新しいエクステントを割り当てます。

エクステントの割当て

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

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

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、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。

Oracle8*i* より前のリリースでは、表領域はすべてディクショナリ管理表領域として作成されていました。ディクショナリ管理表領域では、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ます。Oracle8*i* からは、データ・ディクショナリ表のかわりにビットマップを使用して使用済み領域と空き領域が追跡される、ローカル管理表領域を作成できるようになりました。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。

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

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

関連項目：

- 3-11 ページ「表領域内の領域管理」
- 『Oracle9*i* データベース管理者ガイド』

エクステントの割当て方法

Oracle 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 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 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。

関連項目： ディクショナリ管理の表内でエクステントを割り当てる方法は、付録 B「廃止機能に関する情報」を参照してください。

エクステントの割当て解除

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、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 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。

関連項目：

- 『Oracle9*i* データベース管理者ガイド』
- 『Oracle9*i* SQL リファレンス』

エクステントの割当て解除の詳細は、次のマニュアルを参照してください。

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 はブロックに新しい行を挿入します。表の行をすべて削除しても、データ・ブロックが再生されて表領域内の他のオブジェクトがそれを使用できるようになることはありません。

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

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

> **関連項目：** マテリアライズド・ビューとそのログの詳細は、10-21 ページの「マテリアライズド・ビュー」を参照してください。

索引内のエクステント

索引セグメントに割り当てられたエクステントはすべて、その索引が存在するかぎり、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、エクステントは再生されて、表領域内で他の目的に使用できるようになります。

一時セグメント内のエクステント

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

それに対して複数のソートでは、ソート専用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 回だけ割り当てられ、ソート後には戻されずに残るため、他の複数ソートでそのセグメントを使用できます。

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

> **関連項目：**
>
> - 2-14 ページ「一時セグメントの概要」
> - 10-11 ページ「一時表」

ロールバック・セグメント内のエクステント

Oracle では、データベースのロールバック・セグメント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自動的に割当て解除されます。

関連項目： ロールバック・セグメントの詳細は、付録 B「廃止機能に関する情報」を参照してください。

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、その表のデータ・セグメントを形成する 1 つ以上のエクステントが割り当てられ、各索引には、その索引セグメントを形成する 1 つ以上のエクステントが割り当てられます。

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

- データ・セグメントの概要
- 索引セグメントの概要
- 一時セグメントの概要

データ・セグメントの概要

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

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

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

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

注意： Oracle は、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、『Oracle9*i* レプリケーション』を参照してください。
- CREATE 文と ALTER 文の詳細は、『Oracle9*i* SQL 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 では、問合せの処理中に、SQL 文の解析と実行の中間段階で、一時的な作業領域が必要になることがよくあります。Oracle は、**一時セグメント**と呼ばれるこのディスク領域を自動的に割り当てます。通常、一時セグメントは、ソート操作の作業領域として必要です。ソート操作がメモリー内で実行できる場合、または Oracle が索引を使用して操作を実行できる別の方法を見つけた場合、一時セグメントは作成されません。

一時セグメントを必要とする操作

次の文では、一時セグメントの使用が必要になる場合があります。

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

一部の索引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、2 つの一時セグメントが必要になる可能性があります。アプリケーション

で前述の文を頻繁に発行する場合は、データベース管理者が初期化パラメータ SORT_AREA_SIZE を調整してパフォーマンスを改善してください。

> **関連項目：** SORT_AREA_SIZE およびその他の初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

一時表とその索引のセグメント

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

> **関連項目：** 10-11 ページ「一時表」

一時セグメントが割り当てられる方法

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

問合せ用の一時セグメントの割当て 一時セグメントは、ユーザー・セッション中に必要に応じて、文を発行したユーザーの一時表領域内に割り当てられます。この一時表領域は、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削除されます。

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

> **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：** 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 22 章「データベース・アクセスの制御」を参照してください。

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

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

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

> **関連項目：** 10-11 ページ「一時表」

自動 UNDO 管理

自動 UNDO 管理は、UNDO 表領域ベースです。異なるサイズの多数のロールバック・セグメントを割り当てるかわりに、数個の UNDO 表領域の形式で領域を割当てます。

> **関連項目：** UNDO 表領域の作成方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

自動 UNDO 管理を使用すると、UNDO 保存を明示的に制御できます。システム・パラメータ（UNDO_RETENTION）の使用によって、データベースに保存するコミット済みのロールバック情報の量を指定できます。このパラメータは時計時間（たとえば、30 秒）として指定します。保存制御では、長い問合せを正常に実行できるように、システムを構成できます。

V$UNDOSTAT ビューを使用すると、データベース・システムの監視と構成が可能になるため、UNDO 領域を効率的に使用できるようになります。V$UNDOSTAT は、インスタンスで使用されている UNDO 領域のサイズなど、様々な UNDO とトランザクションの統計値を表示します。

注意： 以前のリリースでは、ロールバック・セグメントを使用して UNDO 領域管理を実行していました。現行バージョンでは、この方法を手動 UNDO 管理モードと呼んでいます。

UNDO モード

UNDO モードを使用すると、手動 UNDO 管理から自動 UNDO 管理へと柔軟に移行できます。データベース・システムは、手動 UNDO 管理モードまたは自動 UNDO 管理モードで実行できます。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ます。また、このモードはどの互換性レベルでもサポートされます。新機能を利用するために Oracle9*i* を実行する必要があり、まだ自動 UNDO 管理モードに切り替える準備が完了していない場合は、手動 UNDO 管理モードを使用してください。

自動 UNDO 管理モードでは、UNDO 領域は UNDO 表領域で管理されます。自動 UNDO 管理モードを使用するために、データベース管理者に必要な操作は、各インスタンスに UNDO 表領域を作成し、UNDO_MANAGEMENT 初期化パラメータを AUTO に設定することのみです。自動 UNDO 管理モードは、Oracle9*i* 以上での互換性レベルでサポートされています。手動 UNDO 管理モードはサポートされていますが、自動 UNDO 管理モードでの実行をお薦めします。

関連項目：

- DDL 文の構文とセマンティクスについては、『Oracle9*i* データベース管理者ガイド』を参照してください。
- 手動 UNDO 管理モードの詳細は、付録 B「廃止機能に関する情報」を参照してください。

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。手動 UNDO 管理モードの場合、このような可能性は MAXEXTENTS 値を小さくしてロールバック・セグメントのサイズを制限すれば制御できます。ただし、実行が長いトランザクションは、SET TRANSACTION USE ROLLBACK SEGMENT 文を使用して、より大きなロールバック・セグメントに明示的に割り当てる必要があります。これは、簡単な方法ではありません。

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。

UNDO 保存制御

読込み一貫性操作に必要なロールバック情報が使用できないため、長時間実行する問合せでは失敗があります。これは、コミット済みの UNDO ブロックがアクティブなトランザクションに上書きされた場合に起ります。

自動 UNDO 管理では、UNDO 領域が再利用できるとき、つまり UNDO 情報をどれくらいの期間保存するのかを明示的に制御できます。データベース管理者は、UNDO_RETENTION パラメータを使用して、保存期間を指定できます。たとえば、UNDO_RETENTION を 30 分に設定すると、システムにあるコミット済みのすべての UNDO 情報は、最低 30 分間保存されます。これによって、実行時間が 30 分以内のすべての問合せでは、通常の環境では、OER エラー「スナップショットが古すぎます」は発生しません。

起動時に UNDO_RETENTION を設定するか、ALTER SYSTEM 文で動的に変更できます。次の例では、保存を 20 分に設定します。

```
ALTER SYSTEM SET UNDO_RETENTION = 1200;
```

UNDO_RETENTION パラメータを設定しない場合、Oracle は、問合せが少なく短いほとんどの OLTP システムで、小さくても適切なデフォルト値を使用します。

一般に保存の値は、UNDO 表領域が対応可能な限界値の近くに設定しないことをお薦めします。UNDO セグメント間で領域の移動が過度に生じることがあるからです。UNDO 領域の 20% のバッファを設定することをお薦めします。

外部ビュー

トランザクションと UNDO 情報を V$TRANSACTION と V$ROLLSTAT で監視します。自動 UNDO 管理の場合、その UNDO セグメントの動作は V$ROLLSTAT の情報に反映されます。

V$UNDOSTAT ビューは、統計データのヒストグラムを表示してシステムの動作状況を示します。UNDO 使用率、トランザクションの並行性、インスタンスで実行される問合せの長さなどの統計値を表示できます。このビューを使用すると、現在のワークロードに必要な UNDO 領域の量を的確に予測できます。このビューは、自動 UNDO 管理モードと手動 UNDO 管理モードの両方で使用可能です。

> **関連項目：** V$UNDOSTAT の使用の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

3

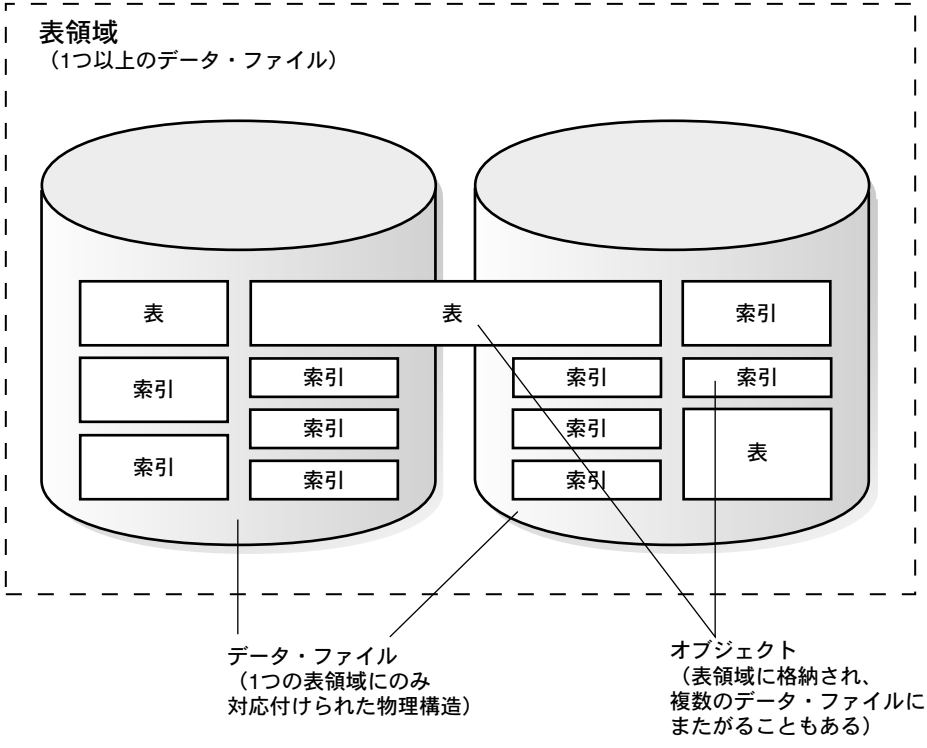
表領域、データ・ファイルおよび制御ファイル

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

図 3-1 データ・ファイルと表領域



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

- Oracle データベースは、データベースのすべてのデータがまとめて格納される、表領域と呼ばれる 1 つ以上の論理記憶単位からなっています。
- Oracle データベース内の各表領域は、データ・ファイルと呼ばれる 1 つ以上のファイルからなっています。データ・ファイルは、Oracle が稼働中のオペレーティング・システムに準拠する物理構造です。

- データベースのデータは、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）。

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 では、内部的に標準ファイルが使用されます。

システムは、必要に応じて次のデータベース構造にファイルを作成し、削除するためのインタフェースを提供します。

- 表領域
- オンライン 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 では、一意のファイルである Oracle 管理ファイルが作成され、不要になると削除されます。

関連項目：『Oracle9*i* データベース管理者ガイド』

データベースへの多くの領域の割当て

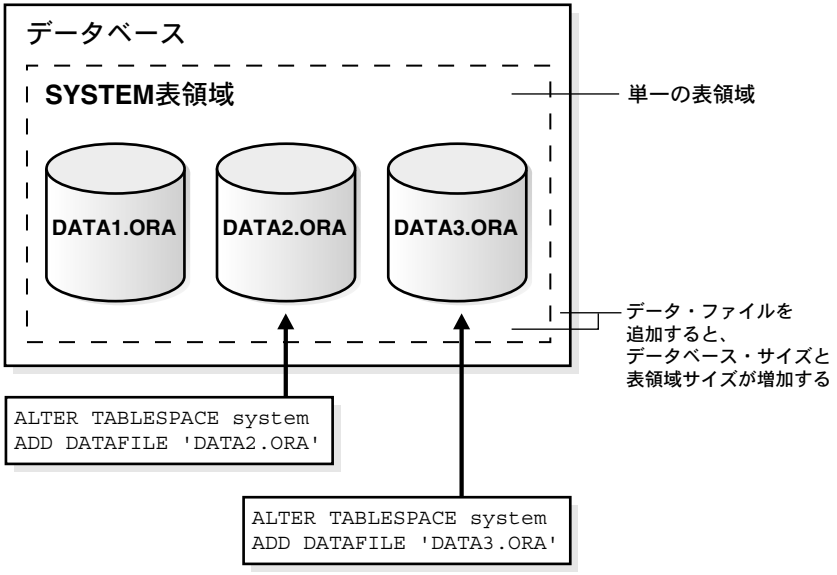
表領域のサイズは、表領域を構成するデータ・ファイルのサイズです。データベースのサイズは、データベースを構成する表領域の合計サイズです。

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。

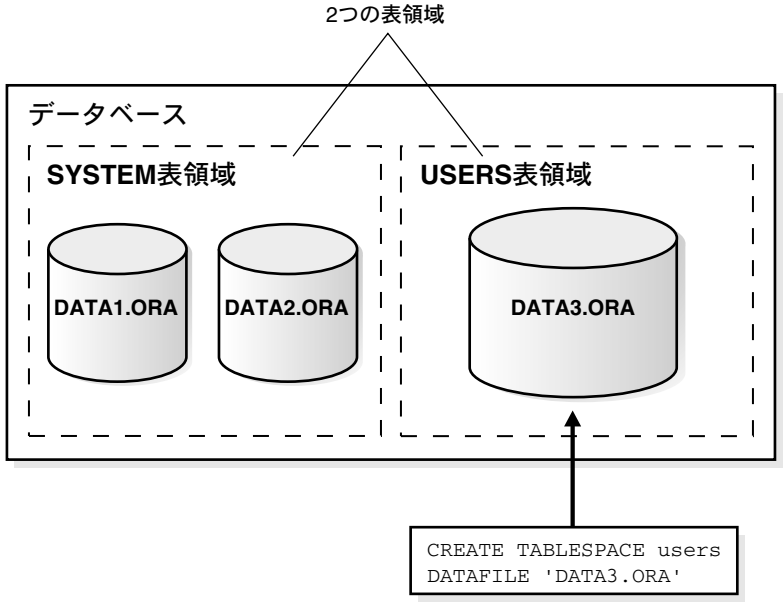
既存の表領域にデータ・ファイルを追加すると、その表領域に割り当てられているディスク領域の容量を増やすことになります。図 3-2 は、このようにして領域を拡大する方法を示しています。

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



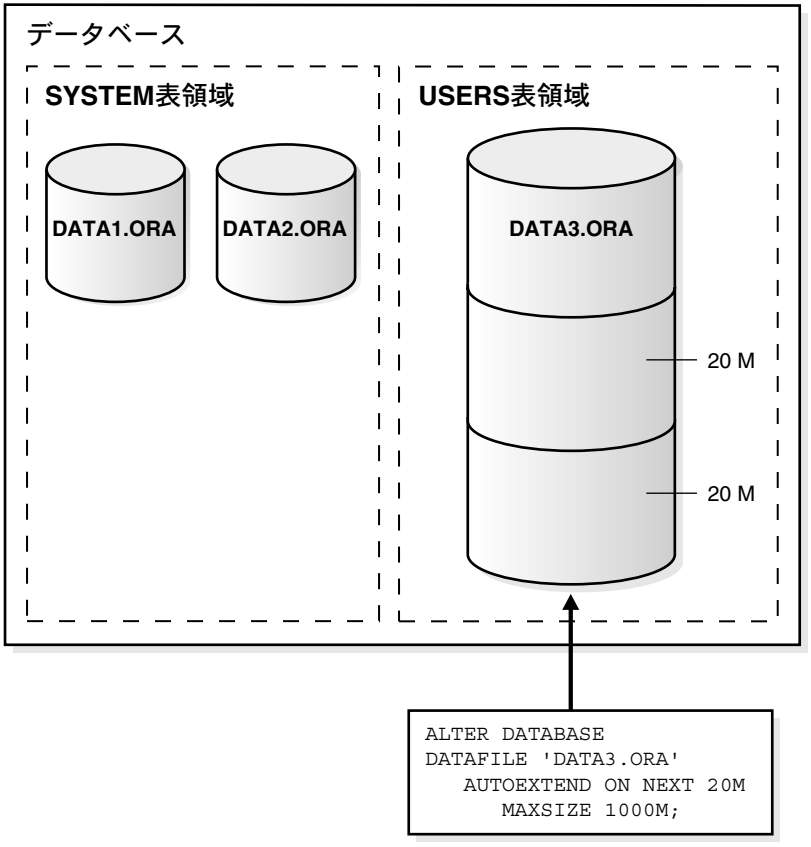
別の方法として、新しい表領域を作成し（少なくとも 1 つのデータ・ファイルを作成）、データベースのサイズを大きくすることもできます。図 3-3 は、この方法を示しています。

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、『Oracle9*i* データベース管理者ガイド』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。

- SYSTEM 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
- ソート操作用の一時表領域
- 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。

SYSTEM 表領域

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

注意： 表領域をローカル管理にした後は、ディクショナリ管理に戻すことはできません。

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、『Oracle9*i* データベース管理者ガイド』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、3-14 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 14 章「SQL、PL/SQL および Java」および第 17 章「トリガー」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。各データベースには、UNDO 表領域が確保されない場合もあります。自動 UNDO 管理モードでは、各 Oracle インスタンスには 1 つの UNDO 表領域のみ割り当てられます。取り消されたデータは、Oracle により自動的に作成、維持される UNDO セグメントを使用して、UNDO 表領域内で処理されます。

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 セグメントおよび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。

注意： 最初の UNDO 表領域を作成し、オンライン化するまではユーザー・トランザクションは実行しないでください。

各 UNDO 表領域は、UNDO ファイルのセットで構成され、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。

UNDO 表領域の作成

データベース管理者は、CREATE UNDO TABLESPACE 文を使用して、UNDO 表領域を個別に作成します。CREATE DATABASE 文を使用して、データベース作成時に同時に作成することもできます。ファイルのセットが、新規に作成された各 UNDO 表領域に割り当てられます。通常の表領域と同様、UNDO 表領域の属性は、ALTER TABLESPACE 文を使用して変更し、DROP TABLESPACE 文を使用して削除できます。

注意： UNDO 表領域をインスタンスが使用している場合またはトランザクションのリカバリに必要なロールバック情報が含まれている場合は、UNDO 表領域は削除できません。

UNDO 表領域の割当て

UNDO 表領域を次のいずれかの方法でインスタンスに割り当てます。

- インスタンス起動時。UNDO 表領域を初期化ファイルに指定することも利用可能な UNDO 表領域をシステムに選択させることもできます。
- インスタンス実行時。アクティブな UNDO 表領域を別の UNDO 表領域で置換するには、ALTER SYSTEM SET UNDO_TABLESPACE を使用します。この方法はほとんど使用されません。

ALTER TABLESPACE 文を使用してデータ・ファイルを UNDO 表領域に追加することで、UNDO 表領域に領域を追加できます。

2 つ以上の UNDO 表領域を確保して、これらを切り替えることができます。Database Resource Manager を使用して、UNDO 表領域のユーザーへの割当て制限を設定します。ロールバック情報の保存周期を指定できます。

> **関連項目：** UNDO 表領域の作成と管理の詳細は、『Oracle9*i* データベース管理者ガイド』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の場合で、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。

デフォルトの一時表領域を削除すると、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。

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、『Oracle9*i* SQL リファレンス』を参照してください。

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、表領域を使用して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込み / 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
- 表領域の削除

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み領域を追跡できます。

- **ローカル管理表領域** : 表領域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。選択した方法を後で変更することはできません。

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。

関連項目： 2-9 ページ「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 はブロックの新しい状態を示すためにビットマップ値を変更します。この変更では、データ・ディクショナリ内の表は更新されないため（表領域割当て制限情報などの特殊ケースを除く）、ロールバック情報は生成されません。

ローカル管理表領域には、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、システムが動的に決定できます。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

  このキーワードにより、セグメント内の空き領域を管理するためにビットマップを使用することが Oracle に伝えられます。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 では、ビットマップを使用することで空き領域の管理がより自動化されるため、この形式の領域管理は、自動セグメント領域管理と呼ばれます。

- MANUAL

  このキーワードを指定することで、セグメント内の空き領域を管理するために空きリストを使用することが Oracle に伝えられます。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。MANUAL がデフォルトです。

関連項目：

- SQL 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- SQL 文の管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- 2-9 ページ「エクステントの数とサイズの決定」
- 一時表領域の詳細は、3-16 ページの「ソート操作用の一時表領域」を参照してください。

ディクショナリ管理表領域

以前のバージョンの Oracle 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。データ・ディクショナリを使用してエクステントが管理される表領域の場合、エクステントが割り当てられたり再利用のために解放されるたびに、Oracle はデータ・ディクショナリ内の適切な表を更新します。また、ディクショナリ表のそれぞれの更新に関するロールバック情報も格納されます。ディクショナリ表とロールバック・セグメントは

データベースの一部であるため、これらが占める領域は、他のすべてのデータと同じ領域管理操作の対象となります。

マルチ・ブロック・サイズ

SYSTEM 表領域のブロック・サイズは、**標準ブロック・サイズ**です。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。

標準ブロック・サイズに加えて、ブロック・サイズを 4 つまで指定できます。初期化ファイルで、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：

- 3-17 ページ「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、『Oracle9*i* 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。SYSTEM 表領域は、データベースのオープン中は常にオンラインになっています。Oracle がデータ・ディクショナリを常に使用できるようになっている必要があるためです。

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。

表領域がオフラインになった場合

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

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

表領域をオンラインにできるのは、その表領域を作成したデータベース内のみに限られます。これは、必要なデータ・ディクショナリ情報がそのデータベースの SYSTEM 表領域内に維持されているためです。オフライン表領域は、Oracle 以外のユーティリティで読み込んだり、編集できません。したがって、オフライン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。

ある種のエラーが発生すると、Oracle は表領域をオンラインからオフラインへ自動的に切り替えます。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・ファイルに書き込もうとして、何度か失敗した場合、Oracle は表領域をオンラインからオフラインへ自動的に切り替えます。オフライン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は、3-16 ページの「ソート操作用の一時表領域」を参照してください。
- データ転送ツールの詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

特殊なプロシージャに対する表領域の使用方法

タイプの異なるデータを分離するために複数の表領域を作成する場合は、各種プロシージャに対する特定の表領域をオフラインで取得します。その他の表領域はオンラインのままなので、表領域内の情報は使用できます。ただし、表領域をオフライン化すると、特殊な状況が発生することがあります。たとえば、2 つの表領域を使用して表データを索引データから分離する場合は、次のことに注意してください。

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

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

読取り専用表領域

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

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

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 / 書込み可能にしたことがないかぎり、バックアップを繰り返す必要はありません。また、データベースをリカバリする必要が生じたときにも、読取り専用表領域は修正されていないため、リカバリする必要がありません。

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、『Oracle9*i* データベース管理者ガイド』を参照してください。
- `ALTER TABLESPACE` 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- リカバリの詳細は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。

ソート操作用の一時表領域

ソート操作用の領域は、ソート専用に指定した**一時表領域**を使用すると、さらに効率的に管理できます。それにより、ソート領域の割当ておよび割当て解除に伴う領域管理操作のシリアル化を効率的に解消できます。

結合、索引作成、順序付け、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。Real Application Clusters を使用することで、パフォーマンスが向上します。

ソート・セグメント

一時表領域は、ソート・セグメントにのみ使用できます。ユーザーが一時セグメント用に指定する表領域とは異なり、一時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

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

> **関連項目：**
>
> - TEMPFILES の詳細は、3-19 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。
> - CREATE TABLESPACE 文、CREATE TEMPORARY TABLESPACE 文および ALTER TABLESPACE 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
> - ソートおよびハッシュ結合用に一時表領域を設定する方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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、同一プラットフォーム上の 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（同一プラットフォーム上の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。

注意： 表領域を移動できるのは、同一キャラクタ・セットを使用して、同一ハードウェア・ベンダーが提供する互換性のあるプラットフォーム上で実行される Oracle データベース間のみです。

表領域の別のデータベースへの移動またはコピー方法

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。

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：

- 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、『Oracle9*i* データベース管理者ガイド』を参照してください。
- インポート / エクスポートの詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。
- 3-7 ページ「SYSTEM 表領域」

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 は、指定されたディスクの容量に、ファイル・ヘッダーに必要なオーバーヘッドの量を加えた容量のファイルを割り当てて、表領域のデータ・ファイルを作成します。データ・ファイルが作成されるとき、そのファイルが Oracle に割り当てられる前に、Oracle が実行されているオペレーティング・システムによってそのファイルから古い情報や認可が消去されます。ファイルが大きい場合には、この処理にかなりの時間がかかることがあります。どのデータベースでも最初の表領域は常に SYSTEM 表領域であるため、データベースの最初のデータ・ファイルは、データベースの作成時に自動的に SYSTEM 表領域に割り当てられます。

> **関連項目：** オペレーティング・システムでデータ・ファイルのファイル・ヘッダーに必要な領域の量については、オペレーティング・システム固有の Oracle マニュアルを参照してください。

データ・ファイルの内容

データ・ファイルが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。しかし、Oracle は、対応付けられた表領域内に将来のデータ・セグメントを保持する領域を確保します。この領域は Oracle が専用で使用します。表領域内のデータが増加すると、Oracle は対応付けられたデータ・ファイル内の空き領域を使用してセグメントにエクステントを割り当てます。

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。ただし、スキーマ・オブジェクトは、特定のデータ・ファイルには対応しません。データ・ファイルは、特定の表領域内の任意のスキーマ・オブジェクトのデータを保持するためのリポジトリです。スキーマ・オブジェクトに関連するデータのための領域は、表領域の 1 つ以上のデータ・ファイル内に割り当てられます。このため、1 つのスキーマ・オブジェクトが 1 つ以上のデータ・ファイルにまたがる場合があります。表の**ストライプ化**（データが複数のディスク上に分散される）を使用しないかぎり、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、『Oracle9*i* データベース管理者ガイド』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。
- 一時ファイルは改名できません。
- ALTER DATABASE 文では一時ファイルを作成できません。
- 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限りません。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。制御ファイルはデータベースの使用時に Oracle によって絶えず更新されるため、データベースがオープンされている間は書込み可能になっている必要があります。なんらかの理由で制御ファイルにアクセスできない場合、データベースは正常に機能できなくなります。

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。

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 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・ファイルとオンライン REDO ログ・ファイルの名前と位置
- 表領域情報
- データ・ファイルのオフライン範囲
- ログ履歴
- アーカイブ・ログ情報
- バックアップ・セットとバックアップ・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・ファイルやオンライン REDO ログ・ファイルが追加、改名または削除されるたびに、制御ファイルが更新され、この物理構造の変化が反映されます。これらの変更は、次の目的のために記録されます。

- データベースの起動時に、オープンするデータ・ファイルとオンライン REDO ログ・ファイルを Oracle が識別できるようにします。
- データベースのリカバリが必要な場合に必要なファイルや使用可能なファイルを Oracle が識別できるようにします。

したがって、データベースの物理構造を変更した後は（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、オンライン REDO ログ内のチェックポイント位置に関する情報を、制御ファイルに 3 秒ごとに記録します。この情報は、データベースのリカバリ時に使用され、オンライン REDO ログ・グループの特定のポイントより前に記録されている REDO エントリはデータベースのリカバリには不要である（すでにデータ・ファイルに書き込まれている）ことを Oracle に伝えます。

関連項目：

- 『Oracle9*i* Recovery Manager ユーザーズ・ガイド』
- 『Oracle9*i* ユーザー管理バックアップおよびリカバリ・ガイド』

データベースの制御ファイルのバックアップを作成する方法は、次のマニュアルを参照してください。

多重制御ファイル

オンライン REDO ログ・ファイルと同様に、同一の制御ファイルを同時に複数オープンして、同じデータベースの情報を書き込めます。

1 つのデータベースについての複数の制御ファイルを異なるディスクに格納することによって、単一地点の障害から制御ファイルを保護できます。制御ファイルが格納されているディスクがクラッシュした場合に、Oracle がこの破損した制御ファイルにアクセスしようとすると、現行インスタンスがエラーになります。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簡単に再起動できます。

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。したがって、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

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 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、3-7 ページの「SYSTEM 表領域」を参照してください。

データ・ディクショナリの構造

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 だけです。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 ユーザー SYS が所有しています。したがって、Oracle ユーザーは、SYS スキーマに含まれている行またはスキーマ・オブジェクトを決して変更（UPDATE、DELETE または INSERT）しないでください。そのような操作により、データ整合性が損なわれることがあります。セキュリティ管理者は、このアカウントを厳しく管理する必要があります。

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 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 は、データ定義言語（DDL）文が発行されるたびにデータ・ディクショナリを変更します。
- Oracle ユーザーは、データベースについての情報の読取り専用リファレンスとしてデータ・ディクショナリを使用できます。

Oracle によるデータ・ディクショナリの使用方法

データ・ディクショナリの実表内のデータは、Oracle を機能させるために必要です。したがって、データ・ディクショナリ情報を書き込んだり変更するのは、Oracle のみにする必要があります。データベースがアップグレードまたはダウングレードされたときは、Oracle により、データ・ディクショナリ表を変更するスクリプトが提供されます。

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

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

データ・ディクショナリ・ビューのパブリック・シノニム

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

高速アクセスのためのデータ・ディクショナリのキャッシュ

ユーザー・アクセスの妥当性チェックやスキーマ・オブジェクト状態の検証のために、Oracle はデータベース操作中に絶えずデータ・ディクショナリにアクセスするため、データ・ディクショナリ情報の大部分は**ディクショナリ・キャッシュ**内の SGA 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 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。

Oracle によるデータ・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たします。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 ユーザーがアクセスできるビューもいくつかありますが、その他のビューはデータベース管理者のみが使用するように設計されています。

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

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

表 4-1 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、『Oracle9*i* データベース・リファレンス』を参照してください。

接頭辞が USER のビュー

通常のデータベース・ユーザー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、『Oracle9*i* データベース管理者ガイド』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 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、『Oracle9*i* SQL リファレンス』を参照してください。

動的パフォーマンス表

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

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

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

> **関連項目：** 動的パフォーマンス・ビューのシノニムと列の完全なリストは、『Oracle9*i* データベース・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、『Oracle9*i* 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。

第 III 部

Oracle インスタンス

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

第 III 部には、次の章が含まれています。

5

データベースとインスタンスの起動と停止

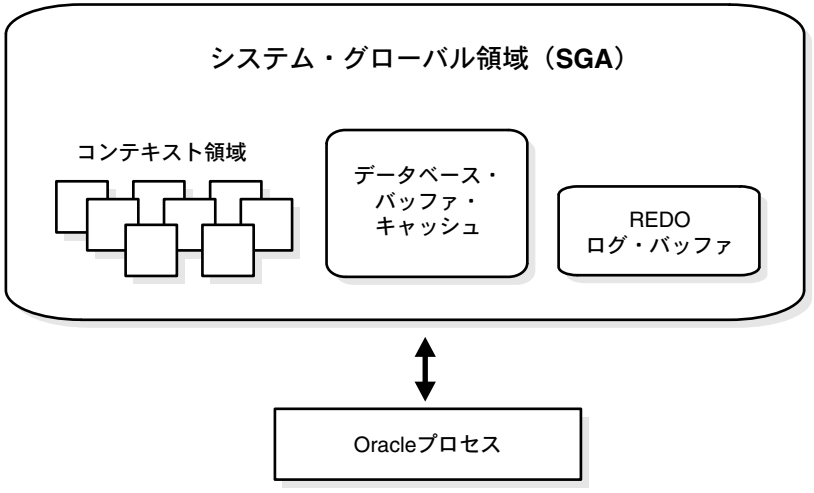
この章では、Oracle のインスタンスおよびデータベースの起動と停止に関連する手順を説明します。この章の内容は、次のとおりです。

Oracle インスタンスの概要

稼働中のすべての Oracle データベースは、Oracle インスタンスに対応付けられます。データベース・サーバー上で（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 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 プロセスが開始されます。この SGA と Oracle 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 5-1 に Oracle インスタンスを示します。

図 5-1 Oracle インスタンス



関連項目：

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

インスタンスとデータベース

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

複数のインスタンスを同時に同じコンピュータ上で実行し、それぞれ専用の物理データベースにアクセスさせることができます。クラスタ化された大規模パラレル・システム（MPS）では、Real Application Clusters により、1 つのデータベースを複数のインスタンスがマウントできます。

データベース管理者のみが、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情報にアクセスできません。

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

関連項目： 詳細は、『Oracle9*i* Real Application Clusters 概要』を参照してください。

管理者権限での接続

データベースの起動と停止は強力な管理オプションであり、管理者権限を使用して Oracle に接続するユーザーのみがこれを行うことができます。ユーザーの管理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、そのユーザーは SYS が所有するスキーマ内に置かれます。SYSOPER 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 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 22 章「データベース・アクセスの制御」を参照してください。

初期化パラメータ・ファイル

インスタンスを起動するには、Oracle は**初期化パラメータ・ファイル**を読み込む必要があります。初期化パラメータ・ファイルとは、そのインスタンスとデータベースの構成パラメータのリストを含むファイルです。これらのパラメータに特定の値を設定して、Oracle インスタンスのメモリーとプロセスの設定の多くを初期化します。ほとんどの初期化パラメータは、次のグループのいずれかに属しています。

- 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。

主に、初期化パラメータは Oracle に次のことを伝えます。

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

> **関連項目：** 初期化パラメータ・ファイルの例については、『Oracle9*i* データベース管理者ガイド』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することもできます。サーバー・パラメータ・ファイル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影響します。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・ファイルを使用すると、データベースの停止時と起動時で変更内容が一致するように、ディスク上のパラメータを更新できます。

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、『Oracle9*i* データベース管理者ガイド』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、『Oracle9*i* データベース・リファレンス』を参照してください。
- SGA に影響するパラメータの詳細は、7-5 ページの「動的 SGA」を参照してください。

インスタンスとデータベースの起動

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

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 Enterprise Manager 管理者ガイド』

インスタンスの起動方法

Oracle は、インスタンスを起動するときに、初期化パラメータ・ファイルを読み込んで初期化パラメータの値を判別してから、SGA（データベース情報のために使用される共有メモリー領域）を割り当てて、バックグラウンド・プロセスを作成します。この時点では、これらのメモリー構造とプロセスにデータベースは対応付けられていません。

関連項目：

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

インスタンスの起動の制限モード

インスタンスを制限モードで起動したり、既存のインスタンスを制限モードに変更できます。これにより、接続は `RESTRICTED SESSION` システム権限を付与されているユーザーのみに制限されます。

異常な状況での強制起動

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、直前のインスタンスの残りの Oracle プロセスをすべて終了してから、新しいインスタンスを起動する必要があります。

データベースのマウント方法

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したパラメータ・ファイルの CONTROL_FILES 初期化パラメータに指定されます。その後で Oracle は制御ファイルを読み込み、データベースのデータ・ファイルと REDO ログ・ファイルの名前を取得します。

マウント直後のデータベースはまだクローズされているため、データベース管理者のみがそのデータベースにアクセスできます。データベース管理者は、特定のメンテナンス操作を完了するまでの間、データベースをクローズしたままにしておくことができます。ただし、データベースに対して通常の操作を行うことはまだできません。

Real Application Clusters を使用したデータベースのマウント方法

注意： この項で説明している機能は、Oracle9*i* Enterprise Edition と Real Application Clusters を購入した場合にのみ使用できます。

Oracle で同時に複数のインスタンスが同じデータベースをマウントできる場合、データベース管理者は初期化パラメータ CLUSTER_DATABASE を使用して、データベースを複数インスタンスに使用可能にすることができます。

CLUSTER_DATABASE パラメータのデフォルト値は false です。Real Application Clusters をサポートしていない Oracle のバージョンでは、CLUSTER_DATABASE パラメータに設定できる値は false のみです。

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：

- 『Oracle9*i* Real Application Clusters 概要』
- 『Oracle9*i* Real Application Clusters セットアップおよび構成』
- 『Oracle9*i* Real Application Clusters 管理』
- 『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』

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、次のマニュアルを参照してください。

スタンバイ・データベースのマウント方法

スタンバイ・データベースは、プライマリ・データベースの複製をメンテナンスし、障害発生時も引き続き使用できるようにします。

スタンバイ・データベースは、常にリカバリ・モードになっています。スタンバイ・データベースをメンテナンスするには、`ALTER DATABASE` 文を使用してスタンバイ・モードでマウントし、プライマリ・データベースによって生成されるアーカイブ REDO ログを適用する必要があります。

スタンバイ・データベースは、読取り専用モードでオープンして、一時的なレポート用データベースとして使用できます。スタンバイ・データベースを読取り / 書込みモードでオープンすることはできません。

関連項目：

- 『Oracle9*i* Data Guard 概要および管理』
- スタンバイ・データベースを読取り専用モードでオープンする方法については、5-9 ページの「データベースの読取り専用モードでのオープン」を参照してください。

クローン・データベースのマウント方法

クローン・データベースは、表領域の Point-in-Time リカバリに使用できる、データベースの特殊コピーです。表領域の Point-in-Time リカバリを実行する場合は、クローン・データベースをマウントし、表領域を目的の時点までリカバリします。次に、クローンからプライマリ・データベースにメタデータをエクスポートし、リカバリされた表領域からデータ・ファイルをコピーします。

関連項目：

クローン・データベースと表領域の Point-in-Time リカバリの詳細は、次のマニュアルを参照してください。

- 『Oracle9*i* Recovery Manager ユーザーズ・ガイド』
- 『Oracle9*i* ユーザー管理バックアップおよびリカバリ・ガイド』

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。

データベースをオープンすると、Oracle はオンライン・データ・ファイルとオンライン REDO ログ・ファイルをオープンします。データベースを前回停止したときに表領域がオフラインであった場合は、データベースを再オープンしたとき、その表領域と対応するデータ・ファイルはオフラインのままです。

データベースをオープンしようとしたときに、データ・ファイルまたは REDO ログ・ファイルのうち存在していないファイルがあると、Oracle からエラーが戻されます。そのデータベースをオープンするには、破損または欠落したファイルのバックアップからリカバリする必要があります。

> **関連項目：** オフライン表領域をオープンする方法については、3-14 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。

インスタンス・リカバリ

データベース管理者がインスタンスを終了させたため、または停電などのためにデータベースが異常にクローズされた場合、Oracle はデータベースの再オープン時にリカバリを自動的に実行します。

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、インスタンスは 1 つ以上の UNDO 表領域またはロールバック・セグメントを取得しようとします。自動 UNDO 管理モードと手動 UNDO 管理モードのどちらでインスタンスを起動するかを、初期化パラメータ `UNDO_MANAGEMENT` により決定します。サポートされる値は、`AUTO` と `MANUAL` です。`AUTO` の場合、インスタンスは自動 UNDO 管理モードで起動されます。デフォルト値は、`MANUAL` です。

- UNDO 表領域を使用する場合は、自動 UNDO 管理モードを使用します。このモードを使用することをお薦めします。
- UNDO 領域管理のロールバック・セグメントを使用する場合は、手動 UNDO 管理モードを使用します。

> **関連項目：** UNDO 領域の管理の詳細は、2-16 ページの「自動 UNDO 管理」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、『Oracle9*i* データベース管理者ガイド』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・ファイルなど、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を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、『Oracle9*i* データベース管理者ガイド』を参照してください。

データベースとインスタンスの停止

データベースとそれに対応付けられたインスタンスを停止するには、次の 3 つの操作を行います。

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 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 Enterprise Manager 管理者ガイド』

データベースのクローズ

データベースをクローズすると、SGA にあるすべてのデータベース・データとリカバリのためのデータが、それぞれデータ・ファイルと REDO ログ・ファイルに書き込まれます。次に、Oracle がすべてのオンライン・データ・ファイルとオンライン REDO ログ・ファイルをクローズします。（オフライン表領域のオフライン・データ・ファイルは、すでにクローズされています。オフラインになっていた表領域とそのデータ・ファイルは、この後でデータベースを再オープンしたとき、それぞれオフラインでクローズされたままになっています。）この時点でデータベースはクローズされているため、通常の操作は実行できません。データベースがクローズされても、まだマウントされていれば、制御ファイルはオープンされたままになります。

インスタンスの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。この後、データベースを再オープンすると、Oracle がリカバリを自動的に実行します。

注意： データベースがオープンされているときにシステム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 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、データベースの制御ファイルはクローズされます。

インスタンスの停止

データベース停止の最後の操作は、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、最初に、残っている前のインスタンスを削除してから新しいインスタンスを起動するか、Enterprise Manager で `SHUTDOWN ABORT` 文を発行することにより、新しいインスタンスの起動を強制実行できます。

> **関連項目：** インスタンスとデータベースの起動および停止の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

6

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 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 データベース・システム環境に適用されます。

この章の内容は、次のとおりです。

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 つの部分に分割されています。フロントエンド、すなわち**クライアント**部分およびバックエンド、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 ソフトウェアが実行され、Oracle データベースへの同時実行の共有データ・アクセスに必要な機能が処理されます。

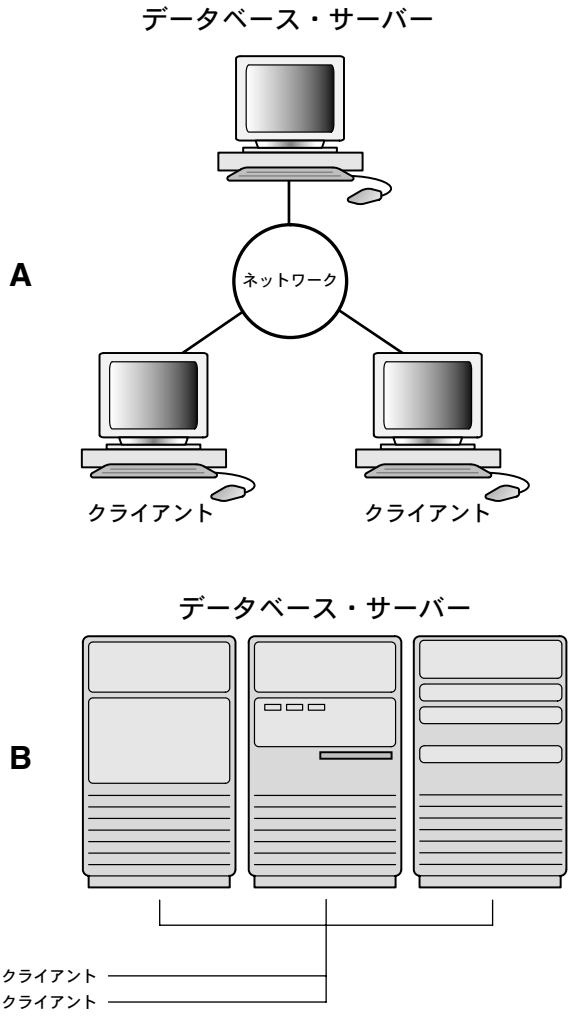
クライアント・アプリケーションと Oracle を同じコンピュータで実行することもできますが、クライアント部分とサーバー部分を別々のコンピュータで実行し、それらのコンピュータをネットワークで接続するほうがさらに効率的です。次の各項では、Oracle クライアント / サーバー・アーキテクチャの様々な形態について説明します。

分散処理とは、複数のプロセッサを使用して個々の作業を処理することです。図 6-1 に、Oracle データベース・システムでの分散処理の例を示します。

- 図の A では、クライアントとサーバーを別々のコンピュータに配置し、各コンピュータをネットワークで接続しています。Oracle データベース・システムのサーバーとクライアントは、Oracle のネットワーク・インタフェースである 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 を別々のプロセッサで実行しています。

注意： この章の内容は、1 つのサーバー上に 1 つのデータベースがある環境に適用されます。**分散データベース**では、あるサーバー（Oracle）から別のサーバー上のデータベースへのアクセスが必要となる場合があります。

図 6-1 クライアント / サーバー・アーキテクチャと分散処理



分散処理環境での Oracle クライアント / サーバー・アーキテクチャには、次のような利点があります。

- クライアント・アプリケーションではデータ処理は実行されません。そのかわり、クライアント・アプリケーションは、ユーザーに入力を要求し、必要なデータをサーバーに要求し、そのデータを分析してクライアント・ワークステーションまたは端末の表示機能（グラフィックスやスプレッドシート）を使用して表示します。

- クライアント・アプリケーションはデータの物理的な位置に依存しません。データを他のデータベース・サーバーに移動したり分散しても、アプリケーションはほとんど、またはまったく変更することなく、機能を続行できます。
- Oracle は、基礎となるオペレーティング・システムのマルチタスキング機能と共有メモリ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並行性、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 は、システムの拡大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 をミニコンピュータやメインフレームなどに移動して、より大規模なシステムのパフォーマンス上の利点を活用（**垂直スケール**）できます。どちらの場合も、Oracle にはシステム間での移植性があるため、すべてのデータとアプリケーションをほとんど、またはまったく変更することなく維持できます。
- ネットワーク化された環境の場合、共有データは、システムのすべてのコンピュータに格納されるのではなく、サーバーに格納されます。このため、同時アクセスの管理が簡単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取りされるのは要求と結果のみであるため、ネットワーク通信量は最小限ですみます。

関連項目：

- Oracle Net Services の詳細は、6-7 ページの「Oracle Net Services」を参照してください。
- 分散データベースでのクライアントとサーバーの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

複数層アーキテクチャ

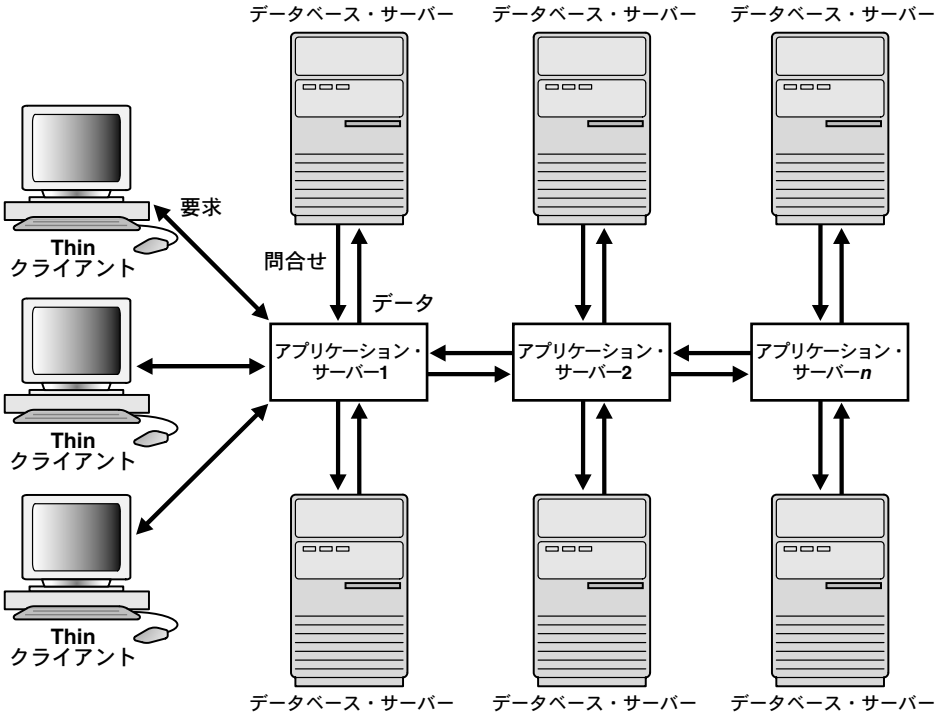
複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントとデータベース・サーバー間のインタフェースとして機能します。このアーキテクチャは、インターネットの普及により重要度が増しています。

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 6-2 は、複数層アーキテクチャの例を示しています。

図 6-2 複数層アーキテクチャ環境の例



クライアント

クライアントは、データベース・サーバー上で実行される操作について要求を出します。このクライアントは、Web ブラウザでも他のエンド・ユーザー・プロセスでもかまいません。複数層アーキテクチャでは、クライアントは 1 つ以上のアプリケーション・サーバーを介してデータベース・サーバーに接続します。

アプリケーション・サーバー

アプリケーション・サーバーは、クライアントにデータ・アクセスを提供します。また、クライアントと、さらに高度なセキュリティ・レベルを提供する 1 つ以上のデータベース・サーバーとの間のインタフェースとして機能します。さらに、クライアントのために一部の問合せ処理も実行するため、データベース・サーバーの負荷がある程度軽減されます。

アプリケーション・サーバーは、クライアントのためにデータベース・サーバー上で操作を実行するとき、そのクライアントの識別を想定します。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：** 複数層環境におけるセキュリティの注意点の詳細は、22-10 ページの「複数層の認証と認可」を参照してください。

Oracle Net Services

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 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。

次の項では、一般的なネットワーク構成におけるいくつかの Oracle Net Services での解決方法を説明します。

接続性

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ます。Oracle Net Services は、クライアント・アプリケーションとデータベース・サーバー間の接続の確立と維持の他、これらの間でのメッセージの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。

管理性

Oracle Net Services により、位置の透過性、構成と管理の集中管理および迅速なインストールと構成が実現されます。

インターネットにおける拡張性

Oracle Net Services では、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 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拡張性とデータベースに同時に接続できるクライアント数が増大します。**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。

インターネットでのセキュリティ

ネットワークのセキュリティは、データベース・アクセス制御および Oracle Advanced Security などの機能を使用して強化します。

> **関連項目：** これらの機能の詳細は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

Oracle Net Services の機能

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 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 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 プロトコルは、Oracle アプリケーションのインタフェースから SQL 文を取り出し、それらをパッケージ化してから、サポートされている業界標準の高レベルのプロトコルまたはプログラム・インタフェースを介して Oracle に送信します。また、Oracle からの応答を取り込んでパッケージ化し、同じ高レベルの通信メカニズムを介して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 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 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 の機能の詳細は、『Oracle9*i* 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は、**リスナー・プロセス**によって Oracle への通信経路が確立されます。ユーザー・プロセスが接続要求を出すと、リスナーは共有サーバー・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。Real Application Clusters など、単一のコンピュータで複数のデータベースやインスタンスが稼働している場合は、**サービス名**を使用すると、インスタンスを同じマシン上の他のリスナーに自動的に登録できます。サービス名では複数のインスタンスを識別でき、インスタンスは複数のサービスに属することができます。サービスに接続するクライアントは、必要なインスタンスを指定する必要がありません。

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。登録済み情報により、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：

- サーバー・プロセスの詳細は、8-17 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。
- サーバー・プロセスの詳細は、8-22 ページの「専用サーバー構成」を参照してください。
- リスナーの詳細は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。
- Real Application Clusters でのインスタンス登録とクライアント / サーバー接続の詳細は、『Oracle9*i* Real Application Clusters セットアップおよび構成』および『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』を参照してください。

7

メモリー・アーキテクチャ

この章では、Oracle インスタンスのメモリー・アーキテクチャについて説明します。この章の内容は、次のとおりです。

- Oracle メモリー構造の概要
- システム・グローバル領域（SGA）の概要
- プログラム・グローバル領域（PGA）の概要
- 専用サーバーと共有サーバー
- ソフトウェア・コード領域

Oracle メモリー構造の概要

Oracle は、メモリーを使用して次のような情報を格納します。

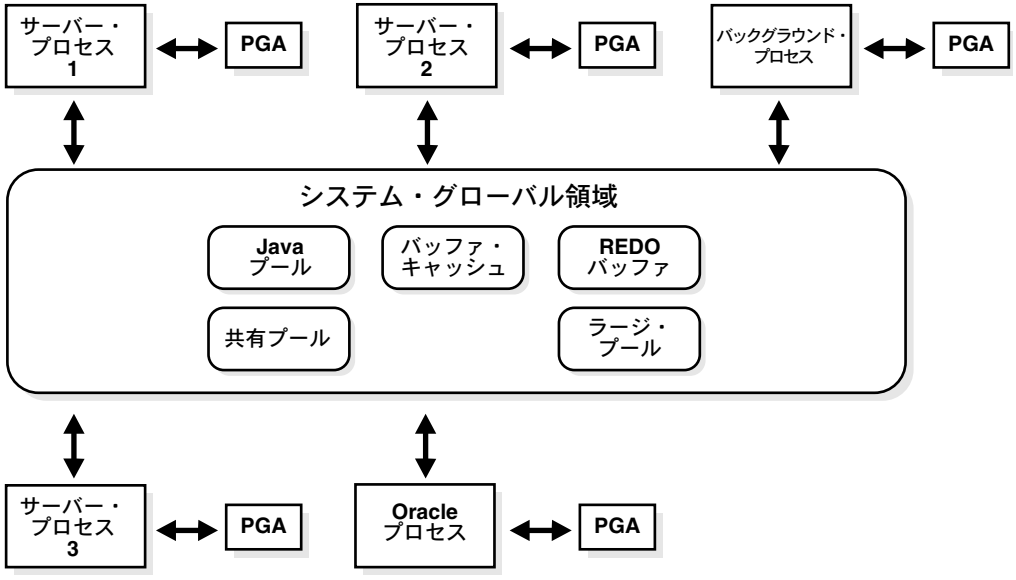
- プログラム・コード
- 現在はアクティブかどうかを問わず、接続されているセッションについての情報
- プログラムの実行時に必要な情報（行をフェッチしている問合せの現在の状態など）
- Oracle プロセス間で共有され、やり取りされる情報（ロック情報など）
- 周辺記憶装置にも永続的に格納されるキャッシュ・データ（データ・ブロックや REDO ログ・エントリなど）

Oracle に関連する基本的なメモリー構造は、次のような領域で構成されます。

- 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る、システム・グローバル領域（SGA）。SGA は次の領域を保持します。
  - 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（構成された場合）
- 各サーバーおよびバックグラウンド・プロセスに対しプライベートなプログラム・グローバル領域（PGA）。各プロセスには PGA が 1 つ存在します。PGA は次の領域を保持します。
  - スタック領域
  - データ領域

図 7-1 にこれらメモリー構造の関係を示します。

図 7-1 Oracle のメモリー構造



ソフトウェア・コード領域は、7-23 ページで説明するもう 1 つの基本的なメモリー構造です。

関連項目：

- 7-4 ページ「システム・グローバル領域（SGA）の概要」
- 7-18 ページ「プログラム・グローバル領域（PGA）の概要」

システム・グローバル領域（SGA）の概要

システム・グローバル領域（SGA）とは、共有メモリー構造のグループで、1 つの Oracle データベース・インスタンスのためのデータと制御情報が含まれています。複数のユーザーが同じインスタンスに同時に接続した場合、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ます。このため、SGA は**共有グローバル領域**と呼ばれることもあります。

SGA と Oracle プロセスによって、1 つの Oracle インスタンスが構成されます。インスタンスを起動すると、Oracle が自動的に SGA 用のメモリーを割り当て、インスタンスを停止すると、オペレーティング・システムがそのメモリーの割当てを解除します。各インスタンスには、それぞれ専用の SGA があります。

SGA は読取り / 書込み可能です。マルチ・プロセス・データベースのインスタンスに接続しているすべてのユーザーは、そのインスタンスの SGA に含まれている情報を読み込むことができます。また、Oracle の稼働中に、複数のプロセスが SGA に書き込むことができます。

SGA には、次のようなデータ構造が含まれています。

- データベース・バッファ・キャッシュ
- REDO ログ・バッファ
- 共有プール
- Java プール
- ラージ・プール（オプション）
- データ・ディクショナリ・キャッシュ
- その他の情報

SGA の一部には、バックグラウンド・プロセスがアクセスする必要のある、データベースとインスタンスの状態に関する一般的な情報が含まれています。この部分を**固定 SGA** と呼びます。ここには、ユーザー・データは格納されません。SGA には、ロック情報などのプロセス間でやり取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、要求キューと応答キュー、および PGA 領域の内容の一部は SGA に格納されます。

関連項目：

- Oracle インスタンスの詳細は、5-2 ページの「Oracle インスタンスの概要」を参照してください。
- 7-18 ページ「プログラム・グローバル領域（PGA）の概要」
- 8-18 ページ「ディスパッチャの要求キューと応答キュー」

動的 SGA

動的 SGA インフラストラクチャを使用して、インスタンスを停止することなく、バッファ・キャッシュ、共有プール、ラージ・プールおよびプロセス・プライベート・メモリーのサイズを変更できます。

Oracle では、実行時に、動的 SGA によって、Oracle が SGA 用に使用する仮想メモリー・サイズの上限を設定できます。Oracle では、構成中のインスタンスを起動して、そのインスタンスがより多くのメモリーを使用できるように SGA コンポーネントを増やすことができます（SGA_MAX_SIZE の最大値まで）。初期化パラメータ・ファイルで指定した SGA_MAX_SIZE が、初期化時に指定またはデフォルト設定された全コンポーネントの合計より少ない場合は、初期化パラメータ・ファイルでの設定は無視されます。

多くのシステムで最適なパフォーマンスを実現するには、SGA 全体が実メモリーに収まることが必要です。SGA 全体が実メモリーに収まらず、その一部を格納するために仮想メモリーが使用される場合、オペレーティング・システムが SGA の一部についてページング（ディスクの読取り / 書込み）を実行するため、データベース・システム全体のパフォーマンスが大幅に低下する可能性があります。SGA 内のすべての共有領域に専用に割り当てられるメモリーの量も、パフォーマンスに影響します。

SGA のサイズは、いくつかの初期化パラメータによって決定されます。次のパラメータは、SGA サイズに最も大きく影響します。

パラメータ	説明
DB_CACHE_SIZE	標準ブロックのキャッシュ・サイズ。
LOG_BUFFER	REDO ログ・バッファに割り当てられるバイト数。
SHARED_POOL_SIZE	共有 SQL 文と共有 PL/SQL 文に割り当てられる領域のサイズ（単位はバイト）。
LARGE_POOL_SIZE	ラージ・プールのサイズ。デフォルトは 0（ゼロ）。

Enterprise Manager または SQL*Plus を使用しているときには、インスタンスの SGA に割り当てられたメモリーがインスタンスの起動時に表示されます。SQL*Plus の SHOW 文に SGA 句を指定して、現行インスタンスの SGA サイズを表示することもできます。

動的 SGA のグラニュル

動的 SGA での割当て単位はグラニュルと呼ばれます。バッファ・キャッシュ、共有プール、Java プールおよびラージ・プールなどのコンポーネントは、SGA 領域をグラニュル単位で割り当て、解放します。Oracle では、SGA メモリーの使用が SGA コンポーネント別に整数値のグラニュル数で追跡されます。グラニュルに関するすべての情報は、対応するグラニュル・エントリに格納されます。Oracle では、グラニュル・エントリ内の各グラニュルの状態とグラニュル・タイプがメンテナンスされます。

グラニュル・サイズは、SGA 全体のサイズによって決定されます。ほとんどのプラットフォームでは、SGA 全体のサイズが 128MB より小さければグラニュル・サイズは 4MB、128MB 以上の場合は 16MB です。プラットフォームへの依存性が存在する場合があり、たとえば 32 ビットの Windows NT では、128MB 以上の SGA のグラニュル・サイズは 8MB です。

現在 SGA に使用されているグラニュル・サイズは、V$SGA_DYNAMIC_COMPONENTS ビューで表示できます。SGA 内のすべての動的コンポーネントには、同じグラニュル・サイズが使用されます。

> **注意：** コンポーネント用にグラニュル・サイズの倍数でないサイズを指定すると、最も近似の倍数に切り上げられます。たとえば、グラニュル・サイズが 4MB の場合に DB_CACHE_SIZE を 10MB として指定すると、実際には 12MB が割り当てられます。

Oracle では、コンポーネントとそのグラニュルに関する情報がスコアボードに格納されます。スコアボードには、グラニュルを所有するコンポーネントごとに、そのコンポーネントに割り当てられたグラニュル数、そのコンポーネントに対するペンディング操作、グラニュル数でのターゲット・サイズおよびターゲット・サイズに対する進捗が含まれます。操作の開始時刻もログに記録されます。Oracle では、コンポーネントごとに初期のグラニュル数と最大グラニュル数がメンテナンスされます。

グラニュル数を変更する操作については、その操作、ターゲット・サイズおよび適切な SGA コンポーネントに対する開始時刻のログがスコアボードに記録されます。Oracle では、操作が完了するまで進捗フィールドが更新されます。操作が完了すると、現行サイズがターゲット・サイズで置換され、ターゲット・サイズ・フィールドと進捗フィールドが消去されます。操作が完了した時点で、データベース管理者はグラニュル数の変化を調べることができます。初期化パラメータの値は、更新された SGA 使用量を反映するように更新されます。

Oracle では、スコアボードに対して実行された過去 100 回の操作の循環バッファがメンテナンスされます。固定ビューには、スコアボードの状態と、スコアボードに対する過去 100 回の操作の現在の内容が表示されます。

起動時のグラニュルの割当て 起動時に、Oracle は初期化パラメータ・ファイル内の値を読み込み、オペレーティング・システムのメモリー制限を問い合せて、SGA 用の仮想アドレス空間を割り当てます。初期化パラメータ SGA_MAX_SIZE では、インスタンスの存続期間中

の SGA の最大サイズをバイト単位で指定します。このパラメータの値は、次のグラニュル・サイズに切り上げられます。

コンポーネントへのグラニュルの追加 データベース管理者は、ALTER SYSTEM 文を使用して初期化パラメータの値を変更し、コンポーネントによる SGA 使用量を増やします。Oracle は、16MB に最も近似の倍数に切り上げられた新規サイズを使用し、ターゲット・サイズに合せてグラニュル数を増減させます。Oracle には、要求を十分に満たせる空きグラニュルが必要です。現在の SGA メモリー量が SGA_MAX_SIZE より小さい場合、Oracle は SGA サイズが SGA_MAX_SIZE に達するまでグラニュルを割り当てることができます。

関連項目：

- メモリー割当ての詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- Enterprise Manager を使用して SGA サイズを表示する方法は、『Oracle Enterprise Manager 管理者ガイド』を参照してください。
- SQL*Plus で SGA のサイズを表示する方法は、『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。
- V$SGASTAT の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- オペレーティング・システム固有の情報は、該当する Oracle インストレーション・ガイドまたはユーザーズ・ガイドを参照してください。

データベース・バッファ・キャッシュ

データベース・バッファ・キャッシュは SGA の一部であり、データ・ファイルから読み込んだデータ・ブロックのコピーが保持される領域です。インスタンスに同時接続されたユーザー・プロセスはすべて、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。

データベース・バッファ・キャッシュと共有 SQL キャッシュは、論理的にセグメント化されて複数の集合になります。このように複数の集合へ編成すると、マルチ・プロセッサ・システム上での競合が減少します。

データベース・バッファ・キャッシュの編成

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ます。書込みリストおよび最低使用頻度リスト（LRU）です。**書込みリスト**は、使用済みバッファを保持します。使用済みバッファとは、修正されたが、まだディスクに書き込まれていないデータを含むバッファのことです。**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みバッファを保持します。**使用可能バッファ**は、有効なデータが含まれておらず、使用可能なバッファです。**使用中バッファ**は、現在アクセスされているバッファです。

Oracle プロセスがバッファにアクセスするとき、プロセスはそのバッファを LRU リストの最高使用頻度（MRU）側に移動します。さらに多くのバッファが LRU リストの MRU 側へ移動されるにつれて、使用済みバッファは LRU リストの LRU 側に向かって古くなります。

Oracle ユーザー・プロセスでデータの特定の部分が初めて必要になると、プロセスは データベース・バッファ・キャッシュ内のデータを検索します。キャッシュ内にデータが見つかった場合（**キャッシュ・ヒット**）、プロセスはデータをメモリーから直接読み込むことができます。キャッシュ内にデータが見つからなかった場合（**キャッシュ・ミス**）、プロセスはデータにアクセスする前に、ディスク上のデータ・ファイルからキャッシュ内のバッファにデータ・ブロックをコピーする必要があります。キャッシュ・ヒットによるデータのアクセスは、キャッシュ・ミスによるデータのアクセスよりも高速です。

プロセスはキャッシュ内にデータ・ブロックを読み込む前に、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み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・ブロックを読み込んで、バッファを LRU リストの MRU 側に移動します。

使用可能バッファが見つからず、検索したバッファ数が制限しきい値に達すると、Oracle ユーザー・プロセスがは LRU リストの検索を停止し、使用済みバッファの一部をディスクに書き込むように、DBW0 バックグラウンド・プロセスに信号を送ります。

> **関連項目：** DBW*n* プロセスの詳細は、8-9 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込んで LRU リストの（MRU 側ではなく）LRU 側に入れます。通常、全表スキャンされた表は一時的に必要なので、使用頻度の高いブロックをキャッシュ内に残しておくために、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、『Oracle9*i* SQL リファレンス』を参照してください。

データベース・バッファ・キャッシュのサイズ

Oracle は、データベース内で複数のブロック・サイズをサポートします。サポートされるのはデフォルトのブロック・サイズで、SYSTEM 表領域で使用されます。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 から 32K です。

標準ブロック・サイズ・キャッシュのサイズを指定するには、初期化パラメータ DB_CACHE_SIZE を設定します。オプションで、DB_KEEP_CACHE_SIZE および DB_RECYCLE_CACHE_SIZE を設定すると、2 つの追加バッファ・プール KEEP および RECYCLE のサイズも設定できます。これら 3 つのパラメータは、互いに独立しています。

> **関連項目：** KEEP および RECYCLE バッファ・プールの詳細は、7-10 ページの「複数バッファ・プール」を参照してください。

非標準ブロック・サイズのバッファのサイズと数は、次のパラメータで指定します。

```
DB_2K_CACHE_SIZE
DB_4K_CACHE_SIZE
DB_8K_CACHE_SIZE
DB_16K_CACHE_SIZE
DB_32K_CACHE_SIZE
```

各パラメータで、対応するブロック・サイズのキャッシュ・サイズを指定します。

> **注意：** ブロック・サイズの上限値に関してはプラットフォーム固有の制限が適用されるため、プラットフォームによっては指定したサイズが適用できないこともあります。

ブロック・サイズとキャッシュ・サイズの設定例

```
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=1024M
DB_2K_CACHE_SIZE=256M
DB_8K_CACHE_SIZE=512M
```

前述の例では、パラメータ DB_BLOCK_SIZE は、データベースの標準ブロック・サイズを 4K に設定しています。標準ブロック・サイズ・バッファのキャッシュ・サイズは 1024MB です。さらに、バッファ・サイズがそれぞれ 256MB および 512MB である 2K および 8K のキャッシュも構成されます。

注意： DB_*n*K_CACHE_SIZE パラメータで標準ブロック・サイズのキャッシュ・サイズを指定することはできません。DB_BLOCK_SIZE の値が *n*K の場合、DB_*n*K_CACHE_SIZE に値を設定することはできません。標準ブロック・サイズのキャッシュ・サイズは必ず DB_CACHE_SIZE の値から決定します。

キャッシュのサイズには制限があるため、ディスク上のすべてのデータをキャッシュに入れられるとはかぎりません。キャッシュがいっぱいになった後でキャッシュ・ミスが発生すると、Oracle は新しいデータ用の領域を確保するために、すでにキャッシュ内にある使用済みデータをディスクに書き込みます。（バッファが使用済みでなければ、新しいブロックをバッファに読み込む前にディスクに書き込む必要はありません。）その後、ディスクに書き込まれたデータにアクセスすると、それもキャッシュ・ミスになります。

データを要求したときにキャッシュ・ヒットになる確率は、キャッシュのサイズによって決まります。キャッシュが大きければ、要求されたデータがキャッシュに入っている可能性は高くなります。キャッシュのサイズを大きくすると、データ要求がキャッシュ・ヒットになる確率が高くなります。

インスタンスの実行時でも、データベースを停止せずにバッファ・キャッシュのサイズを変更できます。この操作には ALTER SYSTEM 文を使用します。詳細は、7-16 ページの「SGA のメモリーの使用方法の制御」を参照してください。

個々のキャッシュ・コンポーネントのサイズおよび保留中のサイズ変更操作を追跡するには、固定ビュー V$BUFFER_POOL を使用します。

関連項目： バッファ・キャッシュのチューニング方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

複数バッファ・プール

異なるバッファ・プールを持つデータベース・バッファ・キャッシュを構成して、バッファ・キャッシュ内にデータを保持するか、またはデータ・ブロックの使用直後に新しいデータがバッファを使用するかを指定できます。その後、特定のスキーマ・オブジェクト（表およびクラスタ、索引およびパーティション）を適切なバッファ・プールに割り当てて、キャッシュからデータ・ブロックのエージングを行う方法を制御できます。

- KEEP バッファ・プールは、スキーマ・オブジェクトのデータ・ブロックをメモリーに保持します。
- RECYCLE バッファ・プールは、データ・ブロックが不要になると、すぐにそれをメモリーから排除します。
- DEFAULT バッファ・プールには、どのバッファ・プールにも割り当てられていないスキーマ・オブジェクトのデータ・ブロックと、明示的に DEFAULT プールに割り当てられたスキーマ・オブジェクトが含まれます。

KEEP バッファ・プールと RECYCLE バッファ・プールを構成する初期化パラメータは、DB_KEEP_CACHE_SIZE と DB_RECYCLE_CACHE_SIZE です。

注意： 複数バッファ・プールは、標準ブロック・サイズに対してのみ使用可能です。非標準のブロック・サイズのキャッシュの DEFAULT プールは 1 つです。

関連項目：

- 複数バッファ・プールの詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- STORAGE 句の BUFFER_POOL 句の構文については、『Oracle9*i* SQL リファレンス』を参照してください。

REDO ログ・バッファ

REDO ログ・バッファとは、SGA 内の循環バッファであり、データベースに加えられた変更についての情報を保持します。この情報は、**REDO エントリ**に格納されます。REDO エントリには、INSERT、UPDATE、DELETE、CREATE、ALTER または DROP の各操作によってデータベースに加えられた変更の再構築または再実行に必要な情報が含まれます。REDO エントリは、必要に応じてデータベースのリカバリ時に使用されます。

REDO エントリは、Oracle のサーバー・プロセスによってユーザーのメモリー領域から SGA 内の REDO ログ・バッファにコピーされます。REDO エントリは、バッファ内で連続した領域を占めます。バックグラウンド・プロセス LGWR は、REDO ログ・バッファをディスク上のアクティブなオンライン REDO ログ・ファイル（または REDO ログ・ファイルのグループ）に書き込みます。

関連項目：

- REDO ログ・バッファがディスクに書き込まれる方法の詳細は、8-10 ページの「ログ・ライター・プロセス（LGWR）」を参照してください。
- オンライン REDO ログ・ファイルとグループの詳細は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。

初期化パラメータ LOG_BUFFER は、REDO ログ・バッファのサイズをバイト単位で設定します。一般的に（トランザクションが長いか、数が多い場合には特に）、設定する値を大きくするほどログ・ファイルの I/O は少なくなります。デフォルト設定は、512KB、または 128KB と CPU_COUNT パラメータの設定値を乗算した値のうち、いずれか大きい方です。

共有プール

SGA の共有プール部分には、次の 3 つの主要領域が含まれます。ライブラリ・キャッシュ、ディクショナリ ・キャッシュ、パラレル実行メッセージおよび制御構造です。

注意： 初期化パラメータ PARALLEL_AUTOMATIC_TUNING の設定が true の場合は、これらのバッファはラージ・プールから割り当てられます。

共有プールの合計サイズは、初期化パラメータ SHARED_POOL_SIZE によって決まります。このパラメータのデフォルト値は、32 ビットのプラットフォームでは 8MB、64 ビットのプラットフォームでは 64MB です。このパラメータの値を増やすと共有プール用に確保するメモリー量が増加します。

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（複数トランザクション・サーバーの場合）、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 は、実行する各 SQL 文を、**共有 SQL 領域**と**プライベート SQL 領域**によって表します。Oracle は、2 人のユーザーが同じ SQL 文を実行する場合にそれを認識し、これらのユーザーのために共有 SQL 領域を再利用します。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 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 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 は、新しい SQL 文が解析されると共有プールからメモリーを割り当て、共有 SQL 領域に格納します。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みの場合、Oracle は、新しい文の共有 SQL 領域のための空き領域が十分になるまで、修正した LRU アルゴリズムを使用してその共有プールからエントリの割当てを解除できます。Oracle が共有 SQL 領域を割当て解除する場合、対応付けられていた SQL 文は次の実行時に再解析され、別の共有 SQL 領域に再び割り当てられます。

関連項目：

- 7-18 ページ「プライベート SQL 領域」
- 『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』

PL/SQL プログラム・ユニットと共有プール

Oracle は、PL/SQL プログラム・ユニット（プロシージャ、ファンクション、パッケージ、無名ブロックおよびデータベース・トリガー）を、個々の SQL 文の処理と同様の方法で処理します。プログラム・ユニットを解析およびコンパイル済みの形で保持するために、共有領域が割り当てられます。Oracle では、ローカル変数、グローバル変数、パッケージ変数（パッケージ・インスタンス化とも呼ばれる）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、プライベート領域が割り当てられます。複数のユーザーが同じプログラム・ユニットを実行している場合、単一の共有領域はすべてのユーザーによって使用されますが、各ユーザーは、自分のセッションに固有の値を保持するプライベート SQL 領域のコピーを個別にメンテナンスします。

PL/SQL プログラム・ユニット内に含まれる個々の SQL 文は、先の項で説明したように処理されます。PL/SQL プログラム・ユニット内の起点には関係なく、これらの SQL 文は解析された表現を保持するために共有領域を使用し、その文を実行するセッションごとにプライベート SQL 領域を使用します。

ディクショナリ・キャッシュ

データ・ディクショナリとは、データベース、データベースの構造およびそのユーザーについての参照情報を含むデータベースの表およびビューの集合のことです。Oracle は、SQL 文の解析時にデータ・ディクショナリに頻繁にアクセスします。このアクセスは、Oracle の操作を続けるために不可欠です。

データ・ディクショナリは Oracle によって頻繁にアクセスされるので、ディクショナリ・データを保持するために、メモリー内に 2 箇所の特別な場所が指定されています。一方の領域は、データのブロック全体を保持するバッファのかわりに行としてデータを保持するため、**データ・ディクショナリ・キャッシュ**または**行キャッシュ**と呼ばれます。ディクショナリ・データを保持するメモリー内の他方の領域は、ライブラリ・キャッシュです。すべての Oracle ユーザー・プロセスは、データ・ディクショナリ情報にアクセスするために、これら 2 つのキャッシュを共有します。

関連項目：

- 第 4 章「データ・ディクショナリ」
- 7-12 ページ「ライブラリ・キャッシュ」

共有プール内のメモリーの割当てと再利用

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限りメモリーに残ります。その結果、マルチユーザー Oracle システムに関連する SQL 文のオーバーヘッドと処理が、最小限に抑えられます。

SQL 文が実行のために Oracle に送られると、Oracle は次のメモリー割当ての手順を自動的に実行します。

1. 同一の文について共有 SQL 領域がすでに存在しているかどうかを確認するため、共有プールをチェックします。その文のための共有 SQL 領域がすでに存在する場合、その領域は、その文の後続の新しいインスタンスを実行するために使用されます。一方、その文のための共有 SQL 領域が存在しない場合、Oracle は新しい共有 SQL 領域を 共有プール内に割り当てます。どちらの場合も、ユーザーのプライベート SQL 領域が、その文を含む共有 SQL 領域に対応付けられます。

   注意： 共有 SQL 領域がオープンしているカーソルに対応していても、しばらく使用されていないカーソルであれば、その共有 SQL 領域を共有プールからフラッシュできます。オープンしているカーソルが、その後その文を実行するために使用される場合、その文は再解析されて新しい共有 SQL 領域が共有プール内に割り当てられます。

2. セッションのためにプライベート SQL 領域を割り当てます。プライベート SQL 領域の位置は、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 は、次のような場合にも共有 SQL 領域を共有プールからフラッシュします。

- 表、クラスタまたは索引の統計を更新または削除するために `ANALYZE` 文を使用する場合、分析されたスキーマ・オブジェクトを参照する文を含んでいるすべての共有 SQL 領域は、共有プールからフラッシュされます。フラッシュされた文を次に実行するときには、その文はスキーマ・オブジェクトの新しい統計を反映するように新しい共有 SQL 領域で解析されます。
- スキーマ・オブジェクトが SQL 文で参照され、なんらかの方法で修正されると、共有 SQL 領域は**無効**になり（無効のマークが付けられる）、その文を次に実行するときには解析する必要があります。
- データベースのグローバル・データベース名を変更すると、すべての情報が共有プールからフラッシュされます。
- 管理者はインスタンス起動後のパフォーマンス（データ・バッファ・キャッシュではなく、共有プールに関するパフォーマンス）を査定するために、共有プール内のすべての情報を手動でフラッシュできます。こうすると、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、7-12 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 15 章「スキーマ・オブジェクト間の依存性」を参照してください。
- ALTER SYSTEM FLUSH SHARED_POOL の使用方法は、『Oracle9*i* SQL 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、『Oracle9*i* データベース・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 のバックアップおよびリストア操作
- 初期化パラメータ PARALLEL_AUTOMATIC_TUNING の設定が true の場合は、パラレル実行メッセージ・バッファ（false の場合は、これらのバッファは共有プールに割り当てられる）

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、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することができます。

さらに、Oracle のバックアップおよびリストア操作と I/O サーバー・プロセス用、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、8-17 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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- 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、『Oracle9*i* データベース・パフォーマンス・プランニング』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、18-8 ページの「並列度」を参照してください。

SGA のメモリーの使用方法の制御

動的 SGA では、Oracle が使用する物理メモリーの増減を外部制御できます。動的バッファ・キャッシュ、共有プールおよびラージ・プールとともに動的 SGA では、次のことが可能です。

- SGA は、オペレーティング・システムが指定する最大値および `SGA_MAX_SIZE` の指定値まで、データベース管理文により拡張可能。
- SGA は、Oracle が規定する最小値、（通常はオペレーティング・システムが推奨する限度まで）データベース管理文により縮小可能。
- バッファ・キャッシュと SGA プールは、ある内部的な Oracle の管理方針に従って実行時に拡張と縮小が可能。

その他の SGA 初期化パラメータ

複数の初期化パラメータを使用して、SGA によるメモリーの使用方法を制御できます。

物理メモリー

LOCK_SGA パラメータは、SGA を物理メモリーにロックします。

SGA 開始アドレス

SHARED_MEMORY_ADDRESS パラメータと HI_SHARED_MEMORY_ADDRESS パラメータは、実行時の SGA の開始アドレスを指定します。これらのパラメータはほとんど使用されません。64 ビットのプラットフォームでは、HI_SHARED_MEMORY_ADDRESS は 64 ビット・アドレスの上位 32 ビットを指定します。

拡張バッファ・キャッシュ・メカニズム

USE_INDIRECT_DATA_BUFFERS パラメータは、4GB を超える物理メモリーのサポート機能を持った 32 ビット・プラットフォームで、拡張バッファ・キャッシュ・メカニズムを使用可能にします。

ただし、動的バッファ・キャッシュ機能では、すべてのバッファが有効な仮想アドレスを持つことが必要になります。これは、基礎となる割当て単位のグラニュルが、仮想アドレスで識別されるためです。このため、現行バージョンでは拡張キャッシュ機能を使用できません。

関連項目：

- USE_INDIRECT_DATA_BUFFERS パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- オペレーティング・システム固有の情報は、該当する Oracle インストレーション・ガイドまたはユーザーズ・ガイドを参照してください。

プログラム・グローバル領域（PGA）の概要

プログラム・グローバル領域（PGA）とは、1 つのサーバー・プロセスのためのデータと制御情報を含むメモリー・リージョンです。PGA は、サーバー・プロセスの起動時に、Oracle によって作成される非共有メモリーです。PGA へのアクセスは起動時のサーバー・プロセスに限定され、読込みと書込みは PGA のために機能する Oracle コードでのみ行われます。Oracle インスタンスに連結した各サーバー・プロセスにより割り当てられた全 PGA メモリーは、インスタンスにより割り当てられた**集計 PGA メモリー**と呼ばれることもあります。

> **関連項目：** セッションについては、8-5 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。しかし、一般的に、PGA メモリーは次のように分類できます。

プライベート SQL 領域

プライベート SQL 領域は、バインド情報などのデータとランタイム・メモリー構造を格納します。SQL 文を発行するそれぞれのセッションには、プライベート SQL 領域があります。同一の SQL 文を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。

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。

- バインド情報などを格納する、持続領域。持続領域は、カーソルのクローズ時にのみ解放されます。
- 実行終了時に解放される、ランタイム領域。

Oracle は、実行要求の最初の処理でランタイム領域を作成します。`INSERT`、`UPDATE` および `DELETE` 文では、Oracle は文の実行後にランタイム領域を解放します。問合せの場合、Oracle は、すべての行がフェッチされた後、または問合せが取り消された後にのみ、ランタイム領域を解放します。

プライベート SQL 領域の位置は、セッションのために確立される接続のタイプによって異なります。セッションが専用サーバーを介して接続されている場合、プライベート SQL 領域はサーバー・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。

関連項目：

- PGA の詳細は、7-18 ページの「プログラム・グローバル領域（PGA）の概要」を参照してください。
- セッションの詳細は、8-5 ページの「接続とセッション」を参照してください。
- ソート、ハッシュ結合、ビットマップ作成またはビットマップ・マージ実行中の SELECT ランタイムの詳細は、7-20 ページの「SQL 作業領域」を参照してください。
- 共有サーバーの概要は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

カーソルと SQL 領域 Oracle プリコンパイラ・プログラムや OCI プログラムのアプリケーション開発者は、**カーソル**、つまり特定のプライベート SQL 領域へのハンドルを明示的にオープンし、それらのカーソルをプログラムの実行中に名前付きリソースとして使用できます。Oracle が一部の SQL 文のために暗黙的に発行する再帰カーソルも共有 SQL 領域を使用します。

プライベート SQL 領域は、ユーザー・プロセスが管理します。ユーザー・プロセスが割り当てることのできるプライベート SQL 領域の数は初期化パラメータ OPEN_CURSORS によって制限されますが、プライベート SQL 領域の割当ておよび割当て解除は、使用するアプリケーション・ツールに大きく依存します。このパラメータのデフォルト値は 300 です。

プライベート SQL 領域は、対応するカーソルがクローズされるか、文ハンドルが解放されるまで存在します。Oracle は文の実行が完了した後にランタイム領域を解放しますが、持続領域は待機し続けます。持続領域を解放し、アプリケーションのユーザーが必要とするメモリー容量を最小限に抑えるには、オープンされているカーソルのうち再利用しないものを、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： 14-6 ページ「カーソル」

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）。

SQL 作業領域

複雑な問合せでは（たとえば、意思決定支援の問合せ）、次に示すようなメモリー集中型の演算子により、ランタイム領域の大部分が作業領域に専用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る）を使用して、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、入力データを小さな部分に分割します。これにより、メモリー内でデータの一部を処理することができます。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、一時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稼働します。

作業領域のサイズは、制御およびチューニング可能です。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。

専用モードに対する PGA メモリー管理

SQL 作業領域を、自動的かつグローバルに管理できます。データベース管理者は、初期化パラメータ PGA_AGGREGATE_TARGET を設定し、Oracle インスタンス専用の PGA メモリーの合計サイズを指定します。指定した数値（たとえば、2G）は、Oracle インスタンスのグローバル・ターゲットであるため、Oracle は、すべてのデータベース・サーバー・プロセスに割り当てた PGA メモリーの合計がこのターゲット値を超えないことを確認します。

注意： 以前のリリースでは、データベース管理者はパラメータ SORT_AREA_SIZE、HASH_AREA_SIZE、BITMAP_MERGE_AREA_SIZE および CREATE_BITMAP_AREA_SIZE を設定することで、SQL 作業領域の最大サイズを制御していました。ただし、最大作業領域サイズは、データ入力サイズとシステム内でアクティブな作業領域の合計数から選択するのが理想であるため、これらのパラメータの設定は困難です。これら 2 つの要因は、作業領域および時間により大幅に変動します。このため、各種 *_AREA_SIZE パラメータを最適な状況の下でチューニングすることは困難です。

PGA_AGGREGATE_TARGET を使用すると、すべての専用セッションの作業領域のサイズ設定が自動的に行われ、これらのセッションについてはすべての *_AREA_SIZE パラメータが無視されます。ある時点での、インスタンスのアクティブな作業領域で使用可能な PGA メモリーの合計は、パラメータ PGA_AGGREGATE_TARGET から自動的に導出されます。この量は、PGA_AGGREGATE_TARGET の値から、システムの他の構成要素により割り当てられた PGA メモリー（たとえば、セッションにより割り当てられた PGA メモリー）を差し引いた値に設定されます。結果として、PGA メモリーは、特定のメモリー要件に基づいて個々のアクティブな作業領域に割り当てられます。

注意： 初期化パラメータ WORKAREA_SIZE_POLICY は、セッション・レベルおよびシステム・レベルのパラメータで、設定できる値は MANUAL または AUTO の 2 つのみです。デフォルトは AUTO です。データベース管理者は、PGA_AGGREGATE_TARGET を設定して、次にメモリー管理モードを自動から手動に切り替えます。

PGA メモリーの使用統計を提供する固定ビューと列が用意されています。これらの統計の多くは、PGA_AGGREGATE_TARGET を設定することで使用できます。

- 作業領域メモリーの割当ておよび使用に関する統計は、次の動的ビューで表示できます。

  V$SYSSTAT
  V$SESSTAT
  V$PGASTAT

```
V$SQL_WORKAREA
V$SQL_WORKAREA_ACTIVE
```

- V$PROCESS ビューの次の 3 つの列では、Oracle プロセスによって割り当てられ使用されている PGA メモリーがレポートされます。

```
PGA_USED_MEM
PGA_ALLOCATED_MEM
PGA_MAX_MEM
```

注意： 自動 PGA メモリー管理モードは、専用サーバーが割り当てた作業領域にのみ適用します。共有サーバーが割り当てた作業領域のサイズは、旧 *_AREA_SIZE パラメータで制御します。これらの作業領域は、PGA ではなく SGA に主に割り当てられるからです。

関連項目：

- ビューについては、『Oracle9*i* データベース・リファレンス』を参照してください。
- これらのビューの使用方法は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

専用サーバーと共有サーバー

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、メモリーの割当て方法が変わることがあります。表 7-1 にこれらの相違点を示しています。

表 7-1 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	SGA
DML/DDL 文のランタイム領域の位置	PGA	PGA

ソフトウェア・コード領域

ソフトウェア・コード領域とは、実行中または実行される可能性があるコードを格納するためのメモリー部分です。Oracle のコードはソフトウェア領域に格納されますが、これはユーザー・プログラムが格納される領域の位置とは異なる位置、つまり排他的で保護された位置になります。

ソフトウェア領域のサイズは通常固定されており、ソフトウェアの更新時か再インストール時にかぎり変化します。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。可能なときには、メモリー内に Oracle コードの複数のコピーを持たずにすべての Oracle ユーザーが Oracle コードにアクセスできるようにするため、Oracle コードは共有されます。これにより、実際の主メモリーが節約され、全体のパフォーマンスが改善されます。

ユーザー・プログラムは共有でも非共有でもかまいません。Forms Developer 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 の複数のインスタンスが同じコンピュータ上で実行されている場合、それらのインスタンスは異なるデータベースにおいても同じ Oracle コード領域を使用できます。

注意： ソフトウェアを共有でインストールするオプションは、すべてのオペレーティング・システムで使用できるわけではありません（たとえば、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・システム固有の Oracle マニュアルを参照してください。

8

プロセス・アーキテクチャ

この章では、Oracle データベース・システムのプロセスと、Oracle システムで使用可能な各種構成について説明します。この章の内容は、次のとおりです。

- プロセスの概要
- ユーザー・プロセスの概要
- Oracle プロセスの概要
- 共有サーバー・アーキテクチャ
- 専用サーバー構成
- プログラム・インタフェース

プロセスの概要

接続している Oracle ユーザーはすべて、Oracle データベース・インスタンスにアクセスするために、2 つのコード・モジュールを実行する必要があります。

- アプリケーションまたは Oracle 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 サーバー・コード : 各ユーザーは自分のために Oracle サーバー・コードをいくつか持ち、これがアプリケーションの SQL 文を解釈して処理します。

これらのコード・モジュールは、プロセスによって実行されます。**プロセス**は制御のスレッド、つまり一連の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。

マルチ・プロセス Oracle システム

マルチ・プロセス Oracle（**マルチユーザー Oracle** とも呼ばれる）は、Oracle コードの各部分とユーザーのためのその他のプロセスを実行するために、複数のプロセスを使用します。ユーザー用プロセスは、接続中のユーザーごとに個別のプロセスの場合や、複数のユーザーが共有する 1 つ以上のプロセスの場合があります。データベースの主な利点の 1 つが、複数のユーザーが同時に必要とするデータを管理できることにあるため、多くのデータベース・システムはマルチユーザー・システムです。

Oracle インスタンスにおける各プロセスは、特定のジョブを実行します。Oracle とデータベース・アプリケーションの作業を複数のプロセスに分けることにより、システムの優れたパフォーマンスを維持しながら、複数のユーザーやアプリケーションが同時に 1 つのデータベース・インスタンスに接続できます。

プロセスのタイプ

Oracle システム内のプロセスは、次の 2 つの主要グループに分類できます。

- ユーザー・プロセスは、アプリケーションまたは Oracle のツール製品のコードを実行します。
- Oracle プロセスは、Oracle 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。

プロセス構造は、オペレーティング・システムと、選択する Oracle オプションによって、Oracle 構成ごとに異なります。接続ユーザー用のコードは、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 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 サーバー・コードを実行するプロセスが異なります。Oracle サーバー・コードを実行する各サーバー・プロセス（**共有サーバー・プロセス**）は、マルチ・ユーザー・プロセスとして機能します。

図 8-1 に、専用サーバー構成を示します。接続中の各ユーザーは別々のユーザー・プロセスを持ち、複数のバックグラウンド・プロセスが Oracle を実行します。

図 8-1 Oracle インスタンス

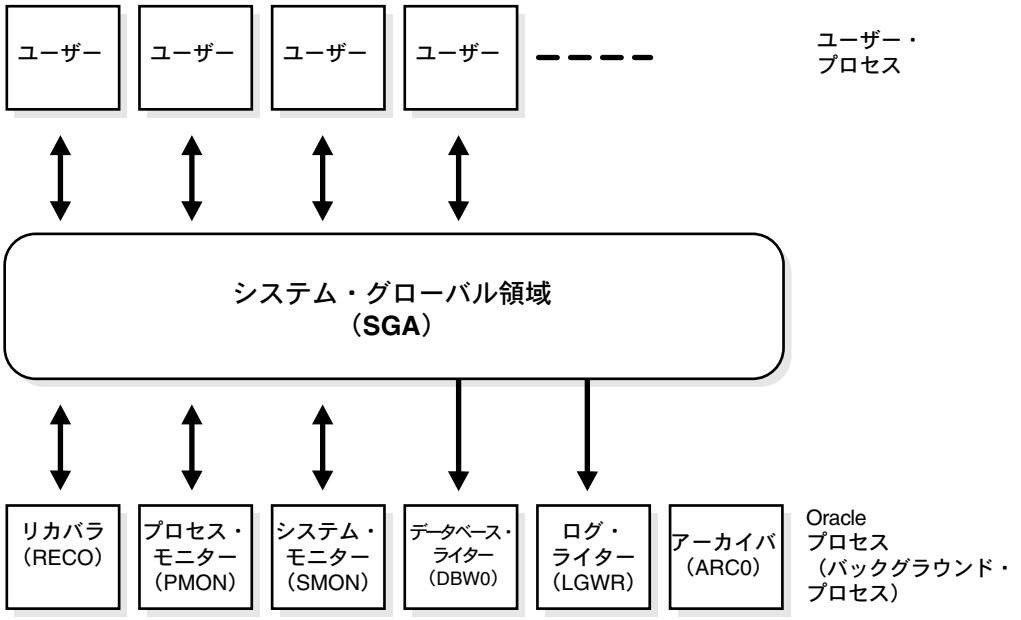


図 8-1 は、複数の同時実行ユーザーが、Oracle と同一のマシン上にあるアプリケーションを実行していることを表しています。このような特別の構成は、通常、メインフレームまたはミニコンピュータで稼働します。

関連項目：

- 8-5 ページ「ユーザー・プロセスの概要」
- 8-6 ページ「Oracle プロセスの概要」
- 8-22 ページ「専用サーバー構成」
- 8-17 ページ「共有サーバー・アーキテクチャ」
- 構成の選択肢の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

ユーザー・プロセスの概要

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（Enterprise Manager や SQL*Plus など）を実行すると、Oracle はユーザーのアプリケーションを実行するために**ユーザー・プロセス**を作成します。

接続とセッション

接続および**セッション**という用語は、**ユーザー・プロセス**という用語と密接に関連していますが、意味的には大きな差異があります。

接続とは、ユーザー・プロセスと Oracle インスタンスの間の通信経路のことです。通信経路は、使用可能なプロセス間通信メカニズム（1 台のコンピュータでユーザー・プロセスと Oracle の両方を実行する場合）、またはネットワーク・ソフトウェア（別々のコンピュータがデータベース・アプリケーションと Oracle を実行し、ネットワークを介して通信する場合）を使用して確立されます。

セッションとは、ユーザーがユーザー・プロセスを介して Oracle インスタンスに接続するときの、特定の接続のことです。たとえば、ユーザーは、SQL*Plus を開始するときに、有効なユーザー名とパスワードを入力する必要があります。そうすることにより、そのユーザーのためのセッションが確立されます。セッションは、ユーザーが接続した時点から、接続を切断するかデータベース・アプリケーションを終了する時点まで続きます。

1 人の Oracle ユーザーに対して複数のセッションを作成し、それらを同じユーザー名で同時に存在させることができます。たとえば、`SCOTT/TIGER` というユーザー名 / パスワードを持つユーザーは、同じ Oracle インスタンスに何度も接続できます。

共有サーバー以外の構成では、Oracle は、各ユーザー・セッションのためにサーバー・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： 8-17 ページ「共有サーバー・アーキテクチャ」

Oracle プロセスの概要

この項では、Oracle サーバー・コードを実行する 2 種類のプロセス（サーバー・プロセスとバックグラウンド・プロセス）と、Oracle プロセスのデータベース・イベントが記録されるトレース・ファイルとアラート・ファイルについて説明します。

サーバー・プロセス

Oracle では、インスタンスに接続されたユーザー・プロセスの要求を処理するために、**サーバー・プロセス**が作成されます。アプリケーションと Oracle が同一のマシン上で稼働しているときには、システムのオーバーヘッドを軽減するために、ユーザー・プロセスとそれに対応するサーバー・プロセスを 1 つのプロセスに結合できる場合もあります。ただし、アプリケーションと Oracle がそれぞれ別のマシン上で稼働している場合は、ユーザー・プロセスは常に独立したサーバー・プロセスを通じて Oracle と通信します。

各ユーザー・アプリケーションのために作成されたサーバー・プロセス（または、結合されたユーザー / サーバー・プロセスのサーバー部）は、次の 1 つ以上の操作を実行できます。

- アプリケーションを介して発行された SQL 文を解析し、実行します。
- 必要なデータ・ブロックが SGA 内に存在しない場合、そのブロックをディスク上のデータ・ファイルから SGA の共有データベース・バッファに読み込みます。
- アプリケーションが情報を処理できるような方法で結果を戻します。

バックグラウンド・プロセス

パフォーマンスを最大にし、多数のユーザーが使用できるように、マルチ・プロセス Oracle システムでは、この他に**バックグラウンド・プロセス**という複数の Oracle プロセスを使用します。

1 つの Oracle インスタンスは、多数のバックグラウンド・プロセスを持つことができます。ただし、常にすべてが存在するわけではありません。Oracle インスタンスのバックグラウンド・プロセスには、次のものがあります。

- データベース・ライター（DBW0 または DBW*n*）
- ログ・ライター（LGWR）
- チェックポイント（CKPT）
- システム・モニター（SMON）
- プロセス・モニター（PMON）
- アーカイバ（ARC*n*）
- リカバラ（RECO）
- ロック・マネージャ・サーバー（LMS）- Real Application Clusters のみ

- キュー・モニター（QMN*n*）
- ディスパッチャ（D*nnn*）
- サーバー（S*nnn*）

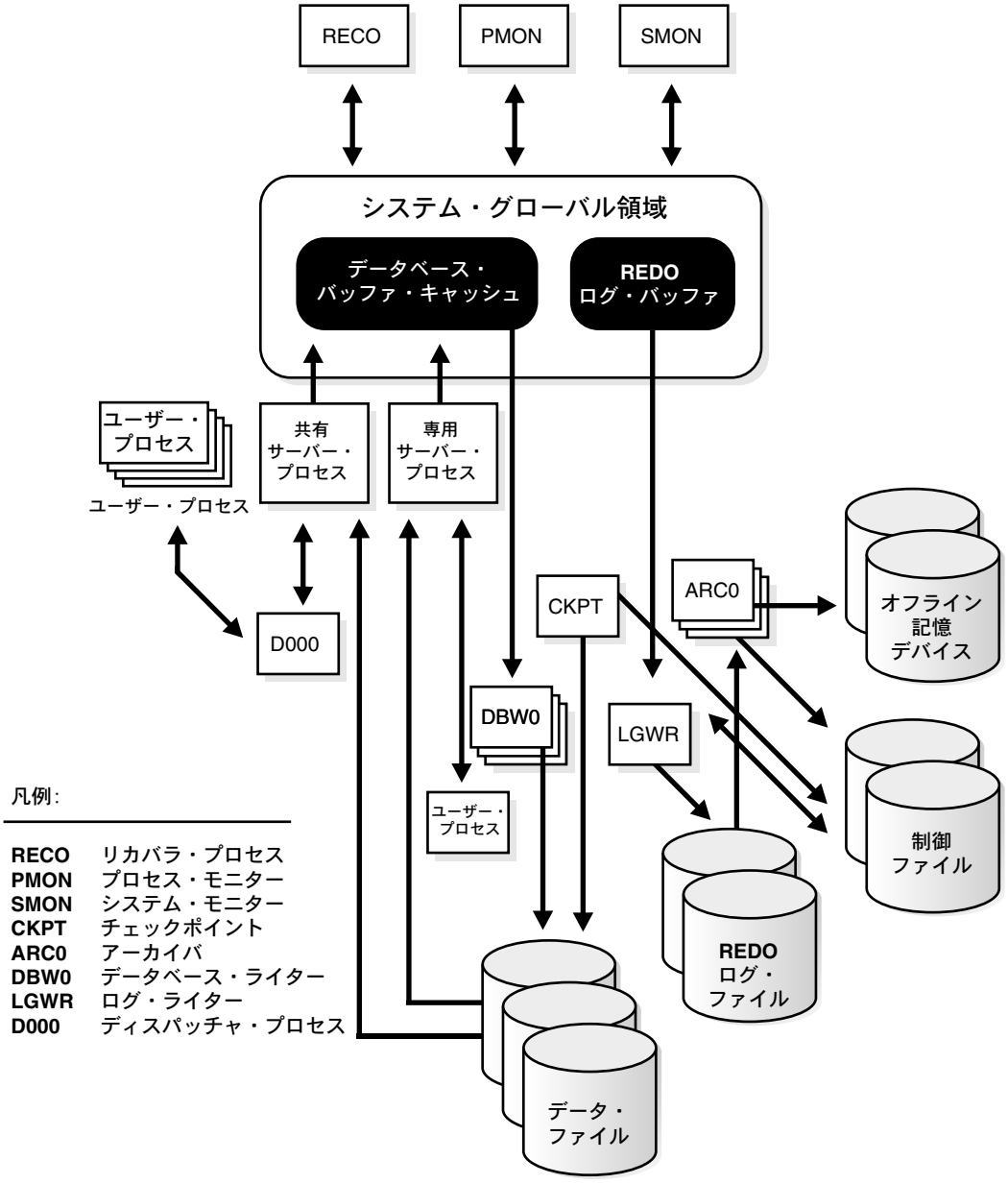
多くのオペレーティング・システムでは、インスタンスが起動するときに、バックグラウンド・プロセスが自動的に作成されます。

図 8-2 に、Oracle データベースの各部分とそれぞれのバックグラウンド・プロセスとの相互作用を示します。その後、各プロセスについて説明します。

関連項目：

- 詳細は、『Oracle9*i* Real Application Clusters 概要』を参照してください。図 8-2 には、Oracle9*i* 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 8-2 マルチ・プロセス Oracle インスタンスのバックグラウンド・プロセス



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（DBW*n*）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（使用済み）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**使用済み**のマークが付きます。**コールド**・バッファとは、LRU アルゴリズムに従って最近使用されたことがないバッファです。DBW*n* プロセスは、ユーザー・プロセスが新規ブロックをキャッシュに読み込むために使用できるクリーンなコールド・バッファを検出できるように、使用済みのコールド・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みの状態になると、使用可能バッファの数が減少します。使用可能バッファの数が少なすぎると、ディスクからキャッシュにブロックを読み込む必要があるユーザー・プロセスは、使用可能バッファを検出できません。ユーザー・プロセスが使用可能バッファを常に検出できるように、DBW*n* はバッファ・キャッシュを管理します。

使用済みのコールド・バッファをディスクに書き込むことにより、DBW*n* は、使用頻度の高いバッファをメモリー内に保ちながら、使用可能バッファを検索するときのパフォーマンスを向上させます。たとえば、頻繁にアクセスされる小さな表または索引の一部であるブロックは、それらをディスクから再び読み込まなくても済むように、キャッシュ内に置かれます。LRU アルゴリズムは、バッファの内容をディスクに書き込むときに、すぐに使用されそうなデータが書き込まれてしまわないように、より頻繁にアクセスされるブロックをバッファ・キャッシュ内に保持します。

DBW*n* プロセスの個数は、初期化パラメータ `DB_WRITER_PROCESSES` で指定します。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、CPU 数とプロセッサ・グループ数に基づいて `DB_BLOCK_PROCESSES` の設定が判断されます。

DBW*n* プロセスは、次のような場合に使用済みバッファをディスクに書き込みます。

- サーバー・プロセスがしきい値の数までバッファをスキャンしても、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み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み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：

- 7-7 ページ「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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- 『Oracle9*i* バックアップおよびリカバリ概要』

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* プロセスが修正済みのバッファをディスクに書き込むとき

注意： DBW*n* が修正済みバッファを書き込むには、バッファの変更に関連するすべての REDO レコードがディスクに書き込まれている必要があります（**事前書込みプロトコル**）。DBW*n* は、いくつかの REDO レコードが書き込まれていないことを見つけた場合に、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オンライン REDO ログ・ファイルのグループにも同時に書き込みます。グループ内のファイルのいずれかが破損している場合や使用できない場合、LGWR はそのグループ内の他のファイルへの書込みを続け、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・ファイルに記録します。グループ内のす

べてのファイルが破損している場合や、アーカイブされていないためにグループ全体が使用できない場合、LGWR は機能を続行できません。

ユーザーが COMMIT 文を発行すると、LGWR は REDO ログ・バッファ内にコミット・レコードを入れ、トランザクションの REDO エントリとともにそれを即時にディスクに書き込みます。対応するデータ・ブロックの変更は、それらをより効率的に書き込めるようになるまで延期されます。これを**高速コミット**・メカニズムと呼びます。トランザクションのコミット・レコードを含む REDO エントリのアトミックな書込みは、トランザクションがコミットされたかどうかを判別する 1 つのイベントです。Oracle は、データ・バッファがまだディスクに書き込まれていない場合でも、コミットしたトランザクションに成功コードを戻します。

注意： より多くのバッファ領域が必要な場合に、LGWR はトランザクションが コミットされる前に REDO ログ・エントリを書き込むこともあります。後でトランザクションがコミットされた場合にのみ、これらのエントリは確定します。

ユーザーがトランザクションをコミットすると、そのトランザクションには**システム変更番号（SCN）**が割り当てられます。Oracle は、このシステム変更番号を該当するトランザクションの REDO エントリとともに REDO ログに記録します。SCN は、Oracle9*i* 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**を使用してオンライン REDO ログ・ファイルに書き込むこともあります。たとえば、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オンライン REDO ログ・ファイルに書き込んでコミットすることはできません。最初のトランザクションのエントリをオンライン REDO ログ・ファイルに書き込んだ後、待機中の（まだコミットされていない）トランザクションの REDO エントリのリスト全体を 1 回の操作でディスクに書き込むことができるため、トランザクションのエントリを個々に処理するときよりも、必要となる I/O を低減できます。したがって、ディスク I/O は最小になり、LGWR のパフォーマンスは最大になります。コミットの要求が高い頻度で続くと、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：

- 7-11 ページ「REDO ログ・バッファ」
- 8-16 ページ「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、『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』を参照してください。
- SCN およびその使用方法の詳細は、『Oracle9*i* データベース管理者ガイド』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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

チェックポイント（CKPT）プロセス

チェックポイントが発生すると、すべてのデータ・ファイルのヘッダーはそのチェックポイントの詳細を記録するために更新される必要があります。この処理は、CKPT プロセスによって実行されます。CKPT プロセスは、ブロックをディスクに書き込みません。この書込みは、常に DBW*n* が実行します。

Enterprise Manager のシステム統計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： Real Application Clusters による CKPT の詳細は、『Oracle9*i* Real Application Clusters 管理』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み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、『Oracle9*i* Real Application Clusters 管理』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・プロセスの状態も定期的にチェックし、実行を停止したサーバー・プロセスがあれば再起動します（ただし、Oracle が意図的に終了させたプロセスは除きます）。また、PMON は、インスタンスおよびディスパッチャ・プロセスに関する情報をネットワーク・リスナーに登録します。

SMON と同じように、PMON は、処理が必要かどうかを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。

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、『Oracle9*i* データベース管理者ガイド』を参照してください。

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 インスタンスのプロシージャとして計画するのに使用できるスケジューラ・サービスとして表示できます。開始日付と間隔が設定されると、ジョブ・キュー・プロセスは、次に発生するジョブの実行に備えます。

Oracle9*i* からは、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、大量のジョブを一定の間隔で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・プロセスの実行内容を次に示します。

1. CJQ0 という名前のコーディネータ・プロセスは、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、動的にジョブ・キュー・スレーブ・プロセス（J000 ... J999）を起動して、ジョブを実行します。
3. ジョブ・キュー・プロセスは、CJQ プロセスが選択したジョブを実行します。ジョブ・キュー・プロセスは、一度に 1 つのジョブを実行します。
4. ジョブ・キュー・プロセスは 1 つのジョブの実行を終えると、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。ジョブ・キュー・プロセスが新規ジョブをまったく検出しなかった場合、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・プロセスがジョブ実行に使用可能であるとは想定しません。

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、『Oracle9*i* データベース管理者ガイド』を参照してください。

アーカイバ・プロセス（ARC*n*）

アーカイバ・プロセス（ARC*n*）は、ログ・スイッチの発生後、オンライン REDO ログ・ファイルを指定の記憶デバイスにコピーします。ARC*n* プロセスが存在するのは、データベースが ARCHIVELOG モードで、かつ、自動アーカイブが使用可能になっている場合のみです。

Oracle インスタンスは、最大 10 個の ARC*n* プロセス（ARC0 ～ ARC9）を持つことができます。LGWR プロセスは、現行の ARC*n* プロセス数ではワークロードを処理できなくなると、新しい ARC*n* プロセスを起動します。アラート・ファイルには、LGWR が新しい ARC*n* プロセスを開始した時刻が記録されます。

データのバルク・ロード中など、アーカイブに伴う大量の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 を使用して複数のアーカイバ・プロセスを指定できます。ALTER SYSTEM 文では、このパラメータの値を動的に変更し、ARC*n* プロセス数を増減させることができます。ただし、データベースのワークロードに対応しきれなくなると、必要な ARC*n* プロセス数がシステムによって自動的に判断され、

LGWR が追加の ARC*n* プロセスを自動的に起動するため、このパラメータをデフォルト値の 1 から変更する必要はありません。

関連項目：

- 8-16 ページ「トレース・ファイルとアラート・ログ」
- 『Oracle9*i* バックアップおよびリカバリ概要』
- 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。

ロック・マネージャ・サーバー（LMS）プロセス

Oracle9*i* Real Application Clusters のロック・マネージャ・サーバー（LMS）プロセスは、内部インスタンス・リソース管理を備えています。

関連項目： このバックグラウンド・プロセスの詳細は、『Oracle9*i* Real Application Clusters 概要』を参照してください。

キュー・モニター・プロセス（QMN*n*）

キュー・モニター・プロセス（QMN*n*）は、メッセージ・キューを監視する Oracle Advanced Queuing のオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。これらのプロセスは、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 バックグラウンド・プロセスと異なります。

関連項目： Oracle アドバンスト・キューイングの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

トレース・ファイルとアラート・ログ

それぞれのサーバーとバックグラウンド・プロセスは、対応付けられた**トレース・ファイル**に書き込むことができます。プロセスによって内部エラーが検出されると、エラーについての情報がそのプロセスのトレース・ファイルに書き込まれます。内部エラーが発生して情報がトレース・ファイルに書き込まれた場合、管理者はオラクル社カスタマ・サポート・センターに連絡してください。

バックグラウンド・プロセスに対応付けられているすべてのトレース・ファイルのファイル名には、そのトレース・ファイルを生成したプロセスの名前が含まれます。唯一の例外は、ジョブ・キュー・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。

各データベースには、`alert.log` もあります。データベースのアラート・ファイルは、次のようなメッセージとエラーの履歴ログです。

- 発生したすべての内部エラー（ORA-00600）、ブロック障害エラー（ORA-01578）およびデッドロック・エラー（ORA-00060）。
- 管理的な操作。たとえば、SQL 文の `CREATE/ALTER/DROP DATABASE/TABLESPACE/ROLLBACK SEGMENT` と、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 は、これらのイベントの記録を保管するのに、オペレータのコンソール上に情報を表示するかわりに、アラート・ファイルを使用します（多くのシステムではコンソール上にもこの情報が表示されます）。管理操作が成功すると、タイムスタンプとともに「completed」というメッセージがアラート・ファイルに書き込まれます。

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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- エラー・メッセージの詳細は、『Oracle9*i* データベース・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 ではなく Oracle Net Services の一部です）
- 1 つ以上のディスパッチャ・プロセス
- 1 つ以上の共有サーバー・プロセス

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には、ユーザー・プロセスは、Oracle インスタンスと同一のマシン上で実行されている場合でも、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 に接続するときの通信経路をオープンして確立します。その後、各ディスパッチャ・プロセスは、接続要求をリスニングするアドレスをリスナー・プロセスに割り当てます。データベース・クライアントで使用するネットワーク・プロトコルごとに、最低 1 つ以上のディスパッチャ・プロセスを構成し起動する必要があります。

ユーザー・プロセスが接続を要求すると、リスナーはその要求を調べてユーザー・プロセスが共有サーバー・プロセスを使用できるかどうかを決定します。使用できる場合には、リスナー・プロセスは負荷が最も軽いディスパッチャ・プロセスのアドレスを戻し、ユーザー・プロセスはディスパッチャに直接接続します。

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。

関連項目：

- 8-21 ページ「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、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

拡張性

Oracle 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拡張性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。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。

ディスパッチャの要求キューと応答キュー

ユーザーからの要求は、ユーザーの SQL 文の一部である単一のプログラム・インタフェース・コールです。ユーザーがコールすると、そのディスパッチャが要求を**要求キュー**に入れ、次に使用可能な共有サーバー・プロセスがそこから要求を取り出します。

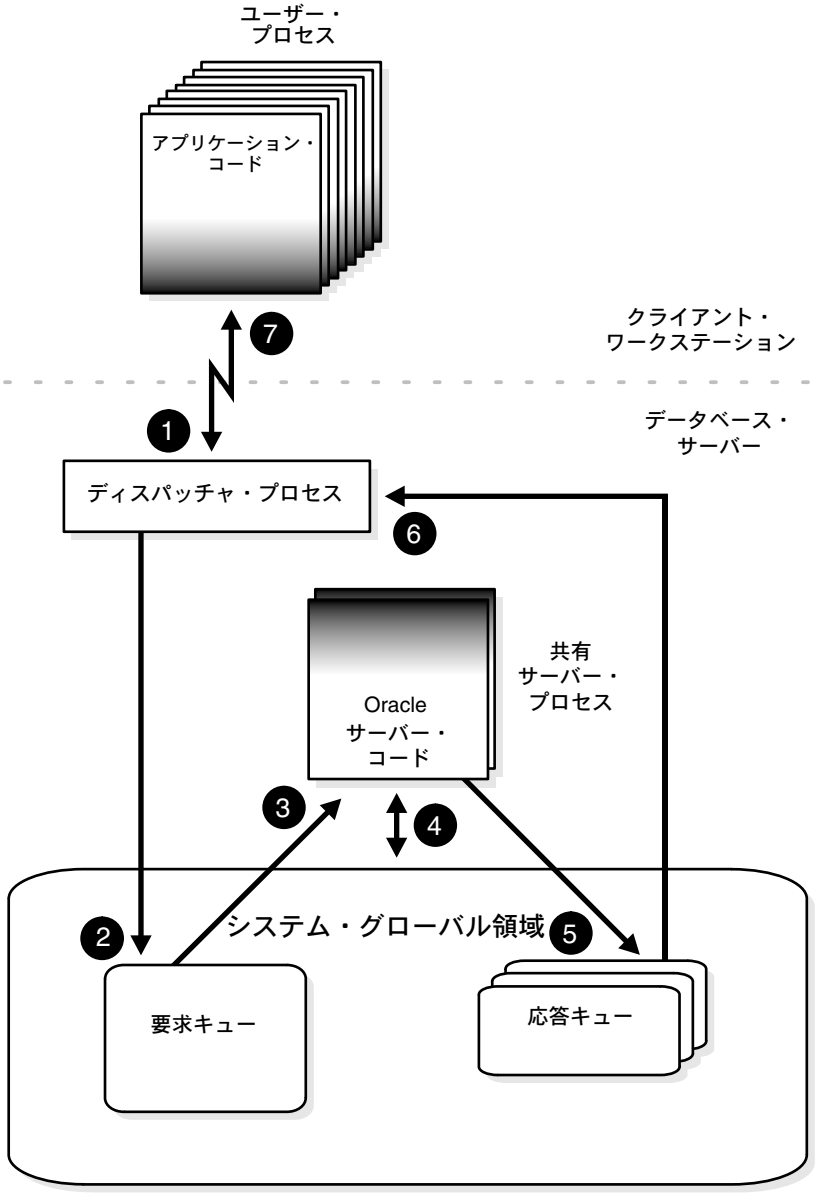
要求キューは SGA 内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。共有サーバー・プロセスは、共通の要求キューをチェックして新しい要求がないかどうかを調べ、先入れ先出し方式に基づいて新しい要求をピックアップします。1 つの共有サーバー・プロセスがキュー内の要求を 1 つピックアップし、その要求を完了するのに必要なデータベースに対するコールをすべて出します。

サーバーは要求を完了すると、コール・ディスパッチャの**応答キュー**に応答を入れます。各ディスパッチャには、SGA 内に固有の応答キューがあります。ディスパッチャは、完了した要求を適切なユーザー・プロセスに戻します。

たとえば、注文入力システムでは、それぞれの事務担当のユーザー・プロセスがディスパッチャに接続し、事務担当が出したそれぞれの要求がそのディスパッチャに送られ、そしてディスパッチャがその要求を要求キューに入れます。次に使用可能な共有サーバー・プロセスは、要求をピックアップして処理し、応答キューに応答を入れます。事務担当の要求が完了しても、事務担当はディスパッチャに接続されたままですが、その要求を処理した共有サーバー・プロセスは解放されるため、別の要求で使用できます。1 人の事務担当が顧客と話し合っている間に、別の事務担当は同じ共有サーバー・プロセスを使用できます。

図 8-3 に、ユーザー・プロセスがプログラム・インタフェースを介してディスパッチャと通信する方法と、ディスパッチャがユーザー要求を共有サーバー・プロセスに伝達する方法を示します。

図 8-3 共有サーバー構成と共有サーバー・プロセス



ディスパッチャ（D*nnn*）プロセス

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マシン上で稼働するようなクライアント / サーバー環境では、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 で使用する各ネットワーク・プロトコルごとに、少なくても 1 つのディスパッチャを作成する必要があります。データベース管理者は、プロセス当たりの接続数についてのオペレーティング・システムの制限に応じて、ディスパッチャ・プロセスを適切な数だけ起動する必要があります。また、インスタンスの実行中にディスパッチャ・プロセスを追加および削除できます。

注意： ディスパッチャに接続するそれぞれのユーザー・プロセスは、両方のプロセスが同一のマシン上で実行されている場合であっても、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 インスタンスの一部ではなく、Oracle とともに作動するネットワーキング・プロセスの一部です。

関連項目：

- 8-17 ページ「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、『Oracle9*i* 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。サーバーがどのセッションからの要求でも処理できるように、すべてのセッションのデータ領域に各共有サーバー・プロセスからアクセスできる必要があります。セッションのデータ領域ごとに、SGA 内に領域が割り当てられます。ユーザーのプロファイル内のリソース制限 `PRIVATE_SGA` を必要な領域の容量に設定すると、セッションが割り当てることのできる領域の容量を制限できます。

Oracle は、要求キューの長さに基づいて、共有サーバー・プロセスの数を動的に調整します。作成できる共有サーバー・プロセスの数は、初期化パラメータ `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、7-18 ページの「プログラム・グローバル領域（PGA）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 22 章「データベース・アクセスの制御」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`）。

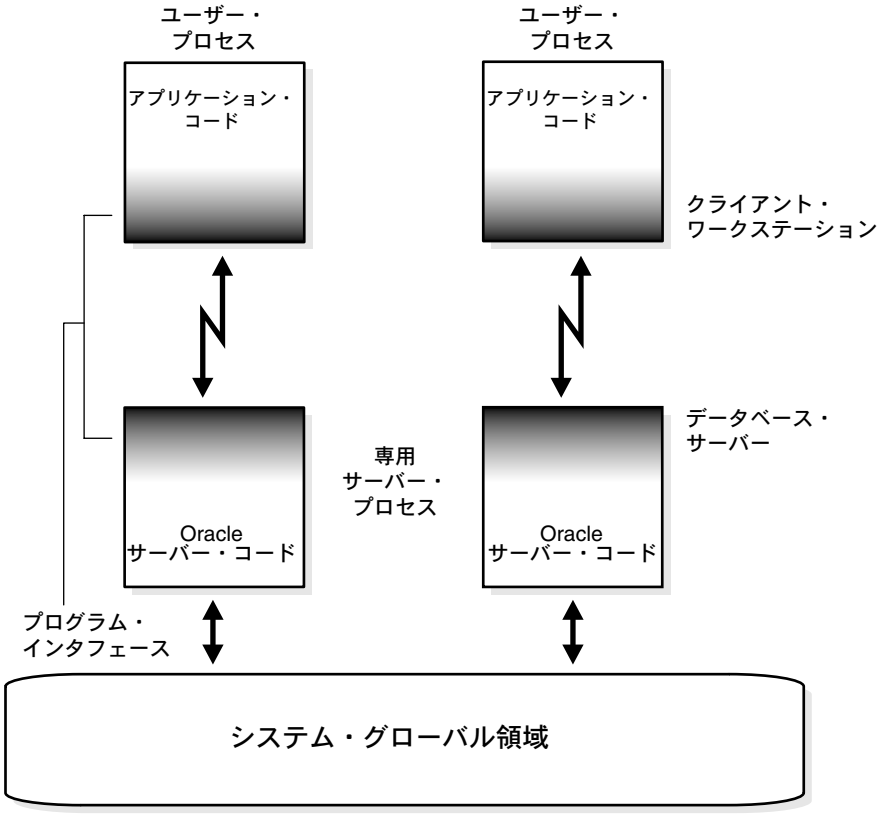
関連項目：

- オペレーティング・システム固有のマニュアルを参照してください。
- 正しい接続文字列の構文については、『Oracle9*i* 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 8-4 は、専用サーバー・アーキテクチャを使用し、2 台のコンピュータ上で稼働している Oracle を示しています。この構成では、データベース・アプリケーションが 1 台のマシン上のユーザー・プロセスによって実行され、対応付けられた Oracle サーバーがもう 1 台のマシン上のサーバー・プロセスによって実行されます。

図 8-4 専用サーバー・プロセスを使用する Oracle



ユーザー・プロセスとサーバー・プロセスは、相互に分離した別のプロセスです。ユーザー・プロセスごとに作成された別々のサーバー・プロセスは、このサーバー・プロセスが対応付けられたユーザー・プロセスのためにのみ活動するため、**専用サーバー・プロセス**（または**シャドウ・プロセス**）と呼ばれます。

この構成では、ユーザー・プロセス数とサーバー・プロセス数の比率が 1 対 1 に 維持されます。ユーザーが活発にデータベース要求をしていない場合でも、専用サーバー・プロセスはそのまま残ります（ただし、活動していない状態の場合、オペレーティング・システムによってはページアウトされることもあります）。

図 8-4 に、ネットワークを介して接続されている別々のコンピュータ上で実行されるユーザー・プロセスとサーバー・プロセスを示します。専用サーバー・アーキテクチャは、同じコンピュータ上でクライアント・アプリケーションと Oracle サーバー・コードの両方が実行される場合にも使用されますが、この 2 つのプログラムが 1 つのプロセスで実行されている場合は、ホスト・オペレーティング・システムはその 2 つの分離を維持できません。UNIX はそのようなオペレーティング・システムの一例です。

専用サーバー構成では、ユーザー・プロセスとサーバー・プロセスは異なるメカニズムを使用して通信します。

- ユーザー・プロセスと専用サーバー・プロセスを同じコンピュータで実行するようにシステムが構成される場合、プログラム・インタフェースは、ホスト・オペレーティング・システムのプロセス間通信メカニズムを使用してジョブを実行します。
- ユーザー・プロセスと専用サーバー・プロセスが別々のコンピュータ上で実行される場合、プログラム・インタフェースはプログラム間の通信メカニズム（ネットワーク・ソフトウェアや 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を参照してください。
  - 『Oracle9*i* Net Services 管理者ガイド』

  通信リンクの詳細は、次のマニュアルを参照してください。

プログラム・インタフェース

プログラム・インタフェースとは、データベース・アプリケーションと Oracle の間のソフトウェア・レイヤーです。プログラム・インタフェースには次のような機能があります。

- セキュリティ・バリアを実現し、クライアント・ユーザー・プロセスによる SGA への破壊的なアクセスを防ぎます。
- 通信メカニズムとして機能し、情報要求の書式設定、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 コードは、Oracle ソフトウェアとオペレーティング・システム固有のソフトウェアの両方によって提供されるいくつかの部分で構成されます。

プログラム・インタフェースの構造

プログラム・インタフェースは、次の要素から構成されます。

- Oracle Call Interface（OCI）または Oracle ランタイム・ライブラリ（SQLLIB）
- クライアント側またはユーザー側のプログラム・インタフェース（**UPI** とも呼ばれる）
- 各 **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
- サーバー側または Oracle 側のプログラム・インタフェース（OPI とも呼ばれる）

ユーザー側と Oracle 側の両方のプログラム・インタフェースは、どちらもドライバのような仕組みで Oracle ソフトウェアを実行します。

Oracle Net Services はプログラム・インタフェースの一部であり、これによってクライアント・アプリケーション・プログラムと Oracle サーバーが通信ネットワーク内の別々のコンピュータ上に常駐できるようになります。

プログラム・インタフェース・ドライバ

ドライバとは、通常はネットワークを介してデータを転送するソフトウェアの一部です。ドライバは、接続、切断、エラーの通知、エラーのテストなどの操作を実行します。ドライバは、通信プロトコルに固有であり、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 つのプロセスでも、異なる Oracle Net Services ドライバを使用して、1 つのデータベースまたは複数のデータベース（ローカルまたはリモートのどちらでも）に同時に接続できます。

関連項目：

- ドライバの選択、インストールおよび追加方法の詳細は、システムのインストレーションおよび構成ガイドを参照してください。
- Oracle へのアクセス中にドライバを実行時に選択する方法については、システムの Oracle Net Services マニュアルを参照してください。
- 『Oracle9*i* 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

ユーザー側のプログラム・インタフェースと、Oracle 側のプログラム・インタフェースを接続している最下位層のソフトウェアは、ホスト・オペレーティング・システムによって提供される通信ソフトウェアです。たとえば、DECnet、TCP/IP、LU6.2、ASYNC などです。通信ソフトウェアはオラクル社から提供されることもありますが、通常、ハードウェア・ベンダーまたはサード・パーティのソフトウェア・サプライヤから別途、購入します。

関連項目： システムの通信ソフトウェアの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

9

データベース・リソースの管理

この章では、複数のユーザー・グループへのリソースの割当てでデータベース管理者を支援する Oracle の Database Resource Manager の機能を説明します。この章の内容は、次のとおりです。

- Database Resource Manager の概要
- Database Resource Manager の機能
- リソース・プランとリソース・コンシューマ・グループ
- リソース割当て方法とリソース・プラン・ディレクティブ
- オペレーティング・システムのリソース制御との相互作用

Database Resource Manager の概要

従来より、Oracle データベースも含み、システム上で稼働する各アプリケーション間のリソース管理の調整は、オペレーティング・システムに任されています。Oracle8*i* 以下のリリースでは、ある Oracle のセッションを他のセッションに優先させる方法はありませんでした。Database Resource Manager を使用することで、データベース管理者は企業のビジネス目標に沿ったリソース割当てを実現するために、リソース管理における意思決定をより一層制御できるようになります。

注意： Oracle8*i* から、Database Resource Manager は、Oracle Enterprise Edition で使用できます。

Database Resource Manager 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 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。

Database Resource Manager を使用して、データベース管理者は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。
- 様々なユーザーとアプリケーションに、比率を指定して CPU タイムを割り当てて、使用可能な処理リソースを分配します。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し、より多くの比率が指定されます。
- ユーザーが所属するグループのメンバーが実行する操作の並列度に上限を設定します。
- **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されるまでキューに格納されますが、タイムアウト周期を指定できるので、タイムアウト後にはキューに格納されたジョブは終了されます。

- 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。ユーザーが所属する特定のグループのメンバーが、指定した時間より長く実行されるセッションを作成した場合、そのセッションは、リソース要件の異なる、ユーザーが所属する別のグループに自動的に切り替えられます。
- 事前定義した上限より長い時間実行されると予測される操作の実行を防止します。
- **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。

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。

> **関連項目：** Database Resource Manager の使用方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

Database Resource Manager の概要

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・グループにより、管理者は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、次の点でユーザー・ロールとは異なります。複数のリソース・コンシューマ・グループに割り当てられた複数のセッションを所有できるのは、1 人のデータベース・ユーザーです。

リソース割当て方法により、特定のリソースを割り当てる際に使用する方針を定めます。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にパラメータを指定することで、コンシューマ・グループに特定のプランを割り当てたり、コンシューマ・グループ間のリソースを分割する手段です。

Database Resource Manager では、サブプランと呼ばれるプランをプラン内に作成できます。**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。

簡単なリソース・プランの例

これらの概念を説明するために、架空の会社、ABC 社の例を取り上げます。ABC 社はインターネットを介して家庭用電化製品を販売しています。オンラインの顧客に対し最高のパフォーマンスを提供するために、CPU リソースの少なくとも 85% は、顧客用に割り当てます。残りのリソースの内、10% は出荷オーダーを処理するユーザーに、5% は請求処理をするユーザーに割り当てます。

このようなリソース割当てを Oracle データベースに構成するために、データベース管理者は、次の 3 つのリソース・コンシューマ・グループを作成します。

- オンライン顧客用の ONLINE
- 出荷担当ユーザー用の SHIPPING
- 請求担当ユーザー用の BILLING

データベース管理者は、次に表 9-1 に示すようなリソース・プランを作成します。

表 9-1 簡単なリソース割当てプラン（ABCUSERS）

コンシューマ・グループ	CPU リソースの割当て
ONLINE	85%
SHIPPING	10%
BILLING	5%

表 9-1 に示されたプランでは、CPU サイクルの 85% が ONLINE グループ・セッションに割り当てられ、10% が SHIPPING グループ・セッション、残りの 5% が BILLING グループ・セッションに割り当てられるように指定されています。ここでは非常に簡単な使用例で説明していますが、Database Resource Manager は、制御可能なリソース割当ての方針を Oracle データベースに実装するための強力なメカニズムをデータベース管理者に提供します。

> **関連項目：** これらのプランを作成するための PL/SQL コードについては、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

Database Resource Manager の機能

Database Resource Manager 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、Database Resource Manager 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）。

より多くの CPU リソースが割り当てられているコンシューマ・グループに属しているセッションは、割当ての少ないグループやサブプランに属しているセッションよりも CPU をより長く使用できます。

注意： UNIX プラットフォームでは、オペレーティング・システムのプロセス実行順位を変更するのに、**`nice`** 文は使用しないでください。この文を使用すると、Oracle サーバーの動作が不安定かつ予測できなくなることがあります。詳細は、9-18 ページの「オペレーティング・システムのリソース制御との相互作用」を参照してください。

リソース制御

Database Resource Manager の基本的な目的は、ビジネス目標に合わせ、システムのスループットを最大限に高めることにあります。したがって、コンシューマ・グループに必要なリソースが確保されているかぎり、CPU 割当て比率の制限が適用されることはありません。

リソース制御の例

表 9-1 のプランを考えてみます。他のグループにおいて、割り当てられたリソースをすべて使用するようなアクティブなセッションが存在しない場合に、CPU が 1 つのシステムでこのプランがアクティブ化されると、1 つのコンシューマ・グループで CPU リソースをすべて（100%）使用することができます。したがって、SHIPPING グループと BILLING グループ内にアクティブなセッションが存在しない場合には、ONLINE グループのセッションは、割当ての上限が 85% に設定されている場合でも、CPU リソースの 100% を使用できます。

同様に、データベースが 3 つの CPU からなるシステムで稼働していて、それぞれのグループのアクティブなセッションが 1 つのみの場合は、各セッションは 3 つの CPU 内のいずれかで実行されます。この場合、割当て上限の設定にかかわらず、33.33% のリソースが割り当てられます。ただし、すべてのコンシューマ・グループに、使用可能な CPU リソースのすべてを使用するようなアクティブなセッションが存在する場合、Database Resource Manager により、プラン・ディレクティブで指定する割当ての方針が規定されます。

Database Resource Manager の有効性

Database Resource Manager の効果は、システム使用率が高いビジーな環境において顕著です。

マルチプロセッサ・システムでは、オペレーティング・システム・レベルでのプロセッサのアフィニティ・スケジューリングにより、十分に利用されていないシステムの CPU 割当てが正しく行われないことがあります。複数の CPU を持つシステムにおいて、1 つの CPU に使用可能なリソースがあり、他の CPU の使用率が 100% の場合、オペレーティング・システムは、ビジーなプロセッサの実行キューから十分に利用されていないプロセッサへのプロセスの移行を行います。ただし、この移行が即時実行されるわけではありません。

多くのプロセスがある負荷の大きいシステムでは、プロセッサのアフィニティによりパフォーマンスが向上します。現在の CPU キャッシュを無効にして、新しいキャッシュを搭載するのには多額の費用がかかるためです。ほとんどのプラットフォームではプロセッサのアフィニティがサポートされているため、多くのプロセスの実行ではシステムを 100% 利用します。

データベースの整合性

Database Resource Manager は、データベース・セキュリティ・システムに完全に統合されています。PL/SQL パッケージ DBMS_RESOURCE_MANAGER により、データベース管理者はリソース・プランとリソース・コンシューマ・グループの作成、更新および削除ができます。管理者は、ユーザーのデフォルト・コンシューマ・グループおよびユーザーが持つ権限を定義します（DBMS_RESOURCE_MANAGER_PRIVS パッケージを使用）。ユーザーにそのコンシューマ・グループへ切り替える権限が付与されている場合、ユーザーまたはセッションはリソース・コンシューマ・グループを切り替えて（DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP を使用）、実行の優先順位を変更できます。さらに、ユーザーまたはセッションは本番システムのデータベース管理者によりグループ間を移動でき、CPU リソースの使用方法が動的に変更されます。

各アプリケーションのユーザーが複数のデータベース・ユーザー名を使用してデータベースにログオンする環境で、Database Resource Manager を使用するのは非常に簡単です。アプリケーションが汎用データベース・ログインを使用する環境に実装することも、さほど困難ではありません。Database Resource Manager は、実際にはセッション・レベルでのリソース使用率を制御するため、両方のセッションが同じデータベースのユーザーに属している場合でも、セッション間で優先順位を付けることができます。このため、次に示すように、ユーザーのアプリケーション・ロールのためにあるセッションを希望するコンシューマ・グループへ切り替えることも、DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP プロシージャを使用することで実現できます。

```
DECLARE default_group VARCHAR2(30);
BEGIN DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('desired_consumer_group', 'default_
group', false);
END; /
```

Oracle は Database Resource Manager を使用して、ユーザーのリソース制限とプロファイルに対するサポートを継続します。Database Resource Manager により、様々なサービス要求と定義したリソース割当てプラン内のそれぞれとの間が均衡化されます。一方、プロファイルは、ユーザーのリソース使用を制限するのに使用されます。

Database Resource Manager と自動並列度（ADOP）機能は統合されています。ADOP は、現在のシステムの負荷および Database Resource Manager の並列度ディレクティブを基準にパラレル問合せ操作の並列度を自動調整することで、システム使用率を最適化します。

パフォーマンスのオーバーヘッド

Database Resource Manager では、最小限のオーバーヘッドでリソース管理が効率的に行えます。ユーザーが数百人のシステムでは、Database Resource Manager はコンテキストの切替えおよびラッチの競合を削減することにより、パフォーマンスを実際に改善することができます。

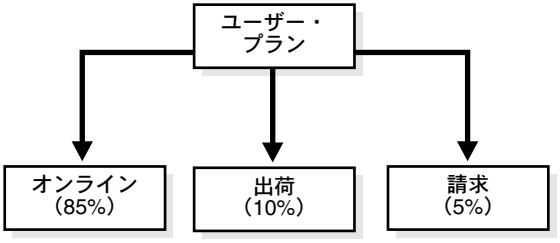
- Database Resource Manager は、オペレーティング・システムのフェア・シェア・スケジューラが行うような頻度でプロセスを切り替えることはありません。
- Database Resource Manager では、同時に実行されるプロセスはより少なく、ラッチを保持しているプロセスのコンテキストの切替えは行いません。

Database Resource Manager を適切に使用すると、パフォーマンスが低下することはありません。ただし、リソース・プランの深さまたは複雑度により、実行する処理を選択するプロセスが妨げられることもあります。したがって、深すぎるリソース・プランは避けることをお薦めします。プラン・スキーマがより多くのレベルを持つと、Database Resource Manager は実行するセッションを選択するために、より多くの作業を行う必要があります。

リソース・プランとリソース・コンシューマ・グループ

リソース・プランとは、リソース・コンシューマ・グループの集合をグループ化して、グループ間でリソースを分割する方法を指定する手段です。次のように図示化される表 9-1 の例を考えてみます。

図 9-1 ABC のリソース割当てプラン（ABCUSERS）



リソース・プランのアクティブ化

データベース内に、必要な数のリソース・プランを作成できます。ただし、一度にアクティブ化できるのは 1 プランのみです。持続と動的のいずれかの方法で、リソース・プランをアクティブ化できます。

持続

RESOURCE_MANAGER_PLAN初期化パラメータの値をアクティブ化するプランに設定します。たとえば、ABC 社の簡単なリソース・プランである abcusers をアクティブ化するには、初期化パラメータ・ファイルを変更して次の行を含めます。

```
RESOURCE_MANAGER_PLAN='ABCUSERS'
```

初期化パラメータ・ファイルを変更するときは、停止したデータベースでのリソース・プランの持続性を確認します。ただし、初期化パラメータ・ファイルにおける変更が有効になるのは、データベースが再起動されるときのみです。この方法を使用してデータベースのデフォルトのリソース・プランを設定します。

動的

ALTER SYSTEM SET RESOURCE_MANAGER_PLAN 文を発行します。同じ例を使用して、次の文を発行します。

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN ='ABCUSERS';
```

ALTER SYSTEM SET RESOURCE_MANAGER_PLAN 文を発行すると、インスタンスの再起動も必要なく、指定したプランはすぐにアクティブ化されます。ただし、データベースは、次に起動するときに、初期化パラメータ・ファイルのデフォルト設定に戻されます。

たとえば、動的方法の使用では、管理者は業務時間用と夜間用の 2 種類のプランを作成できます。業務時間用のプランでは、オンライン・ユーザーに多くのリソースを割り当てますが、夜間用のプラン（オンライン・ユーザーがアクティブではないとき）では、バッチ・ジョブに多くのリソースを割り当てるようにします。このため、データベース管理者は、ALTER SYSTEM 文を使用して、業務時間および夜間のプランの間をデータベース・サービスを中断することなく行き来できます。

ALTER SYSTEM SET RESOURCE_MANAGER_PLAN 文は、リソース・プランを動的にアクティブ化する場合、変更または解除する場合に使用します。

関連項目： 9-19 ページの「オペレーティング・システムのリソース制御との相互作用」の手順 3 を参照してください。

リソース・プランのグループ

リソース・プランを使用して他のリソース・プランをグループ化することもできます。これにより、種類の異なるアプリケーション間でのリソースの分割が可能になります。たとえば、ABC 社では開発者および管理者が重要なメンテナンス操作を行えるように、最小限のリソースを確保する必要がある場合があります。

DEVELOPERS と ADMINISTRATORS の 2 つのリソース・コンシューマ・グループに属するセッション用に、少なくとも 25% の使用可能な CPU サイクルの確保が必要であると考えてみます。これらの CPU サイクルを、DEVELOPERS と ADMINISTRATORS の比率が 60 対 40 となるように割り当てます。この目標を達成するために、データベース管理者は次の仕様のメンテナンス・プランを最初に作成します。

表 9-2 メンテナンス・プランのサンプル（ABCMAINT）

コンシューマ・グループ	CPU リソースの割当て
DEVELOPERS	60%
ADMINISTRATORS	40%

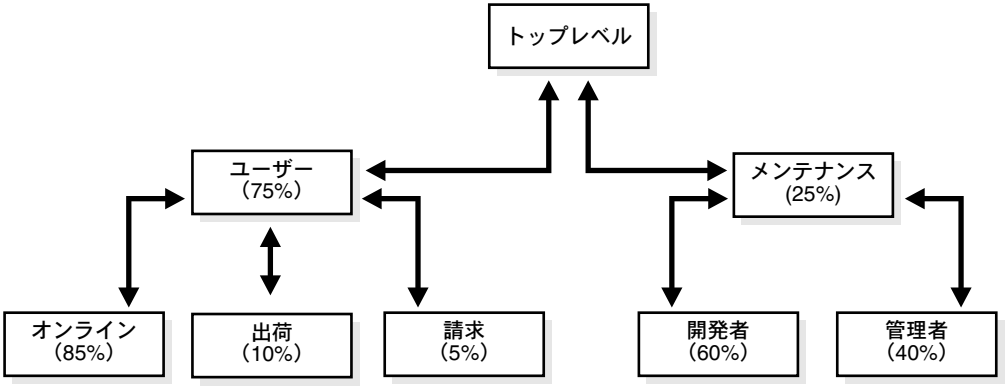
表 9-2 に示されるプランがアクティブ化されたとき、使用可能なすべてのリソースが、開発者および管理者のセッションに 60 対 40 の比率で分配されていることを確認します。ただし、メンテナンス用に確保されるのは使用可能なすべてのリソースの 25% のみで、75% は abcusers プラン用です。これらは、表 9-3 に示すように、abcusers と abcmaint をそのメンバーとするトップレベルのプランを作成することで実現できます。

表 9-3 トップレベルのプラン（ABCTOP）

サブプラン	CPU リソースの割当て
ABCUSERS	75%
ABCMAINT	25%

ユーザー・グループとメンテナンス・グループは、ABCTOP のサブプランとなります。結果として作成されたプラン・ツリーを図 9-2 に図示します。

図 9-2 サブプランを含んだリソース・プラン



サブプランまたはコンシューマ・グループは、複数の親を持つことができます。たとえば、図 9-2 に示すプランで、コンシューマ・グループの管理者は、ユーザー・プランおよびメンテナンス・プランの両方の一部とすることも可能です。複数の親が許可されているため、プランを独立して確保することもできます。サブプランをトップレベルのプランにロールアップする場合、サブプランの変更は必要ありません。

プラン・ツリーには、必要に応じていくらでも階層レベルを設定できます。ただし、階層レベルの数が増加するにつれ、リソース制御に関連するオーバーヘッドが増加します。このため、次に実行するプロセスの判別は、サブプランを含むすべてのレベルごとに行う必要があります。一方、サブプランでは、高水準のデータベース・リソースが複数アプリケーション間で適切に分割され、アプリケーション内ではそれらが各種ユーザー間で再分割されます。アプリケーション内の特定のグループが割当てを使用しなかった場合、同じアプリケーション内の他のグループが未使用のリソースを最初に使用できます。アプリケーション内のいずれのグループも使用可能になったリソースを使用しなかった場合には、未使用のリソースは親プランに戻され、親プランが戻されたリソースをサブプラン間に分配します。

> **関連項目：** これらのプランを作成するための PL/SQL コードについては、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

リソース割当て方法とリソース・プラン・ディレクティブ

Database Resource Manager では、割当て方法およびプラン・ディレクティブを使用して、コンシューマ・グループ間（グループ間）およびコンシューマ・グループ内（グループ内）でのリソースの分配を制御できます。

- プランレベル・リソース割当て方法およびプランレベル・リソース・ディレクティブにより、コンシューマ・グループ間でのリソースの割当て方法を指定します。
- コンシューマ・グループ方法およびコンシューマ・グループ・ディレクティブは、コンシューマ・グループに属するセッション間でのリソースの分配を制御します。

Database Resource Manager は、セッション実行のスケジューリングを、次の手順で行います。

1. リソース割当ての方針とディレクティブを計画して、次に実行するコンシューマ・グループを決定します。
2. グループレベル割当て方法とグループレベル・ディレクティブにより、CPU の実行キューに送るセッションを、選択したグループ内から選択します。

たとえば、表 9-1 に示す ABC 社のユーザー・プランでは、プランレベル方法とプランレベル・ディレクティブによりリソース分配が指定され、ONLINE コンシューマ・グループ・セッションには 85% の実行時間が、SHIPPING および BILLING グループに属するセッションにはそれぞれ 10%、5% の CPU タイムが割り当てられます。

ABCUSERS のプランレベル・ディレクティブにより、ONLINE グループは SHIPPING および BILLING グループに比べより頻繁にピックアップされ実行できることが保証されます。ただし、ONLINE グループには実行待ちのアクティブなセッションがいくつか存在するのが普通です。このため、グループレベルのディレクティブによりこれらのセッションを実行する順序が決められます。

リソース・プラン・ディレクティブ

リソース・コンシューマ・グループへのリソースの割当て方法は、リソース割当てディレクティブに指定します。Database Resource Manager は、リソース割当ての手段をいくつか備えています。

CPU 方法

この方法では、コンシューマ・グループまたはサブプラン間に CPU リソースを割り当てる方法を指定します。CPU リソースを複数レベルに割り当てることで（8 レベルまで）、プラン・スキーマ内の CPU 使用に優先順位を付ける手段が与えられます。レベル 1 がすべてのリソースを使用できない場合のみ、レベル 2 はリソースを使用できます。複数レベルのリソース割当てでは、優先順位が付けられるだけでなく、プライマリ・リソースおよび残りのリソースの使用方法を明示的に指定できます。

キューイングを伴うアクティブなセッションのプール

コンシューマ・グループ内で許可される、同時にアクティブなセッションの最大数を制御できます。この最大数でアクティブなセッションのプールが指定されます。プールがいっぱいになっているためにセッションが開始できない場合は、そのセッションはキューに置かれます。アクティブなセッションが完了すると、キュー内の 1 番目のセッションが実行されます。実行キュー内のジョブ（実行待ちの）がタイムアウトになった後に、エラーとして終了させるタイムアウト周期も指定できます。

パラレル実行セッション全体が、1 つのアクティブなセッションとして数えられます。

並列度の上限

並列度の上限を指定することで、コンシューマ・グループ内の操作に対する最大並列度を制御できます。

コンシューマ・グループの自動切替え

この方法では、ある条件を満たした場合に、自動的に別のコンシューマ・グループにセッションが切り替わるような基準を指定することでリソースを制御します。切替えの判断に使用する基準を次に示します。

- SWITCH_GROUP - 別の（次の）基準を満たした場合に、このセッションを切り替えるコンシューマ・グループを指定します。
- SWITCH_TIME - 別のコンシューマ・グループに切り替えられる前にセッションが実行できる時間を指定します。
- SWITCH_ESTIMATE - 操作の実行時間を推定するのに、Oracle 独自の見積りを使用するのかどうかを指定します。

セッションが SWITCH_TIME 秒以上アクティブな場合、Database Resource Manager は稼働中のセッションを SWITCH_GROUP に切り替えます。アクティブとはセッションが稼働中でリソースを使用していることを意味していて、ユーザーからの入力や CPU サイクルを待機していることは意味しません。新しいグループ用のアクティブなセッションのプールがいっぱいの場合でも、セッションの稼働を継続できます。このような条件では、コンシューマ・グループは、アクティブなセッションのプールで指定した数より多いセッションを稼働できます。セッションが操作を終了してアイドルになると、そのセッションは元のグループに再び切り替えられます。

SWITCH_ESTIMATE が true に設定されている場合、Database Resource Manager は、操作完了にかかる時間の予測値を使用します。Oracle の予測値が SWITCH_TIME として指定した値より長い場合、Oracle はセッションの実行が開始される前にそのセッションを切り替えます。このパラメータが設定されていない場合は、通常どおりに操作が開始され、他の切替え基準が満たされた場合にのみグループが切り替えられます。

実行時間の上限

操作に許可される最大実行時間を指定できます。ある操作の実行時間が、指定した最大実行時間より長いと Oracle が予測した場合、その操作はエラーとして終了させられます。このエラーはトラップされ、操作が再開されます。

UNDO プール

各コンシューマ・グループごとに UNDO プールを指定できます。UNDO プールは、コンシューマ・グループが生成できる UNDO の合計数を制御します。コンシューマ・グループが生成した UNDO の合計が UNDO の上限を超えた場合、REDO を生成する現在の DML 文は終了させられます。UNDO 領域がプールから解放されるまでは、コンシューマ・グループの他のメンバーはすべて、データ操作を実行できません。

CPU リソースの割当て

データベース管理者は 8 レベルまでのリソース割当てに対しリソースを与え、これらのレベルのそれぞれのコンシューマ・グループ間でのリソース分配方法を指定することで、競合するコンシューマ・グループ内のセッション間でのリソースの分配を制御できます。

表 9-1 に示すユーザー・プランは、単一レベルのリソース割当てを使用する簡単なリソース分配方法を表しています。未使用のリソースを最初に管理者に割り当てて（メンテナンス操作用に）、次にバッチ・ジョブに割り当てるようにこのプランを変更できます。表 9-4 に、変更後のプランを示します。

表 9-4 マルチレベル・ユーザー・プラン 1

コンシューマ・グループ	CPU_LEVEL1	CPU_LEVEL2	CPU_LEVEL3
ONLINE	85%	0%	0%
SHIPPING	10%	0%	0%
BILLING	5%	0%	0%
ADMIN	0%	100%	0%
BATCH	0%	0%	100%

表 9-4 に示す変更後のユーザー・プランでは、次のことが実現できます。

- CPU_LEVEL1 によって、CPU リソースの少なくとも 85% が ONLINE に属するセッションで使用可能であり、SHIPPING および BILLING コンシューマ・グループではそれぞれ 10%、5% を使用できることが保証されます。
- CPU_LEVEL2 は、ONLINE、SHIPPING および BILLING で使用されなかったリソースを ADMIN コンシューマ・グループに提供します。
- CPU_LEVEL3 は、残りのリソースを BATCH コンシューマ・グループで使用できるようにします。

マルチレベル・ユーザー・プラン 1 は、レベルの異なるコンシューマ・グループ間へリソースを与えることで設定した目標を満たします。ONLINE、SHIPPING および BILLING グループに属するセッションに対しては、常に最初に実行する機会が与えられますが、リソースの使用はそれぞれ 85%、10% および 5% に制限されます。未使用のリソースはレベル 2 で利用され、レベル 2 で許可される割当て内でコンシューマ・グループ間に分配されます。未使用のリソースがまだ残っている場合は、次の下位レベルで利用できます。

ただし、レベル 1 のすべてのグループがビジーの場合には、ADMIN グループはかなり待たされることもあります。同様に、レベル 1 とレベル 2 のグループがすべてのリソースを使用した場合、BATCH グループはセッションをまったく実行できなくなることもあります。このような動作は、環境によっては受け入れられません。CPU リソースのほとんどを ONLINE、SHIPPING および BILLING に割り当てて、かつ ADMIN および BATCH グルー

プへも最小限の CPU サイクルの可用性を保証するようなプランこそ本当に必要とされます。変更後のマルチレベル・ユーザー・プランを表 9-5 に示します。

表 9-5 マルチレベル・ユーザー・プラン 2

コンシューマ・グループ	CPU_LEVEL1	CPU_LEVEL2	CPU_LEVEL3
ONLINE	75%	0%	0%
SHIPPING	10%	0%	0%
BILLING	5%	0%	0%
ADMIN	0%	80%	0%
BATCH	0%	20%	0%

表 9-5 に示す変更後のマルチユーザー・プランでは、次のことが実現できます。

- CPU_LEVEL1 は、今回、使用可能な CPU タイムの 75% までを ONLINE グループに割り当てて、SHIPPING および BILLING にはそれぞれ 10%、5% を割り当てます。
- CPU_LEVEL2 は、残りの 10% の CPU タイムを ADMIN および BATCH グループの間で 80 対 20 の比率に分割します。これにより、ADMIN グループのセッションは、使用可能なすべての CPU タイムの少なくとも 8%（10% の内の 80%）が与えられることが保証されます。また、BATCH グループのセッションには少なくとも 2% が与えられます。

マルチレベル・ユーザー・プラン 2 では、最小限、CPU リソースの 10% を ADMIN および BATCH グループに保証します。ONLINE、SHIPPING および BILLING グループに割り当てられたリソースが余った場合は、これらレベル 2 グループには、より多くの CPU タイムが割り当てられます。レベル 1 グループにアクティブなセッションが存在しない場合、ADMIN グループのセッションは CPU タイムの 80% を、BATCH グループのセッションは CPU タイムの 20% をそれぞれ実行できます。

CPU 割当てルール

表 9-4 および表 9-5 に示すマルチレベル・ユーザー・プランでは、強調方法による CPU リソースの割当てが一連のルールに従うことを説明しています。これらのルールを次に示します。

1. レベル 1 で 0% 以外のパーセンテージが指定されているリソース・コンシューマ・グループのセッションには、常に最初に実行する機会が与えられます。
2. CPU リソースは、指定したパーセンテージに基づいた特定のレベルで分配されます。
   - あるコンシューマ・グループに割り当てられたリソースが使用されなかった場合、未使用のリソースは、同じレベルの他のグループではなく、次のレベルに割り当てられます。
   - 特定レベルのすべてのリソース・コンシューマ・グループに実行機会が与えられた後は、残っているリソースが次のレベルに割り当てられます。

3. どのレベルの強調度の合計も、100 以内にする必要があります。
4. コンシューマ・グループのすべてのレベルに CPU タイムが割り当てられた後、アクティブでないことや割当て制限のために未使用となった CPU タイムはリサイクルされます。すなわち、この CPU タイムは、レベル 1 で開始されるコンシューマ・グループに再度割り当てられます。
5. プラン・ディレクティブが明示的に指定されていないレベルの場合（たとえば、レベル 3）、すべてのサブプランまたはコンシューマ・グループについて暗黙的に 0% になります。

Database Resource Manager は、任意の時点で、アクティブなセッションがあるグループ間に CPU リソースを割り当てます。実行させるグループを決める場合、履歴情報は使用しません。たとえば、ONLINE コンシューマ・グループにアクティブなセッションが 2 時間存在しなかったとします。この期間は、ONLINE コンシューマ・グループのリソースは他のコンシューマ・グループで使用可能です。その後、ONLINE コンシューマ・グループに属するセッションがアクティブになったとしても、それらに割り当てられるのは CPU リソースの 75% のみです。コンシューマ・グループは、アクティブなセッションが存在しなかった期間のクレジットを蓄積しません。

レベルと優先順位

レベルは優先順位に類似しています。レベル 1 のコンシューマ・グループには、下位レベルのコンシューマ・グループの前にリソースが割り当てられます。表 9-6 は、Database Resource Manager のプラン・ディレクティブを使用して優先順位を設定する方法の説明です。

表 9-6 簡単な優先順位プラン

サブプランまたはグループ	CPU_LEVEL1	CPU_LEVEL2	CPU_LEVEL3
高優先順位	100%	0%	0%
標準優先順位	0%	100%	0%
低優先順位	0%	0%	100%

表 9-6 で、標準優先順位グループまたはサブプランに属するセッションは、高優先順位グループまたはサブプラン内にアクティブなセッションが存在しなくなった場合にのみ、実行できます。同様に、低優先順位のセッションは、高優先順位グループ、標準優先順位グループまたはサブプランのいずれかに属するアクティブなセッションが存在しなくなった場合にのみ、実行する機会が与えられます。

ただし、表 9-6 に示すようなプランでは、リソースが不足する可能性があります。高優先順位グループが CPU リソースの 100% を使用するかぎり、標準優先順位グループまたは低優先順位グループに属するセッションは、実行できません。すなわち、高優先順位グループに属する一連のリソース集中型のセッションは、標準優先順位グループまたは低優先順位グループに属するセッションの処理を完全にストールさせます。

これを避けるには、相対優先順位に応じた最小限のリソースが確実にすべてのコンシューマ・グループに割り当てられるようにプランを作成します。たとえば、表 9-6 のプランを次のように変更できます。

表 9-7 変更後の優先順位プラン

サブプランまたはグループ	CPU_LEVEL1	CPU_LEVEL2	CPU_LEVEL3
高優先順位	80%	0%	0%
標準優先順位	10%	0%	0%
低優先順位	10%	0%	0%

表 9-7 の変更後のプランによると、高優先順位グループにほとんどの CPU タイムが割り当てられた場合でも、他のグループが完全に停止することはありません。

オペレーティング・システムのリソース制御との相互作用

Oracle9*i* は静的構成を見込んで、データベースの起動時に検出された使用可能なリソースからラッチなどの内部リソースを割り当てます。そのため、リソース構成が頻繁に変更されるとデータベースが最適な状態で実行されないこともあり、不安定になります。したがって、オペレーティング・システムのリソース制御を Oracle データベースとともに使用する場合は、次の指針に従って正しく使用します。

1. オペレーティング・システムのリソース制御を Database Resource Manager と同時に使用しないでください。これらは、両方ともお互いの存在を意識していません。このため、オペレーティング・システムおよび Database Resource Manager の両者は、Oracle データベースに対し、予測できないような動作および不安定性をもたらす方法でリソース割当てを制御しようとします。
   - インスタンス内のリソース分配を制御する場合は、Database Resource Manager を使用して、オペレーティング・システムのリソース制御をオフにします。
   - ノード上に複数のインスタンスがあり、それらにリソースを分配する場合は、Database Resource Manager ではなく、オペレーティング・システムのリソース制御を使用します。

   注意： Oracle9*i* では、両ツールの同時使用はサポートしていません。将来のリリースでは、一定の制限のもとで両者の対話が許可されるようになります。

2. Oracle の環境では、次の条件がすべて満たされた場合にのみ、HP 社のプロセス・リソース・マネージャ（PRM）または Sun 社の Solaris Resource Manager（SRM）などのオペレーティング・システムのリソース・マネージャの使用がサポートされます。
   - 各インスタンスを専用のオペレーティング・システムのリソース・マネージャ・グループまたは専用の管理エンティティに割り当てる必要があります。
   - インスタンスのすべてのプロセスが稼働する専用のエンティティは、1 つの優先順位（またはリソース使用）レベルで実行する必要があります。
   - プロセスの優先順位管理を使用可能にしないでください。

注意： 優先順位レベルの異なる個々の Oracle プロセスの管理（たとえば、UNIX プラットフォーム上での **nice** 文の使用）はサポートされていないことに注意してください。これらの使用により、インスタンスのクラッシュなどの重大な障害が発生することがあります。Oracle のインスタンスが確保されたメモリーの管理がオペレーティング・システムのリソース制御に許可された場合、好ましくない同じような障害が発生することがあります。

3. オペレーティング・システムのリソース制御の使用を選択した場合には、Database Resource Manager を必ずオフにしてください。デフォルトでは、Database Resource Manager はオフに設定されています。オフに設定されていない場合は、次の文を発行することでオフにできます。

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN='';
```

データベースが次回起動されたときに、Database Resource Manager がアクティブ化されないように、初期化パラメータ・ファイル内のこのパラメータも必ずリセットします。

動的な再構成

Sun 社の Processor Set および Dynamic System Domain のようなツールは、Oracle データベースとの協調動作に優れています。CPU の数を変更しても、インスタンスを再起動する必要はありません。

Oracle は使用可能な CPU 数の変化を動的に検出し、内部リソースを再割当てします。ほとんどのプラットフォームでは、Oracle は CPU_COUNT の値を使用可能な CPU の数に合せて自動的に調整します。

関連項目： CPU_COUNT の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

第 IV 部

データ

第 IV 部では、データベース管理に必要なデータについて説明します。

第 IV 部には、次の章が含まれています。

- 第 10 章「スキーマ・オブジェクト」
- 第 11 章「パーティション表とパーティション索引」
- 第 12 章「システム固有のデータ型」
- 第 13 章「オブジェクト・データ型とオブジェクト・ビュー」

10

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
- 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。

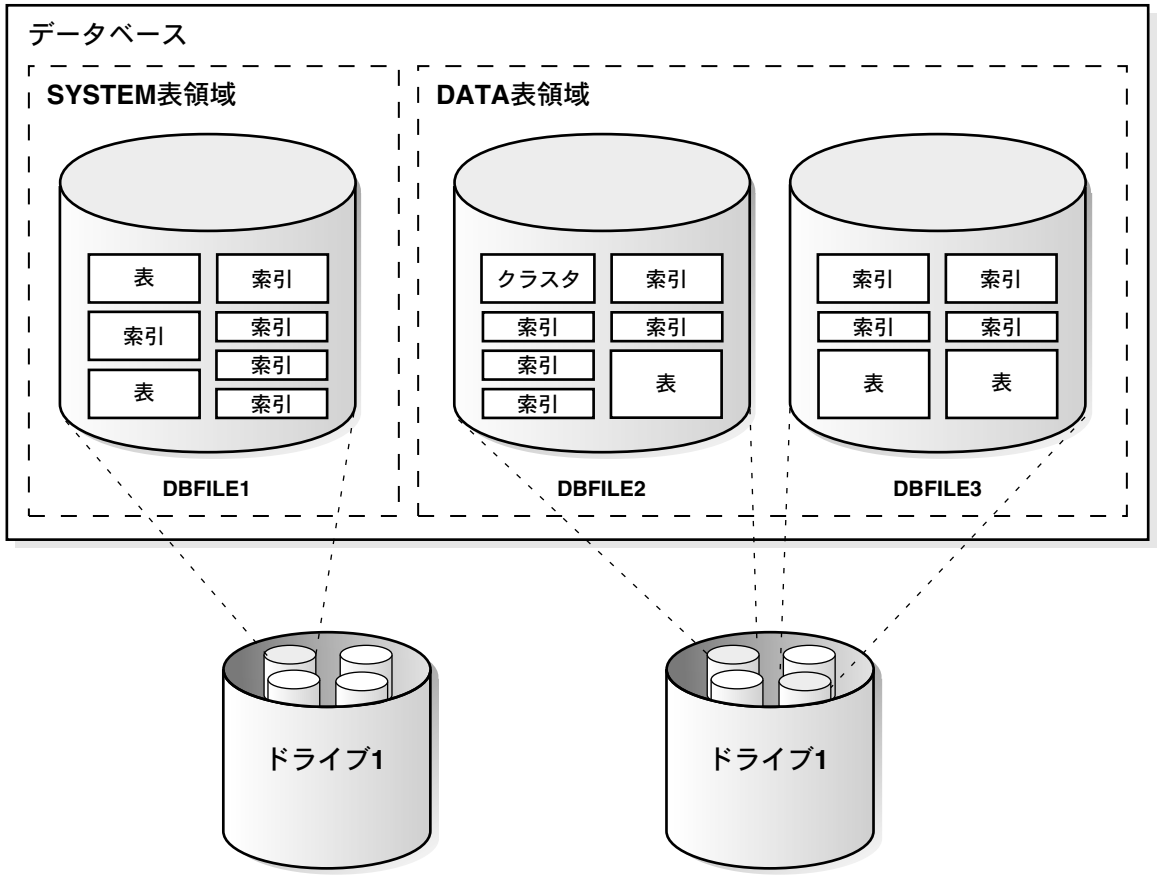
- コンテキスト
- ディレクトリ
- プロファイル
- ロール
- 表領域
- ユーザー
- ロールバック・セグメント

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 では対応しませんが、データベースの表領域内に論理的に格納されます。各オブジェクトのデータは、物理的には、表領域の 1 つ以上のデータ・ファイルに格納されます。表、索引およびクラスタなど、いくつかのオブジェクトについては、表領域のデータ・ファイル内のオブジェクトに対して割り当てるディスク領域の容量を指定できます。

スキーマと表領域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 10-1 に、オブジェクト、表領域およびデータ・ファイルの関連を示します。

図 10-1 スキーマ・オブジェクト、表領域およびデータ・ファイル



関連項目：

- 『Oracle9*i* データベース管理者ガイド』
- 14-20 ページ「ストアド・プロシージャとファンクション」
- 第 14 章「SQL、PL/SQL および Java」
- 第 17 章「トリガー」

表

表は、Oracle データベースにおけるデータ記憶域の基本単位です。データは**行**と**列**に格納されます。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。行は、1 つのレコードに対応する列情報の集まりです。

表の各列に対してルールを指定できます。これらのルールのことを**整合性制約**と呼びます。たとえば、整合性制約の 1 つに NOT NULL 整合性制約があります。この制約により、どの行の列にも必ず値が入るようになります。

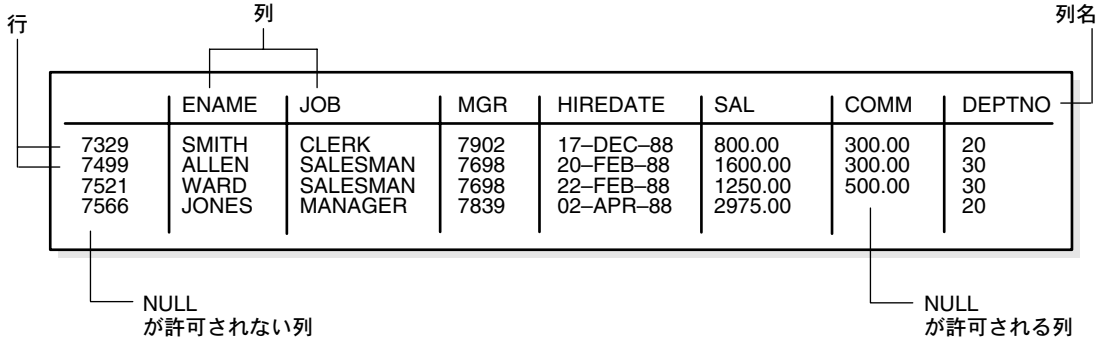
表の作成後に、SQL 文を使用してデータ行を挿入します。挿入した表データは、SQL を使用して問合せ、削除または更新できます。

図 10-2 に、`EMP` という名前のサンプル表を示します。

関連項目：

- Oracle データ型の詳細は、第 12 章「システム固有のデータ型」を参照してください。
- 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。

図 10-2 EMP 表



表データの格納方法

表の作成時に、表の将来のデータを保持するために、データ・セグメントが表領域内に自動的に割り当てられます。表のデータ・セグメントに対する領域の割当てと使用は、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。

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

クラスタ化されていない表のデータ・セグメントが含まれる表領域は、表所有者のデフォルト表領域または CREATE TABLE 文で明確に指定した表領域です。

関連項目： 22-15 ページ「ユーザー表領域の設定と割当て制限」

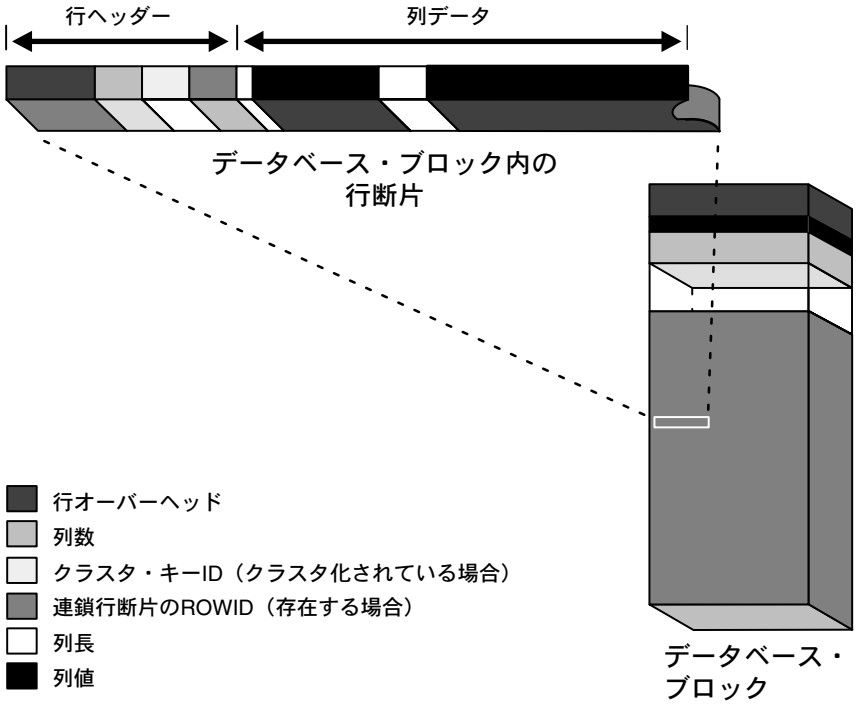
行の形式とサイズ

255 列以下のデータを含むデータベース表の各行は、1 つ以上の行断片として格納されます。1 つの行全体を単一のデータ・ブロックに挿入できる場合、その行は 1 つの行断片として格納されます。ただし、1 行のデータを単一のデータ・ブロックには挿入できない場合や、既存の行の更新によって行がデータ・ブロックのサイズを超えてしまう場合は、複数の行断片を使用して行が格納されます。1 つのデータ・ブロックには、通常、各行に 1 つの行断片のみが格納されます。1 つの行を複数の行断片に格納する必要がある場合は、複数のブロックにわたって行が**連鎖**されます。

表に 256 以上の列が含まれる場合、256 列目以降のデータを含む行は、通常同じブロック内に連鎖されます。このような処理を**ブロック内連鎖**と呼びます。連鎖行の各断片は、それぞれの断片の ROWID を使用して連鎖されます。ブロック内連鎖により、ユーザーは同じブロック内のすべてのデータを受け取ります。行がブロックに収まる場合、行の残りの部分を取得する特別の I/O 操作は必要ないため、ユーザーが I/O パフォーマンスへの影響を目にすることはありません。

それぞれの行断片は、連鎖されていても連鎖されていなくても、行の全部または一部の列の**行ヘッダー**とデータを含みます。また、個々の列が複数の行断片にまたがり、結果として複数のデータ・ブロックにまたがっている場合もあります。図 10-3 に、行断片の形式を示します。

図 10-3 行断片の形式



行ヘッダーは、データの前に位置し、次のような情報を含んでいます。

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

1 つのブロックに完全に収まる行は、最低 3 バイトの行ヘッダーを持っています。各行は、行ヘッダー情報の後に列の長さ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。NULL 列のデータは格納されません。また、末尾にある NULL 列の場合は、列の長さも格納されません。

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

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

関連項目：

- クラスタ化された行や表の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- 10-62 ページ「クラスタ」
- 2-7 ページ「行連鎖と移行」
- 10-9 ページ「値がないことを意味する NULL」
- 2-5 ページ「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。ROWID が行断片に割り当てられると、対応する行の削除や、エクスポート／インポート・ユーティリティによるエクスポートまたはインポートが実行されるまで、行断片はその ROWID を保持します。クラスタ化表の場合、ある行のクラスタ・キー値が変化しても、その行はそれまでと同じ ROWID を保持しますが、同時に新しい値に関する追加のポインタ ROWID も取得します。

ROWID は、行断片の存続期間中は一定であるため、SELECT、UPDATE および DELETE などの SQL 文で ROWID を参照すると便利です。

関連項目：

- 10-62 ページ「クラスタ」
- 12-18 ページ「物理 ROWID」

列の順序

列の順序は、表内のすべての行について同一です。列は、通常、CREATE TABLE 文にリストした順序で格納されますが、そのことが保証されているわけではありません。たとえば、LONG データ型の列を持つ表を作成する場合、Oracle は常にこの列を最後に格納します。また、表を変更して新しい列を追加すると、その新しい列は、最後に格納されます。

一般的には、行に必要な領域を少なくするために、NULL を頻繁に格納する列を最後の列にします。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。

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 制約または主キー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の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 10-31 ページ「索引と NULL」
- 10-50 ページ「ビットマップ索引と NULL」

列のデフォルト値

新しい行が挿入され、列に対する値が省略されたか、またはキーワードの 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ができます。列のデフォルト値は、実際に INSERT 文が値を指定している場合と同様に機能します。

デフォルトのリテラルまたは式のデータ型は、その列のデータ型に一致しているか、あるいはそれに変換可能である必要があります。

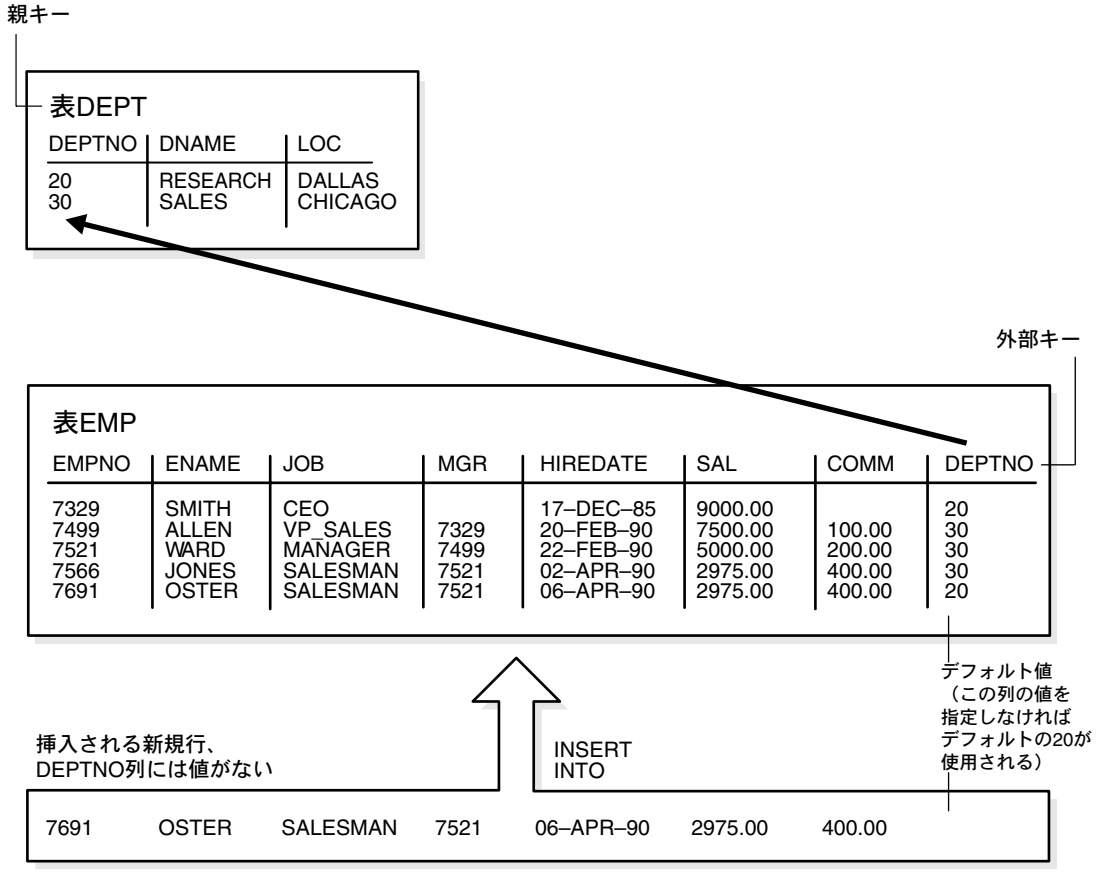
列に対してデフォルト値が明示的に定義されていない場合、その列のデフォルトは暗黙的に NULL に設定されます。

デフォルト値の挿入と整合性制約のチェック

デフォルト値を持つ行が挿入されると、整合性制約チェックが行われます。たとえば、図 10-4 では、従業員の部門番号に対する値を持たない行が、EMP 表に挿入されています。部門番号に値が指定されていないため、DEPTNO 列のデフォルト値 20 が挿入されます。デフォルト値の挿入後に、DEPTNO 列に定義されている外部キー制約がチェックされます。

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

図 10-4 デフォルトの列値



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することができます。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。

関連項目： 第 11 章「パーティション表とパーティション索引」

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 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：

- 13-11 ページ「ネストした表の説明」
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

一時表

Oracle では、永続的な表だけでなく、トランザクションまたはセッションの期間中にのみ存在するセッションのプライベート・データが保持される**一時表**を作成できます。

CREATE GLOBAL TEMPORARY TABLE 文では、トランザクション固有またはセッション固有の一時表が作成されます。トランザクション固有の一時表では、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。

CREATE INDEX 文を使用すると、一時表の索引を作成できます。一時表に作成された索引と、その索引のデータは、一時表のデータと同じセッションまたはトランザクション・スコープを持ちます。

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

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

セグメントの割当て

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

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

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

関連項目： 2-12 ページ「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。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 は、Oracle の列型を示す部分と外部データの Oracle データ列へのマッピングを説明する部分（アクセス・パラメータ）の 2 つの部分で構成されています。

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、外部表レイヤーがデータをサーバーに提示する際の方法が示されます。データ・ファイルのデータに必要な変換を行って外部表の定義に合わせるのは、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。

アクセス・ドライバ

データベース・サーバーは外部ソースのデータにアクセスする必要がある場合、適切なアクセス・ドライバをコールして、外部ソースから望ましい構成のデータを取得します。Oracle は、ファイルのデータにアクセスする際の要件のほとんどを満たすデフォルトのアクセス・ドライバを提供します。

重要なことは、データ・ソースにあるデータの説明は外部表の定義とは別であるという点です。ソース・ファイルのフィールドは、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。また、データ・ソース内のフィールドのデータ型は、表内の列と異なるものにすることもできます。アクセス・ドライバはデータ・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します。

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、『Oracle9*i* データベース管理者ガイド』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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- インポートおよびエクスポートの詳細は、『Oracle9*i* データベース・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、『Oracle9*i* SQL リファレンス』を参照してください。

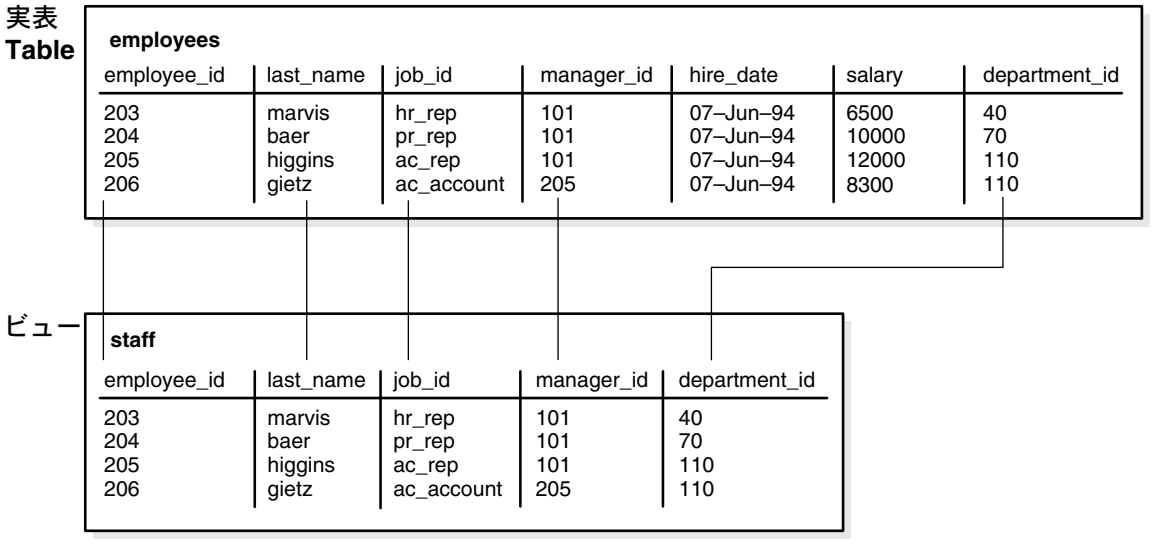
ビュー

ビューとは、1 つ以上の表または他のビューに含まれているデータを調整して表現したものです。ビューは問合せの出力を受け取り、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 10-5 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 10-5 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。

注意： 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 では、ビューの論理的制約の定義をサポートしています。

関連項目：『Oracle9*i* SQL リファレンス』

ビューの格納方法

表と違い、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。様々なユーザーに合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 10-5 は、`staff` ビューが実表 `employees` の列 `salary` と `commission_pct` をどのように隠すかを示しています。

- データの複雑さを隠します。

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

- ユーザー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 ビューおよび UNION ビューの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ビューのメカニズム

ビューの定義は、ビューを定義する問合せのテキストとしてデータ・ディクショナリに格納されます。SQL 文でビューを参照すると、Oracle では次の操作が実行されます。

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

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

ビューのグローバリゼーション・サポート・パラメータ

文字列リテラルが含まれているビューや、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）を評価する場合、Oracle 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り出します。これらのデフォルト値は、ビューの定義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・サポートの詳細は、『Oracle9*i* 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 がビューに対する問合せで索引を使用するかどうかは、元の問合せがビュー定義の問合せとマージされるときにどのように変換されるかによって決まります。

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, departments
    WHERE employees.department_id = departments.department_id AND
                        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 によって作成される最終的な問合せは次のようになります。

```
SELECT 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 はビューに対する問合せを、そのビューを定義している問合せおよび基礎となるビューの問合せとマージします。マージされた問合せは、ビューを参照せずに発行された場合と同じように最適化されます。そのため、列がビュー定義で参照されても、またはビューに対するユーザーの問合せで参照されても、参照される実表の列に対する索引を使用できます。

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

> **関連項目：** 問合せの最適化の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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 は、ビューの依存性を自動的に処理します。たとえば、ビューの実表が削除された後で再作成された場合、Oracle は、新しい実表がそのビューの既存の定義と一致するかどうかを判断します。

> **関連項目：** データベースの依存性の詳細は、第 15 章「スキーマ・オブジェクト間の依存性」を参照してください。

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、その他）も使用しないビューと定義されます。

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 句および 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。詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

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：**
>
> - 更新可能ビューの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
> - 17-12 ページ「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：

- 第 13 章「オブジェクト・データ型とオブジェクト・ビュー」
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

インライン・ビュー

インライン・ビューは、スキーマ・オブジェクトではありません。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。

たとえば、次の問合せでは、集計表 SUMTAB を TIME 表で定義されているインライン・ビュー V に結合して T.YEAR を取得し、SUMTAB の集計を YEAR レベルにロールアップしています。

```
SELECT v.year, s.prod_name, SUM(s.sum_sales)
    FROM sumtab s,
        (SELECT DISTINCT t.month, t.year FROM time t) v
    WHERE s.month = v.month
    GROUP BY v.year, s.prod_name;
```

関連項目： 副問合せの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

マテリアライズド・ビュー

マテリアライズド・ビューは、データの集計、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。

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

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

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

関連項目：

- 10-28 ページ「索引」
- 第 11 章「パーティション表とパーティション索引」
- 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、『Oracle9*i* 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 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 では、ビューを制約できます。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 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 では、マテリアライズド・ビューがマスター表の変更後にリフレッシュされ、そこに含まれるデータがメンテナンスされます。リフレッシュ方法には、増分リフレッシュ（**高速リフレッシュ**）または完全リフレッシュがあります。高速リフレッシュ方法を使用する場合、マスター表に対する変更は**マテリアライズド・ビュー・ログ**または**ダイレクト・ローダー・ログ**に記録されます。

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

マテリアライズド・ビュー・ログ

マテリアライズド・ビュー・ログとは、マスター表に定義されているマテリアライズド・ビューの増分リフレッシュを実行できるように、マスター表の変更を記録するスキーマ・オブジェクトです。

マテリアライズド・ビュー・ログは、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、『Oracle9*i* 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、『Oracle9*i* レプリケーション』を参照してください。

ディメンション

ディメンションは、1 対の列または列セット間の階層（親子）関係を定義します。子レベルの値は、それぞれが親レベルの 1 つの値にのみ対応付けられます。階層関係は、ある階層レベルから次のレベルへの**機能上の依存関係**です。ディメンションは、列相互の論理関係のコンテナで、それにはデータ記憶域は割り当てられません。

CREATE DIMENSION 文では、次の事項を指定します。

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

ディメンション内の列は、同じ表のもの（**非正規化**）でも複数の表のもの（**完全正規化**または**一部正規化**）でもかまいません。複数の表からの列によるディメンションを定義するには、HIERARCHY 句の JOIN 句を使用して表を接続します。

たとえば、正規化されている time ディメンションには、date 表、month 表および year 表と、各 date 行を month 行に、各 month 行を year 行に接続する結合条件を含めることができます。完全な非正規化の time ディメンションでは、date、month および year 列はすべて同じ表に格納されます。正規化されているかどうかに関係なく、列相互の階層関係を CREATE DIMENSION 文で指定する必要があります。

関連項目：

- ウェアハウス環境におけるディメンションの使用方法については、『Oracle9*i* 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については、『Oracle9*i* SQL 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

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ル化（2 つのトランザクションの文が連続番号を同時に生成する必要がある状態）が低減されます。このシリアル化を避けることによって、トランザクションのスループットが改善され、ユーザーの待ち時間が大幅に短縮されます。

順序番号とは、データベース内で定義される最大 38 桁の整数です。順序の定義には、次のように一般的な情報を指定します。

- 順序の名前
- 昇順または降順
- 数値の増分値
- 生成される順序番号の集合をメモリー内にキャッシュするかどうか

データベースごとにすべての順序定義が、SYSTEM 表領域内の単一のデータ・ディクショナリ表に行として格納されます。したがって、SYSTEM 表領域は常時オンラインであるため、すべての順序定義がいつでも使用できます。

順序番号は、順序を参照する SQL 文によって使用されます。文を発行して、新しい順序番号を生成することも、カレント順序番号を使用することもできます。ユーザーのセッション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、カレント順序番号にアクセスできます。

順序番号は表からは独立して生成されます。そのため、同じシーケンス・ジェネレータを複数の表に対して使用できます。順序番号の生成は、データに対して一意の主キーを自動的に生成する場合や、複数の行または表にまたがるキーを統合する場合に有効です。最終的にロールバックされたトランザクションで生成されて使用された個々の順序番号はスキップされます。必要に応じて、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： すべての順序番号に対するアカウンタビリティが必要な場合、つまりアプリケーションが決して順序番号を紛失してはならない場合は、Oracle 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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- `CREATE SEQUENCE` 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

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、プロシージャ、ファンクションまたはパッケージに対する別名です。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶域は必要ありません。

シノニムは、セキュリティと便利さのために使用されます。たとえば、次のような利点があります。

- オブジェクトの名前と所有者を隠します。
- 分散データベースのリモート・オブジェクトの位置の透過性を実現します。
- データベース・ユーザー側の SQL 文を単純化します。
- 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。

ここで、データベースのユーザーが、次のような SQL 文を使用して SALES_DATA 表を問い合せます。

```
SELECT * FROM jward.sales_data;
```

問合せを実行するには、表を含むスキーマと表の名前の両方を含める必要があることに注意してください。

データベース管理者が次の SQL 文によって、パブリック・シノニムを作成するとします。

```
CREATE PUBLIC SYNONYM sales FOR jward.sales_data;
```

パブリック・シノニムが作成されると、ユーザーは、次のような単純な SQL 文で SALES_DATA 表を問い合せることができます。

```
SELECT * FROM sales;
```

パブリック・シノニム SALES は、表 SALES_DATA の名前とその表を含むスキーマ名を隠していることに注意してください。

索引

索引は、表とクラスタに対応付けるオプションの構造体です。表の 1 つ以上の列に索引を作成し、その表に対する SQL 文の実行を高速化できます。このマニュアルでも、索引を使用すると特定の情報をすばやく見つけることができます。同様に、Oracle の索引を使用すると表データに高速にアクセスできます。適切に使用すると、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限り、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);
```

表の 1 列のみを参照する索引がすでに存在している場合、この種の索引は作成できません。

Oracle は、パフォーマンスの機能性を補足する複数の索引体系を提供しています。

- B 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・ジョイン・インデックス

また、アプリケーションまたはカートリッジ固有のファンクション・ベース索引やドメイン索引もサポートしています。

索引の有無によって、SQL 文の表現を変更する必要はありません。索引は、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 によって自動的にメンテナンスされ、使用されます。行の新規追加、更新または削除など、データに対する変更は、関連するすべての索引に自動的に反映され、ユーザーによる操作は不要です。

新たにいくつかの行が挿入されても、索引付きのデータ検索のパフォーマンスはほとんど一定です。ただし、表に対して数多くの索引が存在すると、更新、削除および挿入のパフォーマンスは低下します。これは、表に関連する索引も更新する必要があるためです。

オプティマイザは、既存の索引を使用して別の索引を作成できます。この結果、索引作成がはるかに高速になります。

一意索引と非一意索引

索引には、一意索引と非一意索引の 2 種類があります。一意索引によって、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、表の一意制約を有効にして作成するのではなく、明示的に作成することをお薦めします。

そのかわりに、必要な列に対して一意制約を定義できます。Oracle は、一意キーに対して自動的に一意索引を定義して、一意制約を規定します。ただし、一意索引を含む、問合せのパフォーマンスのために存在する索引は、明示的に作成することをお薦めします。

> **関連項目：** 一意索引を明示的に作成する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

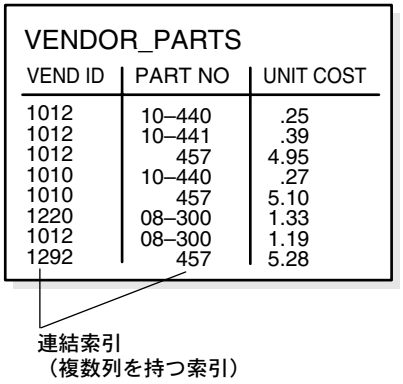
コンポジット索引

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

図 10-6 コンポジット索引の例



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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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、整合性制約に対応しています。これは、データベースのビジネス・ルールを規定する Oracle のもう 1 つの機能です。

Oracle 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見なされます。ただし、索引内に NULL でない同一の値を持つ行が 2 行以上存在する場合は、それらの行は同一と見なされます。したがって、一意索引では、NULL 値を含む行は同一のものとして扱われなくなります。この規則は、NULL 以外の値が存在しない場合、つまり、行全体が NULL の場合には適用されません。

表のうち、すべてのキー列が NULL の表の行には索引は設定されません。ただし、ビットマップ索引の場合や、クラスタ・キーの列値が NULL の場合は例外です。

関連項目： 10-50 ページ「ビットマップ索引と NULL」

ファンクション・ベース索引

表に索引を設定する場合、その表の 1 つ以上の列を含むファンクションと式について、索引を作成できます。**ファンクション・ベース索引**では、ファンクションや式の値が計算され、索引に格納されます。ファンクション・ベース索引は、B ツリーまたはビットマップ索引として作成できます。

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 ファンクション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・ベース索引を作成できません。また、オブジェクト型に LOB、REF またはネストした表が含まれる場合も作成できません。

関連項目：

- 「ビットマップ索引」
- ファンクション・ベース索引の使用方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

ファンクション・ベース索引の使用方法

ファンクション・ベース索引は、WHERE 句にファンクションを含む文を効率的に評価するメカニズムを提供します。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、文を処理するために従来どおりファンクションを評価する必要があります。

たとえば、次の索引を作成する場合を考えます。

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
```

これにより、次のような問合せを処理するときに、この索引を使用できます。

```
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

UPPER(*column_name*) または LOWER(*column_name*) でファンクション・ベース索引を定義すると、大文字 / 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・ベース索引は、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、『Oracle9*i* 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・ベース索引による最適化

オプティマイザを使用するには、ファンクション・ベース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。ルールベース最適化には、ファンクション・ベース索引は使用されません。

コストベース最適化では、WHERE 句の式による問合せに、ファンクション・ベース索引に対する索引レンジ・スキャンを使用できます。たとえば、次の問合せでは、

```
SELECT * FROM t WHERE a + b < 10;
```

a+b の索引が作成されていれば、オプティマイザは索引レンジ・スキャンを使用できます。レンジ・スキャンのアクセス・パスは、述語（WHERE 句）の選択性が低い場合に特に便利です。また、式がファンクション・ベース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・ベース索引を比較して、式のマッチングを実行します。この比較は大文字 / 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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

ファンクション・ベース索引の依存性

ファンクション・ベース索引は、その索引を定義している式に使用されたファンクションに依存しています。ファンクションが PL/SQL 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・ベース索引の定義には、PL/SQL ファンクション DETERMINISTIC を使用する必要があります。索引所有者には、定義するファンクションに対する EXECUTE 権限が必要です。EXECUTE 権限が取り消されると、ファンクション・ベース索引に DISABLED マークが設定されます。

関連項目：

- PL/SQL ファンクション DETERMINISTIC の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。
- ファンクション・ベース索引の依存性と権限の詳細は、15-8 ページの「ファンクション・ベース索引の依存性」を参照してください。

索引の格納方法

索引を作成すると、索引のデータを保持するための索引セグメントが表領域内に自動的に割り当てられます。索引セグメントへの領域の割当てと、確保された領域の使用は、次の方法で制御できます。

- 索引セグメントのエクステントの割当ては、その索引セグメントに対して記憶域パラメータを設定すると制御できます。
- 索引セグメントのエクステントを構成するデータ・ブロック内の空き領域は、その索引セグメントの PCTFREE パラメータを設定すると制御できます。

索引セグメントの表領域は、所有者のデフォルト表領域または CREATE INDEX 文で指定された表領域です。索引を、その索引に対応する表と同じ表領域に格納する必要はありません。また、Oracle では索引と表のデータの両方をパラレルにして検索できるため、索引とその索引に対応する表をそれぞれ別のディスク・ドライブ上の異なる表領域内に格納すると、問合せのパフォーマンスが向上します。

関連項目： 22-15 ページ「ユーザー表領域の設定と割当て制限」

索引ブロックの形式

索引データに対して使用可能な領域は、Oracle ブロック・サイズから、ブロック・オーバーヘッド、エントリ・オーバーヘッド、ROWID および索引を付ける値 1 つ当たり 1 バイトの合計を引いたものです。索引ブロックのオーバーヘッドに必要なバイト数は、オペレーティング・システムによって異なります。

> **関連項目：** 索引ブロックのオーバーヘッドの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

索引の作成時には、索引を付ける列がフェッチされてソートされます。次に、各行について ROWID と索引値が一緒に格納されます。その後、索引が一番下から順にロードされます。たとえば、次の文を考えてみます。

```
CREATE INDEX employees_last_name ON employees(last_name);
```

Oracle は last_name 列に基づいて employees 表をソートし、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。索引を使用する場合、Oracle 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 列を持つ行を見つけます。

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

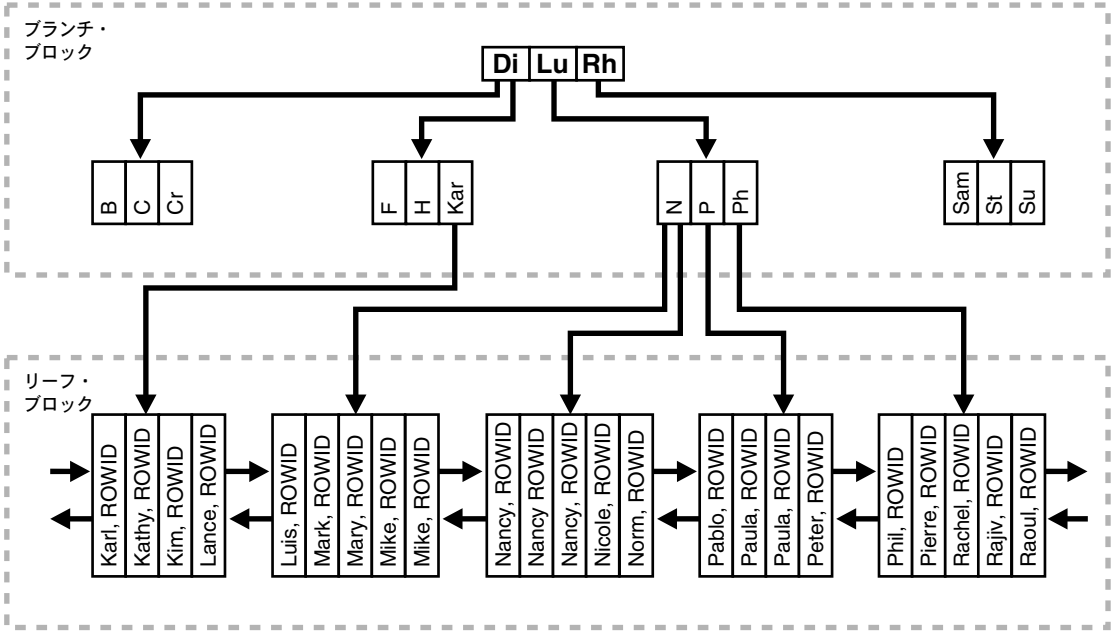
索引の内部構造

Oracle は、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。当然、データの量が多くなるほど、あまり正確な結果は得られません。

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 索引の基本原理です。

図 10-7 に、B ツリー索引の構造を示します。

図 10-7 B ツリー索引の内部構造



B ツリー索引の上位ブロック（**ブランチ・ブロック**）には、下位レベルの索引ブロックを指す索引データが含まれます。最下位レベルの索引ブロック（**リーフ・ブロック**）には、すべての索引対象のデータ値と、実際の行を検出するための ROWID が含まれます。リーフ・ブロックは二重にリンクされます。文字データを持つ列の索引は、データベース・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。

索引の検索方法

索引の一意スキャン

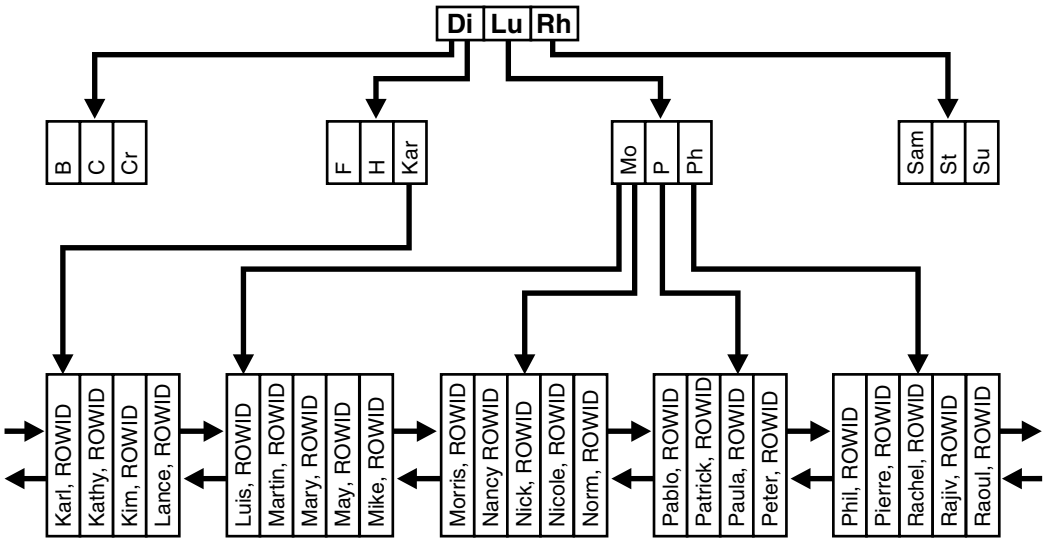
索引の一意スキャンは、データへの最も効率的なアクセス方法の 1 つです。このアクセス方法は、B ツリー索引からデータを戻す際に使用されます。一意（B ツリー）索引のすべての列が等価条件で指定されると、オプティマイザは一意スキャンを選択します。

索引の一意スキャンの手順

1. ルート・ブロックから開始します。
2. >= 値となる最小のキーのブロック・キーを検索します。
3. キー > 値の場合、このキーの前のリンクをたどって子ブロックに到達します。
4. キー = 値の場合、このリンクをたどって子ブロックに到達します。
5. 手順 2 で >= 値となるキーがなかった場合、ブロック内で最上位となるキーの後のリンクをたどります。
6. 子ブロックがブランチ・ブロックの場合、手順 2 から手順 4 を繰り返します。
7. = 値となるキーのリーフ・ブロックを検索します。
8. キーが見つかった場合は、`ROWID` を戻します。
9. キーが見つからない場合は、行は存在しません。

図 10-8 に、索引の一意スキャンの例を示します。下の解説はこの図の説明です。

図 10-8 索引の一意スキャンの例



Patrick を検索する場合。

- ルート・ブロックでは、Rh が >= Patrick となる最小のキーです。
- Rh の前のリンクをたどって、ブランチ・ブロック（Mo、P、Ph）に到達します。
- このブロックでは、Ph が >= Patrick となる最小のキーです。
- Ph の前のリンクをたどって、リーフ・ブロック（Pablo、Patrick、Paula、Peter）に到達します。
- このブロックで、Patrick = Patrick のキーを探します。
- Patrick = Patrick が見つかった場合は、(`KEY`, `ROWID`) を戻します。

Meg を検索する場合。

- ルート・ブロックでは、Rh が >= Meg となる最小のキーです。
- Rh の前のリンクをたどって、ブランチ・ブロック（Mo、P、Ph）に到達します。
- このブロックでは、Mo が >= Meg となる最小のキーです。
- Mo の前のリンクをたどって、リーフ・ブロック（Luis、...、May、Mike）に到達します。

- このブロックで、キー = Meg を探します。
- キー = Meg が見つからない場合は、行は戻しません。

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。

索引レンジ・スキャンの動作 索引レンジ・スキャンは一意索引と非一意索引の両方に行われます。B ツリーの非一意索引は、一意の B ツリー索引と同一です。ただし、同じキーには複数の値を認めます。

レンジ・スキャンでは、等価条件を指定できます。次に例を示します。

- 名前 = 'ALEX' - スタート・キー = 'ALEX', エンド・キー = 'ALEX'

または、スタート・キーとエンド・キーで境界付けた間隔を指定します。次に例を示します。

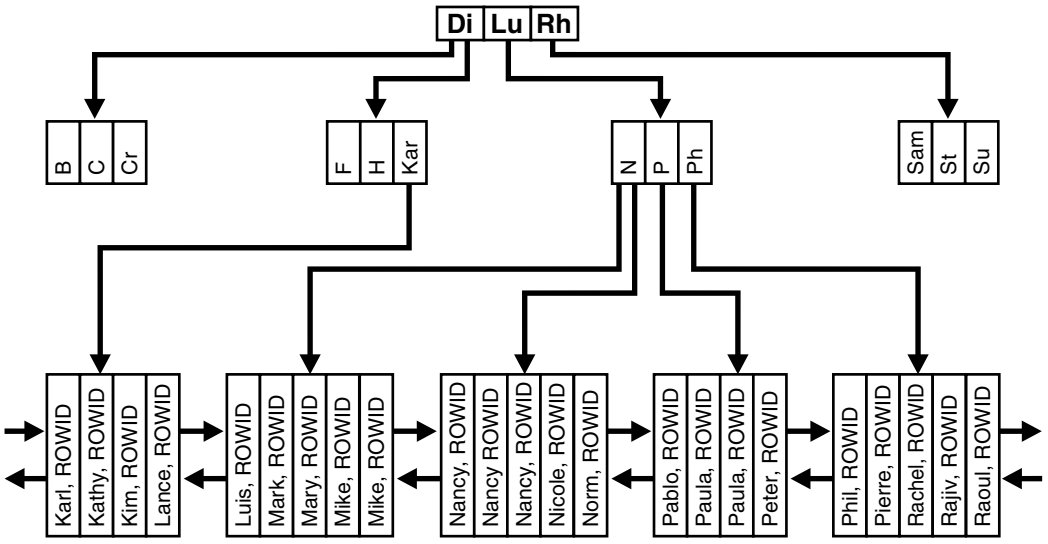
- 名前 LIKE 'AL%' - スタート・キー = 'AL, エンド・キー < 'AM'
- 100 と 120 の間の order_id - スタート・キー = 100, エンド・キー = 120

または、スタート・キーかエンド・キーのどちらか一方のみを指定します（非有界レンジ・スキャンの場合）。次に例を示します。

- order_book_date > SYSDATE - 30（先月受けた注文）
- employee_hire_date < SYSDATE - 3650 （勤続 10 年以上の従業員）

図 10-9 に、境界レンジ・スキャンの例を示します。下の解説はこの図の説明です。

図 10-9 境界レンジ・スキャンの例



境界レンジ・スキャンの手順

1. ルート・ブロックから開始します。
2. >= スタート・キーとなる最小のキーのブロック・キーを検索します。
3. キー > スタート・キーの場合、このキーの前のリンクをたどって子ブロックに到達します。
4. キー = スタート・キーの場合、このリンクをたどって子ブロックに到達します。
5. 手順 2 で >= スタート・キーとなるキーがなかった場合、ブロック内で最上位となるキーの後のリンクをたどります。
6. 子ブロックがブランチ・ブロックの場合、手順 2 から手順 4 を繰り返します。
7. >= スタート・キーとなる最小のキーのリーフ・ブロック・キーを検索します。
8. キー <= エンド・キーの場合は、次のとおりです。
   - キー列が WHERE 句の条件すべてを満たす場合、（値、ROWID）を戻します。
   - リンクを右にたどります。

このレンジ・スキャンでは、リーフ・ノードがすべて左から右へリンクされるという事実を利用しています。手順 7 では、表に ROWID でアクセスする前に、索引列に追加のフィルタ条件を適用できます。

左側に境界のある（右側は非有界の）レンジ・スキャンも、同様に開始されます。ただし、エンドポイントはチェックされません。右端のリーフ・キーに到達するまで、スキャンが続けられます。

右側に境界のあるレンジ・スキャンは、索引ツリーを横断して左端のリーフ・キーに達した後、キー >= 指定された条件となるキーに到達するまで手順 6 および手順 7 の手順を繰り返します。

非一意の B ツリー索引を使用したレンジ・スキャンで、Nancy を検索する場合。

- スタート・キー =Nancy、エンド・キー <Nancy とします。
- ルート・ブロックでは、Rh が >= スタート・キーとなる最小のキーです。
- Rh の前のリンクをたどって、ブランチ・ブロック（N、P、Ph）に到達します。
- このブロックでは、P が >= スタート・キーとなる最小のキーです。
- P の前のリンクをたどって、リーフ・ブロック（Nancy、...、Nicole、Norm）に到達します。
- このブロックでは、Nancy が >= スタート・キーとなる最小のキーです。
- Nancy<= エンド・キーであるため、(KEY、ROWID) を戻します。
- 次のキーも Nancy<= エンド・キーなので、(KEY、ROWID) を戻します。
- 次のキーも Nancy<= エンド・キーなので、(KEY、ROWID) を戻します。
- 次のキーは Nicole > エンド・キーなので、レンジ・スキャンを終了します。

P% を検索する場合。

- スタート・キー =P、エンド・キー <Q とします。
- ルート・ブロックでは、Rh が >= スタート・キーとなる最小のキーです。
- Rh の前のリンクをたどって、ブランチ・ブロック（N、P、Ph）に到達します。
- このブロックでは、P が = スタート・キーとなる最小のキーです。
- このリンクをたどって、リーフ・ブロック（Pablo、...、Peter）に到達します。
- このブロックでは、Pablo が >= スタート・キーとなる最小のキーです。
- Pablo<= エンド・キーであるため、(KEY、ROWID) を戻します。
- 次のキーは Paula<= エンド・キーなので、(KEY、ROWID) を戻します。
- 次のキーも Paula<= エンド・キーなので、(KEY、ROWID) を戻します。
- 次のキーは Phil<= エンド・キーなので、(KEY、ROWID) を戻します。

- 次のキーは Pierre<= エンド・キーなので、(KEY、ROWID) を戻します。
- 次のキーは Rachel> エンド・キーなので、レンジ・スキャンを終了します。

索引レンジ・スキャン降順

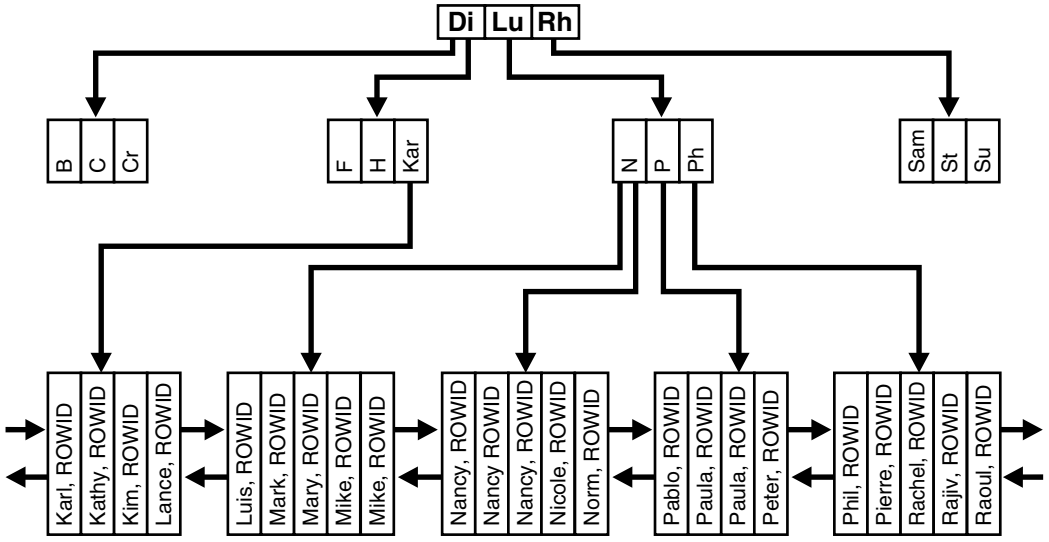
境界降順レンジ・スキャンの手順 降順レンジ・スキャンの場合（通常のレンジ・スキャンと同様）、等価条件または間隔を指定します。

1. ルート・ブロックから開始します。
2. <= エンド・キーとなる最大のキーのブロック・キーを検索します。
3. リンクをたどって子ブロックに到達します。
4. 手順 2 で <= エンド・キーとなるキーがなかった場合、ブロック内で最下位となるキーの前のリンクをたどります。
5. 子ブロックがブランチ・ブロックの場合、手順 2 から手順 4 を繰り返します。
6. <= エンド・キーとなる最大のキーのリーフ・ブロック・キーを検索します。
7. キー >= スタート・キーの場合は、次のとおりです。
   - キー列が WHERE 句の条件すべてを満たす場合、（値、ROWID）を戻します。
   - リンクを左にたどります。

このレンジ・スキャンでは、リーフ・ノードがすべて右から左へリンクされるという事実を利用しています。

図 10-10 に、境界レンジ・スキャンの例を示します。下の解説はこの図の説明です。

図 10-10 非一意の B ツリー索引を使用したレンジ・スキャンの例



Nancy を検索する場合。

- スタート・キー =Nancy、エンド・キー <Nancy とします。
- ルート・ブロックでは、Lu が <= エンド・キーとなる最大のキーです。
- リンクをたどって、ブランチ・ブロック（N、P、Ph）に到達します。
- このブランチ・ブロックでは、N が <= エンド・キーとなる最大のキーです。
- N の後のリンクをたどって、リーフ・ブロック（Nancy、...、Nicole、Norm）に到達します。
- このリーフ・ブロックでは、Nancy が <= エンド・キーとなる最大のキーです。
- Nancy>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーも Nancy>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーも Nancy>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーは Mike< スタート・キーなので、レンジ・スキャンを終了します。

P% を検索する場合。

- スタート・キー =P、エンド・キー <Q とします。

- ルート・ブロック・キーでは、Lu が <= エンド・キーとなる最大のキーです。
- リンクをたどって、ブランチ・ブロック（N、P、Ph）に到達します。
- このブランチ・ブロックでは、Ph が <= エンド・キーとなる最大のキーです。
- リンクをたどって、リーフ・ブロック（Phil、...、Raoul）に到達します。
- このリーフ・ブロックでは、Pierre が <= エンド・キーとなる最大のキーです。
- Pierre>= スタート・キーであるため、(KEY、ROWID) を戻します。
- その前のキーは Phil>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーは Peter>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーは Paula>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーは Pablo>= スタート・キーなので、(KEY、ROWID) を戻します。
- その前のキーは Norm< スタート・キーなので、レンジ・スキャンを終了します。

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、グループ化要素に ROWID を追加して一意要素を提供します。キー圧縮は、複数の一意要素で共有できるように、グループ化要素を分割し、格納する方法です。

接頭辞エントリと接尾辞エントリ

キー圧縮により、索引キーが接頭辞エントリ（グループ化要素）と接尾辞エントリ（一意要素）に分割されます。圧縮するために、接頭辞エントリが索引ブロック内の接尾辞エントリ間で共有されます。圧縮されるのは、B 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、重複キーが格納されて ROWID がキーに追加され、重複行が分割されます。キー圧縮を使用すると、重複キーは ROWID を除き接頭辞エントリとして索引ブロックに格納されます。残りの行は、ROWID のみからなる接尾辞エントリです。
- これと同じ動作は、(`stock_ticker, transaction_time`) など、(**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だけ格納されます。索引ブロックの他のエントリでは、`transaction_time` 値が共通の `stock_ticker` 接頭辞エントリを参照する接尾辞エントリとして格納されます。
- `VARRAY` または `NESTED TABLE` データ型を含む索引構成表の場合は、コレクション・データ型の要素ごとにオブジェクト ID（OID）が繰り返されます。キー圧縮を使用すると、重複する OID 値を圧縮できます。

ただし、キー圧縮を使用できない場合があります。たとえば、単一の属性キーを持つ一意索引の場合、一意要素はありますが、共有するグループ化要素がないため、キー圧縮は使用できません。

関連項目： 10-56 ページ「索引構成表」

逆キー索引

逆キー索引を作成する場合は、標準の索引とは違って、列の順序は保ちながら、索引が定義されている各列（ROWID を除く）のバイトを逆にします。索引に対する変更が少数のリーフ・ブロックに集中する Oracle9*i* 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方が、OLTP の Oracle9*i* Real Application Clusters アプリケーションが高速になることもあります。たとえば、電子メール・アプリケーションでメール・メッセージの索引を保存する場合、ユーザーが古いメッセージを保存していれば、索引はそれらのメッセージに対するポインタも、最新のメールに対するポインタと同様に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

注意： ビットマップ索引は、Oracle9*i* Enterprise Edition を購入した場合にのみ使用可能です。

Oracle9*i* および Oracle9*i* Enterprise Edition で使用可能な機能の詳細は、『Oracle9*i* データベース新機能』を参照してください。

索引を使用する目的は、特定のキー値が含まれる行へのポインタを提供することです。通常の索引では、そのために、キー値と、そのキー値を持つ行の ROWID の対応リストを格納します。Oracle は、キー値とそれに対応する ROWID を繰り返し格納します。**ビットマップ索引**では、ROWID のリストのかわりに、各キー値のビットマップを使用します。

ビットマップの各ビットは、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たします。異なるキー値の数が多くない場合、ビットマップ索引は領域を有効に使用できます。

ビットマップ索引は、`WHERE` 句に指定されたいくつかの条件に対応する索引を効率的にマージします。一部の条件は満たしているがすべては満たしていない行については、表自体にアクセスする前に除外します。これによって、応答時間が短縮されます。多くの場合は劇的に改善されます。

データ・ウェアハウス・アプリケーションの場合の利点

ビットマップ索引は、大量のデータと非定型問合せを扱うものの、同時実行トランザクションのレベルは高くないデータ・ウェアハウス・アプリケーションに対して効果的です。この種のアプリケーションに対するビットマップ索引の利点は次のとおりです。

- 多くの種類の非定型問合せの応答時間が短縮されます。
- 他の方式の索引と比べて使用領域が実質的に縮小されます。
- 機能の低いハードウェアでもパフォーマンスが劇的に向上します。
- パラレル DML とロードを効果的に実行できます。

従来の B 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 ツリー索引のほうが適しています。ビットマップ化された索引は、AND、OR、NOT 問合せまたは等価問合せでのみ有効です。

ビットマップ索引は、Oracle のコストベース最適化アプローチおよび実行エンジンと統合されます。また、他の Oracle 実行メソッドとシームレスに組み合せて使用できます。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、従来の索引だけでなく、ビットマップ索引に対しても機能します。パーティション表のビットマップ索引は、ローカル索引であることが必要です。索引のパラレル作成と連結索引もサポートされています。

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 10-1 に、ある会社の顧客データの一部を示します。

表 10-1 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 10-2 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 10-2 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 つの行に対応しており、各ビットの値は対応する行の値に依存しています。たとえば、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 10-11 に示すとおり、結果のビットマップから該当する値を数えて、この問合せを非常に効果的に処理できます。基準に該当する特定の顧客を識別するときは、結果のビットマップを使用して表にアクセスできます。

図 10-11 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。

ビットマップ索引と NULL の例 1

```
SELECT COUNT(*) FROM employees;
```

NULL データを持つ行を含め、すべての表の行の索引が作成されるため、この問合せには任意のビットマップ索引を使用できます。NULL の索引が作成されていない場合、オプティマイザは NOT NULL 制約を持つ列の索引のみを使用できます。

ビットマップ索引と NULL の例 2

```
SELECT COUNT(*) FROM employees WHERE commission_pct IS NULL;
```

この問合せは、commission_pct のビットマップ索引によって最適化できます。

ビットマップ索引と NULL の例 3

```
SELECT COUNT(*)
FROM customers
WHERE cust_gender = 'M' AND cust_state_province != 'CA';
```

この問合せは、cust_gender = 'M' のビットマップを検索し、cust_state_province = 'CA' のビットマップを除くことによって回答されます。cust_state_province に NULL 値が含まれる場合（つまり、NOT NULL 制約がない場合）は、cust_state_province = 'NULL' のビットマップも結果から除く必要があります。

パーティション表に対するビットマップ索引

他の索引の場合と同じように、ビットマップ索引もパーティション表に対して作成できます。ただし、制限が 1 つだけあります。ビットマップ索引はパーティション表に対してローカルであることが必要です。グローバル索引にはできません。グローバル・ビットマップ索引は、非パーティション表でのみサポートされます。

関連項目：

- パーティション表と、ローカルおよびグローバル索引の詳細は、第 11 章「パーティション表とパーティション索引」を参照してください。
- ビットマップ索引の使用方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

ビットマップ・ジョイン・インデックス

結合索引は、結合によって 1 つ以上の表の列を含んでいる、1 つの表の索引です。

最も単純な構成のビットマップ・ジョイン・インデックスの例は、D1,...,Dn 表の列に基づく F 表のビットマップ索引です。ここでは、10-54 ページの「ビットマップ・ジョイン・インデックスの作成」に説明があるように、スター・スキーマまたはスノーフレーク・スキーマで、Di が F と結合しています。データ・ウェアハウス環境では通常、表 F はファクト表、表 Di はディメンション表であり、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。わかりやすくするため、今後は ROWID がビットマップ化された表を **ファクト表**と呼び、ビットマップ・ジョイン・インデックスの結合に関わるその他の表を**ディメンション表**と呼ぶことにします。

結合として使用される結合索引がすでに計算済みの場合は、結合するデータのボリュームを削減できます。さらに、複数のディメンション表を含む結合索引は、既存のビットマップ索引でのスター型変換の際に必要なビット単位の操作を省略できます。最後に、ビットマップ・ジョイン・インデックスは格納に関しては、ファクト表の ROWID を圧縮しない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。

4 つの結合モデル

次に、スター・クエリー・フレームワークにおける 4 つの結合モデルを示し、そのビットマップ・ジョイン・インデックスによるアドレス指定方法を解説します。図は、それぞれ SQL 文で解説します。

表記規則

`F`*`i`* -- ファクト表 *i*
`D`*`i`* -- ディメンション表 *i*
`pk` -- ディメンション表の主キー列
`fk` -- ディメンション表と結合されるファクト表の列
`sales` -- ファクト表の測定列

図 10-12 ディメンション表の 1 列とファクト表 1 つの結合

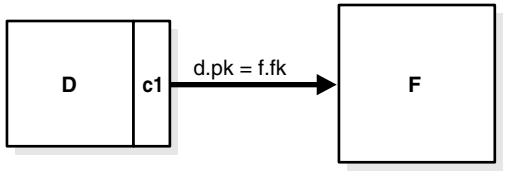


図 10-12 の場合、`F`（`D.c1`）のビットマップ・ジョイン・インデックスを示すモデルは次の SQL 文で表すことができます。

```
CREATE BITMAP INDEX bji ON f (d.c1) FROM f, d WHERE d.pk = f.fk
```

ビットマップ・ジョイン・インデックスにアクセスして次の問合せを実行すると、結合の操作を省略できます。

```
SELECT SUM(f.sales)
FROM d, f
WHERE d.pk = f.fk and d.c1 = 2
```

マテリアライズド結合ビューと同様、ビットマップ・ジョイン・インデックスは結合を計算し、それをデータベース・オブジェクトとして格納します。異なる点は、マテリアライズド結合ビューは結合を表に示すのに対し、ビットマップ・ジョイン・インデックスはビットマップ索引に示すということです。

図 10-13 ディメンション表の複数列とファクト表 1 つの結合

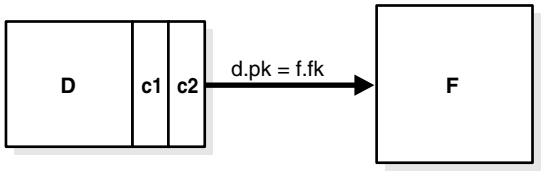


図 10-13 は図 10-12 の単純な拡張例で、次のような連結ビットマップ・ジョイン・インデックスで表されるものです。

```
CREATE BITMAP INDEX bji ON f (d.c1, d.c2)
FROM F, d
WHERE d.pk = f.fk;
```

ビットマップ・ジョイン・インデックス bji にアクセスすると、次の問合せの結果を取得できます。

```
SELECT SUM(f.sales)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1 AND d.c2 = 3;
```

索引キーの先頭部分のみを参照する別の問合せでも、次のようにビットマップ・ジョイン・インデックス bji を使用できます。

```
SELECT SUM(f.sales)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1
```

図 10-14 複数のディメンション表とファクト表 1 つの結合

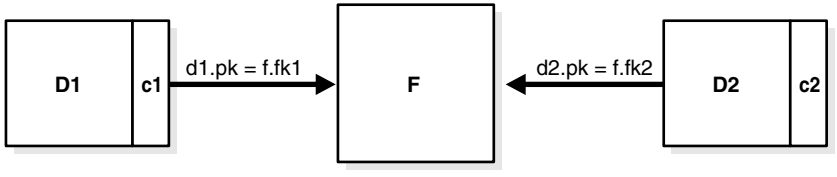


図 10-14 は、連結ビットマップ・ジョイン・インデックスが必要となる、3 つめのモデルです。

```
CREATE BITMAP INDEX bji ON f (d1.c1, d2.c2)
FROM f, d1, d2
WHERE d1.pk = f.fk1 AND d2.pk = f.fk2
```

図 10-15 スノーフレーク・スキーマ

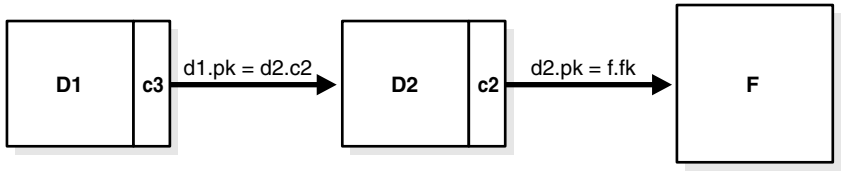


図 10-15 は、複数のディメンション表の結合を含んでいます。これはビットマップ・ジョイン・インデックスで表すことができます。ビットマップ・ジョイン・インデックスは、索引設定するディメンション表内の列の数によって、単一または連結にできます。d1.c3 に関する、d1 と d2 の結合との、および d2 と f の結合とのビットマップ・ジョイン・インデックスは、次のように作成できます。

```
CREATE BITMAP INDEX bji ON f (d1.c3)
FROM f, d1, d2
WHERE d1.pk = d2.c2 AND d2.pk = f.fk;
```

ビットマップ・ジョイン・インデックスは、こういったモデルの組合せの結合を表現できる必要があります。

ビットマップ・ジョイン・インデックスの作成

10-51 ページの「ビットマップ・ジョイン・インデックス」に定義されているような、1 つのファクト表 F と複数のディメンション表 D1,.., Dn によるスター・スキーマまたはスノーフレーク・スキーマを考えてみます。F を D1,.., Dn と結合するビットマップ・ジョイン・インデックスの制限事項を次に示します。

- ビットマップ・ジョイン・インデックスは、単一表 F 上にあります。
- FROM 句には、同じ表を二度使用することはできません。
- 結合はスター・スキーマまたはスノーフレーク・スキーマを構成します。結合はすべて、主キーまたは次のような一意制約を持つキーを通して行われます。
  - ファクト表と結合されるディメンション表の列は、主キー列または一意制約を持つキー列である必要があります。
  - 結合が D1><D2><F となるスノーフレーク・スキーマでは、D1><D2 の結合に関わる D1 の列は、主キー列または一意制約を持つキー列である必要があります。
  - ディメンション表の複合主キーの場合、キーの各列が結合に関わる必要があります。
- 結合はすべて等価内部結合であり、必ず AND で接続されます。

- 通常のビットマップ索引の作成に関する現在の制限事項も、ビットマップ・ジョイン・インデックスに適用されます。たとえば、一意属性を持つビットマップ索引は作成できません。他の制限事項については、『Oracle9*i* SQL リファレンス』を参照してください。
- ファクト表がパーティション化されていない場合、ビットマップ・ジョイン・インデックスはパーティション化できません。ファクト表がパーティション化されている場合は、対応するビットマップ・ジョイン・インデックスはそのファクト表でローカル・パーティションする必要があります。グローバル・パーティションのビットマップ・ジョイン・インデックスはサポートされていません。

IOT でのビットマップ・ジョイン・インデックス、機能ビットマップ・ジョイン・インデックス、および一時ビットマップ・ジョイン・インデックスは、まだ認められていません。

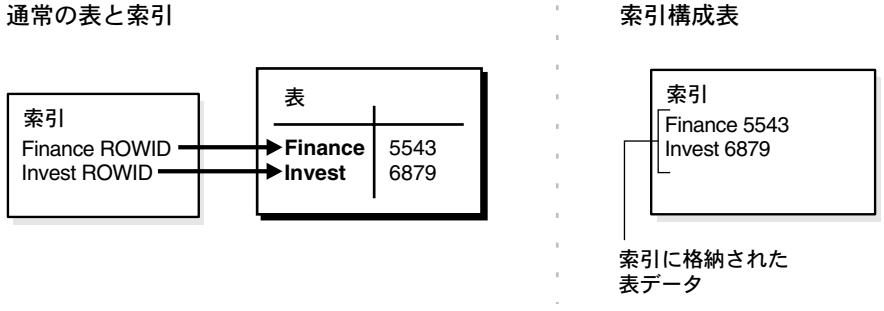
主キーまたは一意制約の要件は、ビットマップ・ジョイン・インデックスの正確性に関わる問題です。通常のビットマップ索引の場合、ビットマップのビットのセットと実表の ROWID には 1 対 1 のマッピング・リレーションがあります。ビットマップ・ジョイン・インデックスの場合、結合の結果セットの各行とファクト表の ROWID にも 1 対 1 のマッピング・リレーションが必要になります。主キーや一意制約は、この 1 対 1 マッピングを施行するために使用されます。

索引構成表

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）と違い、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 10-16 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 10-16 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 10-3 に、索引構成表と通常の表の違いをまとめます。

表 10-3 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。

表 10-3 索引構成表と通常の表の比較 （続き）

通常の表	索引構成表
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。

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記憶域オプション（下記参照）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）オーバーフロー記憶域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：

- 10-44 ページ「キー圧縮」
- 10-59 ページ「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、『Oracle9*i* データベース管理者ガイド』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。

この問題を処理するために 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記憶域にソートされます。

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記憶域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、『Oracle9*i* データベース管理者ガイド』を参照してください。

索引構成表の 2 次索引

索引構成表の 2 次索引がサポートされることにより、主キーでもその接頭辞でもない列を使用して、索引構成表に効率よくアクセスできます。

Oracle は、論理行識別子を使用して索引構成表の 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、必要に応じて、行のブロック位置を識別する**物理推測**が含まれます。Oracle は、これらの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： 12-22 ページ「論理 ROWID」

索引構成表のビットマップ索引

Oracle では、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 RANGE または HASH を指定して、索引構成表をパーティション化できます。パーティション化列は、主キー列のサブセットを形成する必要があります。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。Oracle9*i* 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
- 時系列アプリケーション

アプリケーション・ドメイン索引

Oracle は、複合データ型（文書、空間データ、イメージおよびビデオ・クリップなど）の索引に対応し、特化した索引作成テクニックを使用するために、**拡張索引作成機能**を提供します。この機能を使用すると、アプリケーション固有の索引管理ルーチンを**索引タイプ**・スキーマ・オブジェクトとしてカプセル化し、オブジェクト型の表の列や属性の**ドメイン索引**（アプリケーション固有の索引）を定義できます。また、拡張可能索引作成機能により、アプリケーション固有の**演算子**を効率的に処理できます。

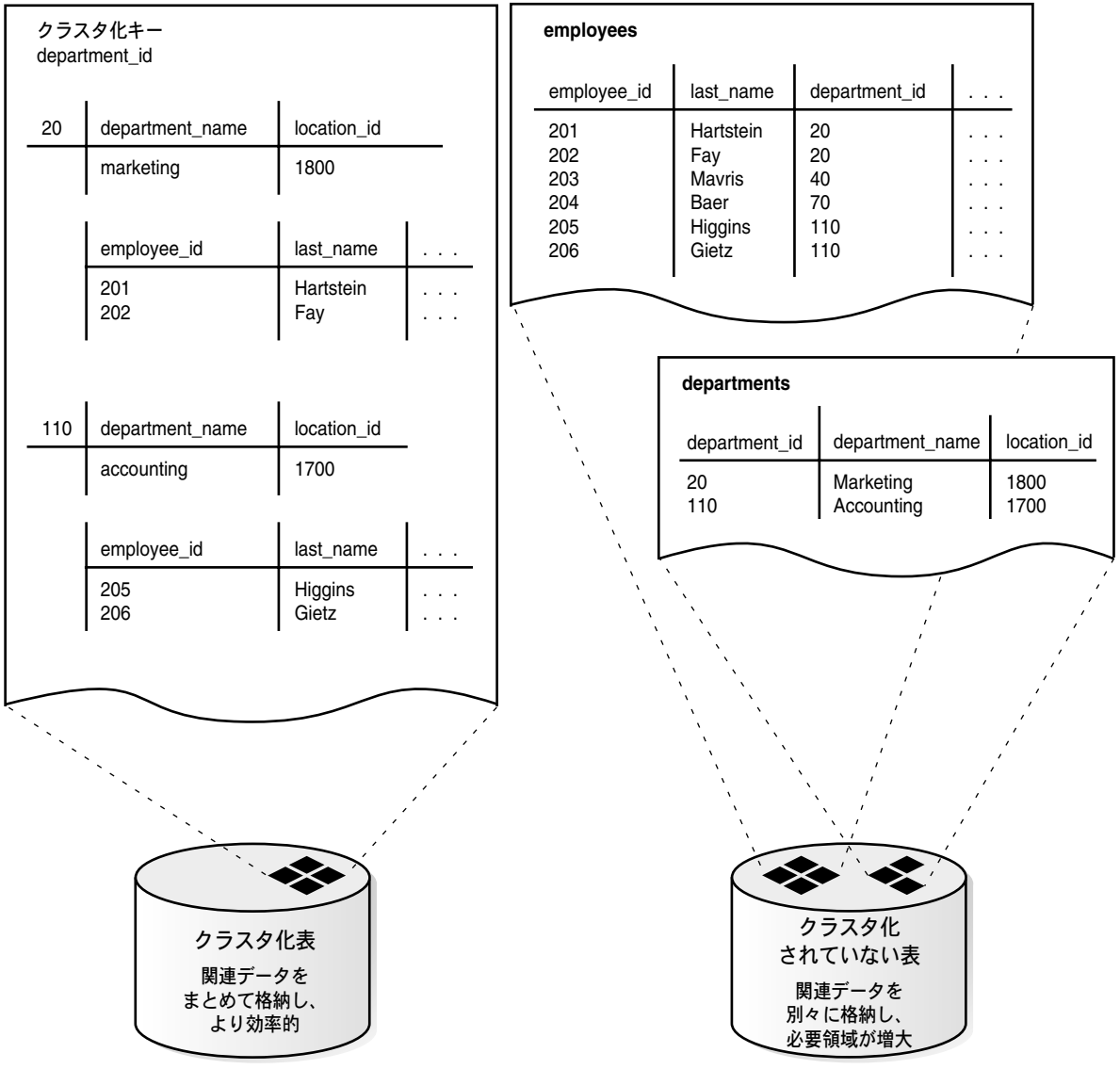
ドメイン索引の構造と内容は、**カートリッジ**と呼ばれるアプリケーション・ソフトウェアによって制御されます。Oracle サーバーは、ドメイン索引の作成、メンテナンスおよび検索のために、このアプリケーションと対話します。索引構造そのものは、索引構成表として Oracle データベースに格納するか、ファイルとして外部に格納できます。

> **関連項目：** Oracle の拡張性アーキテクチャにおけるデータ・カートリッジの使用方法については、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

クラスタ

クラスタは、表データを格納するためのオプションの方法です。クラスタは同じデータ・ブロックを共有する表のグループです。これらの表グループは共通の列を共有し、頻繁に併用されます。たとえば、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、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 10-17 に、employees 表と departments 表をクラスタ化する際の例を示します。

図 10-17 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、各クラスタ・キー値は、その値を持つ行がいくつの表に含まれているとしても、クラスタとクラスタ索引にそれぞれ一度だけ格納されます。そのため、クラスタとして関連付けた表データと索引データを格納するために必要な記憶域は、クラスタ化していない表で必要な記憶域よりも少なくなります。たとえば、図 10-17 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： クラスタの作成と管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

ハッシュ・クラスタ

ハッシュ・クラスタは、通常の索引クラスタ（ハッシュ関数ではなく索引がキーになっているクラスタ）の場合と同様の方法で表データをグループ化します。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。表の行はハッシュ・クラスタ内に物理的に格納され、ハッシュ関数の結果に従って検索されます。

ハッシュ関数は、**ハッシュ値**と呼ばれる、数値の分布を表す値を生成するために使用します。この値は、特定のクラスタ・キー値に基づくものです。ハッシュ・クラスタのキーは、索引クラスタのキーと同様に、単一の列またはコンポジット・キー（複数列のキー）にすることができます。ハッシュ・クラスタ内で行を検索または格納するために、Oracle はハッシュ関数を行のクラスタ・キー値に適用します。結果として得られるハッシュ値は、クラスタ内の 1 つのデータ・ブロックに対応し、Oracle は発行された文による読取り / 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、別個の索引に格納されているキー値に基づいて表の中で行が配置されます。索引付きの表またはクラスタの行を検索または格納するには、次のように最低 2 回の I/O を実行する必要があります。

- 索引内でキー値を検索または格納するために 1 回以上の I/O
- 表またはクラスタ内の行を読取り / 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、『Oracle9*i* データベース管理者ガイド』を参照してください。

11

パーティション表とパーティション索引

この章では、パーティション表とパーティション索引について説明します。この章の内容は、次のとおりです。

注意： Oracle では、表、表の索引、マテリアライズド・ビュー、およびマテリアライズド・ビューの索引のパーティション化のみをサポートしています。クラスタ化表やクラスタ化表の索引のパーティション化はサポートしていません。

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。OLTP システムは管理性と可用性の改良によって改善されることが多く、データ・ウェアハウス・システムはパフォーマンスと管理性によって改善されます。

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。

関連項目：

- 3-13 ページ「マルチ・ブロック・サイズ」
- パーティション化の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パーティション化には次のような利点があります。

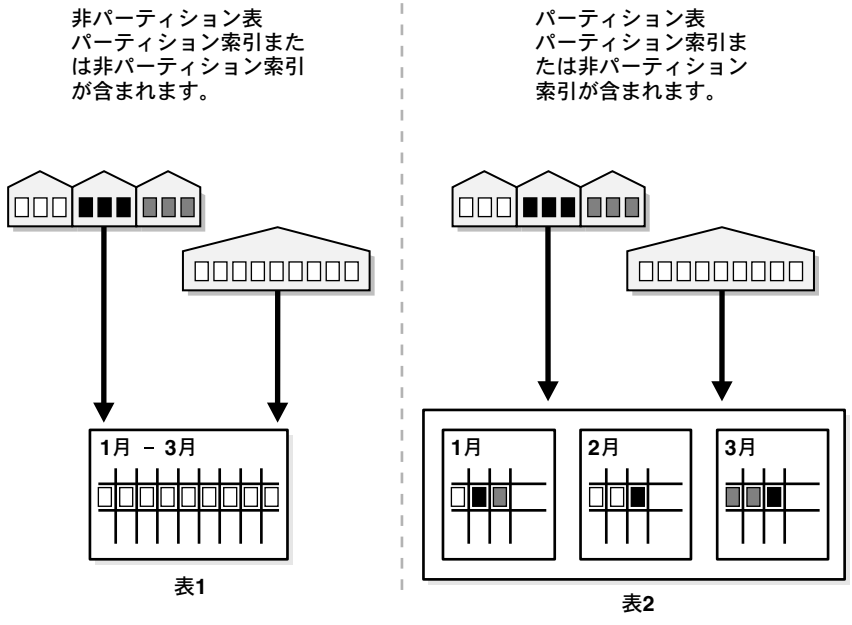
- パーティション化により、表全体でなくパーティション・レベルでのデータのロード、索引の作成や再作成、バックアップやリカバリといったデータ管理操作が可能になります。このため、これらの操作にかかる時間が大幅に短縮されます。
- パーティション化により、問合せのパフォーマンスが改善されます。多くの場合、問合せの結果は表全体ではなくパーティションのサブセットにアクセスして得ることができます。一部の問合せでは、このテクニック（**パーティション・プルーニング**という）によってパフォーマンスを大幅に改善できます。
- パーティション化により、定期的なメンテナンス操作による停止時間の影響を大幅に軽減できます。

  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。

- メンテナンス操作の時間枠やリカバリ時間を短縮し、障害による影響を軽減させる目的でクリティカルな表と索引がパーティションに分けられている場合、パーティション化により、ミッション・クリティカルなデータベースの可用性が高まります。
- 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。

図 11-1 は、パーティション表と非パーティション表の相違点を図示したものです。

図 11-1 パーティション表のビュー



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。Oracle9*i* ではパーティション・キーを使用して、挿入、更新および削除の操作を適切なパーティションに自動的に指示します。パーティション・キーには次のような特長があります。

- 1 から 16 の順序付けられた列のリストから成ります。
- 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 64,000 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。

パーティション索引構成表

索引構成表はレンジ・パーティション化できます。この機能は索引構成表の管理性、可用性およびパフォーマンスの向上に非常に有効です。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。この一般的な例は、Image Cartridge および *inter*Media Cartridge です。

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。

- レンジ・パーティション化およびハッシュ・パーティション化のみがサポートされています。
- 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。

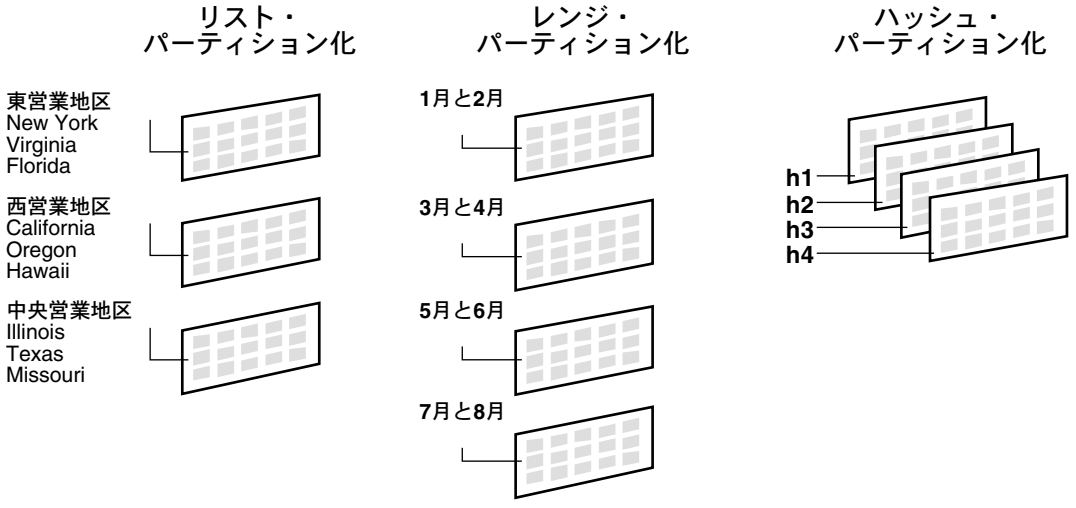
パーティション化の方法

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
- リスト・パーティション化
- ハッシュ・パーティション化
- コンポジット・パーティション化

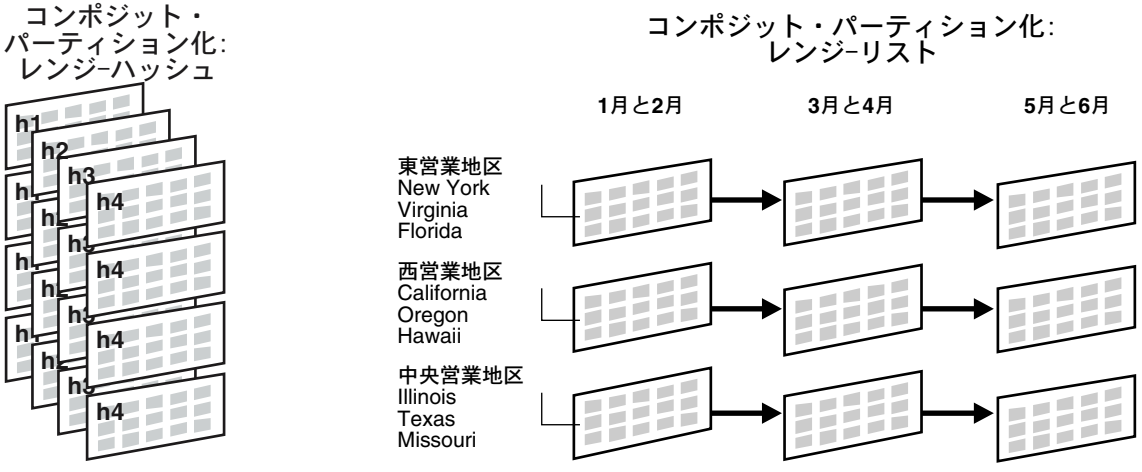
図 11-2 は、パーティション化方法の例を図示したものです。

図 11-2 リスト・パーティション化、レンジ・パーティション化およびハッシュ・パーティション化



コンポジット・パーティション化は、他のパーティション化方法を組み合せたものです。Oracle は現在、レンジ - ハッシュ・コンポジット・パーティション化とレンジ - リスト・コンポジット・パーティション化をサポートしています。図 11-3 は、レンジ - ハッシュ・コンポジット・パーティション化とレンジ - リスト・コンポジット・パーティション化を図示したものです。

図 11-3 コンポジット・パーティション化



レンジ・パーティション化

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。これは最も一般的なタイプのパーティション化で、多くの場合日付とともに使用されます。たとえば、売上データを月別パーティションに分割する場合などです。

レンジ・パーティション化を使用する際は、次のルールを考慮する必要があります。

- 各パーティションには VALUES LESS THAN 句があり、これがパーティションの上限（この値は含まれない）を指定します。このリテラル以上となるパーティション・キーのバイナリ値は、次のパーティションに追加されます。
- 最初の 1 つを除くすべてのパーティションには、前のパーティションの VALUES LESS THAN 句に指定された暗黙的な下限があります。
- 最上位のパーティションには、MAXVALUE リテラルを定義できます。MAXVALUE は、仮想の無限大値を示しており、そのパーティション・キーの他のどんな可能な値（NULL 値を含む）よりも高い値としてソートされます。

代表的な例を次に示します。この文は、sales_date フィールドでレンジ・パーティション化される表（sales_range）を作成します。

レンジ・パーティション化の例

```
CREATE TABLE sales_range
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
);
```

リスト・パーティション化

リスト・パーティション化では、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。これは各パーティションの記述で、パーティション化キーの不連続な値のリストを指定して行います。これは値の範囲がパーティションに対応付けられているレンジ・パーティション化や、ハッシュ関数がパーティションへの行のマッピングを制御するハッシュ・パーティション化とは異なります。リスト・パーティション化の利点は、関連もなく順序付けもされていないデータのセットを、自然な形でグループ化および編成できることです。

リスト・パーティション化の詳細は、例示して説明するのが最も有効です。ここでは、売上表を領域別にパーティション化することにします。つまり、次の例のように、州を地理的な位置によってグループ分けします。

リスト・パーティション化の例

```
CREATE TABLE sales_list
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_state   VARCHAR2(20),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY LIST(sales_state)
(
PARTITION sales_west VALUES('California', 'Hawaii'),
PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
PARTITION sales_central VALUES('Texas', 'Illinois')
PARTITION sales_other VALUES(DEFAULT)
);
```

行は、行のパーティション化列の値が、そのパーティションを示した値のセットの範囲内にあるかどうかをチェックすることによって、パーティションにマップされます。たとえば、各行は次のように挿入されます。

- (10,'Jones','Hawaii',100,'05-JAN-2000')は、パーティション sales_west にマップされます。
- (21,'Smith','Florida',150,'15-JAN-2000')は、パーティション sales_east にマップされます。
- (32,'Lee','Colorado',130,'21-JAN-2000')は、表内のどのパーティションにもマップされません。

レンジ・パーティション化やハッシュ・パーティション化と違い、リスト・パーティション化ではマルチカラム・パーティション・キーはサポートされていません。表をリストによってパーティション化する場合、パーティション化キーは常に表の単一列から成ります。

DEFAULT パーティションを使用すると、リスト・パーティション表に可能な値をすべて指定する必要がなく、他のどのパーティションにもマップされない行でもエラーは生成されません。

ハッシュ・パーティション化

ハッシュ・パーティション化では、レンジ・パーティション化やリスト・パーティション化ができないデータを容易にパーティション化できます。これは単純な構文で行われ、実装も容易です。レンジ・パーティション化よりハッシュ・パーティション化のほうが適しているのは、次のような場合です。

- 特定の範囲にマップされるデータ量が事前に把握できない場合。
- 各レンジ・パーティションのサイズが大幅に異なるか、または手動で調整するのが困難な場合。
- レンジ・パーティション化を行うとデータが不適切にクラスタ化されてしまう場合。
- パラレル DML、パーティション・プルーニングおよびパーティション・ワイズ結合といったパフォーマンス機能が重要な場合。

ハッシュ・パーティションには、パーティションの分割、削除およびマージの概念は当てはまりません。そのかわり、ハッシュ・パーティションは追加および結合ができます。

ハッシュ・パーティション化の例

```
CREATE TABLE sales_hash
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
week_no       NUMBER(2))
PARTITION BY HASH(salesman_id)
```

```
PARTITIONS 4
STORE IN (data1, data2, data3, data4);
```

この文は、salesman_id フィールドでハッシュ・パーティション化される表 sales_hash を作成します。表領域名は、data1、data2、data3 および data4 です。

コンポジット・パーティション化

コンポジット・パーティション化では、データはレンジ方式でパーティション化されてから、各パーティション内でハッシュまたはリスト方式でサブパーティション化されます。レンジ - ハッシュ・コンポジット・パーティション化では、レンジ・パーティション化の優れた管理性や、ハッシュ・パーティション化のデータ配置、ストライプ化およびパラレル化といった利点が生かされています。レンジ - リスト・コンポジット・パーティション化では、レンジ・パーティション化の管理性と、サブパーティションに対するリスト・パーティション化の明示的な制御が生かされています。

コンポジット・パーティション化では新しいレンジ・パーティションの追加といった履歴操作がサポートされていますが、同時にサブパーティション化を通して、DML 操作のより高い並列度やさらにきめの細かいデータ配置も可能になっています。

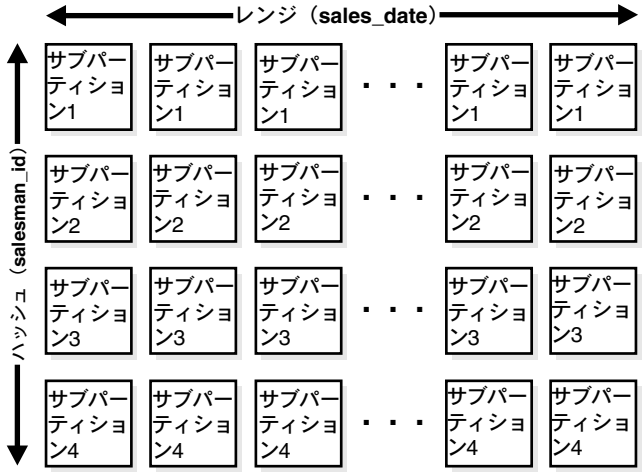
レンジ - ハッシュ・コンポジット・パーティション化の例

```
CREATE TABLE sales_composite
(salesman_id  NUMBER(5),
 salesman_name VARCHAR2(30),
 sales_amount  NUMBER(10),
 sales_date    DATE)
PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(salesman_id)
SUBPARTITION TEMPLATE(
SUBPARTITION sp1 TABLESPACE data1,
SUBPARTITION sp2 TABLESPACE data2,
SUBPARTITION sp3 TABLESPACE data3,
SUBPARTITION sp4 TABLESPACE data4)
(PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY'))
 PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY'))
 PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY'))
 PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
 PARTITION sales_may2000 VALUES LESS THAN(TO_DATE('06/01/2000','DD/MM/YYYY')));
```

この文は、sales_date フィールドでレンジ・パーティション化され、かつ salesman_id でハッシュ・サブパーティション化される表 sales_composite を作成します。テンプレートを使用すると、パーティション名、アンダースコアおよびテンプレートからのサブパーティション名が連結されて、サブパーティション名が作成されます。このサブパーティションは、テンプレートに指定されている表領域に格納されます。前述の文では、sales_jan2000_sp1 が作成されて表領域 data1 に格納され、sales_jan2000_sp4 が作成され

て表領域 data4 に格納されます。同様に、sales_apr2000_sp1 が作成されて表領域 data1 に格納され、sales_apr2000_sp4 が作成されて表領域 data4 に格納されます。図 11-4 は、前述の例を図示したものです。

図 11-4 レンジ - ハッシュ・コンポジット・パーティション化

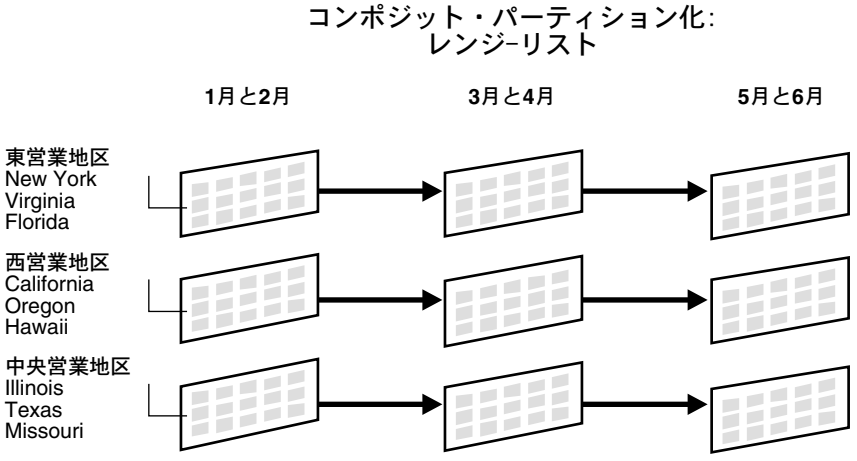


レンジ - リスト・コンポジット・パーティション化の例

```
CREATE TABLE bimonthly_regional_sales
(deptno NUMBER,
 item_no VARCHAR2(20),
 txn_date DATE,
 txn_amount NUMBER,
 state VARCHAR2(2))
PARTITION BY RANGE (txn_date)
SUBPARTITION BY LIST (state)
SUBPARTITION TEMPLATE(
    SUBPARTITION east VALUES('NY', 'VA', 'FL') TABLESPACE ts1,
    SUBPARTITION west VALUES('CA', 'OR', 'HI') TABLESPACE ts2,
    SUBPARTITION central VALUES('IL', 'TX', 'MO') TABLESPACE ts3)
(
PARTITION janfeb_2000 VALUES LESS THAN (TO_DATE('1-MAR-2000','DD-MON-YYYY')),
PARTITION marapr_2000 VALUES LESS THAN (TO_DATE('1-MAY-2000','DD-MON-YYYY')),
PARTITION mayjun_2000 VALUES LESS THAN (TO_DATE('1-JUL-2000','DD-MON-YYYY'))
);
```

この文は、txn_date フィールドでレンジ・パーティション化され、かつ state でリスト・サブパーティション化される表 bimonthly_regional_sales を作成します。テンプレートを使用すると、パーティション名、アンダースコアおよびテンプレートからのサブパーティション名が連結されて、サブパーティション名が作成されます。このサブパーティションは、テンプレートに指定されている表領域に格納されます。前述の文では、janfeb_2000_east が作成されて表領域 ts1 に格納され、janfeb_2000_central が作成され表領域 ts3 に格納されます。同様に、mayjun_2000_east が作成されて表領域 ts1 に格納され、mayjun_2000_central が表領域 ts3 に格納されます。図 11-5 は、表 bimonthly_regional_sales とその 9 個のサブパーティションを図示したものです。

図 11-5 レンジ - リスト・コンポジット・パーティション化



表をパーティション化する場合

ここでは表をパーティション化する場合についての提案を示します。

- 2GB 以上の表は常に、パーティション化の考慮対象となります。
- 最新のパーティションに新しいデータが追加されるような履歴データを含む表も、パーティション化の考慮対象となります。この典型的な例は、現在月のデータのみ更新可能で他の 11 か月は読取り専用になっている履歴表です。

パーティション索引

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拡張性を改善します。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。

> **関連項目：** パーティション索引の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

ローカル・パーティション索引

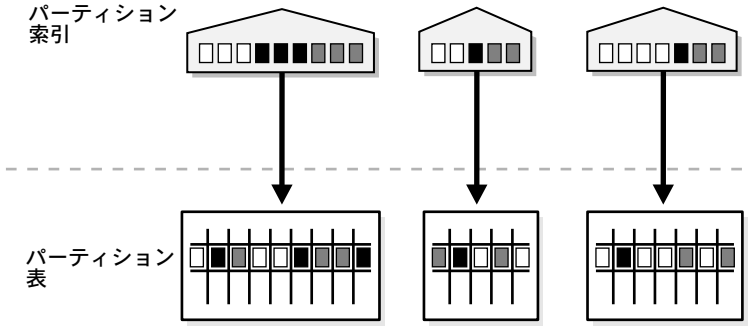
ローカル・パーティション索引は、他のタイプのパーティション索引よりも管理が容易です。ローカル・パーティション索引でも可用性が拡張されており、DSS 環境では一般的となっています。その理由は、ローカル索引の各パーティションが必ず表の 1 つのパーティションに対応付けられている、同一レベル・パーティション化にあります。これにより、索引のパーティションを自動的に表のパーティションと同時保存し、かつ表と索引の各ペアをそれぞれ独立させることができます。1 つのパーティションのデータを無効または使用不可能にする処理は、単一パーティションのみに影響します。

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。

図 11-6 は、ローカル・パーティション索引を図示したものです。

図 11-6 ローカル・パーティション索引



グローバル・パーティション索引

グローバル・パーティション索引は、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。新しい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・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 GLOBAL 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。

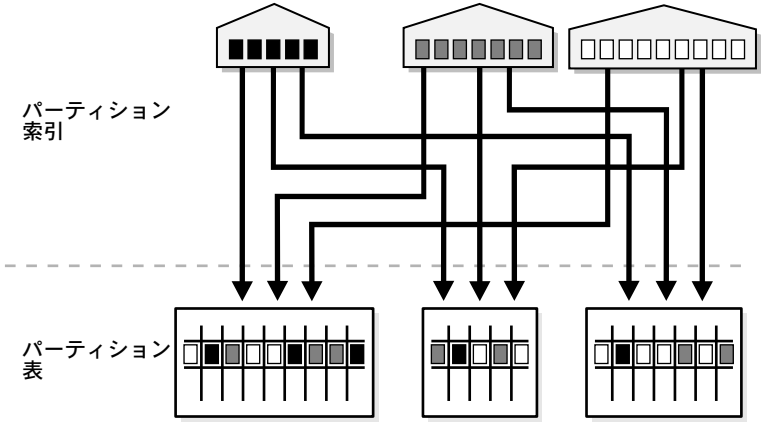
例： `ALTER TABLE DROP PARTITION P1 UPDATE GLOBAL INDEXES`

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 GLOBAL INDEX 句の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

図 11-7 は、グローバル・パーティション索引を図示したものです。

図 11-7 グローバル・パーティション索引

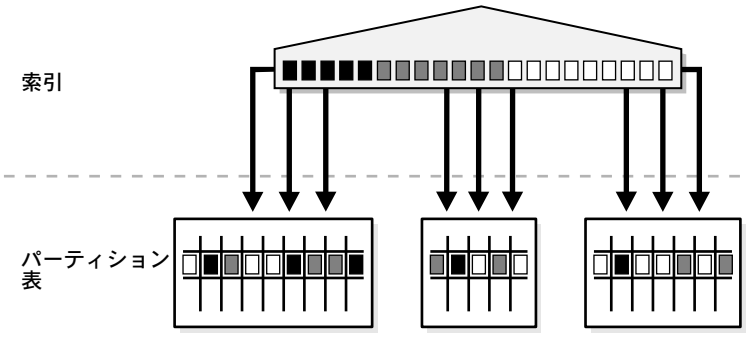


グローバル非パーティション索引

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。

図 11-8 は、グローバル非パーティション索引を図示したものです。

図 11-8 グローバル非パーティション索引



パーティション索引の例

索引の作成例 : 例に使用している表の冒頭部分

```
CREATE TABLE employees
(employee_id NUMBER(4) NOT NULL,
 last_name VARCHAR2(10),
 department_id NUMBER(2))
PARTITION BY RANGE (department_id)
(PARTITION employees_part1 VALUES LESS THAN (11) TABLESPACE part1,
 PARTITION employees_part2 VALUES LESS THAN (21) TABLESPACE part2,
 PARTITION employees_part3 VALUES LESS THAN (31) TABLESPACE part3);
```

ローカル索引の作成例

```
CREATE INDEX employees_local_idx ON employees (employee_id) LOCAL;
```

グローバル索引の作成例

```
CREATE INDEX employees_global_idx ON employees(employee_id);
```

グローバル・パーティション索引の作成例

```
CREATE INDEX employees_global_part_idx ON employees(employee_id)
GLOBAL PARTITION BY RANGE(employee_id)
(PARTITION p1 VALUES LESS THAN(5000),
 PARTITION p2 VALUES LESS THAN(MAXVALUE));
```

パーティション化された索引構成表の作成例

```
CREATE TABLE sales_range
(
salesman_id   NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE,
PRIMARY KEY(sales_date, salesman_id))
ORGANIZATION INDEX INCLUDING salesman_id
OVERFLOW TABLESPACE tabsp_overflow
PARTITION BY RANGE(sales_date)
(PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY'))
 OVERFLOW TABLESPACE p1_overflow,
 PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY'))
 OVERFLOW TABLESPACE p2_overflow,
 PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY'))
 OVERFLOW TABLESPACE p3_overflow,
 PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
 OVERFLOW TABLESPACE p4_overflow);
```

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。

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挙げます。

- レンジ・パーティション化グローバル索引のみがサポートされています。
- 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
- パラレル DML

パーティション・プルーニング

Oracle サーバーは、パーティションとサブパーティションを明示的に認識します。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。

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・ベース索引でない限り、SQL 文によって索引付きの列に関数が適用されると、オ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限ります。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

orders と呼ばれるパーティション表があります。orders のパーティション・キーは order_date です。orders には 1 月から 6 月までの 6 か月分のデータがあり、各月のデータが 1 つのパーティションになっているとします。ここで次のような問合せを実行した場合、

```
SELECT SUM(value)
FROM 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 つの表を結合する際に使用できる最適化のことです。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、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パラレル DML

パラレル実行を行うと、主に意思決定支援システムやデータ・ウェアハウスに対応付けられた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劇的に短縮されます。従来型の表に加え、レンジ・パーティションおよびハッシュ・パーティションの表でもパラレル問合せやパラレル DML が使用できます。これにより、バッチ操作の拡張性およびパフォーマンスを拡張できます。

パラレル DML セッションの方法および制限事項は、索引構成表の使用の有無によらず変わりません。

> **関連項目：** パラレル DML とそのパーティション表における使用方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

12

システム固有のデータ型

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、次のとおりです。

- 文字データ型
  - CHAR データ型
  - VARCHAR2 および VARCHAR データ型
  - NCHAR および NVARCHAR2 データ型
  - LONG データ型
- NUMBER データ型
- DATE データ型
- LOB データ型
  - BLOB データ型
  - CLOB および NCLOB データ型
  - BFILE データ型
- RAW および LONG RAW データ型
- ROWID および UROWID データ型
  - 物理 ROWID
  - 論理 ROWID
  - Oracle 以外のデータベースの ROWID

注意： PL/SQL には、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 データ型と各 Oracle データ型の特性の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
- 組込みデータ型の使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

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

文字データ型は、文字コード体系（通常、キャラクタ・セットまたはコード・ページと呼ばれる）に対応するバイト値によって、文字（英数字）データを文字列に格納します。

データベースのキャラクタ・セットは、データベースの作成時に設定されます。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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- 文字データ変換の詳細は、『Oracle9*i* 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。

CHAR データ型

CHAR データ型は、固定長の文字列を格納します。CHAR 列を含む表を作成するときは、CHAR 列の文字列の長さを 1 から 2000 までの値で指定する必要があります（単位は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、固定長に合せて空白が埋め込まれます。
- 末尾に空白が続く長い値を与えると、固定長に合せて値から空白が切り捨てられます。
- 値が大きすぎると、Oracle からエラーが戻されます。

Oracle は、空白埋め比較方法を使用して CHAR 値を比較します。

関連項目： 空白埋め比較方法の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

VARCHAR2 および VARCHAR データ型

VARCHAR2 データ型には、可変長の文字列が格納されます。VARCHAR2 列を含む表を作成するときは、VARCHAR2 列の文字列の最大長を 1 から 4000 までの値で指定します（単位はバイト数または文字数）。各行の値は、可変長フィールドとして列に格納されます。値が列の最大長を超えている場合は、Oracle からエラーが戻されます。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 は、非空白埋め比較方法を使用して VARCHAR2 値を比較します。

> **関連項目：** 非空白埋め比較の詳細は、『Oracle9*i* SQL 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・セットを処理したり、それらのキャラクタ・セット間の変換が可能です。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、NLS_LENGTH_SEMANTICS 初期化パラメータを INIT.ORA に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：

- 12-6 ページ「Oracle データベース内の 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、『Oracle9*i* 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と適切な Unicode キャラクタ・セットの選択の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- 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、『Oracle9*i* データベース移行ガイド』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、必ずキャラクタ・セマンティクス内になります。キャラクタ・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。

例 12-1 列の最大バイト長の定義

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。

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、『Oracle9*i* 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 データベース内の 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使用する Oracle ユーザーは、Oracle データベースに必ず 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 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、必ずキャラクタ・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・タイプでは、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 のすべての暗黙的な変換に加えて、Oracle は、NCHAR/NVARCHAR2 の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 4GB の文字データ（CLOB）または各国語キャラクタ・セット・データ（NCLOB）を格納できます。LOB データ型は、LONG データ型の機能性と置き換えられます。

関連項目： 12-13 ページ「LOB データ型」

LONG データ型

注意： LONG データ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションでは、大量の文字データを格納するには CLOB および NCLOB データ型を使用してください。

LONG と定義されている列には、最大 2GB までの情報を含む可変長の文字データを格納できます。LONG データは、異なるシステム間で移動しても適切に変換されるテキスト・データです。

LONG データ型の列は、ビュー定義のテキストを格納するために、データ・ディクショナリで使用されます。LONG 列は、SELECT 構文のリスト、UPDATE 文の SET 句および INSERT 文の VALUES 句で使用できます。

関連項目：

- LONG データ型の制限の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- LONG RAW データ型の詳細は、12-16 ページの「RAW および LONG RAW データ型」を参照してください。

NUMBER データ型

NUMBER データ型には、固定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 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 バージョン 5 データベースからのインポート時にのみ生成されます）。

数値列の場合は、次のように列を指定できます。

```
column_name NUMBER
```

また、任意で次のように**精度**（全体の桁数）と**スケール**（小数点以下の桁数）を指定することもできます。

```
column_name NUMBER (precision, scale)
```

精度を指定しないと、値は与えられたとおりに列に格納されます。スケールを指定しないと、スケールは 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 12-1 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 12-1 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

表 12-1 スケール変更係数が数値データの格納に与える影響 （続き）

入力データ	指定	格納方法
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(精度を超えているため、受け入れられない)
7,456,123.89	NUMBER(7,-2)	7456100

負のスケールを指定すると、小数点の左側の指定した桁数で実際のデータが丸められます。たとえば、(7,-2) という指定は、表 12-1 に示されているとおり、Oracle によって 100 の位で丸めることを意味します。

数値の入力と出力において、標準 Oracle のデフォルト小数点文字はピリオドです。たとえば、1234.56 というようになります。小数点文字は、数値の整数部と小数部を分離する文字です。デフォルトの小数点文字は、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 は、数値データを可変長形式で格納します。それぞれの値は科学表記法で格納され、指数を格納するために 1 バイト、仮数を格納するために最大 20 バイトが使用されます。結果の数値の精度は 38 桁に制限されます。Oracle は、先行ゼロと後続ゼロは格納しません。たとえば、数値 412 は 4.12×10^2 という形式で格納され、指数（2）を格納するために 1 バイト、仮数の 3 桁の有効数字（4、1、2）を格納するために 2 バイトが使用されます。負数の長さには、符号が含まれます。

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

```
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 データベース・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。

DATE データ型

DATE データ型には、ある時点の値（日付と時刻）が表に格納されます。DATE データ型には、年（世紀を含む）、月、日、時、分および秒（真夜中から数える）が格納されます。

Oracle では、ユリウス暦の西暦前 4712 年 1 月 1 日から西暦 4712 年 12 月 31 日までの日付を格納できます。BCE（書式マスクでは 'BC'）が明確に指定されないかぎり、デフォルトとして日付は西暦として扱われます。

Oracle は、日付を格納するために固有の内部形式を使用します。日付データは、それぞれ世紀、年、月、日、時、分および秒に対応する 7 バイトの固定長フィールドに格納されます。

日付の入出力に対する標準的な Oracle の日付書式は、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 は、時刻を HH:MI:SS の 24 時間形式で格納します。デフォルトでは、時刻部分に何も指定しない場合、日付フィールドの時刻部分は 00:00:00 A.M.（真夜中）になります。時刻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.'));
```

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 は簡略化された方法を使用しているため、結果は整数値になります。ユリウス日付は、様々な計算方法と解釈があります。Oracle 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 は、データを世紀情報と一緒に格納します。たとえば、Oracle データベースには、単に 96 または 01 ではなく、1996 または 2001 が格納されます。DATE データ型は常に 4 桁の年を内部に格納し、データベース内部に格納される他のすべての日付も 4 桁の年となります。インポート、エクスポートおよびリカバリなどの Oracle ユーティリティでも、年を 4 桁で処理しています。

夏時間のサポート

Oracle9*i* 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 と TIMESTAMP WITH LOCAL TIME ZONE は、タイム・ゾーンです。

関連項目：

- 世紀と日付の書式マスクの詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- 日付書式コードの詳細は、『Oracle9*i* SQL 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。次のような違いがあります。

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・タイム・ゾーンに格納されます。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。

例：

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 ファンクション 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 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。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、『Oracle9*i* SQL リファレンス』を参照してください。

LOB データ型

LOB データ型 BLOB、CLOB、NCLOB および BFILE は、非構造化データ（テキスト、グラフィック・イメージ、ビデオ・クリップおよびサウンド・ウェーブなど）の大きなブロックを格納します。最大サイズは 4GB です。このデータ型を使用すると、個々のデータに対する効率的なランダム・アクセスが可能です。オラクル社では、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。
- LOB の最大サイズは 4GB ですが、LONG の最大サイズは 2GB です。
- LOB はデータへのランダム・アクセスをサポートしていますが、LONG は順次アクセスしかサポートしていません。
- LOB データ型（NCLOB を除く）をユーザー定義オブジェクト型の属性として指定できますが、LONG データ型には指定できません。

- ローカル変数のように機能する一時 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、ユーザーの一時表領域に作成され、表には依存しません。ただし、LONG データ型の場合、一時構造は使用できません。
- 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 ロケータを使用する）または外部ファイル（BFILE データ型）のいずれかに格納できます。

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 ファンクションを使用できます。

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、『Oracle9*i* SQL リファレンス』を参照してください。
- LOB 記憶域と LOB ロケータの詳細は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

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 4GB 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 4GB 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。可変幅データベース・キャラクタ・セットの場合は、CLOB 値は、固定幅の 2 バイトの Unicode キャラクタ・セットを使用して、データベースに格納されます。Oracle は、格納された Unicode 値をクライアントまたはサーバーに必要なキャラクタ・セットに変換します。このキャラクタ・セットは、固定幅の場合と可変幅の場合があります。可変幅キャラクタ・セットを使用して CLOB 列にデータを挿入すると、Oracle はそのデータを Unicode に変換してからデータベースに格納します。

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 属性を持ったオブジェクト型は作成できませんが、オブジェクト型のメソッドで NCLOB パラメータを指定することはできます。

> **関連項目：** 各国語キャラクタ・セット・データと Unicode キャラクタ・セットの詳細は、『Oracle9*i* 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。BFILE は最大 4GB のバイナリ・データを格納できます。

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（順次読込みはサポートしていない）、トランザクションには参加しません。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 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

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は、大量のバイナリ・データを格納するには BLOB および BFILE データ型を使用してください。

RAW データ型と LONG RAW データ型は、Oracle が解釈しない（異なるシステム間での移動時には変換されない）データに使用します。これらのデータ型は、バイナリ・データやバイト文字列用に用意されています。たとえば、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、RAW または LONG RAW データの送信時に、Oracle Net Services（ユーザー・セッションをインスタンスに接続）とインポート／エクスポート・ユーティリティでは文字変換は実行されません。それとは対照的に、Oracle Net Services とインポート／エクスポート・ユーティリティは、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（ALTER SESSION 文の NLS_LANGUAGE パラメータで設定）が異なる場合に、これらのキャラクタ・セット間で CHAR、VARCHAR2 および LONG データを自動的に変換します。

Oracle が RAW または LONG RAW データと CHAR データとの間の自動変換を実行する場合、バイナリ・データは 16 進数で表され、これらの 16 進文字は 1 文字で RAW データ 4 ビット分のデータを表します。たとえば、1 バイトの RAW データのビットが 11001011 の場合、これは 'CB' として表示または入力されます。

LONG RAW データに索引を付けることはできませんが、RAW データには索引を付けることができます。

関連項目： LONG RAW データ型に関する他の制限の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

ROWID および UROWID データ型

ROWID データ型には、データベース内の各行のアドレス（**ROWID**）が格納されます。

- **物理 ROWID** は、通常の表（索引構成表を除く）、クラスタ化表、表パーティションとサブパーティション、索引および索引パーティションとサブパーティションに行のアドレスを格納します。
- **論理 ROWID** は、索引構成表に行のアドレスを格納します。

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、COMPATIBLE 初期化パラメータの値を 8.1 以上に設定する必要があります。

関連項目： 12-23 ページ「Oracle 以外のデータベースの ROWID」

ROWID 疑似列

Oracle データベースのそれぞれの表は、ROWID という名前の**疑似列**を内部的に持っています。SQL*Plus で SELECT * FROM ... 文または DESCRIBE ... 文を実行して表の構造のリストを表示しても、この疑似列は表示されません。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 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 句で使用する）、その ROWID はデータベースに格納されているわけではなく、またそのデータベースのデータでもありません。ROWID データ型の列を含む表を作成することはできますが、そのような列の ROWID 値が有効であるかどうかは保証できません。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： 12-21 ページ「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。その行が存在するかぎり、ROWID は変化しないことが保証されます。このようなパフォーマンスと安定性により、アプリケーションが行の集合を選択し、それに対してなんらかの操作を実行し、選択した一部の行に更新などの目的で再度アクセスする場合に、ROWID が役立ちます。

クラスタ化されてない表のすべての行には、行の行断片（行が複数の行断片にわたって連鎖している場合は、最初の行断片）の物理アドレスに対応する、一意の ROWID が割り当てられています。クラスタ化表の場合、同じデータ・ブロックにある異なる表の行の ROWID が同一の場合もあります。

行に割り当てられている ROWID は、その行がインポートおよびエクスポート・ユーティリティを使用してエクスポートおよびインポートされないかぎり変更されません。表から行を削除し、その操作を含むトランザクションがコミットされると、削除された行に対応していた ROWID は、後続のトランザクションで挿入される行に割り当てられることがあります。

物理 ROWID のデータ型は、次の 2 つの書式のどちらかです。

- **拡張 ROWID** 形式は、表領域関連のデータ・ブロック・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。
- Oracle7 以前のリリースで開発されたアプリケーションとの下位互換性を保つために、**制限付き ROWID** 形式も使用できます。

拡張 ROWID

拡張 ROWID は、それぞれの選択された行の物理アドレスを 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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

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。このため、同じブロック番号の行が、同じ表領域の 2 つの異なるデータ・ファイルに存在することもあります。
- その行を含むブロックの中の**行**（例では行 0、1、2）。各ブロックの行番号は、必ず 0 で始まります。
- その行を含む**データ・ファイル**（例ではファイル 1）。すべてのデータベースの最初のデータ・ファイルは必ずファイル 1 であり、ファイル番号はデータベース内で一意です。

ROWID の使用例

関数 SUBSTR を使用して、ROWID のデータをコンポーネントに分割できます。たとえば、SUBSTR を使用して拡張 ROWID を 4 つのコンポーネント（データベース・オブジェクト、ファイル、ブロックおよび行）に分割します。

```
SELECT ROWID,
       SUBSTR(ROWID,1,6) "OBJECT",
       SUBSTR(ROWID,7,3) "FIL",
       SUBSTR(ROWID,10,6) "BLOCK",
       SUBSTR(ROWID,16,3) "ROW"
       FROM products;

ROWID               OBJECT  FIL  BLOCK   ROW
------------------  ------  ---  ------  ----
AAAA8mAALAAAAQkAAA  AAAA8m  AAL  AAAAQk  AAA
AAAA8mAALAAAAQkAAF  AAAA8m  AAL  AAAAQk  AAF
AAAA8mAALAAAAQkAAI  AAAA8m  AAL  AAAAQk  AAI
```

または、SUBSTR を使用して制限付き ROWID を 3 つのコンポーネント（ブロック、行およびファイル）に分割します。

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE",
       SUBSTR(ROWID,1,8) "BLOCK",
       SUBSTR(ROWID,10,4) "ROW"
       FROM products;

ROWID               FILE  BLOCK     ROW
------------------  ----  --------  ----
00000DD5.0000.0001  0001  00000DD5  0000
00000DD5.0001.0001  0001  00000DD5  0001
00000DD5.0002.0001  0001  00000DD5  0002
```

ROWID は、表データの物理的な格納に関する情報を明らかにするのに便利です。たとえば、表の行の物理的な位置を確認する場合（表のストライプ化の場合など）、次のように拡張 ROWID を問い合せると、指定の表の行がいくつのデータ・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：

- 『Oracle9*i* SQL リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』

ROWID の使用例の詳細は、次のマニュアルを参照してください。

ROWID の使用方法

Oracle は、ROWID を内部的に使用して索引を構築します。索引のそれぞれのキーは、高速アクセスのために、対応する行のアドレスを指す ROWID に結び付けられています。エンド・ユーザーとアプリケーション開発者は、次のような重要な用途に ROWID を使用することもできます。

- ROWID は特定の行にアクセスする最も高速な方法です。
- ROWID は表の編成を把握するために使用できます。
- ROWID は表の中の行の一意識別子です。

DML 文の中で ROWID を使用する前に、ROWID が変更されないことを検証して確認する必要があります。目的の行を、削除されないようにロックしてください。無効な ROWID を使用してデータを要求すると、状況によっては、文が失敗します。

また、ROWID データ型を使用して定義した列を持つ表を作成することもできます。たとえば、データ型 ROWID の列を持つ例外表を定義して、整合性制約に違反するデータベース行の ROWID を格納できます。ROWID データ型を使用して定義されている列の動作は、表の他の列と類似しています。たとえば、値を更新できます。データ型 ROWID として定義されている列のそれぞれの値を、適切な列データとして格納するには 6 バイト必要です。

論理 ROWID

索引構成表の中の行は、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 は、これらの論理 ROWID を使用して、索引構成表の 2 次索引を構築します。

2 次索引に使用される論理 ROWID ごとに、**物理推測**を含めることができます。これは、推測時、つまり、2 次索引の作成時または再作成時に、索引構成表の中の行のブロック位置を識別するものです。

Oracle は、推測を使用して、全キー検索を回避し、リーフ・ブロックを直接プローブできます。これにより、不揮発性の索引構成表の ROWID アクセスで、通常の表の物理 ROWID アクセスに匹敵するパフォーマンスを得られることが保証されます。ただし、揮発性の表では、推測が陳腐化するとプローブが失敗することがあります。その場合は、主キー検索を実行する必要があります。

2 つの論理 ROWID の値は、同じ主キー値および異なる推測を持つ場合に同一と見なされます。

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` 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： 12-17 ページ「ROWID および UROWID データ型」

論理 ROWID での推測

行の物理位置が変化した場合、論理 ROWID は推測が含まれていても引き続き有効ですが、推測は陳腐化し、その行へのアクセスが低速になることがあります。推測情報を動的に更新することはできません。ただし、索引構成表の 2 次索引に対し、索引を再作成して最新の推測を取得できます。索引構成表の 2 次索引を再作成するには、通常の表の索引を再作成する場合とは異なり、実表を読み込む必要があるため注意してください。

Oracle で推測が不用意に使用されないように、DBMS_STATS パッケージまたは ANALYZE 文で索引統計を収集し、推測の陳腐化を追跡する必要があります。特に、UROWID 列に推測を使用した ROWID を永続的に格納し、その ROWID を後から取り出して行のフェッチに使用するアプリケーションの場合は、この作業が重要になります。

DBMS_STATS パッケージまたは ANALYZE 文を使用して索引の統計を収集すると、既存の推測がまだ有効かどうかがチェックされ、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、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

Oracle 以外のデータベースの ROWID

Oracle データベース・アプリケーションは、SQL*Connect または Oracle Transparent Gateway を使用して、Oracle 以外のデータベース・サーバーに対しても実行可能です。そのような場合、ROWID の形式は、その Oracle 以外のシステムの特性に応じて異なります。また、VARCHAR2 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。
- 12-17 ページの「ROWID および UROWID データ型」

ANSI データ型、DB2 データ型および SQL/DS データ型

表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 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、表 12-2 および表 12-3 に示す変換に基づいて、その列のデータを Oracle データ型に格納します。

表 12-2 ANSI データ型から Oracle データ型への変換

ANSI SQL データ型	Oracle データ型
CHARACTER(n)	CHAR(n)
CHAR(n)	
CHARACTER VARYING(n)	VARCHAR(n)
CHAR VARYING(n)	
NATIONAL CHARACTER(n)	NCHAR(n)
NATIONAL CHAR(n)	
NCHAR(n)	
NATIONAL CHARACTER VARYING(n)	NVARCHAR2(n)
NATIONAL CHAR VARYING(n)	
NCHAR VARYING(n)	
NUMERIC(p,s)	NUMBER(p,s)
DECIMAL(p,s)[a]	
INTEGER	NUMBER(38)
INT	
SMALLINT	
FLOAT(b)[b]	NUMBER
DOUBLE PRECISION[c]	
REAL[d]	

[a] NUMERIC および DECIMAL データ型では、固定小数点数のみを指定できます。これらのデータ型の場合、s はデフォルトの 0（ゼロ）に設定されます。

[b] FLOAT データ型は、2 進精度が b の浮動小数点数です。このデータ型のデフォルト精度は、2 進の場合は 126、10 進の場合は 38 です。

[c] DOUBLE PRECISION データ型は、2 進精度が 126 の浮動小数点数です。

[d] REAL データ型は、2 進精度が 63 または 10 進精度が 18 の浮動小数点数です。

表 12-3 SQL/DS および DB2 データ型から Oracle データ型への変換

SQL/DS または DB2 データ型	Oracle データ型
CHARACTER(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)
LONG VARCHAR(n)	LONG
DECIMAL(p,s)[a]	NUMBER(p,s)
INTEGER	NUMBER(38)
SMALLINT	
FLOAT(b)[b]	NUMBER

[a] DECIMAL データ型では、固定小数点数のみを指定できます。このデータ型の場合、*s* はデフォルトの 0（ゼロ）に設定されます。

[b] FLOAT データ型は、2 進精度が *b* の浮動小数点数です。このデータ型のデフォルト精度は、2 進の場合は 126、10 進の場合は 38 です。

次の SQL/DS データ型と DB2 データ型は、対応する Oracle データ型がないため、列の定義に使用しないでください。

- GRAPHIC
- LONG VARGRAPHIC
- VARGRAPHIC
- TIME

TIME 型のデータは、Oracle の DATE データとしても表せることに注意してください。

XML データ型

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 を表やビューの列のデータ型として使用したり、XMLType の変数を 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 および Java で使用したり、JDBC や OCI を介して使用します。

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：

- 『Oracle9*i* XML Developer's Kits ガイド - XDK』
- 『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』
- Oracle Advanced Queuing での XMLType の使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 第 1 章「Oracle サーバーの基礎知識」

URI データ型

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：『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』

データ変換

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、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。使用される関数には、次のものがあります。

```
TO_NUMBER()
TO_CHAR()
TO_NCHAR()
TO_DATE()
TO_CLOB()
TO_NCLOB()
CHARTOROWID()
ROWIDTOCHAR()
ROWIDTONCHAR()
HEXTORAW()
RAWTOHEX()
RAWTONHEX()
REFTOHEX()
```

関連項目： 暗黙的なデータ型変換の規則は、『Oracle9*i* SQL リファレンス』を参照してください。

13

オブジェクト・データ型とオブジェクト・ビュー

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。

この章の内容は、次のとおりです。

オブジェクト・データ型の概要

リレーショナル・データベース管理システム（RDBMS）は、ビジネス・データを管理する標準的なツールです。このツールを使用すると、世界中で毎日行われている幾百万ものビジネスについての膨大なデータに、信頼性をもってアクセスできます。

Oracle は、**オブジェクト・リレーショナル**・データベース管理システム（ORDBMS）です。これは、ユーザーが各種のデータを追加定義し（データの構造とその構造に対する操作の両方を指定する）、リレーショナル・モデル内でそれらのデータ型を使用できることを意味しています。このアプローチにより、データベースに格納されているデータにさらに価値が付加されます。オブジェクト・データ型によって、アプリケーション開発者は、イメージ、オーディオおよびビデオなどの複合データを容易に処理できます。オブジェクト型では、構造化されたビジネス・データが自然な形で格納され、アプリケーションもそれらのデータを自然な形で取り出すことができます。これにより、オブジェクト指向プログラミング技法を使用して開発されたアプリケーションを効率よく処理できます。

複合データ・モデル

Oracle サーバーでは、複合ビジネス・モデルを SQL で定義し、それをデータベース・スキーマの一部にすることができます。データを管理し共有するアプリケーションに入っている必要があるのは、データ・ロジックではなく、アプリケーション・ロジックのみです。

複合データ・モデルの例

たとえば、発注情報を使用して、購買、買掛管理、出荷および売掛管理の機能を編成しているとします。

1 件の発注情報には、関連するサプライヤまたは顧客と、不特定数の品目が含まれています。さらに、アプリケーションでは、発注に関してステータス情報を動的に計算する機能が必要になることがあります。たとえば、出荷済みまたは未出荷の品目の現行の数量が必要になることがあります。

アプリケーションですべての発注データのテンプレートとして使用することになる、**オブジェクト型**と呼ばれるスキーマ・オブジェクトを定義する方法は、この章の後半で説明します。オブジェクト型では、発注などの構造化されたデータ単位を形成する個々の要素を指定します。この要素のことを**属性**と呼びます。属性そのものが、別の構造化されたデータ単位になっていることもあります（明細項目リストなど）。オブジェクト型では、1 件の発注の合計金額を計算するなど、そのデータ単位に対して実行できる操作も指定します。この操作のことを**メソッド**と呼びます。

テンプレートと一致する発注情報を作成し、数値や日付とまったく同じように表の列に格納できます。

また、発注情報を**オブジェクト表**に格納することもできます。その場合、表の各行が 1 件の発注情報に対応し、表の各列が発注情報の属性に対応します。

発注情報の構造と動作のロジックはスキーマに入っているため、アプリケーションはその詳細を知る必要はなく、最新の変更内容を反映する必要もありません。

Oracle では、オブジェクト型に関するスキーマ情報を使用して、実質的な伝送効率を向上させます。クライアント側のアプリケーションは、サーバーに発注情報を要求したとき、すべての関連データを 1 回の伝送で受信できます。その後、アプリケーションは、データが格納されている位置や実現方法の詳細を知らなくても、サーバーからのそれ以上の伝送なしで、関連データ項目間をナビゲートできます。

マルチメディア・データ型

数値、日付および文字などの基本的なデータ型の管理を最適化することにより、データベース・システムはかなりの程度まで効率が良くなります。また、値の比較、値の分散状況の判別、効率的な索引の作成およびその他の最適化などを実行するための機能があります。

これらの基本的なデータ型に適合しない重要なビジネス・エンティティの例には、テキスト、ビデオ、サウンド、グラフィックスおよび空間データなどがあります。Oracle Enterprise Edition は、これらの複合データ型のモデル化と実現をサポートしています。

オブジェクト・データ型のカテゴリ

オブジェクト・データ型には、次の 2 つのカテゴリがあります。

- オブジェクト型
- コレクション型

オブジェクト・データ型では、組込みデータ型と他のユーザー定義データ型を、アプリケーション・データの構造と動作をモデル化するデータ型のビルディング・ブロックとして使用します。

オブジェクト型はスキーマ・オブジェクトです。これらのオブジェクトの使用は、他のスキーマ・オブジェクトと同様の管理制御を受けます。

関連項目：

- 第 12 章「システム固有のデータ型」
- 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』

オブジェクト型

オブジェクト型は、アプリケーション・プログラムが取り扱う実社会のエンティティ（発注など）を抽象化したものです。オブジェクト型は、次のような 3 種類のコンポーネントのあるスキーマ・オブジェクトです。

- **名前** − これは、そのスキーマ内でオブジェクト型を一意に識別するものです。
- **属性** − 実社会のエンティティの構造と状態をモデル化します。属性は、組込み型か他のユーザー定義型です。
- **メソッド** − PL/SQL や Java で作成されてデータベースに格納された関数やプロシージャ、または、C などの言語で作成されて外部に格納された関数やプロシージャのことです。メソッドは、実社会のエンティティに対してアプリケーションが実行できる操作を実装します。

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

発注情報の例

ここで説明する例では、external_person、lineitem および purchase_order というオブジェクト型を定義する方法を示します。

オブジェクト型 external_person および lineitem は、組込み型の属性を持ちます。オブジェクト型 purchase_order の構造はさらに複雑で、実際の発注情報の構造と厳密に一致するものになっています。

purchase_order の属性は、id、contact および lineitems です。属性 contact はオブジェクトで、属性 lineitems はネストした表です。

```
CREATE TYPE external_person AS OBJECT (
  name        VARCHAR2(30),
  phone       VARCHAR2(20) );

CREATE TYPE lineitem AS OBJECT (
  item_name   VARCHAR2(30),
  quantity    NUMBER,
  unit_price  NUMBER(12,2) );

CREATE TYPE lineitem_table AS TABLE OF lineitem;

CREATE TYPE purchase_order AS OBJECT (
  id          NUMBER,
  contact     external_person,
  lineitems   lineitem_table,

  MEMBER FUNCTION
  get_value   RETURN NUMBER );
```

ここに記載したコード例は簡略化してあります。get_value メソッドの本体を指定する方法は示していません。また、実際の発注情報の複雑さも完全には反映させていません。

オブジェクト型はテンプレートです。オブジェクト型を定義しても、記憶域は割り当てられません。SQL 文で NUMBER や VARCHAR2 などのデータ型を使用可能な場所であれば、ほとんどの場所に lineitem、external_person または purchase_order を使用できます。

たとえば、次のようなリレーショナル表を定義して、発注先の担当者（contact）を追跡し記録できます。

```
CREATE TABLE contacts (
  contact     external_person
  date        DATE );
```

contacts 表は、列の 1 つがオブジェクト型で定義されているリレーショナル表です。リレーショナル表の列を占めるオブジェクトを**列オブジェクト**と呼びます。

関連項目：

- 13-11 ページ「ネストした表の説明」
- 13-8 ページ「行オブジェクトと列オブジェクト」
- 発注例の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

メソッドの型

オブジェクト型のメソッドは、オブジェクトの動作をモデル化したもので、次の 3 つのカテゴリに大別できます。

- **メンバー**・メソッドは、最初のパラメータとして常に暗黙的 SELF パラメータを持ち、その型にオブジェクト型を含むファンクションまたはプロシージャです。
- **静的**メソッドは、暗黙的 SELF パラメータを持たないファンクションまたはプロシージャです。この種のメソッドは、TYPE_NAME.METHOD() のように、メソッドを型名で修飾して起動できます。静的メソッドは、ユーザー定義のコンストラクタやキャスト・メソッドを指定するときに役立ちます。
- **比較**メソッドは、オブジェクトのインスタンスを比較するときに使用します。

Oracle は、PL/SQL、Java および C で型メソッドの実装の選択をサポートしています。

この例の場合、purchase_order には get_value というメソッドがあります。それぞれの発注情報オブジェクトには、専用の get_value メソッドがあります。たとえば、*x* と *y* が発注情報オブジェクトを保持する PL/SQL 変数、*w* と *z* が数値を保持する変数である場合、次の 2 つの文による *w* と *z* の結果値は異なる可能性があります。

```
w = x.get_value();
z = y.get_value();
```

これらの文を実行すると、*w* には変数 *x* で表される発注情報の値、*z* には変数 *y* で表される発注情報の値が代入されます。

x.get_value() の項により、メソッド get_value が起動されます。メソッド定義にパラメータを組み込むこともできますが、get_value メソッドにパラメータは必要ありません。起動時に結び付けられたオブジェクトの属性からすべての引数を検索できるためです。つまり、1 番目のサンプル文では発注情報 *x* の属性を使用して値を計算します。2 番目のサンプル文では、発注情報 *y* の属性を使用して値を計算します。これを**自己参照的**メソッドの起動と呼びます。

すべてのオブジェクト型には、特定のオブジェクトに結び付けられない、暗黙に定義されるメソッド、つまりオブジェクト型のコンストラクタ・メソッドも 1 つあります。

オブジェクト型のコンストラクタ・メソッド すべてのオブジェクト型には、オブジェクト型の仕様に基づいて新しいオブジェクトを作成する、システム定義の**コンストラクタ・メソッド**があります。コンストラクタ・メソッドの名前は、そのオブジェクト型の名前と同じであり、コンストラクタ・メソッドのパラメータの名前と型は、オブジェクト型の属性の名前と型です。コンストラクタ・メソッドはファンクションです。新しいオブジェクトを値として戻します。

たとえば、次のような式があります。

```
purchase_order(
  1000376,
  external_person ("John Smith","1-800-555-1212"),
  NULL )
```

この式は、次のような属性を持つ発注情報オブジェクトを表します。

```
id         1000376
contact    external_person("John Smith","1-800-555-1212")
lineitems  NULL
```

式 external_person ("John Smith","1-800-555-1212") により、オブジェクト型 external_person のコンストラクタ・ファンクションが起動されます。これにより戻されるオブジェクトが、発注情報の contact 属性になります。

また、独自のコンストラクタ・ファンクションを定義し、システムによって各オブジェクト型に暗黙的に定義されるコンストラクタ・ファンクションのかわりに使用することもできます。

> **関連項目：**『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』

比較メソッド メソッドは、オブジェクトの比較でも役割を果たします。Oracle には、特定の組込みタイプの 2 つのデータ項目（たとえば 2 つの数値）を比較し、一方が他方より大きい、等しい、または小さいかどうかを判別する機能があります。しかし、Oracle では、定義者がさらに指示を与えないかぎり、任意のユーザー定義型の 2 つの項目を比較することはで

きません。Oracle は、特定のオブジェクト型のオブジェクト間に順序関係を定義するために 2 つの方法を用意しています。マップ・メソッドとオーダー・メソッド です。

マップ・メソッドでは、組込み型を比較する Oracle の機能を使用します。たとえば、`rectangle`（四角形）というオブジェクト型に `height`（高さ）および `width`（幅）という属性を定義した場合を考えます。area（面積）というマップ・メソッドを定義して、数値、つまり四角形の `height` 属性と `width` 属性の積を戻すようにします。この場合は、面積によって 2 つの四角形を比較できます。

オーダー・メソッドは、より一般的な方法です。オーダー・メソッドは、独自の内部論理を使用して、特定のオブジェクト型に属する 2 つのオブジェクトを比較します。それは、順序関係を符号化した値を戻します。たとえば、最初のオブジェクトのほうが小さければ -1 を戻し、どちらも同じ大きさであれば 0（ゼロ）を戻し、最初のオブジェクトのほうが大きければ 1 を戻す、などです。

たとえば、`address`（住所）というオブジェクト型に `street`（番地）、`city`（市町村）、`state`（都道府県）、`zip`（郵便番号）という属性を定義した場合を考えます。アプリケーションで住所を比較して、**より大きい**か**より小さい**かを判別することは無意味ですが、2 つの住所が等しいかどうかを判別するには、複雑な計算を必要とする場合が考えられます。

オブジェクト型を定義する際、マップ・メソッドかオーダー・メソッドのどちらかを指定できますが、両方は指定できません。オブジェクト型に比較メソッドがない場合、Oracle はその型の 2 つのオブジェクト間の大小関係を判別できません。ただし、その型の 2 つのオブジェクトが等しいかどうかの判別は試行できます。

比較メソッドを持たない型の 2 つのオブジェクトを比較する場合、Oracle は、対応する属性を比較します。

- すべての属性が NULL 以外であり、等しい場合、Oracle はその 2 つのオブジェクトが等しいとレポートします。
- 2 つのオブジェクトに等しくない NULL 以外の値がある場合、Oracle はその 2 つのオブジェクトが等しくないとレポートします。
- その他の場合、Oracle は比較できないことをレポートします（NULL）。

> **関連項目：** 比較メソッドを指定して使用する例は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

オブジェクト表

オブジェクト表は特殊な種類の表で、オブジェクトを保持し、各オブジェクトの属性のリレーショナル・ビューを提供します。

たとえば、次の文は、以前に定義した `external_person` 型のオブジェクトのオブジェクト表を定義します。

```
CREATE TABLE external_person_table OF external_person;
```

Oracle では、次の 2 つの方法でこの表を表示できます。

- 単一列表。それぞれのエントリが external_person オブジェクトになっています。
- 複数列表。オブジェクト型 external_person のそれぞれの属性、つまり name と phone が列を 1 つずつ占有します。

たとえば、次の命令を実行できます。

```
INSERT INTO external_person_table VALUES (
       "John Smith",
       "1-800-555-1212" );

SELECT VALUE(p) FROM external_person_table p
        WHERE p.name = "John Smith";
```

最初の命令は、複数列表としての external_person_table に external_person オブジェクトを挿入します。2 番目の命令は、単一列表として external_person_table から選択します。

行オブジェクトと列オブジェクト オブジェクト表に表示されるオブジェクトのことを**行オブジェクト**と呼びます。表の列に表示されるオブジェクト、または他のオブジェクトの属性として表示されるオブジェクトのことを**列オブジェクト**と呼びます。

オブジェクト識別子

オブジェクト表のすべての行オブジェクトには、論理オブジェクト識別子（OID）が対応付けられています。デフォルトで、各行オブジェクトには OID として長さ 16 バイトの一意のシステム生成識別子が割り当てられます。

オブジェクト表の OID 列は、非表示列です。OID 値そのものはオブジェクト・リレーショナル・アプリケーションにほとんど意味を持ちませんが、Oracle はこの値を使用して行オブジェクトへのオブジェクト参照を組み立てます。アプリケーションは、オブジェクトのフェッチとナビゲートに使用されるオブジェクト参照のみに注意する必要があります。

行オブジェクトの OID の目的は、そのオブジェクトをオブジェクト表内で一意に識別することです。そのために、Oracle はオブジェクト表の OID 列に索引を暗黙的に作成してメンテナンスします。システム生成の一意識別子には、分散およびレプリケート環境でオブジェクトが明確に識別されること以外にも、多数の利点があります。

主キーに基づくオブジェクト識別子 システム生成のグローバル一意識別子が提供する機能性を必要としないアプリケーションでは、各オブジェクトとともに余分の 16 バイトを格納し、その索引をメンテナンスしても、効率的でない場合があります。Oracle では、行オブジェクトのオブジェクト ID としてその主キー値を指定する方法を選択できます。

また、主キーに基づく識別子には、より効率的かつ容易にオブジェクト表をロードできるという利点もあります。これに対して、システム生成のオブジェクト識別子は、特にその参照も永続的に格納される場合には、なんらかのユーザー指定キーを使用して再マップする必要があります。

オブジェクト・ビューの説明

オブジェクト・ビューは、仮想的なオブジェクト表です。オブジェクト・ビューの行は、行オブジェクトです。Oracle は、永続的には格納しないオブジェクト識別子を、基礎を形成する表やビューの主キーから具象化します。

関連項目： 13-22 ページ「オブジェクト・ビューの概要」

REF

リレーショナル・モデルの場合、外部キーは多対 1 リレーションシップを表します。Oracle オブジェクト型では、多対 1 リレーションシップで 1 の側が行オブジェクトである場合に、このリレーションシップを表すための効率的な方法が提供されています。

Oracle は、指定したオブジェクト型の行オブジェクトの参照をカプセル化するために、REF という組込みデータ型を提供します。モデル化の観点から見ると、REF を使用すると 2 つの行オブジェクト間の関連付けを獲得できます。Oracle は、この種の REF を組み立てるためにオブジェクト識別子を使用します。

REF を使用して、参照先のオブジェクトを検査および更新できます。また、REF によって、参照先のオブジェクトのコピーを取得できます。REF に対して実行できる変更は、同じオブジェクト型の異なるオブジェクトへの参照に内容を置き換えるか、NULL 値を割り当てるという変更のみです。

有効範囲付 REF 列型、コレクション要素またはオブジェクト型の属性を REF として宣言する際に制約を適用し、指定したオブジェクト表への参照のみを含めることができます。このような REF は**有効範囲付** REF と呼びます。有効範囲付 REF は記憶領域が少なく、有効範囲なし REF よりも効率的にアクセスできます。

参照先がない REF REF で識別されるオブジェクトが削除されたり権限が変更されて、そのオブジェクトを使用できなくなる可能性があります。このような REF を**参照先がない** REF と呼びます。Oracle の SQL では、この条件について REF をテストする述語（IS DANGLING）を提供しています。

REF の参照解除 REF で参照しているオブジェクトにアクセスすることを、REF の**参照解除**と呼びます。Oracle には、そのための DEREF 演算子が用意されています。参照先がない REF への参照を解除すると、結果は NULL オブジェクトになります。

Oracle は、REF の**暗黙的参照解除**を提供しています。たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (
  name    VARCHAR2(30),
  manager REF person );
```

x が PERSON 型のオブジェクトを表している場合、次の式について考えます。

```
x.manager.name
```

x の manager 属性が参照する person オブジェクトの name 属性を含む文字列を表します。つまり、前述の式は、次の式の短縮形です。

```
y.name, where y = DEREF(x.manager)
```

REF の取得 行オブジェクトへの REF は、オブジェクト表からそのオブジェクトを選択し、REF 演算子を適用すると取得できます。たとえば、識別番号 1000376 の発注情報への REF は、次のようにして取得できます。

```
DECLARE OrderRef REF to purchase_order;

SELECT REF(po) INTO OrderRef
               FROM purchase_order_table po
               WHERE po.id = 1000376;
```

> **関連項目：** REF の使用例は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

コレクション型

それぞれのコレクション型は、すべて同じデータ型の不特定数の要素で構成されるデータ単位を記述します。コレクション型には、**配列型**と**表型**があります。

配列型および表型は、スキーマ・オブジェクトです。それぞれに対応するデータ単位を、**VARRAY** および**ネストした表**と呼びます。混乱の恐れがない場合は、コレクション型のことを単に VARRAY およびネストした表と呼ぶこともよくあります。

コレクション型には、コンストラクタ・メソッドがあります。コンストラクタ・メソッドの名前はコレクション型の名前と同じで、その引数は新しいコレクションの要素をカンマで区切ったリストです。コンストラクタ・メソッドはファンクションです。新しいコレクションを値として返します。

コレクション型の名前に空のカッコを付けた式は、その型を持つ空のコレクションを作成するコンストラクタ・メソッドのコールを表します。空のコレクションは、NULL コレクションとは異なります。

VARRAY

配列とは、データ**要素**を順番に並べた集合のことです。特定の配列のすべての要素は、同じデータ型で構成されます。各要素には**索引**があり、これは配列内におけるその要素の位置に対応する番号です。

配列内の要素の数が、配列の**サイズ** になります。Oracle の配列のサイズは可変であるため、VARRAY と呼ばれます。配列型を宣言するときは、最大サイズを指定する必要があります。

たとえば、次のような文で配列型を宣言します。

```
CREATE TYPE prices AS VARRAY(10) OF NUMBER(12,2);
```

これは、prices 型の VARRAY で、最大 10 個までの要素を入れることができ、各要素のデータ型は NUMBER(12,2) です。

配列型を作成しても、領域は割り当てられません。単にデータ型を定義しているだけです。このデータ型は、次の用途に使用できます。

- リレーショナル表の列のデータ型
- オブジェクト型の属性
- PL/SQL の変数、パラメータ、ファンクションの戻り型

通常、VARRAY はひとまとめに格納されます。つまり、行に入っているその他のデータと同じ表領域に格納されます。ただし、サイズが十分に大きい場合には、Oracle はこれを BLOB として格納します。

> **関連項目：** VARRAY の使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

ネストした表の説明

ネストした表は、すべて同じデータ型のデータ**要素**を順不同で 並べた集合です。この表には単一の列があり、この列の型は組込み型かオブジェクト型です。オブジェクト型の場合は、この表を、オブジェクト型の各属性に対応する列のある、複数列の表とみなすこともできます。Oracle9*i* またはそれ以上への互換性を設定すれば、ネストした表は他のネストした表を含めることができます。

たとえば、発注情報の例では、次の文により、品目を入れるネストした表のために使用する表型を宣言します。

```
CREATE TYPE lineitem_table AS TABLE OF lineitem;
```

表型を定義しても、領域は割り当てられません。単に型を定義しているだけです。この型は、次の用途に使用できます。

- リレーショナル表の列のデータ型
- オブジェクト型の属性
- PL/SQL の変数、パラメータ、ファンクションの戻り型

表型が、リレーショナル表の列の型、またはオブジェクト表の基礎を形成するオブジェクト型の属性として使用された場合、Oracle はネストした表のすべてのデータを単一の表に格納します。Oracle は、その表を、それを含むリレーショナル表またはオブジェクト表と対応付けます。たとえば、次の文は、オブジェクト型 purchase_order に対してオブジェクト表を定義します。

```
CREATE TABLE purchase_order_table OF purchase_order
   NESTED TABLE lineitems STORE AS lineitems_table;
```

2 番目の行は、purchase_order_table にあるすべての purchase_order オブジェクトの lineitems 属性を格納する表として、lineitems_table を指定しています。

ネストした表の要素に個々にアクセスする簡便な方法は、ネストしたカーソルを使用することです。

関連項目：

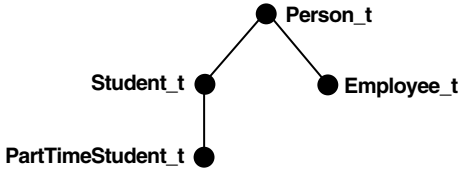
- ネストしたカーソルの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- ネストした表の使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

型の継承

オブジェクト型は、既存オブジェクト型のサブタイプとして作成できます。単一継承モデルがサポートされています。サブタイプが導出可能な親タイプは 1 つのみです。型は、その直接のスーパータイプの属性とメソッドをすべて継承します。新しい属性とメソッドを追加できます。また、継承したどのメソッドに対してもオーバーライドが可能です。

図 13-1 は、Person_t の下に作成された 2 つのサブタイプ Student_t と Employee_t を示します。

図 13-1 型の階層



また、サブタイプは、その下に別のサブタイプを定義して型の階層を作成すると、さらに細分化できます。前述の図では、PartTimeStudent_t はサブタイプ Student_t から導出されています。

FINAL 型および NOT FINAL 型

サブタイプを作成する場合は、型宣言で NOT FINAL キーワードが必要です。デフォルトでは、型が FINAL であるため、その型にはサブタイプは作成されません。これによって、下位互換性が維持されています。

NOT FINAL オブジェクト型の作成例

```
CREATE TYPE Person_t AS OBJECT
( ssn NUMBER,
  name VARCHAR2(30),
  address VARCHAR2(100)) NOT FINAL;
```

Person_t は NOT FINAL 型であることが宣言されています。このため、Person_t のサブタイプの定義が可能です。

FINAL 型は、NOT FINAL に変更できます。さらに、サブタイプを持たない NOT FINAL 型は、FINAL に変更できます。

NOT INSTANTIABLE 型およびメソッド

型は NOT INSTANTIABLE を宣言できます。これは、その型にコンストラクタ（デフォルトまたはユーザー定義）がないことを意味します。このため、この型のインスタンスは作成できません。この型の代表的な使用例は、次のような型についてインスタンス化可能なサブタイプを定義することです。

```
CREATE TYPE Address_t AS OBJECT(...) NOT INSTANTIABLE NOT FINAL;
CREATE TYPE USAddress_t UNDER Address_t(...);
CREATE TYPE IntlAddress_t UNDER Address_t(...);
```

ある型のメソッドは NOT INSTANTIABLE に宣言できます。NOT INSTANTIABLE としてメソッドを宣言することは、その型がそのメソッドを実装していないことを意味します。さらに、インスタンス化が不可能なメソッドを含む型は、必ず NOT INSTANTIABLE に宣言する必要があります。

次に例を示します。

```
CREATE TYPE T AS OBJECT
(
  x NUMBER,
  NOT INSTANTIABLE MEMBER FUNCTION func1() RETURN NUMBER
) NOT INSTANTIABLE;
```

NOT INSTANTIABLE 型のサブタイプは、そのスーパータイプのインスタンス化が不可能なメソッドをオーバーライドできます。また、具体的な実装を提供します。インスタンス化が不可能なメソッドが残っている場合、そのサブタイプも必ず NOT INSTANTIABLE に宣言する必要があります。

インスタンス化が不可能なサブタイプは、インスタンス化可能なスーパータイプの下に定義できます。インスタンス化が不可能な型を FINAL に宣言することはできません。

関連項目：『PL/SQL ユーザーズ・ガイドおよびリファレンス』

ユーザー定義集計関数

Oracle は、MAX、MIN および SUM などの集計関数の基本セットをサポートしています。この他に、ユーザー定義の集計操作ロジックによる新しい集計関数を実装できるメカニズムも用意されています。

ユーザー定義の集計関数を使用する理由

ユーザー定義集計関数（UDAG）とは、ユーザーが指定した集計操作セマンティクスによる集計関数を言います。ユーザーは新しい集計関数を作成できます。また、一連のルーチンから集計操作ロジックも提供できます。いったん集計関数を作成すると、ユーザー定義集計関数は、組込み集計と同じように、SQL DML 文で使用できます。Oracle サーバーは、ユーザーが提供する集計ルーチンを適切に起動して、UDAG を評価します。

データベースは、イメージ、空間、オーディオ、ビデオなどの複雑なデータの格納が徐々に増加します。一般に複雑なデータは、オブジェクト型、不透明型または LOB を使用してデータベースに格納されます。ユーザー定義の集計は主に、データの新しいドメインなどに集計操作を指定する場合に便利です。

さらに、UDAG を使用すると、財務や科学関係のアプリケーションで使用する従来のスカラー・データ型について、新しい集計関数を作成できます。集計のすべてのフォームについて固有のサポートを提供できないため、アプリケーション開発者が新しい集計関数を追加できるように柔軟性に富んだメカニズムを提供することが望まれます。

関連項目：

- ユーザー定義集計の実装に関する詳細は、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。
- データ・ウェアハウスにおける UDAG の使用方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。
- 不透明型の詳細は、第 12 章「システム固有のデータ型」を参照してください。

UDAG の作成と使用

ユーザー定義集計を実装する手順は次のとおりです。

1. ODCIAggregate インタフェース・ルーチンをオブジェクト型のメソッドとして実装します。
2. CREATE FUNCTION 文で、UDAG を作成し、手順 1 で作成した実装タイプを指定します。

   ```
   CREATE FUNCTION MyUDAG ... AGGREGATE USING MyUDAGRoutines;
   ```

3. SQL DML 文で組込み集計と同じ方法で、UDAG を使用します。

   ```
   SELECT col1, MyUDAG(col2) FROM tab GROUP BY col1;
   ```

集計関数の動作

集計関数は概念的には、入力として一連の値を受け取り、単一の値を返します。集計操作に必要な一連の値は、通常は、GROUP BY 句を使用して識別します。次に例を示します。

```
SELECT AVG(T.Sales)
FROM AnnualSales T
GROUP BY T.State
```

集計関数の評価は、3 つの基本操作に分割できます。前述の例 AVG() について考えると、次のようになります。

1. 初期化 : 計算を初期化します。

   ```
   runningSum = 0; runningCount = 0;
   ```

2. 反復 : 新しい入力値を処理します。

   ```
   runningSum += inputval; runningCount++;
   ```

3. 終了 : 結果を計算します。

   ```
   return (runningSum/runningCount);
   ```

前述の例にある変数 runningSum と runningCount は、集計操作の**状態**を判断します。このように、**集計操作コンテキスト**は runningSum と runningCount 属性を含むオブジェクトとして表示できます。初期化メソッドは集計操作コンテキストを初期化します。反復メソッドではそれを更新し、終了メソッドはそのコンテキストを使用して結果の集計値を戻します。

2 つの集計操作コンテキストをマージして、新しいコンテキストを作成するためには、さらにもう 1 つの基本操作が必要です。この操作は、サブセットに対する集計操作の結果を結合して、セット全体の集計結果を得る場合に必要です。このような状況は、集計をシリアルおよびパラレルで評価しているときに生じます。

4. マージ : 2 つの集計操作コンテキストを結合し、単一のコンテキストを戻します。

```
runningSum = runningSum1 + runningSum2;
runningCount = runningCount1 + runningCount2;
```

Oracle では、前述した基本操作を独自に実装しているため、ユーザーは新しい集計関数を登録できます。

アプリケーション・インタフェース

Oracle には、アプリケーション・プログラムでオブジェクト・データ型を使用するための機能が多数あります。

- SQL
- PL/SQL
- Pro*C/C++
- OCI
- OTT
- JPublisher
- JDBC
- SQLJ

SQL

Oracle SQL データ定義言語（DDL）は、オブジェクト・データ型のために次のようなサポートを提供しています。

- オブジェクト型、ネストした表および配列の定義
- 権限の指定
- オブジェクト型の表の列の指定
- オブジェクト表の作成

Oracle SQL DML は、オブジェクト・データ型のために次のようなサポートを提供しています。

- オブジェクトとコレクションの問合せおよび更新
- REF の操作

関連項目： SQL 構文の詳細説明は、『Oracle9*i* SQL リファレンス』を参照してください。

PL/SQL

PL/SQL は、SQL を拡張する手続き型言語です。パッケージおよびデータのカプセル化、情報の非表示、オーバーロード、例外処理などの機能が提供されています。ほとんどのストアド・プロシージャは、PL/SQL で記述されています。

PL/SQL では、ファンクションおよびプロシージャ内から、オブジェクト型をサポートする SQL 機能を使用できます。PL/SQL ファンクションとプロシージャのパラメータと変数は、ユーザー定義型にすることができます。

PL/SQL は、オブジェクト型に結び付けるメソッドを実装するのに必要なすべての機能を備えています。これらのメソッド（ファンクションとプロシージャ）は、ユーザーのスキーマの一部としてサーバーに格納されます。

> **関連項目：** PL/SQL の詳細説明は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

Pro*C/C++

Oracle Pro*C/C++ プリコンパイラを使用すると、プログラマは C および C++ プログラムでオブジェクト・データ型を使用できます。Pro*C を使用する開発者は、Object Type Translator（OTT）を使用して、Oracle のオブジェクト型とコレクションを C のデータ型にマップし、Pro*C アプリケーションで使用できます。

Pro*C は、オブジェクト型とコレクションのコンパイル時型チェック機能と、データベース型から C データ型への自動型変換を提供しています。Pro*C は、オブジェクトを作成したり破棄したりするための SQL 構文 `EXEC` 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 プログラムに埋め込まれている SQL 文と PL/SQL ファンクションまたはプロシージャ。
- オブジェクト・キャッシュへの単純なインタフェース。横断ポインタでオブジェクトにアクセスし、その後サーバー上で変更および更新します。

> **関連項目：**
>
> - 13-19 ページ「OCI」
> - Pro*C プリコンパイラの詳細説明は、『Pro*C/C++ Precompiler 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 DBMS 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、`LNOCIAnyData` と `LNOCIAnyDataSet` の作成とアクセスが可能になります。

- LNOCIAnyData 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。
- LNOCIAnyDataSet 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 LNOCIType に対応します。
- SYS.ANYDATA は LNOCIAnyData に対応します。
- SYS.ANYDATASET は LNOCIAnyDataSet 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。

新しい 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（LNOCIAnyData）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、例外になります。
- **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：

- 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』
- 『Oracle Call Interface プログラマーズ・ガイド』

OCI

Oracle Call Interface（OCI）は、Oracle サーバーへの C 言語インタフェースの集合です。OCI を使用すると、プログラマはサーバーの機能の使用に関する柔軟性を得られます。

OCI の重要な要素は、オブジェクト・キャッシュと呼ばれる作業領域をアプリケーション・プログラムから使用できるようにするコールの集合です。**オブジェクト・キャッシュ**は、クライアント側にあるメモリー・ブロックで、プログラムがオブジェクト全体を格納し、サーバーとの間を往復せずにオブジェクト間をナビゲートできるようにします。

オブジェクト・キャッシュは、そのオブジェクト・キャッシュを使用するアプリケーション・プログラムが完全に制御および管理します。Oracle サーバーは、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。

LNOCI は、次の機能を持つ関数を提供しています。

- SQL を使用してサーバー上のオブジェクトにアクセスします。
- 横断ポインタまたは `REF` によってオブジェクト・キャッシュ内のオブジェクトをアクセスし、操作し、管理します。
- Oracle の日付および文字列、数値を C データ型に変換します。
- オブジェクト・キャッシュのメモリーのサイズを管理します。
- 一時型記述を作成します。一時型記述は、DBMS に永続的に格納されません。Oracle9*i* 以上に互換性を設定する必要があります。

LNOCI の並行性が向上し、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。

LNOCI を使用する開発者は、Object Type Translator（OTT）を使用して、Oracle オブジェクト型に対応する C データ型を生成できます。

関連項目：『Oracle Call Interface プログラマーズ・ガイド』

OTT

Object Type Translator（OTT）は、オブジェクト型に対応する C 言語の構造体の宣言を自動生成するプログラムです。OTT は、Pro*C プリコンパイラと OCI サーバー・アクセス・パッケージを使用するときの手助けをします。

関連項目：

- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Pro*C/C++ Precompiler プログラマーズ・ガイド』

JPublisher

Java Publisher（JPublisher）は、データベース内のオブジェクト型に対応する Java クラス定義を自動的に生成するプログラムです。JPublisher は、SQLJ および JDBC サーバー・アクセス・パッケージの使用を容易にします。

関連項目：『Oracle9*i* JPublisher ユーザーズ・ガイド』

JDBC

Java Database Connectivity（JDBC）は、Oracle サーバーへの Java インタフェースの集合です。Oracle の JDBC の機能は、次のとおりです。

- 動的 SQL を通じて、Java プログラムからデータベース内で定義されたオブジェクト型とコレクション型にアクセスできるようにします。
- データベース内で定義されている型を、デフォルトまたはカスタマイズ可能なマッピングを通じて Java のクラスに変換します。

関連項目：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』

SQLJ

SQLJ により、開発者は Java プログラムでオブジェクト・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アプリケーションで使用される Java クラスにマップできます。

SQLJ は、Java コードに埋め込まれた SQL 文を使用して、サーバー・オブジェクトへのアクセスを提供します。また、コンパイル時には、SQL 文中のオブジェクト型とコレクションの型チェックを提供します。

構文は、ANSI 規格（SQLJ コンソーシアム）に基づいています。

SQLJ オブジェクト型

Oracle では、SQL ユーザー定義オブジェクト型として Java クラスを指定できます。この SQLJ 型の列と行を定義できます。また、この型のオブジェクトは SQL 基本型のように、問合せや操作が可能です。

さらに、次の操作を実行できます。

- あるクラスの静的フィールド SQL で参照できるようにします。
- ユーザーに 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に依存性を維持します。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』
- 『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』

データ型の発展

オブジェクト・データ型は、次のスキーマ・オブジェクトによって参照できます。

- 表またはサブテーブル
- 型またはサブタイプ
- プログラム・ユニット（PL/SQL ブロック）: プロシージャ、ファンクション、パッケージ、トリガー
- 索引タイプ
- ビュー（オブジェクト・ビューを含む）
- 機能索引
- 演算子

これらのオブジェクトが、別の型やサブタイプから直接または間接的にある型を参照すると、その型に依存したオブジェクトになります。型が変更されると、必ずすべての依存プログラム・ユニット、ビュー、演算子および索引タイプに無効のマークが付けられます。これらの無効なオブジェクトが次回参照されると、新しい型定義を使用して再度有効にされます。正常に再コンパイルされると有効になるため、もう一度使用できるようになります。

ある型に型依存または表依存がある場合、既存の永続データは現行の型定義に依存しているため、型定義の変更はさらに複雑になります。

オブジェクト型を変更し、それが依存する型と表に型変更を伝播させることができます。`ALTER TYPE` によって、既存の型からメソッドや属性を追加および削除できます。またオプションで、依存する型、表や表データへもこれらの変更を伝播できます。ある型の属性も変更できます。

関連項目：

- 構文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 型指定と本体のコンパイルの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
- 型バージョンの管理の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

オブジェクト・ビューの概要

ビューが仮想表であるのと同様に、**オブジェクト・ビュー**は仮想的なオブジェクト表です。

Oracle は、基本的なリレーショナル・ビュー・メカニズムへの機能拡張としてオブジェクト・ビューを提供します。オブジェクト・ビューを使用すると、データベースのリレーショナル表またはオブジェクト表の列に格納されているデータ（組込み型またはユーザー定義型）から、仮想的なオブジェクト表を作成できます。

オブジェクト・ビューは、データベース内のデータおよびオブジェクトへの特化または制限付きのアクセスを提供する機能を実現します。たとえば、オブジェクト・ビューを使用して、機密データを含む属性がなく、削除方法がないバージョンの従業員オブジェクト表を提供できます。

オブジェクト・ビューにより、オブジェクト指向アプリケーションでリレーショナル・データを使用できるようになります。ユーザーは、オブジェクト・ビューを使用して次の操作ができます。

- 既存の表を変換せずにオブジェクト指向プログラミング技法を試してみます。
- リレーショナル表からオブジェクト・リレーショナル表へ、データを段階的かつ透過的に変換します。
- 既存のオブジェクト指向アプリケーションで従来の RDBMS データを使用します。

オブジェクト・ビューの利点

オブジェクト・ビューを使用すると、パフォーマンスを改善できます。オブジェクト・ビューの 1 行を構成するリレーショナル・データは、1 つの単位としてネットワークを横断するため、何度も往復する必要がありません。

リレーショナル・データをクライアント側のオブジェクト・キャッシュにフェッチして、C または C++ の構造体にマップできるため、3GL アプリケーションはそのデータをネイティブな構造体と同じように処理できます。

オブジェクト・ビューは、従来のデータを段階的にアップグレードしていくための手段になります。オブジェクト・ビューにより、リレーショナル・アプリケーションとオブジェクト指向アプリケーションの共存が可能になり、既存のリレーショナル・データへのオブジェクト指向アプリケーションの導入も、パラダイムの大幅な変更なしに容易に行えます。

オブジェクト・ビューには、同じリレーショナル・データまたはオブジェクト・データを複数の方法で表示できるという柔軟性もあります。したがって、データベースにデータを格納する方法を変えなくても、様々なアプリケーションに応じて異なるメモリー内オブジェクト表現を使用できます。

オブジェクト・ビューの定義方法

概念上は、オブジェクト・ビューを定義するプロセスは簡単です。このプロセスでは次の処理を行います。

- オブジェクト・ビューの行によって表現するオブジェクト型を定義します。
- どのリレーショナル表のどのデータに、その型のオブジェクトの属性が入っているかを指定する問合せを作成します。
- オブジェクト・ビューのオブジェクト（行）を REF で参照できるように、基礎となるデータの属性に基づいてオブジェクト識別子を指定します。

オブジェクト識別子は、Oracle がオブジェクト表の行として自動的に生成する一意のオブジェクト識別子と対応しています。ただし、オブジェクト・ビューの場合は、基礎を形成するデータの中でなんらかの一意の値（主キーなど）を宣言する必要があります。

表または別のオブジェクト・ビューに基づくオブジェクト・ビューにオブジェクト識別子が指定されない場合、Oracle は、元の表またはオブジェクト・ビューのオブジェクト識別子を使用します。

複合オブジェクト・ビューを更新できるようにするには、さらに次の処理を実行する必要があります。

- アプリケーション・プログラムがオブジェクト・ビューのデータを更新しようとするたびに Oracle で実行されるように、INSTEAD OF トリガー・プロシージャを作成します。

この 4 つの処理を実行すると、オブジェクト・ビューをオブジェクト表と同じように使用できます。

たとえば、次の SQL 文は、オブジェクト・ビューを定義します。

```
CREATE TABLE emp_table  (
    empnum   NUMBER (5),
    ename    VARCHAR2 (20),
    salary   NUMBER (9, 2),
    job      VARCHAR2 (20) );

CREATE TYPE employee_t  AS OBJECT(
    empno    NUMBER (5),
    ename    VARCHAR2 (20),
    salary   NUMBER (9, 2),
    job      VARCHAR2 (20) );

CREATE VIEW emp_view1 OF employee_t
    WITH OBJECT OID (empno) AS
        SELECT   e.empnum, e.ename, e.salary, e.job
            FROM     emp_table e
            WHERE    job = 'Developer';
```

このオブジェクト・ビューは、ユーザーには、基礎を形成する型が employee_t であるオブジェクト表のように見えます。それぞれの行には employee_t 型のオブジェクトが含まれており、それぞれの行のオブジェクト識別子は一意です。

Oracle は、指定されたキーに基づいてオブジェクト識別子を構築します。ほとんどの場合、実表の主キーを指定します。ただし、オブジェクト・ビューを定義する問合せに結合が含まれている場合は、オブジェクト・ビューの行を一意に識別できるように、その結合に含まれるすべての表からのキーを指定する必要があります。

注意： WITH OBJECT OID 句の列（この例では empno）は、基礎を形成するオブジェクト型（この例では employee_t）の属性にもなっている必要があります。これにより、トリガー・プログラムが実表の対応する行を一意に識別しやすくなります。

関連項目：

- オブジェクト・ビューの定義の具体的な手順は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- INSTEAD OF トリガーの記述方法の詳細は、13-25 ページの「オブジェクト・ビューの更新」を参照してください。

オブジェクト・ビューの利用

オブジェクト・ビューの行データが 2 つ以上の表から取られている場合もありますが、そのオブジェクトは 1 回の操作でネットワークを横断します。インスタンスがクライアント側のオブジェクト・キャッシュに入っているときは、プログラマにとってそのインスタンスは C または C++ の構造体、あるいは PL/SQL のオブジェクト変数のように見えます。このインスタンスは、他のネイティブな構造体と同じように処理できます。

SQL 文の中で、オブジェクト・ビューは、オブジェクト表を参照するときと同じ方法で参照できます。たとえば、オブジェクト・ビューは、SELECT 構文のリスト、UPDATE SET 句または WHERE 句に指定できます。オブジェクト・ビューに対するオブジェクト・ビューを定義することもできます。

オブジェクト表のオブジェクトに使用するときと同じ OCI コールを使用して、クライアント側でオブジェクト・ビューのデータにアクセスできます。たとえば、REF を確保するときに LNOCIObjectPin() を使用し、オブジェクトをサーバーにフラッシュするときに LNOCIObjectFlush() を使用できます。オブジェクト・ビュー内のオブジェクトをサーバーに更新またはフラッシュすると、Oracle はオブジェクト・ビューを更新します。

関連項目： OCI コールの詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

オブジェクト・ビューの更新

オブジェクト・ビューのデータは、オブジェクト表に使用するのと同じ SQL DML を使用して更新、挿入および削除できます。あいまいでなければ、Oracle はオブジェクト・ビューの実表を更新します。

ビューの問合せに、結合、集合演算子、集計関数、GROUP BY または DISTINCT が含まれる場合、そのビューは更新不可能です。ビューの問合せに疑似列、つまり式が含まれている場合、対応するビューの列は更新不可能です。オブジェクト・ビューには、しばしば結合が含まれています。

このような問題を克服するために、Oracle は **INSTEAD OF トリガー**を提供しています。Oracle は実際の DML 文のかわりにこのトリガー本体を実行するので、これらのトリガーのことを INSTEAD OF トリガーと呼びます。

INSTEAD OF トリガーにより、透過的な方法でオブジェクト・ビューまたはリレーショナル・ビューを更新できます。オブジェクト表の場合と同じ SQL DML（INSERT、DELETE および UPDATE）文を記述します。Oracle は、SQL 文のかわりに該当するトリガーを起動し、そのトリガー本体に指定されているアクションを実行します。

関連項目：

- INSTEAD OF トリガーを使用する発注 / 明細項目の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。
- 第 17 章「トリガー」

ビュー内のネストした表の列の更新

ネストした表を変更するには、新しい要素を挿入して更新するか、既存の要素を削除します。ビュー内と同様に、ネストした表の列が仮想または合成の場合、通常その列は更新できません。この種の列を更新するために、Oracle ではこれらの列に INSTEAD OF トリガーを作成できます。

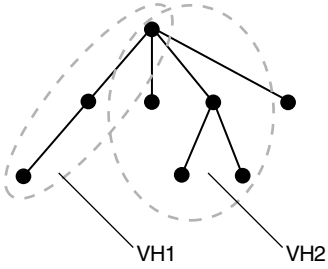
ビューのネストした表の列で定義されている INSTEAD OF トリガーは、その列が変更されると起動します。親行の更新によってコレクション全体が置き換えられると、ネストした表の列の INSTEAD OF トリガーは起動しません。

関連項目： ネストした表の列で INSTEAD OF トリガーを使用する発注 / 明細項目の例は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

ビューの階層

オブジェクト・ビューは、別のオブジェクト・ビューのサブビューとして作成できます。スーパービューのタイプは、作成するオブジェクト・ビューのタイプの即時スーパータイプにする必要があります。つまり、タイプの階層と 1 対 1 の対応関係を持つオブジェクト・ビュー階層を作成できます。しかし、すべてのビュー階層が対応するタイプ階層すべてにまたがる必要はありません。ビュー階層はタイプ階層のどのサブタイプからでも構成できます。さらに、サブ階層全体を網羅する必要はありません。

図 13-2 複数のビューの階層



デフォルトでは、ビューの階層におけるオブジェクト・ビューの行には、そのビューの列に表示されるすべてのサブビュー（直接および間接）の全行が含まれます。

特定のサブタイプに対応する特定のビューのサブビューとして作成できるのは、オブジェクト・ビュー 1 つのみです。つまり、同じビューを複数のビュー階層で使用することはできません。オブジェクト・ビューは、1 つのスーパービューのサブビューとしてのみ作成できます。多重継承はサポートされていません。

サブビューはスーパービューからオブジェクト識別子（OID）を継承します。サブビュー内で明示的に指定することはできません。

第 V 部

データ・アクセス

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

第 V 部には、次の章が含まれています。

- 第 14 章「SQL、PL/SQL および Java」
- 第 15 章「スキーマ・オブジェクト間の依存性」
- 第 16 章「トランザクションの管理」
- 第 17 章「トリガー」

14

SQL、PL/SQL および Java

この章では、SQL（Structured Query Language）、Oracle が開発した SQL への手続き型言語機能拡張である PL/SQL、Java 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。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

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。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、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します。

SQL は、IBM 研究所で開発および定義され、ANSI/ISO によりリレーショナル・データベース管理システムの標準言語として改良されました。SQL-99 規格準拠の最小レベルはコアと呼ばれます。SQL-99 コアは、SQL-92 エントリ・レベル仕様のスーパーセットです。Oracle9*i* は、SQL-99 コア仕様に準拠しています。

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、文を追加することができます。SQL*Plus および Oracle Enterprise Manager などの Oracle のツール製品では、Oracle データベースに対して ANSI/ISO 標準の SQL 文を実行できるほか、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。

アプリケーション・プログラマは、Oracle SQLJ を使用して、Java の設計思想に準拠した静的な SQL 操作を Java コードに埋め込むことができます。SQLJ プログラムとは、ANSI 標準の SQLJ 言語リファレンス構文に準拠した、静的埋込み SQL 文を含む Java プログラムです。

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 に組み込まれているセキュリティが活用されず、データのセキュリティと整合性が損なわれます。

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、『Oracle9*i* SQL リファレンス』を参照してください。
- 『Oracle Enterprise Manager 管理者ガイド』
- 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 操作を Java コードに埋め込む方法の詳細は、『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』を参照してください。

SQL 文

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

SQL 文は、コンピュータ・プログラムまたは命令です。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;
```

実行できるのは完全な SQL 文のみです。次のような不完全な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 SQL 文は、次のカテゴリに分類できます。

- 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 17 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を取り出します（SELECT）。フェッチでのスクロールが可能です（14-7 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');
```

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）。
- 権限とロールの付与および取消しを行います（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、先行するトランザクションを暗黙的にコミットし、新しいトランザクションを開始します。次に、DDL 文の例をいくつか示します。

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

DROP TABLE plants;

GRANT SELECT ON employees TO scott;

REVOKE DELETE ON employees FROM scott;
```

関連項目：

- 第 22 章「データベース・アクセスの制御」
- 第 23 章「権限、ロールおよびセキュリティ・ポリシー」
- 第 24 章「監査」

トランザクション制御文

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

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

セッション制御文

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

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

システム制御文

システム制御文は、Oracle サーバー・インスタンスのプロパティを変更します。システム制御文は、ALTER SYSTEM のみです。この文は、設定値（共有サーバーの最小数など）の変更、セッションの停止およびその他の作業のために使用します。

埋込み SQL 文

埋込み SQL 文は、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、解放（DECLARE CURSOR、OPEN、CLOSE）
- データベースの指定と、Oracle への接続（DECLARE DATABASE、CONNECT）
- 変数名の割当て（DECLARE STATEMENT）
- 記述子の初期化（DESCRIBE）
- エラー条件と警告の処理方法の指定（WHENEVER）
- SQL 文の解析と実行（PREPARE、EXECUTE、EXECUTE IMMEDIATE）
- データベースからデータの取出し（FETCH）

非標準 SQL の識別

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

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

関連項目：

- 『Pro*C/C++ Precompiler プログラマーズ・ガイド』
- 『Pro*COBOL Precompiler プログラマーズ・ガイド』
- 『SQL*Module for Ada Programmer's Guide』

再帰的 SQL

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

カーソル

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

ほとんどの Oracle ユーザーは Oracle ユーティリティの自動カーソル処理を使用しますが、プログラム・インタフェースを利用すると、アプリケーションの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 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。そ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

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

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

> **関連項目：** 共有 SQL の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』および『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

解析

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

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

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

アプリケーションが SQL 文の解析コールを出すことと、Oracle が実際にその文を解析することには、次のような違いがあります。**アプリケーション**が**解析コール**を出すと、SQL 文はプライベート SQL 領域に対応付けられます。この対応付けにより、アプリケーションが解析コールを発行しなくても、その文を繰り返し実行できます。Oracle が**解析操作**を実行した場合は、SQL 文に共有 SQL 領域が割り当てられます。文に共有 SQL 領域が割り当てられた後は、再解析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： 14-16 ページ「PL/SQL の概要」

SQL の処理

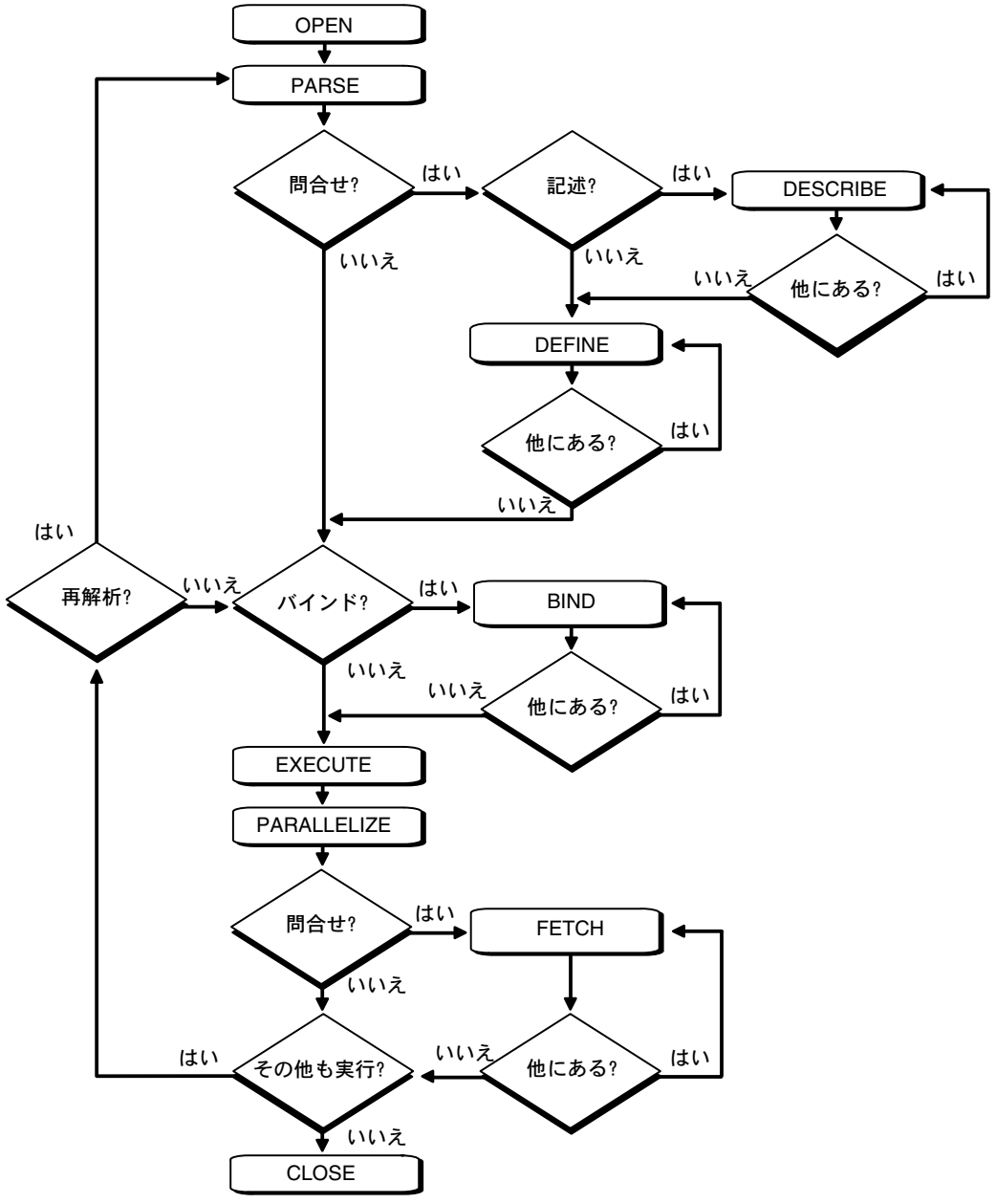
この項では、SQL 処理の基本について説明します。この項の内容は、次のとおりです。

SQL 文の実行

図 14-1 に、SQL 文を処理して実行するための一般的な段階を示します。場合によっては、順序が少し異なることもあります。たとえば、`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 14-1 SQL 文の処理の段階



DML 文の処理

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

Pro*C プログラムを使用して、ある部門の従業員全員の給与を増額する処理を実行するとします。現在使用しているプログラムは、Oracle 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。

各タイプの文の処理では、次の段階が必要です。

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

オプションとして、次の段階を含めることもできます。

- 段階 6: 文のパラレル化

図 14-1 に示されているとおり、問合せ（SELECT）では、さらにいくつかの段階が必要です。

- 段階 3: 問合せの結果の記述
- 段階 4: 問合せの出力の定義
- 段階 8: 問合せの行のフェッチ

関連項目： 14-11 ページ「問合せの処理」

段階 1: カーソルの作成 プログラム・インタフェース・コールによってカーソルが作成されます。カーソルは、SQL 文とは別に、独立して作成されます。カーソルは SQL 文からの要求で作成されます。ほとんどのアプリケーションでは、カーソルは自動的に作成されます。ただし、プリコンパイラ・プログラムでは、暗黙的にカーソルが作成されたり、カーソル作成が明示的に宣言されることもあります。

段階 2: 文の解析 解析段階では、SQL 文がユーザー・プロセスから Oracle に渡され、解析済みの表現が共有 SQL 領域にロードされます。文処理のこの段階では、多くのエラーを捕捉できます。

解析には、次のような処理が関係しています。

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

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

解析段階には、文の実行回数に関係なく一度だけ実行する必要のある処理要件があります。Oracle はそれぞれの SQL 文を一度だけ変換し、後でその文が参照されると、解析済み文を再実行します。

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

関連項目： 14-7 ページ「共有 SQL」

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。

```
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。

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

段階 3: 問合せの結果の記述 記述段階が必要なのは、問合せがユーザーにより対話式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 4: 問合せの出力の定義 問合せの定義段階では、フェッチされた各値を受け取るために定義された変数の位置、サイズおよびデータ型を指定します。Oracle は、必要に応じてデータ型変換を実行します。

段階 5: 変数のバインド この時点で、Oracle は SQL 文の意味を認識していますが、文を実行するための情報がまだ不足しています。Oracle は、文に含まれている変数の値を必要とします。例では、`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 ユーティリティはアプリケーションのエンド・ユーザーに新しい値の入力を要求するプロンプトを表示するのみであるため、エンド・ユーザーはバインド変数を指定しているということを認識していない可能性があります。

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

Oracle でデータ型変換を実行する必要がある場合は、暗黙的にまたはデフォルトで指定されていないかぎり、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：

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

- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Pro*C/C++ Precompiler プログラマーズ・ガイド』（「動的 SQL 方法 4」を参照）
- 『Pro*COBOL Precompiler プログラマーズ・ガイド』（「動的 SQL 方法 4」を参照）

段階 6: 文のパラレル化 Oracle では、問合せ（SELECT、INSERT、UPDATE、MERGE、DELETE）および索引作成などの DDL 操作をパラレル化して、副問合せを持つ表およびパーティションに対する操作を作成できます。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 18 章「SQL 文のパラレル実行」

段階 7: 文の実行 この時点で、Oracle には必要なすべての情報とリソースが揃ったため、文を実行できます。文が問合せや INSERT 文の場合は、変更されるデータがないため、行をロックする必要はありません。ただし、UPDATE 文または DELETE 文の場合、その文の影響を受けるすべての行は、トランザクションの COMMIT、ROLLBACK または SAVEPOINT が次に実行される時点まで、データベースの他のユーザーが使用できないようにロックされます。この処理により、データ整合性を確実に維持できます。

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

段階 8: 問合せの行のフェッチ フェッチ段階では、行が選択され、順序付け（問合せで要求された場合）されます。最後の行がフェッチされるまで、毎回のフェッチで結果の行が連続して取り出されます。

段階 9: カーソルのクローズ SQL 文の処理の最後の段階は、カーソルのクローズです。

DDL 文の処理

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

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

トランザクションの制御

一般に、1 つのトランザクションとしてまとめるアクションのタイプに配慮するのは、Oracle へのプログラム・インタフェースを使用するアプリケーション設計者のみです。作業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。

アプリケーションを設計するときは、トランザクションに含めるアクションのタイプのみでなく、短い非分散型のトランザクションのパフォーマンスを向上させるために、BEGIN_DISCRETE_TRANSACTION プロシージャを使用する時期も判断する必要があります。

関連項目： 16-12 ページ「ディスクリート・トランザクションの管理」

オプティマイザの概要

オプティマイザは、SQL 文の最も効率的な実行方法を決定します。この処理は、SELECT、INSERT、UPDATE、MERGE または DELETE など、データ操作言語（DML）文の処理において重要です。SQL 文を実行する場合、表や索引にアクセスする順序などによって、実行方法が様々になることがあります。文を実行するときに使用する手順は、文の実行速度に大きく影響します。オプティマイザは、代替アクセス・パスの多数の要因を考慮します。コストベースまたはルールベースのアプローチを使用します。通常は、コストベースのアプローチを使用します。ルールベースのアプローチは、既存のアプリケーションのために使用します。

注意： オプティマイザは、Oracle のバージョンが変わると、同じ決定をしない可能性があります。最近のバージョンでは、入手可能な情報の改善により、オプティマイザは様々な決定が行えるようになりました。

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。PL/SQL パッケージ DBMS_STATS を使用して、コストベース・オプティマイザ（CBO）の統計も収集できます。

特定のアプリケーションのデータについて、オプティマイザよりもさらに詳細な知識を持つアプリケーション・デザイナのほうが、SQL 文をより効率的に実行する方法を選択できることもあります。アプリケーション・デザイナは、SQL 文内にヒントを使用して文の実行方法を指定できます。

関連項目：

- DBMS_STATS の使用方法は、『Oracle9*i* 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。
- コストベース・オプティマイザ、ルールベース・オプティマイザおよび拡張可能オプティマイザの詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

実行計画

DML 文を実行するために、Oracle で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 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。

ストアド・アウトライン ストアド・アウトラインとは、アウトライン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。

関連項目： 実行計画とストアド・アウトラインの使用方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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 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 によってデータベースに格納され、アプリケーションから名前でコールできる PL/SQL ブロックのことです。ストアド・プロシージャを作成すると、Oracle はそのプロシージャを解析し、その解析済みの表現をデータベースに格納します。さらに Oracle では、ファンクション（プロシージャによく似ているもの）やパッケージ（プロシージャとファンクションをまとめたもの）を作成して保管することもできます。

関連項目：

14-32 ページ「Java の概要」

第 17 章「トリガー」

PL/SQL の実行方法

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 サーバーにリンクされます。）

関連項目：『PL/SQL ユーザーズ・ガイドおよびリファレンス』

解析済みの実行

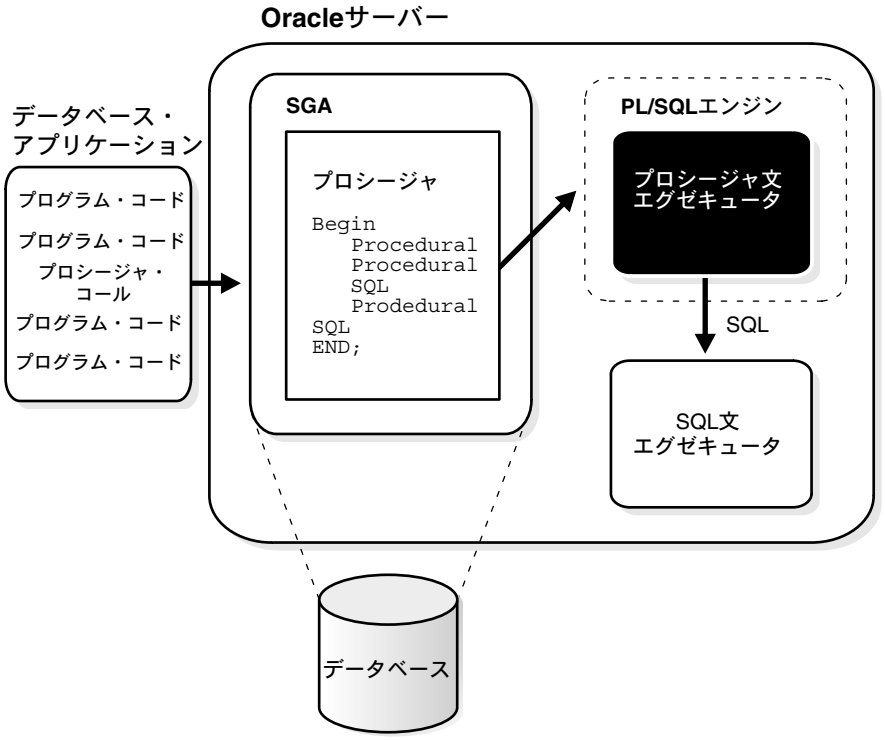
Oracle9*i* より前のバージョンでは、PL/SQL のソース・コードは、その名のとおり常にバイトコード表現にコンパイルされ、Oracle サーバーの一部として、また Oracle Forms などの製品に実装される移植性のある仮想マシンによって実行されていました。Oracle9*i* からは、システム固有の実行または解析済みの実行を選択できます。

PL/SQL エンジンは、PL/SQL プログラム・ユニットの定義、コンパイルおよび実行に使用するツールです。このエンジンは、Oracle サーバー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 データベースに格納でき、Oracle サーバーの PL/SQL エンジンを使用して処理できるプログラム・ユニットを対象にして説明します。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 14-2 に、Oracle サーバーに含まれている PL/SQL エンジンを示します。

図 14-2 PL/SQL エンジンと Oracle サーバー



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 は、コンパイル済みのプログラム・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 サーバー
- 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（バージョン 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 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

この項では、それぞれの構成メンバーについて概説します。

関連項目： 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

変数と定数

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

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

カーソル

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

関連項目： 14-7 ページ「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 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 の動的 SQL

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

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

- 動的 SQL 文を PL/SQL ブロックに埋め込みます。
- `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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- 動的 SQL の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
- 『Oracle9*i* 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 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++ Precompiler 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 Precompiler 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 14-3 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 14-3 ストアド・プロシージャ

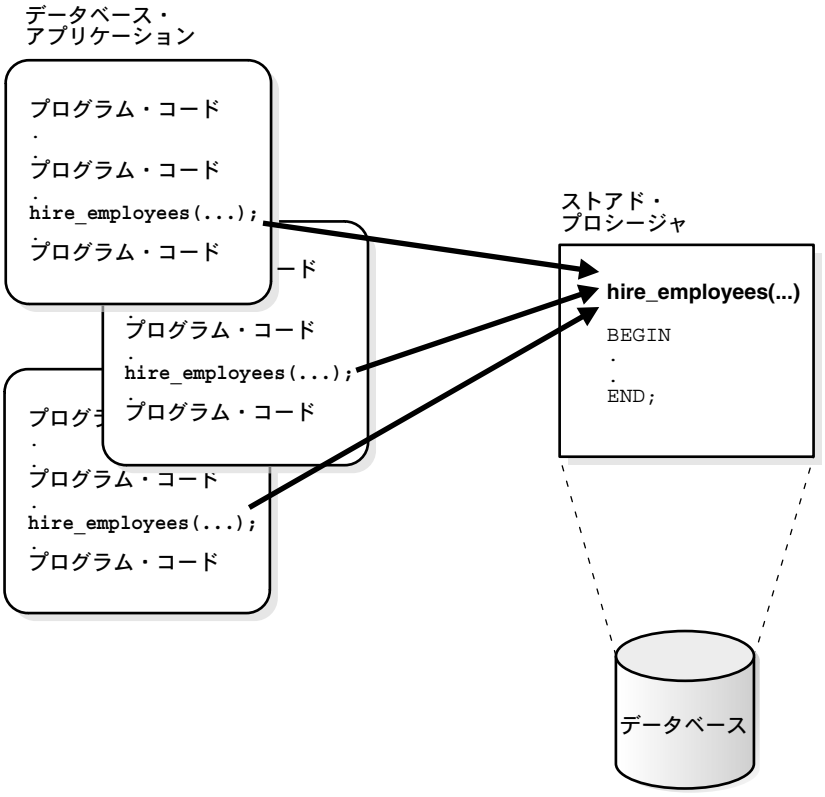


図 14-3 のストアド・プロシージャは、従業員レコードを employees 表に挿入します。このプロシージャを図 14-4 に示します。

図 14-4 ストアド・プロシージャの例

```
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;
```

図 14-3 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： 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

プロシージャの利点 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： 14-26 ページ「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 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み形式をそのまま使用できるため、実行時にコンパイルする必要がありません。
  - 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 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すみます。同じコードを何人ものユーザーが共有すれば、アプリケーションに必要な Oracle メモリーを実質的に削減できます。

- 生産性の向上

  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することができます。

たとえば、プロシージャやファンクションが正確な結果を戻すかどうかをテストし、検証後は、そのプロシージャとファンクションを必要な数のアプリケーションで再利用できます。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 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 サーバーに送信し、無名 PL/SQL ブロックを作成できます。Oracle によって PL/SQL ブロックがコンパイルされ、SGA の共有プールにそのコンパイル済みバージョンが入れられます。ただし、そのソース・コードやコンパイル済みバージョンは、現行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 が実行時に不必要な再コンパイルを実行することがなくなり、アプリケーションと Oracle の全体的なパフォーマンスが向上します。

スタンドアロン・プロシージャ パッケージのコンテキスト内で定義されていないストアド・プロシージャのことを、**スタンドアロン・プロシージャ**と呼びます。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、14-27 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 は、そのような依存性を自動的に追跡し、管理します。たとえば、プロシージャが参照している表の定義を変更した場合は、そのプロシージャを再コンパイルして、引き続き設計どおりに機能することを確認する必要があります。通常、Oracle はこのような依存性の管理を自動的に処理します。

> **関連項目：** 依存性の追跡の詳細は、第 15 章「スキーマ・オブジェクト間の依存性」を参照してください。

外部プロシージャ Oracle サーバー上で実行される 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 サーバー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、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

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 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 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 ETL（抽出－変換－ロード）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 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することができます。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：

- 『Oracle9*i* Data Cartridge Developer's Guide』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

テーブル・ファンクションのアカウントの詳細は、次のマニュアルを参照してください。

PL/SQL パッケージ

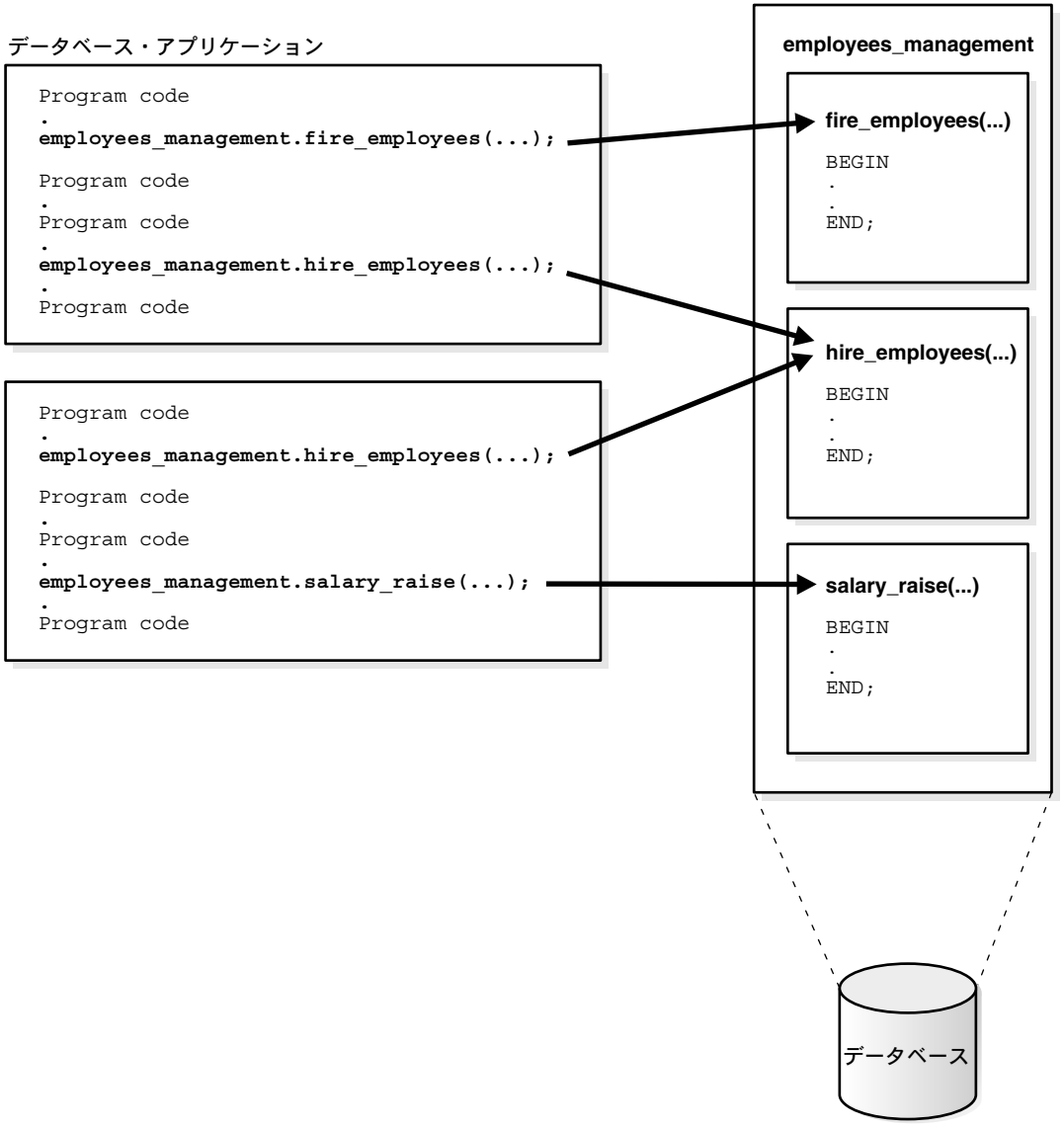
パッケージとは、関連するプロシージャとファンクションおよびこれらのプロシージャとファンクションが使用するカーソルと変数をグループとしてまとめたもので、1 単位として継続的に使用できるようにデータベースに格納されています。スタンドアロン・プロシージャやファンクションと同様に、パッケージ・プロシージャとファンクションは、アプリケーションやユーザーが明示的にコールできます。

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。

図 14-5 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 14-5 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：

- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle9*i* 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、プライベートはパッケージのユーザーから隠され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 22 章「データベース・アクセスの制御」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み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。

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：** コレクションとレコードの使用方法の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

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 Cartridge
- WebDB

> **関連項目：** PL/SQL Server Page の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

Java の概要

Java は、オブジェクト指向でありアプリケーション・レベルのプログラムに効率的であるため、オブジェクト指向のプログラミング言語の選択肢として普及してきました。Java の機能は、次のとおりです。

- Java Virtual Machine（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。
- 自動記憶域管理機能。この機能の特長はガベージ・コレクションで確認できます。
- C 言語に基づく言語構文。強い型指定が必要です。

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 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。経験豊富な Java プログラマは、通常はこれらの用語を熟知しています。

クラス

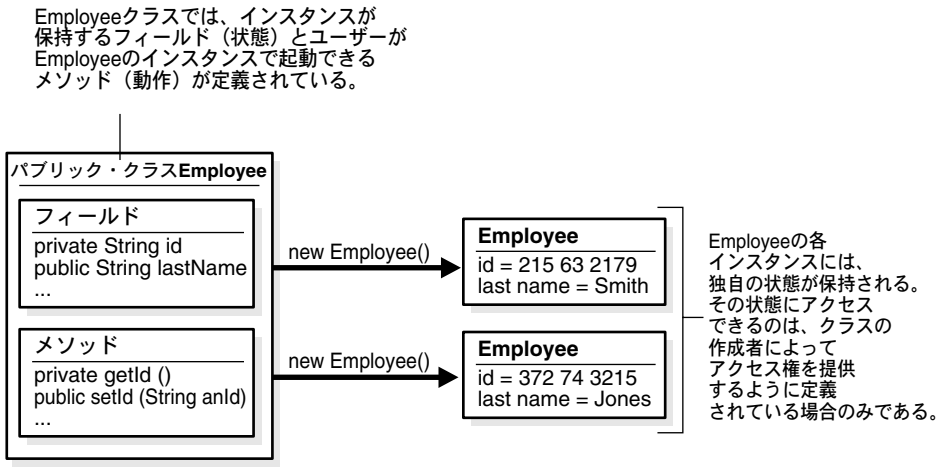
オブジェクト指向プログラミング言語はすべて、クラスの概念をサポートしています。表の定義では、クラスから共通の特性を共有するオブジェクト用のテンプレートが提供されます。それぞれのクラスには次の項目を含めることができます。

- 属性 — 特定クラスの各オブジェクトが所有する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 14-6 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 14-6 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 14-6 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。

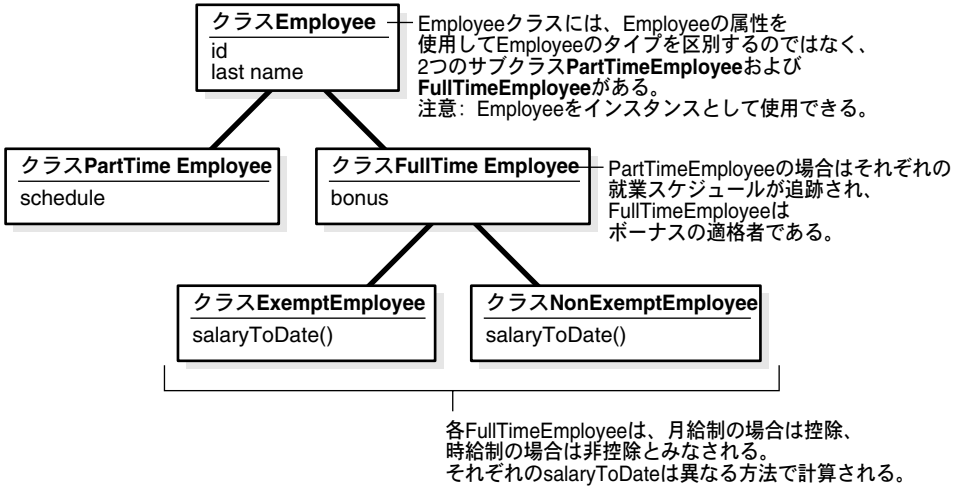
また、カプセル化では、Employee.getId() メソッドをプライベートとして宣言すること、あるいは Employee.getId() メソッドを記述しないでおくことができます。カプセル化を行うと、再利用可能なプログラム、または不正に使用されることのないプログラムを容易に記述できるようになります。カプセル化では、パブリックの宣言を行ったオブジェクトの機能のみがパブリックになります。他のすべてのフィールドとメソッドはプライベートです。プライベート・フィールドとプライベート・メソッドは、内部オブジェクトの処理に使用できます。

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 14-7 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 フィールドが含まれます。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 14-7 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 を追加するには、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();
}
```

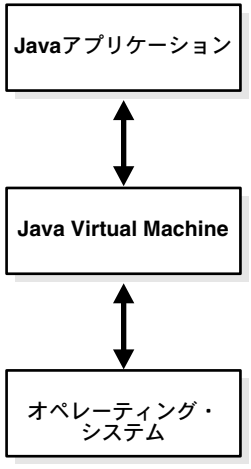
メソッド名 compensationToDate() を共通して使用することで、使用している従業員の種類を知らなくても、同じメソッドを異なるクラスで起動して異なる結果を受け取ることができます。FullTimeEmployees と PartTimeEmployees を処理するために特殊なメソッドを記述する必要はありません。このように、異なるオブジェクトに対して同じメッセージに異なる方法で対応する機能は、ポリモフィズムと呼ばれます。

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 Virtual Machine（JVM）

Java ソースは、他の高水準言語と同様、低水準のマシン語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 バイトの記憶単位だからです）。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、マシン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 Virtual Machine（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に最適化された個別のプログラムです。図 14-8 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 14-8 Java コンポーネント構造

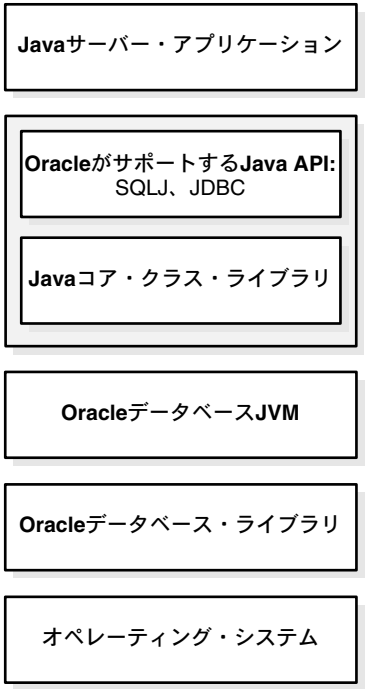


Java プログラムの開発では、Java 言語で記述された事前定義のコア・クラス・ライブラリを使用します。Java コア・クラス・ライブラリは、基本言語サポート（java.lang）、I/O（java.io）およびネットワーク・アクセス（java.net）など、一般的な機能を提供するパッケージに論理的に分割されています。JVM とコア・クラス・ライブラリにより、Java プログラマの開発するプログラムが、Java をサポートするすべてのハードウェアおよびオペレーティング・システムで、確実に実行できるようなプラットフォームが提供されます。この概念は、「1 回プログラミングすれば、どこでも実行可能」という Java の考え方を推進するものです。

図 14-9 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 データベース・ライブラリと対話します。

図 14-9 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マシン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 で Java を使用する理由

データベース内で Java アプリケーションを記述してロードできるのは、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するために開発されました。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 言語に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 言語の主要な拡張性機能の 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 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 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拡張性は向上しません。データベースの埋込み拡張性を使用すると、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拡張性が実現します。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 Oracle9*i* 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、拡張性が限定される可能性があります。この種の実装の場合、高水準の拡張性は十分には実証されていません。
- Oracle9*i* 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 Virtual Machine 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 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。

拡張性の観点から、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。

システム固有のコンパイラによるパフォーマンスの改善 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。

次の表に、システム固有の 2 つのコンパイル方法を示します。

システム固有のコンパイル方法	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）のマシン・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 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み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、サーバー環境では次の問題を伴います。

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 Virtual Machine 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が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 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、配置および管理のためのエンド・トゥ・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 データベース・サーバーと統合された Java Virtual Machine 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 および SQLJ 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。

Java ストアド・プロシージャ

Java を利用する PL/SQL プログラマは、Java ストアド・プロシージャに関心を持つことになります。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、PRO*、JDBC または SQLJ など、すべての Oracle Net クライアントからアクセスできます。

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、『Oracle9*i* Java Stored Procedures Developer's Guide』を参照してください。

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 には、Java プログラミング言語と JVM の全面準拠の実装が用意されています。

PL/SQL の統合と Oracle RDBMS 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。

Oracle には、Java 開発者が SQL データにアクセスできるように、JDBC および SQLJ という 2 種類の Application Program Interface（API）が用意されています。どちらの API も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配置できます。

- JDBC ドライバ ― クライアント / サーバーの 2 層アプリケーションを作成するために使用されます。
- SQLJ （Java での埋込み SQL）― 静的 SQL へのアクセスに使用されます。列名を知る必要があります。

JDBC ドライバ JDBC は、データベースに接続し、その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。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。

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。

ドライバ	説明
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 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、同じマシン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。Oracle8*i* 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、配置時にコードの再処理を必要としません。

SQLJ （ Java での埋込み SQL）オラクル社は IBM 社、Tandem 社、Sybase 社および Sun 社などの他のベンダーと共同で、Java プログラムに SQL 文を埋め込む標準的な方法である SQLJ を開発しました。この作業の結果が、JDBC より単純で生産性の高いプログラミング API に関する新たな規格（ANSI x.3.135.10-1998）です。ユーザーは、この上位レベルの API に対してアプリケーションを記述し、プリプロセッサを使用して、JDBC コールを使用する標準的な Java ソースにプログラムを変換します。このプログラムでは、実行時に標準的な JDBC ドライバを使用して複数ベンダー・データベースと通信できます。

SQLJ は、Java からデータベースにアクセスするクライアント側と中間層のアプリケーションを開発する、単純でしかも強力な手段を提供します。SQLJ は、Oracle 環境でストアド・プロシージャ、トリガー、メソッド内で使用できます。また、SQLJ プログラムを JDBC と併用できます。

SQLJ Translator は、Java ソース・コード内の埋込み SQL を JDBC ベースの Pure Java コードに変換する Java プログラムです。Oracle は完全な Java 環境を提供するため、サーバー上で実行するためにクライアント上で SQLJ プログラムをコンパイルするのみでなく、サーバー上で直接コンパイルできます。Oracle はインターネット標準に準拠しているため、ニーズに合った開発スタイルを選択できます。

関連項目：『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』

15

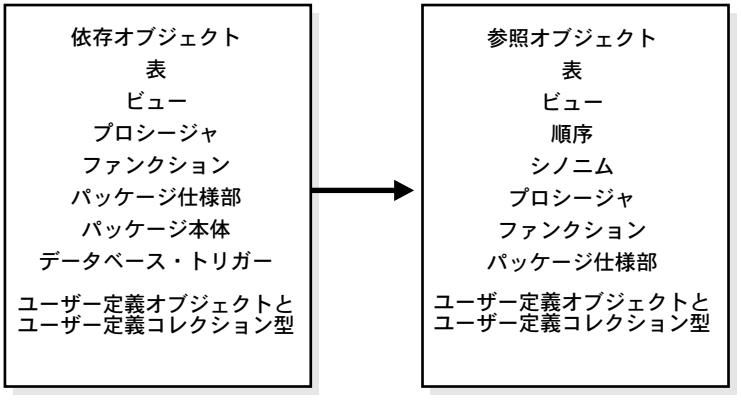
スキーマ・オブジェクト間の依存性

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

依存性の問題の概要

スキーマ・オブジェクトのタイプによっては、定義の一部として他のオブジェクトを参照できるものがあります。たとえば、ビューは表や他のビューを参照する問合せによって定義されます。また、プロシージャ本体には、データベースの他のオブジェクトを参照する SQL 文を組み込めます。定義の一部として他のオブジェクトを参照するオブジェクトを依存オブジェクトと呼び、参照されるオブジェクトを**参照**オブジェクトと呼びます。図 15-1 に、各種の依存オブジェクトと参照オブジェクトを示します。

図 15-1 様々な依存スキーマ・オブジェクトと参照スキーマ・オブジェクト



参照オブジェクトの定義を変更した場合、その変更の内容によって、依存オブジェクトがエラーなしで機能し続ける場合と、そうでない場合があります。たとえば、表を削除した場合、その表に基づくビューは使用できなくなります。

Oracle ではオブジェクト間の依存性が自動的に記録されるため、データベース管理者とユーザーは依存性管理の複雑な作業から解放されます。たとえば、複数のストアド・プロシージャが依存している表を変更すると、依存プロシージャは、次に参照された（再度実行またはコンパイルされた）時点で自動的に再コンパイルされます。

スキーマ・オブジェクト間の依存性を管理するために、データベース内のすべてのスキーマ・オブジェクトはステータスを持ちます。

- 有効なスキーマ・オブジェクトはコンパイルされており、参照するとただちに使用できます。

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

  無効になる可能性があるのは、依存しているオブジェクトのみです。表、順序およびシノニムは、常に有効です。

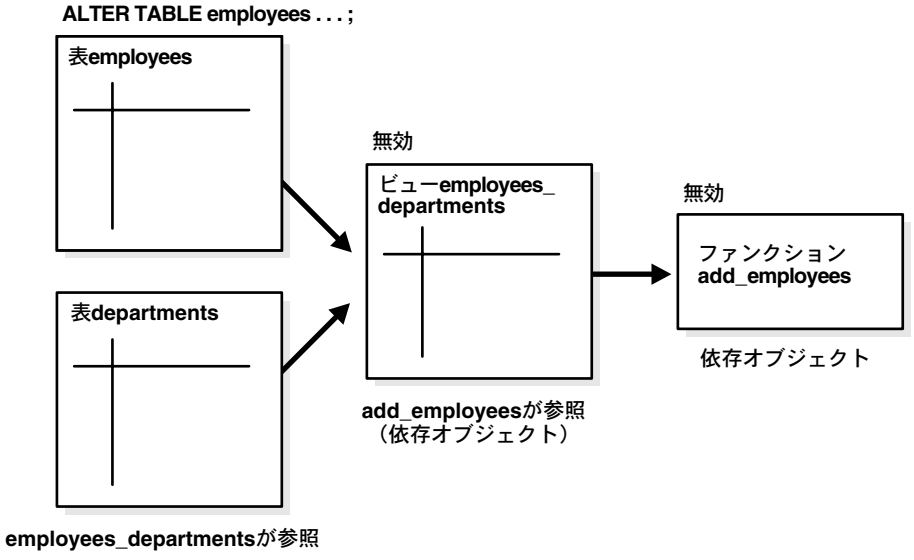
  ビュー、プロシージャ、ファンクションまたはパッケージが無効である場合、Oracle がそのオブジェクトをコンパイルしようとしても、そのオブジェクトに関連したエラーが発生します。たとえば、ビューのコンパイル時に、実表の 1 つが存在しなかったり、実表の正当な権限がなかったりする場合があります。パッケージのコンパイル時には、PL/SQL または SQL の構文エラーが発生したり、参照されているオブジェクトについての適切な権限が存在しない場合があります。このような問題があるスキーマ・オブジェクトは無効のままです。

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

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

たとえば、ビューを直接参照するストアド・プロシージャを考えます。ストアド・プロシージャは、このビューの実表を間接的に参照します。このため、実表を変更するとビューは無効になり、さらにストアド・プロシージャも無効になります。図 15-2 に、間接的な依存性を示します。

図 15-2 間接的な依存性



スキーマ・オブジェクトの依存性の解決

スキーマ・オブジェクトが SQL 文によって直接的に、または依存オブジェクトの参照を介して間接的に参照されるときに、Oracle は SQL 文に明示的に指定されているオブジェクトと参照オブジェクトのステータスを、必要に応じてチェックします。Oracle の処理は、SQL 文で直接および間接的に参照されるオブジェクトのステータスに応じて異なります。

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

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

ビューと PL/SQL プログラム・ユニットのコンパイル

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

- ビューまたはプログラム・ユニットの定義が正しいこと。すべての SQL 文と PL/SQL 文が適切な構成メンバーである必要があります。
- 参照オブジェクトが存在し、適切な構成であること。たとえば、ビューを定義する問合せに列が含まれる場合、その列が実表に存在している必要があります。
- ビューまたはプログラム・ユニットの所有者に、参照オブジェクトに対する必要な権限が付与されていること。たとえば、プロシージャ内の SQL 文によって表に行が挿入される場合、プロシージャの所有者にはその参照表に対する `INSERT` 権限が必要です。

ビューと実表

ビューは、そのビューの定義問合せで参照されている実表またはビューに依存しています。たとえば、SELECT * FROM *table* のように、ビューの定義問合せで参照される列が明示的に定義されていなければ、定義問合せはデータ・ディクショナリへの格納時に展開され、参照先の実表内の現行の列がすべて含まれるようになります。

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

> **注意：** 表、索引およびビューを作成した後に索引を削除すると、その表に依存するビュー、パッケージ、パッケージ本体、ファンクションおよびプロシージャなどのオブジェクトはすべて無効になります。

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

- ビューの定義問合せが参照しているすべての実表が存在している必要があります。ビューの実表を改名または削除すると、そのビューは無効になり、使用できなくなります。無効なビューを参照すると、その参照文は失敗します。ビューをコンパイルできるのは、実表を元の名前に改名した場合、または実表を再作成した場合のみです。
- 実表を変更したり、同じ列を含む実表を再作成した場合に、その実表の 1 つ以上の列のデータ型が変更されていても、依存ビューは正常に再コンパイルできます。
- ビューの実表が、少なくとも同じ列集合を含むように変更または再作成された場合、そのビューは有効にすることができます。実表が新しい列を含むように再作成され、その結果の表中にもはや存在しない列をビューが参照していると、そのビューは有効になりません。後者のケースは、SELECT * FROM *table* の問合せで定義されたビューに特に当てはまります。このような定義問合せはビューの作成時にデータ・ディクショナリ内で展開され、そのまま永続的に格納されるためです。

プログラム・ユニットと参照オブジェクト

参照オブジェクトの定義を変更すると、プログラム・ユニットは自動的に無効になります。たとえば、スタンドアロン・プロシージャに表、ビュー、別のスタンドアロン・プロシージャおよびパブリック・パッケージ・プロシージャを参照する複数の文が含まれているとします。このような場合は、次のような条件が成立します。

- 参照表を変更すると、依存プロシージャは無効になります。
- 参照ビューの実表を変更すると、ビューと依存プロシージャが無効になります。
- 参照スタンドアロン・プロシージャを置き換えると、依存プロシージャが無効になります。

- 参照パッケージの本体を置き換えても、依存プロシージャには影響しません。ただし、参照パッケージの**仕様部**を置き換えると、依存プロシージャは無効になります。これが、パッケージを使用することによってプロシージャおよび参照オブジェクトの間の依存性を最小にするメカニズムです。
- 表、索引およびビューを作成した後に索引を削除すると、その表に依存するビュー、パッケージ、パッケージ本体、ファンクションおよびプロシージャなどのオブジェクトはすべて無効になります。

データ・ウェアハウスの考慮事項

一部のデータ・ウェアハウスでは、ロード時間を速めるために夜間に索引を削除します。ただし、索引が削除された表に依存するすべてのビューは無効になります。このことは、その後実行されるパッケージがこれら削除されたビューを参照すると、そのパッケージが無効になることを意味します。

表、索引およびビューを作成した後に索引を削除すると、その表に依存するビュー、パッケージ、パッケージ本体、ファンクションおよびプロシージャなどのオブジェクトはすべて無効になることに注意してください。このことにより、更新可能な結合ビューが保護されます。

ビューを再び有効にするには、次の文を使用します。

```
SELECT * FROM vtest;
```

または

```
ALTER VIEW vtest compile
```

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されて再コンパイルされると、セッションのパッケージのすべてのインスタンシエーションが、状態も含めて失われる可能性があります。

セキュリティ認可

ユーザーや PUBLIC に対して DML オブジェクト権限やシステム権限の付与または取消しが実行された場合、Oracle はその所有者のすべての依存オブジェクトを自動的に無効にします。Oracle は依存オブジェクトを無効にして、その依存オブジェクトの所有者が参照オブジェクトに対する必要な権限を引き続き持っていることを検証します。Oracle は内部的に、そのようなオブジェクトを再コンパイルする必要はないことを確認します。つまり、オブジェクト構造を検証するのではなく、セキュリティ認可のみを検証します。このように最適化すると、不要な再コンパイルが省略され、依存オブジェクトのタイム・スタンプを変更する必要もありません。

関連項目： 無効なビューとプログラム・ユニットを強制的に再コンパイルする方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

ファンクション・ベース索引の依存性

ファンクション・ベース索引は、その索引を定義している式に使用されたファンクションに依存しています。PL/SQL ファンクションまたはパッケージ・ファンクションに変更があると、索引は使用禁止になります。

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

要件

ファンクション・ベース索引を作成する手順は次のとおりです。

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

ファンクション・ベース索引を使用する手順は次のとおりです。

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

この後の各項では、追加の要件について説明します。

関連項目： 10-31 ページ「ファンクション・ベース索引」

DETERMINISTIC 関数

ファンクション・ベース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： 『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』

定義する関数に対する権限

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

ファンクション・ベース索引の依存性の解決

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

オブジェクト名の解決

SQL 文で参照するオブジェクト名は、複数の要素をピリオドで区切って指定できます。Oracle では、次の方法でオブジェクト名が解決されます。

1. Oracle は、SQL 文で参照されている名前の最初の要素を修飾しようとします。たとえば、hr.employees では、hr が最初の要素です。要素が 1 つしかない場合は、その要素が最初の要素とみなされます。

   a. 現行のスキーマ内で、オブジェクト名の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。一致するオブジェクトが見つからない場合は、手順 b に進みます。

   b. 名前の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。一致するパブリック・シノニムが見つからない場合は、手順 c に進みます。

   c. オブジェクト名の最初の要素と一致する名前を持つスキーマが検索されます。一致するスキーマが見つかると、手順 b に戻り、今度は名前の 2 番目要素がオブジェクトとして使用され、修飾されたスキーマ内で検索されます。2 番目要素が以前に修飾されたスキーマ内のオブジェクトに対応していない場合、または 2 番目要素が存在しない場合は、エラーが戻されます。

   手順 c でスキーマが見つからない場合、オブジェクトは修飾できず、エラーが戻されます。

2. スキーマ・オブジェクトは修飾されています。名前の残りの要素は、見つかった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、hr.employees.department_id が名前の場合、hr はスキーマ、employees は表としてそれぞれ修飾され、department_id は 1 列に対応する必要があります (employees が表であるため)。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 による参照の解決では、オブジェクトが、他のオブジェクトが**存在しない**という事実に依存する場合があります。このような状況は、依存オブジェクトが使用している参照が、別のオブジェクトが存在する場合に異なって解釈される可能性がある場合に発生します。

関連項目：『Oracle9*i* データベース管理者ガイド』

共有 SQL の依存性管理

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

ローカルおよびリモートの依存性の管理

依存性の追跡や必要な再コンパイルは、Oracle によって自動的に実行されます。**ローカル依存性の管理**が発生するのは、Oracle が単一データベース内のオブジェクト間の依存性を管理する場合です。たとえば、プロシージャ内の文が、同じデータベース内の表を参照する場合があります。

リモート依存性の管理が発生するのは、Oracle がネットワークを介した分散環境で依存性を管理する場合です。たとえば、Oracle Forms トリガーは、データベース内のスキーマ・オブジェクトに依存する場合があります。また、分散データベースにおいて、ローカル・ビューの定義問合せがリモート表を参照する場合があります。

ローカル依存性の管理

Oracle は、データベース内部の依存性表を使用してローカル依存性を管理します。この表は、スキーマ・オブジェクトごとに依存オブジェクトを追跡し、記録するものです。参照オブジェクトが修正されると、この依存性表を使用して依存オブジェクトが識別され、それらの依存オブジェクトが無効にされます。

たとえば、ストアド・プロシージャ `UPDATE_SAL` が表 `JWARD.employees` を参照している場合を考えます。この表の定義になんらかの変更があると、ストアド・プロシージャ `UPDATE_SAL` を含めて、`JWARD.employees` を参照するすべてのオブジェクトのステータスが無効に変更されます。そのため、このプロシージャは、再コンパイルされて有効にされるまで実行できません。同様に、ユーザーの DML 権限を取り消すと、そのユーザーのスキーマ内の依存オブジェクトはすべて無効になります。ただし、認可が取り消されたために無効になったオブジェクトは、再認可によって有効にできます。その場合は、完全に再コンパイルする必要はありません。

リモート依存性の管理

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

ローカルおよびリモートのデータベース・プロシージャ間の依存性

ファンクション、パッケージおよびトリガーなど、ストアド・プロシージャの相互間の依存性は、分散データベース・システムにおいては**タイム・スタンプのチェック**または**シグネチャのチェック**を使用して管理されます。

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

> **関連項目：** タイム・スタンプまたはシグネチャによるリモート依存性管理の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

タイム・スタンプのチェック タイム・スタンプ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、コンパイル済みプロシージャには、それが参照するリモート・プロシージャごとに、スキーマ、パッケージ名、プロシージャ名、タイム・スタンプなどの情報も含まれています。

依存プロシージャが使用されると、Oracle は、コンパイル時に記録されたリモート・タイム・スタンプと、リモートで参照されたプロシージャのカレント・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。ただし、無効なプロシージャ・コールより前に実行された DML 文があると、すべてロールバックされます。

シグネチャのチェック Oracle には、**シグネチャ**を使用するリモート依存性の機能もあります。このシグネチャ機能の影響を受けるのは、リモート依存性のみです。ローカル依存性は、その環境でいつでも再コンパイルできるため影響を受けません。

プロシージャのシグネチャには、次の項目に関する情報が含まれます。

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

> **注意：** 重要なのはパラメータの型とモードのみです。パラメータの名前はシグネチャには影響しません。

シグネチャ依存性モデルが有効である場合に、依存単位に親単位のプロシージャへのコールが含まれており、かつ、そのプロシージャのシグネチャが互換性のない方法で変更されていると、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。

その他のリモート・スキーマ・オブジェクトの間の依存性

Oracle では、ローカル・プロシージャとリモート・プロシージャの間の依存性を除いては、リモート・スキーマ・オブジェクト間の依存性は管理されません。

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

表の変更後にビューとプロシージャが使用され、そのビューとプロシージャがエラーとなった場合でも、ローカル・ビューとプロシージャは無効になりません。この場合、ビューとプロシージャを、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、プリコンパイラおよびSQL*Moduleアプリケーションは、無名PL/SQLブロックを発行できます。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。

> **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。

16

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。

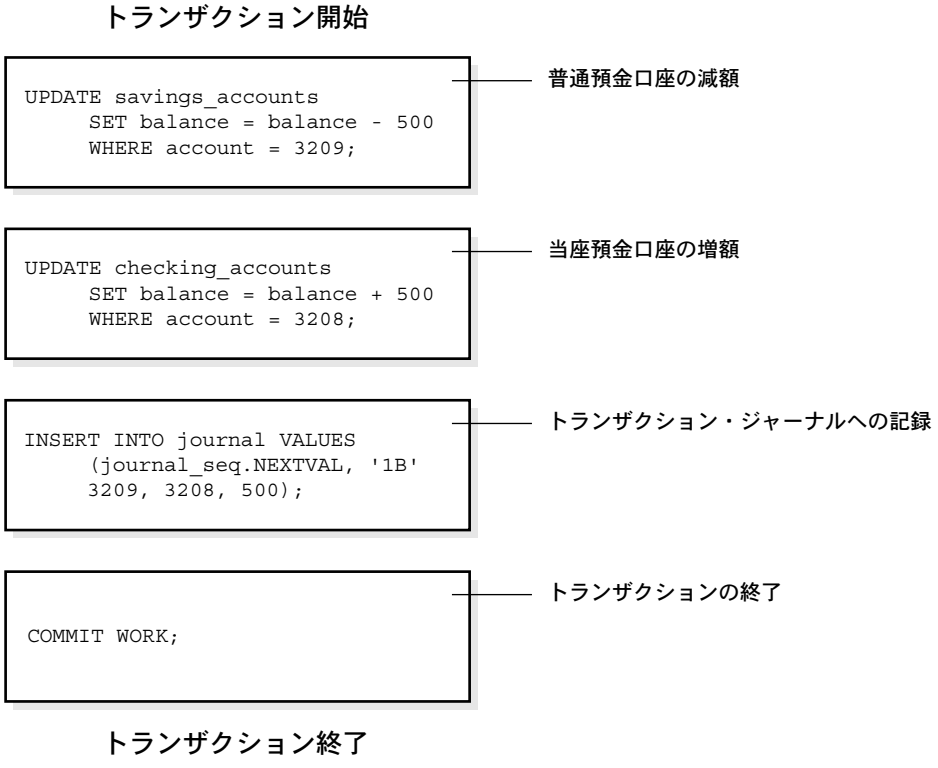
トランザクションの概念を具体的に説明するため、銀行業務データベースについて考えてみます。銀行の顧客が普通預金口座から当座預金口座へ預金を振り替える場合、トランザクションは次の 3 つの別々の操作で構成されます。

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

図 16-1 に銀行のトランザクションの例を示します。

図 16-1 銀行のトランザクション



文の実行とトランザクションの制御

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

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

ただし、その文が含まれているトランザクションがコミットされるまでは、トランザクションをロールバックしてその文のすべての変更を取り消せます。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。明示的な要求とは、ユーザーが COMMIT 文を発行することを意味します。暗黙的な要求は、アプリケーションの正常終了を介して、またはデータ定義言語などを介して出すことができます。トランザクションがコミットされて初めて、トランザクションの SQL 文による変更内容が確定され、他のユーザーがこれらの変更を参照できるようになります。このトランザクションよりも後に開始したトランザクションのみが、コミットされた変更を参照できます。

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： 16-10 ページ「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。

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

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

関連項目： 20-20 ページ「デッドロック」

再開可能領域割当て

Oracle 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。この機能は、**再開可能領域割当て**と呼ばれ、影響を受けた文は**再開可能文**と呼ばれます。

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については、『Oracle9*i* データベース管理者ガイド』を参照してください。

トランザクション管理の概要

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

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

トランザクションは、次のいずれかの状況が発生すると終了します。

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

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

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

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 により、ロールバック・セグメント・データを格納する SGA 内のバッファに、ロールバック・セグメント・レコードが生成されます。このロールバック情報には、トランザクションの SQL 文によって変更された古いデータ値が入れられます。
- Oracle により、SGA の REDO ログ・バッファに REDO ログ・エントリが生成されます。REDO ログ・レコードには、データ・ブロックおよびロールバック・ブロックに対する変更内容が含まれています。これらの変更は、トランザクションがコミットされる前にディスクに移動することがあります。
- SGA のデータベース・バッファに変更が加えられます。これらの変更は、トランザクションが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*）によって必ずしも即座にデータ・ファイルに書き込まれるわけではありません。データベースにとって最も効率的な時点で書き込まれます。この書込みは、トランザクションがコミットされる前に実行されたり、トランザクションがコミットされた後しばらくしてから実行されます。

トランザクションをコミットすると、次の処理が実行されます。

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

関連項目：

- 20-4 ページ「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、8-6 ページの「Oracle 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 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 では、コミットされていないトランザクション全体をロールバックできます。また、コミットされていないトランザクションの後半部分をセーブポイントと呼ばれるマーカーまでロールバックすることもできます。

次のすべてのタイプのロールバックで、同じ手順が使用されます。

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

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

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

関連項目：

- 16-9 ページ「トランザクションのセーブポイント」
- 20-4 ページ「ロックのメカニズムの概要」
- リカバリ中にコミット済みかどうかを問わず変更に対して実行される処理の詳細は、『Oracle9*i* Recovery Manager 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 はロールバック文が取得したデータ・ロックを解放します。以前にロックされていたリソースを待機していた他のトランザクションは、続行できます。以前にロックされていた行を更新しようとする他のトランザクションは、それらの行を更新できます。

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

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

トランザクションはアクティブであり、継続できます。

注意： 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ときには、異常終了を生じさせないために、次を使用します。

```
FOR UPDATE ... NOWAIT
```

UPDATE または DELETE 文を発行する前に行います。

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することができます。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・ディクショナリ表内の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、『Oracle9*i* データベース管理者ガイド』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、『Oracle9*i* SQL 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、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、またはすべてをロールバックするかのどちらか一方のみになるように保証するものです。また、2 フェーズ・コミット・メカニズムは、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙的 DML 操作も保護します。

Oracle の 2 フェーズ・コミット・メカニズムは、分散トランザクションを発行するユーザーからはまったく意識されません。事実、ユーザーは、トランザクションが分散していることも認識する必要はありません。トランザクションの終了を示す `COMMIT` 文は、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 Oracle サーバーの `RECO` プロセスが、関係するすべてのノード上でインダウト分散トランザクションを自動的にコミットするか、またはロールバックします。

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

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

関連項目：『Oracle9*i* Heterogeneous Connectivity 管理者ガイド』

ディスクリート・トランザクションの管理

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

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

ディスクリート・トランザクション中には、次のイベントが発生します。

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

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

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

> **関連項目：** BEGIN_DISCRETE_TRANSACTION プロシージャの詳細は、『Oracle9*i* 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。

自律型トランザクション

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

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

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

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

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

関連項目：『PL/SQL ユーザーズ・ガイドおよびリファレンス』

自律型ブロック内のトランザクション制御文

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

```
SET TRANSACTION
COMMIT
ROLLBACK
SAVEPOINT
ROLLBACK TO SAVEPOINT
```

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

関連項目：『PL/SQL ユーザーズ・ガイドおよびリファレンス』

17

トリガー

データベース・トリガーについて説明します。データベース・トリガーとは、PL/SQL、Java または C 言語で記述され、データベースに格納されているプロシージャであり、表またはビューが変更された場合、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。特定のスキーマ・オブジェクトに関する DML 文の発行、スキーマまたはデータベース内での DDL 文の発行、ユーザーによるイベントへのログオンまたはイベントからのログオフ、サーバーのエラー、データベースの起動またはインスタンス停止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。

この章の内容は、次のとおりです。

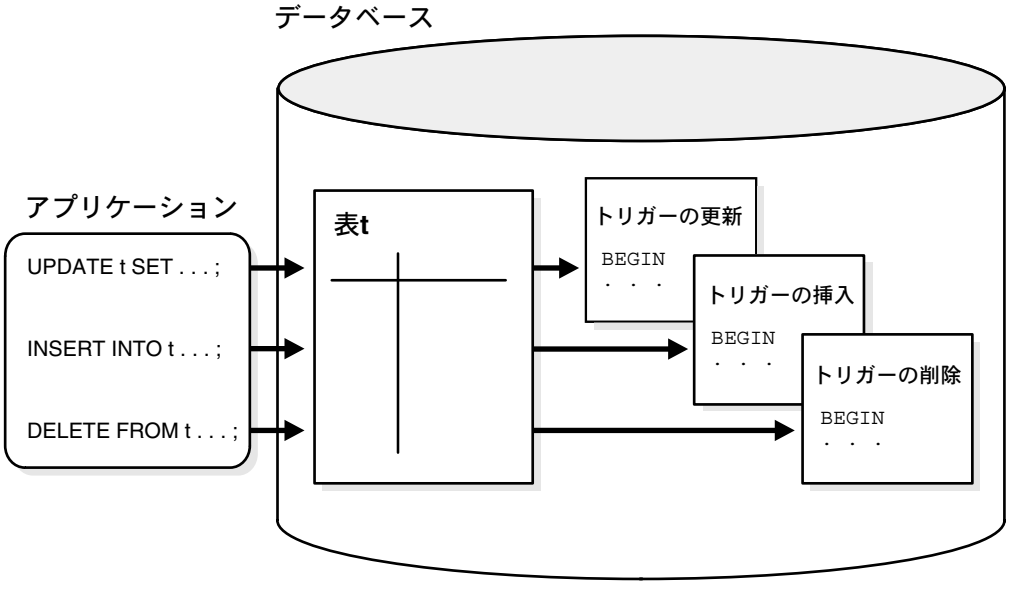
トリガーの概要

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

トリガーは、ストアド・プロシージャに似ています。データベースに格納されているトリガーには、1 単位として実行可能な SQL や PL/SQL 文を格納することができ、また、トリガーは他のストアド・プロシージャを起動できます。ただし、トリガーとプロシージャとでは、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暗黙的に起動されます。

図 17-1 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 17-1 トリガー



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

トリガーを起動するイベントは、次のとおりです。

- 表中のデータを変更する DML 文（INSERT、UPDATE または DELETE）。
- DDL 文
- 起動、停止およびエラー・メッセージなどのシステム・イベント
- ログオンやログオフなどのユーザー・イベント

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

関連項目：

- トリガーとストアド・プロシージャの類似点の詳細は、第 14 章「SQL、PL/SQL および Java」を参照してください。
- 17-7 ページ「トリガー・イベントまたはトリガーを実行する文」

トリガーの使用方法

トリガーは Oracle の標準機能を補い、高度にカスタマイズされたデータベース管理システムを提供します。たとえば、トリガーによって、表に対する DML 操作を通常の業務時間内のみに制限できます。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

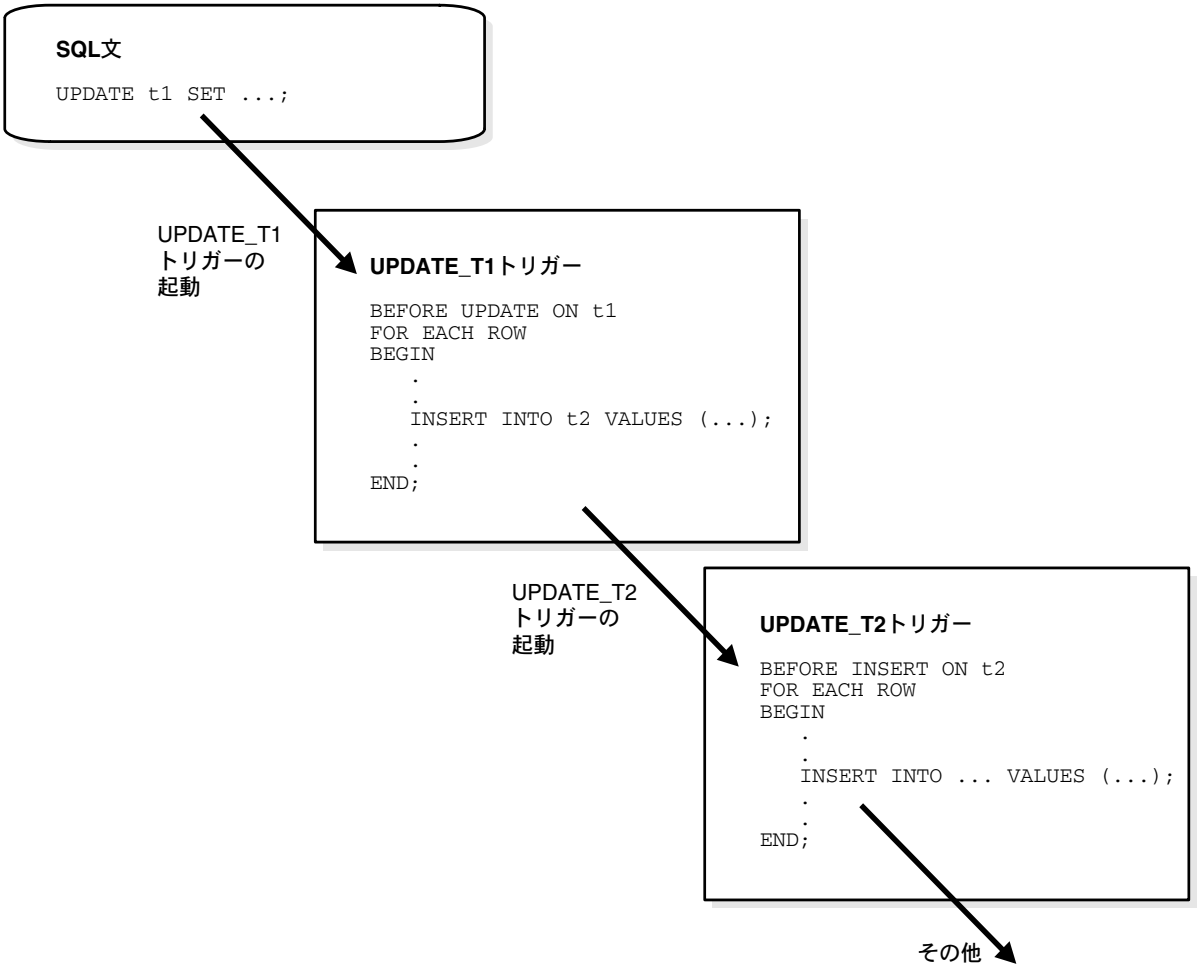
- ビューに対して DML 文が発行された場合の表データの変更
- データベース・イベント、ユーザー・イベントおよび SQL 文に関する情報の、サブスクライバ・アプリケーションに対する発行

> **関連項目：** トリガーの使用例は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

トリガーの使用上の注意

トリガーはデータベースのカスタマイズに使用すると便利ですが、必要な場合にのみ使用してください。トリガーを使用しすぎると相互依存性が複雑になるため、大規模アプリケーションのメンテナンスが困難になります。たとえば、トリガーが起動すると、そのトリガー・アクションに含まれる SQL 文によって他のトリガーが起動され、**トリガーが連鎖状態になる**ことがあります。これにより、意図しない影響を生じるおそれがあります。図 17-2 に、この状態を示します。

図 17-2 連鎖的なトリガー



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
  - CHECK
  - DELETE CASCADE
  - DELETE SET NULL

関連項目： 整合性制約の詳細は、21-4 ページの「Oracle がデータ整合性を規定する方法」を参照してください。

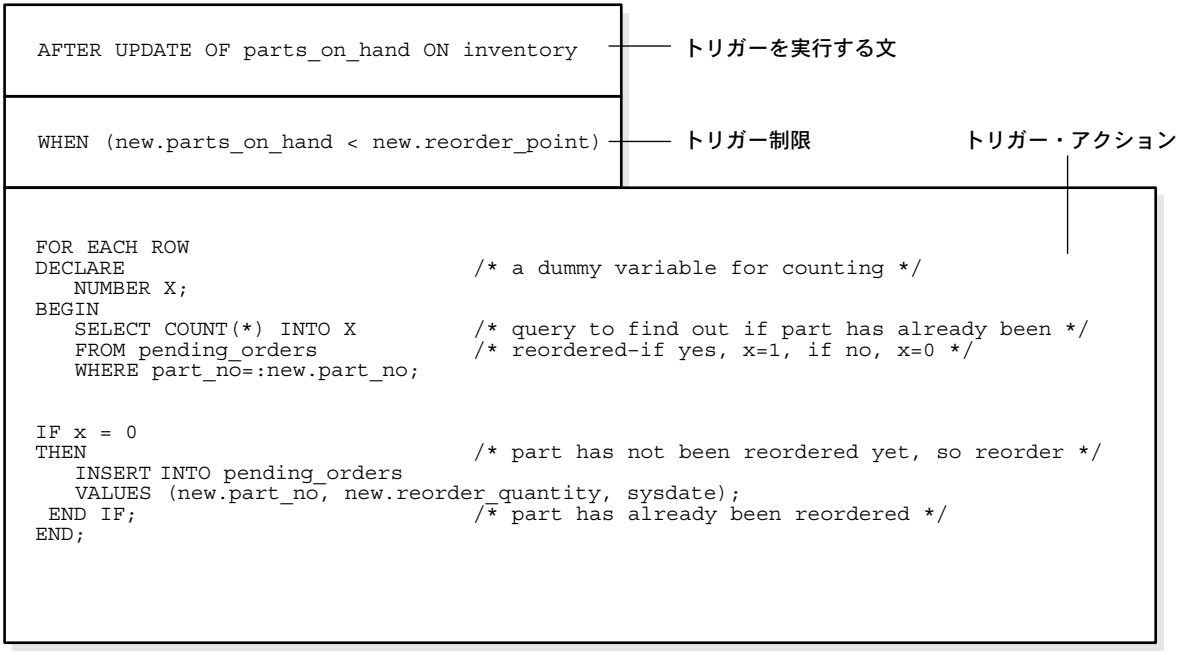
トリガーの各部分

トリガーには次の 3 つの基本部分があります。

- トリガー・イベントまたはトリガーを実行する文
- トリガー制限
- トリガー・アクション

図 17-3 に、トリガーの各部分を示します。この図は、正確な構文を示すためのものではありません。トリガーの各部分については、この後で詳しく説明します。

図 17-3 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、図 17-3 のトリガーを実行する文は次のとおりです。

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

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

トリガー・イベントには、次のように複数の SQL 文を指定できます。

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

この場合、inventory 表に対して INSERT、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・プロシージャのコール

トリガーが行トリガーの場合、トリガー・アクション内の文から、トリガーによって処理されている行の列値にアクセスできます。各列の古い値と新しい値には、相関名によってアクセスできます。

トリガーのタイプ

この項では、様々なタイプのトリガーについて説明します。

- 行トリガーと文トリガー
- BEFORE トリガーと AFTER トリガー
- INSTEAD OF トリガー
- システム・イベントとユーザー・イベントのトリガー

行トリガーと文トリガー

トリガーを定義するときに、次のようにトリガー・アクションの実行回数を指定できます。

- 多数の行を更新する UPDATE 文によって起動されるトリガーなどの場合、トリガーを実行する文の影響を受ける行ごとに 1 回ずつ
- 影響を受ける行数に関係なく、トリガーを実行する文ごとに 1 回ずつ

行トリガー

行トリガーは、表がトリガーを実行する文の処理の影響を受けるたびに実行されます。たとえば、UPDATE 文が表の複数の行を更新した場合、UPDATE 文が処理する行ごとに 1 つずつ行トリガーが起動されます。トリガーを実行する文の影響を受ける行がなければ、行トリガーは実行されません。

トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータまたは影響を受ける行数に依存する場合には、行トリガーが便利です。たとえば、図 17-3 は、トリガーを実行する文の影響を受ける各行の値を使用する行トリガーの一例です。

文トリガー

文トリガーは、トリガーを実行する文の影響を受ける表の行数とは関係なく、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 トリガーと AFTER トリガー

トリガーを定義するときに、**トリガーのタイミング**を指定できます。これは、トリガー・アクションを、トリガーを実行する文の前に実行するのか、後に実行するのかを指定するものです。BEFORE 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：

- 17-12 ページ「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、17-14 ページの「システム・イベントとユーザー・イベントのトリガー」を参照してください。

BEFORE トリガー

BEFORE トリガーは、トリガー文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 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。

- **AFTER 文トリガー**

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

ある表に対する 1 つの文に、同じタイプのトリガーを複数指定できます。たとえば、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 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。

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

> **関連項目：** トリガーの適用例は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

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 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。

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。

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込む場合、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：

- 第 13 章「オブジェクト・データ型とオブジェクト・ビュー」
- 『Oracle Call Interface プログラマーズ・ガイド』
- INSTEAD OF トリガーの例は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

変更不可能なビュー

INSTEAD OF トリガーを使用せずにデータを挿入、更新または削除でき、かつ後述の条件に一致する場合、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 句、CONNECT BY 句および 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： 10-19 ページ「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：

- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』
- CREATE TRIGGER 文の詳細は、『Oracle9*i* SQL 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
  - サーバー・エラー・メッセージ・イベント
- ユーザー・イベント
  - ユーザーのログオンとログオフ
  - DDL 文（CREATE、ALTER および DROP）
  - DML 文（INSERT、DELETE および UPDATE）

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

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

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

イベントの発行

イベントの発行では、Oracle アドバンスト・キューイングのパブリッシュ / 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たします。トリガーは、特定のシステム・イベントまたはユーザー・イベントの発生時に、DBMS_AQ パッケージを使用してメッセージをエンキューします。

関連項目：

- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』
- 『Oracle9*i* 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』

イベントの属性

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

システム・イベント

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

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

ユーザー・イベント

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

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

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

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

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

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

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

- BEFORE INSERT および AFTER INSERT トリガーは、表に挿入される行ごとに起動します。
- BEFORE UPDATE および AFTER UPDATE トリガーは、表中で更新される行ごとに起動します。

- BEFORE DELETE および AFTER DELETE トリガーは、表から削除される行ごとに起動します。

関連項目：

- 17-9 ページ「行トリガー」
- 17-10 ページ「BEFORE トリガーと AFTER トリガー」
- システム・イベントとユーザー・イベントのトリガーを使用したイベント発行の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

トリガーの実行

トリガーには次の 2 つのモードがあります。

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 は次の処理を自動的に行います。

- 1 つの SQL 文により複数のトリガーが起動される場合は、指定どおりの起動順序で各トリガーを実行します。
- 様々なタイプのトリガーに対して、指定された時点で整合性制約のチェックを実行し、整合性制約の違反を防止します。
- 問合せと制約に対して、読込み一貫性ビューを提供します。
- トリガー・アクションのコードで参照されるトリガーとスキーマ・オブジェクトの間の依存性を管理します。
- トリガーが分散データベースのリモート表を更新する場合には、2 フェーズ・コミットを使用します。
- 特定の 1 つの文について同じタイプのトリガーが複数存在する場合、順序不定で各トリガーが起動されます。

トリガーの実行モデルと整合性制約のチェック

1 つの SQL 文は、最大 4 種類のトリガーを起動します。

- BEFORE 行トリガー
- BEFORE 文トリガー
- AFTER 行トリガー
- AFTER 文トリガー

トリガーを実行する文またはトリガー内の文によって、1 つ以上の整合性制約のチェックが実施されます。またトリガーには、他のトリガーを起動する文を含めることもできます（連鎖的なトリガー）。

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

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

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

元の SQL 文が発行されます。

1. BEFORE 行トリガーが起動されます。
   a. BEFORE 行トリガー内の UPDATE 文により、AFTER 文トリガーが起動します。
      i. AFTER 文トリガーの文が実行されます。
      ii. AFTER 文トリガーによって変更された表の整合性制約がチェックされます。
   b. BEFORE 行トリガーの文が実行されます。
   c. BEFORE 行トリガーによって変更された表の整合性制約がチェックされます。
2. SQL 文が実行されます。

3. SQL 文から整合性制約がチェックされます。

この再帰には、次の 2 つの例外があります。

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

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

たとえば、前述の手順 1 ～ 2 を正しく実行し、手順 3 で整合性制約に違反があったとします。その違反の結果として、SQL 文によるすべての変更（手順 2）、起動された BEFORE 行トリガー（手順 1-b）および起動された AFTER 文トリガー（手順 1-a-i）がロールバックされます。

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

トリガーのデータ・アクセス

トリガーが起動されると、トリガー・アクション内で参照される表は、他のユーザーのトランザクション内の SQL 文によって変更されている最中である可能性があります。トリガー内で実行される SQL 文は、常に単独の SQL 文と同じ規則に従います。起動されたトリガーが読込み（問合せ）または書込み（更新）の対象にする値が、まだコミットされていないト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、参照先の表の読込み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。

次に、これらを具体的に説明する例を示します。

トリガーのデータ・アクセスの例 1 salary_check トリガー（本体）に、次の SELECT 文が含まれているとします。

```
SELECT min_salary, max_salary INTO min_salary, max_salary
  FROM jobs
  WHERE job_title = :new.job_title;
```

この例で、トランザクション T1 に jobs 表の max_salary 列に対する更新が含まれているとします。その更新の時点で、トランザクション T2 内の文によって salary_check トリガーが起動されます。起動されたトリガー内の SELECT 文（T2 に由来）は、コミットされていないトランザクション T1 による更新を参照しません。そのトリガーの問合せは、トランザクション T2 の読込み一貫性ポイントの値として、max_salary の古い値を戻します。

トリガーのデータ・アクセスの例 2 total_salary トリガーが、部門内のメンバー全員の給与総額が格納されている導出列をメンテナンスするものとします。

```
CREATE TRIGGER total_salary
AFTER DELETE OR INSERT OR UPDATE OF department_id, salary ON employees
 FOR EACH ROW BEGIN
  /* assume that department_id and salary are non-null fields */
   IF DELETING OR (UPDATING AND :old.department_id != :new.department_id)
   THEN UPDATE departments
  SET total_salary = total_salary - :old.salary
  WHERE department_id = :old.department_id;
   END IF;
   IF INSERTING OR (UPDATING AND :old.department_id != :new.department_id)
   THEN UPDATE departments
    SET total_salary = total_salary + :new.salary
    WHERE department_id = :new.department_id;
   END IF;
   IF (UPDATING AND :old.department_id = :new.department_id AND
    :old.salary != :new.salary )
```

```
   THEN UPDATE departments
    SET total_salary = total_salary - :old.salary + :new.salary
   WHERE department_id = :new.department_id;
  END IF;
END;
```

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

PL/SQL トリガーの記憶域

PL/SQL トリガーは、ストアド・プロシージャと同じように、コンパイル済みの形式で格納されます。CREATE TRIGGER 文がコミットされると、P コード（疑似コード）と呼ばれるコンパイル済み PL/SQL コードがデータベースに格納され、トリガーのソース・コードは共有プールからフラッシュされます。

> **関連項目：** PL/SQL コードのコンパイルと格納の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

トリガーの実行

Oracle は、プロシージャ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。この点を除けば、トリガーはストアド・プロシージャと同じ方法でチェックされ、実行されます。

> **関連項目：** ストアド・プロシージャの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

トリガーの依存性のメンテナンス

プロシージャと同様に、トリガーも参照しているオブジェクトに依存します。トリガー・アクションで参照されるスキーマ・オブジェクトに対するトリガーの依存性は、Oracle によって自動的に管理されます。トリガーの依存性についての問題は、ストアド・プロシージャの依存性についての問題と同じです。トリガーは、ストアド・プロシージャと同様に扱われます。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 15 章「スキーマ・オブジェクト間の依存性」

第 VI 部

パラレル SQL とダイレクト・ロード・インサート

第 VI 部では、SQL 文のパラレル実行とダイレクト・ロード・インサート機能について説明します。第 VI 部には、次の章が含まれています。

- 第 18 章「SQL 文のパラレル実行」
- 第 19 章「ダイレクト・パス・インサート」

18

SQL 文のパラレル実行

この章では、SQL 文のパラレル実行について説明します。この章の内容は、次のとおりです。

- パラレル実行の概要
- パラレル実行の動作
- パラレル化できる SQL 操作

注意： この章で説明するパラレル実行機能は、Oracle9*i* Enterprise Edition を購入した場合にのみ使用できます。Oracle9*i* Enterprise Edition の詳細は、『Oracle9*i* データベース新機能』を参照してください。

パラレル実行の概要

Oracle で SQL 文をパラレル実行する場合は、複数のプロセスが同時に作業を実行して、単一の SQL 文を実行します。1 つの文を実行するのに必要な作業を複数のプロセスに分けることにより、1 つのプロセスで文を実行する場合に比べて、Oracle はさらに高速で実行できます。これらを**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、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パラレル・プロセス（MPP）システムにおいては、パラレル実行を活用して最大のパフォーマンスを引き出せます。1 つの Oracle システムにある複数の CPU に文の処理を分散させることができるためです。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。

パラレル実行によって、システムはハードウェア・リソースの使用が最適化され、パフォーマンスが一段と向上します。システムの CPU とディスク・コントローラの負荷がすでに大きい場合は、パフォーマンスを改善するため、パラレル実行を使用する前にシステムの負荷を軽減するか、またはハードウェア・リソースを増やす必要があります。

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：** パラメータ・ファイルとデータベースをチューニングして、パラレル実行を活用できるようにする方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パラレル実行を実装する場合

ほとんどの OLTP システムでは、業務時間中にパラレル実行が使用されることはありません。ただし、業務時間外には、パラレル実行により大量のバッチ操作が効率的に処理されます。たとえば、銀行では、口座への利子の適用に必要な数百万件の更新作業に、パラレル化バッチ・プログラムを使用できます。

パラレル実行の最も一般的な例は DSS での使用です。非常に大きな表の結合や検索を含む複合問合せは、パラレルでの実行が最適です。

パラレル実行は、大量のデータにアクセスする様々な操作に有効です。パラレル実行により、次の操作のパフォーマンスが向上します。

- 問合せ
- 大規模な索引の作成
- 大量な挿入、更新および削除
- 集計とコピー

パラレル実行は、次の特長をすべて備えたシステムに有効です。

- 対称型マルチプロセッサ（SMP）、クラスタまたは大規模パラレル・システム（複数 CPU など）
- 十分な I/O 帯域幅
- 使用率の低い CPU または断続的に使用される CPU（たとえば、CPU の通常の使用率が 30% 未満のシステム）
- ソート、ハッシングおよび I/O バッファなどのメモリー集中型の処理もサポートする十分なメモリー

現在のシステムがこれらの特長を 1 つでも欠く場合には、パラレル実行による大幅なパフォーマンス改善は期待できません。実際、非常に使用率の高いシステムまたは I/O 帯域幅が十分ではないシステムでは、パラレル実行によりパフォーマンスが低下することがあります。

> **関連項目：** パラレル実行を実装する時期に関する詳細は『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パラレル実行を実装しない場合

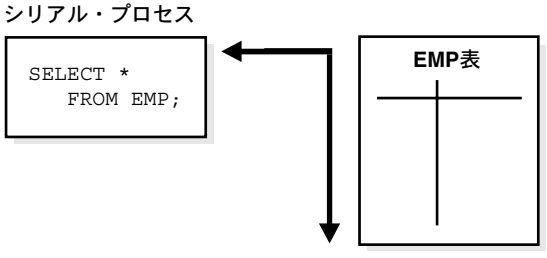
通常、次の場合には、パラレル実行は有用ではありません。

- 通常使用する問合せまたはトランザクションが非常に短い（数秒以下）環境。これにはほとんどのオンライン・トランザクション・システムが含まれます。パラレル実行サーバーの調整に関連するコストがかかるため、これらの環境におけるパラレル実行は有用ではありません。短いトランザクションでは、この調整にかかるコストがパラレル化の効果を上回ることがあります。
- CPU、メモリーまたはI/Oリソースの利用率が非常に高い環境。パラレル実行は使用可能なハードウエア・リソースを活用するように設計されています。使用可能なリソースが存在しない場合は、パラレル実行による効果が得られないばかりかパフォーマンスに悪影響を及ぼす可能性があります。

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 employees`など）を実行するには、図18-1に示すように、1つのプロセスで操作全体を実行します。

図18-1 シリアルの全表スキャン

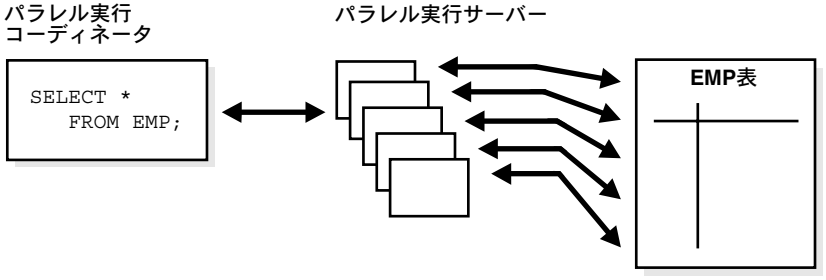


パラレル実行では、複数の**パラレル処理**を使用して操作がパラレルで実行されます。**パラレル実行コーディネータ**という1つのプロセスが、文の実行を複数の**パラレル実行サーバー**に分配し、すべてのサーバー・プロセスからの結果を調整してユーザーに送り返します。

図18-2では、表`employees`のスキャンを実行するパラレル実行サーバーを説明しています。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込まれます。グラニュルはコーディネータにより生成されます。各グラニュルの大きさは、表`employees`の物理ブロックの範囲にあります。実行サーバーへのグラニュルのマッピングは静的に行われるのではなく、実行時間で決定されます。実行サーバーがグラニュルに対応する表`employees`の行の読込みを完了したときにグラニュルが残っている場合は、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用されるまで続けられます。すなわち、表`employees`全体が読み込まれるまで続けられます。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 18-2 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：

- マルチユーザー環境において作業を分割する方法と DOP を処理する方法は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。
- グラニュルの詳細は、第 7 章「メモリー・アーキテクチャ」を参照してください。

SQL 文のパラレル化

各 SQL 文ごとに、解析時に最適化プロセスおよびパラレル化プロセスが実行されます。このため、データが変更されて、さらに最適な実行計画やパラレル化計画が使用可能になると、Oracle はその新しい状況に自動的に対応します。

オプティマイザが文の実行計画を決定した後で、パラレル実行コーディネータは、実行計画に含まれる各操作のパラレル化の方法を決定します。コーディネータは、操作をパラレルで実行できるかどうかと、実行する場合のパラレル実行サーバーの数を決定する必要があります。操作に使用するパラレル実行サーバーの数のことを並列度と呼びます。

操作間のパラレル化

イントラ・オペレーション並列化とインター・オペレーション並列化を理解するために、次の文を考えてみます。

```
SELECT * FROM employees ORDER BY employee_id;
```

実行計画では、employees 表の全体スキャンと、その後に行う、取り出した行の employee_id 列の値に基づくソート操作が実装されます。この例では、last_name 列には索引が作成されていないものとします。また、この問合せの並列度は 4 に設定されているとします。つまり、それぞれの操作について 4 つのパラレル実行サーバーをアクティブにできることになります。

同時に実行される前述の 2 つの操作（スキャンとソート）には、それぞれ専用のパラレル実行サーバーの集合が用意されます。したがって、両操作はパラレル化されます。個別操作のパラレル化では、パラレル実行サーバーにより、比較的小さな行集合に同じ操作が行われますが、これにより、**イントラ・オペレーション並列化**と呼ばれるパラレル化が実現します。ある操作から別の操作にデータが流れるパラレル実行サーバーの複数の集合において、2 つの操作が同時に実行されると、**インター・オペレーション並列化**と呼ばれるパラレル化が実現します。

Oracle サーバーの操作にはプロデューサ（作成者）およびコンシューマ（使用者）という特性があるため、特定のツリーに含まれる操作のうち、実行時間を最小にするために同時に実行する必要がある操作は 2 つのみです。

図 18-3 に、この例の問合せのパラレル実行を示します。

図 18-3 インター・オペレーション並列化と動的パーティション化

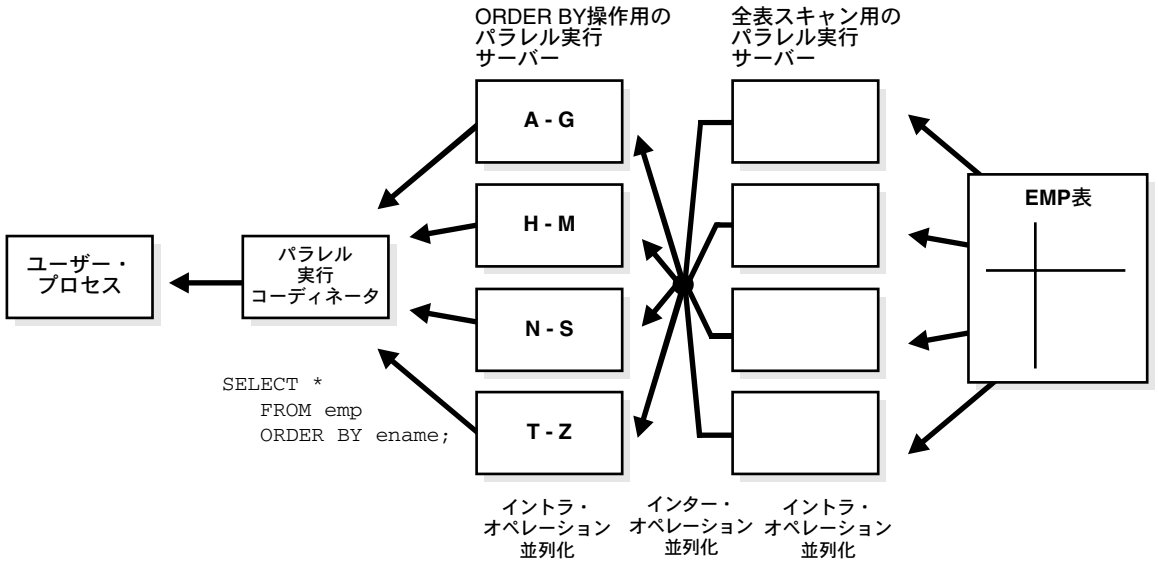


図 18-3 からわかるとおり、並列度は 4 ですが、問合せに実際に関与しているのは 8 つのパラレル実行サーバーです。これは、親オペレータと子オペレータを同時に実行できるためです（インター・オペレーション並列化）。

スキャン操作に含まれるすべてのパラレル実行サーバーが、ソート操作を実行するパラレル実行サーバーのうちの適切なプロセスに行を送っていることにも注目してください。パラレル実行サーバーによってスキャンされた行の ename 列の内容が A ～ G の間の値であれば、その行は最初の ORDER BY パラレル実行サーバーへ送られます。スキャン操作が完了すると、ソート・プロセスはソート結果をコーディネータへ戻し、コーディネータは問合せ結果の全体をユーザーに戻します。

注意： 一連のパラレル実行サーバーの操作が完了すると、データ・フローの中で次の順位にある操作に移ります。たとえば、前述の図で、ORDER BY の後に別の ORDER BY 操作が続いている場合は、表スキャンを実行しているパラレル実行サーバーが、表スキャン完了後に 2 番目の ORDER BY 操作を実行します。

並列度

パラレル実行コーディネータは、1 つの SQL 文の処理のために、そのインスタンスのパラレル実行サーバーを 2 つ以上登録することがあります。1 つの操作に関連付けられたパラレル実行サーバーの数を**並列度**と呼びます。

この並列度が直接適用されるのは、イントラ・オペレーション並列化のみです。インター・オペレーション並列化が可能であれば、パラレル実行サーバーの合計数は、指定した並列度の 2 倍になることがあります。パラレル実行サーバーの 3 つ以上の集合を同時に実行することはできません。パラレル実行サーバーの各集合は、複数の操作を実行できます。最適なインター・オペレーション並列化を保証するには、アクティブにするパラレル実行サーバーの集合を 2 つに限定します。

パラレル実行は、複数の CPU とディスクを効率的に使用して問合せに迅速に応答するように設計されています。複数のユーザーが同時にパラレル実行を使用すると、使用可能な CPU、メモリーおよびディスク・リソースは急速に消費されます。

Oracle には、次のように、パラレル実行環境に伴うリソース使用状況を管理するために、複数の方法が用意されています。

- マルチユーザー問合せ調整アルゴリズム — システムにかかる負荷が大きくなるにつれて、並列度を低下させます。このオプションを切り替えるには、`ALTER SYSTEM` 文または初期化パラメータ・ファイルの `PARALLEL_ADAPTIVE_MULTI_USER` パラメータを使用します。
- ユーザーのリソース制限とプロファイル — ユーザーのセキュリティ・ドメインの一部として、各ユーザーが使用できる各種のシステム・リソースの容量に制限を設定できます。
- Database Resource Manager — 様々なユーザー・グループにリソースを割当てできます。

関連項目：

- `PARALLEL_ADAPTIVE_MULTI_USER` については、『Oracle9*i* データベース・リファレンス』を参照してください。
- SQL 文 `ALTER SYSTEM` の構文は、『Oracle9*i* SQL リファレンス』を参照してください。
- 『Oracle9*i* データ・ウェアハウス・ガイド』

パラレル問合せのイントラ・オペレーションとインター・オペレーションの例

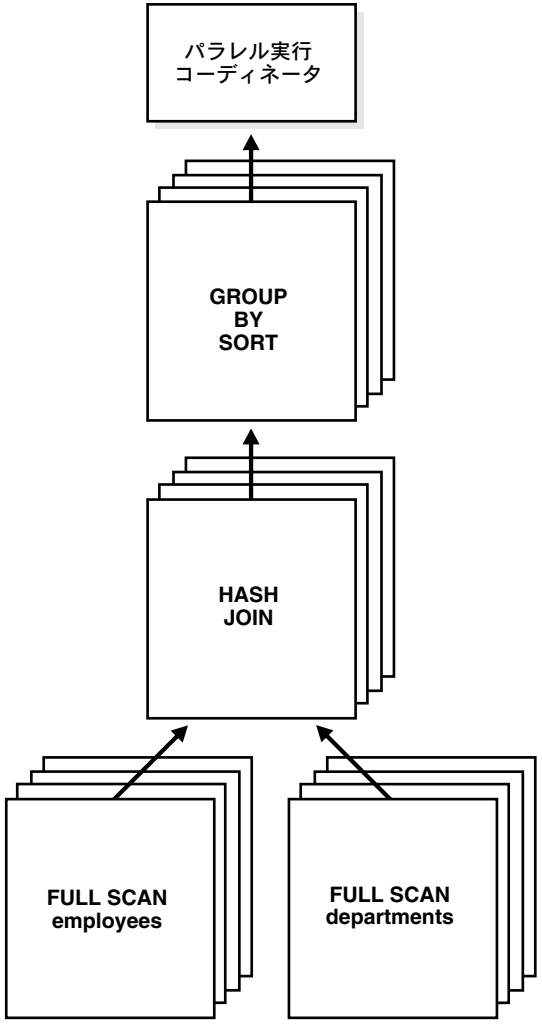
パラレル問合せのイントラ・オペレーション並列化とインター・オペレーション並列化の例として、複雑な問合せを考えてみます。

```
SELECT /*+ PARALLEL(employees 4) PARALLEL(departments 4) USE_HASH(employees) ORDERED
*/
       MAX(salary), AVG(salary)
FROM employees, departments
WHERE employees.department_id = departments.department_id
GROUP BY employees.department_id;
```

結合順序と結合方法を規定し、表 `employees` と `departments` の並列度（DOP）を指定するために、問合せ内でヒントが使用されていることに注意してください。通常は、オプティマイザにより結合順序と結合方法が決定されます。

この問合せに対応する問合せ計画またはデータ・フロー図式を図 18-4 に示します。

図 18-4 表の結合のデータ・フロー・ダイアグラム



パラレル実行サーバーの 2 つの集合、SS1 と SS2 では、この計画が次のように実行されます。各サーバー集合（SS1 と SS2）には、DOP を指定する問合せ内の PARALLEL ヒントにより 4 つの実行プロセスが与えられます。つまり、パラレル実行サーバーの各集合には 4 つのプロセスがあるため、DOP は 4 になります。

スレーブ集合の SS1 が最初に表 employees をスキャンします。SS2 は SS1 から行をフェッチして、その行のハッシュ表を構築します。つまり、SS2 内の親サーバーと SS2 内の子サーバーは次のように同時に作業します。一方が employees をパラレルでスキャンして、他方が SS1 から送られる行を使用してハッシュ結合用のハッシュ表をパラレルで構築します。これがインター・オペレーション並列化の例です。

SS1 が表 employees 全体のスキャンを終了すると（すなわち、employees のすべてのグラニュルまたはタスク単位が使用される）、表 departments をパラレルでスキャンします。SS1 は行を SS2 内のサーバーに送ります。SS2 はハッシュ結合を完了するため、プローブをパラレルで実行します。SS1 は表 departments のパラレル・スキャンおよび SS2 への行の送付を終了すると、GROUP BY のパラレル実行を開始します。これが、2 つのサーバーの集合を同時に実行して、各操作のパラレル実行によるイントラ・オペレーション並列化を実現する一方、問合せツリー内の各種演算子に対するインター・オペレーション並列化を実現する方法です。

パラレル実行でもう 1 つの重要なのは、あるサーバーが別のサーバーに設定されているときに、サーバーから行の送信が行われた場合、行が再パーティション化されることです。図 18-4 の問合せ計画では、SS1 内のサーバー・プロセスが employees の行のスキャンを完了した後、SS2 のどのサーバー・プロセスが行を受け取るのでしょうか。問合せツリーからあふれた行のパーティション化は、行が流入する演算子によって決定されます。この場合、employees のパラレル・スキャンを行う SS1 からあふれた行を、パラレル・ハッシュ結合を行う SS2 にパーティション化する処理は、結合列の値に基づくハッシュ・パーティション化によって行われます。つまり、employees をスキャンするサーバー・プロセスは、列 employees.employee_id の値のハッシュ関数を計算して、列を受け取る SS2 内のサーバー・プロセスの数を決定します。パラレル問合せで使用するパーティション化の方法は、問合せの EXPLAIN PLAN に明示的に示されます。実行サーバーの集合間に送られる行のパーティション化を、ハッシュ、範囲その他の方法によって表の分割が行われる Oracle のパーティション化機能と混同しないでください。

> **関連項目：** パラレル問合せを伴う EXPLAIN PLAN の例は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パラレル化できる SQL 操作

ほとんどの操作をパラレル化できます。次の項目は、パフォーマンスを改善するために通常パラレル化されます。

- パラレル問合せ
- パラレル DDL
- パラレル DML
- SQL*Loader

> **関連項目：** パラレル DML の制限事項およびウェアハウス設計の考慮事項に関する特定の情報は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

パラレル問合せ

問合せと副問合せは、SELECT 文のみでなく、DDL 文と DML 文（INSERT、UPDATE および DELETE）の問合せ部でパラレル化できます。ただし、リモート・オブジェクトを参照している DDL 文や DML 文の問合せ部は、パラレル化できません。問合せ部でリモート・オブジェクトを参照するパラレルの DML 文または DDL 文を発行すると、操作は自動的にシリアルで実行されます。

> **関連項目：** パラレル問合せ文の構文と制限の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

パラレル DDL

パラレル DDL は、通常の DDL を使用する環境で使用できます。ただし、データベースの設計に際し、考慮すべき事項もあります。制限事項として重要なのは、パラレル DDL が、オブジェクトまたは LOB 列を持つ表では使用できないことです。

パラレル化できる DDL 文

CREATE TABLE AS SELECT 文、CREATE INDEX 文および ALTER INDEX REBUILD 文をパラレル化できます。表がパーティション化されると、ALTER TABLE MOVE または [SPLIT または COALESCE] 文もパラレル化できます。索引がパーティション化されると、ALTER INDEX REBUILD [または SPLIT] のパラレル化も使用できます。

これらのすべての DDL 操作は、パラレル実行でもシリアル実行でも、NOLOGGING モードで実行できます。

索引構成表の CREATE TABLE 文は、AS SELECT 句を使用してもしなくてもパラレル化できます。

操作ごとに異なるパラレル化が使用されます。パーティション表のパラレル CREATE TABLE AS SELECT およびパーティション索引のパラレル CREATE INDEX は、パーティション数と同じ並列度で実行されます。

最適なパラレル操作を実行するには、正確な統計が必要です。

関連項目：

- パラレル DDL 文の構文と制限の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- LOB の制限事項は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

パラレル DML

パラレル DML（パラレル INSERT、パラレル UPDATE およびパラレル DELETE）では、パラレル実行メカニズムを使用して、大規模のデータベース表や索引への大がかりな DML 操作のスピードアップやスケールアップを図ります。単一の DML 文の一部として INSERT ... SELECT 文を使用して、複数表へ行を挿入することもできます。パラレル DML は、通常の DML を使用する環境で使用できます。

通常、データ操作言語（DML）には問合せが含まれますが、パラレル DML は、パラレルで実行される挿入、更新、アップサートおよび削除のみを指します。

関連項目：

- パラレル DML 文の構文と制限の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- パラレル DML の制限事項およびウェアハウス設計の考慮事項に関する特定の情報は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

SQL*Loader

大量のデータが定期的に発生する SQL*Loader の使用をパラレル化できます。ロードをスピードアップするには、次の例に示すようなパラレル・ダイレクト・パス・ロードを使用します。

```
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD1.CTL DIRECT=TRUE PARALLEL=TRUE
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD2.CTL DIRECT=TRUE PARALLEL=TRUE
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD3.CTL DIRECT=TRUE PARALLEL=TRUE
```

パラメータ・ファイルを使用しても同じことができます。

パラレル・ロードの間は索引がメンテナンスされないことに注意してください。

> **関連項目：** パラレル・ロードの構文と制限の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

パラレルで実行する文の作成方法

文をパラレルで実行する方法は、パラレル操作のタイプにより異なります。次の 3 つのパラレル操作があります。

- パラレル問合せ
- パラレル DDL
- パラレル DML

パラレル問合せ

SQL 問合せ文をパラレル化するには、スキャンされる表の 1 つ以上がパラレル属性を持つ必要があります。

パラレル DDL

SQL の DDL 文をパラレル化するには、パラレル句を指定する必要があります。

パラレル DML

シリアル DML とパラレル DML ではロック方法が異なるため、使用前に必ずパラレル DML を明示的に使用可能にします。SQL の DML 文をパラレル化するには、まずセッション内でパラレル DML を使用可能にします。

```
ALTER SESSION ENABLE PARALLEL DML;
```

これにより、PDML 制限に違反しないかぎり、パラレル属性を持つ表に対して発行されるすべての DML がパラレル化されます。次に例を示します。

```
INSERT INTO mytable SELECT * FROM origtable;
```

関連項目：

- パラレル化を実装する構文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- パラメータ・ファイルからパラレル化を実装するための構文の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- パラレル DML の制限に関する特定の情報は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

19

ダイレクト・パス・インサート

この章では、シリアル挿入またはパラレル挿入用の Oracle ダイレクト・パス・インサート機能について説明します。ダイレクト・パス・インサートといくつかの DDL 文で使用可能な NOLOGGING 機能についても説明します。この章の内容は、次のとおりです。

- ダイレクト・パス・インサートの概要
- ダイレクト・パス・インサートの利点
- シリアルおよびパラレルのダイレクト・パス・インサート
- パーティション表および非パーティション表へのダイレクト・パス・インサート
- ダイレクト・パス・インサートとロギング・モード
- ダイレクト・パス・インサートのその他の考慮事項

注意： この章で説明するパラレル・ダイレクト・パス・インサート機能は、Oracle9*i* Enterprise Edition を購入した場合にのみ使用可能です。詳細は、『Oracle9*i* データベース新機能』を参照してください。

関連項目：

- パラレル実行 INSERT の注意点の詳細は、第 18 章「SQL 文のパラレル実行」を参照してください。
- 『Oracle9*i* データ・ウェアハウス・ガイド』

ダイレクト・パス・インサートの概要

Oracle では、次の 2 種類の方法で表にデータを挿入します。

- **通常の挿入オペレーション**の実行中に Oracle は、表内の空き領域を再利用して新しく挿入されたデータを既存のデータにインターリーブします。この操作の間、Oracle は、参照整合性制約も維持します。
- **ダイレクト・パス・インサート・オペレーション**の実行中に Oracle は、表内の既存データの後ろに、挿入されたデータを追加します。データはデータファイルに直接書き込まれ、バッファ・キャッシュはバイパスされます。既存データの空き領域は再利用されず、参照整合性制約は無視されます。これらの手順を組み合せることでパフォーマンスを強化できます。

ダイレクト・パス・インサート文または Oracle のダイレクト・パス・ローダー・ユーティリティの SQL*Loader の使用により、ダイレクト・パス・インサート・オペレーションを実装できます。この項では、ダイレクト・パス・インサートを説明します。

関連項目：

- ダイレクト・パス・ロードと SQL*Loader の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。
- ダイレクト・パス・インサートの制限事項のリストは、『Oracle9*i* SQL リファレンス』を参照してください。

ダイレクト・パス・インサートの利点

ダイレクト・パス・インサートには、次のようなパフォーマンス上の利点があります。

- ダイレクト・パス・インサートの実行中、REDO および UNDO エントリのロギングを使用禁止にすることができます。それに比べて、従来の挿入オペレーションでは、これらのオペレーションが空き領域を再利用して参照整合性を維持するので、これらのエントリに常にログしている必要があります。
- 既存の表にあるデータから新しい表を作成する場合、新しい表を作成してからデータを挿入する方法か CREATE TABLE ... AS SELECT 文を実行する方法を選択できます。表を作成し、ダイレクト・パス・インサート・オペレーションを使用することにより、挿入オペレーション中にターゲット表で定義された索引を更新します。それに比べて、CREATE TABLE ... AS SELECT 文により作成された表では、定義された索引が存在しないため、後で定義する必要があります。
- ダイレクト・パス・インサート・オペレーションでは、パラレル・モードで実行されるときでも、トランザクションのアトム性が保証されます。ただし、パラレル・ダイレクト・パス・ロード（SQL*Loader を使用）の実行中は、アトム性は保証されません。
- パラレル・ダイレクト・パス・ロードの実行中にエラーが発生した場合、索引には、ロード終了時に UNUSABLE とマークが付けられます。それに比べ、パラレル・ダイレクト・パス・インサートでは、索引の更新時にエラーが発生すると文がロールバックされます。

シリアルおよびパラレルのダイレクト・パス・インサート

パラレル DML モードでの挿入では、ダイレクト・パス・インサートがデフォルトです。パラレル DML モードで実行するには、次の要件を満たしている必要があります。

- Oracle Enterprise Edition がインストールされている必要があります。
- セッション内のパラレル DML を使用可能にする必要があります。そのため、次の文を実行します。

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- ターゲット表の作成時または作成後も引き続き、ターゲット表にパラレル属性が指定されている必要があります。パラレル属性が指定されていないと、挿入オペレーションごとに PARALLEL ヒントを指定する必要があります。

ダイレクト・パス・インサートを使用禁止にするには、各 INSERT 文に NOAPPEND ヒントを指定します。これにより、パラレル DML モードがオーバーライドされます。

> **関連項目：** ヒントの使用方法の詳細は、『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

シリアル・モードでの挿入では、INSERT 文の副問合せの INSERT キーワードの直後または SELECT キーワードの直後の各挿入文に APPEND ヒントを指定することで、ダイレクト・パス・インサートをアクティブ化する必要があります。

注意： ダイレクト・パス・インサートでサポートされるのは INSERT 文の副問合せ構文のみで、VALUES 句はサポートされません。INSERT 文の副問合せ構文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

パーティション表および非パーティション表へのダイレクト・パス・インサート

パーティション表および非パーティション表の両方でダイレクト・パス・インサートを使用できます。

パーティション表および非パーティション表へのシリアル・ダイレクト・パス・インサート

シングル・プロセスでは、データは、表のセグメントまたは各パーティションのセグメントの現行の最高水位標を超えて挿入されます。(**最高水位標**は、ブロックがデータ受入れ用にフォーマットされたことのないレベルです。) COMMIT を実行すると、最高水位標が新しい値に更新され、ユーザーからもデータが見えるようになります。

パーティション表へのパラレル・ダイレクト・パス・インサート

この状況はシリアル・ダイレクト・パス・インサートに類似しています。各パラレル実行サーバーに 1 つ以上のパーティションが割り当てられ、1 つのパーティションで 1 つのプロセスのみ作動します。各パラレル実行サーバーは、割り当てられたパーティション・セグメントのカレント最高水位標を超えてデータを挿入します。COMMIT を実行すると、各パーティション・セグメントの最高水位標が新しい値に更新され、ユーザーからもデータが見えるようになります。

非パーティション表へのパラレル・ダイレクト・パス・インサート

各パラレル実行サーバーが新しい一時セグメントを割り当てて、そこにデータを挿入します。COMMIT を実行すると、パラレル実行コーディネータは新しい一時セグメントを 1 次表セグメントにマージします。このため、ユーザーからもセグメントが見えるようになります。

ダイレクト・パス・インサートとロギング・モード

ダイレクト・パス・インサートでは、挿入オペレーション実行中に REDO 情報およびロールバック情報のログを取るかどうかを選択します。

- 表、パーティション、索引または LOB 記憶域へのロギング・モードを、これらの作成時（CREATE 文に）または作成後（ALTER 文に）に指定できます。
- これらの作成時または作成後に、LOGGING または NOLOGGING を指定しなかった場合は、次のようになります。
  - パーティションのロギング属性のデフォルト値は、その表のロギング属性に設定されます。
  - 表または索引のロギング属性のデフォルト値は、それらが常駐する表領域のロギング属性に設定されます。
  - LOB 記憶域に CACHE を指定した場合は、LOB 記憶域のロギング属性のデフォルト値は LOGGING に設定されます。CACHE を指定しなかった場合は、ロギング属性のデフォルト値は、LOB 値が常駐する表領域の値に設定されます。
- 表領域のロギング属性を CREATE TABLESPACE または ALTER TABLESPACE 文に設定します。

注意： データベースまたは表領域が FORCE LOGGING モードの場合、ダイレクト・パス・インサートでは、LOGGING または NOLOGGING のどちらを指定していても、常にログが記録されます。

ロギングありのダイレクト・パス・インサート

このモードでは、Oracle は、インスタンスおよびメディア・リカバリ用に完全な REDO ログを実行します。データベースが ARCHIVELOG モードの場合、オンライン REDO ログをテープにアーカイブすることができます。データベースが NOARCHIVELOG モードの場合、ディスク障害の場合を除きインスタンスのクラッシュをリカバリできます。

ロギングなしのダイレクト・パス・インサート

このモードでは、Oracle は、データは挿入されますが、REDO または UNDO ログは生成しません。（ただし、新しいエクステントが無効であることをマークするために最小限のロギングが実行されます。またディクショナリの変更については常にログが記録されます。）このモードでは、パフォーマンスが改善されます。ただし、後でメディア・リカバリを実行する必要がある場合は、エクステント無効化レコードにより、論理的な破損としてブロックの範囲にマークが付けられます。これらの REDO データのログが記録されなかったためです。したがって、挿入オペレーションなどの後ではデータのバックアップが重要です。

関連項目：

- リカバリ情報は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。
- 挿入以外のオペレーションでのロギング・モードに関する情報は、『Oracle9*i* SQL リファレンス』を参照してください。

ダイレクト・パス・インサートのその他の考慮事項

ダイレクト・パス・インサートでの索引のメンテナンス

索引が付けられた表（パーティション表または非パーティション表）のダイレクト・パス・インサート・オペレーションの最後に、Oracle は、索引のメンテナンスを実行します。この索引のメンテナンスは、パラレル・ダイレクト・パス・インサートの場合にはパラレル実行サーバーによって行われ、シリアル・ダイレクト・パス・インサートの場合には単一のプロセスによって行われます。INSERT オペレーションの前に索引を削除して後で再作成することにより、索引メンテナンスによるパフォーマンスの影響を回避できます。

ダイレクト・パス・インサートでの領域に関する考慮事項

ダイレクト・パス・インサートは、セグメントの空きリスト内の既存の領域を使用しないため、従来型パス・インサートより多くの領域が必要です。

すべてのシリアル・ダイレクト・パス・インサート・オペレーションとパーティション表へのパラレル・ダイレクト・パス・インサートは、影響を受けたセグメントの最高水位標を超えてデータを挿入します。この操作には追加の領域が必要です。

非パーティション表へのパラレル・ダイレクト・パス・インサートでは、各並列度ごとに一時セグメントを作成するため、より多くの領域が必要です。自動モードでローカルで管理される表領域内に非パーティション表が存在しない場合は、NEXT 記憶域パラメータと PCTINCREASE 記憶域パラメータおよび MINIMUM EXTENT 表領域パラメータの値を変更して、一時セグメントに対し十分な（しかし、過剰ではない）記憶域を提供します。これらのパラメータには、次のような値を選択してください。

- 各エクステントが小さくなりすぎない程度のサイズ（1MB 以上）。この設定は、オブジェクト内の合計エクステント数に影響します。
- 各エクステントが、パラレル・インサートによって必要以上に大きいセグメントの領域を無駄にすることのない程度のサイズ。

  関連項目： これらのパラメータの設定方法は、『Oracle9*i* SQL リファレンス』を参照してください。

ダイレクト・パス・インサート・オペレーションが完了すると、これらのパラメータをシリアル操作に対し、より適切な設定にリセットできます。

ダイレクト・パス・インサートでのロックに関する考慮事項

ダイレクト・パス・インサートの実行中、Oracle は、表（または、パーティション表のすべてのパーティション）を排他ロックします。このため、ユーザーは、表に対する挿入、更新または削除などの操作を同時に実行することはできません。また、索引の作成および構築などの操作を同時に行うことも許可されません。問合せの同時実行はサポートされていますが、その問合せで戻されるのは、挿入オペレーション以前の情報のみです。

第 VII 部

データの保護

第 VII 部では、Oracle のデータベース内におけるデータ保護方法、およびデータベース管理者によってデータ保護をさらに強化する方法について説明します。

第 VII 部には、次の章が含まれています。

- 第 20 章「データの並行性と整合性」
- 第 21 章「データ整合性」
- 第 22 章「データベース・アクセスの制御」
- 第 23 章「権限、ロールおよびセキュリティ・ポリシー」
- 第 24 章「監査」

20

データの並行性と整合性

この章では、マルチユーザー・データベース環境で Oracle がどのようにデータ整合性を維持するかについて説明します。この章の内容は、次のとおりです。

- マルチユーザー環境におけるデータの並行性と整合性の概要
- データの並行性と整合性の管理方法
- データをロックする方法
- フラッシュバック問合せ

マルチユーザー環境におけるデータの並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並行性と整合性の制御が重要になります。

- **データ並行性**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ル化可能なモードでは、複数のトランザクションは、同時にではなく 1 つずつ（シリアルに）実行されるように見えます。

一般に、この程度までのトランザクションの分離が望ましいとされますが、このモードで多数のアプリケーションを実行すると、アプリケーションのスループットがかなり低下する可能性があります。同時に実行される各トランザクションの完全な分離とは、あるトランザクションが問合せを実行している表に対して、他のトランザクションが挿入を実行できない状態を指します。つまり、現実には、トランザクションの完全な分離とパフォーマンスとの間の妥協点を考慮する必要があります。

Oracle には 2 つの分離レベルがあり、これによってアプリケーションの開発者は、一貫性を保ちながら高いパフォーマンスを実現する各操作モードを使用できます。

> **関連項目：** データベースに対応付けられたビジネス・ルールを規定するデータ整合性の詳細は、第 21 章「データ整合性」を参照してください。

回避可能な現象とトランザクション分離レベル

ANSI/ISO SQL 規格（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込み : まだコミットされていないトランザクションが書き込んだデータを、別のトランザクションが読み込む現象
- 非リピータブル・リード（ファジー読込み）: あるトランザクションが以前に読み込んだデータをもう一度読み込んだときに、コミットされた別のトランザクションによってそのデータが変更または削除されたことが明らかになる現象
- 仮読込み : あるトランザクションが検索条件を満たす一連の行を戻す問合せを二度実行する間に、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 20-1 に、各分離レベルを示します。

表 20-1 分離レベル別による回避可能な読込み現象

分離レベル	内容を保証しない読込み	非リピータブル・リード	仮読込み
非コミット読込み	あり	あり	あり
コミット読込み	なし	あり	あり
リピータブル・リード	なし	なし	あり
シリアル化可能	なし	なし	なし

Oracle 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、コミット読込みおよびシリアル化可能です。デフォルトはコミット読込みです。

> **関連項目：** コミット読込みとシリアル化可能な分離レベルの詳細は、20-5 ページの「データの並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並行性、一貫性、および整合性の問題を解決しています。**ロック**は、同じリソースにアクセスしている複数のトランザクションの間で破壊的な相互作用が起きないようにするメカニズムです。

リソースには、次の 2 つの一般的なタイプのオブジェクトが含まれます。

- ユーザー・オブジェクト。たとえば、表と行（構造体とデータ）。
- ユーザーには見えないシステム・オブジェクト。たとえば、メモリー内の共有データ構造やデータ・ディクショナリ行。

関連項目： ロックの詳細は、20-18 ページの「データをロックする方法」を参照してください。

データの並行性と整合性の管理方法

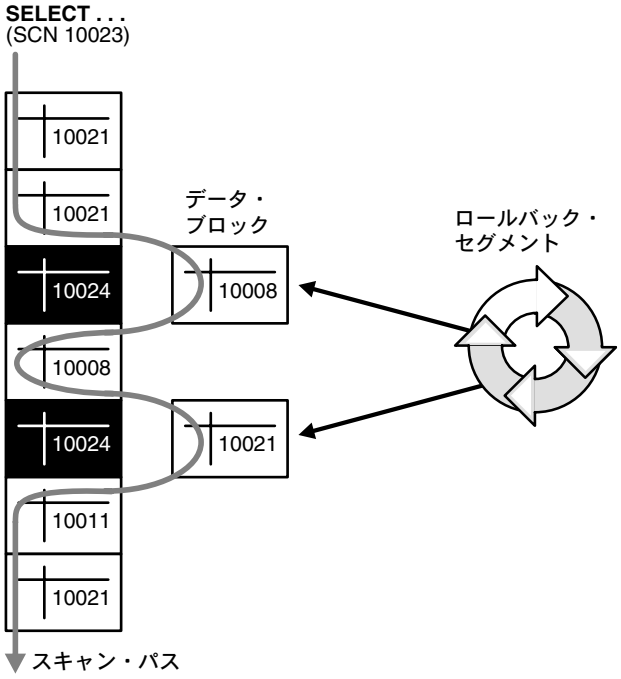
Oracle は、マルチバージョン一貫性モデルや、様々なタイプのロックおよびトランザクションを使用することによって、マルチユーザー環境でのデータ整合性を維持します。この項の内容は、次のとおりです。

マルチバージョン並行性制御

Oracle では、ある問合せで参照されるデータのすべてが同一時点でのデータになるように、問合せに対して自動的に読込み一貫性が提供されます（**文レベルの読込み一貫性**）。読込み一貫性は、1 つのトランザクション内のすべての問合せに対しても提供できます（**トランザクション・レベルの読込み一貫性**）。

Oracle は、これらの一貫したデータの参照を実現するために、ロールバック・セグメント内に保持される情報を使用します。ロールバック・セグメントは、コミットされていないトランザクション、または最近コミットされたトランザクションによって変更されたデータの古い値が含まれています。図 20-1 に、Oracle がロールバック・セグメント内のデータを使用して、文レベルの読込み一貫性をどのように提供するかを示します。

図 20-1 トランザクションと読込み一貫性



問合せが実行段階に入ると、現行のシステム変更番号（SCN）が決定されます。図 20-1 では、このシステム変更番号が 10023 になっています。問合せのためにデータ・ブロックを読み込むと、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みのすべてのデータが戻されます。問合せの実行中に発生する他のトランザクションの変更は参照されません。このようにして、各問合せに対して一貫性のあるデータが戻されることが保証されます。

文レベルの読込み一貫性

Oracle では、常に**文レベル**の読込み一貫性が保たれています。これによって、1 つの問合せによって戻されるすべてのデータは、その問合せが開始された時点のものであることが保証されます。そのため、問合せは、内容が保証されないデータも、その問合せの実行中にコミットされたトランザクションによって変更されたデータもまったく参照しません。問合せの実行中にその問合せで参照できるのは、その問合せが開始される前にコミットされたデータのみです。問合せは、文の実行の開始後にコミットされた変更は参照しません。

あらゆる問合せに対して一貫した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。

トランザクション・レベルの読込み一貫性

Oracle では、**トランザクション・レベルの読込み一貫性**を保つこともできます。シリアル化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。つまり、シリアル化可能トランザクションが発行した問合せではそのトランザクション自身が実行した変更が参照されるという点を除けば、同一のトランザクション内にあるどの問合せで参照されるデータも、1 つの時点を基準とした一貫性が保たれているということです。トランザクション・レベルの読込み一貫性によってリピータブル・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。

Real Application Clusters における読込み一貫性

Real Application Clusters では、読込み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。Real Application Clusters 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、高速で待機時間の短いインターコネクトを使用してこれを行います。

> **関連項目：** 詳細は、『Oracle9*i* Real Application Clusters 概要』を参照してください。

Oracle の分離レベル

Oracle には次のトランザクション分離レベルがあります。

分離レベル	説明
コミット読込み	デフォルトのトランザクション分離レベル。あるトランザクションによって実行されるそれぞれの問合せで参照されるのは、その問合せ（トランザクションではなく）が開始される前にコミットされたデータのみです。Oracle の問合せでは、内容が保証されない（コミットされていない）データが読み込まれることはありません。 Oracle では、問合せで読み込まれたデータを他のトランザクションで変更できるため、問合せを 2 回実行する間に最初の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・リードと仮読込み（実体のない読込み）の現象の両方が起こり得ます。
シリアル化可能	シリアル化可能トランザクションでは、そのトランザクションを開始した時点でコミット済みの変更と、そのトランザクション自身が INSERT 文、UPDATE 文および DELETE 文で実行した変更のみが参照されます。シリアル化可能トランザクションでは、非リピータブル・リードや仮読込みの現象は起きません。
読取り専用	読取り専用トランザクションでは、そのトランザクションを開始した時点でコミット済みの変更のみが参照され、INSERT 文、UPDATE 文および DELETE 文は使用できません。

分離レベルの設定

アプリケーション・デザイナ、アプリケーション開発者およびデータベース管理者は、アプリケーションとワークロードに応じて、それぞれのトランザクションに適した分離レベルを選択できます。トランザクションの分離レベルは、トランザクションの開始時に次の文のいずれかを使用して設定できます。

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION ISOLATION LEVEL READ ONLY;
```

各トランザクションを SET TRANSACTION 文で開始する場合のネットワー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、『Oracle9*i* SQL リファレンス』を参照してください。

コミット読込み分離

Oracle のデフォルト分離レベルは、コミット読込みです。このレベルの分離は、競合するトランザクションの数が少ない環境に適しています。問合せごとに、それぞれ独自のマテリアライズド・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・リードと仮読込みが発生することもありますが、高いスループットが得られます。コミット読込み分離は、競合するトランザクションの数が少ない環境に適した分離レベルです。

シリアル化可能な分離

シリアル化可能な分離は、次のような環境に適しています。

- 大規模データベースを使用し、短いトランザクションでごく少数の行しか更新しない環境。
- 2 つの同時実行トランザクションが同一の行を更新するようなことが比較的少ない環境。
- 比較的長時間にわたって実行されるトランザクションが主に読取り専用である環境。

シリアル化可能な分離では、同時実行トランザクションが実行できる変更は、トランザクションが順に 1 つずつ実行される場合に実行できるデータベース変更のみです。特に、シリアル化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、シリアル化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、データ・ブロック内に格納されている制御情報、つまりブロック内の行のうち、どの行にコミットされた変更が含まれていて、どの行にコミットされてない変更が含まれているかを示す情報が使用されます。ある意味で、ブロックには、ブロック内の各行に影響を与えたトランザクションの最新の履歴が含まれていると考えられます。ここに保存される履歴の量は、CREATE TABLE および ALTER TABLE の INITRANS パラメータによって制御されます。

場合によっては、ごく最近のトランザクションで行が更新されたかどうかを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。それにより、ブロックにアクセスした最新のトランザクションの履歴を記録するのに十分な記憶域が各ブロックに割り当てられます。

シリアル化可能トランザクションが、その開始後にコミットされたトランザクションによって変更されたデータを更新または削除しようとすると、次のようなエラーが生成されます。

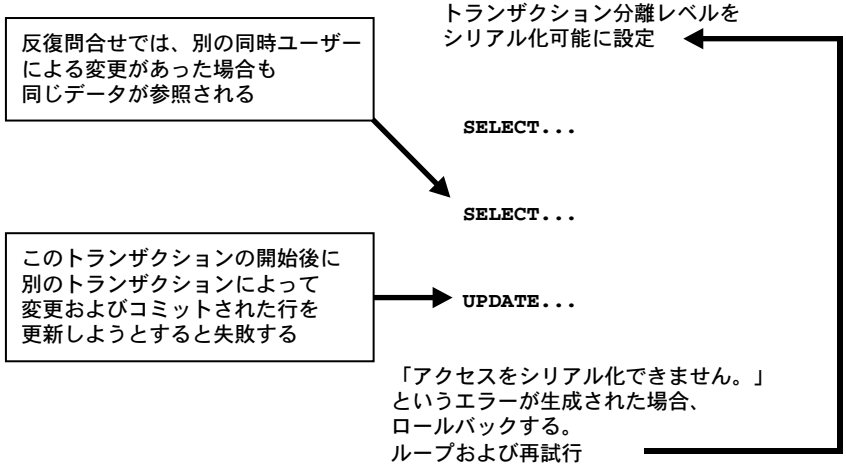
```
ORA-08177: このトランザクションのアクセスをシリアル化できません。
```

シリアル化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ロールバックします。

図 20-2 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 20-2 シリアル化可能トランザクションの失敗



コミット読込み分離とシリアル化可能な分離の比較

Oracle では、異なる特性を持つ 2 つのトランザクション分離レベルのどちらかを選択できます。コミット読込みとシリアル化可能のどちらの分離レベルでも、高度な一貫性と並行性が提供されます。どちらの分離レベルでも、Oracle の読込み一貫性によるマルチバージョン並行性制御モデルと、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配置を考慮して設計されています。

トランザクション集合の一貫性

Oracle におけるコミット読込みとシリアル化可能な分離レベルを理解するため、次のような使用例を考えます。データベースの表の集合（またはデータの任意の集合）があり、それらの表内の行を特定の順序で何度か読み込み、一連のトランザクションをある特定の時点でコミットするとします。ある操作（問合せまたはトランザクション）のどの読込みでも、コミット済みのトランザクションの同一の集合によって書き込まれたデータが戻される場合、その操作には**トランザクション集合の一貫性**があります。一部の読込みにはあるトランザクション集合による変更が反映されており、他の読込みには別のトランザクション集合による変更が反映されている場合、その操作にはトランザクション集合の一貫性がありません。トランザクション集合の一貫性のない操作では、事実上、コミット済みの 1 つのトランザクション集合が反映された状態のデータベースを一貫して参照できません。

Oracle では、コミット読込みモードで実行されるトランザクションには文単位でのトランザクション集合の一貫性が提供されます。シリアル化可能なモードでは、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 20-2 に、Oracle でのコミット読込みトランザクションとシリアル化可能トランザクションの主な違いをまとめます。

表 20-2 コミット読込みトランザクションとシリアル化可能トランザクション

	コミット読込み	シリアル化可能
内容を保証しない書込み	なし	なし
内容を保証しない読込み	なし	なし
非リピータブル・リード	あり	なし
仮読込み	あり	なし
ANSI/ISO SQL 92 への準拠	あり	あり
マテリアライズド・ビュー読込み時間	文	トランザクション
トランザクション集合の一貫性	文レベル	トランザクション・レベル
行レベル・ロック	あり	あり
読込みが書込みをブロックするか	なし	なし

表 20-2 コミット読込みトランザクションとシリアル化可能トランザクション（続き）

	コミット読込み	シリアル化可能
書込みが読込みをブロックするか	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

コミット読込みトランザクションとシリアル化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ロールバックされてロックが解除されるまで、待ち状態になります。この最初のトランザクションがロールバックした場合、どの分離モードでも、待ち状態のトランザクションは最初のトランザクションが存在しなかったかのように、以前にロックされた行を変更できるようになります。

ただし、阻止している最初のトランザクションがコミットされてロックが解除された場合、コミット読込みトランザクションは、目的の更新処理を続行します。また、シリアル化可能トランザクションの場合には、そのシリアル化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、エラー、「アクセスをシリアル化できません。」を出して失敗します。

参照整合性

Oracle は、読込み一貫性トランザクションでもシリアル化可能トランザクションでも読込みロックを使用しないため、あるトランザクションで読み込んだデータが別のトランザクションによって上書きされる可能性があります。アプリケーション・レベルでデータベースの一貫性のチェックを実行するトランザクションでは、データの変更がトランザクションから見えなくても、読み込んだデータがトランザクションの実行中に変更されることはないとは想定しないでください。このことを考慮してアプリケーション・レベルの一貫性チェックをコーディングしないかぎり、シリアル化可能トランザクションを使用しても、データベースの一貫性が損なわれることがあります。

> **関連項目：** 参照整合性とシリアル化可能トランザクションの詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

注意： Real Application Clusters では、コミット読込みトランザクションとシリアル化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、**シリアル化可能**トランザクションに関与するすべてのサーバーで、シリアル化可能な分離モードがサポートされている必要があります。

シリアル化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ル化可能トランザクションが更新しようとすると、そのトランザクションにはエラーが戻されます。トランザクションがロールバックして再試行できるのは、リモート・サーバーがシリアル化可能トランザクションをサポートしている場合のみです。

これとは対照的に、**コミット読込み**トランザクションでは、シリアル化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：『Oracle9*i* データベース管理者ガイド』

分離レベルの選択

アプリケーション・デザイナおよびアプリケーション開発者は、アプリケーションのコーディング以外にアプリケーションのパフォーマンスと一貫性のニーズに基づいて分離レベルを選択する必要があります。

多数の同時実行ユーザーが次々にトランザクションを送る環境では、トランザクションの予想到着率および応答時間の要求の面から、トランザクションのパフォーマンス要件を評価する必要があります。高いパフォーマンスが必要な環境では、分離レベルを選択する際に一貫性と並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込み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 の 2 つの分離モードは、行レベル・ロックと Oracle のマルチバージョン並行性制御システムとの組合せによって、高水準の一貫性、並行性およびパフォーマンスを提供します。Oracle では、読込み側と書込み側が互いに処理をブロックしません。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、コミット読込みとシリアル化可能などちらの分離レベルでも、コミットされていない（内容が保証されない）データを読むことなく、高パフォーマンスを実現する高水準の並行性を提供します。

コミット読込み分離

多くのアプリケーションでは、コミット読込みが最適な分離レベルです。コミット読込み分離では、並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、仮読込みや非リピータブル・リードのために、一貫性のない結果が生じる危険性が多少高くなります。

トランザクションの到着率が高く、高いパフォーマンスが求められる多くの環境では、シリアル化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、仮読込みおよび非リピータブル・リードによって間違った結果が生じる危険性が非常に低くなります。コミット読込み分離は、いずれの環境にも適しています。

Oracle のコミット読込み分離では、すべての問合せについてトランザクション集合の一貫性が提供されます。つまり、どの問合せでも一貫性のある状態のデータが参照されます。したがって、マルチバージョン並行性制御を使用しない他のデータベース管理システム上で実行される場合にはさらに高水準の分離を必要とするようなアプリケーションでも、多くの場合、コミット読込み分離で十分に対応できます。

コミット読込み分離モードでは、アプリケーションのロジックで「アクセスを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、仮読込みおよび非リピータブル・リードの防止は重要ではありません。したがって、前述のようなエラー・チェックや再試行のコードを各トランザクションで記述する必要を避けるために、多くの開発者はコミット読込みを選択します。

シリアル化可能な分離

Oracle のシリアル化可能な分離は、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。

シリアル化可能な分離モードは、仮読込みと非リピータブル・リードを防止することによってある程度まで一貫性を向上できるため、読込み / 書込みトランザクションが 1 つの問合せを複数回実行する場合に重要になることがあります。

他の処理系のシリアル化可能な分離では書込みのみでなく読込みについてもブロックがロックされるのに対し、Oracle では非ブロッキング問合せときめ細かい行レベル・ロックが提供され、それらによって書込み / 読込みの競合が少なくなります。読込み / 書込みの競合が多く発生するアプリケーションでは、Oracle のシリアル化可能な分離は、他のシステムより大幅に高いスループットを提供します。このため、Oracle 上ではシリアル化可能な分離に適していても、他のシステムではシリアル化可能な分離に適さないアプリケーションもあります。

Oracle のシリアル化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、読込み / 書込みトランザクションで複数の一貫した問合せを発行する必要がある場合は、この分離レベルが適しています。シリアル化可能なモードでは、`READ ONLY` トランザクションで実現される一貫性が確保される一方で、`INSERT`、`UPDATE` および

DELETE トランザクションも実行できるため、サマリー・データを生成してそのデータをデータベースに格納する報告書作成アプリケーションでは、シリアル化可能なモードを使用してください。

> **注意：** 副問合せを持つ DML 文を含むトランザクションは、読込み一貫性を保証するためにシリアル化可能な分離を使用する必要があります。

アプリケーション開発者がシリアル化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ロールバックおよび再試行のために、追加の作業が必要になります。他のデータベース管理システムでデッドロックを管理する際にも、同様の追加コーディングが必要です。社内標準を遵守する場合や、複数のデータベース管理システム上で実行するアプリケーションを作成する場合には、シリアル化可能なモードに対応したトランザクションの設計が必要なことがあります。シリアル化の可能性エラーをチェックして、再試行するトランザクションは、Oracle のコミット読込みモード、つまりシリアル化の可能性エラーを生成しないモードで使用できます。

シリアル化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。従来型の読込みをロックしたシリアル化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、読込みトランザクションであっても、短い更新トランザクションの処理を阻止したり、逆に短いトランザクションが長時間実行のトランザクションの処理を阻止する場合があるためです。

アプリケーション開発者は、シリアル化可能なモードを使用するときはトランザクションのロールバックと再試行に要するコストを考慮する必要があります。デッドロックが頻繁に発生する読込みロックのシステムと同様に、シリアル化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、シリアル化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 のシリアル化可能なモードには SQL-92 との互換性があり、読込みロックの処理系と比較して多くの利点がありますが、意味としてはそれらのシステムと同じではありません。アプリケーション設計者は、Oracle での読込みは、他のシステムとは違って、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。シリアル化可能なモードを使用しているアプリケーションを他の環境から Oracle に移植する場合には、この問題を検討する必要があります。

データベースの静止

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。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。次に例を示します。

  1. データベース表のスキーマを変更する。
  2. PL/SQL プロシージャを、このデータベース表の新しいスキーマを使用する新しいバージョンに更新する。

  手順 1 と 2 の間には、新しい表のスキーマは PL/SQL プロシージャの実装と一貫性がありません。この非一貫性のため、PL/SQL プロシージャを同時に実行しようとするユーザーに不都合な影響が出ます。

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。

Database Resource Manager 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。ALTER SYSTEM QUIESCE RESTRICTED 文を発行できるのは、ユーザー SYS および SYSTEM のみです。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 はすべてのインスタンスの Database Resource Manager 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 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、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 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、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 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 は Database Resource Manager 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、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。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* データベース管理者ガイド』

データをロックする方法

ロックは、同じ**リソース**、つまりユーザー・オブジェクト（表や行など）またはユーザーには見えないシステム・オブジェクト（メモリー内の共有データ構造やデータ・ディクショナリ行など）のどちらかにアクセスする複数のトランザクションの間で、破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 によって必要なロックが自動的に取得されます。そのため、ユーザーがそのような詳細事項にかかわる必要はありません。Oracle は、最も高度なデータ並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。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： 20-22 ページ「ロックの種類」

トランザクションとデータ並行性

Oracle ではロック・メカニズムを使用して、トランザクション間のデータの並行性と整合性を実現します。Oracle のロック・メカニズムはトランザクション制御と密接に結び付けられているため、アプリケーション・デザイナがトランザクションを適切に定義するだけで、ロックは Oracle によって自動的に管理されます。

Oracle のロックは完全に自動化されていて、ユーザー・アクションを必要としません。すべての SQL 文について暗黙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 のデフォルトのロック・メカニズムは、最高度のデータ並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： 20-34 ページ「明示的（手動）データ・ロック」

ロックのモード

Oracle では、マルチユーザー・データベースで 2 つのロック・モードを使用します。

- 排他ロック・モードは、関連リソースが共有されないようにします。このロック・モードはデータを変更するために取得します。リソースを排他的にロックした最初のトランザクションは、その排他ロックが解除されるまで、そのリソースを変更できる唯一のトランザクションになります。
- 共有ロック・モードでは、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。データの読込みを実行する複数のユーザーはそのデータを共有でき、また共有ロックを保持することによって、排他ロックを必要とする書込みユーザーの同時アクセスを防ぎます。複数のトランザクションが同じリソースについて共有ロックを取得できます。

ロックの期間

あるトランザクション内の文によって取得されたすべてのロックは、そのトランザクションの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込み、更新内容の消失、有害な DDL 操作など）が防止されます。あるトランザクションの SQL 文によって加えられた変更を参照できるのは、そのトランザクションがコミットされた後に開始される別のトランザクションのみです。

トランザクションをコミットまたはロールバックすると、そのトランザクション内の文によって取得されたすべてのロックが解除されます。また、セーブポイントまでロールバックした場合には、そのセーブポイントより後で取得されたロックが解除されます。ただし、ロックを解除されて使用可能となったリソースに対してロックを取得できるのは、ロック中だったリソースを待機していなかったトランザクションのみです。ロック中のリソースを待機していたトランザクションは、元のトランザクションが完全にコミットされるかロールバックされるまで待機し続けます。

データ・ロック変換とロックの段階的拡大の比較

トランザクションは、トランザクション内で挿入、更新または削除の対象になる行すべてに対して排他ロックを保持します。行ロックは、制限の最も高い程度で取得されるため、ロック変換はまったく必要なく、実行されません。

Oracle は、低い制限の表ロックを、より高い制限の適切な表ロックに自動的に変換します。たとえば、トランザクションが表の行をロックするために、`FOR UPDATE` 句を指定した `SELECT` 文を使用する場合を考えます。その結果、排他行ロックとその表に対する行共有表ロックが取得されます。後で、トランザクションがロック中の 1 つ以上の行を更新する場合、行共有表ロックは自動的に行排他表ロックに変換されます。

ロックの段階的拡大が発生するのは、あるレベル（行レベルなど）で多数のロックが保持されている場合に、各ロックをデータベースが上位レベル（表レベルなど）の別のロックに変更した場合です。たとえば、あるユーザーが表の中の多数の行をロックした場合、データベースによっては、そのユーザーが保持する複数の行ロックを単一の表ロックに自動的に拡大する場合があります。ロックの数は少なくなりますが、ロックされている対象の制限は大きくなります。

Oracle では、ロックの段階的拡大が発生することはありません。ロックの段階的拡大はデッドロックの可能性を大幅に増します。たとえば、システムがトランザクション T1 のためにロックを拡大しようとしますが、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。

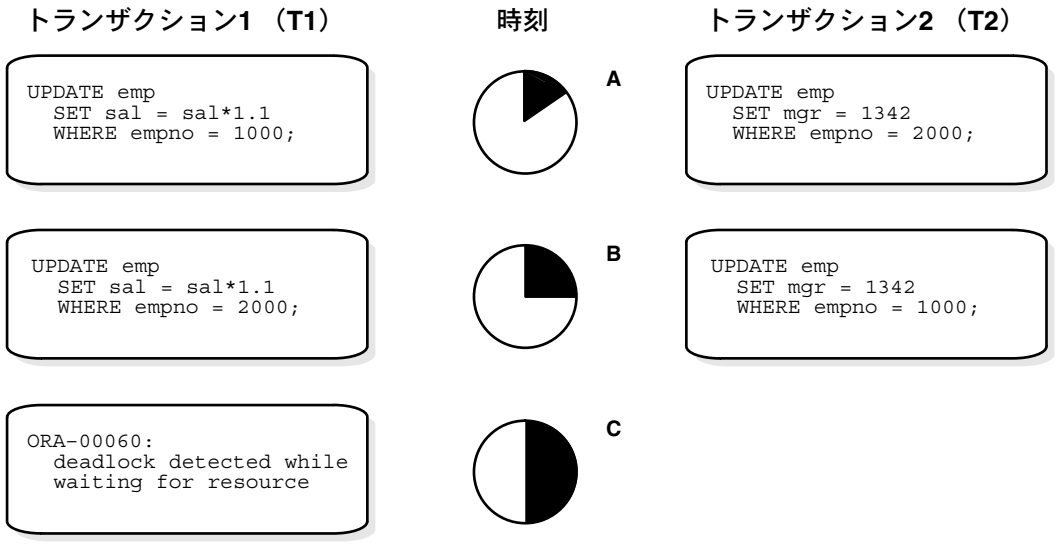
関連項目： 20-24 ページ「表ロック（TM）」

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 20-3 に、デッドロック状態になっている 2 つのトランザクションを示します。

図 20-3 において、各トランザクションはそれ自体が更新しようとする行に対して行ロックを保持するため、時刻 A では問題は存在しません。各トランザクションの処理は、終了せずに進行します。ただし、各トランザクションは、次に、他方のトランザクションが現在保持している行を更新しようとします。したがって、どちらのトランザクションも処理の進行や終了に必要となるリソースを取得できないため、結局、時刻 B でデッドロックが発生します。デッドロックになるのは、それぞれのトランザクションがいくら待機しても、競合するロックが保持されるためです。

図 20-3 デッドロック状態の 2 つのトランザクション



デッドロックの検出

Oracle は、デッドロック状況を自動的に検出し、そのデッドロックに含まれる文の 1 つをロールバックして、競合する一連の行ロックを解放することにより、デッドロックを解決します。また、対応するメッセージが、文レベルのロールバックの対象となったトランザクションに戻されます。ロールバックされる文は、デッドロックが検出されたトランザクションに属しています。通常、通知されたトランザクションは明示的にロールバックする必要がありますが、待機した後でロールバック文を再試行できます。

注意： 分散トランザクションの場合、ローカル・デッドロックは待機グラフを分析することによって検出され、グローバル・デッドロックはタイムアウトによって検出されます。いったん検出されると、非分散デッドロックも分散デッドロックも、データベースとアプリケーションによって同じように処理されます。

デッドロックが最も頻繁に発生するのは、トランザクションが Oracle のデフォルト・ロックを明示的にオーバーライドする場合です。Oracle 自体はロックの段階的拡大を実行せず、また、問合せに読込みロックを使用せず、また行レベルのロック（ページ・レベルのロックではありません）を使用するため、Oracle ではデッドロックはまれにしか起こりません。

> **関連項目：** 手動によるロックの取得の詳細は、20-34 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 は、データへの同時アクセスを制御し、各ユーザー間の有害な干渉を防止するために、様々なタイプのロックを自動的に使用します。Oracle は、トランザクションのためにリソースを自動的にロックし、他のトランザクションが同じリソースの排他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 はロックされるリソースと実行される操作に応じて、様々な制限レベルの様々なタイプのロックを自動的に取得します。

Oracle のロックは次の 3 つのうちいずれか 1 つに分類されます。

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（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。たとえば、Oracle の DML ロックでは、一度に 1 つのトランザクションのみが表の中の特定の行を更新すること、また、コミットされていないトランザクションに表への挿入操作が含まれている場合はその表が削除されないことが保証されます。

DML 操作では、2 つの異なるレベルでデータ・ロックを取得できます。1 つは特定の行に対するロック、もう 1 つは表全体に対するロックです。

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。

行ロック（TX）

Oracle が自動的に取得する DML ロックは、行レベルのロックのみです。1 つの文またはトランザクションで保持できる行ロックの数に制限はありません。また、Oracle が行レベルのロック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、最高の並行性とスループットが得られます。

マルチバージョン並行性制御と行レベル・ロックを組み合せると、同じデータに関して複数のユーザーが競合するのは、同じ行にアクセスする場合のみになります。特に、次のような場合です。

- データの読込みが、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、同じデータ行の読込みを待機しない場合。ただし、読込みのロックを要求する SELECT ... FOR UPDATE を使用していない場合。
- 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、データを読み込むために、同じデータ・ブロックへの書込みの待機が必要になることもあります。

INSERT 文、UPDATE 文、DELETE 文、および FOR UPDATE 句を含む SELECT 文のいずれかで変更された行ごとに、トランザクションは排他 DML ロックを取得します。

変更された行は、ロックを保持しているトランザクションがコミットされるかロールバックされるまで、他のユーザーがその行を変更できないように**常に**排他的にロックされます。ただし、インスタンス障害のためにトランザクションが終了すると、トランザクション全体がリカバリされる前に、ブロック・レベルのリカバリによって行が使用可能になります。前述の文の結果として、行ロックは、常に Oracle によって自動的に取得されます。

ある行について行ロックを取得したトランザクションは、その行に対応する表についての表ロックも取得します。表ロックがあると、カレント・トランザクション内のデータ変更を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： 20-31 ページ「DDL ロック」

表ロック（TM）

INSERT 文、UPDATE 文、DELETE 文、FOR UPDATE 句付きの SELECT 文および LOCK TABLE 文の各 DML 文で表が変更される場合、トランザクションは表ロックを取得します。これらの DML 操作が表ロックを必要とするのは、トランザクションのために表への DML アクセスを確保すること、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 20-3 に、文が取得する表ロックのモードと、各ロックで許可されている操作と禁止されている操作を示します。

表 20-3 表ロックのまとめ

SQL 文	表ロックのモード	許可されるロック・モード				
		RS	RX	S	SRX	X
SELECT...FROM *table*...	なし	可	可	可	可	可
INSERT INTO *table* ...	RX	可	可	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 ...	RS	可 *	可 *	可 *	可 *	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： 20-34 ページ「明示的（手動）データ・ロック」

行共有表ロック（RS）行共有表ロック（**副共有表ロック、SS** とも呼ぶ）は、この表ロックを保持しているトランザクションが、その表の行をロックし、それらの行を更新する予定であることを示します。次の SQL 文のいずれかが実行された場合は、表の行共有表ロックが自動的に取得されます。

```
SELECT ... FROM table ... FOR UPDATE OF ... ;

LOCK TABLE table IN ROW SHARE MODE;
```

行共有表ロックは、最も制限の緩やかなモードの表ロックであり、最高度の並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、表の中の行に対して 1 つ以上の更新を行ったことを示します。行排他表ロックは、次のタイプの文によって修正される表について自動的に取得されます。

```
INSERT INTO table ... ;

UPDATE table ... ;

DELETE FROM table ... ;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

行排他表ロックでは、行共有表ロックよりも少し多くの制限が課されます。

許可される操作 : トランザクションによって保持される行排他表ロックでは、他のトランザクションは、同じ表の中の行に対して問合せ、挿入、更新、削除またはロックを同時に実行できます。そのため、行排他表ロックによって、複数のトランザクションが同時に、同じ表について行排他ロックと行共有表ロックを取得できます。

禁止される操作 : トランザクションによって保持される行排他表ロックは、他のトランザクションが排他的な読込みや書込みを実行するために表を手動でロックすることを防止します。そのため、他のトランザクションは、次の文を使用して同時に表をロックすることはできません。

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

共有表ロック（S）共有表ロックは、次の文で指定される表について自動的に取得されます。

```
LOCK TABLE table IN SHARE MODE;
```

許可される操作 : トランザクションによって保持される共有表ロックでは、他のトランザクションは、表の問合せ、SELECT ... FOR UPDATE 文による特定行のロック、または LOCK TABLE ... IN SHARE MODE 文の実行のみが許可されます。他のトランザクションによる更新は許可されません。複数のトランザクションが、同じ表について同時に共有表ロックを保持できます。ただし、この場合、トランザクションは表を更新できません（FOR UPDATE 句を指定した SELECT 文の結果として行ロックを保持できたとしても更新できません）。そのため、共有表ロックを保持しているトランザクションがその表を更新できるのは、他のトランザクションが同じ表について共有表ロックを保持していない場合のみです。

禁止される操作 : トランザクションによって保持される共有表ロックは、他のトランザクションが同じ表を変更するのを禁止します。また、他のトランザクションが次の文を実行することも禁止します。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

共有行排他表ロック（SRX）共有行排他表ロック（**共有副排他表ロック、SSX** とも呼ぶ）は、共有表ロックよりも多くの制限を課します。共有行排他表ロックは、次の文で指定された表について取得されます。

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

許可される操作 : 一度に 1 つのトランザクションのみが、特定の表の共有行排他表ロックを取得できます。トランザクションによって保持される共有行排他表ロックでは、他のトランザクションは、問合せ、または FOR UPDATE 句を指定した SELECT 文による特定行のロックを許可されますが、表の更新は許可されません。

禁止される操作 : トランザクションによって保持される共有行排他表ロックは、他のトランザクションによる行排他表ロックの取得、および同じ表の変更を防止します。また、共有行排他表ロックでは、他のトランザクションが共有ロック、共有行排他ロックおよび排他表

ロックを取得することが禁止されます。つまり、他のトランザクションが次の文を実行できなくなります。

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

排他表ロック（X）排他表ロックは、最も多くの制限が課されるモードの表ロックであり、ロックを保持するトランザクションが表に排他的に書込みアクセスすることを許可します。排他表ロックは、次の文で指定された表について取得されます。

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

許可される操作 : 1 つのトランザクションのみが、表の排他表ロックを取得できます。排他表ロックでは、他のトランザクションは同じ表に対する問合せのみを実行できます。

禁止される操作 : トランザクションによって保持されている排他表ロックは、他のトランザクションによる DML 文の実行とその表に対するロックの適用を禁止します。

DML 文について自動的に取得される DML ロック

これまで、各種データ・ロックのタイプと、どのモードで保持できるか、いつ取得できるか、いつ取得されるか、何を禁止するかについて説明しました。この後の項では、各種の DML 操作を実行するために Oracle がどのようにデータを自動的にロックするかをまとめます。

表 20-4 に、次の項で説明する情報をまとめておきます。

表 20-4 DML 文が取得するロック

DML 文	行ロック	表ロックのモード
SELECT ... FROM *table*		
INSERT INTO *table* ...	X	RX
UPDATE *table* ...	X	RX
DELETE FROM *table* ...	X	RX
SELECT ... FROM *table* ... FOR UPDATE OF ...	X	RS
LOCK TABLE *table* IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X
	X: 排他 RX: 行排他	RS: 行共有 S: 共有 SRX: 共有行排他

問合せのデフォルト・ロック 問合せはデータを読み込むのみであるため、他の SQL 文に対してほとんど干渉しない SQL 文です。INSERT 文、UPDATE 文および DELETE 文には、文の一部として暗黙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することができます。FOR UPDATE 句が指定されていない問合せでは、データ・ロックが取得されないので他の操作はブロックされないため、Oracle ではこの問合せを**非ブロック化問合せ**と呼ぶことがあります。
- 問合せでは、データ・ロックの解除を待つ必要がなく、常に処理を進めることができます。（待ち状態の分散トランザクションがある場合、ごくまれに、問合せにデータ・ロックの待機が必要な場合があります。）

INSERT、UPDATE、DELETE および SELECT ... FOR UPDATE のデフォルト・ロック INSERT 文、UPDATE 文、DELETE 文および SELECT ... FOR UPDATE 文のロックの特性は次のとおりです。

- DML 文を含むトランザクションは、その文の修正対象の行に対して排他行ロックを取得します。ロックしているトランザクションがコミットまたはロールバックされるまで、その他のトランザクションは、ロックされている行の更新や削除は実行できません。
- DML 文を含むトランザクションは、副問合せや暗黙的問合せ（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 はこのロックを行排他表ロックに自動的に変換します。

DDL ロック

処理中のデータ・ディクショナリ・ロック（DDL）操作によってスキーマ・オブジェクトが影響を受けたり参照される間、そのオブジェクトの定義は DDL ロックによって保護されます。DDL 文がトランザクションを暗黙のうちにコミットすることに注意してください。たとえば、ユーザーがプロシージャを作成する場合を考えます。そのユーザーの単一文トランザクションのために、Oracle はプロシージャ定義で参照されるすべてのスキーマ・オブジェクトについての DDL ロックを自動的に取得します。その DDL ロックにより、プロシージャのコンパイル完了前に、プロシージャで参照されるオブジェクトの変更または削除が防止されます。

Oracle は、ディクショナリ・ロックを必要とする DDL トランザクションのために、ディクショナリ・ロックを自動的に取得します。ユーザーは DDL ロックを明示的に要求できません。DDL 操作中には、修正や参照の対象となる個々のスキーマ・オブジェクトのみがロックされます。データ・ディクショナリ全体がロックされることはありません。

DDL ロックは、排他 DDL ロック、共有 DDL ロックおよびブレーク可能解析ロックの 3 つに分類されます。

排他 DDL ロック

次の項の共有 DDL ロックに示されている DDL 操作を除き、ほとんどの DDL 操作では、同じスキーマ・オブジェクトの変更や参照を実行する可能性のある他の DDL 操作によって破壊的な干渉が起きないように、リソースの排他 DDL ロックが必要です。たとえば `ALTER TABLE` 操作で表に列を追加している間は、`DROP TABLE` 操作でその表を削除できません。その逆も同様です。

排他 DDL ロックの取得では、別の操作によってすでにそのスキーマ・オブジェクトに対して別の DDL ロックが保持されている場合、その取得は古い DDL ロックが解除されるまで待機し、その後に処理されます。

また、DDL 操作は変更対象のスキーマ・オブジェクトに対する DML ロック（データ・ロック）も取得します。

共有 DDL ロック

ある種の DDL 操作では、競合する DDL 操作によって破壊的な干渉が起きないようにし、かつ一方では類似の DDL 操作についてデータの並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 [OR REPLACE] VIEW/ PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/TRIGGER、CREATE SYNONYM、CREATE TABLE（CLUSTER パラメータが含まれていない場合）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 15 章「スキーマ・オブジェクト間の依存性」

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 または SQL*Plus 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。

ラッチ

ラッチは、システム・グローバル領域（SGA）内の共有データ構造を保護するための単純な下位レベルのシリアル化メカニズムです。たとえば、ラッチは現在データベースにアクセスしているユーザーのリストと、バッファ・キャッシュ内のブロックを説明しているデータ構造を保護します。サーバー・プロセスまたはバックグラウンド・プロセスは、これらの構造の 1 つを操作したり参照する、非常に短い間のみラッチを取得します。ラッチのインプリメンテーション（特に、プロセスがラッチを待機するかどうか、およびどれくらいの時間待機するか）は、オペレーティング・システムに依存します。

内部ロック

内部ロックは、ラッチよりも高水準で複雑なメカニズムであり、いろいろな目的のために機能します。

ディクショナリ・キャッシュ・ロック これらのロックの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することが必要です。ロールバック・セグメントは、1 つのセグメントに 1 つのインスタンスしか書き込めないようにロックされています。

明示的（手動）データ・ロック

データの並行性、整合性および文レベルの読込み一貫性を確保するために、Oracle は常に自動的にロックを実行します。ただし、デフォルトのロック・メカニズムを置き換えることもできます。デフォルト・ロックの置換えは、次のような状況で有効です。

- アプリケーションで、トランザクション・レベルの読込み一貫性または**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・レベルの読込み一貫性は、明示的ロック、読取り専用トランザクションまたはシリアル化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 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 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 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、データ並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、『Oracle9*i* SQL リファレンス』を参照してください。

明示的なロックの下でのデータ並行性の例

LOCK TABLE 文や FOR UPDATE 句を指定した SELECT 文が使用されるときに、Oracle がデータの並行性、整合性および一貫性をどのように維持するかを次に示します。

> **注意：** 簡潔にするため、ORA-00054 のメッセージ・テキストは記載しません。そのメッセージ・テキストは「リソース・ビジー、NOWAIT が指定されていました。」というものです。ユーザーが入力するテキストは**太字**になっています。

トランザクション 1	時点	トランザクション 2
LOCK TABLE scott.dept ** IN ROW SHARE MODE;** Statement processed	1	
	2	**DROP TABLE scott.dept;** DROP TABLE scott.dept * * ORA-00054 *(exclusive DDL lock not possible because of T1's table lock)*
	3	**LOCK TABLE scott.dept** ** IN EXCLUSIVE MODE NOWAIT;** ORA-00054
	4	**SELECT LOC** ** FROM scott.dept** ** WHERE deptno = 20** ** FOR UPDATE OF loc;** LOC - - - - - - - DALLAS 1 row selected
UPDATE scott.dept ** SET loc = 'NEW YORK'** ** WHERE deptno = 20;** *(waits because T2 has locked same rows)*	5	
	6	**ROLLBACK;** *(releases row locks)*
1 row processed. **ROLLBACK;**	7	

トランザクション 1	時点	トランザクション 2
LOCK TABLE scott.dept ** IN ROW EXCLUSIVE MODE;** Statement processed.	8	
	9	**LOCK TABLE scott.dept** ** IN EXCLUSIVE MODE** ** NOWAIT;** ORA-00054
	10	**LOCK TABLE scott.dept** ** IN SHARE ROW EXCLUSIVE** ** MODE NOWAIT;** ORA-00054
	11	**LOCK TABLE scott.dept** ** IN SHARE ROW EXCLUSIVE** ** MODE NOWAIT;** ORA-00054
	12	**UPDATE scott.dept** ** SET loc = 'NEW YORK'** ** WHERE deptno = 20;** 1 row processed.
	13	**ROLLBACK;**
SELECT loc ** FROM scott.dept** ** WHERE deptno = 20** ** FOR UPDATE OF loc;** LOC - - - - - - DALLAS 1 row selected.	14	
	15	**UPDATE scott.dept** ** SET loc = 'NEW YORK'** ** WHERE deptno = 20;** *(waits because T1 has locked same rows)*
ROLLBACK;	16	
	17	1 row processed. *(conflicting locks were released)* **ROLLBACK;**

トランザクション1	時点	トランザクション2
LOCK TABLE scott.dept ** IN SHARE MODE** Statement processed	18	
	19	**LOCK TABLE scott.dept** ** IN EXCLUSIVE MODE NOWAIT;** ORA-00054
	20	**LOCK TABLE scott.dept** ** IN SHARE ROW EXCLUSIVE** ** MODE NOWAIT;** ORA-00054
	21	**LOCK TABLE scott.dept** ** IN SHARE MODE;** Statement processed.
	22	**SELECT loc** ** FROM scott.dept** ** WHERE deptno = 20;** LOC - - - - - - DALLAS 1 row selected.
	23	**SELECT loc** ** FROM scott.dept** ** WHERE deptno = 20** ** FOR UPDATE OF loc;** LOC - - - - - - DALLAS 1 row selected.
	24	**UPDATE scott.dept** ** SET loc = 'NEW YORK'** ** WHERE deptno = 20;** *(waits because T1 holds conflicting table lock)*
ROLLBACK;	25	
	26	1 row processed. *(conflicting table lock released)* **ROLLBACK;**

トランザクション 1	時点	トランザクション 2
LOCK TABLE scott.dept ** IN SHARE ROW** ** EXCLUSIVE MODE;** Statement processed.	27	
	28	**LOCK TABLE scott.dept** ** IN EXCLUSIVE MODE** ** NOWAIT;** ORA-00054
	29	**LOCK TABLE scott.dept** ** IN SHARE ROW** ** EXCLUSIVE MODE** ** NOWAIT;** ORA-00054
	30	**LOCK TABLE scott.dept** ** IN SHARE MODE NOWAIT;** ORA-00054
	31	**LOCK TABLE scott.dept** ** IN ROW EXCLUSIVE** ** MODE NOWAIT;** ORA-00054
	32	**LOCK TABLE scott.dept** ** IN SHARE MODE NOWAIT;** ORA-00054
	33	**SELECT loc** ** FROM scott.dept** ** WHERE deptno = 20;** LOC - - - - - - DALLAS 1 row selected.
	34	**SELECT loc** ** FROM scott.dept** ** WHERE deptno = 20** ** FOR UPDATE OF loc;** LOC - - - - - - DALLAS 1 row selected.

トランザクション 1	時点	トランザクション 2
	35	**UPDATE scott.dept** **SET loc = 'NEW YORK'** **WHERE deptno = 20;** *(waits because T1 holds conflicting table lock)*
UPDATE scott.dept **SET loc = 'NEW YORK'** **WHERE deptno = 20;** *(waits because T2 has locked same rows)*	36	*(deadlock)*
Cancel operation **ROLLBACK;**	37	
	38	1 row processed.
LOCK TABLE scott.dept **IN EXCLUSIVE MODE;**	39	
	40	**LOCK TABLE scott.dept** **IN EXCLUSIVE MODE;** ORA-00054
	41	**LOCK TABLE scott.dept** **IN ROW EXCLUSIVE MODE** **NOWAIT;** ORA-00054
	42	**LOCK TABLE scott.dept** **IN SHARE MODE;** ORA-00054
	43	**LOCK TABLE scott.dept** **IN ROW EXCLUSIVE** **MODE NOWAIT;** ORA-00054
	44	**LOCK TABLE scott.dept** **IN ROW SHARE MODE** **NOWAIT;** ORA-00054

トランザクション 1	時点	トランザクション 2
	45	**SELECT loc** **FROM scott.dept** **WHERE deptno = 20;** LOC - - - - - - DALLAS 1 row selected.
	46	**SELECT loc** **FROM scott.dept** **WHERE deptno = 20** **FOR UPDATE OF loc;** *(waits because T1 has conflicting table lock)*
UPDATE scott.dept **SET deptno = 30** **WHERE deptno = 20;** 1 row processed.	47	
COMMIT;	48	
	49	0 rows selected. *(T1 released conflicting lock)*
SET TRANSACTION READ ONLY;	50	
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - BOSTON	51	
	52	**UPDATE scott.dept** **SET loc = 'NEW YORK'** **WHERE deptno = 10;** 1 row processed.
SELECT loc **FROM scott.dept** **WHERE deptno = 10;** LOC - - - - - - BOSTON *(T1 does not see uncommitted data)*	53	
	54	**COMMIT;**

トランザクション 1	時点	トランザクション 2
SELECT loc ** FROM scott.dept** ** WHERE deptno = 10;** LOC - - - - - - *(same results seen even after T2 commits)*	55	
COMMIT;	56	
SELECT loc ** FROM scott.dept** ** WHERE deptno = 10;** LOC - - - - - - NEW YORK *(committed data is seen)*	57	

Oracle のロック・マネージメント・サービス

アプリケーションの開発者は、Oracle のロック・マネージメント・サービスを使用して次のような処理をする文を PL/SQL ブロックに含めることができます。

- 特定のタイプのロックを要求します。
- そのロックに、同一のインスタンスまたは別のインスタンス内にある別のプロシージャからも識別できる、一意の名前を指定します。
- ロック・タイプを変更します。
- ロックを解除します。

確保したユーザー・ロックは、Oracle ロックと同一とみなされるため、デッドロックの検出などの Oracle ロックの機能をすべて備えています。ユーザー・ロックは接頭辞 UL で識別されるため、Oracle ロックと矛盾することはありません。

Oracle のロック・マネージメント・サービスは、DBMS_LOCK パッケージ内のプロシージャを介して利用できます。

関連項目：

- Oracle ロック・マネージメント・サービスの詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- DBMS_LOCK の詳細は、『Oracle9*i* 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。

フラッシュバック問合せ

フラッシュバック問合せを使用すると、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 のマルチバージョン読込み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。管理者は、UNDO をデータベースに保存する期間を指定するだけで、UNDO 保存を構成できます。フラッシュバック問合せを使用すると、ユーザーは今朝、昨日または先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。

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：** AS OF 句の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

フラッシュバック問合せの利点

- 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は UNDO は行いません。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、変更の取消しに使用でき、非常に効率的です。理論的には、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。
- 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。これはアプリケーションによっては処理が難しくなります。

フラッシュバック問合せの用途

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。

E 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、メールを削除した経験がある場合は、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：

- フラッシュバック問合せの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- `DBMS_FLASHBACK` パッケージの説明は、『Oracle9*i* 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については、『Oracle9*i* データベース管理者ガイド』を参照してください。

21

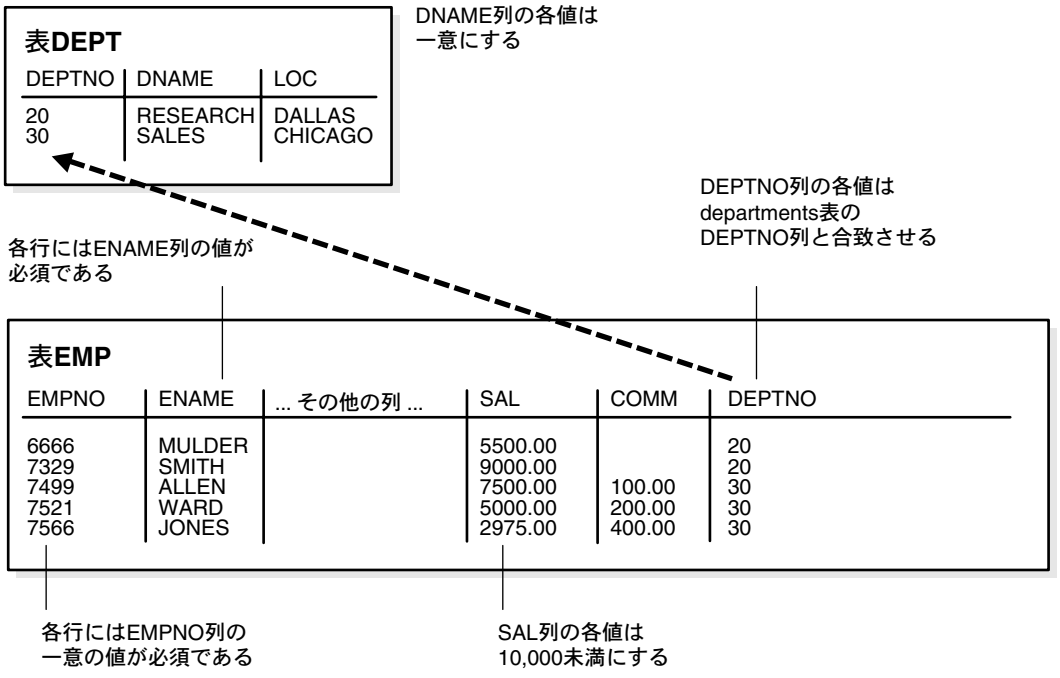
データ整合性

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

データが、データベース管理者やアプリケーションの開発者によって事前に定義された規則に準拠しているのは重要なことです。データ整合性の例として、図 21-1 に示す employees 表と departments 表、およびこれらの各表の情報に関するビジネス・ルールについて考えてみましょう。

図 21-1 データ整合性の例



それぞれの表のある列には、その列に入っているデータに制約をかける固有の規則があることに注意してください。

データ整合性のタイプ

この項では、様々な種類のデータ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 は、単一の列に対して定義される規則で、その列に NULL が入っている（値がない）行の挿入や更新を許可または禁止します。

一意の列値

列（または列の集合）に対して定義される一意の列値は、その列（または列の集合）に含まれる値が一意である場合に限って、行の挿入または更新を許可します。

主キー値

キー（列または列の集合）に対して定義される主キー値は、表の中の各行がキーの値によって一意に識別できることを指定します。

参照整合性規則

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 はデフォルト・アクションとして NO ACTION を使用します。）

複雑な整合性チェック

複雑な整合性チェックは、列（または列の集合）に対して設定されるユーザー定義の規則であり、その列の値に基づいて、行の挿入、更新、削除を許可または禁止します。

Oracle がデータ整合性を規定する方法

Oracle では、前述のデータ整合性規則をそれぞれ定義し、規定できます。これらの規則のほとんどは、整合性制約またはデータベース・トリガーを使用して簡単に定義できます。

整合性制約の説明

整合性制約は、表の列に規則を定義する宣言手法です。Oracle では、次の整合性制約をサポートしています。

- NOT NULL 制約。列に含まれる NULL に関連した規則です。
- 一意キー制約。一意の列値に関連した規則です。
- 主キー制約。プライマリ識別値に関連した規則です。
- 外部キー制約。参照整合性に対応付けられた規則です。Oracle では、次の参照整合性アクションを定義する際に外部キー制約の使用をサポートしています。
  - UPDATE NO ACTION と DELETE NO ACTION
  - DELETE CASCADE
  - DELETE SET NULL
- CHECK 制約。複雑な整合性規則の場合に使用します。

注意： 子表と親表が分散データベースの別のノードにある場合、宣言整合性制約を使用して参照整合性の規定はできません。ただし、データベース・トリガーを使用して、分散データベースで参照整合性を規定することはできます（後述）。

データベース・トリガー

Oracle では、データベース・トリガー（挿入、更新または削除の各操作で自動的に起動されるストアド・データベース・プロシージャ）を使用することによって、非宣言アプローチの整合性規則を規定できます。

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 17 章「トリガー」を参照してください。

整合性制約の概要

Oracle では、データベースの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 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` 列に整合性制約を定義するとします。この表では、どの行でもこの列に 10,000 より大きい数値を入れないという規則を、整合性制約として規定します。`INSERT` 文または `UPDATE` 文がこの整合性制約に違反すると、Oracle はその文をロールバックし、通知エラー・メッセージを戻します。

Oracle で実装されている整合性制約は、ANSI X3.135-1989 と ISO 9075-1989 の標準規格に完全に準拠しています。

整合性制約の利点

この項では、整合性制約が他の代替方法よりも優れている点をいくつか説明します。代替方法には次のようなものがあります。

- データベース・アプリケーションのコードでビジネス・ルールを規定します。
- ストアド・プロシージャを使用してデータへのアクセスを完全に制御します。
- トリガーされるストアド・データベース・プロシージャを使用してビジネス・ルールを規定します。

関連項目： 第 17 章「トリガー」

宣言の容易さ

整合性制約は、SQL 文を使用して定義します。表を定義したり変更したりするときに追加のプログラミングをする必要はありません。SQL 文は記述が容易で、プログラミングのエラーを避けられます。Oracle は SQL 文の機能を制御します。これらの理由で、宣言整合性制約は、アプリケーション・コードやデータベース・トリガーよりも優れています。また、ストアド・プロシージャを使用してデータ・アクセスを制御することによってデータ整合性を解決するという方法もありますが、整合性制約の場合は非定型のデータ・アクセスという柔軟性を犠牲にすることがないため、宣言アプローチを使用するほうがストアド・プロシージャより優れています。

規則の集中化

整合性制約は、表に対して（アプリケーションに対してではなく）定義され、データ・ディクショナリに格納されます。どのアプリケーションから入力されるデータも、表に対応付けられている同じ整合性制約を遵守する必要があります。ビジネス・ルールをアプリケーション・コードから集中管理された整合性制約に移行することにより、どのデータベース・アプリケーションが情報を操作したとしても、データベース表には有効なデータが入っていることが保証されます。ストアド・プロシージャには、集中管理された規則を表に格納する場合のような利点がありません。また、データベース・トリガーでは、規則を集中管理できるという利点はありますが、それを実現する方法は、整合性制約で使用されている宣言アプローチよりはるかに複雑です。

アプリケーション開発の生産性の最大化

整合性制約によって規定されるビジネス・ルールを変更する場合、管理者が整合性制約を変更するだけで、すべてのアプリケーションは変更後の制約を自動的に遵守するようになります。それに対して、それぞれのデータベース・アプリケーションのコードでビジネス・ルールを規定する場合、開発者はすべてのアプリケーションのソース・コードを修正して、その修正したアプリケーションを再コンパイルし、デバッグしてテストする必要があります。

ユーザーへの即時フィードバック

Oracle は、各整合性制約ごとに、特定の情報をデータ・ディクショナリに格納します。Oracle が SQL 文を実行しチェックする前でも、データベース・アプリケーションは、その情報を使用して整合性制約の違反をユーザーに即時にフィードバックするように、データベース・アプリケーションを設計できます。たとえば SQL*Forms アプリケーションは、文を発行する前でも、データ・ディクショナリに格納されている整合性制約の定義を使用して、フォームのフィールドに入力される値の違反をチェックできます。

優れたパフォーマンス

整合性制約宣言の意味は明確に定義されており、個々の宣言規則ごとにパフォーマンスの最適化が実現されます。Oracle 問合せオプティマイザは、宣言を使用して、データをより詳細に認識し、問合せのパフォーマンスを全体として向上させます。（また、アプリケーション・コードとデータベース・トリガーから整合性規則を取り去ることによって、必要なときのみチェックが行われることが保証されます。）

データ・ロードの柔軟性と整合性違反の識別

大量のデータをロードする場合、制約チェックによるオーバーヘッドをなくすために、整合性制約を一時的に使用禁止にできます。データのロードが完了した後、整合性制約を使用可能にするのは簡単であり、整合性制約に違反した新しい行を別の例外表に自動的に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

列値の入力時の制限として課すことのできる整合性制約には、次のものがあります。

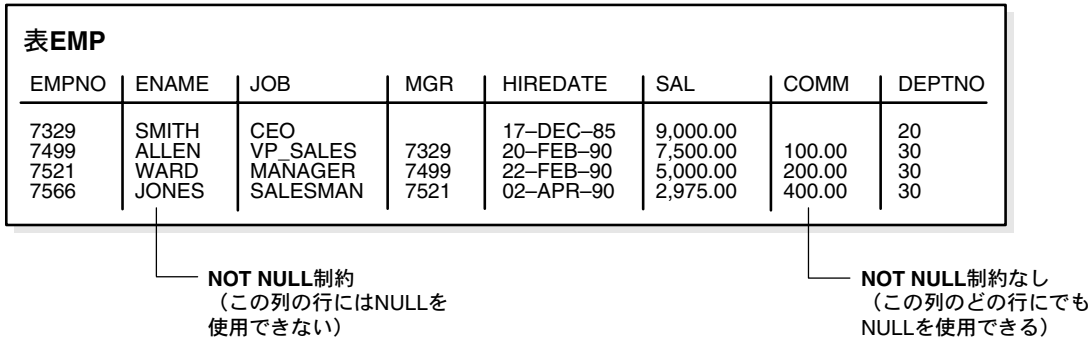
- NOT NULL 制約
- 一意キー制約
- 主キー制約
- 参照整合性制約
- CHECK 制約

NOT NULL 制約

デフォルトでは、表のすべての列で NULL を使用できます。**NULL** は、値がないことを意味します。NOT NULL 制約がある場合、表の列値が NULL でないということが求められます。たとえば、`employees` 表のすべての行で `last_name` 列に必ず値を入力するように求める NOT NULL 制約を定義できます。

図 21-2 に、NOT NULL 制約を示します。

図 21-2 NOT NULL 制約

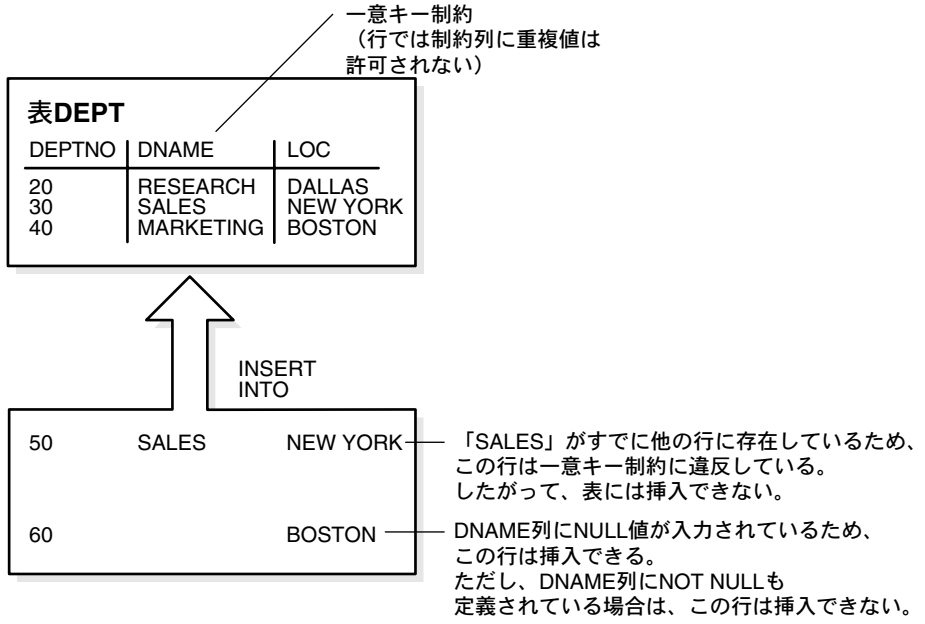


一意キー制約

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。

たとえば、図 21-3 では、departments 表の DNAME 列に一意キー制約が定義されており、複数の行での部門名の重複を禁止しています。

図 21-3 一意キー制約

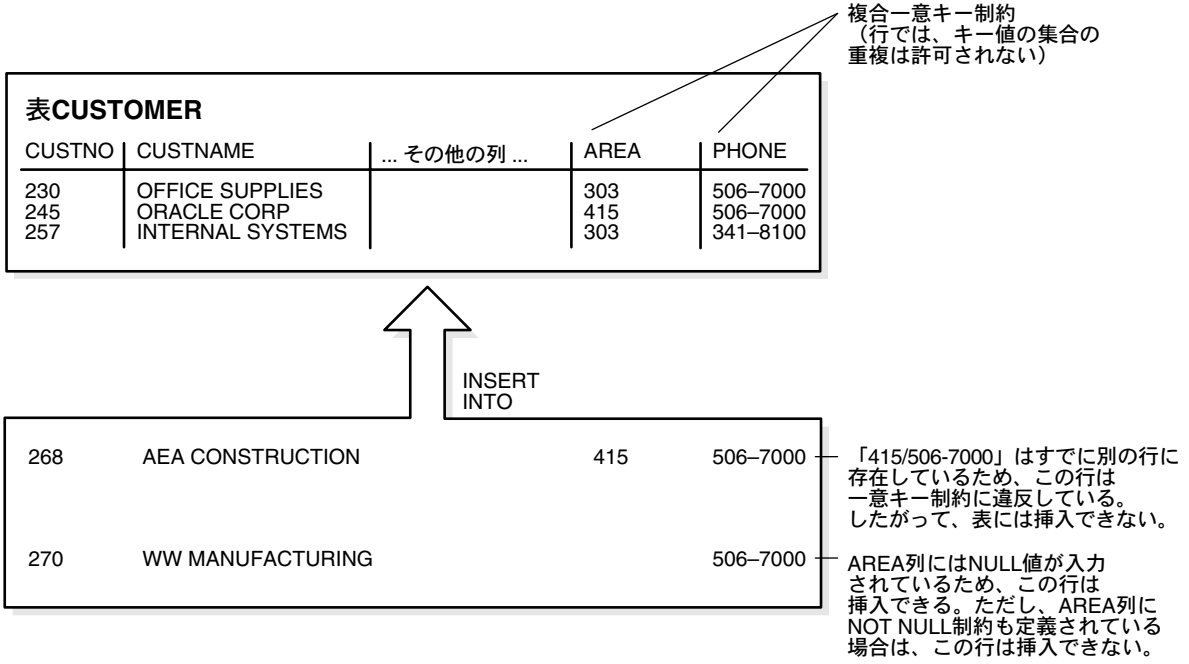


一意キー

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**という用語は、誤って**一意キー制約**や**一意索引**のシノニムとして使用されることがあります。ただし、**キー**という用語は整合性制約の定義に使用されている列または列の集合のみを指して使用されるため、注意してください。

一意キーが 2 つ以上の列で構成されている場合、列のそのグループのことを**複合一意キー**と呼びます。たとえば、図 21-4 の例では、customer 表の複合一意キー（area 列と phone 列）に対して一意キー制約が定義されています。

図 21-4 複合一意キー制約



この場合の一意キー制約では、同じ市外局番と電話番号を何回でも入力できますが、市外局番と電話番号の特定の組合せを同じ表の中に重複しては入力できません。これにより、誤って電話番号が重複するのを避けられます。

一意キー制約と索引

Oracle は、索引を使用して一意の整合性制約を規定します。たとえば、図 21-4 で、Oracle は、複合一意キーに対する一意索引を暗黙的に作成することにより、一意キー制約を規定しています。したがって、複合一意キー制約には、コンポジット索引に対して課されているのと同じ制限があります。複合一意キーを構成する最大列数は 32 個です。

注意： 互換性が Oracle9*i* 以上に設定されている場合、キー値のバイトで示される合計サイズはブロック・サイズに近くなります。以前のリリースでは、キーのサイズは対応するデータベースのブロック・サイズのおよそ半分を超えることはできませんでした。

一意キー制約が作成されたときに使用できる索引がある場合、制約は新しい索引を暗黙的に作成するかわりにその索引を使用します。

一意キー制約と NOT NULL 制約との組合せ

図 21-3 と図 21-4 では、同じ列に NOT NULL 制約も定義しないかぎり、一意キー制約は NULL の入力を許可します。実際、NULL はどの値とも等しいとみなされないため、NOT NULL 制約のない列では、その列が NULL である行が何行存在してもかまいません。1 つの列に NULL がある（または複合一意キーのすべての列に NULL がある）場合は、一意キー制約はいつも満たされます。

一意キーと NOT NULL 制約の両方が指定された列が、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 の複合一意キー制約の非 NULL 列で同一の値は許されません。

主キー制約

データベースのそれぞれの表には、最大 1 つの主キー制約を指定できます。この制約が指定されている 1 つ以上の列グループの値は、その行の一意識別子を構成します。つまり、それぞれの行は、その主キー値によって指定されます。

Oracle に実装されている主キー制約では、次の 2 つのことが保証されます。

- 表の指定された列または列の集合の中に、2 つの行が重複する値を持つことはありません。
- 主キー列では、NULL は許可されません。つまり、それぞれの行の主キー列には値が必ず存在します。

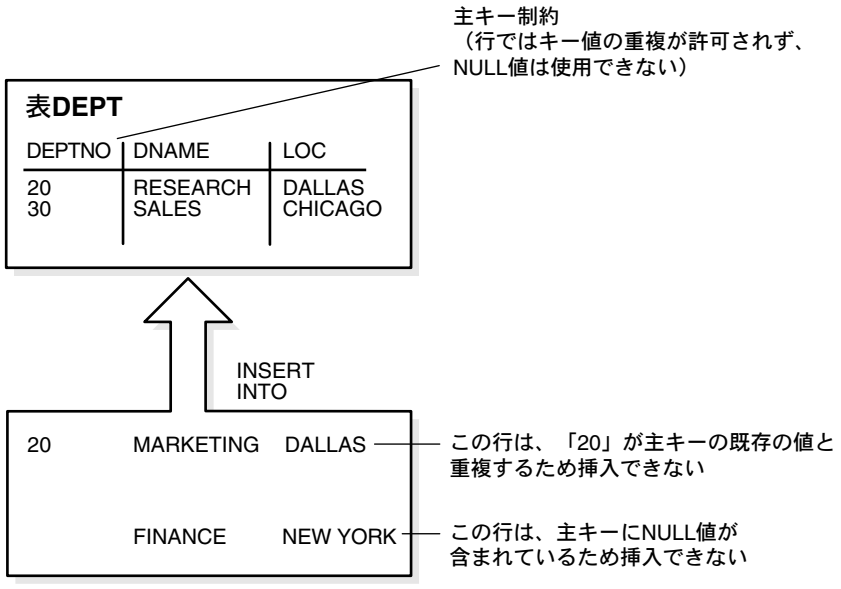
主キー

表の主キー制約の定義に含まれている列を、主キーと呼びます。主キーは必須ではありませんが、次の利点を考慮して、すべての表に主キーを指定してください。

- 表のそれぞれの行を一意に識別できます。
- 重複する行が表に存在しません。

図 21-5 に、`departments` 表での主キー制約と、制約に違反する行の例を示します。

図 21-5 主キー制約



主キー制約と索引

Oracle では、すべての主キー制約が索引を使用して規定されます。図 21-5 では、`department_id` 列に対して作成された主キー制約は、次の暗黙的な作成によって規定されます。

- その列に対して一意の索引
- その列に対する NOT NULL 制約

Oracle は、索引を使用して主キー制約を規定し、コンポジット主キー制約は、コンポジット索引に課されるのと同じ 32 以下の列という制限を受けます。この索引の名前は、制約の名前と同じ名前です。また、制約を作成に使用する `CREATE TABLE` 文または `ALTER TABLE` 文に `ENABLE` 句を組み込んで、この索引に記憶領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。

参照整合性制約に関連する用語は、次のとおりです。

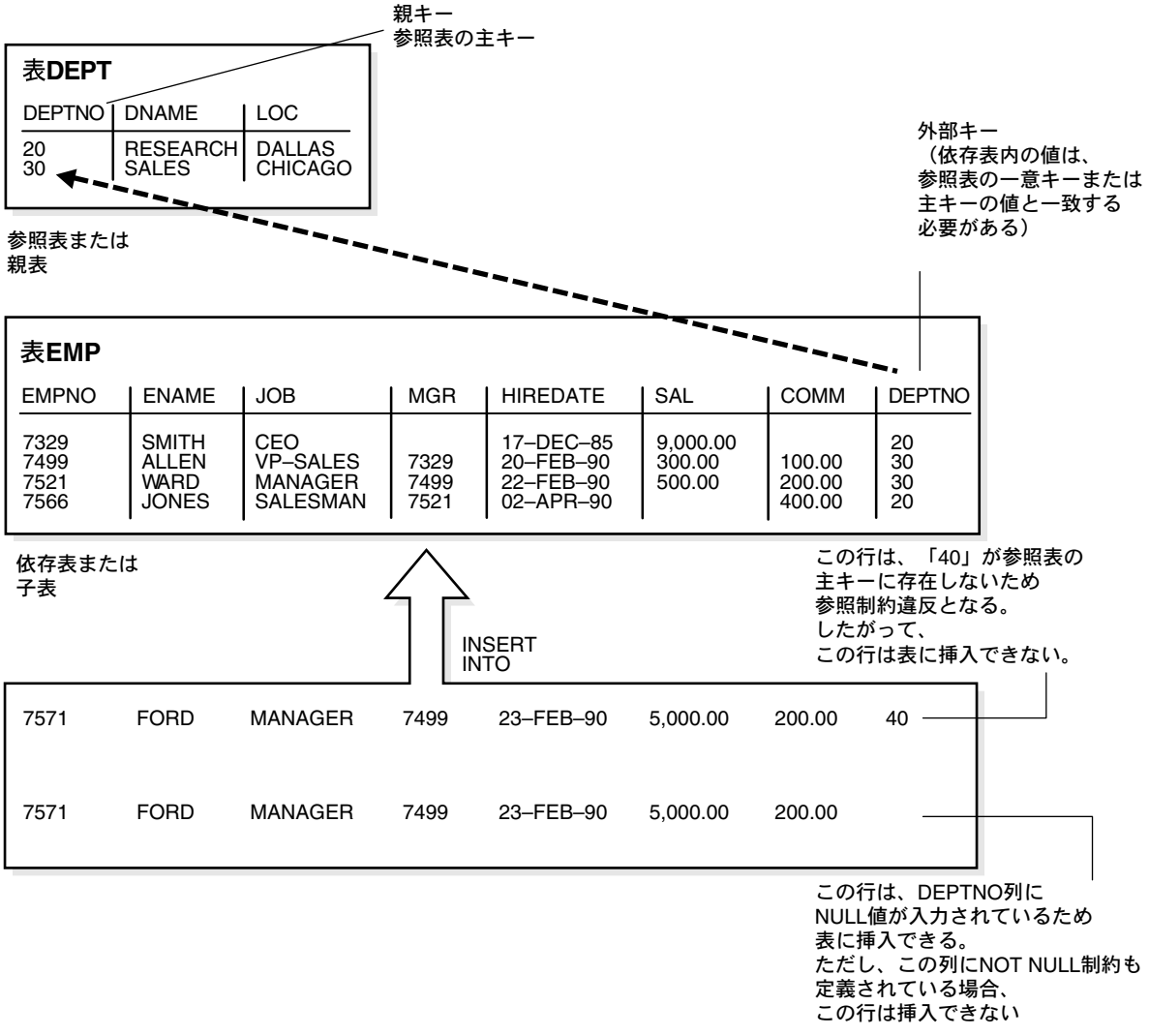
用語	定義
外部キー	参照整合性制約の定義に含まれている列または列の集合のうち、参照キーを参照するもの。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。

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 21-6 に、employees 表の department_id 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 departments 表の主キー（すなわち department_id 列）の値と一致することが保証されます。このため、employees 表の department_id 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 21-6 参照整合性制約

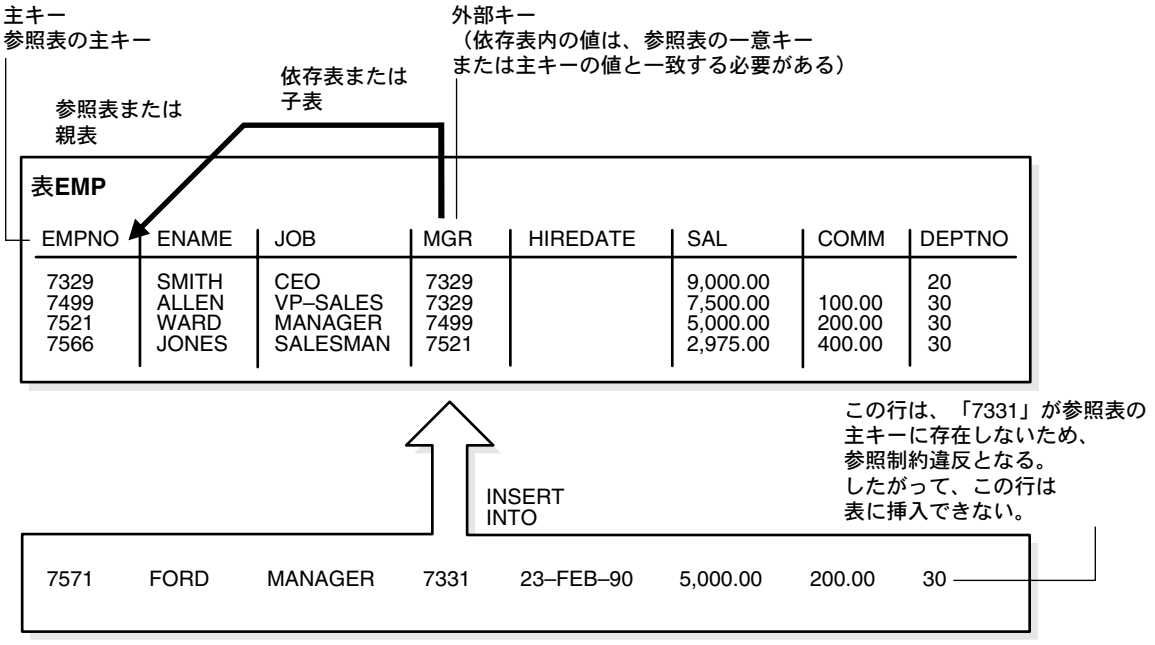


自己参照型整合性制約

図 21-7 に示す別のタイプの参照整合性制約のことを、自己参照型整合性制約と呼びます。このタイプの外部キーは、同じ表の親キーを参照します。

図 21-7 では、参照整合性制約によって、employees 表の manager_id 列のすべての値は、必ずしも同じ行でなくても、同じ表の employee_id 列に現在存在する値と一致することが保証されます。これは、すべての管理職は従業員でもある必要があるためです。この整合性制約により、間違った従業員番号が manager_id 列に存在する可能性はなくなります。

図 21-7 単一表の参照制約



NULL と外部キー

リレーショナル・モデルでは、外部キーの値が、参照主キーか一意キーの値、または NULL のどちらかに一致することが許されています。コンポジット外部キーのいずれかの列が NULL の場合、そのキーの NULL 以外の部分は、親キーの対応部分と一致している必要はありません。

参照整合性制約によって定義されるアクション

参照整合性制約では、参照先の親キー値が修正された場合に子表の依存行に対して実行される特定のアクションを指定できます。Oracle の外部キー制約によってサポートされる参照アクションは、UPDATE NO ACTION、DELETE NO ACTION および DELETE CASCADE です。

注意： Oracle の外部キー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 17 章「トリガー」を参照してください。

UPDATE NO ACTION と DELETE NO ACTION NO ACTION（デフォルト）オプションは、結果のデータが参照整合性制約に違反する場合に参照キー値を更新または削除できないことを指定します。たとえば、主キー値が外部キーの中の値によって参照されている場合は、依存データであるため、参照先の主キー値を削除できません。

DELETE CASCADE 参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行も削除されます。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 21-1 に、親表の主キー値または一意キー値および子表の外部キー値に対する異なる参照アクションごとに可能な DML 文の概要を示します。

表 21-1 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。

表 21-1 UPDATE NO ACTION と DELETE NO ACTION で許可される DML 文（続き）

DML 文	親表に対して発行	子表に対して発行
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

外部キーには、ほぼ常に索引を付けます。唯一の例外は、対応する一意キーまたは主キーの更新や削除が発生しないことが確実な場合です。

Oracle は、親キーと依存している外部キー値に関して、並行性制御を最適化します。両者の関連性の維持に使用される並行性のためのメカニズムを制御できるので、状況によっては、これが大きな利点をもたらすことがあります。ここでは、考えられる状況と個々の推奨事項について説明します。

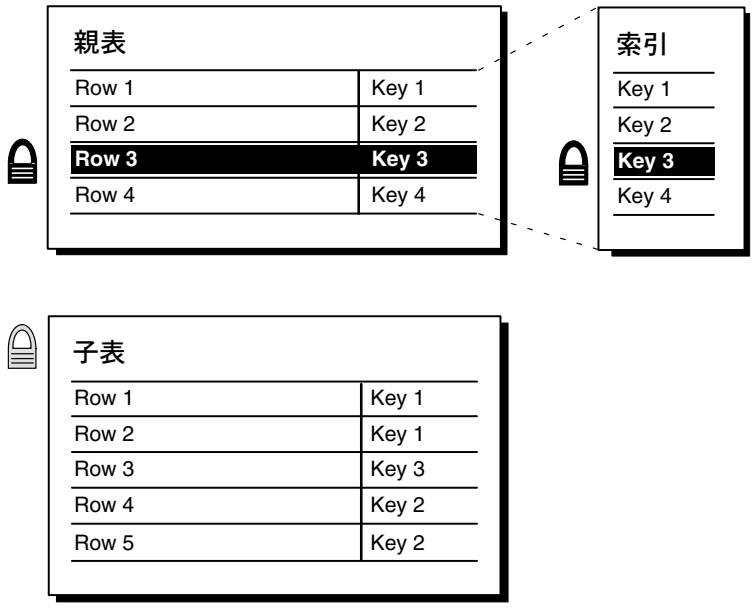
外部キーの索引がない場合 図 21-8 は、外部キーの索引が定義されていない場合と、親表の行が更新または削除される場合に、Oracle で使用されるロック・メカニズムを示しています。親表に対する挿入の場合、子表のロックは必要ありません。

Oracle では、主キーの更新または削除を行う際、索引なしの外部キーに共有ロックは必要ありません。表レベルの共有ロックは従来どおり取得されますが、すぐに解放されます。複数の主キーを更新または削除する場合、ロックは各行につき一度ずつ取得され解放されます。

以前のリリースでは、親表に対する DELETE 文を含むトランザクションがコミットされるまでは、子表全体の共有ロックが必要でした。外部キーで ON DELETE CASCADE が指定された場合、DELETE 文では子表に表レベルの共有副排他ロックが発生していました。また、子表で参照される列に影響する親表に対して UPDATE 文を発行する場合も、子表全体の共有ロックが必要でした。共有ロックでは読込みしかできません。そのため、UPDATE または DELETE を含むトランザクションがコミットされるまでは、子表に対して INSERT、UPDATE または DELETE 文は発行できませんでした。子表に対する問合せは可能でした。

子表に対する INSERT、UPDATE および DELETE 文は親表のロックを取得しませんが、INSERT 文と UPDATE 文は親表の索引の行ロックが解放されるまで待機します。

図 21-8 外部キーに索引が定義されていない場合のロック・メカニズム

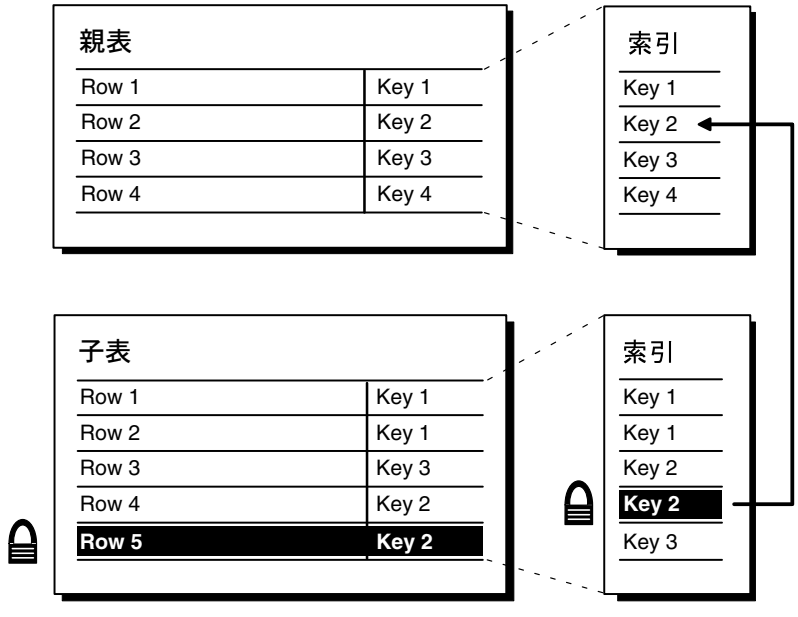


外部キーの索引 図 21-9 は、外部キーの索引が定義されており、子表に新規行が挿入されるか、更新または削除される場合に、Oracle で使用されるロック・メカニズムを示しています。

挿入、更新または削除が発生しても、親表またはその索引については、どんな種類の表ロックも取得されないことに注目してください。したがって、親表には、INSERT、UPDATE、DELETE および問合せなど、任意の DML 文を発行できます。

この状況が望ましいのは、子表に対する更新処理の発生中に、親表に対する更新または削除処理が発生する場合です。親表に対する INSERT、UPDATE および DELETE には子表のロックは必要ありませんが、UPDATE および DELETE は、子表の索引に対する行レベルのロックが解放されるまで待機します。

図 21-9 外部キーに索引が定義されている場合のロック・メカニズム



子表で ON DELETE CASCADE が指定されている場合、親表から削除すると子表からも削除されることがあります。この場合、待機とロックに関するルールは、親表からの DELETE を実行した後に子表から自身を削除した場合と同じです。

CHECK 制約

列または列の集合に対する CHECK 制約では、その表のすべての行について、指定した条件が TRUE または UNKNOWN であることが必要です。DML 文の結果で CHECK 制約の条件が FALSE に評価される場合、その文はロールバックされます。

チェック条件

CHECK 制約を使用すると、チェック条件を指定することによって、特定の目的の整合性規則を規定できます。CHECK 制約の条件には次のような制限があります。

- 挿入または更新する行の値を使用して評価されるブール式であることが必要です。
- 副問合せ、順序、SQL ファンクション 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 ファンクション（TO_CHAR、TO_DATE および TO_NUMBER など）が組み込まれている CHECK 制約を評価する際、デフォルトでは Oracle はデータベースの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、CHECK 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すれば、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、『Oracle9*i* 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 CHECK 制約

単一の列に、定義の中でその列を参照する複数の CHECK 制約を指定できます。1 つの列に対して定義できる CHECK 制約の数に制限はありません。

単一の列に対して複数の CHECK 制約を作成する場合は、その目的が競合しないように慎重に設計してください。また、条件が特定の順序で評価されるとは考えないでください。CHECK の条件が矛盾しないかどうかは検証されません。

制約チェックのメカニズム

制約が存在する場合に許可されるアクションのタイプを把握する上で、Oracle が実際に制約をチェックするタイミングを理解しておくと役立ちます。このことを具体的に説明するため、いくつかの例を紹介します。この例では、次のような状況を想定します。

- employees 表は、21-14 ページの図 21-7 で定義されたものです。
- 自己参照型制約によって、manager_id 列のエントリが employee_id 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、employees 表の employee_id 列と manager_id 列のみを対象にします。

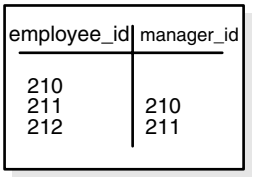
employees 表に最初の行を挿入する場合を考えます。現在は行が存在しません。manager_id 列が employee_id 列の既存の値を参照できない場合に行を入力する方法について考えてみます。この操作を実行するには、次の 3 つの方法が考えられます。

- manager_id 列に NOT NULL 制約が定義されていない場合には、第 1 行の manager_id 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。
- employee_id 列と manager_id 列の両方に同じ値を入力できます。このことから、Oracle が文の実行完了後に制約チェックを実行することがわかります。親キーと外部キーに同じ値を指定した行の入力を許可するために、Oracle は、文を実行（つまり、新しい行を挿入）してから、その新しい行の manager_id 列に対応する employee_id が入っている行がその表に含まれているかどうかをチェックします。
- SELECT 文のネストを伴う INSERT 文など、複数行を挿入する INSERT 文により、相互に参照しあう行を挿入できます。たとえば、最初の行は employee_id 列が 200 で manager_id 列が 300、2 番目の行は employee_id 列が 300 で manager_id 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。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 21-10 を参照）。

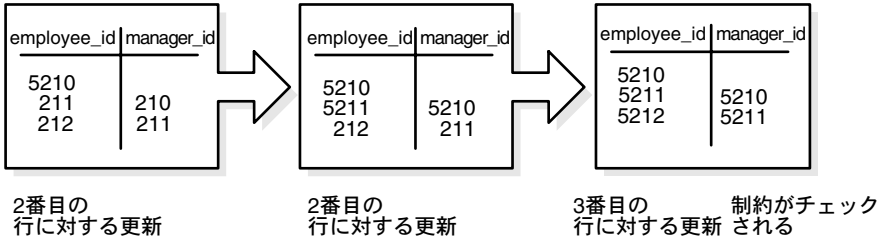
図 21-10 更新前の EMP 表




```
UPDATE employees
  SET employee_id = employee_id + 5000,
      manager_id = manager_id + 5000;
```

制約は、各 manager_id 値が employee_id 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 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 21-11 に、制約チェックの前に SQL 文全体のアクションが実行される流れを示します。

図 21-11 制約チェック



この項の例は、INSERT 文と UPDATE 文を実行した場合の制約チェックのメカニズムを示しています。UPDATE 文、INSERT 文および DELETE 文など、すべてのタイプの DML 文でも、同じメカニズムを使用しています。

また、これらの例は、自己参照型整合性制約を使用してチェックのメカニズムを示すものでした。これと同じメカニズムが、次に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
- CHECK

関連項目： 21-22 ページ「遅延制約チェック」

デフォルト列値と整合性制約チェック

デフォルト値は、文の解析前に INSERT 文の一部として組み込まれます。このため、デフォルトの列値はすべての整合性制約チェックの対象になります。

遅延制約チェック

制約の妥当性のチェックは、トランザクション終了時まで**遅延**できます。

- **遅延制約**では、制約が満たされているかどうかのチェックがコミット時にしか実行されません。遅延制約違反があると、コミット時にそのトランザクションはロールバックされます。
- **即時制約**の場合（つまり遅延制約でない場合）、チェックはそれぞれの文が終わった時点で実行されます。制約違反があると、その文は即時にロールバックされます。

制約によって**アクション**（DELETE CASCADE など）が発生する場合は、遅延制約か即時制約かに関係なく、そのアクションは、アクションを発生させた文の一部として実行されます。

制約の属性

制約は、**遅延可能**または**遅延不可**、および**初期遅延**または**初期即時**のどちらかに定義できます。これらの属性は、制約ごとに異なるものを指定できます。それらの定義は、CONSTRAINT 句の中で次のキーワードを使用して指定します。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE

制約について、追加、削除、使用可能と使用禁止の切替え、または妥当性チェックを実行できます。また、制約の属性も変更できます。

関連項目：

- 制約属性とそのデフォルト値の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 21-24 ページ「制約の状態」
- 21-25 ページ「制約の状態変更」

SET CONSTRAINTS モード

SET CONSTRAINTS 文は、特定のトランザクションのために、制約を DEFERRED または IMMEDIATE のどちらかに指定します（構文的にも意味的にも ANSI 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。CHECK 制約が一貫しており、他に SET CONSTRAINTS 文が発行されない場合、Oracle はまずトランザクション内で事前に遅延されている制約をチェックし、次にそのトランザクション内のそれ以降の文の制約を即時にチェックします。制約チェックが失敗すると、エラーが通知されます。この場合、COMMIT を発行するとトランザクション全体がロールバックされます。

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。

制約の状態

CREATE TABLE 文または ALTER TABLE 文を使用すると、整合性制約を表レベルで使用可能または使用禁止にできます。また、次のように、制約を VALIDATE または NOVALIDATE に設定し、ENABLE または DISABLE と組み合せることもできます。

- ENABLE を指定すると、入ってくるすべてのデータが制約に準拠していることが保証されます。
- DISABLE を指定すると、入ってくるデータは、制約に準拠しているかどうかに関係なく許可されます。
- VALIDATE を指定すると、既存のデータが制約に準拠していることが保証されます。
- NOVALIDATE は、一部の既存データが制約に準拠していない可能性があることを意味します。

さらに、次のようになります。

- ENABLE VALIDATE は ENABLE と同じです。制約がチェックされ、すべての行が保持されることが保証されます。
- ENABLE NOVALIDATE は、制約はチェックされますが、すべての行について TRUE でなくても許されることを意味します。これにより、既存の行が制約に違反することは許され、しかも、新しい行や変更した行に対してはすべて有効であることが保証されます。

  ALTER TABLE 文の ENABLE NOVALIDATE オプションを使用すると、表内のすべてのデータの妥当性を最初にチェックせずに、使用禁止にした制約について制約チェックを再開できます。
- DISABLE NOVALIDATE は DISABLE と同じです。制約はチェックされず、TRUE でなくてもかまいません。
- 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キーが自動的に作成されます。同様に、一意キーまたは主キーが ENABLE から DISABLE に移る場合に、一意索引で使用可能にされていれば、一意索引は削除されます。
- 制約が NOVALIDATE 状態から VALIDATE 状態に移った場合は、すべてのデータをチェックする必要があります。（この操作は、きわめて低速の場合があります。）ただし、VALIDATE から NOVALIDATE に移っても、データがチェックされたことが無視されるだけです。
- 単一の制約を ENABLE NOVALIDATE 状態から ENABLE VALIDATE 状態に移動すると、読込み、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： ENABLE、DISABLE、VALIDATE および NOVALIDATE CONSTRAINT 句の使用方法の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

制約の状態変更

ALTER TABLE 文の MODIFY CONSTRAINT 句を使用すると、次の制約の状態を変更できます。

- DEFERRABLE または NOT DEFERRABLE
- INITIALLY DEFERRED または INITIALLY IMMEDIATE
- RELY または NORELY
- USING INDEX ...
- ENABLE または DISABLE
- VALIDATE または NOVALIDATE
- EXCEPTIONS INTO ...

関連項目： これらの制約の状態の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

22

データベース・アクセスの制御

この章では、Oracle データベースへのアクセスを制御する方法を説明します。この項の内容は、次のとおりです。

- データベース・セキュリティの概要
- スキーマ、データベース・ユーザーおよびセキュリティ・ドメイン
- ユーザー認証
- Oracle Internet Directory
- ユーザー表領域の設定と割当て制限
- ユーザー・グループ PUBLIC
- ユーザー・リソースの制限とプロファイル

データベース・セキュリティの概要

データベース・セキュリティには、データベースとデータベースに格納されているオブジェクトに対するユーザー・アクションを許可したり禁止する機能が関係しています。Oracle では、スキーマとセキュリティ・ドメインを使用して、データへのアクセスを制御したり、様々なデータベース・リソースの使用を制限します。

Oracle では、包括的な任意アクセス制御が提供されています。**任意アクセス制御**は、特定のオブジェクトに対するすべてのユーザー・アクセスを、権限によって調整します。権限とは、特定のオブジェクトに規定の方法でアクセスするための許可です。たとえば、表への問合せを実行する許可はその一例です。権限は、他のユーザーの裁量で任意に付与されるものであるため、**任意アクセス制御**という語を使用しています。

関連項目： 第 23 章「権限、ロールおよびセキュリティ・ポリシー」

スキーマ、データベース・ユーザーおよびセキュリティ・ドメイン

ユーザー（**ユーザー名**とも呼ぶ）は、データベース内で定義されている名前であり、この名前を使用してオブジェクトに接続したりアクセスできます。**スキーマ**は、表、ビュー、クラスタ、プロシージャおよびパッケージなどのオブジェクトの集合に名前を付けたものです。スキーマとユーザーは、データベース管理者がデータベース・セキュリティを管理するのに役立ちます。

エンタープライズ・ユーザーはディレクトリ内で管理されるユーザーで、各データベース内でアカウントやスキーマを作成しなくても、複数のスキーマおよびデータベースへのアクセス権を取得できます。これによりユーザーや DBA の処理をより簡単にし、また、その権限を一元的に変更できるため、セキュリティが改善されます。

新たにデータベースを作成したり、既存のデータベースを変更する場合、セキュリティ管理者は、ユーザーのセキュリティ・ドメインについていくつか決定する必要があります。たとえば、次のようなことを決定します。

- ユーザー認証情報を、データベース、オペレーティング・システムまたはネットワーク認証サービスのどれを使用して維持管理するか。
- ユーザーのデフォルト表領域と一時表領域の設定。
- ユーザーがアクセス可能な表領域のリスト（存在する場合）と、そのリストに含まれている表領域に対応付ける割当て制限。
- ユーザーのリソース制限プロファイル。ユーザーが使用できるシステム・リソースの量を制限します。
- データベース操作の実行に必要な、オブジェクトへの適切なアクセス権をユーザーに提供する権限、ロールおよびセキュリティ・ポリシー。

この章では、このうち最初の 4 つのセキュリティ・ドメイン・オプションについて説明します。

注意： この章の情報は、ユーザー定義データベースのすべてのユーザーに適用されます。特殊なデータベース・ユーザーである SYS および SYSTEM には適用されません。これらの特殊なユーザーのセキュリティ・ドメインの設定は、絶対に変更しないでください。

関連項目：

- 第 23 章「権限、ロールおよびセキュリティ・ポリシー」
- エンタープライズ・ユーザーの詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。
- 特別なユーザーである SYS および SYSTEM の詳細と、セキュリティ管理者の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

ユーザー認証

データベース・ユーザー名が無許可で使用されないようにするため、Oracle では次のように複数の異なる方法で標準データベース・ユーザーの妥当性をチェックします。認証は次のものによって実行できます。

- オペレーティング・システム
- ネットワーク・サービス
- 対応付けられた Oracle データベース
- ユーザーに代ってトランザクションを実行する中間層アプリケーションの Oracle データベース
- Secure Socket Layer（SSL）プロトコル

通常は、検査を簡素化するため、前述のどれか 1 つの方法を使用してデータベースのすべてのユーザーを認証します。ただし、Oracle では、同じデータベース・インスタンス内で前述の方法をすべて使用できます。

さらに、Oracle では、ネットワーク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。

データベース管理者は特別なデータベース操作を実行するため、データベース管理者に対しては特別な認証手順が必要になります。

オペレーティング・システムによる認証

一部のオペレーティング・システムでは、オペレーティング・システムの維持している情報を、Oracle がユーザーの認証のために使用できます。オペレーティング・システムによる認証の利点は、次のとおりです。

- ユーザーがより簡単に Oracle に接続できます。ユーザー名やパスワードを指定する必要はありません。たとえば、ユーザーは SQL*Plus を起動する際に次のように入力して、ユーザー名とパスワードのプロンプトを省略できます。

  ```
  SQLPLUS /
  ```

- ユーザー認証はオペレーティング・システムで集中管理されます。Oracle がユーザー・パスワードを格納したり管理する必要がなくなります。ただし、ユーザー名は Oracle データベース内に保持されます。
- データベースのユーザー名エントリと、オペレーティング・システムの監査証跡が対応します。

データベース・ユーザーの認証にオペレーティング・システムを使用する場合は、分散データベース環境とデータベース・リンクに関していくつかの特別な考慮事項があります。

関連項目：

- 『Oracle9*i* データベース管理者ガイド』
- オペレーティング・システムによる認証の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

ネットワークによる認証

Oracle は、後述のようにネットワークによる複数の認証方法をサポートしています。

サードパーティベースの認証テクノロジ

ネットワーク認証サービス（DCE、Kerberos または SESAME など）を使用できる場合、Oracle はこれらのネットワーク・サービスによる認証を受け入れることができます。Oracle でネットワーク認証サービスを使用するには、Oracle9*i* Enterprise Edition と Oracle Advanced Security が必要です。

関連項目：

- ネットワーク認証の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。ネットワーク認証サービスを使用する場合は、ネットワーク・ロールとデータベース・リンクについて特別な考慮事項があります。
- Oracle Advanced Security オプションの詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。

公開鍵インフラストラクチャベースの認証

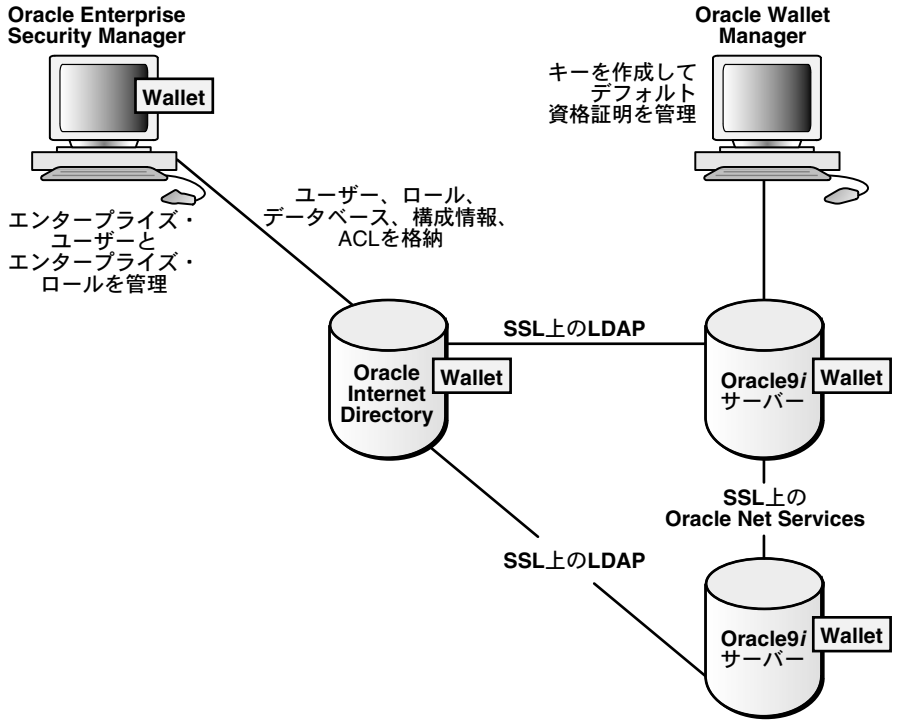
公開鍵暗号化システムに基づく認証システムは、ユーザー・クライアントにデジタル証明書を発行します。ユーザー・クライアントはそれを使用し、認証サーバーを直接関与させないで、社内のサーバーを直接認証します。Oracle は、公開鍵と証明書を使用するための公開鍵インフラストラクチャ（PKI）を提供します。PKI のコンポーネントは、次のとおりです。

- Secure Sockets Layer（SSL）による認証および保護セッション・キー管理。
- Oracle Call Interface（OCI）および PL/SQL 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**。信頼できるサード・パーティの識別です。信頼できるとは、実在者との同一性が確認された時に使用されます。一般的に、信頼されている認証局によってユーザーの証明書が発行されます。複数レベルの信頼できる証明書がある場合、証明連鎖における下位レベルの信頼できる証明書は、それより上のレベルの証明書をすべて再検証する必要はありません。
- **Oracle Wallet**。これは、ユーザーの秘密鍵、ユーザー証明書および一連のトラスト・ポイント（ユーザーが信頼するルート証明書のリスト）を含むデータ構造です。

- **Oracle Wallet Manager**。Oracle 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。Wallet Manager には、次のような機能があります。
  - ユーザー・キーを保護します。
  - Oracle のクライアントとサーバー上で X.509v3 証明書を管理します。
  - 公開鍵と秘密鍵のペアを生成し、認証局に提出する証明書要求を作成します。
  - エンティティの証明書をインストールします。
  - エンティティの信頼できる証明書を構成します。
  - Wallet をオープンして、PKI ベースのサービスにアクセスできるようにします。
  - Oracle Enterprise Login Assistant でオープンできる Wallet を作成します。
- X.509v3 証明書。Oracle 外部の認証局から取得します。この証明書は、エンティティの公開鍵が、信頼されているエンティティ（Oracle 外部の認証局）によって署名されたときに有効となります。この証明書は、そのエンティティの情報が正しいこと、および公開鍵がそのエンティティに属していることを保証します。証明書が Oracle Wallet にロードされ、認証が使用可能になります。
- **Oracle Enterprise Security Manager**。管理を容易にし、セキュリティ・レベルを高めるために、集中的な権限管理を提供します。ロールが Lightweight Directory Access Protocol（LDAP）をサポートしている場合は、Oracle Enterprise Security Manager により、Oracle Internet Directory にロールを格納し、取り出すことができます。また、ロールが Oracle スキーマと関連アクセス制御リストのインストールをサポートする場合は、そのロールを他の LDAP v3 準拠のディレクトリ・サーバーに格納できます。
- **Oracle Internet Directory**。Oracle9*i* データベース上に作成された LDAP v3 準拠のディレクトリです。X.509 証明書を使用して認証されたユーザーについて、セキュリティ属性や権限など、ユーザーおよびシステム構成環境を管理できます。Oracle Internet Directory は認証レベルのアクセス制御を規定します。これにより、特定のユーザー（社内のセキュリティ管理者など）のみに、ディレクトリの特定の属性の読込み、書込みまたは更新権限を限定できます。また、SSL 暗号化を介したディレクトリの問合せと応答の保護と認証もサポートしています。
- **Oracle Enterprise Login Assistant**。ユーザーの 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。このツールには、Oracle Wallet Manager で有効性が確認される機能のサブセットがあります。Wallet を事前に Oracle Wallet Manager で構成する必要があります。

図 22-1 に、Oracle の公開鍵インフラストラクチャを示します。

図 22-1 Oracle の公開鍵インフラストラクチャ



注意： Oracle で公開鍵インフラストラクチャベースの認証を使用するには、Oracle9*i* Enterprise Edition と Oracle Advanced Security が必要です。

リモート認証

Oracle 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。

注意： Oracle で RADIUS を介したユーザーのリモート認証を使用するには、Oracle9*i* Enterprise Edition と Oracle Advanced Security が必要です。

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

Oracle データベースによる認証

Oracle では、データベースに格納されている情報を使用して、データベースに接続しようとするユーザーを認証できます。

Oracle でデータベースによる認証を使用する場合は、対応するパスワードを指定してそれぞれのユーザーを作成します。ユーザーは、接続の確立時に正しいパスワードを入力します。この方法により、データベースの無許可使用が防止されます。ユーザーのパスワードは、暗号化された形式でデータ・ディクショナリに格納されます。ユーザーは、いつでも自分のパスワードを変更できます。

接続時のパスワード暗号化

パスワードの機密性を保護するため、Oracle ではネットワーク（クライアント / サーバーおよびサーバー / サーバー）接続時にパスワードを暗号化できます。クライアントおよびサーバーのマシンでこの機能を使用可能にすると、パスワードは修正 DES（データ暗号化規格）アルゴリズムを使用して暗号化され、その後ネットワーク経由で送信されます。パスワードをネットワークへの侵入者から保護するために、接続のパスワード暗号化を使用可能にすることをお薦めします。

関連項目： ネットワーク・システムにおけるパスワード暗号化の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

アカウントのロック

指定された試行回数の範囲内でユーザーがシステムにログインできない場合、ユーザーのアカウントがロックされることがあります。アカウントの構成方法によって、一定の時間の後に自動的にロックが解除されるか、またはデータベース管理者がロックを解除する必要があります。

`CREATE PROFILE` 文は、ユーザーが試行できるログインの失敗回数、および自動ロック解除前にアカウントがロックされたままになっている時間を構成するために使用します。

データベース管理者は、手動でアカウントをロックすることもできます。その場合、アカウントは自動的にはロック解除されないため、データベース管理者による明示的なロック解除が必要です。

関連項目： 22-20 ページ「プロファイル」

パスワードの存続期間と期限切れ

パスワードの存続期間と時間切れのオプションを使用すると、データベース管理者はパスワードの存続期間を指定できます。その期間を過ぎると、パスワードは期限切れになり、そのアカウントにログインする前にパスワードを変更する必要があります。パスワードの期限が切れた後にデータベース・アカウントへの最初のログイン試行で、ユーザーのアカウントは猶予期間に入り、その猶予期間が終わるまで、ユーザーがログインするたびに警告メッセージが発行されます。

ユーザーは、猶予期間内にパスワードを変更するよう求められます。パスワードが猶予期間内に変更されなければ、そのアカウントはロックされ、それ以降、そのアカウントにはデータベース管理者の介入がなければログインできなくなります。

また、データベース管理者がパスワードの状態を時間切れに設定することもできます。この場合は、ユーザーのアカウントの状態が時間切れに変更され、そのユーザーがデータベースにログインするには、自分で、またはデータベース管理者に依頼してパスワードを変更する必要があります。

パスワード履歴

パスワード履歴オプションは、新しく指定される各パスワードを調べて、指定された期間内に、または指定されたパスワード変更回数の間に、同じパスワードが再利用されないようにするためのものです。データベース管理者は、`CREATE PROFILE` 文を使用してパスワードの再利用のルールを構成できます。

パスワードの複雑度の検証

複雑度の検証は、パスワードを推定してシステムに入ろうとする侵入者に対して、各パスワードが十分保護可能な複雑なものであることをチェックすることです。

Oracle において、デフォルトのパスワード複雑度検証ルーチンでは、各パスワードに次の条件が求められます。

- 4 文字以上の長さ。
- ユーザー ID と同じでないこと。
- 少なくとも 1 文字のアルファベット、1 文字の数字および 1 文字の句読点が含まれていること。
- welcome、account、database、user など、簡単な語からなる内部リストにある単語と一致しないこと。
- 前のパスワードと 3 文字以上異なっていること。

複数層の認証と認可

複数層環境では、Oracle は権限を制限し、すべての層を通じてクライアント識別を保持し、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できることが重要です。ただし、中間層が持つ利点の 1 つに**接続プーリング**があります。これにより、複数のユーザーが個別に接続しなくても 1 つのデータ・サーバーにアクセスできます。この種の環境では、接続を迅速に確立および切断できる機能が必要です。Oracle は、これらの環境で Oracle Call Interface を介して**軽量セッション**を作成する機能を提供します。これらの軽量セッションにより、各ユーザーをデータベース・パスワードで認証でき、個別のデータベース接続によるオーバーヘッドは生じません。また、中間層を介して実際のユーザーの識別性が保たれます。

パスワードあり、またはパスワードなしで軽量セッションを作成できます。中間層が外部またはファイアウォールにある場合は、軽量ユーザー・セッションごとに、パスワードを伴う軽量セッションを確立する方法が適しています。内部アプリケーション・サーバーの場合は、パスワードを必要としない軽量セッションを作成する方法が適しています。

クライアント、アプリケーション・サーバーおよびデータベース・サーバー

複数層アーキテクチャ環境では、アプリケーション・サーバーはクライアントにデータを提供し、クライアントと 1 つ以上のデータベース・サーバーとの間のインタフェースとして機能します。

このアーキテクチャでは、アプリケーション・サーバーを使用して Web ブラウザなどのクライアントの資格証明を検査できます。また、データベース・サーバーは、アプリケーション・サーバーが自身のために実行する操作と、クライアントのためにアプリケーション・サーバーが実行する操作を監査できます。たとえば、前者の操作はデータベース・サーバーへの接続要求などで、後者の操作はクライアント上で表示する情報の要求などです。

複数層環境における認証は、次のようなトラスト領域に基づいています。

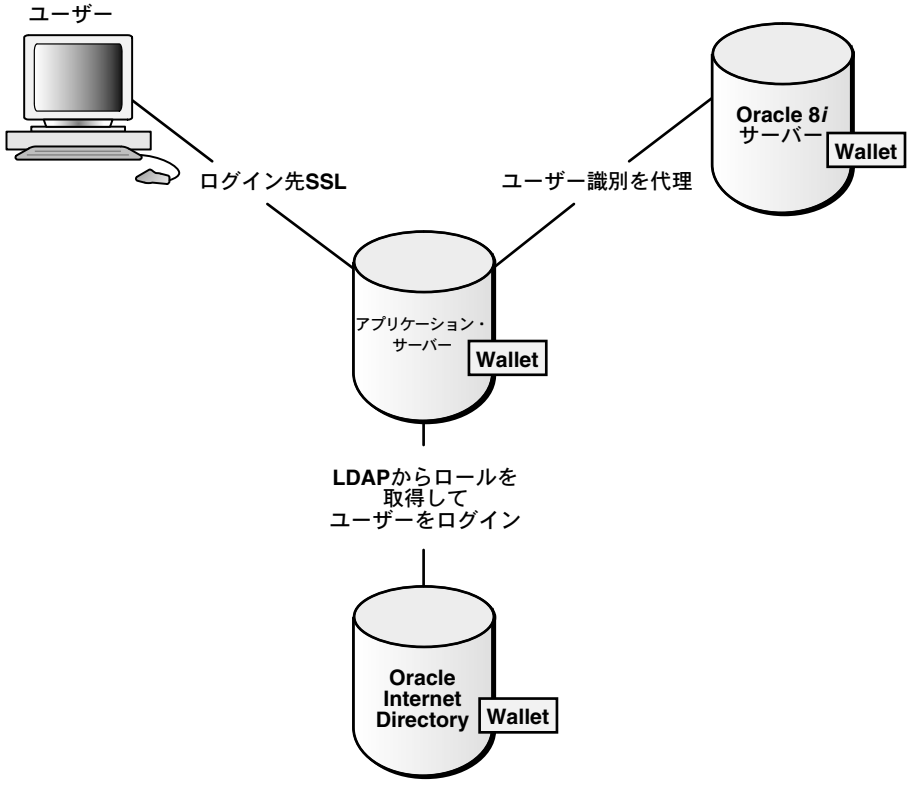
- クライアントは、通常はパスワードまたは X.509 証明書を使用して、アプリケーション・サーバーに認証の証明を提供します。
- アプリケーション・サーバーは、クライアント認証を検査し、自身をデータベース・サーバーに対して認証します。
- データベース・サーバーは、アプリケーション・サーバー認証をチェックし、クライアントが存在するかどうかを検証し、アプリケーション・サーバーがこのクライアントへの接続権限を持っているかどうかを検証します。

アプリケーション・サーバーは、接続中のクライアントのロールを使用可能にすることもできます。また、これらのロールは、認証リポジトリとして機能するディレクトリから取得できます。アプリケーション・サーバーが要求できるのは、これらのロールを使用可能にすることだけです。データベースは、次のことを検証します。

- クライアントがこれらのロールを付与されているかどうか、内部のロール・リポジトリをチェックします。
- アプリケーション・サーバーは、ユーザーのロールを使用して、そのユーザーのために接続する権限を付与されているかどうかをチェックします。

図 22-2 に、複数層認証の例を示します。

図 22-2 複数層の認証



中間層アプリケーションのセキュリティの問題

中間層アプリケーションには、次のようにセキュリティ上の問題が多数あります。

- アカウンタビリティ : データベース・サーバーは、クライアントのアクションと、アプリケーションがクライアントに代って実行するアクションを区別できる必要があります。この両方のアクションを監査できることが必要です。
- 区別 : データベース・サーバーは、Web サーバー・トランザクション、ブラウザ・クライアントのための Web サーバー・トランザクションおよびデータベースに直接アクセスするクライアントを、区別できる必要があります。
- 最小限度の権限 : ユーザーと中間層には、そのジョブを実行するための必要最小限度の権限を与える必要があります。

複数層環境における識別の問題

複数層アプリケーションは、すべての接続層を通じてクライアントの識別をメンテナンスします。この操作を必要とするのは、要求元クライアントの識別が失われると、有効な監査レコードをメンテナンスできなくなるためです。また、アプリケーション・サーバーがクライアントのために実行する操作と、アプリケーション自身のための操作を区別できなくなります。

複数層環境における制限付きの権限

複数層環境での権限は、要求された操作の実行に必要な権限に限定されます。

クライアントの権限 クライアントの権限は、複数層環境では最小限度になります。操作は、クライアントにかわってアプリケーション・サーバーが実行します。

アプリケーション・サーバーの権限 複数層環境におけるアプリケーション・サーバーの権限は、クライアント操作の実行中に望ましくない操作や不要な操作を実行できないように制限されます。

> **関連項目 :** 複数層認証の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

Secure Socket Layer プロトコルによる認証

Secure Socket Layer（SSL）プロトコルは、アプリケーション・レイヤー・プロトコルです。SSL は、Oracle Internet Directory でのグローバル・ユーザー管理とは関係なく、データベースに対するユーザー認証に使用できます。つまり、ユーザーは SSL を使用してデータベースを認証でき、そのディレクトリ・アクセスについて何も暗黙的に指定する必要がありません。ただし、エンタープライズ・ユーザー機能を使用してユーザーとその権限をディレクトリ内で管理する場合、ユーザーは SSL を使用してデータベースを認証する必要があります。SSL の使用は、初期化ファイルのパラメータで指定されます。

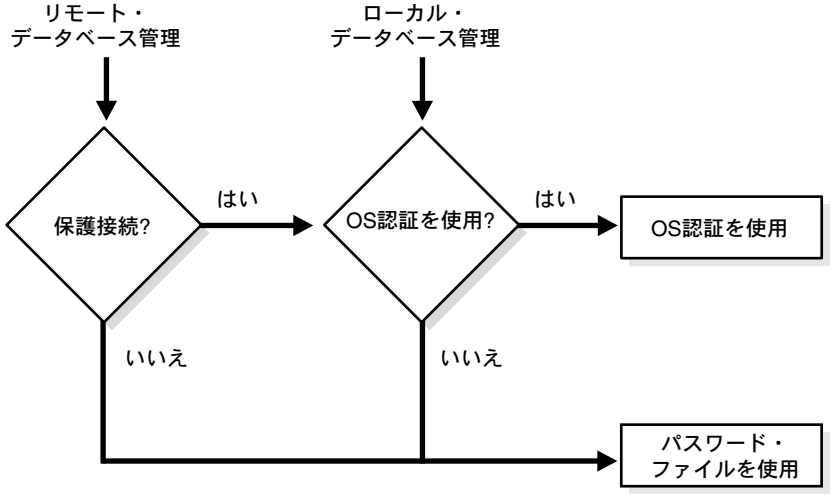
データベース管理者の認証

データベース管理者は、通常のデータベース・ユーザーが実行できない特別な操作（データベースの停止や起動など）を実行します。Oracle では、データベース管理者のユーザー名に対して、さらに安全性の高い認証方式を使用します。

データベース管理者の認証方式として、オペレーティング・システムによる認証とパスワード・ファイルによる認証のどちらを使用するかを選択できます。

図 22-3 に、データベースをローカルに（データベースが存在する同じマシン上で）管理するか、1 つのリモート・クライアントから多数の異なるデータベース・マシンを管理するかに応じた、データベース管理者の認証方式の選択肢を示します。

図 22-3 データベース管理者の認証方式



ほとんどのオペレーティング・システムでは、データベース管理者のオペレーティング・システム認証には、データベース管理者のオペレーティング・システム・ユーザー名を特別なグループ（UNIX システムでは **dba** グループ）に入れること、またはそのオペレーティング・システム・ユーザー名に特別な処理を実行する権利を与えることが含まれます。

データベースは、パスワード・ファイルを使用することによって、SYSDBA または SYSOPER 権限を付与されたデータベース・ユーザー名を追跡します。

- SYSOPER の権限があると、データベース管理者は STARTUP、SHUTDOWN、ALTER DATABASE OPEN/MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG および RECOVER を実行できます。また、RESTRICTED SESSION 権限もこの権限に含まれます。
- SYSDBA 権限には、ADMIN OPTION および SYSOPER システム権限とともに、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：

- データベース管理者のオペレーティング・システム認証の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。
- 『Oracle9*i* データベース管理者ガイド』

Oracle Internet Directory

Oracle Internet Directory は、Oracle データベース上のアプリケーションとして実装されているディレクトリ・サービスです。分散ユーザーおよびネットワーク・リソースの情報を検索できます。Oracle Internet Directory は、オープンなインターネット規格のディレクトリ・アクセス・プロトコルである Lightweight Directory Access Protocol（LDAP）バージョン 3 に、Oracle サーバーが提供する高パフォーマンス、拡張性、耐久性および可用性を組み合せたものです。

Oracle Internet Directory には、次のものが含まれます。

- 要員およびリソースに関する情報を求めるクライアントの要求に応答し、TCP/IP 上の複数層アーキテクチャ・ディレクトリを使用してこれらの情報を更新する Oracle ディレクトリ・サーバー
- Oracle ディレクトリ・サーバー間で LDAP データをレプリケートする Oracle ディレクトリ・レプリケーション・サーバー
- Graphical User Interface（GUI）管理ツールの Oracle Directory Manager
- 各種コマンドライン管理ツールおよびデータ管理ツール

関連項目：『Oracle Internet Directory 管理者ガイド』

ユーザー表領域の設定と割当て制限

データベース管理者は、すべてのユーザーのセキュリティ・ドメインの一部として、表領域の使用についていくつかのオプションを設定できます。

デフォルト表領域のオプション

ユーザーがオブジェクトを入れる表領域を指定しないでスキーマ・オブジェクトを作成すると、そのオブジェクトはそのユーザーのデフォルト表領域に格納されます。ユーザーのデフォルト表領域は、ユーザー作成時に設定し、ユーザー作成後に変更できます。

一時表領域のオプション

一時セグメントの作成を必要とする SQL 文をユーザーが実行すると、ユーザーの一時表領域にそのセグメントが割り当てられます。

表領域のアクセスと割当て制限

データベースのどの表領域についても、ユーザーごとに**表領域割当て制限**を設定できます。これにより、次の 2 つのことが可能になります。

- ユーザーが適切な権限を付与されている場合は、そのユーザーが指定された表領域を使用してスキーマ・オブジェクトを作成することを許可できます。
- 指定された表領域にあるユーザーのスキーマ・オブジェクトの記憶域に割り当てられたスペースの量を制限できます。

デフォルトでは、各ユーザーはデータベースの表領域での割当て制限を課されていません。このため、ユーザーになんらかのタイプのスキーマ・オブジェクトを作成する権限がある場合、そのユーザーには、オブジェクトを作成するときの表領域割当て制限が設定されているか、十分な表領域割当て制限を設定されている別のユーザーのスキーマにそのオブジェクトを作成する権限も必要になります。

ユーザーには、2 種類の表領域割当て制限を設定できます。1 つは表領域内のディスク領域を特定の量に制限する割当て制限（バイト、キロ・バイトまたはメガ・バイト単位）、もう 1 つは表領域内のディスク領域を無制限とする割当て制限です。ユーザーのオブジェクトが表領域内の領域を使用しすぎないように、特定量の割当て制限を設定する必要があります。

表領域の割当て制限および一時セグメントは、相互に影響し合いません。

- 一時セグメントが作成されても、それはユーザーに課されている割当て制限の領域には加算されません。Oracle によって一時セグメントに自動的に作成されるスキーマ・オブジェクトは、SYS の所有となるため、割当て制限は適用されません。
- 一時セグメントは、ユーザーが割当て制限を課されていない表領域内に作成できます。

ユーザーの表領域の割当て制限は、そのユーザーの作成時に設定できます。その割当て制限を変更したり、後で別の割当て制限を追加できます。

ユーザーの表領域アクセスを取り消すには、そのユーザーの現行の割当て制限を 0（ゼロ）に変更します。割当て制限を 0（ゼロ）にすると、そのユーザーのオブジェクトは取り消したその表領域内に残りますが、それらのオブジェクトに新しい領域を割り当てることはできません。

ユーザー・グループ PUBLIC

各データベースには、PUBLIC というユーザー・グループがあります。PUBLIC ユーザー・グループは、表やビューなど、特定のスキーマ・オブジェクトへのパブリック・アクセスを提供し、すべてのユーザーに特定のシステム権限を提供します。すべてのユーザーは、自動的に PUBLIC ユーザー・グループに所属します。

PUBLIC のメンバーとして、ユーザーは接頭辞が USER および ALL であるすべてのデータ・ディクショナリ表を参照（SELECT FROM）できます。さらに、ユーザーは PUBLIC に対して権限やロールを付与できます。すべてのユーザーは、PUBLIC に付与されている権限を使用できます。

PUBLIC には、任意のシステム権限、オブジェクト権限またはロールを付与または取消しできます。ただし、アクセス権に対する厳密なセキュリティを維持するため、PUBLIC に付与する権限とロールは、すべてのユーザーに関係のあるもののみにしてください。

PUBLIC に対してなんらかのシステム権限やオブジェクト権限を付与するか取り消すと、データベース中のすべてのビュー、プロシージャ、ファンクション、パッケージおよびトリガーが再コンパイルされます。

PUBLIC には次の制限があります。

- PUBLIC に対して表領域割当て制限は設定できませんが、UNLIMITED TABLESPACE システム権限は設定できます。
- データベース・リンクおよびシノニムは、（CREATE PUBLICDATABASE LINK/SYNONYM を使用して）PUBLIC として作成できますが、その他のスキーマ・オブジェクトを PUBLIC の所有とすることはできません。たとえば、次の文は無効です。

  ```
  CREATE TABLE public.employees ... ;
  ```

注意： キーワード PUBLIC を指定してロールバック・セグメントを作成することはできますが、そのセグメントは PUBLIC ユーザー・グループが所有することにはなりません。すべてのロールバック・セグメントは SYS が所有します。

関連項目：

- 第 2 章「データ・ブロック、エクステントおよびセグメント」
- 第 23 章「権限、ロールおよびセキュリティ・ポリシー」

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・リソースに多額の費用がかかる大規模なマルチ・ユーザー・システムにおいて、このリソース制限機能はたいへん有効です。1 人以上のユーザーが過度にリソースを使用すると、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。シングル・ユーザー・データベースまたは小規模のマルチ・ユーザー・データベースの場合、ユーザーがシステム・リソースを消費してもそれほど有害な影響はないため、システム・リソース機能はそれほど重要ではありません。

ユーザーのリソース制限とパスワード管理の作業環境は、そのユーザーのプロファイルを使用して管理します。プロファイルとは、そのユーザーに割り当てることができる一連のリソース制限に名前を付けたものです。それぞれの Oracle データベースに指定できるプロファイルの数に、制限はありません。Oracle において、セキュリティ管理者は、プロファイルによるリソース制限の規定を全体的に使用可能または使用禁止に設定できます。

リソース制限を設定した場合、ユーザーがセッションを作成すると、パフォーマンスがわずかに低下します。これは、ユーザーがデータベースに接続した時点で、そのユーザーのすべてのリソース制限データがロードされるためです。

関連項目： セキュリティ管理者の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

システム・リソースのタイプと制限

Oracle では、CPU タイムと論理読込みを含め、いくつかのタイプのシステム・リソースの使用を制限できます。一般に、それぞれのリソースは、セッション・レベル、コール・レベル、またはその両方で制御できます。

- セッション・レベル

  ユーザーがデータベースに接続するたびに、セッションが作成されます。それぞれのセッションは、Oracle を実行するコンピュータの CPU タイムとメモリーをコンシュームします。複数のリソース制限をセッション・レベルで設定できます。

  ユーザーがセッション・レベルのリソース制限を超えると、現行の文は終了（ロールバック）し、セッションの制限に達したことを示すメッセージが戻されます。この時点では、カレント・トランザクション内のそれ以前のすべての文の結果はそのまま残っています。ユーザーが実行できる操作は、COMMIT、ROLLBACK または接続の切断（この場合、カレント・トランザクションはコミットされます）のみです。それ以外の操作を実行すると、エラーが発生します。トランザクションがコミットまたはロールバックされた後も、カレント・セッションではユーザーはどんな作業も完了できません。

- コール・レベル

  SQL 文が実行されるたびに、いくつかのステップが実行されて文が処理されます。この処理では、データベースに対して複数のコールが異なる実行フェーズの一部として発行されます。1 回のコールで過度にシステムが使用されないように、複数のリソース制限をコール・レベルで設定できます。

  ユーザーがコール・レベルのリソース制限を超えると、Oracle は文の処理を停止し、その文をロールバックし、エラーを戻します。ただし、カレント・トランザクションのそれ以前の文の結果はそのまま残り、そのユーザーのセッションは接続されたままになります。

CPU タイム

SQL 文やその他のタイプのコールが Oracle に発行されると、そのコールを処理するために一定量の CPU 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 コールに使用される CPU タイムの合計を制限できます。これらの制限は、コールやセッションに使用される 1/100 秒（0.01 秒）単位の CPU 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 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・ブロック読込み数を制限できます。論理データ・ブロック読込みには、メモリーとディスクの両方からの論理データ・ブロック読込みが含まれます。これらの制限は、1 コールまたは 1 セッション中に実行されるブロック読込みの数として設定し、測定されます。

その他のリソース

Oracle では、さらに、その他のいくつかのセッション・レベルのリソース制限が提供されています。

- ユーザー当たりの**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 コールと Oracle コールの間隔がアイドル制限時間に達すると、カレント・トランザクションはロールバックされ、セッションは異常終了し、そのセッションのリソースはシステムに戻されます。次のコールは、ユーザーがインスタンスから切断されたことを示すエラーを受け取ります。この制限は、分単位の経過時間として設定します。

  注意： セッションがアイドル時間の制限を超えたために異常終了すると、その少し後に、異常終了したセッションの後処理としてプロセス・モニター（PMON）・バックグラウンド・プロセスがクリーン・アップを実行します。PMON がこのプロセスを完了するまでは、異常終了したセッションも、セッションまたはユーザー・レベルのリソース制限に加算されます。

- セッション当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。

注意： Oracle は、経過アイドル時間や経過接続時間を絶えず監視しているわけではありません。絶えず監視するとすれば、システム・パフォーマンスが低下します。そのかわり数分ごとにチェックします。このため、Oracle がこの制限を規定してからセッションを異常終了させるまでの間に、セッションはこの制限をわずかに（5 分など）超える可能性があります。

- セッションのプライベート SGA 領域（プライベート SQL 領域に使用）の容量の制限。この制限が重要になるのは、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、『Oracle9*i* データベース管理者ガイド』を参照してください。

プロファイル

プロファイルとは、Oracle データベースの有効なユーザー名に対して割り当てることができるリソース制限に名前を付けた集合です。プロファイルを使用すると、リソース制限を簡単に管理できます。また、パスワード方針を管理する手段としても使用できます。

プロファイルを使用する場合

ユーザー・プロファイルを作成して管理する必要があるのは、リソース制限がデータベース・セキュリティ・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する必要があります。これらの値は、典型的なユーザーが実行する操作のタイプを基準として決定できます。たとえば、あるクラスのユーザーが通常は大量の論理データ・ブロック読込みを実行しない場合、`LOGICAL_READS_PER_SESSION` および `LOGICAL_READS_PER_CALL` の制限は控えめに設定してください。

通常、ユーザー・プロファイルの適切なリソース制限値を決定するには、それぞれのタイプのリソースの使用状況について履歴情報を収集するのが最善です。たとえば、データベース管理者やセキュリティ管理者は、`AUDIT SESSION` 句を使用すると、`CONNECT_TIME`、

LOGICAL_READS_PER_SESSION および LOGICAL_READS_PER_CALL の制限値についての情報を収集できます。

その他の制限値の統計情報は、Oracle Enterprise Manager（または SQL*Plus）のモニター機能、特に統計モニターを使用して収集できます。

関連項目： 第 24 章「監査」

23

権限、ロールおよびセキュリティ・ポリシー

この章では、ユーザーが実行できるシステム操作とスキーマ・オブジェクトに対して実行できるアクセスを、権限、ロールおよびセキュリティ・ポリシーによって制御する方法について説明します。この章の内容は、次のとおりです。

権限の概要

権限は、特定のタイプの SQL 文を実行するため、または別のユーザーのオブジェクトにアクセスするための権利です。たとえば、次のことをする権利が権限です。

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

権限をユーザーに付与すると、それらのユーザーが業務で必要な作業を実行できるようになります。なお、権限は、必要な作業を実行する上で本当にその権限を必要としているユーザーにのみ付与します。必要でない権限まで付与すると、セキュリティを維持できなくなる可能性があります。ユーザーは次の 2 つの方法で権限を受け取ることができます。

- 権限を明示的にユーザーに付与します。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します。

権限は次の 2 つに分類できます。

- システム権限
- スキーマ・オブジェクト権限

> **関連項目：** すべてのシステム権限およびスキーマ・オブジェクト権限の詳細リストと、権限管理の手順は、『Oracle9*i* データベース管理者ガイド』を参照してください。

システム権限

システム権限とは、特定のアクションを実行する権限、または特定のタイプのスキーマ・オブジェクトに対してアクションを実行する権限のことです。たとえば、表領域を作成する権限や、データベース内の任意の表から行を削除する権限などがシステム権限です。システム権限には 60 以上の種類があります。

システム権限の付与と取消し

システム権限は、ユーザーやロールに対して付与したり取り消すことができます。システム権限をロールに付与すると、そのロールを使用してシステム権限を管理できます。たとえば、ロールを使用すると権限を選択的に使用できるようになります。

> **注意：** 通常、システム権限は管理者やアプリケーション開発者にのみ付与するようにしてください。エンド・ユーザーは一般的に、システム権限に関連する機能を必要としません。

次のどちらかを使用して、ユーザーやロールにシステム権限を付与したり、その権限を取り消します。

- Oracle Enterprise Manager のコンソール
- SQL 文 GRANT および REVOKE

システム権限を付与または取消しできるユーザー

他のユーザーにシステム権限を付与したり、他のユーザーのシステム権限を取り消すことができるのは、ADMIN OPTION によって特定のシステム権限を付与されているユーザー、あるいは GRANT ANY PRIVILEGE または GRANT ANY OBJECT PRIVILEGE システム権限を付与されているユーザーのみです。

スキーマ・オブジェクト権限

スキーマオブジェクト権限とは、次のように特定のスキーマ・オブジェクトに対して特定のアクションを実行する権限です。

- 表
- ビュー
- 順序
- プロシージャ
- ファンクション
- パッケージ

各タイプのスキーマ・オブジェクトごとに、異なるオブジェクト権限があります。たとえば、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。

たとえば、jward.emp という表があり、それに jward.employee というシノニムがある場合に、ユーザー jward が次の文を発行するとします。

```
GRANT SELECT ON emp TO swilliams;
```

ユーザー swilliams は、名前またはシノニム jward.employee を使用して表を参照することによって、jward.emp を問い合せることができます。

```
SELECT * FROM jward.emp;
SELECT * FROM jward.employee;
```

表、ビュー、プロシージャ、ファンクションまたはパッケージに対するオブジェクト権限を、そのオブジェクトの**シノニム**に付与した場合、結果はシノニムを使用しない場合と同じです。たとえば、jward が emp 表に対する SELECT 権限を swilliams に付与する場合、jward は次のどちらの文でも使用できます。

```
GRANT SELECT ON emp TO swilliams;
GRANT SELECT ON employee TO swilliams;
```

シノニムが削除された場合、そのシノニムを指定して権限が付与されていた場合でも、基礎になるスキーマ・オブジェクトに対して付与された権限はすべて有効なままです。

スキーマ・オブジェクト権限の付与と取消し

スキーマ・オブジェクト権限は、ユーザーやロールに対して付与したり取り消すことができます。オブジェクト権限をロールに付与する場合、その権限を選択的に使用可能にできます。ユーザーやロールのオブジェクト権限を付与したり取り消すには、次のものを使用します。

- SQL 文 GRANT および REVOKE
- Oracle Enterprise Manager

スキーマ・オブジェクト権限を付与できるユーザー

ユーザーは、自分のスキーマに含まれているスキーマ・オブジェクトに関しては、自動的にすべてのオブジェクト権限が付与されています。ユーザーは、自分が所有するスキーマ・オブジェクトに対するオブジェクト権限を、他のユーザーまたはロールに付与できます。GRANT ANY OBJECT PRIVILEGE 権限を付与されているユーザーは、GRANT 文の GRANT OPTION 句を使用してもしなくても、他のユーザーに対して指定したオブジェクト権限を付与したり取り消すことができます。この句を指定しない場合、権限受領者はその権限を使用できますが、別のユーザーには権限を付与できません。

たとえば、ユーザー SCOTT が表 t2 の所有者の場合：

```
SQL>GRANT grant any object privilege TO U1;
SQL> connect u1/u1
Connected.
SQL> GRANT select on scott.t2 TO U2;
SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
 WHERE TABLE_NAME = 'employees';

GRANTEE                        OWNER
------------------------------ ------------------------------
GRANTOR                        PRIVILEGE                                GRA
------------------------------ ---------------------------------------- ---
U2                             SCOTT
SCOTT                          SELECT                                   NO
```

関連項目：『Oracle9*i* SQL リファレンス』

表のセキュリティ

表に対するスキーマ・オブジェクト権限は、DML および DDL 操作のレベルで表のセキュリティ機能を実現します。

データ操作言語（DML）操作

表またはビューに対する DELETE、INSERT、SELECT および UPDATE DML 操作を使用する権限を付与できます。これらの権限は、表のデータを問い合せたり変更する必要があるユーザーとロールにのみ付与してください。

表に対する INSERT と UPDATE の各権限は、表の特定の列に制限できます。選択的な INSERT 権限を付与されたユーザーは、選択された列の値を行に挿入できます。その他の列には、NULL またはその列のデフォルト値が挿入されます。選択的な UPDATE 権限によって、ユーザーは行の特定の列に限ってその値を更新できます。機密データに対するユーザー・アクセスを制限するには、INSERT 権限と UPDATE 権限を選択的に使用します。

たとえば、データ入力者が employees 表の salary 列を変更しないようにするには、salary 列を含めずに、選択的な INSERT 権限または UPDATE 権限を付与します。（また、salary 列を含まないようにしたビューを使用すると、同じことをさらに高いセキュリティ・レベルで実現できます。）

> **関連項目：** これらの DML 操作の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

データ定義言語（DDL）操作

ALTER、INDEX および REFERENCES の各権限は、表に対する DDL 操作の実行を許可します。これらの権限によって、他のユーザーは表の依存性を変更または作成できるようになるため、この権限は慎重に付与する必要があります。表に対して DDL 操作を実行するユーザーには、さらにその他のシステム権限やオブジェクト権限が必要です。たとえば、表にトリガーを作成するには、その表に対する ALTER TABLE オブジェクト権限と CREATE TRIGGER システム権限の両方が必要です。

INSERT 権限および UPDATE 権限と同じように、表の特定の列に REFERENCES 権限を付与できます。REFERENCES 権限を付与されたユーザーは、付与の対象となった表を、自分の表の中に作成する外部キーの親キーとして使用できます。外部キーが存在すると、親キーに対して実行される可能性のあるデータ操作や表の変更が制限されるため、この動作は特殊な権限によって制御されます。列固有の REFERENCES 権限によって、権限受領者が使用できるのは、指定された列（当然、親表の主キーまたは一意キーを最低 1 つ含む）に制限されます。

> **関連項目：** 主キー、一意キーおよび整合性制約の詳細は、第 21 章「データ整合性」を参照してください。

ビューのセキュリティ

ビューに対するスキーマ・オブジェクト権限は、ビューの導出元の実表に実際に影響する様々な DML 操作を許可します。表に対する DML オブジェクト権限は、ビューに対しても同じように適用されます。

ビューの作成に必要な権限

ビューを作成するには、次の要件を満たしている必要があります。

- 次のどちらかのシステム権限が、明示的に、またはロールを介して付与されている必要があります。
  - CREATE VIEW システム権限（自分のスキーマ内にビューを作成するため）
  - CREATE ANY VIEW システム権限（別のユーザーのスキーマ内にビューを作成するため）
- 次のいずれかの権限が明示的に付与されている必要があります。
  - ビューの基礎となるすべてのベース・オブジェクトに対する SELECT、INSERT、UPDATE または DELETE オブジェクト権限
  - SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE または DELETE ANY TABLE システム権限
- さらに、ビューに対するアクセス権を他のユーザーに付与する場合は、ベース・オブジェクトに対するオブジェクト権限が GRANT OPTION 句付きで付与されているか、または ADMIN OPTION 句によって適切なシステム権限が付与されている必要があります。これらの権限がなければ、権限受領者はビューにアクセスできません。

  関連項目：『Oracle9*i* SQL リファレンス』

ビューによる表セキュリティの強化

ビューを使用するのに必要な権限は、そのビュー自体に対する適切な権限のみです。ビューの基礎となるベース・オブジェクトに対する権限は必要ありません。

ビューを使用することにより、表に対してさらに 2 つのセキュリティ・レベル（列レベルのセキュリティと値ベースのセキュリティ）が追加されます。

- ビューは、実表の中から選択した列のアクセスを提供できます。たとえば、employees 表の中から employee_id、last_name および manager_id の 3 列のみを表示するようにビューを定義できます。

```
CREATE VIEW employees_manager AS
    SELECT last_name, employee_id, manager_id FROM employees;
```

- ビューにより、表の情報に対して、値ベースのセキュリティを実現できます。ビュー定義の中に WHERE 句を使用すると、実表の中から選択した行のみが表示されます。次の 2 つの例を考えてみます。

```
CREATE VIEW lowsal AS
    SELECT * FROM employees
    WHERE salary < 10000;
```

LOWSAL ビューでは、employees 表のうち給与値が 10000 未満のすべての行にアクセスできます。LOWSAL ビューでは、employees 表のすべての列にアクセスできることに注目してください。

```
CREATE VIEW own_salary AS
    SELECT last_name, salary
    FROM employees
    WHERE last_name = USER;
```

own_salary ビューでは、last_name がビューの現行のユーザーに一致している行にのみアクセスできます。own_salary ビューは user 関数を使用します。この関数の値は、常に現行のユーザーを参照します。なお、このビューでは、列レベルのセキュリティと値ベースのセキュリティを併用しています。

プロシージャのセキュリティ

スタンドアロン・プロシージャ、ファクションおよびパッケージを含め、プロシージャに対する**スキーマ・オブジェクト権限**は、EXECUTE のみです。この権限は、プロシージャの実行、またはそのプロシージャをコールする他のプロシージャのコンパイルを必要とするユーザーにのみ付与してください。

プロシージャの実行とセキュリティ・ドメイン

特定のプロシージャに対する EXECUTE オブジェクト権限を持っているユーザーは、そのプロシージャを実行したり、それを参照するプログラム・ユニットをコンパイルできます。このプロシージャがコールされるときには、実行時権限チェックは実行されません。EXECUTE ANY PROCEDURE システム権限を付与されているユーザーは、データベース内の任意のプロシージャを実行できます。

プロシージャの実行権限は、ロールを通してユーザーに付与できます。

実行者権限プロシージャの場合は、参照オブジェクトに対する追加の権限が必要ですが、定義者権限プロシージャの場合は不要です。

関連項目： 23-21 ページ「PL/SQL ブロックとロール」

定義者権限 定義者権限プロシージャのユーザーに必要なのは、そのプロシージャを実行する権限のみで、そのプロシージャがアクセスする基礎となるオブジェクトの権限は不要です。これは、定義者権限プロシージャは、その実行者に関係なく、所有者のセキュリティ・

ドメインで動作するからです。プロシージャの所有者は、参照オブジェクトに対する必要なオブジェクト権限をすべて持つ必要があります。定義者権限プロシージャのユーザーに付与する権限が少ないほど、データベース・アクセスを厳密に制御できます。

定義者権限プロシージャを使用して、プライベート・データベース・オブジェクトへのアクセスを制御し、データベースのセキュリティ・レベルを強化できます。定義者権限プロシージャを記述し、ユーザーには EXECUTE 権限のみを付与することにより、ユーザーがそのプロシージャを介さなければ参照オブジェクトにアクセスできないようにできます。

実行時には、定義者権限ストアド・プロシージャの所有者の権限が、常にそのプロシージャを実行する前にチェックされます。参照オブジェクトに対して必要な権限が、定義者権限プロシージャの所有者から取り消されていると、所有者もその他のユーザーも、そのプロシージャを実行できません。

注意： トリガーの実行には、定義者権限プロシージャと同じパターンが適用されます。ユーザーは、実行する権限を付与されている SQL 文を実行できます。SQL 文の実行結果として、トリガーが起動されます。トリガー・アクション内の文は、トリガーを所有するユーザーのセキュリティ・ドメインで実行されます。

関連項目： 第 17 章「トリガー」

実行者権限 実行者権限プロシージャは、実行者のすべての権限で実行されます。実行者権限プロシージャが定義者権限プロシージャによって直接または間接的にコールされた場合でなければ、ロールが使用可能になります。実行者権限プロシージャのユーザーには、そのプロシージャが実行者のスキーマ内で解決される外部参照を介してアクセスする、オブジェクトに対する権限が（直接、またはロールを介して）必要です。

実行者には、DML 文または動的 SQL 文に埋め込まれているプログラム参照にアクセスする権限が実行時に必要です。これは、この種のプログラム参照は実質的に実行時に再コンパイルされるためです。

PL/SQL ファンクション・コールなど、その他のすべての外部参照の場合、所有者の権限はコンパイル時にチェックされ、実行時チェックは行われません。したがって、実行者権限プロシージャのユーザーには、DML 文や動的 SQL 文の外側にある外部参照に対する権限は不要です。また、実行者権限プロシージャの開発者による権限の付与は、実行者権限プロシージャによって直接参照されるすべてのオブジェクトに対してではなく、プロシージャ自体に対する権限を自分に付与するだけでよいことになります。

ほとんどの DBMS_* パッケージなど、Oracle が提供する多数のパッケージは、実行者権限で実行されます。つまり、所有者（SYS）ではなく現行のユーザーとして実行されます。ただし、DBMS_RLS パッケージなど、いくつか例外があります。

複数のプログラム・ユニットからなるソフトウェア・バンドルを作成し、一部のプログラム・ユニットには定義者権限を付与し、残りのプログラム・ユニットには実行者権限を付与して、プログラムのエントリ・ポイントを限定できます（**コントロール・ステップイン**）。

エントリ・ポイント・プロシージャを実行する権限を持つユーザーは、内部プログラム・ユニットを間接的に実行できますが、内部プログラムを直接コールすることはできません。

関連項目：

- 23-24 ページ「ファイングレイン・アクセス・コントロール」
- オラクル社が提供するパッケージの詳細は、『Oracle9*i* 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。

プロシージャの作成または変更に必要なシステム権限

プロシージャを作成するには、ユーザーには CREATE PROCEDURE または CREATE ANY PROCEDURE **システム権限**が必要です。プロシージャを変更する、つまりプロシージャを手動で再コンパイルするには、そのプロシージャを所有しているか、ALTER ANY PROCEDURE システム権限を持っている必要があります。

プロシージャを所有するユーザーは、プロシージャ本体で参照されるスキーマ・オブジェクトに対する権限も持っている必要があります。プロシージャを作成するには、プロシージャによって参照されるすべてのオブジェクトに対して、必要な権限（システム権限やオブジェクト権限）を明示的に付与されている必要があります。ロールを介してはそれらの権限を取得できません。これには、作成するプロシージャ内からコールするプロシージャに対する EXECUTE 権限も含まれます。

また、トリガーでは、参照オブジェクトに対する権限をトリガーの所有者に対して明示的に付与することが必要です。権限が明示的に、またはロールを介して付与されていても、無名 PL/SQL ブロックは任意の権限を使用できます。

パッケージとパッケージ・オブジェクト

パッケージに対する EXECUTE オブジェクト権限を付与されているユーザーは、パッケージ内の任意のパブリック・プロシージャまたはファンクションを実行したり、任意のパブリック・パッケージ変数の値をアクセスまたは変更できます。パッケージの構成メンバーに対して特定の EXECUTE 権限は付与できません。したがって、データベース・アプリケーションのプロシージャ、ファンクションおよびパッケージを開発する場合は、セキュリティの確立に関して 2 つの選択肢を考慮してください。これらの選択肢について、次に示す例で具体的に説明します。

パッケージとパッケージ・オブジェクト : 例 1 この例では、2 つのパッケージの本体の中に 4 つのプロシージャを作成します。

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO employees . . .
    END hire;
  PROCEDURE fire(...) IS
```

```
    BEGIN
      DELETE FROM employees . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE employees SET salary = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE employees SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

このプロシージャを実行するためのアクセス権限を付与するには、次の文を使用して、パッケージに対する EXECUTE 権限を付与します。

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

つまり、EXECUTE 権限はパッケージに対して付与されるため、これによってパッケージ・オブジェクト全体にわたる一様なアクセスが提供されます。

パッケージとパッケージ・オブジェクト : 例 2 この例では、単一のパッケージ本体にある 4 つのプロシージャ定義を示します。さらに 2 つのスタンドアロン・プロシージャと 1 つのパッケージを作成し、メイン・パッケージ内で定義したプロシージャへのアクセスを提供します。

```
CREATE PACKAGE BODY employee_changes AS
  PROCEDURE change_salary(...) IS BEGIN ... END;
  PROCEDURE change_bonus(...) IS BEGIN ... END;
  PROCEDURE insert_employee(...) IS BEGIN ... END;
  PROCEDURE delete_employee(...) IS BEGIN ... END;
END employee_changes;

CREATE PROCEDURE hire
  BEGIN
    employee_changes.insert_employee(...)
  END hire;

CREATE PROCEDURE fire
  BEGIN
    employee_changes.delete_employee(...)
  END fire;
```

```
PACKAGE raise_bonus IS
  PROCEDURE give_raise(...) AS
    BEGIN
      employee_changes.change_salary(...)
    END give_raise;

  PROCEDURE give_bonus(...)
    BEGIN
      employee_changes.change_bonus(...)
    END give_bonus;
```

この方法を使用すると、実際に作業を実行するプロシージャ（employee_changes パッケージ内のプロシージャ）は 1 つのパッケージ内で定義され、宣言されたグローバル変数やカーソルなどを共有できます。トップ・レベルのプロシージャの hire と fire、および追加のパッケージ raise_bonus を宣言することにより、選択的な EXECUTE 権限をメイン・パッケージ内のプロシージャに対して付与できます。

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

型のセキュリティ

この項では、型、メソッドおよびオブジェクトに対する権限について説明します。

名前付きの型に対するシステム権限

Oracle では、名前付きの型（オブジェクト型、VARRAY およびネストした表）について、表 23-1 のシステム権限が定義されています。

表 23-1 名前付きの型に対するシステム権限

権限	許可される操作
CREATE TYPE	自分のスキーマ内で名前付きの型を作成します。
CREATE ANY TYPE	任意のスキーマ内で名前付きの型を作成します。
ALTER ANY TYPE	任意のスキーマ内で名前付きの型を変更します。
DROP ANY TYPE	任意のスキーマ内で名前付きの型を削除します。
EXECUTE ANY TYPE	任意のスキーマ内で名前付きの型を使用および参照します。

CONNECT ロールと RESOURCE ロールには、CREATE TYPE システム権限が含まれています。DBA ロールには、これらの権限すべてが含まれています。

オブジェクト権限

名前付きの型に適用されるオブジェクト権限は、EXECUTE のみです。名前付きの型に対する EXECUTE 権限があれば、ユーザーはその型を使用して次の操作を実行できます。

- 表を定義します。
- リレーショナル表に列を定義します。
- 名前付きの変数またはパラメータを宣言します。

EXECUTE 権限により、ユーザーは、型コンストラクタを含め、その型のメソッドを起動できます。これは、ストアド PL/SQL プロシージャに対する EXECUTE 権限と同じです。

メソッド実行モデル

メソッドの実行は、他のストアド PL/SQL プロシージャと同じです。

関連項目： 23-8 ページ「プロシージャのセキュリティ」

型の作成と型を使用した表の作成に必要な権限

型を作成するには、次の要件を満たしている必要があります。

- 自分のスキーマ内で型を作成するには CREATE TYPE システム権限が、他のユーザーのスキーマ内で型を作成するには CREATE ANY TYPE システム権限が必要です。この 2 つの権限は、明示的に、またはロールを介して取得できます。
- 型の所有者には、その型の定義で参照されている他のすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要があります。所有者が、ロールを介して必要な権限を取得することはできません。
- 型の所有者が他のユーザーに型へのアクセス権を付与する場合は、GRANT OPTION の参照された型への EXECUTE 権限、または ADMIN OPTION の EXECUTE ANY TYPE システム権限が必要になります。どちらかの権限がない場合は、権限不足のため、型の所有者は型へのアクセス権を他のユーザーに付与できません。

型を使用して表を作成するには、表の作成要件のみでなく、次の要件を満たす必要があります。

- 表の所有者には、その表で参照されているすべての型にアクセスするための EXECUTE オブジェクト権限が明示的に付与されているか、EXECUTE ANY TYPE システム権限が付与されている必要があります。所有者が、ロールを介して必要な権限を取得することはできません。
- 表の所有者が他のユーザーに表へのアクセス権を付与する場合は、GRANT OPTION の参照された型への EXECUTE 権限、または ADMIN OPTION の EXECUTE ANY TYPE システム権限が必要になります。どちらかの権限がない場合は、権限不足のため、表の所有者は型へのアクセス権を他のユーザーに付与できません。

関連項目： 表の作成要件については、23-6 ページの「表のセキュリティ」を参照してください。

型の作成と型を使用した表の作成に必要な権限の例

CONNECT および RESOURCE ロールを持つユーザーが存在するとします。

- user1
- user2
- user3

User1 は、自分のスキーマで次の DDL を実行します。

```
CREATE TYPE type1 AS OBJECT (
  attr1 NUMBER);

CREATE TYPE type2 AS OBJECT (
  attr2 NUMBER);

GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

User2 は、自分のスキーマで次の DDL を実行します。

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT (
  attr3 user1.type2);
CREATE TABLE tab2 (
  col1 user1.type2);
```

次の文は、正常に実行されます。user2 は、user1 の TYPE2 に対して GRANT OPTION 付きの EXECUTE 権限を持っているためです。

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT on tab2 TO user3;
```

ただし、次の権限付与は正しく実行されません。user2 は、user1 の TYPE1 に対して GRANT OPTION 付きの EXECUTE を持っていないためです。

```
GRANT SELECT ON tab1 TO user3;
```

User3 は、次の文を正常に実行できます。

```
CREATE TYPE type4 AS OBJECT (
  attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

型アクセスおよびオブジェクト・アクセスの権限

DML 文について列レベルと表レベルの権限が存在する場合は、列オブジェクトと行オブジェクトの両方に適用されます。Oracle では、オブジェクト表について、表 23-2 の権限が定義されています。

表 23-2 オブジェクト表に対する権限

権限	許可される操作
SELECT	表からオブジェクトとその属性にアクセスします。
UPDATE	表の行を構成するオブジェクトの属性を変更します。
INSERT	表に新規オブジェクトを作成します。
DELETE	行を削除します。

同様に、列オブジェクトには表権限と列権限が適用されます。インスタンスを検索するだけでは、型情報は明らかになりません。ただし、クライアントは、型インスタンスのイメージを解釈するために、名前付きの型の情報にアクセスする必要があります。クライアントからこの種の型情報を要求されると、Oracle は型の EXECUTE 権限をチェックします。

次のスキーマを考えてみます。

```
CREATE TYPE emp_type (
    eno NUMBER, ename CHAR(31), eaddr addr_t);
CREATE TABLE emp OF emp_t;
```

さらに、次の 2 つの問合せについて考えます。

```
SELECT VALUE(emp) FROM emp;
SELECT eno, ename FROM emp;
```

どちらの問合せの場合も、Oracle は、emp 表に対するユーザーの SELECT 権限をチェックします。最初の問合せの場合、ユーザーは、emp_type の型情報を取得してデータを解釈する必要があります。問合せによって emp_type 型がアクセスされると、Oracle はユーザーの EXECUTE 権限をチェックします。

ただし、2 番目の問合せを実行しても、名前付きの型が含まれていないため、Oracle は型の権限をチェックしません。

さらに、前の項のスキーマを使用して、user3 は、次の問合せを実行できます。

```
SELECT tab1.col1.attr2 FROM user2.tab1 tab1;
SELECT attr4.attr3.attr2 FROM tab3;
```

user3 は基礎となる型に対する明示的な権限を持っていませんが、USER3 によるどちらの SELECT 文も成功するので注意してください。型および表の所有者が、GRANT OPTION 付きの必要な権限を持っているためです。

Oracle は、次のイベントに対する権限をチェックし、クライアントがそのアクションに必要な権限を持っていなければエラーを戻します。

- オブジェクトの REF 値を使用してオブジェクト・キャッシュ内でオブジェクトを確保するときには、Oracle は、そのオブジェクトを含んでいるオブジェクト表に対する SELECT 権限をチェックします。
- 既存のオブジェクトを修正したり、オブジェクト・キャッシュからオブジェクトをフラッシュするときには、Oracle は目的のオブジェクト表に対する UPDATE 権限をチェックします。
- 新しいオブジェクトをフラッシュするときには、Oracle は目的のオブジェクト表に対する INSERT 権限をチェックします。
- オブジェクトを削除するときには、Oracle は目的の表に対する DELETE 権限をチェックします。
- 名前付きの型のオブジェクトを確保するときには、Oracle はそのオブジェクトに対する EXECUTE 権限をチェックします。

クライアントの 3GL アプリケーション内でオブジェクトの属性を変更するときには、Oracle はオブジェクト全体を更新します。したがって、ユーザーにはオブジェクト表に対する UPDATE 権限が必要です。アプリケーションがオブジェクト表の特定の列に対応する属性を変更する場合でも、その列のみに対する UPDATE 権限では不十分です。したがって、Oracle は、オブジェクト表に対する列レベルの権限はサポートしていません。

型の依存性

プロシージャや表などのストアド・オブジェクトと同様に、他のオブジェクトから参照される型を、依存性があると呼びます。表が型に依存する特殊な問題点がいくつかあります。アクセス時に型定義に依存するデータが表に含まれている場合は、その型を変更すると、その表に格納されているすべてのデータにアクセスできなくなります。変更によってこのような結果になるのは、型に必要な権限が取り消される場合や、型または依存型が削除される場合です。このどちらかのアクションが発生すると、表は無効になり、アクセスできなくなります。

必要な権限が再び付与されると、権限がないために無効になった表は自動的に有効になり、アクセスできるようになります。依存型が削除されていたために無効になった表は、アクセス不能のままで、実行できるアクションは削除のみです。

型に対する権限を取り消したり型を削除すると重大な影響が生じるため、デフォルトでは SQL 文 REVOKE および DROP TYPE の実装は意図的に限定されています。つまり、どちらの文でも、名前付きの型に表または型が依存していると、エラーが戻されて文は異常終了します。ただし、どちらの文も、FORCE 句を使用すると常に正常終了し、表が依存している場合はその表は無効になります。

> **関連項目：** REVOKE、DROP TYPE および FORCE 句の使用方法の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

ロールの概要

Oracle では、ロールを使用することで簡単かつ制御された権限管理を実現できます。**ロール**は、関連する権限のグループに名前を付けたもので、ユーザーまたは他のロールに付与します。ロールは、エンド・ユーザーのシステム権限とスキーマ・オブジェクト権限を容易に管理できるように設計されています。ただし、ロールは、アプリケーション開発者が使用することを目的としたものではありません。ストアド・プログラム構成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からです。

ロールの次のような特性によって、データベース内での権限管理がさらに容易になります。

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一の権限セットを明示的に付与するかわりに、関連するユーザー・グループのための権限をまとめて 1 つのロールに付与しておき、そのグループの各メンバーにはそのロールを付与するだけですみます。
動的な権限管理	あるグループの権限の変更が必要な場合、そのロールの権限を修正するだけですみます。グループのロールを付与した全ユーザーのセキュリティ・ドメインには、そのロールに対して加えられる変更が自動的に反映されます。
権限の選択的な可用性	あるユーザーに付与したロールを、選択的に使用可能または使用禁止にできます。この機能によって、どのような状況でもユーザー権限を個々に制御できます。
アプリケーションによる認識	ロールの存在はデータ・ディクショナリに記録されます。したがって、ユーザーが特定のユーザー名でアプリケーションを実行したときに、アプリケーションがディクショナリに問い合せ、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：

- プロシージャに関する制限の詳細は、23-22 ページの「データ定義言語の文とロール」を参照してください。
- アプリケーションからロールを使用可能にする手順は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

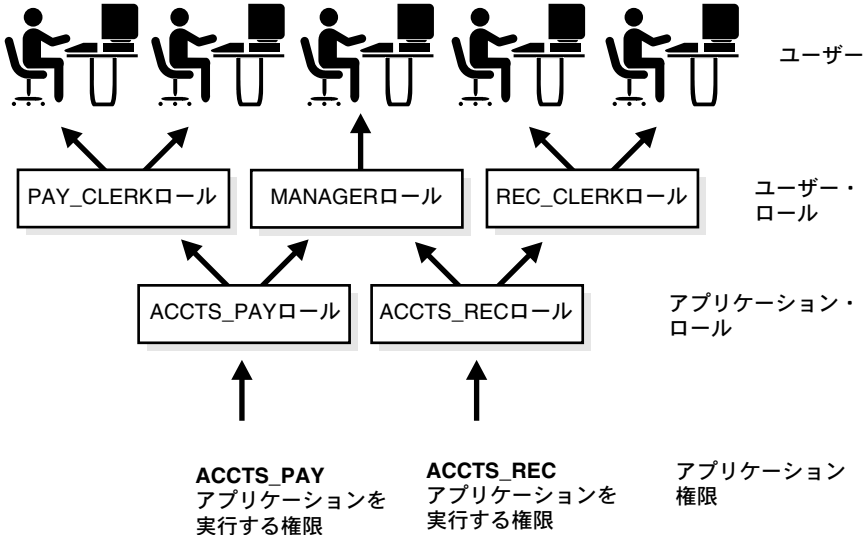
ロールの一般的な使用方法

通常は、次のどちらかの目的でロールを作成します。

- データベース・アプリケーションに対する権限の管理
- ユーザー・グループに対する権限の管理

図 23-1 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 23-1 ロールの一般的な使用方法



アプリケーション・ロール

アプリケーション・ロールには、特定のデータベース・アプリケーションを実行するために必要な権限をすべて付与します。そして、その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 では、データベース・アプリケーションとユーザーがロールを使用可能または使用禁止にできるため、権限を選択的に使用できます。
- 間接的に付与されたロールとは、ロールに対して付与されたロールです。この種のロールは、ユーザーに対して明示的に使用可能または使用禁止にすることができます。ただし、別のロールを含んだロールを使用可能にすることによって、直接的に付与されたロールに含まれる間接的に付与された全ロールは、すべて暗黙のうちに使用可能になります。

ロールの付与と取消し

ユーザーや別のロールに対してロールを付与したり取り消すには、次の方法があります。

- Oracle Enterprise Manager
- SQL 文 GRANT および REVOKE

ロールに対して権限を付与または取り消す場合にも同じオプションを使用します。また、ロールは、Oracle を実行しているオペレーティング・システムや、ネットワーク・サービスを使用することによっても、ユーザーに対して付与したり取り消したりできます。

> **関連項目：** ロール管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

ロールの付与と取消しを実行できるユーザー

GRANT ANY ROLE システム権限を付与されたユーザーは、グローバル・ロールを除き、他のユーザーの任意のロールまたはデータベースのロールの付与または取消しを実行できます。このシステム権限は非常に強力であるため、付与するときは注意が必要です。

ADMIN OPTION 付きでロールを付与されたユーザーは、データベースの他のユーザーやロールに対してロールを付与したりそのロールを取り消すことができます。つまり、このオプションにより、選択的なロールの管理が可能になります。

> **関連項目：** グローバル・ロールについては、『Oracle9*i* データベース管理者ガイド』を参照してください。

ロール名

各ロール名はデータベース内で一意にする必要があり、ユーザー名とロール名を同一にすることはできません。スキーマ・オブジェクトとは異なり、ロールはスキーマに含まれているわけではありません。そのため、ロールを作成したユーザーを削除しても、そのロールに影響はありません。

ロールとユーザーのセキュリティ・ドメイン

各ロールと各ユーザーは、それぞれ独自のセキュリティ・ドメインを持っています。ロールのセキュリティ・ドメインには、ロールそのものに付与されている権限と、そのロールに付与されたロールに対して付与されている権限が含まれます。

ユーザーのセキュリティ・ドメインには、対応するスキーマ内のすべてのスキーマ・オブジェクトの権限、そのユーザーに対して付与されている権限、およびそのユーザーに対して付与されていて**現在使用可能**になっているロールの権限が含まれます。(1 つのロールをあるユーザーに対して使用可能にし、別のユーザーに対しては使用禁止にすることもできます。)さらに、ユーザーのセキュリティ・ドメインには、ユーザー・グループ PUBLIC に対して付与されている権限とロールも含まれます。

PL/SQL ブロックとロール

PL/SQL ブロック内でのロールの使用方法は、それが無名ブロックであるか名前付きブロック（ストアド・プロシージャ、ファンクションまたはトリガー）であるか、および定義者権限と実行者権限のどちらで実行されるかに応じて異なります。

定義者権限を持つ名前付きブロック

定義者権限で実行される名前付き PL/SQL ブロック（ストアド・プロシージャ、ファンクションまたはトリガー）では、すべてのロールは使用禁止になっています。ロールは権限チェックに使用されず、定義者権限プロシージャ内ではロールを設定できません。

SESSION_ROLES ビューは、現在使用可能になっているすべてのロールを示します。定義者権限で実行される名前付き PL/SQL ブロックが SESSION_ROLES を問い合せると、問合せは行を戻しません。

関連項目：『Oracle9*i* データベース・リファレンス』

実行者権限と無名ブロック

実行者権限で実行される名前付き PL/SQL ブロックと、無名 PL/SQL ブロックは、使用可能になっているロールを通じて付与された権限に基づいて実行されます。実行者権限を持つ PL/SQL ブロック内での権限チェックにはカレント・ロールが使用され、動的 SQL を使用してセッション中にロールを設定できます。

関連項目：

- 実行者権限と定義者権限の説明は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
- 14-19 ページ「PL/SQL の動的 SQL」

データ定義言語の文とロール

ユーザーがデータ定義言語（DDL）文を正常に実行するには、その文に応じて 1 つ以上の権限が必要になります。たとえば、表を作成するには、CREATE TABLE または CREATE ANY TABLE システム権限が必要です。別のユーザーの表のビューを作成するには、CREATE VIEW または CREATE ANY VIEW システム権限のみでなく、その表に対する SELECT オブジェクト権限または SELECT ANY TABLE システム権限も必要です。

Oracle では、特定の DDL 文での特定の権限の使用を制限することによって、ロールを介して受け取った権限への依存性を回避します。次の規則は、DDL 文に対する権限の制限を示しています。

- DDL 操作の実行をユーザーに許可するすべてのシステム権限およびスキーマ・オブジェクト権限は、ロールを介して受け取った場合でも使用可能。たとえば、次のようなものがあります。
  - システム権限 : CREATE TABLE、CREATE VIEW および CREATE PROCEDURE 権限。
  - スキーマ・オブジェクト権限 : 表に対する ALTER および INDEX の各権限。

  例外 : 表に対する REFERENCES オブジェクト権限は、それがロールを介して付与された場合には、表の外部キー定義には使用できません。
- DDL 文の発行に必要な DML 操作の実行をユーザーに許可するすべてのシステム権限およびオブジェクト権限は、ロールを介して受け取った場合は使用不可。次に例を示します。
  - 表に対する SELECT ANY TABLE システム権限や SELECT オブジェクト権限がロールを介して付与されている場合、それらの権限を使用して別のユーザーの表に基づくビューを作成することはできません。

ロールを介して受け取った権限の使用許可と使用制限について、次の例で具体的に説明します。

次のようなユーザーを想定します。

- CREATE VIEW システム権限を持つロールが付与されています。
- employees 表の SELECT オブジェクト権限を含むロールが付与されていますが、この employees 表の SELECT オブジェクト権限は間接的に付与されています。
- departments 表に対する SELECT オブジェクト権限が直接付与されています。

前述の権限がこのユーザーに直接的および間接的に付与されているとすると、

- このユーザーは employees 表と departments 表の両方に対して SELECT 文を発行できます。

- このユーザーには employees 表の CREATE VIEW 権限と SELECT 権限がロールを介して付与されていますが、employees 表の SELECT オブジェクト権限がロールを介して付与されているため、employees 表に基づく使用可能なビューは作成できません。作成されたビューにアクセスすると、エラーになります。
- CREATE VIEW 権限がロールを介して付与され、departments 表の SELECT 権限が直接付与されているため、departments 表のビューは作成できます。

事前定義済みのロール

次のロールは、Oracle データベースに対して自動的に定義されます。

- CONNECT
- RESOURCE
- DBA
- EXP_FULL_DATABASE
- IMP_FULL_DATABASE

これらのロールは、旧バージョンの Oracle との下位互換のために提供されており、Oracle データベース内の他のロールと同じ方法で変更できます。

オペレーティング・システムとロール

環境によっては、オペレーティング・システムでデータベース・セキュリティを管理できる場合もあります。オペレーティング・システムを使用して、データベース・ロールの付与と取消しや、パスワードの認証を管理できます。この機能は、すべてのオペレーティング・システムで利用できるとは限りません。

関連項目： オペレーティング・システムによるロールの管理方法の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

分散環境におけるロール

分散データベース環境でロールを使用する場合は、必要なすべてのロールを分散（リモート）セッションのデフォルト・ロールとして設定する必要があります。ローカル・データベース・セッション内からリモート・データベースに接続しているときは、ロールを使用可能にできません。たとえば、リモート・サイトでロールを使用可能にしようとするリモート・プロシージャは実行できません。

関連項目： 『Oracle9*i* Heterogeneous Connectivity 管理者ガイド』

ファイングレイン・アクセス・コントロール

ファイングレイン・アクセス・コントロールにより、ファンクションにセキュリティ・ポリシーを適用し、セキュリティ・ポリシーを表、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 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を表します。

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、作成したポリシーを追加、削除、使用可能と使用禁止の切替えおよびリフレッシュを行います。

関連項目：

- パッケージのインプリメンテーションの詳細は、『Oracle9*i* 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。
- セキュリティ・ポリシーの設定方法と例は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

動的な述語

作成したセキュリティ・ポリシーを実現するファンクションまたはパッケージは、述語（WHERE 条件）を戻します。この述語によって、ポリシーで設定されたとおりにアクセスが制御されます。再書込みされた問合せは、完全に最適化され、共有可能になります。

アプリケーション・コンテキスト

アプリケーション・コンテキストにより、ファイングレイン・アクセス・コントロールの実現が容易になります。また、ファンクションにセキュリティ・ポリシーを実装し、そのセキュリティ・ポリシーをアプリケーションに対応付けることができます。各アプリケーションには、固有のコンテキストを対応付けることができます。ユーザーは、そのコンテキストを任意に（SQL*Plus などを通じて）変更することは許されません。

アプリケーション・コンテキストにより、アプリケーションに関係する属性に基づき、パラメータベースの柔軟なアクセス制御が可能になります。たとえば、人事管理アプリケーションのコンテキスト属性には「職位」、「部門」および「国」などを含め、受注管理アプリケーションの属性には「顧客番号」や「営業地区」などを含めることができます。

次のことができます。

- コンテキスト値の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。

アプリケーション・コンテキストを定義する手順

1. アプリケーションのコンテキストを検査して設定する関数を使用して、PL/SQL パッケージを作成します。ログイン時にイベント・トリガーを使用して、ログイン・ユーザーの初期コンテキストを設定できます。
2. `CREATE CONTEXT` を使用して一意のコンテキスト名を指定し、作成した PL/SQL パッケージに対応付けます。
3. 次のどちらかの操作を行います。
   - ファイングレイン・アクセス・コントロールを実装するポリシー関数内で、アプリケーション・コンテキストを参照します。
   - ログイン時にイベント・トリガーを使用して、ユーザーの初期コンテキストを設定します。たとえば、ユーザーの従業員番号を問い合せて、「従業員番号」コンテキスト値として設定できます。
4. アプリケーション・コンテキストを参照します。

関連項目：

- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle9*i* 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』
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

保護アプリケーション・ロール

Oracle 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。

以前のリリースでは、パスワードはソース・コードに埋め込まれるか、または表に格納されていました。このリリースでは、アプリケーション開発者はパスワードをアプリケーション内に埋め込んでロールを保護する必要がなくなります。そのかわり、保護アプリケーション・ロールを作成して、ロールの有効化を認可する PL/SQL パッケージを指定します。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。アプリケーションはロールを有効化する前に、認証を実行します。

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をチェックするなどの、カスタマイズ認可を実行できます。

注意： 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。

保護アプリケーション・ロールの作成

保護アプリケーション・ロールは、CREATE ROLE ... IDENTIFIED USING 文を使用して作成します。次に例を示します。

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

この例から次のようなことがわかります。

- 作成される admin_role は、保護アプリケーション・ロールです。
- このロールは、PL/SQL パッケージ hr.admin 内で定義されたモジュールによってのみ有効化できます。

この文を実行するには、システム権限 CREATE ROLE が必要になります。

実行者権限プロシージャ内で有効化されたロールは、プロシージャの終了後も有効になります。そのため、ロールの有効化を扱う専用プロシージャを作成し、セッションの終了まで使用することができます。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle9*i* 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』
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

24

監査

この章では、Oracle の監査機能について説明します。この章の内容は、次のとおりです。

監査の概要

監査とは、選択したユーザー・データベース・アクションを監視して記録する処理のことです。監査は、通常次のような目的で使用されます。

- 動作が不審なアクティビティの調査。たとえば、無許可ユーザーが表からデータを削除しようとした場合、セキュリティ管理者は、そのデータベースへのすべての接続と、そのデータベースにあるすべての表からの行の削除（成功および失敗）をすべて監査できます。
- 特定のデータベース・アクティビティに関するデータの監視と収集。たとえば、データベース管理者は、更新された表、実行された論理 I/O の回数、またはピーク時に接続していた同時実行ユーザーの数などに関する統計を収集できます。

監査の機能

ここでは、Oracle の監査メカニズムの概要について説明します。

監査のタイプ

Oracle は次の 3 つのタイプの一般監査機能をサポートします。

監査のタイプ	説明
文監査	SQL 文が処理する特定のスキーマ・オブジェクトではなく、文のタイプに関してのみ実行する SQL 文の選択的な監査。文監査のオプションは通常、適用範囲が広く、オプションごとに何種類かの関連したアクションの使用方法を監査します。たとえば、`AUDIT TABLE` は、それがどの表に対して発行されたかには関係なく、いくつかの DDL 文を追跡します。文監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
権限監査	`AUDIT CREATE TABLE` など、システム・アクションを実行する強力なシステム権限の使用方法を監査する選択的な監査。権限監査は、対象となる権限の使用方法のみを監査するため、監査対象が文監査より限定されています。権限監査は、データベースの選択したユーザーまたはすべてのユーザーを監査するように設定できます。
スキーマ・オブジェクト監査	`AUDIT SELECT ON 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選択的な監査。スキーマ・オブジェクト監査は対象が非常に限定されており、特定のスキーマ・オブジェクトに対する特定の文のみを監査します。スキーマ・オブジェクト監査は、データベースのすべてのユーザーに常に適用されます。
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。

監査の対象

Oracle では、監査オプションの対象範囲を広くしたり狭くできます。次の監査ができます。

- 文の正常な実行、失敗した実行、またはその両方
- ユーザー・セッションごと、または文が実行されるごとの文の実行
- すべてのユーザーまたは特定のユーザーのアクティビティ

監査レコードと監査証跡

監査レコードには、監査された操作、操作を実行したユーザーおよび操作の日時などの情報が記録されます。監査レコードは、**データベース監査証跡**と呼ばれるデータ・ディクショナリ表か、オペレーティング・システムの監査証跡のどちらかに格納できます。

データベース監査証跡は、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
- セッション識別子
- 端末識別子
- アクセスされたスキーマ・オブジェクトの名前
- 実行または試行された操作
- 操作の完了コード
- 日時のタイム・スタンプ
- 使用されたシステム権限

オペレーティング・システム監査証跡はコード化されていて読むことはできませんが、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。これらのコードとその記述のリストは、AUDIT_ACTIONS データ・ディクショナリ表にあります。
- **使用された権限**は、操作の実行に使用されたシステム権限の記述です。これらのコードとその記述は、SYSTEM_PRIVILEGE_MAP 表にあります。
- **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 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、『Oracle9*i* データベース管理者ガイド』を参照してください。
- 完了コードのリストは、『Oracle9*i* データベース・エラー・メッセージ』を参照してください。

監査機能のメカニズム

この項では、Oracle の監査機能で使用されているメカニズムについて説明します。

監査レコードが生成される場合

監査情報の記録機能は、使用可能または使用禁止にできます。この機能により、許可されたデータベース・ユーザーは監査オプションをいつでも設定できますが、監査情報の記録を制御する操作はセキュリティ管理者のために確保されています。

データベースの監査機能が使用可能になっている場合、監査レコードは文実行の実行フェーズで生成されます。

PL/SQL プログラム・ユニット内の SQL 文は、プログラム・ユニットの実行時に必要に応じて監査されます。

監査証跡レコードの生成と挿入は、ユーザーのトランザクションからは独立して実行されます。そのため、ユーザーのトランザクションがロールバックされても、監査証跡レコードはコミットされたままになります。

注意： 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から分離された 位置への送信により、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、『Oracle9*i* データベース管理者ガイド』を参照してください。
- SQL 文処理の様々なフェーズや共有 SQL については、第 14 章「SQL、PL/SQL および Java」を参照してください。

オペレーティング・システムの監査証跡に必ず記録されるイベント

データベース監査が使用可能であるかどうかに関係なく、Oracle はある種のデータベース関連アクションをオペレーティング・システムの監査証跡に常に記録します。

- インスタンス起動時には、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も記録されるため、管理者が、データベース監査使用禁止の状態のままデータベースを再起動して、監査されないアクションを実行できないようになっています。
- インスタンス停止時には、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、Oracle に管理者権限によって接続したオペレーティング・システム・ユーザーの詳細を記述した監査レコードが生成されます。この監査レコードにより、管理者権限によって接続したユーザーに関する情報が提供されます。

監査証跡が Oracle からはアクセスできるようになっていないオペレーティング・システムの場合、これらの監査証跡レコードは、バックグラウンド・プロセスのトレース・ファイルと同じディレクトリにある Oracle 監査証跡ファイルに格納されます。

> **関連項目：** オペレーティング・システムの監査証跡の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

監査オプションが有効になるタイミング

データベース・ユーザーがデータベースに接続した時点で有効になっていた文監査オプションと権限監査オプションは、そのセッションの持続期間中は有効です。文監査または権限監査のオプションの設定または変更の結果は、セッション中には有効になりません。修正した文監査オプションまたは権限監査オプションは、カレント・セッションを終了し、新しいセッションを作成する時点で有効になります。一方、オブジェクト監査オプションについて変更した内容は、カレント・セッションでただちに有効になります。

分散データベースの監査

監査機能には、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 ノードは、リモート・データベースで発生するアクションを監査できません。リモート接続はデータベース・リンクのユーザー・アカウントを介して確立されるため、リモート Oracle ノードはデータベース・リンクの接続を介して発行された文を監査します。

> **関連項目：** 『Oracle9*i* データベース管理者ガイド』

オペレーティング・システム監査証跡に対する監査

オペレーティング・システムの監査証跡が使用可能になっている場合に、Oracle は監査証跡レコードをオペレーティング・システムの監査証跡に送ることができます。その他のオペレーティング・システムの場合、これらの監査レコードが、他の Oracle トレース・ファイルと似た形式でデータベース外のファイルに書き込まれることもあります。

> **関連項目：** この機能がオペレーティング・システムで実装されているかどうかを調べるには、オペレーティング・システム固有の Oracle マニュアルを参照してください。

Oracle では、オペレーティング・システムの監査証跡（または監査レコードを含むオペレーティング・システム・ファイル）に監査レコードを記録できない場合にも、常に特定のアクションを引き続き監査できます。オペレーティング・システムの監査証跡に記録できないのは、多くの場合、オペレーティング・システムの監査証跡またはファイル・システムがいっぱいになっており、新しいレコードが入らないことが原因です。

オペレーティング・システム監査を構成するシステム管理者は、監査証跡またはファイル・システムがいっぱいにならないようにしてください。ほとんどのオペレーティング・システムでは、このような状況を回避できるように十分な情報と警告が管理者に提供されます。ただし、データベースの監査証跡を使用するように監査を構成すれば、その危険性を回避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 サーバーによって防止されるためです。

文監査

文監査とは、関連する文のグループに対する選択的な監査のことです。文は、次の 2 つに分類できます。

- 特定のタイプのデータベース構造やスキーマ・オブジェクトに関する DDL 文。ただし、監査の対象として特定の構造やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT TABLE はすべての CREATE TABLE 文と DROP TABLE 文を監査します）。
- 特定のタイプのデータベース構造やスキーマ・オブジェクトに関する DML 文。ただし、監査の対象として特定の構造やスキーマ・オブジェクトを指定することはできません（たとえば、AUDIT SELECT TABLE は、表やビューとは無関係に、すべての SELECT ... FROM TABLE/VIEW 文を監査します）。

文監査では、監査の対象範囲を広げてすべてのデータベース・ユーザーのアクティビティを監査したり、範囲を限定して特定のデータベース・ユーザーのアクティビティのみを監査できます。

権限監査

権限監査は、システム権限の使用を許可されている文を選択的に監査します。たとえば、SELECT ANY TABLE システム権限の監査は、SELECT ANY TABLE システム権限を使用して実行されるユーザーの文を監査します。任意のシステム権限の使用を監査できます。

権限監査では、システム権限の監査の前に、所有者権限とスキーマ・オブジェクト権限が必ず検査されます。所有者権限とスキーマ・オブジェクト権限がアクションを許可するのに十分である場合、そのアクションは監査されません。

類似の文監査オプションと権限監査オプションを両方設定しても、監査レコードは 1 つしか生成されません。たとえば、文の句 TABLE とシステム権限の CREATE TABLE の両方を監査すると、表が作成されるたびに監査レコードが 1 つのみ生成されます。

権限監査のそれぞれのオプションでは、特定のタイプの文のみが監査され、関連する一連の文が監査されるわけではないため、権限監査の対象は文監査よりも限定されます。たとえば、文監査句 TABLE では CREATE TABLE、ALTER TABLE および DROP TABLE 文が監査されますが、権限監査オプション CREATE TABLE では CREATE TABLE 文のみが監査されます。これは、CREATE TABLE 権限を必要とするのが CREATE TABLE 文のみであるためです。

文監査と同様に、権限監査では、すべてのデータベース・ユーザーのアクティビティを監査したり、特定のデータベース・ユーザーのアクティビティを監査できます。

スキーマ・オブジェクト監査

スキーマ・オブジェクト監査は、特定のスキーマ・オブジェクトの特定の DML 文（問合せを含む）、および GRANT 文と REVOKE 文の選択的な監査です。スキーマ・オブジェクト監査では、特定の表に対する SELECT 文や DELETE 文など、スキーマ・オブジェクト権限によって許可される操作と、それらの権限を制御する GRANT 文と REVOKE 文が監査されます。

表、ビュー、順序、**スタンドアロン**のストアド・プロシージャとストアド・ファンクションおよびパッケージを参照する文を監査できます。パッケージ内のプロシージャは、個別には監査できません。

クラスタ、データベース・リンク、索引またはシノニムを参照する文は、直接監査できません。ただし、実表に影響を与える操作を監査することにより、これらのスキーマ・オブジェクトへのアクセスを間接的に監査できます。

スキーマ・オブジェクト監査オプションは、常にすべてのデータベース・ユーザーに対して設定されます。これらのオプションは、特定のユーザーのリストに対しては設定できません。監査可能なすべてのスキーマ・オブジェクトに対して、デフォルトのスキーマ・オブジェクト監査オプションを設定できます。

> **関連項目：** 監査できるスキーマ・オブジェクトの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ビューとプロシージャのスキーマ・オブジェクト監査オプション

ビューとプロシージャ（ストアド・ファンクション、パッケージおよびトリガーを含む）は、その定義の基礎を形成するスキーマ・オブジェクトを参照します。そのため、ビューとプロシージャに関する監査には、いくつかの固有の特性があります。ビューやプロシージャを使用すると、その結果として複数の監査レコードが生成されます。ビューやプロシージャの使用は、使用可能になっている監査オプションに依存します。ビューやプロシージャを使用した結果として発行される SQL 文は、ベース・スキーマ・オブジェクトの使用可能監査オプション（デフォルトの監査オプションも含む）に依存します。

次の一連の SQL 文について考えます。

```
AUDIT SELECT ON employees;

CREATE VIEW employees_departments AS
  SELECT employee_id, last_name, department_id
    FROM employees, departments
    WHERE employees.department_id = departments.department_id;

AUDIT SELECT ON employees_departments;

SELECT * FROM employees_departments;
```

この employees_departments に対する問合せの結果、2 つの監査レコードが生成されます。1 つは employees_departments ビューの問合せについての監査レコード、もう 1 つ

は実表 employees の問合せ（employees_departments ビューを介した間接的な問合せ）についての監査レコードです。実表の SELECT 監査オプションが使用可能になっていないため、実表 departments に対して問合せを実行しても監査レコードは生成されません。すべての監査レコードは、employees_departments ビューの問合せを発行したユーザーに属します。

ビューやプロシージャの監査オプションは、そのビューまたはプロシージャを最初に使用して共有プールに配置する時点で、判別されます。そのビューまたはプロシージャをいったんフラッシュして共有プールに再配置するまで、これらの監査オプションは設定されたままになります。スキーマ・オブジェクトを監査すると、キャッシュ内のスキーマ・オブジェクトが無効になるため、スキーマ・オブジェクトを再ロードすることになります。ベース・スキーマ・オブジェクトの監査オプションに対する変更は、共有プール内のビューとプロシージャには認識されません。

前述の例の続きとして、employees 表に対する SELECT 文の監査をオフにすると、employees_departments ビューを使用しても employees 表の監査レコードは生成されなくなります。

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。データベースのビルトインの監査メカニズムにより、ユーザーは監査を回避できません。Oracle のトリガーでは、INSERT、UPDATE および DELETE などの DML 処理を監視することもできます。ただし、SELECT で監視を行うとコストもかかり、場合によっては有効に機能しないこともあります。また、ユーザーは単に監査証跡に監査レコードを挿入するだけでなく、独自の警告アクションを定義する場合もあります。この機能は、表やビューの SELECT 文を監査する、拡張可能なインタフェースを提供します。

DBMS_FGA パッケージは、このような値ベースの監査方針を管理します。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。問合せブロックから戻された行のいずれかが監査条件に一致した場合（その行は**関連**行として参照されます）、ユーザー名、SQL テキスト、バインド変数、ポリシー名、セッション ID、タイム・スタンプ、およびその他の属性を含む監査イベント・エントリが、監査証跡に挿入されます。拡張フレームワークの一環として、管理者はイベントの処理のためにオプションで適切なイベント・ハンドラ、**監査イベント・ハンドラ**を定義することもできます。たとえば、監査イベント・ハンドラは管理者に警告ページを送ります。

関連項目：『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

文、権限およびスキーマ・オブジェクトの監査対象の限定

Oracle では、文、権限およびスキーマ・オブジェクトのそれぞれの監査の対象を、次の 3 つの分野に限定できます。

- 監査される SQL 文の正常な実行と失敗した実行
- BY SESSION 監査と BY ACCESS 監査
- データベースの特定のユーザーまたはすべてのユーザー（文監査と権限監査のみ）

文の正常な実行と失敗した実行の監査

文、権限およびスキーマ・オブジェクトの監査では、文の正常な実行、失敗した実行、またはその両方の文実行を選択的に監査できます。このため、監査する文が成功しなかった場合にも、アクションを監視できます。

失敗した文の実行を監査できるのは、有効な SQL 文を実行しても適切な認可がないために失敗した場合や、存在しないスキーマ・オブジェクトを参照したために失敗した場合のみです。有効でなかったために失敗した文は監査できません。たとえば、失敗した文の実行を監査するように権限監査オプションが設定されている場合は、対象のシステム権限を使用しても他の原因で失敗した文が監査されます（たとえば CREATE TABLE を設定したが、指定した表領域に対する割当て制限を設定していなかったために CREATE TABLE 文が失敗した場合など）。

どちらの形式の AUDIT 文にも、次の句を指定できます。

- WHENEVER SUCCESSFUL 句。監査対象の文の正常な実行のみを監査する場合に使用します。
- WHENEVER NOT SUCCESSFUL 句。監査対象の文の失敗した実行のみを監査する場合に使用します。
- 前述のどちらの句も使用しません。監査対象の文の正常な実行と失敗した実行の両方を監査する場合に使用します。

監査文の BY SESSION 句と BY ACCESS 句

ほとんどの監査オプションでは、1 つのユーザー・セッションで監査対象の文が複数回発行された場合の監査レコードの生成方法を指定できます。ここでは、AUDIT 文の BY SESSION 句と BY ACCESS 句の相違点について説明します。

関連項目：『Oracle9*i* SQL リファレンス』

BY SESSION

どのタイプの監査（スキーマ・オブジェクト、文、権限）の場合も、BY SESSION は、監査されるアクションが含まれているセッション中に、ユーザーおよびスキーマ・オブジェクト当たり 1 つの監査レコードのみを監査証跡に挿入します。

セッションとは、ユーザーが Oracle データベースに接続してから切断するまでの期間です。

BY SESSION 例 1 この例では、次のような状況を想定します。

- SELECT TABLE 文監査オプションを BY SESSION に設定します。
- JWARD でデータベースに接続し、departments という表に対して 5 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。
- SWILLIAMS でデータベースに接続し、employees 表に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

この場合、監査証跡には 8 つの SELECT 文に対して 2 件（SELECT 文を発行したセッションごとに 1 件）の監査レコードが記録されます。

BY SESSION 例 2 別の例として、次の条件を想定します。

- SELECT TABLE 文監査オプションを BY SESSION に設定します。
- JWARD でデータベースに接続し、departments という表に対して 5 つの SELECT 文、表 employees に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

この場合、監査証跡には、ユーザーがセッション中に SELECT 文を発行した各スキーマ・オブジェクトごとに 1 件ずつ、2 件のレコードが記録されます。

注意： オペレーティング・システムの監査証跡に監査レコードを記録する際に BY SESSION 句を使用すると、Oracle ではアクセスするたびに監査レコードが生成され格納されます。したがって、この監査構成の場合、BY SESSION は BY ACCESS と等価です。

BY ACCESS

監査を BY ACCESS に設定すると、カーソル内で監査対象の操作が実行されるたびに、監査証跡に監査レコードが 1 つ挿入されます。カーソルを再利用させるイベントは、次のとおりです。

- カーソルを再利用するためにオープンしておく、Oracle Forms などのアプリケーション
- 新しいバインド変数を使用してカーソルを再実行すること
- PL/SQL ループ内で実行される文で、1 つのカーソルを再利用するために PL/SQL エンジンにより文が最適化される場合

監査は、カーソルが共有されているかどうかの影響を**受けない**ことに注意してください。それぞれのユーザーは、カーソルの最初の実行時に自分の監査証跡レコードを作成します。

たとえば、次のような場合を想定します。

- SELECT TABLE 文監査オプションを BY ACCESS に設定します。
- JWARD でデータベースに接続し、departments という表に対して 5 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。
- SWILLIAMS でデータベースに接続し、departments 表に対して 3 つの SELECT 文を発行した後、そのデータベースとの接続を切断します。

1 つの監査証跡に、8 つの SELECT 文に対応する 8 件のレコードが記録されます。

デフォルトと除外される操作

AUDIT 文には、BY SESSION と BY ACCESS のどちらかを指定できます。ただし、次のような一部の監査オプションでは、BY ACCESS しか設定できません。

- DDL 文を監査するすべての文監査オプション
- DDL 文を監査するすべての権限監査オプション

他のすべての監査オプションでは、デフォルトで BY SESSION が設定されています。

ユーザー別の監査

文監査オプションと権限監査オプションでは、任意のユーザーが発行した文を監査するか、特定のユーザー・リストに含まれるユーザーが発行する文を監査するかを選択できます。特定のユーザーに限定すると、生成される監査レコードの数は最小限に抑えられます。

ユーザー別の監査の例 表やビューを問い合せたり更新するためにユーザー SCOTT および BLAKE によって発行される文を監査するには、次の文を発行します。

```
AUDIT SELECT TABLE, UPDATE TABLE
     BY scott, blake;
```

> **関連項目：** ユーザー別の監査の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

複数層環境における監査

複数層環境では、Oracle はクライアントの識別情報をすべての階層を通して保持します。これにより、クライアントのために実行された処理を監査できます。これを行うには、AUDIT 文で BY **proxy** 句を使用します。

この句を使用すると、オプションで次のことを実行できます。

- 特定のプロキシのために、そのプロキシによって発行された SQL 文の監査
- 特定のユーザーのために実行された文の監査
- ユーザーのために実行されたすべての文の監査

中間層では、データベースのセッションで軽量のユーザー ID を設定し、監査証跡で表示させることができます。クライアントの識別子の設定には、OCI または PL/SQL を使用します。

> **関連項目：**
>
> - 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』
> - 『Oracle Call Interface プログラマーズ・ガイド』
> - 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

A

オペレーティング・システム固有の情報

このマニュアルでは、特定のオペレーティング・システムで Oracle を使用する際の詳細情報について、他の Oracle マニュアルに言及していることがあります。正式名称は実際のオペレーティング・システムによって異なりますが、これらの Oracle マニュアルを**インストレーションおよび構成ガイド**と呼びます。

この付録では、このマニュアル内でオペレーティング・システム別の Oracle のマニュアルを参照しているすべての箇所と、オペレーティング・システムによって異なる初期化パラメータのリストを記載します。Oracle を複数のオペレーティング・システムで使用している場合は、それらのオペレーティング・システム間でアプリケーションの移植性を確保するために、この付録の情報が参考になります。

次のリストは、このマニュアルのオペレーティング・システム別のトピックを、ページ順に示したものです。

- 管理者権限、前提条件 : 5-3 ページの「管理者権限での接続」、接続文字列の構文 : 『Oracle9*i* Net Services 管理者ガイド』
- 監査 : 24-5 ページの「オペレーティング・システムの監査証跡に必ず記録されるイベント」および 24-6 ページの「オペレーティング・システム監査証跡に対する監査」
- DBA の認証 : 5-3 ページの「管理者権限での接続」および 22-13 ページの「データベース管理者の認証」
- ユーザーの認証 : 22-13 ページの「オペレーティング・システムによる認証」
- バックグラウンド・プロセス、ARC*n*: 8-14 ページの「アーカイバ・プロセス（ARCn）」および『Oracle9*i* バックアップおよびリカバリ概要』
- バックグラウンド・プロセス、作成 : 8-6 ページの「バックグラウンド・プロセス」
- バックグラウンド・プロセス、DBW*n* プロセス : 8-9 ページの「データベース・ライター・プロセス（DBWn）」
- クライアント / サーバー通信 : 8-22 ページの「専用サーバー構成」
- 通信ソフトウェア : 8-25 ページの「オペレーティング・システムの通信ソフトウェア」

- Oracle の構成 : 8-3 ページの「プロセスのタイプ」、専用サーバー（2 タスクの Oracle）については、8-22 ページの「専用サーバー構成」、共有サーバーについては、8-17 ページの「共有サーバー・アーキテクチャ」
- データ・ブロック、サイズ : 2-4 ページの「データ・ブロックの概要」
- データ・ファイル、ファイル・ヘッダーのサイズ : 3-18 ページの「データ・ファイルの概要」
- 専用サーバー、管理操作のための要求 : 8-21 ページの「共有サーバーの限定的運用」
- 索引、索引ブロックのオーバーヘッド : 10-34 ページの「索引ブロックの形式」
- オペレーティング・システム監査証跡 : 監査証跡が Oracle からはアクセスできるようになっていないオペレーティング・システムの場合、これらの監査証跡レコードは、バックグラウンド・プロセスのトレース・ファイルと同じディレクトリにある Oracle 監査証跡ファイルに格納されます。

  オペレーティング・システムの監査証跡が使用可能になっている場合に、Oracle は監査証跡レコードをオペレーティング・システムの監査証跡に送ることができます。その他のオペレーティング・システムの場合、これらの監査レコードが、他の Oracle トレース・ファイルと似た形式でデータベース外のファイルに書き込まれることもあります。

  この機能がオペレーティング・システムで実装されているかどうかを調べるには、プラットフォーム固有の Oracle マニュアルを参照してください。
- Oracle Net Services、ネットワーク・ドライバの選択とインストール : 8-25 ページの「プログラム・インタフェース・ドライバ」
- Oracle Net Services、Oracle Net Services ソフトウェア付属のドライバ : 6-8 ページの「Oracle Net Services の機能」および『Oracle9*i* Net Services 管理者ガイド』
- パスワード・ファイルと認証方式 : 22-13 ページの「データベース管理者の認証」
- プログラム・グローバル領域（PGA）: 7-20 ページの「SQL 作業領域」
- オペレーティング・システムによるロール管理 : 23-23 ページの「オペレーティング・システムとロール」
- ソフトウェア・コード領域、共有または非共有 : 7-23 ページの「ソフトウェア・コード領域」

B

廃止機能に関する情報

この付録の内容は、次のとおりです。

- ディクショナリ管理表領域内でのエクステントの割当て
- ロールバック・セグメントの概要
- PCTFREE、PCTUSED と行連鎖

旧バージョンの Oracle でデータベースを作成した場合は、これらの機能を使用している可能性があります。ディクショナリ管理表領域よりローカル管理表領域、ロールバック・セグメントを使用する手動 UNDO 領域管理より自動 UNDO 領域管理を使用することをお薦めします。

ディクショナリ管理表領域内でのエクステントの割当て

Oracle8*i* より前のリリースでは、表領域はすべてディクショナリ管理表領域として作成されていました。ディクショナリ管理表領域では、ディクショナリ表に依存して領域の使用率が追跡されます。Oracle8*i* からは、データ・ディクショナリ表のかわりにビットマップを使用して使用済み領域と空き領域が追跡される、ローカル管理表領域を作成できるようになりました。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。

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。

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

1. 次のアルゴリズムにより、空き領域（そのセグメントを含む表領域内）を検索し、増分エクステントより大きいサイズを持つ最初の連続した空きデータ・ブロックの集合を探します。

   a. 内部的な断片化を少なくするために、新しいエクステントのサイズに 1 ブロックを加えたサイズと一致する、連続したデータ・ブロックの集合を検索します。（必要に応じて、そのサイズは表領域の最小エクステント・サイズの単位にまで切り上げられます。）たとえば、新しいエクステントに 19 個のデータ・ブロックが必要な場合は、正確に 20 個の連続したデータ・ブロックを検索します。ただし、新しいエクステントが 5 ブロック以下の場合は、必要なブロック数に余分のブロックは追加しません。

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

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

c. 必要なサイズ以上の連続したデータ・ブロックの集合が見つからない場合は、対応する表領域内の空いている隣接したデータ・ブロックを結合して、より大きな連続データ・ブロックの集合を形成します。(SMON バックグラウンド・プロセスも、連続した空き領域を定期的に結合します。) 表領域のデータ・ブロックを結合した後、1a および 1b で説明した検索を再度実行します。

d. この 2 回目の検索の後でもエクステントを割り当てることができない場合には、自動拡張によりファイルのサイズ変更を試みます。ファイルのサイズ変更ができない場合、エラーが戻されます。

2. 表領域内に必要な空き領域を見つけて割り当てた後、Oracle は、増分エクステントのサイズに相当する空き領域の一部を割り当てます。エクステントに必要な容量よりも大きな空き領域が見つかった場合、残りは空き領域として残します（5 個以上の連続したブロックの場合)。

3. 新しいエクステントが割り当てられたことと、その割り当てられた領域が使用できなくなったことを示すように、セグメント・ヘッダーとデータ・ディクショナリを更新します。

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

ロールバック・セグメントの概要

旧リリースでは、ロールバック・セグメントを使用して UNDO 領域管理を実行していました。現行バージョンでは、この方法を手動 UNDO 管理モードと呼んでいます。また、このモードはどの互換性レベルでもサポートされます。新機能を利用するために Oracle9*i* を実行する必要があり、まだ自動 UNDO 管理モードに切り替える準備が完了していない場合は、手動 UNDO 管理モードを使用してください。

注意： 手動 UNDO 管理モードはサポートされていますが、自動 UNDO 管理モードでの実行をお薦めします。

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

注意： 自動 UNDO 管理を使用することをお薦めします。この項には、以前のリリースとの下位互換性についての情報のみが含まれています。

関連項目：

- 2-16 ページの「自動 UNDO 管理」
- 読込み一貫性の詳細は、20-5 ページの「マルチバージョン並行性制御」を参照してください。
- 16-8 ページの「トランザクションのロールバック」

ロールバック・セグメントの内容

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

データベース・ユーザーと管理者のいずれも、ロールバック・セグメントへのアクセスや読取りを実行できません。Oracle のみが書込みや読取りを実行できます。（ロールバック・セグメントは、実際に作成したユーザーとは関係なく、ユーザー SYS によって所有されます。）

ロールバック・エントリのログ記録の方法

ロールバック・エントリはロールバック・セグメント内のデータ・ブロックを変更します。Oracle は、データ・ブロックのすべての変更（ロールバック・エントリを含む）を REDO ログに記録します。この 2 次的なロールバック情報の記録は、システム・クラッシュの時点でのアクティブ・トランザクション（まだコミットまたはロールバックされていないトランザクション）にとって非常に重要です。システム・クラッシュが発生すると、アクティブ・トランザクションのロールバック・エントリを含むロールバック・セグメント情報は、インスタンス・リカバリまたはメディア・リカバリの一部として自動的にリストアされます。リカバリが完了すると、Oracle はシステム・クラッシュの時点でコミットまたはロールバックされていなかったトランザクションのロールバックを実行します。

ロールバック情報が必要になる場合

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

ロールバック・セグメントは、各トランザクションでデータが変更される前に、そのデータを記録します。各トランザクションについて、新しい変更がそれぞれ前の変更とリンクされます。トランザクションをロールバックする必要がある場合は、一連の変更が、データを直前の状態にリストアするのに必要な順序でデータ・ブロックに適用されていきます。

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

トランザクションとロールバック・セグメント

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

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

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

トランザクションをコミットした時点で、ロールバック情報は解放されますが、すぐに破棄されるわけではありません。トランザクションがコミットされる前に開始された問合せに対して適切なデータの読取り一貫ビューを作成できるようにするために、その情報はロールバック・セグメントに残っています。そのようなビューのためにロールバック・データを可能なかぎり長く使用可能にするために、Oracle は、ロールバック・セグメントのエクステントを順次方式で書き込みます。ロールバック・セグメントの最後のエクステントがいっぱいになると、折り返してセグメント内の最初のエクステントに上書きしてロールバック・データの書込みが継続されます。長時間実行のトランザクション（アイドル状態またはアクティブ状態）では、新しいエクステントをロールバック・セグメントに割り当てる必要が生じることがあります。

トランザクションでロールバック・セグメントのエクステントを使用する方法の詳細は、B-7 ページの図 B-1、B-8 ページの図 B-2 および B-9 ページの図 B-3 を参照してください。

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

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

ロールバック・セグメントの作成時には、記憶域パラメータを指定して、そのセグメントへのエクステントの割当てを制御できます。各ロールバック・セグメントには、最低 2 つのエクステントが割り当てられる必要があります。

1 つのトランザクションは、1 つのロールバック・セグメントに順次方式で書込みを行います。各トランザクションは、どの時点でもロールバック・セグメントの 1 つのエクステントにのみ書き込みます。多数の**アクティブ**なトランザクションが、同時に 1 つのロールバック・セグメントに書込みを実行できます。これはロールバック・セグメントの同じエクステントでも同様です。ただし、ロールバック・セグメントのエクステントにある各データ・ブロックに含めることができる情報は、1 つのトランザクションに関するもののみです。

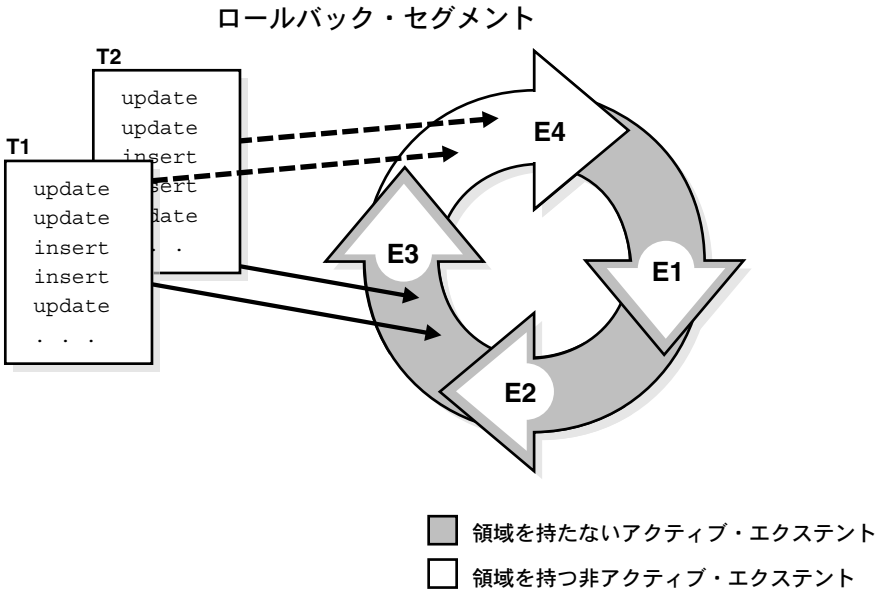
トランザクションがカレント・エクステント領域をすべて使用し尽くしても書込みを継続する必要がある場合、Oracle は、次の 2 つの 方法のどちらかによって、同じロールバック・セグメントから使用可能なエクステントを見つけます。

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

ロールバック領域をさらに取得する必要のあるトランザクションは、ロールバック・セグメントの次のエクステントを調べます。ロールバック・セグメントの次のエクステントにアクティブなトランザクションからの情報が含まれていない場合、Oracle はそれをカレント・エクステントにします。これにより、さらに領域を必要とするすべてのトランザクションは、新しいカレント・エクステントにロールバック情報を書き込めるようになります。

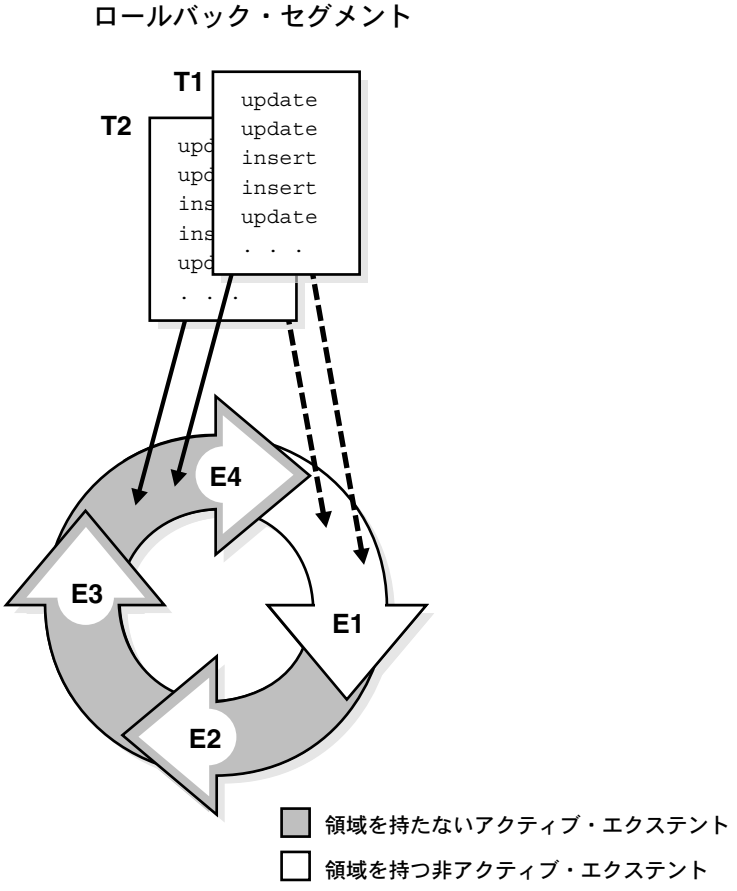
図 B-1 は、ロールバック・セグメントの第 3 エクステント（E3）への書込みを開始し、続いて第 4 エクステント（E4）に書き込む、2 つのトランザクション T1 および T2 を示しています。

図 B-1 ロールバック・セグメントに割り当てられたエクステントの使用



トランザクションが書込みを継続し、カレント・エクステントがいっぱいになると、Oracle はロールバック・セグメントにすでに割り当てられている次のエクステントを調べ、そのエクステントが使用可能かどうかを判断します。図 B-2 で、E4 が完全にいっぱいであるときに T1 と T2 がさらに書込みをする場合は、ロールバック・セグメントに割り当てられている次に使用可能なエクステントに書込みます。この図では、E1 が次のエクステントに相当します。この図は、ロールバック・セグメント内のエクステントが循環的に使用されることを示しています。

図 B-2 ロールバック・セグメントに割り当てられたエクステントの循環的な使用

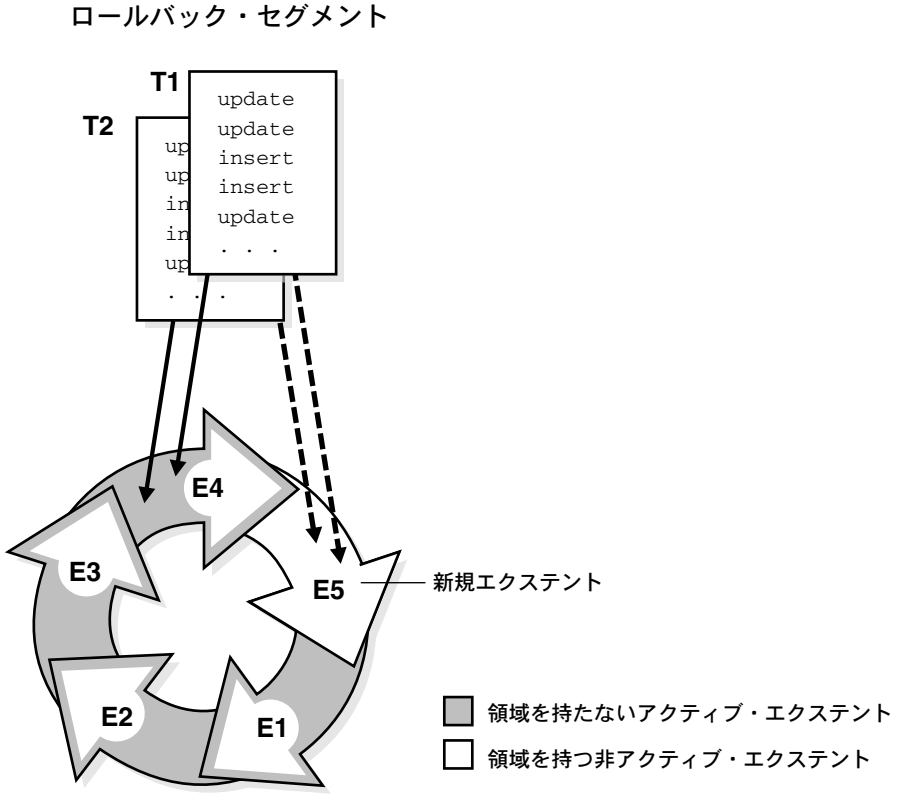


トランザクションのロールバック情報の書込みを継続するときに、Oracle は常にリング内の次のエクステントを最初に再利用しようとします。ただし、次のエクステントにアクティブ・トランザクションからのデータが含まれている場合は、新しいエクステントを割り当てる必要があります。エクステントの数が、ロールバック・セグメントの記憶域パラメータ MAXEXTENTS に設定されている値に達するまで、Oracle は、ロールバック・セグメントに新しいエクステントを割り当てることができます。

図 B-3 に、ロールバック・セグメントに割り当てられた新しいエクステントを示します。コミットされていないトランザクションが長時間実行されています（アイドル状態、アクティブ状態または永続のインダウト分散トランザクション）。この時点で、トランザクションはロールバック・セグメントの第 4 エクステント（E4）に書き込んでいます。しかし、ロール

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

図 B-3 ロールバック・セグメントへの新しいエクステントの割当て



ロールバック・セグメントからのエクステントの割当て解除

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

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

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

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

SYSTEM ロールバック・セグメント

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

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

Oracle インスタンスとロールバック・セグメントのタイプ

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

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

1. インスタンスは、最低 1 つのロールバック・セグメントを取得する必要があります。データベースにアクセスしているインスタンスが唯一のインスタンスの場合は、SYSTEM セグメントを取得します。あるインスタンスが、Real Application Clusters 環境のデータベースにアクセスしている複数のインスタンスの 1 つである場合、そのインスタンスは、SYSTEM ロールバック・セグメントと、それ以外に最低 1 つのロールバック・セグメントを取得します。これらのセグメントを取得できないと、エラーが戻されて、インスタンスはデータベースをオープンできません。

2. インスタンスは、少なくとも、次の初期化パラメータの値の比率と等しい数のロールバック・セグメントを常に取得しようとします。

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

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

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

3. インスタンスは、SYSTEM ロールバック・セグメントを取得した後、そのインスタンスの ROLLBACK_SEGMENTS パラメータに指定されているプライベート・ロールバック・セグメントを すべて取得しようとします。Oracle Real Application Clusters 内のインスタンスがデータベースをオープンして、別のインスタンスがすでに要求しているプライベート・ロールバック・セグメントを取得しようとした場合、ロールバック・セグメン

トを取得しようとしている 2 番目のインスタンスは起動時にエラーを受け取ります。また、存在しないプライベート・ロールバック・セグメントを取得しようとするインスタンスも、エラーを受け取ります。

4. インスタンスが、手順 3 で十分な数のプライベート・ロールバック・セグメントを取得できた場合は、それ以上のアクションは必要ありません。ただし、さらにロールバック・セグメントが必要な場合は、インスタンスはパブリック・ロールバック・セグメントを取得しようとします。

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

Oracle9*i* Real Application Clusters が使用するデータベースは、パブリック・セグメントとプライベート・セグメントの両方を持っています。プライベート・セグメントの使用をお薦めします。

関連項目：

Real Application Clusters でロールバック・セグメントを使用する方法の詳細は、次のマニュアルを参照してください。

- 『Oracle9*i* Real Application Clusters 概要』
- 『Oracle9*i* Real Application Clusters 管理』

ロールバック・セグメントの状態

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

ロールバック・セグメントの状態は、次のとおりです。

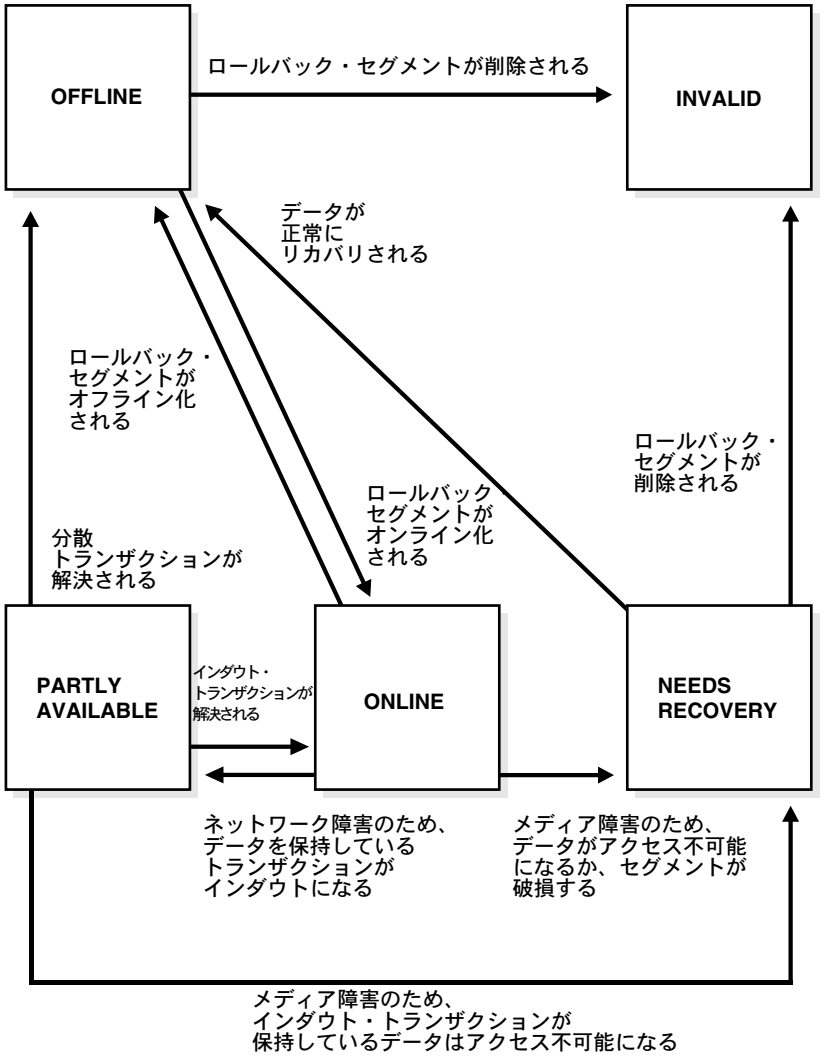
`OFFLINE`	インスタンスによって取得されていません（オンライン化されていません）。
`ONLINE`	インスタンスによって取得されています（オンライン化されています）。アクティブ・トランザクションのデータが入っていることがあります。
`NEEDS RECOVERY`	ロールバックできず（関係するデータ・ファイルにアクセスできないため）、コミットされていないトランザクションのデータが入っているか、データが破損しています。

PARTLY AVAILABLE	インダウト・トランザクション（つまり、未解決の分散トランザクション）のデータが含まれています。
INVALID	削除されています（このロールバック・セグメントに割り当てられていた領域は、後の新しいロールバック・セグメントの作成時に使用されます。）

データ・ディクショナリ表 DBA_ROLLBACK_SEGS には、各ロールバック・セグメントの状態と、その他のロールバック情報がリストされます。

図 B-4 に、ロールバック・セグメントの状態の変化を示します。

図 B-4 ロールバック・セグメントの状態と状態遷移



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

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

PARTLY REMOVE_AVAILABLE のロールバック・セグメントをオンライン化する（文を使用するか、インスタンスの起動時に）と、そのセグメントは新しいトランザクションで使用できるようになります。ただし、インダウト・トランザクションがトランザクション表の一部のエントリをそのまま保持しているため、そのロールバック・セグメントを使用できる新しいトランザクションの数は制限されます。

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

関連項目：

- 分散トランザクションでの障害の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- トランザクション表の詳細は、B-5 ページの「ロールバック情報が必要になる場合」を参照してください。

遅延ロールバック・セグメント

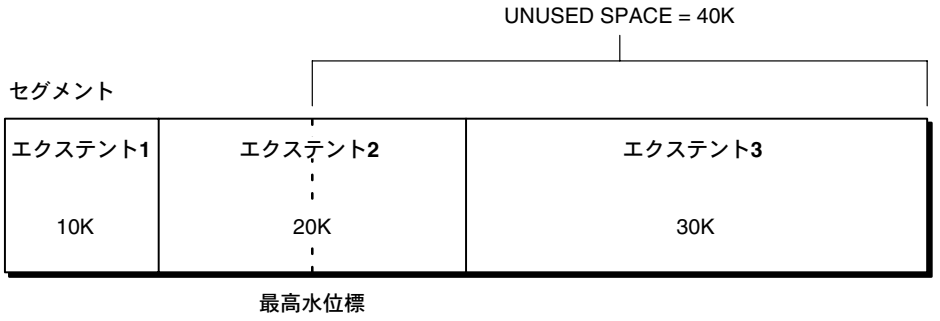
表領域がオフラインになったためにトランザクションをすぐにロールバックできない場合、Oracle は**遅延ロールバック・セグメント**に書き込みます。遅延ロールバック・セグメントには、表領域に適用できなかったロールバック・エントリが入っており、その表領域がオンラインに戻ったときに適用できるようになっています。表領域がオンラインに戻ってリカバリされると、これらのセグメントは消滅します。遅延ロールバック・セグメントは、SYSTEM 表領域内に自動的に作成されます。

最高水位標

最高水位標は、セグメント内の使用済み領域と未使用領域間の境界です。受け取った新しい空きブロックへの要求が既存の空きリストでは満たせない場合、最高水位標が示すブロックは使用済みブロックになり、最高水位標は次のブロックに進められます。言い換えると、最高水位標の左にあるセグメント領域は使用済みであり、右にある領域は未使用です。

図 B-5 は、それぞれの領域が、10K、20K、30K の 3 つのエクステントで構成されているセグメントを示します。最高水位標は、2 つめのエクステントの中央にあります。このように、このセグメントには、最高水位標の左に使用済み領域が 20K、右に未使用領域が 40K あります。

図 B-5 最高水位標



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。

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。

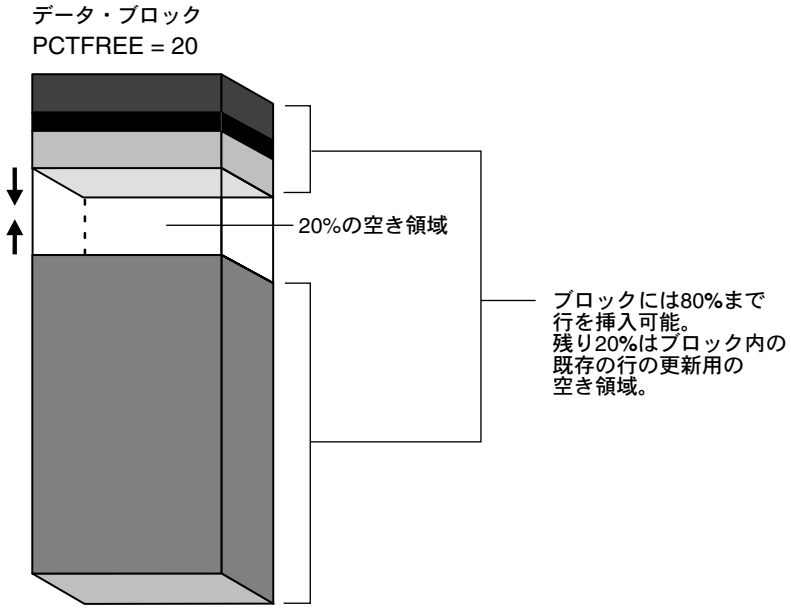
詳細は、12-13 ページの「LOB データ型」を参照してください。

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

```
PCTFREE 20
```

これによって、この表のデータ・セグメント内の各データ・ブロックの 20% が空き状態で維持されます。この空き領域は、各ブロック内の既存の行が更新される場合に使用されます。行データとオーバーヘッドの合計が合計ブロック・サイズの 80% になるまで、新規の行を行データ領域に追加し、それに対応する情報をオーバーヘッド領域の可変部分に追加できます。図 B-6 は、PCTFREE を示しています。

図 B-6 PCTFREE

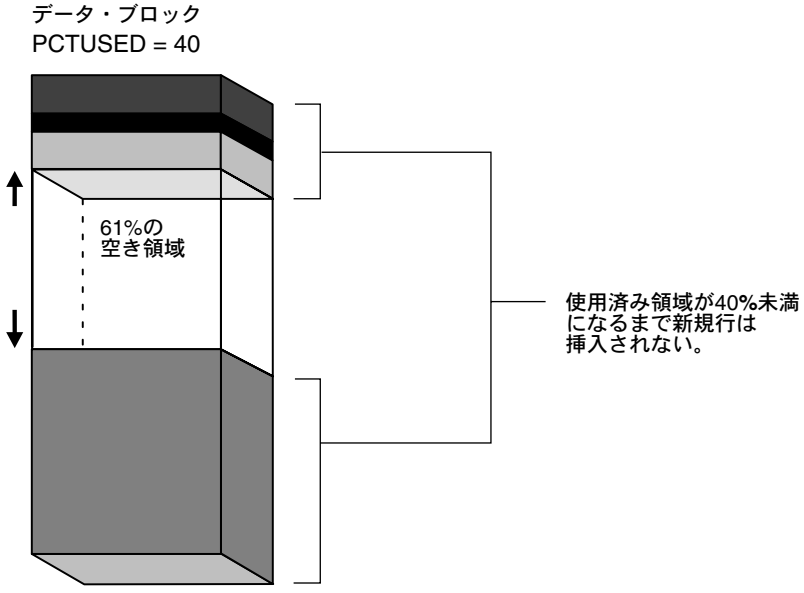


PCTUSED パラメータ PCTUSED パラメータは、新しい行をブロックに追加するときに、行データとオーバーヘッドに使用できるブロックの割合の最小値を設定します。データ・ブロックが PCTFREE で指定した限界値まで満たされると、そのブロックにおける割合がパラメータ PCTUSED の値を下回るまで、Oracle はそのブロックを新しい行の挿入には使用できないものとみなします。この PCTUSED の値に到達するまでは、データ・ブロックの空き領域は、すでにデータ・ブロックに入っている行の更新にのみ使用されます。たとえば、CREATE TABLE 文で次のようにパラメータを指定するとします。

```
PCTUSED 40
```

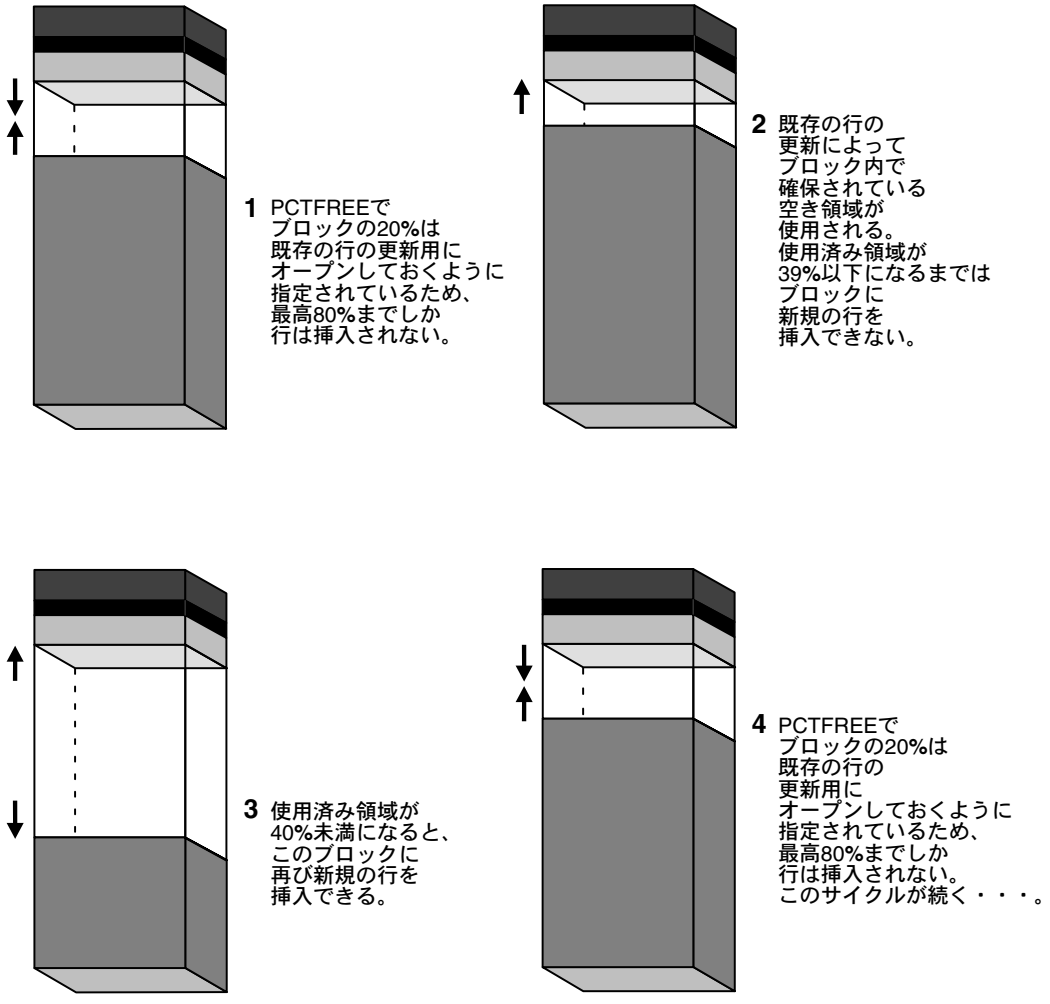
この場合、この表のデータ・セグメントとして使用されるデータ・ブロックは、ブロックの使用領域の総量が 39% 以下になるまでは、新しい行の挿入に使用できないものとみなされます（ブロックの使用領域がそれ以前に PCTFREE に達していたと仮定します）。図 B-7 は、このことを示しています。

図 B-7 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 B-8 は、この 2 つのパラメータの相互作用を示しています。

図 B-8 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 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、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、空きリストから外されます。1 つのセグメントに複数の空きリストがあれば、複数の挿入が同時に行われるときに、空きリストの競合を軽減できます。

DELETE 文または UPDATE 文が発行されると、Oracle はその文を処理して、ブロック内の使用中の領域が PCTUSED よりも小さくなったかどうかを調べます。小さい場合、そのブロックはトランザクション空きリストの先頭に登録され、そのトランザクションで最初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。

用語集

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 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、文のトリガーと行のトリガーの両方に適用される。

関連項目：「トリガー（trigger）」

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 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、文のトリガーと行のトリガーの両方に適用される。

関連項目：「トリガー（trigger）」

CHECK 制約（CHECK constraint）

列または列の集合に対する CHECK 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 CHECK 制約の条件が FALSE に評価される場合、その文はロールバックされる。

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（roll back）」

DBTZ

データベース・タイム・ゾーン。

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。

DOP

操作の並列度。

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。

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ツール。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 Oracle9*i* 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 value）」

NULL 値（NULL value）

1 つの行の 1 つの列に値がないこと。NULL は、欠落しているデータ、不明なデータ、または適用できないデータを示す。他の値（0（ゼロ）など）を暗示する目的では、NULL は使用できない。

Oracle XA

Oracle XA ライブラリは、Oracle サーバー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 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（database）」、「プロセス（process）」、「サーバー（server）」

Oracle プロセス（Oracle process）

Oracle プロセスは、Oracle サーバーのコードを実行する。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（process）」、「サーバー・プロセス（server process）」、「バックグラウンド・プロセス（background process）」、「ユーザー・プロセス（user process）」

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」

Real Application Clusters

Oracle9*i* Enterprise Edition のオプション。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。

関連項目：「インスタンス（instance）」

Recovery Manager（RMAN）

バックアップおよびリカバリの操作を管理する Oracle ユーティリティ。データベース・ファイル（データ・ファイル、制御ファイルおよびアーカイブ REDO ログ・ファイル）のバックアップを作成したり、バックアップを使用してデータベースのリストアやリカバリを行う。

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みデータベース・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（online redo log）」、「アーカイブ REDO ログ（archived redo log）」

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（system global area: SGA）」

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。

SDTZ

カレント・セッション・タイム・ゾーン。

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て指定されたタスクを実行するためのプロシージャを自動的に生成する。

関連項目：「SQL*Plus」、「PL/SQL」

SQL*Plus

Oracle データベースに対して SQL 文を実行するために使用する Oracle のツール製品。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」、「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）

Oracle9*i* 以上の NCHAR、NVARCHAR2 または NCLOB 型の列。Unicode 列は、Unicode を保持できることが保証されている。

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。

アーカイブ REDO ログ（archived redo log）

オンライン REDO ログ・ファイルがいっぱいになった場合、アーカイブ REDO ログを作成して再利用する前にアーカイブすることもできる。アーカイブ REDO ログは、アーカイブ済み（オフライン）REDO ログ・ファイルによって構成される。

関連項目：「REDO ログ（redo log）」

アーキテクチャ（architecture）

関連項目：「Oracle アーキテクチャ（Oracle architecture）」

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（integrity constraint）」、「キー（key）」

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作業領域が必要なときに、Oracle 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（extent）」、「セグメント（segment）」

インスタンス（instance）

Oracle インスタンスは、システム・グローバル領域（SGA）および Oracle 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 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（background process）」、「システム・グローバル領域（system global area: SGA）」

エクステント（extent）

論理データベース記憶域の第 2 レベル。特定数の連続したデータ・ブロック。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る。

関連項目：「データ・ブロック（data block）」、「セグメント（segment）」

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。

オフライン REDO ログ（offline redo log）

関連項目：「アーカイブ REDO ログ（archived redo log）」

オンライン REDO ログ（online redo log）

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

関連項目：「REDO ログ（redo log）」、「システム・グローバル領域（system global area: SGA）」、「バックグラウンド・プロセス（background process）」、「ログ・ライター・プロセス（log writer process: LGWR）」

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。

また、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 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（integrity constraint）」、「主キー（primary key）」

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。キーは、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を記述するものである。

関連項目：「整合性制約（integrity constraint）」、「外部キー（foreign key）」、「主キー（primary key）」

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である。

関連項目：「列（column）」、「表（table）」

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（dedicated server）」

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一意の SQL 文それぞれを処理するために、共有 SQL 領域が必要になる。

関連項目：「システム・グローバル領域（system global area: SGA）」、「SQL」

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（client/server architecture）」、「サーバー（server）」

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。

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（client）」、「クライアント / サーバー・アーキテクチャ（client/server architecture）」

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 と対話する。

関連項目：「プロセス（process）」、「ユーザー・プロセス（user process）」

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、その表に対する SQL 文の実行を高速化できる。

関連項目：「クラスタ（cluster）」

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（index）」、「セグメント（segment）」

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。

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る。

関連項目：「キー（key）」

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（instance）」

シノニム（synonym）

表、ビュー、順序またはプログラム・ユニットの別名。

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。1 つの表には 1 つの主キーのみが定義できる。

関連項目：「主キー制約（PRIMARY KEY constraint）」

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（integrity constraint）」

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。

スーパータイプ（supertype）

関連項目：「サブタイプ（subtype）」

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（logical structures）」

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オンライン 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（physical structures）」、「REDO ログ（redo log）」

整合性（integrity）

関連項目：「データ整合性（data integrity）」

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（extent）」、「データ・ブロック（data block）」

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 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（connection）」、「インスタンス（instance）」、「ユーザー・プロセス（user process）」

接続（connection）

ユーザー・プロセスと Oracle インスタンスとの間の通信経路。

関連項目：「セッション（session）」、「ユーザー・プロセス（user process）」

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（shared server）」

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（並行性）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：「並行性（concurrency）」

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（integrity constraint）」、「トリガー（trigger）」

データ・セグメント（data segment）

それぞれの非クラスタ化表には、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。クラスタ内のすべての表のデータが、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（cluster）」、「エクステント（extent）」、「セグメント（segment）」

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（data file）

Oracle で作成されたディスク上のオペレーティング・システム物理ファイルで、表や索引などのデータ構造が含まれる。データ・ファイルは、1 つのデータベースにのみ属することができる。

関連項目：「索引（index）」、「物理構造（physical structures）」

データ・ブロック（data block）

Oracle データベース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（extent）」、「セグメント（segment）」

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。

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（system global area: SGA）」

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（system global area: SGA）」

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（使用済み）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（buffer cache）」

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（shared server）」

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」

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：「COMMIT」、「ロールバック（roll back）」

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。

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 は、インスタンスごとに 1 組のバックグラウンド・プロセスを作成する。

関連項目：「インスタンス（instance）」、「プロセス（process）」、「Oracle プロセス（Oracle process）」、「ユーザー・プロセス（user process）」

バッファ・キャッシュ（buffer cache）

データベース・バッファ・キャッシュは SGA の一部で、データ・ファイルから読み込んだデータ・ブロックのコピーを保持する。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、データベース・バッファ・キャッシュへのアクセスを共有する。

関連項目：「システム・グローバル領域（system global area: SGA）」

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合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。ビュー上で実行する操作は、そのビューの実表に影響を与える。

表（table）

Oracle データベース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（column）」、「行（row）」

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（logical structures）」

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 データベースの物理データベース構造。

関連項目：「論理構造（logical structures）」

フラッシュバック問合せ（flashback query）

Oracle のマルチバージョン読込み一貫性機能を使用しており、必要に応じて取消しを適用することでデータをリストアする。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、履歴データの表示、修復およびデータベースへの問合せを実行できる。

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。サーバー・プロセスの起動時に、Oracle によって作成される。PGA 内の情報は Oracle の構成によって異なる。

プロセス（process）

Oracle インスタンスにおける各プロセスは、特定のジョブを実行する。Oracle とデータベース・アプリケーションの作業を複数のプロセスに分割することにより、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（Oracle process）」、「ユーザー・プロセス（user process）」

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。

並行性（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、適切な並行性制御が必要である。

関連項目：「データ整合性（data consistency）」

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、表データへの間接アクセスを提供する。

関連項目：「ビュー（view）」

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（process）」、「Oracle プロセス（Oracle process）」

ユーザー名（user name）

Oracle サーバーと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 データベース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。

読込み一貫性（read consistency）

マルチユーザー環境では、Oracle の読込み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（文レベルの読込み一貫性）。
- 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：「並行性（concurrency）」、「データ整合性（data consistency）」

ラージ・プール（large pool）

Oracle 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（system global area: SGA）」、「プロセス（process）」、「共有サーバー（shared server）」、「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（row）」、「表（table）」

ロールバック（roll back）

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。トランザクションをコミットした後は、ロールバックできない。

Oracle は、ロールバック・セグメントを使用して古い値を格納する。REDO ログには、変更の履歴が記録される。

関連項目：「COMMIT」、「トランザクション（transaction）」、「ロールバック・セグメント（rollback segment）」

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。

関連項目：「COMMIT」、「論理構造（logical structures）」、「セグメント（segment）」

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（redo log）」

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 データベースの論理構造。物理構造と論理構造が分離されているので、論理的な記憶構造へのアクセスに影響を与えずに、データの物理的な記憶域を管理できる。

関連項目：「物理構造（physical structures）」

索引

数字

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

ね

の

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

れ

ろ

わ